Comparing Graphs

Algorithms & Applications

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r N a t u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Nils Morten Kriege

Dortmund
2015

Nils Morten Kriege
Lehrstuhl XI - Algorithm Engineering
Fakultät für Informatik
Technische Universität Dortmund
Otto-Hahn-Str. 14
44227 Dortmund

Tag der mündlichen Prüfung:	**28.09.2015**
Dekan:	**Prof. Dr. Gernot A. Fink**
Gutachter:	**Prof. Dr. Petra Mutzel** (TU Dortmund, Fakultät für Informatik) **Prof. Dr. Kristian Kersting** (TU Dortmund, Fakultät für Informatik)

Abstract

Graphs are a well-studied mathematical concept, which has become ubiquitous to represent structured data in many application domains like computer vision, social network analysis or chem- and bioinformatics. The ever-increasing amount of data in these domains requires to efficiently organize and extract information from large graph data sets. In this context techniques for comparing graphs are fundamental, e.g., in order to obtain meaningful similarity measures between graphs. These are a prerequisite for the application of a variety of data mining algorithms to the domain of graphs. Hence, various approaches to graph comparison evolved and are wide-spread in practice. This thesis is dedicated to two different strategies for comparing graphs: maximum common subgraph problems and graph kernels.

We study maximum common subgraph problems, which are based on classical graph-theoretical concepts for graph comparison and are NP-hard in the general case. We consider variants of the maximum common subgraph problem in restricted graph classes, which are highly relevant for applications in cheminformatics. We develop a polynomial-time algorithm, which allows to compute a maximum common subgraph under *block and bridge preserving* isomorphism in series-parallel graphs. This generalizes the problem of computing maximum common biconnected subgraphs in series-parallel graphs. We show that previous approaches to this problem, which are based on the separators represented by standard graph decompositions, fail. We introduce the concept of potential separators to overcome this issue and use them algorithmically to solve the problem in series-parallel graphs. We present algorithms with improved bounds on running time for the subclass of outerplanar graphs. Finally, we establish a sufficient condition for maximum common subgraph variants to allow derivation of graph distance metrics. This leads to polynomial-time computable graph distance metrics in restricted graph classes. This progress constitutes a step towards solving practically relevant maximum common subgraph problems in polynomial time.

The second contribution of this thesis is to graph kernels, which have their origin in specific data mining algorithms. A key property of graph kernels is that they allow to consider a large (possibly infinite) number of features and can support graphs with arbitrary annotation, while being efficiently computable. The main contributions of this part of the thesis are (i) the development of novel graph kernels, which are especially designed for attributed graphs with arbitrary annotations and (ii) the systematic study of implicit and explicit mapping into a feature space for computation of graph kernels w.r.t. its impact on the running time and the ability to consider arbitrary annotations. We propose graph kernels based on bijections between subgraphs and walks of fixed length. In an experimental study we show that these approaches provide a viable alternative to known techniques, in particular for graphs with complex annotations.

Acknowledgements

Many people supported and encouraged me during the work on this thesis. First of all, I want to thank my advisor Prof. Petra Mutzel for giving me the opportunity to work in her research group at the TU Dortmund. I am deeply grateful for her scientific guidance during the past years and for giving me the freedom to pursue my own research ideas. She also made various conference trips all over the world possible, which I enjoyed a lot.

I would also like to thank my co-advisor Prof. Kristian Kersting for the interesting scientific discussions and for immediately agreeing to serve as the second referee of this thesis. Furthermore, I would like to thank Prof. Johannes Fischer and Dr. Falko Bause for agreeing to serve as members of my defense committee.

I am grateful to the co-authors of my papers for sharing their experience and knowledge. I would also like to thank our collaborators from the department of chemical biology, in particular, the members of the group of Dr. Oliver Koch for sharing their chemical knowledge.

I would like to thank my colleagues at the Chair XI for the pleasant and productive atmosphere as well as for the relaxing coffee breaks. Special thanks go to Dr. Karsten Klein and my roommates Bernd Zey and Till Schäfer for numerous interesting discussions. I would like to thank Fritz Bökler, Andre Droschinsky, Christopher Morris, Till Schäfer and Dr. Jan Kriege for proofreading parts of this thesis and making valuable suggestions.

Finally, I would like to express my deep gratitude to my family, in particular, my parents, and Karen, who supported me in every possible way in all times.

Nils Morten Kriege
Essen, June 2015

Contents

Ein Unterschied, der dem Verstand
nichts gibt, ist kein Unterschied.

Johann Wolfgang von Goethe
(1749–1832)

Chapter 1

Introduction

Similarity between objects is subjective and there is no absolute standard measure. Yet recognizing similarities is a fundamental principle in perception and facilitates to simplify and understand complex relations. A concept of similarity is as well the key for various data mining methods used to discover patterns and relations in possibly large data sets of objects representing real-world entities. Graphs are a versatile data structure used to represent structured objects in many application domains such as biology, chemistry, pattern recognition or social networks analysis. The breadth of problems requiring to deal with graphs is growing rapidly and a great number of approaches to their comparison have been developed and are widely used in practice. This thesis is dedicated to two different strategies for comparing graphs: maximum common subgraph problems and graph kernels. While the first is a classical graph-theoretical approach, graph kernels inherently transform graphs into vector data and have their origin in specific data mining algorithms.

It would go beyond the scope of this thesis to review graph comparison techniques comprehensively and we confine ourselves to representative techniques with fundamentally different characteristics. Figure 1.1 gives an overview over important techniques, which can be divided into three categories. A traditional approach is to represent graphs by vectors, which is in particular appealing since such data is easy to manage and compare. Vector representations of graphs typically count the number of occurrences of specific *features* of a graph. Each element of the vector is associated with one feature, for example, a specific subgraph or path. In case of binary fingerprints just the presence of a feature is indicated by a bit, but the number of occurrences is not counted. Since features typically are of limited size and number, vector representations can be generated and compared efficiently. This technique is widely used in application domains like cheminformatics, where molecules are represented by so-called *chemical fingerprints*. Graph kernels can be considered a generalization of the above technique and do not necessarily require the size and number of features to be limited to compute a similarity measure

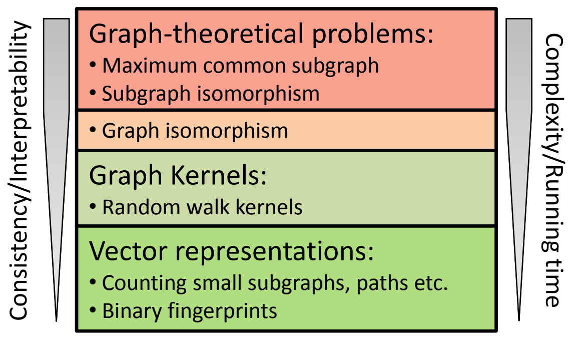


Figure 1.1: Techniques for graph comparison

efficiently. Notably, the number of all walks two graphs have in common can be taken into account by a polynomial-time computable similarity measure, although the length and number of walks is infinite. However, more complex features, e.g., without repeated vertices, may lead to graph kernels that no longer can be computed in polynomial time, unless $\mathsf{P} = \mathsf{NP}$.

A graph-theoretical problem closely related to graph comparison asks for a *maximum common subgraph* of two given graphs, i.e., a largest possible subgraph that is contained in both graphs. This problem is well-known to be NP-hard, but has the unique advantage that a derived similarity measure is well interpretable by visual inspection of the common subgraph. The problem generalizes the subgraph isomorphism problem, which is as well NP-hard, and the graph isomorphism problem. The graph isomorphism problem asks whether two graphs have the same structure and has neither been shown to be NP-complete nor polynomial-time solvable. This indicates the high complexity inherent to these approaches to compare graphs.

The techniques briefly introduced above have varying characteristics and differ in terms of computational complexity and their general notion of similarity. The most important—and most challenging—property of a similarity measure is to which extent it provides valid results with respect to a specific standard or task. As stated initially, there is no absolute standard similarity measure and this is as well the case for many concrete applications of graph comparison techniques. However, it is possible to assess the validity by investigating, for example, how well the result of a data mining algorithm based on a specific similarity measure aligns with experimental results or the expectations of domain specialists.

1.1 Applications in Cheminformatics

We motivate the importance of graph comparison techniques by examples from the domain of cheminformatics. As this thesis can offer only a glimpse into this topic, we refer the reader to textbooks like (Gasteiger and Engel, 2003; Bajorath, 2004; Bunin et al., 2007; Bajorath, 2011) for a more comprehensive introduction. We already use basic concepts from graph theory in this section, which are formally defined in Section 2.

The term *cheminformatics* (or synonymously *chemoinformatics*) is relatively new and was coined in the late nineties by Brown (1998). Gasteiger and Engel (2003) concisely define the term as follows:

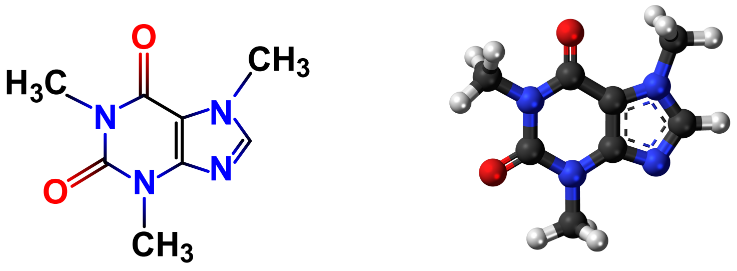
> "Chemoinformatics is the application of informatics methods to solve chemical problems."

Notably, such methods have been used in chemistry long before the term cheminformatics has been introduced and their application can be dated back to the advent of computers in the 1940s (Brown, 2009). However, it is apparent that there is a growing need for computer aided methods caused, among others, by the increasing amount of available chemical data.

Drug discovery is the main application area of cheminformatics and aims at the development of new medications to cure particular diseases. Most drugs are so called *small molecules* with a low molecular weight. The search for a new drug is a time consuming and rather expensive process, which typically starts with the identification of a target protein related to the biological process that should be influenced. In a next step, small molecules that bind to the target protein are searched, which are referred to as *hits*. This process involves the test of large synthesized libraries of chemical compounds for biological activity against the target protein in automated high-throughput screening methods. The discovery of hits is just the starting point, which is followed by the selection of promising molecules and their modification and optimization, e.g., to reduce side effects, increase the oral availability or avoid patent issues. The bioactivity results of screening tests as well as further information are stored for each molecule together with its structure in *chemical databases*. This information is increasingly made publicly available by projects like PubChem or ChEMBL (Gaulton et al., 2012). PubChem by now contains more than 68 million[1] chemical compounds and is growing rapidly.

The term *chemical space* refers to the infinite set of all theoretically possible molecules. The subset of druglike molecules contains only those molecules that satisfy certain criteria, which are essential for the availability as drug. These criteria are not clearly defined and estimates of the number of druglike molecules range from 10^{12} to 10^{180} molecules (Brown, 2009). In any case only an extremely small fraction of these compounds is of interest for a specific disease and the search for a potential new drug is therefore often compared to "searching a needle in a haystack". Clearly, it is not possible to synthesize and test all druglike molecules and, hence, it is essential to explore the space of these molecules systematically in order to find molecules with the desired

[1]PubChem compound search result, 12 May 2015, `https://pubchem.ncbi.nlm.nih.gov`



(a) Structural formula (b) Ball-and-stick model

Figure 1.2: Two representations of the caffeine molecule.[2]

properties and the required biological activity. One starting point is the space spanned by molecules contained in publicly available databases of compounds that have been synthesized or are even available for purchase. Furthermore, large data sets of molecules that never have been synthesized can be generated *in silico* by computer aided methods according to chemical rules. Since the structure of a molecule is closely related to its properties, a key technique is to organize and mine molecules based on their structural similarity (Maggiora and Shanmugasundaram, 2011). We first review how the structure of a molecule can be represented as graph and discuss the relation between typical tasks in cheminformatics and the associated graph theoretical problems.

1.1.1 Graphs in Cheminformatics

Long before computers and the term cheminformatics emerged, chemistry was one of the early application areas of graph theory (see Biggs et al., 1986, Chapter 4): *Isomers* are molecules with the same number of atoms of each element, but different chemical structure. From the middle of the nineteenth century the fact that isomers may exhibit different properties was explained by considering the chemical bonds between the individual atoms. In this context first graphical notations for molecules similar to structural formulas as they are common today were introduced, cf. Figure 1.2(a). The term *graph* was originally introduced by the mathematician J. J. Sylvester as an abbreviating term for these notations (Biggs et al., 1986). The close relation between the field of chemistry and graph theory also becomes apparent by the chemical term *valency*, which is synonymously used for the degree of a vertex in graph theory.

It is apparent that a structural formula as shown in Figure 1.2(a) can be directly interpreted as graph. For some tasks the 3D arrangement of atoms as shown in Figure 1.2(b) is more adequate. However, representing

[2] Figure 1.2(b), image source: `http://en.wikipedia.org/wiki/File:Caffeine_%281%29_3D_ball.png`.

this information by a graph is less intuitive and there are different approaches, e.g., by incorporating distances between atoms (see Marialke et al., 2007). We will come back to such models later in Section 4.7.2 and for now focus on the first case.

The *molecular graph* of a chemical compound is the labeled graph, where vertices represent the atoms and edges the bonds of the molecule. The vertex labels correspond to the atom types identified by their chemical symbol, e.g., `C`, `N`, `O` etc. for carbon, nitrogen and oxygen, respectively. The edge labels encode the type of chemical bonds, e.g., `-`, `=` etc. for single and double bonds, respectively. Since the hydrogen atoms are essentially determined by the atom type and the bonds between non-hydrogen atoms, they are often not explicitly represented by the molecular graph.

Molecular graphs derived from small molecules typically have several characteristic properties:

- Molecular graphs exhibit a characteristic distribution of vertex and edge labels. Kriege (2009) considered a data set of 187 266 molecular graphs and observed that 45.1% of all vertices represent hydrogen atoms, 40.8% carbon atoms, 6.5% oxygen and 5.2% nitrogen atoms. The vertices representing other atom types each constitute less than 1% of the total number of vertices.

- The maximum degree of molecular graphs is bounded by a small constant (≤ 4 with very few exceptions) as a result of the atom valency (also see Kriege, 2009, Section 3.4.1).

- Almost all molecular graphs are planar,[3] and most are even outerplanar. Horváth et al. (2010) considered the NCI data set[4] of 250 251 chemical compounds and recognized 8.8% of these graphs as trees and 94.3% as being outerplanar. For every block of these graphs the authors considered the number of edges that are not incident to the outer face in an outerplanar embedding. At most 11 such edges were found in any block and 99.9% of these graphs exhibit at most 5 such edges in any block. Hence, these graphs are outerplanar 6-almost trees.

- Molecular graphs typically are tree-like. Yamaguchi et al. (2003) computed the tree width of 9 712 molecular graphs derived from the LIGAND database (Goto et al., 2002) and found that there is only one graph with tree width 4, while all other have tree width at most 3. Actually, 19.4% of the graphs are trees and 95% are partial 2-trees, i.e., series-parallel graphs.

[3]Note, that we mean planar in a graph theoretical sense, while a molecule typically is considered planar if all atoms are on the same plane w.r.t. their 3D arrangement.

[4]National Cancer Institute (NCI) Open Database Compounds, `http://cactus.nci.nih.gov/download/nci/`

These properties can be exploited in order to obtain algorithms, which are able to compare molecular graphs efficiently.

1.1.2 Comparing Molecular Graphs

A chemical database can be interpreted as a graph database containing a set $\mathcal{D} = \{G_1, \ldots, G_n\}$ of molecular graphs. There are several common tasks in cheminformatics closely related to graph isomorphism problems.

Unique identification: A database should not contain duplicates and it must be possible to test if a molecule with a given structure is contained in the database. Hence, it is beneficial to associate a unique identifier with each molecule, which is computed from its structure. So-called *canonical SMILES* (Brown, 2009) are one example of standard methods for this purpose, which can be considered a complete invariant for labeled graphs, cf. Problem 2.2.

Substructure search: It is a common task to search for the molecules in the database that contain a specified structure. Interpreting the specified structure as graph Q, this task requires to decide for every graph $G \in \mathcal{D}$ if a subgraph isomorphism from Q to G exists, cf. Problem 2.3. Since chemical databases must be able to answer a large number of substructure queries with a short response time, typically *graph indexing* techniques are employed. These techniques build an index once in a preprocessing step in order to be able to answer subsequent queries more efficiently (see Klein et al., 2011, and references therein).

Structural similarity: Structural similarity is a pervasive concept in cheminformatics and there are various tasks that require to determine a meaningful similarity between molecular graphs. Analogously to substructure search one might be interested in the k molecules of the database that are most similar to a given query structure. Another common task is to identify homogeneous subsets of similar molecules by clustering techniques. Such a clustering can, for example, be employed to explore the database systematically or to find representative subsets. Finally, it is desirable to predict the biological properties of untested molecules based on a subset of molecules with known properties. All these tasks are closely related to similarity measures between graphs.

1.1.3 Exploring Chemical Space with Scaffold Hunter

We further motivate the importance of graph comparison in the domain of cheminformatics by presenting the tool Scaffold Hunter, which was developed with the goal to facilitate drug discovery by means of visual analytics techniques (Wetzel et al., 2009; Klein et al., 2012; 2013a). After introducing the

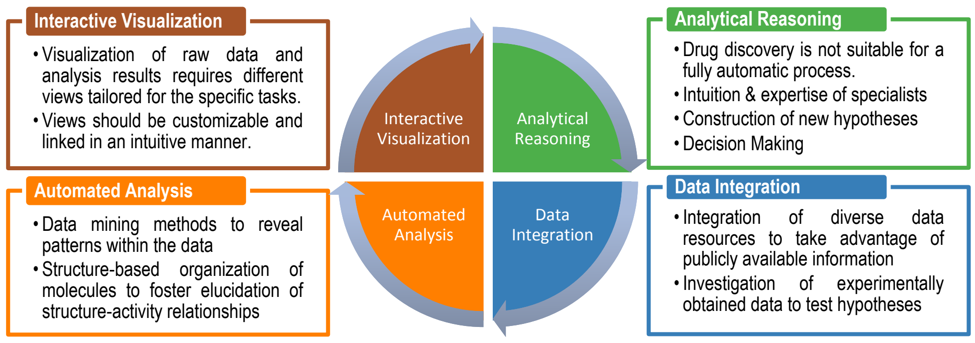

Interactive Visualization
• Visualization of raw data and analysis results requires different views tailored for the specific tasks.
• Views should be customizable and linked in an intuitive manner.
Automated Analysis
• Data mining methods to reveal patterns within the data
• Structure-based organization of molecules to foster elucidation of structure-activity relationships
Interactive Visualization
Analytical Reasoning
Automated Analysis
Data Integration
Analytical Reasoning
• Drug discovery is not suitable for a fully automatic process.
• Intuition & expertise of specialists
• Construction of new hypotheses
• Decision Making
Data Integration
• Integration of diverse data resources to take advantage of publicly available information
• Investigation of experimentally obtained data to test hypotheses

Figure 1.3: Challenges for visual analytics approaches in drug discovery

software and the underlying concepts, we discuss the use of graph comparison techniques in the context of the software. Scaffold Hunter is a publicly available[5] Java-based open source tool that has been continuously developed since its start in 2007 as a student project group at TU Dortmund (Gorecki et al., 2008).

Visual Analytics for Chemical Databases

Visual analytics is the science of analytical reasoning facilitated by interactive visual interfaces (Thomas and Cook, 2005). Hence, it combines techniques from several fields like data mining and information visualization. The drug discovery process requires a large amount of time, money and other resources and suffers from a small and even decreasing success rate. Although automated computational methods have become a standard tool, drug discovery essentially depends on the intuition and expertise of specialists. This observation suggests that drug discovery can greatly benefit from visual analytics techniques, which is also supported by the fact that an increasing number of software tools incorporating visualization techniques have been developed recently (see, e.g., Lounkine et al., 2010; Bertini et al., 2011; Gutlein et al., 2012). Figure 1.3 illustrates the knowledge discovery process typically underlying visual analytics approaches and highlights the specific challenges arising when this concept is applied to the domain of drug discovery. In drug discovery a straight forward process from raw data over analysis to visualization is not appropriate.

The architecture of Scaffold Hunter has been designed to support a workflow in accordance with the visual analytics concepts, see Figure 1.4. When first analyzing and browsing a compound library, typically new hypotheses are generated and it is desirable to refine the current data set by filtering, or to perform new specific computational analysis. Moreover, further experiments may be conducted and the results obtained should again be integrated into

[5] `http://scaffoldhunter.sourceforge.net`

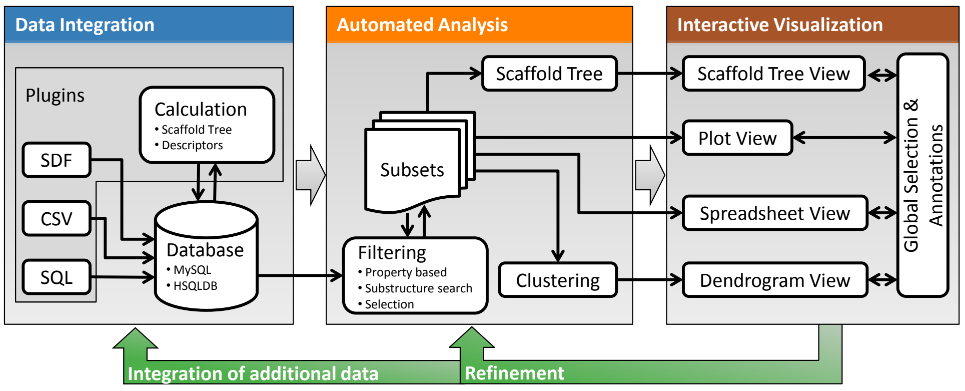


Figure 1.4: Realization of visual analytics concepts in Scaffold Hunter

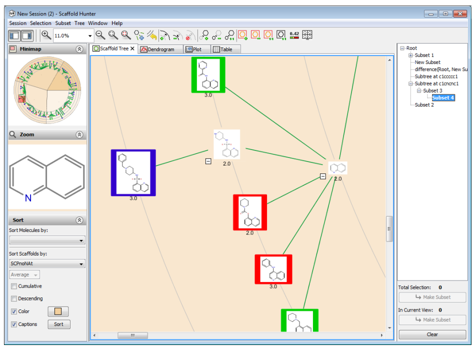
the database for detailed investigation. The concepts and their implementation have been shown to be useful in practice for several cheminformatics tasks (Wetzel et al., 2009; Klein et al., 2013a).

The tool offers a flexible plugin mechanisms to allow integration of chemical data form various resources into an internal database. The data can be further enhanced by the automatic calculation of chemical descriptors like different chemical fingerprints. Scaffold Hunter provides a subset management, which allows to organize molecules into hierarchies of subsets, e.g., to partition the data into smaller sets or to store interesting intermediate results. Subsets can be created by manual selection of molecules or by means of a versatile filtering mechanism based on properties stored in the database. In addition, an integrated structure editor allows the user to perform filtering based on substructures. Subsets can be analyzed and visually explored by several methods in order to allow the user to view the data from different perspectives.

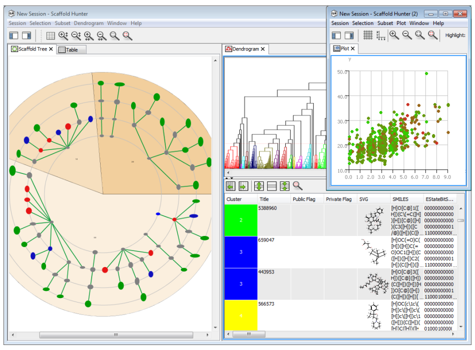
Methods and Views

Scaffold Hunter supports multiple views on the data opened simultaneously, cf. Figure 1.5(b), and allows to quickly switch between views and to open new views for specific subsets. A global selection concept links the simultaneous representations of compounds over all views. Numerical property values can be represented by mapping colors to the visual representations or by scaling objects, see Figure 1.5.

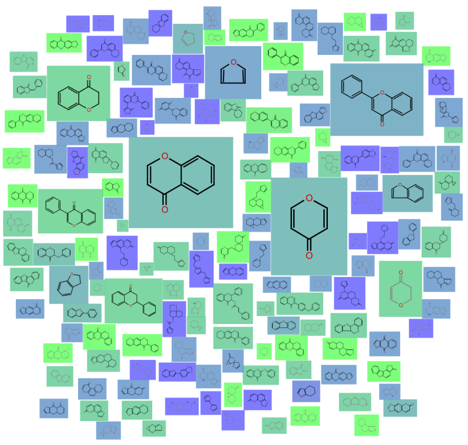
Scaffold based approaches. The scaffold tree algorithm computes a hierarchical classification for chemical compound sets based on their common core structures referred to as *scaffolds*. We only summarize the basic ideas of the algorithm, please refer to (Schuffenhauer et al., 2007) for further details. Essentially the scaffold tree algorithm proceeds as follows: Each compound is



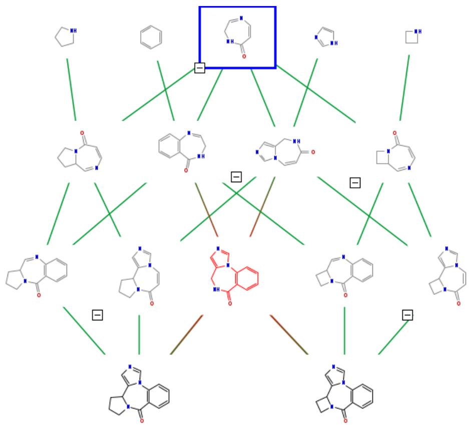
(a) Scaffold tree view



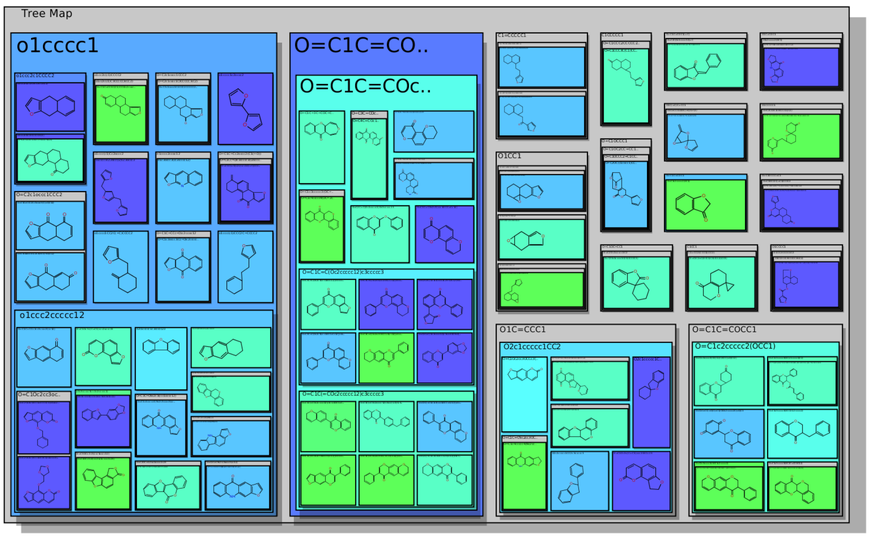
(b) Multiple linked views



(c) Molecule cloud



(d) Scaffold network



(e) Tree map

Figure 1.5: Different views on chemical compound data sets

associated with its unique scaffold that is obtained by cutting off all terminal side chains preserving double bonds directly attached to the core structure. Then each scaffold is simplified stepwise by removing a single ring, such that the obtained ancestor scaffold remains connected. In the case that multiple rings can be pruned, different parent scaffolds can be generated. From these a unique scaffold is selected by a set of deterministic rules based on structural considerations with the aim to preserve the most characteristic ring structure. The procedure terminates as soon as a scaffold consisting of a single ring is obtained. Typically multiple molecules in a data set share a common scaffold and ancestors generated in the successive process of simplification coincide. By merging the scaffolds that occur more than once, the scaffold tree is obtained. Scaffold Hunter allows to visualize the scaffold tree based on different tree layout techniques, e.g., a radial layout as shown in Figure 1.5(b). A different representation of the scaffold tree by means of *tree maps* (Johnson and Shneiderman, 1991) is as well supported, see Figure 1.5(e). This alternative space-filling approach represents the scaffold tree structure by nested rectangles.

The scaffold tree concept has proven to be useful for various research tasks (see, e.g., Renner et al., 2009; Wetzel et al., 2010; Bon and Waldmann, 2010). The selection of a single parent scaffold simplifies the structural relations and allows for intuitive visualization. However, there is no single optimal solution to select a unique parent scaffold for a specific task and, for example, bioactivity was employed as well as a selection criterion (Renner et al., 2009). Considering all parent relations leads to a directed acyclic graph and visualization of large data sets becomes a challenging task. Figure 1.5(d) shows an example of a small data set visualized within Scaffold Hunter using the Sugiyama layout techniques of the graph drawing library OGDF (Chimani et al., 2013).

Molecule cloud. Word clouds are a popular method to visualize key words of large textual data sets. The concept was recently transferred to the domain of molecular databases: The *molecule cloud* proposed by Ertl and Rohde (2012) represents scaffolds as cloud diagram, where the size of a scaffold represents the number of molecules associated with it. The approach has been integrated as a view module in Scaffold Hunter, see Figure 1.5(c). Going beyond the original concept, the molecule cloud view utilizes layout algorithms for *semantics-preserving* word clouds (Barth et al., 2014), which allow to place semantically related objects close to each other. Different similarity measures for chemical compounds are supported to arrange the scaffolds of a cloud in a meaningful manner.

Clustering. Scaffold Hunter includes sequential agglomerative hierarchical non-overlapping (SAHN) clustering techniques, which realize a classification concept orthogonal to the scaffold tree concept (Schuffenhauer and Varin, 2011). Starting with individual objects as clusters, SAHN clustering techniques successively merge the two clusters that are closest to each other w.r.t. to some distance function. The distance between two clusters typically is defined based on the distances between the individual elements they contain. Several alternative approaches to this task, so-called *linkage* methods, are known. The result of this merging process is a binary tree, where every inner node is associated with a merge operation and the corresponding distance between this pair. This tree is commonly visualized as a dendrogram, see Figure 1.5(b) (right part of the main window). SAHN clustering methods are heavily applied in cheminformatics (Downs and Barnard, 2003) and users may be accustomed to dendrogram representations. Scaffold Hunter supports various distance measures defined on either the numeric properties of molecules or their chemical fingerprints, which encode their structure. For large data sets a heuristic SAHN clustering algorithm has been integrated, which is applicable provided that the employed distance function satisfies the triangle inequality. The approach obtains subquadratic running time in practice with a linear number of distance computations (Kriege et al., 2014b).

Applications of Graph Comparison Techniques

There are several graph comparison techniques implemented in Scaffold Hunter and various possible extensions that could improve the usability of the software further. We summarize the use of these techniques and the requirements for adequate approaches stemming from the concrete application.

The internal database relies on unique identifiers of molecules and scaffolds realized by canonical SMILES. These are used as well for the identification of scaffolds obtained multiple times in the course of the scaffold tree algorithm. The substructure search functionality is implemented by an efficient graph-based system employing fingerprints for filtering and a subgraph isomorphism algorithm tailored to molecular graphs for verification of false positives (Klein et al., 2011). The exploration process could be supported by techniques for similarity searching, which are not yet integrated.

Methods to determine the structural (dis)similarity between molecules are employed by several views: Clustering techniques rely on a distance function between molecules. The efficient heuristic approach integrated in Scaffold Hunter has been combined with various graph distance metrics derived from (i) molecular fingerprints, (ii) the size of maximum common subgraphs, and (iii) graph kernels (Kriege et al., 2014b). The experimental comparison of these approaches suggests that fingerprints perform worst in combination with the clustering algorithm, where the task was to recreate planted cluster structures in synthetic graphs. The heuristic approach was shown to be able to cluster considerably larger data sets on standard hardware, which is crucial for the data sets typically considered in cheminformatics. This has been achieved by exploiting the triangle inequality, which is satisfied by the employed distance measures. Hence, there is a demand for similarity measures that satisfy certain mathematical properties. This is as well motivated by so-called kernel methods like support vector machines, which can be used for classification. Incorporating such techniques allows to predict properties of untested molecules based on a set of experimentally tested molecules and would be a useful extension of Scaffold Hunter.

Almost all views already allow or could greatly benefit form incorporating structural similarity, since elucidating the relation between structure and chemical properties is a key task in drug discovery. The general idea that similar molecules should be located close to each other is, for example, realized by ordering the children of all nodes in the scaffold tree according to their similarity and is an integral part of the molecule cloud view. The tree map could benefit as well from placing similar compounds near to each other. This direction has been explored by Gronemann et al. (2013), where the degrees of freedom in arranging nested substructures by a tree mapping algorithm are exploited to optimize secondary constraints. Here, similarities cannot be expected to be represented without distortion and, hence, efficient approaches, which give less accurate results, seem to be adequate.

Two requirements for graph comparison techniques used in this context are apparent: First, interactive visualization requires a fast response time and, hence, graph comparison techniques employed must be efficient. Second, the similarity should be well interpretable, e.g., by highlighting the parts two molecules have in common.

1.2 Challenges in Graph Comparison

Efficient techniques for comparing graphs are the key to organize, manage and mine structured data efficiently. The requirements and constraints for these techniques heavily depend on the specific task and the application domain and sometimes cannot yet be reconciled. In particular, one wants to obtain measures of similarity, which (i) are easy to interpret, (ii) are valid and meaningful w.r.t. the considered task, (iii) can be computed efficiently, (iv) satisfy specific mathematical properties, (v) are applicable to graphs with the annotations that are required to model real-world objects adequately.

There is a complex interplay between these requirements: The employed graph model clearly has a crucial impact on the computational efficiency and the validity of a similarity measure. For example, molecules could be represented by so-called *reduced graphs*, a summary representation, where groups of connected atoms are collapsed into a single vertex (Birchall and Gillet, 2011). Clearly, reduced graphs are smaller and often less complex, potentially allowing for more efficient algorithms. When using such graph models, one might want to annotate vertices of the graphs by additional information, e.g., the molecular weight of the group they represent in order to obtain a valid similarity measure. In this case techniques for graph comparison must be able to take such information into account.

A similarity measure is easily interpretable if it allows to understand why two graphs are considered similar. This is often desirable for similarity measures used in data mining and is even more essential when employed for visual analytics. We have detailed that the drug discovery process depends on the expertise and intuition of specialists. The similarity value of two molecules computed based on their fingerprints does not allows for intuitive interpretation while common substructures could be highlighted and visualized. However, computing common subgraphs of maximum size is a hard computational problem while fingerprint based methods allow for efficient computation.

In order to obtain valid results, standard graph theoretic problems may be inadequate or must be modified. The success of scaffold based approaches suggests that rings and chains of molecules could be handled separately to some extent. Instead of collapsing rings into single vertices, one could, for example, require that common substructures must preserve these in some kind of way. Such modifications again may have an impact on the complexity of the related computational problem. Finally, it is desirable that measures of similarity satisfy certain mathematical properties. For example, every object should be considered more similar to itself than to any other objects. A dissimilarity between objects possibly should satisfy the triangle inequality.

Considering these partly conflicting demands, a single approach to graph comparison that satisfies all needs simultaneously cannot be expected.

1.3 Contribution and Organization of this Thesis

In Chapter 2 we provide basic definitions and results from graph theory. Following we consider two techniques for comparing graphs, which exhibit distinct characteristics and pursue fundamentally different goals.

Chapter 3 is dedicated to common subgraph problems. Computing common subgraphs of maximum possible size exceptionally allows to derive easily interpretable similarity measures, but involves solving a computationally hard problem. In order to obtain efficient algorithms for such problems, we focus on structurally simple graphs with the goal to extend the class of graphs for which efficient algorithms are known. This is motivated by the application in cheminformatics, where molecular graphs are known to have certain characteristics as discussed above. Furthermore, additional constraints for meaningful common subgraphs can be applied. In particular, we study a variation, which distinguishes between rings and chains—the blocks and bridges of the molecular graph—to obtain polynomial running time in series-parallel graphs. We discuss under which conditions graph distance metrics can be derived from the different notions of common subgraphs.

The second contribution of this thesis is to the relatively novel field of graph kernels and presented in Chapter 4. Graph kernels can be considered a generalization of traditional techniques based on vector embeddings. Graph kernels satisfy the properties of an inner product in a feature space and thereby allow to apply various machine learning algorithms to the domain of graphs. A key advantage of graph kernels compared to classical methods is that they allow to consider a large (possibly infinite) number of features and can support graphs with arbitrary annotation, while being efficiently computable. Notably, the flexibility of annotating graphs with arbitrary attributes allows more adequate graph models of real-world objects and may ultimately lead to more valid similarity measures. The main contributions of this part of the thesis are (i) the development of novel graph kernels, which fully exploit these advantages and (ii) the systematic study of the decline in efficiency caused by the flexibility to consider arbitrary annotations.

Chapters 3 and 4 both present an overview on the associated related work, put our achievements in the context of the state of the art and give an outlook on future work. Finally, Chapter 5 concludes by summarizing the contributions presented in this thesis and their impact on graph comparison in general.

1.4 Corresponding Publications

The results presented in this thesis have partially been published as listed below. The software Scaffold Hunter briefly described in Section 1.1.3 has been presented in the following publications:

- Karsten Klein, Nils Kriege, and Petra Mutzel. Scaffold Hunter – visual analysis of chemical compound databases. In *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications (GRAPP & IVAPP)*, pages 626–635, 2012

- Karsten Klein, Nils Kriege, and Petra Mutzel. Scaffold Hunter: Facilitating drug discovery by visual analysis of chemical space. In Gabriela Csurka, Martin Kraus, Robert S. Laramee, Paul Richard, and José Braz, editors, *Computer Vision, Imaging and Computer Graphics. Theory and Application*, volume 359 of *Communications in Computer and Information Science*, pages 176–192. Springer Berlin Heidelberg, 2013b. ISBN 978-3-642-38240-6

- Karsten Klein, Oliver Koch, Nils Kriege, Petra Mutzel, and Till Schäfer. Visual analysis of biological activity data with Scaffold Hunter. *Molecular Informatics*, 32 (11-12):964–975, 2013a. ISSN 1868-1751

Chapter 3 contains results from the following articles. Further details are presented at the beginning of the chapter.

- Nils Kriege and Petra Mutzel. Finding maximum common biconnected subgraphs in series-parallel graphs. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014*, volume 8635 of *Lecture Notes in Computer Science*, pages 505–516. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44464-1

- Nils Kriege, Florian Kurpicz, and Petra Mutzel. On maximum common subgraph problems in series-parallel graphs. In Kratochvíl Jan, Mirka Miller, and Dalibor Froncek, editors, *International Workshop on Combinatorial Algorithms, IWOCA 2014*, volume 8986 of *Lecture Notes in Computer Science*, pages 200–212. Springer International Publishing, 2014a. ISBN 978-3-319-19314-4. Journal version submitted to the European Journal of Combinatorics

- Andre Droschinsky, Nils Kriege, and Petra Mutzel. Efficient enumeration algorithms for common subgraph problems in outerplanar graphs. 2015. In preparation

Chapter 4 contains results previously published in the following articles. Further details are presented at the beginning of the chapter.

- Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012

- Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit versus implicit graph feature maps: A computational phase transition for walk kernels. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 881–886, Dec 2014c

Sehr geringe Unterschiede begründen
manchmal sehr große Verschiedenheiten.

Marie von Ebner-Eschenbach
(1830–1916)

Chapter 2

Preliminaries

In this section we introduce the notation and terminology used throughout this thesis. Basic concepts of graph theory are presented here, additional data structures that are only relevant for individual sections are introduced where needed. For a comprehensive introduction to graph theory, please refer to a standard textbook, e.g., (Diestel, 2005).

2.1 Numbers, Sets, Relations and Functions

We refer to the set of natural numbers including zero by $\mathbb{N} = \{0, 1, 2, \dots\}$, $\mathbb{N}^+ = \mathbb{N}\setminus\{0\}$ denotes the natural numbers without zero. The real numbers are denoted by $\mathbb{R}$ and we refer by $\mathbb{R}_{\geq 0}$ and $\mathbb{R}^+$ to the non-negative and positive real numbers, respectively. We denote the set of all k-element subsets of a set M by $[M]^k$. The union of two sets M_1 and M_2 is denoted by $M_1 \cup M_2$ and the disjoint union by $M_1 \uplus M_2$. A *partition* of a set M is a set $\mathcal{M} = \{M_1, \dots, M_k\}$ of disjoint non-empty subsets of M with $M = \biguplus_{i=1}^{k} M_i$. The subsets $M_1, \dots, M_k$ are said to be the *cells* of the partition. We denote vectors $\mathbf{v} \in K^n$ and matrices $\mathbf{M} \in K^{n \times n}$ over (a field) K in bold, using capital letters for the latter. For a vector $\mathbf{v}$ we refer to the element at position i either by v_i or $[\mathbf{v}]_i$. We denote the *inner product* of two vectors $\mathbf{v}, \mathbf{w} \in K^n$ by $\langle \mathbf{v}, \mathbf{w} \rangle$. In this thesis, the inner product is usually realized by the *dot product* $\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w} = v_1 w_1 + \cdots + v_n w_n$, where $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$. The *(Euclidean) norm* of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$ and $\|\mathbf{v} - \mathbf{w}\|$ consequently is the *(Euclidean) distance* between $\mathbf{v}$ and $\mathbf{w}$. For a matrix $\mathbf{M}$ we refer to the element in the i-th row and j-th column either by m_{ij} or $[\mathbf{M}]_{ij}$.

A *relation* on the sets $M_1, \dots, M_k$ is a subset $R \subseteq M_1 \times \cdots \times M_k$. A *binary relation* on M is a relation $\sim \subseteq M^2$ and we write $a \sim b$ if and only if $(a, b) \in \sim$, and $a \not\sim b$ otherwise. An *equivalence relation* is a binary relation on M that is

(i) reflexive, i.e., $\forall x \in M : x \sim x$,
(ii) symmetric, i.e., $\forall x, y \in M : x \sim y \Longrightarrow y \sim x$, and

(iii) transitive, i.e., $\forall x, y, z \in M : x \sim y \land y \sim z \Longrightarrow x \sim z$.

The *equivalence class* of $x \in M$ under an equivalence relation $\sim$ is defined as $[x]_\sim = \{y \in M \mid x \sim y\}$; the set of equivalence classes forms a partition of M. A binary relation that is symmetric and transitive, but not necessarily reflexive, is called *partial equivalence relation.*

A *function* from X to Y is a relation $f \subseteq X \times Y$ such that for every element $x \in X$ there is exactly one element $y \in Y$ with $(x, y) \in f$. In this case, we use the notation $f : X \to Y$ instead of $f \subseteq X \times Y$, and $f(x) = y$ instead of $(x, y) \in f$. Given a function $f : X \to Y$, we refer to the *domain* of f by $\mathrm{dom}(f) = X$ and the *codomain* of f by $\mathrm{codom}(f) = Y$. The *image* of a subset $A \subseteq X$ under f is defined as $f(A) = \{y \in Y \mid \exists x \in A : f(x) = y\}$; the *image* $\mathrm{img}(f)$ of the function f is $f(X)$. A function $f : X \to Y$ is

(i) an *injection* if $f(a) = f(b) \Longrightarrow a = b$ holds for all $a, b \in X$,

(ii) a *bijection* if it is an injection and $\mathrm{img}(f) = \mathrm{codom}(f)$.

A *partial function* f from X to Y, written $f : X \rightsquigarrow Y$, is a function $f : X' \to Y$ for some $X' \subseteq X$; in this case we have $\mathrm{dom}(f) = X'$.

2.2 Graphs

Unless explicitly stated otherwise throughout this thesis we consider simple undirected graphs according to the following definition; the set of all such graphs is denoted by $\mathcal{G}$.

Definition 2.1 (Graph). *A* graph $G = (V, E)$ *consists of a finite set* $V(G) = V$ *of* vertices *and a finite set* $E(G) = E \subseteq [V]^2$ *of* edges.

For convenience, in the following an edge $\{u, v\}$ is denoted by uv or vu, both refer to the same edge. Two vertices u and v are said to be *adjacent* if $uv \in E$ and referred to as *endpoints* of the edge uv. The edge uv is said to be *incident* to its endpoints; two edges are *adjacent* if they share one endpoint. The *degree* $\deg(v)$ of a vertex v is the number of edges incident to v; a vertex v with $\deg(v) = 0$ is said to be *isolated.* We denote the *maximum degree* of G by $\Delta(G) = \max\{\deg(v) \mid v \in V(G)\}$. The vertices adjacent to a vertex v are denoted by $\mathrm{N}(v) = \{u \in V \mid uv \in E\}$ and referred to as *neighbors of* v. The *order* of a graph G is its number of vertices and denoted by $|G| = |V(G)|$. The *size* of a graph G is its number of edges and denoted by $\|G\| = |E(G)|$. A graph G with $|G| = 0$ is said to be *empty.* An *adjacency matrix* of a graph G of order n is a symmetric matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ with $a_{ij} = 1$ if $v_i v_j \in E(G)$ and 0 otherwise, for some numbering of vertices. A *walk* is a sequence of vertices $(v_0, \ldots, v_n)$ such that $v_i v_{i+1} \in E$ for $0 \leq i < n$. The vertices and the edges connecting consecutive vertices are said to be *contained* in the walk. The *length* of a walk is the number of edges it contains. A walk with no repeated vertices is called *path.* A *cycle* is a walk of length at least 3 with no repeated vertices except $v_0 = v_n$. A *chord* is an edge e connecting two vertices of a cycle that does not contain the edge e. A cycle is called *chordless* if it does not possess a chord. We say a graph is a path (cycle) if there is a path (cycle) containing all the vertices and edges of the graph. Let $G = (V, E)$ be a graph, the graph $\overline{G} = (V, [V]^2 \setminus E)$ is referred to as *complement graph* of G.

2.2.1 Multigraphs and Digraphs

A *multigraph* G is a generalization of Definition 2.1 where the edge set $E(G)$ and every edge $e \in E(G)$ are multisets. Multisets may contain the same element multiple times. An edge $\{v, v\}$ is called a *self-loop* and two edges are said to be *parallel* if they connect the same endpoints.

A *directed graph* or *digraph* $G = (V, E)$ consists of a finite set V of vertices and a finite set $E \subseteq V \times V$ of edges. We denote an edge (u, v) of a digraph as uv, for short. Note that in contrast to graphs, for digraphs uv and vu refer to different edges. For an edge uv the vertices u and v are referred to as *source vertex* and *target vertex*, respectively; uv is an *incoming edge* of v and an *outgoing edge* of u. The *in-degree* $\deg^-(v)$ of a vertex v is the number of incoming edges of v, the *out-degree* $\deg^+(v)$ the number of outgoing edges. We denote the *maximum in-degree* of a digraph G by $\Delta^-(G) = \max\{\deg^-(v) \mid v \in V(G)\}$ and the *maximum out-degree* by $\Delta^+(G) = \max\{\deg^+(v) \mid v \in V(G)\}$. The definition of adjacency matrices carries over from graphs to digraphs, for which the resulting matrix is not necessarily symmetric. A *(directed) walk* from a *start vertex* s to an *end vertex* t is a sequence of vertices $(s = v_0, \ldots, v_n = t)$ such that $v_i v_{i+1} \in E$ for $0 \leq i < n$. A *(directed) path* is a directed walk with no repeated vertices. A directed walk of length at least 2 with no repeated vertices except $s = t$ is a *(directed) cycle*. A digraph without directed cycles is called *acyclic*. A digraph G that contains an edge vu for every edge $uv \in E(G)$ can be interpreted as an undirected graph. In this sense digraphs generalize graphs according to Definition 2.1.

2.2.2 Subgraphs

The concept of subgraphs transfers the inclusion relation of sets to the domain of graphs. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$, written $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. If G' is a subgraph of G, then G is said to be a *supergraph* of G'. A subgraph $G' \subseteq G$ is said to be *proper* if $G' \neq G$ and we write $G' \subset G$. Let $V' \subseteq V$ and $E' = \{uv \in E \mid u, v \in V'\}$, then $G' = (V', E')$ is said to be a *(vertex-)induced* subgraph of G denoted by $G' \sqsubseteq G$. We say that V' *induces* G' in G and refer to the induced subgraph by $G[V']$. Let $E' \subseteq E$ and V' be the set of vertices that appear as endpoint of at least one edge in E', then $G' = (V', E')$ is said to be an *edge-induced* subgraph of G. We say E' *induces* G' in G and refer to the induced subgraph by $G[E']$. Edge-induced subgraphs do not contain isolated vertices and, vice versa, every subgraph not containing isolated vertices is an edge-induced subgraph for some subset of edges. A graph G is called *minimal* (*maximal*) with some property if G itself has the property, but no proper subgraph $G' \subset G$ (supergraph $G' \supset G$) does. A subgraph $G' \subseteq G$ is said to be a *spanning subgraph* of G if $V(G') = V(G)$. A subset of edges $E' \subseteq E$ is said to *span* $G = (V, E)$ if $G[E']$ is a spanning subgraph of G.

2.2.3 Separators and Connectivity

A graph is *connected* if at least one path between any two vertices exists and is *disconnected* otherwise. A *(connected) component* of G is a maximal connected subgraph of G. We write $G \setminus S$ for the graph $G[V(G) \setminus S]$ obtained from G by deleting the vertices $S \subseteq V(G)$. For a subset of edges $T \subseteq E(G)$ we define $G \setminus T = (V(G), E(G) \setminus T)$. A set $S \subseteq V(G)$ is called $|S|$*-separator* or *separator* of a graph G if $G \setminus S$ is disconnected. A graph G with $|G| > k$ is called k*-connected* if it does not contain a j-separator with $j < k$; a 2-connected graph is also called *biconnected.* A *block* is a maximal biconnected subgraph of G and a *bridge* is an edge that is not contained in a block. If $\{v\}$ is a separator then v is called *cutvertex.* Any two blocks and bridges of G may have at most one vertex in common, which is a cutvertex. A graph G is said to have connectivity $\kappa(G) = k$ if it is k-connected, but not $(k+1)$-connected. A separator S is called (a, b)*-separator* if $G \setminus S$ contains two distinct connected components C and D, such that $a \in V(C)$ and $b \in V(D)$; an (a, b)-separator S is said to *separate* a from b. A separator S is said to separate $A \subseteq V(G)$ from $B \subseteq V(G)$ if S is an (a, b)-separator for all $a \in A$ and $b \in B$. A separator S is called *minimal* if there are vertices $a, b \in V(G)$, such that S is an (a, b)-separator, but there is no (a, b)-separator S' with $S' \subset S$. Note that a minimal separator may be contained in another minimal separator. A separator S is said to *cross* another separator T if $G \setminus T$ contains components C, D such that $S \cap V(C) \neq \emptyset$ and $S \cap V(D) \neq \emptyset$. In other words: S crosses T if there are vertices $a, b \in S$ that are separated by T. Let S, T be minimal separators, then S crosses T if and only if T crosses S (Parra and Scheffler, 1995, Lemma 3). Two non-crossing separators are said to be *parallel.*

2.2.4 Specific Graph Classes

The subset of graphs that have a special property are referred to as *graph class.* We briefly review some graph classes that are relevant for this thesis and refer the reader to the textbook by Brandstädt et al. (1999) for a comprehensive survey. A graph is called *complete* if every pair of distinct vertices is adjacent. The complete graph on n vertices is denoted by K_n. Given a graph $G = (V, E)$, two disjoint sets $U \subseteq V$ and $W \subseteq V$ with $U \uplus W = V$ form a *bipartition* of G if there is no edge $uv \in E$ with u and v both in U or both in W. A graph G is called *bipartite* if a bipartition of G exists and we use the notation $G = (U, W, E)$ to explicitly refer to the bipartition U, W. The *complete bipartite* graph on $n = |U|$ and $m = |W|$ vertices is the graph with bipartition U, W such that for every $u \in U$ and $v \in W$ the edge uv exists;

this graph is denoted by $K_{n,m}$. A *forest* is a graph containing no cycles. A *(free) tree* is a connected forest. A *rooted tree* is a tree T together with a distinguished vertex $r \in V(T)$. The vertex u is the *parent* of v if $(v, u, \ldots, r)$ is a path in T with root r. If u is the parent of v, then v is referred to as *child* of u.

The following concepts intuitively allow to measure how "tree-like" a graph is. A graph G is called a *k-almost tree* if $\|B\| < |B| + k$ is satisfied for every block B in G. The 1-almost trees are known as *cactus graphs* (Brandstädt et al., 1999).

Definition 2.2 (Tree Decomposition). *A* tree decomposition *of a graph G is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X} = (X_i)_{i \in V(T)}$ a family of vertex subsets $X_i \subseteq V(G)$ called* bags *satisfying:*

T1 $V(G) = \bigcup_{i \in V(T)} X_i$,

T2 *for every edge $uv \in E(G)$ there is a node $i \in V(T)$ with $\{u, v\} \subseteq X_i$,*

T3 *for every $i, j, k \in V(T)$, if j lies on the unique path with endpoints i and k then $X_i \cap X_k \subseteq X_j$.*

We refer to the vertices of the tree T as *nodes* to distinguish them from the vertices of the input graph. The *width* of a tree decomposition $(T, \mathcal{X})$ is defined as $\max\{|X_i| - 1 \mid i \in V(T)\}$ and the *tree width* $\operatorname{tw}(G)$ of a graph G is the minimum width among all possible tree decompositions of G. The graphs with tree width less or equal to k are also known as *partial k-trees.* A crucial feature of tree decompositions is their close relation to graph separators: Let $(T, \mathcal{X})$ be a tree decomposition of a connected graph G and ab any edge in T. Let T_a and T_b be the connected components of $T \setminus \{ab\}$ and $A = \bigcup_{i \in V(T_a)} X_i$, $B = \bigcup_{i \in V(T_b)} X_i$, then either (i) the intersection of the bags $X_a \cap X_b = S$ separates $A \setminus S$ from $B \setminus S$ in G; or (ii) $A \subseteq B$ or $B \subseteq A$. Therefore, $\kappa(G) \leq \operatorname{tw}(G)$ and the equation $\kappa(G) = \operatorname{tw}(G) = k$ is satisfied for k-connected partial k-trees only. The partial 2-trees are also called *series-parallel* graphs.

A graph is *planar* if it admits a *planar drawing*, i.e., a drawing in the plane such that no two edges cross. A planar drawing of a graph divides the plane into connected regions called *faces*, which are enclosed by edges. The unbounded region is referred to as the *outer face.* An edge and a face are said to be *incident* if the edge lies on the boundary of the face. Two faces are *adjacent* if they are incident with a common edge. A *combinatorial embedding* of a graph specifies for every vertex the cyclic, clockwise order of all incident edges. A planar drawing defines a combinatorial embedding and every planar drawing that defines the same combinatorial embedding yields the same set of faces. A *planar embedding* is a combinatorial embedding with a distinguished outer face. A planar graph G with at least three vertices has at most $3|G| - 6$ edges (Diestel, 2005, Corollary 4.2.10). A graph is called *outerplanar* if it admits a planar drawing, in which every vertex lies on the boundary of the outer face. The associated planar embedding is referred to as *outerplanar embedding.* Every outerplanar graph is series-parallel, but, vice versa, not every series-parallel graph is outerplanar.

The following concepts are essential to characterize graph classes by forbidden substructures. An edge uv is *subdivided* by replacing uv with a new vertex w and two edges uw and wv. The graph M is a *topological minor* of G if a subgraph of G can be obtained[1] by subdividing edges of M. An edge uv is *contracted* by merging its endpoints into a new vertex w, which becomes adjacent to all the former neighbors of u and v. A graph M is a *minor* of a graph G if M can be obtained from G by successively (i) contracting edges, (ii) deleting edges and (iii) deleting isolated vertices. If M is a topological minor of a graph G, then M also is a minor of G. Hence, the class of graphs that do not contain M as a topological minor is at least as general as the class of graphs that do not contain M as a minor.

We provide alternative characterizations of some of the graph classes defined above: A graph is a cactus if and only if it does not contain the complete graph on four vertices missing one edge as minor (El-Mallah and Colbourn, 1988). A graph is outerplanar if and only if it contains neither K_4 nor $K_{2,3}$ as a minor (Brandstädt et al., 1999; Diestel, 2005); it is series-parallel if and only if it does not contain K_4 as a minor (Diestel, 2005, Proposition 12.4.2). Finally, a graph is planar if and only if it contains neither K_5 nor $K_{3,3}$ as a minor (Diestel, 2005, Theorem 4.4.6). This famous result was originally shown for topological minors by Kuratowski (1930) and later for minors by Wagner (1937).

2.3 Isomorphism Problems in Graphs

In this section we introduce basic concepts to describe that graphs exhibit the same structure or substructures. We briefly review the complexity status and algorithms for the related computational problems.

A graph isomorphism is a mapping of vertices that preserves adjacencies.

Definition 2.3 (Graph Isomorphism). *Let G and H be two graphs, a bijection $\psi : V(G) \to V(H)$ is a* (graph) isomorphism *if*

$$uv \in E(G) \iff \psi(u)\psi(v) \in E(H) \qquad \forall u, v \in V(G). \tag{2.1}$$

Two graphs G and H are said to be isomorphic*, written $G \simeq H$, if an isomorphism between G and H exists.*

Note that an isomorphism implicates a bijection between the edges of a graph. An *automorphism* of a graph G is an isomorphism $\psi : V(G) \to V(G)$. The set of automorphisms of G is denoted by $\mathrm{Aut}(G)$. Definition 2.3 leads to the following well-studied decision problem.

[1]More precisely a graph “with the same structure” must be obtained according to the concept of isomorphism introduced later by Definition 2.3.

Problem 2.1 (GI). Graph Isomorphism

Input: *Two graphs G and H.*

Task: *Decide if $G \simeq H$ holds.*

GI is one of few problems belonging to NP, which has neither been shown to be NP-complete nor polynomial-time solvable (Garey and Johnson, 1979). Although the problem has been extensively studied (see, e.g., Read and Corneil, 1977; Arvind and Torán, 2005, and references therein), the complexity status still remains open (Johnson, 2005; Grohe, 2013). For many graph classes the problem can be solved in polynomial time (Köbler, 2006): These include planar graphs (Hopcroft and Wong, 1974), partial k-trees (Bodlaender, 1990) and, more generally, graphs with forbidden minors; polynomial-time algorithms are also known for graphs of bounded degree (Luks, 1982). All these previous results were recently unified and generalized by Grohe and Marx (2015) for graphs with forbidden topological minors. In practice, the problem can be solved fairly well (McKay, 1981; McKay and Piperno, 2014) and difficult instances are often hard to find and generated synthetically. For general graphs the best known algorithm for more than three decades obtains a running time of $\exp(\mathcal{O}(\sqrt{n \log n}))$ for graphs of order n as reported by Babai et al. (1983). The method is based on the approach for graphs of bounded degree by Luks (1982) and employs a (color) degree reduction technique due to Zemlyachenko et al. (1985). Hence, GI can be solved in *moderately-exponential time*, meaning that the running time is bounded by $\exp(\mathcal{O}(n^\alpha))$ with $\alpha < 1$. A problem is said to be GI-hard if it is at least as hard as graph isomorphism and GI-complete if it is as hard as graph isomorphism w.r.t. polynomial-time reduction (for details, see Booth and Colbourn, 1979). The graph isomorphism problem, for example, remains GI-complete when the input graphs are restricted to connected graphs or bipartite graphs. Isomorphism problems for other classes of mathematical objects like automata and semigroups and certain combinatorial problems are also known to be GI-complete (Booth and Colbourn, 1979).

A *graph invariant* is a function I on $\mathcal{G}$, such that $G \simeq H \Longrightarrow I(G) = I(H)$. In the case that I not only provides a necessary, but also sufficient condition for graph isomorphism, i.e., $G \simeq H \Longleftrightarrow I(G) = I(H)$, the graph invariant I is said to be *complete*. The *canonical form* $C(G)$ of a graph G is a unique adjacency matrix of G such that $G \simeq H \Longleftrightarrow C(G) = C(H)$ and, hence, C is a complete invariant. Gurevich (1997) showed that every complete invariant can be used to obtain a canonical form. The above definition leads to the following well-studied computational problem closely related to GI.

Problem 2.2 (CAN). Graph Canonization

Input: *A graph G.*

Task: *Compute a complete graph invariant $C(G)$.*

Clearly, CAN is GI-hard, while it is an open problem if CAN is polynomial time reducible to GI (Arvind and Torán, 2005). For the above mentioned graph classes for which polynomial-time GI algorithms are known CAN can as well be solved in polynomial time. In fact, GI algorithms often compute a complete graph invariant in order to test graphs for isomorphism. This also is the case for the most general approach by Grohe and Marx (2015) as well as for most practical algorithms (McKay and Piperno, 2014). The best known algorithm for general graphs was proposed by Babai and Luks (1983) and achieves a moderately-exponential running time asymptotically bounded by $\exp(n^{1/2+o(1)})$, where n is the order of the input graph. The algorithm is based on the same approach used by Babai et al. (1983) to solve GI and closely matches its running time (Arvind and Torán, 2005). Canonization is of high practical relevance since it allows, for example, to easily test if a graph is contained in a large database of graphs, when these are stored in canonical form.

An (induced) subgraph isomorphism from G to H is an isomorphism between G and an (induced) subgraph H' of H.

Definition 2.4 (Subgraph Isomorphism). *Let G and H be two graphs. An injection $\psi : V(G) \to V(H)$ is a* subgraph isomorphism *from G to H if*

$$uv \in E(G) \Longrightarrow \psi(u)\psi(v) \in E(H) \qquad \forall u, v \in V(G). \tag{2.2}$$

Definition 2.5 (Induced Subgraph Isomorphism). *Let G and H be two graphs. An injection $\psi : V(G) \to V(H)$ is an* induced subgraph isomorphism *from G to H if*

$$uv \in E(G) \Longleftrightarrow \psi(u)\psi(v) \in E(H) \qquad \forall u, v \in V(G). \tag{2.3}$$

Clearly, each induced subgraph isomorphism also is a subgraph isomorphism, but not vice versa. Again, there are well-studied computational problems related to the two above definitions.

Problem 2.3 (SI). Subgraph Isomorphism

Input: *Two graphs G and H.*

Task: *Decide if a subgraph isomorphism from G to H exists.*

Problem 2.4 (ISI). Induced Subgraph Isomorphism

Input: *Two graphs G and H.*

Task: *Decide if an induced subgraph isomorphism from G to H exists.*

In the context of subgraph isomorphism problems as above we refer to G as the *pattern graph.* If G and H are of the same order and size, SI and ISI algorithms decide if G and H are isomorphic. However, several well-known NP-complete problems can also be reduced in polynomial time to subgraph isomorphism. This includes deciding if a clique of size k (cf. the related optimization Problem 2.6) or a Hamiltonian cycle exists in a graph. Thus, SI and ISI are NP-complete for general graphs even if the pattern graph has a very special structure, i.e., it either is complete or a cycle. The complexity has been studied in detail. Let $k = |G|$ be the order of the pattern graph and $n = |H|$ the order of H. If the pattern graph G is a tree and H is a forest, SI and ISI can be solved in time $\mathcal{O}\left(k^{1.5}n/\log k\right)$ (Matula, 1978; Reyner, 1977; Verma and Reyner, 1989; Shamir and Tsur, 1999). Vice versa, if G is a forest and H is a tree, the problems are NP-complete (Garey and Johnson, 1979). SI is NP-complete if G and H are outerplanar even when G is connected (Sysło, 1982; Lingas, 1989). If G is biconnected and both input graphs are outerplanar, ISI can be solved in time $\mathcal{O}(n^2)$ (Sysło, 1982) and SI in $\mathcal{O}(kn^2)$ (Lingas, 1989). Lingas and Sysło (1988) showed that SI can be solved in time $\mathcal{O}(n^3 + k^{4.5}n^2)$ if G is biconnected and both input graphs are series-parallel. SI can be solved in time $\mathcal{O}(k^{k+1}n)$ if G is a connected, bounded-degree graph and H is a partial k-tree (Matoušek and Thomas, 1992). Generalizing the result for bounded-degree graphs Hajiaghayi and Nishimura (2007) showed that polynomial running time time can still be achieved when G is a *log-bounded fragmentation graph*, i.e., the removal of at most k vertices results in $\mathcal{O}(k \log n)$ connected components, where $n = |G|$. SI can be solved in time $\mathcal{O}(k^{k+1}n^{3.5})$ if G is k-connected and H is a partial k-tree (Matoušek and Thomas, 1992; Gupta and Nishimura, 1994). Dessmark et al. (2000) improved the running time to $\mathcal{O}(n^{k+2})$ for arbitrary k and $\mathcal{O}(n^{3.5})$ for $k \in \{2, 3\}$. If G and H are partial k-paths[2] SI can be solved in time $\mathcal{O}(n^3)$ for arbitrary k (Gupta and Nishimura, 1996a). On the contrary, Gupta and Nishimura (1996a;b) proved that SI on partial k-trees is NP-complete if G is not k-connected or has more than k vertices of unbounded degree. Eppstein (1999) showed that SI (as well as ISI) on planar graphs can be solved in $\mathcal{O}(n)$ for fixed k. The superexponential dependency on k of the SI test can be reduced from $k^{\mathcal{O}(k)}$ to $2^{\mathcal{O}(k)}$ (Dorn, 2010).

In practice, backtracking algorithms for general graphs are often employed successfully. These approaches typically maintain (either explicit or implicit) a set of candidate vertices of H for every vertex of G and extend a mapping step-by-step performing backtracking if the current mapping cannot be completed. Whenever the mapping is extended the candidate sets are filtered to some extent. Further the ordering, in which the mapping is extended turned out to be crucial in practice. The early algorithm by Ullmann (1976) uses elaborate candidate filtering referred to as *refinement.* Later computational less demanding filtering techniques in combination with certain vertex orderings were shown to be more efficient for practical instances. Cordella et al. (1999) proposed the VF algorithm and experimentally showed it to outperform Ullmann's algorithm on several synthetically generated in-

[2]A graph is a *partial k-path* if it admits a tree decomposition $(T, \mathcal{X})$ of width at most k, where the tree T is a path.

stances (Cordella et al., 1999). The approach subsequently was improved, in particular with respect to memory requirements, referred to as VF2 (Cordella et al., 2001), and shown to be efficient in an extensive experimental comparison (Foggia et al., 2001; Cordella et al., 2004). For molecular graphs Ullmann's algorithm has been considered most convenient for years (Barnard, 1993; Willett, 1999). More recently, it was shown that straight-forward approaches similar to VF2 not managing candidate sets explicitly are highly efficient for molecular graphs in practice (Kriege, 2009; Klein et al., 2011). Subgraph isomorphism was considered in the context of constraint programming (see, e.g., Zampelli, 2008, and references therein) and recently less demanding filtering techniques were again proposed by Ullmann (2011) and combined with efficient bit-vector representations.

Definition 2.6 (Common Subgraph (Isomorphism)). *Let G and H be two graphs, an isomorphism ψ between $G' \subseteq G$ and $H' \subseteq H$ is called* common subgraph isomorphism *of G and H; the graph $G' \simeq H'$ is a* common subgraph *of G and H. The isomorphism ψ induces the subgraph isomorphisms $\psi_G : V(H') \to V(G)$ from H' to G and $\psi_H : V(G') \to V(H)$ from G' to H.*

Note that a common subgraph isomorphism is a partial function $\psi : V(G) \rightsquigarrow V(H)$ w.r.t. to the vertices of G. We say a common subgraph (isomorphism) is *induced* if G' and H' both are induced subgraphs; it is a *common edge subgraph (isomorphism)* if both are edge-induced subgraphs. A common subgraph isomorphism is said to be *maximal* if it cannot be extended. Note that a common subgraph isomorphism may be a maximal common induced subgraph isomorphism and at the same time a common edge subgraph isomorphism that is not maximal. Clearly, in general a common subgraph may be connected or disconnected. Combining these properties leads to several variants of common subgraph problems, which differ regarding their complexity. The related optimization problems are discussed in detail in Chapter 3.

2.3.1 Isomorphism Problems in Labeled Graphs

In practice graphs typically are used to model complex structures like chemical compounds or proteins. Then vertices and edges represent real-world objects like atoms and molecular bonds and must typically be annotated by additional information like the atom or bond type.

Definition 2.7 (Labeled Graph). *Let $\mathcal{L} = \mathcal{L}_V \cup \mathcal{L}_E$ be a finite set of* labels*, a* labeled graph *is a graph G equipped with a function $\tau : V(G) \uplus E(G) \to \mathcal{L}$, which assigns a* vertex label *$\tau(v) \in \mathcal{L}_V$ to every vertex $v \in V(G)$ and an* edge label *$\tau(e) \in \mathcal{L}_E$ to every edge $e \in E(G)$.*

We refer to $\mathcal{L}$ as *label alphabet*, which could be, e.g., $\mathcal{L} = \{\texttt{red}, \texttt{green}, \texttt{blue}\}$ or the possible atom and bond types of a molecular graph. The elements of these sets can typically not be naturally ordered and we refer to them as *categorical* or *discrete* labels in order to distinguish them from the more general annotations of attributed graphs introduced later in Section 4.3, which can be continuous and multi-dimensional.

For two graphs G, H with labels τ_1 and τ_2 a bijection $\psi : V(G) \to V(H)$ is said to *preserve labels* if

$$\tau_1(v) = \tau_2(\psi(v)) \qquad \forall v \in V(G) \tag{2.4}$$

$$\tau_1(uv) = \tau_2(\psi(u)\psi(v)) \qquad \forall uv \in E(G). \tag{2.5}$$

Two labeled graphs G and H are isomorphic if there is a bijection $\psi : V(G) \to V(H)$ that (i) preserves adjacencies according to Equation (2.1), Definition 2.3; and (ii) simultaneously preserves labels according to Equations (2.4), (2.5). A graph G' with labels τ' is a subgraph of a graph G with labels τ if $G' \subseteq G$ and τ' is the function τ restricted to the domain $V(G') \uplus E(G')$. The definitions of vertex- and edge-induced subgraphs as well as the definition of the various isomorphism problems can directly be transferred to labeled graphs. This includes automorphism, (induced) subgraph isomorphism and common (induced) subgraph isomorphism. When applied to labeled graphs, we in general assume that labels must be preserved by isomorphisms. Occasionally, we refer to Definition 2.3 even for labeled graphs as *isomorphism between the underlying unlabeled graphs.*

Research with a focus on theoretical results typically does not consider labels. For graphs with uniform labels, Equations (2.4), (2.5) always hold and we obtain the ordinary isomorphism problems. Hence, negative complexity results for isomorphism problems on unlabeled graphs carry over to labeled graphs. Notably, the NP-hardness proof by Akutsu and Tamura (2012) for a common subgraph problem in degree-bounded partial k-trees requires graphs to be labeled, but the authors conjecture that the result holds as well for unlabeled graphs. Vice versa, the proposed polynomial-time algorithms can typically be adjusted to take labels into account. Moreover, in practice, the presence of labels often reduces the search space and can even be used algorithmically to improve running times (Kriege, 2009; Klein et al., 2011).

2.4 Fundamental Graph Problems and Algorithms

A *matching* in a graph G is a subset of edges $M \subseteq E(G)$ such that no two edges in M share a common vertex, i.e., $e \cap e' = \emptyset$ for all distinct edges $e, e' \in M$. A matching is *perfect* if every vertex of G is incident to exactly one edge in M.

Problem 2.5 (MWBM). Maximum Weight Bipartite Matching

Input: *A bipartite graph $G = (U, V, E)$ with edge weights $w : E \to \mathbb{R}$.*

Task: *Determine the maximum weight $w(M) = \sum_{e \in M} w(e)$ of a matching $M \subseteq E$ in G.*

For convenience, we occasionally allow the edge weight $-\infty$. Note that a maximum weight matching does not contain any edge e with weight $w(e) < 0$ and we may delete all edges with non-positive weight without affecting the optimal solution. The Maximum Weight Bipartite Perfect Matching (MWBPM) is defined analogous to Problem 2.5, but asks for the maximum weight of a perfect matching (provided that the input graph admits a perfect matching). MWBM can be reduced to MWBPM by constructing a graph that contains two copies of the MWBM instance and additional edges with weight zero between every two copies of the same vertex (Duan and Su, 2012). The Assignment problem asks for the minimum weight of a perfect matching. By negating all weights of an MWBPM instance, we can transform it into an equivalent assignment problem and vice versa. The assignment problem can be solved in time $\mathcal{O}(n^3)$ by an elaborated implementation of the Hungarian method, where G is assumed to be a complete bipartite graph of order n. Fredman and Tarjan (1987) obtained a running time of $\mathcal{O}(n^2 \log n + nm)$, where $n = |G|$ and $m = \|G\|$, by employing Fibonacci heaps in combination with Dijkstra's algorithm to solve the subproblem of finding shortest paths. In case of integral edge weights from the set $\{-N, \dots, N\}$ the assignment problem can be solved in $\mathcal{O}(\sqrt{n}m \log(nN))$ by means of scaling techniques (Gabow and Tarjan, 1989). The best known scaling algorithm for MWBM, Problem 2.5, runs in $\mathcal{O}(m\sqrt{n} \log N)$ time (Duan and Su, 2012). The Maximum (Cardinality) Bipartite Matching problem is a special case of Problem 2.5, where all edges have weight one. The classical algorithm by Hopcroft and Karp (1973) solves this problem in $\mathcal{O}(m\sqrt{n})$ time.

Given a graph $G = (V, E)$, a *clique* is a subset of vertices $C \subseteq V$ such that $G[C]$ is a complete graph. A clique C is said to be *maximal* if there is no clique C' with $C \subset C'$ and *maximum* if there is no clique C' with $|C| < |C'|$. The following Problem 2.6 is known to be NP-complete (Garey and Johnson, 1979).

Problem 2.6 (CLIQUE). Maximum Clique

Input: *A graph G.*

Task: *Determine the size of a maximum clique in G.*

Given a graph $G = (V, E)$, a *vertex cover* is a subset of vertices $C \subseteq V$ such that for all edges $uv \in E$ either $u \in C$ or $v \in C$ or both. A vertex cover C is said to be *minimal* if there is no vertex cover C' with $C' \subset C$. A vertex cover C is said to be *minimum* if there is no vertex cover C' with $|C'| < |C|$.

Problem 2.7 (VC). Minimum Vertex Cover

Input: *A graph G.*

Task: *Determine the size of a minimum vertex cover in G.*

Given a graph $G = (V, E)$, an *independent set* is a subset of vertices $S \subseteq V$ such that $G[S]$ contains no edges, i.e., $\forall u, v \in S : uv \notin E$. An independent set S is said to be *maximal* if there is no independent set S' with $S \subset S'$. An independent set S is said to be *maximum* if there is no independent set S' with $|S| < |S'|$.

Problem 2.8 (IS). Maximum Independent Set

Input: *A graph G.*

Task: *Determine the size of a maximum independent set in G.*

The above problems are closely related: The set $C \subseteq V(G)$ is a vertex cover in G if and only if the set $S = V(G) \setminus C$ is an independent set in G. Furthermore, C is a minimal (minimum) vertex cover if and only if S is a maximal (maximum) independent set. The set $S \subseteq V$ is an independent set in G if and only if S is a clique in $\overline{G}$. Furthermore, S is a maximal (maximum) independent set in G if and only if S is a maximal (maximum) clique in $\overline{G}$.

2.5 Notation

Notation and symbols used throughout this thesis are summarized in Tables 2.1, 2.2, 2.3. The symbols presented in Table 2.3 are introduced in Chapter 4 and listed here for completeness.

Table 2.1: General notation and symbols.

Symbol	Meaning
$\mathbb{N}$, $\mathbb{N}^+$	Natural numbers with and without zero
$\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{R}^+$	All, non-negative, positive real numbers
$M_1 \uplus M_2$	Disjoint union of sets M_1 and M_2
$\mathcal{P}(M)$	Power set of M
$[M]^k$	Subsets of the set M with k-element
$v_i = [\mathbf{v}]_i$	Element at position i of vector $\mathbf{v}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	Dot product
$\lVert \mathbf{v} \rVert$, $\lVert \mathbf{v} - \mathbf{w} \rVert$	Norm and distance
$m_{ij} = [\mathbf{M}]_{ij}$	Element at (i, j) of matrix $\mathbf{M}$
$\mathrm{dom}(f)$	Domain of function f
$\mathrm{codom}(f)$	Codomain of function f
$\mathrm{img}(f)$	Image of function f

Table 2.2: Notation and symbols for graphs.

Symbol	Meaning
$\mathcal{G}$	Set of graphs
$\lvert G \rvert$	Order of G, i.e., $\lvert V(G) \rvert$
$\lVert G \rVert$	Size of G, i.e., $\lvert E(G) \rvert$
$\deg(v)$	Degree of vertex v
$\Delta(G)$	Maximum degree of graph G
$\mathrm{N}(v)$	Neighbors of v
$\overline{G}$	Complement graph of G
$G[V]$	Subgraph induced by V
$\subseteq$, $\sqsubseteq$	Subgraph, induced subgraph relation
$G \setminus S$	Graph obtained by deleting S
$\kappa(G)$	Connectivity of G
K_n, $K_{n,m}$	Complete and complete bipartite graph
$\mathrm{tw}(G)$	Treewidth of G
ψ, $\simeq$	Isomorphism, isomorphism relation
$\mathrm{Aut}(G)$	Automorphisms of G
$\mathcal{L}, \tau$	Label alphabet, labeling function
$\mathcal{A}, \alpha$	Attributes, attributing function
$G \times H$, $\times_w$	Direct product graph, weighted —
$G \nabla H$, ∇_w	Association graph, weighted —

Table 2.3: Notation and symbols for kernels.

Symbol	Meaning
K, k	Kernel functions
K_{RBF}	Gaussian RBF kernel
K_δ	Dirac kernel
$\hat{K}$	Normalized kernel
$\mathcal{X}$	Objects compared by kernels
$\mathcal{D}$	Data set (of graphs)
κ_V, κ_E	Vertex, edge kernel
$\mathcal{H}$	Feature space
ϕ	Feature map $\mathcal{X} \to \mathcal{H}$
Φ	Feature vector representing an element in $\mathcal{H}$

Gemeinsamkeit ist immer etwas
Hinzugekommenes, und wir
wissen nicht was.

Hermann Bahr
(1863–1934)

Chapter 3

Common Subgraph Problems

In this chapter common subgraph problems and closely related variations are considered. The most elementary problem of that kind arguably is the following.

Problem 3.1 (MCIS). Maximum Common Induced Subgraph

Input: *Two graphs G and H.*

Task: *Determine the maximum order of a common induced subgraph of G and H.*

MCIS is well-known to be NP-hard[1] (Johnston, 1976). In Section 3.1 we summarize known bounds on the running time of exact exponential-time algorithms. In addition, we discuss a modification of a folklore algorithm which improves over the worst-case time bound of the fastest exact MCIS algorithm for general graphs. Subsequently polynomial-time solvable special cases are considered, which become tractable either by restricting to specific common subgraphs, by considering only specific graph classes or—most fruitful—a combination of both. After giving an overview on relevant algorithms and complexity results, we present an algorithm to compute the order of a maximum common biconnected subgraph of two series-parallel graphs. While this problem was believed to be solved, we identify key obstacles that were overlooked by previous approaches and present new solutions to overcome them. For the special case of outerplanar graphs we present a new algorithm, which is shown to yield quadratic running time. We extend our approach to solve a problem of high practical relevance, which asks to find a maximum common subgraph that preserves the structure of blocks and bridges of the input graphs. Following this, we discuss graph distance metrics derived from maximum common subgraph problems. Finally, we summarize our results and point out promising directions of future research.

The main contributions of this chapter are the following:

[1]Please note that the common subgraph problems considered in this chapter are optimization problems. For convenience, we say that an optimization problem P is NP-hard if there is a polynomial-time reduction from an NP-complete decision problem to P.

- We present an exponential-time exact MCIS algorithm in Section 3.1. While not suitable for practical purpose, we obtain a worst-case bound on its running time that constitutes an improvement over the fastest algorithms currently known. This result has not been published previously.
- In Section 3.4 we propose an algorithm to find a maximum common biconnected subgraph in series-parallel graphs based on the novel concept of potential separators. Parts of this section have been published in (Kriege and Mutzel, 2014).
- For the same problem we present a different algorithm in Section 3.4.4 that is restricted to outerplanar graphs, where it achieves quadratic running time. This result is intended for publication as part of (Droschinsky, Kriege, and Mutzel, 2015).
- We consider a related problem, where the task is to find a maximum common connected subgraph that preserves the blocks and bridges of the input graphs. In Section 3.5 we present a polynomial-time algorithm for this problem in series-parallel graphs, which utilizes the algorithm developed in Section 3.4. The basic idea of the approach is due to the author of this thesis, the results were initially elaborated by Kurpicz (2014) as part of his Master's thesis and have also been published in (Kriege, Kurpicz, and Mutzel, 2014a). The algorithms and the text presented here were revised and widely rewritten by the author of this thesis. Parts of Section 3.5 have been submitted for publication to the *European Journal of Combinatorics*.
- In Section 3.6 we introduce the concept of triangle consistency as a sufficient condition for maximum common subgraph variants to allow derivation of graph distance metrics. In combination with our algorithms we obtain polynomial-time computable graph distance metrics for cactus graphs. The results of this section have not been published previously.

3.1 Exact Exponential-Time Algorithms

Considerable effort has been spent into developing and improving maximum common subgraph algorithms for practical purposes (see, e.g., Raymond and Willett, 2002; Raymond et al., 2002a;b; Conte et al., 2007; Ehrlich and Rarey, 2011). In addition, maximum common subgraph problems have been formulated as integer linear programs (see Manić et al., 2009; Bahiense et al., 2012; Piva and de Souza, 2012) and constraint satisfaction problems (Vismara and Valery, 2008), such that general purpose solvers can be applied. However, only few algorithms have been thoroughly analyzed in terms of running time.

We briefly summarize the fundamentals of a classical approach and then review recently proposed algorithms, for which better bounds on the worst-case running time have been shown. Finally, we propose a new algorithm, which is shown to obtain the best worst-case bound that is currently known for exact MCIS algorithms on general graphs.

3.1.1 Reduction to the Clique Problem

The most common approach in practice is to exploit the one-to-one correspondence between common induced subgraph isomorphisms of the input graphs and cliques in the association graph,[2] which was already described by Levi (1973).

Definition 3.1 (Association Graph). *For two graphs $G = (V, E)$, $H = (V', E')$, the* association graph *is denoted by $G\nabla H = (\mathcal{V}, \mathcal{E})$ and defined as*

$$\begin{aligned} \mathcal{V} &= V \times V' \\ \mathcal{E} &= \Big\{(u,u')(v,v') \in [\mathcal{V}]^2 \;\Big|\; u \neq v \wedge u' \neq v' \wedge \\ &\qquad (uv \in E \wedge u'v' \in E' \;\vee\; uv \notin E \wedge u'v' \notin E')\Big\}. \end{aligned}$$

Each vertex of the association graph represents a mapping of a vertex of G to a vertex of H. Two vertices in the association graph are adjacent if their corresponding vertex mappings are compatible, i.e., the involved vertices exhibit the same relation (adjacent or non-adjacent) in both graphs.

Levi (1973) has shown that two graphs G and H have a common induced subgraph with k vertices if and only if there is a clique with k vertices in $G\nabla H$. Further, there is a one-to-one correspondence between the common induced subgraph isomorphisms and cliques in the association graph. Let $C = \{(v_1, v'_1), \dots, (v_k, v'_k)\}$ be a clique in $G\nabla H$, $U = \{v_1, \dots, v_k\} \subseteq V(G)$ and $U' = \{v'_1, \dots, v'_k\} \subseteq V(H)$, then $\psi(v_i) = v'_i$, for $i \in \{1, \dots, k\}$, is an isomorphism between $G[U]$ and $H[U']$. Consequently, the maximum cliques in the association graph correspond to the isomorphisms between maximum common induced subgraphs. Note that there may be different isomorphisms acting on the same sets of vertices of the two input graphs.

This relation allows to reduce the maximum common induced subgraph problem (Problem 3.1) to the maximum clique problem (Problem 2.6). There is a multitude of algorithms devised for maximum clique detection, see the surveys by Pardalos and Xue (1994) and Bomze et al. (1999) for an overview. The algorithm by Bron and Kerbosch (1973) enumerates all maximal cliques

[2]Also commonly called *compatibility graph* (Koch, 2001; Durand et al., 1999) or *product graph* (Koch, 2001; Cazals and Karande, 2005). In the context of a more systematic study of graph products Hammack et al. (2011) refers to the association graphs as *weak modular product graph.*

and is often used in practice (see, e.g., Koch, 2001). If one is only interested in an arbitrary maximum clique or just the size of a maximum clique, *branch-and-bound* techniques are used to speed up the search. These essentially allow to ignore unfruitful branches in the search tree, whenever an upper bound of the possible solution size within the branch drops below the best known current solution. Consequently, these techniques benefit from heuristics to find a good solution early. One example of these algorithms is due to Wood (1997), which is employed by Stahl et al. (2005). Conte et al. (2007) performed an extensive experimental comparison including several clique detection algorithms (e.g., Balas and Yu, 1986). However, the study does not reveal clear evidence for one algorithm being preferable in practice. The theoretical fastest known maximum clique algorithm is due to Robson (2001) and requires time $\mathcal{O}(2^{0.249N}) = \mathcal{O}(1.1888^N)$ to find a maximum clique in a graph of order N.[3] The order of the association graph clearly is nm, where $n = |G|$ and $m = |H|$. Hence, the best known worst-case bound for this approach is $\mathcal{O}(2^{0.249nm})$.

3.1.2 Recent Algorithms

Suters et al. (2005) more recently proposed an exact algorithm termed *clique branching* to compute a maximum common induced subgraph, which exploits the specific structure of association graphs. It is observed that the complement of an association graph contains a large number of cliques. This fact is used algorithmically when computing the minimum vertex cover in this graph.[4] The method is shown to require $\mathcal{O}((m+1)^n)$ time. Moreover, the authors state that this worst-case bound is superior to those known for algorithms that where analyzed previously.

Motivated by that claim Akutsu and Tamura (2012) presented and analyzed straight-forward exponential-time algorithms for several variations of the maximum common subgraph problem. Algorithm 3.1 shows the general approach, which simply enumerates all induced subgraphs of the two input graphs and checks all pairs for isomorphism—basically implementing a naive brute-force approach. Let $T_{\simeq}(n)$ be the time to solve the graph isomorphism problem between two induced subgraphs of order n, cf. line 4. The number of induced subgraphs corresponds to the number of subsets of the vertices, which is 2^n for a set of n vertices. Let n, m be the order of the two input graphs as above and assume w.l.o.g. that $n \leq m$. Algorithm 3.1 then requires

[3]The algorithm by Robson (2001) has not been published and is difficult to follow. Notably, there is a simple algorithm for which a worst-case bound of $\mathcal{O}(2^{0.288N})$ has been shown by advanced analysis techniques (Fomin et al., 2006).

[4]Actually these cliques directly correspond to independent sets in the association graph and are due to the fact that a vertex (u, v) cannot be adjacent to a vertex (u, w) for any $w \in V(H)$ according to Definition 3.1. Thus, a clique algorithm directly operating on the association graph should allow the same bound on running time. The article states only vague arguments for solving the complementary problem instead.

Algorithm 3.1: A folklore algorithm for exact MCIS

```
Input  : Two graphs $G$, $H$.
Output : Order of a maximum common subgraph.
1 $mcs \leftarrow 0$
2 forall the $G' \sqsubseteq G$ do
3     forall the $H' \sqsubseteq H$ do
4         if $G' \simeq H'$ then                 ▷ Graph isomorphism algorithm
5             $mcs \leftarrow \max\{mcs, |G'|\}$
6 return $mcs$
```

Algorithm 3.2: Improved exact MCIS by graph canonization

```
Input  : Two graphs $G$, $H$.
Output : Order of a maximum common subgraph.
1 $\mathcal{A} \leftarrow \emptyset$; $\mathcal{B} \leftarrow \emptyset$
2 for $k \leftarrow 1$ to $\min\{|G|, |H|\}$ do
3     forall the $V' \in [V(G)]^k$ do
4         $\mathcal{A} \leftarrow \mathcal{A} \cup C(G[V'])$        ▷ Graph canonization algorithm
5     forall the $V' \in [V(H)]^k$ do
6         $\mathcal{B} \leftarrow \mathcal{B} \cup C(H[V'])$
7 $\mathcal{C} \leftarrow \mathcal{A} \cap \mathcal{B}$                    ▷ Set of all common subgraphs
8 return maximum order $|G'|$ of a graph $G' \in \mathcal{C}$
```

$\mathcal{O}(2^n \cdot 2^m \cdot T_{\simeq}(n))$. For many graph classes GI can be solved in polynomial time, see Section 2.3. For general graphs a running time $T_{\simeq}(n) = \exp(\mathcal{O}(\sqrt{n \log n}))$ can be obtained (Babai et al., 1983), which for Algorithm 3.1 yields a total running time of $\mathcal{O}(2^{n+m+c\sqrt{n \log n}})$, where c is a constant. Please note that it is necessary to introduce the constant c as $\exp(\mathcal{O}(n)) = 2^{\mathcal{O}(n)} \neq \mathcal{O}(2^n)$. Since $c\sqrt{n \log n} = o(n)$, reducing the term $n + m$ in the exponent means an asymptotic improvement of the running time for any constant c. This folklore algorithm yet is asymptotically faster than the $\mathcal{O}((m+1)^n)$ time algorithm by Suters et al. (2005).

3.1.3 A Faster Algorithm by Graph Canonization

In the following we refine this simple algorithm and obtain a better bound on the running time. The key idea for improvement is not to check all pairs of subgraphs for isomorphism, but to compute the canonical form individually for each graph in order to allow a linear time isomorphism test between pairs of preprocessed graphs afterwards. The pseudo-code of the procedure is presented as Algorithm 3.2. For both input graphs the set of all induced

Table 3.1: Summary on worst-case bounds for maximum common subgraph algorithms for two arbitrary input graphs of order n and m, where $n \leq m$.

Algorithm	Running Time
Clique reduction (Levi, 1973; Robson, 2001)	$\mathcal{O}\left(2^{0.249nm}\right)$
Clique branching (Suters et al., 2005)	$\mathcal{O}\left((m+1)^n\right)$
Algorithm 3.1 (Akutsu and Tamura, 2012)	$\mathcal{O}\left(2^{n+m+c\sqrt{n \log n}}\right)$
Algorithm 3.2	$\mathcal{O}\left(2^{m+n^{1/2+o(1)}}\right)$

subgraphs in canonical form is computed. Let $T_{\text{CAN}}(n)$ be the time required to compute the canonical form of a graph of order n, cf. lines 4 and 6. Two graphs in canonical form can then be tested for isomorphism simply by verifying that the $\mathcal{O}(n^2)$ elements of their adjacency matrices are equal. Since there are $\mathcal{O}(2^k)$ induced subgraphs of a graph of order k, lines 2 to 6 can be computed in time $\mathcal{O}((2^n + 2^m)T_{\text{CAN}}(n))$, where $n = |G|$ and $m = |H|$ and w.l.o.g. we again assume $n \leq m$. We can implement the set intersection in line 7 as follows when using lists to store the canonical forms.[5] First the lists are sorted lexicographically and then we scan both lists simultaneously using a pointer for each list starting at the first element. By successively either increasing the pointer of the lists that addresses the lexicographically smaller element or, in case of equality, keeping an element as result and increasing both pointers, we obtain the intersection. Sorting a list with 2^m elements lexicographically, where a comparison takes $\mathcal{O}(n^2)$ time due to the size of the adjacency matrices, requires $\mathcal{O}(n^2 \cdot 2^m \cdot \log 2^m) = \mathcal{O}(n^2 \cdot m \cdot 2^m)$ total time. The scan of both lists as detailed above requires time $\mathcal{O}(n^2 \cdot 2^m)$ resulting in a total running time of $\mathcal{O}(n^2 \cdot m \cdot 2^m)$ for the set intersection. Consequently, Algorithm 3.2 requires a total time of $\mathcal{O}(2^m \cdot T_{\text{CAN}}(n) + n^2 \cdot m \cdot 2^m)$. For general graphs the best known algorithm for canonization achieves a running time of $T_{\text{CAN}}(n) = \exp(n^{1/2+o(1)})$ (Babai and Luks, 1983). Then the second addend is dominated by the first for increasing n and the total running time simplifies to $\mathcal{O}(2^m \cdot \exp(n^{1/2+o(1)})$.

The bounds on running time for the different algorithms are summarized in Table 3.1. Figure 3.1 illustrates the growth of functions closely related to the asymptotic running times, where we assume $T_{\simeq}(n) = T_{\text{CAN}}(n) = \mathcal{O}(2^{\sqrt{n \log n}})$ for simplicity. Note that for many graph classes GI can be solved in polynomial time and typically there is an accompanying canonization approach with comparable running time (Babai and Luks, 1983, also see Section 2.3). Even for general graphs, $T_{\simeq}$ and T_{CAN} both grow moderately exponentially for the best known algorithms, whereas the number of induced

[5]The approach essentially corresponds to a slightly modified version of the conquer-step of the divide and conquer algorithm Mergesort.

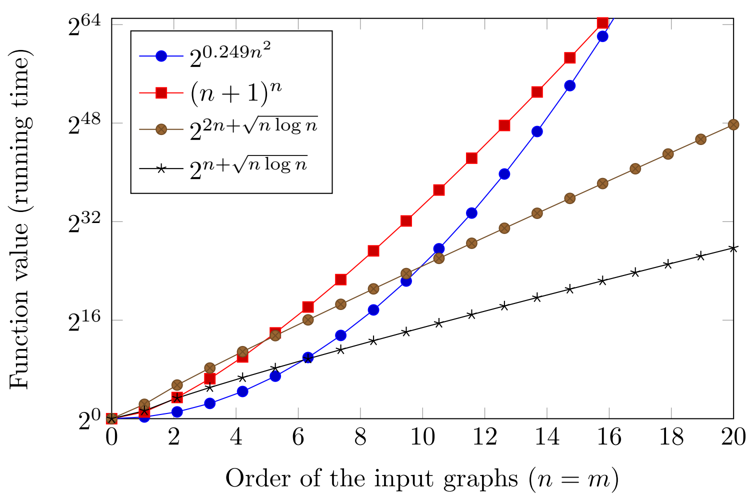


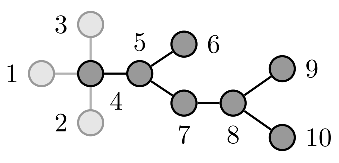
Figure 3.1: Growth of functions closely related to the worst-case bounds of the four exact maximum common subgraph algorithms.

subgraphs increases exponentially (with linear exponent) w.r.t. the number of vertices. Clearly, the algorithm with the best theoretical worst-case bound on running time presented in this section is not likely to be useful in practice. Even for small graphs the running time is prohibitive. Moreover, the exponential space requirement of Algorithm 3.2 would be problematic.

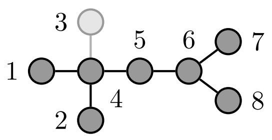
The idea of preprocessing elements individually in order to simplify a related problem between all pairs of objects, also turned out to be useful in the context of graph kernels, cf. Chapter 4.

3.2 A Polynomial-Time Algorithm for the Maximum Common Subtree Problem

The asymptotically best known algorithms consider all possible induced subgraphs of the input graphs. The exponential number of subgraphs renders polynomial-time computation for these approaches impossible. The key idea to obtain polynomial-time algorithms for restricted graph classes is to consider certain well-defined subproblems for a specific set of subgraphs. We present a fundamental approach for the maximum common subtree problem, which was sketched by Matula (1978) in the context of subtree isomorphism. The idea to solve the more general maximum common subtree problem involves multiple maximum weighted bipartite matching problems, cf. Problem 2.5, and is attributed to Jack R. Edmonds. Therefore we refer to the approach as *Edmonds-Matula algorithm*. Similar techniques have later been described on several occasions in literature, for example, see (Akutsu, 1993) for trees or (Gupta and Nishimura, 1998; Kao et al., 2001) for rooted trees.



(a) Tree G



(b) Tree H

Figure 3.2: Two input trees G and H with a maximum common subtree of order 7. Light gray vertices are not part of the maximum common subtree.

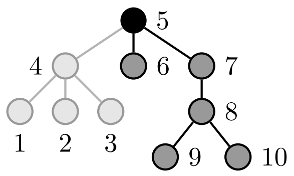
Problem 3.2 (MCST). Maximum Common Subtree

Input: *Two trees G and H.*

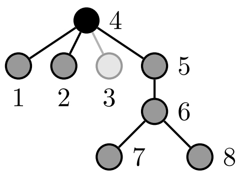
Task: *Determine the maximum order of a common subtree of G and H.*

Figure 3.2 shows an example of two trees and a maximum common subtree. Note that the desired common subgraph must be a tree, in particular, it must be connected. This requirement is essential to obtain a polynomial-time algorithm, since Problem 3.1 remains NP-hard even when the input graphs are restricted to trees (Brandenburg, 2000). This result can be obtained by a simple modification of the classical reduction of 3-Partition to Subforest Isomorphism (Garey and Johnson, 1979, Theorem 4.6).

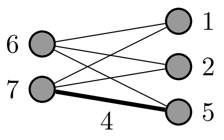
The Edmonds-Matula algorithm systematically decomposes a tree T into rooted subtrees. For an edge $uv \in E(T)$ we denote by T_v^u the subtree rooted at v, where the child u and its descendants are deleted, see Figure 3.3. Likewise T_u^v refers to the other subtree associated with the same edge, which is rooted at u and does not contain v and its descendants. For every pair of rooted subtrees of the two input graphs G and H the maximum common subtree under the restriction that the roots are mapped to each other is computed. The results of these subproblems are stored in a table D that is filled by dynamic programming. Let G_s^i and H_t^j be two rooted subtrees and $M_s = \{s_1, \dots, s_n\}$ and $M_t = \{t_1, \dots, t_m\}$ the children of s in G_s^i and t in H_t^j, respectively.



(a) Subtree G_5^4



(b) Subtree H_4^3



(c) Matching problem

Figure 3.3: Two rooted subtrees (a) and (b) and the associated weighted bipartite matching problem (c). Light gray vertices and edges are not part of the rooted subtrees, root vertices are shown in solid black; edges without label in (c) have weight 1.

Then $D(G^i_s, H^j_t) = 1 + \textsc{MwbMatching}(M_s, M_t, w)$, where MwbMatching is the size of a maximum weight bipartite matching in the complete bipartite graph with vertex set $M_s \uplus M_t$ and edge weights w, cf. Problem 2.5. The edge weights correspond to the solutions for pairs of smaller rooted subtrees and are determined according to $w(s_k, t_l) = D(G^s_{s_k}, H^t_{t_l})$, $k \in \{1, \ldots, n\}$, $l \in \{1, \ldots, m\}$. The matching defines the mapping of the children of the two roots, cf. Figure 3.3(c). The table D is filled by processing subproblems in order of increasing size of the involved rooted subtrees, such that the required partial solutions are always available from D. Finally, the solution of the original problem is determined by combining all pairs of corresponding rooted subtrees according to Algorithm 3.3.

Algorithm 3.3: Maximum Common Subtree

```
Input  : Trees $G$ and $H$.
Data   : Table $D$ containing solutions for all subproblems.
Output : Order of a maximum common subtree of $G$ and $H$.

1 $mcs \leftarrow 0$
2 forall the $(is, jt) \in E(G) \times E(H)$ do
3     $p \leftarrow D(G^i_s, H^j_t) + D(G^s_i, H^t_j)$      ▷ Mapping $i$ to $j$ and $s$ to $t$.
4     $q \leftarrow D(G^i_s, H^t_j) + D(G^s_i, H^j_t)$      ▷ Mapping $i$ to $t$ and $s$ to $j$.
5     $mcs \leftarrow \max\{mcs, p, q\}$
6 return $mcs$
```

The size of the table D is $2\,\|G\| \cdot 2\,\|H\|$ and each cell can be computed in time $\mathcal{O}(n^3)$ by the Hungarian method, where n is the order of the complete bipartite graph (see Section 2.4). Consequently, the total running time of the algorithm is $\mathcal{O}(n^5)$, where $n = \max\{|G|, |H|\}$. Typically, not all the matching instances have size n, but yet the bound on running time is tight. This is seen for the case when G and H both are star graphs, i.e., trees with all but one vertex of degree one. However, the running time can be improved to $\mathcal{O}(n^4)$ by considering only a specific subset of the rooted subtrees of one input graph, which is sufficient to guarantee that the optimal solution is found (Droschinsky et al., 2015).[6] Further improvement is possible by exploiting the following two properties: First, the emerging matching instances are closely related since the symmetric difference of the children of T^u_v and T^w_v contains exactly the two vertices u and w. Second, the matching instances exhibit integral edge weights with a largest possible magnitude of $\mathcal{O}(n)$. Hence, scaling algorithms are more efficient for these graphs than the general Hungarian method, cf. Section 2.4.

[6]This was observed earlier by Schietgat et al. (2013), who state an even better bound on the running time of $\mathcal{O}(n^{2.5})$ for a more general problem, which yet remains subject of discussion (private communication with the authors).

3.3 Variants of Maximum Common Subgraph Problems and their Complexity

There are various different variants of maximum common subgraph problems considered in literature. As discussed in the previous section MCIS is NP-hard in trees, but polynomial-time solvable when the common subgraph must be connected. This fact already suggests that there is a complex relation between the problem definition and the properties of the input graphs. Some variants of common subgraph problems are even motivated by the fact that more general definitions are known to be NP-hard, but restrictions still yield meaningful results in practice. We first summarize basic definitions of different variants and then discuss their relation before providing known complexity results in Section 3.3.3.

Several authors consider common subgraph problems that do not ask for induced subgraphs as Problem 3.1, but require the desired common subgraph to be edge-induced. Consequently, the size of a solution in terms of its number of edges is maximized for this variant.

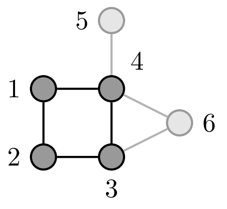
Problem 3.3 (MCES). Maximum Common Edge Subgraph

Input: *Two graphs G and H.*

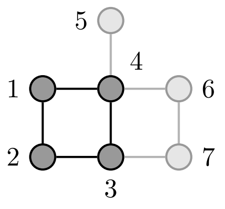
Task: *Determine the maximum size of a common edge-induced subgraph of G and H.*

If in addition the common subgraph is required to be connected, we refer to the problem as Maximum Common Connected Edge Subgraph (MCCES) and Maximum Common Connected Induced Subgraph (MCCIS) problem, respectively. The connectivity of the common subgraph has an essential influence on the complexity of the problem in general as we already have seen for trees. Even subgraph isomorphism is NP-complete when the pattern graph is a tree and the other is outerplanar (Sysło, 1982). Therefore, the concept of *block and bridge preserving* (BBP) subgraph isomorphism has been proposed by Horváth et al. (2006; 2010) and was later transferred to common subgraph problems (Schietgat et al., 2007; 2008; 2011; Akutsu and Tamura, 2013). This restriction renders efficient algorithms for outerplanar graphs possible. Notably, finding maximum common subgraphs under BBP isomorphism yields meaningful results for molecular graphs in practice and even compares favorably to ordinary maximum common subgraphs in empirical studies (Schietgat et al., 2008; 2011). Formally, a common subgraph isomorphism ψ between two graphs G and H is said to be *block and bridge preserving* if and only if the associated subgraph isomorphisms ψ_G and ψ_H to the input graphs G and H both satisfy:

BBP1 any two edges from different blocks in the common subgraph are mapped to edges from different blocks in the input graph;



(a) Graph G



(b) Graph H

EXTENSION	CES	CIS	BBP
$5 \mapsto 5$	✓	✓	✓
$5 \mapsto 5, 6 \mapsto 6$	✓	✗	✗
$5 \mapsto 6$	✓	✓	✗
$5 \mapsto 6, 6 \mapsto 5$	✓	✗	✗

(c) Common subgraph isomorphisms

Figure 3.4: Different common subgraph problems. Assume we have a common subgraph isomorphism $\psi : V(G) \rightsquigarrow V(H)$ with $\psi(\mathbf{x}) = \mathbf{x}$ for $\mathbf{x} \in \{1, 2, 3, 4\}$ between the two input graphs (a), (b). The table (c) shows several extensions of this isomorphism that are all associated with common connected edge-induced subgraphs, but not necessarily induced subgraphs and block and bridge preserving isomorphisms.

BBP2 each bridge in the common subgraph is mapped to a bridge in the input graph.

Figure 3.4 illustrates possible isomorphisms between common subgraphs and the constraints they satisfy. The BBP constraint can be applied to the different maximum common subgraph problems by taking only common subgraphs into account, for which a BBP isomorphism exists. So far, only BBP-MCCES in outerplanar graphs has been studied (Schietgat et al., 2007; 2008; 2011; Akutsu and Tamura, 2013), although BBP-MCCIS is as well of interest. BBP-MCCIS is equivalent to Problem 3.2 when the input graphs are restricted to trees, since trees do not contain any blocks. Whereas, when the input graphs are required to be biconnected, i.e., they do not contain bridges, BBP-MCCIS computes a maximum common biconnected induced subgraph. Otherwise the common subgraph must contain at least one bridge, whereas the input graphs do not—hence, condition (BBP2) must be violated. Connectivity is essential for the complexity of maximum common subgraph problems in tree-like graphs leading to the following problem.

Problem 3.4 (k-MCIS). MAXIMUM COMMON k-CONNECTED INDUCED SUBGRAPH IN PARTIAL k-TREES

Input: *Two k-connected partial k-trees G and H.*

Task: *Determine the maximum order of a common k-connected induced subgraph of G and H.*

The closely related problem, where the maximum size of an edge-induced common subgraph is to be searched, is analogously referred to as k-MCES. Note that 1-MCIS is equivalent to the maximum common subtree problem, cf. Problem 3.2. We discuss several fundamental relations between problem variants and algorithmic modifications applicable to MCIS algorithms in order to solve specific variants.

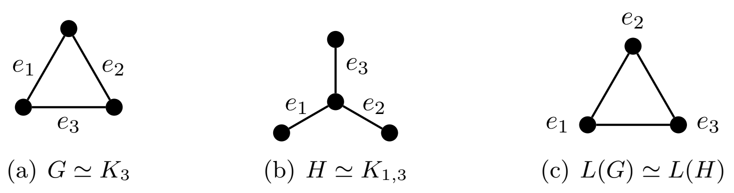


Figure 3.5: The non-isomorphic graphs (a) and (b) have the same line graph (c).

3.3.1 Vertex- and Edge-Induced Common Subgraphs

The exact algorithms described in Section 3.1 solve MCIS. However, in certain application domains like cheminformatics common edge subgraphs are sometimes considered more desirable. It is typically possible to adapt algorithms to directly solve the edge-induced variant. However, general approaches that reduce common edge subgraph problems to induced subgraph problems in adequately modified input graphs are widely-applied in practice.

Approaches Based on Line Graphs

Nicholson et al. (1987) first proposed a general solution based on the reduction to the clique problem, cf. Section 3.1.1, which was later applied on various occasions (Tonnelier et al., 1990; Durand et al., 1999; Koch, 2001). Instead of directly creating the association graph of the two input graphs, the association graph of the line graphs[7] is created.

Definition 3.2 (Line Graph). *The* line graph $L(G)$ *of a graph* G *is the graph with vertex set* $V(L(G)) = E(G)$*, in which two vertices are adjacent if and only if the two corresponding edges are adjacent in* G*.*

Obviously, two isomorphic graphs have isomorphic line graphs. A classical result states that almost always the reverse holds as well.

Theorem 3.1 (Whitney, 1932). *Given two connected graphs* G *and* H*. If* $L(G) \simeq L(H)$ *then* $G \simeq H$*, unless* $G \simeq K_{1,3}$ *and* $H \simeq K_3$ *or vice versa.*[8]

The single exception is referred to as *ΔY-exchange* (Raymond and Willett, 2002; Raymond et al., 2002b) or *triode-triangle interchange* (Durand et al., 1999) and depicted in Figure 3.5.

The family of line graphs is closed under vertex removal, i.e., each induced subgraph of a line graph again is a line graph. Furthermore one can observe

[7]Often referred to as *edge graph* or, in the context of molecular graphs, *bond graph* (Tonnelier et al., 1990).

[8]The result was originally formulated without the notion of line graphs and instead directly relates graph isomorphism to edge isomorphism. An *edge isomorphism* is a bijection between edges such that any two edges are adjacent in one graph if and only if the associated edges in the other graph are adjacent.

that each vertex-induced subgraph of $L(G)$ directly corresponds to an edge-induced subgraph of G and vice versa. Based on this idea common edge subgraphs can be detected, if ΔY-exchanges are handled adequately: For a common induced subgraph isomorphism between two line graphs that maps a K_3, it must be verified that the corresponding edge-induced subgraphs in the original graphs are either both K_3 or both $K_{1,3}$. The check can be incorporated in the clique detection algorithm, but the same idea also can be used for other exact MCIS algorithms. Tonnelier et al. (1990) as well as Durand et al. (1999) and later Koch (2001) directly define an equivalent edge association graph without requiring the notion of line graphs.

An Approach Based on Subdivision Graphs

Vismara and Valery (2008) proposed an alternative to the classical method based on line graphs. Here the subdivision graphs of the two input graphs are considered.

Definition 3.3 (Subdivision Graph). *Let G be a graph. The* subdivision graph *$S(G)$ is obtained from G by subdividing every edge.*

We can distinguish two types of vertices in a subdivision graph: Those contained in the original graph and those that represent edges of the original graph. The subdivision graph is bipartite with respect to these two sets. Clearly, every edge-induced subgraph of G corresponds to an induced subgraph of $S(G)$. But, vice versa, not every induced subgraph of $S(G)$ corresponds to an edge-induced subgraph of G.

There are several technicalities that must be considered to obtain a maximum common (connected) edge-induced subgraph isomorphism between the original graphs from their subdivision graphs. First, vertices representing original vertices must not be mapped to vertices representing edges and vice versa. Furthermore, a subgraph of a subdivision graph is only valid, if every vertex that represents an edge has degree two in the subgraph. Finally, the size of the solution corresponds to the number of vertices that represent edges in the common induced subgraph isomorphism between the subdivision graphs. For further details of the approach as well as additional constrains to restrict to connected common edge subgraphs the reader is referred to (Vismara and Valery, 2008). The resulting problem again corresponds to a special clique problem in a product graph and is formulated as a constraint satisfaction problem.

3.3.2 Finding Connected Common Subgraphs

Tonnelier et al. (1990) considered the problem of enumerating all maximal connected common edge subgraph isomorphisms between two graphs G and H. An algorithm is proposed, which enumerates all maximal cliques in the association graph $L(G)\nabla L(H)$ obtained from the line graphs of G and H. In a subsequent filtering step disconnected subgraphs are removed. Koch (2001) avoids the exhaustive enumeration by directly restricting the solution space searched by backtracking. The key idea of the approach is to distinguish between two types of edges in the association graph: Edges that are due to common adjacencies are so called *c-edges*, all other edges are referred to as *d-edges* and represent the absence of adjacency in both graphs, cf. Definition 3.1. A clique C in an association graph A is called *c-clique* if $A[C]$ is spanned by c-edges. Koch (2001) showed that a clique in the association graph corresponds to an isomorphism between common connected edge subgraphs if and only if it is a c-clique.

Based on this observation the algorithm by Bron and Kerbosch (1973) is extended to find only c-cliques, which considerably reduces the solution space. Some flaws in the algorithm were later corrected by Cazals and Karande (2005).

3.3.3 Polynomial-Time Algorithms and Hardness Results

While the general problem MCIS clearly is NP-hard, several special cases for restricted graph classes are known that allow to obtain polynomial-time algorithms. There is a complex interplay between graph classes, the properties of the desired common subgraph and the complexity of the problem, which is yet not fully understood. Table 3.2 summarizes known results that are particularly relevant for our work.

As previously mentioned MCIS/MCES in trees is NP-hard, while MCCIS and MCCES can be solved in polynomial time. For two rooted trees the problem can be solved in $\mathcal{O}(n^{2.5}\log n)$ (Gupta and Nishimura, 1998), which can be improved to $\mathcal{O}(\sqrt{d}n^2\log\frac{2n}{d})$, where d denotes the maximum vertex degree (Kao et al., 2001). The approach by Gupta and Nishimura (1998) also allows to solve MCST w.r.t. topological embedding, which allows to map edges of the common subtree to vertex-disjoint paths of the input trees. MCCES in connected k-almost trees is NP-hard for any $k \geq 1$, but can be solved in polynomial time in connected almost trees of bounded degree (Akutsu, 1993). MCCIS can also be solved in polynomial time when one of the input graphs is a bounded-degree partial k-tree and the other is a connected graph with a polynomial number of possible spanning trees (Yamaguchi and Mamitsuka, 2003; Yamaguchi et al., 2004). This is a generalization of the result for almost trees by Akutsu (1993) since almost trees are a subclass of these graphs.

Solving BBP-MCCES in outerplanar graphs can be achieved in polynomial time as shown in several publications (Schietgat et al., 2007; 2008; 2013; Akutsu and Tamura, 2013). By non-trivial modification of an algorithm for BBP-MCCES Akutsu and Tamura (2013) proved that MCCES in outerplanar graphs of bounded degree is polynomial-time solvable. Since this work focuses on the complexity of the general problem with bounded degree constraint,

Table 3.2: Summary on complexity results for the maximum common induced subgraph problem and its variants. Note that the graph classes are related as follows: trees $\subset$ outerplanar graphs $\subset$ series-parallel graphs $\subset$ partial k-trees; trees $\subset$ k-almost trees $\subset$ partial k-trees (Bodlaender, 1986); (Δ-bounded — the maximum degree is bounded by a constant; $\star$ — the result was originally shown for the edge-induced variant).

Input graph class	Constraint	Complexity
Trees	—	NP-hard (Brandenburg, 2000)
	connected	P (Matula, 1978, Sec. 3.2)
k-almost trees	connected	NP-hard for any $k \geq 1$ (Akutsu, 1993)
Δ-bounded	connected	P (Akutsu, 1993)
Outerplanar graphs	connected	NP-hard (Sysło, 1982)
	biconnected	P (Sec. 3.4.4)
	connected, BBP	P (Kriege et al., 2014a, Sec. 3.5)
Δ-bounded	connected	P (Akutsu and Tamura, 2013)*
Series-parallel graphs	biconnected	P (Kriege and Mutzel, 2014, Sec. 3.4)
	connected, BBP	P (Kriege et al., 2014a, Sec. 3.5)
biconnected, one vertex of unbounded degree	connected	NP-hard (Kriege et al., 2014a)
Partial k-trees	j-connected, $j < k$	NP-hard for any $k \geq 1$ (Brandenburg, 2000)
	k-connected	Open for $k > 2$ (Sec. 3.4.1, 3.7)
Δ-bounded, $k = 11$ (vertex-labeled)	connected	NP-hard (Akutsu and Tamura, 2012)
Δ-bounded	connected	Open for $k \in \{2, \ldots, 10\}$ (Sec. 3.7)

the authors only gave a rough bound of $\mathcal{O}(n^{10})$ for their BBP-MCCES algorithm, where $n = \max\{|G|, |H|\}$. The BBP-MCCES approach by Schietgat et al. (2007) was shown to be efficient in practice (Schietgat, 2010; Schietgat et al., 2013) and different worst-case bounds have been provided over time in consecutive publications: Starting with $\mathcal{O}(n^7)$ (Schietgat et al., 2007), then $\mathcal{O}(n^5)$ (Schietgat et al., 2008; Schietgat, 2010) and finally $\mathcal{O}(n^2\sqrt{n})$ (Schietgat et al., 2013).[9]

Note that the subgraph isomorphism problem, i.e., to determine if G is isomorphic to a subgraph of H, can be reduced to the maximum common subgraph problem: There is an induced subgraph isomorphism from G to H if and only if a maximum common induced subgraph between G and H has order $|G|$. Therefore, the NP-completeness results presented in Section 2.3 imply the NP-hardness of the maximum common subgraph problem. However, the case where G and H have a certain property that the desired common subgraph not necessarily must have does not directly follow from such a reduction. The complexity of subgraph isomorphism in partial k-trees is exceptionally well studied. In a nutshell, subgraph isomorphism is solvable in polynomial time in partial k-trees if the pattern graph either is k-connected or has bounded degree (Matoušek and Thomas, 1992; Gupta and Nishimura, 1994). In contrast, it is NP-complete when the pattern graph is not k-connected or has more than k vertices of unbounded degree (Gupta and Nishimura, 1996b).

Unfortunately such a clear demarcation for partial k-trees is not known for maximum common subgraph problems and their complexity in partial k-trees is still not fully understood. However, since practically relevant graphs for maximum common subgraph computation, e.g., derived from small molecules, often have small treewidth (Yamaguchi et al., 2004; Horváth and Ramon, 2010), it is highly relevant to develop polynomial-time algorithms for tractable graph classes and to clearly identify graph classes, where the problem remains NP-hard. The above mentioned polynomial-time solvable cases of common subgraph problems impose constraints that are comparable to those that render subgraph isomorphism tractable.

[9]The proofs of the worst-case bounds are not the main focus of the publication and are rather brief. The analysis leading to $\mathcal{O}(n^2\sqrt{n})$ leaves several questions open:

- The authors propose to use the algorithm by Hopcroft and Karp (1973) to solve maximum weight bipartite matching problems (defined as in Problem 2.5) in time $\mathcal{O}(n^2\sqrt{n})$. Since this algorithm computes a maximum cardinality matching in bipartite graphs and is not designed to take weights into account, it remains unclear how to apply it to the instances that occur.
- The analysis of the method RMCSCOMPOUND relies on the fact that a vertex v contributes at most $\deg(v)$ vertices to a matching instance. While this certainly is true, the analysis does not seem to take into account that there may be up to $\deg(v) + 1$ matching instances of that size for a vertex v of the second input graph.

Unfortunately, these issues could not yet be clarified by contacting the authors. Using the Hungarian method for solving the matching problems and taking account of the additional instances would yield a total running time of $\mathcal{O}(n^4)$ for the approach.

However, the positive results for common subgraph problems are mostly limited to outerplanar graphs, which are a subclass of the series-parallel graphs, i.e., the partial 2-trees. As a matter of fact, the more general positive results for subgraph isomorphism in partial k-trees cannot be transferred to common subgraph problems: Recently, Akutsu and Tamura (2012) have shown that MCCIS and MCCES both are NP-hard in vertex-labeled partial 11-trees of bounded degree—a class of graphs which allows for polynomial-time subgraph isomorphism algorithms (Matoušek and Thomas, 1992; Gupta and Nishimura, 1994). Essentially the complexity of maximum common connected subgraph problems is unknown for graphs of bounded-degree that are not outerplanar and have tree width less than 11.

We consider the positive result that subgraph isomorphism in k-connected partial k-trees can be solved in polynomial time, which is closely related to k-MCIS and k-MCES. Since a tree is a graph, where every edge is a bridge, BBP-MCCIS generalizes the maximum common subtree problem as defined in Problem 3.2, which is equivalent to 1-MCIS. Applying BBP-MCCIS and BBP-MCCES to biconnected series-parallel graphs actually solves 2-MCIS and 2-MCES, respectively. Hence, 2-MCES in outerplanar graphs, a subclass of the partial 2-trees, was solved as a subproblem of BBP-MCCES (Schietgat et al., 2013; Akutsu and Tamura, 2013). Schietgat (2010) stated that their algorithm for 2-MCES as part of the approach for BBP-MCCES is related to the subgraph isomorphism algorithm for biconnected outerplanar graphs by Lingas (1989). The concept of block and bridge preserving isomorphisms cannot be expected to render common subgraph problems tractable in partial k-trees with $k > 2$: Computing the maximum common $(k-1)$-connected subgraph of two k-connected partial k-trees is known to be NP-hard (Brandenburg, 2000). This was shown using techniques of Gupta and Nishimura (1996b) for proving that subgraph isomorphism is NP-complete when the pattern graph is a partial $(k-1)$-tree and the other a partial k-tree. However, the positive result for subgraph isomorphism in k-connected partial k-trees was reported to be transferred to solve k-MCIS and k-MCES for arbitrary k (Bachl et al., 2004; Hajiaghayi and Nishimura, 2007).[10] We will show in Section 3.4.1 that the mentioned approach is flawed and fails even for series-parallel graphs. While we present a different solution for these graphs, the general problem remains open for $k > 2$.

3.4 Finding Maximum Common Biconnected Induced Subgraphs in Series-Parallel Graphs

We consider 2-MCIS, where the input graphs are biconnected partial 2-trees, i.e., series-parallel graphs, and the common induced subgraph is required to

[10]In both references the result is attributed to an unpublished manuscript by Brandenburg (2000), which was kindly provided by its author.

be biconnected. When the input graphs are not biconnected, it is sufficient to solve 2-MCIS for all pairs of blocks of the two input graphs to obtain a maximum common biconnected induced subgraph. It was believed that k-MCIS can be solved by means of normalized tree decompositions that were previously applied to solve subgraph isomorphism (Brandenburg, 2000). However, we show that this is actually not the case and that approaches directly based on (normalized) tree decompositions in general must fail even for $k = 2$. We discuss key obstacles of tree decompositions arising for common subgraph problems. These issues do neither occur in outerplanar graphs, for which the more general BBP-MCCES problem is polynomial-time solvable (Schietgat et al., 2007; Akutsu and Tamura, 2013), nor for the related subgraph isomorphism problem.

We introduce the concept of *potential separators*, i.e., separators of a subgraph to be searched that not necessarily are separators of the input graph. We characterize these separators and propose a polynomial-time solution for 2-MCIS based on the SP-tree data structure. In order to obtain polynomial-time algorithms for BBP-MCCIS in series-parallel graphs it is necessary to solve 2-MCIS in polynomial time. Vice versa, it is possible to solve BBP-MCCIS in series-parallel graphs by a combination of slightly modified algorithms for 1- and 2-MCIS applied to the blocks and bridges of the input graphs as we will show later in Section 3.5. In addition, the concepts introduced in the following may form the basis for solving k-MCIS for $k > 2$.

3.4.1 Tree Decompositions and Common Subgraph Problems

We first introduce graph-theoretical concepts and data structures that are especially relevant for the 2-MCIS problem considered here. Then we discuss the problems that occur when these decompositions are applied in order to solve common subgraph problems before we present our novel algorithm.

Normalized Tree Decompositions

As a consequence of the close relation between tree decompositions and separators, cf. Sections 2.2.3, 2.2.4, the inequality $\kappa(G) \leq \text{tw}(G)$ holds for any graph G. The equality $\kappa(G) = \text{tw}(G) = k$ is satisfied for k-connected partial k-trees only—a class of graphs that turned out to be tractable for subgraph isomorphism. To obtain a polynomial-time algorithm for subgraph isomorphism in k-connected partial k-trees Gupta and Nishimura (1994) proposed a normalized tree decomposition making k-separators explicit as separator nodes, which are distinguished from clique nodes. Here, we call a tree decomposition $(T, \mathcal{X})$ *normalized* if $V(T)$ can be divided into the two disjoint sets S and C of separator and clique nodes, respectively, such that

N1 S and C is a bipartition of T and all leaves of T are clique nodes,

N2 all separator and clique nodes have bags of size k and $k+1$, respectively,

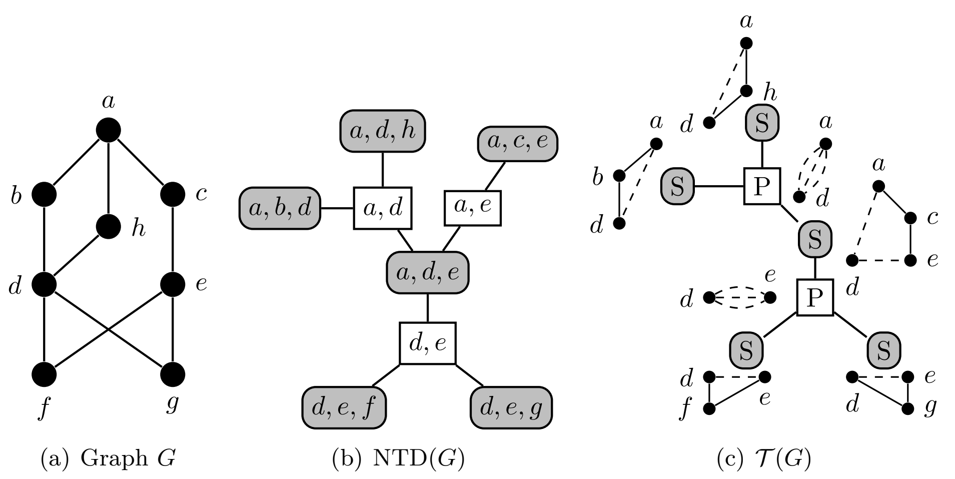


Figure 3.6: A biconnected partial 2-tree G (a), a normalized tree decomposition of G (b) and an SP-tree of G (c) with the associated skeleton graphs (dashed lines represent virtual edges).

N3 for each path (i, j, k) in T: $X_j = X_i \cup X_k$ if $j \in C$ and $X_j = X_i \cap X_k$ if $j \in S$.

The bags associated with separator nodes form a set of pairwise parallel separators. Figure 3.6(b) shows an example of a normalized tree decomposition, denoted by NTD(G). Note that in general a (normalized) tree decomposition is not unique for a given graph. Therefore, a so-called *tree decomposition graph* was used by Gupta and Nishimura (1994), which is a directed acyclic graph incorporating all possible normalized tree decompositions. Normalized tree decompositions and tree decomposition graphs can be computed in time $\mathcal{O}(n^2)$ and $\mathcal{O}(n^{k+2})$, respectively (Gupta and Nishimura, 1994).

SP-tree Data Structure

For biconnected partial 2-trees, SP-trees are a well-known data structure, which reflects their series parallel composition, cf. Figure 3.6(c). We use a notation and definition common for SPQR-trees (see, e.g., Chimani and Hliněný, 2011), a generalization of SP-trees, which can be computed in linear time (Gutwenger and Mutzel, 2001).

Definition 3.4 (SP-tree). *Let G be a biconnected partial 2-tree with at least three vertices. The* SP-tree $\mathcal{T} = \mathcal{T}(G)$ *is the smallest tree that satisfies the following properties:*

SP1 *Each node*[11] *μ of $\mathcal{T}$ is associated with a* skeleton *graph $S_\mu = (V_\mu, E_\mu)$.*

[11]As for tree decompositions, we refer to the vertices of the tree $\mathcal{T}$ as *nodes* to distinguish them from the vertices of the graph G.

Each edge uv of E_μ is either a real *edge with $uv \in E(G)$ or a* virtual *edge where $\{u, v\}$ forms a 2-separator of G.*

SP2 *$\mathcal{T}$ has two different node types with the following skeleton structures:*

S: *The skeleton S_μ is a cycle, i.e., μ represents a series composition.*

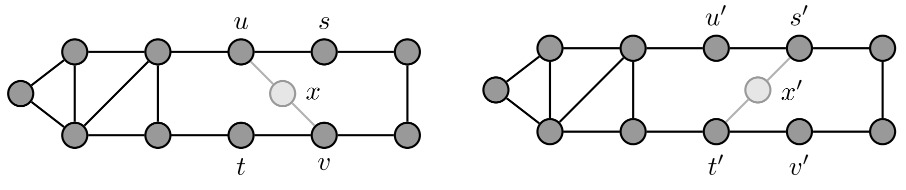
P: *The skeleton S_μ is a multigraph consisting of two vertices and multiple parallel edges between them, i.e., μ represents a parallel composition.*

SP3 *For two adjacent nodes μ and ν the skeleton S_μ contains a virtual edge e_ν that represents S_ν and vice versa. The node ν is called* pertinent *to the edge e_ν.*

SP4 *The original graph G can be obtained by merging the skeletons of adjacent nodes, where the virtual edges e_μ and e_ν are not part of the resulting graph and their common endpoints are merged for all $\mu\nu \in E(\mathcal{T})$.*

The S-nodes $V_\mathrm{S}(\mathcal{T})$ and the P-nodes $V_\mathrm{P}(\mathcal{T})$ form a bipartition of $\mathcal{T}$. Since a vertex v of a graph may occur in multiple skeleton graphs of the SP-tree $\mathcal{T}$, we denote by $\mu(v)$ the representative of v in the skeleton S_μ. For the sake of simplicity we do not distinguish vertices of the original graph and their representatives in skeleton graphs, where clear from context. Let $\mathcal{T}$ be the SP-tree of G and $r = uv$ be an arbitrary edge in G. The SP-tree *rooted at* r is obtained by rooting $\mathcal{T}$ at the node τ such that r is a real edge in S_τ. Let μ be a child of ν with respect to the parent-child relationship induced by the root τ. The virtual edge e_ν in S_μ is called *reference edge* of μ, denoted by $\mathrm{ref}(\mu)$. The edge r is considered the reference edge of τ. For a node μ of the SP-tree $\mathcal{T}(G)$, there is a direct correspondence between the virtual edges of the skeleton graph S_μ and the connected components of $\mathcal{T} \setminus \mu$. Note that the cyclic ordering of virtual edges in the skeleton of an S-node μ induces an ordering of the neighbors of μ in $\mathcal{T}$, whereas the neighbors of a P-node are unordered.

A normalized tree decomposition of a graph G can be obtained from its SP-tree by successively selecting and splitting an S-node μ that has a skeleton graph with more than three vertices. Two arbitrary non-adjacent vertices u, v of S_μ form a separator of G. We create a new P-node ν adjacent to two new S-nodes μ' and μ^*, such that $V(S_\nu) = \{u, v\}$ and there are two virtual edges in the skeleton graph with pertinent nodes μ' and μ^*. The skeleton graphs $S_{\mu'}$ and S_{μ^*} represent the two parts of S_μ separated by $\{u, v\}$ and the node μ is replaced by μ' and μ^* and ν in the SP-tree, such that (SP4) still holds. Note that this tree still fulfills the properties (SP1)–(SP4), but no longer is smallest possible as required for an SP-tree by Definition 3.4. When there is no S-node left with a skeleton with more than three vertices, the tree corresponds to a normalized tree decomposition, whereas P-nodes map to separator nodes and S-nodes to clique nodes. Conversely, the SP-tree can be obtained from a normalized tree decomposition of a partial 2-tree by successively merging



(a) Graph G (b) Graph H

Figure 3.7: Example where straight-forward algorithms based on tree decomposition fail.

each separator node s with its adjacent clique nodes if (i) $\deg(s) = 2$ and (ii) the vertices in the associated bag X_s are not adjacent in G.

For all P-nodes ν of $\mathcal{T}(G)$ the set $V(S_\nu)$ is a bag of a separator node of every normalized tree decomposition of G. SP-trees are unique, which follows from the fact that the triconnected components represented by the more general SPQR-trees are unique (Hopcroft and Tarjan, 1973). In contrast, normalized tree decompositions are not unique. Note that different normalized tree decompositions can be obtained from an SP-tree with the procedure detailed above depending on the vertices selected to split S-nodes with more than three vertices.

Application to Common Subgraph Problems

A key property of tree decompositions is their close relation to separators, which allows to systematically divide graphs into subgraphs along the tree structure. Based on solutions for these parts, typically a solution for the input graphs is computed. This is the case for the Edmonds-Matula algorithm presented in Section 3.2. A key observation for correctness of the approach is the following.

Observation 3.1. *Let C be a common subtree of the two trees G and H with isomorphism ψ. If v is a 1-separator of C, then $\psi_G(v)$ and $\psi_H(v)$ are 1-separators of G and H, respectively.*

This fact allows to determine a solution based on results for pairs of subtrees of the input graphs determined by the separating vertices. Figure 3.7 illustrates that this observation does not hold in a similar manner for the 2-separators of biconnected series-parallel graphs: The maximum common subgraph in the given example consists of the parts of the graphs depicted in black. Note that in every normalized tree decomposition of G and H there is a separator node with bag $\{u, v\}$ and $\{s', t'\}$, respectively. This follows from (Dessmark et al., 2000, Theorem 3.3) and is apparent from the relation between separator nodes of a normalized tree decomposition and P-nodes of the SP-tree, which are unique. For a maximum common subgraph C with isomorphism ψ we must have $\psi(\mathbf{x}) = \mathbf{x}'$ for $\mathbf{x} \in \{u, v, s, t\}$. Let $a = \psi_G^{-1}(u)$, $b = \psi_G^{-1}(v)$, then $\{a, b\}$ is a separator of C, but $\{\psi_H(a) = u', \psi_H(b) = v'\}$ is not a separator of H. When computing common subgraphs based on normalized tree decompositions, typically all vertices of a bag in NTD(G) are matched to the vertices of a bag in NTD(H). One obstacle here is that normalized tree decompositions are not unique. This can be overcome by means of tree decomposition graphs. However, in the example, there are no bags with corresponding vertices in *any* normalized tree decomposition. Therefore, it is difficult to apply normalized tree decompositions to solve common subgraph problems. In contrast to the subgraph isomorphism problem, where normalized tree decompositions have been successfully applied, a key obstacle here is that the subgraph to be searched may be a subgraph of both input graphs. Hence, each input graph may contain parts that are not contained in the common subgraph, but constrain their possible normalized tree decompositions. This is the reason why the approach of Brandenburg (2000) mentioned in (Bachl et al., 2004; Hajiaghayi and Nishimura, 2007) does not work.

To overcome this issue, in Section 3.4.2 we introduce the concept of a *potential separator*, i.e., a separator of a subgraph to be searched that not necessarily is a separator of the input graphs. For a given potential separator P we characterize the maximal subgraph that is separated by P and propose a decomposition of series-parallel graphs into split graphs based on all possible potential separators analogously to the approach by Edmonds-Matula for trees. In Section 3.4.3 we provide an algorithm based on SP-trees that solves 2-MCIS by implicitly matching the split graphs of the two input graphs.

3.4.2 Theoretical Background for Novel Approaches

We say $P = \{u, v\}$ is a *potential separator* of a biconnected partial 2-tree G if there is a biconnected induced subgraph $G' \sqsubseteq G$ with separator P.

Observation 3.2. *Let C be a biconnected common subgraph of the biconnected partial 2-trees G and H with isomorphism ψ. If $\{u, v\}$ is a 2-separator of C, then $\{\psi_G(u), \psi_G(v)\}$ and $\{\psi_H(u), \psi_H(v)\}$ are potential separators of G and H, respectively.*

In the following we characterize the maximal biconnected induced subgraph G_P^* that is separated by a potential separator P. Since a potential separator not necessarily separates G, there may be parts of G keeping $G \setminus P$ connected, which can consequently not be contained in G_P^*. We show that these parts are associated with specific separators of G that we refer to as critical. We call a 2-separator S of a biconnected partial 2-tree G *compulsive* if every normalized tree decomposition NTD(G) contains a separator node i with bag $X_i = S$. If S is compulsive for G and there is an induced biconnected subgraph $G' \sqsubseteq G$, such that S is a non-compulsive separator of G', we say S is *critical*.

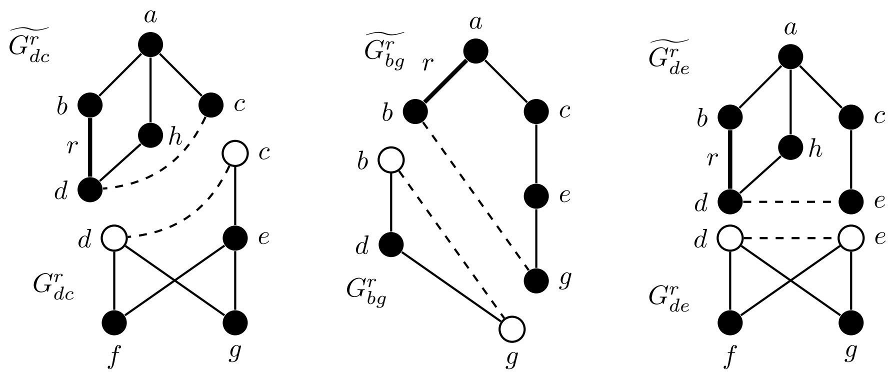
Lemma 3.1. *Let G be a biconnected partial 2-tree and $S = \{u, v\} \subset V(G)$. The following statements are equivalent:*

(i) The set S is a critical separator of G,
(ii) S is compulsive for G and $uv \notin E(G)$,
(iii) there is a P-node ν in the SP-tree $\mathcal{T}(G)$ with $V(S_\nu) = S$ and $uv \notin E(G)$,
(iv) the graph $G \setminus S$ has at least three connected components and $uv \notin E(G)$,
(v) there is no 2-separator that crosses S in G, but in a biconnected induced subgraph $G' \sqsubseteq G$.

Proof. A separator $S = \{u, v\}$ of G is compulsive if and only if (a) the graph $G \setminus S$ has at least three connected components (see Dessmark et al., 2000, Theorem 3.3) or (b) the vertices u and v are adjacent and $G \setminus S$ has at least two connected components. Condition (b) and the equivalence follows from the relation between normalized tree decompositions and SP-trees and the fact that P-nodes are unique. Note that if u and v are adjacent then condition (b) also holds for every induced biconnected subgraph $G' \sqsubseteq G$ that is separated by S. Consequently a critical separator must consist of non-adjacent vertices, i.e., condition (a) must hold in G, but not in G'. Assume u and v are not adjacent and condition (a) holds and let $C_1, \dots, C_l$ be the connected components of $G \setminus S$, $l \geq 3$. Then $G' = G[V(C_1) \uplus V(C_2) \uplus S]$ is a biconnected subgraph of G with separator S such that S is not compulsive for G', since neither (a) nor (b) apply. Therefore S is critical. This proves the equivalence of the statements (i) and (ii) of Lemma 3.1. The equivalence of the statements (ii), (iii) and (iv) directly follows from the characterization of compulsive separators. Let G' be defined as above, then there are two vertices $i \in V(C_1)$, $j \in V(C_2)$ such that $T = \{i, j\}$ separates G' and crosses S. However, T does not separate G since there is a path from u to v through C_3. Therefore, (iv) implies (v). Vice versa, the existence of a 2-separator T in an induced subgraph $G' \sqsubseteq G$ that crosses S requires $uv \notin E(G)$ since otherwise T would not separate G'. Assume $G \setminus S$ has two connected components, then T also separates G, contradicting the assumption that there is no 2-separator of G that crosses S. □

The graph in Figure 3.6(a), for example, contains the compulsive separators $S_1 = \{a, d\}$ and $S_2 = \{d, e\}$, which are both critical. Extending the symmetric relation of crossing separators, we say a 2-separator S *crosses* a set of two vertices $T = \{u, v\}$ if and only if S is an (u, v)-separator. Lemma 3.2 provides the key for our characterization of the maximal biconnected subgraph of G that is separated by a potential separator P (see Corollary 3.1).

Lemma 3.2. *Let G be a biconnected partial 2-tree and S a critical separator that crosses a potential separator $P = \{u, v\}$. Let C_u and C_v denote the components of $G \setminus S$ containing the vertex u and v, respectively. Every biconnected induced subgraph $G' \sqsubseteq G$ separated by P is a subgraph of the biconnected graph $G^S_P = G[V(C_u) \uplus V(C_v) \uplus S]$.*



(a) Split at $\{d, c\}$ (b) Shear split at $\{b, g\}$ (c) Split at $\{d, e\}$

Figure 3.8: Splitting the graph shown in Figure 3.6(a).

Proof. Let $S = \{s, t\}$ be a critical separator that crosses the potential separator $P = \{u, v\}$. Let G' be a biconnected induced subgraph of G that is separated by P. Since G is biconnected and S is a (u, v)-separator of G, there must be two disjoint paths from u to v containing s and t, respectively. Therefore, s and t must also be contained in G'. According to Lemma 3.1 the graph $G \setminus S$ has components $C_1, \ldots, C_l$, $l \geq 3$, and there are disjoint paths from s to t in G through each component, since G is biconnected. Assume some vertices of a component C_i, $i \in \{1, \ldots, l\}$, neither containing the vertex u nor v, i.e., $C_i \neq C_u$ and $C_i \neq C_v$, are contained in G'. If there is a path from s to t through these vertices then P would not separate G'. If there is no such path then G' cannot be biconnected. Hence, vertices of C_i cannot be contained in G' and therefore $G' \subseteq G^S_P$. □

Corollary 3.1. *Let P be a potential separator of G and $\mathcal{S} = \{S_1, \ldots, S_l\}$ the set of critical separators that cross P. The graph $G^*_P = G[\bigcap_{S \in \mathcal{S}} V(G^S_P)]$ is the maximal biconnected subgraph of G with separator P.*

In other words: The graph G^*_P is the subgraph in which for all critical separators S_i that cross P the vertices in connected components of $G \setminus S_i$ neither containing u nor v are removed. This fact allows us to decompose biconnected partial 2-trees at the potential separators and can be used algorithmically to compute partial solutions for well-defined subgraphs. For a potential separator $P = \{u, v\}$ and a distinguished edge $r \in E(G^*_P)$, $r \neq uv$, we say P *splits* G into the two subgraphs G^r_{uv} and $\widetilde{G^r_{uv}}$, referred to as *split graphs*. Let $C_1, \ldots, C_l$ be the connected components of $G^*_P \setminus P$ and w.l.o.g. let C_1 contain at least one endpoint of the distinguished edge r. Then $\widetilde{G^r_{uv}}$ denotes the subgraph $G[V(C_1) \uplus \{u, v\}]$ and G^r_{uv} refers to $G[\biguplus_{i=2}^{l} V(C_i) \uplus \{u, v\}]$, where u and v are called *base vertices*. In case of a potential separator P that is not a separator of G, i.e., $G^*_P \neq G$, the operation is referred to as *shear split*.

Figure 3.8 illustrates the different split operations. When the vertices u, v are not a potential separator, but adjacent, G^r_{uv} is defined as the edge $e = uv$, if $e \neq r$, and the graph $(V, E \setminus e)$ otherwise. Note that we may assume split graphs to be biconnected which can be achieved by inserting a virtual edge between the two base vertices as illustrated by dashed lines in Figure 3.8. The split graphs shown in Figure 3.8(b) are determined by the potential separator $P = \{b, g\}$ that is not a separator of G. Note that the critical separators $S_1 = \{a, d\}$ and $S_2 = \{d, e\}$ both cross P and hence the vertices h and f are not contained in G^*_P and the split graphs associated with P.

3.4.3 A Polynomial-Time Algorithm for 2-MCIS

We present a polynomial-time algorithm for 2-MCIS that is based on the SP-tree data structure and considers the split graphs of the input graphs defined by potential separators. Let $\Upsilon(v) = \{\mu \in V(\mathcal{T}) \mid u \in V(S_\mu), \mu(v) = u\}$ be the set of *allocation nodes* of a vertex v, i.e., the nodes of $\mathcal{T}$ whose skeleton contains a representative of v. We define the *shear path* $P(u, v)$ as the shortest path in the SP-tree between an S-node $\mu_1 \in \Upsilon(u)$ and an S-node $\mu_l \in \Upsilon(v)$. Lemma 3.3 characterizes the potential separators of a biconnected partial 2-tree by means of the SP-tree.

Lemma 3.3. *Let $P(u, v) = (\mu_1, \nu_1, \ldots, \nu_{l-1}, \mu_l)$ be a shear path, then the set $T = \{u, v\}$ is*

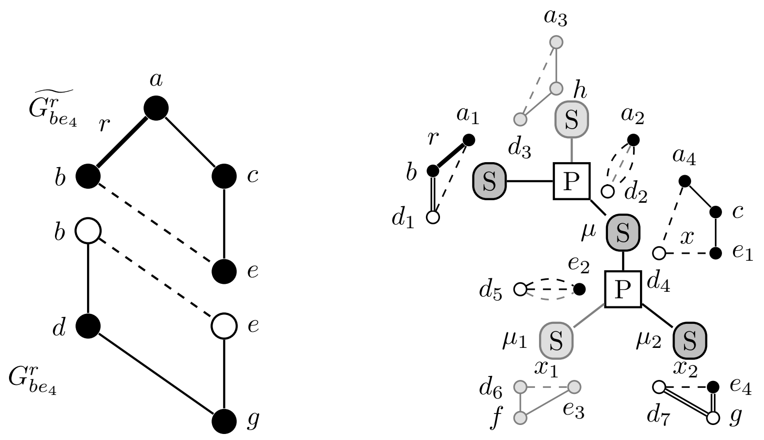
(i) a potential separator of G if and only if there is no P-node ν_i, $i \in \{1, \ldots, l-1\}$, with S_{ν_i} containing a real edge,

(ii) a separator of G if and only if $l = 1$.

In case (i), T is crossed by the critical separators $V(S_{\nu_i})$, $i \in \{1, \ldots, l-1\}$.

Proof. The structure of the SP-tree assures that all compulsive separators that cross T are associated with P-nodes on the shear path. Note that the vertices of these separators must be contained in a biconnected subgraph separated by T. If there is a P-node ν such that S_ν contains a real edge, the edge must also be contained in the induced subgraph and T cannot be a potential separator. According to Lemma 3.1 other P-nodes correspond to critical separators. If the length of the shear path is 1, both vertices of T are contained in the skeleton of the same S-node, which is a cycle, and therefore T is a separator. □

The maximal biconnected subgraph G^*_P of G that is separated by a potential separator $P = \{u, v\}$ of G can be obtained based on the shear path by keeping the components determined by μ_i and μ_{i+1} at each critical separator $V(S_{\nu_i})$. Note that a representative contained in a skeleton graph is associated with a unique node of the SP-tree. We extend the definition of split paths and split graphs to representatives from skeleton graphs of S-nodes. Given two representatives $P = \{u, v\}$, the end vertices of the shear path $P(u, v)$ are the



(a) Subgraphs (b) SP-tree and skeleton graphs

Figure 3.9: The graph $\widetilde{G^r_{be_4}}$ shown with filled black vertices and edges in (b) represents the vertices already considered and $G^r_{be_4}$, non-filled in (b), the unmapped vertices of the graph G, cf. Figure 3.6(a). Parts no longer considered for solutions are shown in gray. In the example we have $\text{childS}(x) = \{\mu_1, \mu_2\}$, $\text{parentS}(d_7) = \text{parentS}(d_6) = d_4$ and $\mu(a) = \mu(a_i) = a_4, i \in \{1, \ldots, 4\}$.

unique allocation nodes of u and v. We can still use Lemma 3.3 to determine if P is a potential separator and define the graph G^*_P and the split graphs accordingly based on the critical separators. Further split graphs emerge for representatives u, v in a skeleton S_μ, where μ is an S-node with $\text{ref}(\mu) = uv$, e.g., in Figure 3.9(b) the split graph $G^r_{d_7 e_4}$ would consist of S_μ. Note that all split graphs defined by potential separators can as well be obtained using certain representatives. Figure 3.9 illustrates a shear split defined by representatives as well as the corresponding split graphs and provides an example for the additional notation: For a virtual edge e in the skeleton of an S-node we denote the children of the P-node pertinent to e by $\text{childS}(e)$. For a vertex v that is an endpoint of the reference edge in a skeleton S_μ we refer to the representative of v in the next S-node on the path to the root by $\text{parentS}(v)$.

Algorithm 3.4 computes the SP-tree for both input graphs G, H and implicitly extends a partial mapping ψ between split graphs step-by-step. In each state the graphs $\widetilde{G^r_{uv}}$ and $\widetilde{H^{r'}_{u'v'}}$ are already processed and the mapping is extended to the unexplored subgraphs G^r_{uv} and $H^r_{u'v'}$, where $\psi(u) = u'$ and $\psi(v) = v'$. The main procedure starts with pairs of S-nodes, maps two edges of their skeleton graphs and roots the SP-trees at these edges by the function Root. Once set in the main procedure, the roots remain unchanged in all recursive calls to subprocedures, which make use of the induced parent-child relationship. Initially edges of the skeleton graphs are considered only if their endpoints are adjacent in the input graph. Note that this includes all

Algorithm 3.4: 2-MCIS

```
Input  : Biconnected partial 2-trees $G$ and $H$.
Output : Order of a maximum common biconnected subgraph.
1  $\mathcal{T} \leftarrow \mathcal{T}(G)$; $\mathcal{U} \leftarrow \mathcal{T}(H)$        ▷ Compute SP-tree decomposition
2  $mcs \leftarrow 0$
3  forall the $(\mu, \mu') \in V_S(\mathcal{T}) \times V_S(\mathcal{U})$ do        ▷ Pairs of series components
4      $r \leftarrow$ arbitrary edge $uv$ in $S_\mu$ with $uv \in E(G)$
5      Root$(\mathcal{T}, r)$        ▷ Root SP-tree $\mathcal{T}$ at the edge $r$
6      forall the edges $r' = u'v'$ in $S_{\mu'}$ with $r' \in E(H)$ do
7          Root$(\mathcal{U}, r')$        ▷ Root SP-tree $\mathcal{U}$ at the edge $r'$
8          $p \leftarrow$ McisSeries$(u, v, u', v')$
9          $q \leftarrow$ McisSeries$(u, v, v', u')$        ▷ Alternative edge mapping
10         $mcs \leftarrow \max\{mcs, p, q\}$
11 return $mcs + 2$
```

real edges contained in the skeleton graph as well as the virtual edges with a pertinent P-node ν such that S_ν contains a real edge. In case of a virtual edge the subgraph represented by the pertinent P-node is not considered subsequently. This is admissible since common subgraphs containing parts of these components can be obtained starting with root edges included in these components.

Ongoing with this mapping the procedure McisSeries essentially follows the cyclic skeleton graphs of the two S-nodes simultaneously and successively extends the mapping by the next unmapped vertices. Let μ be an S-node with reference edge ts and a skeleton graph S_μ consisting of a cycle $(c_k, c_1, \dots, c_k)$, where $c_k = t$ and $c_1 = s$. The function Next(v, S_μ) returns the edge $(v = c_i, c_j)$, where $j = i + 1 \mod k$, i.e., the next edge in the cyclic order given by the skeleton graph. The direction in which the cycle is traversed is determined by the ordering of the parameters of McisSeries. Different cases apply based on the type of edges that are mapped. The basic case is that the edges can be matched (line 11). Then the size of the maximum common subgraph depends on (i) MatchEdge, i.e., the maximum common subgraph of the split graphs associated with the edges and (ii) the result of the recursive call McisSeries with the new base vertices. Furthermore, the size of the mapping is increased by one for the vertex w, which is mapped to w'. When the next vertex corresponds to the first base vertex, i.e., the split graph consists of a single edge, the recursion ends (line 8). However, if this does not hold for both split graphs, the mapping cannot be completed as indicated by the return value $-\infty$ (line 10). Assume the edge under consideration is virtual and the vertices are not adjacent in G, then a critical separator is reached (line 12). In this case the algorithm recursively proceeds with a representative of the second base vertex from a skeleton of a different S-node that is a child of the P-node

Algorithm 3.5: Comparing split graphs of series components.

```
Input  : Base vertices $u, v$ of $G$, $v \in V(S_\mu)$ and $u', v'$ of $H$,
         $v' \in V(S_{\mu'})$.
Output : Order of a 2-MCIS of $G^r_{uv}$ and $H^{r'}_{u'v'}$ under the restriction
         that $u$ is mapped to $u'$ and $v$ to $v'$.

Procedure MCISSERIES($u, v, u', v'$)
    $e = vw \leftarrow$ NEXT($v, S_\mu$)                              ▷ Next edge in $S_\mu$
    $e' = v'w' \leftarrow$ NEXT($v', S_{\mu'}$)                          ▷ Next edge in $S_{\mu'}$
    if $e = \text{ref}(\mu)$ then                                ▷ Return to parent S-node
        return MCISSERIES($u, \text{parentS}(v), u', v'$)
    if $e' = \text{ref}(\mu')$ then                              ▷ Return to parent S-node
        return MCISSERIES($u, v, u', \text{parentS}(v')$)
    if $w = u$ and $w' = u'$ then                               ▷ Completed skeleton
        return MATCHEDGE($v, w, v', w'$)
    if $w = u$ xor $w' = u'$ then                               ▷ Incompletable mapping
        return $-\infty$
    $mcs \leftarrow$ MATCHEDGE($v, w, v', w'$) + MCISSERIES($u, w, u', w'$) + 1
    if $e \notin E(G)$ or $e' \notin E(H)$ then              ▷ Consider critical separators
        if $e \in E(G)$ then $M \leftarrow \{\mu\}$ else $M \leftarrow \text{childS}(e)$
        if $e' \in E(H)$ then $M' \leftarrow \{\mu'\}$ else $M' \leftarrow \text{childS}(e')$
        forall the $(\eta, \eta') \in M \times M'$ do
            $p \leftarrow$ MCISSERIES($u, \eta(v), u', \eta'(v')$)        ▷ Perform shear split
            $mcs \leftarrow \max\{mcs, p\}$
    return $mcs$
```

pertinent to the edge. All possible pairs of such S-nodes are considered and the best solution is selected (line 17). Note that it is eventually required to go back to the previous S-node to finally complete a mapping. This step is realized when the reference edge is reached in line 4 and 6, respectively.

The function MATCHEDGE is called whenever the mapping is extended. Note that the parameters are adjacent vertices from skeleton graphs and the corresponding edges may be real or virtual. Assume a given edge is virtual in the skeleton, then an edge between these vertices may or may not exist in G. We do not obtain a valid mapping between induced subgraphs if such an edge exists in one of the graphs, but not in the other (line 2). However, we can still map a virtual edge to a real edge if the skeleton of the pertinent P-node contains a real edge (line 4). In this case the subgraph represented by the descendants of the P-node is not part of the common subgraph. Finally, assume both edges are virtual, then the endpoints are 2-separators, see (SP1), and the 2-MCIS between the associated unmapped connected components has to

Algorithm 3.6: Comparing split graphs associated with edges.

```
Input  : Vertices $u, v, u', v'$ with $uv \in E(S_\mu)$ and $u'v' \in E(S_{\mu'})$.
Output : Order of a 2-MCIS of $G^r_{uv}$ and $H^{r'}_{u'v'}$ under the restriction
         that $u$ is mapped to $u'$ and $v$ to $v'$.

Procedure MatchEdge($u, v, u', v'$)                    ▷ Let $e = uv$, $e' = u'v'$
    if $e \in E(G)$ xor $e' \in E(H)$ then
        return $-\infty$                                ▷ Subgraph not induced
    if $e$ is real in $S_\mu$ or $e'$ is real in $S_{\mu'}$ then
        return 0                                       ▷ Valid mapping
    $M \leftarrow$ childS($e$);  $M' \leftarrow$ childS($e'$)
    forall the $m = (\eta, \eta') \in M \times M'$ do  ▷ Pairs of S-node children
        $w(m) \leftarrow$ McisSeries($\eta(u), \eta(v), \eta'(u'), \eta'(v')$)
    $p \leftarrow$ MwbMatching($M, M', w$)  ▷ Compute max. weight matching
    if $p = 0$ and $e \notin E(G)$, $e' \notin E(H)$ then
        return $-\infty$                                ▷ Not biconnected
    else return $p$
```

be added to the result. To this end, MwbMatching is performed between the children of the pertinent P-nodes (line 8), where the edge weights w are determined by McisSeries on pairs of the associated split graphs (line 7). Note that if the result of the matching is 0 and there is no edge between the base vertices, there is no path between the base vertices through the split graph considered. Since in this case the common subgraph would not be biconnected, the value $-\infty$ is returned (line 10). Otherwise the result corresponds to the size of the matching (line 11). Note that for two edges $e = uv$ and $e' = u'v'$ the result of MatchEdge(u, v, u', v') may differ from MatchEdge(u, v, v', u') since the ordering of the parameters determines the mapping of the base vertices. The two base vertices also contribute to the size of the solution. These are taken into account by the main procedure (Algorithm 3.4, line 11), but not in the procedures McisSeries and MatchEdge handling pairs of split graphs.

Analysis

We argue that Algorithm 3.4 solves 2-MCIS in polynomial time. A biconnected subgraph G' can be obtained from a biconnected partial 2-tree only by deleting components of $G \setminus S$ for compulsive separators S. A separator may remain compulsive for the subgraph, either because the two vertices are adjacent in G or because $G' \setminus S$ has at least three components. These cases are handled for the two input graphs by the procedure MatchEdge. If the separator does not remain compulsive, there are separators of G' that do not separate G, but are potential separators of G. Algorithm 3.4 considers all possible potential separators of both input graphs by building all possible valid shear paths in McisSeries, cf. Lemma 3.3. Furthermore, it considers the associated split graphs, which is sufficient according to Corollary 3.1.

In order to analyze the running time we first consider the number of possible split graphs.

Lemma 3.4. *Let G be a biconnected partial 2-tree and $n = |G|$. The number of split graphs of G is $\mathcal{O}(n^2)$.*

Proof. Each split graph G^r_{uv} is defined by a potential separator $P = \{u, v\}$ and a deleted component that is determined by the edge r. There are $\mathcal{O}(n^2)$ potential separators P. Let $c(P)$ denote the number of connected components of $G^*_P \setminus P$. For each potential separator the number of split graphs is $2c(P)$. We distinguish between potential separators P with $c(P) = 2$ and those with $c(P) > 2$. Note that potential separators that are no separators of G always yield exactly two connected components. In total $\mathcal{O}(n^2)$ split graphs can be caused by potential separators P with $c(P) = 2$. According to Lemma 3.1 there is a one-to-one correspondence between separators P with $c(P) > 2$ and P-nodes in the SP-tree. Since the number of P-nodes is well-known to be $\mathcal{O}(n)$ and $c(P) < n$, the number of split graphs caused by these separators also is $\mathcal{O}(n^2)$. □

Theorem 3.2. *The problem 2-MCIS can be solved in time $\mathcal{O}(n^6)$, where $n = \max\{|G|, |H|\}$.*

Proof. Each call of the recursive methods McisSeries and MatchEdge computes 2-MCIS for two split graphs defined by the parameter list. We can easily transform the methods into dynamic programming algorithms filling a table indexed by split graphs. Thus, as soon as a cell of the table has been computed, every successive call for the same pair of split graphs can be answered in $\mathcal{O}(1)$. Note that the number of vertices and edges, either virtual or real, in skeleton graphs is linear in the number of vertices of the input graph. According to Lemma 3.4 a table of size $\mathcal{O}(n^4)$ is sufficient. We consider the function McisSeries first. Assume that when computing McisSeries the results for all smaller split graphs are already determined and stored in the table. The running time of each call is then dominated by the loop in line 15 that requires $\mathcal{O}(n^2)$ time. Therefore the total running time spend in McisSeries is $\mathcal{O}(n^6)$. The function MatchEdge involves MwbMatching which can be solved in $\mathcal{O}(n^3)$ by the Hungarian algorithm, cf. Section 2.4. Non-trivial matching problems must only be solved when the given edges e and e' both are virtual. Since there can be at most $\mathcal{O}(n^2)$ such pairs the total running time spend in MatchEdge is $\mathcal{O}(n^5)$. The total running time of a dynamic programming variant of Algorithm 3.4 is then dominated by the function McisSeries and consequently $\mathcal{O}(n^6)$. □

3.4.4 Solving 2-MCIS in Outerplanar Graphs

The outerplanar graphs are a subset of the series-parallel graphs and exhibit special characteristics that can be exploited algorithmically. The problem 2-MCES was solved in the context of BBP-MCCES in outerplanar graphs, but not carefully analyzed, see Section 3.3.3 for the best known bounds on the running time for this related problem.

We first show that the running time of Algorithm 3.4 presented in Section 3.4.3 is cubic when the input graphs are restricted to outerplanar graphs. Then we develop a new algorithm, which is specifically designed for outerplanar graphs and achieves quadratic time complexity. Being on par with the best known subgraph isomorphism algorithm for these graphs in terms of asymptotic complexity, the approach clearly beats the running time of known BBP-MCCES algorithms.

Tighter Bounds for the General Approach

The following property is the key to our improved analysis of Algorithm 3.4 when applied to outerplanar graphs.

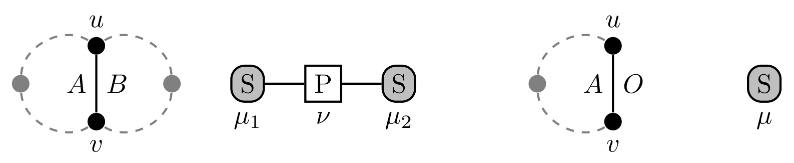
Lemma 3.5. *Let G be a biconnected partial 2-tree, G is outerplanar if and only if all P-nodes of $\mathcal{T}(G)$ have degree two.*

Proof. It is well-known that a graph is outerplanar if and only if it does not contain K_4 or $K_{2,3}$ as a minor (Diestel, 2005). Assume there is a P-node ν in $\mathcal{T}$ with degree at least 3. Then each adjacent S-node contains a path connecting the two vertices in $V(S_\nu)$ and consequently G contains $K_{2,3}$ as minor and is not outerplanar. If G contains $K_{2,3}$ as minor then the two vertices of degree three are a compulsive separator of G (cf. Proof of Lemma 3.1) and lead to a P-node with at least three adjacent S-nodes, each of which represents a component containing exactly one of the vertices with degree two. □

According to Lemma 3.5 combined with Lemma 3.1 there cannot be any critical separators in outerplanar graphs and, hence, every potential separator also is a separator. Therefore parts of Algorithm 3.4 become trivial and we obtain the following result.

Theorem 3.3. *Algorithm 3.4 solves 2-MCIS in outerplanar graphs in time $\mathcal{O}(n^3)$, where $n = \max\{|G|, |H|\}$.*

Proof. Since there are no critical separators, lines 13–17 of MCISSERIES are never reached. The sets M and M' in line 5 of MATCHEDGE always consist of single S-nodes, because all P-nodes have degree two. Therefore both functions are computed in $\mathcal{O}(1)$ except the running time required for recursive calls. Algorithm 3.4 starts with pairs of edges and causes $\mathcal{O}(n^2)$ calls of MCISSERIES. Each again involves $\mathcal{O}(n)$ recursive calls of the procedures MCISSERIES and MATCHEDGE leading to a total running time of $\mathcal{O}(n^3)$. □



(a) Two faces sharing an edge (b) The outer face

Figure 3.10: In (a) an edge uv is shown that is incident to two faces A and B in an outerplanar embedding. In the associated SP-tree each face corresponds to an S-node and the edge to a P-node adjacent to both S-nodes. In (b) the situation for an edge incident to the outer face O is depicted.

An Improved Algorithm for Outerplanar Graphs

We can obtain an even better algorithm for 2-MCIS in outerplanar graphs based on the observation that the whole matching process in outerplanar graphs is uniquely determined by certain starting parameters. Our approach exploits the fact that biconnected outerplanar graphs have a unique outerplanar embedding in the plane, up to the mirror image (see, e.g., Sysło, 1982). Thus, every edge is incident to exactly two faces that are uniquely defined. We say a face is mapped by an isomorphism ψ when all the vertices bordering the face are mapped by ψ. 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 ψ between biconnected induced subgraphs. Assume the edge uv is incident to the faces A and B in G and $u'v'$ is incident to A' and B' in H. At least one face incident to uv must be mapped by ψ, since the common subgraph must be biconnected. For the sake of simplicity of the case distinction, we also associate the two faces that are not mapped if applicable. The isomorphism ψ may map the endpoints of the edges uv and $u'v'$ in two different ways—just as the two incident faces. We assume that there are arbitrary total orders $\prec$ on the vertices and faces, respectively. W.l.o.g. assume that $u \prec v$, $u' \prec v'$ and $A \prec B$, $A' \prec B'$. Then we can distinguish the following cases:

Type 1:	$u \mapsto u'$,	$v \mapsto v'$	and	$A \mapsto A'$,	$B \mapsto B'$
Type 2:	$u \mapsto v'$,	$v \mapsto u'$	and	$A \mapsto A'$,	$B \mapsto B'$
Type 3:	$u \mapsto u'$,	$v \mapsto v'$	and	$A \mapsto B'$,	$B \mapsto A'$
Type 4:	$u \mapsto v'$,	$v \mapsto u'$	and	$A \mapsto B'$,	$B \mapsto A'$

Given an isomorphism ψ between biconnected common induced subgraphs that maps the two endpoints of an edge e, let the function $\text{type}(e, \psi) \in \{1, \ldots, 4\}$ determine the type of the mapping as above. The following result is the key to obtain an efficient 2-MCIS algorithm.

Lemma 3.6. *Let ψ and ψ' 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 ψ and ψ', then

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

Proof. It is obvious that the direction $\Longleftarrow$ is correct. We prove the implication $\Longrightarrow$: Since the common subgraph is required to be biconnected, ψ and ψ' must map at 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 two vertices lying on this cycle is fixed, the mapping of all vertices on the border of the face is unambiguously determined. Every mapping of connected subgraphs can be obtained by adding further adjacent vertices to an initial mapping. Since the common subgraph here is required to be biconnected, the 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 and all isomorphisms between biconnected common induced subgraphs that map e according to the given type can be obtained. Consequently, $\psi(u) = \psi'(u)$ holds for all $u \in \text{dom}(\psi) \cap \text{dom}(\psi')$. Since ψ and ψ' are maximal it is not possible that one of them can be extended and, hence, we must have $\text{dom}(\psi) = \text{dom}(\psi')$ and the result follows. ☐

We develop an efficient algorithm for 2-MCIS by means of SP-trees, which conveniently encode the adjacency relation between the faces of outerplanar graphs. Figure 3.10 illustrates the correspondence between faces and S-nodes in the SP-tree. Each face is associated with an S-node with exception of the outer face. SP-trees easily allow to computationally distinguish the above cases and the type distinction. The isomorphism ψ induces a mapping of faces of G to faces of H which corresponds to a mapping of S-nodes of $\mathcal{T}(G)$ to S-nodes of $\mathcal{T}(H)$. This allows to (i) determine $\text{type}(e, \psi)$ for a given edge e and isomorphism ψ as well as (ii) to initialize a mapping of an edge according to a specific type fixing the mapping of incident faces. Note that for some edge pairs not all four types of mappings are possible. Clearly, an isomorphism between biconnected subgraphs can only map the vertices of faces bordered by the same number of vertices. We say two S-nodes μ and μ' representing faces are *compatible* if $|S_\mu| = |S_{\mu'}|$. The skeleton graphs of two compatible S-nodes can always be mapped. We say that 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 occur in S-nodes μ of $\mathcal{T}(G)$ and μ' of $\mathcal{T}(H)$ that are compatible. Note that it is possible to map an edge that occurs in a single S-node μ, cf. Figure 3.10(b), to an edge that occurs in two S-nodes μ_1, μ_2, cf. Figure 3.10(a). Assume μ is compatible with μ_1 and μ_2, then all four types are valid. However, if μ is compatible with μ_1, but not with μ_2 and $\mu_1) \prec \mu_2$, then only types 1 and 2 are valid.

The proof of Lemma 3.6 constructively shows how to obtain a maximal solution given an edge of both input graphs together with the mapping of their endpoints and incident faces. This approach is realized by the procedures MatchCycle and MatchEdge of Algorithm 3.7. The function MatchCycle matches the skeleton graph of an S-node, i.e., all vertices on the border of a face. For each edge on the border the function MatchEdge is called, which extends the mapping beyond the current face to an adjacent face if possible. In case of two virtual edges, the other S-nodes of the pertinent P-nodes are mapped by the function MatchCycle if they are compatible. Combining the mappings returned by these procedures yields the unique maximal isomorphism ψ that maps the two given edges according to the specified type. The running time to compute a maximal solution ψ by these procedures is $\mathcal{O}(|\psi|)$.

The main procedure of Algorithm 3.7 uses a table $D(e, e', t)$, $e \in E(G)$, $e' \in E(H)$ and $t \in \{1, \dots, 4\}$, storing the size of the unique maximal isomorphism between biconnected induced subgraphs containing a type t mapping of the edge e to the edge e'. The size of the table is $4\,\|G\|\,\|H\| = \mathcal{O}(nm)$, where $n = |G|$ and $m = |H|$. This bound is valid, since $\|G\| = \mathcal{O}(|G|)$ holds for every planar graph G and, hence, in particular for outerplanar graphs. 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. The size of this isomorphism ψ is then stored in D for all pairs of edges mapped by ψ considering the type of the mapping. Keeping these values allows to avoid generating the same isomorphism multiple times. Finally, the maximum size of any maximal isomorphisms is returned, which is the size of a 2-MCIS.

The time to compute SP-trees is linear (Gutwenger and Mutzel, 2001) and therefore does not dominate the running time of the algorithm. The main procedure loops over the $\mathcal{O}(n^2)$ pairs of edges and the four possible mappings for each pair. A mapping ψ is computed by the procedures MatchCycle and MatchEdge in time $\mathcal{O}(|\psi|) = \mathcal{O}(n)$. Improved analysis gives the following result.

Theorem 3.4. *Algorithm 3.7 solves 2-MCIS in outerplanar graphs in time $\mathcal{O}(n^2)$, where $n = \max\{|G|, |H|\}$.*

Proof. We allocate the costs for calls of MatchCycle and MatchEdge to the cells of the table D. An isomorphism containing k edges is computed in time $\mathcal{O}(k)$ and as a result exactly k cells of the table D are filled with a value. The value of a cell is computed at most once: Line 4 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

Algorithm 3.7: 2-MCIS in outerplanar graphs

```
Input  : Biconnected outerplanar graphs $G$ and $H$.
Output : Order of a 2-MCIS.
Data   : Table $D(e, e', t)$, $e \in E(G)$, $e' \in E(H)$, $t \in \{1, \dots, 4\}$ storing
         the size of a 2-MCIS $\psi$ with $e \mapsto e'$ and type$(e, \psi) = t$.

$\mathcal{T} \leftarrow \mathcal{T}(G)$; $\mathcal{U} \leftarrow \mathcal{T}(H)$          ▷ Compute SP-tree decomposition
forall the edge pairs $(uv, u'v') \in E(G) \times E(H)$ do
    for $t \leftarrow 1$ to $4$ do
        if type $t$ valid and $D(uv, u'v', t)$ undefined then
            ▷ Let $\psi$ and $\mu, \mu'$ be initialized according to type $t$.
            $\psi \leftarrow \psi \uplus$ MatchEdge$(u, v, \psi(u), \psi(v), \mu, \mu')$
            $\psi \leftarrow \psi \uplus$ MatchCycle$(u, v, \psi(u), \psi(v), \mu, \mu')$
            [Output $\psi$]          ▷ Optionally output maximal isomorphism
            forall the edges $e \in E(G)$ mapped to $e' \in E(H)$ by $\psi$ do
                $D(e, e', \text{type}(e, \psi)) \leftarrow |\psi|$
return maximum entry in $D$

Procedure MatchCycle$(u, v, u', v', \mu, \mu')$
    Input  : Adjacent vertices $u, v \in V(S_\mu)$ and $u', v' \in V(S_{\mu'})$.
    Output : Extension of the isomorphism with $u \mapsto u'$ and $v \mapsto v'$
             fixed to the faces represented by $\mu$ and $\mu'$.
    Data   : Skeleton graphs $S_\mu$ and $S_{\mu'}$ containing the cycles
             $(v = v_0, \dots, v_k = u, v)$ and $(v' = v'_0, \dots, v'_k = u', v')$.
    $\psi_\mu \leftarrow \{v_i \mapsto v'_i \mid 1 \leq i < k\}$
    for $i \leftarrow 1$ to $k$ do
        $\psi_\mu \leftarrow \psi_\mu \uplus$ MatchEdge$(v_{i-1}, v_i, v'_{i-1}v'_i, \mu, \mu')$
    return $\psi_\mu$

Procedure MatchEdge$(u, v, u', v', \mu, \mu')$
    Input  : Adjacent vertices $u, v \in V(S_\mu)$ and $u', v' \in V(S_{\mu'})$.
    Output : Extension of the isomorphism with $u \mapsto u'$ and $v \mapsto v'$
             fixed to pertinent nodes of $uv \in E(S_\mu)$ and $u'v' \in E(S_{\mu'})$.
    if $uv$ virtual in $S_\mu$ and $u'v'$ virtual in $S_{\mu'}$ then
        ▷ Let childS$(uv) = \{\eta\}$ and childS$(u'v') = \{\eta'\}$.
        if $\eta$ and $\eta'$ are compatible then
            return MatchCycle$(u, v, u', v', \eta, \eta')$
    return $\emptyset$
```

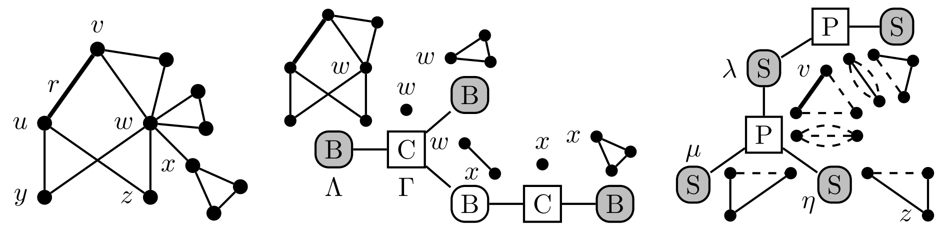
undefined cells according to Lemma 3.6. Therefore the total costs of the algorithm can be allocated to cells of D, such that each cell pays $\mathcal{O}(1)$. This proves that the total running time is bounded by the size of the table, which is $\mathcal{O}(n^2)$. □

We argue that the running time of Algorithm 3.7 essentially corresponds to the running time of the best known induced subgraph isomorphisms algorithm for biconnected outerplanar graphs: Sysło (1982) presented an algorithm that decides if G is isomorphic to an induced subgraph of H in $\mathcal{O}(tpr)$, where t is $\|H\| + 1$, p is the size of a face of G, which can be selected arbitrary, and r is the number of faces of size p in H. The best possible bound of this running time in terms of the number of vertices also is $\mathcal{O}(n^2)$, where $n = |H|$.

We can easily modify our algorithm to enumerate all maximal isomorphisms between biconnected common induced subgraphs of two biconnected outerplanar graphs without affecting the total running time by adding line 7. To enumerate all maximum solutions we may proceed as follows: First we run the unmodified version of Algorithm 3.7 once to obtain the size $W_{\max}$ of a maximum solution. Then we run a modified version that outputs an isomorphism ψ as soon as it is found only if it has size $W_{\max}$. This approach can as well be adapted to handle labeled graphs, which requires minor modifications (see Droschinsky et al., 2015, for further information).

3.5 Finding BBP-MCCIS in Series-Parallel Graphs

In this section we consider the task to find a maximum common connected induced subgraph that preserves the structure of blocks and bridges of the input graphs (BBP-MCCIS). The concept was introduced by Schietgat et al. (2007), who considered the related problem BBP-MCCES in outerplanar graphs. According to the definition in Section 3.3 a solution must fulfill the two conditions (BBP1) and (BBP2), which can as well be applied to series-parallel graphs, where different algorithmic challenges occur. In order to solve this problem, we make use of the following observation: Every two edges in an input graph that are not in the same block cannot be in the same block in any common subgraph. Hence, together with condition (BBP1) and (BBP2) we may infer that vertices in one block of G can only be mapped to vertices contained in the same block of H such that the mapped vertices form a biconnected common subgraph. This means that blocks and bridges can be considered separately to some extent and allows us to solve BBP-MCCIS by systematically decomposing the input graphs and solving the associated subproblems. We use the BC-tree data structure to decompose the connected input graphs into tree-like parts consisting of bridges and maximal biconnected subgraph, i.e., the blocks. Every block is then again decomposed by means of SP-trees and is handled by a modified version of the 2-MCIS algorithm presented in Section 3.4. Clearly, whenever two vertices of blocks are



(a) Input graph G (b) BC-tree $\mathfrak{T}(G)$ (c) SP-tree $\mathcal{T}_\Lambda = \mathcal{T}(S_\Lambda)$

Figure 3.11: A series-parallel graph (a) and its BC-tree (b) with the associated skeleton graphs. Block B-nodes have a gray background color, while bridge B-nodes are not filled. The SP-tree of the skeleton graph associated with the block B-node Λ is shown in (c) together with its skeleton graphs.

mapped that are both cutvertices of the input graphs, the components beyond these cutvertices must be considered for BBP-MCCIS. Solving MCCIS for the bridges that appear in the BC-tree is similar to the problem 1-MCIS and handled by an algorithm based on the ideas of the approach described in Section 3.2.

3.5.1 Decomposing Series-Parallel Graphs

We now consider connected series-parallel graphs that contain at least one edge. In the case that input graphs are disconnected, we treat all pairs of connected components individually. The problem is trivial when one input graph (or one connected component) consists of an isolated vertex. We first decompose graphs by means of BC-trees to distinguish between blocks and bridges.

Definition 3.5 (BC-tree). *Given a connected graph G with at least two vertices, let C denote the set of cutvertices,* Bl *the set of blocks,* Br *the set of bridges and $\mathcal{B} = \mathrm{Br} \uplus \mathrm{Bl}$. The* BC-tree *$\mathfrak{T} = \mathfrak{T}(G)$ is the tree with nodes $\mathcal{B} \uplus C$ and edges between nodes $B \in \mathcal{B}$ and $c \in C$ if and only if $c \in V(B)$.*

We refer to the nodes of $\mathfrak{T}$ representing cutvertices as C-nodes and distinguish between bridge and block B-nodes; given a BC-tree $\mathfrak{T}$, we refer to the three sets of nodes by $V_{\mathrm{C}}(\mathfrak{T})$, $V_{\mathrm{Br}}(\mathfrak{T})$ and $V_{\mathrm{Bl}}(\mathfrak{T})$, respectively. Each node Λ in a BC-tree has a skeleton graph S_Λ consisting of the vertices and edges of G represented by that node, i.e., a single vertex for C-nodes, two vertices connected by an edge for bridge B-nodes and a biconnected subgraph for block B-nodes, cf. Figure 3.11. SP-trees allow to decompose biconnected series-parallel graphs and can consequently be applied to the skeleton graphs of block B-nodes. In the following we associate an SP-tree $\mathcal{T}_\Lambda = \mathcal{T}(S_\Lambda)$ with every block B-node Λ, cf. Figure 3.11(c). For the readers convenience, we use Greek uppercase letters to refer to B- and C-nodes of the BC-tree, while Greek lowercase letters denote S- and P-nodes of an SP-tree, respectively.

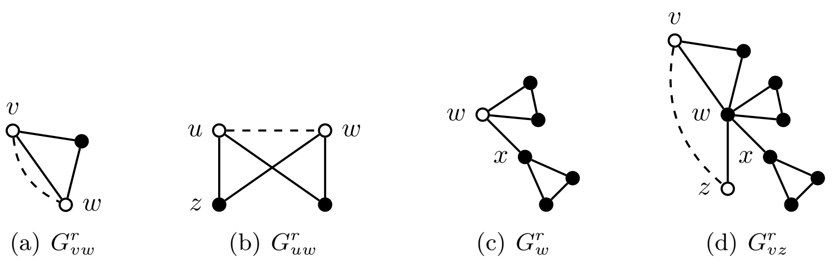


Figure 3.12: Some split graphs obtained from the graph depicted in Figure 3.11(a): In (a) and (b) the graph is split at different 2-separators, in (c) at a cutvertex and in (d) at a potential separator that does not separate G. Base vertices are not filled.

Our BBP-MCCIS algorithm computes the solution based on subproblems for well-defined subgraphs of the input graphs closely related to these two data structures. Note that the split graphs defined in Section 3.4 for biconnected series-parallel graphs were obtained from 2-separators since these graph contain no 1-separators, i.e., cutvertices. We generalize the notion of split graphs for connected series-parallel input graphs. Given a set of vertices S, let C_i^S denote the connected components of $G \setminus S$ and assume w.l.o.g. that at least one endpoint of a distinguished edge r is contained in C_1^S. Let $\mathcal{C}_S^r = \biguplus_{i\geq 2} V(C_i^S)$. We now consider the following split graphs: Let $S = \{v\}$, where v is a cutvertex, then G_v^r denotes the graph $G[\mathcal{C}_S^r \uplus S]$ with *base vertex v*. Let $S = \{u, v\}$ be a minimal separator, then G_{uv}^r denotes the graph $G[\mathcal{C}_S^r \setminus (\mathcal{C}_{\{u\}}^r \uplus \mathcal{C}_{\{v\}}^r) \uplus S]$; we say u and v are the base vertices of G_{uv}^r. Figure 3.12 shows an example of several split graphs obtained from the graph depicted in Figure 3.11(a).

Note that there is a one-to-one correspondence between split graphs obtained from cutvertices and specific *rooted subtrees* of the BC-tree of G. Let G_v^r be a split graph, consider the C-node Γ representing the base vertex v and the unique B-node Λ with $r \in E(S_\Lambda)$, then the tree rooted at Γ, where the child subtree containing Λ is deleted, is associated with the split graph. This is similar to the rooted subtrees defined in the classical approach for MCST, cf. Section 3.2. For example, the split graph depicted in Figure 3.12(c) corresponds to the subtree of the BC-tree shown in Figure 3.11(b) that is obtained by rooting the tree at Γ and deleting Λ.

For handling block B-nodes we build on the algorithm described in Section 3.4, which is based on split graphs closely related to the SP-tree data structure. The approach obtains additional split graphs from potential separators, which are not only the 2-separators of the graph G, but also the 2-separators of an induced biconnected subgraph of G. In a series-parallel graph all pairs of non-adjacent vertices that are contained in the same chordless cycle are potential separators. Note that the split graphs used here again can readily be obtained from a potential separator P based on the graph G_P^*,

Algorithm 3.8: BBP-MCCIS

```
Input  : Two non-empty series-parallel graphs $G$ and $H$.
Output : Order of a BBP-MCCIS of the graphs $G$ and $H$.
Data   : BC-trees $\mathfrak{T}$ and $\mathfrak{U}$ of $G$ and $H$, respectively; SP-trees $\mathcal{T}_\Lambda$ for
         every block B-node $\Lambda$ in $\mathfrak{T}$ and $\mathfrak{U}$.
$mcs \leftarrow -\infty$
forall the $(\Lambda, \Lambda') \in V_{\mathrm{Bl}}(\mathfrak{T}) \times V_{\mathrm{Bl}}(\mathfrak{U})$ do        ▷ Pairs of block B-nodes
    Root($\mathfrak{T}, \Lambda$); Root($\mathfrak{U}, \Lambda'$)
    forall the $(\mu, \mu') \in V_{\mathrm{S}}(\mathcal{T}_\Lambda) \times V_{\mathrm{S}}(\mathcal{T}_{\Lambda'})$ do        ▷ Pairs of S-nodes
        $r \leftarrow$ arbitrary $uv \in E(S_\mu) \cap E(G)$; Root($\mathcal{T}_\Lambda, r$)
        forall the edges $r' = u'v' \in E(S_{\mu'}) \cap E(H)$ do
            Root($\mathcal{T}_{\Lambda'}, r'$)
            $p \leftarrow$ BbpMcisSeries($u, v, u', v'$)
            $p \leftarrow$ BbpMcisSeries($u, v, v', u'$)        ▷ Alternative mapping
            $mcs \leftarrow \max\{mcs, p, q\}$
forall the $(\Lambda, \Lambda') \in V_{\mathrm{Br}}(\mathfrak{T}) \times V_{\mathrm{Br}}(\mathfrak{U})$ do        ▷ Pairs of bridge B-nodes
    Root($\mathfrak{T}, \Lambda$); Root($\mathfrak{U}, \Lambda'$)
    $r = uv \leftarrow E(S_\Lambda)$; $r' = u'v' \leftarrow E(S_{\Lambda'})$
    $p \leftarrow$ BbpMatchVertex($u, u'$) + BbpMatchVertex($v, v'$)
    $q \leftarrow$ BbpMatchVertex($u, v'$) + BbpMatchVertex($v, u'$)
    $mcs \leftarrow \max\{mcs, p, q\}$
return $\max\{1, mcs + 2\}$
```

cf. Corollary 3.1. For example, in Figure 3.11(a) the set $S = \{v, z\}$ would be a potential separator, but not a separator of the graph, since u remains connected to w in $G \setminus S$ through the vertex y. Consequently, this vertex cannot be contained in the subgraph that is separated by S. Applying the definition of split graph to G^*_P yields the split graph shown in Figure 3.12(d).

3.5.2 A Polynomial-Time Algorithm for BBP-MCCIS

We now present a polynomial-time algorithm, which solves BBP-MCCIS in series-parallel graphs based on the decomposition of input graphs by means of BC- and SP-trees. We apply the idea presented in Section 3.2 for MCST to the BC-trees and combine solutions for smaller subproblems between rooted subtrees by means of MwbMatching. In a similar manner the block B-nodes are handled by means of SP-trees, where certain additional technicalities must be considered as described in Section 3.4.

We assume that for the two given series-parallel graphs the associated BC- and SP-trees are already computed. Our approach computes the size of a BBP-MCCIS between the two input graphs by recursively dividing the problem into smaller subproblems defined between split graphs obtained from these data structures. The main procedure, presented as Algorithm 3.8, starts with all pairs of B-nodes in the BC-trees of the input graphs. Since blocks and bridges must be preserved, only pairs of block B-nodes (line 2–10) and pairs of bridge B-nodes (line 11–15) are considered:

In the former case, the algorithm loops over all possible pairs of S-nodes in the SP-trees of the two blocks. Two distinguished edges serve as starting point by fixing the mapping of their endpoints. The resulting subproblem then is handled by the procedure BbpMcisSeries. This part is similar to the main procedure of our 2-MCIS approach, see Algorithm 3.4. The main procedure does not take into account that the vertices u and u' (or v and v') may be cutvertices or that the edges r and r' could be virtual. These cases are not considered since the possible better result would be obtained anyway for a different starting configuration.

In the latter case, the edges associated with the bridge B-nodes serve as starting point of the mapping. Again, the two possible mappings of endpoints are considered and the solution is determined based on the return values of the procedure BbpMatchVertex, see Algorithm 3.10. Assume we have the bridges $r = uv$ of G and $r' = u'v'$ of H and map the vertex u to u' and v to v', then the result is obtained by computing the BBP-MCCIS between the split graphs G^r_u and $G^{r'}_{u'}$ and add the result for the opposing split graphs G^r_v and $G^{r'}_{v'}$. Finally, the procedure returns the best solution found and accounts for the two base vertices, which have been mapped, by adding two. It is possible that no solution has been found, e.g., when one graph consists of bridges only and the other merely of blocks. Note that a single vertex can always be mapped without violating (BBP1) and (BBP2). Hence, in this case the value 1 is returned (line 16), where we assume the input graphs to be non-empty.

Whenever we compute the BBP-MCCIS between two split graphs, we require that the mapping of base vertices is fixed and compute the maximum possible solution under this constraint. Whenever a procedure is called the BC-trees and SP-trees considered have been rooted by the procedure Root at distinct B-nodes and at distinct edges, respectively. Therefore, we can make use of the parent-child relationship between adjacent nodes. In particular, we may infer which parts of the graphs have already been considered, since these are associated with the branches containing the root. We will give the details of the procedures called by Algorithm 3.8 in the following.

The procedure BbpMcisSeries—detailed as Algorithm 3.9 for the sake of completeness—exactly corresponds to the procedure McisSeries with a single crucial difference in line 7 (highlighted in the pseudocode). Here the next vertex according to the cyclic order of the skeleton graph is mapped and the problem is divided into several subproblems. In addition to the subproblems considered for 2-MCIS the new procedure BbpMatchVertex is called in order to take cutvertices into account. We illustrate the subproblems and how they align with our definition of split graphs in the following by considering the example shown in Figure 3.11(a). Assume that the main

Algorithm 3.9: Variant of Algorithm 3.5 considering cutvertices.

```
Input  : Base vertices $u, v$ of $G$, $v \in V(S_\mu)$ and $u', v'$ of $H$,
         $v' \in V(S_{\mu'})$.
Output : Order of a BBP-MCCIS between $G^r_{uv}$ and $H^{r'}_{u'v'}$ under the
         restriction that $u$ is mapped to $u'$ and $v$ to $v'$.
Procedure BbpMcisSeries($u, v, u', v'$)
    $e = vw \leftarrow$ Next($v, S_\mu$)                                   ▷ Next edge in $S_\mu$
    $e' = v'w' \leftarrow$ Next($v', S_{\mu'}$)                            ▷ Next edge in $S_{\mu'}$
    if $e = \text{ref}(\mu)$ then return BbpMcisSeries($u, \text{parentS}(v), u', v'$)
    if $e' = \text{ref}(\mu')$ then return BbpMcisSeries($u, v, u', \text{parentS}(v')$)
    if $w = u$ and $w' = u'$ then return BbpMatchEdge($v, w, v', w'$)
    if $w = u$ xor $w' = u'$ then return $-\infty$                    ▷ Incompletable mapping
    $mcs \leftarrow$ BbpMatchEdge($v, w, v', w'$) + BbpMatchVertex($w, w'$)
            +BbpMcisSeries($u, w, u', w'$) + 1
    if $e \notin E(G)$ or $e' \notin E(H)$ then                      ▷ Consider critical separators
        if $e \in E(G)$ then $M \leftarrow \{\mu\}$ else $M \leftarrow \text{childS}(e)$
        if $e' \in E(H)$ then $M' \leftarrow \{\mu'\}$ else $M' \leftarrow \text{childS}(e')$
        forall the $(\eta, \eta') \in M \times M'$ do
            $p \leftarrow$ BbpMcisSeries($u, \eta(v), u', \eta'(v')$)
            $mcs \leftarrow \max\{mcs, p\}$
    return $mcs$
```

procedure has started with a pair of block B-nodes, where the first is $\Lambda \in V(\mathfrak{T})$. Further, assume that the S-node μ has been selected from the SP-tree associated with the block, which has been rooted at the edge r. Then, the next vertex in S_μ is w, which is connected to the base vertex v by the virtual edge vw. The situation corresponds to the case that is handled in line 7 of the procedure BbpMcisSeries. The problem here is divided into three subproblems: First, if possible, the part of the input graph represented by the virtual edge must be considered, i.e., the split graph G^r_{vw}, see Figure 3.12(a). This is done by calling the procedure BbpMatchEdge. Second, the vertex w is a cutvertex in G and the split graph G^r_w, see Figure 3.12(c), must be taken into account whenever w is mapped to a cutvertex in H. This is handled by the new procedure BbpMatchVertex, which is explained in detail later. Finally, the remainder of the skeleton S_μ must be considered, which is done by the recursive call of BbpMcisSeries, where the vertex w replaces v as base vertex. Consequently, this procedure tries to map the split graph G^r_{uw}, see Figure 3.12(b). Since the vertex w is mapped, one is added to the sum over the values returned by these procedures.

The procedure BbpMatchEdge works exactly as MatchEdge (Algorithm 3.6) used as part of the 2-MCS algorithm with the difference that in

Algorithm 3.10: Comparing split graphs associated with cutvertices.

```
Input  : Two vertices $u \in V(G)$, $u' \in V(H)$.
Output : Order of a BBP-MCCIS between $G^r_u$ and $H^{r'}_{u'}$ under the
         restriction that $u$ is mapped to $u'$; 0 if $u$ and $u'$ not both are
         cutvertices.

Procedure BbpMatchVertex($u, u'$)
    if $u$ and $u'$ not both are cutvertices then
        return 0
    $M \leftarrow \text{childB}(u)$; $M' \leftarrow \text{childB}(u')$        ▷ Get B-node children
    forall the $m = (\Lambda, \Lambda') \in M \times M'$ do
        if $\Lambda$ and $\Lambda'$ both are bridge B-nodes then
            $uv \leftarrow E(S_\Lambda)$; $u'v' \leftarrow E(S_{\Lambda'})$        ▷ Get associated bridges
            $w(m) \leftarrow$ BbpMatchVertex($v, v'$) $+ 1$
        else if $\Lambda$ and $\Lambda'$ both are block B-nodes then
            ▷ Get S-nodes containing an edge incident to the base vertex
            $N \leftarrow \{\mu \in V_\mathrm{S}(\mathcal{T}_\Lambda) \mid \exists uv \in E(S_\mu) \cap E(G)\}$
            $N' \leftarrow \{\mu' \in V_\mathrm{S}(\mathcal{T}_{\Lambda'}) \mid \exists u'v' \in E(S_{\mu'}) \cap E(H)\}$
            forall the $(\mu, \mu') \in N \times N'$ do        ▷ Pairs of relevant S-nodes
                $r \leftarrow$ arbitrary edge $uv \in E(S_\mu) \cap E(G)$; Root($\mathcal{T}_\Lambda, r$)
                forall the edges $r' = u'v' \in E(S_{\mu'}) \cap E(H)$ do
                    Root($\mathcal{T}_{\Lambda'}, r'$)
                    $p \leftarrow$ BbpMcisSeries($u, v, u', v'$)
                    $w(m) \leftarrow \max\{w(m), p\}$
        else $w(m) \leftarrow -\infty$        ▷ Non BBP matching
    return MwbMatching($M, M', w$)
```

line 7 the weights are determined by the adapted procedure BbpMcisSeries, of course. The procedure constructs maximum weight bipartite matching instances, cf. Problem 2.5, to determine the mapping between connected components obtained from 2-separators.

The new procedures BbpMatchVertex relies on the same approach to find the best possible mapping between split graphs obtained from 1-separators, i.e., cutvertices. The method is presented as Algorithm 3.10 and computes the size of a BBP-MCCIS of two split graphs obtained from cutvertices. Therefore, 0 is returned if the given vertices u and u' not both are cutvertices. Otherwise, we consider their child B-nodes childB(u) and childB(u') in the rooted BC-trees. To this end, we again create a weighted complete bipartite graph with vertex partition childB(u) $\uplus$ childB(u'). The weight of an edge is the size of a BBP-MCCIS of the two split graphs associated with its endpoints. Note that the node sets contain block B-nodes as well as bridge B-nodes. Edges connecting different types of B-nodes obtain weight $-\infty$ (line 17) as mapping them contradicts restriction (BBP1). This assures that these edges are not contained in any maximum weight matching. The weight of an edge between two bridge B-nodes is determined by a recursive call of BbpMatchVertex, where the current base vertices are replaced by the other endpoints of the bridges. In case of two block B-nodes the BBP-MCCIS is determined in a similar manner as in the main procedure with the difference that only S-nodes are considered that contain a representative of the base vertex with an appropriate incident edge $r \in E(G)$, which is mapped to an edge $r' \in E(H)$. These two edges serve as roots of the SP-trees and for each pair the procedure BbpMcisSeries is called. The maximum value returned by any of these calls yields the edge weight. Note that the mapping of the base vertices is fixed and—in contrast to the main procedure—we do not need to consider all the possible mappings.

Analysis

We analyze Algorithm 3.8 and show that BBP-MCCIS can be solved in polynomial time for series-parallel graphs. We give improved bounds on the running time for the case that both input graphs are outerplanar.

Theorem 3.5. *The problem BBP-MCCIS in series-parallel graphs can be solved in time $\mathcal{O}(n^6)$, where $n = \max\{|G|, |H|\}$.*

Proof. The correctness of the algorithm is based on the argumentation above and the results presented in Section 3.4. The BC- as well as SP-tree data structure can be computed in linear time (Gutwenger and Mutzel, 2001). This also holds for our data structure, where we have an SP-tree for every block B-node. To prove the running time, we again transform the algorithm in a dynamic programming approach as for the proof of Theorem 3.2 using a table of size $\mathcal{O}(n^4)$ for all pairs of split graphs from 2-separators. In addition, we now have to consider split graphs that are obtained from cutvertices, for which another table of size $\mathcal{O}(n^2)$ is sufficient.

Again assume that for each split graph that is smaller than the split graph considered by the current call of one of the procedures, the BBP-MCCIS has been computed. Then all calls to procedures are answered in constant time by a lookup in the table. Consequently, the total running time required by the procedure BbpMcisSeries remains unchanged compared to McisSeries and we obtain $\mathcal{O}(n^6)$ with the same arguments used previously. The total running time of BbpMatchEdge still is $\mathcal{O}(n^5)$ due a quadratic number of possible MwbMatching calls. The procedures BbpMatchVertex also is based on MwbMatching. Here, each matching problem corresponds to a pair of rooted subtrees of the BC-trees. There can be at most $\mathcal{O}(n^2)$ such pairs and each matching problem can be solved in $\mathcal{O}(n^3)$ by the Hungarian method. Therefore, the total running time of this procedure also is $\mathcal{O}(n^5)$.

Finally, we consider the main procedure, Algorithm 3.8. Since we may assume that all calls to procedures are answered in constant time by lookups in the tables, the running time of the procedure is $\mathcal{O}(n^2)$. Consequently, the total running time of the algorithm is dominated by the procedure BbpMcisSeries and is $\mathcal{O}(n^6)$. □

If we restrict to outerplanar graphs, we can again use the fact, that each P-node in the SP-trees has degree two, cf. Lemma 3.5.

Theorem 3.6. *The problem BBP-MCCIS in outerplanar graphs can be solved in time $\mathcal{O}\left(n^5\right)$, where $n = \max\{|G|, |H|\}$.*

Proof. The proof is similar to the proof of Theorem 3.5 and uses arguments previously used for the proof of Theorem 3.3. Since all P-nodes in SP-trees of outerplanar graphs have degree two according to Lemma 3.5, the total running time of BbpMcisSeries reduces to $\mathcal{O}(n^4)$, since the sets M and M' considered in line 11 contain only one element. Moreover, there is no need to call MwbMatching in the procedure BbpMatchEdge. Consequently the running time of a single call of BbpMatchEdge becomes constant. For the procedure BbpMatchVertex the restriction to outerplanar graphs does not allow improved bounds on the running time since the number of rooted subtrees of the BC-tree does not change. Therefore the total running time is $\mathcal{O}(n^5)$. □

For the proof of Theorem 3.6 we make use of a table to store results between all pairs of split graphs. Please note that it would be possible to utilize more efficient 2-MCIS algorithms specifically designed for outerplanar graphs as described in Section 3.4.4, which rely on a smaller table. However, since the total running time is dominated by the procedure BbpMatchVertex, this would not directly allow us to obtain an improved result. Nevertheless, since this method essentially corresponds to the MCST algorithm presented in Section 3.2, further improvement of the running time is possible (see Droschinsky et al., 2015).

3.6 Graph Distance Metrics

The size of an optimal solution to a maximum common subgraph problem can be used to quantify the (dis)similarity between two graphs. A metric is a dissimilarity measure that satisfies certain conditions like the triangle inequality. These properties are not only desirable because they align with an intuitive notion of dissimilarity, but can also be used algorithmically to speed up distance-based algorithms. For example, searching a graph database for the graphs that are highly similar to a given input graph can be performed more efficiently in metric spaces (Zezula et al., 2006; Shang et al., 2010).

The k-means clustering approach by Elkan (2003), which exploits the triangle inequality satisfied by a metric, has been applied in the domain of graphs (Jain and Obermayer, 2010). A similar idea was employed by Kriege et al. (2014b) to develop an hierarchical clustering algorithm, which has been used for graphs by means of maximum common subgraph based distance metrics. Remarkably, the conditions that must be satisfied to obtain a graph distance metric from maximum common subgraphs have not yet received considerable attention. In the following, we first give elementary definitions and the relation between graph distance metrics and the graph isomorphisms problem. We then briefly review maximum common subgraph based distance metrics previously used and finally discuss their applicability for polynomial-time computable maximum common subgraph variants.

3.6.1 Basic Definitions and Complexity

A *dissimilarity function* on a set of objects $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that computes a dissimilarity value for a pair of objects from $\mathcal{X}$. Intuitively, the dissimilarity value should be low for similar objects and large for dissimilar objects. A dissimilarity function that satisfies specific properties is called metric and—in the domain of graphs—is defined as follows.

Definition 3.6 (Graph Distance Metric). *A dissimilarity function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a* graph distance metric*, if and only if it satisfies the following conditions for all $G, H, F \in \mathcal{G}$:*

$$\begin{aligned} d(G,H) &\geq 0 && \textit{non-negativity} && (3.1)\\ d(G,H)=0 &\Leftrightarrow G \simeq H && \textit{identity} && (3.2)\\ d(G,H) &= d(H,G) && \textit{symmetry} && (3.3)\\ d(G,F) &\leq d(G,H)+d(H,F) && \textit{triangle inequality} && (3.4) \end{aligned}$$

Note that a graph distance metric must be able to recognize if two graphs are isomorphic to satisfy Equation (3.2).

Proposition 3.1. *Computing any graph distance metric is* GI*-hard.*

Proof. Let d be a graph distance metric. We can decide if two graphs G and H are isomorphic by computing $d(G,H)$ and verifying Equation (3.2). □

Vice versa, one may ask if there is any graph distance metric, such that its computation is in GI. We give a positive answer to this question by providing a trivial graph distance metric. Consider the dissimilarity function

$$d_{\simeq}(G,H) = \begin{cases} 0 & \text{if } G \simeq H, \\ 1 & \text{otherwise.} \end{cases} \tag{3.5}$$

Proposition 3.2. *The dissimilarity function $d_{\simeq}$ is a graph distance metric.*

Proof. Conditions (3.1), (3.2),(3.3) are trivially fulfilled by $d_{\simeq}$. Assume condition (3.4) does not hold. Then there must be three graphs G, H, F, such that $d(G,F) > d(G,H) + d(H,F)$. This inequality does only hold when the left hand side is strictly positive, i.e., $G \not\simeq F$, and the right hand side is zero, i.e., $G \simeq H$ and $H \simeq F$. Since the isomorphism relation is transitive, this contradicts the assumption. □

No polynomial-time algorithm for graph isomorphism in general graphs is known. Weakening the identity requirement without giving up the other properties of a metric facilitates the design of polynomial-time computable dissimilarity functions for graphs. A dissimilarity function is a *pseudometric*, if it satisfies Equations (3.1), (3.3), (3.4) and

$$d(G,H) = 0 \quad \Leftarrow \quad G \simeq H. \tag{3.6}$$

A pseudometric does not necessarily distinguish non-isomorphic graphs. However, we pursue the goal to obtain graph distance metrics for specific graph classes, which as a sideline solve graph isomorphism and can be computed in polynomial-time.

3.6.2 Distance Metrics from Maximum Common Subgraphs

Several graph distance metrics have been proposed that are derived from the size of a maximum common subgraph. We refer to the size of a maximum common subgraph by $[\![\textsc{Mcs}(G,H)]\!]$ and do not distinguish between the different variants of the problem and possible realizations of a size function $[\![\cdot]\!]$ for now. The following dissimilarity functions were shown to yield a metric:

$$d_1(G,H) = 1 - \frac{[\![\textsc{Mcs}(G,H)]\!]}{\max\{[\![G]\!],[\![H]\!]\}} \tag{BS}$$

$$d_2(G,H) = 1 - \frac{[\![\textsc{Mcs}(G,H)]\!]}{[\![G]\!] + [\![H]\!] - [\![\textsc{Mcs}(G,H)]\!]} \tag{WSKR}$$

$$d_3(G,H) = [\![G]\!] + [\![H]\!] - 2[\![\textsc{Mcs}(G,H)]\!] \tag{FV}$$

Each of these graph distance metrics was proposed for a specific definition of the maximum common subgraph problem. Bunke and Shearer (1998) proposed (BS) for MCIS in labeled graphs, where $[\![G]\!] = |G|$. Wallis et al. (2001) modified the denominator of this graph distance metric to obtain (WSKR), where the denominator reminds on the cardinality of the union of sets. This change is motivated by the fact that the “graph union” can be considered a more appropriate measure of the problem size than the maximum size of the graphs. The authors state that the result holds as well for MCES. The dissimilarities (BS) and (WSKR) are normalized, i.e., they range between 0 and 1. Fernández and Valiente (2001) proposed (FV) for MCES with

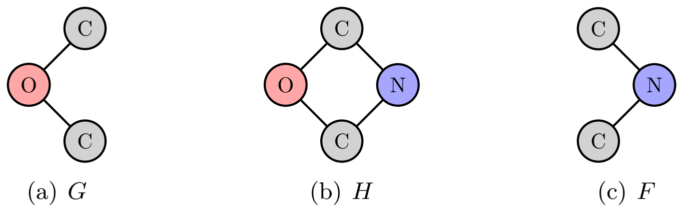


Figure 3.13: Labeled graphs for which known metrics do not fulfill the triangle inequality under MCCIS.

$[\![G]\!] = |G| + \|G\|$, and observed that the size of a minimum common supergraph is closely related to the size of a maximum common subgraph. The same graph distance metric was obtained by De Raedt and Ramon (2009) based on MCIS with $[\![G]\!] = |G|$.

None of the above metrics employs a polynomial-time computable variant of the maximum common subgraph problem. We discuss the application of these metrics to different problem variants in the following.

3.6.3 Polynomial-time Computable Graph Distance Metrics

We have developed several polynomial-time algorithms for variations of maximum common subgraph problems in restricted graph classes. An obvious question is if the result computed by these algorithms can be used to obtain a graph distance metric. The above distance metrics were proposed for maximum common subgraphs without the constraint of being connected. However, MCIS and MCES without connection constraint are NP-hard even in trees, cf. Section 3.3.3. Do the above dissimilarities yield a graph distance metric when employing MCCES or MCCIS algorithms? We give a negative answer by providing a counterexample.

Proposition 3.3. *The dissimilarities* (BS), (WSKR) *and* (FV) *are no graph distance metrics for MCCIS in labeled graphs with* $[\![G]\!] = |G|$.

Proof. Consider the example of labeled graphs shown in Figure 3.13. Employing MCCIS we have $|\text{MCS}(G,H)| = 3$, $|\text{MCS}(H,F)| = 3$ and $|\text{MCS}(G,F)| = 1$ and consequently:

d_1	G	H	F
G	0	1/4	2/3
H	1/4	0	1/4
F	2/3	1/4	0

d_2	G	H	F
G	0	1/4	4/5
H	1/4	0	1/4
F	4/5	1/4	0

d_3	G	H	F
G	0	1	4
H	1	0	1
F	4	1	0

The triangle inequality (3.4), is violated by all three dissimilarity functions, since $d_i(G,F) > d_i(G,H) + d_i(H,F)$ for all $i \in \{1,2,3\}$. □

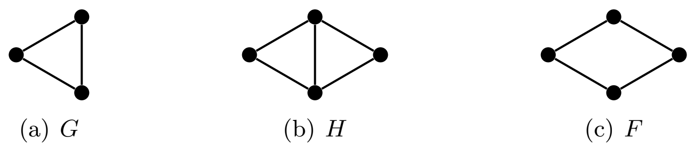

(a) G (b) H (c) F

Figure 3.14: Outerplanar graphs for which known metrics do not satisfy the triangle inequality under BBP-MCCES.

A similar counterexample can be constructed to show that (BS), (WSKR) and (FV) violate the triangle inequality for MCCES in labeled graphs with $[\![G]\!] = \|G\|$.

Schietgat et al. (2007; 2013) proposed an algorithm for BBP-MCCES and claimed that the approach yields a metric in combination with the dissimilarity (BS) and $[\![G]\!] = |G| + \|G\|$. We show that this is not the case.

Proposition 3.4. *The dissimilarities* (BS)*,* (WSKR) *and* (FV) *are no graph distance metrics for BBP-MCCES with $[\![G]\!] = |G| + \|G\|$.*

Proof. Consider the example shown in Figure 3.14. Employing BBP-MCCES with $[\![G]\!] = |G| + \|G\|$, we have $[\![\textsc{Mcs}(G,H)]\!] = 6$, $[\![\textsc{Mcs}(H,F)]\!] = 8$ and $[\![\textsc{Mcs}(G,F)]\!] = 1$[12] and consequently:

d_1	G	H	F
G	0	1/3	7/8
H	1/3	0	1/9
F	7/8	1/9	0

d_2	G	H	F
G	0	1/3	12/13
H	1/3	0	1/9
F	12/13	1/9	0

d_3	G	H	F
G	0	3	12
H	3	0	1
F	12	1	0

The triangle inequality (3.4) is violated by all three dissimilarity functions, since $d_i(G,F) > d_i(G,H) + d_i(H,F)$ for all $i \in \{1,2,3\}$. □

Based on the example shown in Figure 3.14 we can easily verify that (BS), (WSKR) and (FV) also violate the triangle inequality for BBP-MCCES with $[\![G]\!] = |G|$ and $[\![G]\!] = \|G\|$.

We have seen that the relation between the precise definition of the maximum common subgraph problem and the validity of the triangle inequality is non-trivial and neglecting this is a serious pitfall. We have not yet obtained a graph distance metric from any of the polynomial-time solvable problem variants. In the following we focus on maximum common induced subgraph problems with $[\![G]\!] = |G|$.

Definition 3.7 (Triangle Consistency). *An MCIS variant is* triangle consistent *if for all graphs $G, H, F \in \mathcal{G}$ there are maximum common subgraph isomorphisms $\psi_1 : V(G) \rightsquigarrow V(H)$ and $\psi_2 : V(H) \rightsquigarrow V(F)$ such that*

$$|\textsc{Mcs}(G,F)| \geq |\operatorname{img}(\psi_1) \cap \operatorname{dom}(\psi_2)|. \tag{3.7}$$

[12] Here, we assume that a single vertex is a BBP-MCCES, which would be coherent with the size function used. However, the same result holds for the different interpretation.

We show that triangle consistency is a sufficient condition to ensure that the dissimilarities (WSKR) and (FV) satisfy the triangle inequality. We have not investigated (BS) further, since it is similar in spirit to (WSKR) and differs only w.r.t. the normalizing denominator.

Lemma 3.7. *The triangle inequality is satisfied for the dissimilarity* (WSKR) *if the maximum common subgraph definition is triangle consistent.*

Proof. We consider arbitrary graphs $F, G, H \in \mathcal{G}$ and assume that there are maximum common subgraph isomorphisms $\psi_1 : V(G) \rightsquigarrow V(H)$ and $\psi_2 : V(H) \rightsquigarrow V(F)$. Further, we assume that the vertex sets of all three graphs are disjoint and define $V_G = (V(G) \setminus \operatorname{dom}(\psi_1)) \uplus \operatorname{img}(\psi_1)$ and $V_F = (V(F) \setminus \operatorname{img}(\psi_2)) \uplus \operatorname{dom}(\psi_2)$. Then $|\textsc{Mcs}(G,H)| = |V_G \cap V(H)|$ and $|G| + |H| - |\textsc{Mcs}(G,H)| = |V_G \cup V(H)|$. Consequently, the triangle inequality $d_2(G,F) \leq d_2(G,H) + d_2(H,F)$ holds if and only if

$$d_2(G,F) \leq 1 - \frac{|V_G \cap V(H)|}{|V_G \cup V(H)|} + 1 - \frac{|V_F \cap V(H)|}{|V_F \cup V(H)|}. \tag{3.8}$$

Since we assume triangle consistency, we have $|\textsc{Mcs}(G,F)| \geq |\operatorname{img}(\psi_1) \cap \operatorname{dom}(\psi_2)| = |V_G \cap V_F|$ and, hence,

$$d_2(G,F) = 1 - \frac{|\textsc{Mcs}(G,F)|}{|G| + |F| - |\textsc{Mcs}(G,F)|} \geq 1 - \frac{|V_G \cap V_F|}{|V_G \cup V_F|}. \tag{3.9}$$

We may plug in the right hand side of inequality (3.9) into (3.8) and observe that the triangle inequality holds for d_2 w.r.t. the graphs F, G, H as above if the triangle inequality $d_J(V_G, V_F) \leq d_J(V_G, V(H)) + d_J(V(H), V_F)$ holds, where

$$d_J(X,Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}.$$

The dissimilarity d_J is known as *Jaccard distance* or *Tanimoto distance* and indeed satisfies the triangle inequality for any finite sets as proven, e.g., by Lipkus (1999). □

We obtain the same result for (FV) by a straight-forward proof of the triangle inequality.

Lemma 3.8. *The triangle inequality is satisfied for the dissimilarity* (FV) *if the maximum common subgraph definition is triangle consistent.*

Proof. We consider arbitrary graphs $F, G, H \in \mathcal{G}$, plugging (FV) into the triangle inequality and rearranging yields:

$$\underbrace{|\textsc{Mcs}(G,H)|}_{=|\operatorname{img}(\psi_1)|} + \underbrace{|\textsc{Mcs}(H,F)|}_{=|\operatorname{dom}(\psi_2)|} \leq \underbrace{|H|}_{=|V(H)|} + \underbrace{|\textsc{Mcs}(G,F)|}_{\geq|\operatorname{img}(\psi_1)\cap\operatorname{dom}(\psi_2)|}$$

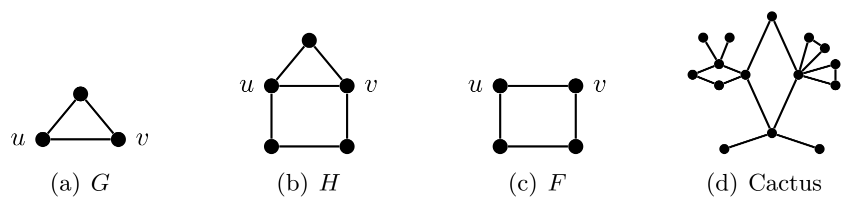


Figure 3.15: Outerplanar graphs (a)–(c) for which known metrics do not satisfy the triangle inequality under BBP-MCCIS; example of a cactus graph (d).

We assume that the maximum common subgraph definition is triangle consistent. Hence, there are maximum common subgraph isomorphisms $\psi_1 : V(G) \rightsquigarrow V(H)$ and $\psi_2 : V(H) \rightsquigarrow V(F)$ and Equation (3.7) holds. This allows to express the size of the maximum common subgraphs in the above inequality by the set cardinalities related to the image and domain of ψ_1 and ψ_2, respectively, as specified by the terms below the braces. Note that all these sets are subsets of $V(H)$ and for any two sets $A, B \subseteq C$, we know that $|A| + |B| \leq |C| + |A \cap B|$. Hence, the triangle inequality is satisfied. $\square$

We have introduced the concept of triangle consistency and seen that it is a sufficient condition for maximum common subgraph variants to obtain graph distance metrics. We consider BBP-MCCIS, for which we have presented polynomial-time algorithms in series-parallel graphs, cf. Section 3.5. Note that the problem generalizes MCST/1-MCIS and 2-MCIS. The series-parallel graphs include the outerplanar graphs, which again include the cactus graphs.

Consider the graphs shown in Figure 3.14, we can transform this counterexample used to prove Proposition 3.4 into an instance, for which BBP-MCCIS violates triangle consistency by subdividing every edge. Note that the resulting graphs do not all remain outerplanar, but are series-parallel.

Observation 3.3. *BBP-MCCIS violates triangle consistency in series-parallel graphs.*

In the above example, for any BBP-MCCIS isomorphisms $\psi_1 : V(G) \rightsquigarrow V(H)$ and $\psi_2 : V(H) \rightsquigarrow V(F)$ we have $|\operatorname{img}(\psi_1)| = 6$ and $|\operatorname{dom}(\psi_2)| = 8$. Hence, at least 5 vertices must be contained in the intersection of both since the common superset $V(H)$ consist of 9 vertices. However, $|\textsc{Mcs}(G, F)| = 1$ and, hence, triangle consistency (3.7) does not hold for BBB-MCCIS in series-parallel graphs.

Observation 3.4. *BBP-MCCIS violates triangle consistency in outerplanar graphs.*

This can be seen from the example depicted in Figures 3.15(a)–(c), where the edge uv incident to the two faces in H is contained in ψ_1 and ψ_2, but not in a BBP isomorphism between G and F, which contains at most one vertex. An edge incident to two faces does not occur in cactus graphs, cf. Figure 3.15(d). More specifically the above example shows that 2-MCIS is not triangle consistent in outerplanar graphs. We can also observe that the triangle inequality actually is not satisfied by (BS), (WSKR) and (FV) for the above examples.

However, when restricting the graph class further, we obtain the following positive result for cactus graphs.

Lemma 3.9. *BBP-MCCIS is triangle consistent in cactus graphs.*

Proof. Consider cactus graphs $F, G, H \in \mathcal{G}$ and let $\psi_1 : V(G) \rightsquigarrow V(H)$ and $\psi_2 : V(H) \rightsquigarrow V(F)$ be BBP-MCCIS isomorphisms and $H_1 = H[\text{img}(\psi_1)]$ and $H_2 = H[\text{dom}(\psi_2)]$ the corresponding common subgraphs. Let $\psi_3 : V(G) \rightsquigarrow V(F)$ be the mapping between G and F such that $\psi_3(v) = \psi_2(\psi_1(v))$ for all vertices $v \in \text{dom}(\psi_1)$ with $\psi_1(v) \in \text{dom}(\psi_2)$. It suffices to show that ψ_3 is a BBP common connected induced subgraph isomorphism (BBP-CCISI). In this case every BBP-MCCIS must have at least the size of ψ_3 and $|\textsc{Mcs}(G, F)| \geq |\psi_3| = |\text{img}(\psi_1) \cap \text{dom}(\psi_2)|$ is satisfied. This trivially holds if $|\psi_3| = 0$ or $|\psi_3| = 1$ since the empty mapping and the mapping of a single vertex is a BBP-CCISI. We consider the case where at least two vertices are mapped by ψ_3 and consider the common subgraph $H_3 = H[\text{img}(\psi_1) \cap \text{dom}(\psi_2)]$. We show that (i) H_3 is connected, (ii) ψ_3 satisfies (BBP2); and (iii) ψ_3 satisfies (BBP1).

To prove (i), we assume H_3 is disconnected. Let C_1 and C_2 be connected components of H_3. No vertex of C_1 is adjacent to any vertex of C_2 in H since H_3 is an induced subgraph of H. All the vertices of C_1 and C_2 are contained in the connected graphs H_1 and H_2. Hence, there must be a path p_1 in H_1 and p_2 in H_2 from a vertex in C_1 to a vertex in C_2. The path p_1 contains at least one vertex that is not contained in p_2 and vice versa. Let m_1 and m_2 be the first vertices on the paths p_1 and p_2 that are not contained in H_3 with $m_1 \neq m_2$. The two vertices m_1 and m_2 must be contained in the same block of H. Hence, ψ_1 and ψ_2 either violate (BBP2) or H cannot be outerplanar: Assume the edges incident to m_1 contained in p_1 are part of a block in H_1. There are two vertex-disjoint path between any two vertices contained in this block. In the supergraph H, there must be another path between two vertices of this block through m_2. Hence, there must be a pair of vertices connected by at least three vertex-disjoint paths of length at least 2. Consequently, H contains $K_{2,3}$ as a minor and cannot be outerplanar, in particular not a cactus graph. This contradicts the assumption and, hence, H_3 must be connected.

To prove (ii), we consider a bridge e of G. Since ψ_1 is BBP, it must map e to a bridge e' of H. Since ψ_2 is as well BBP it must map e' to a bridge e'' of F. Hence, ψ_3 maps a bridge of G only to a bridge of H_3. Assume H_3 contains a bridge e that is mapped to a block edge of an input graph by one of the two subgraph isomorphisms associated with ψ_3. Since ψ_1 and ψ_2 are BBP, the edge e must be mapped to a block edge in both input graphs. Since in cactus graphs BBP isomorphisms must map all vertices of a block, both ψ_1 and ψ_2 must map the whole block and, hence, the edge e must be a block edge in H_3, contradicting the assumption.

We prove (iii) by assuming that ψ_3 violates (BBP1). Since H_3 is connected, it must contain two blocks B_1, B_2 sharing a cutvertex c and $V(B_1) \cup V(B_2)$ are contained in the same block of H and consequently of H_1 and H_2. There must be two vertex-disjoint paths connecting B_1 and B_2 that do not contain c. Similar arguments as used above to prove (i) show that H cannot be outerplanar and consequently not a cactus graph, contradicting the assumption. □

Note that the above result holds as well for BBP-MCCIS applied to labeled graphs. We conclude this section with the following theorem.

Theorem 3.7. *The dissimilarities* (WSKR) *and* (FV) *are polynomial-time computable graph distance metrics for BBP-MCCIS in connected cactus graphs.*

Proof. The dissimilarity functions both clearly are non-negative and symmetric. Two connected cactus graphs of order n are isomorphic if and only if their BBB-MCCIS is of order n. Hence, the identity requirement of a metric is fulfilled. According to Lemmas 3.7, 3.8 and 3.9 the triangle inequality holds and, hence, the dissimilarities (WSKR) and (FV) are graph distance metrics under the above assumptions. Polynomial running time for computation follows from Theorem 3.6. □

3.7 Summary and Future Work

Whenever the task is to elucidate structural similarities between graphs, common subgraph problems naturally occur. Consequently, these problems are practically relevant in various application domains, where real world objects are modeled as graphs. In cheminformatics graphs are derived from molecules and have several properties that can be exploited algorithmically—nevertheless common subgraph problems remain difficult to solve, both, from a theoretical as well as a practical point of view. We have identified the problem k-MCIS, whose complexity is open although it is fundamental. We have considered the class of series-parallel graphs and studied 2-MCIS, for which we presented a polynomial-time algorithm. In addition, we discussed the special case of 2-MCIS, where input graphs are outerplanar, and gave improved algorithms and bounds on the running time. The problem 2-MCIS appears as a subproblem of BBP-MCCIS in series-parallel graphs, which is highly relevant in cheminformatics. For this problem we have designed a polynomial-time algorithm as well.

In previous work a tree decomposition graph was proposed that incorporates all possible tree decompositions of a graph. However, a key problem for using this data structure for common subgraph problems is that it does not contain all tree decompositions of *subgraphs* of the graph. To overcome this issue we proposed the new concept of potential separators. One may ask if one can extend tree decompositions to obtain a correspondence between subtrees of the tree decomposition and the subgraphs of the input graphs. This would require to take the potential separators into account and remains future work. The concept of potential separators on its own as well as such a data structure may turn out to be useful as well for other related problems, e.g., the Isomorphic Subgraph problem considered by Bachl et al. (2004).

Since we only considered 2-MCIS the following problem remains open.

Open Problem 3.1. *Can k-MCIS be solved in polynomial time for $k > 2$?*

In the affirmative case, it would be interesting to generalize the concept of block and bridge preserving common subgraphs, such that k-connected components (Holberg, 1992) or k-blocks (Carmesin et al., 2014) are preserved.

For outerplanar graphs BBP-MCCES was fundamental for solving unrestricted MCCES in outerplanar graphs of bounded degree (Akutsu and Tamura, 2013). We have obtained a BBP-MCCIS algorithm for series-parallel graphs by extending our 2-MCIS approach. Furthermore, it was shown by Kurpicz (2014) that the degree-bound is crucial for the complexity: Finding an MCCIS of two biconnected series-parallel graphs with all but one vertex of degree bounded by three is NP-hard (also see Kriege et al., 2014a). Can our BBP-MCCIS approach serve as a starting point for solving MCCIS in series-parallel graphs, where all vertices have bounded degree, as it was possible for outerplanar graphs? More generally one may ask if MCCIS in partial k-trees of bounded degree is tractable. Akutsu and Tamura (2012) have proven the negative result that MCCIS is NP-hard even for (vertex-labeled) partial 11-trees with maximum degree 6. Consequently, the following question is open.

Open Problem 3.2. *Can MCCIS be solved in polynomial time for partial k-trees of bounded degree for some $k \in \{2, \ldots, 10\}$?*

While we focused on induced subgraph problems, in practice often edge-induced subgraphs are desired, which give rise to closely related maximum common subgraph problems, cf. Section 3.3.1. MCES can be reduced to MCIS by considering the line graphs of the input graphs or subdivision graphs. Both reductions require to handle technicalities like the ΔY-exchange or mappings between original vertices and those that subdivide edges. Apart from this the graph class of the input graphs not necessarily is preserved: Figure 3.16(b) shows a simple example demonstrating that partial k-trees are not closed under taking line graphs. However, the subdivision graph of a partial k-tree obviously again is a partial k-tree making this approach promising to transfer MCIS algorithms for partial k-trees to MCES. We have presented an algorithm for 2-MCIS in outerplanar graphs with quadratic running time. The

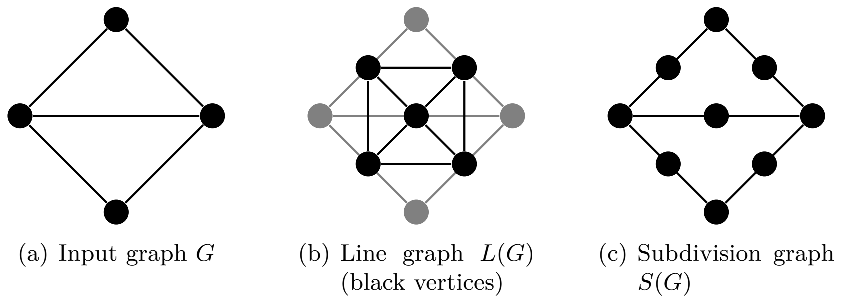

(a) Input graph G
(b) Line graph $L(G)$ (black vertices)
(c) Subdivision graph $S(G)$

Figure 3.16: An outerplanar graph (a), its line graph (b) and subdivision graph (c). While the input graph is a partial 2-tree its line graph has tree width 3. The subdivision graph has the same treewidth as the input graph, but is no longer outerplanar.

problem 2-MCIS generalizes the induced subgraph isomorphism problem in biconnected outerplanar graphs, for which the best known algorithm due to Sysło (1982) also has quadratic running time. For the subgraph isomorphism problem in biconnected outerplanar graphs the best known algorithm has cubic running time (Lingas, 1989). However, since the subdivision graph of an outerplanar graph not necessarily again is outerplanar as demonstrated by Figure 3.16(c), we can not directly apply Algorithm 3.7. It remains future work to modify our algorithms based on SP-trees for 2-MCES in outerplanar graphs—this might yield asymptotically better running times than those obtained by BBP-MCCES algorithms.

We have first studied the conditions under which maximum common subgraph variants allow to define graph distance metrics. We have shown that our algorithms can be used to obtain graph distance metrics for connected cactus graphs including the class of trees. This is a rather restricted graph class, but—as we have shown—even in outerplanar graphs polynomial-time computable maximum common subgraph variants do not yield a metric in combination with common dissimilarity functions. Graph distance metrics are important in practice, e.g., when graphs are compared in order to perform similarity searching in graph databases or clustering. Hence, as future work one could devise maximum common subgraph variants that can be used to obtain graph distance metrics while at the same time allow for polynomial-time computation in restricted graph classes.

In this chapter several novel algorithms have been proposed. While we focused on the theoretical side, the problems considered are also of high practical relevance. It is promising future work to implement these algorithms and further engineer them in order to obtain high efficiency on real-world instances. Graphs derived from molecules, for example, are typically annotated by vertex and edge labels. It is in general possible to adapt our algorithms to consider labels and we presented an adapted version of Algorithm 3.7 in (Droschinsky et al., 2015). However, for the more complex algorithms adequate elaborated modifications are likely to drastically improve the running time in practice. Developing efficient implementations taking labels into account as well as their experimental comparison on real-world graphs remains future work.

Here, we essentially only considered the task to compute the size of an optimal solution. While it is straight-forward to modify our algorithms to obtain a single optimal solution, typically there are multiple solutions of the same size. The development of enumeration algorithms for common subgraph problems, which output *all* these solutions, is a promising direction of future research. Algorithm 3.7 can easily be modified to output all maximal or maximum isomorphisms between biconnected common induced subgraphs of biconnected outerplanar graphs in polynomial time. However, in general the number of solutions is not polynomially bounded in the input size. In this case, it is of interest to develop special enumeration algorithms with a running time bounded in the size of the input and the output. The first enumeration algorithms for maximum common subgraph problems with provable running time were recently proposed for maximum common subtree isomorphisms (Droschinsky et al., 2014) and BBP-MCCIS isomorphisms in outerplanar graphs (Droschinsky et al., 2015). The algorithms are based on the classical approach described in Section 3.2 and the combination with Algorithm 3.7, respectively, and were shown to have *polynomial-delay*. This means that the 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 in the input size (Johnson et al., 1988). It remains future work to develop efficient enumeration algorithms for other variants of common subgraph problems.

Only recently first positive results on the parameterized complexity of MCIS were obtained by Abu-Khzam (2014), where the parameter k is a bound on the size of the minimum vertex covers of the input graphs. Finding other parameterization for which MCIS is fixed parameter tractable is promising future research. The initial quote of this chapter suggests that finding the common parts in fact is equivalent to finding the parts that were added. In many applications like cheminformatics highly similar input graphs are of special interest. It remains an interesting task—both, in theory and practice—to focus on the additional parts and to solve MCIS efficiently, where $p = |G| + |H| - 2|\textsc{Mcs}(G, H)|$ is considered as a parameter or (small) constant.

The totality is not, as it were,
a mere heap, but the whole is
something besides the parts.

ARISTOTLE
(384–322 BCE)

Chapter 4

Graph Kernels

In order to successfully apply machine learning algorithms in the domain of graphs, meaningful (dis)similarity measures between graphs are a prerequisite. We have seen in Section 3.6 that the size of a maximum common subgraph can be used to derive a distance metric on graphs. However, finding maximum common subgraphs efficiently is non-trivial even for highly restricted graph classes. In addition, not all polynomial-time solvable problem variants can be used to obtain a graph distance metric from the result.

In this chapter we consider graph kernels, i.e., (similarity) functions *between*[1] two graphs that satisfy specific mathematical properties. While in principle akin to similarity functions between vector embeddings of graphs, graph kernels provide unique advantages over these classical methods. We make several contributions to the theory of graph kernels, in particular w.r.t. implicit and explicit computation schemes. In this context we develop and analyze algorithms for both methods of computation, demarcate their ability to support attributed graphs and analyze their running times in theory and practice. Although the main focus of our work lies on algorithmic aspects, comparative experimental results including the application to classification tasks on various real-world data sets are presented.

In Section 4.1 the fundamentals of kernels and kernel methods are summarized. In Section 4.2 we discuss kernels for structured data. In particular, we bridge the gap between graph distance metrics and so-called *complete* graph kernels. Subsequently, we discuss two aspects of graph kernels in detail that are essential for our work: In Section 4.3 we consider possible annotations of graphs, how graph kernels can cope with general attributes and under which conditions labeled graphs are an adequate model. The section gives the required background and justification to distinguish between graph kernels for attributed graphs and those for labeled graphs. In Section 4.4 we discuss implicit and explicit methods of computation for kernels in general

[1]These must be distinguished from kernels that compute similarities between two vertices of the same graph, which are sometimes also called graph kernels (see, e.g., Kondor and Lafferty, 2002; Neumann et al., 2013).

and analyze the running time of both approaches when applied to graph data sets. Moreover, we show under which conditions R-convolution kernels can be computed by explicit mapping. Then, in Section 4.5, we discuss related work on graph kernels with a special focus on these two aspects, i.e., support for arbitrary labels and implicit or explicit computation. In the following we propose graph kernels based on walks of fixed length and develop implicit and explicit computation schemes (Section 4.6). These are subsequently compared experimentally w.r.t. to their running time in practice. Section 4.7 is dedicated to a novel graph kernel based on common subgraph isomorphisms which are scored with respect to the mapped attributes of vertices and edges. After presenting experiments for this kernel with a focus on running time, we present an extensive comparative experimental evaluation with state-of-the-art graph kernels. Finally, we summarize and discuss directions of future work.

The main contributions of this chapter are the following:

- In Section 4.4 we analyze implicit and explicit computation schemes for kernels in order to provide the algorithmic background for the computation of graph kernels. Further we observe a relation between binary kernels and (partial) equivalence relations. Based on this we provide a sufficient condition to derive explicit computation schemes for general R-convolution kernels. A preliminary version of this section has been published in (Kriege, Neumann, Kersting, and Mutzel, 2014c).

- We propose efficient algorithms for fixed length walk kernels, based on explicit mapping and product graph based implicit computation in Section 4.6. The product graph based computation fully supports arbitrary attributes and benefits from restrictive vertex and edge kernels. No algorithm for random walk kernels based on explicit computation has previously been proposed. Our experimental evaluation shows that both methods of computation behave complementary in terms of running time depending on data set size and label diversity of the graphs. Parts of this section have been published in (Kriege, Neumann, Kersting, and Mutzel, 2014c)

- In Section 4.7 we propose a graph kernel that is based on subgraphs and allows to score all possible mappings by vertex and edge kernels. This is the first kernel based on subgraphs that also supports graphs with complex attributes. We observe a contrasting trend in terms of running time for explicit and implicit computation as for walk-based kernels. Parts of this section have been published in (Kriege and Mutzel, 2012).

- In Section 4.8 we present an experimental evaluation of our novel graph kernels and compare them to several other state-of-the art kernels on benchmark graphs with simple labels and attributes. A preliminary version of this section has been published in (Kriege and Mutzel, 2012).

4.1 Kernels and Kernel Methods

In this section we briefly summarize fundamental definitions and properties of kernels and their application as an integral part of support vector machines for classification. For a comprehensive introduction to kernel methods in machine learning, please refer to (Schölkopf and Smola, 2001; Vert et al., 2004; Hofmann et al., 2008; Steinwart and Christmann, 2008).

A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is *positive semidefinite (p.s.d.)* if $\mathbf{c}^\top \mathbf{A}\mathbf{c} \geq 0$ for any $\mathbf{c} \in \mathbb{R}^n$ and *positive definite* if $\mathbf{c}^\top \mathbf{A}\mathbf{c} > 0$ for any non-zero $\mathbf{c} \in \mathbb{R}^n$.

Definition 4.1 (Kernel, Kernel Matrix). *A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* kernel *on $\mathcal{X}$ if the $n \times n$ matrix $\mathbf{K}$ with entries $k_{ij} = K(x_i, x_j)$ is symmetric and p.s.d. for any $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$, $n \in \mathbb{N}$.*[2] *The matrix $\mathbf{K}$ is referred to as* kernel matrix*.*

Assume $\mathcal{X} = \mathbb{R}^d$, the dot product between vectors always gives rise to a matrix that is p.s.d. and symmetric. Consequently,

$$K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = \sum_{x=1}^{d} [\mathbf{x}]_i [\mathbf{y}]_i, \tag{4.1}$$

is a kernel on $\mathcal{X}$. Roughly speaking, every kernel can be expressed as a dot product: A function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if there is a real Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$, such that $K(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$ for all $x, y \in \mathcal{X}$ (Steinwart and Christmann, 2008, Theorem 4.16). We call ϕ a *feature map* and $\mathcal{H}$ a *feature space* of the kernel K. A Hilbert space $\mathcal{H}$ is a vector space equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ that is complete with respect to the norm induced by the inner product. The concept of Hilbert spaces can be considered a generalization of the Euclidean space to any finite or infinite number of dimensions. An example of an infinite-dimensional Hilbert space is the space of all infinite sequences $x = (x_1, x_2, \ldots)$ of real numbers with $\sum_{i=1}^{\infty} x_i^2 < \infty$, where the inner product is realized by $\langle x, y \rangle = \sum_{i=1}^{\infty} x_i y_i$. For our purpose—as usually when learning with kernels (Schölkopf and Smola, 2001)—it is sufficient to think of $\langle \cdot, \cdot \rangle$ as a dot product.

The fact that every kernel can be considered a dot product in some feature space and, vice versa, every dot product yields a kernel, is fundamental and has been exploited on various occasions in machine learning.

4.1.1 Classification and Support Vector Machines

The most prominent example of kernel methods arguably are Support Vector Machines (SVMs), which can be employed for classification. To provide a readily understandable context for the application of graph kernels, we

[2] In the literature on kernel methods it is also common to use the term *positive definite* instead of positive semidefinite if $\mathbf{c}^\top \mathbf{K}\mathbf{c} \geq 0$ for any $\mathbf{c} \in \mathbb{R}^n$.

briefly explain classification problems, review the development of SVMs and, in particular, their relation to kernels.

Problem 4.1 (CLASS). CLASSIFICATION

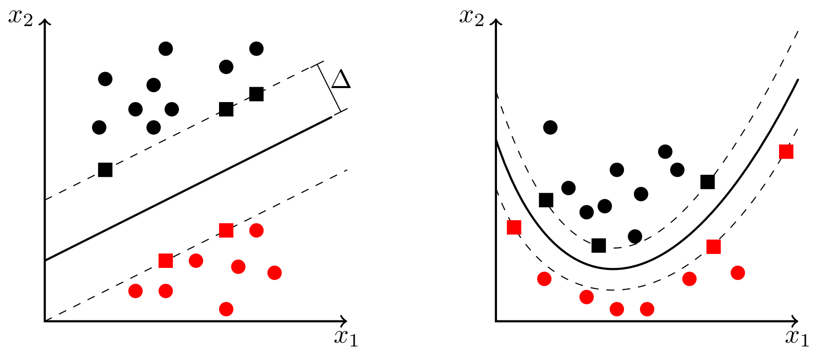
Input: *Training set* $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ *of objects* $x_i \in \mathcal{X}$ *with class labels* $y_i \in \mathcal{Y}$ *for all* $1 \leq i \leq n$.

Task: *Learn a function* $f : \mathcal{X} \to \mathcal{Y}$ *predicting class labels.*

In the following, we consider the BINARY CLASSIFICATION PROBLEM, where $\mathcal{Y} = \{-1, 1\}$. Although, we are eventually interested in the classification of graphs, i.e., $\mathcal{X} = \mathcal{G}$, let us first assume that $\mathcal{X} = \mathbb{R}^d$, $d \in \mathbb{N}$. The key idea of SVMs is to find a hyperplane in $\mathbb{R}^d$ that separates the objects in the training set with label -1 from those with label $+1$. As soon as such a hyperplane is found, the label of every object without known class label can be predicted by deciding on which side of the hyperplane it is located.

Given a training set T, let $X^+ = \{\mathbf{x} \mid (\mathbf{x}, +1) \in T\}$, $X^- = \{\mathbf{x} \mid (\mathbf{x}, -1) \in T\}$ and $X = X^+ \uplus X^-$. The sets X^+ and X^- are said to be *linearly separable* if there is a vector $\mathbf{w} \in \mathbb{R}^d$ and a real number $b \in \mathbb{R}$ such that every $\mathbf{x} \in X^+$ satisfies $\langle \mathbf{w}, \mathbf{x} \rangle + b > 0$ whereas every $\mathbf{x} \in X^-$ satisfies $\langle \mathbf{w}, \mathbf{x} \rangle + b < 0$. Let us first assume that the training data set is linearly separable like the data points in Figure 4.1(a). In this case, there is at least one separating hyperplane and, in the general case, multiple different separating hyperplanes exist. The classical approach by Vapnik and Lerner (1963) selects the hyperplane that maximizes the *margin*, i.e., the minimum distance between the hyperplane and any data point. Finding this hyperplane gives rise to a quadratic optimization problem, which is not discussed here. For details we refer the reader to a standard text book such as (Schölkopf and Smola, 2001). Let $(\mathbf{w}, b)$ describe the desired hyperplane. The normal vector $\mathbf{w}$ can be defined as as a linear combination of elements from X such that only the vectors that lie on the margin have a non-zero coefficient—these vectors are said to *support* the hyperplane and can be obtained from a solution to the optimization problem. The sign of the function $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ indicates on which side of the hyperplane the unclassified data point $\mathbf{x}$ lies and, thus, predicts a class label.

The classical approach clearly is not applicable when the training data is not linearly separable. Further, the concept of maximizing the margin is highly sensitive to outliers. Relaxing the strict constraint, so-called *soft-margin* SVMs allow data points lying within the margin: The class of C-SVMs introduces a penalty term into the objective function for violating the margin constraint. The regularization constant C allows to control the trade-off between constraint violation and the width of the margin. Schölkopf et al. (2000) modified the approach to obtain a more intuitive and interpretable parameter leading to ν-SVMs. It is essential to note that the optimization problems arising for the different types of SVMs can be formulated such that



(a) Linear classification (b) Non-linear classification

Figure 4.1: Learning a classifier for a binary classification problem on $\mathcal{X} = \mathbb{R}^2$; black and red dots represent X^+ and X^-, respectively. In (a) the two sets are linearly separable and the separating hyperplane that maximizes the margin Δ is shown. The corresponding support vectors are depicted as rectangles. In (b) the data points are not linearly separable in the input space $\mathbb{R}^2$, but in the feature space associated with the kernel used for classification.

all training data points appear only in dot products. Let S be the set of support vectors of a solution and α_s the coefficient associated with $s \in S$, then

$$f(\mathbf{x}) = b + \langle \mathbf{w}, \mathbf{x} \rangle = b + \left\langle \sum_{\mathbf{s} \in S} \alpha_s \mathbf{s}, \mathbf{x} \right\rangle = b + \sum_{\mathbf{s} \in S} \alpha_s \langle \mathbf{s}, \mathbf{x} \rangle. \tag{4.2}$$

Hence, deciding on which side of the corresponding hyperplane an unclassified data point lies, is also possible by computing dot products between data points only.

By now we assumed that the training data is linearly separable. However, we cannot expect this property for real-world data. Consider the example shown in Figure 4.1(b), the data points cannot be separated by a hyperplane. However, by mapping the data points from the input space in some feature space, where they become linearly separable, would allow us to apply SVMs in the feature space in order to solve the original problem. Assume there is such a function $\phi : \mathcal{X} \to \mathcal{H}$ and we have computed a separating hyperplane P_ϕ in $\mathcal{H}$, then $P = \{\mathbf{x} \in \mathcal{X} \mid \phi(\mathbf{x}) \in P_\phi\}$ may form, for example, a curve in $\mathcal{X}$ as depicted in Figure 4.1(b). This generalization of a hyperplane is referred to as *decision boundary*. In order to predict the class label of any object, we map it into feature space and decide on which side of the hyperplane it lies. This approach allows to obtain a non-linear classification in the input space $\mathcal{X}$. Remind the correspondence between kernels and dot products in some feature space and the fact that SVMs merely require the dot product between data points, cf. Equation (4.2). Instead of providing the training data points $X \subseteq \mathbb{R}^d$ as input for an SVM algorithm, we may provide the training data $X \subseteq \mathcal{X}$ and a kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, or simply the corresponding kernel matrix. This reduces the classification problem to the task of designing adequate kernel functions allowing us to obtain non-linear classification in the input space. A kernel function may be efficiently computable, although a corresponding feature space is of high or even infinite dimension. In this case it would be prohibitive to explicitly map elements into feature space. Kernels allow to operate in this high-dimensional feature space implicitly avoiding the computational expensive mapping. Exploiting this fact is known as *kernel trick*. Moreover, note that the input of SVM algorithms no longer is restricted to data points from $\mathbb{R}^d$, but we may use arbitrary objects as long as we provide a kernel on these objects. Thus, SVMs become a powerful tool even for learning in structured data like graphs by means of *graph kernels*, i.e., a kernel function $K : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ on graphs.

4.1.2 Standard Kernels and Closure Properties

Various kernel functions for vector data are known (Genton, 2002; Hofmann et al., 2008), which will again be of interest for kernels on graphs that are annotated by vertex and edge attributes. As we have stated earlier every kernel corresponds to the dot product in some feature space. Directly applying the dot product to vector data according to Equation (4.1) is referred to as *linear kernel*. Other kernels obtain non-linear decision boundaries in the input space. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ in the following, where $d \in \mathbb{N}$. The *polynomial kernel* is defined as

$$K_{\text{poly}}(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^p, \tag{4.3}$$

where $p \in \mathbb{N}$ and $c \geq 0$ are parameters. Note that the linear kernel is obtained as a special case of the polynomial kernel for $c = 0$ and $p = 1$.

A popular kernel is the *Gaussian radial basis function* or *(Gaussian) RBF kernel*, which is defined as

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right), \tag{4.4}$$

where σ is a parameter. Since the kernel only depends on the distance vector $\mathbf{x} - \mathbf{y}$ and not directly on $\mathbf{x}$ and $\mathbf{y}$, it is said to be *translation invariant.* This kernel is frequently and often successfully used in practice and generally considered a reasonable first choice for classification of vector data (Hsu et al., 2003). The classifier learned corresponds to the sum of Gaussian centered on the support vectors and, thus, rather complex decision boundaries can be obtained (Vert et al., 2004). The Gaussian RBF kernel matrix is positive definite (Hofmann et al., 2008), implying that the associated feature space has an infinite number of dimensions. Thus, it is prohibitive to compute the feature map explicitly, while evaluating the kernel according to Equation (4.4) allows to implicitly operate in this feature space. Nevertheless, for computational reasons, cf. Section 4.4, it was shown to be practical to compute an explicit mapping in a low-dimensional feature space, such that the dot product in this feature space approximates the Gaussian RBF kernel (Rahimi and Recht, 2008; Cotter et al., 2011). Since the RBF kernel is *globally supported*, i.e., $K_{\text{RBF}}(\mathbf{x}, \mathbf{y}) > 0$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, it gives rise to dense kernel matrices. However, *compactly supported* RBF kernels have been investigated, which yield sparse kernel matrices with a certain information loss (see Zhang and Genton, 2004; Zhu, 2012, and references therein). The arising sparsity can lead to substantial savings in storage and computation time when exploited adequately by kernel-based algorithms.

The simplest kernels one may think of is $K_c(x, y) = c$ for all $x, y \in \mathcal{X}$, where c is a constant. For $c = 0$ and $c = 1$ this kernel is referred to as *zero kernel* and *one kernel*, respectively. The *Dirac kernel* or *delta kernel* is widely-used and defined as

$$K_\delta(x, y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise}, \end{cases} \tag{4.5}$$

where $x, y \in \mathcal{X}$. The zero, one and Dirac kernel do not require that $\mathcal{X} = \mathbb{R}^d$.

Kernels are closed under certain operations allowing to combine known kernels to obtain new functions that again are valid kernels. Let K_1 and K_2 be kernels on $\mathcal{X}$, then the linear combination with non-negative coefficients $a_1, a_2 \in \mathbb{R}_{\geq 0}$

$$K(x, y) = a_1 K_1(x, y) + a_2 K_2(x, y) \tag{4.6}$$

is again a kernel. Further, the product of two kernels yields a kernel:

$$K(x, y) = K_1(x, y) \cdot K_2(x, y) \tag{4.7}$$

This allows, for example, to obtain a compactly supported kernel from a globally supported RBF kernel by multiplication with a known compactly supported kernel.

4.1.3 Kernels as Similarity Measures

We have mathematically defined what kernels are. In order to design novel kernel functions it is useful to have a more intuitive interpretation. From a statistics perspective kernels can be considered covariances (Genton, 2002). However, in the following we will think of a kernel function as a measure of similarity between objects, see (Vert et al., 2004) for a detailed justification of this viewpoint. This perspective might not readily align with the geometric interpretation of the dot product in some feature space. For example, $\langle \mathbf{x}, \mathbf{x} \rangle < \langle \mathbf{x}, 2\mathbf{x} \rangle$ for $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{x} \neq \mathbf{0}$, although according to intuition one might want to require that every object exhibits a higher similarity to itself than to any other object. One can obtain a valid kernel $\hat{K}$ with this property from any kernel K by *normalization* according to

$$\hat{K}(x, y) = \frac{K(x, y)}{\sqrt{K(x, x) K(y, y)}}. \tag{4.8}$$

Note that $\hat{K}$ may range between -1 and 1 with $\hat{K}(x,x) = 1$ for all $x \in \mathcal{X}$, i.e., all entries on the main diagonal are one for every kernel matrix derived from $\hat{K}$. Equation (4.8) is referred to as *cosine normalization* since

$$\hat{K}(x,y) = \left\langle \frac{\phi(x)}{\|\phi(x)\|}, \frac{\phi(y)}{\|\phi(y)\|} \right\rangle = \cos\theta, \tag{4.9}$$

where θ is the angle between $\phi(x)$ and $\phi(y)$ in feature space (Ah-Pine, 2010). Although other methods for normalization are known we refer to Equation (4.8) when speaking of normalization.

4.2 Kernels for Structured Data

The standard kernels discussed in Section 4.1.2 are, with few exceptions, only applicable to vector data. However, kernel methods can be readily applied to any kind of data as long as a kernel is available. Consequently, kernels for structured data, which is common in real-world applications, have been proposed, among them are graph kernels. Please refer to (Gärtner, 2009) for a more profound overview on kernels for structured data.

4.2.1 Kernels on Sets

We first consider kernels on sets, which are assumed to be finite. Let A and B be sets from a universe U, the function

$$K_\cap(A,B) = |A \cap B| \tag{4.10}$$

is a kernel on $\mathcal{P}(U)$ referred to as *intersection kernel.* Let $\phi : \mathcal{P}(U) \to \{0,1\}^{|U|}$ be the mapping of sets to their characteristic vectors, then $K_\cap(A,B) = \phi(A)^\top \phi(B)$ and, thus, a p.s.d. function. Note that this immediate explicit mapping scheme of the intersection kernel does not directly carry over to multisets, where the characteristic vector would specify the multiplicity of each element in the multiset.

Assume we have given a kernel k on U and, in order to compare two sets $A, B \subseteq U$, we want to use k to compare the individual elements of the two sets. The *cross product kernel* is defined as

$$K_\times(A,B) = \sum_{a\in A}\sum_{b\in B} k(a,b). \tag{4.11}$$

For $k = K_\delta$ this kernel coincides with the intersection kernel, i.e., $K_\times(A,B) = K_\cap(A,B)$. Hence, for this choice of k, the cross product kernel also corresponds with the dot product between the characteristic vectors.

More general kernels based on similar ideas can be obtained for infinite sets as well as for multisets. The set intersection, for example, yields a valid kernel on semirings of sets for all measures (Gärtner, 2009, Proposition 3.4). The kernels above can be applied to sets, but not yet to complex structured data.

4.2.2 Convolution Kernels

Haussler (1999) proposed R-convolution kernels, which provide a generic framework to define kernels between composite objects. Suppose $\mathbf{x} \in \mathcal{R} = \mathcal{R}_1 \times \cdots \times \mathcal{R}_d$ are the parts of $X \in \mathcal{X}$ according to some decomposition. Let $R \subseteq \mathcal{R} \times \mathcal{X}$ be a relation such that $(\mathbf{x}, X) \in R$ if and only if X can be decomposed into the parts $\mathbf{x}$. Let $R^{-1}(X) = \{\mathbf{x} \mid (\mathbf{x}, X) \in R\}$ and assume $R^{-1}(X)$ is finite for all $X \in \mathcal{X}$. The function

$$K(X,Y) = \sum_{\mathbf{x} \in R^{-1}(X)} \sum_{\mathbf{y} \in R^{-1}(Y)} \underbrace{\prod_{i=1}^{d} k_i([\mathbf{x}]_i, [\mathbf{y}]_i)}_{k(\mathbf{x},\mathbf{y})} \tag{4.12}$$

is a valid kernel provided that k_i is a valid kernel on $\mathcal{R}_i$ for all $i \in \{1, \dots, d\}$ and referred to as *R-convolution kernel*. Note that $k(\mathbf{x}, \mathbf{y})$ clearly is a valid kernel on $\mathcal{R}$ if k_i are valid kernels on $\mathcal{R}_i$, since kernels are closed under taking the product, cf. Section 4.1.2. In fact, Equation (4.12) for arbitrary d can be easily obtained from the case $d = 1$ for an appropriate choice of $\mathcal{R}_1$ and k_1 as noted by Shin and Kuboyama (2010). The class of R-convolution kernels is popular to define kernels on structured data and most graph kernels can be seen as instances of R-convolution kernels (Vishwanathan et al., 2010), where $d = 1$ is a common choice. In fact R-convolution kernels are closely related to the cross product kernel: If we assume $d = 1$ and $\mathcal{R} = \mathcal{R}_1 = U$, the R-convolution kernel essentially boils down to the decomposition of structured objects X into sets $\{x \in U \mid x \in R^{-1}(X)\}$, which are then compared by the kernel k_1 on U according to Equation (4.11). This view is sufficient for our approach to develop explicit mapping schemes for graph kernels in Section 4.4.3.

4.2.3 Distance Metrics and Complete Kernels

We have discussed graph distance metrics derived from maximum common subgraphs in Section 3.6. In this section we bridge the gap between graph kernels and graph distance metrics.

Every kernel K on $\mathcal{X}$ is associated with a feature map $\phi : \mathcal{X} \to \mathcal{H}$ into a feature space. We can compute the distance in a feature space $\mathcal{H}$ of a kernel K on $\mathcal{X}$ by employing the kernel trick—no matter if feature vectors can be computed explicitly or not—according to

$$d_K(x,y) = \|\phi(x) - \phi(y)\| = \sqrt{K(x,x) + K(y,y) - 2K(x,y)}, \tag{4.13}$$

which is referred to as *kernel metric*. Note that d_K yields a metric on $\mathcal{H}$ and in general a pseudo-metric on $\mathcal{X}$. However, it is a metric on $\mathcal{X}$ if and only if ϕ is an injection (see, e.g., Steinwart and Christmann, 2008).

The feature map $\phi : \mathcal{G} \to \mathcal{H}$ of a kernel K on graphs can be viewed as a graph invariant. Thus, an interesting property of a graph kernel K

is, whether the associated function ϕ is a complete graph invariant. If it is not, there are non-isomorphic graphs $G, H \in \mathcal{G}$ with $\phi(G) = \phi(H)$ that cannot be distinguished by K. Gärtner et al. (2003) introduced the concept of *complete* graph kernels as kernels on graphs, where the feature map ϕ is an injection. This exactly is the case if and only if ϕ is a complete graph invariant. Consequently d_K according to Equation (4.13) yields a graph distance metric if K is a complete graph kernel and a pseudo metric otherwise. Thus, with Proposition 3.1 we obtain the result that computing any complete graph kernel is GI-hard, which was as well observed by Gärtner et al. (2003).

Vice versa, it is not straightforward to derive a valid graph kernel from arbitrary graph distance metrics. Consider the Gaussian RBF kernel, cf. Equation (4.4), which relies on the Euclidean distance $\|\mathbf{x} - \mathbf{y}\|$ between the two input vectors $\mathbf{x}$ and $\mathbf{y}$. It might seem natural to replace the Euclidean distance by a graph distance pseudo-metric to obtain a graph kernel. However, in general this approach does not yield a p.s.d. kernel. Haasdonk and Bahlmann (2004) have shown that satisfying the properties of a metric is a necessary, but not sufficient condition for this approach to yield a valid kernel. Let us assume we obtained a valid kernel by plugging-in a graph distance metric into a Gaussian RBF kernel. The Gaussian RBF kernel gives rise to positive definite kernel matrices implying that the associated feature map is an injection. Thus, this approach would indeed yield a complete graph kernel. For other approaches to derive kernels from distances, see (Haasdonk and Bahlmann, 2004; Neuhaus and Bunke, 2006).

4.3 Graph Data: Labels and Attributes

To model real-world objects adequately by means of graphs it is required to annotate vertices and edges. Molecules provide an intuitive example, where graphs are annotated by categorical labels from a finite set. We have introduced labeled graphs in Definition 2.7 to model this kind of annotations. Apart from that, one may as well think of labels that are continuous, e.g., edges may be annotated with real-valued Euclidean distances. Furthermore, labels could be multi-dimensional, i.e., vertices and edge are annotated with multiple categorical or real-valued properties. Even more general, we may have to deal with arbitrary objects as vertex or edge annotation, e.g., a vertex of a graph may again be associated with another graph. We obtain a general definition of such graphs by allowing arbitrary objects from a set $\mathcal{A}$ as annotations. We refer to these as *attributed graphs* to distinguish them from labeled graphs.

Definition 4.2 (Attributed Graph). *An* attributed graph *is a graph G equipped with a function $\alpha : V(G) \uplus E(G) \to \mathcal{A}$, which assigns an* attribute *$\alpha(v)$ to every vertex $v \in V(G)$ and $\alpha(e)$ to every edge $e \in E(G)$.*

Note that for attributed graphs, we do not make any assumptions regarding the structure of $\mathcal{A}$. Labeled graphs are a special case of attributed graphs obtained when $\mathcal{A}$ is a finite set of categorical values. In this case we say that the graph has *simple labels*. Finally, we refer to attributed graphs with uniform labels, e.g., $\mathcal{A} = \{\epsilon\}$, as *unlabeled*. Hence, labeled graphs generalize unlabeled graphs and are in turn generalized by attributed graphs.

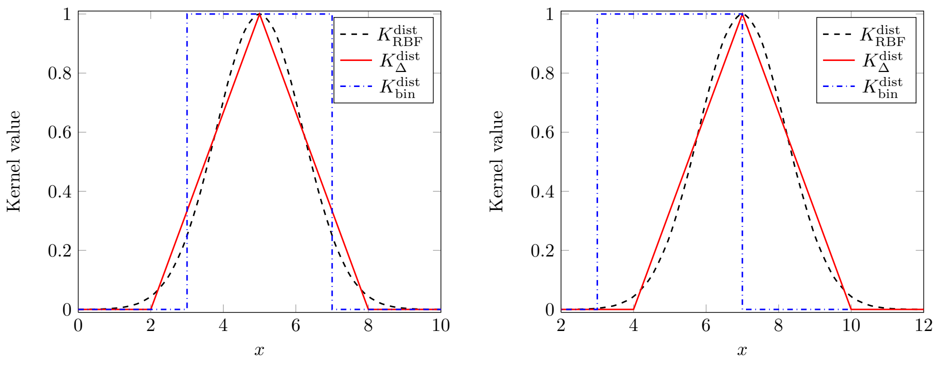
4.3.1 Taking Annotations into Account

Clearly, graph kernels should be able to make best possible use of the annotations provided. Most graph kernels decompose graphs and compare their parts, cf. Section 4.2.2, which eventually are their annotated vertices and edges. The proposed graph kernels differ in their ability to handle labels or attributes. We distinguish kernels for unlabeled, labeled and attributed graphs. Kernels for unlabeled graphs consider all vertices and edges as "compatible". Kernels for labeled graphs take into account pairs of vertices and edges only if they have equal labels, which is typically adequate for categorical labels.

For general attributed graphs the requirement of strict equality of attributes is not appropriate and prior knowledge on how to compare attributes is required. In order to support graphs with attributes one can make use of the fact that kernels can be combined according to the rules detailed in Sections 4.1.2 and 4.2. Hence, we say that a kernel supports attributed graphs when it admits two kernel functions κ_V and κ_E on $\mathcal{A}$ as input and uses them internally to compare vertex and edge attributes. Where clear from context, we occasionally neglect the index and assume that $\kappa(x)$ means $\kappa_V(x)$ if x is a vertex and $\kappa_E(x)$ if x is an edge. For convenience, we write $\kappa(x)$ instead of $\kappa(\alpha(x))$.

A kernel on attributed graphs can be applied to graphs with simple labels by setting $\kappa_V = \kappa_E = K_\delta$, i.e., the Dirac kernel according to Equation (4.5). This kernel then realizes the strict equality requirement inherent to kernels for graphs with simple labels. However, the flexibility of vertex and edge kernels also allows to design kernels on labels that penalize the matching of certain non-equal labels while other different label combinations are completely forbidden or allowed. In a molecular graph, for example, matching an oxygen atom and a nitrogen atom may be considered acceptable when the task is to predict biological activity by means of graph kernels. On the contrary, matching an oxygen atom and a carbon atom possibly should not be allowed because of their different effect w.r.t. the biological activity. When graphs have continuous, real-valued attributes, requiring exact equality is not appropriate; the possibility of specifying kernel function to obtain an adequate similarity measure between attributes overcomes this problem.

As a running example, we consider attributed graphs, where edges are annotated by real-valued distances, and discuss three possible choices for κ_E



(a) Varying x, $y = 5$ fixed (b) Varying x, $y = 7$ fixed

Figure 4.2: Kernel functions to compare distances, where x is varied and y is fixed at either 5 (a) or 7 (b). The Gaussian RBF kernel $K^{\text{dist}}_{\text{RBF}}$ with $\sigma = 1.2$ and the triangular kernel K^{dist}_{Δ} with $c = 3$ have a peak at y and decrease with the difference. The kernel $K^{\text{dist}}_{\text{bin}}(x, y)$ does not change for $y = 5$ and $y = 7$ fixed, since both values fall in the same bin.

as input for kernels supporting attributed graphs. These have been reported to be used in the context of graph kernels for the comparison of real-valued distances $x, y \in \mathbb{R}$ in real-world problems:

$$K^{\text{dist}}_{\text{RBF}}(x, y) = \exp\left(-\frac{|x-y|^2}{2\sigma^2}\right) \tag{4.14}$$

$$K^{\text{dist}}_{\Delta}(x, y) = \frac{1}{c} \cdot \max\{0, c - |x-y|\} \tag{4.15}$$

$$K^{\text{dist}}_{\text{bin}}(x, y) = \begin{cases} 1 & \text{if } \operatorname{bin}(x) = \operatorname{bin}(y), \\ 0 & \text{otherwise,} \end{cases} \tag{4.16}$$

where bin : $\mathbb{R} \to \mathbb{N}$ is a function performing some kind of *binning* as, for example,

$$\operatorname{bin}(x) = \begin{cases} 1 & \text{if } x < 3, & \text{“near”} \\ 2 & \text{if } 3 \leq x \leq 7, & \\ 3 & \text{if } x > 7. & \text{“far”} \end{cases} \tag{4.17}$$

The three functions are illustrated in Figure 4.2. Equation (4.14) is obtained from the standard Gaussian RBF kernel. The *triangular kernel* K^{dist}_{Δ} according to Equation (4.15) depends on the threshold parameter $c \in \mathbb{R}$. It ranges between zero and one, assigning one to equal distances, zero if the discrepancy exceeds the threshold c, and a value anti-proportional to the difference otherwise. This kernel was, for example, implemented in the toolkit ChemCPP (Perret and Mahé, 2006) for the pharmacophore kernel (Mahé et al., 2006, see Sections 4.5, 4.7.2) and used to compare distances between atoms in a 3D model of molecules. Borgwardt et al. (2005) used a variant of this function without the normalizing coefficient $1/c$ for the comparison of distances between secondary structure elements of proteins and referred to it as *Brownian Bridge kernel.* The idea of binning as in Equation (4.16) has been used on several occasions for the comparison of graphs with continuous labels and was reported to be successful, e.g., for continuous distances between atoms in (Mahé et al., 2006). However, it is not clear if this approach is generally suitable: Consider again our example of a real-valued edge attribute with the binning function according to Equation (4.17). Assume the edge in one graph is annotated with $x = 7$ and the other with $y = 7 + \epsilon$, for some small $\epsilon > 0$. Then these two edge labels would yield $K_{\text{bin}}^{\text{dist}}(x, y) = 0$, since only the second distance is considered "far". This issue is illustrated in Figure 4.2(b) and has been referred to as *discontinuity on bin-boundaries problem* by Fober et al. (2012). Binning also is possible for more complex attributes, which are, e.g., multi-dimensional. In order to perform binning for arbitrary attributes, a function $\text{bin} : \mathcal{A} \to \mathbb{N}$ is sufficient, which could, for example, be obtained by hashing techniques (Neumann et al., 2012b). However, the problem of discontinuity on bin-boundaries can be expected to become more serious for multi-dimensional continuous attributes.

From the above discussion it becomes apparent that the three possible solutions differ in their ability to score the discrepancy between distances adequately. The three kernel functions also have distinct characteristics that crucially affect the computation of graph kernels when used as edge kernels. The Gaussian RBF kernel is globally supported and always yields a non-zero value, while the triangular kernel is compactly supported. We will later see that this difference drastically influences the running time required for computing kernels for attributed graphs. The binning kernel also is compactly supported and is closely related to the discretization of attributes to transform attributed graphs into labeled graphs.

4.3.2 Transforming Attributes to Simple Labels

We have observed that a kernel for attributed graphs, where vertex and edge kernels are realized by the Dirac kernel, compares labels in a similar manner as kernels for graphs with simple labels. Most graph kernels yet were proposed with labeled graphs in mind. Thus, an obvious question is if we can, the other way round, apply graph kernels designed for unlabeled or simply labeled graphs to attributed graphs? Clearly, we can compare attributed graphs by applying any graph kernel to the underlying unlabeled graphs—which will not exploit the possible rich information of annotations at all. Directly applying a kernel for labeled graphs to attributed graphs in general is hardly promising, since, for example, distances that are "nearly" equal typically should also contribute to the kernel value. A better approach would be to transform these attributed graphs into graphs with meaningful simple labels. The binning kernel, cf. Equation (4.16), is based on a function, which maps attributes to

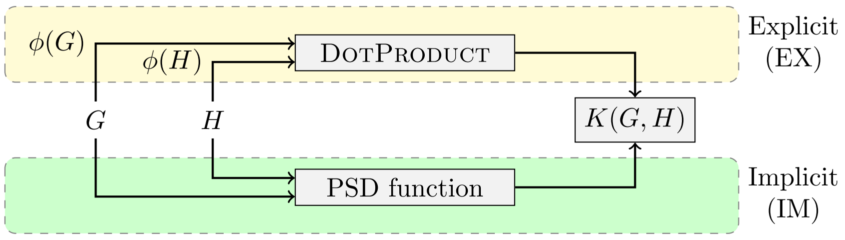


Figure 4.3: Explicit and implicit computation of a kernel K between two graphs G and H.

discrete values. A function bin : $\mathcal{A} \to \mathcal{L}$, where $\mathcal{L}$ is a finite set, can as well be realized by hashing. Then, we can use this function to transform attributed graphs into labeled graphs by simply relabeling all vertices and edges.

Assume we have a kernel for attributed graphs and employ a binning vertex and edge kernel according to Equation (4.16) for some binning function. Applying this kernel to a data set would yield exactly the same result as applying the same graph kernel with $\kappa_V = \kappa_E = K_\delta$ to the data set after transformation to simply labeled graphs based on the same binning function. In general, after transformation, we can apply every graph kernel designed for labeled graphs to the data set. Note that this is only possible, because we mimic a very restricted kernel to compare edge labels; this would not be possible for edge kernels according to Equations (4.14), (4.15). As we will see later, the flexibility of supporting arbitrary vertex and edge kernels for attributes is closely related to the employed method of computation.

4.4 Explicit and Implicit Kernel Computation

Although the analysis and results presented in this section are valid for kernels in general, we occasionally refer to graph kernels in the following and give concrete examples arising in the domain of graph data. As previously mentioned, every kernel $K(G, H)$ can be considered as a function that gives rise to a p.s.d. kernel matrix or, equivalently, as a dot product $\langle \phi(G), \phi(H) \rangle$, where ϕ is a mapping into a feature space. These two aspects are related to the algorithmic strategies employed to compute graph kernels, cf. Figure 4.3:

(i) One way is functional computation, e.g., by computing kernel values for the individual substructures obtained for some decomposition, which are then combined. In this case the feature map not necessarily is known and the feature space may be of infinite dimension. Therefore, we refer to this approach closely related to the famous kernel trick as *implicit* computation.

(ii) The other strategy is to compute the feature vector $\phi(G)$ for each graph G *explicitly* in order to obtain the kernel values from the dot product between pairs of feature vectors. These feature vectors commonly count how often certain substructures occur in a graph.

The running time of kernel methods may heavily depend on the strategy used: Consider, for example, the SVM classifier according to Equation (4.2). When computing feature maps explicitly, the normal vector of the hyperplane can be constructed explicitly as well and evaluating the classifier once requires computing a single dot product only. Using implicit computation the number of kernel computations required depends on the number of support vectors defining the hyperplane. We do not consider a specific kernel method in the following, but analyze the running time for computing a kernel matrix.

4.4.1 Computing Kernel Matrices

Applying kernel methods often involves computing a kernel matrix, which stores the kernel values for all pairs of data objects. For the moment we do not want to go into the computational details of any specific graph kernel, but consider the difference in running time of explicit and implicit computation schemes when computing a kernel matrix. Algorithm 4.1 generates the kernel matrix in a straightforward manner by directly computing the kernel functions, thus applying a mapping into feature space implicitly. Here, we assume that the procedure ComputeKernel does not internally generate the feature vectors of the two graphs passed as parameters to compute the kernel function, of course. While this would in principle be possible, it would involve computing the feature vector of every graph $\mathcal{O}(n)$ times. When explicit mapping is applied, the feature vectors can be generated initially once for each graph of the data set. Then the matrix is computed by taking the dot product between these feature vectors, cf. Algorithm 4.2.

Both approaches differ in terms of running time, which depends on the complexity of the individual procedures that must be computed in the course of the algorithms.

Proposition 4.1. *Algorithm 4.1 computes an $n \times n$ kernel matrix in time $\mathcal{O}(n^2 T_k)$, where T_k is the running time of* ComputeKernel *to compute a single kernel value.*

Algorithm 4.1: Computation by implicit mapping into feature space.

```
Input  : A set of graphs $\mathcal{D} = \{G_1, \dots, G_n\}$.
Output : Symmetric $n \times n$ kernel matrix $\mathbf{K}$ with entries $k_{ij}$.
1 for $i \leftarrow 1$ to $n$ do
2     for $j \leftarrow i$ to $n$ do
3         $k_{ij} \leftarrow$ ComputeKernel($G_i, G_j$)          ▷ $k_{ji} = k_{ij}$
```

Algorithm 4.2: Computation by explicit mapping into feature space.

```
Input  : A set of graphs $\mathcal{D} = \{G_1, \dots, G_n\}$.
Data   : Feature vectors $\Phi_i$, $i \in \{1, \dots, n\}$.
Output : Symmetric $n \times n$ kernel matrix $\mathbf{K}$ with entries $k_{ij}$.
1 for $i \leftarrow 1$ to $n$ do
2     $\Phi_i \leftarrow$ FeatureMap($G_i$)                ▷ Compute $\phi(G_i)$
3 for $i \leftarrow 1$ to $n$ do
4     for $j \leftarrow i$ to $n$ do
5         $k_{ij} \leftarrow$ DotProduct($\Phi_i, \Phi_j$)       ▷ $k_{ji} = k_{ij}$
```

Proposition 4.2. *Algorithm 4.2 computes an $n \times n$ kernel matrix in time $\mathcal{O}(nT_\phi + n^2 T_{dot})$, where T_ϕ is the running time of* FeatureMap *to compute the feature vector for a single graph and T_{dot} the running time of* DotProduct *for computing the dot product between two feature vectors.*

Clearly, an algorithm based on explicit mapping can only be competitive with implicit computation, when the time T_{dot} is smaller than T_k. In this case, however, even a time-consuming feature mapping T_ϕ pays off with increasing data set size. The running time T_{dot}, thus, is crucial for explicit computation and depends on the data structure used to store feature vectors.

4.4.2 Storing Feature Vectors

Although feature vectors may be real-valued, of course, we obtain an adequate and more intuitive notion when thinking of integer valued vectors, which count the occurrences of features. This restricted view is almost always sufficient for the explicit computation schemes we develop in the following. Then the i-th element of a feature vector of a graph G counts the occurrences of the i-th feature in G. The ordering of the features must be consistent over all feature vectors, but in principle is arbitrary. In addition, feature vectors are typically sparse and many of the theoretically possible features do not occur at all in a specific data set. When considering the label sequences of walks in a molecular graph, for example, a label sequence `H=H` would correspond to a hydrogen atom with a double bond to another hydrogen atom. This does not occur in valid chemical structures, but cannot be excluded in advance without domain knowledge. Therefore, we may adequately represent feature vectors by a binary relation

$$\phi(G) = \{(i, c) \in \mathbb{N} \times \mathbb{R}^+ \mid \text{Feature } i \text{ occurs } c > 0 \text{ times in } G\}, \tag{4.18}$$

where i is a unique identifier of a feature, which we here assume to be a natural number. An appropriate implementation of this dictionary data structure is a hash table. The function DotProduct(Φ_1, Φ_2) can then be computed in time $T_{dot} = \mathcal{O}(\min\{|\Phi_1|, |\Phi_2|\})$ in the average case and depends only on the size of the smaller feature vector. In particular, the running time does not depend on the theoretical dimension of the feature space, but on the number of different features that actually occur in the individual objects.

We introduce some notation for feature vectors as specified above and define basic operations on feature vectors used in the following. As above, we write Φ to denote a feature vector in $\mathcal{H}$ and $\phi : \mathcal{G} \to \mathcal{H}$ for the feature map. Given a feature vector Φ, where more convenient, we interpret it as a function $\Phi : \mathbb{N} \to \mathbb{R}^+$ according to

$$\Phi(i) = \begin{cases} c & \text{if } (i, c) \in \Phi, \\ 0 & \text{otherwise.} \end{cases} \tag{4.19}$$

For two feature vectors Φ_1 and Φ_2 we define addition by

$$\Phi_1 + \Phi_2 = \left\{ (i, c) \in \mathbb{N} \times \mathbb{R}^+ \;\middle|\; c = \Phi_1(i) + \Phi_2(i) > 0 \right\} \tag{4.20}$$

and the multiplication of a feature vector Φ with a scalar $s \in \mathbb{R}$ as

$$s \cdot \Phi = \left\{ (i, c) \in \mathbb{N} \times \mathbb{R}^+ \;\middle|\; c = s \cdot \Phi(i) > 0 \right\}. \tag{4.21}$$

4.4.3 Explicit Mapping of R-convolution Kernels

We have introduced general R-convolution kernels in Section 4.2.2 as a generic framework to define kernels between composite objects. Here, we establish a sufficient condition that allows to construct feature maps of these kernels. This idea will later be applied in order to derive algorithms for the explicit computation for several concrete graph kernels, which typically can be seen as instances of R-convolution kernels (Vishwanathan et al., 2010). Consider Equation (4.12) and assume that each term of the summation is determined by a kernel k on $\mathcal{R}$ with $\mathrm{img}(k) = \{0, 1\}$. We show that under this assumption we can systematically construct a mapping into a feature space.

We say a kernel k on $\mathcal{X}$ is *binary* if $k(x, y)$ is either 0 or 1 for all $x, y \in \mathcal{X}$. Given a binary kernel, we refer to

$$\sim_k = \{(x, y) \in \mathcal{X} \times \mathcal{X} \mid k(x, y) = 1\} \tag{4.22}$$

as the relation on $\mathcal{X}$ *induced by* k. Next we will establish several properties of this relation that will turn out to be useful for the construction of a feature map.

Lemma 4.1. *Let k be a binary kernel on $\mathcal{X}$, then $x \sim_k y \implies x \sim_k x$ holds for all $x, y \in \mathcal{X}$.*

Proof. Assume there are $x, y \in \mathcal{X}$ such that $x \not\sim_k x$ and $x \sim_k y$. By the definition of $\sim_k$ we obtain $k(x, x) = 0$ and $k(x, y) = 1$. The symmetric kernel matrix obtained by k for $X = \{x, y\}$ thus is either $\left(\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right)$ or $\left(\begin{smallmatrix}0&1\\1&1\end{smallmatrix}\right)$, where we assume that the first row and column is associated with x. Both matrices are not p.s.d. and, thus, k is not a kernel contradicting the assumption. □

Lemma 4.2. *Let k be a binary kernel on $\mathcal{X}$, then $\sim_k$ is a partial equivalence relation meaning that the relation $\sim_k$ is*
(i) symmetric, i.e., $\forall x, y \in \mathcal{X} : x \sim_k y \Longrightarrow y \sim_k x$ and
(ii) transitive, i.e., $\forall x, y, z \in \mathcal{X} : x \sim_k y \wedge y \sim_k z \Longrightarrow x \sim_k z$.

Proof. Property (i) follows from the fact that k must be symmetric according to Definition 4.1. Assume property (ii) does not hold. Then there are $x, y, z \in \mathcal{X}$ with $x \sim_k y \wedge y \sim_k z$ and $x \not\sim_k z$. Since $x \neq z$ must hold according to Lemma 4.1 we can conclude that $X = \{x, y, z\}$ are pairwise distinct. We consider a kernel matrix $\mathbf{K}$ obtained by k for X and assume that the first, second and third row as well as column is associated with x, y and z, respectively. There must be entries $k_{12} = k_{21} = k_{23} = k_{32} = 1$ and $k_{13} = k_{31} = 0$. According to Lemma 4.1 the entries of the main diagonal $k_{11} = k_{22} = k_{33} = 1$ follow. Consider the coefficient vector $\mathbf{c}$ with $c_1 = c_3 = 1$ and $c_2 = -1$, we obtain

$$\mathbf{c}^\top \mathbf{K} \mathbf{c} = \begin{pmatrix}1 & -1 & 1\end{pmatrix} \begin{pmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 1\end{pmatrix} \begin{pmatrix}1\\ -1\\ 1\end{pmatrix} = -1 < 0.$$

Hence, $\mathbf{K}$ is not p.s.d. and k is not a kernel contradicting the assumption. □

Corollary 4.1. *Let $\mathcal{X}_{\text{ref}} = \{x \in \mathcal{X} \mid x \sim_k x\}$, then $\sim_k$ is an equivalence relation on $\mathcal{X}_{\text{ref}}$.*

We now again consider Equation (4.12), where $R^{-1}(X) \subseteq \mathcal{R}$ for all $X \in \mathcal{X}$ and assume k is a binary kernel on $\mathcal{R}$. Let $C_k : \mathcal{R}_{\text{ref}} \to \mathbb{N}$ be a function that assigns each element $x \in \mathcal{R}_{\text{ref}}$ to (an identifier of) its equivalence class $[x]_{\sim_k}$ under the relation $\sim_k$. We define

$$F_i(X) = \left\{ x \in R^{-1}(X) \cap \mathcal{R}_{\text{ref}} \;\middle|\; C_k(x) = i \right\} \tag{4.23}$$

for all equivalence classes i and the feature map by

$$\phi(X) = \left\{ (i, c) \in \mathbb{N} \times \mathbb{N}^+ \;\middle|\; c = |F_i(X)| > 0 \right\}. \tag{4.24}$$

Theorem 4.1. *Let K be an R-convolution kernel according to Equation* (4.12) *with k a binary kernel on $\mathcal{R}$. Let ϕ be defined as above, then*

$$K(X, Y) = \phi(X)^\top \phi(Y). \tag{4.25}$$

Proof. Note that an element $x \in \mathcal{R} \setminus \mathcal{R}_{\text{ref}}$ does not contribute to the sum in Equation (4.12) according to Lemma 4.1. Thus it suffices to consider the elements in $\mathcal{R}_{\text{ref}}$, where $\sim_k$ yields an equivalence relation according to

Corollary 4.1. Consider an equivalence class i under $\sim_k$. Clearly, only two elements from the same equivalence class contribute with the value one to the sum in Equation (4.12). Assume there are a elements of $R^{-1}(X)$ in the equivalence class i and b elements of $R^{-1}(Y)$. Then pairs of elements from these classes contribute a total of ab to the sum in Equation (4.12). Note that (i, a) and (i, b) are contained in the feature vectors $\phi(X)$ and $\phi(Y)$, respectively. Therefore, the features associated with the class i by the function C_k exactly contributes ab to the dot product. The result follows, since the observation holds for all equivalence classes. $\square$

Note that the sufficient condition of Theorem 4.1 not necessarily requires that k is the Dirac kernel. One may, for example, also think of kernel functions for continuous attributes that perform some kind of binning according to Equation (4.16). In the case that k is not a binary kernel, but yields either 0 or t, where $t \in \mathbb{R}^+$, we can multiply feature vectors by $\sqrt{t}$ to obtain a mapping in a feature space. More general, assume that $k(x, y) = \lambda(C_{k'}(x)) \cdot k'(x, y)$, where k' is a binary kernel and λ a weight function, which associates a non-negative weight to each equivalence class of the relation induced by k'. Then we can modify the feature map according to

$$\phi(X) = \left\{ \left(i, c \cdot \sqrt{\lambda(i)} \right) \in \mathbb{N} \times \mathbb{N}^+ \;\middle|\; c = |F_i(X)| > 0 \right\}, \tag{4.26}$$

such that Theorem 4.1 and Equation (4.25) hold for this feature map.

4.5 Related Work on Graph Kernels

In recent years various graph kernels have been proposed and were applied to different domains. We discuss graph kernels in detail here that are related to our work and briefly summarize other graph kernels.

4.5.1 Random Walk Graph Kernels

Gärtner et al. (2003) and Kashima et al. (2003) simultaneously proposed graph kernels based on random walks, which count the number of walks two graphs have in common. These kernels were proposed for digraphs and applied to real-world graphs, which are mostly undirected, by inserting an edge vu for every edge uv.

The description of the random walk kernel by Kashima et al. (2003) is motivated by a probabilistic view on kernels and based on the idea of so-called *marginalized* kernels. It is stated that the features considered by the kernel are the label sequences produced by random walks. Since the length of the walks is not bounded, the feature space must be of infinite dimension. A method of computation is proposed based on a recursive reformulation of the kernel, which at the end boils down to finding the stationary state of a discrete-time linear system. Since this kernel was later generalized by Vishwanathan et al. (2006; 2010) we do not go into the mathematical details of the original publication. The approach fully supports attributed graphs, since vertex and edge labels encountered on walks are compared by user-specified kernels. However, it is unclear how this aligns with the specified feature space, where each dimension should be the count of a label sequence, cf. Section 4.6.2 for a more detailed discussion of this. Subsequently Mahé et al. (2004) extended this formulation of random walk kernels with a focus on application in cheminformatics (Mahé et al., 2005) to improve the scalability and relevance as similarity measure: An unfavorable characteristic of random walks is that they may visit the same vertex several times. Walks even allow to traverse an edge from u to v and instantly return to u via the same edge, a problem referred to as *tottering*. These repeated consecutive vertices do not provide useful information and even may harm the validity as similarity measure. Hence, the marginalized graph kernel was extended to avoid tottering by replacing the underlying 1st-order Markov random walk model by a 2nd-order Markov random walk model (Mahé et al., 2004; 2005). A 2nd-order Markov random walk on a graph $G = (V, E)$ can be carried out by a 1st-order random walk on a transformed graph with $|V| + |E|$ vertices. Thus, the complexity of the kernel computation increases. This technique to prevent tottering only eliminates walks $(v_1, \dots, v_n)$ with $v_i = v_{i+2}$ for some i, but it does not require the considered walks to be paths, i.e., repeated vertices still occur. Especially for the classification of chemical compounds the kernel has been combined with an enrichment of vertex labels. For each vertex, information on its environment is added to its label by employing the Morgan algorithm (Mahé et al., 2004; 2005). This leads to a reduced number of compatible vertices and thus reduces the running time of the kernel computation in practice.

Gärtner et al. (2003) also explicitly define the feature space of their random walk kernel as label sequences derived from walks, but propose a different method of computation based on the direct product graph of two input graphs with simple labels.

Definition 4.3 (Direct Product Graph). *For two labeled digraphs*[3] $G = (V, E)$ *and* $H = (V', E')$ *the* direct product graph *is denoted by* $G \times H = (\mathcal{V}, \mathcal{E})$ *and defined as*

$$\begin{aligned}\mathcal{V} &= \{(v, v') \in V \times V' \mid \tau(v) = \tau(v')\} \\ \mathcal{E} &= \{(u, u')(v, v') \in \mathcal{V} \times \mathcal{V} \mid uv \in E \wedge u'v' \in E' \wedge \tau(uv) = \tau(u'v')\}\,.\end{aligned}$$

A vertex (edge) in $G \times H$ *has the same label as the corresponding vertices (edges) in* G *and* H*.*

[3]The definition is analogously valid for graphs: If both, G and H are graphs, the direct product graph can again be interpreted as graph, cf. Section 2.2.1.

There is a one-to-one correspondence between walks in $G \times H$ and walks in the graphs G and H with the same label sequence. The *direct product kernel* is then defined as

$$K_{\mathrm{rw}}(G, H) = \sum_{i,j=1}^{|\mathcal{V}|} \left[\sum_{l=0}^{\infty} \lambda_l \mathbf{A}_{\times}^{l} \right]_{ij}, \tag{4.27}$$

where $\mathbf{A}_{\times}$ is the adjacency matrix of $G \times H$ and $\lambda = (\lambda_0, \lambda_1, \dots)$ a sequence of weights such that the sum converges. It was shown that this is the case for $\lambda_i = \gamma^i$, $i \in \mathbb{N}$, and $\gamma < \frac{1}{a}$, where $a \geq \min\{\Delta^+(G \times H), \Delta^-(G \times H)\}$. For this choice of weights the closed form expression

$$K_{\mathrm{rw}}(G, H) = \sum_{i,j=1}^{|\mathcal{V}|} \left[(\mathbf{I} - \gamma \mathbf{A}_{\times})^{-1} \right]_{ij} \tag{4.28}$$

with $\mathbf{I}$ the identity matrix exists, which can be computed by matrix inversion. Since the expression reminds of the geometric series transferred to matrices, Equation (4.28) is referred to as *geometric random walk kernel*. The running time to compute the geometric random walk kernel between two graphs is dominated by the inversion of the adjacency matrix associated with the direct product graph. For two graphs of order n we obtain a $n^2 \times n^2$ matrix in the worst-case, which results in a running time of $\mathcal{O}(n^{2\omega})$, where $\omega < 2.376$, by employing the matrix multiplication algorithm by Coppersmith and Winograd (1987) for matrix inversion (see Cormen et al., 2001, Section 28.4). However, this approach is not efficient in practice and in (Gärtner et al., 2003; Vishwanathan et al., 2010) the running time is given as roughly $\mathcal{O}(n^6)$. The approach was extended to handle transition graphs, where edges are annotated by transition probabilities, which gives rise to a direct product graph with edge weights. As already observed by Mahé et al. (2004) the approach is closely related to the marginalized kernel, which can as well be computed by means of the direct product graph, where vertices and edges are annotated with certain weights. For random walk kernels the definition of a feature map was provided for theoretical purpose or to show that the function indeed is a valid kernel. Explicit kernel computation would be prohibitive because of the infinite dimension of the feature space.

Vishwanathan et al. (2010) propose a generalizing framework for random walk based graph kernels and argue that the approach by Kashima et al. (2003) and Gärtner et al. (2003) can be considered special cases of this kernel. The publication does not address vertex labels and makes extensive use of the Kronecker product between matrices denoted by $\otimes$ and lifts it to the feature space associated with an (edge) kernel. Given an edge kernel κ_E on the attributes $\mathcal{A}$, let $\phi : \mathcal{A} \to \mathcal{H}$ be a feature map. For an attributed graph G the feature matrix $\Phi(G)$ then is defined as $[\Phi(G)]_{ij} = \phi(\alpha(v_i v_j))$ if $v_i v_j \in E(G)$ and $\mathbf{0}_{\mathcal{H}}$ otherwise. Then $W_{\times} = \Phi(G) \otimes \Phi(H)$ yields a weight matrix of the direct product graph $G \times H$.[4] The proposed kernel is defined as

$$K_{\text{rw}}(G, H) = \sum_{l=0}^{\infty} \mu_l \mathbf{q}_\times^\top \mathbf{W}_\times^l \mathbf{p}_\times, \tag{4.29}$$

where $\mathbf{p}_\times$ and $\mathbf{q}_\times$ are initial and stopping probability distributions and μ_l coefficients such that the sum converges. Several methods of computation are proposed, which yield different running times depending on a parameter specific to that approach. The parameter k either denotes the number of fixed-point iterations, power iterations or the effective rank of $\mathbf{W}_\times$. The running times to compare graphs of order n also depend on the edge labels of the input graphs and the desired edge kernel: For unlabeled graphs the running time $\mathcal{O}(n^3)$ is achieved and $\mathcal{O}(dkn^3)$ for labeled graphs, where $d = |\mathcal{L}|$ is the size of the label alphabet. The same running time is obtained for edge kernels with a d-dimensional feature space, while $\mathcal{O}(kn^4)$ time is required in the infinite case. For sparse graphs $\mathcal{O}(kn^2)$ is obtained in all cases, where a graph G is said to be sparse if $\|G\| = \mathcal{O}(|G|)$. Further improvements of the running time were subsequently obtained by non-exact algorithms based on low rank approximations of $\mathbf{W}_\times$ (Kang et al., 2012).

Random walk kernels have been applied to protein-protein interaction (PPI) networks by Borgwardt et al. (2007). In order to take missing edges into account, which is crucial for PPI networks, the kernel

$$K_{\text{comp}}(G, H) = K_{\text{rw}}(G, H) + K_{\text{rw}}(\overline{G}, \overline{H}), \tag{4.30}$$

was proposed, which is the sum of a random walk kernel K_{rw} applied to the original graphs as well as to their complement graphs. Borgwardt et al. (2005) also proposed random walk kernels to predict protein function. To this end, proteins are modeled by attributed graphs, where vertices represent secondary structure elements of the proteins and edges their geometric relation and sequence order. Both, vertices and edges, are annotated by various categorical and real-valued properties. A specific kernel for the comparison of walks in these graphs is designed, which takes these attributes into account and leads to a weight matrix similar to $\mathbf{W}_\times$. However, only walks up to a predetermined length are taken into account by the kernel used for protein function prediction. This might suggest that it is not necessary or even not beneficial to consider the infinite number of possible walks to obtain a satisfying prediction accuracy. Harchaoui and Bach (2007) applied kernels based on walks of a fixed length to image classification and developed a dynamic programming approach for their computation. Remarkably, the flexibility of selecting a certain walk length ℓ here is considered an advantage compared to the approaches that are able to consider walks of unbounded length, while ℓ earlier was referred to as “unwanted parameter” by Kashima et al. (2003).

[4]Here vertex labels are ignored, i.e., $V(G \times H) = V(G) \times V(H)$.

We refer to these kernels that consider only walks of (at most) a specified length as *fixed length walk kernels.* There is yet no empirical evidence that fixed length walk kernels are preferable to those allowing infinite walk lengths or vice versa.

4.5.2 Subgraph and Graphlet Kernels

A drawback of random walk kernels is that walks are structurally simple, for example, it is not clear to what extent the presence of a certain substructure like a functional group of a molecule is unambiguously encoded by walks. An obvious choice is to define a graph kernel based on subgraphs.

Definition 4.4 (Subgraph Kernel). *Given two graphs $G, H \in \mathcal{G}$ and a weight function $\lambda : \mathcal{G} \to \mathbb{R}_{\geq 0}$. The* subgraph kernel *is defined as*

$$K_{\subseteq}(G, H) = \sum_{G' \subseteq G} \sum_{H' \subseteq H} \lambda(G') \cdot k_{\simeq}(G', H'), \tag{4.31}$$

where $k_{\simeq} : \mathcal{G} \times \mathcal{G} \to \{0, 1\}$ is the isomorphism kernel, i.e., $k_{\simeq}(G', H') = 1$ if and only if G' and H' are isomorphic.

Instead of decomposing graphs according the general subgraph relation $G' \subseteq G$, one may as well restrict to induced subgraphs $G' \sqsubseteq G$; we refer to the corresponding kernel as *induced subgraph kernel* denoted by $K_{\sqsubseteq}$. A similar kernel based on counting common subgraphs of unbounded size was defined by Gärtner et al. (2003) and its computation was shown to be NP-hard. Thus, another direction in the development of graph kernels focuses on small subgraphs of a fixed size. The *graphlet kernel* proposed by Shervashidze et al. (2009) for unlabeled graphs explicitly constructs feature vectors. Each entry counts the occurrences of induced subgraphs of size $k \in \{3, 4, 5\}$. By sampling techniques and subgraph enumeration algorithms tailored to graphs of bounded degree, the approach becomes applicable to large graphs. Further sampling techniques for graphlets were proposed by Shi et al. (2009) and combined with hashing to obtain a compact representation. These approaches are primarily designed for unlabeled graphs. Taking simple labels into account also was shown to be tractable for molecular graphs (see, e.g., Wale et al., 2008). However, attributed graphs are not supported by these approaches.

4.5.3 Path, Tree Pattern and Further Kernels

As seen above it is a promising strategy to modify the input graphs before computing a kernel between them, e.g., by refining vertex labels (Mahé et al., 2004) or by taking the complement graph, cf. Equation (4.30). This again is the idea of the *shortest-path kernel* (Borgwardt and Kriegel, 2005), which compares all shortest paths in the two graphs according to their lengths and vertex attributes. The shortest-path kernel is defined as

$$K_{\mathrm{sp}}(G,H) = \sum_{u,v \in V(G)} \sum_{w,z \in V(H)} \kappa_V(u,w) \cdot \kappa_d(d_{uv}, d_{wz}) \cdot \kappa_V(v,z), \qquad (4.32)$$

where d_{uv} denotes the length of a shortest path from u to v, the kernel κ_d compares path lengths and κ_V the vertex attributes of the associated start and end vertices of the paths. Computation is essentially performed in two steps: For each graph G of the data set the complete graph G' with vertex set $V(G)$ is generated, where an edge uv is annotated with the length of a shortest path from u to v. The shortest-path kernel then is equivalent to the walk kernel with fixed length $\ell = 1$ between these transformed graphs, where the kernel essentially compares all pairs of edges. The kernel κ_d used to compare path lengths may, for example, be realized by the Brownian Bridge kernel or the Dirac kernel, cf. Section 4.3.1. A drawback of the shortest-path kernel is that the vertices lying on shortest paths cannot be taken into account. Recently the *GraphHopper kernel* was proposed, which allows to compare all the vertices encountered while hopping along shortest paths by a given vertex kernel (Feragen et al., 2013). In order to compute the kernel, for every vertex v of the graphs under comparison a matrix $\mathbf{M}_v$ is generated such that the entry $[\mathbf{M}_v]_{ij}$ is the number of times the vertex v appears as the i-th vertex on a shortest path containing j vertices. Using these matrices the GraphHopper kernel is computed according to

$$K_{\mathrm{gh}}(G,H) = \sum_{u \in V(G)} \sum_{w \in V(H)} \langle \mathbf{M}_v, \mathbf{M}_w \rangle \kappa_V(v,w), \qquad (4.33)$$

where $\langle \mathbf{M}_v, \mathbf{M}_w \rangle = \sum_{ij} [\mathbf{M}_v]_{ij} [\mathbf{M}_w]_{ij}$. Note that the GraphHopper kernel adds up the vertex kernel values for vertices on shortest paths, while the shortest-path kernel is based on a product. Hence, the notion of compatible paths that contribute to the total kernel value is quite different for the two approaches: Assume there are two paths $(u, \ldots, v)$ in G and $(w, \ldots, z)$ in H with the same number of vertices and $\kappa_V(u,w) = 0$ and $\kappa_V(v,z) > 0$. Note that these two paths would in any case contribute to the GraphHopper kernel, but in no case to the shortest-path kernel, cf. Equation (4.32).

In contrast to subgraphs, tree patterns are allowed to contain repeated vertices just like random walks and were initially proposed by Ramon and Gärtner (2003). This approach is based on all common subtree patterns of a specified height contained in the graphs and was developed for graphs with simple labels. However, the computation schemes based on implicit mapping can be extended to support attributed graphs. The concept of tree patterns was refined by Mahé and Vert (2009) to obtain kernels for molecular graphs, which were then applied in toxicity and anti-cancer activity prediction tasks. Harchaoui and Bach (2007) modified the approach for image classification, were graphs derived from images typically have a fixed embedding in the plane. These kernels can be considered predecessors of the *Weisfeiler-Lehman* *kernels* proposed by Shervashidze and Borgwardt (2009); Shervashidze et al. (2011), which are based on vertex label refinement. These kernels employ the classical 1-dimensional Weisfeiler-Lehman heuristic for graph isomorphism testing and consider subtree patterns consisting of the entire neighborhood of each vertex up to given distance. For a parameter h and a graph G with simple labels τ, a sequence of refined labels $(\tau_0, \dots, \tau_h)$ is computed, where $\tau_0 = \tau$ and τ_i is obtained from τ_{i-1} by the following procedure: Sort the multiset of labels $\{\tau_{i-1}(u) \mid u \in \mathrm{N}(v)\}$ for every vertex v lexicographically to obtain a unique string of labels by their concatenation and add $\tau_{i-1}(v)$ as prefix. Assign a new label $\tau_i(v)$ to every vertex v by mapping the string to a new label. The mapping of strings to new labels is referred to as *label compression* and performed in order to avoid that the length of strings increases over several iterations, which comes at the cost of an increasing label alphabet. Given an arbitrary graph kernel used as base kernel, the Weisfeiler-Lehman kernel then is the sum of the results obtained by the base kernel applied to pairs of graphs with label τ_i after the i-th refinement step. The *Weisfeiler-Lehman subtree kernel* is obtained for a base kernel counting common vertex labels; another base kernel considered is the shortest-path kernel. Compared to the earlier tree pattern kernels this approach reduces the number of features and the required running time considerably since computation typically can be achieved by explicit mapping, depending on the used base kernel, of course. Hido and Kashima (2009) independently developed a highly efficient *neighborhood hash kernel* which is similar in spirit to the Weisfeiler-Lehman subtree kernel, but represents simple labels by bit vectors and uses logical operations and hashing to encode the direct neighborhood. Bai et al. (2014) propose a label refinement technique that is essentially equivalent to the refinement step of Weisfeiler-Lehman graph kernels and combine it with an information theoretical kernel on the vertex label distribution after different iterations of refinement. The kernel is proposed for "attributed graphs", but the authors do not discuss continuous labels and—using the clear distinction we have introduced—the kernel seems to be designed for graphs with simple labels only. Propagation kernels proposed by Neumann et al. (2012b) provide a generic framework to define kernels on graphs based on an information propagation scheme for labels and attributes. Propagation, e.g., based on random walks, is performed individually on the two input graphs and a kernel is obtained by comparing label distributions after every propagation step. In this respect, the approach is similar to the Weisfeiler-Lehman subtree kernel and also efficiently computed by explicit mapping. In case of continuous (multi-dimensional) attributes a binning kernel according to Equation (4.16) is employed, where the binning function is realized by locality sensitive hashing. The extensive experimental evaluation shows highly promising results regarding running time and prediction accuracy, even for the attributed graph data sets used for comparison. By adapting the propagation scheme, the approach becomes, for example, also applicable to partially labeled graphs (Neumann et al., 2012a). The *Neighborhood Subgraph Pairwise Distance Kernel* (NSPDK) proposed by Costa and Grave (2010) associates a string representing the neighborhood subgraph with every vertex. For computational reasons graph invariants are used to encode neighborhood subgraphs avoiding graph canonization. By this means, pairs of neighborhood graphs that are likely to be isomorphic to another pair of neighborhood graphs that exhibits exactly the same distance are taken into account. The approach is similar to the Weisfeiler-Lehman shortest-path kernel.

Menchetti et al. (2005) proposed a weighted decomposition kernel, which determines matching substructures by a restrictive kernel (a so-called *selector*) and weights each matching by a kernel defined on attribute frequencies in the context of the matching, e.g., the environment of an atom of a molecular graph. The approach was used for biological sequences and molecular graphs (Menchetti et al., 2005; Ceroni et al., 2007). Horváth et al. (2004) proposed the *cyclic pattern kernel* which decomposes (molecular) graphs into cycles and trees. Let $G' \trianglelefteq G$ denote that G' either is a cycle in G or a maximal connected subgraph containing only bridges of G. Each graph G is then represented by the set $\{C(G') \mid G' \trianglelefteq G\}$, where C is a complete graph invariant for labeled cycles and trees. Graph canonization allows for a unique set representation and the kernel is then defined as intersection kernel between these sets. A different approach to define kernels on graphs is based on vector embeddings of graphs with respect to the (dis)similarities to a fixed set of objects called *prototypes* (see Riesen and Bunke, 2009, and references therein). Given a set of prototypes $\mathfrak{P} = \{P_1, \dots, P_n\}$, e.g., as set of graphs, every graph G of the data set is represented by a vector $(d(G, P_1), \dots, d(G, P_n))^\top$, where $d(\cdot, \cdot)$ is any graph dissimilarity measure. After embedding graphs into this vector space, kernel methods become readily applicable.

4.5.4 Graph Kernels in Cheminformatics

Several graph kernels were tailored especially to molecular graphs and general graph kernels were refined for the application in cheminformatics (Horváth et al., 2004; Swamidass et al., 2005; Ceroni et al., 2007; Mahé and Vert, 2009). However, so-called *fingerprints* are a well-established classical technique in cheminformatics to represent molecules by feature vectors. Commonly features are obtained by (i) enumeration of all substructures of a certain class contained in the molecular graphs, (ii) taken from a predefined dictionary of relevant substructures or (iii) generated in a preceding data-mining phase. Fingerprints are then used to encode the number of occurrences of a feature or only its presence or absence by a single bit per feature. Often hashing is used to reduce the fingerprint length to a fixed size at the cost of information loss (see, e.g., Daylight, 2008). These techniques can then easily be combined with kernels on vector data and similarity measures common for fingerprints like the *Tanimoto coefficient* are closely related to kernels (Ralaivola et al., 2005). Approaches of the first category include, for example, techniques based on all paths contained in a graph (Daylight, 2008) or all subgraphs up to a certain size (Wale et al., 2008), similar to graphlets. Ralaivola et al. (2005) experimentally compared random walk kernels to kernels derived from path-based fingerprints and has shown that these reach similar classification performance on molecular graph data sets. *Extended connectivity fingerprints* encode the neighborhood of atoms iteratively as the Weisfeiler-Lehman kernel and can be considered a standard tool in cheminformatics for more than a decade (Rogers and Hahn, 2010). Predefined dictionaries compiled by experts with domain-specific knowledge exist, e.g., MACCS/MDL Keys for drug discovery (Durant et al., 2002). Techniques like frequent subgraph mining can be combined with feature selection to generate discriminative features for a specific task in a preceding data-mining phase (Deshpande et al., 2005).

The *pharmacophore kernel* was proposed by Mahé et al. (2006) to compare chemical compounds based on characteristic features together with their relative spatial arrangement and thus has to deal with continuous distances. Kernels for chemical compounds have been successfully employed for various tasks in cheminformatics including the prediction of mutagenicity, toxicity and anti-cancer activity (Swamidass et al., 2005). Especially for attributed molecular graphs the *optimal assignment kernel* was proposed (Fröhlich et al., 2005), which computes a vertex mapping with a maximum score, in which vertices are in turn compared by kernel functions taking the neighborhood and various attributes into account. However, the proposed function was subsequently shown not to be p.s.d. (Vert, 2008; Vishwanathan et al., 2010). Nevertheless, such a function can be used for classification, e.g., by subsequent transformation of the kernel matrix. Mohr et al. (2010) proposed to derive a graph similarity measure by computing the maximum common subgraph and employs a P-SVM (Hochreiter and Obermayer, 2006) for classification, since the derived function is not a valid kernel.

4.5.5 Summary and Motivation of our Contribution

The proposed techniques can be classified into approaches that use explicit feature mapping and those that directly compute a kernel function. If explicit representations are manageable, these approaches usually outperform other kernels with respect to running time on large data sets. This is in accordance with our analysis in Section 4.4. However, these approaches typically do not support attributed graphs, cf. Table 4.1.

The development of graph kernels is commonly and justifiably motivated as follows: Techniques like SVMs have been developed for vectorial data while real-world data often is structured and adequately modeled by graphs. To overcome this issue the key idea is to define kernels on graphs in order to plug them into kernel methods. As we have observed most state-of-the-art graph kernels recently proposed employ explicit mapping for computation.

Table 4.1: Summary on selected graph kernels regarding computation by explicit (EX) and implicit (IM) feature mapping (ϕ) and support for attributed graphs (Attr.). * — attributes not considered in publication, but method can be extended; † — vertex attributes only).

Graph Kernel	ϕ	Attr.
Random Walk (Gärtner et al., 2003; Kashima et al., 2003)	IM	✓
Tree Pattern (Ramon and Gärtner, 2003; Mahé and Vert, 2009)	IM	✓*
Shortest-Path (Borgwardt and Kriegel, 2005)	IM	✓†
Graphlet (Shervashidze et al., 2009)	EX	✗
NSPDK (Costa and Grave, 2010)	EX	✗
Weisfeiler-Lehman (Shervashidze et al., 2011)	EX	✗
Propagation (Neumann et al., 2012b)	EX	✗
Subgraph Matching (Kriege and Mutzel, 2012, see Sec. 4.7)	IM	✓
GraphHopper (Feragen et al., 2013)	IM	✓†

These approaches, thus, in fact transform graph data to vector data explicitly to apply kernel methods, which typically just is the linear kernel. As a consequence one may ask if graph kernels essentially are nothing more than techniques to transform graph data into vector data? Such techniques were actually commonly used, e.g., in cheminformatics, cf. Section 4.5.4, long before the advent of graph kernels. Moreover, techniques highly similar to those recently proposed in the context of graph kernels have long been considered state-of-the art in application domains as cheminformatics. Is the research on graph kernels actually reinventing the wheel?

The unique advantage of graph kernels over the transformation into vector data is that the kernel trick can be employed, i.e., efficient computation may be possible by operating in a high or infinite dimensional feature space implicitly only. Two possible advantages of the kernel trick for graphs are apparent:

(i) An infinite number of parts may be taken into account while the kernel remains efficiently computable. This was the case for the first graph kernels based on random walks.

(ii) For the comparison of annotations one may resort to the whole toolbox of known kernels and compose a graph kernel from vertex and edge kernels. The feature spaces of these kernels for attributed graphs may have an infinite dimension, which stems from the employed vertex and edge kernels.

Remarkably most recent research does not address any of these two aspects but focuses primarily on graphs with simple labels and uses a restricted number of parts. There is yet no empirical evidence that an infinite number of parts is beneficial for graph classification tasks. This explains the success of graph kernels that are both, efficiently computed by explicit mapping and at the same time provide excellent prediction accuracy for simply labeled graphs like the Weisfeiler-Lehman and related kernels. We also observe this experimentally by considering fixed length walk kernels which are shown to be competitive or even superior compared to their infinite length counterparts on standard benchmark sets.

By now only few data sets with attributed graphs are available for comparative experimental studies, which is most likely due to the fact that their comparison is non-trivial and kernels for attributed graphs only recently received considerable attention. However, it has been observed on several occasions in concrete applications that the prediction accuracy can be increased by annotating vertices or edges with additional attributes (see, e.g., Borgwardt et al., 2005; Fröhlich et al., 2005; Harchaoui and Bach, 2007). There are first empirical indications that kernels designed for graphs with simple labels combined with binning techniques are not suitable for attributed graphs in general (Mahé et al., 2006; Fober et al., 2012; Feragen et al., 2013) for the reasons detailed in Section 4.3.1, although for certain data sets satisfying results have been reported (Mahé et al., 2006; Neumann et al., 2013). Walk-based kernels are not competitive for graphs with simple labels, but the computational techniques proposed for random walk and tree pattern kernels directly allows or can be extended to compare vertex and edge attributes by kernel functions. Thus, for attributed graphs, these early graph kernels again are of interest. We present graph kernels based on walks of fixed length in Section 4.6 and develop an implicit computation scheme for attributed graphs and an explicit computation scheme for labeled graphs.

Approaches based on walks or tree patterns have the drawback that they are based on simple features including repeated vertices. This problem has been overcome by novel graph kernels like subgraph and graphlet kernels, cf. Section 4.5.2, which are though restricted to graphs with simple labels. Therefore, we develop subgraph matching kernels in Section 4.7, which are closely related to these kernels but fully support attributed graphs. We employ an implicit computation scheme to provide the flexibility to compare vertex and edge attributes by means of arbitrary kernel functions.

4.6 Fixed Length Walk Kernels

We propose an explicit and implicit computation scheme for a walk-based graph kernel. Walk kernels are a prominent example of graph kernels and are widely used in practice. Our product graph based implicit computation scheme fully supports arbitrary vertex and edge kernels and exploits their sparsity. Previously no algorithms based on explicit mapping for computation of walk-based kernels have been proposed. To obtain authoritative experimental results we carefully implemented and engineered algorithms for both approaches. We give the details of our algorithms in the following. Finally, we experimentally compare the running times of both computation strategies systematically with respect to the label diversity, data set sizes and the walk length. As it turns out, there exists a computational phase transition for our walk kernels from explicit to implicit computations. We identify the label diversity and walk lengths as key parameters affecting the running time, either contrary or to a strongly different extent.

4.6.1 Basic Definitions

A fixed length walk kernel counts “common” walks of length ℓ. We generalize the concept of common walks for attributed graphs. Note that for any two consecutive vertices of a walk the connecting edge is uniquely defined. For convenience, we include these edges in the sequence of adjacent vertices. In the following, a walk of length ℓ in a graph G is a sequence of vertices and edges $(v_0, e_1, v_1, \ldots, e_\ell, v_\ell)$ such that $e_i = v_{i-1}v_i \in E(G)$ for $i \in \{1, \ldots, \ell\}$. The set of walks of length ℓ in a graph G is denoted by $\mathcal{W}_\ell(G)$.

Definition 4.5 (ℓ-walk kernel). *The* ℓ-walk kernel *between two attributed graphs $G, H \in \mathcal{G}$ is defined as*

$$K_\ell^{=}(G,H) = \sum_{w \in \mathcal{W}_\ell(G)} \sum_{w' \in \mathcal{W}_\ell(H)} k_W(w, w'), \tag{4.34}$$

where k_W is a kernel between walks.

Definition 4.5 is very general and does not specify how to compare walks. An obvious choice is to decompose walks and define k_W in terms of vertex and edge kernel functions, denoted by κ_V and κ_E, respectively. We consider

$$k_W(w, w') = \prod_{i=0}^{\ell} \kappa_V(v_i, v'_i) \prod_{i=1}^{\ell} \kappa_E(e_i, e'_i), \tag{4.35}$$

where $w = (v_0, e_1, \ldots, v_\ell)$ and $w' = (v'_0, e'_1, \ldots, v'_\ell)$ are two walks.[5] Assume the graphs in a data set have simple vertex and edge labels $\tau : V \uplus E \to \mathcal{L}$. An appropriate choice then is to use the Dirac kernel for both, vertex and edge kernels, between the associated labels. In this case two walks are considered equal if and only if the labels of all corresponding vertices and edges are equal. We refer to this kernel by

$$k_W^\delta(w, w') = \prod_{i=0}^{\ell} K_\delta(\tau(v_i), \tau(v'_i)) \prod_{i=1}^{\ell} K_\delta(\tau(e_i), \tau(e'_i)), \tag{4.36}$$

[5]The same idea to compare walks was proposed by Kashima et al. (2003) as part of the marginalized kernel between labeled graphs.

where K_δ is the Dirac kernel. For graphs with continuous or multi-dimensional annotations this choice is not appropriate and κ_V and κ_E should be selected depending on the application-specific vertex and edge attributes.

A variant of the ℓ-walk kernel can be obtained by considering all walks up to length ℓ.

Definition 4.6 (Max-ℓ-walk kernel). *The* Max-ℓ-walk kernel *between two attributed graphs $G, H \in \mathcal{G}$ is defined as*

$$K_\ell^{\leq}(G,H) = \sum_{i=0}^{\ell} \lambda_i K_i^{=}(G,H), \tag{4.37}$$

where $\lambda_0, \dots, \lambda_\ell \in \mathbb{R}_{\geq 0}$ are weights.

In the following we primary focus on the ℓ-walk kernel, although our algorithms and results can be easily transferred to the Max-ℓ-walk kernel.

4.6.2 Walk and R-convolution Kernels

We show that the ℓ-walk kernel is p.s.d. if k_W is a valid kernel by seeing it as an instance of an R-convolution kernel. We use this fact to develop an algorithm for explicit mapping later based on the ideas presented in Section 4.4.3.

Theorem 4.2. *The ℓ-walk kernel is positive semidefinite if k_W is defined according to Equation* (4.35) *and κ_V and κ_E are valid kernels.*

Proof. Equation (4.34) with k_W defined according to Equation (4.35) is the R-convolution kernel directly obtained when graphs are decomposed into walks $w = (v_0, e_1, v_1, \dots, e_\ell, v_\ell) = (x_0, \dots, x_{2\ell})$ and

$$k_i = \begin{cases} \kappa_V & \text{if } i \text{ even,} \\ \kappa_E & \text{otherwise} \end{cases}$$

for $i \in \{0, \dots, 2\ell\}$. Then k_W equals $\prod_{i=0}^{2\ell} k_i(x_i, x_i')$, implying that the ℓ-walk kernel is a valid kernel if κ_V and κ_E are valid kernels. □

Since kernels are closed under taking linear combinations with non-negative coefficients, cf. Section 4.1.2, we obtain the following corollary.

Corollary 4.2. *The Max-ℓ-walk kernel is positive semidefinite.*

If the employed kernel on walks satisfies the conditions discussed in Section 4.4.3, explicit feature mapping could be obtained by assigning walks to the equivalence classes of the relation induced by the kernel. We assume graphs to have simple labels from the alphabet $\mathcal{L}$ and consider the kernel k_W^δ given by Equation (4.36). This kernel clearly satisfies the condition to derive an explicit feature map and is a natural choice. A walk w of length ℓ is then associated with a label sequence $\tau(w) = (\tau(v_0), \tau(e_1), \ldots, \tau(v_\ell)) \in \mathcal{L}^{2\ell+1}$. In this case graphs are decomposed into walks and and two walks w and w' are considered equivalent if and only if $\tau(w) = \tau(w')$; each label sequence can be considered an identifier of an equivalence class of $\sim_{k_W^\delta}$. This gives rise to the feature map $\phi_\ell^=$ defined by

$$\phi_\ell^=(G) = \left\{(i, c) \in \mathcal{L}^{2\ell+1} \times \mathbb{N}^+ \;\middle|\; c = |\{w \in \mathcal{W}_\ell(G) \mid \tau(w) = i\}| > 0\right\}. \quad (4.38)$$

According to Theorem 4.1 we have

$$K_\ell^=(G, H) = \phi_\ell^=(G)^\top \phi_\ell^=(H).$$

A feature map of the Max-ℓ-walk kernel can be obtained from the feature maps of all i-walk kernels with $i \leq \ell$ according to

$$\phi_\ell^\leq(G) = \sum_{i=0}^{\ell} \sqrt{\lambda_i} \cdot \phi_i^=(G), \quad (4.39)$$

where each value in a feature vector is multiplied by the square root of the associated weight. Note that the identifiers stored in $\phi_i^=(G)$ and $\phi_j^=(G)$ are disjoint for $i \neq j$.

4.6.3 Implicit Kernel Computation

An essential part of the implicit computation scheme is the generation of the product graph that is then used to compute the ℓ-walk kernel.

Computing Direct Product Graphs

In order to support graphs with arbitrary attributes, vertex and edge kernels κ_V and κ_E are considered as part of the input. Product graphs can be used to represent these kernel values between pairs of vertices and edges of the input graphs in a compact manner. We avoid to create vertices and edges that would represent incompatible pairs with kernel value zero. The following definition can be considered a weighted version of the direct product graph introduced by Gärtner et al. (2003) for kernel computation, cf. Definiton 4.3.[6]

Definition 4.7 (Weighted Direct Product Graph). *For two attributed graphs $G = (V, E)$, $H = (V', E')$ and given vertex and edge kernels κ_V and κ_E, the* weighted direct product graph *(WDPG) is denoted by $G \times_w H =$*

[6]Note that we consider undirected graphs while Definiton 4.3 refers to directed graphs as in (Gärtner et al., 2003).

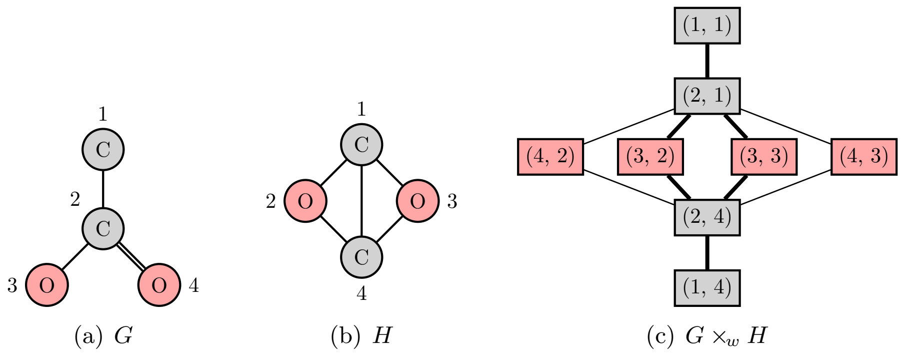


Figure 4.4: Two attributed graphs G (a) and H (b) and their weighted direct product graph $G \times_w H$ (c). We assume the vertex kernel to be the Dirac kernel and κ_E to be 1 if edge labels are equal and $\frac{1}{2}$ if one edge label is "=" and the other is "-". Thin edges in $G \times_w H$ represent edges with weight $\frac{1}{2}$, while all other edges and vertices have weight 1.

$(\mathcal{V}, \mathcal{E}, w)$ *and defined as*

$$\begin{aligned}
\mathcal{V} &= \{(v, v') \in V \times V' \mid \kappa_V(v, v') > 0\} \\
\mathcal{E} &= \Big\{(u, u')(v, v') \in [\mathcal{V}]^2 \;\Big|\; uv \in E \wedge u'v' \in E' \wedge \kappa_E(uv, u'v') > 0\Big\} \\
w(v) &= \kappa_V(u, u') && \forall v = (u, u') \in \mathcal{V} \\
w(e) &= \kappa_E(uv, u'v') && \forall e \in \mathcal{E}, \textit{where } e = (u, u')(v, v').
\end{aligned}$$

An example with two graphs and their weighted direct product graph obtained for specific vertex and edge kernels is shown in Figure 4.4. Algorithm 4.3 computes a weighted direct product graph and does not consider edges between pairs of vertices (v, v') that have been identified as incompatible, i.e., $\kappa_V(v, v') = 0$.

Since the weighted direct product graph is undirected, we must avoid that the same pair of edges is processed twice. Therefore, we suppose that there is an arbitrary total order $\prec$ on the vertices $\mathcal{V}$, such that for every pair $(u, s), (v, t) \in \mathcal{V}$ found either $(u, s) \prec (v, t)$ or $(v, t) \prec (u, s)$ holds. In line 8 we restrict the edge pairs that are compared to one of these cases.

Proposition 4.3. *Let $n = |G|$, $n' = |H|$ and $m = \|G\|$, $m' = \|H\|$. Algorithm 4.3 computes the weighted direct product graph in time $\mathcal{O}(nn'T_{\kappa_V} + mm'T_{\kappa_E})$, where T_{κ_V} and T_{κ_E} is the running time to compute vertex and edge kernels, respectively.*

Note that in case of a sparse vertex kernel, which yields zero for most of the vertex pairs of the input graph, $|G \times_w H| \ll |G| \cdot |H|$ holds. Algorithm 4.3 compares two edges by κ_E only in case of matching endpoints (cf. lines 7, 8), therefore in practice the running time to compare edges (line 7–13) might be

Algorithm 4.3: Weighted Direct Product Graph

```
Input  : Graphs $G$, $H$, vertex and edge kernels $\kappa_V$ and $\kappa_E$.
Output : Graph $G \times_w H = (\mathcal{V}, \mathcal{E}, w)$.

Procedure WDPG($G, H, \kappa_V, \kappa_E$)
    forall the $v \in V(G)$, $v' \in V(H)$ do
        $w \leftarrow \kappa_V(v, v')$
        if $w > 0$ then
            create vertex $z = (v, v')$
            $\mathcal{V} \leftarrow \mathcal{V} \uplus \{z\}$
            $w(z) = w$
    forall the $(u, s) \in \mathcal{V}$ do
        forall the $v \in \mathrm{N}(u), t \in \mathrm{N}(s)$ with $(v, t) \in \mathcal{V}$, $(u, s) \prec (v, t)$ do
            $w \leftarrow \kappa_E(uv, st)$
            if $w > 0$ then
                create edge $e = (u, s)(v, t)$
                $\mathcal{E} \leftarrow \mathcal{E} \uplus \{e\}$
                $w(e) = w$
```

considerably less than suggested by Proposition 4.3. We show this empirically in Section 4.6.6. In case of sparse graphs, i.e., $|E| = \mathcal{O}(|V|)$, and vertex and edge kernels which can be computed in time $\mathcal{O}(1)$ the running time of Algorithm 4.3 is $\mathcal{O}(n^2)$, where $n = \max\{|G|, |H|\}$.

Counting Weighted Walks

Given an undirected graph G with adjacency matrix $\mathbf{A}$, let a^ℓ_{ij} denote the element at (i, j) of the matrix $\mathbf{A}^\ell$. It is well-known that a^ℓ_{ij} is the number of walks from vertex i to j of length ℓ. The number of ℓ-walks of G consequently is

$$\sum_{i,j} a^\ell_{i,j} = \mathbf{1}^\top \mathbf{A}^\ell \mathbf{1} = \mathbf{1}^\top \mathbf{r}_\ell, \tag{4.40}$$

where $\mathbf{r}_\ell = \mathbf{A}\mathbf{r}_{\ell-1}$ with $\mathbf{r}_0 = \mathbf{1}$. The i-th element of the recursively defined vector $\mathbf{r}_\ell$ is the number of walks of length ℓ starting at vertex i.

Note that even for sparse (connected) graphs $\mathbf{A}^\ell$ quickly becomes dense with increasing walk length ℓ. The ℓ-th power of an $n \times n$ matrix $\mathbf{A}$ can be computed naively in time $\mathcal{O}(n^\omega \ell)$ and $\mathcal{O}(n^\omega \log \ell)$ using exponentiation by squaring, where ω is the exponent of matrix multiplication. The vector $\mathbf{r}_\ell$ can be computed by means of matrix-vector multiplications, where the matrix $\mathbf{A}$ remains unchanged over all iterations. Further methods to compute matrix powers are known, e.g., based on diagonalization of the adjacency matrix. However, the approach we use is closely related to vector-matrix multiplication and turned out to be efficient in practice for reasonable choices of the walk length ℓ.

In order to compute the ℓ-walk kernel we do not want to count the walks, but sum up the weights of each walk, which in turn are the product of vertex and edge weights. Let k_W be defined according to Equation (4.35), then we can formulate the ℓ-walk kernel as

$$K_\ell^=(G,H) = \sum_{w\in\mathcal{W}_\ell(G)} \sum_{w'\in\mathcal{W}_\ell(H)} k_W(w,w') = \sum_{v\in V(G\times_w H)} r_\ell(v), \tag{4.41}$$

where r_ℓ is determined recursively according to

$$\begin{aligned} r_i(u) &= \sum_{uv\in E(G\times_w H)} w(u)\cdot w(uv)\cdot r_{i-1}(v) && \forall u \in V(G\times_w H) \\ r_0(u) &= w(u) && \forall u \in V(G\times_w H). \end{aligned}$$

Note that r_i can as well be formulated as matrix-vector products. Since the direct product graph tends to be sparse we present a graph-based approach for computation, see Algorithm 4.4.

Algorithm 4.4: Implicit computation of ℓ-walk kernel

```
Input  : Graphs $G$, $H$, kernels $\kappa_V$, $\kappa_E$ and length parameter $\ell$.
Output : Value $K_\ell^=(G,H)$ of the $\ell$-walk kernel.
1 $(\mathcal{V},\mathcal{E},w) \leftarrow$ WDPG$(G,H,\kappa_V,\kappa_E)$        ▷ Compute $G \times_w H$
2 forall the $v \in \mathcal{V}$ do
3     $r_0(v) \leftarrow w(v)$                                    ▷ Initialization
4 for $i \leftarrow 1$ to $\ell$ do
5     forall the $u \in \mathcal{V}$ do
6         $r_i(u) \leftarrow 0$
7         forall the $v \in \mathrm{N}(u)$ do                     ▷ Neighbors of $u$ in $G \times_w H$
8             $r_i(u) \leftarrow r_i(u) + w(u)\cdot w(uv)\cdot r_{i-1}(v)$
9 return $\sum_{v\in\mathcal{V}} r_\ell(v)$
```

Theorem 4.3. *Let $n = |\mathcal{V}|$, $m = |\mathcal{E}|$. Algorithm 4.4 computes the ℓ-walk kernel in time $\mathcal{O}(n+\ell(n+m)+T_{\text{WDPG}})$, where T_{WDPG} is the time to compute the weighted direct product graph.*

Note that the running time depends on the size of the product graph and $n \ll |G|\cdot|H|$ and $m \ll \|G\|\cdot\|H\|$ is possible as discussed in Section 4.6.3.

The Max-ℓ-walk kernel is the sum of the j-walk kernels with $j \leq \ell$ and, hence, with Equation (4.41) we can also formulate it recursively as

$$K_\ell^\leq(G,H) = \sum_{i=0}^{\ell} \lambda_i K_i^=(G,H) = \sum_{i=0}^{\ell} \lambda_i \sum_{v\in V(G\times_w H)} r_i(v). \tag{4.42}$$

This value can be obtained from Algorithm 4.4 by simply changing the return statement in line 9 according to the right-hand side of the equation without affecting the asymptotic running time.

4.6.4 Explicit Kernel Computation

Provided that the kernel k_W on walks satisfies the conditions discussed in Section 4.4.3, explicit computation schemes could be obtained by enumerating all walks and counting the members of the equivalence classes of the relation $\sim_{k_W}$. We again consider graphs with simple labels from the alphabet $\mathcal{L}$ and the kernel k_W^{δ} given by Equation (4.36). We develop an efficient algorithm tailored to this setting, which computes the feature vectors as specified in Equation (4.38).

A straightforward approach would require to enumerate all walks in order to count the label sequences associated with them. We propose a more elaborated approach that exploits the simple composition of walks similar to the approach for weighted walks, cf. Section 4.6.3. A walk of length ℓ can be decomposed into a walk of length $\ell - 1$ with an additional edge and vertex added at the front. This allows to obtain the number of walks of length ℓ with a given label sequence starting at a fixed vertex v by concatenating $(\tau(v), \tau(vu))$ with all label sequences for walks starting from a neighbor u of the vertex v. Algorithm 4.5 provides the pseudo code of this computation.

Algorithm 4.5: Generating feature vectors of the ℓ-walk kernel

```
Input  : Graph $G$, length parameter $\ell$.
Output : Feature vector $\phi_\ell^=(G)$ of label sequences and counts
         associated with length $\ell$ walks in $G$.
Data   : Feature vectors $\Phi_i^v : \mathcal{L}^{2i+1} \to \mathbb{N}$ of label sequences associated
         with $i$-walks starting at $v$.

1 forall the $v \in V(G)$ do
2     $\Phi_0^v(\tau(v)) \leftarrow 1$                          ▷ Initialization, length 0 walks
3 for $i \leftarrow 1$ to $\ell$ do
4     forall the $u \in V(G)$ and $v \in \mathrm{N}(u)$ do
5         forall the $w$ with $\Phi_{i-1}^v(w) > 0$ do
6             $w' \leftarrow (\tau(u), \tau(uv)) + w$          ▷ Concatenate label sequence
7             $\Phi_i^u(w') \leftarrow \Phi_i^u(w') + \Phi_{i-1}^v(w)$
8 return $\sum_{v \in V(G)} \Phi_\ell^v$                       ▷ Combine vectors
```

Theorem 4.4. *Given a graph G with $n = |G|$ vertices and $m = \|G\|$ edges, Algorithm 4.5 computes the ℓ-walk kernel feature vector $\phi_\ell^=(G)$ in time $\mathcal{O}(n + \ell(n+m)s)$, where s is the maximum number of different label sequences of $(\ell - 1)$-walks staring at a vertex of G.*

Assume Algorithm 4.5 is applied to unlabeled sparse graphs, i.e., $\|G\| = \mathcal{O}(|G|)$, then $s = 1$ and the feature mapping can be performed in time $\mathcal{O}(n + \ell n)$. With Proposition 4.2 we have a total running time to compute a kernel matrix for d graphs of order n of $\mathcal{O}(d\ell n + d^2)$, for $\ell > 0$.

4.6.5 Application to Shortest-Path Kernels

The shortest-path kernel, cf. Equation (4.32), is computed by applying the 1-walk kernel to complete graphs, where edges are annotated by shortest-path distances (Borgwardt and Kriegel, 2005). Note that κ_E then compares shortest-path lengths and can, for example, be realized by the kernels (4.14) or (4.15) or the Dirac kernel. Vertices may be annotated with simple labels or continuous attributes and are compared by κ_V. Therefore, our two approaches to compute walk kernels of fixed length directly yield efficient explicit and implicit computation schemes for the shortest-path kernel. Clearly, the explicit version is only applicable when shortest-path lengths and vertex labels are compared by the Dirac kernel. For the binning kernel (4.16), we could relabel edges as detailed in Section 4.3.2 to obtain graphs with simple labels, which are manageable by the explicit computation scheme.

4.6.6 Experimental Evaluation

We compare the implicit and explicit computation schemes for fixed length walk kernels experimentally with a focus on running times. We present a more comprehensive experimental comparison including prediction accuracies and a large number of different graph kernels at the end of this chapter in Section 4.8. The discussion of running times for walk kernels in Sections 4.6.3 and 4.6.4 suggested that

(i) implicit computation benefits from sparse vertex and edge kernels,

(ii) explicit computation is promising for graphs with a uniform label structure, which exhibit few different features, and then scales to large data sets.

We experimentally analyze this trade-off between label diversity and running time for synthetic and real-world data sets. Finally, we use our walk kernels to compare graphs after applying different levels of label refinement using the Weisfeiler-Lehman method (Shervashidze et al., 2011) and to compute the shortest-path kernel. Both computation schemes consistently produced the same kernel matrix in all experiments. While for the shortest-path kernel explicit mapping—when applicable—clearly outperforms implicit computation, we observe a computational phase transition for walk kernels.

All algorithms were implemented in Java and the default Java HashMap implementation was used to store feature vectors, see Section 4.4.2. Experiments were conducted using Java OpenJDK v1.7.0 on an Intel Core i7-3770

CPU at 3.4GHz (Turbo Boost disabled) with 16GB of RAM using a single processor only. The reported running times are average values over 5 runs. We performed classification experiments using the C-SVM implementation LIBSVM (Chang and Lin, 2011). We report mean prediction accuracies obtained by 10-fold cross-validation repeated 10 times with random fold assignment. Within each fold the regularization parameter C was chosen from $\{10^{-5}, 10^{-4}, \dots, 10^{5}\}$ by cross-validation based on the training set.

Synthetic Data Sets

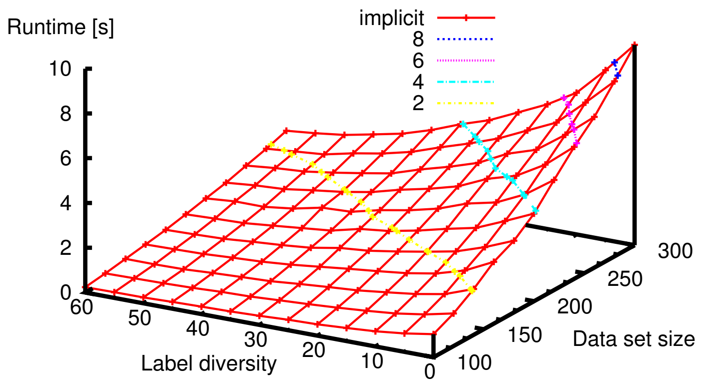
In order to systematically vary the label diversity we generated synthetic graphs by the following procedure: The number of vertices was determined by a Poisson distribution with mean 20. Edges were inserted between a pair of vertices with probability 0.1. The label diversity depends on the parameter p_V. Edges were uniformly labeled; a vertex obtained the label 0 with probability $1 - p_V$. Otherwise the labels 1 or 2 were assigned with equal probability. In addition, we vary the data set size d between 100 and 300 adding 20 randomly generated graphs in each step. We analyze the running time to compute the $d \times d$ kernel matrix. For the product graph based computation we used the Dirac kernel as vertex and edge kernel according to Equation (4.36).

The results are depicted in Figure 4.5, were a label diversity of 50 means that $p_V = 0.5$. Figure 4.5(a) shows that the running time for implicit computation increases with the data set size and decreases with the label diversity. This observation is in accordance with our hypotheses. When the label diversity increases, there are less compatible pairs of vertices and the weighted direct product graph becomes smaller. Consequently, its computation and the counting of weighted walks require less running time. For explicit computation we observe a different trend: While the running time increases with the size of the data set, the approach is extremely efficient for graphs with uniform labels ($p_V = 0$) and becomes slower when the label diversity increases. Combining both results, cf. Figure 4.5(c), shows that both approaches yield the same running time for a label diversity of $p_V \approx 0.3$, while for higher values of p_V implicit computation is preferable and explicit otherwise.

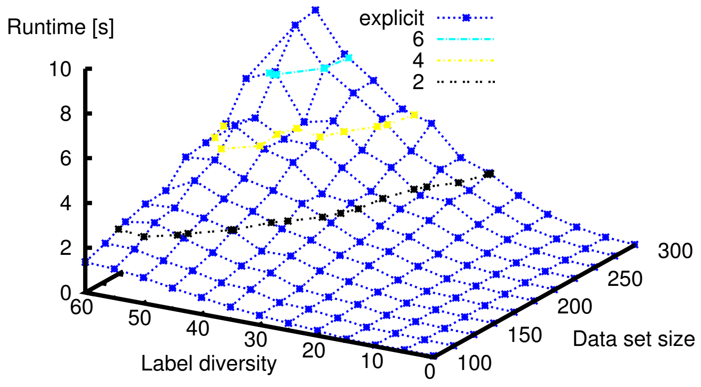
Molecular Data Sets

In the previous section we have observed how both approaches behave when the label diversity is varied. We use a data set of graphs derived from small molecules[7] to analyze the running time on a real-world data set with a predetermined label diversity. Vertex labels correspond to the atom types and edge labels represent single, double, triple and aromatic bonds, respectively. This time we vary the walk length and the data set size by starting with a random

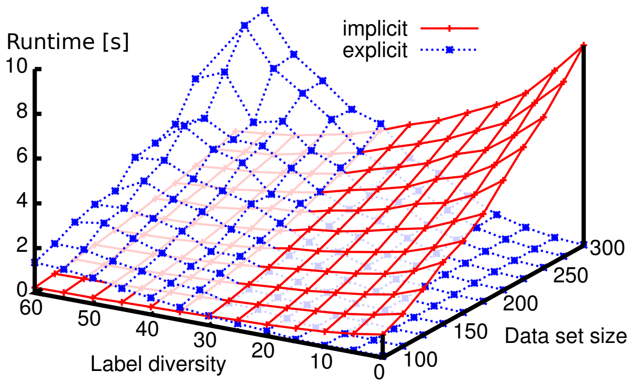
[7]NCI Open Database, GI50, U251; `http://cactus.nci.nih.gov`



(a) Implicit computation



(b) Explicit computation



(c) Implicit and explicit computation

Figure 4.5: Running time to generate the kernel matrix by implicit and explicit computation of walk kernels with fixed length 7 for synthetic data sets with varying label diversity. Figures (a) and (b) show contour lines obtained by linear interpolation.

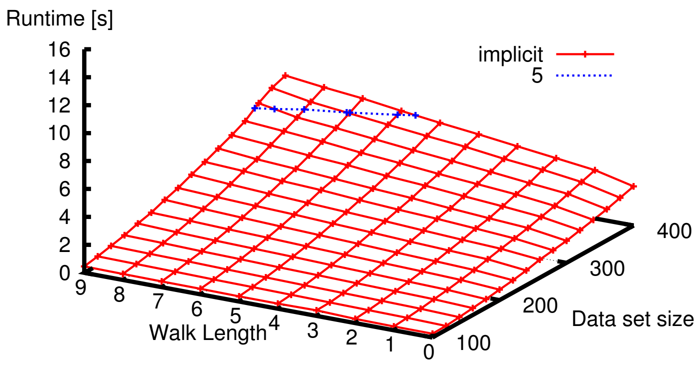
subset and adding additional graphs that were selected randomly from the remaining graphs of the data set.

Figure 4.6(a) shows that the running time of the implicit computation scheme heavily depends on the size of the data set. The increase with the walk length is less considerable. This can be explained by the time T_{WDPG} required to compute the product graph, which is always needed independent of the walk length. For short walks explicit computation is very efficient, even for larger data sets, cf. Figure 4.6(b). However, when a certain walk length is reached the running time increases drastically. This can be explained by the growing number of different label sequence. Notably for walks of length 8 and 9 the running time also largely increases with the data set size. This indicates that the time T_{dot} has a considerable influence on the running time. In the following section we analyze the running time of the different procedures of the two algorithms in more detail. Figure 4.6(c) shows that for walk length up to 7 explicit computation beats implicit computation on the molecular data set.

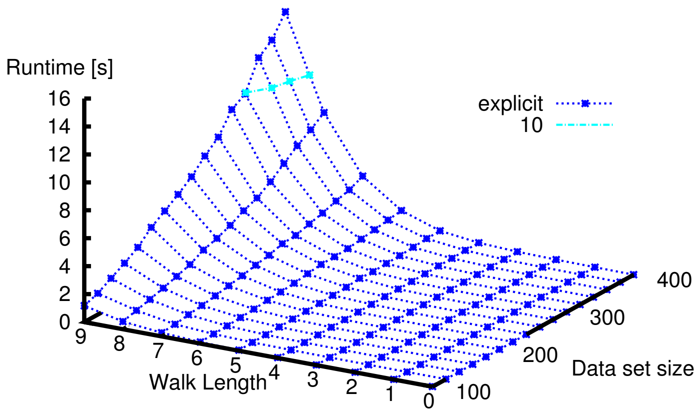
Enzymes and Mutag

We have shown that up to a certain walk length explicit computation is more efficient than implicit computation. We want to clarify the relation between the walk length and the prediction accuracy in a classification task. In addition, we analyze the ratio between the time T_ϕ for computing the explicit mapping and T_{dot} for taking dot products. For the implicit computation scheme we want to clarify the running time of T_{WDPG} and the time required for counting weighted walks. We apply both algorithms to two widely-used data sets, Mutag (188 graphs, 2 classes) and Enzyme (600 graphs, 6 classes), and vary the walk length. See Section 4.8.1 for details on these data sets.

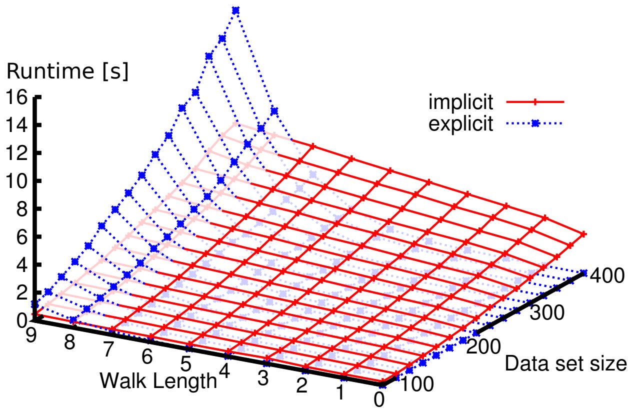
Figure 4.7 shows the running time of both algorithms depending on the walk length and gives the time for product graph computation and explicit mapping, respectively. In addition, the prediction accuracy is presented. For both data sets we observe that up to a walk length of 7 explicit mapping is more efficient. Notably a peak of the accuracy is reached for walk length smaller than 7 in both cases. For the Mutag data set walks of length 3 provide the best results and walks of length 6 for the Enzyme data set, i.e., in both cases explicit mapping should be preferred when computing a walk kernel of fixed length. The running time of the product graph computation is constant and does not depend on the walk length. For explicit mapping the time required to compute the dot product becomes dominating when the walk length is increased. This can be explained by the fact that the generation of the kernel matrix involves a quadratic number of dot product computations, see Proposition 4.2. Note that the given times include a quadratic number of product graph computations while the times for generating the feature vectors include only a linear number of operations.



(a) Implicit computation

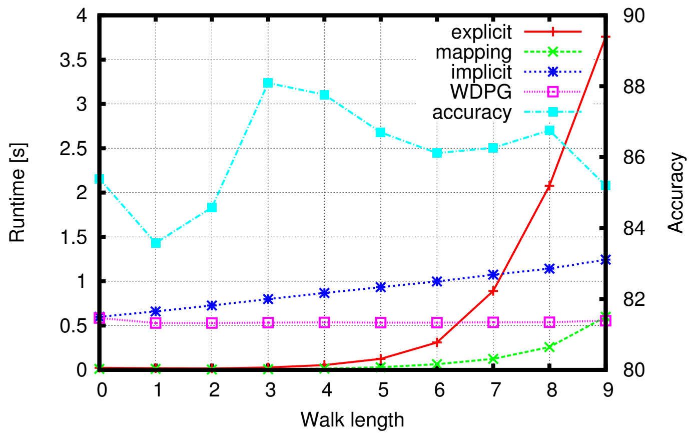


(b) Explicit computation

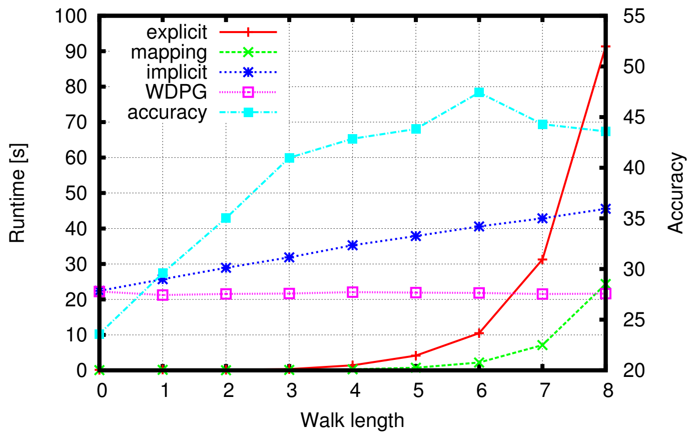


(c) Implicit and explicit computation

Figure 4.6: Running time to generate the kernel matrix by implicit and explicit computation of walk kernels with varying length for the molecular data set. Figures (a) and (b) show contour lines obtained by linear interpolation.



(a) Mutag



(b) Enzyme

Figure 4.7: Running time to generate the kernel matrix and prediction accuracy on the Enzyme and Mutag data sets depending on the walk length.

As a side note, we also compared the accuracy of our kernels based on walks of fixed length to the accuracy reached by the geometric random walk kernel (GRW) according to Equation (4.28), which considers arbitrary walk lengths. The parameter γ of the geometric random walk kernel was selected by cross-validation from $\{10^{-5}, 10^{-4}, \dots, 10^{-2}\}$. We observed that the accuracy of our kernel is competitive on the Mutag data set (GRW 87.3), and considerably better on the Enzyme data set (GRW 31.6), cf. Figure 4.7. This is remarkable, since our approach with walk length 6 yields best results and is efficiently computed by explicit mapping, which would be impossible for the geometric random walk kernel.

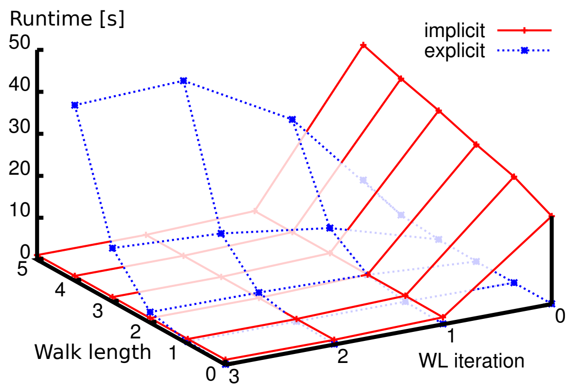


Figure 4.8: Running time to generate the kernel matrix by implicit and explicit computation of walk kernels with varying walk length and iterations of Weisfeiler-Lehman refinement on the Enzyme data set.

Weisfeiler-Lehman Label Refinement

Walk kernels have been successfully combined with label refinement techniques (Mahé et al., 2004). Weisfeiler-Lehman label refinement (WL) has recently been applied to develop graph kernels (Shervashidze et al., 2011). To further analyze the sensitivity w.r.t. label diversity, we again use the Enzyme data set, which consists of graphs with three vertex and two edge labels initially, and apply our algorithms after 0 to 3 iterations of WL, see Figure 4.8.

If no refinement is applied, the explicit mapping approach beats the product graph based algorithm for the used walk lengths. However, as soon as a single iteration of label refinement is performed, the product graph based algorithm becomes competitive for walk length 0 and 1 and outperforms the explicit mapping approach for higher walk lengths. The running times do not change substantially for more iterations of refinement. This indicates that a single iteration of Weisfeiler-Lehman refinement results in a high label diversity that does not increase considerably for more iterations on the Enzyme data set. When using our walk-based kernel as base kernel of a Weisfeiler-Lehman graph kernel (Shervashidze et al., 2011), our observation suggests to start with explicit computation and switch to the implicit computation scheme after few iterations of refinement.

Shortest-Path Kernel

For the shortest-path kernel we found that explicit mapping clearly outperforms implicit computation by several orders of magnitude with respect to running time. This is in accordance with our theoretical analysis and our results suggest to always use explicit computation schemes for this kernel whenever applicable.

4.7 Subgraph Matching Kernels

Recently developed graph kernels primarily focus on large data sets of graphs with simple labels, while kernels for attributed graphs have obtained less attention, cf. Section 4.5. This is remarkable since graph kernels have the unique advantage over vector representations to make use of vertex and edge kernel functions, which compare the annotations. We propose kernels for attributed graphs based on subgraph matchings, i.e., structure-preserving bijections between subgraphs. State-of-the-art kernels based on common subgraphs (Wale et al., 2008; Shervashidze et al., 2009) were shown to be successful for graphs with simple labels, but are not applicable to attributed graphs. Our approach, on the contrary, allows to rate mappings of subgraphs by a flexible scoring scheme comparing vertex and edge attributes by kernels. We show that subgraph matching kernels generalize several known kernels and can be computed by a graph-theoretical algorithm inspired by classical approaches to the maximum common subgraph problem based on association graphs, cf. Section 3.1.1. Instead of deriving a similarity measure from a maximum common subgraph, our approach considers all mappings between subgraphs up to a fixed size and therefore has polynomial running time.

4.7.1 Basic Definitions

According to Definition 2.6 a common subgraph isomorphism between two graphs is an isomorphism between their subgraphs. We consider attributed graphs G, H and denote the set of all common induced subgraph isomorphisms (CISI) between the underlying unlabeled graphs as $\mathcal{B}(G,H)$. In case of labeled graphs we refer to the subset of $\mathcal{B}(G,H)$ that also preserves labels by $\mathcal{I}(G,H)$, cf. Section 2.3.1.

Based on this definition we define the following function and will see later that it is p.s.d. and, thus, a valid kernel.

Definition 4.8 (CISI Kernel). *Let $\mathcal{I}(G,H)$ denote the set of all label preserving CISIs of two labeled graphs G and H and $\lambda : \mathcal{G} \to \mathbb{R}_{\geq 0}$ a weight function. The function*

$$K_{\text{cisi}}(G,H) = \sum_{\psi \in \mathcal{I}(G,H)} \lambda(G'), \tag{4.43}$$

where $G' = G[\text{dom}(\psi)]$, is called common induced subgraph isomorphism kernel.

When vertices and edges are annotated with arbitrary attributes it is inappropriate to require a mapping to preserve labels exactly. To this end, we generalize Definition 4.8 to allow for a more flexible scoring of isomorphisms referred to as *graph matching*.[8]

[8]Please note that the term matching is used ambiguously in the context of graphs and

Definition 4.9 (Subgraph Matching Kernel). *Given two graphs G and H with attributes, let $\mathcal{B}(G,H)$ denote the set of all CISIs between the underlying unlabeled graphs and let $\lambda : \mathcal{G} \to \mathbb{R}_{\geq 0}$ be a weight function. The* subgraph matching kernel *is defined as*

$$K_{\text{sm}}(G,H) = \sum_{\psi \in \mathcal{B}(G,H)} \lambda(G') \prod_{v \in V'} \kappa_V(v, \psi(v)) \prod_{uv \in E(G')} \kappa_E(uv, \psi(u)\psi(v)),$$

where $V' = \operatorname{dom}(\psi)$, $G' = G[V']$ and κ_V, κ_E are vertex and edge kernels.

Theorem 4.5. *The subgraph matching kernel is positive semidefinite.*

Proof. The structure of a graph $G = (V,E)$ with n vertices can be encoded by a tuple $(\mathbf{v}, \mathbf{E})$, where $\mathbf{v} = (v_0, \ldots, v_n)^\top$ is a vector of the vertices V for some fixed ordering and $\mathbf{E}$ is a $n \times n$ matrix of elements $E \uplus \{\epsilon\}$, such that

$$e_{ij} = \begin{cases} v_i v_j & \text{if } v_i v_j \in E, \\ \epsilon & \text{otherwise.} \end{cases}$$

By extending $\mathbf{v}$ and $\mathbf{E}$ by additional rows and columns filled with ϵ-elements we can encode graphs of different size into the same space. Each permutation of the vertices of a graph yields a valid encoding and a graph can be decomposed into all its encodings. This allows us to define a graph kernel by specifying an R-convolution, see Section 4.2.2. Let $R(\mathbf{v}, \mathbf{E}, G)$ be a relation, where $\mathbf{v}$ and $\mathbf{E}$ are defined as above, G is a graph and $R(\mathbf{v}, \mathbf{E}, G) = 1$ if and only if $(\mathbf{v}, \mathbf{E})$ is an encoding of G. Let $R^{-1}(G) = \{(\mathbf{v}, \mathbf{E}) \mid R(\mathbf{v}, \mathbf{E}, G) = 1\}$ be the set of encodings of G. We can now specify the R-convolution kernel

$$k_{\text{enc}}(G,H) = \sum_{\substack{(\mathbf{u},\mathbf{E}) \in R^{-1}(G) \\ (\mathbf{v},\mathbf{F}) \in R^{-1}(H)}} \prod_i \kappa'_V(u_i, v_i) \prod_{i,j} \kappa'_E(e_{ij}, f_{ij}), \tag{4.44}$$

where $\kappa'_V(\epsilon, v) = \kappa'_V(v, \epsilon) = 0$ for all vertices v, $\kappa'_V(\epsilon, \epsilon) = 1$ and $\kappa'_V = \kappa_V$ otherwise. The kernel κ'_E is defined analogously. The product in Equation (4.44) yields one for isomorphic graphs G and H, but only if their encodings are equal, and zero otherwise. For a graph of order n there are $n!$ possible encodings and, thus, $n!n!$ pairs of encodings for two graphs, $n!$ of which correspond to the same bijection. Combining this kernel with a convolution kernel based on the decomposition into induced subgraphs and a suitable chosen weight function yields

$$K(G,H) = \sum_{G' \sqsubseteq G} \sum_{H' \sqsubseteq H} \frac{\lambda(G')}{|G'|!} k_{\text{enc}}(G', H'). \tag{4.45}$$

may refer to a subset of edges in a single graph as in Problem 2.5 or to a bijection between the vertices of two different graphs.

The coefficient accounts for the $n!$ pairs of encodings of two graphs with n vertices, which correspond to the same bijection. Further, the weight $\lambda(G')$ distributes over the addition in Equation (4.44) and each pair of encodings corresponding to an isomorphism ψ is weighted by $\lambda(G') = \lambda(G[\mathrm{dom}(\psi)])$. Consequently, the kernel in Equation (4.45) is equivalent to K_{sm}. □

We can identify Definition 4.8 as a special case of Definition 4.9, where

$$\kappa_V(v,v') = \begin{cases} 1 & \text{if } \tau_1(v) = \tau_2(v'), \\ 0 & \text{otherwise and} \end{cases} \qquad \kappa_E(e,e') = \begin{cases} 1 & \text{if } \tau_1(e) = \tau_2(e'), \\ 0 & \text{otherwise.} \end{cases}$$

Using these vertex and edge kernels, only isomorphisms that preserve labels according to Equations (2.4), (2.5) contribute to the kernel value. Hence, the CISI kernel is a special case of the subgraph matching kernel and we may state the following corollary.

Corollary 4.3. *The CISI kernel is positive semidefinite.*

4.7.2 Relations to other Kernels

In this section we show how the CISI kernel as special case of the subgraph matching kernel is related to the subgraph kernel. Further we show how an explicit computation scheme can be obtained for CISI by means of the general approach for R-convolution kernels described in Section 4.4.3. Finally, we observe that subgraph matching kernels can be used to compute the pharmacophore kernel proposed for chemical compounds.

Relation to the Subgraph Kernel

We consider the induced variant of the subgraph kernel according to Definition 4.4. The induced subgraph kernel basically counts isomorphic induced subgraphs, while the CISI kernel counts the number of isomorphisms between induced subgraphs. Since there may be more than one isomorphism between a pair of isomorphic induced subgraphs that exhibit non-trivial automorphisms, both concepts differ in detail.

Theorem 4.6. *Let $K_{\sqsubseteq}$ be the induced subgraph kernel with weight function $\lambda_{\sqsubseteq}$ and K_{cisi} the CISI kernel with weight function*

$$\lambda_{\mathrm{cisi}}(G) = \frac{\lambda_{\sqsubseteq}(G)}{|\operatorname{Aut}(G)|}. \tag{4.46}$$

Then $K_{\mathrm{cisi}}(G,H) = K_{\sqsubseteq}(G,H)$ for all graphs $G, H \in \mathcal{G}$.

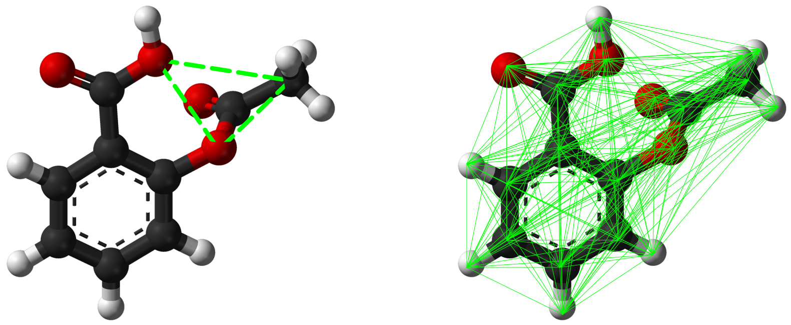
Proof. For each pair (G', H') that contributes to the sum of Equation (4.31), $G' \simeq H'$ holds. CISIs exist for these pairs of graphs only. There are $|\operatorname{Aut}(G')| = |\operatorname{Aut}(H')|$ isomorphism between G' and H', each of which is contained in $\mathcal{I}(G,H)$ and contributes to Equation (4.43). This is compensated by the correction term $|\operatorname{Aut}(G')|^{-1}$. □

We have developed a general approach to derive feature maps of R-convolution kernels in Section 4.4.3 and have already observed that the subgraph matching kernel is an instance of a R-convolution kernel, cf. proof of Theorem 4.5. The decomposition into induced subgraphs is crucial for the induced subgraph kernel and the R-convolution kernel specified in Equation (4.45). We consider the CISI kernel, i.e., the subgraph matching kernel where κ_V and κ_E are Dirac kernels, and chose $\lambda(G) = |\operatorname{Aut}(G)|^{-1}$. Following the proof of Theorem 4.5 and the above argument in Theorem 4.6 we obtain

$$K_{\text{sm}}(G,H) = \sum_{G' \sqsubseteq G} \sum_{H' \sqsubseteq H} \frac{k_{\text{enc}}(G',H')}{|G'|! \cdot |\operatorname{Aut}(G')|} = \sum_{G' \sqsubseteq G} \sum_{H' \sqsubseteq H} k_{\simeq}(G',H').$$

The isomorphism kernel $k_{\simeq}$ is binary and, thus, according to Theorem 4.1 we can derive a feature map based on a function $C : \mathcal{G} \to \mathbb{N}$ with $C(G) = C(H) \iff k_{\simeq}(G,H) = 1$. Let $F(G,i) = \{G' \sqsubseteq G \mid C(G') = i\}$, then the corresponding feature vector $\phi(G)$ distinguishes subgraphs up to isomorphism and counts their number of occurrences in G. Computing such a function C corresponds to graph canonization, Problem 2.2, which is well-studied. By solving the graph canonization problem instead of graph isomorphism we can explicitly compute a feature map of the subgraph (matching) kernel. Although graph canonization clearly is at least as hard as graph isomorphism, the number of canonizations required is linear in the number of subgraphs, while a quadratic number of isomorphism tests would be required for all pairs of subgraphs. The same principle has been applied in Section 3.1.3 to obtain a maximum common subgraph algorithm with improved theoretical bounds on running time. Here, the gap in terms of running time even increases when computing a whole kernel matrix and not just the kernel value for two graphs, cf. Section 4.4.1. We will show this experimentally in Section 4.7.4.

Recently, several graph kernels have been proposed that are based on subgraphs, see Section 4.5.2. The graphlet kernel is an instance of the induced subgraph kernel and computed by an explicit mapping scheme. However, only unlabeled graphs of order five or less are considered by this kernel, such that the canonizing function can be computed easily. Furthermore the number of features, i.e., different subgraphs of small size, stays easily manageable when labels are ignored. However, the same approach was also taken in (Wale et al., 2008) considering larger connected subgraphs of labeled graphs derived from molecular structures. The arguments above suggest that explicit computation of the subgraph matching kernel is far more efficient. Note that we considered a very restricted version of the subgraph matching kernel, for which we could derive explicit feature vectors which generalize those of several known kernels for graphs with highly restricted labels. Subgraph matching kernels allow to specify arbitrary vertex and edge kernel and we propose an implicit computation scheme in Section 4.7.3. This flexibility when comparing attributes comes at the cost that efficient explicit computation schemes become difficult or impossible, since the condition of Theorem 4.1 is not necessarily fulfilled.



(a) Pharmacophore (b) Molecular distance graph $G(M)$

Figure 4.9: Structure of Aspirin in 3D, where a pharmacophore according to the model used by Mahé et al. (2006) is highlighted in green (a) and the edges of the molecular distance graph (b).[9]

Relation to the Pharmacophore Kernel

Mahé et al. (2006) proposed a kernel to compare chemical compounds based on characteristic features together with their relative spatial arrangement, so-called *pharmacophores*. To this end, a molecule is represented by a set of pairs $M = \{(x_i, l_i) \in \mathbb{R}^3 \times \mathcal{A}\}_i$, where x_i are the coordinates of a feature i in a 3-dimensional space and l_i is an associated annotation. A pharmacophore is a triple of pairwise distinct features, see Figure 4.9(a), and the set of pharmacophores associated with a molecule M is

$$\mathfrak{P}(M) = \{(a_1, a_2, a_3) \in M^3 \mid a_1 \neq a_2, a_1 \neq a_3, a_2 \neq a_3\}.$$

The pharmacophore kernel between two molecules M and M' is defined as

$$K_{\mathrm{p}}(M, M') = \sum_{p \in \mathfrak{P}(M)} \sum_{p' \in \mathfrak{P}(M')} k_{\mathrm{i}}(p, p') k_{\mathrm{s}}(p, p')$$

and measures the similarity of two molecules based on triples of similar characteristic features with a similar spatial arrangement, which is quantified by the two kernels k_{i} and k_{s}, respectively. These are defined as

$$k_{\mathrm{i}}(p, p') = \prod_{i=1}^{3} k_{\mathrm{feat}}(l_i, l'_i) \quad \text{and}$$

$$k_{\mathrm{s}}(p, p') = \prod_{i=1}^{3} k_{\mathrm{dist}}(\|x_i - x_{i+1}\|, \|x'_i - x'_{i+1}\|),$$

[9] The illustration is based on the ball-and-stick model of the aspirin molecule published by Ben Mills, `http://commons.wikimedia.org/wiki/File:Aspirin-B-3D-balls.png`.

where the index $i+1$ is taken modulo 3, $p = ((x_1, l_1), (x_2, l_2), (x_3, l_3))$ and $p' = ((x'_1, l'_1), (x'_2, l'_2), (x'_3, l'_3))$. The kernels k_{feat} and k_{dist} can be specified for the comparison of annotations and distances, respectively.

From the representation M of a molecule as used by the pharmacophore kernel we can construct the *molecular distance graph* $G(M) = (V, E)$ with attributes α, which is the complete graph defined as

$$\begin{aligned} V &= \{v_1, \dots, v_{|M|}\} && \text{with } \alpha(v_i) = l_i && 1 \le i \le |M| \\ E &= [V]^2 && \text{with } \alpha(v_i v_j) = \|x_i - x_j\| && 1 \le i < j \le |M|. \end{aligned}$$

An example of the molecular distance graph derived from the drug Aspirin is shown in Figure 4.9(b).

Theorem 4.7. *Let K_{p} be a pharmacophore kernel and K_{sm} the subgraph matching kernel with weight function*

$$\lambda(G) = \begin{cases} 6 & \textit{if } |G| = 3, \\ 0 & \textit{otherwise} \end{cases}$$

and vertex and edge kernels defined as

$$\begin{aligned} \kappa_V(v, v') &= k_{\text{feat}}(\alpha(v), \alpha(v')) \quad \textit{and} \\ \kappa_E(e, e') &= k_{\text{dist}}(\alpha(e), \alpha(e')). \end{aligned}$$

Then $K_{\text{p}}(M, M') = K_{\text{sm}}(G(M), G(M'))$ holds.

Proof. The weight function λ ensures that only subgraphs with three vertices contribute to the value of K_{sm}. Since $G(M)$ is a complete graph, every common subgraph induced by three vertices is a triangle, i.e., all subsets with three features and their pairwise distances are taken into account. For a pair of two subgraphs with three vertices, there are 6 bijections considered by the subgraph matching kernel. For each subset with three elements there are six different triples representing all possible permutations. The pharmacophore kernel considers all pairs of such triples in $\mathfrak{P}(M) \times \mathfrak{P}(M')$. For two subsets with three elements, there are 36 such pairs and $3! = 6$ combinations correspond to the same bijection between the elements. Thus, multiplying the value of 3-element subgraph matchings by 6 compensates for this. □

4.7.3 Kernel Computation

In this section we propose an algorithm to implicitly compute the CISI and subgraph matching kernel. Our technique is inspired by a classical result of Levi (1973) who observed a relation between common subgraphs of two graphs and cliques in their association graph. This idea is commonly used to reduce the maximum common induced subgraph problem to a maximum clique problem as detailed in Section 3.1.1.

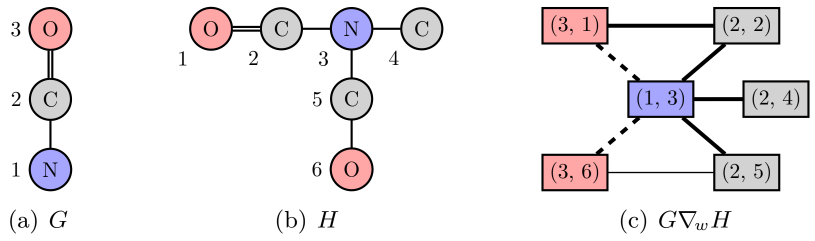


Figure 4.10: Two attributed graphs G (a) and H (b) and their weighted association graph $G\nabla_w H$ (c). We assume the vertex kernel to be the Dirac kernel and κ_E to be 1 if edge labels are equal and $\frac{1}{2}$ if one edge label is "=" and the other is "-". Thin edges in $G\nabla_w H$ represent edges with weight $\frac{1}{2}$, while all other edges and vertices have weight 1; dashed lines represent d-edges.

Here, we use the fact that for two graphs G and H the vertex subset C is a clique in the association graph $G\nabla H$ defined according to Definition 3.1 if and only if there is a corresponding CISI $\psi \in \mathcal{I}(G, H)$. As a consequence we can enumerate (or count) all CISIs by enumerating (counting) the cliques of the association graph. In order to compute the subgraph matching kernel we adapt the approach and the definition of the association graph. Similar to the weighted direct product graph used to compute walk kernels, we annotate the association graph with the values of vertex and edge kernels.

Definition 4.10 (Weighted Association Graph). *For two graphs $G = (V, E)$, $H = (V', E')$ with attributes and given vertex and edge kernels κ_V and κ_E, the* weighted association graph *(WAG) is denoted by $G\nabla_w H = (\mathcal{V}, \mathcal{E}, w)$ and defined by the vertex set*

$$\mathcal{V} = \{(v, v') \in V \times V' \mid \kappa_V(v, v') > 0\}$$

and the edge set $\mathcal{E} = \mathcal{E}_c \uplus \mathcal{E}_d$ consisting of c-edges *$\mathcal{E}_c$ and* d-edges *$\mathcal{E}_d$, where*

$$\begin{aligned} \mathcal{E}_c &= \left\{(u, u')(v, v') \in [\mathcal{V}]^2 \;\middle|\; uv \in E \wedge u'v' \in E' \wedge \kappa_E(uv, u'v') > 0\right\} \\ \mathcal{E}_d &= \left\{(u, u')(v, v') \in [\mathcal{V}]^2 \;\middle|\; uv \notin E \wedge u'v' \notin E' \wedge u \neq v \wedge u' \neq v'\right\}. \end{aligned}$$

The weights $w : \mathcal{V} \uplus \mathcal{E} \to \mathbb{R}$ of vertices and edges are determined according to

$$\begin{aligned} w(v) &= \kappa_V(v, v') & & \forall v = (v, v') \in \mathcal{V} \\ w(e) &= \begin{Bmatrix} \kappa_E(uv, u'v') & \text{if } e \in \mathcal{E}_c \\ 1 & \text{otherwise} \end{Bmatrix} & & \forall e \in \mathcal{E}, \text{where } e = (u, u')(v, v'). \end{aligned}$$

An example of two graphs and their weighted direct product graph obtained for specific vertex and edge kernels is shown in Figure 4.4. The distinction between c-edges, which represent common adjacency, and d-edges representing common non-adjacency is due to Koch (2001). This allows to characterize the cliques corresponding to isomorphisms between connected common subgraphs, also see Section 3.3.2. We will make use of edge types later. The weight of a clique then is the product of the weights of all vertices and edges contained in it. In Figure 4.10, there are two maximum cliques $C_1 = \{(1,3),(2,2),(3,1)\}$ and $C_2 = \{(1,3),(2,5),(3,6)\}$ of size three, which both correspond to an isomorphism between G and a subgraph of H. The clique C_2 contains the thin edge with weight one half since the edge with label "`=`" is mapped to an edge with label "`-`" according to the mapping associated with the clique. Thus, the weight of C_1 is one and the weight of C_2 is one half. There is an interesting relation between the weighted association graph and the weighted direct product graph. Note that the only difference according to their definition is that the weighted association graph contains additional d-edges, which all have weight one. The set of d-edges encode common non-adjacency and, thus, these edges exactly correspond to the edges of the direct product of the complement of the input graphs. Formally, we have

$$\begin{aligned} V(G\nabla_w H) &= V(G \times_w H) \\ E(G\nabla_w H) &= \underbrace{E(G \times_w H)}_{\mathcal{E}_c} \uplus \underbrace{E(\overline{G} \times_w \overline{H})}_{\mathcal{E}_d}, \end{aligned}$$

where the weights of vertices and edges are inherited from the weighted direct product graph. Here, we assume the edges of the complement graph to be unlabeled and the edge kernel to be the one kernel, i.e., all edges in $\mathcal{E}_d$ obtain weight one. As a consequence, if at least one of the graphs G and H is complete, $G\nabla_w H = G \times_w H$ holds. The weighted association graph can easily be computed analogously to the weighted direct product, cf. Algorithm 4.3.

Algorithm 4.6 makes use of the weighted association graph in order to compute the subgraph matching kernel by enumeration of cliques as follows: A current clique is extended stepwise by all vertices preserving the clique property. These vertices form the candidate set P. Whenever the current clique C is extended by a new vertex v, the weight of the vertex itself (line 8) and all the edges connecting v to a vertex in C (line 10) are multiplied with the weight c of the current clique to obtain the weight c' of the new clique C'. Then the global variable, which at the end holds the result, is updated by adding the weight of the new clique weighted by $\lambda(G[C'])$ (line 11). Given a clique $C = \{(v_1, v_1'), \dots, (v_n, v_n')\}$ in $G\nabla_w H$, in slight abuse of notation we write $G[C]$ to refer to the induced subgraph $G[\{v_1, \dots, v_n\}]$. In order to extend the clique further the procedure SMKernel is called recursively with the new clique and a modified candidate set as parameter. The algorithm effectively avoids duplicates by removing a vertex from the candidate set after all cliques containing it have been exhaustively explored (line 13). Note that the candidate set P in Algorithm 4.6 can be implemented using a single array passed to each recursive call and two local indices. The array contains all vertices in $\mathcal{V}$ and the indices indicate the first and last element of the current

Algorithm 4.6: Subgraph matching kernel

```
Input  : Graphs $G$, $H$, kernels $\kappa_V$, $\kappa_E$, weight function $\lambda$.
Param. : Weight $c$ of the clique $C$, candidate set $P$.
Data   : Global variable $value$.
Output : Value $K_{\mathrm{sm}}(G, H)$ of the subgraph matching kernel.

$(\mathcal{V}, \mathcal{E}, w) \leftarrow$ WAG$(G, H, \kappa_V, \kappa_E)$        ▷ Compute $G \nabla_w H$
$value \leftarrow 0$
SMKernel$(1, \emptyset, \mathcal{V})$
return $value$

Procedure SMKernel$(c, C, P)$
    while $|P| > 0$ do
        $v \leftarrow$ arbitrary element of $P$
        $C' \leftarrow C \uplus \{v\}$
        $c' \leftarrow c \cdot w(v)$                        ▷ Multiply by vertex weight
        forall the $u \in C$ do
            $c' \leftarrow c' \cdot w(uv)$                   ▷ Multiply by edge weights
        $value \leftarrow value + c' \cdot \lambda(G[C'])$    ▷ Increase value
        SMKernel$(c', C', P \cap \mathrm{N}(v))$              ▷ Extend clique
        $P \leftarrow P \setminus \{v\}$
```

candidate set occupying a contiguous interval of the array. In preparation of a recursive call the indices are adjusted and some vertices are swapped within the current interval, if necessary, such that the new candidate set again occupies a contiguous section of the array. The candidate set used in a recursive call always forms a subset of the current candidate set and it does not matter in which order vertices are extracted from P. Hence, swapping elements within the current interval does not harm correctness and it is not necessary to restore the array when performing backtracking.

Restriction to Subgraph Classes

In this section we discuss restrictions to certain classes of subgraphs, their relation to cliques in the association graph and appropriate modifications of the enumeration algorithm. The restricted variants remain p.s.d., since they are equivalent to the general subgraph matching kernel with a suitably chosen weight function. The intention of restricting the class of subgraphs is twofold: On the one hand this may prune the search space of the clique detection algorithm and thus reduce the running time. On the other hand it depends on the application domain which graphs yield a meaningful similarity measure. The first constraint introduced here restricts the search space to subgraphs of a specified maximum size, the second to connected subgraphs.

Since finding a maximum clique or a maximum CISI is known to be an NP-hard problem, it is required in practice to restrict the size of the subgraphs considered. Modifying Algorithm 4.6 to stop the recursion whenever a fixed maximum size k has been reached, effectively restricts the size of the cliques and thereby the size of the matched subgraphs, which is quantified by the number of vertices (see Algorithm 4.7, line 19).

Restricting to connected subgraphs may also substantially reduce the search space, especially when graphs are sparse. Shervashidze et al. (2009) observed that for unlabeled graphs using disconnected subgraphs is beneficial in classification tasks. However, in general disconnected subgraphs convey only limited structural information and may therefore be considered less relevant or even inappropriate to determine a meaningful similarity. This depends on the specific data set and application and we will discuss this question later based on experimental results. We can realize the constraint by an adaption of a technique proposed by Koch (2001); Cazals and Karande (2005). Let *c-cliques* be defined as in Section 3.3.2 as cliques where each vertex of the clique is connected to all other vertices of the clique via at least one path containing only c-edges. There is a one-to-one correspondence between c-cliques and isomorphisms between connected subgraphs. For example, in Figure 4.10 the clique $C = \{(1,3),(3,6)\}$ corresponds to an isomorphism between the disconnected subgraphs consisting of two isolated vertices. Since the single edge connecting vertices of the clique is a d-edge, C is not a c-clique. We obtain a larger clique $C' = C \uplus \{(2,5)\}$ by adding one vertex. Then C' is a c-clique and corresponds to a mapping of connected subgraphs. Algorithm 4.6 can be adapted to enumerate only c-cliques by making sure that only vertices are added that are adjacent to a vertex in the current clique via at least one c-edge. This modification is presented as Algorithm 4.7, which maintains two candidate sets P and D. The set P contains vertices that are adjacent to all vertices of the current clique and are connected to at least one of them via a c-edge. The set D contains vertices connected to all vertices of the current clique exclusively by d-edges. Since only vertices in P are added to the current c-clique the new clique again is c-clique. Note that a vertex u can change over from D to P after adding a vertex v to C if and only if uv is a c-edge (cf. line 20). The set of vertices adjacent to a vertex v via a c-edge is denoted by $\mathrm{N}_c(v) = \{u \in \mathrm{N}(v) \mid uv \in \mathcal{E}_c\}$ and $\mathrm{N}_d(v) = \{u \in \mathrm{N}(v) \mid uv \in \mathcal{E}_d\}$ is defined analogously for d-edges.

Running Time Analysis

In this section we analyze the worst-case running time of the proposed algorithm to compute the subgraph matching kernel and discuss practical considerations.

Algorithm 4.7: Connected subgraph matching kernel with fixed size

```
Input  : Graphs $G$, $H$, kernels $\kappa_V$, $\kappa_E$, weight function $\lambda$, size $k > 0$.
Param. : Weight $c$ of the clique $C$, candidate sets $P$ and $D$.
Data   : Global variable $value$.
Output : Value of the connected subgraph matching kernel.

$(\mathcal{V}, \mathcal{E}, w) \leftarrow$ WAG$(G, H, \kappa_V, \kappa_E)$          ▷ Compute $G \nabla_w H$
$value \leftarrow 0$
$P \leftarrow \mathcal{V}$
while $|P| > 0$ do
    $v \leftarrow$ arbitrary element of $P$
    $C \leftarrow \{v\}$
    $value \leftarrow value + \lambda(G[C]) \cdot w(v)$
    if $|C| < k$ then                                  ▷ Check size restriction
        CSMKernel$(w(v), C, P \cap \mathrm{N}_c(v), P \cap \mathrm{N}_d(v))$   ▷ Extend clique
    $P \leftarrow P \setminus \{v\}$
return $value$

Procedure CSMKernel$(c, C, P, D)$
    while $|P| > 0$ do
        $v \leftarrow$ arbitrary element of $P$
        $C' \leftarrow C \uplus \{v\}$
        $c' \leftarrow c \cdot w(v)$                     ▷ Multiply by vertex weight
        forall the $u \in C$ do
            $c' \leftarrow c' \cdot w(uv)$               ▷ Multiply by edge weights
        $value \leftarrow value + \lambda(G[C']) \cdot c'$   ▷ Increase value
        if $|C'| < k$ then                             ▷ Check size restriction
            $P' \leftarrow (P \cap \mathrm{N}(v)) \uplus (\mathrm{N}_c(v) \cap D)$
            $D' \leftarrow D \cap \mathrm{N}_d(v)$
            CSMKernel$(c', C', P', D')$                ▷ extend clique
        $P \leftarrow P \setminus \{v\}$
```

Complexity. The running time of Algorithm 4.6 depends on the number of cliques in the association graph. Since there is a one-to-one correspondence between cliques and bijections contributing to the kernel value, we can derive an upper bound for the number of cliques in the weighted association graph $G\nabla_w H$ by considering the number of possible bijections. There are $\binom{n}{k}$ induced subgraphs of size k in a graph with n vertices and up to $k!$ isomorphisms between graphs of size k. Thus, we have at most

$$\mathfrak{C}(k) = \sum_{i=0}^{k} i! \binom{n}{i}\binom{m}{i} \leq \sum_{i=0}^{k} \binom{nm}{i}$$

cliques of size up to k in $G\nabla_w H$, where $n = |G|$ and $m = |H|$. Note that just before every recursive call of SMKernel the global variable value is increased by the weight of a new clique. Hence, the number of recursive calls is bounded by $\mathfrak{C}(k)$. The running time to compute the new clique and its weight (line 7–10, Algorithm 4.6) is $\mathcal{O}(k)$, when k is the size of the current clique. The running time required to compute the new candidate set by intersection in preparation of the recursive call is linear in the order of the weighted association graph. Therefore the worst-case running time of Algorithm 4.6 (modified to stop recursion at depth k) is $\mathcal{O}(T_{\text{WAG}} + N \cdot \mathfrak{C}(k)) = \mathcal{O}(T_{\text{WAG}} + k \cdot N^{k+1})$, where $N = nm$ is the order of the weighted association graph and T_{WAG} the time required to compute the weighted association graph. With the same arguments this bound also holds for Algorithm 4.7. Although the number c-cliques is expected to be considerably less in practice, in particular for sparse graphs. However, for complete input graphs G, H, every clique in $G\nabla_w H$ is a c-clique.

Practical considerations. The analysis of the complexity shows that a reasonable performance in practice can only be expected when the maximum size of the subgraphs considered is restricted. Therefore, the approach competes against subgraph or graphlet kernels counting all subgraphs up to a certain size two graphs have in common. Besides the differences described in Section 4.7.2, the methods of computation exhibit substantially different characteristics: The running time of our algorithm heavily depends on the number of allowed mappings of subgraphs, which is related to the size and density of the product graph. For instances with diverse labels in combination with a restrictive vertex kernel (e.g., the Dirac kernel) the size of the product graph is typically considerably reduced, such that $|G\nabla_w H| \ll |G| \cdot |H|$. In a similar way diverse edge labels may diminish the number of edges. Due to d-edges sparse graphs tend to have dense product graphs and contain a large number of cliques. However, in this case the number of enumerated cliques can be substantially reduced by restricting to c-cliques. The computation of the graphlet kernel is based on explicit mapping into feature space. While this is beneficial for certain data sets, the number of subgraphs quickly becomes very large for graphs with diverse labels rendering explicit mapping prohibitive (Shervashidze et al., 2009). Furthermore, subgraph kernels are not applicable to attributed graphs. In these respects, our approach is complementary to subgraph kernels and shows promise for instances for which these approaches fall short. The discussion of practical considerations is similar as for the implicit and explicit computation schemes of walk-based kernels, where we experimentally observed a computational phase transition, cf. Section 4.6. We again systematically investigate the running time of explicit and implicit computation of the subgraph (matching) kernel in the following.

4.7.4 Experimental Evaluation

In this section we compare the (connected) subgraph matching kernel to closely related kernels, i.e., the pharmacophore kernel and the subgraph kernel, with focus on running times. We present a comparative experimental evaluation in Section 4.8. For the pharmacophore kernel (Mahé et al., 2006) we used the `C++` implementation provided by Perret and Mahé (2006). All other algorithms were implemented in Java and experiments were conducted using Sun Java JDK v1.6.0 on an Intel Xeon E5430 machine at 2.66GHz with 8GB of RAM using a single processor only.

Implicit vs. Explicit Computation

As detailed in Section 4.7.2 a restricted version of the subgraph matching kernel computes results closely related to the subgraph kernel, a special instance of which is the graphlet kernel. The explicit computation scheme, which then becomes possible is expected to be favorable in terms of running time. We have reimplemented a variation of the graphlet kernel taking connected induced subgraphs with three vertices and simple vertex and edge labels into account. Since the only possible features are triangles and paths of length two, graph canonization is realized by selecting the lexicographically smallest string obtained by concatenation of labels. For a path $(u, \dots, v)$, two possible strings can be obtained, namely $\tau(u) \dots \tau(v)$ and $\tau(v) \dots \tau(u)$; for a triangle, each choice of a start vertex and the direction for traversing the cycle yields a total number of six strings. Our implementation is similar to the approach used by Shervashidze et al. (2011) as extension of the original graphlet kernel (Shervashidze et al., 2009) to the domain of labeled graphs. We refer to this method as graphlet kernel in the following. We compared the graphlet kernel to the connected subgraph matching kernel with $\lambda(G) = \lambda_{\sqsubseteq}(G) = 1$ if $|G| = 3$ and 0 otherwise. In order not to penalize the running time of the connected subgraph matching kernel by additional automorphism computations, the weight function does not consider the number of automorphisms as in Equation (4.46) and, consequently, not the same kernel values are computed.

For real-world instances we observed that explicit computation outperforms implicit computation by several orders of magnitude, see Section 4.8.

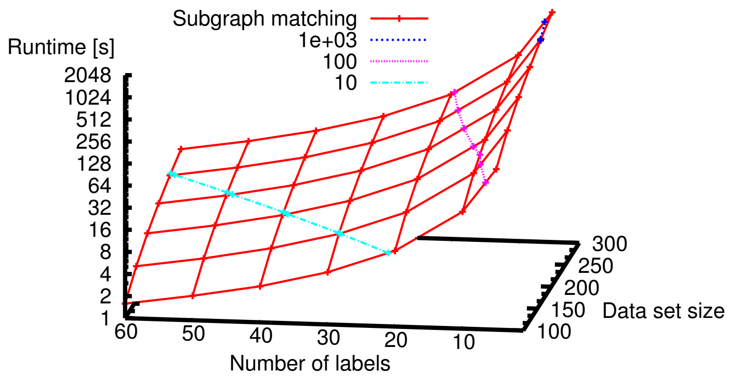
This in accordance with our theoretical analysis. However, the practical considerations suggest that explicit and implicit computation behave complementary and subgraph matching kernels become competitive if a sufficient small and sparse weighted product graphs is generated, which occurs for graphs with increasing label diversity as for the walk-based kernels. Hence, we randomly generated graphs with the following procedure: The number of vertices was determined by a Poisson distribution with mean 60. Edges were inserted between a pair of vertices with probability 0.5. Labels for vertices and edges were assigned with equal probability, whereas the size of the label alphabet $\mathcal{L} = \mathcal{L}_V = \mathcal{L}_E$ is varied from 1, i.e., uniform labels, to 65. Note that the graphs obtained by this procedure have different characteristics than those used to show the computational phase transition for walk-based kernels. We vary the data set size m between 100 and 300 adding 50 randomly generated graphs in each step. We analyze the running time to compute the $m \times m$ kernel matrix. For the subgraph matching kernel we used the Dirac kernel as vertex and edge kernel.

Figure 4.11 shows a computational phase transition: For this synthetic data set the subgraph matching kernel is more efficient than the graphlet kernel for instances with 20-30 different labels and its running time increases exponentially when the number of labels decreases. The graphlet kernel in contrast is more efficient for graphs with uniform or few labels. For more than 10 different labels, there is only a moderate increase in running time. This can be explained by the fact that the number of features contained in the graphs does not increase considerably as soon as a certain number of different labels is reached. The enumeration of triangles dominates the running time for this relatively dense synthetic data set. The running time behavior of the subgraph matching is as expected and is directly related to the size and number of edges in the weighted association graph.

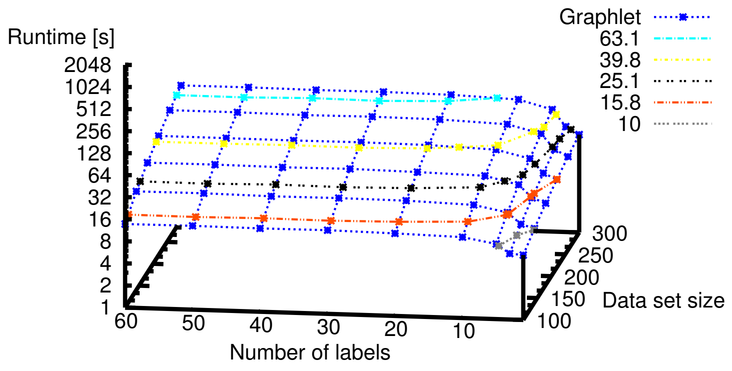
Our synthetic data set differs from typical real-world instances, since we generated dense graphs with many different labels, which are assigned uniformly at random. For real-world data sets the graphlet kernel consistently outperforms the subgraph matching kernel by orders of magnitude, see Section 4.8. It would be interesting to further investigate where this computational phase transition occurs for larger subgraphs and to analyze if the implicit computation scheme then becomes competitive for instances of practical relevance. This requires the implementation of non-trivial graph canonization algorithms and remains future work. The results we obtained clearly suggest to prefer the explicit computation schemes when no flexible scoring by vertex and edge kernels is required.

Subgraph Matching vs. Pharmacophore Kernel

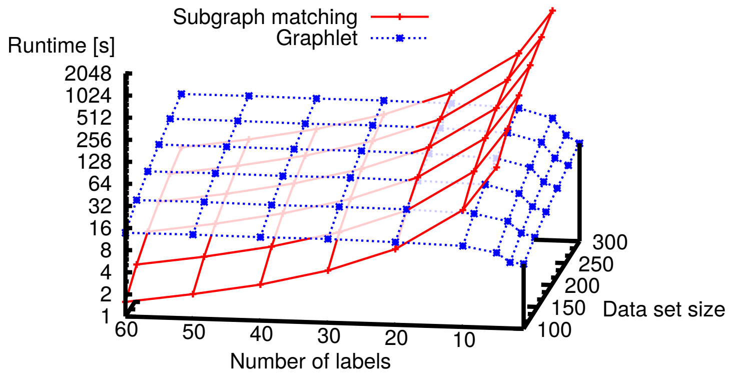
The subgraph matching kernel can be used to compute the pharmacophore kernel, see Section 4.7.2. We compared our implementation of the subgraph



(a) Subgraph matching kernel (implicit)



(b) Graphlets with three vertices (explicit)



(c) Implicit and explicit computation

Figure 4.11: Running time to generate the kernel matrix by implicit and explicit computation for synthetic data sets with varying size of the label alphabet. Figures (a) and (b) show contour lines obtained by linear interpolation.

Table 4.2: Running time in minutes required to compute the kernel matrix.

	BZR	COX2	DHFR	ER
PH with $\kappa_E = K_{\text{RBF}}^{\text{dist}}$	419.98	728.85	776.02	2100.77
SM with $\kappa_E = K_{\text{RBF}}^{\text{dist}}$	94.96	176.52	148.49	394.06
PH with $\kappa_E = K_{\Delta}^{\text{dist}}$	154.59	261.73	333.24	861.15
SM with $\kappa_E = K_{\Delta}^{\text{dist}}$	1.75	2.76	3.44	6.37

Table 4.3: Mean classification accuracy and standard deviation.

	BZR	COX2	DHFR	ER
PH with $\kappa_E = K_{\text{RBF}}^{\text{dist}}$	76.49±1.42	**72.39**±1.51	77.87±1.44	81.15±1.00
SM with $\kappa_E = K_{\text{RBF}}^{\text{dist}}$	76.29±1.25	**72.39**±1.51	77.97±1.37	81.15±1.00
PH with $\kappa_E = K_{\Delta}^{\text{dist}}$	**77.59**±1.36	72.12±1.12	**79.29**±0.93	**82.56**±0.72
SM with $\kappa_E = K_{\Delta}^{\text{dist}}$	**77.56**±1.34	72.15±1.08	**79.39**±1.05	**82.60**±0.72

matching kernel to the implementation of pharmacophore kernel contained in the library ChemCPP provided by Perret and Mahé (2006). In order to demonstrate the usefulness of the pharmacophore kernel, Mahé et al. (2006) used the four data sets COX2, BZR, DHFR and ER (Sutherland et al., 2003), which come with 3D coordinates. For comparison purpose we generated benchmark data sets from the publicly available data[10] by removing highly similar structures and assigning class labels as in (Sutherland et al., 2003; Mahé et al., 2006). We generated complete graphs from these compounds, where edges are labeled with distances as described in Section 4.7.2 and vertex labels correspond to atom types without encoding charges. The molecular distance graphs without hydrogen atoms have the characteristics shown in Table 4.5.

Mahé et al. (2006) used a Gaussian RBF kernel to compare distances according to Equation (4.14). Since this kernel is globally supported, i.e., $K_{\text{RBF}}^{\text{dist}}(x, y) > 0$ for all $x, y \in \mathbb{R}$, using this kernel gives rise to a dense product graph with our approach. Hence, we also used the compactly supported triangular kernel according to Equation (4.15) as replacement, which results in a sparse product graph. This kernel is also implemented in ChemCPP, but the computation of the pharmacophore kernel does not fully exploit the arising sparsity. The parameter optimization reported in the original publication suggests using $\sigma = 0.1$ for the Gaussian RBF kernel, where distances are expressed in ångström (Å). For the triangular kernel we selected $c = 0.25$, which promises to yield comparable results.

[10]Supporting information accompanying the article (Sutherland et al., 2003) available at `http://pubs.acs.org/doi/suppl/10.1021/ci034143r`.

We have computed the kernel matrix for all data sets using the pharmacophore kernel (PH) and our algorithm (SM) applying the Gaussian RBF kernel and the triangular kernel, respectively, with the above-mentioned parameters. The results of the experimental comparison are summarized in Tables 4.2 and 4.3. The improvement w.r.t. running time of the subgraph matching kernel compared to the pharmacophore kernel can be explained by two reasons: First the subgraph matching kernel is based on bijections instead of comparing triples of vertices, which reduces the combinatorial explosion. This effect can be expected to be even more crucial when using more than 3 vertices, which is not supported by the pharmacophore kernel, but is considered a worthwhile extension (Mahé et al., 2006). The second reason is that our algorithm exploits the sparsity of the product graph. This particularly explains the difference in running time when using the triangular kernel instead of the Gaussian RBF edge kernel. Regarding classification accuracy, there is, of course, no noticeable difference between the pharmacophore kernel and the subgraph matching kernel using the same kernel to compare distances. Marginal discrepancies can be explained by numerical inaccuracies. Using a triangular kernel instead of the Gaussian RBF kernel has only minor effects on the classification accuracy, which has even improved for three of the four data sets. At the same time using the triangular kernel drastically reduced the running time for computing the subgraph matching kernel.

Our approach clearly beats the pharmacophore kernel regarding running time when using the triangular kernel. This allows to apply our approach to larger data sets. Furthermore, our technique is not limited to three vertices and we consider larger subgraphs in the experimental comparison presented in Section 4.8

4.8 Comparative Experimental Evaluation

In Sections 4.6.6 and 4.7.4 we have compared different algorithms that compute conceptually equivalent kernels. Consequently, we focused on their difference in terms of running time, in particular, w.r.t. implicit and explicit computation. In this section we compare the performance of the proposed kernels to a wide range of state-of-the-art graph kernels regarding prediction accuracy as well as running time. We report classification results on widely-used benchmark data sets containing graphs with simple labels as well as attributed graphs derived from real-world application in chem- and bioinformatics.

4.8.1 Method and Data Sets

We performed classification experiments using the C-SVM implementation LIBSVM (Chang and Lin, 2011). We report mean prediction accuracies as well as standard deviations obtained by 10-fold cross-validation repeated 10 times with random fold assignment. Within each fold the regularization parameter C was chosen from $\{10^{-3}, 10^{-2}, \dots, 10^{3}\}$ by cross-validation based on the training set.

We compared the fixed length ℓ-walk kernel (FLW, see Section 4.6) and the subgraph matching kernel (SM and CSM with connection constraint, see Section 4.7) to several state-of-the-art graph kernels summarized in Section 4.5. We implemented graph kernels based on tree patterns (TP) and adapted the approach to support attributed graphs. The definition by Ramon and Gärtner (2003) is vague regarding subtrees without children. We require subtrees to have at least one child, see (Mahé and Vert, 2009) for a detailed discussion of this issue. The kernels (C)SM, FLW and TP support attributed graphs without restrictions. Furthermore, we compare to the geometric random walk (GRW), shortest-path (SP), graphlet (GL, see Section 4.7.4 for details), GraphHopper (GH), Weisfeiler-Lehman subtree (WL) and Weisfeiler-Lehman shortest-path (WLSP) kernel. Our implementation of these kernels supports graphs with simple vertex and edge labels whenever technically possible. SP and GH do not take edge annotations into account. WLSP is similar to the Neighborhood Subgraph Pairwise Distance Kernel recently proposed for graphs with simple labels (Costa and Grave, 2010). Computation of GRW is based on matrix inversion and we used the standard library LAPACK[11] for this. Since the library is not optimized for sparse matrices derived from the (weighted) direct product graph, better running times could be obtained with specialized implementations as reported by Vishwanathan et al. (2010), which were not available for Java.

These graph kernels can be tuned by several parameters. The maximum size of (C)SM was chosen from $k \in \{1, 2, \dots, 7\}$ and a uniform weight function was used. FLW refers to the ℓ-walk kernel according to Definition 4.5 and was computed for walks of length $\ell \in \{0, 1, \dots, 7\}$. The running times reported here for FLW differ from those presented in the publication (Kriege and Mutzel, 2012), since the algorithm and the implementation has been optimized subsequently as described in Section 4.6. GRW takes walks of infinite length into account, where the weight of walks decreases with their length depending on the parameter γ, cf. Equation (4.28). This parameter γ for GRW was chosen from $\{10^{-5}, 10^{-4}, \dots, 10^{-2}\}$. For TP we used a uniform weight λ chosen from $\{10^{-5}, 10^{-4}, \dots, 10^{-2}\}$ with height $h \in \{1, 2, 3, 4\}$. The number of iterations of WL/WLSP was chosen from $h \in \{0, 1, \dots, 5\}$. These intervals were chosen on the basis of reported results in prior publications. All parameters were optimized by means of cross-validation on the training data sets only. For SP and WLSP we tested different kernels to compare path lengths, but found the Dirac kernel which requires paths to have exactly equal length to yield comparable results. This choice allows to reduce computational costs (see Borgwardt and Kriegel, 2005) and makes explicit computation possible;

[11] `http://www.netlib.org/lapack/`

when the vertex kernel is binary we can employ our walk-based kernel, see Section 4.6.5. Both speed-up techniques are exploited by our implementation. As remarked by Wale et al. (2008) and Costa and Grave (2010), kernels using features of different size are typically biased towards large features. Therefore, we also normalized kernel values separately for each feature size where applicable. Kernels that can be written as $K(G,H) = \sum_{i=1}^{d} K_i(G,H)$ with K_i the kernel for feature size i and d the maximum feature size, were in addition normalized according to $\tilde{K}(G,H) = \sum_{i=1}^{d} \hat{K}_i(G,H)$. We used this normalization technique for SM, CSM, WL and WLSP. The values $K_i(G,H)$ for each feature size i for (C)SM can easily be obtained by slightly modifying Algorithms 4.6 and 4.7, respectively, such that for each $i \in \{1,\dots,k\}$ a global variable is maintained, which stores the sum over weights of size i cliques.

Since the running times may depend on the selected parameters, we report the time required to compute the kernel matrix using parameters frequently selected by the parameter optimization process. For a fair comparison all kernels were implemented in Java with exception of the pharmacophore kernel (PH), for which we again used the implementation provided by the authors as in Section 4.7.4. Experiments were conducted using Sun Java JDK v1.6.0 on an Intel Xeon E5430 machine at 2.66GHz with 8GB of RAM using a single processor only.

Graphs with simple labels

We employed benchmark data sets containing molecules[12] and proteins: The MUTAG data set consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium. The PTC data set contains compounds labeled according to carcinogenicity on rodents divided into male mice (MM), male rats (MR), female mice (FM) and female rats (FR). Molecules can naturally be represented by graphs, where vertices represent atoms and edges represent chemical bonds, cf. Section 1.1. We have removed explicit hydrogen atoms and labeled vertices by atom type and edges by bond type (single, double, triple or aromatic). The properties of these data sets are summarized in Table 4.4.

We have obtained the data set ENZYME from Borgwardt et al. (2005), which is associated with the task of assigning 600 enzymes to one of the 6 EC top level classes, which reflect the chemical reaction an enzyme catalyzes. Vertices represent secondary structure elements (SSE) and are annotated by their type, i.e., helix, sheet or turn. Two vertices are connected by an edge if they are neighbors along the amino acid sequence or one of three nearest neighbors in space. Edges are annotated with their type, i.e., structural or sequential.

[12]Both data sets are widely used (see, e.g., Kashima et al., 2003) and can be obtained from `http://cdb.ics.uci.edu`.

Table 4.4: Properties of the graph data sets with simple labels.

	Mutag	MM	FM	MR	FR	Enzyme
$\lvert\mathcal{D}\rvert$	188	336	349	344	351	600
Class size	125/63	129/207	143/206	152/192	121/230	6×100
Max. $\lvert V\rvert$	28	64	64	64	64	126
Avg. $\lvert V\rvert$	17.93	13.97	14.11	14.29	14.56	32.63
Max. $\lvert E\rvert$	33	71	71	71	71	149
Avg. $\lvert E\rvert$	19.79	14.32	14.48	14.69	15.0	62.14
Max. deg.	4	4	4	4	4	9
Avg. deg.	2.208	2.05	2.053	2.057	2.061	3.808
$\lvert\mathcal{L}_V\rvert$	7	20	18	18	19	3
$\lvert\mathcal{L}_E\rvert$	4	4	4	4	4	2

Attributed graphs

Benchmark data sets containing attributed graphs yet are less wide-spread. The ENZYME data set originally contains a rich set of attributes (for details, see Borgwardt et al., 2005). When all attributes are taken into account, we observed that a simple kernel just comparing edges, i.e., FLW with $\ell = 1$, is sufficient for highest prediction accuracy. Hence, we decided not to consider all attributes to make the graph structure crucial for the classification task. We added a single continuous attribute representing the 3D length of the SSE in ångström to each vertex of the ENZYME data set with simple labels. The vertex kernel was defined as the product of the Dirac kernel on the type attribute and the Brownian bridge kernel with parameter $c = 3$ originally used on the length attribute, see (Borgwardt et al., 2005). The edge kernel remains the Dirac kernel on the type attribute.

Further classification problems were derived from the chemical compound data sets BZR, COX2, DHFR and ER we have already used in Section 4.7.4. Since PH actually is not a graph kernel, it was directly applied to the original representation; for the other kernels the molecular distance graphs were used. Since there is no need to compute shortest paths, SP was directly applied to these graphs. Thus, SP reduces to a kernel basically comparing edges and was realized by FLW with $\ell = 1$. The shortest-path computation of GH was based on the computed distances as well, where we discretized lengths to values in $\{0, \dots, 1000\}$ since the approach is unreliable for real-valued edge lengths. SM, FLW and TP support multi-dimensional edge attributes. For these kernels we enriched the edge attributes by the type of the chemical bond, where edges not corresponding to bonds obtained an additional distinct label. The employed edge kernel was defined as the product of the Dirac kernel on the bond type and the triangular kernel according to Equation (4.15) on the distances. For PH the same triangular kernel was used to compare

Table 4.5: Properties of the attributed graph data sets. For discrete attributes the number of different values is listed and for continuous attributes the minimum and maximum value, respectively. For multi-dimensional attributes each dimension is considered individually.

	Enzyme	BZR	COX2	DHFR	ER
$\lvert\mathcal{D}\rvert$	600	306	303	393	446
Class size	6×100	157/149	148/155	126/267	181/265
Max. $\lvert V\rvert$	126	33	36	39	43
Avg. $\lvert V\rvert$	32.63	21.3	26.28	23.87	21.33
Max. $\lvert E\rvert$	149	528	630	741	903
Avg. $\lvert E\rvert$	62.14	225.1	335.1	283.0	234.8
Max. deg.	9	32	35	38	42
Avg. deg.	3.808	21.13	25.51	23.72	22.02
Vertex attr.	$\begin{bmatrix}3\\ 0\text{-}149\end{bmatrix}$	8	7	7	10
Edge attr.	2	$\begin{bmatrix}5\\ 1.14\text{-}16.6\end{bmatrix}$	$\begin{bmatrix}5\\ 1.16\text{-}16.9\end{bmatrix}$	$\begin{bmatrix}5\\ 1.1\text{-}19.8\end{bmatrix}$	$\begin{bmatrix}5\\ 1.13\text{-}24.0\end{bmatrix}$

distances. In all cases the parameter c of the triangular kernel was chosen from $\{0.1, 0.25, 0.5, 1.0\}$ by cross-validation. Since the input graphs are complete, SM and CSM yield the same results and we used the implementation of SM, which in this case is more efficient. Note that in contrast to the experiments presented in Section 4.7.4, we do not restrict SM to subgraphs of size 3.

4.8.2 Results and Discussion

The classification accuracies and running times are summarized in Tables 4.6 and 4.7. In terms of classification accuracy on graphs with simple labels no general suggestion which kernel performs best can be derived. CSM performs best on FM, where walk-based kernels perform slightly worse. The kernels GRW and FLW in contrast reach high accuracy on MM, where CSM is less competitive. CSM and GL are closely related, where the subgraph size of CSM is determined by cross-validation and is fixed to 3 vertices for GL. This explains why CSM reaches a higher accuracy on most data sets. Surprisingly GL performs better than CSM on MUTAG. This might be explained by the influence of automorphisms or be caused by disadvantageous decisions taken in the process of cross-validation. CSM beats the accuracy of GL clearly on the multiclass classification problem ENZYME with simple labels and yields results comparable to WL and WLSP, while others perform worse. This observation also holds for ENZYME with attributes, where WL and WLSP can no longer be applied. All approaches benefit substantially from the additional vertex annotations, which indicates the importance of attributes, and CSM reaches the highest classification accuracy. On molecular distance graphs we observed that SM performs best in two of four cases and competitive on the

Table 4.6: Mean classification accuracy, standard deviation and running time in seconds on graphs with simple labels. (Parameters used when measuring running time: CSM $k = 5$, FLW $\ell = 5$, TP $h = 2$, WL/WLSP $h = 3$; * — computation by explicit mapping)

Method	Mutag		MM		FM		MR		FR		Enzyme	
CSM	85.4±1.2	30.5	63.3±1.7	29.2	**63.8**±1.0	30.9	58.1±1.6	34.8	65.5±1.4	36.6	60.4±1.6	98m
TP	86.7±0.9	10.0	66.1±1.1	14.0	60.5±1.3	14.9	56.5±2.2	16.0	**68.0**±0.6	17.2	42.7±1.6	38m
GRW	87.4±1.1	40.1	**66.4**±0.7	46.4	59.2±1.2	49.2	57.7±1.1	53.6	**67.9**±0.5	56.7	31.6±1.3	229m
FLW	87.1±0.7	1.83	**66.6**±0.5	2.52	57.1±1.4	2.79	56.9±1.4	2.82	65.8±0.8	3.02	37.9±1.9	78.1
FLW*		0.35		0.31		0.36		0.33		0.37		11.7
GH	86.1±1.8	4.35	62.8±1.3	7.16	62.0±1.4	7.41	55.4±1.6	8.89	64.7±1.1	8.98	33.8±1.6	215.4
SP*	83.3±1.4	0.19	61.0±2.1	0.16	60.4±1.7	0.17	56.2±2.7	0.17	**67.9**±1.5	0.18	39.4±0.5	1.01
GL*	**87.7**±1.0	0.06	61.2±1.1	0.04	58.4±1.1	0.04	56.5±1.8	0.03	65.6±0.9	0.04	38.3±1.1	0.46
WL*	86.7±1.4	0.04	64.8±1.2	0.08	61.1±0.8	0.08	58.5±1.7	0.08	67.2±0.7	0.09	56.4±0.9	0.54
WLSP*	85.4±1.2	0.94	**66.6**±1.9	1.16	60.4±1.3	1.32	**59.7**±1.6	1.27	65.7±1.3	1.40	**62.9**±1.0	25.7

Table 4.7: Mean classification accuracy, standard deviation and running time in seconds on attributed graphs. (Parameters used when measuring running time: (C)SM $k = 5$, FLW $\ell = 5$, TP $h = 3$; Triangular Kernel $c = 0.25$)

Method	Enzyme		BZR		COX2		DHFR		ER	
(C)SM	**69.8**±0.7	115.5	**79.4**±1.2	85.5	**74.4**±1.7	195.5	79.9±1.1	116.2	**82.0**±0.8	163.5
TP	55.5±0.6	69.8	—	>24h	—	>24h	—	>24h	—	>24h
PH	—		77.9±1.6	192m	**74.6**±1.5	364m	**80.8**±1.2	418m	81.4±0.6	18h
FLW	63.9±0.3	15.8	77.9±1.1	49.8	**74.4**±1.5	82.0	79.1±1.1	71.1	81.6±1.1	84.2
GH	61.3±1.0	216.8	71.9±1.1	8.9	66.4±1.3	12.9	67.8±0.2	14.3	72.7±1.1	13.0
SP	65.7±1.1	282.2	78.2±1.2	40.6	**74.5**±1.3	67.4	77.6±1.5	59.9	79.9±1.3	69.7

other data sets. However, the differences here are rather small. Selecting the threshold parameter c by cross-validation explains the slight improvement compared to the accuracies presented in Section 4.7.4. Mahé et al. (2006) suggested to extend the pharmacophore kernel to take more than 3 points into account. At least for the instances we have tested, we observed that this does not lead to a considerable increase in classification accuracy. On the contrary, even SP, which just compares 2 points, yields competitive results. Nevertheless, the extension might prove useful where more complex binding mechanism must be considered.

The results on running time for graphs with simple labels clearly show that computation schemes based on explicit mapping outperform other approaches. For FLW we measured the running time for both, explicit and implicit computation, and found the explicit algorithm to be more efficient for all data sets, cf. Table 4.4. WLSP is by far the slowest of the approaches based on explicit mapping. At first sight, it seems surprising that the observed running times are even higher than the sum of the running times obtained for SP and WL, since WLSP actually is obtained by a combination of these two kernels. However, for approaches based on explicit mapping the time required for taking dot products is very important and depends on the number of different features contained in the individual graphs, cf. Section 4.4. The number of different features increases drastically when combining the two approaches and this explains the observed running times. Kernels computed by implicit mapping are slower and CSM, TP and GRW lie in the same order of magnitude, whereas CSM is the slowest. Alone our implicit computation of FLW is competitive to the explicit mapping algorithms. For attributed graphs with continuous labels WL and WLSP cannot be applied. On these graphs SP cannot be computed by explicit mapping. This leads to a considerable increase in running time of SP on the ENZYME data set, where it is now the slowest of the five tested approaches, while the other kernels noticeably benefit from the sparsity introduced by the vertex kernel taking the length attribute into account. TP could not be employed to molecular distance graphs, since the running time to compute a kernel matrix here exceeded 24h even for the most restrictive edge kernel with $c = 0.1$. This can be explained by the fact that this class of graphs contains vertices with large sets of matching neighbors, all subsets of which are considered by TP. The running time of PH also is very high rendering the approach infeasible for large data sets. We observed that the running time of SM increased with the parameter c, which is as expected, since the product graph becomes more dense when the threshold parameter is raised. The implicit computation of FLW is shown to be more efficient than SM and is one of the most efficient algorithms for attributed graphs, where the fact that SP on molecular graphs is directly computed by FLW with $\ell = 1$ without computing shortest paths must be considered. We also compared CSM to SM and observed that CSM is considerably faster on sparse graphs, while reaching a comparable prediction accuracy. Shervashidze et al. (2009) noticed in the context of unlabeled graphlets that taking into account disconnected subgraphs increases the prediction accuracy. We cannot confirm this observation for subgraph matching kernels.

We discuss the results for GH in more detail since this kernel was particularly proposed for attributed graphs (following our publication of the subgraph matching kernel) and has already been compared to CSM by Feragen et al. (2013)[13] and Neumann et al. (2012b). We observed that GH provides only moderate accuracy results, in particular for the used attributed graph data sets. On molecular distance graphs the approach is the fastest, but obtains the worst accuracy results. This cannot be explained by the fact that GH does not take the bond types into account since SP and PH both provide a high prediction accuracy on these graphs without using this information. A possible explanation is that GH is not able to incorporate the graph structure adequately for the prediction task: Since GH is a weighted sum over vertex kernels, cf. Equation (4.33), it does not guarantee that any two or more vertex pairs that contribute to the total kernel value exhibit a similar distance. For the ENZYME data set GH again reaches only moderate accuracy results and is one of the slowest kernels. This is remarkable since Feragen et al. (2013) and Neumann et al. (2012b) report competitive results for GH on the ENZYME data set. The reason for this most likely is, that we only used a subset of the available attributes, while for the other experiments all attributes were considered. As stated earlier, when using all attributes the classification problem is greatly simplified and the graph structure is only marginally relevant. Our results suggest that CSM—when applicable—remains a viable alternative in cases where the graph structure and attributes must be taken into account simultaneously.

4.9 Summary and Future Work

We have given an algorithmic view on the computation of graph kernels and analyzed techniques based on implicit and explicit feature maps and their ability to cope with attributed graphs. In particular, a sufficient condition for R-convolution kernels to determine feature maps and derive explicit computation schemes has been given. We have reviewed previous work in this context, identified key advantages of graph kernels over traditional vector based approaches, and—as the main contribution of this chapter—proposed fixed length walk and subgraph matching kernels. For fixed length walk kernels implicit and explicit computation schemes have been developed. For subgraph matching kernels we have proposed an algorithm for implicit computation and identified its relation to kernels for labeled graphs based on subgraphs and explicit feature maps. Our implicit computation schemes fully support

[13]Please note that the Erratum contains updated experimental results.

attributed graphs, while explicit computation schemes are well-suited for labeled graphs and often more efficient for real-world instances. In extensive experimental comparisons, we verified theoretical and practical considerations and observed computational phase transitions for both kernels depending on the characteristics of graphs with simple labels. We believe that graph kernels provide a unique potential over traditional approaches, which yet has not been fully exploited. We discuss several directions of future work reaching from incremental extensions of the kernels we have proposed to general aspects in the development of graph kernels.

Walk kernels. Kernels based on (random) walks were among the first kernels for graphs and received considerable attention. A remarkable property of these kernels is that they consider an infinite number of substructures stemming from walks of unbounded length and yet allow polynomial-time computation by closed-form expressions. Thus, the computation of these kernels employs the kernel trick and necessarily is implicit. Our results for kernels of fixed length suggest that the benefit of considering an infinite number of walks is limited. We have obtained similar accuracy results with walks of a fixed length on several data sets. Moreover, for the walk lengths that yield best accuracy results on these data sets explicit computation often turned out to be faster. While it is known that computing graph kernels based on paths of unbounded length is NP-hard (Gärtner et al., 2003), kernels based on paths of fixed length clearly can be computed in polynomial time. Paths would completely avoid the problem of tottering, which is known to harm prediction accuracy, and reduce the number of features that must be considered, since paths are a subset of walks. While the running time required to compute feature vectors is likely to increase compared to walks, the time for kernel matrix computation is dominated by taking dot products, which crucially depends on the number of features, cf. Section 4.4.1. Indeed techniques based on paths of a fixed length are commonly used in cheminformatics and used to determine the similarity between molecular graphs (Daylight, 2008; Ralaivola et al., 2005). Our experimental evaluation provides new insights in the running time behavior of implicit and explicit computation of walk-based kernels and suggests that implicit computation is preferable when graphs exhibit a high label diversity. The most important advantage of implicit computation schemes is that they fully support attributed graphs. Developing efficient kernels based on paths of fixed length that support attributed graphs would be interesting future work. Our algorithm for implicit computation of walk-based kernels can serve as starting point for this.

Subgraph matching kernels. We have proposed a novel graph kernel, which takes subgraphs into account and supports attributed graphs without restriction. The experimental evaluation shows promising results for attributed graphs from chem- and bioinformatics. Thus, we believe subgraph matching kernels are a viable alternative to existing approaches for attributed graphs. Our approach already works well in practice for medium-sized graphs, large graphs when vertex and edge kernels are sparse, or when restricted to small or connected subgraphs. Improving the running time for large-scale data sets and large graphs remains future work.

It is often sufficient to consider rather small subgraphs and, hence, small cliques in the product graph. The pharmacophore kernel, for example, can be computed based on 3-cliques, i.e., triangles. Recently, algorithms for the enumeration of triangles were subject to an extensive experimental evaluation by Ortmann and Brandes (2014). Modifying these algorithms, such that the weights of triangles are computed in the course of the enumeration process, allows to apply them to weighted direct product graphs with the goal to compute the subgraph matching kernel for $k = 3$ faster.

Computing the CISI kernel can be done by counting cliques instead of enumerating them. For just counting the number of cliques, algorithms with a running time superior to the running time achieved by our algorithm are known: For $k = 1$ and $k = 2$ the number of k-cliques corresponds to the number of vertices and edges, respectively. The number of 3-cliques is $\frac{1}{6}\operatorname{tr}(\mathbf{A}^3)$, where A is the adjacency matrix of $G\nabla H$ and $\operatorname{tr}(\cdot)$ is the trace of a matrix, i.e., essentially the number of walks of length 3 with the same start and end vertex. Since matrix multiplication of an $n \times n$-matrix can be computed in $\mathcal{O}(n^{\omega})$ with $\omega < 2.376$ (Coppersmith and Winograd, 1987), the number of 3-cliques can be determined in $\mathcal{O}(n^{\omega})$. Cliques of size $k = 4$ can be counted in $\mathcal{O}(n^{\omega+1})$ and general k-cliques in $\mathcal{O}(n^{\omega\lfloor k/3\rfloor+(k \mod 3)})$ (Nešetřil and Poljak, 1985; Alon et al., 1997). Can the sum of weighted cliques be computed in a similar manner? Such a modification could be useful in order to compute the subgraph matching kernel.

The technique described in Section 3.3.1 based on line graphs allows to solve MCES using an MCIS algorithm and is frequently combined with a clique detection algorithm for the corresponding association graph. The same ideas could be used to develop a variant of the subgraph matching kernel, that considers edge-induced common subgraphs. For MCES of molecular graphs the approach was shown to be able to improve the running time (Nicholson et al., 1987) and this might as well be the case when applied to the subgraph matching kernel. However, a key obstacle might be the presence of ΔY-exchanges causing cliques in the association graph that correspond to subgraphs with a different structure.

Kernels have attractive properties that allow their combination. The strength of the subgraph matching kernel when comparing distance graphs was shown in Section 4.7.4 for molecular graphs. The following approach can be considered an extension of the shortest-path kernel: We could first compute complete graphs, where edges are annotated with shortest-path distances and then apply the subgraph matching kernel to these graphs. This kernel would not only consider vertex pairs with similar distances as the shortest-path kernel, but could be used to take k-tuples with pairwise similar distances into account. The resulting kernel could as well be combined with vertex label refinement (see, e.g., Mahé et al., 2004; Rogers and Hahn, 2010; Costa and Grave, 2010; Shervashidze et al., 2011) based on the original graphs to obtain an extension of the NSDPK or Weisfeiler-Lehman shortest-path kernel. According to our analysis the running time of our approach is expected to decrease with the number of refinement steps. The goal of such a kernel is to take the relative position of vertices with the same environment into account.

Implicit and explicit computation. Graph kernels are typically proposed with one method of computation. We have shown that implicit and explicit computation schemes of the same kernel may behave complementary with respect to running time. Thus, for future work, one should develop explicit computation schemes for other implicit graph kernels and vice versa. While explicit computation schemes indeed often show superior running times in practice, they are highly similar to traditional approaches transforming graphs into vectors in order to apply machine learning tasks. A striking advantage of graph kernels is that they allow to compare attributes again by kernel functions. This is yet only possible with implicit mapping schemes. An open question is the following: If we assume that vertex and edge kernels can be computed (or approximated) by explicit mapping into a feature space, can we then obtain an explicit computation scheme of a (non-trivial) graph kernel? How can we define the feature space of such a graph kernel and in which way does its dimension depend on the dimension of the feature space of the vertex and edge kernels? This is a promising direction of future research with the goal to overcome the problem of discontinuity on bin-boundaries discussed in Section 4.3.1 while preserving the efficiency of explicit computation.

Complete graph kernels. We feel that this fundamental concept deserves more attention in order to systematically study the discriminative power of graph kernels, both, in theory and practice. None of the graph kernels typically used in practice is complete. This actually is inherent to the concept underlying almost all graph kernels, which are based on the decomposition of graphs into substructures. The initial quote of this chapter suggests that—when transferred to graphs—considering the parts, i.e., substructures, is not sufficient to determine the whole graph. The famous *graph reconstruction conjecture*, states that graphs are uniquely determined by *all* their proper subgraphs. Thus, even for kernels based on all proper subgraphs, completeness is not yet proven. The converse is easy to show for the subgraph sizes typically considered by graph kernels as the graphlet kernel (Shervashidze et al., 2011). We have obtained graph distance metrics by a fundamentally different principle: The maximum common subgraph algorithms of Chapter 3 do not consider parts separately and can be used to derive polynomial-time computable distance metrics for restricted graph classes. However, as detailed in Section 4.2.3 it is not straightforward to obtain valid kernels from these approaches and this remains promising future work.

One reason why the concept of complete graph kernels has been neglected so far, might be that kernels are meant to compute valid similarity measures and there is typically no need to distinguish all possible graphs up to isomorphism in order to obtain high prediction accuracies. However, it is absolutely necessary to distinguish non-isomorphic graphs with different class labels. Thus, the following notions might proof useful: For a given data set $\mathcal{D} = \{(G_1, y_1), \ldots, (G_n, y_n)\}$ of graphs $G_i \in \mathcal{G}$ with class labels $y_i \in \mathcal{Y}$ for all $1 \leq i \leq n$, we say a graph kernel K is *complete for $\mathcal{D}$* if for all graphs G_i, G_j the implication $\phi(G_i) = \phi(G_j) \implies y_i = y_j$ holds. Counting the number of graph pairs in a data set with different class labels that cannot be distinguished by a graph kernel then yields a value that limits the classification accuracy of a kernel on a specific data set. It would be interesting to investigate how this measures aligns with the observed prediction accuracy. If a graph kernel can distinguish all graphs (with different class labels) of a data set, but at the same time provides poor classification accuracies, then the classes are not separable in feature space by the used SVM. But does this necessarily mean that the graph kernel is not adequate? It is an interesting question if we can apply the same techniques that are commonly used for vector data to obtain linear separability in feature space in the context of graph kernels. We discuss this in the following paragraph.

Authoritative experimental comparison. The number of graph kernels proposed grows rapidly, while obtaining fair authoritative experimental comparisons becomes increasingly difficult. As we have seen, many graph kernels explicitly compute feature vectors and thus basically transform graph data to vector data. Typically these kernels then just apply the linear kernel to these vectors ob obtain a graph kernel. This is surprising since it is well-known that for vector data often better results can be obtained by a polynomial or Gaussian RBF kernel. These, however, are not used in combination with graph kernels. By applying the kernel trick such kernels can as well be used with graph kernels based on implicit computation. The Gaussian RBF kernel could, for example, be combined with the kernel metric obtained from the subgraph matching kernel to obtain a modified graph kernel. Experimental comparisons—including the one presented here—typically do not consider such modification although this would be most interesting.

There are yet several benchmark data sets containing graphs with simple labels, unfortunately data sets with attributed graphs are yet not wide-spread. This might be explained by the fact that only more recently graph kernels for attributed graphs obtained considerable attention. Since then, several kernels supporting attributed graphs have been proposed. These now allow to develop novel, more complex graph models with non-discrete annotations. Adding attributes to vertices and edges greatly enhances the freedom in modeling domain-specific problems and promises improved prediction accuracies when taken into account by kernels adequately. The advantages of graph kernels for attributed graphs compared to approaches for graphs with simple labels or based on transformation into vector data should be analyzed in more detail. In particular, the transformation of attributes to discrete labels by binning techniques as detailed in Section 4.3.2 should be further investigated. Experimental comparison could give evidence when kernels for attributed graphs can be replaced by kernels for simple labels, which are often more efficient.

Large-scale comparative evaluations on a wide range of data sets with the various graph kernels yet proposed would be highly desirable and—with the growing number of available graph kernels—become both, increasingly important and difficult to accomplish. Graph kernels often depend on parameters like the walk length and subgraph size. Authoritative experiments must consequently perform the best possible parameter optimization for all kernels and should combine each with different normalization techniques. Experimental evaluation would benefit from a common software framework for graph kernels, since implementations typically differ with respect to the graph file format, their ability to handle edge and vertex labels or attributes as well as basic graph data structures. Moreover, many recently proposed graph kernels rely on a base graph kernel or are obtained by combining standard modifications to graphs like label refinement. Many graph problems reappear in the context of graph kernels: This includes the enumeration of subgraphs, subtrees, cycles, paths and walks as well as the related canonization problems; computing graph invariants, (Weisfeiler-Lehman) label refinements and shortest paths is often required. A common framework would highly facilitate the combination of kernels, their fair comparison and the prototyping of new kernels. As part of the experimental evaluation presented in this thesis, not only kernels proposed by us have been implemented, but various kernels of other authors have been reimplemented. Releasing this software package in the near future is planned.

Das Vergleichen ist das Ende des Glücks
und der Anfang der Unzufriedenheit.

Søren Aabye Kierkegaard
(1813–1855)

Chapter 5

Conclusion and Outlook

Comparing graphs and developing meaningful similarity measures between graphs is of utmost importance and a premise in order to apply data mining algorithms to graphs. Consequently such techniques have been studied extensively. In this thesis, we considered two different techniques.

An essential advantage of solving maximum common subgraph problems is that the result can be inspected visually. This is crucial, for example, in cheminformatics, where visual analytics techniques become increasingly popular. In this application domain graphs often have a simple structure. Nevertheless, in practice almost exclusively algorithms with exponential worst-case running time are employed. Recently, progress in polynomial-time computable maximum common subgraph problems in molecular graphs has been made. Working around the limits established by known complexity results, rings and chains—or blocks and bridges in the language of graphs—were considered separately. This does not only allow to obtain polynomial running time for certain graph classes, but also yields chemical meaningful results. This has been shown experimentally and the idea is as well supported by the success of scaffold based approaches in cheminformatics. Building on these ideas we have extended the graph class for which polynomial-time algorithms were known from the outerplanar graphs to the series-parallel graphs. This requires to overcome the limitations of graph decompositions that are based on graph separators. We have introduced the concept of potential separators and used them algorithmically to find a maximum common biconnected subgraph of two biconnected series-parallel graphs. This result finally allowed us to compute block and bridge preserving maximum common connected subgraphs in series-parallel graphs. This progress constitutes a step towards solving maximum common subgraph problems that are highly relevant in practice in worst-case polynomial time. Further progress in this direction is required and might ultimately change the practical use of maximum common subgraph algorithms in cheminformatics and other application domains, where structurally simple graphs arise.

Another contribution of this thesis is to the field of graph kernels, a more novel development in graph comparisons. This technique generalizes traditional techniques based on vector representations. A unique advantage of graph kernels is that a large, possibly infinite, number of features can be considered efficiently and attributed graphs with continuous annotations can be compared. We have developed novel kernels especially designed for attributed graphs and studied under which conditions graph kernels can be computed by explicit vector representations. This allowed us to systematically study the impact of implicit and explicit computation of graph kernels on the running time. Our results suggest that explicit computation often is favorable, while implicit computation schemes provide a higher flexibility when comparing attributes. The advantage of annotating vertices and edges by attributes is not yet fully exploited by widely used graph models. The development of more sophisticated models with complex annotations requires domain specific expert knowledge and is promising future work.

If I have seen further it is by
standing on the shoulders of giants.

ISAAC NEWTON
(1643–1727)

Bibliography

Faisal N. Abu-Khzam. Maximum common induced subgraph parameterized by vertex cover. *Information Processing Letters*, 114(3):99 – 103, 2014. ISSN 0020-0190.

Julien Ah-Pine. Normalized kernels as similarity indices. In Mohammed J. Zaki, Jeffrey Xu Yu, B. Ravindran, and Vikram Pudi, editors, *Advances in Knowledge Discovery and Data Mining*, volume 6119 of *Lecture Notes in Computer Science*, pages 362–373. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13671-9.

Tatsuya Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76-A (9), September 1993.

Tatsuya Akutsu and Takeyuki Tamura. On the complexity of the maximum common subgraph problem for partial k-trees of bounded degree. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation*, volume 7676 of *Lecture Notes in Computer Science*, pages 146–155. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35260-7.

Tatsuya Akutsu and Takeyuki Tamura. A polynomial-time algorithm for computing the maximum common connected edge subgraph of outerplanar graphs of bounded degree. *Algorithms*, 6(1):119–135, 2013. ISSN 1999-4893.

Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997. ISSN 0178-4617.

Vikraman Arvind and Jacobo Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.

László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 171–183, New York, NY, USA, 1983. ACM. ISBN 0-89791-099-0.

László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 162–171, Nov 1983.

Sabine Bachl, Franz-Josef Brandenburg, and Daniel Gmach. Computing and drawing isomorphic subgraphs. *J. Graph Algorithms Appl.*, 8(2):215–238, 2004.

Laura Bahiense, Gordana Manić, Breno Piva, and Cid C. de Souza. The maximum common edge subgraph problem: A polyhedral investigation. *Discrete Applied Mathematics*, 160(18):2523 – 2541, 2012. ISSN 0166-218X. V Latin American Algorithms, Graphs, and Optimization Symposium — Gramado, Brazil, 2009.

Lu Bai, Horst Bunke, and Edwin R. Hancock. An attributed graph kernel from the jensen-shannon divergence. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 88–93, Aug 2014.

Jürgen Bajorath, editor. *Chemoinformatics: Concepts, Methods, and Tools for Drug Discovery*, volume 275 of *Methods in Molecular Biology.* Humana Press, Totowa, New Jersey, 2004. ISBN 1-58829-261-4.

Jürgen Bajorath, editor. *Chemoinformatics and Computational Chemical Biology*, volume 672 of *Methods in Molecular Biology.* Humana Press, 2011.

Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, 1986. ISSN 0097-5397.

John M. Barnard. Substructure searching methods: Old and new. *Journal of Chemical Information and Computer Sciences*, 33(4):532–538, 1993.

Lukas Barth, Stephen G. Kobourov, and Sergey Pupyrev. Experimental comparison of semantic word clouds. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, volume 8504 of *Lecture Notes in Computer Science*, pages 247–258. Springer International Publishing, 2014. ISBN 978-3-319-07958-5.

Enrico Bertini, Hendrik Strobelt, Joachim Braun, Oliver Deussen, Ulrich Groth, Thomas U. Mayer, and Dorit Merhof. HiTSEE: A visualization tool for hit selection and analysis in high-throughput screening experiments. In *Biological Data Visualization (BioVis), 2011 IEEE Symposium on*, pages 95–102, 2011.

Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph Theory 1736-1936.* Clarendon Press, New York, NY, USA, 1986. ISBN 0-198-53916-9.

Kristian Birchall and Valerie J Gillet. Reduced graphs and their applications in chemoinformatics. *Methods Mol Biol*, 672:197–212, 2011.

Hans L. Bodlaender. Classes of graphs with bounded treewidth. Technical Report RUU-CS-86-22, Department of Computer Science, Utrecht University, 1986.

Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990.

Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume A, pages 1–74. Kluwer Academic Publishers, 1999.

Robin S. Bon and Herbert Waldmann. Bioactivity-guided navigation of chemical space. *Acc Chem Res*, 43(8):1103–1114, Aug 2010.

Kellogg S. Booth and Charles J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, University of Waterloo, Computer Science Department, June 1979.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 74–81, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2278-5.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 Suppl 1:i47–i56, Jun 2005.

Karsten M. Borgwardt, Hans-Peter Kriegel, S. V. N. Vishwanathan, and Nicol N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. *Pacific Symposium on Biocomputing*, pages 4–15, 2007.

Franz J. Brandenburg. Subgraph isomorphism problems for k-connected partial k-trees. Unpublished Manuscript, 2000.

Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. ISBN 0-89871-432-X.

Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16:575–577, September 1973. ISSN 0001-0782.

Frank K. Brown. Chapter 35 - chemoinformatics: What is it and how does it impact drug discovery. volume 33 of *Annual Reports in Medicinal Chemistry*, pages 375 – 384. Academic Press, 1998.

Nathan Brown. Chemoinformatics - an introduction for computer scientists. *ACM Comput. Surv.*, 41(2), 2009.

Barry A. Bunin, Brian Siesel, Guillermo A. Morales, and Jürgen Bajorath, editors. *Chemoinformatics: Theory, Practice, & Products*. Springer Verlag, 2007.

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259, 1998. ISSN 0167-8655.

Johannes Carmesin, Reinhard Diestel, Fabian Hundertmark, and Maya Stein. Connectivity and tree structure in finite graphs. *Combinatorica*, 34(1):11–46, 2014. ISSN 0209-9683.

Frédéric Cazals and Chinmay Karande. An algorithm for reporting maximal c-cliques. *Theor. Comput. Sci.*, 349(3):484–490, 2005.

Alessio Ceroni, Fabrizio Costa, and Paolo Frasconi. Classification of small molecules by two- and three-dimensional decomposition kernels. *Bioinformatics*, 23(16):2038–2045, Aug 2007.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2: 27:1–27:27, May 2011. ISSN 2157-6904. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Markus Chimani and Petr Hliněný. A tighter insertion-based approximation of the crossing number. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *Lecture Notes in Computer Science*, pages 122–134. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22005-0.

Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The Open Graph Drawing Framework (OGDF). In R. Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 17, pages 543–569. CRC Press, 2013.

Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1):99–143, 2007.

Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7.

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, Francesco Tortorella, and Mario Vento. Graph matching: a fast algorithm and its evaluation. In *In Proc. of the 14th International Conference on Pattern Recognition*, pages 1582–1584, 1999.

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. An improved algorithm for matching large graphs. In *3rd IAPR-TC15 Workshop on Graph-based Representations*, 2001.

Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004. ISSN 0162-8828.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, September 2001. ISBN 0262531968.

Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 255–262, 2010.

Andrew Cotter, Joseph Keshet, and Nathan Srebro. Explicit approximations of the gaussian kernel. *CoRR*, abs/1109.4603, 2011.

Chemical Information Systems Daylight. Daylight theory manual v4.9. http://www.daylight.com/dayhtml/doc/theory, January 2008.

Luc De Raedt and Jan Ramon. Deriving distance metrics from generality relations. *Pattern Recogn. Lett.*, 30(3):187–191, February 2009. ISSN 0167-8655.

Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Knowledge and Data Engineering, IEEE Transactions on*, 17(8): 1036 – 1050, aug. 2005. ISSN 1041-4347.

Anders Dessmark, Andrzej Lingas, and Andrzej Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. *Algorithmica*, 27:337–347, 2000. ISSN 0178-4617.

Reinhard Diestel. *Graph Theory.* Springer-Verlag, Heidelberg, 3 edition, 2005.

Frederic Dorn. Planar subgraph isomorphism revisited. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 263–274, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-16-3.

Geoff M. Downs and John M. Barnard. *Clustering Methods and Their Uses in Computational Chemistry*, pages 1–40. John Wiley & Sons, Inc., 2003. ISBN 9780471433514.

Andre Droschinsky, Bernhard Heinemann, Nils Kriege, and Petra Mutzel. Enumeration of maximum common subtree isomorphisms with polynomial-delay. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation*, Lecture Notes in Computer Science, pages 81–93. Springer International Publishing, 2014. ISBN 978-3-319-13074-3.

Andre Droschinsky, Nils Kriege, and Petra Mutzel. Efficient enumeration algorithms for common subgraph problems in outerplanar graphs. 2015. In preparation.

Ran Duan and Hsin-Hao 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 '12, pages 1413–1424. SIAM, 2012.

Paul J. Durand, Rohit Pasari, Johnnie W. Baker, and Chun-che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2:1–12, 1999.

Joseph L. Durant, Burton A. Leland, Douglas R. Henry, and James G. Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, 42(5):1273–1280, 2002.

Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011. ISSN 1759-0884.

Ehab S. El-Mallah and Charles J. Colbourn. The complexity of some edge deletion problems. *Circuits and Systems, IEEE Transactions on*, 35(3): 354–362, Mar 1988. ISSN 0098-4094.

Charles Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 147–153, 2003.

David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms & Applications*, 3(3):1–27, 1999.

Peter Ertl and Bernhard Rohde. The molecule cloud - compact visualization of large collections of molecules. *Journal of Cheminformatics*, 4(1):12, 2012. ISSN 1758-2946.

Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen D. Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In C.j.c. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 216–224, 2013. Erratum available at `http://image.diku.dk/aasa/papers/graphkernels_nips_erratum.pdf`.

Mirtha-Lina Fernández and Gabriel Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6–7):753 – 758, 2001. ISSN 0167-8655.

Thomas Fober, Marco Mernberger, Gerhard Klebe, and Eyke Hüllermeier. Fingerprint kernels for protein structure comparison. *Molecular Informatics*, 31(6-7):443–452, 2012. ISSN 1868-1751.

Pasquale Foggia, Carlo Sansone, and Mario Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, pages 176–187. Citeseer, 2001.

Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 18–25, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. ISBN 0-89871-605-5.

Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

Holger Fröhlich, Jörg K. Wegner, Florian Sieker, and Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 225–232, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979. ISBN 0-7167-1044-7.

Thomas Gärtner. *Kernels for structured data.* Series in machine perception and artificial intelligence. World Scientific, Singapore, 2009.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In Bernhard Schölkopf and Manfred Warmuth, editors, *Learning Theory and Kernel Machines*, volume 2777 of *Lecture Notes in Computer Science*, pages 129–143. Springer Berlin / Heidelberg, 2003.

Johann Gasteiger and Thomas Engel, editors. *Chemoinformatics: A Textbook.* Wiley VCH, October 2003. ISBN 3527306811.

Anna Gaulton, Louisa J. Bellis, A. Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, and John P. Overington. Chembl: a large-scale bioactivity database for drug discovery. *Nucleic Acids Research*, 40 (D1):D1100–D1107, 2012.

Marc G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, March 2002. ISSN 1532-4435.

Adalbert Gorecki, Anke Arndt, Arbia Ben Ahmed, Andre Wiesniewski, Cengizhan Yücel, Gebhard Schrader, Henning Wagner, Michael Rex, Nils Kriege, Philipp Büderbender, Sergej Rakov, and Vanessa Bembenek. *ChemBioSpace Explorer: PG504 Endbericht*, 2008.

Susumu Goto, Yasushi Okuno, Masahiro Hattori, Takaaki Nishioka, and Minoru Kanehisa. Ligand: database of chemical compounds and reactions in biological pathways. *Nucleic Acids Research*, 30(1):402–404, 2002.

Martin Grohe. Logical and structural approaches to the graph isomorphism problem. In Krishnendu Chatterjee and Jirí Sgall, editors, *Mathematical Foundations of Computer Science 2013*, volume 8087 of *Lecture Notes in Computer Science*, pages 42–42. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40312-5. Slides available at: `https://ist.ac.at/mfcs13/slides/gi.pdf`.

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

Martin Gronemann, Michael Jünger, Petra Mutzel, and Nils Kriege. Molmap – visualizing molecule libraries as topographic maps. In *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications (GRAPP & IVAPP)*, 2013.

Arvind Gupta and Naomi Nishimura. Sequential and parallel algorithms for embedding problems on classes of partial k-trees. In Erik Schmidt and Sven Skyum, editors, *Algorithm Theory — SWAT '94*, volume 824 of *Lecture Notes in Computer Science*, pages 172–182. Springer Berlin / Heidelberg, 1994. ISBN 978-3-540-58218-2.

Arvind Gupta and Naomi Nishimura. Characterizing the complexity of subgraph isomorphism for graphs of bounded path-width. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96*, volume 1046 of *Lecture Notes in Computer Science*, pages 453–464. Springer Berlin / Heidelberg, 1996a. ISBN 978-3-540-60922-3.

Arvind Gupta and Naomi Nishimura. The complexity of subgraph isomorphism for classes of partial k-trees. *Theoretical Computer Science*, 164 (1–2):287 – 298, 1996b. ISSN 0304-3975.

Arvind Gupta and Naomi Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21:183–210, 1998. ISSN 0178-4617.

Yuri Gurevich. From invariants to canonization. *Bulletin of the EATCS*, 63, 1997.

Martin Gutlein, Andreas Karwath, and Stefan Kramer. Ches-mapper - chemical space mapping and visualization in 3d. *Journal of Cheminformatics*, 4 (1):7, 2012. ISSN 1758-2946.

Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In Joe Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41554-1.

Bernard Haasdonk and Claus Bahlmann. Learning with distance substitution kernels. In Carl Edward Rasmussen, Heinrich H. Bülthoff, Bernhard Schölkopf, and Martin A. Giese, editors, *Pattern Recognition*, volume 3175 of *Lecture Notes in Computer Science*, pages 220–227. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22945-2.

MohammadTaghi Hajiaghayi and Naomi Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *Journal of Computer and System Sciences*, 73(5):755 – 768, 2007. ISSN 0022-0000.

Richard Hammack, Wilfried Imrich, and Sandi Klavžar. *Handbook of Product Graphs.* Discrete Mathematics and Its Applications. Taylor and Francis, 2011. ISBN 9781439813041.

Zaïd Harchaoui and Francis Bach. Image classification with segmentation graph kernels. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, june 2007.

David Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California, Santa Cruz, CA, USA, 1999.

Shohei Hido and Hisashi Kashima. A linear-time graph kernel. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining*, pages 179 –188, dec. 2009.

Sepp Hochreiter and Klaus Obermayer. Support vector machines for dyadic data. *Neural Computation*, 18(6):1472–1510, 2006.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *Ann. Statist.*, 36(3):1171–1220, 06 2008.

Walter Holberg. The decomposition of graphs into k-connected components. *Discrete Mathematics*, 109(1-3):133 – 145, 1992. ISSN 0012-365X.

John E. Hopcroft and Richard M. Karp. An $\mathrm{n}^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

John E. Hopcroft and Robert E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

John E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. ACM.

Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 158–167, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1.

Tamás Horváth, Jan Ramon, and Stefan Wrobel. Frequent subgraph mining in outerplanar graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 197–206, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

Tamás Horváth, Jan Ramon, and Stefan Wrobel. Frequent subgraph mining in outerplanar graphs. *Data Mining and Knowledge Discovery*, 21:472–508, 2010. ISSN 1384-5810.

Tamás Horváth and Jan Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411 (31–33):2784 – 2797, 2010. ISSN 0304-3975.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003. version updated 2010.

Brijnesh J. Jain and Klaus Obermayer. Elkan's k-means algorithm for graphs. In Grigori Sidorov, Arturo Hernández Aguirre, and Carlos Alberto Reyes García, editors, *Advances in Soft Computing*, volume 6438 of *Lecture Notes in Computer Science*, pages 22–32. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-16772-0.

Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on*, pages 284–291, Oct 1991.

David S. Johnson. The NP-completeness column. *ACM Trans. Algorithms*, 1(1):160–176, July 2005. ISSN 1549-6325.

David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119 – 123, 1988. ISSN 0020-0190.

H. C. Johnston. Cliques of a graph-variations on the bron-kerbosch algorithm. *International Journal of Parallel Programming*, 5:209–238, 1976. ISSN 0885-7458.

U Kang, Hanghang Tong, and Jimeng Sun. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 828–838, 2012.

Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212 – 233, 2001. ISSN 0196-6774.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pages 321–328, 2003.

Karsten Klein, Nils Kriege, and Petra Mutzel. CT-index: Fingerprint-based graph indexing combining cycles and trees. In *IEEE 27th International Conference on Data Engineering (ICDE)*, pages 1115 –1126, April 2011.

Karsten Klein, Nils Kriege, and Petra Mutzel. Scaffold Hunter – visual analysis of chemical compound databases. In *Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications (GRAPP & IVAPP)*, pages 626–635, 2012.

Karsten Klein, Oliver Koch, Nils Kriege, Petra Mutzel, and Till Schäfer. Visual analysis of biological activity data with Scaffold Hunter. *Molecular Informatics*, 32(11-12):964–975, 2013a. ISSN 1868-1751.

Karsten Klein, Nils Kriege, and Petra Mutzel. Scaffold Hunter: Facilitating drug discovery by visual analysis of chemical space. In Gabriela Csurka, Martin Kraus, Robert S. Laramee, Paul Richard, and José Braz, editors, *Computer Vision, Imaging and Computer Graphics. Theory and Application*, volume 359 of *Communications in Computer and Information Science*, pages 176–192. Springer Berlin Heidelberg, 2013b. ISBN 978-3-642-38240-6.

Ina Koch. Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci.*, 250(1-2):1–30, 2001.

Risi Imre Kondor and John D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 315–322, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-873-7.

Nils Kriege. Erweiterte Substruktursuche in Moleküldatenbanken und ihre Integration in Scaffold Hunter. Master's thesis, TU Dortmund, December 2009.

Nils Kriege and Petra Mutzel. Subgraph matching kernels for attributed graphs. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

Nils Kriege and Petra Mutzel. Finding maximum common biconnected subgraphs in series-parallel graphs. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014*, volume 8635 of *Lecture Notes in Computer Science*, pages 505–516. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44464-1.

Nils Kriege, Florian Kurpicz, and Petra Mutzel. On maximum common subgraph problems in series-parallel graphs. In Kratochvíl Jan, Mirka Miller, and Dalibor Froncek, editors, *International Workshop on Combinatorial Algorithms, IWOCA 2014*, volume 8986 of *Lecture Notes in Computer Science*, pages 200–212. Springer International Publishing, 2014a. ISBN 978-3-319-19314-4. Journal version submitted to the European Journal of Combinatorics.

Nils Kriege, Petra Mutzel, and Till Schäfer. Practical SAHN clustering for very large data sets and expensive distance metrics. *Journal of Graph Algorithms and Applications*, 18(4):577–602, 2014b.

Nils Kriege, Marion Neumann, Kristian Kersting, and Petra Mutzel. Explicit versus implicit graph feature maps: A computational phase transition for walk kernels. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 881–886, Dec 2014c.

Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.

Florian Kurpicz. Efficient algorithms for the maximum common subgraph problem in partial 2-trees. Master's thesis, TU Dortmund, 2014.

Johannes Köbler. On graph isomorphism for restricted graph classes. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and JohnV. Tucker, editors, *Logical Approaches to Computational Barriers*, volume 3988 of *Lecture Notes in Computer Science*, pages 241–256. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35466-6.

G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, Jan 1973.

Andrzej Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63(3):295 – 302, 1989. ISSN 0304-3975.

Andrzej Lingas and Maciej Sysło. A polynomial-time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In Timo Lepistö and Arto Salomaa, editors, *Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 394–409. Springer Berlin / Heidelberg, 1988. ISBN 978-3-540-19488-0.

Alan Lipkus. A proof of the triangle inequality for the tanimoto distance. *Journal of Mathematical Chemistry*, 26(1):263–265, October 1999.

Eugen Lounkine, Mathias Wawer, Anne Mai Wassermann, and Jürgen Bajorath. Saranea: a freely available program to mine structure-activity and structure-selectivity relationship information in compound data sets. *J. Chem. Inf. Model.*, 50(1):68–78, Jan 2010.

Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42 – 65, 1982. ISSN 0022-0000.

Gerald M. Maggiora and Veerabahu Shanmugasundaram. Molecular similarity measures. *Methods in Molecular Biology*, 672:39–100, 2011.

Pierre Mahé and Jean-Philippe Vert. Graph kernels based on tree patterns for molecules. *Mach. Learn.*, 75:3–35, April 2009. ISSN 0885-6125.

Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. Extensions of marginalized graph kernels. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 70–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5.

Pierre Mahé, Nobuhisa Ueda, Tatsuya Akutsu, Jean-Luc Perret, and Jean-Philippe Vert. Graph kernels for molecular structure-activity relationship analysis with support vector machines. *J. Chem. Inf. Model.*, 45(4):939–951, 2005.

Pierre Mahé, Liva Ralaivola, Véronique Stoven, and Jean-Philippe Vert. The pharmacophore kernel for virtual screening with support vector machines. *J. Chem. Inf. Model.*, 46(5):2003–2014, 2006.

Gordana Manić, Laura Bahiense, and Cid de Souza. A branch&cut algorithm for the maximum common edge subgraph problem. *Electronic Notes in Discrete Mathematics*, 35(0):47 – 52, 2009. ISSN 1571-0653. Proceedings of the Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS '09).

Jörn Marialke, Robert Körner, Simon Tietze, and Joannis Apostolakis. Graph-based molecular alignment (gma). *J. Chem. Inf. Model.*, 47(2):591–601, 2007.

Jiří Matoušek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.

David W. Matula. Subtree isomorphism in $O(n^{5/2})$. In P. Hell B. Alspach and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91 – 106. Elsevier, 1978.

Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014. ISSN 0747-7171.

Sauro Menchetti, Fabrizio Costa, and Paolo Frasconi. Weighted decomposition kernels. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 585–592, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Johannes Mohr, Brijnesh Jain, Andreas Sutter, Antonius Ter Laak, Thomas Steger-Hartmann, Nikolaus Heinrich, and Klaus Obermayer. A maximum common subgraph kernel method for predicting the chromosome aberration test. *J. Chem. Inf. Model.*, 50(10):1821–1838, Oct 2010.

Michel Neuhaus and Horst Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39(10):1852 – 1863, 2006. ISSN 0031-3203. Similarity-based Pattern Recognition.

Marion Neumann, Roman Garnett, Plinio Moreno, Novi Patricia, and Kristian Kersting. Propagation kernels for partially labeled graphs. In *ICML–2012 Workshop on Mining and Learning with Graphs (MLG–12)*, Edinburgh, UK, 2012a.

Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In *Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 378–393, 2012b. long version `arXiv:1410.3314 [stat.ML]`.

Marion Neumann, Roman Garnett, and Kristian Kersting. Coinciding walk kernels: Parallel absorbing random walks for learning with graphs and few labels. In Cheng Soon Ong and Tu Bao Ho, editors, *Proceedings of the 5th Annual Asian Conference on Machine Learning (ACML 2013)*, volume 29 of *JMLR Proceedings*, pages 357–372, 2013.

Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2): 415–419, 1985.

V. Nicholson, C.-C. Tsai, M. Johnson, and M. Naim. A subgraph isomorphism theorem for molecular graphs. In *Graph Theory and Topology in Chemistry*, number 51 in Stud. Phys. Theoret. Chem., pages 226–230. Elsevier, 1987.

Mark Ortmann and Ulrik Brandes. Triangle listing algorithms: Back from the diversion. In Catherine C. McGeoch and Ulrich Meyer, editors, *Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–8, 2014.

Panos M. Pardalos and Jue Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, April 1994.

Andreas Parra and Petra Scheffler. How to use the minimal separators of a graph for its chordal triangulation. In Zoltán Fülöp and Ferenc Gécseg, editors, *Automata, Languages and Programming*, volume 944 of *Lecture Notes in Computer Science*, pages 123–134. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-60084-8.

Jean-Luc Perret and Pierre Mahé. *ChemCpp User-Guide*, October 2006. Software ChemCPP v1.0.2 available at `http://chemcpp.sourceforge.net`.

Breno Piva and CidCarvalho de Souza. Polyhedral study of the maximum common induced subgraph problem. *Annals of Operations Research*, 199 (1):77–102, 2012. ISSN 0254-5330.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc., 2008.

Liva Ralaivola, Sanjay Joshua Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093 – 1110, 2005. ISSN 0893-6080. Neural Networks and Kernel Methods for Structured Domains.

Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences*, 2003.

John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002.

John W. Raymond, Eleanor J. Gardiner, and Peter Willett. Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316, 2002a.

John W. Raymond, Eleanor J. Gardiner, and Peter Willett. RASCAL: Calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 45(6):631–644, 2002b.

Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977. ISSN 1097-0118.

Steffen Renner, Willem A L. van Otterlo, Marta Dominguez Seoane, Sabine Möcklinghoff, Bettina Hofmann, Stefan Wetzel, Ansgar Schuffenhauer, Peter Ertl, Tudor I. Oprea, Dieter Steinhilber, Luc Brunsveld, Daniel Rauh, and Herbert Waldmann. Bioactivity-guided mapping and navigation of chemical space. *Nat Chem Biol*, 5(8):585–592, Aug 2009.

Steven W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM J. Comput.*, 6(4):730–732, 1977.

Kaspar Riesen and Horst Bunke. Dissimilarity based vector space embedding of graphs using prototype reduction schemes. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 5632 of *Lecture Notes in Computer Science*, pages 617–631. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03069-7.

John M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, Université de Bordeaux I, January 2001.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *J. Chem. Inf. Model.*, 50(5):742–754, May 2010.

Leander Schietgat. *Graph-Based Data Mining for Biological Applications.* PhD thesis, Informatics Section, Department of Computer Science, Faculty of Engineering, May 2010. Hendrik Blockeel and Maurice Bruynooghe (supervisors).

Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time metric for outerplanar graphs. In *Mining and Learning with Graphs, MLG 2007, Firence, Italy, August 1-3, 2007, Proceedings*, 2007.

Leander Schietgat, Jan Ramon, Maurice Bruynooghe, and Hendrik Blockeel. An efficiently computable graph-based metric for the classification of small molecules. In Jean-François Boulicaut, Michael Berthold, and Tamás Horváth, editors, *Discovery Science*, volume 5255 of *Lecture Notes in Computer Science*, pages 197–209. Springer Berlin / Heidelberg, 2008.

Leander Schietgat, Fabrizio Costa, Jan Ramon, and Luc De Raedt. Effective feature construction by maximum common subgraph sampling. *Machine Learning*, 83(2):137–161, 2011. ISSN 0885-6125.

Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics. *Annals of Mathematics and Artificial Intelligence*, 69(4):343–376, 2013. ISSN 1012-2443.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001. ISBN 0262194759.

Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Comput.*, 12(5):1207–1245, May 2000. ISSN 0899-7667.

Ansgar Schuffenhauer and Thibault Varin. Rule-based classification of chemical structures by scaffold. *Molecular Informatics*, 30(8):646–664, 2011. ISSN 1868-1751.

Ansgar Schuffenhauer, Peter Ertl, Silvio Roggo, Stefan Wetzel, Marcus A. Koch, and Herbert Waldmann. The scaffold tree - visualization of the scaffold universe by hierarchical scaffold classification. *J. Chem. Inf. Model.*, 47(1):47–58, January 2007.

Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267 – 280, 1999. ISSN 0196-6774.

Haichuan Shang, Xuemin Lin, Ying Zhang, Jeffrey Xu Yu, and Wei Wang. Connected substructure similarity search. In *SIGMOD Conference*, pages 903–914, 2010.

Nino Shervashidze and Karsten Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1660–1668, 2009.

Nino Shervashidze, S.V.N. Vishwanathan, Tobias H. Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Clearwater Beach, Fl, USA, April, 16-18, 2009, 2009. Society for Artificial Intelligence and Statistics.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, December 2009. ISSN 1532-4435.

Kilho Shin and Tetsuji Kuboyama. A generalization of Haussler's convolution kernel — mapping kernel and its application to tree kernels. *Journal of Computer Science and Technology*, 25:1040–1054, 2010. ISSN 1000-9000.

Martin Stahl, Harald Mauser, Mark Tsui, and Neil R. Taylor. A robust clustering method for chemical structures. *J Med Chem*, 48(13):4358–4366, Jun 2005.

Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 0387772413.

W. Henry Suters, Faisal N. Abu-Khzam, Yun Zhang, Christopher T. Symons, Nagiza F. Samatova, and Michael A. Langston. A new approach and faster exact methods for the maximum common subgraph problem. In Lusheng Wang, editor, *Computing and Combinatorics*, volume 3595 of *Lecture Notes in Computer Science*, pages 717–727. Springer Berlin / Heidelberg, 2005.

Jeffrey J. Sutherland, Lee A. O'Brien, and Donald F. Weaver. Spline-fitting with a genetic algorithm: a method for developing classification structure-activity relationships. *J Chem Inf Comput Sci*, 43(6):1906–1915, 2003.

Sanjay Joshua Swamidass, Jonathan Chen, Jocelyne Bruand, Peter Phung, Liva Ralaivola, and Pierre Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 21 Suppl 1:i359–i368, Jun 2005.

Maciej M. Sysło. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17(1):91 – 97, 1982. ISSN 0304-3975.

James J. Thomas and Kristin A. Cook, editors. *Illuminating the Path: The Research and Development Agenda for Visual Analytics.* August 2005. ISBN 0-7695-2323-4.

Christian Tonnelier, Philippe Jauffret, Thierry Hanser, and Gérard Kaufmann. Machine learning of generic reactions: 3. an efficient algorithm for maximal common substructure determination. *Tetrahedron Computer Methodology*, 3(6):351–358, 1990.

Julian R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1): 31–42, 1976. ISSN 0004-5411.

Julian R. Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *J. Exp. Algorithmics*, 15:1.6:1.1–1.6:1.64, February 2011. ISSN 1084-6654.

Vladimir N. Vapnik and Aleksandr Ya. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

Rakesh M. Verma and Steven W. Reyner. An analysis of a good algorithm for the subtree problem, corrected. *SIAM J. Comput.*, 18(5):906–908, 1989.

Jean-Philippe Vert. The optimal assignment kernel is not positive definite. *CoRR*, abs/0801.4061, 2008.

Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf. A primer on kernel methods. In Jean-Philippe Vert, Koji Tsuda, and Bernhard Schölkopf, editors, *Kernel Methods in Computational Biology*, pages 35–70. MIT Press, 2004.

S. V. N. Vishwanathan, Karsten M. Borgwardt, and Nicol N. Schraudolph. Fast computation of graph kernels. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada*, volume 19, pages 1449–1456, Cambridge, MA, 2006. MIT Press.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Imre Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

Philippe Vismara and Benoît Valery. Finding maximum common connected subgraphs using clique detection or constraint satisfaction algorithms. In HoaiAn Le Thi, Pascal Bouvry, and Tao Pham Dinh, editors, *Modelling, Computation and Optimization in Information Systems and Management*

Sciences, volume 14 of *Communications in Computer and Information Science*, pages 358–368. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-87476-8.

Klaus Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114(1):570–590, 1937. ISSN 0025-5831.

Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, 2008.

Walter D. Wallis, Peter Shoubridge, Miro Kraetzl, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6/7):701–704, 2001.

Stefan Wetzel, Karsten Klein, Steffen Renner, Daniel Rauh, Tudor I. Oprea, Petra Mutzel, and Herbert Waldmann. Interactive exploration of chemical space with scaffold hunter. *Nature Chemical Biology*, 5(8):581–583, August 2009.

Stefan Wetzel, Wolfram Wilk, Samy Chammaa, Bianca Sperl, Anke G Roth, Aybike Yektaoglu, Steffen Renner, Thorsten Berg, Christoph Arenz, Athanassios Giannis, Tudor I Oprea, Daniel Rauh, Markus Kaiser, and Herbert Waldmann. A scaffold-tree-merging strategy for prospective bioactivity annotation of γ-pyrones. *Angew Chem Int Ed Engl*, 49(21):3666–3670, May 2010.

Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932. ISSN 00029327.

Peter Willett. Matching of chemical and biological structures using subgraph and maximal common subgraph isomorphism algorithms. *The IMA Volumes in Mathematics and its Applications*, 108:11–38, 1999.

David R. Wood. An algorithm for finding a maximum clique in a graph. *Operations Research Letters*, 21(5):211–217, 1997. ISSN 0167-6377.

Atsuko Yamaguchi and Hiroshi Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. In Toshihide Ibaraki, Naoki Katoh, and Hirotaka Ono, editors, *Algorithms and Computation (ISAAC)*, volume 2906 of *Lecture Notes in Computer Science*, pages 58–67. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20695-8.

Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Graph complexity of chemical compounds in biological pathways. *Genome Informatics*, 14: 376–377, 2003.

Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Inf. Process. Lett.*, 92(2):57–63, 2004.

Stéphane Zampelli. *A constraint programming approach to subgraph isomorphism*. PhD thesis, Universite catholique de Louvain, 2008.

V.N. Zemlyachenko, N.M. Korneenko, and R.I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985. ISSN 0090-4104. Translated from Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR, Vol. 118, pp. 83–158, 1982.

Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer, 2006. ISBN 0-387-29146-6.

Hao Helen Zhang and Marc Genton. Compactly supported radial basis function kernels. Technical Report 2570, Institute of Statistics, NCSU, 2004.

Sheng-Xin Zhu. Compactly supported radial basis functions: How and why? Technical report, The Mathematical Institute, University of Oxford, 2012.