Tree Comparison

Enumeration and Application to Cheminformatics

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Andre Droschinsky

Dortmund
2021

Andre Droschinsky
Lehrstuhl XI - Algorithm Engineering
Fakultät für Informatik
Technische Universität Dortmund
Otto-Hahn-Str. 14
44227 Dortmund

Tag der mündlichen Prüfung:

Dekan	**Prof. Dr.-Ing. Gernot A. Fink**
Gutachter	**Prof. Dr. Petra Mutzel** (Universität Bonn, Fakultät für Informatik)
	Prof. Dr. Johannes Fischer (TU Dortmund, Fakultät für Informatik)

Abstract

Graphs are a well-known data structure used in many application domains that rely on relationships between individual entities. Examples are social networks, where the users may be in friendship with each other, road networks, where one-way or bidirectional roads connect crossings, and work package assignments, where workers are assigned to tasks. In chem- and bioinformatics, molecules are often represented as molecular graphs, where vertices represent atoms, and bonds between them are represented by edges connecting the vertices. Since there is an ever-increasing amount of data that can be treated as graphs, fast algorithms are needed to compare such graphs. A well-researched concept to compare two graphs is the maximum common subgraph. On the one hand, this allows finding substructures that are common to both input graphs. On the other hand, we can derive a similarity score from the maximum common subgraph. A practical application is rational drug design which involves molecular similarity searches.

In this thesis, we study the maximum common subgraph problem, which entails finding a largest graph, which is isomorphic to subgraphs of two input graphs. We focus on restrictions that allow polynomial-time algorithms with a low exponent. An example is the maximum common subtree of two input trees. We succeed in improving the previously best-known time bound. Additionally, we provide a lower time bound under certain assumptions. We study a generalization of the maximum common subtree problem, the block-and-bridge preserving maximum common induced subgraph problem between outerplanar graphs. This problem is motivated by the application to cheminformatics. First, the vast majority of drugs modeled as molecular graphs is outerplanar, and second, the blocks correspond to the ring structures and the bridges to atom chains or linkers. If we allow disconnected common subgraphs, the problem becomes NP-hard even for trees as input. We propose a second generalization of the maximum common subtree problem, which allows skipping vertices in the input trees while maintaining polynomial running time.

Since a maximum common subgraph is not unique in general, we investigate the problem to enumerate all maximum solutions. We do this for both the maximum common subtree problem and the block-and-bridge preserving maximum common induced subgraph problem between outerplanar graphs. An arising subproblem which we analyze is the enumeration of maximum weight matchings in bipartite graphs. We support a weight function between the vertices and edges for all proposed common subgraph methods in this thesis. Thus the objective is to compute a common subgraph of maximum weight. The weights may be integral or real-valued, including negative values. A special case of using such a weight function is computing common subgraph isomorphisms between labeled graphs, where labels between mapped vertices and edges must be equal. An experimental study evaluates the practical running times and the usefulness of our block-and-bridge preserving maximum common induced subgraph algorithm against state of the art algorithms.

Acknowledgments

First of all, I want to thank my advisor Petra Mutzel for providing the opportunity to work in her group, the scientific guidance, and the freedom to pursue my own research directions. I am incredibly thankful for her ever-encouraging words and the enthusiasm she showed about my research. Further, I am thankful for the opportunity to discover various places on earth during my conference trips and the summer school.

I thank my co-advisor, Johannes Fischer, who filled in at short notice. I further thank Peter Buchholz and Günter Rudolph for agreeing to serve on my defense commission. I also thank Heinrich Müller for mentoring me and the high-level talk about my research. A special thanks to Bernhard Heinemann for accepting me into his research group directly after my diploma. I thank the SFB 876 for a place to pursue my research.

I am most grateful to Nils Kriege for accompanying me on the way from the diploma to the dissertation. His research talks and thematic knowledge were most helpful. Many thanks to Lina Humbeck for guiding my research into meaningful areas in the context of cheminformatics. I am grateful to my co-authors that they supported me in writing down my results structured and clear. Their experience was invaluable. For proofreading parts of my thesis, I thank Lina Humbeck, Roman Kalkreuth, Dominik Koeppl, Nils Kriege, Lutz Oettershagen, Till Schäfer, Henning Timm, and Bernd Zey.

I thank my roommates Dominik Kopczynski and Adalat Jabrayilov for sharing the office with me. I appreciate their research input, their incredible Python and Linux knowledge, and their open ears for every other topic. I thank Gundel Jankord for being absolutely reliable and for all the help I received on the organizational work. For all technical needs, I thank our administrator Helmut Henning. I enjoyed the pleasant and productive atmosphere at Chair XI with all my colleagues, especially with everyone from the Algorithm Engineering group.

Finally, I would like to thank my family and friends for their support and for encouraging me to finish my thesis.

Andre Droschinsky
Dortmund, May 2021

Contents

Abstract . . . iii
Acknowledgments . . . iv

1 Introduction **1**
1.1 Contribution and Organization of this Thesis . . . 4
1.2 Corresponding Publications . . . 6

2 Preliminaries **7**
2.1 Numbers, Sets, Landau Symbols . . . 7
2.2 Graphs . . . 7
2.3 BC-Trees . . . 10
2.4 Enumeration . . . 11
2.5 Notation . . . 12

3 Matchings **15**
3.1 Maximum Cardinality Matching in General Graphs . . . 18
3.2 Maximum Weight Bipartite Matching . . . 20
3.2.1 Recent Algorithms . . . 20
3.2.2 Reduction between Perfect and Non-Perfect Matchings . . . 21
3.2.3 Solving Techniques . . . 21
3.2.4 Matching as Integer Linear Program . . . 21
3.3 Maximum Weight Matching on Unbalanced Bipartite Graphs . . . 23
3.3.1 Maximum Cardinality Matching . . . 23
3.3.2 Arbitrary Weights . . . 24
3.3.3 Integral Weights . . . 25
3.3.4 Maximum Weight Matching of Given Cardinality . . . 25
3.3.5 Insignificant Edges in Unbalanced Bipartite Graphs . . . 26
3.4 All-Cavity Maximum Weight Matching . . . 27
3.4.1 Integral Weights . . . 27
3.4.2 Arbitrary Weights . . . 28
3.5 All-Cavity Maximum Weight Matching on Unbalanced Bipartite Graphs . . . 29
3.5.1 Previous Results . . . 29
3.5.2 New Results . . . 30
3.5.3 Further Improvements . . . 34
3.6 Enumerating Maximum Weight Matchings . . . 36
3.7 Summary and Future Work . . . 42

4 Maximum Common Subtree Isomorphisms and Embeddings **45**
4.1 The Maximum Common Subtree Isomorphism Problem . . . 47
4.2 Rooted Maximum Common Subtree Isomorphism . . . 49
4.2.1 Rooted Labeled Maximum Common Subtree Isomorphism . . . 51
4.2.2 Rooted Maximum Weight Common Subtree Isomorphism . . . 52
4.3 (Unrooted) Maximum Common Subtree Isomorphism . . . 54

4.4 Lower Bounds on the Time Complexity and Optimality . . . 58
4.5 Largest Weight Common Subtree Embedding . . . 59
4.5.1 Gupta and Nishimura's Algorithm . . . 60
4.5.2 Largest Weight Common Subtree Embeddings . . . 61
4.5.3 Unrooted Largest Weight Common Subtree Embeddings . . . 64
4.6 Polynomial Delay Enumeration of Maximum Common Subtree Isomorphisms . . . 68
4.6.1 Example and Basic Approach . . . 69
4.6.2 The Enumeration Tree for the MWCSI Problem . . . 69
4.6.3 Running Time Analysis . . . 70
4.7 Similarity . . . 71
4.8 Metrics . . . 72
4.9 Summary and Future Work . . . 75

5 Block-and-bridge preserving Maximum Common Subgraph 77
5.1 Preliminaries . . . 78
5.2 Biconnected Maximum Common Induced Subgraph . . . 79
5.2.1 2-MCIS under a Non-Negative Weight Function . . . 80
5.2.2 2-MCIS with Arbitrary Weights . . . 83
5.3 Block-and-Bridge Preserving Maximum Common Induced Subgraph Isomorphism 85
5.3.1 Partitioning of the Problem . . . 85
5.3.2 BBP-MCISI under a Non-Negative Weight Function . . . 86
5.3.3 BBP-MCISI with Arbitrary Weights . . . 88
5.4 Problem Variants . . . 90
5.4.1 BBP-MCIS Embedding; LaWeCSE$_u$ Integrated into BBP-MCIS . . . 90
5.4.2 Bioisosteres . . . 92
5.4.3 Non-Outerplanar Graphs . . . 94
5.4.4 BBP-MCES . . . 97
5.5 Polynomial Delay Enumeration of all BBP-MCISIs . . . 99
5.5.1 Enumeration of all 2-MCIS Between Two Blocks . . . 100
5.5.2 Enumeration of all BBP-MCISIs . . . 101
5.5.3 Running Time Analysis . . . 101
5.6 Summary and Future Work . . . 101

6 Experimental Evaluation 103
6.1 Implementation . . . 103
6.1.1 Input Files . . . 103
6.1.2 Implementation Details . . . 106
6.1.3 Available Commands and Output . . . 106
6.1.4 Graphical Output . . . 108
6.2 Test Setup . . . 109
6.3 Synthetic Tests . . . 109
6.4 Evaluation on Molecular Graphs . . . 112
6.4.1 BC-Trees Representing Molecular Graph . . . 112
6.4.2 Outerplanar Molecular Graphs . . . 112
6.4.3 BBP-MCIS as Similarity Coefficient in Cheminformatics . . . 114
6.5 Conclusion . . . 116

7 Conclusion and Outlook 123

Bibliography 125

Sollte dies Kaffee sein, bringen Sie mir bitte Tee. Sollte dies Tee sein, bringen Sie mir bitte Kaffee.

Abraham Lincoln
1809 - 1865

Chapter 1
Introduction

Assume there is a reference object and lots of other objects with various degrees of similarity to said object. When the task is to rank these objects from most similar to least similar, a random person would create an order that depends on their subjective perception of the objects. If the reference object is a cheetah, and the others are a tiger, a cat, and a dog, most people probably would rank in exactly that order. Some might rank the cat first, since there is no absolute standard measure for similarity. If there are not three but a thousand objects to rank, any number of people would probably create that many different rankings. The best-ranked results from the majority could be passed to an expert to check them more closely.

In medicinal drug discovery, researchers face a similar problem. In that domain, the research starts with a target protein. The task is to find small molecules that bind to this protein; these are called hits. The drug-like chemical space is estimated to contain 10^{60} molecules and between 10^{20} and 10^{24} molecules with up to 30 atoms [108]. There are databases, such as ChEMBL [6] and PubChem [66], containing data of millions of chemical compounds, including physical properties, biological activities, and toxicity information. Due to the size of the chemical space and the databases, a search for hits must be automated using fast algorithms. A classical approach is to use molecular fingerprints. These are fixed size vectors and contain information about the structure. An example is the very popular class of Extended-Connectivity Fingerprints (ECFPs), topological fingerprints for molecular characterization. ECFPs are circular, and their features represent the presence of particular substructures [109]. Circular fingerprints store hashed information about the atoms and their close neighbors connected through bonds. Which neighbors are close is determined by a radius (path length). Although algorithms on vectors are often fast, this approach suffers from an irreversible information loss and can lead to inaccurate or reduced discrimination. Therefore, we focus on molecular graphs as input. In this context, the atoms are represented by vertices and the bonds between the atoms by edges. The vertices and edges are attributed by the atom type and bond type, respectively. The common agreement in a graphical representation is that vertices without a label are carbon (C) atoms. The arguably most-desired molecule on earth is shown in Figure 1.1a. Often the atoms are color-coded, e.g., red for oxygen (O). A color-coded 3D-model of the same molecule is depicted in Figure 1.1b.

A computer-calculated similarity ranking between molecular graphs can be conducted as follows. First, for each molecule M to compare, the largest possible graph S, which is a subgraph of both M and the reference molecule R, is computed. This is known as the maximum common subgraph

(a) Structural formula (b) Colored 3D-model

Figure 1.1: The caffeine drug in two different representations.

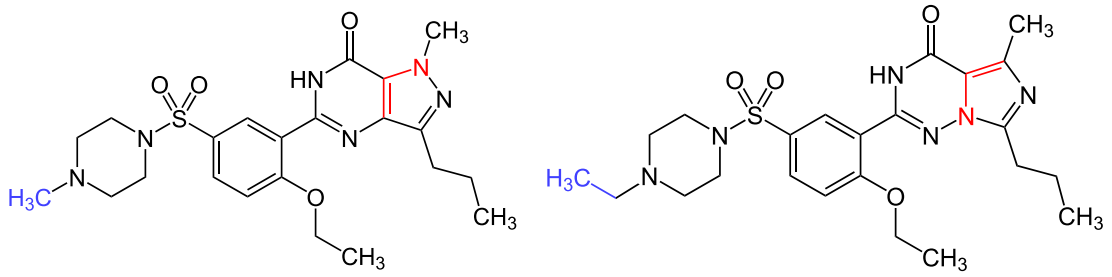
(MCS) problem. Then, from the sizes of S, R, and M, a similarity coefficient is calculated. The objects are then ranked in descending order of that value. A chemist or another expert can further evaluate the top-ranked results. According to Ehrlich and Rarey [33], computing similarity values is the most common application of maximum common subgraph algorithms in cheminformatics. Further applications involve ligand alignment and pharmacophore modeling [33].

However, computing an MCS is NP-hard and therefore not suitable on huge databases. Fortunately, under certain restrictions, polynomial-time algorithms are available. It was shown that the vast majority of drugs modeled as molecular graphs are outerplanar [58]. For the block-and-bridge preserving (BBP) constraint, we propose a novel algorithm to compute a connected maximum common subgraph between outerplanar graphs G and H in time $\mathcal{O}(|G|\,|H|\Delta)$, where Δ is the maximum degree of all vertices. Under this constraint, ring structures (blocks) are mapped to ring structures only. The atom chains or linkers (bridges) are mapped to chains or linkers only. Another property of molecular graphs is that the vertices are of bounded degree. This is due to the limited number of bonds per atom and allows us to solve the BBP maximum common subgraph problem on outerplanar molecular graphs in time $\mathcal{O}(|G|\,|H|)$.

Allowing disconnected common subgraphs improves the quality of the similarity coefficients given that the connected components are arranged consistently in both graphs [82, 113]. However, solving the disconnected variant is NP-hard. A less computationally demanding approach is mapping disjoint paths of bridges (more precisely, the path's endpoints while skipping the inner vertices) to each other. We call this technique *embedding*, following Gupta and Nishimura [50]. To prevent paths of arbitrary length, we introduce a linear penalty depending on the length of such paths. Moreover, small variations of the chemical elements (vertex labels) might be tolerated.

Figure 1.2 shows the two drugs sildenafil and vardenafil, which have a quite similar effect. There are atom substitutions in the bicyclic structure that do not affect the bioactivity, and there is a different chain length on the left side. The classic MCS approach could not map these vertices to each other since their attributes differ. Consequently, a substantial part of the molecules would be omitted. In our algorithm, we allow mapping atoms and bonds with different attributes to each other. In contrast to a labeled approach, we may define an arbitrary weight function, allowing positive and negative weights between the atoms and bonds. Then we maximize the weight of the common subgraph instead of the size. We evaluate these modifications on two benchmark data sets and show that they are beneficial when an adequate weight function and penalty are chosen.

In our BBP-MCS algorithm, several subtasks have to be solved. The first is to decompose the molecular graphs into their blocks and bridges. These blocks, bridges, and the connections between them, are represented by a structure called block-cut tree (BC-tree) [53]. On the BC-trees, we then solve the MCS problem. To this end, we develop a fast dynamic programming approach, where we exploit the similarity between the maximum weight bipartite matching instances that we have to solve [26]. Contrary to the common approach of reducing the maximum weight matching problem to the minimum weight perfect matching problem, we use an algorithm for the maximum



(a) Sildenafil (b) Vardenafil

Figure 1.2: The drugs sildenafil (a) and vardenafil (b). The red colored subgraphs show atoms and bonds that differ between the two molecules. The blue subgraphs differ in the length of the chain.

weight matching problem with a running time depending on the smaller vertex set. We study the maximum weight matching problem and the maximum common subgraph problem on trees extensively, as they are of interest on their own and for various other problems, e.g., in computer vision and pattern recognition [18, 123]. Another subtask of our BBP-MCS algorithm is to find biconnected maximum common subgraphs between blocks. For outerplanar blocks, this is realized by enumerating all maximal (with respect to inclusion) biconnected common subgraphs between the two blocks and deriving a biconnected solution of maximum weight from them. Otherwise, we may use a reduction to the NP-hard maximum clique problem. Our block-based approach reduces the practical running time compared to a pure clique-based algorithm operating on the whole graphs since the computational demanding clique problem must be solved for small components only. In contrast to the BBP-MCS algorithm of [111], the above-described technique enables our algorithm to compute a solution for any two molecular graphs.

Besides algorithmic reasons to use enumeration algorithms (e.g., finding a maximum clique), they find *all* maximum solutions for a given problem. These different maximum solutions may be of interest on their own. In the context of molecular graphs, this increases the probability to find the substructure that is important for the biological activity. In practice, many problems have additional constraints, which are difficult to integrate into an algorithm. Using enumeration, we can select a solution among all maximum solutions which fits best in the current real-world context. For this reason, we develop an enumeration algorithm for the BBP-MCS problem. We also introduce algorithms to enumerate all maximum weight matchings and maximum common subtrees, respectively, since this is essential to enumerate all maximum solutions in our BBP-MCS algorithm.

We implemented our BBP-MCS algorithm in a command-line-based software that includes an optional graphical output.[1] An exemplary console input and output with the graphs from Figure 1.3 is listed in the following. We used the enumeration command to output all maximum solutions. The solution in Figure 1.3 is the fourth line of the console output.

```
./LaWeCSE -c intro.fog 1 2 -d -e

1. BBP-MCS with 5 nodes: (3,3) (2,4) (1,1) (4,2) (5,6)
2. BBP-MCS with 5 nodes: (3,3) (2,4) (1,1) (4,2) (5,5)
3. BBP-MCS with 5 nodes: (3,3) (2,2) (1,1) (4,4) (5,6)
```

[1] See Section 6.1 for details about the software, including implementation, file formats, and command-line parameters.

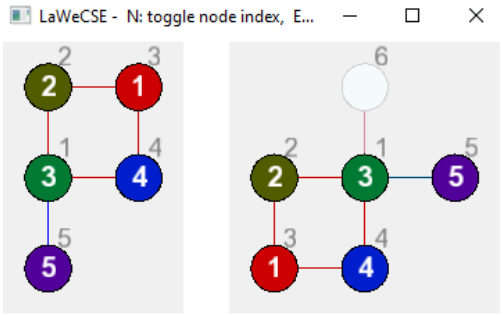


Figure 1.3: Two graphs and their BBP-MCSs. Vertices with the same colors and inner numbers are mapped to each other. The grey numbers are the internal vertex numbers of the input graphs.

```
4. BBP-MCS with 5 nodes: (3,3) (2,2) (1,1) (4,4) (5,5)
Similarity: 0.833333333333333 between Block_Bridge and Block_DualBridge
```

1.1 Contribution and Organization of this Thesis

This work is structured in this introductory chapter (1), a definitions chapter (2), three thematic chapters (3, 4, and 5), an evaluation chapter (6), and a conclusion (7).

In Chapter 2, we introduce basic definitions and terminology from graph theory. Following, we provide definitions for block-cut-trees and the concept of enumeration for listing all optimal solutions. Finally, we list notations, symbols, and abbreviations.

Since matchings are a fundamental subproblem when comparing graphs using the algorithms within this thesis, we dedicate Chapter 3 to this topic. We focus on maximum weight matchings in bipartite graphs. We consider unbalanced graphs, i.e., we provide running time results depending on the sizes of the bipartite vertex sets. Further, we study the computation of maximum weight matchings for a set of similar bipartite graphs, which is known as the all-cavity maximum weight matching problem. Previous results on that problem respect i) the number of edges [63, 17], ii) unbalanced graphs [93], or iii) (possibly integral) weighted edges [63, 93], but to the best of our knowledge, there were no results for i), ii), *and* iii). We present a new algorithm for the all-cavity maximum weight matching problem and prove running time bounds concerning i) to iii), which match or outperform the previously shown time bounds. We further present and analyze an algorithm that enumerates all maximum weight matchings in a given weighted bipartite graph. Previous results consider the unweighted case only, e.g., [125, 124].

Chapter 4 studies the maximum common subtree problem, which asks for a tree isomorphic to subtrees of two input trees. Tree isomorphism is not only relevant for comparing molecular graphs in a tree representation, but there is a large number of applications for this specific problem, e.g., the comparison of binary phylogenetic trees [83]. Several variants of this problem exist. The input trees may be rooted or unrooted, ordered or unordered, vertex labeled, or edge labeled (with a possible weight function between the vertices and between the edges). For the output, we may require that all vertices are connected, allow disjoint paths mapped to each other, or allow arbitrary disconnection. Except for the latter, all these variants are solvable in polynomial time. However, most algorithms have certain restrictions, e.g., only rooted trees [64], only labels on the leaves [83], only vertex labels [123], or no labels at all [50]. The algorithm proposed by Torsello, Rowe, and Pelillo [123] accepts unrooted vertex (but not edge) labeled trees with an

arbitrary weight function as input. The authors proved a running time of $\mathcal{O}(n^3\Delta)$, where n is the number of vertices and Δ the maximum degree of all vertices. We contribute an algorithm that simultaneously supports unordered unrooted trees and integral or real-valued weights between the vertices and edges. In this algorithm, we also allow the mapping of disjoint paths (instead of edges) to each other and support a linear penalty for the number of skipped (inner) vertices. We prove a running time of $\mathcal{O}(n^2\Delta)$ in the worst case. This algorithm generalizes most other maximum common subtree algorithms. If we compute a normalized similarity coefficient from the size or weight of a maximum common subtree (or subgraph), we can use this as a similarity measure in cheminformatics [33]. The skipping vertices approach improves these coefficients' quality, as we show in Chapter 6. We further present and analyze an algorithm to enumerate all maximum common subtree isomorphisms, relying on our enumeration algorithm for maximum weight matchings from Chapter 3.

Chapter 5 covers the block-and-bridge preserving maximum common induced subgraph (BBP-MCIS) problem. Here, the following two properties must hold. First, bridges of the common subgraph must be bridges in both input graphs. Second, any two edges in different biconnected components of the common subgraph must be in different biconnected components of the input graphs. The block-and-bridge preserving constraint was introduced by Schietgat, Ramon, Bruynooghe, and Blockeel [112] and is of high practical relevance in cheminformatics. The authors presented an algorithm to compute a (not necessarily vertex induced) block-and-bridge preserving maximum common edge subgraph between outerplanar graphs. Our contribution is the development of the first algorithm to compute a BBP-MCIS between outerplanar graphs with cubic running time. We address the subtask of computing a biconnected maximum common subgraph (2-MCIS) between outerplanar blocks b_1 and b_2 by enumerating all maximal biconnected common subgraphs between the blocks. Each maximal solution C can be computed in time $\mathcal{O}(|C|)$. The total size of all maximal solutions is $\mathcal{O}(|b_1|\,|b_2|)$. In these maximal solutions, we can find a biconnected maximum weight common subgraph within the same time bound. This generalizes the previous best result for the 2-MCIS problem, which supports unlabeled graphs only. Hence, for all pairs of blocks of outerplanar graphs G and H, this subtask's total running time is $\mathcal{O}(|G|\,|H|)$. We discuss extensions to this algorithm, e.g., bioisosteres and a clique reduction if at least one graph is not outerplanar. Bioisosteres are structurally distinct compounds recognized similar by biological systems [90]. Supporting them can improve the quality of the computed similarity coefficients. Our clique reduction approach computes a maximum solution by enumerating all biconnected c-cliques, a modification to the general approach to solve the maximum common subgraph problem as presented by Koch [67] and corrected by Cazals and Karande [12]. Among them, we keep a biconnected c-clique of maximum size. While the vast majority of drugs may be modeled as outerplanar graphs [58], the support of non-outerplanar graphs allows computing such coefficients for each pair of drugs in a given database. For the BBP-MCIS problem, we also study the enumeration of all maximum solutions.

In Chapter 6, we evaluate our algorithms from Chapters 3, 4, and 5 on synthetic and real-world data and compare the running time against other state of the art algorithms. We implemented the BBP-MCIS algorithm with its backbones, the maximum weight common subtree isomorphism algorithm supporting skipping vertices and the all-cavity maximum weight matching algorithm. We used the C++ programming language and the OGDF framework [15].[2] We further evaluate the quality of the computed similarity coefficients. The results show that our BBP-MCIS algorithm is faster than the BBP-MCES algorithm by Schietgat et al. [112] and much faster than reductions to clique algorithms. Further, we show that the computed similarity coefficients are comparable to the BBP-MCES algorithm and fingerprint-based approaches. Moreover, they improve by allowing skipping vertices.

[2] https://ogdf.uos.de/ – Release 2020-02-09

Finally, Chapter 7 concludes this thesis.

1.2 Corresponding Publications

This thesis partially consists of the results from the following publications, sorted by publication date. Further details are given within the corresponding chapters and sections.

1. A. Droschinsky, P. Mutzel, and E. Thordsen. "Shrinking Trees not Blossoms: A Recursive Maximum Matching Approach". In: *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*. Ed. by G. E. Blelloch and I. Finocchi. SIAM, 2020, pp. 146–160. DOI: 10.1137/1.9781611976007.12

2. N. M. Kriege, A. Droschinsky, and P. Mutzel. "A note on block-and-bridge preserving maximum common subgraph algorithms for outerplanar graphs". In: *J. Graph Algorithms Appl.* 22.4 [2018], pp. 607–616. DOI: 10.7155/jgaa.00480

3. A. Droschinsky, N. M. Kriege, and P. Mutzel. "Largest Weight Common Subtree Embeddings with Distance Penalties". In: *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*. ed. by I. Potapov, P. G. Spirakis, and J. Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 54:1–54:15. DOI: 10.4230/LIPIcs.MFCS.2018.54

4. A. Droschinsky, N. M. Kriege, and P. Mutzel. "Finding Largest Common Substructures of Molecules in Quadratic Time". In: *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017*. Ed. by B. Steffen, C. Baier, M. van den Brand, J. Eder, M. Hinchey, and T. Margaria. Vol. 10139. Lecture Notes in Computer Science. Springer, 2017, pp. 309–321. DOI: 10.1007/978-3-319-51963-0_24

5. A. Droschinsky, N. M. Kriege, and P. Mutzel. "Faster Algorithms for the Maximum Common Subtree Isomorphism Problem". In: *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*. Ed. by P. Faliszewski, A. Muscholl, and R. Niedermeier. Vol. 58. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 33:1–33:14. DOI: 10.4230/LIPIcs.MFCS.2016.33

6. A. Droschinsky, B. Heinemann, N. M. Kriege, and P. Mutzel. "Enumeration of Maximum Common Subtree Isomorphisms with Polynomial-Delay". In: *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*. Ed. by H.-K. Ahn and C.-S. Shin. Vol. 8889. Lecture Notes in Computer Science. Springer, 2014, pp. 81–93. DOI: 10.1007/978-3-319-13075-0_7

For publications 3, 4, 5, and 6, the author of this thesis was the main contributor. This includes all the algorithmic ideas, proofs, and experiments, except for the unlabeled 2-MCIS algorithm in publication 4. For publication 2, the main contributor was Nils Kriege, the author of this thesis contributed roughly a third. For publication 1, Erik Thordsen was the main contributor, the author of this thesis contributed approximately 10 to 15%.

Bei allen Unternehmungen muss vor deren Beginn eine sorgfältige Vorbereitung stehen.

Marcus Tullius Cicero
106 – 43 BCE

Chapter 2

Preliminaries

In this chapter, we introduce basic terminology and notations used in this thesis. We include basic concepts from graph theory consistent with Diestel's book of graph theory [23]. We also introduce the concept of enumeration since we present enumeration algorithms in each of the three following chapters. Additional terminology only relevant for individual chapters is included therein.

2.1 Numbers, Sets, Landau Symbols

We denote the set of all natural numbers by $\mathbb{N} = \{0, 1, \ldots\}$. The set of all integral numbers is denoted by $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$. The set of all real numbers is denoted by $\mathbb{R}$, while $\mathbb{R}^{\geq 0}$ is the set of all non-negative real numbers. For a finite set X, the *cardinality* of X is its number of elements, denoted by $|X|$. For a set X and a natural number $k \leq |X|$, we denote the set of all subsets of X containing exactly k elements by $[X]^k$. If each element $x \in X$ is contained in a set Y, we write $X \subseteq Y$. If we additionally require $X \neq Y$, we write $X \subsetneq Y$. An *alphabet* Σ is a finite (discrete) set of elements. We define *log* as the logarithmic function to the base 2. For two functions f, g, we define $f \circ g \coloneqq f(g)$ as the function composition. We use the Landau symbols, $\mathcal{O}$, Ω, and Θ, as commonly accepted; additionally, we assume each factor or term to be at least 1. This prevents undefined or broken running times when multiple variables are used and allows to omit trivial edge cases in definitions and theorems. For example, a running time of $\mathcal{O}(n \log N)$ for $N = 1$ shall be $\mathcal{O}(n)$, not $\mathcal{O}(0)$. For similar reasons, we define the maximum of an empty subset of a set of numbers as $-\infty$. Given a vector $v = (v_1, \ldots, v_n)$ with n elements, we abbreviate this vector as (v_n).

2.2 Graphs

In this subsection, we define the term *graph* and related structures.

> **Definition 2.1 (Graph).** *A* graph *is a pair $G = (V, E)$ of finite sets, such that $E \subseteq [V]^2$. The elements in V are called* vertices*, the elements in E are called* edges*.*

The vertices are also known as *nodes*. We primarily use the term *vertex*. Most algorithms in this thesis solve specific subproblems on additionally generated graphs. For some of these graphs, we use the term *node*. This choice supports the reader to recognize the relevant structure.

We refer to the set of vertices V by $V(G)$ or V_G and the set of edges E by $E(G)$ or E_G. We may say v is a vertex of G and e is an edge of G, respectively, if $v \in V$ and $e \in E$, respectively. To improve readability, we may denote edges $\{u, v\}$ by their vertices only, i.e., we write uv or vu for the edge $\{u, v\}$. The *order* of G is denoted by $|G|$ and is defined as its number of vertices $|V|$. The *size* of G is defined as its number of edges $|E|$. If not mentioned otherwise, the variables n and m represent the number of vertices and edges, respectively, of the given graph.

Let $e = uv \in E$. We say u and v are *adjacent* to each other. The vertices u and v are *incident* to e. The *neighbors* of a vertex v in G are its adjacent vertices, denoted by $N_G(v) \coloneqq \{u \in V_G \mid vu \in E_G\}$. The *degree* of a vertex v is $\delta_G(v) \coloneqq |N_G(v)|$, the number of its neighbors. The *degree* $\Delta(G)$ of G is the maximum degree of all vertices in G. In the following, if the graph G is clear from the context, we omit the index reference to G, e.g., we write $N(v)$ or $\delta(v)$ instead of $N_G(v)$ or $\delta_G(v)$. A *matching* $\mathcal{M} \subseteq E$ is a set of edges, such that no two edges share a vertex. It is of *maximum cardinality* if there is no matching with more edges than $\mathcal{M}$, and it is perfect if each vertex $v \in V$ is incident to an edge in $\mathcal{M}$. For more details and its problem variants, see Chapter 3.

Definition 2.2 (Directed Graph). *A* directed graph *or* digraph *is a pair* $G = (V, A)$ *of sets, such that* $A \subseteq (V \times V) \setminus \{(v, v) \mid v \in V\}$. *The elements in* A *are called* arcs.

For an arc $(u, v) \in A$ the vertex u is its *startpoint* and the vertex v its *endpoint*. We say (u, v) is an arc from u to v; it is incoming to v and outgoing from u. For an arc (u, v) we call the arc (v, u) the *reversed* arc. We apply Definitions 2.3 to 2.6 to directed graphs by essentially replacing the term *edge* with *arc*.

In many application domains, the vertices and edges of a graph need to be distinguished.

Definition 2.3 (Labeled Graph). *A* labeled *graph* (V, E, l) *is a graph* (V, E) *with a function* $l : V \cup E \to \Sigma$, *assigning labels of an alphabet* Σ *to its vertices and edges.*

It is possible to use different alphabets for edge and vertex labels, respectively. For simplification, and since this is not a restriction, we use a single alphabet Σ for both. Depending on the application, labeled graphs may have labels on their vertices or edges only. In this case, we use a single label for all vertices or all edges, respectively.

Definition 2.4 (Weighted Graph). *A* weighted *graph* (V, E, w) *is a graph endowed with a function* $w : E \to \mathbb{F}$, *where* $\mathbb{F}$ *is a totally ordered field of numbers.*

$\mathbb{F}$ typically is the set of natural or real numbers. This choice influences the running time to solve certain problems, e.g., when computing a matching of maximum weight, cf. Section 3.2.

Definition 2.5 (Bipartite Graph). *If the vertices of a graph* G *can be partitioned into two disjoint sets* U *and* V *such that each edge* $e \in E(G)$ *contains exactly one vertex of* V, *then the graph is called* bipartite.

In many cases, the disjoint sets are given as part of the input. In this case, we write $G = (U \uplus V, E)$, where $|e \cap U| = |e \cap V| = 1$ for each edge $e \in E$. G is *unbalanced*, if $|U| \neq |V|$.

Definition 2.6 (Subgraph). *Let $G' = (V', E')$ and $G = (V, E)$ be any two graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a* subgraph *of G, written $G' \subseteq G$.*

If additionally $E' = [V']^2 \cap E$, then G' is an *induced* subgraph of G; We say G' is induced by V' and refer to that subgraph by $G[V']$. For a subset of vertices $W \subseteq V$ we define $G \setminus W \coloneqq G[V \setminus W]$, i.e., the induced subgraph of G obtained by deleting the vertices W. For a subset $F \subseteq E$ of edges we define $G \setminus F \coloneqq (V, E \setminus F)$, i.e., the subgraph of G obtained by deleting the edges F. By definition, $G \setminus F$ is not an induced subgraph of G, if $F \neq \emptyset$.

Definition 2.7 (Walk, Path, Cycle). *A* walk *is a nonempty sequence of vertices connected through edges (or arcs) $(v_0, e_1, v_1, \ldots, e_l, v_l)$, where $e_i = \{v_{i-1}, v_i\}$ (or $e_i = (v_{i-1}, v_i)$), $i \in \{1, \ldots, l\}$. A walk is a* cycle *if $v_0 = v_l$ and $l \geq 2$. A cycle is* simple *if all vertices but v_0 and v_l are pairwise disjoint. A walk is a* path *if all vertices are pairwise disjoint.*

In a walk, we may alternatively specify the vertices $(v_0, \ldots, v_l)$ or edges $(e_1, \ldots, e_l)$ only. The *length* of a walk is its number of edges l.

Definition 2.8 (Connected Graph). *A graph G is* connected *if there is a path between any two vertices of G.*

The maximal connected subgraphs of a graph are called *connected components*.

Definition 2.9 (Complete Graph). *If all vertices in a graph G are pairwise adjacent, G is* complete.

We denote the complete graph with n vertices by K_n. A bipartite graph $(U \uplus V, E)$ is *complete bipartite* if each vertex in U is adjacent to each vertex in V. We denote such a graph by $K_{m,n}$, where $m = |U|$ and $n = |V|$.

Definition 2.10 (Planar Graph, Outerplanar Graph, Face). *A graph is* planar *if it admits a drawing on the plane such that no two edges cross. The connected regions of the drawing enclosed by the edges are called* faces*, and the unbounded region is referred to as the* outer face*. A graph is called* outerplanar *if it admits a drawing on the plane without crossings, in which every vertex lies on the boundary of the outer face.*

An edge (a vertex) and a face are said to be *incident* if the edge (the vertex) touches the face. Two faces are *adjacent* if they are incident with a common edge.

Definition 2.11 (Tree). *A connected graph containing no cycle is a* tree.

For each tree (V, E), the equation $|V| = |E| + 1$ holds. A *subtree* of a tree T is a connected subgraph of T. An *isomorphism* between graphs G and H is a bijective function $\phi : V_G \to V_H$ such that $uv \in E_G \Leftrightarrow \phi(u)\phi(v) \in E_H$. A *maximum common subtree isomorphism* is an isomorphism between subtrees of two trees with the maximum possible number of vertices. For more details and its problem variants, see Chapter 4.

Definition 2.12 (Rooted Tree). *A tree $T = (V, E)$ with an explicit root vertex $r \in V$ is called a* rooted *tree, denoted by T^r.*

We refer to the root vertex of a rooted tree T^r by $r(T)$. The *depth* of a vertex u in a rooted tree T^r is the length of the unique path from r to u and denoted by $\text{depth}(u)$. Let $e = uv$ be an edge of T^r and $\text{depth}(v) = \text{depth}(u) + 1$. Then v is a *child* of u, and u is the unique *parent* of v. We denote the set of children of a vertex v by $C(v)$ and its parent by $p(v)$, where $p(r) = r$. A rooted tree is binary if there are at most two children per vertex. The *descendants* of a vertex u are the vertex u itself, its children, and recursively all their children. A rooted tree is *ordered* if there is an ordering to each vertex's children. For example, in a binary tree, we may have an explicit left and right vertex. Otherwise, it is *unordered*. In the following, we assume rooted trees to be unordered unless explicitly stated otherwise.

Definition 2.13 (Rooted Subtree). *Let T be a tree. For any two vertices $u, v \in V(T)$, the* rooted subtree T_v^u *is induced by the vertex v and its descendants related to the tree T^u.*

We refer to v as the root of the rooted subtree T_v^u. If the root r of T is clear from the context, we may abbreviate $T_v \coloneqq T_v^r$. For any vertex $v \in V(T)$, the rooted subtree T_v^v is identical to the rooted tree T^v. Figure 4.2 in Chapter 4 illustrates two rooted subtrees.

Definition 2.14 (Rooted Graph). *A graph G with an explicit root vertex $r \in V(G)$ is a* rooted graph.

Note, contrary to trees, we do not have a child/parent relationship.

2.3 BC-Trees

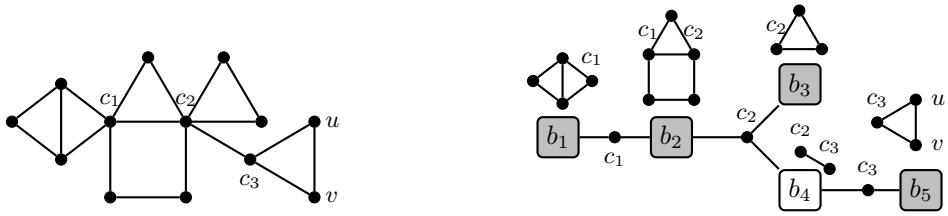
In this subsection, we define the term *BC-tree* and related structures.

Definition 2.15 (Biconnected Graph). *A graph $G = (V, E)$ with $|V| \geq 3$ is called* biconnected *if $G \setminus \{v\}$ is connected for each $v \in V$.*

Commonly the K_2 is also called biconnected. However, requiring $|V| \geq 3$ allows for an easy distinction between *blocks* and *bridges* through the term biconnected.

Definition 2.16 (Block, Bridge, Cutvertex). *A maximal biconnected subgraph of a graph G is called* block*. If an edge $uv \in E(G)$ is not contained in any block of G, the subgraph $(\{u, v\}, \{uv\})$ is called a* bridge*. A vertex v of G is called* cutvertex*, if $G \setminus \{v\}$ consists of more connected components than G.*

We occasionally refer to a bridge by its only edge or its two vertices. For a connected graph G, let C^G denote the set of cutvertices, Bl^G the set of blocks, Br^G the set of bridges, and let $\mathrm{B}^G = \mathrm{Bl}^G \cup \mathrm{Br}^G$. The decomposition of a graph into its blocks, bridges, and cutvertices was first introduced by Harary [53]. This data structure is known as block-cut tree, in short BC-tree. It is used to, e.g., represent chemical structures [94].



(a) Input graph G (b) BC-tree BC^G

Figure 2.1: A connected graph G (a) and its BC-tree BC^G (b). Block nodes have a gray background, while bridge nodes are not filled. The solid black nodes are C-nodes, which represent the cutvertices in G. The corresponding subgraphs of G are shown above the block and bridge nodes.

Definition 2.17 (BC-tree). *The* BC-tree BC^G *of* G *is the tree with nodes* $\mathrm{B}^G \cup \mathrm{C}^G$ *and edges between nodes* $b \in \mathrm{B}^G$ *and* $c \in \mathrm{C}^G$ *if and only if* $c \in V(b)$.

We refer to the vertices of the BC-tree as B- and C-nodes. We use the term *nodes* over *vertices* to easily distinguish them from the vertices of G. Within the B-nodes, we distinguish block nodes from bridge nodes. The C-nodes C^G represent the cutvertices in G. An example of a graph G and its BC-tree BC^G is shown in Figure 2.1. We denote the maximum degree of all C-nodes in the BC-tree of G by $\Delta_C(G)$. For any graph G, we define $\mathrm{CC}(V, U)$ as the connected component of $G[V]$ that includes at least one vertex of U. We allow only such sets U, where the component is unambiguous. For example, in Figure 2.1, $\mathrm{CC}(V_G \setminus V_{b_2}, V_{b_4})$ is the graph $G[\{c_3, u, v\}]$. We use this notation only in conjunction with BC-trees. The BC-tree of a graph can be computed in linear time using depth-first search as shown by Gutwenger and Mutzel [51].

2.4 Enumeration

For several problems, like the maximum cardinality matching problem, an optimal solution is not unique. Consequently, it might be of interest to find all optimal solutions.

Definition 2.18 (Enumeration problem). *Given a problem, return exactly once each solution that satisfies a given property.*

For example, the enumeration problem corresponding to the maximum cardinality matching problem is to return all maximum cardinality matchings in a graph. Depending on the problem, the number of solutions of an enumeration problem can be exponential or even super-exponential in the input size. In this case, the total running time cannot be expected to be polynomial. To overcome this, Johnson et al. [61] introduced classes of *output-sensitive algorithms*. Here, the running time is given in a polynomial in the output size, specifically in the given problem's number of solutions. We refer to that number by α. Johnson et al. introduced the following three classes belonging to a hierarchy.

Polynomial total time. The most general class in that hierarchy is the class of algorithms, where the total time to output all solutions is a polynomial in the input size n and the number of solutions α. More specific, there are constants a, b, such that the total running time is bounded

by $\mathcal{O}(n^a \alpha^b)$. Not for all problems such algorithms are known. For example, the satisfiability problem has no polynomial total time enumeration algorithm unless $\mathsf{P} = \mathsf{NP}$.

Incremental total time. In this class, the output is required to be given within polynomial time in the input size n and the number of previously given solutions i. More specifically, there are constants a and b such that the total running time is bounded by $\mathcal{O}(n^a i^b)$. In this class, the first output is required to be given in polynomial time of the input size, whereas in the class polynomial total time, all outputs could be given after completing the computation.

Polynomial delay. Enumeration algorithms have *polynomial delay* if the running time before the output of the first solution, and after the output of a solution until providing the next solution or halting, is polynomially bounded by $\mathcal{O}(n^a)$, where a is a constant and n the input size. In a polynomial delay algorithm, the pre- and post-processing costs are occasionally given separately. Then we have constants a and b and a total running time of $\mathcal{O}(n^b + \alpha n^a)$, where $\mathcal{O}(n^a)$ is the delay between any two subsequent outputs, and n^b is the time until the first output and from the last solution until halting.

We present polynomial delay algorithms for the maximum weight matching problem in bipartite graphs in Section 3.6, for the maximum common subtree isomorphism problem in Section 4.6, and for the block-and-bridge preserving maximum common induced subgraph problem in Section 5.5.

Enumeration Tree. Enumeration algorithms often compute the solutions recursively. In each recursive step, the problem is divided into two or more subproblems, such that the enumeration of all solutions of all the subproblems equals the solutions of the main problem. This can be interpreted as a *recursion tree* with the parent node as the current step and the children nodes as the subproblems. For the enumeration tree, we use the term *nodes* over *vertices* to distinguish them from our primary graph structures' vertices.

Since the problem size tends to get smaller in each recursive step, it is often possible to distribute the higher computational costs of the parent nodes to the children in an amortized time analysis. In some of these algorithms, solutions are outputted in the inner nodes, e.g., Uno's first enumeration algorithm for perfect matchings [125] or when enumerating cliques [10]. For the latter, however, no polynomial total time algorithm is known. Others output precisely at the recursion tree's leaves, e.g., Uno's second enumeration algorithm for perfect matchings [124]. The polynomial total time to output all solutions is often much smaller than the polynomial delay multiplied by the number of solutions α. This holds, e.g., for the perfect matching enumeration algorithms by Uno and for our algorithms to enumerate maximum common subtree isomorphisms.

To obtain a low polynomial delay, it is advantageous to design an enumeration algorithm so that outputs occur regularly and the enumeration tree is not degenerated. Strategies are to avoid inner nodes of degree 1, a balanced distribution of solutions between the children of an enumeration node, and the simplification of the problem by reducing the problem size in nodes closer to the leaves. More details are provided in Section 2 of [124].

2.5 Notation

The notation and symbols are stated in Table 2.1. We list frequently used abbreviations in Table 2.2.

Table 2.1: Notation and symbols

Symbol	Meaning
$\mathbb{N}$	natural numbers including 0
$\mathbb{Z}$	integral numbers
$\mathbb{R}$, $\mathbb{R}^{\geq 0}$	real numbers, non-negative real numbers
$U \uplus V$	disjoint union of sets U and V
$V(G), V_G$	vertex set of a graph G
$\lvert G\rvert$	order of a graph G, $\lvert G\rvert \coloneqq \lvert V_G\rvert$
$E(G), E_G$	edge set of a graph G
$G[V]$	subgraph induced by V
$(U \uplus V, E)$	bipartite graph with disjoint vertex sets U and V
T^r, $r(T^r) = r$	rooted tree with root r, root of a rooted tree
T^u_v	rooted subtree
$N(v)$	set of neighbors of a vertex v in a graph
$\delta(v)$	degree of v, $\delta(v) \coloneqq \lvert N(v)\rvert$
$\Delta(G)$	degree of G, $\delta(G) \coloneqq \max\{\delta(v) \mid v \in V(G)\}$
$C(v)$	set of children of a vertex v in a rooted tree
$p(v)$	parent vertex of v in a rooted tree T^r, where $p(r) = r$
$\mathcal{M}(v)$	mate of a vertex v of a matched edge uv in a matching $\mathcal{M}$; $\mathcal{M}(v) = u$
cc_i	ith connected component in a graph (ordered arbitrarily)
$\mathrm{CC}(V', U)$	connected component of $G[V']$, that includes at least one vertex of U
$E_1 \oplus E_2$	$\{e \mid e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1)\}$
K_n, $K_{m,n}$	complete and complete bipartite graph
$\mathrm{dom}(f)$	domain of a function $f : A \to B$, $\mathrm{dom}(f) = A$
α	number of solutions in an enumeration algorithm
BC^G	BC-tree (block-cut tree) of a graph G
Bl^G, Br^G	blocks, bridges of BC^G
$B^G \coloneqq \mathrm{Bl}^G \cup \mathrm{Br}^G$	B-nodes in BC^G
C^G	C-nodes in BC^G
$\Delta_C(G)$	maximum degree of all C-nodes in BC^G
$\mathcal{W}(\phi)$	weight of an isomorphism ϕ

Table 2.2: Abbreviations

Abbreviation	Meaning
MWM	maximum weight matching
MWPM	maximum weight perfect matching
MCS	maximum common subgraph
MCSI	maximum common subtree isomorphism
MWCSI	maximum weight common subtree isomorphism
LaCSE	largest common subtree embedding
LaWeCSE	largest weight common subtree embedding
BBP	block-and-bridge preserving
BBP-MCIS	block-and-bridge preserving maximum common induced subgraph
BBP-MCES	block-and-bridge preserving maximum common edge subgraph

Moral ist Zuordnung jedes Augenblicks
unseres Lebens zu einem Dauerzustand!

Robert Musil
1880 – 1942

Chapter 3

Matchings

Matching problems belong to the most prominent combinatorial optimization problems in graphs solvable in polynomial time. In addition to the wide range of direct applications, e.g., assignment problems or steganography [101, 77], matching problems often appear as subtasks of other practically relevant optimization problems, e.g., in the Christofides heuristic for the Traveling Salesperson Problem. In this chapter, we introduce the basic matching problem and the variants relevant to this thesis. The most basic form is as follows.

Definition 3.1 (Matching). *Let $G = (V, E)$ be a graph. A* matching $\mathcal{M} \subseteq E$ *is a set of edges, such that no two edges share a vertex, i.e., $e \cap e' = \emptyset$ for all $e, e' \in \mathcal{M}$, $e \neq e'$.*

The size of a matching is its number of edges $|\mathcal{M}|$. For a matching $\mathcal{M}$ and a graph G, we say an edge $e \in \mathcal{M}$ is *matched by* $\mathcal{M}$; otherwise, i.e., $e \in E \setminus \mathcal{M}$, e is *$\mathcal{M}$-free*. A vertex v incident to an edge $e \in \mathcal{M}$ is *matched by* $\mathcal{M}$; otherwise v is *$\mathcal{M}$-free*. We omit the reference to $\mathcal{M}$, if the matching is clear from the context. For an edge $uv \in \mathcal{M}$, the vertex $\mathcal{M}(v) \coloneqq u$ is the *mate* of v and vice versa. A matching $\mathcal{M}$ is *maximal* if there is no other matching $\mathcal{M}' \supsetneq M$. The following problem is generally referred to as *the matching problem*.

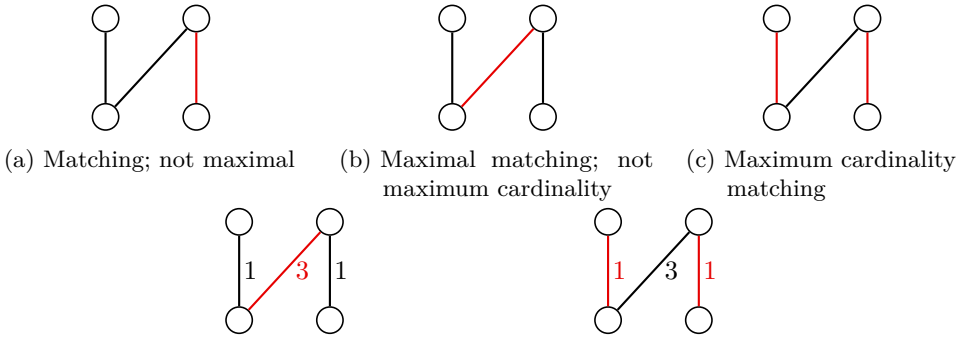
Definition 3.2 (Maximum Cardinality Matching). *A* maximum cardinality matching *is a matching of maximum cardinality, i.e., with the largest possible number of edges.*

In the literature, the term *maximum matching* is as common as the term *maximum cardinality matching*, and both refer to the same problem. We will use the latter to differentiate it from the maximum weight matching, cf. Definition 3.6.

Some graphs admit a *perfect matching*.

Definition 3.3 (Perfect Matching). *A matching $\mathcal{M}$ in a graph $G = (V, E)$ is* perfect *if $|V| = 2|\mathcal{M}|$, i.e., there is no free vertex in G.*

A simple greedy approach allows to find a maximal matching in time $\mathcal{O}(|E|)$. The size of any maximal matching $\mathcal{M}$ is at least $\mathcal{M}'/2$, where $\mathcal{M}'$ is a maximum cardinality matching. Given a matching $\mathcal{M}$, we can iteratively increase the number of matching edges by applying so-called *$\mathcal{M}$-augmenting paths*.



(a) Matching; not maximal (b) Maximal matching; not maximum cardinality (c) Maximum cardinality matching

(d) Maximum weight matching (MWM) (e) Maximum weight perfect matching (MWPM)

Figure 3.1: Different types of matchings. Matchings in red.

Definition 3.4 ($\mathcal{M}$-Alternating Walk, Path, Cycle, Tree; $\mathcal{M}$-Augmenting Path). *Let $\mathcal{M}$ be a matching. A walk, path, or cycle P is* $\mathcal{M}$-alternating *if the sequence of edges is alternating between free edges and edges matched by $\mathcal{M}$. An $\mathcal{M}$-alternating path $P = (v_0, e_1, \ldots, e_l, v_l)$ is* $\mathcal{M}$-augmenting, *if v_0 and v_l are free. An $\mathcal{M}$-alternating tree T is a tree with an $\mathcal{M}$-free root r, where each path from r to any vertex $v \in V(T)$ is $\mathcal{M}$-alternating.*

Each $\mathcal{M}$-augmenting path P is of odd length. Exchanging matched and free edges in P increases the size of $\mathcal{M}$ by 1. We write $\mathcal{M} \oplus P$ for this exchange; formally $\mathcal{M} \oplus P \coloneqq \{e \mid e \in (\mathcal{M} \setminus P) \cup (P \setminus \mathcal{M})\}$. Repeatedly augmenting a matching allows finding a maximum cardinality matching, as proven by Berge [7].

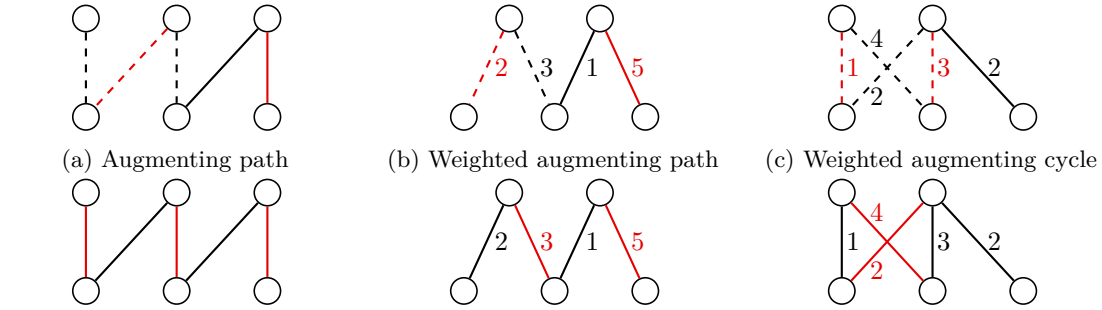
Lemma 3.5 (Berge's Lemma). *A matching $\mathcal{M}$ in a graph G is of maximum cardinality if and only if there is no $\mathcal{M}$-augmenting path.*

If the input graph is bipartite, we can find a maximum cardinality matching with, e.g., the famous algorithm by Hopcroft and Karp [57]. Its running time is $\mathcal{O}(m\sqrt{n})$. This algorithm computes maximal sets of disjoint shortest augmenting paths to increase the size of the matching iteratively. Feder and Motwani [36] proved a running time of $\mathcal{O}(m\sqrt{n}\log(n^2/m)/\log n)$ for the same problem. For dense graphs, this is slightly faster than the algorithm by Hopcroft and Karp.

When considering weighted graphs, we can extend the matching problem to a matching of maximum or minimum weight. The *weight* of a matching $\mathcal{M}$ in a weighted graph $G = (V, E, w)$ is $W(M) \coloneqq \sum_{e \in \mathcal{M}} w(e)$.

Definition 3.6 (Maximum Weight Matching). *Let $G = (V, E, w)$ be a weighted graph. A matching $\mathcal{M}$ of G is a* maximum weight matching (MWM) *if there is no other matching $\mathcal{M}'$ of G with $W(\mathcal{M}') > W(\mathcal{M})$.*

Closely related is the *maximum weight perfect matching* (MWPM) problem. Here, we are interested in a maximum weight matching among all perfect matchings. A *minimum weight matching*, i.e., a matching of minimum weight among all matchings, is also known as *minimum cost matching*. The problem of finding a minimum cost *perfect* matching on a bipartite graph is known as the *assignment problem*. The maximum and minimum weight (perfect) matching



(a) Augmenting path (b) Weighted augmenting path (c) Weighted augmenting cycle

(d) Resulting matching from (a) (e) Resulting matching from (b) (f) Resulting matching from (c)

Figure 3.2: Augmenting paths and cycles (dashed); matchings in red. (a), (b), and (c) depict a matching before augmentation; (d), (e), and (f) depict the resulting matchings.

problems are reducible to each other by multiplying the weights by -1. Consequently, any algorithm for the minimum weight (perfect) matching problem is also applicable to the maximum weight (perfect) matching problem. The different matching types are exemplified in Figure 3.1.

Definition 3.7 (Maximum Weight Matching of Cardinality c). *A matching $\mathcal{M}$ in G is a* maximum weight matching of cardinality c (MWM_c) *if there is no other matching $\mathcal{M}'$ of cardinality c in G with $W(\mathcal{M}') > W(\mathcal{M})$.*

By definition, a maximum weight perfect matching is a special case of an MWM_c, where $c = |V(G)|/2$. An MWM_1 consists of a single edge of the largest weight. In an unbalanced bipartite graph $G = (U \uplus V, E, w)$ with $s := |U| < |V|$, an MWM_s is called *unbalanced maximum weight perfect matching*. The problem to compute a matching of minimum weight among all matchings of cardinality s is also known as the *unbalanced assignment problem* [103].

Definition 3.8 (Augmenting Paths and Cycles in Weighted Graphs). *An $\mathcal{M}$-alternating cycle or path $P = (v_0, e_1, \ldots, e_l, v_l)$ in a weighted graph is* $\mathcal{M}$-augmenting *if the following conditions hold.*

1. *$\mathcal{M} \oplus P$ is a matching.*
2. *$W(\mathcal{M} \oplus P) > W(\mathcal{M})$*

The first condition ensures we actually may exchange the free and matched edges. In both cases (path, cycle), augmenting the matching increases its weight. If we seek a minimum weight matching, an augmenting path or cycle is assumed to decrease the matching's weight. Figure 3.2 illustrates augmenting paths and cycles and the resulting matchings.

The remainder of this chapter is as follows. We present challenges and results for the maximum cardinality matching problem in general graphs in Section 3.1. In Section 3.2, we present known algorithms for the maximum weight matching problem in bipartite graphs. We study the problem on unbalanced bipartite graphs in Section 3.3. The algorithms in Chapter 4 and 5 partly rely on a dynamic programming approach. More specific, solutions from subproblems are combined depending on maximum weight matchings in corresponding bipartite graphs. We can divide all the graphs on which we need to compute those matchings into sets of similar graphs. In each of these sets, there is a weighted graph G and additional graphs $G \setminus \{v\}$ for specific vertices $v \in V(G)$, i.e., in the additional graphs of the set, there is precisely one vertex missing from G.

Given a weighted graph G, the task to compute a maximum weight matching on $G \setminus \{v\}$ for each $v \in V(G)$ is known as the *all-cavity maximum weight matching* problem [65]. One approach to solving the problem is to compute an MWM $\mathcal{M}$ on G and then derive an MWM for each of the graphs $G \setminus \{v\}$ in one step, e.g., by solving a single-source shortest paths problem on a digraph constructed from G and $\mathcal{M}$. This is usually faster than computing each matching individually. We study the all-cavity maximum weight matching for balanced graphs in Section 3.4, followed by known and new results on unbalanced graphs in Section 3.5. In Section 3.6, we present a polynomial delay algorithm to enumerate all maximum weight matchings in a bipartite graph. Finally, we discuss open problems and conclude in Section 3.7.

3.1 Maximum Cardinality Matching in General Graphs [1]

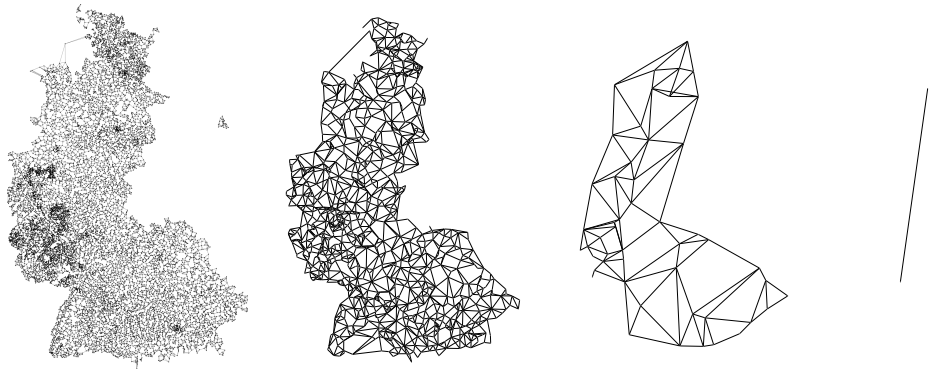
In this section we consider the maximum cardinality matching problem in general (not necessarily bipartite) graphs. While computing matchings in bipartite graphs is easy and can be solved using alternating trees, the problem is quite complex on general graphs. The first polynomial-time algorithm has been provided by Edmonds in his pioneering work titled *Paths, trees, and flowers* [32]. The breakthrough came with his idea to shrink so-called *blossoms*, i.e., odd cycles encountered when growing alternating trees. The algorithm works in rounds. In every round, an alternating tree is constructed by either adding edges or shrinking blossoms until an augmenting path is found. In this case, the current matching is augmented along the path leading to a new matching of size enlarged by one. For the augmentation step, the blossoms along the path need to be expanded. This shrinking and unshrinking procedure leads to a quite complicated algorithm. The algorithm can be implemented in time $\mathcal{O}(n^2m)$ for a graph with n vertices and m edges. Later it was shown that explicit shrinking of odd cycles could be avoided, and the running time can be improved to $\mathcal{O}(n^3)$, e.g., [131, 3, 62, 41, 77].

Unfortunately, the breakthrough by Hopcroft and Karp [57] to find a maximal set of disjoint augmenting paths of shortest length in each round applies to bipartite graphs only. However, Even and Kariv [35] presented a first strong result for general graphs leading to a running time of $\mathcal{O}(\min\{n^{2.5}, \sqrt{n}m \log n\})$. Finally, Micali and Vazirani [91] suggested an algorithm in which they find a maximal set of disjoint augmenting paths of shortest length in each round in time $\mathcal{O}(m)$. This approach is quite complicated, and also the proofs are not easy to understand. This is revealed because Vazirani suggested corrections of the proofs, the running time analysis, and also simplifications of the algorithm [128, 129]. Gabow and Tarjan [43, 42] suggested a data structure that can be used to achieve the running time of $\mathcal{O}(\sqrt{n}m)$.

In the following, we summarize our new techniques to solve the maximum cardinality matching problem on general graphs [29]. We introduced *alternating rooted sets (ARSs)* as a generalization for the state-of-the-art *alternating trees* to find augmenting paths. We revealed how they could grow simultaneously beyond the first contact to create maximal sets of augmenting paths. This is realized by a new data structure, which we call *cherry tree*. Cherry trees act similar to alternating trees but do not require blossom shrinking. These may even be re-used after augmentation steps, contrary to alternating trees. Virtually shrinking those trees induces a metagraph in which any matching corresponds to a set of disjoint augmenting paths in the underlying graph. The metagraph approach is applied recursively. This reduces the problem to a trivial case. Figure 3.3 exemplifies this strategy.

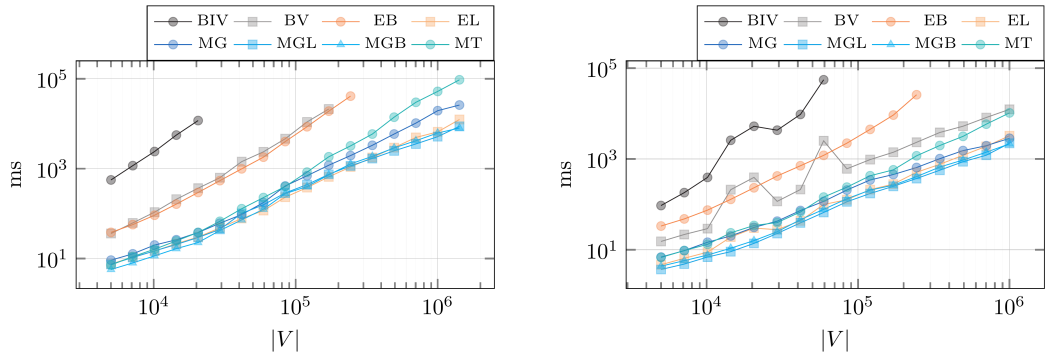
We experimentally evaluated the new metagraph approach on a broad set of benchmark instances, including publically available state-of-the-art software. This includes implementations of Edmonds' algorithm in the LEMON [22, 78] and Boost [9] graph libraries. We further compared

[1] This section consists of and summarizes our findings in *Shrinking Trees not Blossoms: A Recursive Maximum Matching Approach* [29] with Erik Thordsen as the main author.



(a) The base graph (b) First metagraph (c) Second metagraph (d) Third metagraph

Figure 3.3: Different layers of the recursive metagraph approach on the 4 nearest neighbor graph of the brd14051 TSPLib instance [107]. Vertices in the metagraph have the same coordinates as the root of their represented tree.



(a) RD-3 instances (b) TRI instances

Figure 3.4: Average running times for different maximum cardinality matching algorithms.

against Blossom IV [20] and Blossom V [69]. The Blossom algorithms solve the minimum weight perfect matching problem. We used the reduction from Section 3.2.2 to solve the (unweighted) maximum cardinality matching problem using Blossom IV and V. Average running time results are shown in Figure 3.4. Each test case was conducted between 30 times for graphs of at least 1 000 000 vertices and 250 times for graphs of at most 5 000 vertices. MG is the Metagraph approach; MGL and MGB are variants thereof. MT is a multiple cherry tree approach without the metagraphs. BIV and BV are Blossom IV and V. EB and EL are implementations of Edmonds' matching algorithm in the Boost and the LEMON framework. The RD-3 instances are fully randomized graphs with 3 times the number of edges to vertices generated with the random generator from the DIMACS challenge [24]. The TRI instances are Delaunay triangulations on randomized two-dimensional point clouds, generated with Fade2D [118]. For further details about cherry trees, the metagraph approach, and additional test results see [29].

Author	Problem	Running time
Hopcroft and Karp (1973)	Maximum cardinality matching	$\mathcal{O}(m\sqrt{n})$
Feder and Motwani (1991)		$\mathcal{O}(m\sqrt{n}\log(\frac{n^2}{m})/\log n)$
Duan and Su (2012)	MWM, integral	$\mathcal{O}(m\sqrt{n}\log N)$
Gabow and Tarjan (1989) Orlin and Ahuja (1992) Goldberg and Kennedy (1997)	MWPM, integral	$\mathcal{O}(m\sqrt{n}\log(nN))$
Fredman and Tarjan (1987)	MWM, MWPM	$\mathcal{O}(nm + n^2\log n)$

Table 3.1: Worst-case running times for different matching problems on a bipartite graph with n vertices, m edges, and N as maximum edge weight.

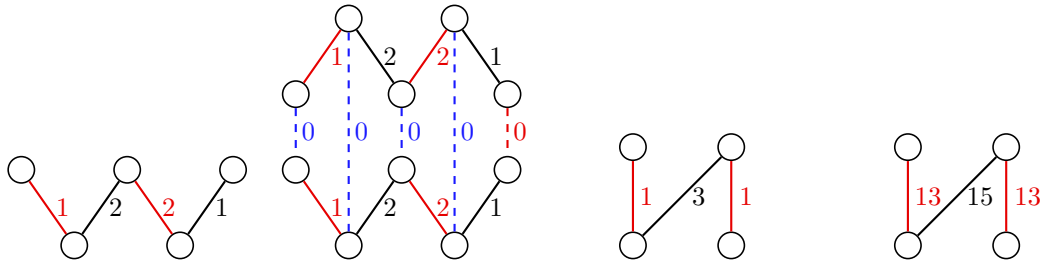
3.2 Maximum Weight Bipartite Matching

This section presents known results on the maximum weight matching problems on bipartite graphs and commonly used techniques to solve them. We consider the MWM, MWPM, and MWM_k problems. We further show the relation between these problems. For the remainder of this thesis, the graphs on which we need to compute matchings are bipartite. So unless stated otherwise, assume any reference to a matching problem relates to bipartite graphs. For the weighted matching problem on general graphs, we refer the reader to, e.g., [30, 59].

3.2.1 Recent Algorithms

Duan and Su [31] presented an overview of previous results on weighted matching problems on bipartite graphs with n vertices and m edges. Some results allow arbitrary weights, and others rely on integral weights of at most N. For the latter case, they presented algorithms for both the maximum weight and maximum weight perfect matching problem. While most algorithms in their overview are deterministic, a few are randomized. Randomized algorithms on integral weights allow running times of $\mathcal{O}(mn)$ [121] or $\mathcal{O}(Nn^\omega)$ [110], where ω is the matrix multiplication exponent. The well known Hungarian method is a deterministic algorithm and solves the MWPM problem in polynomial time, as first shown by Kuhn and Yaw [74]. Fredman and Tarjan [38] improved the running time to $\mathcal{O}(nm+n^2\log n)$. Their result is based on Fibonacci heaps as a priority queue. For integral weights, the Hungarian method allows a running time of $\mathcal{O}(nm + n^2\log\log n)$ [122].

Duan and Su [31] contributed a new algorithm for the MWM problem on integral weights. They improved the previously best-known upper bound of the MWM problem from $\mathcal{O}(m\sqrt{n}\log(nN))$ to $\mathcal{O}(m\sqrt{n}\log N)$. Applying this algorithm to unweighted graphs allows the same running time as the Hopcroft and Karp algorithm [57] for maximum cardinality matchings, which is $\mathcal{O}(m\sqrt{n})$. The MWPM problem is solvable in time $\mathcal{O}(m\sqrt{n}\log(nN))$. This result has been proven by different researchers, e.g., Gabow and Tarjan [44], Goldberg and Kennedy [49], and Orlin and Ahuja [98]. For dense graphs, a slightly improved time bound of $\mathcal{O}(n^{2.5}\log(nN)(\frac{\log\log n}{\log n})^{0.25})$ has been shown [14]. To undercut Duan and Su's running time of $\mathcal{O}(m\sqrt{n}\log N)$ requires $N \in \mathcal{O}(1)$ and $m \in \omega(n^{2-\epsilon})$ for any $\epsilon > 0$ [31]. On graphs with integral weights, it is unknown if the MWPM problem is solvable in the same time bound as the MWM problem. An overview of these results is presented in Table 3.1.



(a) From MWM ... (b) ... to MWPM (c) From MWPM ... (d) ... to MWM

Figure 3.5: Reduction between maximum weight matching and maximum weight perfect matching. (a) Graph including a MWM in red; (b) Reduction to MWPM, copied graph connected through weight 0 edges (dashed), MWPM in red; (c) Graph including a MWPM in red; (d) Reduction to MWM, edge weights increased by $nN = 4 \cdot 3 = 12$, MWM in red.

3.2.2 Reduction between Perfect and Non-Perfect Matchings

The MWM problem and the MWPM problem are reducible to each other, as shown, e.g., by Gabow and Tarjan [44]. To compute an MWM using an algorithm for MWPM, we copy the graph and connect each vertex with its copy by an edge of weight 0. The edges of an MWPM contained in the original graph (alternatively in the copy) are the edges of an MWM. To compute an MWPM with an algorithm for MWM, we add nN to each edge's weight, where n is the number of vertices and N the maximum edge weight (we use the absolute edge weights for N). Note, this can negatively influence the worst-case running time of an algorithm, if it depends on N. If the graph admits a perfect matching, then the MWM algorithm computes a perfect matching of maximum weight in the original graph. Examples for both reductions are depicted in Figure 3.5.

3.2.3 Solving Techniques

Duan and Su describe different approaches to solve the MWPM problem [31]. Firstly, in each iteration, the number of matched edges is increased by one until a perfect matching is computed. Then this matching is of maximum weight. Secondly, the so-called primal methods start with and maintain a perfect matching. They iterate by exchanging edges until the matching is of maximum weight.

For integral weights, weight scaling can be used. This technique is related to the fact that the MWPM problem is a special case of computing a minimum cost integral flow, where the capacity of each flow arc is 1 [102]. Instead of computing an exact solution, in each step, some error ε is allowed. In each iteration (scaling phase), this error is reduced until the computed flow is of minimum cost. The total running time depends on the number of scaling phases and the time needed for each phase. An example of a weight scaling algorithm is the one proposed by Duan and Su [31].

3.2.4 Matching as Integer Linear Program

We may formulate the MWM problem as an integer linear program (ILP). Let $G = (V, E, w)$ be a weighted graph and $x : E \to \{0, 1\}$ be the decision variables, i.e., $x(e)$ indicates if e is contained

in the matching.

$$\max \sum_{e \in E} w(e)x(e) \tag{3.1}$$

$$\text{subject to } x(e) \in \{0,1\} \qquad \forall e \in E \tag{3.2}$$

$$\sum_{uv \in E} x(uv) \leq 1 \qquad \forall u \in V \tag{3.3}$$

The objective function (3.1) maximizes the weight. Condition (3.2) ensures each edge is either matched or free. The last condition, (3.3), ensures each vertex is incident to at most one matched edge. Thus any feasible solution is a matching. If we relax condition (3.2) to

$$0 \leq x(e) \leq 1 \qquad \forall e \in E, \tag{3.4}$$

we obtain the corresponding linear program (LP). Optimal solutions of both the LP and ILP have the same objective function value [56]. We can transform any fractional solution into an integral solution by adjusting the x-values along even length cycles, as shown by Gärtner and Matousek [46].

The dual program (DP) of the above LP is as follows.

$$\min \sum_{u \in V} y(u) \tag{3.5}$$

$$\text{subject to } y(u) + y(v) \geq w(uv) \qquad \forall uv \in E \tag{3.6}$$

$$y(u) \geq 0 \qquad \forall u \in V \tag{3.7}$$

If we are interested in a maximum weight perfect matching, we replace condition (3.3) of the LP by equality, i.e.,

$$\sum_{uv \in E} x(uv) = 1 \qquad \forall u \in V. \tag{3.8}$$

Then the last inequality in the DP, (3.7), is dropped. We refer to the DP of the maximum weight perfect matching problem by *DP:MWPM*.

For a solution of a linear program, we call an inequality *binding* if equality is given. For example, an inequality from Constraint (3.6) is binding if $w(uv) = y(u) + y(v)$ for an edge $uv \in E$. Note, for each primal variable in a linear program there is a corresponding constraint in its dual program. For the jth variable in the linear program let the jth constraint in the dual program be the corresponding constraint. This allows expressing the well known complementary slackness condition easily.

Proposition 3.9 (Complementary slackness). *Let (x_m) be a solution to a primal linear program and (y_n) be a solution to its dual program. Then x and y are optimal if and only if*

1. *$x_j = 0$ or the constraint j in the dual is binding, for each $j \in \{1, \ldots, m\}$*
2. *$y_i = 0$ or the constraint i in the primal is binding, for each $i \in \{1, \ldots, n\}$*

From the above proposition, we obtain the following result for the maximum weight perfect matching problem.

Corollary 3.10. *Let y be a feasible solution to DP:MWPM. If and only if for each edge $uv \in \mathcal{M}$ of a perfect matching $\mathcal{M}$, $w(uv) = y(u) + y(v)$, then $\mathcal{M}$ is of maximum weight and y is optimal.*

Corollary 3.10 is a crucial argument in the Hungarian method to grow the current matching. The Hungarian method to compute an MWPM is a primal-dual algorithm, with its primal and dual linear program as above. In that method, a shortest-path tree or forest is grown from free vertices along those edges uv, where the duality constraint (3.6) is binding. If there is an augmenting path using only binding edges, this path allows increasing the number of matching edges by one while maintaining the dual solution feasible. If there is no such path, the dual values y of the current vertices in the search tree are adjusted, such that the edges in the current search tree (or forest) remain binding and new edges adjacent to the current search tree (forest) become binding while maintaining feasibility of constraint (3.6) for all other edges. At the end of the algorithm, the obtained perfect matching has maximum weight due to Corollary 3.10. A further application of Corollary 3.10 is the enumeration of maximum weight matchings in Section 3.6.

3.3 Maximum Weight Matching on Unbalanced Bipartite Graphs

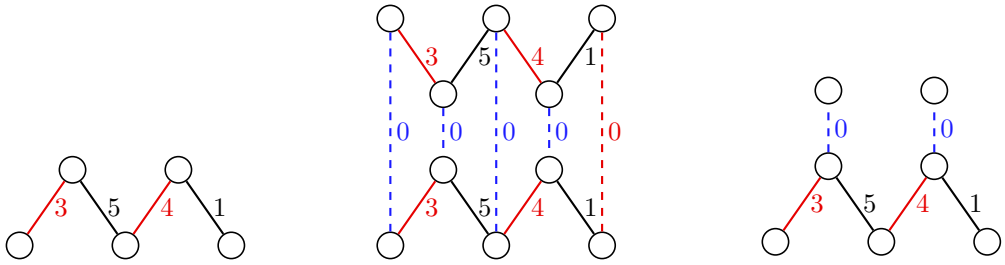
Most researchers do not consider the sizes of the sets U and V of a bipartite graph $(U \uplus V, E)$ separately. Consider the complete bipartite graph $K_{1,n}$ with $n+1$ vertices and n edges. It has a maximum cardinality matching of size 1. The algorithm by Hopcroft and Karp [57] will need a single iteration only. Thus we obtain such a maximum matching in time $\mathcal{O}(n)$. However, the worst-case time without considering unbalanced vertex sets is $\mathcal{O}(m\sqrt{n}) = \mathcal{O}(n\sqrt{n})$ in this case. Indeed, we give an improved upper bound for unbalanced graphs to Hopcroft and Karp's algorithm in Subsection 3.3.1. Following that, we consider weighted graphs. Firstly, with arbitrary edge weights in Subsection 3.3.2. Secondly, with integral weights bounded by N in Subsection 3.3.3. In Subsection 3.3.4, we provide results for finding a MWM_k for a $k \in \{1, \ldots, s\}$, where s is the size of the smaller vertex set in a bipartite graph. The last result in this section discusses a preprocessing step, which allows to upper bound the number of edges m by s^2. In the remainder of this chapter, we assume $n \in \mathcal{O}(m)$, i.e., not asymptotically fewer edges than vertices. Otherwise, we add the term $\mathcal{O}(n)$ to each algorithm's worst-case running time to identify and remove isolated vertices in a preprocessing step.

3.3.1 Maximum Cardinality Matching

In the following, we analyze the running time and space bound of the Hopcroft and Karp algorithm on an unbalanced bipartite graph $G = (U \uplus V, E)$. The algorithm uses breadth-first and depth-first search to compute an MWM. Therefore, the space bound is $\mathcal{O}(|E|)$. Let $\mathcal{M}$ be any maximum cardinality matching in G. In the running time analysis by Hopcroft and Karp, there are two stages, each with at most $\mathcal{O}(\sqrt{|\mathcal{M}|})$ iterations. Since $|\mathcal{M}| \leq \min\{|U|, |V|\}$ and each iteration takes time $\mathcal{O}(|E|)$, we obtain the following result.

Proposition 3.11. *The algorithm by Hopcroft and Karp computes a maximum cardinality matching in a bipartite graph $(U \uplus V, E)$ in time $\mathcal{O}(m\sqrt{s})$ and space $\mathcal{O}(m)$, where $m = |E|$ and $s = \min\{|U|, |V|\}$.*

A more detailed analysis is available in [102], Section 5.2.



(a) G (b) G' (c) $\widetilde{G}$

Figure 3.6: Different reductions from maximum weight matching to maximum weight perfect matching. (a) Graph G including an MWM in red; (b) Standard reduction with graph doubling, MWPM in G' in red; (c) Reduction by Ramshaw and Tarjan [102], one-sided MWPM in $\widetilde{G}$ in red.

3.3.2 Arbitrary Weights [2]

To compute a maximum weight matching on an unbalanced bipartite graph $G = (U \uplus V, E, w)$ with edge weights $w : E \to \mathbb{R}$, we may use the reduction from Subsection 3.2.2 and then use the Hungarian method to solve the problem. Let $s := |U| \leq |V| =: t$. Without further consideration of the individual sizes of the vertex sets, the time bound to compute an MWM is $\mathcal{O}(nm + n^2 \log n)$.

To obtain an improved upper time bound, we first provide an initial matching of size t in time $\mathcal{O}(m)$, such that s further iterations of the Hungarian method are sufficient to obtain an MWPM in the reduced graph and thus an MWM in G. Then we show an upper time bound of $\mathcal{O}(m + t \log t)$ for each iteration. This yields a total time of $\mathcal{O}(ms + st \log t)$ to compute an MWM on G.

In the following, we show how to obtain an initial dual solution and matching for the Hungarian method, as presented in our work [26]. Let G' be the reduced graph according to Subsection 3.2.2. For each vertex $v \in U \uplus V$ of G, we denote its copy v^c, and for each edge $e \in E$, we denote its copy e^c. We set $y(v) := 0$ for all $v \in V$ and $y(u) := \max\{w(uv) \mid v \in V\}$ for all $u \in U$. The copied vertices obtain their values from the original vertices, i.e., $y(v^c) := y(v)$ for all $v \in V \uplus U$. We define an initial matching $\mathcal{M}' := \{vv^c \mid v \in V\}$. Note, $y(v) + y(v^c) = 0 = w(vv^c)$ for all $v \in V$. The dual solution y is feasible and can be computed in time $\mathcal{O}(m)$.

A single iteration of the Hungarian method is possible in time $\mathcal{O}(m + n \log n) = \mathcal{O}(m + t \log t)$. To obtain an MWPM $\mathcal{M}$ from $\mathcal{M}'$ we need to increase the number of matching edges by s. Therefore, the time to compute $\mathcal{M}'$ (which yields an MWM in G) is $\mathcal{O}(ms + st \log t)$.

The initial matching $\mathcal{M}'$ and the Hungarian method require $\mathcal{O}(m)$ space. We summarize this result in the following lemma.

> **Lemma 3.12 ([26]).** *Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $s = |U| \leq |V| = t$. An MWM $\mathcal{M}$ on G can be computed in time $\mathcal{O}(ms + st \log t)$ and space $\mathcal{O}(m)$.*

A strategy to compute an MWM without copying the input graph $G = (U \uplus V, E, w)$ was proposed by Ramshaw and Tarjan [102]. Let $s := |U| \leq |V| =: t$. One difference is that only the vertices in the smaller set U are copied. Then, an edge uu^c of weight 0 for each vertex $u \in U$ is added. We denote the resulting graph $\widetilde{G}$. Figure 3.6 illustrates the difference between the

[2] This subsection is partly based on our findings in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 4.

standard graph doubling technique and the method by Ramshaw and Tarjan. Note, $\widetilde{G}$ is even more unbalanced, and there exists no perfect matching in $\widetilde{G}$. Hence the Hungarian method is not directly applicable. However, the additional edges ensure there is a matching of cardinality s. When computing a one-sided MWPM with exactly s iterations of the Hungarian method as described by Ramshaw and Tarjan, the resulting matching restricted to the vertices in G is an MWM. We summarize this result.

Proposition 3.13 ([102]). *Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $s = |U| \leq |V|$. An MWM on G is computable in time $\mathcal{O}(ms + s^2 \log s)$ and space $\mathcal{O}(m)$.*

The above time bound without considering the number of edges is $\mathcal{O}(s^2 t)$. The authors warn that a shortest-path tree instead of a shortest-path forest may result in a matching that is not maximum. The same is true for local updates, which generally decrease the running time in practice.

When comparing Lemma 3.12 and Proposition 3.13, the latter seems strictly better. However, we can use the dual variables y from the graph doubling method to construct a so-called equality subgraph, on which we can enumerate perfect matchings, cf. Section 3.6. Further, for the all-cavity maximum weight matching problem (cf. Subsection 3.5.2), we have to rely on graph doubling to solve the problem with additional iterations of the Hungarian method. If we are interested in a single maximum solution, the proposed construction by Ramshaw and Tarjan [102] yields a better time bound.

3.3.3 Integral Weights

If the weights are integral and bounded by N, the following result was shown by Goldberg, Hed, Kaplan, and Tarjan.

Proposition 3.14 ([48]). *Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{Z}$ upper bounded by N. Let $s = |U| \leq |V|$. An MWM on G is computable in time $\mathcal{O}(m\sqrt{s} \log N)$.*

Their result considers the minimum weight matching problem. As stated, we solve the maximum weight matching problem by multiplying the weights by -1. For dense graphs, the running time is $\mathcal{O}(s^{1.5} t \log N)$. Unfortunately, there is no space bound given. However, since their algorithm is based on flows, the space bound is probably $\mathcal{O}(m)$. An overview of matching results on unbalanced bipartite graphs is presented in Table 3.2.

3.3.4 Maximum Weight Matching of Given Cardinality

Unbalanced bipartite graphs do not allow a perfect matching. However, one might be interested in a matching of maximum weight among all matchings of cardinality s, where s is the size of the smaller vertex set in that bipartite graph. For certain purposes, even smaller matchings are of interest. We need to compute maximum weight matchings of cardinality 2 as a subproblem in Chapter 4 to find a largest weight common subtree embedding, for example. Ramshaw and Tarjan [103] analyzed the problem to find a maximum weight matching of some cardinality c for integral and arbitrary weights. The result for arbitrary weights is an extension of the Hungarian method.

Author	Problem	Running time
Droschinsky (Proposition 3.11) Ramshaw and Tarjan (2012)	Maximum cardinality matching	$\mathcal{O}(m\sqrt{s})$
Goldberg et al. (2017)	MWM, integral	$\mathcal{O}(m\sqrt{s}\log N)$
Droschinsky et al. (2016) Ramshaw and Tarjan (2012)	MWM; applicable to enumeration MWM	$\mathcal{O}(ms + st\log t)$ $\mathcal{O}(ms + s^2\log s)$
Ramshaw and Tarjan (2012) Ramshaw and Tarjan (2012)	MWM_c, $c \in \{1,\dots,s\}$ MWM_c, integral, $c \in \{1,\dots,s\}$	$\mathcal{O}(mc + c^2\log s)$ $\mathcal{O}(m\sqrt{c}\log(cN))$

Table 3.2: Worst-case running times for different matching problems on unbalanced bipartite graphs $(U \uplus V, E)$, where $s := |U| \leq |V| =: t$, $m := |E|$, and N as maximum edge weight.

Lemma 3.15. *Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $s = |U| \leq |V|$. An MWM_c, $c \in \{1,\dots,s\}$ on G is computable in time $\mathcal{O}(mc + c^2\log s)$ and space $\mathcal{O}(m)$.*

The result for integral weights is from a weight scaling approach.

Lemma 3.16. *Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{Z}$ upper bounded by N. Let $s = |U| \leq |V|$. An MWM_c, $c \in \{1,\dots,s\}$ on G is computable in time $\mathcal{O}(m\sqrt{c}\log(cN))$.*

The results are included in Table 3.2.

3.3.5 Insignificant Edges in Unbalanced Bipartite Graphs

When computing a maximum weight matching, it is obvious we do not have to consider edges of negative weight. However, we can also delete additional edges, as stated in the following lemma.

Lemma 3.17. *Let $G = (U \uplus V, E, w)$ be a bipartite graph. Let $s = |U| \leq |V|$. Let G' be the subgraph of G, where for each vertex $v \in V$ all but the s edges of largest weight adjacent to v are removed. Let $\mathcal{M}$ be an MWM on G and $\mathcal{M}'$ be an MWM on G'. Then $W(\mathcal{M}) = W(\mathcal{M}')$.*

Proof. Let $\mathcal{M}$ be an MWM on G. Assume there exists an edge $uv \in \mathcal{M} \setminus E(G')$. Since the cardinality of $\mathcal{M}$ is at most s, there is at least one free edge $uv' \in E(G')$, such that $\mathcal{M}' := (\mathcal{M} \setminus \{uv\}) \cup \{uv'\}$ is a matching. In other words, we replace the matched edge uv by uv'. From the construction of G' follows $w(uv') \geq w(uv)$ and subsequently $W(\mathcal{M}') \geq W(\mathcal{M})$. We set $\mathcal{M} := \mathcal{M}'$ and repeat this process until there is no other edge $uv \in \mathcal{M} \setminus E(G')$. Since $\mathcal{M}' \subseteq E(G')$ and the weight of $\mathcal{M}'$ is at least that of the initial matching $\mathcal{M}$, the proof is done. □

Lemma 3.18. *The time to construct G' from G in Lemma 3.17 is $\mathcal{O}(|E|)$.*

Proof. First, for each vertex $v \in V$ we identify the edge which has the sth largest weight among all $|N(v)|$ edges adjacent to v, if $|N(v)| > s$; otherwise, we skip that vertex v. Note, edges of the same weight count individually and may be ordered arbitrarily. We can find such an edge e in time $\mathcal{O}(\delta(v))$ [8]. Secondly, we remove all edges uv adjacent to v, where $w(uv) < w(e)$. Lastly, if in the remaining graph $|N(v)| > s$, we arbitrarily remove edges uv of weight $w(e)$, until s edges adjacent to v remain. This may happen if several edges have weight $w(e)$. With reasonable data structures, e.g., adjacency lists for the vertices $v \in V$, the claimed time bound holds. □

As stated in Lemma 3.17, removing edges in a graph as a preprocessing step directly impacts any MWM algorithm on an unbalanced graph.

Corollary 3.19. *Applying Lemma 3.17 as a preprocessing step to any MWM algorithm on a bipartite graph $(U \uplus V, E, w)$ where $s = |U| \leq |V|$ allows to replace each occurrence of m in the associated running time by $\min\{m, s^2\}$ while adding $\mathcal{O}(m)$ as a single term.*

This result should be considered in Table 3.2 and Table 3.3. For example, applying Lemma 3.17 to the running time of maximum cardinality matching, $\mathcal{O}(m\sqrt{s})$, yields an improved time bound of $\mathcal{O}(\min\{m, s^2\}\sqrt{s} + m)$. We decided not retroactively to integrate this lemma into the tables. Firstly, we want to present the original proven running times. Secondly, integrating the result of this lemma complicates the tables.

3.4 All-Cavity Maximum Weight Matching

In this section, we discuss the all-cavity maximum weight matching problem, which is as follows.

Problem 3.1 (All-cavity maximum weight matching). *Given a weighted bipartite graph $G = (U \uplus V, E)$, determine an MWM on $G \setminus \{v\}$ for each vertex $v \in U \uplus V$.*

This problem has been introduced to graphs with integral weights by Kao, Lam, Sung, and Ting [63] and was later used as a backbone to compute maximum agreement subtrees [65]. Another application is the computation of a subgraph homeomorphism between trees and when computing a subtree isomorphism [17]. The time to compute a maximum common subtree isomorphism in Chapter 4 and a block-and-bridge preserving maximum common subgraph in Chapter 5 depends on the time to solve the (unbalanced) all-cavity maximum weight matching problem. This technique is also crucial when computing a block-and-bridge preserving maximum common edge subgraph [111], as we showed in [70]. In practice, it is common that we are interested in an MWM on $G \setminus \{v\}$ for each vertex v in either U or V only. If of relevance, we refer to this problem as the *one-sided* all-cavity maximum weight matching problem. An MWM on $G \setminus \{v\}$ for a single vertex $v \in U \uplus V$ is called *cavity MWM* as opposed to an all-cavity MWM.

3.4.1 Integral Weights

Let $G = (U \uplus V, E, w)$ be a bipartite graph with n vertices and m edges. Let $w : E \to \mathbb{N}$ be upper bounded by N. Kao et al. [63] solved the all-cavity MWM problem on G by reducing it to the single-destination longest paths problem. In this reduction, they constructed a new digraph D, where edges e of $\mathcal{M}$ become arcs from U to V with weight $-w(e)$. Edges e not belonging to $\mathcal{M}$ become arcs from V to U with weight $w(e)$. They also added a target vertex t and arcs of weight 0 from each matched vertex in V and each free vertex in U to t. They showed there is no positive

weighted directed cycle in D. This allows to solve the single-destination longest paths problem with target t in polynomial time. Their time analysis covers three steps.

1. The computation of an MWM $\mathcal{M}$ on G,
2. the construction of the digraph D, and
3. solving the single-destination longest paths problem on D.

The time bound for the second problem is $\mathcal{O}(m)$.[3] The third problem can be solved in time $\mathcal{O}(\sqrt{n}m \log N)$ [47]. Note, this work of Goldberg studies the single-source shortest paths problem for graphs without negative weight cycles. However, the single-destination longest paths and single-source shortest paths problem directly transfer into each other by replacing each arc (u, v) by its reversed arc (v, u) and multiplying the edge weights by −1. When the work of Kao et al. was published, the best-known time bound to compute an MWM on G was $\mathcal{O}(\sqrt{n}m \log(nN))$. However, Duan et al. [31] improved this bound to $\mathcal{O}(\sqrt{n}m \log N)$ as stated in Section 3.2.1. Therefore, the total time bound is $\mathcal{O}(\sqrt{n}m \log N)$. We summarize the result by Kao et al. concerning the improved time bound by Duan and Su for the MWM problem in the following proposition.

Proposition 3.20. *Let $G = (V, E, w)$ be a bipartite graph with n vertices and m edges. Let $w : E \to \mathbb{N}$ be upper bounded by N. The all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(\sqrt{n}m \log N)$.*

Unfortunately, Kao et al. did not take different sizes of U and V into account. We present an improved time bound with the sizes of U and V as parameters in Subsection 3.5.3.

3.4.2 Arbitrary Weights

The result from the previous subsection transfers directly to arbitrary edge weights. We replace the algorithms for the MWM and single-destination longest paths problems on integral weights by algorithms for arbitrary weights. From Section 3.2.1, we know an MWPM, and thus an MWM on G can be computed in time $\mathcal{O}(nm + n^2 \log n)$ using the Hungarian method with Fibonacci heaps. The construction of the digraph D as described by Kao et al. is independent of the edge weights, i.e., real-valued or integral. The single-source shortest paths problem without negative weight cycles can be solved using the well-known Bellman-Ford algorithm [4, 37] in time $\mathcal{O}(nm)$. The time to compute the initial MWM on G dominates the running time. The space bound for each step is $\mathcal{O}(m)$. We summarize the combined result in the following lemma.

Lemma 3.21. *Let $G = (V, E, w)$ be a bipartite graph with n vertices and m edges. Let $w : E \to \mathbb{R}$. The all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(nm + n^2 \log n)$ and space $\mathcal{O}(m)$.*

If an MWM $\mathcal{M}$ on G is known, the all-cavity MWM problem can be solved in time $\mathcal{O}(nm)$. An overview of results for the all-cavity matching problem for both balanced and unbalanced graphs is presented in Table 3.3.

[3] As a reminder, we assume $n \in \mathcal{O}(m)$ for the input graphs in this chapter; otherwise, the term needs to be replaced by $\mathcal{O}(n + m)$

Author	Restriction	Running time
Kao et al. (1997) [4]	integral weights	$\mathcal{O}(\sqrt{n}m \log N)$
Droschinsky; based on Kao et al. (1997)		$\mathcal{O}(nm + n^2 \log n)$
Chung (1987) (cf. Sect. 3.5.1)	unweighted	$\mathcal{O}(m\sqrt{s})$
Milo et al. (2012)		$\mathcal{O}(s^2 t)$
Milo et al. (2013)		$\mathcal{O}(s^3 + st)$
Droschinsky et al. (2016)		$\mathcal{O}(s^2 t + st \log t)$
– considering m (cf. Sect. 3.5.2)		$\mathcal{O}(ms + st \log t)$
Droschinsky et al. (2018)		$\mathcal{O}(s^2 t)$
Droschinsky; Theorem 3.31		$\mathcal{O}(\min\{s^2, m\}s + s^2 \log s + m)$
	integral weights	$\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N + m)$

Table 3.3: Worst-case running times for the all-cavity maximum weight matching problem on a bipartite graph $(U \uplus V, E)$, where $s = |U| \leq |V| = t$, $m = |E|$, $n = s + t$, and N as maximum edge weight.

3.5 All-Cavity Maximum Weight Matching on Unbalanced Bipartite Graphs

As we have seen in Section 3.3, considering the sizes of the vertex sets U and V in a bipartite graph $(U \uplus V, E)$ separately yields improved worst-case running times for the maximum weight matching problem. This section presents known results on the all-cavity maximum weight matching problem for unbalanced graphs in Subsection 3.5.1 and new results in Subsection 3.5.2. In Subsection 3.5.3, we improve and streamline the techniques from Subsection 3.5.2. Table 3.3 lists the different results.

3.5.1 Previous Results

Firstly, we present a result for unweighted graphs, followed by a result for arbitrarily weighted graphs.

All-cavity maximum cardinality matching. Let $G = (U \uplus V, E)$ be a bipartite graph and $\mathcal{M}$ a maximum cardinality matching on G. Let $s := |U| \leq |V| =: t$. Chung [17] showed it is possible to compute a maximum cardinality matching on $G \setminus \{v\}$ for each vertex $v \in V$ in time $\mathcal{O}(st)$ using a single depth-first search on a directed graph constructed from G and $\mathcal{M}$. If we take the number of edges m of G into account, we obtain $\mathcal{O}(m)$ as an improved upper bound. Following the proof, it is easy to argue that we can find a maximum cardinality matching on $G \setminus \{u\}$ for each vertex $u \in U$ in the same time bound. From Proposition 3.11 we know it is possible to compute $\mathcal{M}$ in time $\mathcal{O}(m\sqrt{s})$. We summarize these results.

Lemma 3.22. *Let $G = (U \uplus V, E)$ be a bipartite graph with m edges. Let $s = |U| \leq |V|$. The all-cavity maximum cardinality matching problem on G is solvable in time $\mathcal{O}(m\sqrt{s})$.*

[4] The running time by Kao et al. [63] considers the maximum weight matching result by Duan and Su [31].

All-cavity maximum weight matching with arbitrary weights. For arbitrary weights, the following result was shown by Milo et al. [92]. Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $s := |U| \leq |V| =: t$. The all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(s^2 t)$. This result includes the computation of an MWM on G. Unfortunately, the number of edges m in not considered in this results. However, there is a follow-up result by the same authors [93]. In this result, they bounded the number of edges to s^2, cf. Corollary 3.19. Further, they reduced the MWM problem to the Min-Cost Max-Flow problem. Their reduction is based on a refined approach described by Zhang [133]. This allowed a new time bound of $\mathcal{O}(s^3 + st)$. Additionally, they consider a more generalized MWM problem. This problem also allows vertex weights. They defined the weight of a matching as the sum of its edge weights plus the free vertices' weight.

Lemma 3.23. *Let $G = (V, E, w)$ be a bipartite graph with n vertices and m edges. Let $w : E \to \mathbb{R}$. Let $s = |U| \leq |V|$. The all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(s^3 + st)$.*

3.5.2 New Results

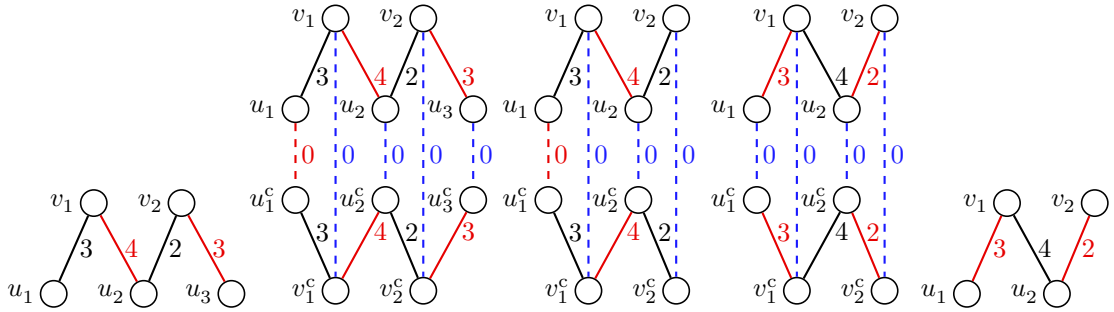
This subsection presents our findings on the one-sided all-cavity maximum weight matching problem through our published works [26] and [28]. These results hold for graphs with arbitrary edge weights. Note, the respective running times are independent of the chosen vertex set U or V. Therefore, we can solve the all-cavity MWM problem in the same time bound.

As before, let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $s := |U| \leq |V| =: t$. The result in [26] is based on additional iterations of the Hungarian method. Here, we achieve a time bound of $\mathcal{O}(s^2 t + st \log t)$ for the all-cavity MWM problem, which we present in the next paragraph. Following that, we show how to improve the time bound to $\mathcal{O}(s^2 t)$ based on our work in [28]. Instead of relying on the Hungarian method only, we consider the degree of unbalance. If it is high, we instead construct digraphs and then solve the all-cavity MWM problem by computing shortest paths on those digraphs. In our publications [26] and [28], we did not consider the number of edges in the graphs. The reason is that the all-cavity MWM problem was only a subproblem. As input for this problem, we had to assume the worst case, i.e., complete bipartite graphs. We improve the time bound by, among others, considering the number of edges in the analysis in Subsection 3.5.3.

Solving the all-cavity maximum weight matching problem using the Hungarian method. [5] Let $G = (U \uplus V, E, w)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. As we showed in Subsection 3.3.2, we can compute an MWM on G in time $\mathcal{O}(ms + st \log t)$, where s is the size of the smaller vertex set and t of the larger vertex set. To do this, we reduced the MWM problem to the MWPM problem. Note, in our original work [26], we upper-bounded the number of edges to st and with it the running time to $\mathcal{O}(s^2 t + st \log t)$.

In the following, we show how to solve the one-sided all-cavity MWM problem on G. Let $G_u := G \setminus \{u\}$ for each $u \in U$. We compute an MWM on each subgraph G_u as follows: Let y be an optimal dual solution (cf. Subsection 3.2.4) from computing an MWPM $\mathcal{M}'$ on the reduced graph (denoted as G') using the Hungarian method (cf. Subsection 3.2.2). Let $\mathcal{M}$ be the MWM on G, derived from the MWPM $\mathcal{M}'$ on G'. If u is free by $\mathcal{M}$, i.e., $u \notin e$ for all $e \in \mathcal{M}$, then $\mathcal{M}_u := \mathcal{M}$ is an MWM of G_u. Otherwise, let G'_u be the reduced graph from G_u. We obtain a feasible dual solution y_u on the bipartite graph G'_u by taking the dual values from y, i.e., $y_u(v) := y(v)$ for

[5] This paragraph mainly consists of our contribution in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 4.



(a) Input graph G (b) Reduced graph G' (c) G'_3 (d) G'_3+MWPM (e) $G_3 \coloneqq G \setminus \{u_3\}$

Figure 3.7: Different stages of computing a cavity MWM. Here, the vertex u_3 is removed. Matchings depicted in red. (a) Weighted bipartite graph G including MWM $\mathcal{M}$ and (b) corresponding reduced graph G' with MWPM $\mathcal{M}'$; (c) reduced graph G'_3 with u_3 and u_3^c removed, initial matching $\mathcal{M}'_3$; (d) reduced graph G'_3 including an MWPM, obtained from a single iteration of the Hungarian method; (e) G_3 with MWM $\mathcal{M}_3$.

all $v \in V(G'_u)$. Note, we have $2(k+l)$ vertices in G', and precisely two less in G'_u, i.e., a perfect matching in G'_u has size $s+t-1$.

We can derive an initial matching $\mathcal{M}'_u$ on G'_u of size $s+t-2$ from $\mathcal{M}'$; $\mathcal{M}'_u$ contains the matching edges that are not incident to the two removed vertices from G' to G'_u. Therefore, only one more iteration of the Hungarian method is needed. We achieve this in time $\mathcal{O}(st + t\log t)$, cf. proof of Lemma 3.12. We then directly obtain an MWM $\mathcal{M}_u$ on G_u from the MWPM $\mathcal{M}'_u$ on G'_u. An example is depicted in Figure 3.7. The pseudo code of this approach is listed in Algorithm 1.

Next, we analyze the total time to compute an MWM on each of the graphs G_u, $u \in U$. We need to compute an MWM different from $\mathcal{M}$ for at most s of those graphs because the size of any matching is at most s. Since we can compute one iteration of the Hungarian method in time $\mathcal{O}(st + t\log t)$, the total time is bounded by $\mathcal{O}(s^2t + st\log t)$, which is identical to the time bound to compute an MWM on G. The space bound is $\mathcal{O}(m)$, since we can dismiss each cavity matching $\mathcal{M}_u$ from memory after outputting it. We summarize this result in the following proposition.

Proposition 3.24. *Let $G = (U \uplus V, E, w)$ be a bipartite graph. Let s be the size of the smaller of the vertex sets U and V, and t be the size of the larger set. Let $w : E \to \mathbb{R}$. The one-sided all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(s^2t + st\log t)$ and space $\mathcal{O}(m)$.*

Solving the all-cavity maximum weight matching problem using single-source shortest paths. [6]
In this paragraph, we provide an improved time bound to solve the one-sided all-cavity maximum weight matching problem compared to our previous result from Proposition 3.24. We start with the result and prove it in the following.

Proposition 3.25. *Let $G = (U \uplus V, E, w)$ be a bipartite graph. Let $s = |U| \leq |V| = t$. Let $w : E \to \mathbb{R}$. The one-sided all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(s^2t)$.*

[6] This paragraph mainly consists of our contribution in *Largest Weight Common Subtree Embeddings with Distance Penalties* [28], Section 5.2.

Algorithm 1: One-sided all-cavity maximum weight matching (Hungarian method)

Input : Bipartite graph $G = (V \uplus U, E, w)$, $|U| \geq 2$, and edge weights $w : E \to \mathbb{R}$
Output : MWM $\mathcal{M}$ on G and MWMs $\mathcal{M}_u$ on $G_u \coloneqq G \setminus \{u\}$ for each $u \in U$.

1 **if** $|V| \leq |U|$ **then** ▷ *Compute an MWM $\mathcal{M}$ on G*
2 Let $G' \coloneqq (V', E')$, where $V' = V \cup U \cup \{v^c \mid v \in V \cup U\}$ and $E' = E \cup \{e^c \mid e \in E\} \cup \{vv^c \mid v \in V \cup U\}$.
3 $w(e^c) \leftarrow w(e)$ for all $e \in E$ ▷ *Weight of copied edges*
4 $w(vv^c) \leftarrow 0$ for all $v \in V \cup U$ ▷ *Weight of additional edges*
5 $y(u^c) \leftarrow y(u) \leftarrow 0$ for all $u \in U$ ▷ *Initial dual values*
6 $y(v^c) \leftarrow y(v) \leftarrow \max\{w(vu) \mid u \in U\}$ for all $v \in V$
7 $\mathcal{M}' \leftarrow \{uu^c \mid u \in U\}$ ▷ *Initial matching*
8 Transform $\mathcal{M}'$ into an MWPM using $|V|$ iterations of the Hungarian method.
9 $\mathcal{M} \leftarrow \{vu \mid vu \in \mathcal{M}', v \in V, u \in U\}$
10 **else**
11 Exchange the vertices of V and U.
12 Compute $\mathcal{M}'$ and $\mathcal{M}$ as in lines 2 to 9 and exchange V and U back.
13 $y \leftarrow$ The dual values obtained from computing the MWPM $\mathcal{M}'$.
14 **foreach** $u \in U$ **do** ▷ *cavity MWMs $\mathcal{M}_u$ on G_u*
15 **if** *u is free by $\mathcal{M}$* **then**
16 $\mathcal{M}_u \leftarrow \mathcal{M}$
17 **else**
18 $G'_u \leftarrow G' \setminus \{u, u^c\}$
19 $\mathcal{M}'_u \leftarrow \mathcal{M}' \cap E(G'_u)$ ▷ *Edges in $\mathcal{M}'$ incident to u, u^c removed*
20 Transform M'_u into an MWPM using a single iteration of the Hungarian method.
21 $\mathcal{M}_u \leftarrow \{vu' \mid vu' \in \mathcal{M}'_u, v \in V, u' \in U\}$

Proof. We distinguish two cases.

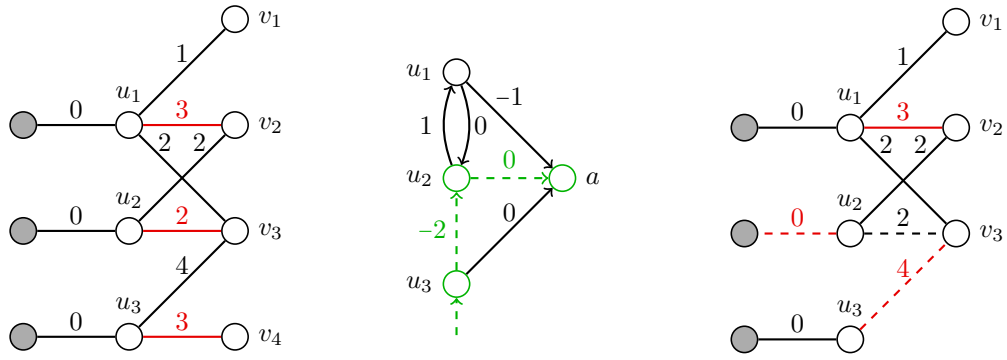
i) $s \geq \log t$. In this case, we use the result from Proposition 3.24. With this result, we can solve the one-sided all-cavity maximum weight matching problem in time $\mathcal{O}(s^2 t + st \log t)$. Under the premise $s \geq \log t$, that is $\mathcal{O}(s^2 t)$.

ii) $s < \log t$. Then one vertex set is much smaller than the other. According to the following Lemma 3.26, we can solve the problem in time $\mathcal{O}(s^4 + s^2 t)$. Under the premise $s < \log t$, that is $\mathcal{O}(s^2 t)$.

□

Lemma 3.26. *Let G be a weighted bipartite graph with vertex sets U and V, $s = |U| \leq |V| = t$. Let either $C = U$ or $C = V$. We can compute an MWM on G and an MWM on $G \setminus \{c\}$, $\forall c \in C$, in total time $\mathcal{O}(s^4 + s^2 t)$.*

Proof. From Proposition 3.13, we know there is an algorithm that computes an MWM on G in time $\mathcal{O}(s^2 t)$. This algorithm first copies the s vertices of U and then adds an edge of weight 0 between each vertex of U and its copy. We denote this graph by $\widetilde{G}$. The algorithm computes an MWM_s $\widetilde{\mathcal{M}}$ on $\widetilde{G}$ ($\widetilde{\mathcal{M}}$ is also an MWM), which corresponds to an MWM on G. An example is depicted in Figure 3.8a.



(a) $\widetilde{G}$ and an MWM $\widetilde{\mathcal{M}}$ (red).

(b) D_{v_4} and a shortest path P from v_4's partner u_3 to a (green, dashed lines).

(c) An MWM (red) on $\widetilde{G}_{v_4}$ obtained from a single augmenting path (dashed lines).

Figure 3.8: (a) An MWM$_s$, $s = |U| = 3$, on $\widetilde{G}$, which is also an MWM. (b) The graph D_{v_4}, on which we compute a shortest path from u_3 to a. Such a path corresponds to an augmenting path of maximal weight in $\widetilde{G}_{v_4}$. (c) Applying the path yields an MWM$_3$ on $\widetilde{G}_{v_4}$, which is also an MWM.

The graph $\widetilde{G}$ with one vertex $c \in C$ removed is denoted by $\widetilde{G}_c$, i.e., $\widetilde{G}_c \coloneqq \widetilde{G} \setminus \{c\}$. If c is not matched, we are done. If c is matched, let u (in case $c \in V$) respectively v (in case $c \in U$) be the mate of c. Let $\widetilde{\mathcal{M}}_c \coloneqq \widetilde{\mathcal{M}} \setminus \{cu\}$ or $\widetilde{\mathcal{M}}_c \coloneqq \widetilde{\mathcal{M}} \setminus \{cv\}$, respectively. We observe $|\widetilde{\mathcal{M}}_c| = s - 1$.

First, assume $c \in V$. In an MWM$_s$ of $\widetilde{G}_c$, each vertex of U including u must be matched. An odd length $\widetilde{\mathcal{M}}_c$-augmenting path P incident to u that maximizes the weight yields an MWM$_s$ on $\widetilde{G}_c$ and thus an MWM on G_c. This follows from the fact that any $\widetilde{\mathcal{M}}_c$-alternating cycle or path on $\widetilde{G}_c$ not incident to u has non-positive weight; otherwise, $\widetilde{\mathcal{M}}$ was no MWM. We can find such a path using the Bellman-Ford algorithm in time $\mathcal{O}(st + s^3)$ as follows. Let $D_c = (U \cup \{a\}, A)$ be the digraph, where A is the union of the following two sets of directed arcs.

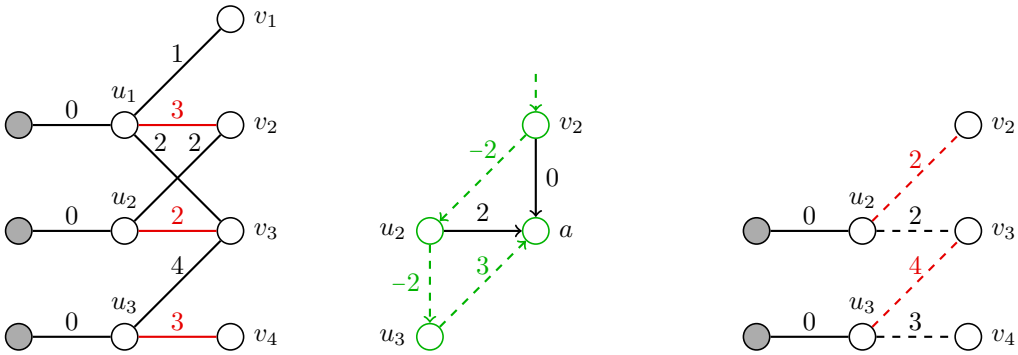
1. For each alternating path $\bar{u}vu'$ in $\widetilde{G}_c$, where $\bar{u}, u' \in U, v \in V$, and vu' is matched, we add the arc $(\bar{u}, u')$ with weight $w(vu') - w(\bar{u}v)$.

2. For each vertex $u' \in U$, let v be a free vertex adjacent to u', such that the edge $u'v$ has maximum weight among all such edges. We add an arc (u', a) of weight $-w(u'v)$.

The time to construct the graph is bounded by $\mathcal{O}(st)$. Since D_c has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we can compute a shortest path P from c's mate u to a in time $\mathcal{O}(s^3)$. We obtain an MWM on $\widetilde{G}_c$ by augmenting $\widetilde{\mathcal{M}}_c$ with the edges that correspond to P in D_c. Figure 3.8b depicts an example for D_c, as well as a shortest path P. Figure 3.8c depicts the resulting MWM. Since at most s vertices of V are matched by M, the total time to compute an MWM on each of the graphs G_c, $c \in V$, is $\mathcal{O}(s^4 + s^2 t)$.

Second, assume $c \in U$. Each vertex in U is matched. Therefore, the cardinality of $\widetilde{\mathcal{M}}_c$ is $s - 1$. This time we need to find an alternating path of even length (we removed a vertex from U) and of maximal weight incident to c's mate v. Any alternating cycle or path not incident to v cannot augment $\widetilde{\mathcal{M}}_c$ to greater weight; otherwise, $\widetilde{\mathcal{M}}$ was no MWM. This path can have length 0, e.g., if $\widetilde{\mathcal{M}}_c$ is an MWM of $\widetilde{\mathcal{M}}$. The total time to compute such a path is $\mathcal{O}(s^3)$ as follows. Let $D_c = (U \cup \{v, a\}, A)$, where A is the union of the following four sets of directed arcs.

1. For each edge $vu \in E(\widetilde{G}_c)$, $u \in U$, we add the arc (v, u) with weight $-w(vu)$.



(a) $\widetilde{G}$ and an MWM $\widetilde{\mathcal{M}}$ (red).

(b) D_{u_1} and a shortest path P from u_1's partner v_2 to a (green, dashed).

(c) An MWM (red) on $\widetilde{G}_{u_1}$ (isolated vertices removed) obtained from a single augmenting path (dashed lines).

Figure 3.9: (a) $\widetilde{G}$ and an MWM as in Figure 3.8a (b) The digraph D_{u_1}, on which we compute a shortest path from v_2 to a. Such a path corresponds to an augmenting path of maximal weight in $\widetilde{G}_{u_1}$. (c) Applying the path yields a MWM_2 on $\widetilde{G}_{u_1}$, which is also an MWM.

2. For each alternating path $uv'u'$ in $\widetilde{G}_c$, where $u, u' \in U, v' \in V$, and uv' is matched, we add the arc (u, u') with weight $w(uv') - w(v'u')$.
3. For each matching edge uv', where $u \in U, v' \in V$, we add the arc (u, a) with weight $w(uv')$.
4. We add the arc (v, a) with weight 0.

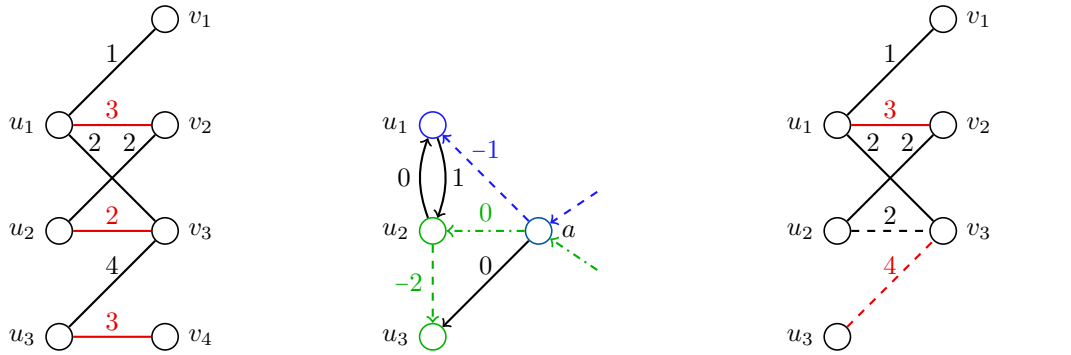
The time to construct the graph is bounded by $\mathcal{O}(s^2)$. Since D_c has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we can compute a shortest path P from c's mate v to a in time $\mathcal{O}(s^3)$. Again, P yields the augmenting path in $\widetilde{G}_c$. An example of constructing D_c and the resulting MWM is depicted in Figure 3.9. Since all the s vertices of U are matched by $\mathcal{M}$, the total time to compute an MWM on each of the graphs G_c, $c \in U$, is $\mathcal{O}(s^4)$. □

3.5.3 Further Improvements

In this subsection, we improve and streamline the shortest paths technique from the previous subsection, such that two digraphs are sufficient. We also consider the number of edges m in our running time analysis. This allows solving the all-cavity maximum weight matching problem on G in time $\mathcal{O}(\min\{s^2, m\}s + s^2 \log s + m)$ for arbitrary weights, where s is the smaller vertex set in G. For integral weights of at most N, the problem is solvable in time $\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N + m)$. Firstly, we start with a corollary to compute an MWM on an unbalanced bipartite graph following Lemma 3.17 for insignificant edges and the previous results for computing an MWM on an unbalanced graph, cf. Proposition 3.13 and 3.14.

> **Corollary 3.27.** *Let $G = (U \uplus V, E, w)$ be a bipartite graph with m edges. Let $s = |U| \leq |V|$. An MWM on G is computable in time $\mathcal{O}(\min\{s^2, m\}s + s^2 \log s + m)$. If the weights are integral bounded by N, it is computable in time $\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N + m)$.*

The following lemma states the relation between an MWM on G and $G \setminus \{c\}$, where c is a matched vertex in G.



(a) G and an MWM $\mathcal{M}$ (red).

(b) D_V and shortest paths from a to u_1 (blue), to u_2 (green, dash-dotted), and to u_3 (green).

(c) An MWM (red) on G_{v_4} obtained from a single augmenting path (dashed lines).

Figure 3.10: (a) An MWM on G. (b) D_V and the 3 shortest paths from a to u_1, u_2, and u_3. The paths correspond to augmenting paths of maximal weight in G. (c) Applying the shortest path to $u_3 = \mathcal{M}(v_4)$ yields an MWM on G_{v_4}.

Lemma 3.28. *Let $\mathcal{M}$ be an MWM on a bipartite graph G, $c \in V(G)$ be a vertex matched by $\mathcal{M}$, $G_c = G \setminus \{c\}$, $v = \mathcal{M}(c)$, and $\mathcal{M}_c = \mathcal{M} \setminus \{cv\}$. If $\mathcal{M}_c$ is no MWM in G_c, we obtain an MWM $\mathcal{M}'_c$ from an augmenting path P starting in v such that $\mathcal{M}_c \oplus P = \mathcal{M}'_c$.*

Proof. Assume there is an $\mathcal{M}_c$-augmenting path or cycle P' in G_c not containing v. Then P' in G would be $\mathcal{M}$-augmenting, which contradicts $\mathcal{M}$ being an MWM. Therefore, P must contain v. Since v is $\mathcal{M}_c$-free, P must start or end in v. Further, two or more paths starting or ending in v cannot be applied simultaneously. Therefore, P, as claimed, must exist. □

We use the above lemma to reduce the one-sided all-cavity MWM problem to the single-source shortest paths problem. Instead of relying on a different digraph per shortest path computation as in the previous subsection, we construct only two digraphs, one for U and V each. The basic idea is to reverse the arcs in the digraphs from the previous subsection, then merge them; one digraph D_V for the cavity matchings of V, another digraph D_U for the cavity matchings of U. Further, we start with $\mathcal{M}$ on G instead of $\widetilde{\mathcal{M}}$ on $\widetilde{G}$. The construction is consistent between D_V and D_U.

Solving the all-cavity maximum weight matching problem. Let $G = (U \uplus V, E, w)$, $\mathcal{M}$ an MWM on G, and $s := |U| \leq |V|$. We first consider the one-sided all-cavity MWM problem on the larger vertex set V. Let G be preprocessed according to Lemma 3.17, i.e., there are at most s edges adjacent to each vertex $u \in G$, and all edges have positive weight. Let $D_V = (U_{\mathcal{M}} \cup \{a\}, A)$ be the digraph, where $U_{\mathcal{M}} = \{u \in U \mid u \text{ is matched by } \mathcal{M}\}$ and A is the union of the following two sets of directed arcs.

1. For each $u \in U_{\mathcal{M}}$ let v be a free vertex adjacent to u, such that the edge $uv \in E$ has maximum weight among all such edges. We add an arc (a, u) of weight $-w(uv)$. If no such vertex v exists, we add an arc (a, u) of weight 0 instead.

2. For each alternating path $u'vu$ in G, where $u', u \in U_{\mathcal{M}}$, $v \in V$, and $u'v \in \mathcal{M}$, we add the arc (u', u) with weight $w(u'v) - w(vu)$.

An example of D_V and applying a shortest path to obtain a cavity MWM (cf. Lemma 3.29) is depicted in Figure 3.10.

Lemma 3.29. *Let $v \in V$ be a vertex matched by an MWM $\mathcal{M}$ in a bipartite graph G. Let P be a shortest path from a to $u = \mathcal{M}(v)$ in D_V. Let $\mathcal{M}_v = \mathcal{M} \setminus \{vu\}$. If the path is of negative weight, then augmenting $\mathcal{M}_v$ by the edges corresponding to P yields an MWM on G_v.*

Proof. First, the arc's weights are chosen such that matching edges increase the path's length, and free edges decrease the path's length. Therefore, the (negative) length of any path in D_V is equal to the increase in the matching's weight after augmentation.

From Lemma 3.28 we know, that any $\mathcal{M}_v$-augmenting path P_v in G_v must start in u. Assume P_v starts with the edge uv', such that v' is matched. Then the next edge on the path is $\{v', \mathcal{M}(v')\}$; otherwise, the result would not be a matching. This is realized by following the arc $(\mathcal{M}(v'), u)$ (step 2 in the construction of D_V). If there is another matched vertex on the path P_v, we can repeat this process. If P_v contains an $\mathcal{M}_v$-free vertex besides u, then this is the last vertex in P_v, because two consecutive free edges are not allowed in an alternating path. This last edge is represented by step 1 in the construction of D_V. □

Note, P_v can never contain an $\mathcal{M}_v$-free vertex $u' \in U \setminus U_{\mathcal{M}}$. Therefore, it is sufficient that $V(D_V)$ consists of the vertices of $U_{\mathcal{M}}$ only. Next, we prove the running time to solve the one-sided all-cavity MWM problem on V.

Lemma 3.30. *Let $G = (U \uplus V, E, w)$ be a bipartite graph with m edges and $s = |U| \leq |V|$. An MWM on each graph G_v, $v \in V$, is computable in time $\mathcal{O}(\min\{s^2, m\}s + s^2 \log s)$ for arbitrary weights and in time $\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N)$ for integral weights of at most N.*

Proof. Let $\mathcal{M}$ be an MWM on G. For the free vertices $v \in V$, the matching $\mathcal{M}$ is also an MWM on G_v. For the matched vertices $v \in V$, we obtain an MWM from D_V in the claimed time bound as follows. The time to construct D_V is $\mathcal{O}(m)$ since each free edge in G is considered for at most one arc of D_V, and the matched edges are only considered indirectly together with the free edges.

D_V has $\mathcal{O}(s)$ vertices and $\mathcal{O}(\min\{m, s^2\})$ edges using Lemma 3.17. Since we have no negative weight cycles, the single-source shortest paths problem is solvable in time $\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N)$ for integral weights [47] and time $\mathcal{O}(\min\{s^2, m\}s)$ for arbitrary weights [4, 37]. Therefore, the proof concludes. □

We solve the one-sided all-cavity MWM problem on the smaller vertex set U analog. The only difference is that we interchange the vertex sets U and V when constructing the digraph D_U. Then $V(D_U) = \{v \in V \mid v \text{ is matched by } \mathcal{M}\} \cup \{a\}$. Consequently, $|D_U| \in \mathcal{O}(s)$ and Lemma 3.30 is also valid for the smaller vertex set U. With Corollary 3.27, we obtain the following theorem.

Theorem 3.31. *Let $G = (U \uplus V, E, w)$ be a bipartite graph with m edges and $s = |U| \leq |V|$. The all-cavity maximum weight matching problem on G is solvable in time $\mathcal{O}(\min\{s^2, m\}s + s^2 \log s + m)$. If the weights are integral bounded by N, it is solvable in time $\mathcal{O}(\min\{s^2, m\}\sqrt{s} \log N + m)$.*

3.6 Enumerating Maximum Weight Matchings

In this section, we study the enumeration of maximum weight matchings with polynomial delay. We discussed the enumeration problem and relevant classes in Section 2.4. Previous

results include the enumeration of perfect matchings [39, 124] and maximum weight perfect matchings [39]. Further efficient enumeration algorithms have been shown for maximal and maximum cardinality matchings in unweighted bipartite graphs [125]. Other results include the enumeration of matchings in general graphs under certain restrictions and further problem variants [76, 99, 13]. Regarding the complexity, it has been shown that the problem to enumerate perfect matching in unweighted bipartite graphs is #P-complete [126]. The class #P contains all problems, such that there exists a polynomial-time nondeterministic Turing machine, which has for each instance of that problem precisely that many accepting branches as there are valid solutions.

The problem to enumerate maximum weight matchings has not been addressed directly. We solve this problem by using two reductions. The first is from the MWM to the MWPM problem. The second is a reduction from the MWPM problem to the (unweighted) perfect matching problem. We address the latter one first.

Reduction from maximum weight perfect matching to perfect matching. This reduction is based on the so-called equality subgraph.

> **Definition 3.32 (Equality subgraph).** *Let $G = (V, E, w)$ be a weighted bipartite graph and y be an optimal solution to DP:MWPM from Section 3.2.4. The graph $G_y = (V, E_y)$, where $E_y = \{uv \in E \mid y(u) + y(v) = w(uv)\}$, is known as* equality subgraph *or* admissible subgraph *to the dual solution y.*

The equality subgraph is not unique, as different optimal solutions y can yield different equality subgraphs G_y. However, any equality subgraph suffices as stated by Fukuda and Matsui [39].

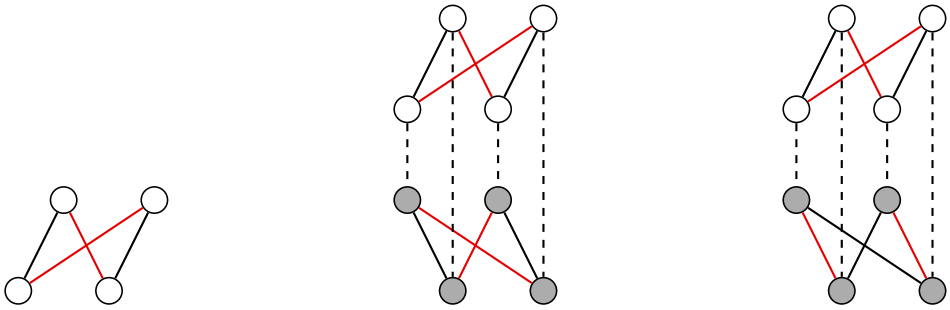
> **Lemma 3.33 ([39]).** *There is a one-to-one relationship between the perfect matchings in the equality subgraph G_y for any optimal dual solution y and the maximum weight perfect matchings in G.*

Proof. The proof follows from Proposition 3.9 (complementary slackness) but is not contained in the work of [39]. Assume there is an edge $uv \notin E_y$, such that $uv \in \mathcal{M}$ for any MWPM $\mathcal{M}$ in G. Let x be the optimal solution corresponding to $\mathcal{M}$, i.e., $x(uv) = 1 \Leftrightarrow uv \in \mathcal{M}$. Since both x and y are optimal, the corresponding dual inequality is binding, i.e., $y(u) + y(v) = w(uv)$. Therefore, the edge uv must be contained in E_y. This is a contradiction.

Assume a solution x corresponding to any perfect matching $\mathcal{M}$ in G_y. Since for each edge in E_y the dual inequality is binding, the first complementary slackness condition is satisfied. Since $\mathcal{M}$ is perfect, each primal inequality is binding. Therefore, x must be optimal. Consequently, $\mathcal{M}$ is an MWPM in G. □

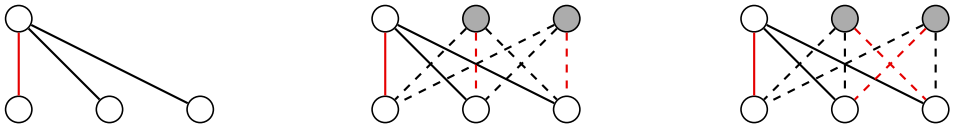
The above result simplifies the enumeration of maximum weight perfect matchings. The Hungarian method yields an optimal dual solution y from which we construct the equality subgraph G_y. Then we can use an algorithm to enumerate perfect matchings on this graph, such as presented in [125, 124].

Reduction from maximum weight matching to maximum weight perfect matching. In this chapter, we showed two different reductions from the maximum weight matching problem. There is another reduction we showed in our previous work [25]. The three reductions are as follows.



(a) MWM $\mathcal{M}$ (red) (b) MWPM $\mathcal{M}_1$ (red) (c) MWPM $\mathcal{M}_2$ (red)

Figure 3.11: Reduction 1; solid edges have weight 1, dashed edges have weight 0. (a) MWM $\mathcal{M}$ in the input graph. (b), (c) Different MWPMs $\mathcal{M}_1$ and $\mathcal{M}_2$ in the reduced graph, which correspond to the same MWM $\mathcal{M}$. Gray vertices added through the reduction.



(a) MWM $\mathcal{M}$ (red) (b) MWPM $\mathcal{M}_1$ (red) (c) MWPM $\mathcal{M}_2$ (red)

Figure 3.12: Reduction 3; solid edges have weight 1, dashed edges have weight 0. (a) MWM $\mathcal{M}$ in the input graph. (b), (c) Different MWPMs $\mathcal{M}_1$ and $\mathcal{M}_2$ in the reduced graph, which correspond to the same MWM $\mathcal{M}$. Gray vertices added through the reduction.

1) The standard reduction to the MWPM problem is via copying the input graph and connecting the copied vertices to the original vertices through edges of weight 0, cf. Figure 3.6b.

2) A reduction to the one-sided MWPM problem is by copying only the smaller vertex set and connecting the new vertices through weight 0 edges to the original vertices, followed by the Hungarian method by Ramshaw and Tarjan [102], cf. Figure 3.6c.

3) In our previous work [25], we could assume the bipartite input graphs to be bipartite complete since the enumeration of all maximum weight matchings was a subproblem to another problem. In that reduction, we added vertices to the smaller vertex set such that both vertex sets had the same size. Then we added edges of weight 0 between the new vertices and all vertices of the previously larger vertex set.

If we are interested in a single optimal solution, any reduction suffices. For enumeration, however, Reduction 1) and 3) do not guarantee a one-to-one relation between maximum weight matchings in the original graph and maximum weight perfect matchings in the reduced graph, as shown in Figure 3.11 and 3.12, respectively. Using Reduction 2) we cannot find any perfect matching, since we have a reduction to the one-sided MWPM problem and, therefore, cannot use any algorithm that enumerates perfect matchings. Even enumerating one-sided perfect matchings on the equality subgraph will not suffice: In the one-sided MWPM problem, we cannot assume each inequality (3.3) to be binding. Therefore, the complementary slackness conditions do not necessarily hold. A trivial counterexample is a graph G containing exactly one edge uv of weight 1. The reduced graph contains an additional edge uu^c of weight 0. Both edges are contained in the equality subgraph obtained from the the Hungarian method by Ramshaw and Tarjan. However, only one edge (that of weight 1) corresponds to a one-sided MWPM in G.

In the following, we present a perfect matching enumeration algorithm for Reduction 1) and another for Reduction 3) such that no two enumerated perfect matchings in the equality subgraph correspond to the same MWM in the input graph.

Polynomial delay enumeration of MWMs using Reduction 3). [7] Reduction 3) is based on a modification of Uno's algorithm for the enumeration of all perfect matchings in a bipartite graph [125]. First, we briefly describe Uno's algorithm. Given a bipartite graph G and a (first) perfect matching $\mathcal{M}$, all edges not contained in alternating cycles are removed, and a remaining edge $e \in \mathcal{M}$ is selected, if existing. The problem is then divided into the enumeration of the perfect matchings containing e and those not containing e. These subproblems lead to a graph $G^+(e)$ with initial matching $\mathcal{M}$ and another graph $G^-(e)$ with initial matching $\mathcal{M}'$ as described below. The total number of edges and vertices in $G^+(e)$ and $G^-(e)$ is less than in G. The enumeration continues recursively until no more edge e can be selected. The selection of e is key to the algorithm. Uno proved that there is another perfect matching if and only if there is an alternating cycle, cf. Definition 3.4. From an alternating cycle C containing e, we obtain another perfect matching $\mathcal{M}' \coloneqq \mathcal{M} \oplus C$. The matching $\mathcal{M}'$ is output in each inner node of the binary enumeration tree. Edges that are not part of any cycle are removed in each recursive step of the algorithm.

On a bipartite graph with n vertices and m edges, a perfect matching can be computed in time $\mathcal{O}(n^{1/2}m)$ [57]. That is the first step in Uno's algorithm. In our case, the initial graph is the equality subgraph G_y; the perfect matching is obtained during the calculation of G_y, i.e., from the Hungarian method. Uno states $\mathcal{O}(n+m)$ time per additional matching, which is the time to find an alternating cycle and improves this with an amortized cost analysis to $\mathcal{O}(n)$.

As shown in Figure 3.12, we cannot use Uno's algorithm as a black box. Instead, we modify it such that each perfect matching regarding the original graph G is enumerated precisely once. If the search for an alternating cycle starts from a vertex of the smaller vertex set of G (white vertices in the top row of Figure 3.12b and 3.12c), the first edge on the cycle will be an edge from G. Therefore, the newly obtained matching $\mathcal{M}'$ will be different from the previous matching M regarding G, i.e., $\mathcal{M}' \cap E(G) \neq \mathcal{M} \cap E(G)$. If there is no such vertex, the current recursion has finished, as no new matchings regarding G can be found. In this sense, we prune the recursion tree. Unfortunately, this means that the time per matching, $\mathcal{O}(n)$, is not valid for our modification, since our algorithm stops the recursion as soon as there is no other perfect matching regarding G. Consequently, the higher costs of the first matchings cannot be allocated to the costs of the later/smaller matching graphs. For example, in the complete bipartite graph $K_{n,n}$, there are $n!$ perfect matchings. Given an initial perfect matching and assuming the original graph G consisting of only 2 vertices in the smaller vertex set, the algorithm's total time to compute the other $\alpha = n(n-1)-1$ perfect matchings regarding G is $\Theta(\alpha n^2) = \Theta(n^4)$, which is not contained in $\mathcal{O}(\alpha n) = \mathcal{O}(n^3)$.

Proposition 3.34. *All α maximum weight matchings of any weighted complete bipartite graph G with n vertices can be enumerated with polynomial delay in total time $\mathcal{O}(n^3 + \alpha n^2)$.*

Proof. The equality subgraph G_y of an optimal dual solution y and the first perfect matching are obtained in time $\mathcal{O}(n^3)$ using the Hungarian method. At this point, the enumeration starts. The solutions are output in the inner nodes of the binary enumeration tree. The running time for each node is $\mathcal{O}(n^2)$ as stated above. □

[7] This paragraph consists of our contribution in *Enumeration of Maximum Common Subtree Isomorphisms with Polynomial-Delay* [25], Section 3.

If the two disjoint sets of G have the same size, we do not need to add vertices or edges. Then we achieve a total time of $\mathcal{O}(n^3 + \alpha n)$ as proven by Uno [125].

Polynomial delay enumeration of MWMs using Reduction 1). As seen in Figure 3.11, different MWPMs in the reduced graph can correspond to the same MWM in the input graph. Let V denote the vertices of the input graph G and V' the vertices of the reduced graph G', where $V' \coloneqq V \cup V^{\mathrm{c}}$ is composed of the vertices V of G (the white vertices in Figure 3.11) and their copies V^{c} (the gray vertices in Figure 3.11). When enumerating perfect matchings on G', we have to assure that no two different perfect matchings $\mathcal{M}$ and $\mathcal{M}'$ are enumerated, such that $\mathcal{M}' \cap E(G) = \mathcal{M} \cap E(G)$. Therefore, we modify another enumeration algorithm for perfect matchings by Uno [124] accordingly. We describe that algorithm and our modifications in the following.

Both algorithm from Uno are based on the fact that any two perfect matchings differ by an alternating cycle. Given a bipartite graph and an initial perfect matching, the newer algorithm [124] enumerates all perfect matchings in that graph as follows.

(1) Firstly, any edge which does not belong to any alternating cycle is removed.

(2) Secondly, resulting isolated vertices are removed.

(3) Thirdly, paths of degree two vertices are shrunk as follows. If there is any edge uv such that $\delta(u) = \delta(v) = 2$ let w_1u and vw_2, respectively, be the only other edge adjacent to u and v. Then the edges w_1u, uv, and vw_2 and the vertices u and v are removed from the graph. Then the edge w_1w_2 is added to the graph. If two of the three removed edges are matched, w_1w_2 is also matched; otherwise, the edge is free. This step is repeated until there is no other such edge.

It was shown that perfect matchings after performing the above three steps correspond one-to-one to perfect matchings in the previous graph. The goal of these steps is to keep the graph's density high while reducing the size of the graph as much as possible. The steps allow lower bounding the number of perfect matchings in such a graph and, consequently, dividing the upper recursive nodes' higher computational costs to the lower nodes in the recursion tree [124].

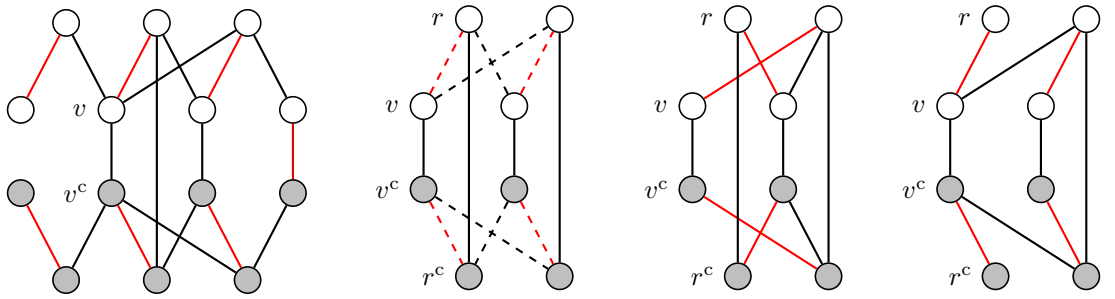
(4) Fourthly, the enumeration problem is divided into perfect matchings containing edges of a set E_1 and other perfect matchings containing edges of a set E_2, such that $E_1 \cup E_2 = N(r)$ and $E_1 \cap E_2 = \emptyset$, where r is a specific vertex of the bipartite graph in the current recursive step. Since all edges that are not contained in any alternating cycle were removed from G', the graph is split into k connected components $cc_1, \ldots, cc_k$, $k \geq 1$. The vertex r is selected from a component cc_i, where $f(i) \coloneqq |E(cc_i)| - |V(cc_i)|$ is minimal. With proper selection of the sets E_1 and E_2, an amortized time of $\mathcal{O}(\log n)$ per perfect matching has been shown.

Before we list our modifications to the algorithm by Uno [124], we observe the following.

Lemma 3.35. *Let G be any bipartite graph, and $G' = (V \cup V^c, E)$ be the reduced graph from Reduction 1). Let G_y be the equality subgraph to an optimal dual solution y to the DP:MWPM on G' and $\mathcal{M}$ a perfect matching on G_y. Let trim(G_y) be the resulting graph from removing all edges not contained in any alternating cycle regarding $\mathcal{M}$ in G_y.*

Then the edges between vertices in V are mirrored *to the edges between vertices in V^c, formally $E(G[V]) = \{e \mid e^c \in E(G[V^c])\}$.*

Proof. The claim follows from Lemma 3.33 and the fact that for each other perfect matching $\mathcal{M}'$ in trim(G_y), there is an alternating cycle C such that $\mathcal{M} \oplus C = \mathcal{M}'$. □



(a) G' of the current iteration including a perfect matching $\mathcal{M}$ in red

(b) trim(G'), alternating cycles C and C' (dashed lines)

(c) Recursive call on trim(G') $\setminus$ E_1, $E_1 = \{rv, r^c v^c\}$

(d) Recursive call on trim(G') $\setminus$ E_2

Figure 3.13: One recursive step of the MWM enumeration algorithm using Reduction 1). Gray vertices added by the reduction. (a) Input graph of the current recursion. (b) Trimmed graph with removed edges on the left and shrunk edges on the right side. The vertex r is arbitrarily chosen since there is only one connected component. C and C' are alternating cycles mirrored to each other. (c) Recursion with the edges from E_1 removed and a new perfect matching $\mathcal{M} \oplus C \oplus C'$. (d) Recursion with the edges from E_2 removed and the perfect matching $\mathcal{M}$.

The goal of our modification is that in each node of the recursion tree, the edges between the vertices of V and between the vertices of V^c of the corresponding bipartite graph are mirrored. The modification allows to select an alternating cycle in a connected component cc_i with a minimal $f(i)$ value in each step of the recursion, as we show next. Let cc_i be the selected component and $\mathcal{M}$ be the current perfect matching in the recursion tree. There are three cases regarding the vertices of cc_i: they are either contained in V only, in V^c only, or simultaneously in both sets V and V^c.

(a) $V(cc_i) \cap V(G') = V$, i.e., cc_i consists of vertices of the original graph. Then any alternating cycle in cc_i yields another perfect matching. Let C be an alternating cycle in cc_i and r be any vertex in C. Then there is another alternating cycle C' mirrored to C, i.e., $E(C') = \{e^c \mid e \in E(C)\}$ in a component cc_j with $f(j) = f(i)$. We define $\mathcal{M}' \coloneqq \mathcal{M} \oplus C \oplus C'$. We select $E_1 \coloneqq \{\{r, \mathcal{M}(r)\}, \{r, \mathcal{M}(r)\}^c\}$ and $E_2 \coloneqq \{\{r, u\}, \{r, u\}^c \mid u \in \{N(r) \setminus \{\mathcal{M}(r)\}\}$. The recursion on $G \setminus E_2$ enumerates all perfect matchings containing the matched edge $r\mathcal{M}(r)$ and its mirrored edge. The recursion on $G \setminus E_1$ enumerates all perfect matchings not containing the edge $r\mathcal{M}(r)$ and its mirrored edge; the initial perfect matching for the recursive call is $\mathcal{M}'$.

(b) $V(cc_i) \cap V(G') = V^c$, i.e., cc_i consists of copied vertices only. This case is analog to (a), where C' consists of edges between vertices in V and C of edges between the vertices in V^c.

(c) If cc_i contains vertices of both V and V^c, then there is an alternating cycle containing vertices of V only or an alternating cycle C, such that $uv \in C \Leftrightarrow u^c v^c \in C$ for all edges $uv \in E(G[V])$. The first case is analog to (a). In the second case, we determine $\mathcal{M}' \coloneqq \mathcal{M} \oplus C$ and continue as in (a).

Technically the algorithm does not pass a full graph and perfect matching in each recursive call as arguments. Instead, it uses an additional data structure to revert the changes made during steps (1) to (3). The data structure requires $\mathcal{O}(n + m)$ additional space for all recursive calls

in total [124]. The recursion stops if $\text{trim}(G')$ is the empty graph. Exactly those nodes output the perfect matchings. Figure 3.13 shows one step of the recursion, including the trimming and selection of E_1 and E_2. Here, case (c) applies, which falls back to the case (a).

Proposition 3.36. *All α maximum weight matchings of a weighted bipartite graph $G = (U \uplus V, E)$ with m edges and $s = |U| \leq |V| = t$ can be enumerated with polynomial delay $\mathcal{O}(ms + st \log t + m^2)$ in total time $\mathcal{O}(ms + st \log t + \alpha m)$.*

Proof. Constructing G' and computing G_y requires time $\mathcal{O}(ms + st \log t)$. In each recursive step, we perform a depth-first search to find alternating cycles. This requires time $\mathcal{O}(m)$. The perfect matchings are output at the leaf nodes in the binary recursion tree, which has a depth of $\mathcal{O}(m)$. □

Uno claimed an amortized time of $\mathcal{O}(\log t)$ per perfect matching [124]. To achieve this, the recursion has to be balanced, i.e., each of the two children of a recursion node has to contain at least a quarter of the parent node's perfect matchings (determined by the f-value). If the initial selection of E_1 and E_2 is unbalanced, the edge sets are computed anew. Unfortunately, this process is indistinct and could not be resolved even by contacting the author. Therefore, it remains unclear if and how a polynomial total time of $\mathcal{O}(ms + st \log t + \alpha \log t)$ can be achieved.

Besides the unresolved selection of E_1 and E_2, a drawback of the newer enumeration algorithm for perfect matchings by Uno [124] is that the solutions are output at the leaf nodes only. If we instead use the previous algorithm by Uno [125] in conjunction with Reduction 1), we obtain a polynomial delay of $\mathcal{O}(ms + st \log t)$ without affecting the polynomial total time. This is achieved by analogously ensuring the edges between the vertices of V and between the vertices of V^{c} are mirrored in each recursive step. Then, however, we cannot hope to lower the term αm in the total running time to $\alpha \log t$ in the future.

Proposition 3.37. *All α maximum weight matchings of a weighted bipartite graph $G = (U \uplus V, E)$ with m edges and $s = |U| \leq |V| = t$ can be enumerated with polynomial delay $\mathcal{O}(ms + st \log t)$ in total time $\mathcal{O}(ms + st \log t + \alpha m)$.*

3.7 Summary and Future Work

In this chapter, we first introduced the matching problem and its variants, e.g., the maximum weight matching problem. We briefly discussed the maximum cardinality matching problem on general graphs in Section 3.1. The rest of the chapter discussed the matching problem on weighted bipartite graphs. Firstly, we presented known algorithms for the maximum weight matching problem in Section 3.2. Secondly, we studied the problem on unbalanced bipartite graphs in Section 3.3 and provided improved running times. We further studied the all-cavity maximum weight matching for balanced graphs in Section 3.4 and unbalanced graphs in Section 3.5. We presented results for the MWM problem and new results for the all-cavity MWM problem for integral and real-valued weight functions. We concluded in Section 3.6 with a newly designed polynomial delay algorithm to enumerate all maximum weight matchings in a bipartite graph.

The running time result from Proposition 3.36 to enumerate all MWMs in a bipartite graph $G = (U \uplus V, E)$ with m edges and $s \coloneqq |U| \leq |V| =: t$ states an additional running time of $\mathcal{O}(m)$ per MWM. The result was achieved by a reduction from the enumeration problem of MWMs to MWPMs to perfect matchings. However, we could not prove the same time bound as shown by Uno for the perfect matching problem, which was $\mathcal{O}(t)$ [125] and $\mathcal{O}(\log t)$ [124], respectively. The reason is, we enumerated perfect matching on the equality subgraph. Since the perfect matchings

on the equality subgraph are not unique regarding G, we had to adjust the perfect matching enumeration to ensure each MWM is enumerated precisely once. It remains open if there is a better time bound than $\mathcal{O}(m)$ per matching.

Open Problem 3.1. *Is there a better time bound than $\mathcal{O}(m)$ per enumerated MWM on a bipartite graph G with m edges?*

The Hungarian method with graph doubling and the equality subgraph has the drawback that running time results (partly) depend on the larger vertex set of size t. However, the faster results for the MWM problem in Section 3.3 do not compute the equality subgraph and do not depend on t. It remains open if we can reduce the initial time to compute a graph similar to the equality subgraph on which we can enumerate perfect matchings in a better time bound than $\mathcal{O}(ms + st \log t)$ or if the equality subgraph is necessary at all.

Open Problem 3.2. *Do we require the equality subgraph to enumerate all MWMs in a bipartite graph? If not, can we improve the initial delay of $\mathcal{O}(ms + st \log t)$?*

Nichts ist für mich mehr Abbild der Welt
und des Lebens als der Baum. Vor ihm
würde ich täglich nachdenken, vor ihm
und über ihn...

Christian Morgenstern
1871 – 1914

Chapter 4

Maximum Common Subtree Isomorphisms and Embeddings

The maximum common subgraph problem asks for a graph with a maximum number of vertices that is isomorphic to induced subgraphs of two input graphs.[1] This problem arises in many domains, where it is essential to find the common parts of objects, which can be represented as graphs. An example of this are chemical structures, which can be interpreted directly as labeled graphs. Computing the maximum common subgraph has been studied extensively in cheminformatics [33, 106, 105, 111]. Although elaborated backtracking algorithms have been developed [87, 106], solving large instances in practice is a great challenge. Multi-core parallelism can reduce the computation time, but this is a non-trivial task [55].

The maximum common subgraph problem is NP-hard and remains so even when the input graphs are restricted to trees [45]. When the input and output graphs are restricted to trees, the problem becomes polynomial-time solvable [84]. This problem is then referred to as the *maximum common subtree isomorphism problem*, and the first algorithm solving it in polynomial time is attributed to Edmonds [84]. A generalization to attributed trees has been proposed by Torsello, Rowe, and Pelillo [123] and is referred to as *maximal similarity common subtree*. In this variant, the vertices are labeled, and a positive weight function between the labels is defined. From their definition, a maximal similarity common subtree is a common subtree of maximum weight. The authors proved a running time of $\mathcal{O}(|T|^2|T'|d)$ on unrooted attributed trees T, T' with d as maximum degree among all vertices.

Also requiring that the common subgraph must be connected (or even partially biconnected), several extensions to tree-like graphs have been proposed by us [27] and others [111, 132], primarily for applications in cheminformatics. Some of these approaches are not suitable for practical applications due to high constants hidden in the polynomial running time. Other algorithms are efficient in practice, but restrict the search space to specific common subgraphs. Instead of developing maximum common subgraph algorithms for more general graph classes, which has proven difficult, a different approach represents molecules simplified as trees [104]. Then, vertices typically represent groups of atoms, and their comparison requires to rate the similarity of two vertices by a weight function. Such a weight function, however, is often not supported

[1] This introduction is partly based on the introduction of our contribution in *Largest Weight Common Subtree Embeddings with Distance Penalties* [28].

by algorithms for tree comparison. Moreover, it can be desirable to map a path in one tree to a single edge in the other tree, skipping the inner vertices. Formally, this is achieved by *graph homeomorphism* instead of isomorphism.

Various variants for comparing trees have been proposed and investigated [127]. Most of them assume rooted trees, which may be ordered or unordered. Algorithms tailored to the comparison of evolutionary trees typically assume only the leaves to be labeled, while others support labels on all vertices or do not consider labels at all. The well-known agreement subtree problem, for example, considers the case where only the leaf vertices are labeled, with no label appearing more than once per tree [83]. Gupta and Nishimura [50] investigated the *largest common embeddable subtree* problem in unlabeled rooted trees. Their definition is based on topological embedding (or homeomorphism) and allows to map edges of the common subtree to vertex-disjoint paths in the input trees. The algorithm uses the classical idea to decompose the problem into subproblems for smaller trees, which are solved via bipartite matching. A solution for two trees with at most n vertices is computed in time $\mathcal{O}(n^{2.5} \log n)$ using a dynamic programming approach. Lozano and Valiente [80] investigated the *maximum common embedded subtree* problem based on edge contraction. In both cases, the input graphs are rooted unlabeled trees. Note, the definition of their problems is not equivalent. The first is polynomial-time solvable, while the second is NP-hard for unordered trees, but polynomial-time solvable for ordered trees. Many algorithms do not support trees, where the leaves and the inner vertices both have labels. A notable exception is the approach by Kao et al. [64], where only vertices with the same label may be mapped. This algorithm generalizes Gupta and Nishimura's approach and improves its running time to $\mathcal{O}(\sqrt{d}D \log \frac{2n}{d})$, where D denotes the number of vertex pairs with the same label and d the maximum degree of all vertices. However, the algorithm by Kao et al. is designed for rooted trees only. A straight forward approach to solve the problem for unrooted trees is to try out all pairs of possible roots, which results in an additional $\mathcal{O}(n^2)$ factor for the running time. We present an algorithm we published in [26] that exploits the fact that there are many similar matching problem instances. We show how we can use the all-cavity maximum weight matching approach from Section 3.5 to speed up the computation. In the end, this allows us to achieve a running time for the unrooted case, which matches that of the rooted case.

This chapter is organized as follows. Section 4.1 introduces the maximum common subtree isomorphism problem (MCSI), which asks for a tree isomorphic to subtrees of both input trees. Thereby we include the different variants of this problem, e.g., the support of labels and the possibility to assign weights between pairs of vertices and edges. For the latter, the objective is to find a common subtree of maximum weight. In the running time analysis, we distinguish between integral and real weights. In Section 4.2, we study the rooted MCSI problem, where the roots of the input trees have to be mapped to each other. We present a basic algorithm solving that problem and improve upon that to achieve faster running times. Following that, we analyze the (unrooted) MCSI problem in Section 4.3. Here, we also compare our algorithm to the approach by Edmonds [84]. We use results from Section 3.5 to achieve running times matching those of the rooted problem variants. The following Section, 4.4, is dedicated to lower bounds on the time complexity. In Section 4.5, we study the largest weight common subtree embedding problem, which asks for the largest possible tree embeddable into two input trees and generalizes the maximum common subtree isomorphism problem. As a maximum solution is not unique in general, we study the enumeration of all maximum weight common subtree isomorphisms in Section 4.6. These results are based on our publications [25, 28] and use the enumeration of maximum weight matchings from Section 3.6 as a subroutine. The weight of a maximum weight common subtree isomorphism can be interpreted as the trees' similarity. We discuss this in Section 4.7. Similarity and metrics are closely related. We briefly present results on this topic in Section 4.8. Finally, we discuss open problems and conclude in Section 4.9.

4.1 The Maximum Common Subtree Isomorphism Problem

In the following, we formalize the maximum common subtree isomorphism problem and its variants. This problem is a special case of the maximum common subgraph problem, which itself generalizes the graph isomorphism problem.

Definition 4.1 (Graph Isomorphism). *Let G and H be graphs. An* isomorphism *between G and H is a bijective function $\phi : V_G \to V_H$ such that $uv \in E_G \Leftrightarrow \phi(u)\phi(v) \in E_H$ for all $u, v \in V_G$. If such an isomorphism exists, G and H are said to be* isomorphic *and we write $G \simeq H$.*

This definition implies a bijective function between the edges. Therefore, we define $\phi(uv) \coloneqq \phi(u)\phi(v)$ for any edge $uv \in E_G$. The graph isomorphism problem is unknown to be in P or NP-hard. However, in practice, this problem is well solvable [89]. A recent contribution is about generating random graphs where the graph isomorphism problem is difficult to solve [21]. The first results on the graph isomorphism problem applied to random graphs are at least 40 years old [2]. Graph isomorphism is very strict in the sense that both graphs require to have exactly the same structure. Therefore, we cannot use it as a meaningful similarity measure.

The intermediate problem between graph isomorphism and maximum common subgraph is the subgraph isomorphism problem.

Definition 4.2 ((Induced) Subgraph Isomorphism). *Let G and H be graphs. A* subgraph isomorphism *from G to H is an injective function $\phi : V_G \to V_H$ such that $uv \in E_G \Rightarrow \phi(u)\phi(v) \in E_H$ for all $u, v \in V_G$. We then call G* subgraph isomorphic *to H and write $G \preceq_\phi H$.*

If additionally $uv \in E_G \Leftarrow \phi(u)\phi(v) \in E_H$ for all $u, v \in V_G$, then ϕ is an induced *subgraph isomorphism.*

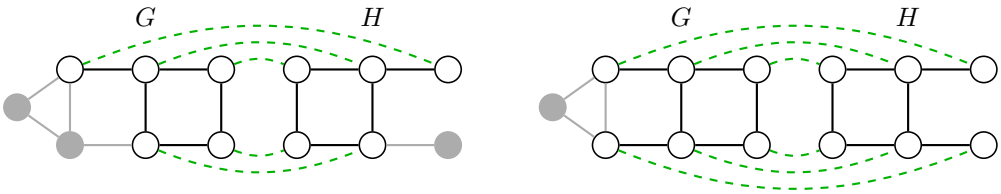
Contrary to the graph isomorphism problem, subgraph isomorphism is known to be NP-hard [19]. This is the case even when G is a forest and H a tree [45], just as when G is a tree and H is outerplanar [120]. On the other hand, for two trees of orders k and n, the induced subgraph isomorphism problem is solvable in time $\mathcal{O}((k^{1.5}/\log k)n)$ [117]. When both graphs are biconnected and outerplanar, induced subgraph isomorphism can be solved in time $\mathcal{O}(n^2)$ [120] and subgraph isomorphism in $\mathcal{O}(n^3)$ [79].

The maximum common subgraph problem naturally generalizes the subgraph isomorphism problem.

Definition 4.3 (Common Subgraph (Isomorphism)). *Let G and H be graphs. Let $G' \subseteq G$ and $H' \subseteq H$ be subgraphs, such that $G' \simeq H'$. Then $G' \simeq H'$ is a* common subgraph *of G and H. An isomorphism ϕ between G' and H' is a* common subgraph isomorphism.

There are four optimization variants regarding the common subgraph $G' \simeq H'$.

1. A connected induced subgraph with the maximum possible number of vertices.
2. A connected subgraph with the maximum possible number of edges.
3. An induced subgraph with the maximum possible number of vertices.
4. A subgraph with the maximum possible number of edges.



(a) Maximum common induced subgraph (b) Maximum common edge subgraph

Figure 4.1: Maximum common induced (a) vs. edge (b) subgraph (black vertices and edges) and isomorphism (green, dashed). Edges and vertices not mapped by the isomorphisms are colored gray.

We refer to the first and third variant as *vertex induced* variants. These are also known as a *(connected) maximum common induced subgraph*. We refer to to the second and fourth as *edge induced* variants. They are also known as a *(connected) maximum common edge subgraph*. The general abbreviation is MCS for maximum common subgraph. Figure 4.1 exemplifies the difference between a maximum common induced subgraph and a maximum common edge subgraph. All four variants can be solved via a reduction to finding a maximum clique in a so-called product graph [12]. We discuss this in Subsection 5.4.3 in conjunction with the block-and-bridge preserving maximum common subgraph on non-outerplanar graphs.

In our definitions, we have the term *common subgraph* on the one hand and *common subgraph isomorphism* on the other hand. The term *common subgraph* is often used equivalently to *common subgraph isomorphism*. Most algorithms regarding the MCS problem, including the ones in this thesis, compute isomorphisms. However, from the computation, we also obtain the common subgraph; more specific, the subgraphs of the input graphs, which are isomorphic to each other. Therefore, the reader should not be confused if we occasionally use one term over the other. For simplicity, we write *common subgraph isomorphism between G and H* instead of *between subgraphs of G and H*. We do this for all further variations of the problem, e.g., when restricted to trees as in the following definition.

Definition 4.4 ((Maximum) Common Subtree Isomorphism). *Let T and T' be trees. A connected common subgraph isomorphism ϕ between T and T' is a* common subtree isomorphism*. If ϕ maps a maximum possible number of vertices, ϕ is a* maximum common subtree isomorphism (MCSI)*. A* rooted MCSI *ϕ is an MCSI between rooted trees T^r and T'^s under the restriction $\phi(r) = s$.*

With trees or rooted trees as input and output, the four optimization variants above break down to a single optimization variant. We can generalize definitions 4.1 to 4.4 to labeled graphs (V, E, l) and (V', E', l'). Then we additionally require $l(v) = l'(\phi(v))$ for each mapped vertex and $l(e) = l'(\phi(e))$ for each mapped edge; i.e., a labeled MCSI is an MCSI that maximizes the number of mapped vertices while respecting the vertex and edge labels.

We can further generalize this by assigning a weight $\omega : \Sigma \times \Sigma \to \mathbb{F}$ to each pair of labels, where $\mathbb{F}$ typically is a field of numbers. In this thesis, we consider ω mapping to the integral numbers $\mathbb{Z}$ and the real numbers $\mathbb{R}$. We restrict the weights to non-negative numbers for some results. We further allow a weight of $-\infty$, which means that the corresponding pair of vertices or edges may not be mapped to each other. We call those pairs *forbidden*. An alternative to $-\infty$ is the common approach to use a weight of $-M$, where M is a sufficient large number. For simplicity, we will omit l and l' for the rest of this thesis and define $\omega(u, v) \coloneqq \omega(l(u), l'(v))$ for any pair of

vertices $(u, v) \in V(T) \times V(T')$. We do the same for edges, i.e., $\omega(e, e') \coloneqq \omega(l(e), l'(e'))$ for any pair of edges $(e, e') \in E(T) \times E(T')$. The weight $\mathcal{W}(\phi)$ of an isomorphism ϕ between T and T' under ω is the sum of the weights $\omega(v, \phi(v))$ and $\omega(e, \phi(e))$ of all vertices and edges mapped by ϕ. A maximum common subgraph isomorphism ϕ under a weight function is one of maximum weight $\mathcal{W}(\phi)$ instead of maximum size $|\operatorname{dom}(\phi)|$.

Definition 4.5 (Maximum Weight Common Subtree Isomorphism). *Let T and T' be trees. Let ω assign a weight to each pair of vertices and each pair of edges between T and T'. A common subtree isomorphism ϕ between T and T' is a* maximum weight common subtree isomorphism (MWCSI) *if there is no other common subtree isomorphism ϕ' with $\mathcal{W}(\phi') > \mathcal{W}(\phi)$.*

We define $\mathcal{W}(T, T')$ as the weight $\mathcal{W}(\phi)$ of an MWCSI ϕ between T and T'. Instead of labels, we could assign (continuous) attribute vectors to the input trees' vertices and edges and then define the function ω on these vectors. Such vectors might be beneficial for practical purposes. However, all the results in this thesis are independent of whether we use discrete labels or attribute vectors to define ω. This is another reason that we defined ω directly on the vertices and edges. There are only a few results regarding the weighted MCSI problem; two results from our publications [28, 26], another from Torsello et al. [123] and Schietgat et al. [111].

To extend the definition of $\mathcal{W}$ to unweighted trees, we define $\omega(u, v) \coloneqq 1$ for all vertices $(u, v) \in V_T \times V_{T'}$ and $\omega(e, e') \coloneqq 0$ for all edges $(e, e') \in E_T \times E_{T'}$. Then, $\mathcal{W}(\phi) = |\operatorname{dom}(\phi)|$ as intended. In the labeled case, we define $\omega(u, v) \coloneqq -\infty$ for all pairs of vertices $l(u) \neq l'(v)$ instead and analog for the edges. It is straightforward to see that the unlabeled case is reducible to the labeled case and the labeled case to the weighted case. Recently McCreesh, Ndiaye, Prosser, and Solnon [85] analyzed different maximum common subgraph algorithms regarding their performance in practice. They showed that the algorithmic choice should depend on whether the edges have labels. Algorithms solving the weighted maximum common subgraph algorithm problem directly are rare. However, we can reduce that problem to the maximum weight clique problem, as we show in Subsection 5.4.3. Recent results for the maximum weight clique problem are discussed in, e.g., [86, 115].

4.2 Rooted Maximum Common Subtree Isomorphism [2]

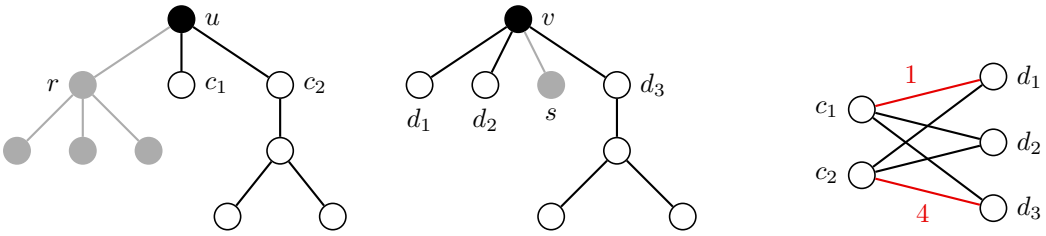
In the following, we introduce the basic techniques to solve the maximum common subtree isomorphism problem following the ideas of Edmonds and Matula [84]. The approach requires to compute maximum weight matchings in bipartite graphs as a subroutine. By fixing the roots of both trees, we can develop an algorithm to solve the rooted MCSI problem. We study the unrooted MCSI problem and Edmond's algorithm in the next section.

Recall Definition 2.13. For a rooted tree T^r, the rooted subtree T^r_u is the subtree induced by u and all its descendants in T^r with root u. Let $x \in V(T)$. Then T^r_u and T^x_u both refer to the same subtree unless x is contained in T^r_u. The key to computing the size of a rooted MCSI between T^r and T'^s is the following recursive formulation:

$$\operatorname{MCSI}_{\text{root}}(T^r, T'^s) = 1 + W(\mathcal{M}), \tag{4.1}$$

where $\mathcal{M}$ is an MWM of the complete bipartite graph on the vertex set $C(r) \uplus C(s)$ with weights $w(uv) = \operatorname{MCSI}_{\text{root}}(T^r_u, T'^s_v)$ for all $u \in C(r)$ and $v \in C(s)$. Hence, each edge weight corresponds

[2] This subsection partly consists of our contribution in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 3, and *Largest Weight Common Subtree Embeddings with Distance Penalties* [28], Section 4.



(a) Rooted subtree T_u^r (b) Rooted subtree $T_v'^s$ (c) MWM on the associated graph

Figure 4.2: (a), (b) Rooted subtrees; gray vertices and edges are not part of the rooted subtrees, root vertices of the subtrees are shown in solid black. (c) The associated bipartite graph and a maximum weight matching in red. Edges without label have weight 1.

to the solution of a problem of the same type for a pair of smaller rooted subtrees, and the recursion naturally stops at the leaves. Each subproblem, the initial and those arising in recursive calls, is uniquely defined by a pair of rooted subtrees and essentially consists of solving a matching instance.

Figure 4.2 illustrates two rooted subtrees T_u^r and $T_v'^s$, the associated bipartite matching graph, and a maximum weight matching on this graph. For the exemplified rooted trees T^r and T'^s, this problem arises on the second level in the recursion of Equation (4.1). We obtain $\mathrm{MCSI}_{\mathrm{root}}(T_u^r, T_v'^s) = 1 + W(\mathcal{M}) = 1 + 1 + 4 = 6$, where $\mathcal{M}$ is the MWM on the biparite graph in Figure 4.2c.

To compute Equation (4.1), we have to solve the subproblems defined by the pairs of rooted subtrees $\mathcal{S}_{\mathrm{root}}(T^r, T'^s) \coloneqq \{(T_u^r, T_v'^s) \mid \mathrm{depth}(u) = \mathrm{depth}(v)\}$.

Proposition 4.6 ([26]). *The size of a maximum common subtree isomorphism between two rooted trees T^r and T'^s can be computed in time $\mathcal{O}(n^3)$, where $n = |T| + |T'|$.*

Proof. For vertices $u, v \in V(T) \times V(T')$, the bipartite graph associated with the subproblem $(T_u^r, T_v'^s)$ contains $k_u + l_v$ vertices, where $k_u \coloneqq |C(u)|$ and $l_v \coloneqq |C(v)|$. We distinguish between $k_u \leq l_v$ and $k_u > l_v$. For $k_u \leq l_v$, we obtain a single MWM in time $\mathcal{O}(k_u l_v (k_u + \log l_v))$ according to Lemma 3.12. Analog, for $k_u > l_v$, we obtain a single MWM in time $\mathcal{O}(k_u l_v (l_v + \log k_u))$.

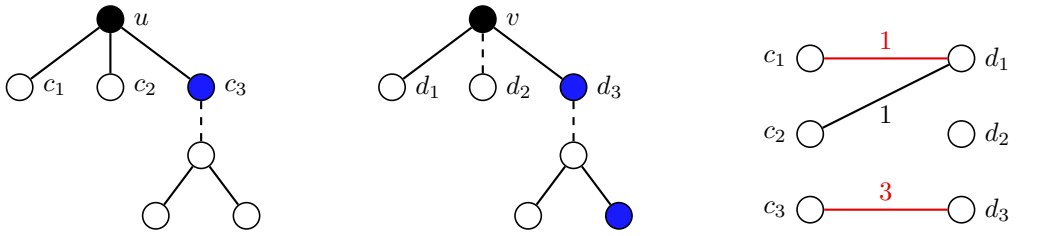
Since $\mathcal{S}_{\mathrm{root}}(T^r, T'^s) \subseteq \{(T_u^r, T_v'^s) \mid u \in V_T, v \in V_{T'}\}$, the total running time is bounded by $\mathcal{O}(n^3)$ as

$$\begin{aligned}
&\sum_{u \in V_T} \sum_{v \in V_{T'}, k_u \leq l_v} k_u l_v (k_u + \log l_v) + \sum_{u \in V_T} \sum_{v \in V_{T'}, k_u > l_v} k_u l_v (l_v + \log k_u) \\
&\quad \leq \sum_{u \in V_T} k_u \sum_{v \in V_{T'}} l_v (l_v + \log l_v) + \sum_{v \in V_{T'}} l_v \sum_{u \in V_T} k_u (k_u + \log k_u) \\
&\quad \leq n \cdot 2n^2 + n \cdot 2n^2 \in \mathcal{O}(n^3). \qquad (4.2)
\end{aligned}$$

□

Note that it is possible to compute a concrete MCSI from the MWMs associated with the computed optimal solution since the matched edges directly transfer to the mapped vertices in the MCSI. We provide more details in Section 4.6, where we enumerate all maximum weight common subtree isomorphisms.

We can improve the result from Proposition 4.6 using Proposition 3.14 for the unbalanced MWM problem. Let $B_{u,v}$ be the bipartite graph associated with the subproblem $(T_u^r, T_v'^s)$. Let



(a) Rooted subtree T_u (b) Rooted subtree T'_v (c) MWM on the graph $B_{u,v}$

Figure 4.3: (a), (b) Rooted subtrees; labels of vertices are indicated by colors, labels of edges by line styles. (c) The associated bipartite graph $B_{u,v}$ and a maximum weight matching in red.

$t \coloneqq \min\{|T|, |T'|\}$. From Equation (4.1), we know the maximum edge weight is bounded by t. Thus we can compute a maximum weight matching on $B_{u,v}$ in time $\mathcal{O}(k_u l_v \sqrt{\min\{k_u, l_v\}} \log t)$. Let $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$. Since $k_u \leq \delta(u)$ and $l_v \leq \delta(v)$, the total time to compute all MWMs is bounded by

$$\begin{aligned}
&\mathcal{O}\left(\sum_{u \in V_T} \sum_{v \in V_{T'}} \delta(u)\delta(v)\sqrt{\min\{\delta(u), \delta(v)\}} \log t\right) \\
&\subseteq \mathcal{O}\left(\sum_{u \in V_T} \delta(u) \sum_{v \in V_{T'}} \delta(v)\sqrt{\Delta} \log t\right) \\
&= \mathcal{O}\left(\sqrt{\Delta} \log t \sum_{u \in V_T} \delta(u)|V(T')|\right) \\
&= \mathcal{O}(|V(T)|\,|V(T')|\,\sqrt{\Delta} \log t). \qquad (4.3)
\end{aligned}$$

This result allows us to provide an improved upper bound to compute an MCSI between two rooted trees.

Theorem 4.7 (Rooted MCSI). *A maximum common subtree isomorphism between two rooted trees T^r and T'^s can be computed in time $\mathcal{O}(|V(T)|\,|V(T')|\,\sqrt{\Delta} \log t)$, where $\Delta = \min\{\Delta(T), \Delta(T')\}$ and $t = \min\{|T|, |T'|\}$.*

4.2.1 Rooted Labeled Maximum Common Subtree Isomorphism

The MCSI problem on labeled rooted trees (T^r, l) and (T'^s, l') requires that only vertices and edges with the same label may be mapped to each other. Consequentially, we only need to solve the MWM problem on those graphs $B_{u,v}$, where $l(u) = l'(v)$. Let $c \in C(u)$ be a child of u and $d \in C(v)$ be a child of v. In the graph $B_{u,v}$, we add the edge cd if and only if $l(c) = l'(d)$ and $l(uc) = l'(vd)$. An example of two labeled rooted subtrees and the associated graph $B_{u,v}$ is depicted in Figure 4.3.

Theorem 4.8 (Rooted Labeled MCSI). *The running time to compute a maximum common subtree isomorphism between labeled rooted trees T^r and T'^s is $\mathcal{O}(D\sqrt{\Delta} \log t + |T|\,|T'|)$, where $t = \min\{|T|, |T'|\}$, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and D is the number of vertex*

$\|T\|$	$\|T'\|$	$\Delta(T)$	$\Delta(T')$	D	Theorem 4.8	Kao et al. [64]
n	n	n	n	n^2	$n^{2.5} \log n$	$n^{2.5}$
n	n	$\mathcal{O}(1)$	n	n^2	$n^2 \log n$	$n^{2.5}$
n	n	$\mathcal{O}(1)$	$\mathcal{O}(1)$	n^2	$n^2 \log n$	$n^2 \log n$
n	n	n	n	n	n^2	$n^{1.5}$
$n^{0.5}$	$n^{1.5}$	$n^{0.5}$	$n^{1.5}$	n^2	$n^{2.25} \log n$	$n^{2.75}$

Table 4.1: Worst-case running times on the rooted MCSI problem for different properties of the input trees T and T'. D is the number of vertex pairs with identical labels.

pairs with identical labels.

Proof. Firstly, the edge weights in the bipartite matching graphs are bounded by t as in the unlabeled case. Secondly, we need to solve an MWM problem only on those graphs $B_{u,v}$, $u \in V(T)$, $v \in V(T')$, where $l(u) = l'(v)$. Thirdly, for each pair of vertices $u \in V(T)$, $v \in V(T')$, where $l(u) \neq l'(v)$, we have one less edge in $B_{p(u),p(v)}$, i.e., in the bipartite graph associated with the parent vertices of u and v. Consequently, for all bipartite graphs $B_{u,v}$, the total number of edges is $\mathcal{O}(D)$. From Proposition 3.14, the time to compute an MWM is $\mathcal{O}(m\sqrt{s}\log t)$ on a graph with m edges and s as the size of the smaller vertex set. Hence, we can bound the time to compute the MWMs as follows.

$$\mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} |E(B_{u,v})|\sqrt{\min\{\delta(u),\delta(v)\}}\log t\right)$$
$$\subseteq \mathcal{O}\left(\sqrt{\Delta}\log t \sum_{u\in V_T}\sum_{v\in V_{T'}} |E(B_{u,v})|\right) \subseteq \mathcal{O}(D\sqrt{\Delta}\log t). \tag{4.4}$$

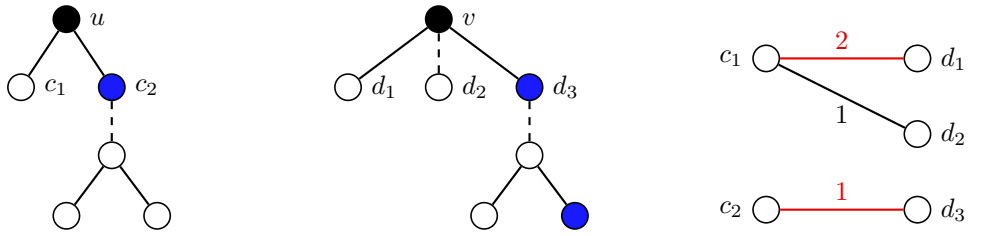
However, it costs $\mathcal{O}(1)$ for each vertex pair $u \in V(T), v \in V(T')$ to check if $l(u) = l'(v)$. Therefore, we add $|V(T)|\,|V(T')|$ to the total running time. □

Kao et al. [64] claimed a time bound of $\mathcal{O}(\sqrt{d}D\log\frac{2n}{d})$, with D as above and d the maximum degree of all vertices. However, D might be very small. Assume two graphs of order $\Theta(n)$, $d \in \Theta(n)$, and $D \in \mathcal{O}(n)$. Then their total time to compute a maximum solution is $\mathcal{O}(n^{1.5})$. To check for all the vertex pairs whether the labels match should take time $\Theta(n^2)$.

An overview of running times for different values of $\Delta(T)$, $\Delta(T')$, D, and different tree sizes between our result (Theorem 4.8) and the result by Kao et al. [64] is presented in Table 4.1. Note, their algorithm does not support edge labels. For graphs with different properties (maximum degree; order of the graphs), their upper bounds are higher than ours.

4.2.2 Rooted Maximum Weight Common Subtree Isomorphism

The maximum weight common subtree isomorphism problem differs from the labeled MCSI problem in the regard that we do not maximize the number of mapped vertices, but the total weight based on the weight function ω between the two trees. Consequently, the weight of matching edges can exceed $\min\{|T|, |T|'\}$. When dealing with a negative weight $\omega(u, v)$ between vertices $u \in |T|$, $v \in |T'|$, we cannot simply stop the recursion on those vertices. This is because descendants of u and v can contribute a positive weight larger than $|\omega(u, v)|$. The formulation to



(a) Rooted subtree T_u (b) Rooted subtree T'_v (c) MWM on the graph $B_{u,v}$

Figure 4.4: (a), (b) Rooted subtrees and a weight function ω as follows: for vertices of the same color, we assume a weight of 2 and for different colors of -3; for edges of the same line style, we assume a weight of 0 and for different styles of -1. (c) The associated bipartite graph $B_{u,v}$ and a maximum weight matching in red. Edges with negative weight are omitted.

compute the weight $\mathcal{W}$ between T^r and T'^s under the weight function ω is

$$\begin{aligned} &\mathrm{MWCSI}_{\mathrm{root}}(T^r, T'^s) = \mathrm{MWCSI}_{\mathrm{root}}(T^r_r, T'^s_s), \text{ and} \\ &\mathrm{MWCSI}_{\mathrm{root}}(T^r_u, T'^s_v) = \omega(u,v) + W(\mathcal{M}), \end{aligned} \tag{4.5}$$

where $\mathcal{M}$ is an MWM of the bipartite graph B with vertex set $C(u) \uplus C(v)$ and edges cd of weight $w(cd) = \mathrm{MWCSI}_{\mathrm{root}}(T^r_c, T'^s_d) + \omega(cu, dv)$ for all $c \in C(u)$ and $d \in C(v)$. By adding $\omega(cu, dv)$ to the weight of each edge $cd \in E(B)$, we include the weight of mapped edges if and only if the incident vertices are mapped to each other. If we allow negative weights in ω, the weight of some edges in B can be negative as well. Since those edges never contribute to an MWM, we may omit them. An example of two rooted subtrees, a weight function ω between its vertices and edges, and the associated graph $B_{u,v}$ is depicted in Figure 4.4.

First, let us assume ω to be integral and bounded by a constant N. This implies a weight of each edge in the associated graphs $B_{u,v}$ of at most $C \coloneqq 2N \cdot \min\{|T|, |T'|\}$, since no more than $2\min\{|T|, |T'|\}$ edges and vertices in total can contribute to the weight. As before, let $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$. From Proposition 3.14 the time bound is to compute all the MWMs is

$$\begin{aligned} &\mathcal{O}\left(\sum_{v \in V_{T'}} \sum_{u \in V_T} \delta(u)\delta(v)\sqrt{\min\{\delta(u), \delta(v)\}} \log C\right) \\ \subseteq\ &\mathcal{O}\left(\sum_{v \in V_{T'}} \delta(v) \sum_{u \in V_T} \delta(u)\sqrt{\Delta} \log C\right) = \mathcal{O}\left(|T|\,|T'|\sqrt{\Delta}\log(N \min\{|T|, |T'|\})\right). \end{aligned} \tag{4.6}$$

Let us assume real weights next. From Proposition 3.13, the time to compute all the MWMs is bounded by

$$\begin{aligned} &\mathcal{O}\left(\sum_{v \in V_{T'}} \sum_{u \in V_T} \delta(u)\delta(v) \min\{\delta(u), \delta(u)\}\right) \\ \subseteq\ &\mathcal{O}\left(\sum_{v \in V_{T'}} \delta(v) \sum_{u \in V_T} \delta(u)\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta). \end{aligned} \tag{4.7}$$

We sum up the results in the following theorem.

Algorithm 2: Edmonds' Maximum Common Subtree

Input : Unlabeled Trees T and T'
Output : Size of an MCSI between T and T'.
Data : Table containing solutions for all subproblems.

1 $m \leftarrow 0$ ▷ *Size of an MCSI*
2 **foreach** $(uv, u'v') \in E(T) \times E(T')$ **do**
3 $\quad m_1 \leftarrow \mathrm{MCSI}_{\mathrm{root}}(T_v^u, T_{v'}^{'u'}) + \mathrm{MCSI}_{\mathrm{root}}(T_u^v, T_{u'}^{'v'})$
4 $\quad m_2 \leftarrow \mathrm{MCSI}_{\mathrm{root}}(T_v^u, T_{u'}^{'v'}) + \mathrm{MCSI}_{\mathrm{root}}(T_u^v, T_{v'}^{'u'})$
5 $\quad m \leftarrow \max\{m, m_1, m_2\}$
6 **return** m

Theorem 4.9 (Rooted MWCSI). *Let T^r and T'^s be rooted trees and let $\Delta = \min\{\Delta(T), \Delta(T')\}$. We can compute a maximum weight common subtree isomorphism between T^r and T'^s under a weight function ω between the vertices and edges in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|,|T'|\}))$ if ω is integral and bounded by N. If ω is real-valued, we can compute it in time $\mathcal{O}(|T|\,|T'|\Delta)$.*

4.3 (Unrooted) Maximum Common Subtree Isomorphism

It is easy to generalize the rooted maximum common subtree isomorphism problem to unrooted input trees T, T' by considering all possible pairs of roots, i.e., we solve

$$\mathrm{MCSI}(T, T') \coloneqq \max\left\{\mathrm{MCSI}_{\mathrm{root}}(T^r, T'^s) \mid r \in V(T), s \in V(T')\right\}. \tag{4.8}$$

We can solve this equation with an additional factor of $\mathcal{O}(|T|\,|T'|)$ to the running time of the rooted MCSI problems. This solution is similar to the approach attributed to Edmonds sketched by Matula [84], which is outlined in Algorithm 2. That algorithm is for unlabeled trees only and is edge-based. The idea is to split the input trees along all pairs of edges (line 2). Then we compute the size of an MCSI between the rooted subtrees on one side of the edges plus the size on an MCSI between the rooted subtrees on the other side (line 3). Then we swap the rooted subtrees in the second tree and do the same (line 4). From all computed solutions, we keep the maximum (line 5). Since Edmonds' algorithm is a dynamic programming approach, intermediate results for each pair of rooted subtrees are stored in a table of size $\mathcal{O}(|T|\,|T'|)$.

We have decided to use a vertex-based approach, since an edge-based approach is difficult to apply to the block-and-bridge preserving maximum common subgraph problem in Chapter 5. However, both Edmonds' algorithm and our approach are based on maximum weight matchings as in Equation (4.5).

Instead of considering all pairs of roots in our vertex-based approach, we still obtain a maximum solution if we fix the root of one input tree as we showed in [26]. We provide the details of this approach next.[3] Following that, we further improve the running time by using algorithms solving the all-cavity maximum weight matching problem.

As in Edmonds' algorithm, a repeated computation can easily be avoided through a lookup table. Let $\textsc{Rt}(T^r) \coloneqq \{T_u^r \mid u \in V(T)\}$ and $\textsc{Rt}(T) \coloneqq \bigcup_{r \in V(T)} \textsc{Rt}(T^r)$. Note that we can uniquely associate the subtree T_u^r with $T_u^{p(u)}$, where $p(u)$ is the parent of u in T^r. For example, in Figure 4.2a,

[3] Partly taken from our contribution in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 3.

Algorithm 3: Maximum Weighted Common Subtree Isomorphism

Input : Trees T and T' under a weight function ω
Output : Weight of an MWCSI between T and T'.
Data : Table $D(u, s, v)$ storing solutions $\mathrm{MWCSI}_{\mathrm{root}}(T_u^r, T_v'^s)$ of subproblems.

```
1  Select an arbitrary root vertex r ∈ V_T.
2  foreach u ∈ V_T in post order traversal on T^r do          ▷ All possible T_u^r ∈ R_T(T^r)
3      U ← C(u) in T^r
4      foreach v ∈ V_T' do
5          foreach s ∈ N(v) ∪ {v} do                          ▷ All possible T_v'^s ∈ R_T(T')
6              V ← C(v) in T'^s
7              if ω(u, v) ≠ −∞ then
8                  B ← bipartite graph with vertices U ⊎ V
9                  foreach pair (u', v') ∈ U × V do
10                     w(u'v') ← ω(uu', vv') + D(u', v, v')
11                 M ← MWM on B
12                 D(u, s, v) ← ω(u, v) + W(M)
13             else D(u, s, v) ← −∞
14 return the maximum entry in D
```

the rooted subtrees $T_{c_2}^r$, $T_{c_2}^u$, and $T_{c_2}^{c_1}$ are the same. Hence, each rooted subtree $T_v^u \in \mathrm{R}_{\mathrm{T}}(T)$ either is the whole tree T with root $u = v$ or is the subtree rooted at v of some edge $uv \in E(T)$, where u is not contained in the subtree. Thus, $\mathrm{R}_{\mathrm{T}}(T) = \{T_v^u \mid v \in V(T) \wedge u \in N(v) \cup \{v\}\}$ is the set of all rooted subtrees of T. In total, the subproblems defined by $\mathcal{S}(T, T') \coloneqq \mathrm{R}_{\mathrm{T}}(T) \times \mathrm{R}_{\mathrm{T}}(T')$ have to be solved.

However, ensuring that each subproblem is solved only once does not allow to improve the bound on the running time, since $\mathcal{S}(T, T')$ still can contain a quadratic number of subproblems of linear size: Let T and T' be two star graphs on n vertices, i.e., trees with all but one vertex of degree one. Each of the $(n-1)^2$ pairs of leaves can be selected as root pair and leads to a different subproblem of size $n - 1$.

We show that it is sufficient to consider only a subset of the subproblems to guarantee that an optimal solution is found. Let

$$\mathrm{MCSI}_{\mathrm{fast}}(T^r, T') \coloneqq \max\left\{\mathrm{MCSI}_{\mathrm{root}}(T_u^r, T'^s) \mid u \in V(T), s \in V(T')\right\}, \tag{4.9}$$

where $r \in V(T)$ is an arbitrary but fixed root of T. To compute Equation (4.9), only the subproblems $\mathcal{S}_{\mathrm{fast}}(T^r, T') \coloneqq \mathrm{R}_{\mathrm{T}}(T^r) \times \mathrm{R}_{\mathrm{T}}(T') \subseteq \mathcal{S}(T, T')$ need to be solved.

Lemma 4.10 ([26]). *Let $MCSI_{fast}$ and $MCSI$ be defined as above and $r \in V(T)$ arbitrary but fixed, then $MCSI_{fast}(T^r, T') = MCSI(T, T')$ for all trees T, T'.*

Proof. Let ϕ be an MCSI. If r is in the domain of ϕ, then $\mathcal{W}(\phi) = \mathrm{MCSI}_{\mathrm{root}}(T^r, T'^{\phi(r)}) = \mathrm{MCSI}_{\mathrm{fast}}(T^r, T')$. Otherwise, the domain of ϕ is contained in the subtree rooted at one child of r. Let u be the unique vertex that is closest to r and mapped by ϕ. Then $\mathcal{W}(\phi) = \mathrm{MCSI}_{\mathrm{root}}(T_u^r, T'^{\phi(u)}) = \mathrm{MCSI}_{\mathrm{fast}}(T^r, T')$. □

Algorithm 3 implements this strategy, where the postorder traversal on T^r (line 2) ensures that the solutions to smaller subproblems are always available when required (line 10). The algorithm

is independent of the MCSI variant (unlabeled, labeled, weighted). We use the weight function ω according to the problem, cf. the paragraph below Definition 4.5. The lookup table contains one entry for each subproblem in $\mathcal{S}_{\text{fast}}(T^r, T')$ and hence requires space $\mathcal{O}(|T|\,|T'|)$. The restriction of the considered subproblems allows improving the bound on the running time.

Proposition 4.11 ([26]). *Algorithm 3 solves the maximum weight common subtree isomorphism problem for two trees T and T' in time $\mathcal{O}(n^4)$, where $n = |T| + |T'|$.*

Proof. According to Lemma 4.10, computing Equation (4.9) yields the optimal solution, and it suffices to solve the subproblems $\mathcal{S}_{\text{fast}}(T^r, T')$ as realized by Algorithm 3. Let k_u be the number of children of u in T_u^r and l_v^s the number of children of v in $T'^s, s \in V(T')$. The subproblems $\mathcal{S}_{\text{fast}}(T, T')$ can be solved in a total time of

$$\begin{aligned} &\mathcal{O}\left(\sum_{u\in V_T}\sum_{s\in V_{T'}}\sum_{v\in V_{T'}} k_u l_v^s \min\{k_u, l_v^s\}\right) \\ &= \mathcal{O}\left(\sum_{u\in V_T}\sum_{s\in V_{T'}}\sum_{v\in V_{T'}} \delta(u)\delta(v)\min\{\delta(u),\delta(v)\}\right) \subseteq \mathcal{O}\left(\sum_{s\in V_{T'}} n^3\right) \subseteq \mathcal{O}\left(n^4\right). \end{aligned} \tag{4.10}$$

□

We could give improved upper bounds for each of the MCSI variants (unlabeled, labeled, weighted) similar to the bounds in Section 4.2, but we leave that to the next subsection, where we introduce additional speed-up techniques based on the relation between the bipartite matching graphs.

Unrooted MCSI and the All-Cavity Maximum Weight Matching Problem. [4] In the following we show the relation between the MCSI problem and the one-sided all-cavity maximum weight matching problem and then give improved time bounds for each MCSI variant. For each pair $u \in V_T$, $v \in V_{T'}$ of vertices, selected in lines 2 and 4 of Algorithm 3, the algorithm computes up to $|N(v)| + 1$ MWMs, cf. lines 5 and 11. A close look reveals this is indeed a one-sided all-cavity maximum weight matching problem: The set U in line 3 is the fixed vertex set. The set V in line 6 consists of the vertices $N(v)$, if $s = v$, and $N(v) \setminus \{s\}$ otherwise. The edge weights depend on the choice of u and v, but not on the choice of s. The choice of s only affects which vertices and edges are present in the graph. The following result is from our findings for the MWCSI problem [26], which is based on Proposition 3.24.

Proposition 4.12 ([26]). *All the MWMs in Algorithm 3 can be computed in total time $\mathcal{O}(|T|\,|T'|(\min\{\Delta(T), \Delta(T')\} + \log\max\{\Delta(T), \Delta(T')\}))$.*

Proof. For each pair $(v, u) \in V(T) \times V(T')$, we solve the one-sided all-cavity maximum weight matching problem on the graphs $(C(v) \uplus N(u), E)$ with edge weights as determined by Equation (4.5). Let $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$ and $d \coloneqq \max\{\Delta(T), \Delta(T')\}$. For all those pairs, we

[4] Partly taken from our contribution in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 4, and *Largest Weight Common Subtree Embeddings with Distance Penalties* [28], Section 5; improved by results from Section 3.5 of this thesis.

obtain a time complexity to compute all the MWMs of

$$\begin{aligned}
&\mathcal{O}\left(\sum_{v\in V_T}\sum_{u\in V_{T'}}\delta(v)\delta(u)(\min\{\delta(v),\delta(u)\}+\log\max\{\delta(v),\delta(u)\})\right)\\
\subseteq\ &\mathcal{O}\left(\sum_{v\in V_T}\delta(v)\sum_{u\in V_{T'}}\delta(u)(\Delta+\log d)\right)\\
=\ &\mathcal{O}\left((\Delta+\log d)\sum_{v\in V_T}\delta(v)|T'|\right)=\mathcal{O}(|T|\,|T'|(\Delta+\log d)).
\end{aligned}\tag{4.11}$$

□

In our newer work [28], we showed how to improve the time bound of the MWCSI problem to $\mathcal{O}(|T|\,|T'|\Delta)$. To do this, we solved the one-sided all-cavity MWM on unbalanced graphs by a series of shortest path problems instead of solely relying on the Hungarian algorithm. The details are presented in Subsection 3.5.2 with the final result in Proposition 3.25. When applying that result to the MWCSI problem, we obtain the following upper time bound for the latter.

$$\begin{aligned}
&\mathcal{O}\left(\sum_{v\in V_T}\sum_{u\in V_{T'}}\delta(v)\delta(u)(\min\{\delta(v),\delta(u)\}\right)\\
\subseteq\ &\mathcal{O}\left(\sum_{v\in V_T}\delta(v)\sum_{u\in V_{T'}}\delta(u)\Delta\right)=\mathcal{O}(|T|\,|T'|\Delta).
\end{aligned}\tag{4.12}$$

Theorem 4.13 (MWCSI). *Let T and T' be trees and $\Delta \coloneqq \min\{\Delta(T),\Delta(T')\}$. We can compute a maximum weight common subtree isomorphism between T and T' under a real-valued weight function ω between the vertices and edges in time $\mathcal{O}(|T|\,|T'|\Delta)$.*

An open question of [28] was whether we might improve the time bound for the unrooted MWCSI problem when the weights are integral and bounded by N. We can answer this question positively now. In Theorem 3.31, we have shown that we can solve the all-cavity MWM problem in time $\mathcal{O}(\min\{s^2,m\}\sqrt{s}\log N+m)$. The weight of each edge in the associated graphs $B_{u,v}$ is at most $C \coloneqq 2N\cdot\min\{|T|,|T'|\}$, since no more than $2\min\{|T|,|T'|\}$ edges and vertices in total can contribute to the weight. As before, let $\Delta \coloneqq \min\{\Delta(T),\Delta(T')\}$. When applying that result to the integral MWCSI problem, we obtain the following upper time bound for the latter.

$$\begin{aligned}
&\mathcal{O}\left(\sum_{v\in V_T}\sum_{u\in V_{T'}}\min\{\min\{\delta(v),\delta(u)\}^2,\delta(v)\delta(u)\}\sqrt{\min\{\delta(v),\delta(u)\}}\log C+\delta(v)\delta(u)\right)\\
\subseteq\ &\mathcal{O}\left(\sum_{v\in V_T}\sum_{u\in V_{T'}}\delta(v)\delta(u)\sqrt{\min\{\delta(v),\delta(u)\}}\log C\right)\\
\subseteq\ &\mathcal{O}\left(\sum_{v\in V_T}\delta(v)\sum_{u\in V_{T'}}\delta(u)\sqrt{\Delta}\log C\right)\\
=\ &\mathcal{O}\left(\sqrt{\Delta}\log t\sum_{v\in V_T}\delta(v)|T'|\right)=\mathcal{O}(|T|\,|T'|\,\sqrt{\Delta}\log(N\min\{|T|,|T'|\})).
\end{aligned}\tag{4.13}$$

Variant	Time bound
MCSI	$\mathcal{O}(nn'\sqrt{\Delta}\log s)$
Labeled MCSI	$\mathcal{O}(nn' + D\sqrt{\Delta}\log s)$
MWCSI, integral	$\mathcal{O}(nn'\sqrt{\Delta}\log(Ns))$
MWCSI, real-valued	$\mathcal{O}(nn'\Delta)$

Table 4.2: Worst-case running times on the different maximum common subtree isomorphism variants (unlabeled, labeled, weighted) on input trees T and T', where $n \coloneqq |T|$, $n' \coloneqq |T'|$, $s \coloneqq \min\{n, n'\}$, and $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$. N is the maximum integral weight, and D is the number of vertex pairs with identical labels. Running times apply to both the rooted and unrooted MCSI variants.

Theorem 4.14 (MWCSI, integral weights). *Let T and T' be trees. Let ω be integral and bounded by N. We can compute a maximum weight common subtree isomorphism between T and T' in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|, |T'|\}))$, where $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$.*

We obtain the following result for the labeled MCSI problem by applying Theorem 3.31 analogously.

Theorem 4.15 (Labeled MCSI). *We can compute a maximum common subtree isomorphism between labeled trees T and T' in time $\mathcal{O}(D\sqrt{\Delta}\log t + |T|\,|T'|)$, where D is the number of vertex pairs with identical labels, $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$, and $t = \min\{|T|, |T'|\}$.*

The result for unlabeled trees is as follows.

Theorem 4.16 (MCSI). *A maximum common subtree isomorphism between two trees T and T' can be computed in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log t)$, where $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$ and $t = \min\{|T|, |T'|\}$.*

We summarize the results for all the rooted and unrooted MCSI variants in Table 4.2.

4.4 Lower Bounds on the Time Complexity and Optimality [5]

Providing a tight lower bound on the time complexity of a problem is generally a non-trivial task. We obtain such a bound for trees of bounded degree and argue why the existence of an algorithm with sub-cubic running time for unrestricted trees is unlikely. To solve the MWCSI with an arbitrary weight function ω for two trees T and T', all values $\omega(u, v)$ for $u \in V(T)$ and $v \in V(T')$ must be considered. This directly leads to the lower bound of $\Omega(|T|\,|T'|)$ for the time complexity of the MWCSI problem. For trees of bounded degree our approach achieves running time $\mathcal{O}(|T|\,|T'|)$ according to Theorem 4.13 and, thus, has an optimal worst-case running time in the considered setting.

For unrestricted trees of order n, our approach has a running time of $\mathcal{O}(n^3)$ according to Theorem 4.13. In the next paragraph, we present a linear time reduction from the assignment

[5] From our contribution in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 5.

problem to the MWCSI problem, which preserves the time complexity. Therefore, solving the MWCSI problem in time $o(n^3)$ yields an algorithm to solve the assignment problem in time $o(n^3)$. The Hungarian method solves the assignment problem in $\mathcal{O}(n^3)$, which is the best-known time bound for bipartite graphs with $\Theta(n^2)$ edges of unrestricted weight for more than 30 years.

Let $B = (U \uplus V, E, w)$ be a weighted bipartite graph on which we want to solve the assignment problem in its maximum variant, i.e., to find a maximum weight perfect matching. We assume weights to be non-negative, which can be achieved by adding a sufficiently large constant to every edge weight to obtain an assignment problem that is equivalent w.r.t. the MWPMs. We construct a star graph T with center c and leaves U and another star graph T' with center c' and leaves V. Let $n = |U| = |V|$ and $N = \max\{w(e) \mid e \in E\}$. We define ω such that $\omega(u, v) = w(uv) + nN$ for all $uv \in E$, $\omega(c, c') = nN$, and $\omega(u, v) = -\infty$ for all other pairs of vertices. For all pairs of edges, we define $\omega(e, e') = 0$. Let ϕ be an MWCSI between T and T' w.r.t. w. It directly follows from the construction that $\mathcal{M} \coloneqq \{uv \in E \mid \phi(u) = v\}$ is an MWM in B with $W(\mathcal{M}) = \mathcal{W}(\phi) - |\operatorname{dom}(\phi)|nN$. Furthermore, the incremented weights ensure that $\mathcal{M}$ is perfect, i.e., $|\operatorname{dom}(\phi)| = n + 1$, whenever B admits a perfect matching. Therefore, we obtain:

Proposition 4.17. *Only if we can solve the assignment problem on a graph with n vertices and $\Theta(n^2)$ edges of unrestricted weight in time $o(n^3)$ we can solve the MWCSI problem on two unrooted trees of order $\Theta(n)$ in time $o(n^3)$.*

4.5 Largest Weight Common Subtree Embedding [6]

In this section, we study the largest common subtree embedding problem. The task is to find a largest possible tree embeddable into two input trees, and the problem generalizes the maximum common subtree isomorphism problem. Thereby we consider all cases from Section 4.2 and 4.3, i.e., rooted and unrooted trees, unlabeled and labeled trees, and a weight function ω between the vertices and edges. Further variants of the problem in labeled and unlabeled rooted trees have been studied, e.g., for comparing evolutionary trees.

We build on the basic ideas of Gupta and Nishimura [50]. To prevent arbitrarily long paths that are mapped to a common edge, we study a linear distance penalty for paths of length greater than 1. By choosing a high distance penalty, we solve the rooted maximum common subtree isomorphism problem as a special case. By choosing weight 1 for equal labels and sufficiently small negative weights otherwise, we solve a problem equivalent to the one studied by Kao et al. [64]. Our initial algorithm [28] for unrooted trees builds on recent results by Ramshaw and Tarjan [103, 102] for unbalanced matchings. We improve those results by our findings for the all-cavity maximum weight matching problem described in Section 3.5. Contrary to the similar algorithms by Gupta and Nishimura and Kao et al., we support arbitrary weights between the vertices and edges, a linear penalty for skipped vertices, and unrooted trees.

This section is organized as follows. We begin by introducing Gupta and Nishimura's algorithm in Subsection 4.5.1. In Subsection 4.5.2, we study the additional support of a weight function and skipped vertices. Following that, we generalize to unrooted trees in Subsection 4.5.3, which is essential, e.g., in comparing chemical structures, as these naturally have no root vertex.

[6] This section consists partly of our findings in *Largest Weight Common Subtree Embeddings with Distance Penalties* [28] with additional results based on the all-cavity maximum weight matching problem from Chapter 3.

4.5.1 Gupta and Nishimura's Algorithm [7]

In this section, we formally define a *Largest Common Subtree Embedding* (LaCSE) and present a brief overview of Gupta and Nishimura's algorithm to compute such an embedding. The following two definitions are based on the work by Gupta and Nishimura [50].

Definition 4.18 (Topological Embedding). *A rooted tree T is* topologically embeddable *in a rooted tree T' if there is an injective function $\psi : V(T) \to V(T')$, such that $\forall a, b, c \in V(T)$*

i) If b is a child of a, then $\psi(b)$ is a descendant of $\psi(a)$.

ii) For distinct children b and c of a the paths from $\psi(a)$ to $\psi(b)$ and from $\psi(a)$ to $\psi(c)$ have exactly $\psi(a)$ in common.

T is root-to-root topologically embeddable *in T' if $\psi(r(T)) = r(T')$.*

In a topological embedding, we map edges in one rooted tree to disjoint paths in another rooted tree.

Definition 4.19 ((Largest) Common Subtree Embedding). *Let T and T' be rooted trees and S be topologically embeddable in both T and T'. For such a rooted tree S, let $\psi : V(S) \to V(T)$ and $\psi' : V(S) \to V(T')$ be topological embeddings.*

- *Then $\varphi : \psi(V_S) \to \psi'(V_S)$, $\varphi(v) = \psi' \circ \psi^{-1}(v)$ is a* common subtree embedding.
- *If there is no other tree S' topologically embeddable in both T and T' with $|S'| > |S|$, then S is a* largest common embeddable subtree*, and φ is a* largest common subtree embedding.
- *A common subtree embedding with $\varphi(r(T)) = r(T')$ is a* root-to-root CSE.
- *A root-to-root CSE is* largest *if it is largest among all root-to-root CSEs.*

The definition of a LaCSE specifically requires two rooted trees as input. The MCSI problem, in comparison, accepts unrooted trees as input.

Algorithm by Gupta and Nishimura. Gupta and Nishimura presented an algorithm to compute the size of a largest common embeddable subtree based on dynamic programming, which is similar to the computation of a rooted maximum common subtree isomorphism as in Section 4.2. Let T and T' be rooted trees, and $\mathcal{L}$ be a table of size $|T|\,|T'|$. For each pair of vertices $u \in T, v \in T'$, the value $\mathcal{L}(u, v)$ stores the size of a LaCSE between the rooted subtrees T_u and T'_v. Gupta and Nishimura proved that an entry $\mathcal{L}(u, v)$ is determined by the maximum of the following three quantities.

- $M_1 = \max\{\mathcal{L}(u, c) \mid c \in C(v)\}$
- $M_2 = \max\{\mathcal{L}(b, v) \mid b \in C(u)\}$
- $M_3 = W(\mathcal{M}) + 1$, where $\mathcal{M}$ is an MWM of the complete bipartite graph $B \coloneqq (C(u) \uplus C(v), \{bc \mid b \in C(u), c \in C(v)\})$ with edge weights $w(bc) = \mathcal{L}(b, c)$ for all edges $bc \in E(B)$.

[7] This subsection consists mainly of our contribution in *Largest Weight Common Subtree Embeddings with Distance Penalties* [28], Section 3.

Here, M_1 represents the case where the vertex v is not mapped. To satisfy ii) from Definition 4.18, we may map at most one child $c \in C(v)$. M_2 represents the case where u is not mapped, and at most one child $b \in C(u)$ is allowed. M_3 represents the case $\varphi(u) = v$. To maximize the number of mapped descendants, we compute a maximum weight matching, where the children of u and v are the vertex sets $C(u)$ and $C(v)$, respectively, of a bipartite graph. The edge weights are determined by the previously computed solutions, i.e., the largest common subtree embeddings between the children of u and v and their descendants, namely between T_b and T'_c for each pair of children (b, c). The algorithm proceeds from the leaves to the roots. From the above recursive formula, we get $\mathcal{L}(u, v) = 1$ if u or v is a leaf.

A maximum value in the table yields the size of a largest common subtree embedding. We obtain the size of a root-to-root LaCSE from M_3 of the root vertices $r(T), r(T')$. Note, with storing $\mathcal{O}(|\mathcal{L}|)$ additional data, it is easy to obtain a (root-to-root) LaCSE φ.

Proposition 4.20 ([50]). *Computing a largest common subtree embedding between two rooted trees of order at most n is possible in time $\mathcal{O}(n^{2.5} \log n)$.*

4.5.2 Largest Weight Common Subtree Embeddings

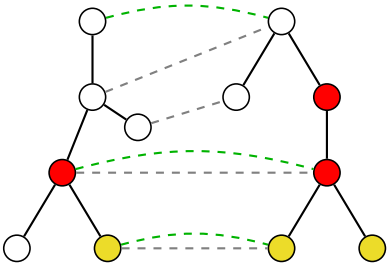
First, we introduce weighted common subtree embeddings between rooted trees. Part of the input is an integral or real weight function between all pairs of vertices and edges. This is similar to the rooted maximum weight common subtree isomorphism problem. We also consider a linear distance penalty for skipped vertices in the input trees. After formalizing the problem and presenting an algorithm, we prove new upper time bounds.

Vertex and edge weights. Instead of maximizing the number of mapped vertices, we want to maximize the sum of the weights $\omega(u, \varphi(u))$ of all vertices u mapped by a common subtree embedding φ. The labeled variant is then defined analog to the labeled MCSI problem, i.e., $\omega(u, v) = 1$ for vertices u, v with the same label and $\omega(u, v) = -\infty$, otherwise.

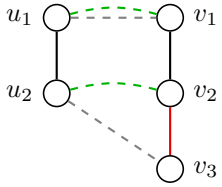
Since in an embedding, inner vertices on mapped paths do not contribute to the weight, we do the same with edges. In other words, both paths need to have length 1 to be considered. Again, we want to maximize the weight $\omega(e, \varphi(e))$ of all edges (paths of length 1) mapped to each other (additional to the mapped vertices' weight).

Distance penalties. Depending on the application purpose, it might be desirable that the paths between the mapped vertices do not have an arbitrary length. Here, we introduce a linear distance penalty for paths of length greater than 1. More precisely, each inner vertex on a path in the input trees corresponding to an edge in the associated common embeddable subtree lowers the embedding's weight by a given penalty p. By assigning p the value ∞ or a sufficiently large number, we effectively compute a rooted maximum (weight) common subtree. The following definition formalizes a LaCSE under a weight function ω and a distance penalty p.

Definition 4.21 (Largest Weight Common Subtree Embedding). *Let T and T' be rooted trees. Let ω be a weight function between the vertices and between the edges of T and T'. Let φ be a common subtree embedding between T and T'. Let $p \in \mathbb{R}^{\geq 0} \cup \{\infty\}$ be a distance penalty. We refer to a path $P = (u_0, e_1, u_1, \ldots, u_k)$ in the tree T corresponding to a single edge in the associated common embeddable subtree as a* topological path. *Let $\varphi(P)$ be the corresponding path $(v_0, e'_1, v_1, \ldots, v_l)$ in T', i.e., $\varphi(u_0) = v_0$, $\varphi(u_k) = v_l$. Then*



(a) Vertex labeled LaCSE (black dotted) and root-to-root LaCSE (green dashed).



(b) The black dotted embedding has weight 1.7, since the vertex v_2 is skipped and, therefore, the penalty p is applied. The green dashed embedding is of largest weight, 5; 2 from the vertices, 3 from the topological path (u_1, u_2) mapped to (v_1, v_2).

Figure 4.5: (a) Vertex labeled LaCSE and root-to-root LaCSE. (b) Two weighted embeddings; one with a skipped vertex, the other where the edge labels contribute to the weight.

- $\omega_p(P, \varphi(P)) = \omega(e_1, e'_1)$, *if* $l = k = 1$, *or*
- $\omega_p(P, \varphi(P)) = -p \cdot (l + k - 2)$, *otherwise.*
- *The* weight $\mathcal{W}(\varphi)$ *is the sum of the weights* $\omega(u, \varphi(u))$ *of all vertices* u *mapped by* φ *plus the weights* $\omega_p(P, \varphi(P))$ *of all topological paths* P.
- *If* φ *is of largest weight among all common subtree embeddings, then* φ *is a* Largest Weight Common Subtree Embedding *(LaWeCSE).*

The definition of root-to-root LaWeCSE is analog to Definition 4.19. A closer look at the definition of ω_p reveals that each inner vertex (the skipped vertices) on a topological path or its mapped path subtracts p from the embedding's weight. Figure 4.5 exemplifies different common subtree embeddings.

The dynamic programming approach. To compute a LaWeCSE, we need to store some additional data during the computation. In Gupta and Nishimura's algorithm, there is a table $\mathcal{L}$ of size $|T|\,|T'|$ to store the weight of LaCSEs between subtrees of the input trees. In our algorithm, we need a table $\mathcal{L}$ of size $2|T||T'|$. An entry $\mathcal{L}(u, v, t)$ stores the weight of a LaWeCSE between the rooted subtrees T_u and T'_v of *type* $t \in \{\curlywedge, \diamond\}$. Type $\curlywedge$ represents a *root-to-root* embedding between T_u and T'_v; $\diamond$ an embedding, where u or v is *skipped*. Skipped in a sense that at least one of u, v will be an inner vertex when mapping some additional ancestor vertices of u or v during the dynamic programming. For type $\diamond$, we subtract the penalty p from the weight for the skipped vertices before storing it in our table. We obtain the weight of a LaWeCSE and a root-to-root LaWeCSE, respectively, from the maximum value of type $\curlywedge$ and $\mathcal{L}(r(T), r(T'), \curlywedge)$, respectively. The following lemma specifies the recursive computation of an entry $\mathcal{L}(u, v, t)$.

Lemma 4.22. *Let* $u \in V(T)$ *and* $v \in V(T')$. *For* $t \in \{\curlywedge, \diamond\}$ *let* $M_t^T = \max\{\mathcal{L}(b, v, t) \mid b \in C(u)\}$ *and* $M_t^{T'} = \max\{\mathcal{L}(u, c, t) \mid c \in C(v)\}$. *Then*

- $\mathcal{L}(u, v, \diamond) = \max\{M_\diamond^T, M_\curlywedge^T, M_\diamond^{T'}, M_\curlywedge^{T'}\} - p$

Let $B = (C(u) \uplus C(v), C(u) \times C(v))$ *be a bipartite graph with edge weights* $w(bc) = \max\{\mathcal{L}(b, c, \diamond), \mathcal{L}(b, c, \curlywedge) + \omega(ub, vc)\}$ *for each pair* $(b, c) \in C(u) \times C(v)$. *Then*

- $\mathcal{L}(u, v, \curlywedge) = \omega(u, v) + W(\mathcal{M})$, *where* $\mathcal{M}$ *is a maximum weight matching on* B.

Proof. Since we defined the maximum of an empty set as $-\infty$, this covers the base case, where one vertex is a leaf, e.g., if u is a leaf, then $\max\{M^T_\diamond, M^T_\curlywedge, M^{T'}_\diamond, M^{T'}_\curlywedge\} = \max\{M^{T'}_\diamond, M^{T'}_\curlywedge\}$. Further, $W(\mathcal{M}) = 0$ if G has no positive weight edges. Then $\mathcal{L}(u, v, \curlywedge) = \omega(u, v)$.

Type $\diamond$ represents the case of an embedding between T_u and T'_v, which is not root-to-root. From the definition of M^T_t, the vertex u is skipped, and from the definition of $M^{T'}_t$, the vertex v is skipped, so it is indeed not root-to-root. Since either u or v was skipped, we subtract the penalty p. This ensures we have taken inner vertices of later steps of the dynamic programming into account.

Type $\curlywedge$ implies that u is mapped to v. Each edge in B represents the weight of a LaWeCSE from one child of u to one child of v. A maximum weight matching yields the best combination, which satisfies Definition 4.18. If a child b of u is mapped to a child c of v, the paths bu and cv have length 1. Then from Definition 4.21, we have to add the weight $\omega(bu, cv)$. Otherwise, at least one path has a length greater than one, and we have to subtract the distance penalty p for each inner vertex. We already did that while computing $\mathcal{L}(b, c, \diamond)$. □

Time and space complexity. We next analyze upper time and space bounds. Thereby we distinguish between real- and integer-valued weight functions ω. If we use dynamic programming starting from the leaves to the roots, we need to compute each value $\mathcal{L}(u, v, t)$ only once.

Theorem 4.23. *Let* T *and* T' *be rooted trees under a weight function* ω. *Let* $\Delta = \min\{\Delta(T), \Delta(T')\}$ *and* p *be a distance penalty.*

- *A LaWeCSE between* T *and* T' *can be computed in time* $\mathcal{O}(|T|\,|T'|\Delta)$ *and space* $\mathcal{O}(|T|\,|T'|)$.
- *If the weights are integral and bounded by a constant* N, *a LaWeCSE can be computed in time* $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|, |T'|\}))$.

Proof. We observe that the entries of type $\curlywedge$ in table $\mathcal{L}$ dominate the computation time. We assume the bipartite graphs on which we compute the MWMs to be complete. We further observe that each edge bc representing the weight of a LaWeCSE between the subtrees T_b and T'_c is contained in precisely one of the matching graphs.

Let us assume real weights first. $\mathcal{L}$ requires $\mathcal{O}(|T|\,|T'|)$ space. We can compute each MWM within the same space bound. This proves the total space bound. From Proposition 3.13, the time to compute all the MWMs is bounded by

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T} |C(u)|\,|C(v)|\min\{|C(u)|, |C(v)|\}\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}} |C(v)|\sum_{u\in V_T} |C(u)|\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$

Let us assume ω to be integral and bounded by a constant N next. This implies a weight of each single matching edge of at most $C \coloneqq 2N \cdot \min\{|T|, |T'|\}$ since no more than $2\min\{|T|, |T'|\}$ edges and vertices in total can contribute to the weight. Negative weight edges never contribute

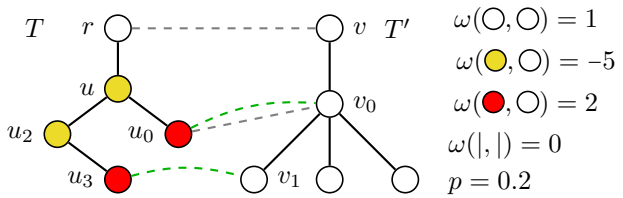


Figure 4.6: The weight of a LaWeCSE between T^r and T'^v is 2.8 (black dotted lines), since we have the skipped vertex u for penalty 0.2. The weight of a LaWeCSE between T^{u_0} and T'^{v_0} is 3.6 (green dashed lines) for two skipped vertices. The latter one is also a LaWeCSE_u.

to an MWM and may safely be omitted. From Proposition 3.14, the time bound is

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T}|C(u)|\,|C(v)|\sqrt{\min\{|C(u)|,|C(v)|\}}\log C\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}}|C(v)|\sum_{u\in V_T}|C(u)|\sqrt{\Delta}\log C\right) = \mathcal{O}\left(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|,|T'|\})\right).$$

□

4.5.3 Unrooted Largest Weight Common Subtree Embeddings

In this section, we consider the LaWeCSE problem between unrooted trees, which is as follows.

> **Problem 4.1 (Unrooted Largest Weight Common Subtree Embedding).** *Given trees T and T' under a weight function ω, and a distance penalty p, determine a LaWeCSE φ between T^r and T'^s, where r and s are selected such that $\mathcal{W}(\varphi)$ is maximum among all possible choices of r and s. We denote this as the* LaWeCSE_u *problem.*

By choosing a high distance penalty, we effectively solve the (unrooted) maximum common subtree isomorphism problem. In the following, we firstly present a basic algorithm to solve the LaWeCSE_u problem and an improvement by fixing the root of T. Following that, we speed up the computation by exploiting similarities between the different chosen roots of T'. We further prove the correctness and upper time bounds of our algorithms.

Basic algorithm and fixing one root. The basic idea is to compute a (rooted) LaWeCSE from T^u to T'^v for each pair of vertices $(u,v) \in V(T)\times V(T')$ and output a maximum solution. Albeit correct, the time bound is $\mathcal{O}(|T|^2\,|T'|^2\Delta)$. In Section 4.3, we showed how to compute a maximum common subtree isomorphism between unrooted trees by arbitrarily choosing one root vertex r of T and then computing MCSIs between T^r and T'^s for all $s \in V(T')$. The key idea in the proof is that for any maximum common subtree isomorphism φ between T and T', either r is mapped by φ, or there exists a unique vertex $u \in V(T)$ with the shortest distance to r, such that all vertices mapped by φ are contained in T_u. The dynamic programming approach for the (rooted) LaWeCSE problem already considers the maximum solutions between the rooted subtrees of T^r and T'^s. However, Figure 4.6 shows that this strategy alone sometimes fails if we want to find a

LaWeCSE_u between the input trees. A LaWeCSE between T^r and T' rooted at any vertex results in a weight of at most 2.8. In contrast, rooting T at u_0 results in a LaWeCSE of weight 3.6.

In a maximum common subtree between trees T and T', let $r \in V(T)$ be an arbitrarily chosen root of T. If any two children u_1, u_2 of their parent vertex $u \in V(T)$ are mapped to vertices of T', then u is also mapped. This statement is independent of the chosen root $r \in V(T)$ since a common subtree is connected. If we want to compute a LaWeCSE, the statement is also true (for the given root). This follows from Definition 4.18 ii). However, if we choose u_1 as root, we may skip u and map u_2, forming the topological path (u_1, u, u_2). Whatever we do, if we skip vertex u as an inner vertex of a topological path, this is the only path containing u; otherwise, we violate Definition 4.18. We record this as a lemma.

Lemma 4.24. *Let T and T' be unrooted trees. Let φ be a LaWeCSE$_u$ from T to T' and $u \in V(T)$ be an inner vertex of a topological path with its neighbors $N(u) = \{u_1, u_2, \ldots, u_k\}$. Then φ maps vertices from precisely two of the rooted subtrees $T^u_{u_1}, \ldots, T^u_{u_k}$ to T'.*

To compute a LaWeCSE_u φ, additionally to the strategy from Section 4.3, we need to cover the case that there exists no single vertex u mapped by φ, such that all vertices mapped by φ are contained in T^r_u, cf. Figure 4.6 with r as chosen root. In this case, let u be the unique inner vertex of a topological path P, such that all vertices mapped by a LaWeCSE_u φ are contained in T^r_u. An example is the yellow vertex u in Figure 4.6. Further, let $P_1 = (u_0, \ldots, u_{i-1}, u_i = u, u_{i+1}, \ldots, u_k)$ be the topological path containing u and $\varphi(P_1) = P_2 = (v_0, \ldots, v_l)$ with $\varphi(u_0) = v_0$ and $\varphi(u_k) = v_l$.

Then there is a LaWeCSE ϕ_1 between the rooted subtrees $T^r_{u_{i-1}}$ and $T'^{v_1}_{v_0}$ containing u_0 and all its descendants mapped by φ. There is another LaWeCSE ϕ_2 between the rooted subtrees $T^r_{u_{i+1}}$ and $T'^{v_0}_{v_1}$ containing u_k and all its descendants mapped by φ. Choosing $T'^{v_{j+1}}_{v_j}$ and $T'^{v_j}_{v_{j+1}}$ for any $j \in \{0, \ldots, l-1\}$ as rooted subtrees yields the same LaWeCSEs ϕ_1 and ϕ_2, i.e., it does not matter where we split the path P_2.

For any vertex $v \in V(T')$, let $\mathcal{L}^v$ refer to the table $\mathcal{L}$ corresponding to T^r and T'^v. Then $L_1 \coloneqq \max\{\mathcal{L}^{v_1}(u_{i-1}, v_0, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_1 minus the penalty for the inner vertices $u_1, \ldots, u_{i-1}$; the penalty is 0 if $i = 1$. $L_2 \coloneqq \max\{\mathcal{L}^{v_0}(u_{i+1}, v_1, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_2 minus the penalty for the inner vertices $u_{i+1}, \ldots, u_{k-1}$ and $v_1, \ldots, v_{l-1}$. In other words, the penalty p for each inner vertex on the paths excluding u is included in $L_1 + L_2$, and we obtain $\mathcal{W}(\varphi) = L_1 + L_2 - p$. Before summarizing this strategy in Lemma 4.25, we exemplify it in Figure 4.6.

The green dashed lines depict the LaWeCSE_u φ with mapping $u_0 \mapsto v_0$ and $u_3 \mapsto v_1$. The yellow inner vertex u fulfills the condition that T^r_u contains all vertices mapped by φ. We have paths $P_1 = (u_0, u, u_2, u_3)$ and $\varphi(P_1) = P_2 = (v_0, v_1)$. Then $L_1 = \mathcal{L}^{v_1}(u_0, v_0, \curlywedge) = 2$ for the mapping $u_0 \mapsto v_0$. Further $L_2 = \mathcal{L}^{v_0}(u_2, v_1, \diamond) = 1.8$ for the mapping $u_3 \mapsto v_1$ and the skipped vertex u_2. We obtain $\mathcal{W}(\varphi) = L_1 + L_2 - p = 2 + 1.8 - 0.2 = 3.6$.

Lemma 4.25. *Let T and T' be trees. Let $r \in V(T)$ be arbitrarily chosen. Let $\mathcal{W}(T^r, T'^v)$ be the weight of a LaWeCSE from T^r to T'^v and $\mathcal{T}(u, v, w) \coloneqq \max\{\mathcal{L}^w(u, v, t) \mid t \in \{\curlywedge, \diamond\}\}$. Then the weight of a LaWeCSE$_u$ is the maximum of the following two quantities.*

- $M_1 = \max\{\mathcal{W}(T^r, T'^v) \mid v \in V(T')\}$
- $M_2 = \max_{u \in V(T),\ vw \in E(T')}\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\} - p$

We next analyze the running time and space to compute a LaWeCSE_u following the improved strategy of Lemma 4.25.

Lemma 4.26. *Let the preconditions be as in Lemma 4.25. Then M_1 can be computed in time $\mathcal{O}(|T|\,|T'|^2\Delta)$ and M_2 in time $\mathcal{O}(|T|\,|T'|)$; both can be computed in space $\mathcal{O}(|T|\,|T'|)$.*

Proof. For any vertices $s, v \in V(T')$ consider the rooted subtree T'^s_v. By definition, $p(v)$ is the parent vertex of v with $p(s) = s$. Then $T'^s_v = T'^{p(v)}_v = T'^w_v$ for each vertex $w \in V(T') \setminus V(T'^s_v)$, i.e., w is a vertex of T', which is not contained in the rooted subtree T'^s_v. Therefore, we can identify each table entry $\mathcal{L}^s(u, v, t)$ by $\mathcal{L}^{p(v)}(u, v, t)$. In other words, all the table entries needed to compute M_1 are determined first by a vertex $u \in V(T)$, and second by either an edge $wv \in E(T')$ or the root vertex s. Therefore, the space needed to store all the table entries and thus compute M_1 is $\mathcal{O}(|T|\,|T'|)$. We will use these values to retrieve $\mathcal{T}(u, v, w)$ in constant time.

From Theorem 4.23, for each $v \in V(T')$, we can compute $\mathcal{W}(T^r, T'^v)$ in time $\mathcal{O}(|T|\,|T'|\Delta)$. Thus, the time for M_1 is bounded by $\mathcal{O}(|T|\,|T'|^2\Delta)$. For any edge $vw \in E(T')$, we observe that the only rooted subtrees from T' to consider are T'^v_w and T'^w_v. Let $L(b, v)$ and $L(b, w)$, $b \in C(u)$, be the weight of a LaWeCSE from T^r_b to T'^w_v and T'^v_w, respectively. Let B be a bipartite graph with vertices $C(u) \uplus \{v, w\}$ and edges between these vertices with weights defined by $L(b, v)$ and $L(b, w)$, respectively. Let $\mathcal{M}$ be an MWM_2 on B, i.e., a matching of maximum weight among all matchings of cardinality 2. Then $W(\mathcal{M}) = \max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$. This follows from the construction of B. From Lemma 3.17, which also holds for an MWM_2, and Lemma 3.18, the time to compute $W(\mathcal{M})$ and thus $\max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$ for given u and vw is $\mathcal{O}(|C(u)|)$. The time to compute M_2 is $\mathcal{O}(\sum_{u \in V_T, vw \in E_{T'}} |C(u)|) = \mathcal{O}(|T|\,|T'|)$.

We can compute M_2 from $\mathcal{L}$ and additional space $\mathcal{O}(|T|)$, which is $\mathcal{O}(|T|\,|T'|)$ space in total. □

In the following, we improve the running time from $\mathcal{O}(|T|\,|T'|^2\Delta)$ to $\mathcal{O}(|T|\,|T'|\Delta)$. To this end, we need to speed up the computation in Lemma 4.22. Specifically, we exploit similarities between the graphs on which we compute the maximum weight matchings. We further need to speed up the computation of $M_t^{T'}$ related to the root vertices from T'. We have to take special care of the sequence in which we compute the table entries to avoid circular dependencies.

Speeding up the computation of $M_t^{T'}$ in Lemma 4.22. The recursion in this lemma computes maximum values among certain table entries. We first include the current root $s \in V(T')$ into the notation. We use the previous definition of $\mathcal{L}^s$ in this subsection referring to the table where s is the root of $V(T')$. Let $u \in V(T)$ and $v \in V(T')$ be the vertices in the current recursion of Lemma 4.22. For all $t \in \{\curlywedge, \diamond\}$ let $M_t^{T,s} = \max\{\mathcal{L}^s(b, v, t) \mid b \in C(u)\}$ and $M_t^{T',s} = \max\{\mathcal{L}^s(u, c, t) \mid c \in C(v)\}$. By definition, $T'^s_v = T'^w_v$ for each vertex $w \in V(T') \setminus V(T'^s_v)$. This implies $M_t^{T,s} = M_t^{T,w}$ and $M_t^{T',s} = M_t^{T',w}$ for all w as before. In other words, it is sufficient to distinguish all $M_t^{T,s}$ and $M_t^{T',s}$ first by a vertex $u \in V(T)$, and second by either an edge $wv \in E(T')$ or a single vertex $s \in V(T')$.

This observation allows us to upper bound the time to compute all values $M_t^{T,s}$ by

$$\mathcal{O}\left(\sum_{u \in V_T, wv \in E_{T'}} |C(u)| + \sum_{u \in V_T, s \in V_{T'}} |C(u)|\right) = \mathcal{O}\left(\sum_{wv \in E_{T'}} |T| + \sum_{s \in V_{T'}} |T|\right) = \mathcal{O}\left(|T|\,|T'|\right).$$

Let $N(v) = \{c_1, c_2, \ldots c_l\}$ and $C_i \coloneqq \{c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_l\}$ for $1 \leq i \leq l$, i.e., C_i contains all the vertices from $N(v)$ except c_i. We observe $M_t^{T',v} = \max\{\mathcal{L}^v(u, c, t) \mid c \in N(v)\}$ and $M_t^{T',c_i} = \max\{\mathcal{L}^{c_i}(u, c, t) \mid c \in C_i\} = \max\{\mathcal{L}^v(u, c, t) \mid c \in C_i\}$ for each $i \in \{1, \ldots, l\}$. Let j be an index, such that $M_t^{T',v} = \mathcal{L}^v(u, c_j, t)$. Then for each $i \neq j$ we have $M_t^{T',c_i} = M_t^{T',v}$. Therefore,

we can compute $M_t^{T',v}$ and M_t^{T',c_i} for all $i \in \{1,\dots,l\}$ in time $\mathcal{O}(\delta(v))$. Hence, the time to compute all the $M_t^{T',s}$ is bounded by

$$\mathcal{O}\left(\sum_{u\in V_T, v\in V_{T'}} \delta(v)\right) = \mathcal{O}\left(\sum_{u\in V_T} |T'|\right) = \mathcal{O}(|T|\,|T'|).$$

Lemma 4.27. *Assume there is a sequence of all pairs $(u,v) \in V_T \times V_{T'}$ such that all necessary values are available to compute $M_t^{T,s}$ and $M_t^{T',s}$ for $s \in N(v) \cup \{v\}$. Then the total time to do this is bounded by $\mathcal{O}(|T|\,|T'|)$.*

Exploiting similarities between the matching graphs. In Lemma 4.22, we need to compute an MWM for each pair of vertices $(u,v) \in V(T) \times V(T')$. When considering all roots $s \in V(T')$, we have one bipartite graph B with vertices $C(u) \uplus N(v)$, $N(v) = \{c_1,\dots,c_l\}$, and l graphs B_{c_i}, $1 \le i \le l$. A graph B_c, $c \in N(v)$, is the same as B except that the vertex c and incident edges are removed. This follows analog to the observation regarding $M_t^{T',v}$ from the previous paragraph. Computing an MWM on each of these graphs is precisely the one-sided all-cavity maximum weight matching problem. Let $s \coloneqq \min\{\delta(u),\delta(v)\}$ and $t \coloneqq \max\{\delta(u),\delta(v)\}$. From Theorem 3.31, the time bound to compute an MWM on B and B_c for all $c \in N(v)$ is $\mathcal{O}(s^2 t)$. However, we yet have to show that there is a sequence such that all the edge weights are available to solve the one-sided all-cavity maximum weight matching problem. We record this intermediate result in a lemma.

Lemma 4.28. *Assume there is a sequence of all pairs (u,v) such that all necessary values are available to compute the MWMs of the LaWeCSE$_u$ approach; then the total time is $\mathcal{O}(|T|\,|T'|\Delta)$.*

Proof. Under the given assumption, the time to compute all the MWMs is

$$\mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\min\{\delta(u),\delta(v)\}\right) \subseteq \mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$

□

Computation sequence. For given vertices $(u,v) \in V(T) \times V(T')$, we call the bipartite graph B without removed vertices as *main instance* and the matching graphs B_c as its *sub instances*. Analog for $t \in \{\curlywedge, \diamond\}$ we define $M_t^{T',v}$ as main instance and $M_t^{T',c}$ for each $c \in N(v)$ as its *sub instances*.

Let $u \in V(T)$ and $vw \in E(T')$. We observe, for type $t \in \{\curlywedge, \diamond\}$, the following values depend circularly on each other. To compute $M_t^{T',w}$ recursively for the vertices (u,w) we need $\mathcal{L}^w(u,v,t)$. Computing $\mathcal{L}^w(u,v,t)$ requires $M_t^{T',v}$ computed recursively for the vertices (u,v). The latter one requires $\mathcal{L}^v(u,w,t)$. Finally, $\mathcal{L}^v(u,w,t)$ requires $M_t^{T',w}$ computed recursively for the vertices (u,w), which was the start of the circular dependency.

We further observe the MWMs depend on table entries of both types. We can break the dependencies by solving at most one sub instance before solving the main instance, as shown next.

We iterate over all roots $s \in V(T')$ and compute a rooted LaWeCSE between T^r and T'^s as in Lemma 4.22. During the recursion on vertices (u,v), the following cases may occur.

1. The first instance to compute on (u, v) is a main instance. Then we instantly compute all its sub instances from it.

2. The first instance to compute on (u, v) is a sub instance. Then we compute only that sub instance without deriving it from the main instance.
   a) If the second instance is a main instance, we instantly compute its sub instances.
   b) Otherwise, let $c_1 \in N(v)$ and $c_2 \in N(v)$ be the vertices corresponding to the first and second sub instance, respectively. Let us consider table entries first. When we computed the sub instance corresponding to c_1, all necessary table entries for $M_t^{T',v}$ except $\mathcal{L}^v(u, c_1, t)$ were available. For the second sub instance, $\mathcal{L}^v(u, c_1, t)$ is also available. Thus we may instantly compute the main instance and all other sub instances, including the one corresponding to c_2. We may argue analog for computing the MWMs.

Theorem 4.29. *A LaWeCSE$_u$ between trees T and T' under a weight function ω and a distance penalty p can be computed in time $\mathcal{O}(|T|\,|T'|\Delta)$ and space $\mathcal{O}(|T|\,|T'|)$, where $\Delta = \min\{\Delta(T), \Delta(T')\}$.*

If the weight function between the edges and vertices maps to integral numbers of at most N, we obtain the following result.

Theorem 4.30. *A LaWeCSE$_u$ between trees T and T' under an integral weight function ω upper bounded by N and an integral distance penalty p can be computed in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|, |T'|\}))$, where $\Delta = \min\{\Delta(T), \Delta(T')\}$.*

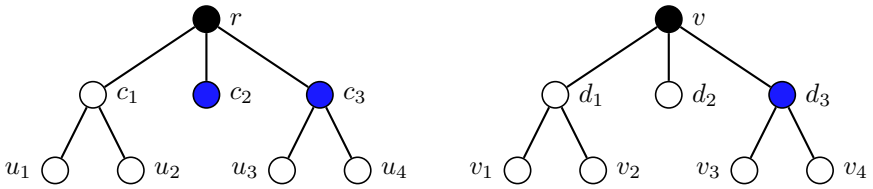
Proof. The proof is analog to the real-valued case. To solve the LaWeCSE$_\mathrm{u}$ problem, we compute the variables $M_t^{T,s}$ and $M_t^{T',s}$ as in Lemma 4.27. This is possible in time $\mathcal{O}(|T|\,|T'|)$. Additionally, we solve a sequence of maximum weight matching and MWM$_2$ problems. The MWM$_2$ problems are also solvable in time $\mathcal{O}(|T|\,|T'|)$. For the computation sequence, as shown before, we replace the computation times for the MWM and one-sided all-cavity MWM problem by those for integral weights, cf. Theorem 3.31. Note, the distance penalty p is not bounded since its value is subtracted from the corresponding edge weights, and negative weight edges may be omitted for the MWM problem. □

Running time results for the labeled and unlabeled case are analog to the maximum common subtree isomorphism problem. In summary, the running times in Table 4.2 are also valid for the LaWeCSE$_\mathrm{u}$ problem.

4.6 Polynomial Delay Enumeration of Maximum Common Subtree Isomorphisms [8]

In this section, we present a polynomial delay algorithm to enumerate all maximum weight common subtree isomorphisms between two input trees with a weight function between their vertices and edges.

[8] Partly based on our findings in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 6.



(a) T (b) T'

Figure 4.7: (a), (b) Vertex labeled input trees. There are in total 4 different MCSIs ϕ_i, $i \in \{1, \ldots, 4\}$ with 7 mapped vertices each. $\phi_i(r) = v$, $\phi_i(c_1) = d_1$, $\phi_i(c_3) = d_3$ for all i. The vertices $u_1, \ldots, u_4$ may be mapped in 4 different manners to $v_1, \ldots, v_4$; e.g., $\phi_1(u_1) = v_2$, $\phi_1(u_2) = v_1$, $\phi_1(u_3) = v_3$, $\phi_1(u_4) = v_4$.

4.6.1 Example and Basic Approach

In Section 3.6, we presented two reductions to enumerate all maximum weight matchings in a bipartite graph. Since the MWCSI problem relies on the computation of MWMs, the enumeration of MWMs is a key to enumerate all MWCSIs. As mentioned before, Algorithm 3 can easily be modified to output the weight of an MWCSI and also an associated isomorphism ϕ. Such a solution ϕ is computed as follows. Let $D(u, s, v)$ be any maximum entry in D, i.e., $D(u, s, v) = \mathcal{W}(T, T')$. Then $\phi(u) = v$. Further mappings are defined by the computed maximum weight matching occurring in Equation (4.5). In the example of Figure 4.7 (for simplicity a labeled MCSI), $D(r, v, v)$ is a maximum entry. Therefore, we have $\phi(r) = v$ and obtain $\phi(c_1) = d_1$ and $\phi(c_3) = d_3$. These steps are repeated for the MWMs defined by the rooted subtrees induced by the vertex pairs (c_1, d_1) and (c_3, d_3).

The basic idea to enumerate all MWCSIs is to compute first the weight $\mathcal{W}(T, T')$ of an MWCSI and fill the table D. Then for each table entry $D(u, v, v) = \mathcal{W}(T, T')$, where $u \in V_T$ and $v \in V_{T'}$, all the different rooted MWCSIs on the rooted subtrees $T_u^r, T_v'^v$ are listed. Note, we omit maximum table entries $D(u, s, v)$, where $s \neq v$. We may do this because $T_v'^s$ is a subtree of $T_v'^v$, and, consequently, every common subtree isomorphism of the rooted subtrees T_u^r and $T_v'^s$ is also a common subtree isomorphism of the rooted subtrees T_u^r and $T_v'^v$. As an example in Figure 4.7 $D(u, v, v) = D(u, d_2, v) = 7$. For both table entries, we obtain the same four different MWCSIs as in the caption of the figure. In the following, we describe the enumeration tree in more detail.

4.6.2 The Enumeration Tree for the MWCSI Problem

The recursion in the enumeration tree occurs as follows. Firstly, we determine the weight $\mathcal{W}_{\max} \coloneqq \mathcal{W}(T, T')$ of an MWCSI. To do this, we use the Hungarian method with graph doubling to compute the necessary MWMs. On all the bipartite graphs on which we compute an MWM, we also compute the equality subgraph G_y and store it in additional data structures. This allows us to use Reduction 1) from Section 3.6 to enumerate all other MWMs. Then the recursion in the enumeration tree is as follows.

1. For all pairs $u \in V_T$, $v \in V_{T'}$, where $D(u, v, v) = \mathcal{W}_{\max}$, set $\phi(u) = v$ and perform step 2.
2. For all non-empty MWMs $\mathcal{M} = \{c_1 d_1, \ldots, c_k d_k\}$ from Reduction 1) corresponding to the rooted subtrees T_u^r and $T_v'^v$ perform step 3. Note, the number of edges k may vary per MWM.
3. For all edges $\{c_i, d_i\} \in \mathcal{M}$, $i \in \{1, \ldots, k\}$, set $\phi(c_i) = d_i$ and recursively perform step 2 corresponding to $T_{c_i}^r$ and $T_{d_i}'^v$. Do this simultaneously for all edges in $\mathcal{M}$, i.e., enumerate all

combinations of all the MWMs as above.

The output of solutions occurs whenever the recursion reaches the leaves, i.e., there are no further matching graphs. For Figure 4.7, we set $\phi(r) = v$. The only MWM yields $\phi(c_1) = d_1$ and $\phi(c_3) = d_3$. This is as before. The subproblems contain two different MWMs each. For the rooted subtrees $T^r_{c_1}$ and $T'^v_{d_1}$, we have $\mathcal{M}_{1,1} = \{u_1v_1, u_2v_2\}$ and $\mathcal{M}_{1,2} = \{u_1v_2, u_2v_1\}$. Analogously, for the rooted subtrees $T^r_{c_3}$ and $T'^v_{d_3}$, we have $\mathcal{M}_{2,1} = \{u_3v_3, u_4v_4\}$ and $\mathcal{M}_{2,2} = \{u_3v_4, u_4v_3\}$. Therefore, we have four different combinations, which yield the additional mapped vertices. The output occurs for each combination since we reached the leaves.

The enumerated solutions are pairwise different between two different maximum entries in D. The proof is similar to the proof of Lemma 4.10. Thus we do not enumerate an MWCSI twice. Further, we do not omit an MWCSI because we consider all necessary maximum table entries and their rooted subtrees and all possible expansions along the MWMs.

The enumeration of all the MWMs associated with a node in the recursion tree can happen multiple times. In the example of Figure 4.7, the matchings $\mathcal{M}_{2,1}$ and $\mathcal{M}_{2,2}$ are enumerated twice: Once for $\mathcal{M}_{1,1}$ and once for $\mathcal{M}_{1,2}$. Only after all different combinations of MWMs associated with the children have been enumerated, the next MWM in the bipartite graph associated with their common parent node is enumerated.

Weight zero matching edges. Edges of weight zero in a matching graph correspond to additional vertices U mapped by the isomorphism, which leave the weight unchanged. If we are interested in a single MWCSI only, we can arbitrarily decide to add the vertices U to the solution or not. In the enumeration process, this choice is not arbitrary. By allowing weight zero edges, we can enumerate different isomorphisms all of the same weight: one isomorphism for each subset of U, e.g., no, all, or some vertices from U. This is a recursive process, i.e., within the vertices of U can be further subsets corresponding to weight zero edges in other matching graphs. By not allowing weight zero edges, we might miss out on some maximum solutions. However, the following running time analysis is not affected by weight zero edges.

4.6.3 Running Time Analysis

Let T and T' be trees, $\Delta \coloneqq \min\{\Delta(T), \Delta(T')\}$, and $d \coloneqq \max\{\Delta(T), \Delta(T')\}$. From Theorem 4.13, we can compute a maximum weight common subtree isomorphism between T and T' under a weight function ω in time $\mathcal{O}(|T|\,|T'|\Delta)$. However, to use Reduction 1), we need the equality subgraph, and thus the upper bound increases to $\mathcal{O}(|T|\,|T'|(\Delta + \log d))$. Thereby we store the computed perfect matching for each equality subgraph.

From Proposition 3.37 and given an initial perfect matching in the equality subgraph, the time to enumerate subsequent MWMs in a pair of rooted subtrees T^r_u, T^s_v is $\mathcal{O}(\delta(u)\delta(v))$. For each MWCSI ϕ, several MWMs may have to be enumerated. Let that number be m_ϕ. We showed in [26], that the total time to enumerate the necessary MWMs is bounded by $\mathcal{O}(\min\{|T|\Delta(T'), \Delta(T)|T'|\})$ as follows.

Let the ith bipartite graph on which we enumerate the MWMs, $i \in \{1, \ldots, m_\phi\}$, contain k_i vertices from T and l_i vertices from T'. Then $\sum_i k_i \leq |T|$ and $\sum_i l_i \leq |T'|$, because all the vertices in all the m_ϕ bipartite graphs are pairwise disjoint. The total time to enumerate an MWM in each bipartite graph can then be bounded by $\sum_i k_i l_i \leq \sum_i k_i \Delta(T') \leq |T|\Delta(T')$ and $\sum_i k_i l_i \leq \sum_i \Delta(T) l_i \leq \Delta(T)|T'|$. Hence, the time to enumerate ϕ is bounded by $\mathcal{O}(\min\{|T|\Delta(T'), \Delta(T)|T'|\})$.

We obtain the following proposition with the initial time to compute a single MWCSI and the above result.

Proposition 4.31 ([26]). *We can enumerate all maximum weight common subtree isomorphisms of trees T and T' with polynomial delay, where $\mathcal{O}(|T|\,|T'|(\Delta + \log d))$ is the time for the first output and $\mathcal{O}(\min\{|T|\Delta(T'), \Delta(T)|T'|\})$ for each subsequent output.*

The above result assumes the worst case, in which for each solution, we traverse the enumeration tree from a leaf back to the root and forth to another leaf. Unfortunately, this worst case can indeed occur, e.g., if the input trees are paths or most matching graphs contain a single maximum weight matching only. However, we can improve the polynomial total time by case analysis on the bipartite matching graphs, where we sum up the amortized time to output a single solution.

1. Bipartite graphs with a single MWM only: Then we store that MWM and can add the mapped vertices in constant time using a reference.
2. Bipartite graphs with more than one MWM: Then we enumerate the MWMs using Reduction 1), cf. Proposition 3.37.

In the first case, the total time per isomorphism can be bounded by $\mathcal{O}(\min\{|T|, |T'|\})$, since this is the maximum possible depth of the enumeration tree. In the second case, we ascend in such a node at most every second time we enumerate an MWM in that node. With amortized time analysis and assuming the lowest nodes contain the computational most demanding problems [9], the total time per solution can be bounded by $\mathcal{O}(\Delta(T)\Delta(T'))$: the worst-case time to compute an additional MWM; on average, for case two, we have to compute at most 2 MWMs per solution.

In total, we obtain the following result.

Theorem 4.32. *We can enumerate all maximum weight common subtree isomorphisms of trees T and T' with polynomial delay as in Proposition 4.31 and in polynomial total time $\mathcal{O}(|T|\,|T'|(\Delta + \log d) + \alpha(\min\{|T|, |T'|\} + \Delta(T)\Delta(T')))$, where α is the number of MWCSIs, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and $d = \max\{\Delta(T), \Delta(T')\}$.*

There is no straightforward approach to transfer this result to integral weighted or even unlabeled graphs while achieving a better time bound. The time to enumerate the matchings is independent of the problem type since we enumerate perfect matchings in the (unweighted) equality subgraph. The faster algorithms from Section 3.5, which solve the all-cavity maximum weight matching problem, do not compute the necessary equality subgraphs to perform Reduction 1). Achieving improved time bounds to enumerate solutions for the restricted MCSI variants is future work.

4.7 Similarity

In Definitions 4.1 through 4.5, we covered the isomorphism, subtree isomorphism, and maximum common subtree isomorphism problem. We indicated that the similarity between two graphs could be related to the maximum size of a common subtree, or more generally, a maximum common subgraph or a largest weight common subtree embedding. This topic and its application have been investigated in, e.g., [64, 33, 104, 132]. There are other similarity measures, such as graph edit distance [40], where the similarity between two graphs depends on the number of singular operations[10] to transform one graph into the other, or fingerprints [105], where features of a graph are encoded into a vector. Further similarity measures are based on neighborhood matching and spectral analysis [40].

[9] For any node b in the recursion tree with its parent a, the enumeration of MWMs in the bipartite graph associated to b occurs at least as often as the enumeration of MWMs in the bipartite graph associated to a.

[10] Insertion and deletion of edges and of isolated vertices as well as relabeling

ID	Coefficient	Range	Reference
1	$\frac{\mathcal{W}}{k+l-\mathcal{W}}$	$[0,1]$	Wallis, Shoubridge, Kraetzl, and Ray (2001)
2	$\frac{\mathcal{W}}{\max\{k,l\}}$	$[0,1]$	Bunke and Shearer (1998)
3	$\frac{\mathcal{W}}{\min\{k,l\}}$	$[0,1]$	Ellis, Furner-Hines, and Willett (1993)
4	$\frac{2\mathcal{W}}{k+l}$	$[0,1]$	Johnson; Raymond and Willett (2002)
5	$\frac{\mathcal{W}^2}{kl}$	$[0,1]$	Johnson; Raymond and Willett (2002)
6	$\frac{\mathcal{W}}{2k+2l-3\mathcal{W}}$	$[0,1]$	Sokal and Sneath; Ellis, Furner-Hines, and Willett (1993)
7	$\frac{\mathcal{W}(k+l)}{2kl}$	$[0,1]$	Kulczynski; Ellis, Furner-Hines, and Willett (1993)
8	$\frac{k\mathcal{W}+l\mathcal{W}-kl}{kl}$	$[-1,1]$	McConnaughey; Ellis, Furner-Hines, and Willett (1993)

Table 4.3: Cost-based similarity coefficients, cf. [105], Table 2. $\mathcal{W}$ is the size of the maximum common subgraph, k, l are the sizes of the input graphs.

For each of these similarity measures, a normalized value is preferable. Table 4.3 lists different coefficients based on the sizes of the input graphs and the maximum common subgraph. These coefficients are also applicable to the MCSI, MWCSI, and LaWeCSE_u problem and also to the block-and-bridge preserving maximum common subgraph problem from Chapter 5. To obtain a value in the range specified in the table, the weight function should be defined such that $\mathcal{W}(T,T) \geq \mathcal{W}(T,T') \leq \mathcal{W}(T',T')$ for any trees (or molecular graphs) T and T', i.e., a graph should be most similar to itself.

Since graphs often represent real-world structures, the similarity value should be consistent with what is deemed similar by, e.g., a ground truth or common agreement. While this is a bit informal, we hold this as a property.

Property 4.33. *Given a reference object r and a set of objects C, a* perfect *similarity function S satisfies the following property:*

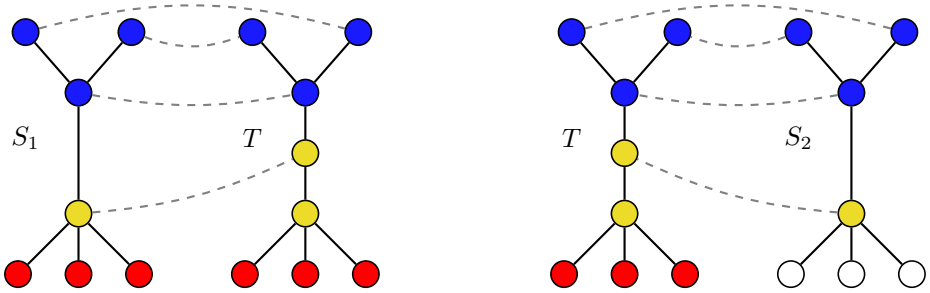
$$\forall a, b \in C. \quad S(a,r) > S(b,r) \text{ if and only if } a \text{ is considered more similar to } r \text{ than } b \text{ to } r.$$

For example, two molecules (represented by molecular graphs) with the same properties should have a normalized similarity value of 1 or near 1, while two completely different molecules should have a similarity of 0 or −1, depending on the coefficient, cf. Table 4.3. We shall note that similarity is subjective and difficult to quantify [54], and this should be considered in Property 4.33.

While we can compute an MWCSI in polynomial time, the restriction to connected solutions can be detrimental to Property 4.33, as seen in Figure 4.8. However, allowing disconnected common subgraphs renders the problem NP-hard. The more general LaWeCSE_u approach suggests an improved similarity value as seen in Figure 4.9 while maintaining polynomial running time. Experimentally evaluated results about this approach and different similarity coefficients are presented in Section 6.4.

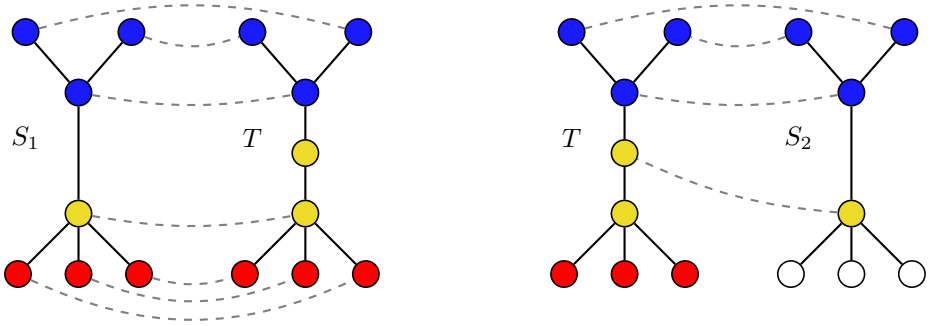
4.8 Metrics

Closely related to a similarity coefficient is a dissimilarity coefficient. Contrary to a similarity coefficient, a dissimilarity coefficient should be 0 for similar objects and large for completely



(a) MCSI between S_1 and T (b) MCSI between S_2 and T

Figure 4.8: Labeled MCSIs between two trees (dotted lines). S_1 is intuitively more similar to T than S_2 to T, but $\mathcal{W}(T, S_1) = \mathcal{W}(T, S_2) = 4$.



(a) LaWeCSE$_u$ between S_1 and T (b) LaWeCSE$_u$ between S_2 and T

Figure 4.9: Largest weight common subtree embeddings (labeled, unrooted) with penalty $p = 0$ between two trees (dotted lines). S_1 is intuitively more similar to T than S_2 to T. This is consistent with the number of mapped vertices: $7 = \mathcal{W}(T, S_1) > \mathcal{W}(T, S_2) = 4$.

different objects. We hold this as an informal property.

Property 4.34. *Given a reference object r and a set of objects C, a* perfect *dissimilarity function d satisfies the following property:*

$$\forall a, b \in C. \quad d(a, r) < d(b, r) \text{ if and only if } a \text{ is considered more similar to } r \text{ than } b \text{ to } r.$$

If a dissimilarity function fulfills the following mathematical properties, it is known as a *metric*.

Definition 4.35 (Metric). *Let C be a set. A function $d : C \times C \to \mathbb{R}$ is a metric on C if and only if the following properties hold for any $x, y, z \in C$:*

$$d(x, y) \geq 0 \qquad \text{non-negativity} \tag{4.14}$$

$$d(x, y) = 0 \Leftrightarrow x = y \qquad \text{identity} \tag{4.15}$$

$$d(x, y) = d(y, x) \qquad \text{symmetry} \tag{4.16}$$

$$d(x, z) \leq d(x, y) + d(y, z) \qquad \text{triangle inequality} \tag{4.17}$$

A metric is normalized *if* $d(x, y) \leq 1 \; \forall x, y \in C$.

In the domain of graphs, such a metric is defined as follows.

Definition 4.36 (Graph Distance Metric). *A* graph distance metric *is a metric on a set of graphs* $\mathcal{G}$*, where Property* (4.15) *from Definition 4.35 is replaced by*

$$d(G, H) = 0 \;\Leftrightarrow\; G \simeq H \qquad \forall\, G, H \in \mathcal{G} \tag{4.18}$$

Torsello et al. showed that a maximum weight common subtree isomorphism could be used to quantify the dissimilarity between two trees.

Proposition 4.37 ([123]). *Let* Σ *be a set of labels,* d *be any normalized metric on* Σ*, and the weight function* ω *be derived from* d *as* $\omega(l, l') \coloneqq 1 - d(l, l')$ *for all* $l, l' \in \Sigma$*. Let* $\mathcal{T}$ *be the set of vertex-labeled trees, where two trees are considered the same if they are isomorphic with respect to their labels.*

Then the following functions on $\mathcal{T}$ *are normalized graph distance metrics.*

$$d_1(T, T') = 1 - \frac{\mathcal{W}(T, T')}{|T| + |T'| - \mathcal{W}(T, T')} \qquad \forall\, T, T' \in \mathcal{T} \tag{4.19}$$

$$d_2(T, T') = 1 - \frac{\mathcal{W}(T, T')}{\max\{|T|, |T'|\}} \qquad \forall\, T, T' \in \mathcal{T} \tag{4.20}$$

Note, for $i \in \{1, 2\}$, $d_i = 1 - S_i$, where S_i is the ith similarity coefficient from Table 4.3. Metrics not only align with an intuitive dissimilarity (cf. Property 4.34) but are, e.g., important components of several machine learning methods, such as instance-based learning or clustering [111] and are required for undistorted embedding in a vector space [123]. From Theorem 4.13 and Proposition 4.37, we can compute a graph distance metric on vertex labeled trees in polynomial time. In contrast, computing a graph distance metric on arbitrary (unrestricted) graphs is NP-hard.

Next, we compute d_1 and d_2 on the graphs in Figure 4.8. Let $d(x, y) = 0$ for vertices x, y of the same color and $d(x, y) = 1$ otherwise. Then d is a normalized metric, and we define $\omega \coloneqq 1 - d$. Note, the definition of ω does not represent a labeled MCSI since vertices of different colors contribute zero to the weight instead of $-\infty$. However, we have $\mathcal{W}(T, S_1) = \mathcal{W}(T, S_2) = 4$ as in the labeled case. The full results for the graph distance metrics d_1 and d_2 are as follows.

MWCSI, d_1

	S_1	S_2	T
S_1	0	3/5	7/11
S_2	3/5	0	7/11
T	7/11	7/11	0

MWCSI, d_2

	S_1	S_2	T
S_1	0	3/7	1/2
S_2	3/7	0	1/2
T	1/2	1/2	0

Both d_1 and d_2 compute the same dissimilarity between T and S_1, S_2. Next, we compute the dissimilarity using the LaWeCSE_u approach on the same graphs using the same metric d as before. The results for d_1 and d_2 are as follows.

LaWeCSE_u, d_1

	S_1	S_2	T
S_1	0	3/5	1/8
S_2	3/5	0	7/11
T	1/8	7/11	0

LaWeCSE_u, d_2

	S_1	S_2	T
S_1	0	3/7	1/8
S_2	3/7	0	1/2
T	1/8	1/2	0

Using the LaWeCSE$_u$ approach, both d_1 and d_2 compute a smaller dissimilarity between T and S_1 compared to S_2. This aligns with Property 4.34. However, we have yet to show if d_1 and d_2 are a metric when we allow skipping vertices. We leave this as an open question.

4.9 Summary and Future Work

In this chapter, we first introduced the maximum common subtree isomorphism problem in Section 4.1. This problem is a specialization of the well known maximum common subgraph problem with trees as input and output. In Section 4.2, we provided running time results for rooted trees as input. Here, we considered unlabeled and labeled trees, as well as a weight function between the vertices and edges. That section concluded with Theorem 4.9 and stated a running time of $\mathcal{O}(|T|\,|T'|\Delta)$ for trees T and T' and a real-valued weight function ω, where $\Delta := \min\{\Delta(T), \Delta(T')\}$. If ω is integral and bounded by N, we proved an upper time bound of $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(N\min\{|T|,|T'|\}))$. In Section 4.3, we considered unrooted trees. We were able to obtain the same running time results as in the rooted case. Key arguments in the proofs were the results from the unbalanced all-cavity maximum weight matching problem from Section 3.5.

Section 4.4 provided a lower bound for the maximum weight common subtree isomorphism problem for trees of bounded degree. We also reasoned about a lower bound for the unrestricted case. In Section 4.5, we introduced the unrooted largest weight common subtree embedding problem. This problem generalizes the MWCSI problem. Here, the edges of the common embedding may map to so-called topological paths in the input graphs. This approach is beneficial for substructure searching (using the similarity coefficients from Section 4.7) in cheminformatics, as shown in Chapter 6. Section 4.6 provided a polynomial delay algorithm to enumerate all maximum weight common subtree isomorphisms between two trees.

The enumeration result from Theorem 4.32 considers a real-valued weight function ω. In Section 4.3, we provided a faster running time for the MWCSI problem if the weights are integral and bounded by a constant N. Unfortunately, we cannot quickly transfer these improved results to our enumeration algorithm. The reason is, we require the equality subgraph, which we obtain from the Hungarian method only. The faster matching algorithms for integral weights use other techniques like minimum cost flows to solve the problem. Therefore, the following problem remains open.

Open Problem 4.1. *Is there a better time bound to enumerate all MWCSIs than* $\mathcal{O}(|T|\,|T'|(\Delta + \log d) + \alpha(\min\{|T|,|T'|\} + \Delta(T)\Delta(T')))$*, where* $\Delta = \min\{\Delta(T), \Delta(T')\}$*,* α *is the number of MWCSIs, and* $d = \max\{\Delta(T), \Delta(T')\}$ *if the weights are integral bounded by* N*?*

The definition of the largest weight common subtree embedding problem allows mapping edges of the common embedding to topological paths of arbitrary length. This may or may not be desirable. To deal with it, we allow a penalty for skipped vertices, i.e., the longer the topological path, so larger the penalty. However, one might be interested in non-linear penalties. These seem difficult to compute since they probably cannot be integrated into the dynamic programming approach.

Open Problem 4.2. *Is there a way to use non-linear penalties in the LaWeCSE$_u$ approach?*

Regarding the LaWeCSE$_u$ approach again, one might be interested in a given maximum length of the topological paths. This should be possible with computing and storing l^2 additional values per pair of vertices if l is the maximum path length. The idea is to store not only a single weight when skipping vertices (cf. Lemma 4.22; $\mathcal{L}(u, v, \diamond)$ therein), but also the number of vertices we yet skipped for *both* the input graphs. This results in computing l^2 values instead of only one for skipping. The details of this approach need to be further elaborated.

Open Problem 4.3. *Can we provide a maximum length for topological paths in the LaWeCSE$_u$ approach? How does this affect the running time?*

A third open problem of the LaWeCSE$_\mathrm{u}$ approach is the enumeration of all maximum solutions. The difficulty lies in outputting each solution exactly once. This problem origins in the arguments of Lemma 4.25, more specific, that we may split the there mentioned path P_2 at an arbitrary position to obtain a maximum solution. Since we do not know those paths in advance and the dynamic programming approach considers each vertex on the path P_2 as a split point, we might generate the same maximum solution from different table entries during the enumeration process. It remains open to identifying such identical solutions with no or only a minor impact on the running time.

Open Problem 4.4. *Can we enumerate all unrooted largest weight common subtree embeddings with polynomial delay?*

Instead of finding all maximum weight common subtree isomorphisms, one might be interested in enumerating all maximal solutions, preferably with polynomial delay. Here, the problem is that maximum solutions are not necessarily maximal. For example, if we use a constant zero weight function, then every subgraph is maximum. Most of these subgraphs are not maximal. Similar to the aforementioned open problem, identical maximal solutions can be generated from different starting points. While we can easily identify such duplicates, this results in not outputting a solution in the recursion tree's current node. Consequently, it is difficult to provide a polynomial delay algorithm since there might be long runs with only duplicate solutions. However, enumerating all maximal solutions is possible in polynomial total time.

Open Problem 4.5. *Can we enumerate all maximal common subgraph isomorphism with polynomial delay?*

We can obtain a graph distance metric from an MWCSI between vertex labeled trees under a weight function ω, which itself is a metric. However, it remains open if the same is true for the LaWeCSE$_\mathrm{u}$ approach, where both vertices and edges are labeled.

Open Problem 4.6. *Can we derive a graph distance metric from computing an unrooted largest weight common subtree embedding?*

Die schwierigste Brücke ist die zwischen Menschen.

Hermann Lahm
1948 –

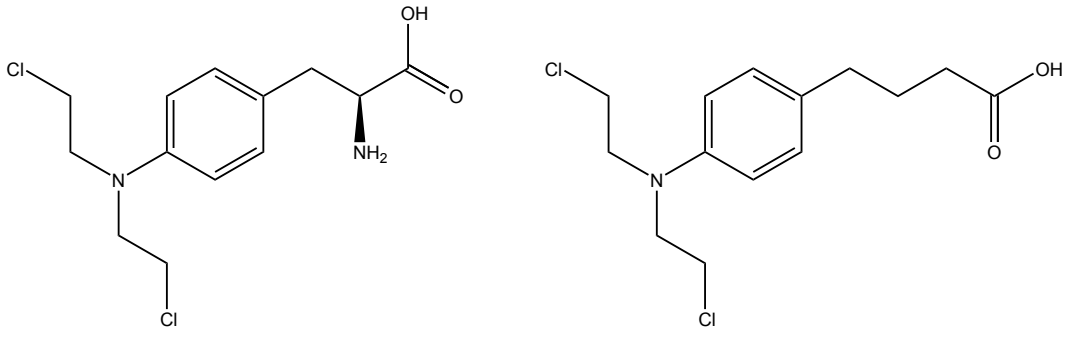
Chapter 5

Block-and-bridge preserving Maximum Common Subgraph

[1] In the previous chapter, we investigated the maximum common subtree isomorphism problem. For that problem, both the input and output graphs are restricted to trees, which allows a polynomial running time. When we represent molecules, these can naturally be modeled as graphs. An example of two different but similar drugs is depicted in Figure 5.1.

In particular, the vast majority of drugs are likely to have an outerplanar representation [58]. Unfortunately, the maximum common subgraph problem on outerplanar graphs remains NP-hard. In cheminformatics, the problem of extracting common structural features of objects represented as graphs has been extensively studied [106, 33, 111]. In this domain, it is often referred to as the maximum or *largest common substructure problem*. From Definition 4.3, we know there exist four optimization variants. In cheminformatics, the edge induced variant is used more frequently

[1] This chapter consists partly of our contribution in *Finding Largest Common Substructures of Molecules in Quadratic Time* [27].



(a) Melphalan (b) Chlorambucil

Figure 5.1: Two molecules represented as labeled molecular graphs. Vertices without labels are carbon (C) atoms. Both graphs are outerplanar.

since it (i) reflects the notion of chemical similarity more adequately [106], and (ii) can reduce the running time of product graph based algorithms [96]. Although such algorithms still have exponential running time in the worst case, they are commonly applied to molecular graphs in practice [106].

In cheminformatics, there is a further variation of the problem of high practical relevance. In this variant, the rings of molecules must not be broken, i.e., the block and bridge structure of the input graphs must be retained by the common subgraph. This is known as the so-called *block-and-bridge preserving* (BBP) constraint [111], which requires the common subgraph to retain the input graphs' local connectivity. We distinguish the edge and vertex induced variants. They are denoted by BBP-MCES and BBP-MCIS, respectively, and are formalized in Section 5.1. Both are polynomial-time computable on outerplanar graphs and also yield meaningful results for cheminformatics [33, 104]. Kriege, Kurpicz, and Mutzel [71] proposed an algorithm for the BBP-MCIS problem, which requires time $\mathcal{O}(n^6)$ in series-parallel and $\mathcal{O}(n^5)$ in outerplanar graphs. The BBP-MCES approach suggested by Schietgat et al. [111] has been shown to outperform state of the art algorithms for the general maximum common subgraph problem on molecular graphs in practice. This algorithm is stated to have a running time of $\mathcal{O}(n^{2.5})$ but leads to a running time of $\Omega(n^4)$ in the worst case, as we showed in [70].

We address the algorithmic and complexity challenges of the BBP constraint on outerplanar graphs and propose a novel BBP-MCIS algorithm that matches the running time of our MWCSI approach from the previous chapter. This leads to a quadratic time complexity for molecular graphs, which have bounded degree. We obtain this result by combining the ideas to speed up the MWCSI computation from Section 4.3 with a new algorithm to compute biconnected maximum common induced subgraphs in biconnected outerplanar graphs. For this subproblem, we develop a quadratic time algorithm, which exploits the fact that the outerplanar embedding of a biconnected outerplanar graph is unique. Moreover, the algorithm allows listing all maximal solutions in quadratic total time.

The experimental comparison on synthetic and real-world data in Chapter 6 shows that our approach is highly efficient in practice and outperforms comparable state of the art algorithms. The experiments show that our vertex induced variant in almost all cases yields the same results as the edge induced variant for molecular graphs under an adequate weight function. Our method outperforms in terms of efficiency the BBP-MCES approach [111] by several orders of magnitude.

This chapter is organized as follows. Section 5.1 formalizes the BBP-MCIS problem and summarizes its application to cheminformatics. Section 5.2 presents an algorithm to compute a biconnected maximum common induced subgraph on outerplanar graphs. This section's content extends our results published in [27] to allow both positive and negative weights. In Section 5.3, we present an algorithm to solve the BBP-MCIS problem on outerplanar graphs. Section 5.4 studies variants of the block-and-bridge preserving maximum common subgraph problem. This includes skipping vertices as in the LaWeCSE_u approach from Subsection 4.5.3, bioisosteres, and the computation of a BBP-MCIS on non-outerplanar graphs. We also present a polynomial delay enumeration algorithm for all maximum weight isomorphisms in Section 5.5 and conclude in Section 5.6.

5.1 Preliminaries

In the following, we formalize the block-and-bridge preserving maximum common induced subgraph isomorphism (BBP-MCISI) problem and its variants. The intermediate problem to the BBP-MCISI problem is the block-and-bridge preserving induced subgraph isomorphism problem.

Definition 5.1 (Block-and-Bridge Preserving Induced Subgraph Isomorphism). *Let G and H be graphs and $G \preceq_\phi H$, where ϕ is an induced subgraph isomorphism (cf. Definition 4.2). If additionally*

(i) each bridge in G maps to a bridge in H,

(ii) any two edges in different blocks in G map to different blocks in H,

then G is block-and-bridge preserving induced subgraph isomorphic *to H, we write $G \sqsubseteq_\phi H$, and denote ϕ as a* block-and-bridge preserving induced subgraph isomorphism*.*

In the previous chapter, we investigated different variants for the maximum common induced subgraph problem: unlabeled, labeled, and weighted. For the BBP-MCISI problem, we provide a definition for the weighted case only; the other variants are defined analogously to the previous chapter.

Definition 5.2 (Block-and-Bridge Preserving Maximum Common Induced Subgraph Isomorphism). *Let G and H be graphs under a weight function ω between each pair of vertices and each pair of edges between G and H. Let S be a connected graph such that $S \sqsubseteq_\phi G$ and $S \sqsubseteq_{\phi'} H$. Then $\varphi : \phi(V_S) \to \phi'(V_S)$, $\varphi(v) = \phi' \circ \phi^{-1}(v)$ is called* block-and-bridge preserving common induced subgraph isomorphism*.*

The weight $\mathcal{W}(\varphi)$ is the sum of the weights $\omega(v, \varphi(v))$ and $\omega(e, \varphi(e))$ of all vertices and edges mapped by φ. If $\mathcal{W}(\varphi)$ is maximum among all such isomorphisms, then φ is a block-and-bridge preserving maximum common induced subgraph isomorphism *(BBP-MCISI).*

Note that we require the common subgraph to be connected. Computing a BBP-MCISI on arbitrary graphs is difficult. However, for *outerplanar graphs*, the problem is solvable in polynomial time, as we show in Section 5.3.

A variant to the BBP-MCISI problem is the *block-and-bridge preserving maximum common edge subgraph isomorphism* (BBP-MCESI) problem [111]. In this variant, we do not require the associated subgraph isomorphisms to be induced.

5.2 Biconnected Maximum Common Induced Subgraph

This section presents an algorithm to determine the weight of a maximum common biconnected induced subgraph isomorphism of two biconnected outerplanar graphs.

Definition 5.3 (Maximum and Maximal Common Biconnected Induced Subgraph Isomorphism). *A common induced subgraph isomorphism ϕ is called* biconnected *if the associated common subgraph is biconnected. A biconnected isomorphism ϕ is* maximum (2-MCIS)*, if it is of maximum weight among all biconnected common induced subgraph isomorphisms; it is* maximal*, if it cannot be extended by additionally mapped vertices.*

In Subsection 5.2.1, we study the computation of a 2-MCIS between biconnected outerplanar graphs G and H on a weight function $\omega : (V_G \times V_H) \cup (E_G \times E_H) \to \mathbb{R}^{\geq 0} \cup \{-\infty\}$. We allow forbidden pairs (weight $-\infty$), which may not be mapped to each other; otherwise, the weights must be positive or 0. In Subsection 5.2.2, we study the problem on $\omega : (V_G \times V_H) \cup (E_G \times E_H) \to \mathbb{R} \cup \{-\infty\}$, i.e., we allow arbitrary weights.

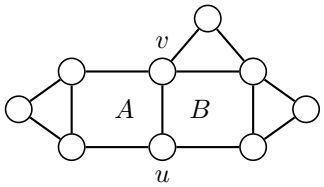


Figure 5.2: A biconnected outerplanar graph with an edge uv incident to the faces A and B.

5.2.1 2-MCIS under a Non-Negative Weight function [2]

We start with the computation of the maximal common biconnected subgraph isomorphisms. Since these may contain forbidden vertex and edge pairs, we then describe how to obtain a maximum solution from them.

Outerplanar graphs are well-studied and have several characteristic properties; see [120] for further information. In particular, our algorithm exploits the fact that biconnected outerplanar graphs have a unique outerplanar embedding in the plane (up to the mirror image). In these embeddings, every edge is incident to exactly two faces that are uniquely defined. We observe that the mapping is determined by starting parameters, i.e., an edge of both input graphs together with the mapping of their endpoints and incident faces.

We say an isomorphism ϕ maps a face if it maps all the vertices bordering the face. We distinguish four cases to describe the mapping of an edge $uv \in E(G)$ to an edge $u'v' \in E(H)$ by an isomorphism ϕ between biconnected induced subgraphs. Assume the edge uv is incident to the faces A and B in G (see Figure 5.2), and $u'v'$ is incident to A' and B' in H. At least one face incident to uv must be mapped by ϕ since the common subgraph must be biconnected. For simplicity of the case distinction, we also associate the two other faces, regardless of whether they are mapped or not. The isomorphism may map the endpoints of the edges in two different ways—just as the two incident faces. We can distinguish the following four cases:

1. $u \mapsto u',\ v \mapsto v',\ A \mapsto A',\ B \mapsto B'$,
2. $u \mapsto v',\ v \mapsto u',\ A \mapsto A',\ B \mapsto B'$,
3. $u \mapsto u',\ v \mapsto v',\ A \mapsto B',\ B \mapsto A'$,
4. $u \mapsto v',\ v \mapsto u',\ A \mapsto B',\ B \mapsto A'$.

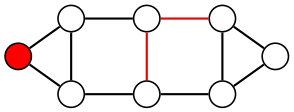
Given an isomorphism ϕ between biconnected common induced subgraphs that maps the two endpoints of an edge e, let the function $\text{type}(e, \phi) \in \{1, \ldots, 4\}$ determine the mapping type as above. The following result is the key to obtain our efficient algorithm.

Lemma 5.4 ([73]). *Let ϕ and ϕ' be maximal isomorphisms between biconnected common induced subgraphs of the biconnected outerplanar graphs G and H. Assume $e \in E(G)$ is mapped to the same edge $e' \in E(H)$ by ϕ and ϕ', then*

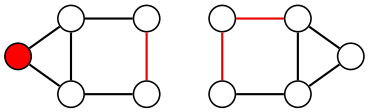
$$\text{type}(e, \phi) = \text{type}(e, \phi') \iff \phi' = \phi.$$

Proof. It is obvious that the direction $\Longleftarrow$ is correct. We prove the implication $\Longrightarrow$. Since the common subgraph is required to be biconnected, the isomorphisms ϕ and ϕ' both must map at

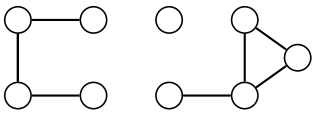
[2] This subsection consists of our findings in *Finding Largest Common Substructures of Molecules in Quadratic Time* [27], Section 3. The findings for the unlabeled case, including Lemma 5.4 and its proof, originate from Kriege [73].



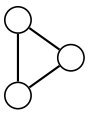
(a) Biconnected common induced subgraph



(b) Graph separated along inner forbidden edges



(c) Forbidden edges and vertices removed



(d) Non-block parts removed

Figure 5.3: Handling forbidden vertices and edges (colored red): (a) A computed maximal solution. (b) The solution is separated along inner forbidden edges. (c) Then all forbidden edges and vertices are removed. (d) Finally, the non-block parts are stripped. We store each component's weight into the corresponding table entries and $-\infty$ for each removed edge.

least one face of G incident to the edge e to a face of H incident to e'. The two faces as well as the mapping of endpoints of the two edges are uniquely determined by the type of the mapping. We consider the mapping of the vertices on the cyclic border of these faces. Since the mapping of the endpoints of e are fixed, the mapping of all vertices on the border of the face is unambiguously determined. Since the common subgraph is required to be biconnected, every extension of the mapping must include all the vertices of a neighboring face. For this face, again, the mapping of the endpoints of the shared edge implicates the mapping of all vertices on the cyclic border and the extension is unambiguous. Therefore, the mapping can be successively extended to an unmapped face. Consequently $\phi(u) = \phi'(u)$ holds for all $u \in \mathrm{dom}(\phi) \cap \mathrm{dom}(\phi')$. Since ϕ and ϕ' are maximal it is impossible that one of them can be extended. Hence, $\mathrm{dom}(\phi) = \mathrm{dom}(\phi')$ and the result follows. □

The proof of Lemma 5.4 constructively shows how to obtain a maximal solution given two edges $uv \in E(G)$, $u'v' \in E(H)$, and a type parameter $t \in \{1, \ldots, 4\}$. We assume that this approach is realized by the procedure MaximalIso$(uv, u'v', t)$, which returns the unique maximal isomorphism that maps the two given edges according to the specified type. The algorithm can be implemented by utilizing a tree structure that encodes the neighboring relation between inner faces, e.g., SP-trees as presented by Kriege et al. [71, 72] or weak dual graphs similar to the approach by Syslo [120]. The running time to compute a maximal solution ϕ then is $\mathcal{O}(|\phi|)$. Note that for some edge pairs, not all four types of mappings are possible. The type $t \in \{1, \ldots, 4\}$ is *valid* for a pair of edges if at least one incident face can be mapped according to type t, i.e., the edges are incident to faces that are bordered by the same number of vertices.

A maximal solution ϕ can map vertex and edge pairs that are forbidden according to the weight function. To obtain the maximum weight, we split ϕ into *split isomorphisms* $\phi_1, \ldots, \phi_k$ such that each (i) has non-negative weight and (ii) again is an isomorphism between biconnected induced common subgraphs. The split isomorphisms can be obtained in time $\mathcal{O}(|\phi|)$ as follows. We consider the graph $G' = G[\mathrm{dom}(\phi)]$. For every forbidden edge uv that is incident to two inner faces in G', we split the graph into $G'_i[V(cc_i) \cup \{u, v\}]$, $i \in \{1, 2\}$, where cc_i is a connected component of $G' \setminus \{u, v\}$. In these graphs, we delete the forbidden vertices and edges and determine the blocks $B_1, \ldots, B_k$. Then ϕ restricted to the vertices $V(B_i)$ of a block B_i yields the split isomorphism ϕ_i for $i \in \{1, \ldots, k\}$. This approach is realized by the function SplitIso(ϕ) used in the following. Every edge $e \in E(G)$ is mapped by at most one of the resulting isomorphisms, referred to by ϕ_e. Every 2-MCIS is a split isomorphism obtained from some maximal solution. The approach to

Algorithm 4: 2-MCIS in outerplanar graphs

Input : Biconnected outerplanar graphs G and H.
Output : Weight of a maximum common biconnected subgraph isomorphism.
Data : Table $D(e, f, t)$, $e \in E(G)$, $f \in E(H)$, $t \in \{1, \dots, 4\}$, storing the weight of a 2-MCIS ϕ, mapping e to f with $\text{type}(e, \phi) = t$.

1 **forall** *$uv \in E(G)$, $u'v' \in E(H)$, and $t \in \{1, \dots, 4\}$* **do**
2 **if** *type t valid for uv and $u'v'$* **and** *$D(uv, u'v', t)$ undefined* **then**
3 $\phi \leftarrow \textsc{MaximalIso}(uv, u'v', t)$
4 $(\phi_1, \dots, \phi_k) \leftarrow \textsc{SplitIso}(\phi)$
5 **forall** *edges $e \in E(G)$ mapped to $f \in E(H)$ by ϕ* **do**
6 $D(e, f, \text{type}(e, \phi)) \leftarrow \begin{cases} \mathcal{W}(\phi_e) & \text{if } e \text{ is mapped by the split isomorphism } \phi_e \\ -\infty & \text{otherwise.} \end{cases}$
7 **return** maximum entry in D

split an isomorphism is exemplified in Figure 5.3.

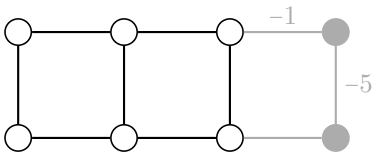
Algorithm 4 uses a table $D(e, f, t)$, $e \in E(G)$, $f \in E(H)$, $t \in \{1, \dots, 4\}$, storing the weight of a 2-MCIS under the constraint that e is mapped to f according to type t. The size of the table is $4|E(G)|\,|E(H)| \in \mathcal{O}(|G|\,|H|)$. The algorithm starts with all pairs of edges and all valid types of mappings between them. For each, the maximal isomorphism between biconnected common induced subgraphs is computed by extending this initial mapping. By splitting the maximal solution, multiple valid isomorphisms with non-negative weight are obtained. These weights are then stored in D for all pairs of edges contained in ϕ considering the type of the mapping. This includes the $-\infty$ weights occurring if there are forbidden vertices or edges. Keeping these values prevents generating the same isomorphism multiple times. The main procedure loops over all pairs of edges and the four possible mappings for each pair. Note that a mapping ϕ and its split isomorphisms are computed in time $\mathcal{O}(|\phi|)$. An improved analysis gives the following result.

Proposition 5.5. *Algorithm 4 computes the weight of a 2-MCIS between biconnected outerplanar graphs G and H under a weight function mapping to $\mathbb{R}^{\geq 0} \cup \{-\infty\}$ in time $\mathcal{O}(|G|\,|H|)$.*

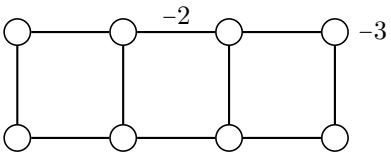
Proof. We assign the costs for a call of MaximalIso followed by SplitIso to cells of table D. A mapping ϕ containing k edges is computed in time $\mathcal{O}(k)$, and as a result, exactly k cells of table D are filled with a value. The value of a cell is computed at most once: Line 2 assures that an edge mapping of a specific type is not used as initial mapping when the corresponding cell is already filled. Every initial mapping that is extended must lead to an isomorphism containing only edge mappings associated with undefined cells, according to Lemma 5.4. Therefore, the algorithm's total costs can be allocated to cells of D, such that each cell pays a constant amount. This proves that the total running time is bounded by the size of the table, which is $\mathcal{O}(|G|\,|H|)$. □

We can easily modify the algorithm to enumerate all split isomorphisms of maximum weight[3] without affecting the total running time. First, we run Algorithm 4 once to obtain the maximum weight $\mathcal{W}_{\max}$ of a common biconnected subgraph isomorphism. Then we run a modified version of Algorithm 4 that outputs every split isomorphism ϕ_i of weight $\mathcal{W}(\phi_i) = \mathcal{W}_{\max}$ as soon as it is found, right after $\textsc{SplitIso}(\phi)$ is called in line 4.

[3] These are not necessarily identical to all the 2-MCISs. We cover the enumeration of all 2-MCISs in Subsection 5.5.1.



(a) 2-MCIS of weight 13. The maximal solution has weight 10.



(b) 2-MCIS of weight 11 including a negative weight edge and vertex.

Figure 5.4: Maximum common biconnected induced subgraphs (colored black) from two different maximal solutions (whole graphs). The numbers specify the vertex and edge weights; 1, if not present.

5.2.2 2-MCIS with Arbitrary Weights [4]

Obtaining a 2-MCIS in the previous subsection results from enumerating all maximal biconnected common subgraphs with further handling of forbidden edges and vertices. For non-negative weights, it is sufficient to consider only maximal solutions: Whatever vertices (and induced edges) are added to an isomorphism ϕ, they cannot decrease $\mathcal{W}(\phi)$. For negative weights, a non-maximal solution can yield a larger total weight, as seen in Figure 5.4a. However, a maximum solution can indeed include edges or vertices of negative weight, cf. Figure 5.4b.

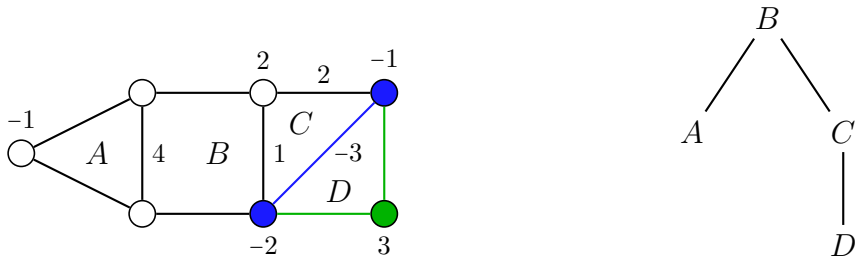
General approach. We can find a 2-MCIS if we also consider each non-maximal biconnected common subgraph isomorphisms. However, enumerating all such subgraphs is problematic, as there can be exponentially many. Fortunately, a closer look reveals similarity to the well known maximum sum subarray problem.

> **Problem 5.1 (Maximum Sum Subarray Problem).** *Given an array of both positive and negative numbers, return a contiguous subarray, such that the sum of those numbers is as large as possible.*

It is known that the maximum sum subarray problem is solvable in linear time [5]. This is achieved by scanning the numbers from left to right. Let a be the array and $a(i)$, $i \geq 1$, be the number at position i. Initially, $g = l = i = 0$, where g is the best global sum up to position i and l the best local sum up to position i. The local sum l must include $a(i)$ or be 0. We iteratively compute $l \coloneqq \max\{0, l + a(i)\}$ and then $g \coloneqq \max\{g, l\}$ until we scanned the whole array. Then, the maximum sum is g. It is easy to store the start and end indices of the maximum contiguous subarray during the computation.

From the proof of Lemma 5.4, we know that the biconnected solutions are expanded by adding all the vertices of neighboring faces of the shared edges. In other words, each biconnected solution consists of some neighboring faces, which include all vertices and edges on that faces' borders. We obtain the so-called *dual graph* if we interpret faces as vertices and the neighborhood of those faces as edges. The dual graph of an embedded outerplanar graph minus the outer face is a tree. The general approach to find a maximum solution is to assign each vertex in the dual graph a number determined by the weight that this face contributes to the isomorphism's weight. Then, similar to solving the maximum sum subarray problem, we scan the dual graph (tree) from the leaves to an

[4] This subsection is based on the findings during the Bachelor's thesis *Berechnung gewichtsmaximaler 2-zusammenhängender gemeinsamer Subgraphen von außenplanaren Graphen* from Dennis Misera, supervised by Prof. Dr. Petra Mutzel and the author of this thesis. The basic approach to solve this problem was studied independently by the author of this thesis in preparation for that Bachelor's thesis.



(a) A maximal biconnected common subgraph G'. Numbers are non-zero edge or vertex weights.

(b) The dual graph (without the outer face) of G' with face B arbitrarily selected as root.

Figure 5.5: A maximal biconnected common subgraph (a) and its dual graph (b). The green vertices and edges are the contribution of face D if also its parents face C is considered for a maximum solution. If C is not considered for a maximum solution, both the blue and green vertices and edges are added to determine the weight of face D. A maximum biconnected common subgraph within G' consists of the vertices and edges of faces B, C, and D; its weight is 6.

arbitrarily selected root vertex. This allows to find a maximum common biconnected subgraph isomorphism within a previously computed maximal isomorphism. The time to do this is linear in the size of the maximal isomorphism. Therefore, the time bound from Proposition 5.5 holds. An example of a biconnected outerplanar graph and its dual graph is depicted in Figure 5.5.

The algorithm. We obtain the common subgraph's dual graph directly from its SP-tree, which we can compute in linear time [51]. Simplified, the SP-tree of a biconnected outerplanar graph consists of nodes representing series (S) and parallel (P) compositions. From the P compositions, we obtain the edges adjacent to two faces. From the S compositions, we obtain all the other edges bordering a face. As seen in Figure 5.5a, edges and also vertices of a common subgraph can be connected to more than one face. To find a maximum solution, we first compute for each vertex and each edge the unique face in the dual tree with which it is incident and that is closest to an arbitrarily chosen root. Let these values be $F_{\text{ctr}}(v)$ and $F_{\text{ctr}}(e)$, respectively. In Figure 5.5, for the green vertices and edges, F_{ctr} is D, for the blue edge and the blue -1 vertex, it is C, and for the blue -2 vertex, it is B. The computation of a maximum solution occurs from the leaves of the dual tree to the selected root. For each face F, we compute two values. Let $w(v)$ and $w(e)$, respectively, be the weight of vertex v and edge e in the common subtree, respectively; $C(F)$ is the set of children of face F in the dual tree.

1. $F_{\text{add}}(F) = \sum_{\{v|F_{\text{ctr}}(v)=F\}} w(v) + \sum_{\{e|F_{\text{ctr}}(e)=F\}} w(e) + \sum_{F'\in C(F)} \max\{0, F_{\text{add}}(F')\}$
2. $F_{\text{maximum}}(F) = \sum_{\{v|v \text{ incident to } F\}} w(v) + \sum_{\{e|e \text{ incident to } F\}} w(e) + \sum_{F'\in C(F)} \max\{0, F_{\text{add}}(F')\}$

The value $F_{\text{add}}(F)$ is the increased weight if we add the vertices and edges of face F (and possibly of its descendants) to the parent face of F. For example, in Figure 5.5, we have $F_{\text{add}}(C) = -1 + 2 - 3 + \max\{0, F_{\text{add}}(D)\} = -2 + 3 = 1$. $F_{\text{maximum}}(F)$ is the total weight we can achieve, if no face above F in the dual tree is considered for a maximum solution, e.g., $F_{\text{maximum}}(C) = 2$ from all the edges and vertices incident to faces C and D. The weight of a maximum biconnected subgraph within the computed maximal solution is then equal to $\max\{F_{\text{maximum}}(F) \mid F \text{ is a face in the dual graph}\}$. In the example, $F_{\text{maximum}}(B) = 6$ is the

maximum. We obtain a maximum biconnected subgraph starting from such a face. We then recursively add the vertices and edges of descending faces F in the dual tree with $F_{\text{add}}(F) > 0$. We could also consider faces F with $F_{\text{add}}(F) = 0$. Vertices and edges from such faces would contribute 0 to the weight in total. It is worth noting that with this approach, it is not necessary to compute split isomorphisms before computing the maximum isomorphisms within the maximal biconnected solutions from Algorithm 4.

Running time. The running time to compute outerplanar embeddings for the input graphs G and H is $\mathcal{O}(|G| + |H|)$. The time to compute all the values F_{add} and F_{maximum} from a maximal biconnected common subgraph G' is $\mathcal{O}(|G'|)$. From Proposition 5.5, the time for all such graphs is $\mathcal{O}(|G|\,|H|)$. The time to compute a 2-MCIS from F_{maximum} is $\mathcal{O}(\min\{|G|, |H|\})$.

Correctness. The sum $F_{\text{add}}(F)$ considers vertices and edges incident to F, but not above F in the dual tree. This sum further considers vertices and edges below F in the dual tree if their weight contribution is positive (or zero). The sum $F_{\text{maximum}}(F)$ considers all vertices and edges incident to face F. The subgraph associated with this sum is biconnected. From the dynamic programming approach similar to the maximum sum subarray problem, all possible biconnected subgraphs are considered.

We obtain the following result.

Theorem 5.6. *We can compute a 2-MCIS between biconnected outerplanar graphs G and H under a weight function mapping to $\mathbb{R} \cup \{-\infty\}$ in time $\mathcal{O}(|G|\,|H|)$.*

5.3 Block-and-Bridge Preserving Maximum Common Induced Subgraph Isomorphism

The previous section has presented an algorithm to compute a 2-MCIS between two biconnected outerplanar graphs. This section generalizes it to compute a BBP-MCISI (cf. Definition 5.2) between two, not necessarily biconnected, outerplanar graphs G and H. In the following, we assume the isomorphisms to be BBP without explicitly mentioning it. We require the input graphs to be connected. Otherwise, we compute a BBP-MCISI for each pair of connected components between G and H and select an arbitrary isomorphism of maximum weight.

This section is organized as follows. In Subsection 5.3.1, using the BC-tree data structure, we partition the set $\mathcal{S}$ of all BBP common subgraph isomorphisms between G and H into subsets with respect to certain conditions. Next, in Subsection 5.3.2, we compute an isomorphism of maximum weight in each of the subsets using a dynamic programming approach similar to Section 4.3, where we solved the maximum common subtree problem. Among the computed isomorphisms, we output one with maximum weight, thus a BBP-MCISI. Initially, we restrict the weights to $\mathbb{R}^{\geq 0} \cup \{-\infty\}$. In Subsection 5.3.3, we describe the necessary steps to also allow negative weights.

5.3.1 Partitioning of the Problem [5]

Given a BBP-MCISI, we can observe that bridges of G are mapped to bridges of H and that edges in one block of G can only be mapped to edges contained in exactly one block of H, such that the mapped edges form a biconnected common subgraph. In other words, the set of all

[5] This subsection and the following largely consist of our findings in *Finding Largest Common Substructures of Molecules in Quadratic Time* [27], Section 4.

BBP isomorphisms is closely related to the BC-tree data structure of G and H. We use this observation to define a partitioning of all BBP isomorphisms $\mathcal{S}$ between G and H into sets $\mathcal{S}_x$, $\mathcal{S} = \bigcup_x \mathcal{S}_x$, depending on the blocks and bridges of G. We further partition each set $\mathcal{S}_x$ into subsets $\mathcal{P}_{xy}$, $\mathcal{S}_x = \bigcup_y \mathcal{P}_{xy}$, depending on the blocks and bridges of H. Within each subset $\mathcal{P}_{xy}$, we compute a maximum solution, then obtain one for each set $\mathcal{S}_x$, and finally get a BBP-MCISI between G and H. In the following, we specify the sets $\mathcal{S}_x$ and subsets $\mathcal{P}_{xy}$.

Partitioning of all BBP isomorphisms $\mathcal{S}$ into $\mathcal{S} = \bigcup_x \mathcal{S}_x$. Let $b \in \mathrm{B}^G$ be an arbitrary block or bridge in the BC-tree of G. We define $\mathcal{S}_1$ to contain all isomorphisms ϕ that map at least one edge in b, i.e., $|\operatorname{dom}(\phi) \cap V(b)| \geq 2$. $\mathcal{S}_2$ is defined to contain all isomorphisms mapping exactly one vertex in b. We can observe that S_1 and S_2 are disjoint, and all other isomorphisms between G and H do not contain any vertices of b. Let $N = \{b_1, \ldots, b_k\} \subseteq \mathrm{B}^G$ be the blocks and bridges that share a cutvertex with b, i.e., for all $b_i \in N$, $i \in \{1, \ldots, k\}$, exists a node $c \in \mathrm{C}^G$, such that bcb_i is a path in BC^G. Any isomorphism ϕ that maps no vertex of b maps vertices of at most one node b_i, because $G[\operatorname{dom}(\phi)]$ is connected by definition. For every b_i, we recursively define sets $\mathcal{S}_x$ of isomorphisms as described above that map only vertices of $\mathrm{CC}(V_G \setminus V_b, V_{b_i})$.

As an example, consider Figure 2.1b and let $b \coloneqq b_2$. $\mathcal{S}_1$ consists of isomorphisms that map at least one edge of b_2 to an edge in H. The isomorphisms in $\mathcal{S}_2$ map precisely one vertex of $V(b)$ to H. The recursion continues on $N = \{b_1, b_3, b_4\}$. We have two additional sets for each $i \in \{1, 3, 4\}$. One set consists of isomorphisms that map at least one edge of $V(b_i)$, but no vertex of $V(b_2)$, operating on $\mathrm{CC}(V_G \setminus V_{b_2}, V_{b_i})$. The other consists of isomorphisms on the same connected component mapping exactly one vertex of $V(b_i)$. The recursion for b_4 continues with $N = \{b_5\}$ and two additional sets. Some of the sets $\mathcal{S}_x$ are empty.

Partitioning of $\mathcal{S}_x$ into $\mathcal{S}_x = \bigcup_y \mathcal{P}_{xy}$. Before computing an isomorphism of maximum weight in a set $\mathcal{S}_x$, we partition $\mathcal{S}_x$ into subsets $\mathcal{P}_{xy}$, $y \in \{1, 2, \ldots\}$. The focus for that separation is on the graph H. We distinguish two cases. If $\mathcal{S}_x$ is a set, where at least one edge of a block (bridge) $b \in B^G$ is mapped, then $\mathcal{S}_x$ is partitioned into one subset for each block (bridge) of H. In other words, for each B-node $\bar{b} \in Bl^H$ ($\bar{b} \in \mathrm{Br}^H$) exists a subset, where the vertices of the B-node b are mapped only to $V(\bar{b})$. In terms of BBP, this is block (bridge) preserving between b and $\bar{b}$, as intended. If in $\mathcal{S}_x$ exactly one vertex of b is mapped, the subsets are defined as follows. For each $(v, \bar{v}) \in V(b) \times V(H)$, where $\omega(v\bar{v}) \neq -\infty$ and v is in the connected component we operate on, we define a subset with the restriction $\phi(v) = \bar{v}$.

As an example, assume Figure 2.1b to be the BC-tree of H. First, let $\mathcal{S}_x$ be a set, where at least one edge of a block is mapped. Then we have one subset mapping at least one edge from b to b_1. There are three additional sets mapping to b_2, b_3, and b_5. Second, let $\mathcal{S}_x$ be a set where exactly one vertex is mapped, and let b contain three vertices. Then we have up to $3 \cdot 13 = 39$ subsets, since $|H| = 13$, and we have one subset for each non-forbidden combination of one vertex in b and one vertex in H.

5.3.2 BBP-MCISI under a Non-Negative Weight Function

In the following, we present the approach to solving the BBP-MCISI problem for any subset $\mathcal{P}_{xy} \subseteq \mathcal{S}_x$. Then we provide an upper time bound to compute a maximum solution for all subsets $\mathcal{P}_{xy}$ of all sets S_x.

Computing a maximum isomorphism in a subset $\mathcal{P}_{xy}$. The basic idea is to recursively extend mappings between some vertices of two single blocks (two single bridges) b and $\bar{b}$ along all pairs of mapped cutvertices into other B-nodes determined by MWMs while preserving bridges and blocks.

The mapping of vertices into neighboring B-nodes is determined by a dynamic programming approach similar to that of trees from Chapter 4. Between the computed isomorphisms, we select one of maximum weight. During the recursion, some restrictions to the possible mappings of the vertices can occur. For example, if we map a cutvertex c of block b to a cutvertex $\bar{c}$ of block $\bar{b}$, then the recursion on the other blocks adjacent to c and $\bar{c}$ must also map $c \mapsto \bar{c}$. We call this *fixed mapping*. In the following, we present the details for the two cases for $\mathcal{P}_{xy}$: at least one edge of $\mathcal{P}_{xy}$ has to be mapped, and precisely one vertex of $\mathcal{P}_{xy}$ has to be mapped.

First, let $\mathcal{P}_{xy}$ be a subset, where at least one edge of a B-node $b \in \mathrm{B}^G$ has to be mapped to an edge of a B-node $\bar{b} \in \mathrm{B}^H$. If b and $\bar{b}$ are bridges, the two possible mappings $V(b) \to V(\bar{b})$ are considered. If both are blocks, all maximal common biconnected subgraph isomorphisms between them are considered (cf. Algorithm 4). Due to the recursive calls, up to one vertex x can be contained in b, but not in the connected component we operate on, cf. Subsection 5.3.1. Such a vertex x is treated as contributing weight $-\infty$ to the isomorphism whatever vertex in $V(\bar{b})$ it is mapped to. If such a vertex x exists and b is a bridge, $\mathcal{P}_{xy}$ is empty. For blocks, this implicates handling of forbidden vertices, cf. Subsection 5.2.1. From previous recursive calls, we may have to map a certain pair of cutvertices to each other. We call a considered isomorphism valid if it respects that mapping and does not contain the possible excluded vertex x. We extend all the valid isomorphisms ϕ of this subset $\mathcal{P}_{xy}$ along all pairs $\phi(c) = \bar{c}, c \neq v$ of mapped cutvertices as follows. Let $B_c \coloneqq \{b_1, \dots b_k\}$ be the B-nodes of B^G, where bcb_i is a path, and $\bar{B}_c \coloneqq \{\bar{b}_1, \dots \bar{b}_l\}$ be the B-nodes of B^H, where $\bar{b}\bar{c}\bar{b}_j$ is a path, $i \in \{1, \dots, k\}, j \in \{1, \dots, l\}$. For each pair $(b_i, \bar{b}_j) \in B_c \times \bar{B}_c$, we recursively calculate a BBP-MCISI φ_{ij} under the following restrictions: (i) The cutvertices must be mapped: $c \mapsto \bar{c}$. (ii) b_i and $\bar{b}_j$ are both bridges or both blocks. (iii) At least one other vertex in the block (bridge) b_i must be mapped, but only to $V(\bar{b}_j)$. Restriction (iii) assures that at least one vertex is added to the isomorphism. We recursively compute φ_{ij} via dynamic programming. In other words, we obtain φ_{ij} from another subset $\mathcal{P}_{x'y'}$, which is a subproblem to $\mathcal{P}_{xy}$. The recursion naturally stops at blocks or bridges, where no further expansion into neighboring blocks or bridges is possible, e.g., because there are none or they are incompatible (the BBP constraint must hold).

After computing φ_{ij} for each pair $(b_i, \bar{b}_j)$, we construct a weighted bipartite graph with vertices $B_c \uplus \bar{B}_c$ for each pair of mapped cutvertices. The weight of each edge $b_i\bar{b}_j$ is determined by the weight of a BBP-MCISI under the above restrictions, subtracted by $\omega(c, \bar{c})$ for the appropriate cutvertices c and $\bar{c}$. If there is no such restricted BBP-MCISI, there is no edge. Computing an MWM on each of the bipartite graphs determines the extension of ϕ. For each matching edge, the corresponding computed isomorphisms are merged with ϕ. After extending all valid isomorphisms, we select one of maximum weight. This concludes the case of mapping at least one edge of $\mathcal{P}_{xy}$.

We now discuss the second case of $\mathcal{P}_{xy}$, where exactly one vertex v of $V(b)$ is mapped. Let $\phi(v) = \bar{v}$. If v is no cutvertex, the only possible expansion is within $V(b)$, which is not allowed in this subset. Therefore, this subset contains exactly one isomorphism, $v \mapsto \bar{v}$. Next, assume v is a cutvertex. If $\bar{v}$ is a cutvertex, we may extend ϕ via dynamic programming analog to the first case of $\mathcal{P}_{xy}$. More precisely, we have exactly one pair of cutvertices v, $\bar{v}$ mapped to each other from where we expand ϕ into the neighboring blocks and bridges B_c and $\bar{B}_c$. In this case, $B_c \coloneqq N_{\mathrm{BC}^G}(c)$ and $\bar{B}_c \coloneqq N_{\mathrm{BC}^H}(\bar{c})$, i.e., we consider *all* blocks and bridges adjacent to c and $\bar{c}$, respectively. The reason is that we have not mapped any other vertices yet. If $\bar{v}$ is no cutvertex, then $\bar{v}$ is contained in precisely one $\bar{b} \in B^H$. We are interested in BBP isomorphisms only. This means, all vertices mapped to $V(\bar{b})$ must be in the same block or bridge $b' \in \mathrm{B}^G$. Therefore, for each $b' \in \mathrm{B}^G$, where bvb' is a path and b' and $\bar{b}$ are of the same type (bridge/block), we compute an isomorphism with fixed mapping $v \mapsto \bar{v}$ and at least one edge of b' mapped to $\bar{b}$. This, again, is possible via dynamic programming. Among the computed isomorphisms, we select one of

maximum weight.

Running time. We obtain the following running time result.

Proposition 5.7. *We can compute a BBP-MCISI between outerplanar graphs G and H under a weight function mapping to $\mathbb{R}^{\geq 0} \cup \{-\infty\}$ in time $\mathcal{O}(|G|\,|H|\,\Delta(G,H))$, where $\Delta(G,H) = 1$ if G or H is biconnected; otherwise $\Delta(G,H) = \min\{\Delta_C(G), \Delta_C(H)\}$.*

Proof. The time to compute a BBP-MCISI essentially depends on the time to compute the BC-trees, the biconnected isomorphisms between the blocks of G and H, and the time to compute all the MWMs. The time to compute the BC-tree of a graph is linear in its number of edges and vertices. Considering all pairs of blocks and Proposition 5.5, we can bound the time for computing all the biconnected isomorphisms by $\mathcal{O}(\sum_b \sum_{\bar{b}} |V_b|\,|V_{\bar{b}}|) \subseteq \mathcal{O}(|G||H|)$. We only need to compute MWMs for the pairs of cutvertices of the two graphs. Analog to Theorem 4.13, this is possible in time $\mathcal{O}(|G|\,|H|\,\min\{\Delta_C(G), \Delta_C(H)\})$. □

Pseudocode. The pseudocode to compute a BBP-MCISI is given in Algorithm 5 and its related procedures. The computation starts in Algorithm 5 with an input of two graphs and an associated weight function. From there, procedure SetSx is called on an arbitrary chosen B-node. Procedure SetSx performs further calls to procedures BBP-Edge (on a subset $\mathcal{P}_{xy}$ where at least one edge is mapped), BBP-SingleVertex (on a subset $\mathcal{P}_{xy}$ where exactly one vertex is mapped), and SetSx (recursion for no mapped vertices in b).

5.3.3 BBP-MCISI with Arbitrary Weights [6]

We studied negative weights within a biconnected component in Subsection 5.2.2. We handled them by computing maximum biconnected common induced subgraphs within a maximal solution G' using a dynamic programming approach derived from the maximum sum subarray problem. For the dual graph of G', we selected an arbitrary root face to solve the problem. However, when computing a maximum solution in a subset $\mathcal{P}_{xy}$, we may have given a fixed mapping $v \mapsto \bar{v}$. To ensure v is included in the solution, we consider a value $F_{\text{maximum}}(F)$ only if v is incident to the face F. To allow the isomorphism to extend as much as possible beyond all faces incident to v, we need to select a face incident to v as root face in the dual graph.

As an example, let the leftmost vertex in Figure 5.5a be the vertex v mapped to $\bar{v}$. Then we must select A as root in the dual graph, and only in doing so, we obtain a maximum isomorphism (which is the whole graph in this case). Using any other face as root face, e.g., face B, would produce a maximum solution without vertex v since $F_{\text{add}}(A) = -1$.

Negative weight vertices within bridges do not need special care. They are processed through the MWMs except for a possible bridge as root face in the dual graph. In total, we achieve the same running time with an arbitrary weight function as with non-negative weights only.

Theorem 5.8. *We can compute a BBP-MCISI of outerplanar graphs G and H under a weight function mapping to $\mathbb{R} \cup \{-\infty\}$ in time $\mathcal{O}(|G|\,|H|\,\Delta(G,H))$, where $\Delta(G,H) = 1$ if G or H is biconnected; otherwise, $\Delta(G,H) = \min\{\Delta_C(G), \Delta_C(H)\}$.*

[6] This subsection is based on the findings during the Bachelor's thesis *Berechnung gewichtsmaximaler 2-zusammenhängender gemeinsamer Subgraphen von außenplanaren Graphen* from Dennis Misera, supervised by Prof. Dr. Petra Mutzel and the author of this thesis.

Algorithm 5: BBP-MCISI of outerplanar graphs

Input : Connected outerplanar graphs G and H,
weight function $\omega : (V_G \times V_H) \cup (E_G \times E_H) \to \mathbb{R}^{\geq 0} \cup \{-\infty\}$.
Output : A BBP-MCISI of G and H.
Data : BC-trees BC^G and BC^H with node sets B^G, C^G, B^H, C^H.

Select an arbitrary B-node (block or bridge) $b \in B^G$.
$\phi \leftarrow \textsc{SetSx}(b, \emptyset)$ ▷ *Initial recursion call.*
[Output ϕ]

Procedure $\textsc{SetSx}(b, X)$
  Input : B-node $b \in B^G$, excluded vertices $X \subseteq V(G)$.
  Output : Isomorphism of maximum weight on $\mathrm{CC}(V_G \setminus X, V_b)$.
  $\varphi \leftarrow (\emptyset \to \emptyset)$ ▷ *Initialize as empty mapping*
  forall *B-nodes* $\bar{b} \in B^H$*, where* b *and* $\bar{b}$ *are both bridges or both blocks* **do**
    $\phi \leftarrow \text{BBP-}\textsc{Edge}(b, \bar{b}, X)$ ▷ $\mathcal{P}_{xy} \subseteq \mathcal{S}_x$*, at least one edge is mapped*
    if $\mathcal{W}(\phi) > \mathcal{W}(\varphi)$ **then** $\varphi \leftarrow \phi$
  forall $(v, \bar{v}) \in (V(b) \setminus X) \times V(H)$ **do**
    if $\omega(v\bar{v}) \neq -\infty$ **then**
      $\phi \leftarrow \text{BBP-}\textsc{SingleVertex}(b, X, v, \bar{v})$ ▷ $\mathcal{P}_{xy} \subseteq \mathcal{S}_x$*, single vertex*
      if $\mathcal{W}(\phi) > \mathcal{W}(\varphi)$ **then** $\varphi \leftarrow \phi$
  forall *paths* bcb' *in* BC^G*, where* $c \notin X$ **do**
    $\phi \leftarrow \textsc{SetSx}(b', V(b))$ ▷ *No vertex of* $V(b)$ *is mapped, recursion*
    if $\mathcal{W}(\phi) > \mathcal{W}(\varphi)$ **then** $\varphi \leftarrow \phi$
  return φ

Procedure $\text{BBP-}\textsc{Edge}(b, \bar{b}, X, v, \bar{v})$
  Input : B-nodes $b \in B^G, \bar{b} \in B^H, X \subseteq V(G)$, mapping $v \mapsto \bar{v}$ (optional).
  Output : Maximum isomorphism φ, where *at least one edge* of b is mapped to $\bar{b}$;
  restricted to $\mathrm{CC}(V_G \setminus X, V_b)$ and $\varphi(v) = \bar{v}$ (if given).
  if *exactly one of* $b, \bar{b}$ *is a block* **then**
    return $\emptyset \to \emptyset$ ▷ *not block-and-bridge preserving*
  forall *valid isomorphisms* $\varphi : V(b) \to V(\bar{b})$ **do**
    forall $(c, \bar{c}) \neq (v, \bar{v})$ *of cutvertices mapped by* φ **do**
      forall $(b_i, \bar{b}_j) \in \mathrm{B}^G \times \mathrm{B}^H$*, where* bcb_i *and* $\bar{b}\bar{c}\bar{b}_j$ *are paths* **do**
        $w(b_i, \bar{b}_j) \leftarrow \mathcal{W}(\text{BBP-}\textsc{Edge}(b_i, \bar{b}_j, X, c, \bar{c})) - \omega(c, \bar{c})$
      Compute MWM $\mathcal{M}$ on the bipartite graph with edge weights $w(b_i, \bar{b}_j)$
      forall *edges* $b_i\bar{b}_j \in \mathcal{M}$ **do**
        Extend φ by $\text{BBP-}\textsc{Edge}(b_i, \bar{b}_j, X, c, \bar{c})$
    if $\mathcal{W}(\phi) > \mathcal{W}(\varphi)$ **then** $\varphi \leftarrow \phi$
  return φ

Procedure BBP-SingleVertex($b, X, v, \bar{v}$)
  Input : B-node $b \in B^G$, excluded vertices $X \subseteq V(G)$, mapping $v \mapsto \bar{v}$.
  Output : Maximum isomorphism φ, where *a single vertex* of $V(b)$ is mapped to $V(\bar{b})$; restricted to $\mathrm{CC}(V_G \setminus X, V_b)$, $\varphi(v) = \bar{v}$.
  $\varphi \leftarrow (v \mapsto \bar{v})$
  if $v \notin C^G$ **then**
    return φ ▷ *No expansion possible*
  if $\bar{v} \in \mathrm{C}^H$ **then**
    forall *pairs* $(b_i, \bar{b}_j) \in \mathrm{B}^G \times \mathrm{B}^H$, *where* bvb_i *is a path and* $\bar{v} \in \bar{b}_j$ **do**
      $w(b_i, \bar{b}_j) \leftarrow \mathcal{W}(\text{BBP-Edge}(b_i, \bar{b}_j, X, v, \bar{v})) - \omega(v, \bar{v})$
    Compute MWM $\mathcal{M}$ on the bipartite graph with edge weights $w(b_i, \bar{b}_j)$
    forall *edges* $b_i\bar{b}_j \in M$ **do**
      Extend φ by BBP-Edge($b_i, \bar{b}_j, X, v, \bar{v}$)
  else
    $\bar{b} \leftarrow$ the unambiguous node of the set B^H that contains the vertex $\bar{v}$
    forall $b_i \in \mathrm{B}^G$, *where* bvb_i *is a path* **do**
      $\phi \leftarrow$ BBP-Edge($b_i, \bar{b}, X, v, \bar{v}$)
      if $\mathcal{W}(\phi) > \mathcal{W}(\varphi)$ **then** $\varphi \leftarrow \phi$
  return φ

5.4 Problem Variants

In this section, we discuss several variants of the BBP-MCIS problem. The variants aim to improve the computed similarity coefficients from Section 4.7. For MCS-based similarity coefficients it was shown, that they produces meaningful results when applied to molecular graphs (cf. Section 4.7 and [68]).

In Section 4.5, we investigated the largest weight common subtree embedding problem, which allows to skip vertices in the input trees and instead map disjoint paths to each other. In Subsection 5.4.1, we discuss the integration of this approach into our block-and-bridge preserving maximum common induced subgraph isomorphism algorithm (Algorithm 5).

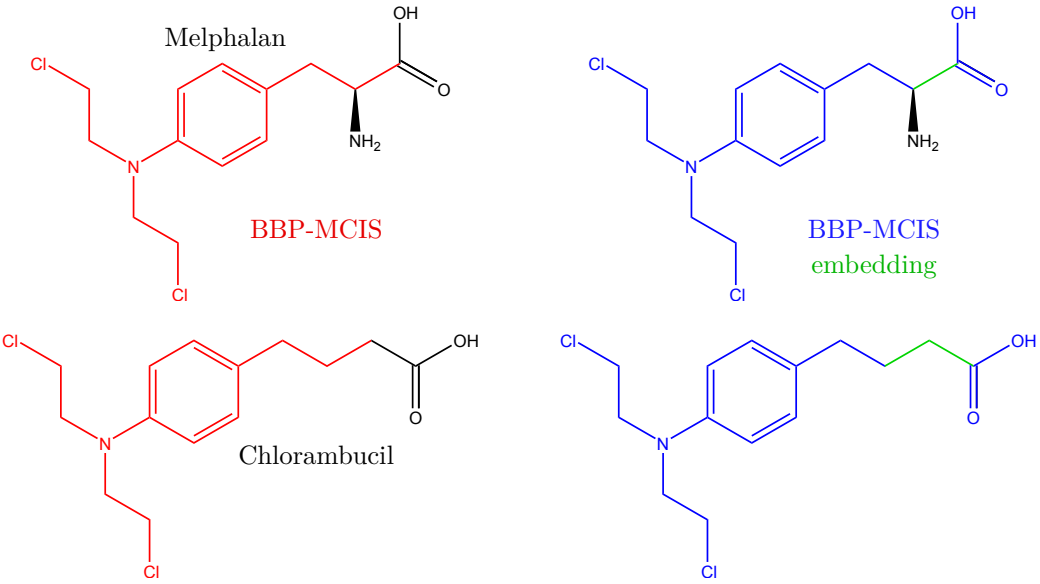
We can further improve the similarity coefficient by incorporating bioisosteres: Molecules with similar physical and chemical properties that produce similar biological properties. In Subsection 5.4.2, we analyze the necessary steps to directly map (non-isomorphic) subgraphs of the input graphs to each other and assigning a weight to this match.

We further present a solution to solve the BBP-MCIS problem on non-outerplanar graphs via a reduction to a modified maximum clique algorithm in Subsection 5.4.3. The modifications are necessary to ensure that the block-and-bridge preserving constraint holds.

In Subsection 5.4.4, we discuss the BBP-MCES algorithm presented by Schietgat et al. [111]. We focus on our suggested improvements [70] to this algorithm.

5.4.1 BBP-MCIS Embedding; LaWeCSE$_u$ Integrated into BBP-MCIS

The tree-based LaWeCSE$_u$ approach from Section 4.5 maps disjoint paths to each other. Unfortunately, that approach requires the input graphs to be trees. When dealing with a BBP-MCIS, we have the biconnected blocks on the one hand and the treelike bridges on the other hand. The



(a) BBP-MCIS (red)

(b) BBP-MCIS Embedding. Edges mapped to each other depicted blue. Path to edge depicted green.

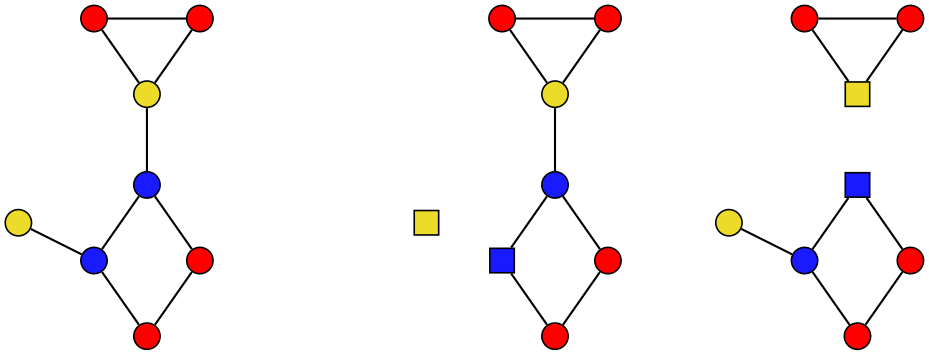
Figure 5.6: Molecular graphs of melphalan (top) and chlorambucil (bottom). The BBP-MCIS (a) maps three less vertices than the BBP-MCIS Embedding (b), where one vertex is skipped. We assume a labeled BBP-MCIS with a penalty $p = 1$.

bridges of each input graph form a forest with trees as its connected components. We can apply the LaWeCSE$_u$ approach between such trees from both input graphs. In other words, we may map disjoint paths formed completely by bridges to each other, while all inner vertices in the paths are skipped. We call this *BBP-MCIS embedding*.

Figure 5.6 depicts a comparison between two molecular graphs, one with a regular BBP-MCIS, one with a BBP-MCIS embedding. Both molecular graphs consist of a central ring (a block) and bridges to both sides; the bridges of each of those molecules form a forest of two trees. In this example, we may skip one vertex in the bottom right molecule, which allows mapping three further atoms. It has also been shown that allowing disconnected common subgraphs improves the quality of the similarity coefficients [82, 113]. In Section 6.4, we evaluate the usefulness of this approach.

Theorem 5.9. *We can compute a BBP-MCIS embedding between outerplanar graphs G and H under a weight function mapping to $\mathbb{R} \cup \{-\infty\}$ in time $\mathcal{O}(|G|\,|H|\,\Delta(G,H))$, where $\Delta(G,H) = 1$ if G or H is biconnected; otherwise, $\Delta(G,H) = \min\{\Delta_C(G), \Delta_C(H)\}$.*

Proof. This follows directly from the running time results of Theorem 5.8 (BBP-MCIS) and Theorem 4.29 (LaWeCSE$_u$). □



(a) G (b) Bioisostere candidates of G

Figure 5.7: (a) Vertex labeled graph G and (b) its four bioisostere candidates rooted at the quadratic vertices. For each bridge exist two candidates.

5.4.2 Bioisosteres [7]

This subsection focuses on improving the BBP-MCIS algorithm specifically for molecular input graphs. For molecules, (graph structural) similarity often goes along with similar chemical properties or similar biological activity (bioactivity). However, there are exceptions to both sides. The case of different structures with similar chemical properties or bioactivity can be approached by using an appropriate weight function. The weight function may, e.g., be defined such that we check for the fconv groups [95] like *hydrogen bond acceptor* or *aromatic* instead of the atom directly. Another approach is to map subgraphs (bioisostere candidates) instead of vertices (atoms) to each other and assign a weight to this match. That weight then depends on the similarity of the chemical properties or bioactivity and is retrieved from a database containing substitution rules between bioisostere candidates.

> **Definition 5.10 (Bioisostere candidate).** *Let (V, E) be a graph. For any bridge $uv \in E$, the two connected components induced by removing the edge uv are bioisostere candidates rooted at u and v, respectively.*

Each candidate is a rooted graph, cf. Definition 2.14. An exemplary graph with its bioisostere candidates is depicted in Figure 5.7.

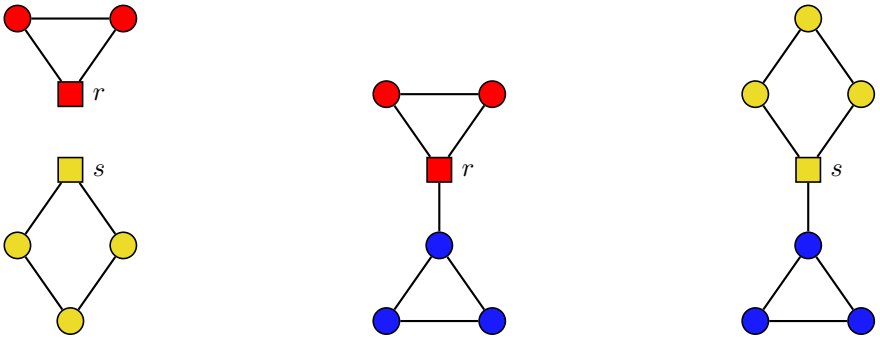
Defining the candidates through bridges is motivated by the fact that bridges represent linkers between parts of the molecule. Removing such a linker separates the molecule. By replacing substructures (bioisostere candidates) on these bridges, we obtain new molecules with possible similar chemical properties, even if their structure differs.

These substitutions are not inherent to the given molecular graphs. Rather, we have an additional data structure that stores a set of substitution rules.

> **Definition 5.11 (Substitution rule).** *Given two rooted graphs B_1^r and B_2^s and a positive weight ω, the triple (B_1^r, B_2^s, ω) is a substitution rule.*

Such a set of substitution rules can be generated from a database of known chemical properties or bioactivities. Then, it can be applied to any pair of molecules and considered in the computation

[7] This subsection presents an overview of the findings achieved during the Master's thesis *Effiziente Algorithmen für maximale gemeinsame Graphen zwischen Molekülen unter Berücksichtigung von Bioisosteren* from Kevin Nikiel, supervised by Prof. Dr. Petra Mutzel and the author of this thesis.



(a) B_1^r, B_2^s (b) G (c) H

Figure 5.8: (a) Bioisostere candidates of a rule $(B_1^r, B_2^s, 4)$ (b), (c) Labeled graphs (indicated by color) G, H. A Bio-BBP-MCISI maps the blue vertices regularly. The red and yellow vertices are mapped from the substitution rule. We obtain a total weight of 7 (3 from the blue vertices plus 4 from the rule).

of a BBP-MCISI ϕ. Given a rule (B_1^r, B_2^s, ω), we may map a bioisostere candidate B_1 in G directly to a bioisostere candidate B_2 in H. This requires r and s to be vertices of a bridge and that the vertices from B_1 and B_2 are not mapped by ϕ or another substitution rule. This substitution then contributes ω to the weight of ϕ. Any number of substitution rules may be applied simultaneously under the above restrictions.

> **Definition 5.12 (Bio-BBP-MCISI).** *A BBP common induced subgraph isomorphism ϕ, where we may additionally map bioisostere candidates from a given substitution rule set to each other (under the above restrictions), is denoted Bio-BBP common induced subgraph isomorphism. An isomorphism of maximum possible weight among them is a* Bio-BBP-MCISI.

An example of a Bio-BBP-MCISI is depicted in Figure 5.8. Finding and mapping the candidates to each other is a non-trivial task. A naive approach requires to solve graph isomorphism problems between the bioisostere candidates of the input graphs and the substitution rules. We use a complete graph invariant on all the substitution rules and the candidates from the input graphs to avoid this.

> **Definition 5.13 (Complete Graph Invariant).** *Let $\mathcal{G}$ be the set of all graphs. A* complete graph invariant *I is a function on $\mathcal{G}$, such that G is isomorphic to H if and only if $I(G) = I(H)$ for any two graph G and H of $\mathcal{G}$.*

An example of a complete graph invariant is the canonical form; it is a unique adjacency matrix. Other graph invariants are unique strings [75, 88]. These invariants accept labeled graphs as input. For a graph G, such a string is of length $\mathcal{O}(|V_G| + |E_G|)$. The invariants can be modified to respect a given root vertex, e.g., by a unique label coding of these root vertices. Using algorithms tailored for planar graphs, we obtain a polynomial running time [75]. For outerplanar graphs, even a linear running time is possible [81]. In practice, we can use a hash table or other means to speed up the string comparison.

Bio-BBP-MCISI Algorithm. To compute a Bio-BBP-MCISI given a set of substitution rules, we first compute a complete graph invariant on all bioisostere candidates from the rules in a

preprocessing step. In practice, it is advisable to do this only once and then store the results for future Bio-BBP-MCISI computations. Then we do the same for the bioisostere candidates from the input graphs. Since the candidates are often subgraphs of other candidates (e.g., in Figure 5.7b, the top right candidate is a subgraph of the candidate to its left), we can reuse the computed strings of the subgraphs to speed up the preprocessing phase further.

The procedures BBP-Edge and BBP-SingleVertex of Algorithm 5 are then modified as follows. In line 7 of Procedure BBP-Edge and line 7 of Procedure BBP-SingleVertex, we check if there is a substitution rule for the bioisostere candidates rooted at c and $\bar{c}$ (BBP-Edge) or v and $\bar{v}$ (BBP-SingleVertex). If yes, and this rule is of greater weight, we consider the rule's weight for $w(b_i, \bar{b}_j)$ instead. We further have to consider the case that the initial chosen B-node is contained in a bioisostere candidate. We find such a candidate by modifying line 4 of Procedure SetSx. If b is a bridge and X is not empty, let r be the unique vertex contained in both X and b. We then check for a bioisostere rule with the candidate rooted at r. This candidate contains the initial chosen B-node. The two candidates from H are rooted at the vertices of $\bar{b}$. We additionally need to modify Algorithm 5 to output the mapping from the rules, if any were used.

Proposition 5.14 ([a]). *Given a preprocessed substitution rule set and outerplanar graphs G and H, we can compute a Bio-BBP-MCISI in time $\mathcal{O}(\Delta(G,H)\cdot|G|^2|H|^2)$, where $\Delta(G,H)=1$ if G or H is biconnected; otherwise, $\Delta(G,H)=\min\{\Delta_C(G),\Delta_C(H)\}$.*

[a] From Corollary 3.6.4 of Kevin Nikiel's master's thesis.

The increased running time over Algorithm 5 is due to the comparison of the used graph invariant. Even with a hash table, we may have to compare the entire coding to determine if a rule applies.

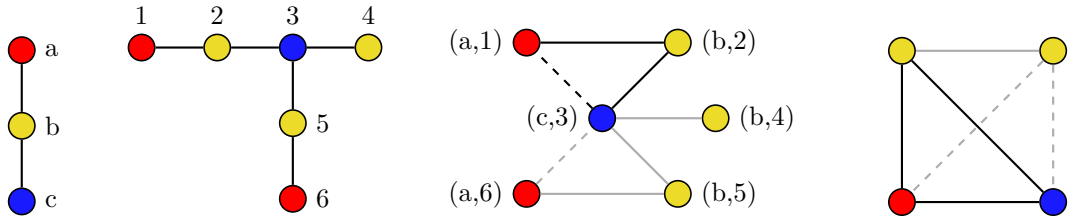
5.4.3 Non-Outerplanar Graphs

In this subsection, we discuss how to compute a BBP-MCIS if the input graphs are not outerplanar. An MCS between non-outerplanar graphs can be computed by reducing the problem to the maximum clique problem [67, 12, 87, 106]. Therefore, a so-called product graph from the input graphs is constructed, on which the clique problem is solved. This yields a result that is not necessarily BBP. We resolve this by applying the clique reduction to non-outerplanar pairs of blocks from the input graphs instead. More precisely, we replace the 2-MCIS algorithm for biconnected outerplanar graphs from Section 5.2, which we use as a subroutine in Algorithm 5: in Procedure BBP-Edge, line 4, the valid isomorphisms are then those computed from the clique reduction if at least one block is not outerplanar. This approach additionally reduces the practical running time in comparison with a pure clique based algorithm, as it is cheaper to perform the reduction on several small graphs than on one large graph.

Maximum clique algorithms usually rely on backtracking [10, 87]. In the unweighted case, there is often some strategy involved, like branch and bound [115] or local search [16, 116]. We can also use constraint models to solve the problem [55]. In the following, we present a reduction to the maximum weight clique problem and the additional steps necessary to ensure that the computed common subgraph is biconnected.

The product graph and reduction to the clique problem. The product graph, also known as vertex product graph or compatibility graph, is defined as follows.

Definition 5.15 (Product Graph). *Given graphs G and H, the* product graph $G\triangledown H$ *consists of* product nodes $V(G\triangledown H)=\{(u,v)\mid u\in V(G), v\in V(H)\}$ *and edges* $E(G\triangledown H)=$



(a) G (b) H (c) $G \triangledown H$ (d) Bicon. c-clique

Figure 5.9: (a), (b) Two vertex labeled graphs (color coded). (c) The product graph of G and H; c-edges solid, d-edges dashed. A maximum c-clique is formed through the black edges. (d) A maximum *biconnected* c-clique formed through black edges.

$$\{\{(u,v),(u',v')\} \mid u \neq u',\ v \neq v',\ \textit{and}\ (u,u') \in E(G) \Leftrightarrow (v,v') \in E(H)\}$$

We use the term *product nodes* to distinguish them from the vertices of the input graphs. Each product node (u, v) represents a mapping from $u \in V(G)$ to $v \in V(H)$. Each edge $\{(u, v), (u', v')\}$ represents the compatibility between two product nodes, i.e., if we may simultaneously map u to v and u' to v' based on the presence or absence of edges between those vertices. Given a product graph, any clique in the product graph represents a common induced subgraph isomorphism ϕ in the original graph, which we derive from the clique's product nodes.

Since $G[\mathrm{dom}(\phi)]$ is not necessarily connected, so-called *c-cliques* were introduced, see e.g., [67, 106]. Each edge $\{(u, v), (u', v')\}$ of the product graph, where $(u, u') \in E(G)$ and $(v, v') \in E(H)$, is called *c-edge* for connecting edge. The other edges are called *d-edges* since they represent disconnection. A c-clique is a clique, which is connected through c-edges, i.e., there is a path consisting only of c-edges between each two different product nodes within the clique.

When dealing with labeled graphs and a label function l, only such product nodes (u, v) are contained in $V(G \triangledown H)$, where additionally $l(u) = l(v)$. For c-edges $\{(u, v), (u', v')\}$, additionally $l(uu') = l(vv')$ is required. This ensures that only vertices and edges with the same label are mapped to each other by the corresponding isomorphism ϕ. An example of two vertex labeled graphs and their product graph is depicted in Figure 5.9.

When dealing with weighted graphs as in our BBP-MCISI algorithm, we assign a weight to each product node and each c-edge defined through the weight function ω. Then we are interested in finding a clique of maximum weight instead of maximum size.

Generally, the product graph is not computed explicitly to save memory. Instead, only the product nodes are stored, and the adjacency is computed on demand from the above definition.

Computing a maximum weight clique. A strategy to find a maximum cardinality clique is to enumerate all maximal cliques [67, 12]. This originates from an algorithm by Bron and Kerbosch [10]. In this approach, four sets are maintained: The c-clique under expansion R; prospective nodes P connected to each node of R through at least one c-edge – these are candidates for expansion; a set Q of nodes connected through d-edges only; and a set of forbidden nodes X – the already processed nodes.

The clique algorithm subsequently starts from each vertex v of the product graph, initializes $R := \{v\}$, and determines the set P. Then, it recursively adds nodes from P to R while adjusting the other sets. When there are no further nodes in P, a maximal clique has been found. When all maximal cliques have been enumerated, a maximum clique among them is outputted. The backtracking occurs by adding other prospective nodes in P first. By taking the forbidden nodes X into account, the algorithm enumerates no solution twice. The pseudocode is listed in [12].

During the building of the c-clique, we keep track of its current weight. Whenever a product node x is shifted from P to R, we add its weight and the weight of all c-edges incident to x to the solution. Among the maximal cliques, we store one of maximum weight. When dealing with negative weights, it is not sufficient to consider maximal cliques only. Instead, we also check on non-maximal cliques if their weight is greater than the previously found best solution. This approach is somewhat similar to non-maximal solutions in our 2-MCIS algorithms for negative weights in Subsection 5.2.2. We also have to ensure that the computed cliques represent biconnected subgraphs. We address that in the next paragraph.

Biconnected c-cliques in the product graph. Since blocks are mapped to blocks, such that they form biconnected subgraphs, the computed clique needs to be biconnected through c-edges, too. As before, it is not sufficient to take only the maximal c-cliques into account. Consider the product graph in Figure 5.9d. It has a single maximal c-clique only: the whole graph. However, that graph is not biconnected through c-edges. Contrary to c-cliques, a direct approach to enumerate biconnected c-cliques is difficult. The reason is, we cannot simply maintain a biconnected c-clique throughout the building process of the maximal clique: Assume two equal long cycles as input graphs; in the clique algorithm, the R-nodes form non-biconnected c-cliques until the very last step, when the clique represents the whole cycle.

To solve the problem, we keep track of the current biconnection status of $G[R]$. We derive the following rules to do this.

1. If $|R| < 3$, then $G[R]$ is not biconnected.

2. If a node x from P is shifted to R, such there is only one c-edge between the product nodes from R and x, then $G[R]$ is not biconnected.

3. If $G[R]$ was biconnected before and a node x from P with at least two distinct c-edges to R is shifted to R, then $G[R]$ remains biconnected.

4. Otherwise, we recompute if $G[R]$ is biconnected via DFS. Here, $G[R]$ was not biconnected before but might be through the new vertex x.

Further, we maintain the sets P and Q, such that for each vertex $x \in P \cup Q$, there are at least two c-edges in total to other product nodes in R, P, and Q. Vertices with less than two c-edges do not allow a biconnected solution and are removed.

Since the clique reduction by Cazals and Karande [12] (with intermediate results, i.e., non-maximal cliques) lists each c-clique exactly once, we obtain each biconnected c-clique exactly once with the above modifications. Among them, we store a clique of maximum weight. Since the maximum biconnected c-clique algorithm is based on clique enumeration, it has no polynomial time bound.

Further approaches. There exist algorithms for the maximum node- and for the maximum edge-weighted clique problem, see e.g., [16, 116, 114]. However, these algorithms do not allow both edge and node weights. The weights are further required to be positive natural numbers. If the given weight function ω allows such restriction, these algorithms might be superior to the above enumeration approach. Even then, the results are not necessarily biconnected through c-edges. It requires further investigation in how far such algorithms are applicable.

There are further approaches to compute a maximum clique, e.g., a constraint programming model, MaxSAT reasoning, and preprocessing steps, as discussed by McCreesh et al. [85].

5.4.4 BBP-MCES [8]

Schietgat, Ramon, and Bruynooghe [111] proposed to determine a block-and-bridge preserving maximum common edge subgraph and developed an algorithm with a claimed running time of $\mathcal{O}(n^{2.5})$ for two outerplanar graphs of order n. While the authors presented promising experimental results on graphs representing molecules, we show that their theoretical analysis of their approach is flawed.

Their BBP-MCES algorithm for outerplanar graphs decomposes the two input graphs into subgraphs with distinct root vertices referred to as *parts* (a formal definition follows later in this subsection). An MCES problem for all compatible pairs of parts is then solved using a dynamic programming strategy. Here, a series of weighted maximal matching instances arise as subproblems. It has been claimed [111, Theorem 2] that for two outerplanar graphs G and H, the proposed BBP-MCES algorithm runs in time

$$\mathcal{O}\left(|V(G)| \cdot |V(H)| \cdot (|V(G)| + |V(H)|)^{\frac{1}{2}}\right),$$

which is $\mathcal{O}(n^{2.5})$ for $|V(G)| = |V(H)| = n$. We show that this bound cannot be obtained by the presented techniques.

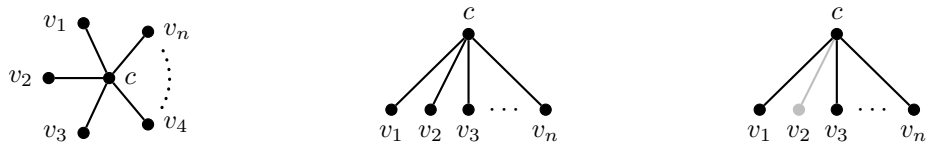
Solving weighted maximum matching problems. The algorithm makes use of a subroutine for solving the maximum weight matching problem in bipartite graphs, where weights are real values. The matching instances arising in the course of the algorithm may be complete bipartite graphs with a quadratic number of edges; see the counterexample discussed in the following paragraph. Hence, the running times given in the following refer to bipartite graphs with n vertices and $\Theta(n^2)$ edges to improve readability. The authors propose to use the algorithm by Hopcroft and Karp [57] to solve an instance of the problem in time $\mathcal{O}(n^{2.5})$. Since this algorithm computes a matching of maximal cardinality but is not designed to take weights into account, it cannot be applied to the instances that occur.

The best-known approaches for the weighted problem allow solving instances with n vertices and $\Theta(n^2)$ edges in time $\mathcal{O}(n^3)$, e.g., the established Hungarian method (cf. Table 3.1). When we assume weights to be integers within the range of $[0..N]$, scaling algorithms would become applicable, which solves the problem in time $\mathcal{O}(n^{2.5} \log N)$. This running time is still worse than the time bound for the algorithm by Hopcroft and Karp by a factor depending logarithmically on N. Moreover, it is desirable to allow that the weight of a common subgraph is measured by a real number depending on the labels of the vertices and edges it contains, cf. [111, Definition 2]. This leads to real edge weights in the matching instances.

In summary, no better bound than $\mathcal{O}(n^3)$ on the worst-case running time can be assumed for the subproblem of solving weighted maximal matching instances with n vertices.

The number of matching instances. We consider a particularly simple counterexample to illustrate that the running time required to solve the matching problems cannot be bounded by $\mathcal{O}(n^{2.5})$. We identify the flaw regarding the analysis, which led to this incorrect result [111, Proof of Theorem 2]. More precisely, we show that for two graphs G and H of order n, the BBP-MCES algorithm performs $\Theta(n)$ calls to the subroutine for weighted maximal matching [111, Algorithm 2, MaxMatch] with instances of size $\Theta(n)$. Since the relationship between the matching instances is not considered in [111], we assume that each instance is solved separately in cubic time, as shown before. Therefore, no better bound than $\mathcal{O}(n^4)$ can be given on the total running time.

[8] This subsection primarily consists of our findings in *A note on block-and-bridge preserving maximum common subgraph algorithms for outerplanar graphs* [70], Section 3.



(a) A star graph H (b) Rooted graph H^c (c) Part in $\mathcal{B}_H$

Figure 5.10: (a) A star graph of order $n+1$, (b) the star graph rooted at the center vertex and (c) an elementary part $H^c \setminus \{v_2\}$ obtained from H^{v_2}, where the gray vertex with its incident edge is deleted.

Let the two graphs G and H both be star graphs of order $n+1$, i.e., trees with all but one vertex of degree one as depicted in Figure 5.10. Since trees are outerplanar, G and H are valid input graphs for BBP-MCES. The algorithm presented in [111] relies on decomposing the two input graphs into their parts.[9] $\mathit{Parts}(T^r)$ of a rooted tree T^r is recursively defined as follows [111, Definitions 20, 23, 26].

(i) $T^r \in \mathit{Parts}(T^r)$,

(ii) if $P^p \in \mathit{Parts}(T^r)$ and p is incident to exactly one edge $\{p, v\}$, then the graph $(P \setminus \{p\})^v$ is in $\mathit{Parts}(T^r)$,

(iii) if $P^p \in \mathit{Parts}(T^r)$ and p is incident to the edges $\{p, v_1\}, \ldots, \{p, v_k\}$, $k \geq 2$, then for each edge $\{p, v_i\}$, $1 \leq i \leq k$, the connected component of the graph $P^p \setminus \{\{p, v_j\} \mid j \neq i\}$ containing p as root is in $\mathit{Parts}(T^r)$.

For the first input graph G, an arbitrary root vertex r is selected to define its parts. Let G be the star graph, r its center vertex and let $L(G)$ denote its leaves, then

$$\mathit{Parts}(G^r) = \{G^r\} \cup \{(\{r, v\}, \{\{r, v\}\})^r \mid v \in L(G)\} \cup \{(\{v\}, \emptyset)^v \mid v \in L(G)\}.$$

The parts of the star graph are the graph itself, the subgraphs consisting of the individual edges and the subgraphs consisting of the leaves. For the second input graph H, its parts are defined as $\mathit{Parts}^*(H) = \cup_{s \in V(H)} \mathit{Parts}(H^s)$ [111, Definition 27]. Therefore,

$$\begin{aligned}\mathit{Parts}^*(H) =& \{H^s \mid s \in V(H)\} \cup \{(e, \{e\})^c \mid e \in E(H)\} \cup \\ & \{(\{v\}, \emptyset)^v \mid v \in L(H)\} \cup \underbrace{\{H^c \setminus \{v\} \mid v \in L(H)\}}_{\mathcal{B}_H},\end{aligned}$$

where c is the unique center vertex of H and $\mathcal{B}_H$ are the subgraphs rooted at c obtained by deleting a single leaf with its incident edge, cf. Figure 5.10c.

To solve the problem, a variant of BBP-MCES, which requires mapping the root of one part to the root of the other, is solved for specific pairs of parts denoted by $\mathit{Pairs}(G, H)$. If the roots of both parts have multiple children, a matching problem between them must be solved. Such parts are referred to as *compound-root graphs*, and the parts associated with the children are *elementary parts*, respectively [111]. Note that this is the case for G^r and all the parts in $\mathcal{B}_H$; according to [111, Definition 28], we have $\{G^r\} \times \mathcal{B}_H \subseteq \mathit{Pairs}(G, H)$. For each pair (G^r, Q), $Q \in \mathcal{B}_H$, a weighted maximal matching instance is constructed, where the vertices correspond to the elementary parts of G^r and Q [111, Algorithm 2, RMCSCompound]. The edge weights are

[9] The approach greatly simplifies for trees, and we have shortened the required definitions accordingly. Please note that [111, Algorithm 4 and Algorithm 3, lines 11-18] will not be required to solve the problem on trees.

determined by the solutions for pairs of smaller parts and depend on the possibly real-valued weights of vertex and edge labels of the common subgraph. The number of elementary parts of G^r is n, the number of elementary parts of each Q in $\mathcal{B}_H$ is $n-1$. Hence, each of these matching instances has $2n-1$ vertices and $n(n-1)$ edges and thus requires time $\mathcal{O}(n^3)$. The number of such pairs is $|\{G^r\} \times \mathcal{B}_H| = n$. If each matching instance is solved separately, no better bound than $\mathcal{O}(n^4)$ on the total running time of the algorithm can be given, and the analysis of [111, Theorem 2] is too optimistic.

Consequently, there must be an error in its proof: The authors claim that every vertex $g \in V(G)$ and every vertex $h \in V(H)$ has at most $\deg(g)$ (resp. $\deg(h)$) elementary parts involved in a maximal matching. While this statement is correct, the subsequent analysis does not take into account that there may be up to $\deg(h)$ matching instances of that size for a vertex $h \in V(H)$. More precisely, the total time spent in RMCSCompound for solving matching instances is claimed to be bounded by

$$T_{\text{comp}} = \sum_{g \in V(G)} \sum_{h \in V(H)} T_{\text{WMM}}(\deg(g) + \deg(h)), \tag{5.1}$$

where $T_{\text{WMM}}(k)$ is the running time for solving a weighted maximal matching instance with k vertices [111, p. 361]. Actually, the procedure considers all pairs of compound-root graphs, where each pair leads to a matching instance containing one vertex for each of the associated elementary parts. The counter-example above shows that for a vertex $h \in V(H)$, there may be $\deg(h)$ compound-root graphs with root h, each with $\deg(h) - 1$ elementary parts. In addition, there is one compound-root graph with root h and $\deg(h)$ elementary parts. Therefore, a correct upper bound is

$$T_{\text{comp}}^{\text{corrected}} = \sum_{g \in V(G)} \sum_{h \in V(H)} (\deg(h) + 1) \cdot T_{\text{WMM}}(\deg(g) + \deg(h)). \tag{5.2}$$

In the counter-example the degree of the center vertex is not bounded, which leads to the additional factor of n appearing in $T_{\text{comp}}^{\text{corrected}}$, but not in T_{comp}.

Exploiting the structure of the matching instances. The matching instances emerging for the counter-example are closely related since the symmetric difference of the elementary parts of $Q_1 \in \mathcal{B}_H$ and $Q_2 \in \mathcal{B}_H$ with $Q_1 \neq Q_2$ contains exactly two elements. In Section 3.5, we showed that this fact could be exploited by solving the all-cavity MWM problem efficiently in one pass. The same technique can be used to improve the running time of their BBP-MCES algorithm. This might allow the same time bound as for our BBP-MCIS algorithm (cf. Theorem 5.8) since all their other subroutines are claimed to be within time $\mathcal{O}(|G|\,|H|)$. We experimentally compared our maximum common subtree algorithm [26] to the BBP-MCS algorithm published in [111] using the implementation provided by the authors. The running times reported for the BBP-MCES algorithm actually suggest a growth of $\Omega(n^5)$ on star graphs.

5.5 Polynomial Delay Enumeration of all Block-and-Bridge Preserving Maximum Common Induced Subgraph Isomorphisms

In Section 4.6, we studied the enumeration of maximum weight common subtree isomorphisms and provided a polynomial delay result. Regarding the BBP-MCISI problem, we not only map bridges to each other but also blocks. Thereby, we separated the solution space into sets S_x and

subsets $\mathcal{P}_{xy}$. To enumerate all BBP-MCISIs, we join both approaches. This additionally requires the enumeration of all 2-MCIS between pairs of blocks. In the following, we first study the latter problem, then we outline the enumeration tree, and finally we provide running time results.

5.5.1 Enumeration of all 2-MCIS between two Blocks

In Section 5.2, we discussed the 2-MCIS problem between biconnected outerplanar graphs. The computation occurred by enumerating all maximal solutions exactly once and then selecting one of maximum weight. Since we allow arbitrary weights between vertices and edges, it is not sufficient to enumerate only the maximal solutions to obtain all maximum weight solutions. We discussed this in Subsection 5.2.2. Another example is a weight function ω defined as constant 0. Then each biconnected common subgraph isomorphism (whether maximal or not) is a maximum solution.

Let us recapitulate the dynamic programming approach to compute a 2-MCIS with negative weights. This approach was related to the maximum sum subarray problem. After computing the dual graph of a maximal common subgraph, we decide for each face F' if it contributes a positive weight $F_{\text{add}}(F') > 0$ to its parent vertex (face) F in the dual graph. If yes, we add this value to $F_{\text{add}}(F)$ and $F_{\text{maximum}}(F)$. For negative weights, we do not add this value. If $F_{\text{add}}(F') = 0$, then adding $F_{\text{add}}(F')$ has no impact on $F_{\text{add}}(F)$ and $F_{\text{maximum}}(F)$. If we want to enumerate all 2-MCISs in the case of $F_{\text{add}}(F') = 0$, the enumeration tree splits into adding face F' to the biconnected subgraph and not adding face F'. Integrated into the BBP-MCISI algorithm, we may have a fixed mapping $v \to \bar{v}$ given. In that case, we assume the dual graph to have a root face incident to v. Further, there may be an excluded vertex x. As before, this vertex contributes weight $-\infty$ to any common subgraph.

The Enumeration Tree for the 2-MCIS Problem. Let $\mathcal{W}_{\max}$ be the weight of a 2-MCIS under the above possible restrictions. Then the enumeration of all 2-MCISs occurs as follows.

1. For all *maximal* biconnected common subgraph isomorphisms ϕ as in line 3 of Algorithm 4, where $\phi(v) = \bar{v}$, perform step 2.
2. For all faces F, where $F_{\text{maximum}}(F) = \mathcal{W}_{\max}$, perform step 3.
3. For all children F' of F, where $F_{\text{add}}(F') > 0$, recursively perform step 2; further let $\mathcal{P}$ be the power set of all children F' of F, where $F_{\text{add}}(F') = 0$. For each set $P \in \mathcal{P}$ recursively and simultaneously perform step 2 on each face $F' \in P$.

The recursion stops if there are no children F' of F, where $F_{\text{add}}(F') \geq 0$.

As an example, consider each weight to be 0 in Figure 5.5. Then the enumeration produces the 2-MCISs defined through the following faces: $D, C, CD, A, B, BA, BC, BCD, BAC, BACD$.

> **Lemma 5.16.** *If we use additional data structures to store (references to) the maximal subgraphs and all faces F, where $F_{maximum}(F) = \mathcal{W}_{\max}$, and all faces F', where $F_{add}(F') \geq 0$, we can enumerate each subsequent 2-MCIS φ in time $\mathcal{O}(|\varphi|)$.*

If we have non-outerplanar blocks, we can directly use the approach from Subsection 5.4.3. There we obtain a maximum solution by enumerating each biconnected common subgraph isomorphism between the blocks. During the enumeration process within the BBP-MCISI algorithm, the biconnected common subgraphs are enumerated multiple times. In practice, it is beneficial to trade space for time by storing the maximum solutions (if their number does not exceed a given threshold) since the clique enumeration is much slower than the 2-MCIS approach for outerplanar graphs.

5.5.2 Enumeration of all Block-and-Bridge Preserving Maximum Common Induced Subgraph Isomorphisms

Similar to the maximum weight common subtree isomorphism problem, we first determine the weight $\mathcal{W}_{\max} \coloneqq \mathcal{W}(G, H)$ with Algorithm 5 using the Hungarian method to compute the MWMs and the equality subgraphs. We also store the returned weight of each call of BBP-Edge and BBP-SingleVertex, i.e., for the subsets $\mathcal{P}_{xy}$.

Then we run a variant of Algorithm 5, where we do not call the procedures BBP-Edge and BBP-SingleVertex to compute an isomorphism; instead, we check if the stored weight for $\mathcal{P}_{xy}$ is equal to $\mathcal{W}_{\max}$. If yes, variants of BBP-Edge and BBP-SingleVertex are called. Here, we firstly consider *all* valid isomorphisms of maximum weight. We do this using the method from Subsection 5.5.1. Additionally, we do not compute a single MWM to map the cutvertices to each other; instead, we enumerate the MWMs using the previously computed equality subgraphs.

Then, similar to the MWCSI problem, we recursively repeat the above steps until we reach the leaves in the BC-Tree.

5.5.3 Running Time Analysis

Let G and H be outerplanar graphs, and α be the number of different BBP-MCISIs. If at least one graph of G and H is biconnected, we directly obtain a polynomial delay algorithm with polynomial total time of $\mathcal{O}(|G|\,|H| + \alpha \min\{|G|, |H|\})$ from Proposition 5.5 and Lemma 5.16.

Otherwise, neither G nor H is biconnected. Then let $\Delta = \min\{\Delta_C(G), \Delta_C(H)\}$ and $d = \max\{\Delta_C(G), \Delta_C(H)\}$. From Theorem 5.8 and computing the equality subgraphs, we can compute the weight of a BBP-MCISI in time $\mathcal{O}(|G|\,|H|(\Delta + \log d))$. The time to add vertices from the blocks is bounded by $\mathcal{O}(\min\{|G|, |H|\})$. The average time to add vertices from the MWMs and the bridges is bounded by $\mathcal{O}(\min\{|G|, |H|\} + \Delta_C(G)\Delta_C(H))$, cf. Theorem 4.32. Thus we obtain the following result, which is also the main enumeration result of this thesis.

Theorem 5.17. *Let G and H be outerplanar graphs under an arbitrary weight function ω. Let α be the number of different BBP-MCISIs between G and H.*

If G or H is biconnected, we can enumerate all α BBP-MCISIs with polynomial delay, where $\mathcal{O}(|G|\,|H|)$ is the time until the first output and $\mathcal{O}(\min\{|G|, |H|\})$ for each subsequent output.

Otherwise, let $\Delta \coloneqq \min\{\Delta_C(G), \Delta_C(H)\}$ and $d \coloneqq \max\{\Delta_C(G), \Delta_C(H)\}$. Then we can enumerate all α BBP-MCISIs with polynomial delay $\mathcal{O}(|G|\,|H|(\Delta + \log d))$ and in total time $\mathcal{O}(|G|\,|H|(\Delta + \log d) + \alpha(\min\{|G|, |H|\} + \Delta_C(G)\Delta_C(H)))$.

5.6 Summary and Future Work

In this chapter, we first introduced the block-and-bridge preserving maximum common induced subgraph problem. A subproblem to this problem is computing a biconnected maximum common induced subgraph between each pair of blocks of the input graphs. We discussed this in Section 5.2. In Section 5.3, we solved the BBP-MCIS problem on outerplanar graphs. For two outerplanar graphs G and H under a weight function mapping to $\mathbb{R} \cup \{-\infty\}$, we proved a running time of $\mathcal{O}(|G|\,|H|\,\Delta(G, H))$, where $\Delta(G, H) = 1$ if G or H is biconnected; otherwise $\Delta(G, H) = \min\{\Delta_C(G), \Delta_C(H)\}$. Allowing negative weights improves our previously published algorithm presented in [27], where only non-negative weights are allowed. In Section 5.4, we outlined the integration of the LaWeCSE$_{\text{u}}$ approach into the BBP-MCIS algorithm. We further discussed bioisosteres, a clique-based approach for non-outerplanar input graphs, and the BBP-MCES

algorithm by Schietgat et al. [111]. For the latter, we discussed flaws in their theoretical running time analysis and suggested improvements based on the results in Section 3.5. Lastly, in Section 5.5, we presented a polynomial delay enumeration algorithm for the BBP-MCIS problem.

This chapter extended the results from the maximum weight common subtree isomorphism problem to the BBP-MCIS problem between outerplanar graphs without affecting the worst-case running time. This is independent of whether the weights are integral or real-valued. There are results for the BBP-MCIS problem on a more general graph class, e.g., series-parallel graphs [71]. However, therein no labels or weights are supported, and the proven time bound is $\mathcal{O}(n^6)$, which is much higher than our proven time bound in Theorem 5.8.

Open Problem 5.1. *Can we compute a BBP-MCIS between two graphs of a more general graph class than outerplanar graphs within the same time bound as in Theorem 5.8?*

The BBP-MCES algorithm presented in [111] computes a single optimal solution only. For the BBP-MCIS problem, we presented a polynomial delay enumeration algorithm. The combination of both approaches, the enumeration of all BBP-MCESs, remains open.

Open Problem 5.2. *Can we enumerate all BBP-MCESs of two outerplanar graphs with polynomial delay? Within which time bound?*

Das Experiment, dem nicht eine Theorie, d.h. eine Idee vorausgeht, verhält sich zur Naturforschung wie das Rasseln einer Kinderklapper zur Musik.

Justus von Liebig
1803 – 1873

Chapter 6

Experimental Evaluation

In this chapter, we experimentally evaluate the algorithms from the previous chapters. Section 6.1 provides details of our implementation in C++. This includes the available command line parameters. Following that, in Section 6.2, we provide details of the hardware the tests run on. In Section 6.3, we conduct synthetic tests to evaluate the theoretically predicted run times. Section 6.4 compares the algorithms to related state of the art algorithms on real-world data.

6.1 Implementation

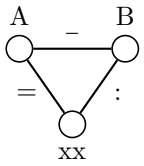
In this section, we discuss the implementation details. The complete source code of our project is available on GitHub.[1] We used the C++ programming language with the minimum required feature standard C++ 11. The software was tested both on Ubuntu[2] using the `GCC` compiler of versions 5 and above and Visual Studio Community 2019.[3] As an additional framework, we used the OGDF library [15]. The software is command-line based, such that test runs can be automated, e.g., using python scripts. We optionally support a graphical output of the graphs and their common vertices and edges, cf. Subsection 6.1.4.

6.1.1 Input Files

We accept two types of input files: labeled graph files and a file coding the weight function. As graph files, we support the *graph modeling language (GML)* and the *FOG* format. Each graph file may contain multiple graphs, and it is possible to mix both formats at will. The labels are interpreted as strings and internally coded to ascending natural numbers.

GML. The graph modeling language (GML) is commonly used, and its reading and parsing are supported directly through the OGDF framework. However, since the graphs' coding is in a meta-language style format, files tend to be rather large, and on slow storage devices, this might bottleneck the computation. A K_3 with vertex and edge labels is shown in Figure 6.1a. The GML code for that graph is shown in Figure 6.2.

[1] https://github.com/AndreDroschinsky/LaWeCSE
[2] https://ubuntu.com/download
[3] https://visualstudio.microsoft.com/de/vs/



(a) The labeled K_3 used in the GML and FOG code examples

```
# K3 3 3
A B xx
1 2 - 2 3 : 3 1 =
```

(b) k3.fog – A labeled K_3 in the FOG format

Figure 6.1: Labeled K_3 as graph and in the FOG format

```
graph [
        directed 0
        id 1
        label "K3"
        node [
                id 1
                label "A"
        ]
        node [
                id 2
                label "B"
        ]
        node [
                id 3
                label "xx"
        ]
        edge [
                source 1
                target 2
                label "-"
        ]
        edge [
                source 2
                target 3
                label ":"
        ]
        edge [
                source 3
                target 1
                label "="
    ]
]
```

Figure 6.2: k3.gml – A labeled K_3 in the graph modeling language

FOG. Schietgat, Ramon, and Bruynooghe use this format to compute a BBP-MCESI [111]. We decided to support it since this file format is very compact and allows for an easier benchmarking of our BBP-MCISI algorithm against their BBP-MCESI algorithm. Each graph is coded through three lines.

1) #<*name*> used in our output; <N> number of vertices; <M> number of edges

2) <label vertex 1>; ...; <label vertex N>

3) M triples of: <start vertex>; <end vertex>; <edge label>

The same K_3 as above is shown in Figure 6.1b.

Weight function. The weight function ω is coded in a file through different commands. Firstly, default vertex and edge weights can be specified separately. If none are specified, then $\omega(l_1, l_2) = 1$ if $l_1 = l_2$, otherwise $\omega(l_1, l_2) = -\infty$. Secondly, we can specify the weight $\omega(l_1, l_2)$ of any labels l_1, l_2 directly via the `L` letter. A matrix style coding is possible via `TARGET` and `V`. A line beginning with `//` is not parsed. Figure 6.3 exemplifies a coding for all different types with comments for general use.

```
// weights are doubles, with '-' as "forbidden";
//  separate entries by ';'

// default weights for same and different node labels
DEFAULT_NODE 1.0 ; -
// default weights for same and different edge labels
DEFAULT_EDGE 0.0 ; -

// label pair weight L <L1>;<L2>;<weight>
//  Sets w(L1,L2)=weight; example:
L C;O;0.5
// this sets w(C,O)=0.5

// target label vector TARGET <L1>;...;<Ln>, used for
//  weight vector V (see next), example:
TARGET C;O;N

// weight vector V <L>;<weight1>;...;<weightn>
//  set w(L,L1)=weight1,...,w(L,Ln)=weightn,
//  empty entries ';;' are skipped; examples:
V O;0.5;1;0
// this sets w(O,C)=0.5; w(O,O)=1; w(O,N)=0
V C; ;0.1
// this sets w(C,O)=0.1 and keeps defaults for w(C,C) and w(C,N)
//  default for w(C,C) from empty entry '; ;'
```

Figure 6.3: An exemplary coding of a weight function ω

6.1.2 Implementation Details

In the following, we provide details about the implementation of the algorithms developed during this thesis. We did not implement algorithms tailored for integral weights; instead we allow arbitrary weights by using the C++ type `double` in our sources.

Maximum Weight Matching. As implementation of a maximum weight matching algorithm, we used a source provided by Fritz Bökler, which origins from an $\mathcal{O}(n^3)$ implementation for the MWPM problem by Papadimitriou and Steiglitz [100]. We also have an experimental implementation for the result from Proposition 3.13. This, however, has not been integrated into the main branch yet. For the all-cavity maximum weight matching problem, we implemented the result from Proposition 3.24. This provides the necessary equality subgraphs, such that the enumeration of all maximum common subgraphs is possible. The MWM algorithms are used as a subroutine for the following algorithms.

Maximum Weight Common Subtree Isomorphism. We implemented the maximum weight common subtree isomorphism approach from Section 4.3. Additionally, we allow skipping vertices as in the LaWeCSE_u approach from Section 4.5. The required matchings are computed by using the MWM implementation as described above. Here, the penalty for skipping vertices is of C++ type `double`.

Block-and-Bridge Preserving Maximum Common Subgraph. For the 2-MCIS problem (Algorithm 4), we used an implementation from Kriege [73] and extended it to allow split isomorphisms [27]. The result from Subsection 5.2.2 for possibly negative weights was not fully implemented, such that the weights should be non-negative or of type forbidden. We further implemented Algorithm 5 to compute a BBP-MCISI. Here, we allow any input graphs. If the graphs are not outerplanar, the computation uses the clique approach from Subsection 5.4.3. The implementation of bioisosteres from Subsection 5.4.2 occurred in a branch by Kevin Nikiel and is developed anew by Augusto Martins from TU Wien to incorporate the recent changes we made to our software.

Enumeration. We implemented the enumeration for maximum weight matchings from Section 3.6 using Reduction 1). We may enumerate all maximum weight common subtree isomorphisms as in Subsection 4.6.2. We also implemented the enumeration of all BBP-MCISIs as in Section 5.5. For the BBP-MCISI problem, the weights should be non-negative since the computation relies on solving the 2-MCIS problem as above.

6.1.3 Available Commands and Output

Generally, the software accepts arbitrary labeled graphs in GML and FOG format. Depending on the input graphs (trees, outerplanar, not outerplanar), the software uses the appropriate algorithms. For simplicity, we refer to the BBP-MCISI algorithm when introducing the parameters. The software has three modes to perform the computations. In the following, we list the modes and their optional parameters and a list of mode-independent parameters.

Computing a single BBP-MCISI.
`-c <GraphFile> <i> <j>` : This computes a BBP-MCISI between the ith and jth graph in the file `GraphFile`. Numbering starts at 1. The output consists of the weight, the isomorphism, and a similarity score. The isomorphism ϕ is listed as a sequence. For example, an output of

$(1,3)(2,4)$ means $\phi(1) = 2$ and $\phi(2) = 4$. The numbers are the vertex numbers according to the input graph. In GML, these numbers may be arbitrary. In FOG, numbering starts at 1. The similarity coefficient is based on Table 4.3 and defaults to the first function therein.

Optional parameters are as follows.

`-e` : This parameter outputs *all* BBP-MCISIs. Each isomorphism is outputted in a separate line.

Comparing a pattern file against a database file.

`-i <PatternFile> <DatabaseFile>` : Computes the weight of an BBP-MCISI pairwise between all graphs from `PatternFile` and all graphs from `DatabaseFile`. The output is one line per BBP-MCISI ϕ and consists of position and label of pattern graph G, position and label of database graph H, $|E_G| + |V_G|$, $|E_H| + |V_H|$, $\mathcal{W}(\phi)$, and the similarity score. Attention should be paid that the pattern file is completely loaded and processed into the main memory. Therefore, it should be of reasonable size.

Optional parameters are as follows.

`-t <N>` : Number of computation threads. $N = 0$ for single thread (default); $N = k$ for k computation and one additional IO thread.

`-rq <N>` : Number of graphs N read ahead from reader thread for threaded computation. Default is 16.

`-cd <N>` : Cleanup delay for threaded computation. For each thread, OGDF garbage collection is called after N BBP-MCISI computations. Default is 4.

`-v` : Console output even if an output file is specified (see `-o` below).

Repeatedly comparing one graph to the following N graphs

`-oneToX <GraphFile> <N>` : Compares the first graph against the following N graphs, and then repeats this process for the entire file `GraphFile`. For example, let $N = 3$, then the comparison is 1st to 2nd, 1st to 3rd, 1st to 4th; 5th to 6th, ..., 5th to 8th; 9th to 10th, and so forth.

Optional parameters for each of the above modes

`-o <outputFile>` : Output into file `outputFile`. Warning: Previous content will be overwritten. The output format is the same as with the `-i` command. If the `-o` parameter is specified, there will be no console output unless parameter `-v` is passed (see above).

`-l <labelFile>` : Reads and processes the weight function from `labelFile` as described in Subsection 6.1.1.

`-s` : Silent / Summary only mode. Outputs only the total running time, no individual isomorphisms. Useful for benchmarking.

`-ct <T>` : Maximum time T in ms per clique computation for non-outerplanar graphs. If no proven maximum solution is found within that time bound, the largest previously found biconnected clique is used. This can produce non-maximum isomorphisms. Default is no timeout.

`-cto` : Append the number of clique computations exceeding the time limit in output. This might help to identify possible non-maximum isomorphisms.

`-sc <1--8>` : Determine the similarity function from Table 4.3. Default is the first.

`-p <D>` : Distance penalty $p = D$ (C++ type `double`) for skipping vertices of the LaWeCSE_u approach. Default is ∞, i.e., no skipped vertices allowed. This setting is incompatible with `-e`.

`-d` : Computes a layout of all read graphs and graphically displays the computed BBP-MCISIs. For details, see the following subsection.

6.1.4 Graphical Output

For graphical output, we use the SFML graphics framework.[4] This is optional and has to be enabled before compilation. To do this, remove the comment in line 7 of the file `include/global.h`, i.e., replace `\\#define GRAPHICS` by `#define GRAPHICS`. Further, the SFML libraries and headers have to be added to the `CMakeLists.txt` file of the LaWeCSE C++ project. We recommend to keep separate executables and use graphical output only with the `-c` parameter. The reason is, the graphical output requires additional memory and computation time, e.g., to compute the layout on the screen. An example of the graphical output is depicted in Figure 6.4.

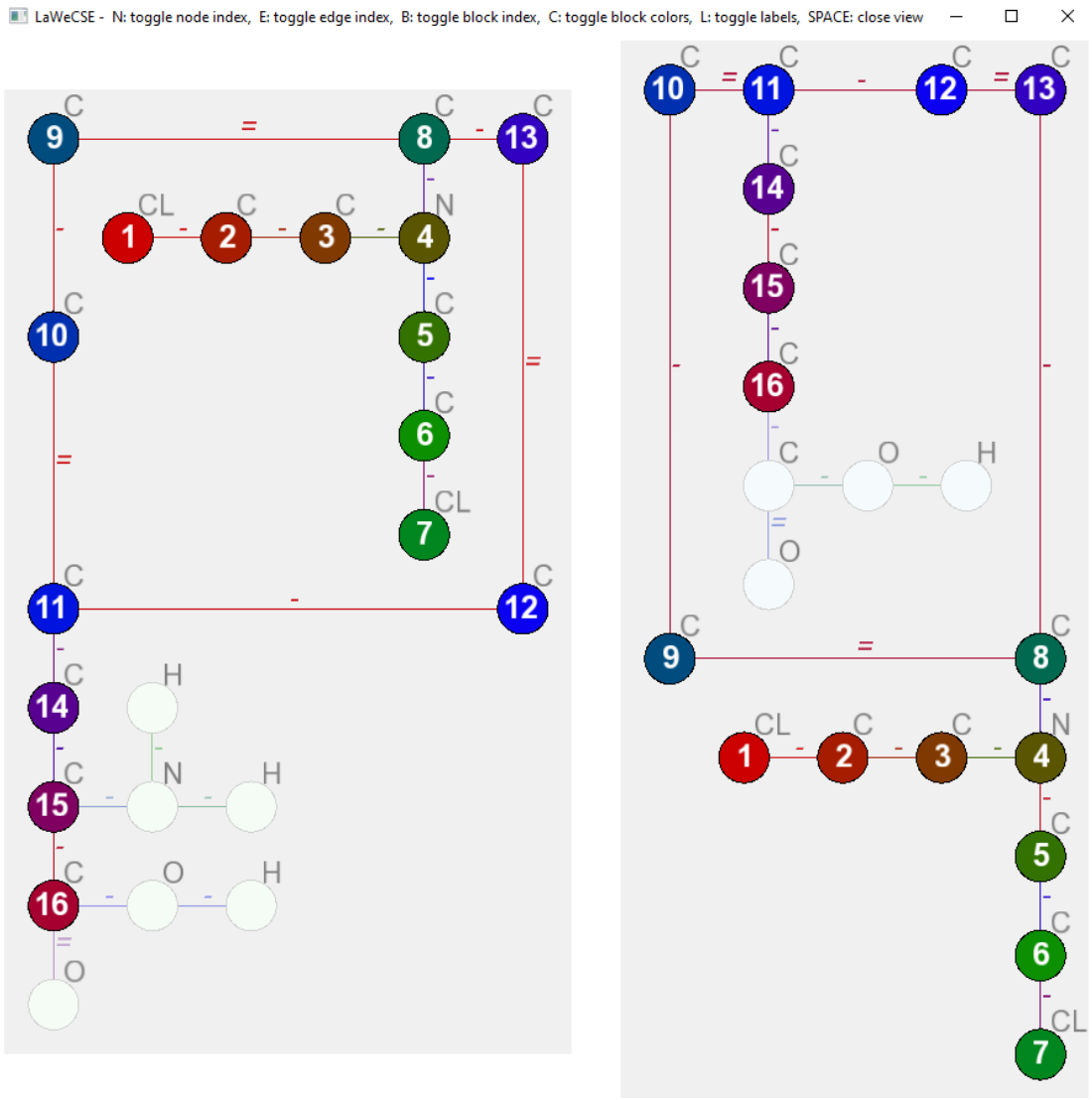


Figure 6.4: Two molecular graphs and one of their BBP-MCISs. Vertices with the same numbers and colors are mapped to each other.

[4] https://www.sfml-dev.org/ – version 2.4.2

Short Form	Specifications
3770	Intel Core i7 3770, 4 Cores, 3.4 GHz, 16 GB RAM, SATA-SSD, Ubuntu, gcc 4.8.4
Cluster	Intel Xeon E5 E5-2650 v3, 10 Cores, 2,3 GHz, Linux

Table 6.1: Hardware setups for the different test runs.

The graphical output automatically adjusts to the window size. While the window is focused, the following key presses are recognized.
`n` : Display the internal vertex index (default: off).
`e` : Display the internal edge index (default: off).
`b` : Display the internal numbering of blocks and bridges next to the edges (default: off).
`c` : Uses separate/uniform colors for different blocks (default: separate).
`l` : Toggles the labels on or off (default: on).
`SPACE` : Display the next computed solution or close the view if it is the last. If used with the `-e` command, all BBP-MCISs are enumerated and displayed. The `SPACE`-key is used to cycle through the solutions.

The layout is computed by the planarization grid layout algorithm in OGDF. That algorithm is based on a publication from Gutwenger and Mutzel [52].

6.2 Test Setup

Some tests were performed on a desktop PC, others on a cluster setup. The specifications are listed in Table 6.1. In any case, main memory was sufficient for all test cases, and no other computational or memory demanding tasks were performed during the test runs. All tests were performed on a single core only unless stated otherwise. Running time tests were performed in silent mode (parameter `-s`). The standard value of 4 as cleanup delay (parameter `-cd`) has been chosen for threaded computation. Higher values mean fewer cleanup calls, but more memory has to be freed in each call. A value of 4 generally showed the best performance with only a small impact on the memory footprint. For single thread computations, a memory cleanup delay did not show any improvement to the running time. Note, the OGDF framework has its own memory manager and requires special care in a threaded environment.

6.3 Synthetic Tests [5]

In this section, we experimentally evaluate the running time of our implementation on synthetic instances. We compare it to the BBP-MCES software of [111]. Since there is no other software available to compute a BBP-MCIS on outerplanar graphs directly, we chose their software. Further, their algorithm was shown to be much faster than clique based MCS approaches. The implementation was provided by the authors as part of the FOG package and implemented in C++.[6] With trees as inputs, their software solves the MCSI problem. Both their and our software

[5] The results in this section are based on our findings in *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 7, and *Finding Largest Common Substructures of Molecules in Quadratic Time* [27], Section 5.
[6] https://dtai.cs.kuleuven.be/software/PMCSFG

Order	Alg. 3	FOG	q
20	$0.9 \pm 8\%$	$40 \pm 7\%$	44.1
40	$3.5 \pm 6\%$	$221 \pm 5\%$	62.7
80	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
160	$58.9 \pm 3\%$	$8\,342 \pm 5\%$	141.7
320	$237.4 \pm 2\%$	$63\,327 \pm 8\%$	266.9

(a) Random trees of the same order

$\lvert H\rvert$	Alg. 3	FOG	q
20	$3.6 \pm 8\%$	$192 \pm 4\%$	53.6
40	$7.3 \pm 7\%$	$504 \pm 4\%$	68.7
80	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
160	$30 \pm 9\%$	$3080 \pm 4\%$	103.3
320	$59.5 \pm 3\%$	$6842 \pm 4\%$	114.9

(b) Random trees with $\lvert G\rvert = 80$ fixed

Order	Alg. 3	FOG	q
10	0.1	18	117.6
20	1	489	458.5
40	8.9	18 722	2109.9
80	77.5	929 784	11 992.1

(c) Star graphs

#labels	Alg. 3	FOG	q
1	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
2	$5.4 \pm 8\%$	$217 \pm 8\%$	40
3	$3.3 \pm 7\%$	$118 \pm 12\%$	36.1
4	$2.6 \pm 8\%$	$83 \pm 9\%$	31.9

(d) Multiple labels, order 80

Table 6.2: Average running time in ms $\pm$ RSD in % and speedup factor q = FOG/Alg. 3.

were compiled and run on the same machine, cf. Table 6.1, 3770. Running times were measured using a single core only. We are interested in how far the theoretical predicted upper time bound equals the measured times.

In a test, we generated random trees as follows. We iteratively added a new vertex and connected it to a random existing vertex starting with a single vertex. We averaged over 40 to 100 pairs of instances, depending on their size. The weight function ω was set to 1 for each pair of vertices and edges, i.e., we computed an isomorphism of maximum size. We also conducted tests with different labels, where ω was set to $-\infty$ for different labels. This matches the setting in FOG. Both algorithms received the same random trees as input.

Table 6.2 summarizes our results. We observe that the running time of Algorithm 3 aligns with our theoretical analysis. Whenever we double the number of vertices of both input trees, the running time increases about four times, cf. Table 6.2a. If we double the size of one tree only, the running time increases about two times, cf. Table 6.2b. In comparison, FOG's running time is much higher and increases to a larger extent with the input size. Table 6.2c shows the running time for star graphs, which are worst-case examples since they require to solve MWM problems in the same order as the star graphs. Our theoretical proven cubic running time matches the experimental results, while FOG's running time increases drastically and suggests a running time of $\Omega(n^5)$ on star graphs. Table 6.2d summarizes the computation time for multiple labels. Both algorithms benefit from the fact that fewer MWMs have to be computed. The running times of both algorithms show a low standard deviation throughout all inputs. The results of Table 6.2a are also shown in Figure 6.5.

We further evaluated the running time on outerplanar graphs. Specifically, we are interested in how the running time is affected by specific properties of the input graphs. We compared our Algorithm 5 and FOG [111] on randomly generated connected outerplanar graphs to evaluate this. Our graph generator takes several parameters as input. With them, we evaluated three different properties: the order of the graph, the average ratio $|E|/|V|$ of edges to vertices, and the average order of the blocks. For any outerplanar graph, the ratio of edges to vertices is less than 2. While evaluating the effect of one property, we preserved the other two. This procedure allows verifying whether our theoretical findings are consistent with the running times observed in practice. We set the weight function ω to 1 for each pair of vertices and edges, which corresponds to uniformly labeled graphs and matches the only possible FOG setting.

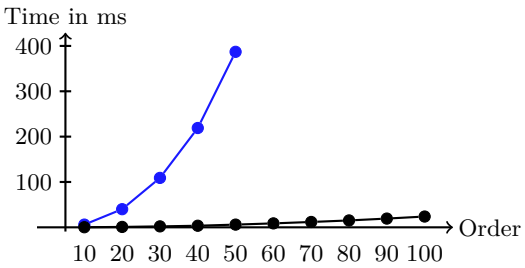


Figure 6.5: Average running time in ms (y-axis) for MCSI computation on random trees of order n (x-axis). Black = Algorithm 3. Blue = FOG implementation.

Order	10	20	40	80	160
MCIS	0.7 ± 0.3 ms	2.3 ± 0.8 ms	8.2 ± 1.6 ms	33.5 ± 3.6 ms	133.2 ± 10.1 ms
MCES	207 ± 118 ms	3.4 ± 6.0 s	38.6 ± 90.6 s	234.2 ± 420.9 s	timeout
$\lvert E\rvert/\lvert V\rvert$	0.98	1.10	1.24	1.46	1.78
MCIS	3.8 ± 0.3 ms	4.0 ± 1.1 ms	8.2 ± 1.6 ms	30.8 ± 4.0 ms	110.3 ± 11.6 ms
MCES	223 ± 16 ms	2.2 ± 2.6 s	38.6 ± 90.6 s	111.0 ± 213.8 s	216.1 ± 288.3 s
BO	3	5	10	20	40
MCIS	27 ± 6.4 ms	13.3 ± 2.4 ms	8.4 ± 1.7 ms	5.5 ± 1.4 ms	4.5 ± 0.9 ms
MCES	132 ± 14 ms	689 ± 548 ms	83.7 ± 118.7 s	30.4 ± 27.8 min	timeout

Table 6.3: Average time ± SD over 100 BBP-MCS computations on random outerplanar graphs, varying one property (graph order, the ratio of edges to vertices $|E|/|V|$, block order BO). Note the units of measurement; timeout—total time exceeds three days.

We first varied the order of the input graphs, while preserving an average ratio of edges to vertices of 1.24 and an average block size of 8. Based on Theorem 5.8, we expected the average time to increase by a factor of a bit more than 4 if we double the order. This follows from the increase in the number of cutvertices (about doubling per graph), while the degree of these vertices only marginally grows. The time to compute the blocks is expected to be dominated by the matching problems. The results in Table 6.3 closely match this expectation.

Next, we evaluated different ratios of edges to vertices. The number of vertices was set to 40 and for blocks on average 8. A higher ratio results in a higher number of faces in the blocks and consequently affects the time required by Algorithm 4. In particular, the table size and, thus, the running time are expected to show a quadratic growth. The increase in running time exceeds our expectations. This might be explained by the increasing size of the data structure used to represent the faces of the blocks.

Finally, we evaluated different average block orders. The graphs had 40 vertices, and the average ratio of edges to vertices was 1.24. Larger blocks mean fewer MWMs to compute, which are the most costly part of the BBP-MCS computation. Therefore, we expected the running time to decrease. The results in Table 6.3 support this. The exact amount of the decrease was hard to predict since that depends on the practical running times for the block versus the matching computations.

6.4 Evaluation on Molecular Graphs

In this section, we evaluate our algorithm on different chemical databases. We first compare our approach against FOG on a database containing molecular graphs in their BC-tree representations. Then we compare the algorithms on outerplanar graphs. Besides measuring the time, we also compare how often the computed common subgraphs differ in size. Finally, we evaluate the computed similarity values w.r.t. state of the art fingerprint-based approaches to verify whether our BBP-MCIS algorithm produces meaningful results. This includes checking the usefulness of skipping vertices from the LaWeCSE_u approach.

6.4.1 BC-Trees Representing Molecular Graphs [7]

From the NCI Open Database[8], a chemical database of thousands of molecules, we extracted 100 pairs of graphs with BC-trees consisting of more than 40 vertices. The running time to compute the MWMs is the dominating factor for our and other MCS algorithms' total running time that operate on outerplanar or series-parallel graphs like in [1, 71, 111]. Again, we used the same hardware and software, cf. Table 6.1, 3770. The average running time of our Algorithm 3 was 11.2 ms, compared to FOG's 481.3 ms. The speedup factor ranges from 24 to 59, with an average of 43. This indicates that the approaches mentioned above could greatly benefit from the techniques presented in Section 3.5 and Section 4.3.

6.4.2 Outerplanar Molecular Graphs [9]

In this subsection, we focus on a comparison between the BBP-MCIS and the BBP-MCES. We are interested in answering the following questions:

(Q1) To what extent differs the BBP-MCIS from the BBP-MCES on molecular graphs?

(Q2) How large is the difference in terms of running time on molecular graphs?

To answer **(Q1)** and **(Q2)**, we extracted 29 000 randomly chosen pairs of outerplanar molecular graphs from the NCI Open Database. The molecules in the database contain up to 104 vertices and 22 vertices on average. The weight function ω was set to 1 for identical vertex and edge labels and $-\infty$ otherwise. This matches the setting in FOG [111].

(Q1) While comparing the weight of the isomorphisms computed by the two algorithms, we observed a difference for only 0.40% of the 29 000 tested molecule pairs. This suggests that a BBP-MCIS yields a valid notion of similarity for molecular graphs, as shown for the BBP-MCES [111].

(Q2) Our algorithm computed a maximum solution 84 times faster on average. The dots in Figure 6.6 represent the computation times of the two algorithms. The results are summarized in Table 6.4. Schietgat et al. [111] compared their BBP-MCES algorithm to a state of the art algorithm for general MCIS. Their algorithm had similar computation times on small graphs and was much faster on large graphs. The maximum time of the general MCIS algorithm was more than 24 hours.[10] In contrast, our computation time never exceeded 41 ms. This clearly indicates that our algorithm is orders of magnitude faster than the general approach and much faster than the BBP-MCES algorithm (>80 on average; >25 on median).

[7] Based on *Faster Algorithms for the Maximum Common Subtree Isomorphism Problem* [26], Section 7.
[8] NCI Open Database, GI50, `http://cactus.nci.nih.gov`
[9] Based on *Finding Largest Common Substructures of Molecules in Quadratic Time* [27], Section 5.
[10] Their results are based on the NCI60 data set http://dtp.nci.nih.gov/docs/cancer/cancer_data.html [111] with an average of 23 vertices.

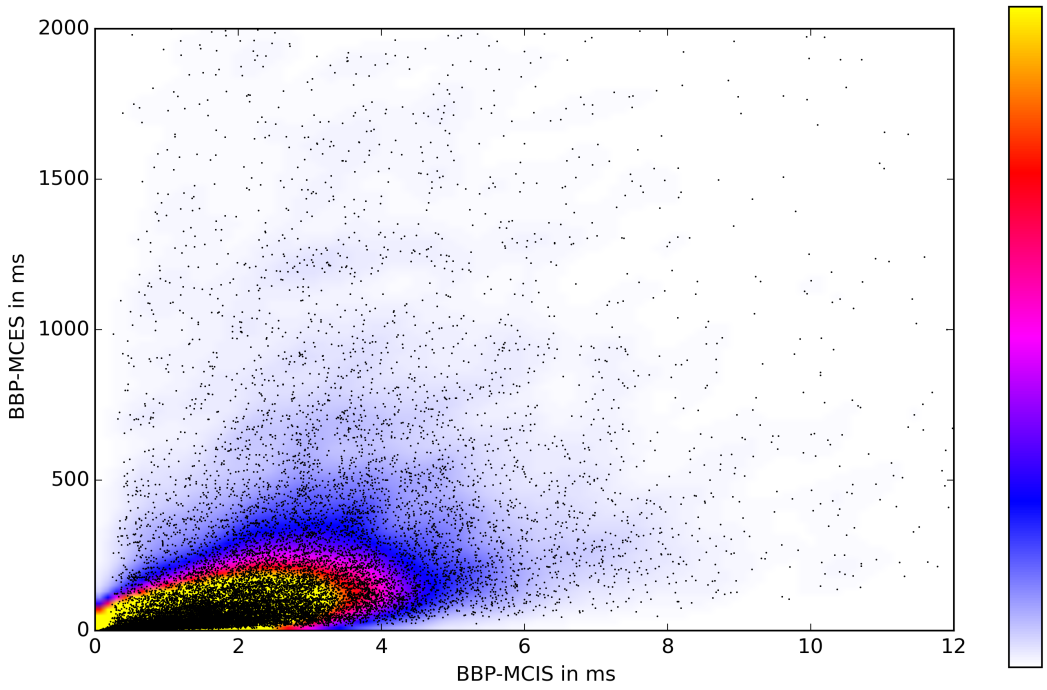


Figure 6.6: Running times in ms for 28 399 BBP-MCS computations. Each black dot represents a BBP-MCS computation on two randomly chosen outerplanar molecular graphs. Colors represent the density of black dots in that area. It directly compares the running time of our algorithm (MCIS, x-axis) and the implementation by Schietgat et al. (MCES, y-axis). Note the different scaling on the axes. The running times of another 601 BBP-MCS computations did not fit into the borders. The distribution of those points extends similar to the visible area.

Algorithm	Average time	Median time	95% less than	Maximum time
MCIS	1.97 ms	1.51 ms	5.28 ms	40.35 ms
MCES	207.08 ms	41.43 ms	871.48 ms	26 353.68 ms
Comparison	**Average factor**	**Median factor**	**Minimum factor**	**Maximum factor**
MCES / MCIS	83.8	25.6	1.8	28912.5

Table 6.4: Upper half: Running times for our implementation (MCIS) and the implementation by Schietgat et al. (MCES). Lower half: Relative differences in computation times. These values are based on the quotients of the individual computation times between the algorithms. For example, the average factor is the average of the 29 000 quotients. Thus, each computation has the same relative impact.

6.4.3 BBP-MCIS as Similarity Coefficient in Cheminformatics

In the previous subsection, we compared the BBP-MCIS to the edge induced variant. This subsection evaluates the quality of the computed similarity coefficients from the BBP-MCIS algorithm to state of the art fingerprint based approaches.[11] To this end, we conducted tests analog to the approach described by O'Boyle and Sayle [97]. There, the authors created two benchmarks: the single-assay and the multi-assay benchmark. The former one consists of rather similar molecules, while the latter one consists of less similar molecules. There are several series of one reference molecule and four other molecules in those benchmarks, ordered concerning their similarity as described in the following paragraphs *Multi-Assay benchmark* and *Single-Assay benchmark*. For both cases, the molecules were randomly chosen (under certain restrictions, cf. [97] for more details). Further, they generated 1 000 repetitions of the data set to assess statistical significance. Each of these repetitions consists of at least 4 563 series of 5 molecules. They used the Spearman correlation to compare the ranking from the fingerprints to the order (reference ranking) in the benchmarks. The main questions of this subsection are:

(QM) Does the BBP-MCIS algorithm produce meaningful results when it is used to rank molecules? How does it compare to fingerprints, and what are the conditions for good results?

To answer **(QM)** we evaluated the quality of the BBP-MCIS algorithm concerning the following questions:

(Q1) Can the BBP-MCIS algorithm discern similar molecules better than a fingerprint based approach?

(Q2) Does the choice of a proper weight function improve the results?

(Q3) Which representations for molecules exist and which produce the best results?

(Q4) To what extent influences the similarity coefficient (cf. Table 4.3) the quality of the result?

(Q5) Can we improve the results with our LaWeCSE_u approach of skipping vertices?

Before answering **(Q1)** to **(Q5)**, we briefly present the benchmarks described by O'Boyle and Sayle and give an overview of the test setup.

Multi-Assay benchmark. The four molecules are ordered by a similarity based on medicinal chemistry programs. More specific, "molecules A and B are similar if a medicinal chemist would be likely to synthesise and test them around the same time as part of the same medicinal chemistry program." [97]. Then, if molecule B is also considered in another medicinal chemistry program with another molecule C, but not A, then C is considered less similar to A than B to A. For this benchmark, the series are generated from such chains of 4 papers, which results in the reference molecule and the four other molecules of decreasing similarity as mentioned above.

Single-Assay benchmark. Here, each set of 5 molecules is taken from the same ChEMBL [6] assay. The reference molecule was chosen as the most active, and the others were ordered by decreasing order of activity.

[11] Lina Humbeck conducted a comprehensive evaluation [60]. Here, we only present the key results.

Test setup. The hardware for all computations is listed in Table 6.1, Cluster. Since not all molecular graphs in our data sets were outerplanar, we used a timeout of 2 seconds (parameter `-ct 2000`) for calculations on non-outerplanar graphs. A pretest with a setting of 1 instead of 2 seconds yielded no difference in the quality. However, without a timeout, sometimes the computation stalled for a long time due to single difficult MCS computations between non-outerplanar blocks.

(Q1) To evaluate the discrimination capabilities of the BBP-MCIS algorithm, we decreasingly ordered the molecules in the ChEMBL database to a reference molecule, *atenolol*, based on the computed similarity coefficient. Among the 1 000 highest rated molecules, the BBP-MCIS approach computed 431 different coefficients. The ECFP4-TC yielded only 239 to 255 (based on the length of the fingerprint) different values. This suggests that the BBP-MCIS approach can better discern different molecules by their similarity coefficient, especially if there are only minor differences between them.

(Q2) Since our algorithm accepts arbitrary weights between the vertices, we evaluated two different weight functions (label files; parameter `-l`). We used them in combination with the fconv_groups (i.e., *fconv_groups*, *ph4*, and *ph4_noH*; see **(Q3)**). These files define the similarity between the atom groups. The first label file was for single fconv_groups, the second for combinations of groups. The second showed better results than the first, and that one better than simple labels. Therefore, all further tests using fconv_groups labels were conducted using the second file. The exact values are listed in Table III.2.3 of [60].

(Q3) We tested different layers to represent the molecules.

1. elements: the chemical elements, like N for nitrogen or H for hydrogen.
2. fconv: file conversion atom types, cf. [95]. There are 139 in total.
3. fconv_groups: file conversion atom *group* types, cf. [95]. There are 8 groups in total. However, atoms may be assigned to more than one group. Therefore, we have 10 additional groups, which represent at least two regular groups. Some other atoms are associated with no group and keep their fconv atom type.
4. ph4: as fconv_groups, but neighboring atoms of the same fconv group are merged into a single vertex.
5. ph4_noH: as ph4, but hydrogen atoms are omitted.

On the single-assay benchmark, the results on the fconv_groups representation were superior to the results from the other four representations. However, on the multi-assay benchmark, the element representation performed best, with fconv_groups as the second-best representation.

(Q4) The similarity coefficient by Bunke and Shearer (1998) (2nd coefficient in Table 4.3) provided the best results on the single-assay benchmark. This is independent of the layer of representation. Contrary to that, the 1st, 4th, and 6th similarity coefficient in Table 4.3 showed the best results on the multi-assay benchmark. All three coefficients perform about equally well. Closely following is the 2nd coefficient.

(Q5) We evaluated the LaWeCSE_u approach to skip vertices (atoms). Here, we considered different penalties $p \in \{0, 1, 2, 3, 5, 8, 13\}$ for skipping the atoms. Allowing to skip vertices showed to be beneficial for most combinations of tested penalties p and representations. It is generally better to allow skipping vertices than not to allow it. On the single-assay benchmark, a value of $p = 3$ showed the best results. This is independent of the molecule representation. On the multi-assay benchmark using the element representation, a penalty $p = 8$ showed the best results. However, for other representations, penalties between 1 and 8 showed the best results.

Ranking Results. Figure 6.7 depicts condensed ranking results on the single-assay benchmark. Here, we show only our best results, firstly of the BBP-MCIS algorithms and secondly the improved results by allowing to skip vertices (BBP embedding). The full results on the single-assay and multi-assay benchmark are depicted in Figure 6.8, Figure 6.9, and Figure 6.10. For all those figures, we used the similarity coefficient by Bunke and Shearer (1998), cf. Table 4.3. Further, the molecule representations are denoted as above, where a possibly following number shows the penalty. For example, *fconv_groups3* is the file conversion atom type group representation with a penalty $p = 3$. The numbers between any two ellipses A and B in the figures show how often the Spearman correlation coefficient of A was higher than that of B (net difference) compared to the reference ranking. The compared fingerprints are listed in [97]. The exact results for all the other tests are presented in [60].

(QM) Our BBP-MCIS implementation shows similar performance to other well known fingerprint methods in the tested benchmarks. It further shows higher discriminating capabilities. An advantage is the fact that our approach computes the similar parts of the molecules and a concrete mapping between those atoms (vertices). The additional feature of skipping vertices showed improved results on the ranking benchmarks. Therefore, we suggest using this feature with a penalty of 3 on typical molecular databases, e.g., the ZINC [119] or ChEMBL database. We also discovered that the molecules' representation has a considerable impact on the quality of the computed similarity coefficients. If the molecules are very similar, we suggest using the fconv_groups representation. For less similar molecules, we suggest using the elements representation.

For the BBP-MCIS algorithm, the other representations (fconv, ph4, ph4_noH) cannot be recommended with the tested settings. Providing an appropriate weight function for the fconv representation should yield better results. However, since there are 139 atom types, defining such a function and coding it into a file is no trivial task. For the ph4 and ph4_noH representations, we did not respect the number of atoms merged into a single vertex. In doing this, we expect improved ranking results.

6.5 Conclusion

In this chapter, we first provided details of our implementation in C++, the command line parameters, and the test setup. We then evaluated our software by conducting synthetic tests to evaluate the theoretically predicted run times. We observed that the predictions closely match the practical running times. We further compared the algorithms to state of the art algorithms on real-world data. We showed that the BBP-MCIS and BBP-MCES approaches compute similar results, while our BBP-MCIS implementation is much faster than the BBP-MCES implementation from Schietgat, Ramon, and Bruynooghe [111]. Experiments on a ranking benchmark revealed that our algorithm's ranking quality is comparable to state of the art fingerprint based approaches. This, however, depends on the choice of the similarity coefficient, the molecule representation, and the distance penalty.

Open Problem 6.1. *Can we obtain better ranking results by incorporating the number of atoms merged into a single vertex in the ph4 and ph4_noH representations?*

Open Problem 6.2. *Can we improve the results by specifying a meaningful weight function for the fconv representation?*

We did not include the Bio-BBP-MCISI approach from Subsection 5.4.2 in our experiments. This is scheduled for testing and publication once the implementation by Augusto Martins is finished.

Open Problem 6.3. *Can we obtain better ranking results by using the Bio-BBP-MCISI approach from Subsection 5.4.2?*

We used the OGDF library with its capabilities to store the graphs and aid us in the computation. However, a low-level optimization might speed up the running time by a constant factor.

Open Problem 6.4. *Can we improve the actual running time by using a low-level approach for current PC hardware?*

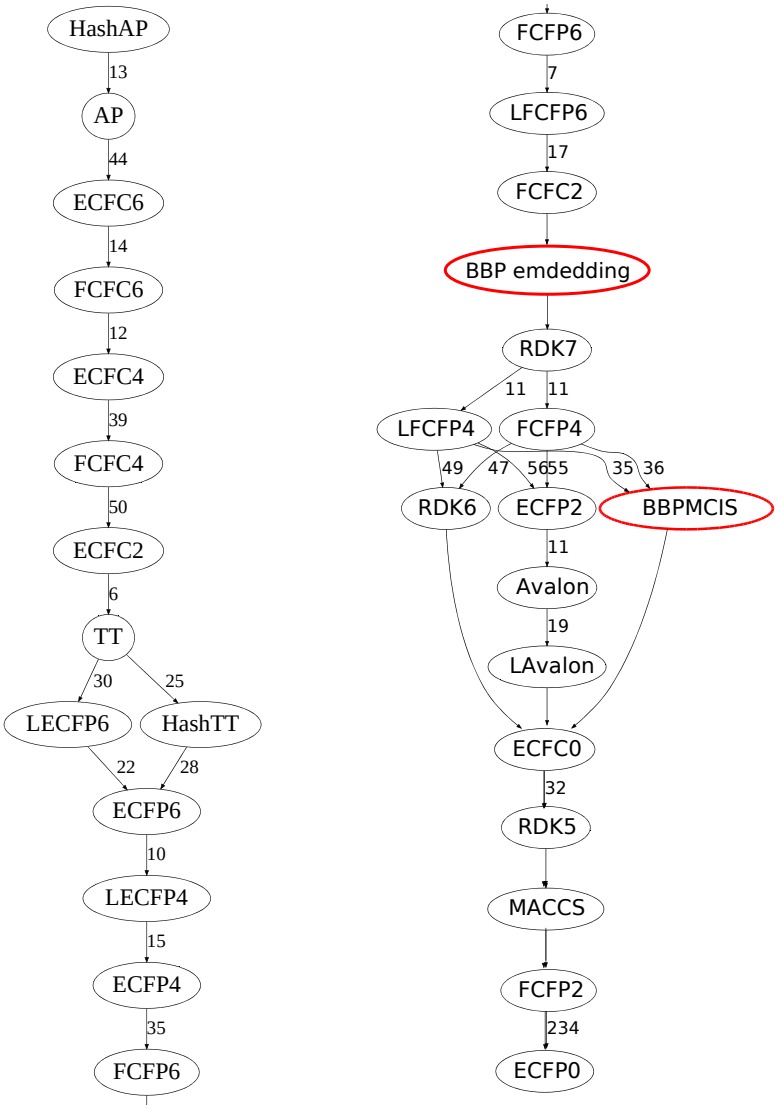


Figure 6.7: Ranking results on the single-assay data set. The ranking continues from the bottom left to the top right. Higher up means better comparative performance. We achieved the best results using the fconv groups representation. When allowing to skip vertices, a penalty of 3 showed the greatest increase in ranking performance (BBP embedding). Similarity coefficient by Bunke and Shearer (1998), cf. Table 4.3.

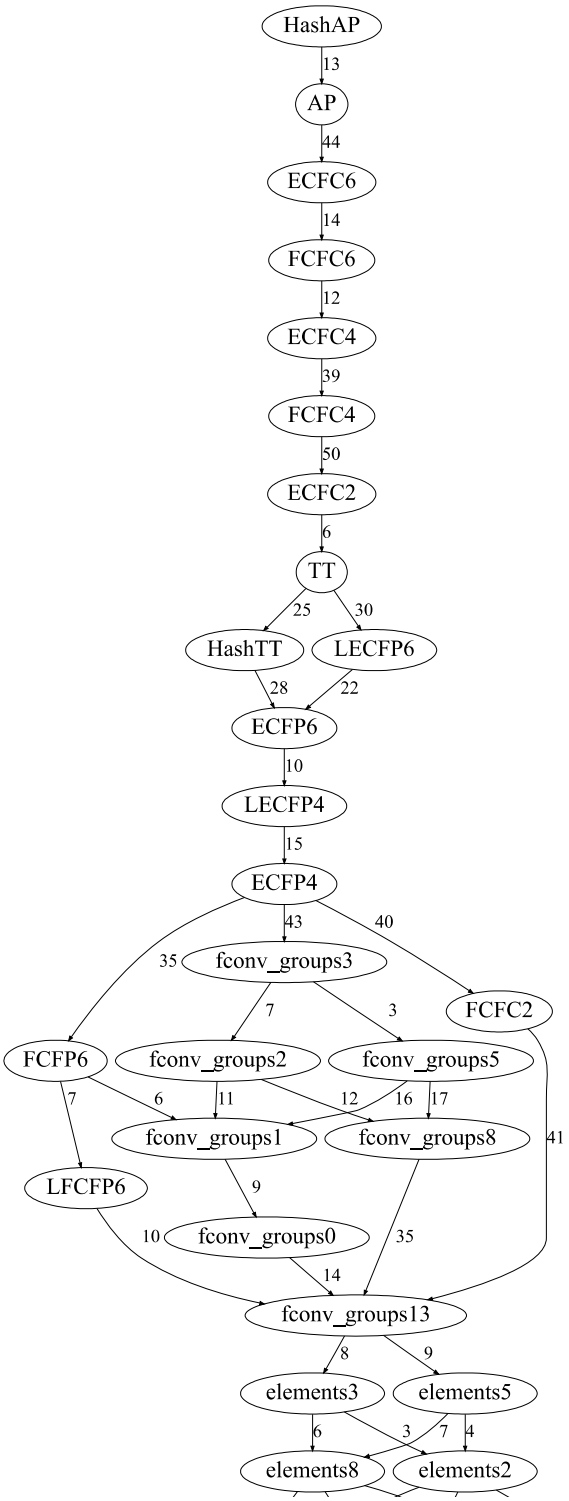


Figure 6.8: Ranking results on the single-assay data set. Higher up means better comparative performance. This is the top half. The bottom half is in the next figure.

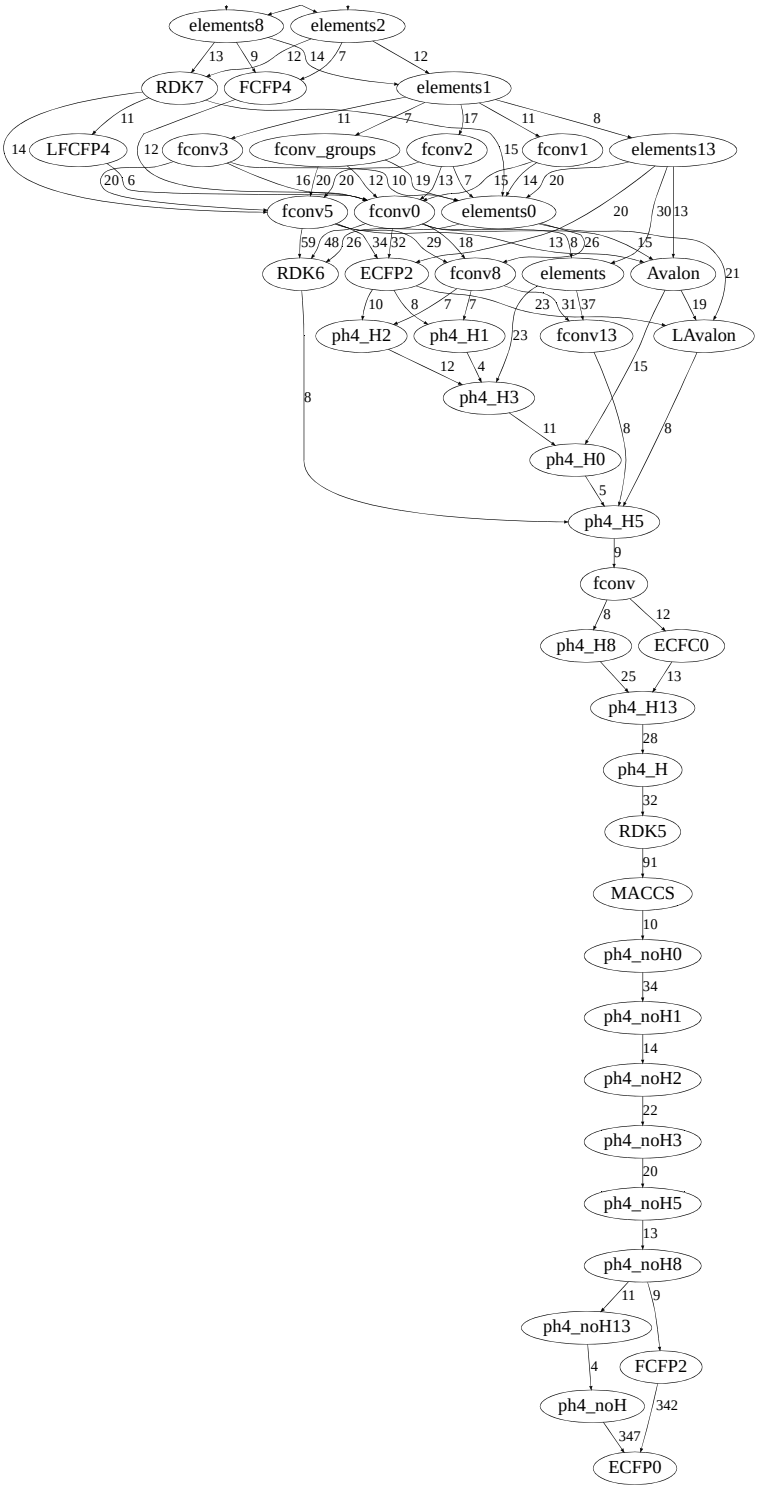


Figure 6.9: Ranking results on the single-assay data set. Higher up means better comparative performance. This is the bottom half. The top half is in the previous figure.

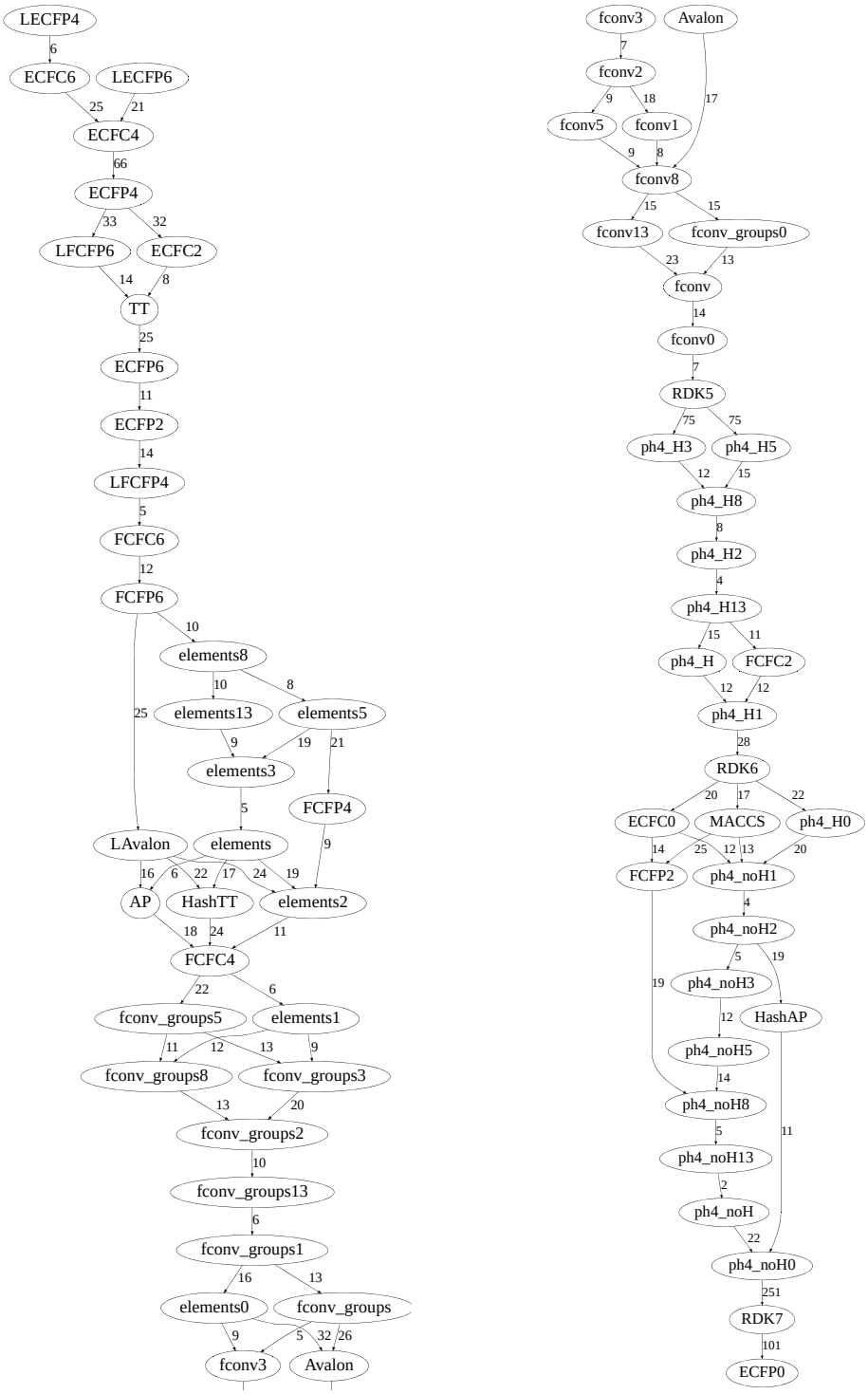


Figure 6.10: Ranking results on the multi-assay data set. The ranking starts top left, continues to the bottom left, then top right, to bottom right.

Was wir mathematisch festlegen, ist nur zum kleinen Teil ein objektives Faktum, zum größeren Teil eine Übersicht über Möglichkeiten.

Werner Karl Heisenberg
1901 – 1976

Chapter 7

Conclusion and Outlook

Determining the similarity between molecules is a fundamental task in drug design. For this problem, numerous chemical fingerprints exist, which allow fast algorithms to compare the molecules. Due to their nature as fixed-size vectors, the established fingerprints lack interpretability and have reduced discrimination capabilities compared to a maximum common subgraph approach, from which we can also obtain a similarity coefficient. This has been shown to produce meaningful results. Contrary to fingerprint algorithms, the maximum common subgraph problem is NP-hard and has a considerably higher running time in practice. As a middle ground between these two approaches, the block-and-bridge preserving constraint for the maximum common subgraph problem has been suggested. This allows a polynomial running time on outerplanar molecular graphs and also produces meaningful results. However, the previously proposed block-and-bridge preserving maximum common edge subgraph algorithm has certain limitations, e.g., no support of weights or only allowing connected solutions.

We approached these limitations and developed a novel block-and-bridge preserving maximum common induced subgraph algorithm tailored to molecular graphs. Our new algorithm allows disconnected solutions by mapping the endpoints of disjoint paths of bridges to each other. Experiments have shown that this increases the quality of the computed similarity coefficients. We also allow arbitrary weights between the vertices and edges in the molecular representation. This weighted approach proved to be superior to a labeled maximum common subgraph. Generally, we improved the running time in theory and practice. Our algorithm relies on efficiently solving the maximum weight common subtree isomorphism problem and the all-cavity maximum weight matching problem. For both subtasks, we developed efficient algorithms that match or outperform previous results. With a newly designed algorithm to compute a biconnected maximum common induced subgraph between non-outerplanar blocks, we can compute a solution for any two molecular graphs. Promising future work lies in improving the running time on non-outerplanar graphs and considering bioactivity by integrating bioisostere mapping into the algorithm. A more efficient implementation that fully utilizes modern computer hardware should further reduce the practical running time.

Another contribution of this thesis are polynomial delay algorithms for various of our proposed algorithms. For the maximum weight matching problem, we proposed the first algorithm realizing this. In the context of molecular graphs, we can provide all block-and-bridge preserving maximum common induced subgraphs. This assists in finding the most relevant common substructure

instead of providing only one solution. However, the running time to compute an initial solution decreases in this process, and further research is necessary to reach the running time of the case where only one optimal solution is sought.

By implementing our algorithms in publicly available software, we aim to provide new practical methods in the field of rational drug design. The optional graphical output allows to visualize the common substructures and, therefore, can be more easily interpreted by chemists. Currently, the software is command-line only. Implementing a user interface and providing a visual representation tailored to molecular graphs that clearly illustrates the common substructures, e.g., overlapping them in a 3D visualization, should further improve the accessibility of the software.

Bibliography

[1] T. Akutsu and T. Tamura. "A Polynomial-Time Algorithm for Computing the Maximum Common Connected Edge Subgraph of Outerplanar Graphs of Bounded Degree". In: *Algorithms* 6.1 (2013), pp. 119–135. DOI: 10.3390/a6010119 (cit. on p. 112).

[2] L. Babai, P. Erdös, and S. M. Selkow. "Random Graph Isomorphism". In: *SIAM J. Comput.* 9.3 (1980), pp. 628–635. DOI: 10.1137/0209047 (cit. on p. 47).

[3] M. L. Balinski. "Labelling to obtain a maximum matching". In: *Combinatorial Mathematics and Its Applications (Proceedings Conference Chapel Hill, North Carolina.* 1967, pp. 585–602 (cit. on p. 18).

[4] R. Bellman. "On a Routing Problem". In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90 (cit. on pp. 28, 36).

[5] J. L. Bentley. "Algorithm Design Techniques". In: *Commun. ACM* 27.9 (1984), pp. 865–871 (cit. on p. 83).

[6] A. P. Bento, A. Gaulton, A. Hersey, L. J. Bellis, J. Chambers, M. Davies, F. A. Krüger, Y. Light, L. Mak, S. McGlinchey, M. Nowotka, G. Papadatos, R. Santos, and J. P. Overington. "The ChEMBL bioactivity database: an update". In: *Nucleic Acids Res.* 42.Database-Issue (2014), pp. 1083–1090. DOI: 10.1093/nar/gkt1031 (cit. on pp. 1, 114).

[7] C. Berge. "Two Theorems in Graph Theory". In: *Proceedings of the National Academy of Sciences* 43.9 (1957), pp. 842–844. DOI: 10.1073/pnas.43.9.842. eprint: https://www.pnas.org/content/43/9/842.full.pdf (cit. on p. 16).

[8] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. "Time Bounds for Selection". In: *J. Comput. Syst. Sci.* 7.4 (1973), pp. 448–461. DOI: 10.1016/S0022-0000(73)80033-9 (cit. on p. 27).

[9] Boost. *Boost C++ Libraries (version 1.55).* https://www.boost.org/doc/libs/1_55_0/. Accessed on 08.08.2019. Nov. 2013 (cit. on p. 18).

[10] C. Bron and J. Kerbosch. "Finding All Cliques of an Undirected Graph (Algorithm 457)". In: *Commun. ACM* 16.9 (1973), pp. 575–576 (cit. on pp. 12, 94 sq.).

[11] H. Bunke and K. Shearer. "A graph distance metric based on the maximal common subgraph". In: *Pattern Recognit. Lett.* 19.3-4 (1998), pp. 255–259. DOI: 10.1016/S0167-8655(97)00179-7 (cit. on pp. 72, 115 sq., 118).

[12] F. Cazals and C. Karande. "An algorithm for reporting maximal c-cliques". In: *Theor. Comput. Sci.* 349.3 (2005), pp. 484–490. DOI: 10.1016/j.tcs.2005.09.038 (cit. on pp. 5, 48, 94 sqq.).

[13] L. Chang, X. Lin, W. Zhang, J. X. Yu, Y. Zhang, and L. Qin. "Optimal Enumeration: Efficient Top-k Tree Matching". In: *PVLDB* 8.5 (2015), pp. 533–544. DOI: 10.14778/2735479.2735486 (cit. on p. 37).

[14] J. Cheriyan and K. Mehlhorn. "Algorithms for Dense Graphs and Networks on the Random Access Computer". In: *Algorithmica* 15.6 (1996), pp. 521–549. DOI: 10.1007/BF01940880 (cit. on p. 20).

[15] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. "The Open Graph Drawing Framework (OGDF)". In: *Handbook on Graph Drawing and Visualization*. Ed. by R. Tamassia. Chapman and Hall/CRC, 2013, pp. 543–569 (cit. on pp. 5, 103).

[16] Y. Chu, B. Liu, S. Cai, C. Luo, and H. You. "An efficient local search algorithm for solving maximum edge weight clique problem in large graphs". In: *J. Comb. Optim.* 39.4 (2020), pp. 933–954. DOI: 10.1007/s10878-020-00529-9 (cit. on pp. 94, 96).

[17] M.-J. Chung. "$O(n^{2.5})$ Time Algorithms for the Subgraph Homeomorphism Problem on Trees". In: *J. Algorithms* 8.1 (1987), pp. 106–112. DOI: 10.1016/0196-6774(87)90030-7 (cit. on pp. 4, 27, 29).

[18] D. Conte, P. Foggia, C. Sansone, and M. Vento. "Thirty Years Of Graph Matching In Pattern Recognition". In: *Int. J. Pattern Recognit. Artif. Intell.* 18.3 (2004), pp. 265–298. DOI: 10.1142/S0218001404003228 (cit. on p. 3).

[19] S. A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. Ed. by M. A. Harrison, R. B. Banerji, and J. D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (cit. on p. 47).

[20] W. J. Cook and A. Rohe. "Computing Minimum-Weight Perfect Matchings". In: *INFORMS J. Comput.* 11.2 (1999), pp. 138–148. DOI: 10.1287/ijoc.11.2.138 (cit. on p. 19).

[21] A. Dawar and K. Khan. "Constructing Hard Examples for Graph Isomorphism". In: *J. Graph Algorithms Appl.* 23.2 (2019), pp. 293–316. DOI: 10.7155/jgaa.00492 (cit. on p. 47).

[22] B. Dezso, A. Jüttner, and P. Kovács. "LEMON - an Open Source C++ Graph Template Library". In: *Electron. Notes Theor. Comput. Sci.* 264.5 (2011), pp. 23–45. DOI: 10.1016/j.entcs.2011.06.003 (cit. on p. 18).

[23] R. Diestel. *Graph Theory: 5th edition*. Springer Graduate Texts in Mathematics. Springer-Verlag, © Reinhard Diestel, 2017 (cit. on p. 7).

[24] *DIMACS Challenge Graph Generators*. http://archive.dimacs.rutgers.edu/pub/netflow/generators/matching/. Accessed on 08.08.2019. July 1991 (cit. on p. 19).

[25] A. Droschinsky, B. Heinemann, N. M. Kriege, and P. Mutzel. "Enumeration of Maximum Common Subtree Isomorphisms with Polynomial-Delay". In: *Algorithms and Computation - 25th International Symposium, ISAAC 2014, Jeonju, Korea, December 15-17, 2014, Proceedings*. Ed. by H.-K. Ahn and C.-S. Shin. Vol. 8889. Lecture Notes in Computer Science. Springer, 2014, pp. 81–93. DOI: 10.1007/978-3-319-13075-0_7 (cit. on pp. 6, 37 sqq., 46).

[26] A. Droschinsky, N. M. Kriege, and P. Mutzel. "Faster Algorithms for the Maximum Common Subtree Isomorphism Problem". In: *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*. Ed. by P. Faliszewski, A. Muscholl, and R. Niedermeier. Vol. 58. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 33:1–33:14. DOI: 10.4230/LIPIcs.MFCS.2016.33 (cit. on pp. 2, 6, 24, 26, 29 sq., 46, 49 sq., 54 sqq., 58, 68, 70 sq., 99, 109, 112).

[27] A. Droschinsky, N. M. Kriege, and P. Mutzel. "Finding Largest Common Substructures of Molecules in Quadratic Time". In: *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017*. Ed. by B. Steffen, C. Baier, M. van den Brand, J. Eder, M. Hinchey, and T. Margaria. Vol. 10139. Lecture Notes in Computer Science. Springer, 2017, pp. 309–321. DOI: 10.1007/978-3-319-51963-0_24 (cit. on pp. 6, 45, 77 sq., 80, 85, 101, 106, 109, 112).

[28] A. Droschinsky, N. M. Kriege, and P. Mutzel. "Largest Weight Common Subtree Embeddings with Distance Penalties". In: *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*. Ed. by I. Potapov, P. G. Spirakis, and J. Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 54:1–54:15. DOI: 10.4230/LIPIcs.MFCS.2018.54 (cit. on pp. 6, 29 sqq., 45 sq., 49, 56 sq., 59 sq.).

[29] A. Droschinsky, P. Mutzel, and E. Thordsen. "Shrinking Trees not Blossoms: A Recursive Maximum Matching Approach". In: *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*. Ed. by G. E. Blelloch and I. Finocchi. SIAM, 2020, pp. 146–160. DOI: 10.1137/1.9781611976007.12 (cit. on pp. 6, 18 sq.).

[30] R. Duan, S. Pettie, and H.-H. Su. "Scaling Algorithms for Weighted Matching in General Graphs". In: *ACM Trans. Algorithms* 14.1 (2018), 8:1–8:35. DOI: 10.1145/3155301 (cit. on p. 20).

[31] R. Duan and H.-H. Su. "A scaling algorithm for maximum weight matching in bipartite graphs". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. Ed. by Y. Rabani. SIAM, 2012, pp. 1413–1424. DOI: 10.1137/1.9781611973099.111 (cit. on pp. 20 sq., 28 sq.).

[32] J. Edmonds. "Paths, trees, and flowers". In: *Canadian Journal of Mathematics* 17.3 (1965), pp. 449–467 (cit. on p. 18).

[33] H.-C. Ehrlich and M. Rarey. "Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1.1 (2011), pp. 68–79. DOI: 10.1002/wcms.5 (cit. on pp. 2, 5, 45, 71, 77 sq.).

[34] D. Ellis, J. Furner-Hines, and P. Willett. "Measuring the degree of similarity between objects in text retrieval systems". In: *Perspectives in Information Management-Annual Review*- 3 (1993), pp. 128–128 (cit. on p. 72).

[35] S. Even and O. Kariv. "An $\mathcal{O}(n^{2.5})$ Algorithm for Maximum Matching in General Graphs". In: *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*. SFCS '75. IEEE Computer Society, 1975, pp. 100–112. DOI: 10.1109/SFCS.1975.5 (cit. on p. 18).

[36] T. Feder and R. Motwani. "Clique Partitions, Graph Compression, and Speeding-Up Algorithms". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by C. Koutsougeras and J. S. Vitter. ACM, 1991, pp. 123–133. DOI: 10.1145/103418.103424 (cit. on pp. 16, 20).

[37] L. R. Ford. *Network Flow Theory.* RAND Corporation, 1956 (cit. on pp. 28, 36).

[38] M. L. Fredman and R. E. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *J. ACM* 34.3 (1987), pp. 596–615. DOI: 10.1145/28869.28874 (cit. on p. 20).

[39] K. Fukuda and T. Matsui. "Finding all minimum-cost perfect matchings in Bipartite graphs". In: *Networks* 22.5 (1992), pp. 461–468. DOI: 10.1002/net.3230220504 (cit. on p. 37).

[40] M. Fyrbiak, S. Wallat, S. Reinhard, N. Bissantz, and C. Paar. "Graph Similarity and its Applications to Hardware Security". In: *IEEE Trans. Computers* 69.4 (2020), pp. 505–519. DOI: 10.1109/TC.2019.2953752 (cit. on p. 71).

[41] H. N. Gabow. "An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs". In: *J. ACM* 23.2 (1976), pp. 221–234. DOI: 10.1145/321941.321942 (cit. on p. 18).

[42] H. N. Gabow. "Set-merging for the Matching Algorithm of Micali and Vazirani". In: *CoRR* abs/1501.00212 (2015). arXiv: 1501.00212 (cit. on p. 18).

[43] H. N. Gabow and R. E. Tarjan. "A Linear-Time Algorithm for a Special Case of Disjoint Set Union". In: *J. Comput. Syst. Sci.* 30.2 (1985), pp. 209–221. DOI: 10.1016/0022-0000(85)90014-5 (cit. on p. 18).

[44] H. N. Gabow and R. E. Tarjan. "Faster Scaling Algorithms for Network Problems". In: *SIAM J. Comput.* 18.5 (1989), pp. 1013–1036. DOI: 10.1137/0218069 (cit. on pp. 20 sq.).

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979 (cit. on pp. 45, 47).

[46] B. Gärtner and J. Matousek. *Understanding and using linear programming*. Universitext. Springer, 2007 (cit. on p. 22).

[47] A. V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM J. Comput.* 24.3 (1995), pp. 494–504. DOI: 10.1137/S0097539792231179 (cit. on pp. 28, 36).

[48] A. V. Goldberg, S. Hed, H. Kaplan, and R. E. Tarjan. "Minimum-Cost Flows in Unit-Capacity Networks". In: *Theory Comput. Syst.* 61.4 (2017), pp. 987–1010. DOI: 10.1007/s00224-017-9776-7 (cit. on pp. 25 sq.).

[49] A. V. Goldberg and R. Kennedy. "Global Price Updates Help". In: *SIAM J. Discret. Math.* 10.4 (1997), pp. 551–572. DOI: 10.1137/S0895480194281185 (cit. on p. 20).

[50] A. Gupta and N. Nishimura. "Finding Largest Subtrees and Smallest Supertrees". In: *Algorithmica* 21.2 (1998), pp. 183–210. DOI: 10.1007/PL00009212 (cit. on pp. 2, 4, 46, 59–62).

[51] C. Gutwenger and P. Mutzel. "A Linear Time Implementation of SPQR-Trees". In: *Graph Drawing, 8th International Symposium, GD 2000, Colonial Williamsburg, VA, USA, September 20-23, 2000, Proceedings*. Ed. by J. Marks. Vol. 1984. Lecture Notes in Computer Science. Springer, 2000, pp. 77–90. DOI: 10.1007/3-540-44541-2_8 (cit. on pp. 11, 84).

[52] C. Gutwenger and P. Mutzel. "An Experimental Study of Crossing Minimization Heuristics". In: *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*. Ed. by G. Liotta. Vol. 2912. Lecture Notes in Computer Science. Springer, 2003, pp. 13–24. DOI: 10.1007/978-3-540-24595-7_2 (cit. on p. 109).

[53] F. Harary. *Graph Theory*. Addison Wesley series in mathematics. Addison-Wesley, 1971 (cit. on pp. 2, 10).

[54] W. C. Herndon and S. H. Bertz. "Linear notations and molecular graph similarity". In: *Journal of Computational Chemistry* 8.4 (1987), pp. 367–374. DOI: 10.1002/jcc.540080413. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.540080413 (cit. on p. 72).

[55] R. Hoffmann, C. McCreesh, S. N. Ndiaye, P. Prosser, C. Reilly, C. Solnon, and J. Trimble. "Observations from Parallelising Three Maximum Common (Connected) Subgraph Algorithms". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*. Ed. by W. J. van Hoeve. Vol. 10848. Lecture Notes in Computer Science. Springer, 2018, pp. 298–315. DOI: 10.1007/978-3-319-93031-2_22 (cit. on pp. 45, 94).

[56] A. Holder. "Understanding and Using Linear Programming by J. Matoušek; Bernd Gärtner; Introduction to Optimization by Pablo Pedregal". In: *Am. Math. Mon.* 116.5 (2009), pp. 471–476 (cit. on p. 22).

[57] J. E. Hopcroft and R. M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM J. Comput.* 2.4 (1973), pp. 225–231. DOI: 10.1137/0202019 (cit. on pp. 16, 18, 20, 23, 39, 97).

[58] T. Horváth, J. Ramon, and S. Wrobel. "Frequent subgraph mining in outerplanar graphs". In: *Data Min. Knowl. Discov.* 21.3 (2010), pp. 472–508. DOI: 10.1007/s10618-009-0162-1 (cit. on pp. 2, 5, 77).

[59] C.-C. Huang and T. Kavitha. "Efficient algorithms for maximum weight matchings in general graphs with small edge weights". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. Ed. by Y. Rabani. SIAM, 2012, pp. 1400–1412. DOI: 10.1137/1.9781611973099.110 (cit. on p. 20).

[60] L. Humbeck. "Betrachtung der Ähnlichkeit von niedermolekularen Verbindungen unter Berücksichtigung der biologischen Aktivität". Dissertation. TU Dortmund University, 2019 (cit. on pp. 114 sqq.).

[61] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. "On generating all maximal independent sets". In: *Information Processing Letters* 27.3 (1988), pp. 119–123. DOI: http://dx.doi.org/10.1016/0020-0190(88)90065-8 (cit. on p. 11).

[62] T. Kameda and J. I. Munro. "A $\mathcal{O}(|V||E|)$ algorithm for maximum matching of graphs". In: *Computing* 12.1 (1974), pp. 91–98. DOI: 10.1007/BF02239502 (cit. on p. 18).

[63] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. "All-Cavity Maximum Matchings". In: *Algorithms and Computation, 8th International Symposium, ISAAC '97, Singapore, December 17-19, 1997, Proceedings*. Ed. by H. W. Leong, H. Imai, and S. Jain. Vol. 1350. Lecture Notes in Computer Science. Springer, 1997, pp. 364–373. DOI: 10.1007/3-540-63890-3_39 (cit. on pp. 4, 27 sqq.).

[64] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. "An Even Faster and More Unifying Algorithm for Comparing Trees via Unbalanced Bipartite Matchings". In: *J. Algorithms* 40.2 (2001), pp. 212–233. DOI: 10.1006/jagm.2001.1163 (cit. on pp. 4, 46, 52, 59, 71).

[65] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting. "Cavity Matchings, Label Compressions, and Unrooted Evolutionary Trees". In: *SIAM J. Comput.* 30.2 (2000), pp. 602–624. DOI: 10.1137/S0097539797332275 (cit. on pp. 18, 27).

[66] S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. E. Bolton. "PubChem in 2021: new data content and improved web interfaces". In: *Nucleic Acids Research* 49.D1 (Nov. 2020), pp. D1388–D1395. DOI: 10.1093/nar/gkaa971. eprint: https://academic.oup.com/nar/article-pdf/49/D1/D1388/35363961/gkaa971.pdf (cit. on p. 1).

[67] I. Koch. "Enumerating all connected maximal common subgraphs in two graphs". In: *Theor. Comput. Sci.* 250.1-2 (2001), pp. 1–30. DOI: 10.1016/S0304-3975(00)00286-3 (cit. on pp. 5, 94 sq.).

[68] O. Koch, N. M. Kriege, and L. Humbeck. "Chemical Similarity and Substructure Searches". In: *Encyclopedia of Bioinformatics and Computational Biology - Volume 2*. Ed. by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach. Elsevier, 2019, pp. 640–649. DOI: 10.1016/b978-0-12-809633-8.20195-7 (cit. on p. 90).

[69] V. Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Math. Program. Comput.* 1.1 (2009), pp. 43–67. DOI: 10.1007/s12532-009-0002-8 (cit. on p. 19).

[70] N. M. Kriege, A. Droschinsky, and P. Mutzel. "A note on block-and-bridge preserving maximum common subgraph algorithms for outerplanar graphs". In: *J. Graph Algorithms Appl.* 22.4 (2018), pp. 607–616. DOI: 10.7155/jgaa.00480 (cit. on pp. 6, 27, 78, 90, 97).

[71] N. M. Kriege, F. Kurpicz, and P. Mutzel. "On Maximum Common Subgraph Problems in Series-Parallel Graphs". In: *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014, Duluth, MN, USA, October 15-17, 2014, Revised Selected Papers*. Vol. 8986. Lecture Notes in Computer Science. Springer, 2014, pp. 200–212. DOI: 10.1007/978-3-319-19315-1_18 (cit. on pp. 78, 81, 102, 112).

[72] N. M. Kriege and P. Mutzel. "Finding Maximum Common Biconnected Subgraphs in Series-Parallel Graphs". In: *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*. Ed. by E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik. Vol. 8635. Lecture Notes in Computer Science. Springer, 2014, pp. 505–516. DOI: 10.1007/978-3-662-44465-8_43 (cit. on p. 81).

[73] N. M. Kriege. "Comparing Graphs: Algorithms & Applications". PhD thesis. Department of Computer Science, TU Dortmund, 2015 (cit. on pp. 80, 106).

[74] H. W. Kuhn and B. Yaw. "The Hungarian method for the assignment problem". In: *Naval Res. Logist. Quart* (1955), pp. 83–97 (cit. on p. 20).

[75] J. P. Kukluk, L. B. Holder, and D. J. Cook. "Algorithm and Experiments in Testing Planar Graphs for Isomorphism". In: *J. Graph Algorithms Appl.* 8.3 (2004), pp. 313–356. DOI: 10.7155/jgaa.00094 (cit. on p. 93).

[76] K. Kurita, K. Wasa, T. Uno, and H. Arimura. "Efficient Enumeration of Induced Matchings in a Graph without Cycles with Length Four". In: *IEICE Transactions* 101-A.9 (2018), pp. 1383–1391. DOI: 10.1587/transfun.E101.A.1383 (cit. on p. 37).

[77] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Books on Mathematics Series. Dover Publications, 2001 (cit. on pp. 15, 18).

[78] LEMON. *LEMON Graph Library (version 1.3.1)*. http://lemon.cs.elte.hu/trac/lemon. Accessed on 08.08.2019. July 2014 (cit. on p. 18).

[79] A. Lingas. "Subgraph Isomorphism for Biconnected Outerplanar Graphs in Cubic Time". In: *Theor. Comput. Sci.* 63.3 (1989), pp. 295–302. DOI: 10.1016/0304-3975(89)90011-X (cit. on p. 47).

[80] A. Lozano and G. Valiente. "On the maximum common embedded subtree problem for ordered trees". In: *In C. Iliopoulos and T Lecroq, editors, String Algorithmics, chapter 7. King's College London Publications*. 2004 (cit. on p. 46).

[81] F. Luccio, A. M. Enriquez, P. O. Rieumont, and L. Pagli. *Bottom-up subtree isomorphism for unordered labeled trees*. 2004 (cit. on p. 93).

[82] J. Marialke, R. Körner, S. Tietze, and J. Apostolakis. "Graph-Based Molecular Alignment (GMA)". In: *J. Chem. Inf. Model.* 47.2 (2007), pp. 591–601. DOI: 10.1021/ci600387r (cit. on pp. 2, 91).

[83] D. M. Martin and B. D. Thatte. "The maximum agreement subtree problem". In: *Discret. Appl. Math.* 161.13-14 (2013), pp. 1805–1817. DOI: 10.1016/j.dam.2013.02.037 (cit. on pp. 4, 46).

[84] D. W. Matula. "Subtree Isomorphism in $\mathcal{O}(n^{5/2})$". In: *Algorithmic Aspects of Combinatorics*. Ed. by P. H. B. Alspach and D. Miller. Vol. 2. Annals of Discrete Mathematics. Elsevier, 1978, pp. 91–106. DOI: 10.1016/S0167-5060(08)70324-8 (cit. on pp. 45 sq., 49, 54).

[85] C. McCreesh, S. N. Ndiaye, P. Prosser, and C. Solnon. "Clique and Constraint Models for Maximum Common (Connected) Subgraph Problems". In: *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Ed. by M. Rueher. Vol. 9892. Lecture Notes in Computer Science. Springer, 2016, pp. 350–368. DOI: 10.1007/978-3-319-44953-1_23 (cit. on pp. 49, 96).

[86] C. McCreesh, P. Prosser, K. A. Simpson, and J. Trimble. "On Maximum Weight Clique Algorithms, and How They Are Evaluated". In: *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. Ed. by J. C. Beck. Vol. 10416. Lecture Notes in Computer Science. Springer, 2017, pp. 206–225. DOI: 10.1007/978-3-319-66158-2_14 (cit. on p. 49).

[87] C. McCreesh, P. Prosser, and J. Trimble. "A Partitioning Algorithm for Maximum Common Subgraph Problems". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by C. Sierra. ijcai.org, 2017, pp. 712–719. DOI: 10.24963/ijcai.2017/99 (cit. on pp. 45, 94).

[88] B. D. McKay. *Practical Graph Isomorphism*. 1981 (cit. on p. 93).

[89] B. D. McKay and A. Piperno. "Practical graph isomorphism, II". In: *J. Symb. Comput.* 60 (2014), pp. 94–112. DOI: 10.1016/j.jsc.2013.09.003 (cit. on p. 47).

[90] N. A. Meanwell. "Synopsis of Some Recent Tactical Application of Bioisosteres in Drug Design". In: *Journal of Medicinal Chemistry* 54.8 (2011). PMID: 21413808, pp. 2529–2591. DOI: 10.1021/jm1013693. eprint: https://doi.org/10.1021/jm1013693 (cit. on p. 5).

[91] S. Micali and V. V. Vazirani. "An $\mathcal{O}(\sqrt{|V|}\,|E|)$ 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*. IEEE Computer Society, 1980, pp. 17–27. DOI: 10.1109/SFCS.1980.12 (cit. on p. 18).

[92] N. Milo, S. Zakov, E. Katzenelson, E. Bachmat, Y. Dinitz, and M. Ziv-Ukelson. "RNA Tree Comparisons via Unrooted Unordered Alignments". In: *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*. Ed. by B. J. Raphael and J. Tang. Vol. 7534. Lecture Notes in Computer Science. Springer, 2012, pp. 135–148. DOI: 10.1007/978-3-642-33122-0_11 (cit. on pp. 29 sq.).

[93] N. Milo, S. Zakov, E. Katzenelson, E. Bachmat, Y. Dinitz, and M. Ziv-Ukelson. "Unrooted unordered homeomorphic subtree alignment of RNA trees". In: *Algorithms for Molecular Biology* 8 (2013), p. 13. DOI: 10.1186/1748-7188-8-13 (cit. on pp. 4, 29 sq.).

[94] T. Nakayama and Y. Fujiwara. "Computer representation of generic chemical structures by an extended block-cutpoint tree". In: *Journal of Chemical Information and Computer Sciences* 23.2 (1983), pp. 80–87. DOI: 10.1021/ci00038a007 (cit. on p. 10).

[95] G. Neudert and G. Klebe. "fconv: format conversion, manipulation and feature computation of molecular data". In: *Bioinform.* 27.7 (2011), pp. 1021–1022. DOI: 10.1093/bioinformatics/btr055 (cit. on pp. 92, 115).

[96] V. Nicholson, C.-C. Tsai, M. Johnson, and M. Naim. "A subgraph isomorphism theorem for molecular graphs". In: *Graph Theory and Topology in Chemistry*. Stud. Phys. Theoret. Chem. 51. Elsevier, 1987, pp. 226–230 (cit. on p. 78).

[97] N. M. O'Boyle and R. A. Sayle. "Comparing structural fingerprints using a literature-based similarity benchmark". In: *J. Cheminformatics* 8.1 (2016), 36:1–36:14. DOI: 10.1186/s13321-016-0148-0 (cit. on pp. 114, 116).

[98] J. B. Orlin and R. K. Ahuja. "New scaling algorithms for the assignment and minimum mean cycle problems". In: *Math. Program.* 54 (1992), pp. 41–56. DOI: 10.1007/BF01586040 (cit. on p. 20).

[99] P. R. J. Östergård and V. Pettersson. "Enumerating Perfect Matchings in n-Cubes". In: *Order* 30.3 (2013), pp. 821–835. DOI: 10.1007/s11083-012-9279-8 (cit. on p. 37).

[100] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982 (cit. on p. 106).

[101] M. Plummer and L. Lovász. *Matching Theory*. North-Holland Mathematics Studies. Elsevier Science, 1986 (cit. on p. 15).

[102] L. Ramshaw and R. Tarjan. "On Minimum-Cost Assignments in Unbalanced Bipartite Graphs". In: *HP Labs technical report HPL-2012-72R1*. Apr. 2012 (cit. on pp. 21, 23–26, 38, 59).

[103] L. Ramshaw and R. E. Tarjan. "A Weight-Scaling Algorithm for Min-Cost Imperfect Matchings in Bipartite Graphs". In: *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 2012, pp. 581–590. DOI: 10.1109/FOCS.2012.9 (cit. on pp. 17, 25 sq., 59).

[104] M. Rarey and J. S. Dixon. "Feature trees: A new molecular similarity measure based on tree matching". In: *J. Comput. Aided Mol. Des.* 12.5 (1998), pp. 471–490. DOI: 10.1023/A:1008068904628 (cit. on pp. 45, 71, 78).

[105] J. W. Raymond and P. Willett. "Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases". In: vol. 16. 1. 2002, pp. 59–71. DOI: 10.1023/A:1016387816342 (cit. on pp. 45, 71 sq.).

[106] J. W. Raymond and P. Willett. "Maximum common subgraph isomorphism algorithms for the matching of chemical structures". In: *J. Comput. Aided Mol. Des.* 16.7 (2002), pp. 521–533. DOI: 10.1023/A:1021271615909 (cit. on pp. 45, 72, 77 sq., 94 sq.).

[107] G. Reinelt. "TSPLIB - A Traveling Salesman Problem Library". In: *INFORMS J. Comput.* 3.4 (1991), pp. 376–384. DOI: 10.1287/ijoc.3.4.376 (cit. on p. 19).

[108] J.-L. Reymond. "The Chemical Space Project". In: *Accounts of Chemical Research* 48.3 (2015). PMID: 25687211, pp. 722–730. DOI: 10.1021/ar500432k. eprint: https://doi.org/10.1021/ar500432k (cit. on p. 1).

[109] D. Rogers and M. Hahn. "Extended-Connectivity Fingerprints". In: *J. Chem. Inf. Model.* 50.5 (2010), pp. 742–754. DOI: 10.1021/ci100050t (cit. on p. 1).

[110] P. Sankowski. "Weighted Bipartite Matching in Matrix Multiplication Time". In: *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I*. Ed. by M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener. Vol. 4051. Lecture Notes in Computer Science. Springer, 2006, pp. 274–285. DOI: 10.1007/11786986_25 (cit. on p. 20).

[111] L. Schietgat, J. Ramon, and M. Bruynooghe. "A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics". In: *Ann. Math. Artif. Intell.* 69.4 (2013), pp. 343–376. DOI: 10.1007/s10472-013-9335-0 (cit. on pp. 3, 27, 45, 49, 74, 77 sqq., 90, 97 sqq., 102, 105, 109 sq., 112 sq., 116).

[112] L. Schietgat, J. Ramon, M. Bruynooghe, and H. Blockeel. "An Efficiently Computable Graph-Based Metric for the Classification of Small Molecules". In: *Discovery Science, 11th International Conference, DS 2008, Budapest, Hungary, October 13-16, 2008. Proceedings.* Ed. by J.-F. Boulicaut, M. R. Berthold, and T. Horváth. Vol. 5255. Lecture Notes in Computer Science. Springer, 2008, pp. 197–209. DOI: 10.1007/978-3-540-88411-8_20 (cit. on p. 5).

[113] R. Schmidt, F. Krull, A. L. Heinzke, and M. Rarey. "Disconnected Maximum Common Substructures under Constraints". In: *J. Chem. Inf. Model.* 61.1 (2021), pp. 167–178. DOI: 10.1021/acs.jcim.0c00741 (cit. on pp. 2, 91).

[114] P. S. Segundo, S. Coniglio, F. Furini, and I. Ljubic. "A new branch-and-bound algorithm for the maximum edge-weighted clique problem". In: *Eur. J. Oper. Res.* 278.1 (2019), pp. 76–90. DOI: 10.1016/j.ejor.2019.03.047 (cit. on p. 96).

[115] P. S. Segundo, F. Furini, and J. Artieda. "A new branch-and-bound algorithm for the Maximum Weighted Clique Problem". In: *Computers & OR* 110 (2019), pp. 18–33. DOI: 10.1016/j.cor.2019.05.017 (cit. on pp. 49, 94).

[116] E. Sevinç and T. Dökeroglu. "A novel parallel local search algorithm for the maximum vertex weight clique problem in large graphs". In: *Soft Comput.* 24.5 (2020), pp. 3551–3567. DOI: 10.1007/s00500-019-04122-z (cit. on pp. 94, 96).

[117] R. Shamir and D. Tsur. "Faster Subtree Isomorphism". In: *J. Algorithms* 33.2 (1999), pp. 267–280. DOI: 10.1006/jagm.1999.1044 (cit. on p. 47).

[118] G. Software. *Fade2D Delaunay Triangulation (version 1.74)*. https://www.geom.at/fade2d/html/. Accessed on 08.08.2019. Mar. 2019 (cit. on p. 19).

[119] T. Sterling and J. J. Irwin. "ZINC 15 - Ligand Discovery for Everyone". In: *J. Chem. Inf. Model.* 55.11 (2015), pp. 2324–2337. DOI: 10.1021/acs.jcim.5b00559 (cit. on p. 116).

[120] M. M. Syslo. "The Subgraph Isomorphism Problem for Outerplanar Graphs". In: *Theor. Comput. Sci.* 17 (1982), pp. 91–97. DOI: 10.1016/0304-3975(82)90133-5 (cit. on pp. 47, 80 sq.).

[121] M. Thorup. "Equivalence between Priority Queues and Sorting". In: *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings.* IEEE Computer Society, 2002, pp. 125–134. DOI: 10.1109/SFCS.2002.1181889 (cit. on p. 20).

[122] M. Thorup. "Integer priority queues with decrease key in constant time and the single source shortest paths problem". In: *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*. Ed. by L. L. Larmore and M. X. Goemans. ACM, 2003, pp. 149–158. DOI: 10.1145/780542.780566 (cit. on p. 20).

[123] A. Torsello, D. H. Rowe, and M. Pelillo. "Polynomial-Time Metrics for Attributed Trees". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.7 (2005), pp. 1087–1099. DOI: 10.1109/TPAMI.2005.146 (cit. on pp. 3 sq., 45, 49, 74).

[124] T. Uno. "A Fast Algorithm for Enumerating Bipartite Perfect Matchings". In: *Algorithms and Computation, 12th International Symposium, ISAAC 2001, Christchurch, New Zealand, December 19-21, 2001, Proceedings.* Ed. by P. Eades and T. Takaoka. Vol. 2223. Lecture Notes in Computer Science. Springer, 2001, pp. 367–379. DOI: 10.1007/3-540-45678-3_32 (cit. on pp. 4, 12, 37, 40, 42).

[125] T. Uno. "Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs". In: *Algorithms and Computation, 8th International Symposium, ISAAC '97, Singapore, December 17-19, 1997, Proceedings*. Ed. by H. W. Leong, H. Imai, and S. Jain. Vol. 1350. Lecture Notes in Computer Science. Springer, 1997, pp. 92–101. DOI: 10.1007/3-540-63890-3_11 (cit. on pp. 4, 12, 37, 39 sq., 42).

[126] L. G. Valiant. "The Complexity of Computing the Permanent". In: *Theor. Comput. Sci.* 8 (1979), pp. 189–201. DOI: 10.1016/0304-3975(79)90044-6 (cit. on p. 37).

[127] G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002 (cit. on p. 46).

[128] V. V. Vazirani. "A Theory of Alternating Paths and Blossoms for Proving Correctness of the $\mathcal{O}(\sqrt{V}E)$ General Graph Maximum Matching Algorithm". In: *Comb.* 14.1 (1994), pp. 71–109. DOI: 10.1007/BF01305952 (cit. on p. 18).

[129] V. V. Vazirani. "An Improved Definition of Blossoms and a Simpler Proof of the MV Matching Algorithm". In: *CoRR* abs/1210.4594 (2012). arXiv: 1210.4594 (cit. on p. 18).

[130] W. D. Wallis, P. Shoubridge, M. Kraetzl, and D. Ray. "Graph distances using graph union". In: *Pattern Recognit. Lett.* 22.6/7 (2001), pp. 701–704. DOI: 10.1016/S0167-8655(01)00022-8 (cit. on p. 72).

[131] C. Witzgall and C. T. Zahn. "Modification of Edmonds' maximum matching algorithm". In: *J. Res. Nat. Bur. Standards Sect. B*. Citeseer. 1965 (cit. on p. 18).

[132] A. Yamaguchi, K. F. Aoki, and H. Mamitsuka. "Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees". In: *Inf. Process. Lett.* 92.2 (2004), pp. 57–63. DOI: 10.1016/j.ipl.2004.06.019 (cit. on pp. 45, 71).

[133] K. Zhang. "A Constrained Edit Distance Between Unordered Labeled Trees". In: *Algorithmica* 15.3 (1996), pp. 205–222. DOI: 10.1007/BF01975866 (cit. on p. 30).