Utilizing Constrained Homomorphisms in the Design of Efficient Graph Kernels

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt
von

Till Hendrik Schulz

aus
Lingen (Ems)

Bonn, 2023

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Stefan Wrobel
2. Gutachter: Prof. Dr. Christian Bauckhage

Tag der Promotion: 13.12.2023
Erscheinungsjahr: 2024

Acknowledgments

While the cover page features only one name, this thesis is the product of the collective efforts of many. A particularly substantial share is attributed to my supervisors, Tamás Horváth, Pascal Welke, and Stefan Wrobel, to whom I am profoundly grateful. Their constant availability and willingness to engage in discussions, as well as their patience and provided freedom, are what made this thesis possible.

I am thankful to have been surrounded by so many great people of the MLAI research group, who not only contributed to the development of this thesis but, more importantly, offered worthwhile distractions thereof. My special thanks go to Fouad Alkhoury, Ewald Bindereif, Michael Kamp, Sebastian Müller, Florian Seiffarth, Patrick Seifner, Eike Stadtländer, Vanessa Toborek, and Daniel Trabold.

Finally, I would like to thank my parents Heidrun and Stefan, whose unwavering support has brought me this far.

Abstract

Learning on graphs, particularly graph classification, requires rich graph representations. A common paradigm to obtain these is by extracting sets of substructures and representing graphs by such sets. The obtained graph representations then enable the application of standard machine learning approaches like support vector machines. Traditionally, graph substructures refer to subgraph patterns which are embedded by subgraph isomorphisms. Identifying subgraph patterns is however computationally infeasible due to the NP-completeness of deciding subgraph isomorphism even when the patterns are restricted to trees. A relaxation of the problem is to consider graph homomorphisms as the pattern matching operator instead, which can in fact be computed in polynomial time for tree patterns. However, graph homomorphisms generally result in less suitable graph representations for classification tasks. A key observation, which has been largely disregarded in the machine learning community, is that subgraph isomorphisms can be regarded as constrained homomorphisms. In this dissertation, we utilize this unifying view of these two pattern embedding operators by considering tractable instances of constrained homomorphisms on tree patterns and design three powerful and efficiently computable graph kernels.

To bridge the gap between graph homomorphisms and subgraph isomorphisms, we first introduce the notion of partially injective homomorphisms which require injectivity only for subsets of the patterns' vertex pairs. Utilizing positive complexity results on deciding homomorphisms from bounded treewidth graphs, we present an algorithm mining frequent trees w.r.t. partially injective homomorphisms in incremental polynomial time. We design a kernel function which measures graph similarity in terms of such mutually occurring patterns and experimentally demonstrate that by bridging the gap between graph homomorphism and subgraph isomorphism, our approach offers an attractive trade-off between efficiency and predictive power.

Subsequently, we turn our attention to the popular Weisfeiler-Lehman method. This label propagation algorithm implicitly constructs tree patterns for which the embedding operator is given by locally bijective homomorphisms, another kind of constrained homomorphisms. While such patterns can be very efficiently computed and yield expressive graph representations, comparing graphs in terms of mutually occurring Weisfeiler-Lehman patterns is an often insufficient similarity measure. We propose two approaches to overcome this drawback.

Utilizing the concept of graph filtrations, we introduce a graph kernel which compares distributions of Weisfeiler-Lehman patterns over multiple graph resolutions. This approach offers a fine-grained graph similarity by comparing existence intervals of patterns, instead of their cardinalities. We show that this kernel is powerful in terms of distinguishing non-isomorphic graphs and even gives rise to complete graph kernels in certain scenarios. Moreover, the kernel can be generalized to arbitrary graph features, enabling an application beyond Weisfeiler-Lehman patterns. We empirically validate our theoretical findings on the expressive power of our kernel and provide experiments on real-world benchmark datasets which show a favorable performance of our approach compared to state-of-the-art graph kernels.

Finally, we propose a graph kernel, which compares graphs using a fine-grained similarity measure on Weisfeiler-Lehman patterns, effectively replacing the traditionally considered similarity defined by equality. This is achieved by a specifically designed tree edit distance which provides a semantically adequate and efficiently computable comparison on Weisfeiler-Lehman tree patterns. The key idea is to cluster similar patterns w.r.t. this distance measure and define a graph kernel that treats two patterns as equivalent if they belong to the same cluster. In an experimental section, we systematically investigate this kernel's predictive performance and show that it significantly outperforms state-of-the-art graph kernels on several graph benchmark datasets beyond the typically considered molecular graphs.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 5
  - 1.1.1 Graph Kernels Based on Partially Injective Homomorphisms . . . 5
  - 1.1.2 Weisfeiler-Lehman Filtration Kernels . . . 6
  - 1.1.3 Generalized Weisfeiler-Lehman Kernels . . . 8
- 1.2 Outline . . . 8
- 1.3 Previously Published Work . . . 9

2 Preliminaries **11**
- 2.1 Sets . . . 11
- 2.2 Graphs . . . 12
- 2.3 Graph Morphisms . . . 13
  - 2.3.1 Graph Equivalence . . . 14
  - 2.3.2 Orders on Graphs . . . 15
- 2.4 Bounded Treewidth Graphs . . . 15
  - 2.4.1 Homomorphisms from Bounded Treewidth Graphs . . . 16
- 2.5 Graph Pattern Mining . . . 18
  - 2.5.1 Complexity Measures for Enumeration Problems . . . 19
- 2.6 Graph Edit Distance . . . 20
  - 2.6.1 Complexity of Computing GEDs . . . 21
- 2.7 The Weisfeiler-Lehman Method . . . 21
  - 2.7.1 Unfolding Trees . . . 22
- 2.8 Kernels and Support Vector Machines . . . 23
  - 2.8.1 Kernel Functions . . . 23
  - 2.8.2 Support Vector Machines . . . 24
  - 2.8.3 Soft-margin SVMs . . . 26
  - 2.8.4 The Kernel Trick . . . 26
- 2.9 Graph Kernels . . . 27
  - 2.9.1 R-convolution Kernels . . . 27
  - 2.9.2 Graph Kernel Expressivity . . . 28
- 2.10 The Wasserstein Distance . . . 28
- 2.11 Datasets . . . 29

3 Related Work **33**
3.1 Constrained Homomorphisms . . . 33
3.1.1 Globally Constrained Homomorphisms . . . 34
3.1.2 Locally Constrained Homomorphisms . . . 35
3.2 Frequent Subgraph Mining . . . 35
3.2.1 Frequent Subtree Mining . . . 36
3.3 Homomorphisms and Weisfeiler-Lehman . . . 37
3.4 Graph Kernels . . . 37
3.5 Graph Neural Networks . . . 42

4 Graph Kernels Based on Partially Injective Homomorphisms **45**
4.1 Partially Injective Homomorphisms . . . 46
4.1.1 Reduction to Ordinary Homomorphisms . . . 47
4.1.2 The Lattice of PIHom Problems . . . 48
4.2 Pattern Mining . . . 48
4.2.1 Efficiently Decidable PIHom Problems . . . 49
4.2.2 PIHom Core Patterns: A Negative Result . . . 50
4.2.3 The Problem Definition . . . 52
4.2.4 The Mining Algorithm . . . 53
4.2.5 The Number of PIHom Patterns . . . 56
4.2.6 Pattern Embedding Computation . . . 56
4.3 The Kernel Function . . . 58
4.4 Experimental Evaluation . . . 59
4.4.1 Predictive Performance . . . 60
4.4.2 Degree of Injectivity vs. Predictive Performance . . . 61
4.4.3 Runtime Analysis . . . 62
4.5 Summary and Concluding Remarks . . . 63

5 Weisfeiler-Lehman Filtration Kernels **65**
5.1 From Filtrations to Distances . . . 67
5.1.1 Feature Persistence . . . 67
5.1.2 The Wasserstein Distance on Filtration Histograms . . . 68
5.2 The Weisfeiler-Lehman Filtration Kernel . . . 69
5.2.1 On the Expressive Power . . . 71
5.3 Filtration Kernels for Arbitrary Graph Features . . . 76
5.3.1 Linear Combination Kernel . . . 77
5.3.2 Product Kernel . . . 78
5.4 Experimental Evaluation . . . 79
5.4.1 Filtration Variants . . . 80
5.4.2 Real-World Benchmarks . . . 81
5.4.3 The Influence of the Filtration Length k . . . 81
5.4.4 Investigation of the Expressive Power . . . 83
5.4.5 Runtime Analysis . . . 84
5.5 Summary and Concluding Remarks . . . 85

6 Generalized Weisfeiler-Lehman Kernels **87**
6.1 The Weisfeiler-Lehman Tree Edit Distance 89
6.1.1 The Structure and Depth Preserving Tree Edit Distance 89
6.1.2 Computing Distances between Unfolding Trees 92
6.2 The Generalized Weisfeiler-Lehman Kernel 94
6.2.1 Unfolding Tree Vectors 95
6.2.2 Wasserstein Distance on Unfolding Tree Vectors 96
6.2.3 Unfolding Tree Barycenters 97
6.2.4 The Wasserstein k-Means Algorithm for Unfolding Trees 97
6.3 A Faster Kernel Variant 97
6.4 Experimental Evaluation 98
6.4.1 Real-world Benchmarks 100
6.4.2 Investigating Noise and Structural Deviation 102
6.5 Summary and Concluding Remarks 104

7 Conclusion **107**
7.1 Summary and Discussion 107
7.2 Outlook 108

Contents

1

Introduction

Graphs are one of the most flexible data representation languages in computer science. They model sets of objects, referred to as nodes, and pairwise relationships between them, called edges. Their ability to express complex dependencies between objects makes them a highly flexible data type. In fact, due to their high representational power, graphs are capable of modeling a majority of common data. For instance, one kind of data that is typically represented as graphs is molecular compounds, in which atoms correspond to nodes and chemical bonds to edges. Another example are social interaction networks, which are usually modeled as graphs consisting of sets of individuals (nodes) and links (edges) between them, representing, e.g., friendships.

However, the broad applicability of graphs is both a blessing and a curse. While they provide a high representational power, evaluations on this kind of data often suffer from complexity problems. Tasks that are simple for less flexible data types, may become a lot more difficult for graphs. In particular, this drawback associated with graphs directly impacts the complexity of knowledge extraction. Whereas, for simpler data types, such as, for example vectors, there exists an obvious correlation between substructures, graphs lack this straightforward correspondence. This complexity issue severely complicates the comparison of graphs, making standard *machine learning* methods harder to apply.

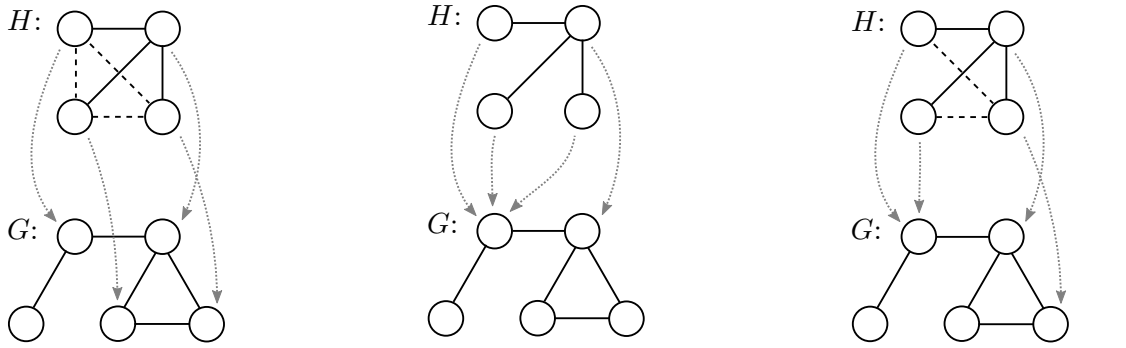
A common approach that enables machine learning on graphs, nonetheless, is by comparing them based on some fixed sets of graph *patterns*. Typically, graphs are compared in terms of their *substructures*. The concept of decomposing graphs into sets of patterns which are subsequently compared as a means to define graph similarities is the core principle of *graph kernels*. Graph kernels are powerful machine learning tools as they enable the application of, e.g., support vector machines (Boser, Guyon, and Vapnik, 1992). In fact, they have long been the dominating approach for graph classification tasks (Gärtner, Flach, and Wrobel, 2003). Even with the emergence of neural network approaches, the predictive capabilities of graph kernel methods remain state-of-the-art. Existing graph kernels primarily differ in the type of patterns. Such patterns generally correspond to fairly simple graph substructures like walks, paths, cycles, trees, or small subgraphs. Most of these graph kernels then compute graph similarities by simply comparing the numbers of co-occurring patterns (see Borgwardt et al., 2020 for a survey on graph kernels).

A specifically interesting type of patterns is *trees*. Trees form a particularly expressive pattern language in a sense that they serve as powerful graph invariants. That is, representing graphs as finite (multi-)sets of tree patterns has been shown to suffice for deciding graph isomorphism in almost all cases (Babai, Erdös, and Selkow, 1980). At the same time, tree patterns often lead to excellent predictive performances when utilized for graph classification purposes (Welke, Horváth, and Wrobel, 2017). Finally, an aspect of trees that is of particular importance to this thesis is the property that many problems which are generally NP-complete, can in fact be solved *efficiently* for this kind of graph patterns.

A substantial step of the substructure-based comparison of graphs is the identification of appropriate pattern sets. The question as to whether a graph pattern is contained in a target graph is decided by the *pattern embedding operator*. A very natural choice for the pattern embedding operator is *subgraph isomorphism*, which indicates whether or not a target graph contains a subgraph that is equivalent (i.e. isomorphic) to a graph pattern. Subgraph isomorphism is the most common definition of embedding operators, such as, for example, in the literature of classical pattern mining (Jiang, Coenen, and Zito, 2013). However, deciding subgraph isomorphism is generally an NP-complete problem, which severely limits this type of pattern embedding operators in practical applications. In fact, the problem remains NP-complete for simple pattern classes like trees and even paths.

An alternative pattern embedding operator is that of *graph homomorphism*. Compared to subgraph isomorphism, it is less restrictive in the sense that it does not require injectivity. A depiction of the conceptual differences between these two embedding operators is illustrated in Fig. 1.1. While homomorphisms are commonly used as the standard subsumption operator for more general finite relational structures (Horváth and Turán, 2001), they are less common in the context of graphs. Whereas, the problem of deciding graph homomorphism is also NP-complete in general, its complexity behaves differently from that of subgraph isomorphism on special graph classes. In particular, homomorphisms can be decided in *polynomial* time for a broad range of pattern classes for which subgraph isomorphism remains NP-complete. For instance, whereas the homomorphism problem from trees into arbitrary target graphs can be efficiently decided, this is not the case for subgraph isomorphism, even when tress are restricted to paths. However, using graph homomorphism as the pattern matching operator for trees suffers from several drawbacks. Patterns generated w.r.t. graph homomorphism often result in a *worse* predictive performance in graph classification tasks when compared to using subgraph isomorphism. Additionally, the set of subgraph patterns embeddable through homomorphism typically contains redundancies and is generally of unbounded cardinality, which ultimately complicates the extraction of a suitable set of meaningful patterns.

Despite their drawbacks, subgraph isomorphism and to a lesser degree graph homomorphism are the most prevalent pattern embedding operators. In fact, the bulk of previous research on graph pattern mining is concerned with subgraph isomorphism (Jiang, Coenen, and Zito, 2013) and occasionally homomorphism (Dries and Nijssen, 2012). Similarly, many of the most prominent graph kernels are based on patterns embedded by either graph homomorphism, as is the case with walks (Gärtner, Flach, and Wrobel, 2003), or subgraph isomorphism such as paths (Borgwardt and Kriegel, 2005), cycles (Horváth, Gärtner, and Wrobel, 2004) and small subgraphs (Shervashidze et al., 2009). However, a key observation connecting both embedding operators has largely been disregarded in this research field: In



(a) Subgraph isomorphism. (b) Graph homomorphism. (c) Partially injective homomorphism.

Figure 1.1: Conceptual visualization of three pattern embeddings from graph H into G, which underlie different injectivity constraints. Dashed edges in H indicate that two vertices need to be mapped onto different vertices in G. Fig. (a) requires all vertices to be mapped distinctly (*subgraph isomorphism*), Fig. (b) has no further injectivity constrains (*homomorphism*), and Fig. (c) requires only a subset of vertex pairs to be mapped distinctly (*partially injective homomorphism*).

fact, subgraph isomorphisms can be regarded as *constrained homomorphisms*. Indeed, they are injective homomorphisms.

Despite its algebraic simplicity, this connection between graph homomorphisms and subgraph isomorphisms has so far been ignored in the machine learning community. In this thesis, we utilize this key observation by studying the *gap* between these two standard embedding operators and other forms of constrained homomorphisms from the point of view of designing new graph kernels. In fact, as we show, one can obtain new, *semantically meaningful* graph kernels that can be calculated *efficiently* by considering constrained homomorphisms, i.e., homomorphisms which underlie additional local or global mapping restrictions.

To demonstrate the usefulness of this *unified view* of the pattern embedding operators used explicitly or implicitly in graph kernels, in this thesis we restrict the pattern language to *trees*. This choice is motivated by the above mentioned properties such as their high expressivity and suitability for classification tasks. Considering the fact that homomorphisms from trees *can* efficiently be decided, while subgraph isomorphisms generally *cannot*, this choice of pattern class is of particular interest. In this thesis, we demonstrate that constrained homomorphisms provide an attractive *trade-off* between the *low time complexity* of deciding graph homomorphisms and the *good predictive performance* achieved by patterns identified through subgraph isomorphisms. We focus on two *efficiently* decidable classes of constrained homomorphisms for *tree* patterns, by noting that our approach is *not* limited to the scenarios below.

1. *Partially injective homomorphisms.* In our first contribution, we bridge the gap between graph homomorphisms and subgraph isomorphisms by introducing the concept of *partially injective homomorphisms*, which require injectivity only for a subset of the

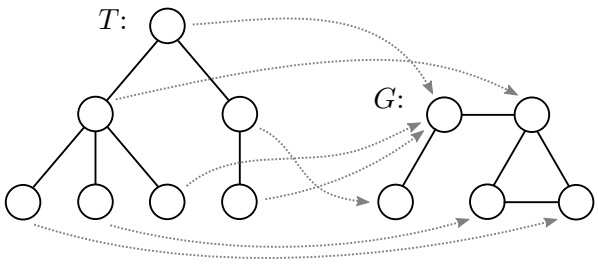


Figure 1.2: A rooted tree T, which encodes a Weisfeiler-Lehman label, is embedded into a graph G by a locally bijective homomorphism φ. That is, φ is a homomorphism from T into G s.t. for all non-leaves v in T, there exists a bijection between the children of v in T and the neighbors of $\varphi(v)$ in G.

pattern's vertices. This concept is visualized in Fig. 1.1(c). Using the class of *bounded treewidth* graphs (Robertson and Seymour, 1986), we define tree pattern embedding operators which guarantee a *maximal* degree of injectivity (w.r.t. bounded treewidth), while remaining polynomially decidable. We propose an *efficient* enumeration algorithm for mining this kind of patterns and use them effectively for graph classification tasks.

2. *Locally bijective homomorphisms.* This thesis puts a particular focus on the popular Weisfeiler-Lehman label propagation scheme (Weisfeiler and Lehman, 1968). While not commonly perceived as such, the Weisfeiler-Lehman method implicitly employs a pattern embedding operator that corresponds to *locally bijective homomorphism*, a particular kind of constrained homomorphisms. The Weisfeiler-Lehman method produces vertex labels, each encoding a *rooted* tree. A graph contains a certain label if the corresponding tree can be embedded into that graph by a locally bijective homomorphism. In simple terms, a homomorphism is locally bijective if its restriction to the neighborhood of every node is bijective. An example of such a mapping is visualized in Fig. 1.2. Motivated by the excellent predictive performance of this kind of pattern for graph classification purposes (Shervashidze et al., 2011; Kriege, Giscard, and Wilson, 2016; Togninalli et al., 2019), we introduce two *novel* kernel methods comparing graphs based on Weisfeiler-Lehman patterns.

   a. Our first approach enriches the conventional R-convolution concept of the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) by considering *graph filtrations*: Using meaningful orders on the set of edges, which allow to construct *sequences* of nested graphs, we consider a graph at multiple granularities. This perception provides access to Weisfeiler-Lehman patterns on different levels of resolution. Rather than to simply compare frequencies of patterns in graphs, it allows for their comparison in terms of *when* and for *how long* they exist in such a graph sequence. Using this insight, we propose a graph kernel that incorporates these existence intervals of patterns and yields a powerful kernel.

b. The ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) defines graphs in terms of mutually occurring Weisfeiler-Lehman patterns. This *binary* valued comparison is, however, arguably too rigid for defining suitable graph kernels. To overcome this limitation, we propose a *generalization* of this Weisfeiler-Lehman subtree kernel which takes into account a *more natural* and *finer* grade of similarity between Weisfeiler-Lehman patterns than equality (i.e., isomorphism). This similarity measure compares the rooted trees, which correspond to Weisfeiler-Lehman patterns, using a specifically designed and semantically meaningful variant of *tree edit distances*.

1.1 Contributions

In this section, we summarize the main contributions of this dissertation as outlined in 1, 2a, and 2b above.

1.1.1 Graph Kernels Based on Partially Injective Homomorphisms

As one specific notion of constrained homomorphisms, we introduce the concept of *partially injective homomorphisms*. Partially injective homomorphisms provide a generalized view of graph homomorphisms as well as subgraph isomorphisms and bridge the gap between the two in order to allow for more flexible pattern embedding operators. Informally, a partially injective homomorphism from a pattern graph H into a target graph G is a homomorphism which additionally respects a set of injectivity constraints C. Such constraints correspond to pairs of nodes in H which are required to be mapped onto distinct nodes in G. In other words, a partially injective homomorphism from H to G respecting constraints C is a homomorphism for which the vertices u and v for every pair $\{u, v\} \in C$ are mapped onto distinct vertices in G. In Fig. 1.1, such constraints have been explicitly visualized in form of dashed edges. The special cases of ordinary graph homomorphism and subgraph isomorphism are obtained by requiring no constrains (i.e., $C = \varnothing$) and constraints between *all* unconnected vertex pairs in H, respectively.

When considering trees as the pattern language, a key observation is that the concept of partially injective homomorphisms not only bridges the gap between graph homomorphisms and subgraph isomorphisms, but simultaneously forms a transition of decision problems in P to a generally NP-complete problems. Given this perspective, we introduce a class of pattern embedding operators that provide a maximal degree of partial injectivity while remaining efficiently decidable. Thus, such embedding operators provide a trade-off between the suitability of subgraph isomorphism for graph classification tasks and the complexity of deciding homomorphisms.

The complexity of deciding partially injective homomorphisms is subject to not only the pattern tree H but also the set of injectivity constraints C. A key observation is that deciding partially injective homomorphism from a pattern H respecting constraints C can be reduced to deciding ordinary homomorphism from the *edge extended* graph $H_C = (V(H), E(H) \cup C)$, i.e., the graph which results from extending H by edges corresponding to vertex pairs in C.

Thus, the complexity of deciding partially injective homomorphism is subject to both, the pattern H and the constraint set C.

While there exist several graph classes from which ordinary homomorphisms can be efficiently decided, we focus on the class of *bounded treewidth* graphs (Robertson and Seymour, 1986). That is, we consider tree patterns H together with constraints C such that the edge extended graphs H_C have treewidth at most k for some constant k. In such cases, the corresponding partially injective homomorphism problems can be decided in polynomial time (Dalmau, Kolaitis, and Vardi, 2002). The choice of bounded treewidth graphs is optimal in the sense that homomorphisms can be decided in polynomial time if and only if the pattern graph has bounded treewidth (Grohe, 2007).[1]

This kind of embedding operators allows to efficiently identify subtree patterns in graphs. However, instead of considering the set of all subtrees, a common approach is to compare graphs based on the set of frequently occurring patterns. These so-called *frequent patterns* have been shown on many occasions to serve as powerful graph representations leading to remarkable predictive performances.

In order to identify the set of frequent patterns, we propose a mining algorithm that enumerates the frequent trees w.r.t. partially injective homomorphism. As opposed to ordinary homomorphism or subgraph isomorphism, partially injective homomorphisms are specific to the pattern at hand. Since the problem of whether a tree pattern H can be embedded into a target graph G directly depends on the set of constraints C, the output of the mining algorithm consists of pairs (H, C).

Following the intention of inducing a *maximal* degree of injectivity while guaranteeing the feasibility of the embedding operator, we consider edge maximal graphs of bounded treewidth, also known as *k-trees*. An interesting aspect about k-trees is that they allow for a very natural algorithmic graph *refinement operator*, i.e, a method that traverses the pattern space by gradually constructing increasingly larger pattern graphs. Using this refinement method, we show that frequent patterns can be efficiently enumerated.

In an empirical evaluation, we analyze the predictive performance of tree patterns mined w.r.t. partially injective homomorphism by using a simple kernel function that computes graph similarity in terms of mutually occurring frequent patterns. Furthermore, we compare the runtimes of the mining algorithm to ordinary enumeration methods based on subgraph isomorphism. Our empirical results clearly show that the predictive performance obtained by our approach is very close to that using ordinary frequent subgraphs as the pattern language. These results could already be achieved for small treewidth values, indicating that our approach provides an attractive trade-off between complexity and predictive performance.

1.1.2 Weisfeiler-Lehman Filtration Kernels

While the Weisfeiler-Lehman subtree patterns have proven to be excellent features for graph prediction tasks on many occasions, a drawback of this kind of patterns is their "specificity". More precisely, such patterns encode k-hop node neighborhoods which are often unique and thus unsuitable for graph representations. This specificity problem is attributed to the Weisfeiler-Lehman method which implicitly utilizes *locally bijective homomorphisms* as

[1] Assuming FPT $\neq$ W[1] which is widely believed to be true.

the pattern embedding operator. As visualized in Fig. 1.2, such a homomorphism φ furthermore requires that for every vertex v in the pattern tree T, there exists a bijection between the children of v and the set of neighbors of $\varphi(v)$ in the target graph G. That is, T is embeddable into G if and only if for every v in T, the children of v and the neighbors of $\varphi(v)$ perfectly match. In order to relax this rigid notion of pattern matching, we propose considering *subsets of edges* in the target graph which are selected according to edge weights.

Using the above concept, the key idea is to regard and compare graphs at *multiple levels of resolution*. This is realized using the notion of *graph filtrations*, which define sequences of nested subgraphs that differ in the sets of edges. Given a graph G, such a sequence has the form $G_1, \ldots, G_k = G$ and can be viewed as incremental refinements that construct the graph G by gradually adding sets of edges. Clearly, with changing sets of edges, the node neighborhoods and hence the sets of Weisfeiler-Lehman subtree patterns change as well. Consequently, a pattern occurring at some point in the sequence may disappear at a later moment. We track such existence intervals of Weisfeiler-Lehman subtree patterns, which ultimately allows for a comparison not only in terms of pattern frequency, but also by *when* and for *how long* they exist.

This comparison of pattern occurrence distributions is realized using the *Wasserstein distance*, for which we show that it yields proper kernel functions on this kind of information. Using this result, we introduce the *Weisfeiler-Lehman filtration kernel*, which defines graph similarities by comparing such pattern occurrence distributions. Concerning the complexity, we show that our kernel increases the complexity of the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) only by a linear factor.

The approach combines several profitable novelties. In particular, it allows to handle continuous edge attributes. Furthermore, it enables the identification of *partial* Weisfeiler-Lehman subtree patterns, i.e., such patterns which appear only in filtration graphs, but not in the original graph itself. Utilizing these advantages, we show that the Weisfeiler-Lehman filtration kernel is a complete kernel for certain filtrations. That is, for suitable choices of filtrations, the kernel is capable of distinguishing all pairs on non-isomorphic graphs. This property is commonly regarded as one of the main ingredients of ideal graph similarity functions. We discuss that this result has implications beyond kernel methods such as graph neural networks.

We note that this definition of graph similarity is not limited to the use of Weisfeiler-Lehman patterns but in fact works with any type of graph feature that yields finite graph multiset representations. We therefore generalize the above approach and formally introduce a family of graph kernels, called *graph filtration kernels*. For this class of kernels, we show that they generalize ordinary dot products between graph multiset representations.

In an experimental evaluation, we empirically validate our theoretical findings on the expressive power of the Weisfeiler-Lehman filtration kernel. By utilizing suitable filtration functions, we demonstrate that our approach can in fact distinguish more non-isomorphic graphs than the ordinary Weisfeiler-Lehman subtree kernel. Furthermore, we evaluate the predictive performance obtained by the Weisfeiler-Lehman filtration kernel and show that it outperforms other state-of-the-art graph kernels on several real-world benchmark datasets. Finally, we provide runtimes which clearly support our claim on the efficiency of our method.

1.1.3 Generalized Weisfeiler-Lehman Kernels

The Weisfeiler-Lehman subtree patterns have most famously been employed in the Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011), which ranks among the best performing graph kernels on most benchmark datasets (Kriege, Johansson, and Morris, 2020). Implicitly, this kernel defines graph similarity in terms of pairwise comparisons of subtree patterns by *equality*. In other words, the (dis-)similarity between two subtree patterns is restricted to whether they are isomorphic or not, and is hence, unable to quantify more fine-grained similarities between them. Thus, the limitation of the Weisfeiler-Lehman subtree kernel is that two subtree patterns which are structurally completely different are treated identically to two subtree patterns which differ only slightly.

Motivated by this observation, we propose a generalization of the Weisfeiler-Lehman subtree kernel by relaxing the above outlined strict comparison of tree patterns. To achieve this, we propose a natural (dis-)similarity measure between Weisfeiler-Lehman subtree patterns that allows to compare them on a much finer scale. This measure is based on a variant of the *tree edit distance*, which provides a semantically adequate comparison for the specific case of Weisfeiler-Lehman subtree patterns. More precisely, we designed a tree edit distance, which specifically respects essential properties of this kind of patterns w.r.t. the implicitly induced pattern embedding operator defined by locally bijective homomorphisms. This distance closely reflects the dissimilarity of node neighborhoods that are represented by Weisfeiler-Lehman subtrees. We show, that in contrast to more general definitions of tree edit distances, this kind of tree edit distance is in fact efficiently computable.

Using this result, we introduce a novel graph kernel, which generalizes the Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) by considering the above fine-grained comparison between subtree patterns. This is achieved by identifying groups of similar Weisfeiler-Lehman patterns and clustering them such that elements within the same cluster are treated as equal. That is, we generalize the ordinary Weisfeiler-Lehman subtree kernel by regarding two subtree patterns as equivalent if they belong to the same cluster, i.e., have a small distance to each other. We show that the partitioning of subtree patterns can be efficiently performed using the concept of *Wasserstein k-means clustering* (Irpino, Verde, and Carvalho, 2014). This choice is motivated by our result that our adaptation of the tree edit distance between subtree patterns can in fact be reformulated in terms of the Wasserstein distance.

In an extensive empirical evaluation, we show that our generalization of the Weisfeiler-Lehman subtree kernel significantly outperforms state-of-the-art graph kernels, including the ordinary Weisfeiler-Lehman subtree kernel, on graph datasets beyond the typically considered molecular graphs. The results clearly indicate that although our more general approach does not improve the predictive performance on small molecular graphs, which are sparse and structurally simple, it considerably outperforms state-of-the-art graph kernels on datasets containing dense and structurally diverse graphs.

1.2 Outline

The remainder of this thesis is structured as follows. In Chapter 2, we recall all necessary notions and notations. We cover several basic definitions and concepts as well as algorithms

and results that are relevant to this thesis. In Chapter 3, we provide background information on a range of related topics and put or contributions into context. The three subsequent chapters are dedicated to the main contributions of this dissertation. In particular, Chapter 4 introduces partially injective homomorphisms and discusses cases for which they can be efficiently decided. It provides an efficient pattern mining algorithm, enabling graph kernels that compare graphs based on such mutual patterns. In Chapter 5, we utilize the concept of graph filtrations in order to introduce graph kernels that compare distributions of Weisfeiler-Lehman patterns over sequences of graphs. We cover the kernel's expressive power and demonstrate the practical use of this graph similarity measure. A different approach comparing graphs based on Weisfeiler-Lehman patterns is presented in Chapter 6. In it, we introduce a semantically meaningful similarity measure between this kind of patterns and propose a corresponding clustering method, leading to graph kernels of remarkable predictive performance. Finally, in Chapter 7, we conclude the dissertation by summarizing the main contributions and outlining potential future research questions.

1.3 Previously Published Work

The contents of this thesis are based on joint work with Tamás Horváth, Pascal Welke and Stefan Wrobel. They have been published in the following conferences and journals.

- Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel (2018). "Mining Tree Patterns with Partially Injective Homomorphisms". In: *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pp. 585–601. DOI: 10.1007/978-3-030-10928-8_35

- Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel (2022). "A generalized Weisfeiler-Lehman graph kernel". In: *Machine Learning* 111.7, pp. 2601–2629. DOI: 10.1007/s10994-022-06131-w

- Till Hendrik Schulz, Pascal Welke, and Stefan Wrobel (2022). "Graph Filtration Kernels". In: *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 8196–8203. DOI: 10.1609/aaai.v36i8.20793

2

Preliminaries

In this chapter, we collect all necessary notions and establish the notation. We provide basic definitions on sets in Sect. 2.1 and discuss orderings and refinement operators. Subsequently, Sect. 2.2 covers all concepts and notations on graphs. Sect. 2.3 is concerned with the two central graph embedding operators: graph homomorphism and subgraph isomorphism. We proceed by introducing the class of bounded treewidth graphs in Sect. 2.4 and recall a polynomial time algorithm for deciding graph homomorphisms from this kind of graphs. We quickly cover the concept of frequent pattern mining as well as a corresponding complexity notion in Sect. 2.5, and discuss graph edit distances in Sect. 2.6. Subsequently, Sect. 2.7 covers one of the central notions of this dissertation, the Weisfeiler-Lehman vertex relabeling method. We proceed by recalling kernel functions and principles of support vector machines in Sect. 2.8, before discussing graph kernels in Sect. 2.9. Finally, we recap the Wasserstein distance in Sect. 2.10, and provide a detailed description on the datasets considered in this dissertation in Sect. 2.11. Readers that are familiar with the notions can skip the corresponding sections.

2.1 Sets

A *set* S is a collection of unique elements. Sets are generalized by *multisets*, which may contain multiple occurrences of elements and are denoted using double curly braces, e.g., $S = \{\{a, a, b, c\}\}$. We denote the *cardinality* of S by $|S|$, and write $\varnothing$ for the *empty set*. A *subset* X of S is denoted by $X \subseteq S$ and $[S]^k = \{X \subseteq S : |X| = k\}$. The *power set* of S is the set containing all subsets of S and is denoted by 2^S. Analogously, $\mathbb{N}^S$ denotes the set of all multisets over S. We write $\mathbb{N}$ for the set of *natural numbers*, and $\mathbb{R}$ for the set of *real numbers*. The set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. An n-dimensional *vector* $x \in \mathbb{R}^n$ is an ordered set of n real numbers $x_1, \ldots, x_n$. The ℓ^p norm of $x \in \mathbb{R}^n$ with integer $p > 0$ is defined by $\|x\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$.

We sometimes define (partial) orders on elements of a set S. For a more detailed description on the following definitions, we refer to, e.g., Davey and Priestley (2002).

Definition 2.1 (Preordered Set). *A* preordered set *is a pair* $(S, \preceq)$*, where* S *is a set and* $\preceq$ *is a binary relation on* S *such that for all* $x, y, z \in S$*:*

(i) $x \preceq x$, *(reflexivity)*

(ii) *if* $x \preceq y$ *and* $y \preceq z$ *then* $x \preceq z$. *(transitivity)*

A special case of preorders are partial orders which additionally require antisymmetry.

Definition 2.2 (Partially Ordered Set). *A preordered set* $(S, \preceq)$ *is a* partially ordered set*, or just* poset*, if it is also antisymmetric, i.e., if for all* $x, y, z \in S$*,* $x \preceq y$ *and* $y \preceq x$ *imply* $x = y$*.*

Some applications require traversing through a given partially ordered set $(S, \preceq)$. This can be achieved by *refinement operators*.

Definition 2.3 (Refinement Operator). *Given a preordered set* $(S, \preceq)$ *and* $x, y, z \in S$*, the function* $\rho : S \to 2^S$ *is called a* refinement operator *if for all* $x \in S : \rho(x) \subseteq \{y \in S : x \preceq y\}$*. It is called*

locally finite if for all $x \in S : \rho(x)$ *is finite and computable,*

complete if for all $x \preceq y$ *there is a chain* $x = r_0, \ldots, r_k = y$ *with* $r_i \in \rho(r_{i-1}), i \in [k]$*,*

proper if for all $x \in S : \rho(x) \subseteq \{y \in S : x \preceq y\}$*.*

We call ρ ideal *if it is locally finite, complete and proper.*

For an in-depth discussion on refinement operators, we refer to, e.g., Laag and Nienhuys-Cheng (1998).

2.2 Graphs

In the following, we collect the necessary notions from graph theory (see, e.g., Diestel, 2012) and fix the notations.

Directed and Undirected Graphs. A *graph* is a pair $G = (V, E)$ consisting of a set V of *vertices* and a set E of *edges*. G is called a *directed* graph if its edges have associated directions, i.e., $E \subseteq V \times V$. Analogously, G is called *undirected* if $E \subseteq \{X \subseteq V : |X| = 2\}$. For a graph G, we refer to its set of vertices as $V(G)$ and to its set of edges as $E(G)$. We often denote an edge $\{u, v\} \in E(G)$ simply by $uv \in E(G)$. A *rooted* graph is a graph with a distinguished vertex $r \in V(G)$, called the root. We sometimes write $r(G)$ for the root of G. The class of all graphs is denoted by $\mathcal{G}$ and the set of all graphs with at most n vertices is denoted by $\mathcal{G}_n$. Within this thesis, we exclusively consider simple graphs, i.e., graphs without loops or multiple edges. Furthermore, if not explicitly stated otherwise, we assume graphs to be undirected.

Labeled, Attributed and Weighted Graphs. A *labeled* graph is a graph G equipped with a *labeling function* $\ell : V(G) \cup E(G) \to \Sigma$ assigning a label from Σ to each vertex and edge in G. We sometimes assume ℓ to be a global function over a set of graphs. A graph is commonly called an *attributed graph* whenever the labeling function assigns real-valued vectors to vertices and edges, i.e., if $\Sigma := \mathbb{R}^d$ for some $d > 0$. Similarly, a *weighted* graph is a graph G equipped with a weight function $\omega : E(G) \to \mathbb{R}_+$ assigning non-negative weights to edges. For simplicity, we assume all graphs to be labeled by noting that labeled graphs can easily be obtained from unlabeled ones by utilizing a trivial labeling function.

Neighborhoods and Degrees. For two vertices u, v of a graph G, u is called a *neighbor* of v if there exists an edge $\{v, u\} \in E(G)$. The set of all neighbors of v is denoted by $\mathcal{N}(v)$ and its cardinality is called the *degree* of v, denoted $\delta(v)$. The *k-hop neighborhood* of v is the set of vertices in G which have shortest-path distance at most k from v in G.

Subgraphs. A graph $G' = (V', E')$ is called a *subgraph* of a graph $G = (V, E)$, if $V' \subseteq V$ and $E' \subseteq E$. If G' is a subgraph of G, and G' contains all edges $uv \in E$ with $u, v \in V'$, then G' is called an *induced subgraph* of G. Given a set $S \subseteq V(G)$, the subgraph of G induced by S contains all edges of G whose endpoints are in S and is denoted by $G[S]$.

Walks, Paths, Cycles and Trees. A *walk* in a graph G is a sequence $v_0, v_1, \ldots, v_k$ of vertices such that $\{v_{i-1}, v_i\} \in E(G)$ for $i \in [k]$ and k is called the length of the walk. A *path* is a graph consisting of a sequence of distinct nodes with consecutive nodes being connected by an edge. Similarly, a *cycle* is a path such that the start and end points are furthermore connected by an edge. G is called *connected* if there exists a path between all vertex pairs. An *acylcic* graph, i.e., a graph which does not contain any cycles, is called a *forest*, while a connected forest is called a *tree*. A tree $T = (V, E)$ has exactly $|V| - 1$ edges and its vertices of degree 1 are referred to as *leaves*. For a rooted tree T and $v \in V(T)$, we denote the subtree of T rooted in v by $T[v]$ and we write $F(v)$ for the set of all subtrees rooted in the children of v.

Complete Graphs and Cliques. A graph $G = (V, E)$ is called *complete* if all vertices in G are pairwise adjacent, i.e., if $u, v \in V$ with $u \neq v$ implies $uv \in E$. A set $S \subseteq V$ with $n = |S|$ forms an *n-clique* in G if the induced subgraph $G[S]$ is complete.

2.3 Graph Morphisms

Graph morphisms are mappings between two graphs that retain certain structural properties. They define relations on graphs which induce notions on equivalence and subsumption. Graph morphisms are of particular interest in this dissertation, as they correspond to *graph embedding operators*. In the following, we discuss the two most common graph morphisms: *graph homomorphisms* and *subgraph isomorphisms*.

Definition 2.4 (Graph Homomorphism). *A* homomorphism *from a graph $G = (V, E, \ell)$ into a graph $G' = (V', E', \ell')$ is a function $\varphi : V \to V'$ preserving all edges and labels, i.e.,*

(i) $\varphi(u)\varphi(v) \in E'$ *for all* $uv \in E$,

(ii) $\ell(v) = \ell'(\varphi(v))$ *for all* $v \in V$, *and*

(iii) $\ell(uv) = \ell'(\varphi(u)\varphi(v))$ *for all* $uv \in E$.

We say G *is* homomorphic *to* G', *denoted* $G \preceq_h G'$, *if such a function exists.*

Graph homomorphisms are closely related to the more general concept of relational homomorphisms. This notion is commonly regarded as the standard subsumption operator in inductive logic programming (Nienhuys-Cheng and Wolf, 1997). Graph homomorphisms play a decisive role in graph theory (Hell and Nesetril, 2004). However, due to the lack of injectivity, graph homomorphisms are largely disregarded as graph embedding operators. Instead, subgraph isomorphisms are commonly considered the standard matching operators on graphs.

Definition 2.5 (Subraph Isomorphism). *For graphs* G *and* G', *the mapping* $\varphi : V(G) \to V(G')$ *is called a* subgraph ismorphism *if*

(i) φ *is a homomorphism, and*

(ii) φ *is injective.*

If such a function exists, we say G *is* subgraph isomorphic *to* G' *and denote it by* $G \subseteq G'$.

2.3.1 Graph Equivalence

Graph morphisms partition the set of all graphs $\mathcal{G}$ into equivalence classes. That is, two graphs are considered equivalent w.r.t. a graph morphism type if there exist such morphisms between the graphs in both directions. The most common notion of equivalence on graphs is defined by graph isomorphism.

Definition 2.6 (Graph Isomorphism). *A* graph isomorphism *from a graph* $G = (V, E, \ell)$ *into a graph* $G' = (V', E', \ell')$ *is a bijection* $\varphi : V \to V'$ *preserving all edges and labels in both directions, i.e.,*

(i) $uv \in E \iff \varphi(u)\varphi(v) \in E'$ *for all* $u, v \in V$,

(ii) $\ell(v) = \ell'(\varphi(v))$ *for all* $v \in V$, *and*

(iii) $\ell(uv) = \ell'(\varphi(u)\varphi(v))$ *for all* $uv \in E$.

We say G *is* isomorphic *to* G' *if such a function exists, and write* $G \equiv G'$.

In other words, $G \equiv G'$ if and only if $G \subseteq G'$ and $G' \subseteq G$. We denote the set of graphs modulo isomorphism by $\mathcal{G}/\equiv$. Analogously, graph homomorphism defines an equivalence relation on $\mathcal{G}$.

Definition 2.7 (Graph Homomorphism Equivalence). *Two graphs* G *and* G' *are* homomorphism equivalent, *denoted* $G \equiv_h G'$, *if* $G \preceq_h G'$ *and* $G' \preceq_h G$.

Comparing the two equivalence relations, we find that $\equiv$ is in fact finer than $\equiv_{\mathrm{h}}$. That is, the partition of $\mathcal{G}$ induced by $\equiv$ is a refinement of that induced by $\equiv_{\mathrm{h}}$.

2.3.2 Orders on Graphs

Graph morphisms define orders on sets of graphs. As subgraph isomorphism is a reflexive, transitive and antisymmetric relation, the following holds:

Proposition 2.1. *The subgraph isomorphism relation $\subseteq$ defines a partial order on $\mathcal{G}/\equiv$.*

In the case of homomorphisms, it can be shown that $\preceq_{\mathrm{h}}$ is a reflexive and transitive relation on $\mathcal{G}/\equiv$. However, $\preceq_{\mathrm{h}}$ is not antisymmetric since $G \preceq_{\mathrm{h}} G'$ and $G' \preceq_{\mathrm{h}} G$ do not imply that G and G' are isomorphic. The following proposition is immediate (Hell and Nesetril, 2004).

Proposition 2.2. *The graph homomorphism relation $\preceq_h$ defines a preorder on $\mathcal{G}/\equiv$.*

However, the relation $\preceq_{\mathrm{h}}$ may be transformed into a partial order for other graph classes. For instance, this can be achieved by identifying a unique representative for each class of homomorphism equivalent graphs. This so called *core* is defined as the smallest graph of a homomorphism equivalence class. More formally:

Definition 2.8 (Core). *A graph G is called a* core *if there exists no homomorphism from G into a proper subgaph of itself. We refer to the core $c \equiv_h G$ as the core of G.*

Using that the core of a graph is unique modulo isomorphism, the following proposition holds (Hell and Nesetril, 2004).

Proposition 2.3. *The graph homomorphism relation $\preceq_h$ defines a partial order on the set of all non-isomorphic cores in $\mathcal{G}$.*

2.4 Bounded Treewidth Graphs

The *treewidth* (Robertson and Seymour, 1986) of a graph can intuitively be interpreted as its tree-likeness. It is defined using the concept of tree-decompositions, which decompose a graph G into a tree whose vertices correspond to specific induced subgraphs of G.

Definition 2.9 (Tree-Decomposition). *A* tree-decomposition *of a graph $G = (V, E)$ is a tuple $TD(G) = (T, X, r)$ where $T = (I, F)$ is an unordered tree with root r and $X = \{B_i : i \in I\}$ is a family of subsets of V, called* bags*, such that:*

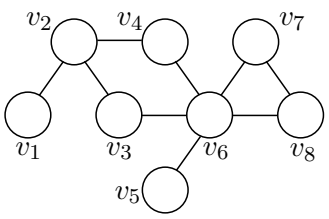
(i) $\bigcup_{i \in I} B_i = V$,

(ii) for every $uv \in E$ there is an $i \in I$ with $\{u, v\} \subseteq B_i$, and

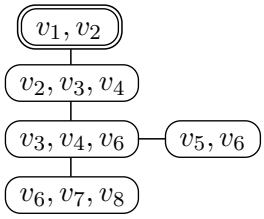
(iii) for every $v \in V$ the set of nodes $\{i | v \in B_i\}$ forms a subtree of T

The width *of a tree-decomposition is defined by the largest cardinality over all bags minus one, i.e., $max_i|B_i| - 1$.*

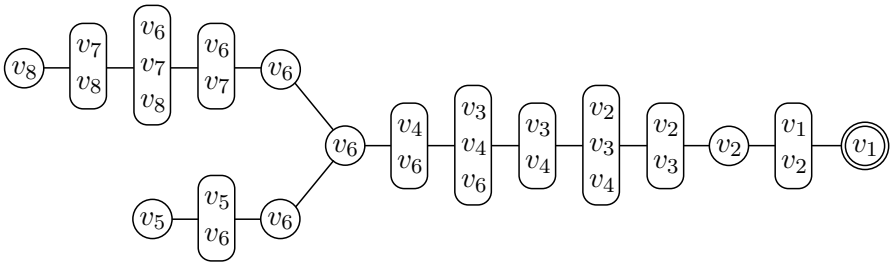
An example of a tree-decomposition is given in Fig. 2.1(b). Clearly, the tree-decomposition of a graph is not unique and the trivial tree-decomposition of a graph G consists of a single bag containing all vertices of G. However, the most relevant tree-decompositions are such which are minimal w.r.t. the width.



(a) Graph G of treewidth 2.



(b) A tree-decomposition of G.



(c) A nice tree-decomposition of G.

Figure 2.1: Example of a graph G and its minimal width (nice) tree-decompositions.

Definition 2.10 (Treewidth). *The* treewidth *of a graph G is the smallest width over all tree-decompositions of G.*

The problem of determining the treewidth for an arbitrary graph is, in general, NP-hard (Arnborg, Corneil, and Proskurowski, 1987). However, for any constant k, there exists a linear time algorithm capable of recognizing whether a graph has treewidth at most k (Bodlaender and Kloks, 1996). Furthermore, given a graph G of treewidth (at most) k, the algorithm outputs a tree-decomposition of width (at most) k in linear time.

A specifically interesting family of graphs are *k-trees* which are maximal graphs of treewidth k, in a sense that no edge can be added without increasing the treewidth (Rose, 1974). k-trees have a simple algorithmic description.

Definition 2.11 (k-Tree). *For $k \in \mathbb{N}$, a $k-$*tree *is recursively defined as follows:*

(i) A clique of $k+1$ vertices is a k-tree, and

(ii) given a k-tree G with n vertices, a k-tree with $n+1$ vertices is obtained from G by adding a new vertex v to G and connecting v to all vertices of a k-clique of G.

2.4.1 Homomorphisms from Bounded Treewidth Graphs

Deciding whether a homomorphism from a graph H of bounded treewidth into an arbitrary target graph G exists can be done in *polynomial time* using a dynamic programming approach. The general idea of this algorithm is to utilize the properties of tree-decompositions and compute a set of partial mappings which are subsequently "stitched together" into a solution.

For the sake of a simpler algorithmic description, we assume the tree-decomposition to fulfill a number of further structural properties. Such tree-decompositions are referred to as *nice* tree-decompositions. The following description is based on Díaz, Serna, and Thilikos (2002).

Definition 2.12 (Nice Tree-Decomposition). *A* nice tree-decomposition *of a graph G is a rooted tree-decomposition $TD(G) = (T, X, r)$ for which every node i in T has at most two children such that:*

(i) if i has exactly one child j, then either $|B_i \backslash B_j| = 1$ or $|B_j \backslash B_i| = 1$, and

(ii) if i has two children j, l, then $B_i = B_j = B_l$.

An example of a nice tree-decomposition can be found in Fig. 2.1(c). Finding a nice tree-decomposition can be done without any significant overhead. In fact, given a tree-decomposition of a graph G, a nice tree-decomposition can be constructed in time $O(|V(G)|)$ (Lemma 2.1 in Díaz, Serna, and Thilikos, 2002).

Nice tree-decompositions have the convenient property that each node belongs to one of four different types. For a nice tree-decomposition (T, X, r), a node $i \in V(T)$ is called:

Start node if i is a leaf,

Introduce node if i has exactly one child j and $|B_i| = |B_j| + 1$,

Forget node if i has exactly one child j and $|B_i| = |B_j| - 1$,

Join node if i has two children j, l.

While, in this thesis, we are merely interested in deciding graph homomorphism from graphs of bounded treewidth, in the following, we outline an algorithm for counting the number of homomorphisms, by noting that the algorithm for both problems is nearly identical. The algorithm utilizes a dynamic programming approach and iterates over the vertices in the nice tree-decomposition in a bottom-up manner. Roughly speaking, the idea is to compute homomorphisms from subgraphs induced by bags which align with the homomorphisms of the bags underneath it.

Alg. 1 describes the procedure for counting homomorphisms from graphs of bounded treewidth. The input is a nice tree-decomposition (T, X, r) of the pattern graph H and a target graph G, and returns the number of homomorphisms from H into G. The algorithm traverses the nodes $i \in V(T)$ in a bottom-up manner (line 2) using an ordering on $V(T)$ which ensures that a node can only be processed once all its descendants have been processed (line 1). Let $M_i = \hom(H[B_i], G)$ be the set of all possible homomorphisms from the subgraph of H induced by the vertices in the bag corresponding to node i (i.e., bag B_i). Each node $i \in V(T)$ is associated with a table that stores a counter for each homomorphism $\varphi \in M_i$, denoted $\#_i(\varphi)$. This value $\#_i(\varphi)$ holds the number of homomorphisms from the subgraph induced by the union of all bags underneath i (and including i) which "align" with φ. More formally, $\#_i(\varphi) = |\{\phi \in \hom(H[\bigcup_{j \in V(T[i])} B_j], G) : \phi[B_i] = \varphi\}|$ with $V(T[i])$ being the set of successor nodes of i in T (and including i itself) and $\phi[S]$ denoting the function ϕ restricted

to domain $S \subseteq V(H)$. The values $\#_i(\varphi)$ are computed depending on the type of node i as follows.

Start For simplicity, we assume that the bag corresponding to i contains exactly one vertex v, i.e., $B_i = \{v\}$. The homomorphisms from $H[\{v\}]$ into G are then simply all mappings $\{(v, x)\}$ with $x \in V(G)$. For each such homomorphism φ, the count is set to 1.

Introduce Let j be the child of i. Bag B_i introduces a new vertex v that is not contained in B_j. For homomorphisms $\varphi \in M_j$, it is checked whether φ can be extended by mappings $\{(v, x)\}$ such that $\varphi \cup \{(v, x)\}$ is a valid homomorphism from $H[B_i]$. If this is the case, we set the count of such $\varphi \cup \{(v, x)\}$ equal to the count of φ.

Forget Let j be the child of i. Bag B_i removes a vertex v which is contained in B_j. Thus, for a homomorphism $\varphi : B_i \to V(G)$, we consider all extended homomorphisms $\varphi \cup \{(v, x)\} \in M_j$ and add up their counts.

Join Node i has two children j and l. It holds that $B_i = B_j = B_l$. Recall that for $\varphi \in M_i$, the counts $\#_j(\varphi)$ and $\#_l(\varphi)$ correspond to the number of homomorphisms from two different subgraphs of H for which φ is a partial mapping. To compute all combinations of such homomorphisms, the counts are multiplied.

Note that in the above description, we assume graphs to be unlabeled. A generalization to labeled graphs is straight-forward. For a detailed discussion on Alg. 1, we refer to Díaz, Serna, and Thilikos (2002).

2.5 Graph Pattern Mining

Frequent pattern mining is concerned with the extraction of valuable insights from graphs. Its objective is the enumeration of graph patterns for which the occurrence count within a graph dataset exceeds a predefined threshold. In the following, we provide a general definition of the frequent pattern mining problem:

Frequent Pattern Mining Problem: *Given*

(i) a finite set $\mathcal{D} \subseteq \mathcal{G}$ of graphs for some graph class $\mathcal{G}$,

(ii) a graph pattern language $\mathcal{P}$,

(iii) an embedding operator $\preceq$ from patterns in $\mathcal{P}$ into graphs in $\mathcal{G}$,

and an integer $t > 0$, *list* all $P \in \mathcal{P}$ such that $\text{freq}(P, \mathcal{D}) \geqslant t$, where $\text{freq}(P, \mathcal{D})$ denotes the (absolute) *frequency* of pattern P in $\mathcal{D}$, i.e., $\text{freq}(P, \mathcal{D}) = |\{G \in \mathcal{D} : P \preceq G\}|$.

The set $\{G \in \mathcal{D} : P \preceq G\}$ is called the *support set* of P. Usually, we define the frequency threshold t as a relative threshold value w.r.t. the size of $\mathcal{D}$. That is, a pattern is required to be contained it at least a certain fraction σ of the dataset graphs, i.e., $t = \sigma|\mathcal{D}|$.

The overwhelming amount of literature concerned with graph pattern mining utilizes subgraph isomorphism as the pattern embedding operator (Jiang, Coenen, and Zito, 2013).

Algorithm 1 Count Homomorphisms from Graphs of Bounded Treewidth

input: Nice tree-decomposition $TD(H) = (T, X, r)$; target graph G
output: Number of homomorphisms from H to G.

Let $(u_1, \ldots, u_n)$ be an ordering on $V(T)$ s.t. $u_n = r$, and for all $p, q \in [n]$, if u_p is a descendant of u_q then $p < q$.
for node i in order of $(u_1, \ldots, u_n)$ **do**
    if i is a *Start* node **then**
        Let $\{v\} = B_i$.
        For all $x \in V(G)$, set $\#_i((v, x)) = 1$.
    if i is a *Introduce* node **then**
        Let j be the child of i and let $\{v\} = B_i \backslash B_j$.
        for all $\varphi \in M_j$ and $x \in V(G)$ **do**
            if for all $u \in \mathcal{N}(v) \cap B_j : (\varphi(u), x) \in E(G)$ **then**
                Set $\#_i(\varphi \cup \{(v, x)\}) = \#_j(\varphi)$
            else
                Set $\#_i(\varphi \cup \{(v, x)\}) = 0$
    if i is a *Forget* node **then**
        Let j be the child of i and let $\{v\} = B_j \backslash B_i$.
        for all $\varphi \in M_i$ **do**
            Set $\#_i(\varphi) = \sum_{x \in V(G)} \#_j(\varphi \cup \{(v, x)\})$
    if i is a *Join* node **then**
        Let j, l be children of i.
        for all $\varphi \in M_i$ **do**
            Set $\#_i(\varphi) = \#_j(\varphi) \cdot \#_l(\varphi)$
return $\sum_{\varphi \in M_r} \#_r(\varphi)$

The problem is therefore also commonly known as the *frequent subgraph mining* problem. We note, that the pattern class $\mathcal{P}$ is almost always restricted to *connected* graphs. This restriction is owed to the convenient reduction of elements in $\mathcal{P}$ as well as simpler pattern candidate generation procedures. Therefore, by pattern graphs we always mean connected graphs.

2.5.1 Complexity Measures for Enumeration Problems

The complexity of the pattern mining problem naturally depends on the size of the output. In fact, for the general graph mining problem above, the output may even be of infinite size, depending on the choices for (i) through (iii). Even for more restricted definitions, such as the standard frequent connected subgraph mining task, where $\mathcal{P}$ is the set of all connected graphs and $\preceq$ corresponds to subgraph isomorphism, the output can be exponential in the size of $\mathcal{D}$. Thus, it is common to consider a more fine-grained notion of complexity for enumeration problems (see, e.g., Johnson, Papadimitriou, and Yannakakis, 1988). This notion considers

the time it takes to generate the output not only relative to the size of the input, but also with respect to the size of the output.

More formally, for input $\mathcal{D}$ and output set $\mathcal{O} = \{P_1, \ldots, P_n\}$, a listing algorithm enumerates the elements of $\mathcal{O}$ with:

polynomial delay if the time before outputting P_1, the time between outputting P_i and P_{i+1} for all $i \in [n-1]$, and the time after outputting P_n is bounded by a polynomial of the size of $\mathcal{D}$,

incremental polynomial time if P_1 is outputted with polynomial delay, the time between outputting P_i and P_{i+1} for all $i \in [n-1]$ is bounded by a polynomial of the combined size of $\mathcal{D}$ and the set $\{P_1, \ldots, P_i\}$, and the time after outputting P_n is bounded by a polynomial of the combined size of $\mathcal{D}$ and $\mathcal{O}$,

output polynomial time if $\mathcal{O}$ is outputted in time polynomial in the combined size of $\mathcal{D}$ and $\mathcal{O}$.

2.6 Graph Edit Distance

The graph edit distance (GED) is a dissimilarity measure between labeled graphs. Intuitively, it can be defined as the minimum effort necessary to transform one graph into another by a sequence of edit operations on the node and edge sets. In its most general definition, the GED is defined as the minimum cost edit path over all possible edit paths. For a detailed discussion on the GED, see e.g., Blumenthal (2019).

Definition 2.13 (Graph Edit Distance). *Let G, H be two labeled graphs over labels Σ and let $\perp \notin \Sigma$ be a special blank symbol. For $\Sigma_\perp = \Sigma \cup \{\perp\}$, we call a function*

$$c : \Sigma_\perp \times \Sigma_\perp \to \mathbb{R}_{\geqslant 0} \tag{2.1}$$

with $c(\alpha, \alpha) = 0$ for all $\alpha \in \Sigma_\perp$ an edit cost function *which assigns costs to edit operations. An edit operation (i) deletes, (ii) inserts or (iii) relabels a node or edge. Deleting and inserting a node or edge with label α has cost $c(\alpha, \perp)$ and $c(\perp, \alpha)$, respectively. Relabeling a node or edge with label α by label β has cost $c(\alpha, \beta)$. Vertices can only be deleted if they are isolated.*

An edit path *is a sequence of edit operations $P = (o_1, o_2, ..., o_k)$ that transform G into a graph isomorphic to H. The cost of an edit path P is defined as the sum of the costs of its edit operations, i.e., $c(P) = \sum_{i=1}^{k} c(o_i)$. Then, the* graph edit distance *between G and H is defined as the cost of a minimum cost edit path, i.e.,*

$$GED(G, H) = \min_{P \in \mathcal{P}(G,H)} c(P) \tag{2.2}$$

with $\mathcal{P}(G, H)$ being the set of all edit paths between G and H.

As there are infinitely many edit paths between two graphs, we generally consider only such paths which edit each node and edge at most once. These restricted edit paths can conveniently be expressed through so called *node maps*.

Definition 2.14 (Node Map). *For graphs G, H, a* node map *is a relation $\pi \subseteq V(G) \times V(H)$ which induces a set of edit paths that transform G into a graph isomorphic to H by specifying the edit operations for all nodes of G and H as follows:*

(i) vertices of G not appearing in π are deleted,

(ii) vertices of H not appearing in π are inserted, and

(iii) for all $(u, v) \in \pi$, node u is relabeled by the label of node v.

A node map further implies for all edges of G and H whether they are deleted, inserted or relabeled. Note that the number of node maps between G and H is finite and that an edit path of minimum cost is always induced by some node map. It is thus sufficient to consider only edit paths induced by node maps (Blumenthal, 2019).

2.6.1 Complexity of Computing GEDs

Computing the GED between two graphs is intractable in general. In fact, it can be shown that even in the case of uniform edit costs, the problem remains NP-hard (Blumenthal, 2019). Furthermore, there exists no α-approximation algorithm for the GED. More specifically, for graph G, H and $\alpha > 0$, there is no polynomial time algorithm that returns an edit path $P \in \mathcal{P}(G, H)$ with $c(P) \leqslant \alpha\,\mathrm{GED}(G, H)$ (Blumenthal, 2019).

The difficulties of computing the graph edit distance remain even for the case that the graph class is restricted to trees. While there exist polynomial time algorithms for ordered trees (Bille, 2005) or restricted variants such as the one introduced in Chapter 6, computing the tree edit distance between two unordered trees is a generally NP-hard problem (Zhang, Statman, and Shasha, 1992).

2.7 The Weisfeiler-Lehman Method

The *Weisfeiler-Lehman* (WL) method (Weisfeiler and Lehman, 1968) is a node relabeling method, which iteratively aggregates node neighborhood information by compressing the labels of each node and its neighbors into new labels. This is done by concatenating a node's label and the ordered multiset of its neighbors' labels. Subsequently this concatenated string is hashed to a new label by a perfect hash function. Thus, with each iteration, labels incorporate increasingly larger substructures. The injectivity of the hash function ensures that different sorted lists of labels cannot be mapped to the same (new) label.

More precisely, let $G = (V, E, \ell_0)$ be a graph with initial vertex label function $\ell_0 : V \to \Sigma_0$, where Σ_0 is the alphabet of the original vertex labels. In case of unlabeled graphs, we assume all vertices to have the same mutual label. Assuming that there is a total order on alphabet Σ_i for all $i \geqslant 0$, the Weisfeiler-Lehman algorithm recursively computes the new label of each $v \in V$ in iteration $i + 1$ by

$$\ell_{i+1}(v) = f_{\#}(\ell_i(v), [\ell_i(u) : u \in \mathcal{N}(v)]) \in \Sigma_{i+1} \tag{2.3}$$

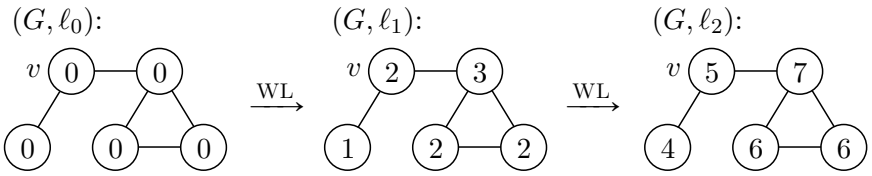


Figure 2.2: Visualization of graph G (under initial labeling ℓ_0) and its first two Weisfeiler-Lehman relabeled versions (G, ℓ_1), resp. (G, ℓ_2). Integers denote Weisfeiler-Lehman labels.

where the list of labels in the second argument of $f_{\#}$ is sorted by the total order on Σ_i and $f_{\#} : \Sigma_i \times \Sigma_i^* \to \Sigma_{i+1}$ is a perfect (i.e., injective) hash function. An example for the Weisfeiler-Lehman relabeling scheme is given in Fig. 2.2.

The Weisfeiler-Lehman method was originally designed for deciding isomorphism between graphs with one-sided error. For that purpose, graphs are first represented as the multisets of their relabeled node vertices and subsequently compared by equality. More specifically, for two graphs G and H, if in some iteration i the corresponding multisets $\{\{\ell_i(v) : v \in V(G)\}\}$ and $\{\{\ell_i(v) : v \in V(H)\}\}$ are different, then G and H are not isomorphic. Otherwise, if after n iterations (where $n = |V(G)| = |V(H)|$) the multisets are identical, G and H may or may not be isomorphic. However, it can be shown that the test is correct with high probability (Babai, Erdös, and Selkow, 1980). More precisely, the fraction of all size n graphs that are not uniquely characterized up to isomorphism by the Weisfeiler-Lehman method converges to 0 as n tends to infinity (Thm. 3.3 in Kiefer, 2020).

The above concept of the Weisfeiler-Lehman method can be generalized to a k-dimensional variant which iteratively recolors vertex k-tuples of a given graph (see e.g., Sect. 2.2. of Kiefer and Neuen, 2019). The ordinary Weisfeiler-Lehman algorithm (Weisfeiler and Lehman, 1968) is therefore also commonly known as the 1-dimensional Weisfeiler-Lehman method. Throughout this dissertation, when referring to Weisfeiler-Lehman, we mean the 1-dimensional version.

2.7.1 Unfolding Trees

An interesting aspect of the Weisfeiler-Lehman method is that it implicitly constructs rooted trees, called *unfolding trees* (Dell, Grohe, and Rattan, 2018). In fact, for $v \in V(G)$ the label $\ell_i(v)$ encodes a rooted tree of depth i which reflects v's i-hop neighborhood. Formally:

Definition 2.15 (Unfolding Tree). *For a graph G, vertex $v \in V(G)$, and $i \in \mathbb{N}$, the depth-i unfolding tree (or simply, i-unfolding tree), denoted $T_i(G, v)$, is a tree T with root $r \in V(T)$ such that all leaves have depth i, and there exists a homomorphism $\varphi : V(T) \to V(G)$ with*

(i) $\varphi(r) = v$, and

(ii) for all non-leaves in $t \in V(T)$, the homomorphism φ induces a bijection between the children of t in T and the neighbors of $\varphi(t)$ in G.

We refer to such a mapping φ as a *locally bijective homomorphism*. In terms of pattern matching, locally bijective homomorphisms can be regarded as the pattern embedding oper-

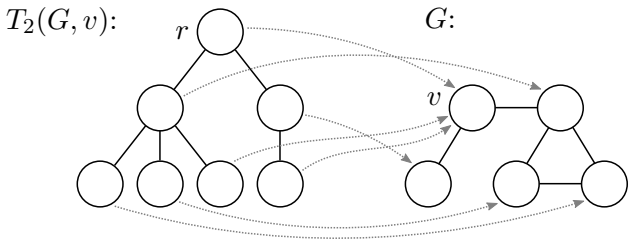


Figure 2.3: $T_2(G, v)$ is the depth-2 unfolding tree of graph G at vertex v. The corresponding locally bijective homomorphism from $T_2(G, v)$ into G is depicted in gray.

ator on Weisfeiler-Lehman unfolding trees. Figure 2.3 provides an example of an unfolding tree and the corresponding locally bijective homomorphism.

Note that there is a one-to-one correspondence between the labels in Σ_i and the set of (non-isomorphic) i-unfolding trees. We will, therefore, often use the notions of Weisfeiler-Lehman labels and unfolding trees interchangeably.

2.8 Kernels and Support Vector Machines

Before discussing graph kernels in Sect. 2.9, we first provide a quick overview of kernel functions in general. For a more detailed discussion on kernel methods, we refer to, e.g., Schölkopf and Smola (2002).

2.8.1 Kernel Functions

A *positive semi-definite kernel* defines a similarity measure between pairs of objects. Formally:

Definition 2.16 ((Positive Semi-definite) Kernel). *Let $\mathcal{X}$ be a non-empty set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* positive semi-definite kernel *if and only if*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geqslant 0 \tag{2.4}$$

holds for any $n \in \mathbb{N}$, any choice $x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$.

In this thesis, we often refer to positive semi-definite kernels as *proper kernels* or just simply *kernels*. A most interesting aspect of kernels is that they correspond to dot products in (potentially unknown) spaces.

Theorem 2.1. *For any kernel k on a set $\mathcal{X}$, there exists a Hilbert space $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \to \mathcal{H}$ such that for all $x, y \in \mathcal{X}$ it holds that*

$$k(x, y) = \langle \phi(x), \phi(y) \rangle. \tag{2.5}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathcal{H}$.

We usually refer to $\phi(x)$ as the *embedding* of x and to $\mathcal{H}$ as the *embedding space* or *feature space*. The above theorem also implies that kernels can directly be defined by dot products of explicit embeddings. That is, an embedding space $\mathcal{H}$ and corresponding embedding function $\phi : \mathcal{X} \to \mathcal{H}$ automatically yield a kernel $k(x, y) = \langle \phi(x), \phi(y) \rangle$.

Kernel functions are closed under several operations, which allows to combine kernel functions into more complex ones. More specifically, for any two kernels k_1, k_2 on the set $\mathcal{X}$ and for all $a \geqslant 0$,

(i) $k(x, y) = a \cdot k_1(x, y)$ is a kernel, *(scalar multiplication)*

(ii) $k(x, y) = k_1(x, y) + k_2(x, y)$ is a kernel, and *(addition)*

(iii) $k(x, y) = k_1(x, y) \cdot k_2(x, y)$ is a kernel. *(multiplication)*

A very rudimentary but substantial kernel is the *Dirac delta* which yields 1 if the inputs are equal and 0 otherwise, i.e.,

$$\delta(x, y) = \begin{cases} 1 & x = y \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The Dirac delta function plays a particularly important role in graph kernel design as it often serves as the comparison operator on graph features.

Positive semi-definite kernels can also be derived from certain distance functions. More precisely, for a non-empty set $\mathcal{X}$ with $x, y \in \mathcal{X}$, and parameter $\gamma > 0$, the kernel

$$k(x, y) = e^{-\gamma d(x,y)} \tag{2.7}$$

is positive semi-definite if and only if the function d is *(conditionally) negative definite* (Schoenberg, 1938).

Definition 2.17 ((Conditionally) Negative Definite Kernel). *Let $\mathcal{X}$ be a non-empty set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a* (conditionally) negative definite kernel *if and only if*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(x_i, x_j) \leqslant 0 \tag{2.8}$$

holds for any $n \geqslant 2$, any choice $x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$ with $\sum_{i=1}^{n} c_i = 0$ (Berg, Christensen, and Ressel, 1984).

We refer to the kind of kernel functions in Eq. 2.7 equipped with (conditionally) negative definite kernels d as *radial basis function kernels*.

2.8.2 Support Vector Machines

Support Vector Machines (SVM) (Boser, Guyon, and Vapnik, 1992) are supervised learning models that are primarily used for classification purposes. Their task is to train a model based on provided data in order to predict the class label of prior unseen data points. More

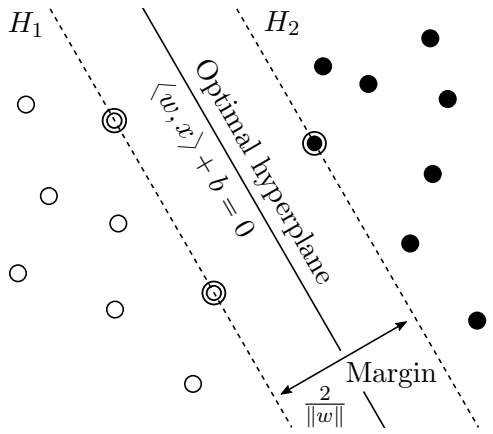


Figure 2.4: SVMs find optimal hyperplanes which maximize the margin between the two classes. The optimal hyperplane lies halfway between the boundary hyperplanes H_1 and H_2. The points lying on these boundary hyperplanes are referred to as *support vectors*.

specifically, given training data $(x_i, y_i), i \in [n]$ with $x_i \in \mathcal{H}$ and corresponding binary valued classes $y_i \in \{-1, 1\}$, the SVM learns a function $f : \mathcal{H} \to \{-1, 1\}$ which classifies new data points. The general approach of SVMs is to find a hyperplane in $\mathcal{H}$ which "best" separates the data points w.r.t. to their classes. Subsequently, the class of a novel data point can be predicted by simply checking on which side of the hyperplane the point lies.

Assuming that the data points are linearly separable, there clearly exist infinitely many such separating hyperplanes. In order to find the most suitable one, the SVM selects the hyperplane which maximizes the distance to the nearest data points on either side. This unique hyperplane corresponds to points x on a line satisfying $\langle w, x \rangle + b = 0$, and is referred to as the *optimal hyperplane*. The optimal hyperplane lies halfway in between two parallel boundary hyperplanes which mark the borders of the two classes. The space between these two boundary hyperplanes is known as the *margin* and the distance between the two is $2/\|w\|_2$. To ensure that all data points lie on the correct side of the margin, the constraint

$$y_i(\langle w, x_i \rangle + b) \geqslant 1 \ , \forall i \in [n] \tag{2.9}$$

is introduced. A visualization of the concept of SVMs can be found in Fig. 2.4. In order to find such two boundary hyperplanes which yield a maximum margin, the goal is to minimize $\|w\|_2^2$, while simultaneously guaranteeing the constraint set of Eq. 2.9. The corresponding optimization problem can then be formulated as follows:

$$\min_{w,b} \frac{1}{2} \|w\|_2^2 \quad \text{s.t.} \quad y_i(\langle w, x_i \rangle + b) \geqslant 1 \ , \forall i \in [n] \tag{2.10}$$

Eq. 2.10 can be solved using the Lagrange multiplier method. More specifically, a positive value $\alpha_i, i \in [n]$ is introduced for each constraint of Eq. 2.9. Finally, the above problem can

be reformulated leading to the *dual problem*:

$$\begin{aligned} \max_{\alpha_1,\ldots,\alpha_n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \alpha_i \geqslant 0, i \in [n]. \end{aligned} \tag{2.11}$$

The resulting hyperplane is then given by

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \ . \tag{2.12}$$

For a more detailed description of this reformulation, see, e.g., Schölkopf and Smola (2002).

2.8.3 Soft-margin SVMs

Up until now it was naively assumed that the data is in fact linearly separable. Clearly, this is not the case for most provided data. In order to handle non-separable cases nonetheless, the constraints of Eq. 2.9 can be relaxed by introducing slack variables $\zeta_i \geqslant 0$, $i \in [n]$ yielding the inequality set

$$y_i(\langle w, x_i \rangle + b) \geqslant 1 - \zeta_i \ , \forall i \in [n] \tag{2.13}$$

The slack variables provide a certain error tolerance when choosing the separating hyperplane. That is, data points may now fall into the margin or even onto the wrong side of the hyperplane. The corresponding optimization problem is formalized as follows:

$$\min_{w,b,\zeta} \frac{1}{2} \|w\|_2^2 + \frac{C}{n} \sum_{i=1}^{n} \zeta_i \quad \text{s.t.} \quad y_i(\langle w, x_i \rangle + b) \geqslant 1 - \zeta_i \ , \forall i \in [n] \tag{2.14}$$

The value C is a positive parameter which governs the trade-off between maximizing the margin and minimizing the sum of distances of misclassified data points to the hyperplane. The identification of a suitable choice of C is usually part of the machine learning process.

The dual problem of this relaxed variant is almost identical to that of Eq. 2.11 since the slack variables and their Lagrange multipliers vanish in this formulation. The only difference is that the α_is are now upper bounded by C, i.e., it needs to hold that $\frac{C}{n} \geqslant \alpha_i \geqslant 0, i \in [n]$.

2.8.4 The Kernel Trick

While the soft-margin SVM relaxes the rigid assumption that the data must be linearly separable, it is not a solution to all problems. An alternative approach is to transform the input data into higher dimensional spaces, in which they become separable.

A most interesting aspect of the dual formulations above is that they compute the optimal hyperplane using dot products between data points. A dot product $\langle x, y \rangle$ with $x, y \in \mathcal{X}$ can easily be replaced by $\langle \phi(x), \phi(y) \rangle$, where $\phi : \mathcal{X} \to \mathcal{H}$ is a function which transforms the input data points into some other, more suitable feature space. The goal is to find a function ϕ which

then allows to separate the data in the corresponding feature space $\mathcal{H}$. However, finding such a function is generally computationally infeasible. Fortunately, there is no need to explicitly calculate this mapping. Instead, it suffices to compute the kernel $k(x, y) = \langle \phi(x), \phi(y) \rangle$. This is known as the *kernel trick*.

Putting everything together, we obtain the following dual formulation for computing the optimal separating hyperplane using kernel function k over $\mathcal{X}$.

$$\begin{aligned} \max_{\alpha_1,\ldots,\alpha_n} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \frac{C}{n} \geqslant \alpha_i \geqslant 0, i \in [n]. \end{aligned} \tag{2.15}$$

Finally, the class prediction of a prior unseen data point x is obtained by the function

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i k(x, x_i) + b \right). \tag{2.16}$$

2.9 Graph Kernels

An interesting aspect about kernel methods is that they allow the application of efficient learning methods such as support vector machines on graph structured data. That is, in contrast to other standard learning methods, which generally require real-vectored data, kernel methods can directly be applied to graphs. For a detailed survey on graph kernels, we refer to Borgwardt et al. (2020) and Kriege, Johansson, and Morris (2020).

2.9.1 R-convolution Kernels

With Haussler's work on *convolution kernels* over discrete structures (Haussler, 1999), kernel methods became widely applicable to graphs. The concept of R-convolution kernels provides a general framework, which can be used to construct graph kernels by defining graph similarity in terms of aggregated substructure similarities.

Definition 2.18 (R-convolution Kernel (simplified)). *Let $\mathcal{G}$ be a set of graphs, $\mathcal{X}$ be a set of substructures, and $R : \mathcal{G} \to \mathbb{N}^{\mathcal{X}}$ be a function that decomposes graphs into multisets of substructures. Then, for $G, H \in \mathcal{G}$, the R-convolution kernel is defined by*

$$k(G, H) = \sum_{x \in R(G)} \sum_{y \in R(H)} \kappa(x, y) \tag{2.17}$$

where $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel on the substructures.

In many graph kernels, the kernel κ of Eq. 2.17 is simply the Dirac delta function, which amounts to 1 if the substructures x and y are equivalent and 0 otherwise. Thus, R-convolution kernels essentially measure graph similarity by counting pairs of equivalent substructures. Such kernels can alternatively be reformulated as substructure frequency products. We refer to this type of kernel as *histogram kernel*.

Definition 2.19 (Histogram Kernel). *Let $\mathcal{G}$ be a set of graphs and $\mathcal{X}$ be a set of substructures. Then, for $G, H \in \mathcal{G}$, the* histogram kernel *over $\mathcal{X}$ is defined by*

$$k(G, H) = \sum_{x \in \mathcal{X}} c_x(G)\ c_x(H) \tag{2.18}$$

where $c_x(G)$ indicates the occurrence count of substructure $x \in \mathcal{X}$ in graph G.

2.9.2 Graph Kernel Expressivity

In the context of graph kernels, the *expressive power* measures the ability of graph representations to distinguish non-isomorphic graphs. Formally, for graph embedding functions $\phi_1 : \mathcal{G} \to \mathcal{H}_1$ and $\phi_2 : \mathcal{G} \to \mathcal{H}_2$, the function ϕ_1 is said to be "more expressive" than ϕ_2 if for any two graphs $G, H \in \mathcal{G}$ it holds that $\phi_1(G) = \phi_1(H) \Rightarrow \phi_2(G) = \phi_2(H)$ and there exist non-isomorphic graphs $G', H' \in \mathcal{G}$ such that $\phi_1(G') \neq \phi_1(H')$ and $\phi_2(G') = \phi_2(H')$.

The expressive power of graph kernels was first addressed by Gärtner, Flach, and Wrobel (2003). The authors refer to graph kernels, which are capable of distinguishing all pairs of non-isomorphic graphs, as *complete*. More formally:

Definition 2.20 (Complete Graph Kernel). *A kernel $k(\cdot, \cdot)$ is called* complete *if its embedding function ϕ with $k(G, H) = \langle \phi(G), \phi(H) \rangle$ satisfies $\phi(G) = \phi(H)$ if and only if G, H are isomorphic, for all $G, H \in \mathcal{G}$.*

As there is no known polynomial algorithm for deciding graph isomorphism, complete graph kernels are generally infeasible in practice.

2.10 The Wasserstein Distance

The Wasserstein distance (Kantorovich, 1960) is a distance function between probability distributions based on the concept of optimal mass transportation. Intuitively, the Wasserstein distance can be viewed as the minimum cost necessary to transform one pile of earth into another. It is, therefore, also known as the *earth movers* distance or *optimal transport* distance.

While Wasserstein distances are sometimes defined more generally, we specifically consider the 1-Wasserstein distance for discrete distributions. More precisely, we restrict our attention to the case that distributions correspond to histograms and the ground cost is equal to a distance. We refer to it as simply the Wasserstein distance. For more general definitions see, e.g., Peyré and Cuturi (2019).

Definition 2.21 (Wasserstein Distance). *Given two vectors $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ with $\|x\|_1 = \|y\|_1$ and a cost matrix $C \in \mathbb{R}^{n \times m}$ containing pairwise distances between entries of x and y, the* Wasserstein distance *is defined by*

$$\mathcal{W}_C(x, y) = \min_{T \in \mathcal{T}(x,y)} \langle T, C \rangle \tag{2.19}$$

with $\mathcal{T}(x, y) \subseteq \mathbb{R}^{n \times m}$ and $T\mathbb{1}_m = x$, $T^\top \mathbb{1}_n = y$ for all $T \in \mathcal{T}(x, y)$, where $\langle \cdot, \cdot \rangle$ is the Frobenius inner product, and $\mathbb{1}_n$ is the one-vector of length n.

Matrix $T \in \mathcal{T}(x, y)$ is called *transport matrix* and a minimizer of Eq. 2.19 is called an *optimal* transport matrix. The costs C between entries of x and y can alternatively be provided in form of a distance function d. We refer to this function as the *ground distance* and write $\mathcal{W}_d$ instead of $\mathcal{W}_C$. If the ground distance is a metric, then the Wasserstein distance is a metric (Peyré and Cuturi, 2019, Sect. 2.4).

In general, the above Wasserstein distance can be computed in time cubic in the size of the vectors using, e.g., the Hungarian method (Kuhn, 1955). However, in case the ground distance is given by the distance on the real line, the resulting Wasserstein distance can be calculated in *linear* time. More precisely, let $x, y \in \mathbb{R}^n$ and let $\bar{x}_i \in \mathbb{R}$ denote the i-th entry of x. Furthermore, let there be a sequence of non-negative, increasing (or alternatively decreasing) values $w_1, w_2, \ldots, w_n$, on which $d^1(w_i, w_j) = |w_i - w_j|$ with $i, j \in [n]$ defines the ground distance. Then, the Wasserstein distance equipped with ground distance d^1 has the following closed form:

$$\mathcal{W}_{d^1}(x, y) = \sum_{i=1}^{n-1} |w_{i+1} - w_i| \left| \sum_{j=1}^{i} \bar{x}_j - \sum_{j=1}^{i} \bar{y}_j \right| \tag{2.20}$$

It can be shown that the distance function of Eq. 2.20 on histograms is a *(conditionally) negative definite* kernel. In fact, this even holds when the ground distance is defined by the shortest-path metric on a tree (Le et al., 2019).

Finally, we can define centers for sets of vectors w.r.t. the Wasserstein distance (Agueh and Carlier, 2011). That is:

Definition 2.22 (Wasserstein Barycenter). *For a set of vectors $x_1, ..., x_k \in \mathbb{R}^n$ and cost matrix $C \in \mathbb{R}^{n \times n}$, we define the* barycenter *as*

$$\operatorname*{argmin}_{a \in \mathbb{R}^n} \sum_{i=1}^{k} \mathcal{W}_C(x_i, a). \tag{2.21}$$

In other words, the barycenter minimizes the sum of Wasserstein distances to the elements in a set of histograms.

2.11 Datasets

The experimental evaluations in this thesis are conducted on a mutual set of real-world benchmark datasets. The datasets range from molecular datasets to large network datasets and are selected with regard to several purposes. For one, they are chosen as a representative sample of well-established benchmark datasets. Furthermore, they contain graphs of varying structural properties. That is, the number of graphs, their average size and the connectivity significantly differs for different datasets. It is noteworthy that for a multitude of commonly used datasets (see, e.g., Morris et al., 2020), very rudimentary methods suffice to be on par with the overall best methods. We therefore selected those datasets which are most suitable for proper graph classification benchmarking purposes.

Molecular graphs represent chemical compounds where node labels correspond to the atom type and edges represent bonds. Due to chemical properties, molecular graphs are of

fairly simple structure. That is, for instance, atoms have small bounded degrees and common subgraphs correspond to functional groups. Additionally, molecular graphs tend to exhibit a tree-like structure, with very few cycles present. In contrast, the large network dataset graphs are more diverse and exhibit a larger degree of noise. They are extracted from densely connected (social) networks and generally have an edge-to-node ratio significantly above $1 : 1$.

In the following, we briefly outline the datasets considered in this dissertation:

DHFR (Sutherland, O'Brien, and Weaver, 2003) is a molecular dataset containing 756 graphs which represent inhibitors of *dihydrofolate reductase*, a process that converts *dihydrofolate* to *tetrahydrofolate*. Each compound is labeled by a binary value according to its inhibitory potential.

MUTAG (Debnath et al., 1991) contains 188 compounds with molecules labeled according to their mutagenic behavior towards the bacteria *salmonella typhimurium*.

NCI1 (Wale and Karypis, 2006) contains 4110 molecular compounds which are classified by whether or not they inhibit non-small cell lung cancer cell line growth.

PTC-MR (Helma et al., 2001) consists of a total of 344 molecular graphs, each labeled by whether or not the compound causes cancer in rats or mice.

IMDB-BINARY (Yanardag and Vishwanathan, 2015) contains 1000 graphs extracted from movie collaboration networks. Each graph represents an ego-network with nodes corresponding to actors and edges indicating that two actors appeared in the same movie. An ego-network is labeled by whether it was extracted from an *action* or *romance* genre collaboration network.

EGO-X are four different real-world benchmark datasets introduced by the author of this thesis. They are extracted from the social networks Buzznet, Digg, Flickr, and LiveJournal. An EGO-X dataset consists of 200 ego-network graphs sampled from the four social networks. Ego-networks are subgraphs induced by a central vertex's neighbors. Each graph is annotated against the social network it was extracted from such that the learning task is to assign an ego-network to the network it belongs to. Graphs within each dataset were randomly chosen from the set of all ego-networks but underlie size- and density-specific constraints to ensure that the graph classification task is non-trivial. The EGO-X datasets contain increasingly larger and more dense ego networks with growing index X.

A detailed summarization of all graph datasets and their structural properties can be found in Table 2.1. All datasets except for EGO-X were provided by Morris et al. (2020).

Dataset	$\|D\|$	$\|C\|$	$\varnothing\|V\|$	$\varnothing\|E\|$	$\varnothing\frac{\|E\|}{\|V\|}$	Δ	$\|\Sigma\|$
DHFR	756	2	42.43 ± 9.06	44.54 ± 9.25	1.05	4	9
MUTAG	188	2	17.93 ± 4.58	19.79 ± 5.68	1.09	4	7
NCI1	4110	2	29.87 ± 13.56	32.3 ± 14.93	1.08	4	37
PTC_MR	344	2	14.29 ± 9.02	14.69 ± 10.05	0.99	4	18
IMDB-BINARY	1000	2	19.77 ± 10.06	96.53 ± 105.6	4.44	135	–
EGO-1	200	4	138.96 ± 5.92	593.53 ± 147.88	4.27	140	–
EGO-2	200	4	178.55 ± 5.92	1444.86 ± 204.65	8.09	180	–
EGO-3	200	4	220.0 ± 6.35	2613.49 ± 275.51	11.88	203	–
EGO-4	200	4	259.77 ± 6.04	4135.8 ± 302.15	15.93	237	–

Table 2.1: Structural information of graph datasets providing the dataset size $|D|$, the number of classes $|C|$, average vertex number $|V|$, average edge number $|E|$, maximum vertex degree Δ, and amount of distinct vertex labels $|\Sigma|$.

3

Related Work

This section explores a body of literature that is relevant to this dissertation. We particularly cover the essential research fields which are connected through this thesis and put our contributions into context. We furthermore discuss state-of-the-art methods which our approaches are compared to and address some orthogonal research topics. Given the abundance of research in each field, the following sections are not intended to be exhaustive.

As one of the central notions of this thesis, we cover the essential literature concerned with *constrained homomorphisms* in Sect. 3.1. We explore the most common types of constrained homomorphisms and briefly address how partially injective homomorphisms and locally bijective homomorphisms fit in. Given the utilization of an explicit pattern mining algorithm in Chapter 4, we review principles of *frequent subgraph mining* in Sect. 3.2. We provide some background information to this extensively studied research field and discuss special instances of the problem. Particularly, we cover *frequent subtree mining* and address literature that is concerned with employing homomorphism as the pattern matching operator. Sect. 3.3 establishes a connection between several central notions of this thesis. While the discussed literature is not of immediate relevance to our contributions, we argue that it nonetheless offers a most interesting tangent. Next, Sect. 3.4 provides a brief overview of *graph kernel methods*. We particularly summarize a range of well-established as well as closely related graph kernels, with an emphasis on graph kernel methods utilizing Weisfeiler-Lehman features. Finally, we briefly discuss *graph neural networks* in Sect. 3.5 due to their close relationship to the Weisfeiler-Lehman method as well as their relevance to the results presented in Chapter 5.

3.1 Constrained Homomorphisms

Graph homomorphisms are a fundamental and well-studied topic in graph theory (Hell and Nesetril, 2004). They are most famously applied in graph colorings. More precisely, the problem of deciding whether a map can be colored with n colors such that no two adjacent regions share the same color can effectively be formulated as a graph homomorphism problem (see, e.g., Lewis, 2016).

A particularly relevant aspect of graph homomorphisms is that they can be regarded as the

most general structure-preserving mappings between graphs, in a sense that other common graph relations can be expressed as *constrained* variants of them. These so called *constraint graph homomorphisms* are homomorphisms which additionally impose further restrictions on the mappings. Constrained homomorphisms have been extensively studied (see Fiala and Kratochvíl, 2008 for a survey) and remain an active research topic to this day. While there exists an abundance of different definitions of constrained homomorphisms, they are often divided into *global* and *local* ones (Long, 2014).

3.1.1 Globally Constrained Homomorphisms

Globally constrained homomorphisms are homomorphisms which satisfy additional conditions that may apply to *any* pairs on nodes. One example that becomes particularly relevant in Chapter 4 is that of *injective* homomorphisms, which corresponds to the concept of *subgraph isomorphisms*. Another common notion is that of graph *isomorphisms* which are simply *bijective* homomorphisms that preserve edges and labels in both directions. Subgraph isomorphism and graph isomorphism are arguably the most established notions of subgraph relation respectively graph equivalence. While *surjective* homomorphisms have gained comparably less attention, they are nonetheless focus of ongoing research (see e.g., Focke, Goldberg, and Zivný, 2019).

Most (constrained) homomorphisms are notoriously hard (i.e., NP-complete) to decide, but become efficiently decidable for certain restrictions on the pattern and target graphs. For instance, even deciding ordinary homomorphism is an NP-complete problem. However, a key result which we utilize in Chapter 4 is that homomorphism from a pattern H into a target graph G is polynomial-time decidable for arbitrary graphs G if H has bounded treewidth (Dalmau, Kolaitis, and Vardi, 2002). In fact, Grohe (2007) showed an even stronger result by proving that homomorphisms can be efficiently decided *if and only if* H has bounded treewidth.

The problem of deciding subgraph isomorphism is a generally even more complex problem. For instance, subgraph isomorphism cannot even be efficiently decided for the case that the pattern graphs are restricted to trees. In fact, due to the NP-completeness of the Hamiltonian path problem, this even holds for paths. However, the subgraph isomorphism problem becomes P-time solvable if H and G are both trees (Matula, 1978). A comprehensive summary covering the complexity of subgraph isomorphisms for multiple graph classes for H and G can be found in Marx and Pilipczuk (2014).

Finally, deciding graph isomorphism belongs to one of the few problems which are neither known to lie in P nor known to be NP-complete (Garey and Johnson, 1979). Nonetheless, Babai (2016) has shown in groundbreaking work that the problem is in fact solvable in quasi-polynomial time.

In Chapter 4, we introduce the notion of partially injective homomorphisms, which can be classified as relaxed globally constrained homomorphisms. Roth (2021) introduces this notion independently, focusing on the complexity of counting constrained homomorphisms. Equivalently to our definition, Roth (2021) defines partially injective homomorphism as a homomorphism $\varphi : V(H) \to V(G)$ which additionally respects a set of vertex pair inequalities I. More precisely for every $uv \in I$, it is required that $\varphi(u) \neq \varphi(v)$.

3.1.2 Locally Constrained Homomorphisms

Locally constrained homomorphisms are homomorphisms that additionally uphold conditions that are restricted to direct node neighborhoods. Specifically, a homomorphism $\varphi : V(H) \to V(G)$ is *locally injective, surjective,* or *bijective* if for all $v \in V(H)$, the restriction of φ to the neighborhood of v is locally injective, surjective, resp. bijective. Locally constrained homomorphisms are an extensively studied field in graph theory. We refer to the survey article by Fiala and Kratochvíl (2008) for a comprehensive discussion on this kind of pattern matching operator. We note that, in this dissertation, locally bijective homomorphisms play an important role, as they correspond to the matching operator of Weisfeiler-Lehman unfolding trees (see Sect. 2.7.1).

Deciding locally injective, surjective, and bijective homomorphisms are generally all NP-complete problems. That is, for arbitrary graphs H and G, there exists no P-time algorithm for deciding this kind of constrained homomorphisms. However, when H is a tree, all three problems become decidable in polynomial time (Chaplick et al., 2015). It was furthermore shown that locally injective, surjective, and bijective homomorphisms become polynomial-time solvable if the pattern graph H has bounded treewidth and H or G have bounded maximum degree (Chaplick et al., 2015).

3.2 Frequent Subgraph Mining

Frequent subgraph mining (FSM) is a fundamental task in the field of data mining, which has gained significant attention over the last 30 years. Its task is to enumerate a set of pattern graphs which are subgraph isomorphic to at least a predefined number of dataset graphs. FSM is widely used in various domains such as bioinformatics (Mrzic et al., 2018), chemical compound analysis (Nijssen and Kok, 2004), and web mining (see e.g., Getoor and Diehl, 2005). FSM is generally restricted to enumerating *connected* pattern graphs and is therefore also often referred to as the frequent connected subgraph mining problem. For a comprehensive survey on FSM, we refer to Jiang, Coenen, and Zito (2013).

The general concept of FSM is to generate potential patterns using an enumeration method and determine if these patterns are frequent within a given dataset. Most practical approaches for frequent subgraph mining therefore primarily focus on suitable enumeration methods which reduce redundancy in the output and on minimizing the number of occurrence counting operations. These two aspects are mainly addressed through different techniques for traversing the pattern search space and generating new pattern candidates.

Methods for traversing the pattern search space follow either a breath-first search (see, e.g., Kuramochi and Karypis, 2004) or a depth-first search (see, e.g., Yan and Han, 2002) approach. Both variants iteratively extend frequent pattern graphs by either a single edge or a vertex-edge pair using a *refinement operator*. Let ρ be such a refinement operator.

The breath-first search, also referred to as *levelwise search*, enumerates all frequent patterns of size i before continuing with size $i+1$ patterns. This allows to make use of the Apriori principle (Agrawal and Srikant, 1994), which was originally designed for the enumeration of frequent itemsets. The Apriori principle states that a pattern can only be frequent if all of its subpatterns are frequent as well. Therefore a candidate pattern H only needs to be tested for

frequency if all its immediate subgraphs $\{H' : \rho(H') = H\}$ have been evaluated and shown to be frequent.

A *depth-first search* pattern enumeration algorithm iterates over the pattern space by recursively computing the set of refinement graphs $\rho(H)$ of a frequent pattern H and continuing with a refinement $H' \in \rho(H)$ if it has not yet been seen. To avoid duplicates, a set $\mathcal{C}$ containing visited patterns may be maintained such that candidate patterns are dismissed if they have already been considered. However, checking whether $H \in \mathcal{C}$ for a pattern H essentially requires solving graph isomorphism instances. While there exist practical isomorphism testing variants such as *nauty* (McKay and Piperno, 2014), most approaches address the problem by sophisticated pattern generation strategies. A common method is to design refinement operators which avoid duplicates in the pattern enumeration process. For instance, Nijssen and Kok (2005) define an order, in which new vertices or edges can be added, leading to a pattern candidate generation process which (ideally) allows each pattern to be generated only through a unique sequence of iterative refinement operations.

3.2.1 Frequent Subtree Mining

A commonly considered special case of general frequent subgraph mining is the restriction of the pattern class to trees. This task is referred to as *frequent subtree mining*. Unfortunately, this restriction to the pattern class does not significantly relax the complexity of the mining task as subtree isomorphism remains an NP-complete problem. Furthermore, the number of potential pattern graphs may still grow exponentially. However, due to the tree isomorphism problem being efficiently solvable, the redundancy issue of the output set can be properly addressed. Often times, a duplicate free enumeration is achieved using canonical tree representations (see, e.g., Chi, Yang, and Muntz, 2003). By utilizing canonical representations, duplicate subtrees can be efficiently eliminated, improving the overall efficiency of frequent subtree mining.

In an effort to overcome the inefficiency of the subtree isomorphism test, several subtree miners have emerged which restrict the class of target graphs to forests (see, e.g., Chi, Yang, and Muntz, 2003). In fact, subtree isomorphism in forests can be decided in polynomial time (Matula, 1968). As an approximation to frequent subtree mining in arbitrary target graphs, Welke, Horváth, and Wrobel (2017) make use of this result and effectively replace graphs by random spanning trees, leading to an incomplete, but efficient subtree enumeration method.

While subgraph isomorphism is the most commonly used pattern matching operator in graph mining, there has been some isolated research that explores the use of homomorphism as the pattern matching operator. One example is the work by Dries and Nijssen (2012), which focuses on enumerating frequent trees using homomorphism. The advantage of using homomorphism is that it is polynomial-time decidable from trees, allowing for an efficient pattern enumeration. However, the use of homomorphism has certain semantic challenges. Dries and Nijssen (2012) identify two essential issues in their mining algorithm which we also encounter in Chapter 4. Firstly, two different (i.e., non-isomorphic) pattern trees may nonetheless be homomorphism equivalent, effectively leading to duplicates in the output set. Secondly, patterns of even unbounded size can be frequent, causing the output set to be of potentially infinite size. To address these issues, the authors propose a specifically designed

pattern refinement method and discuss certain restrictions to the size and labels of patterns.

3.3 Homomorphisms and Weisfeiler-Lehman

In this section, we briefly digress from literature directly related to this dissertation and explore a tangent topic discussed in Dell, Grohe, and Rattan (2018). This article shows that graph homomorphisms and the Weisfeiler-Lehman method are closely related concepts. Even more so, the article merges several central notions of this dissertation such as Weisfeiler-Lehman graph representations, their expressivity, homomorphism counts, and graphs of bounded treewidth.

In his seminal work, Lovász (1967) proves that a graph can be characterized up to isomorphism using homomorphism counts. More precisely, two graphs G and G' are isomorphic if and only if for all graphs H, the number $\#\textsc{Hom}(H, G)$ of homomorphisms from H to G and the number $\#\textsc{Hom}(H, G')$ of homomorphisms from H to G' are identical. Considering that counting graph homomorphisms is NP-complete in general, Dell, Grohe, and Rattan (2018) focus on tractable subproblems. By restricting the set of pattern graphs to trees, they show that the corresponding characterization of graphs is equivalent to that of the 1-dimensional Weisfeiler-Lehman scheme. More specifically, it is shown that two graphs G and G' cannot be distinguished by the 1-dimensional Weisfeiler-Lehman method if and only if for all *trees* T, it holds that $\#\textsc{Hom}(T, G) = \#\textsc{Hom}(T, G')$. Consequently, the problem of whether or not there exists a tree T with $\#\textsc{Hom}(T, G) \neq \#\textsc{Hom}(T, G')$ can be efficiently decided using the Weisfeiler-Lehman test of isomorphism.

Dell, Grohe, and Rattan (2018) furthermore generalize this result to the class of bounded treewidth graphs by relating it to the k-dimensional Weisfeiler-Lehman test of isomorphism. The k-dimensional Weisfeiler-Lehman method is a generalization of the original variant by Weisfeiler and Lehman (1968) which iteratively recolors vertex k-tuples of a given graph (see, e.g., Sect. 2.2. in Kiefer and Neuen, 2019). It is shown that this k-dimensional variant can distinguish two graphs G and G' if and only if for all graphs H of *treewidth* at most k, it holds that $\#\textsc{Hom}(H, G) = \#\textsc{Hom}(H, G')$. Recall that homomorphisms into arbitrary target graphs can be efficiently decided if and only if the source graph has bounded treewidth (Grohe, 2007). Thus, the homomorphism vectors restricted to the class of tractable graph homomorphism problems can directly be linked to the concept of isomorphism tests using vertex recoloring schemes defined by the (k-dimensional) Weisfeiler-Lehman method.

3.4 Graph Kernels

Graph kernels have been the most established approach for graph classification tasks and have gained significant attention during the last 20 years. They have proven to be powerful methods for comparing graphs in terms of their structural similarities. The concept of graph kernels is rooted in the idea of transforming graphs into high-dimensional vector spaces, enabling the application of standard machine learning methods such as support vector machines. Graph kernels are applied in domains such as bioinformatics, chemoinformatics and social network analysis.

With Haussler's seminal work (Haussler, 1999) on convolution kernels over discrete structures, kernel methods became widely applicable to graph structured data. Using this framework for kernel functions, graph kernels can be designed by decomposing graphs into sets of substructures which are subsequently compared. A majority of popular graph kernels is in fact based in this kind of concept. The kernels in this category mainly differ in the kind of substructures which they consider. Thus, the bulk of the R-convolution-based graph kernel design is primarily concerned with feature engineering. Clearly, the type of feature, i.e., the kind of substructure, essentially governs the structural similarity which the kernel measures.

Graph kernels are often distinguished into *explicitly* and *implicitly* computable kernels. Let $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ be a kernel. Then, in the explicit case, the corresponding mapping ϕ is finite and known. Consequently, for two graphs $G, G' \in \mathcal{G}$, the kernel value $k(G, G')$ can simply be computed by $\langle \phi(G), \phi(G') \rangle$. If, otherwise, the mapping ϕ is infinite or unknown, the kernel is called implicit and needs to be computed through the function k. However, according to Kriege et al. (2019), the explicit variant is to be preferred in terms of computation speed and memory consumption.

Gärtner, Flach, and Wrobel (2003) introduced the notion of *complete* graph kernels as a measure of expressiveness. A complete graph kernel k has a corresponding embedding function ϕ which characterizes a graph up to isomorphism. However, computing complete graph kernels is at least as hard as solving graph isomorphism. Since the isomorphism problem is not known to be solvable in polynomial time, complete graph kernels are generally of no practical relevance. Nevertheless, it is a desirable property of graph kernels to distinguish as many pairs of non-isomorphic graphs as possible.

For a comprehensive overview of graph kernels, we refer to the survey by Kriege, Johansson, and Morris (2020) or the book by Borgwardt et al. (2020). Both works provide excellent insights into the theoretical aspects and empirical evaluations of a large body of graph kernel literature.

In the following, we provide a brief review of various graph kernels that are relevant to this dissertation. Due to the rich literature of this line of research, the collection is not meant to be exhaustive. Instead, we discuss a choice of well-established as well as conceptually related graph kernels. We start by introducing a very rudimentary kernel and continue with approaches based on comparing walks, paths and small subgraphs. Subsequently, we cover graph kernels based on tree patterns, with a specific focus on approaches that utilize the Weisfeiler-Lehman labels.

The Label Histogram Kernel The label histogram kernel is unique among all considered graph kernels, as it deliberately disregards all structural information when comparing graphs. This specific kernel variant is primarily meant as a baseline function, which indicates whether more sophisticated graph kernels are of any benefit. It corresponds to the simple R-convolution kernel, where the substructures are nodes and edges. More precisely, for two graphs $G, G' \in \mathcal{G}$, we define the label histogram kernel as follows:

$$k(G, G') = \sum_{v \in V(G)} \sum_{v' \in V(G')} \delta(\ell(v), \ell(v')) + \sum_{e \in E(G)} \sum_{e' \in E(G')} \delta(\ell(e), \ell(e')) \tag{3.1}$$

where $\ell(v)$ and $\ell(e)$ denote the label of node v and edge e, respectively. Surprisingly, this and similar types of structure agnostic baseline kernels achieve excellent accuracies on a large set of benchmark datasets (see, e..g., Kriege, Johansson, and Morris, 2020).

The Random Walk Kernel In their pioneer work, Gärtner, Flach, and Wrobel (2003) introduce a graph kernel that defines similarity in terms of mutually occurring walks. Walks are represented as sequences of node and edge labels, and the kernel's feature space is essentially spanned by such sequences. In order to compute the set of mutual walks occurring in two graphs $G, G' \in \mathcal{G}$, the authors compute a *product graph*, denoted $G_\times$. The task of counting mutual walks may then be reduced to counting walks in $G_\times$. The number of mutual length-k walks corresponds to the sum of all entries in $A(G_\times)^k$ where $A(G)$ denotes the adjacency matrix of G. Then, for graphs G, G' and their product graph $G_\times$, the random walk kernel is defined by

$$k(G, G') := \sum_{i,j=1}^{|V(G_\times)|} \left[\sum_{k=0}^{\infty} \lambda_k \cdot A(G_\times)^k \right]_{ij} \tag{3.2}$$

for a suitable non-negative weight sequence $\lambda_0, \lambda_1, \ldots$ such that the limit of Eq. 3.2 exists. Whenever this limit exists, there is a closed-form expression of Eq. 3.2 such that it can be efficiently computed. The authors state a runtime of $O(n^6)$ for the random walk kernel, which can however be significantly reduced (Vishwanathan et al., 2010). Next to the above kernel, which is often referred to as the *geometric* random walk kernel, Gärtner, Flach, and Wrobel (2003) furthermore propose an additional variant, called the *exponential* random walk kernel.

The Shortest-Path Kernel The shortest-path kernel (Borgwardt and Kriegel, 2005) compares two graphs in terms of the similarities between all pairs of shortest paths. While the similarity between two paths may be any kernel function, the authors suggest comparing the endpoints as well as the lengths. More precisely, for graphs G, G', the shortest-path kernel is defined by

$$k(G, G') = \sum_{p \in P(G)} \sum_{p' \in P(G')} k_{\text{node}}(p_1, p_1') \; k_{\text{length}}(p, p') \; k_{\text{node}}(p_2, p_2') \tag{3.3}$$

where $P(G)$ is the set of all shortest paths in G and p_1, p_2 are the endpoints of path $p \in P(G)$. Furthermore, k_{node} is a kernel on the node labels k_{length} is a kernel on the path lengths. In Borgwardt and Kriegel (2005), k_{node} corresponds to the simple Dirac delta function. For k_{length}, the authors propose the Brownian bridge kernel. The overall runtime of the shortest-path kernel is reported to be in $O(n^4)$.

The Graphlet (Sampling) Kernel The idea of graphlet kernels is to define graph similarity by comparing occurrence counts of small subgraphs. The so-called graphlets are fixed-size *induced* subgraphs and the kernel function is simply computed by the dot product between feature vectors, in which each entry corresponds to the occurrence count of a specific graphlet. More precisely, for graphs $G, G' \in \mathcal{G}$ and predefined graphlet sizes $S \subseteq \mathbb{N}$, the graphlet kernel

can be defined by

$$k(G, G') = \sum_{k \in S} \sum_{g \in \mathcal{G}_k} c_g(G) c_g(G') \tag{3.4}$$

where $\mathcal{G}_k$ denotes the set of all size-k graphlets, and $c_g(G)$ is the number of graphlet g occurences in G. While enumerating all size-k graphlets for fixed values k can be done in polynomial time, it is nonetheless often computationally infeasible. Therefore Shervashidze et al. (2009) propose a method which estimates the true graphlet distributions using a sampling technique. They furthermore show that for graphs of bounded degree d, the exact number of size-k graphlets can be computed in time $O(nd^{k-1})$.

Tree-Based Kernels Tree-based kernels form a class of graph kernels that define similarity in terms of matching subtrees. While the first variants of tree-based kernels have been restricted to tree datasets (Vishwanathan and Smola, 2002), Martino, Navarin, and Sperduti (2012) propose an approach which generalizes this type of kernel to arbitrary graphs. The idea is to first decompose a graph into a set of *directed acyclic graphs* (DAG). More precisely, for every node $v \in G$, a DAG is generated using a BFS on G starting in v. The set of all such DAGs of G is denoted by $ODD(G)$. Using a partial order on vertices in each DAG, Martino, Navarin, and Sperduti (2012) effectively replace DAGs by subtrees in order to apply traditional tree-based kernels. This kernel is formally defined by

$$k(G, G') = \sum_{\substack{D \in \mathrm{ODD}(G) \\ D' \in \mathrm{ODD}(G')}} \sum_{\substack{v \in V(D) \\ v' \in V(D')}} \kappa(T(v), T(v')), \tag{3.5}$$

where $T(v)$ is a tree generated by visiting nodes in the corresponding DAG starting in v, and κ is a kernel between trees. Employing the tree kernel κ as defined in Vishwanathan and Smola (2002), results in a overall runtime complexity $O(n^3 \log(n))$.

The Weisfeiler-Lehman Kernel Shervashidze et al. (2011) employed the Weisfeiler-Lehman method (see Sect. 2.7) to define a family of kernels measuring the similarity between graphs based on their relabeled versions. While the authors introduce a general kernel framework, we restrict our focus to the *subtree kernel*. Its idea is to iteratively assign each vertex $v \in G$ a new label $\ell_i(v)$ in iteration i using the Weisfeiler-Lehman vertex relabeling scheme, and compare graphs in terms of such mutually occurring labels. Formally, the Weisfeiler-Lehman subtree kernel for two graphs G, G' and h iterations is defined by

$$k(G, G') = \sum_{i=0}^{h} \sum_{v \in V(G)} \sum_{v' \in V(G')} \delta(\ell_i(v), \ell_i(v')) \tag{3.6}$$

where δ is the Dirac delta function. In other words, the kernel counts the pairs of matching labels over h Weisfeiler-Lehman iterations. With complexity $O(hm)$, where m is the number of edges, the Weisfeiler-Lehman subtree kernel is highly efficient.

The Weisfeiler-Lehman Optimal Assignment Kernel Next to graph kernels based on the R-convolution framework which compare *all pairs* of substructures, graph kernels can alternatively be defined using *optimal assignments* between the substructure sets. Kriege, Giscard, and Wilson (2016) introduce a family of functions that compute optimal matchings between sets of elements X and Y and show that these functions are positive semi-definite using certain ground kernels that compare the elements of X and Y. They apply this approach to define the Weisfeiler-Lehman optimal assignment kernel, which for two graphs, computes an optimal bijection between the sets of their vertices, using the Weisfeiler-Lehman label hierarchy as similarity measure on vertex labels. More precisely, for graphs G, G' and parameter h, the kernel is defined by

$$k(G, G') = \max_{B \in \mathcal{B}} \sum_{(v,v') \in B} \sum_{i=0}^{h} \delta(\ell_i(v), \ell_i(v')) \tag{3.7}$$

where $\mathcal{B}$ is the set of all bijections between $V(G)$ and $V(G')$. Note that whenever $V(G)$ and $V(G')$ are of unequal size, a padding by dummy nodes is necessary. Using a reduction to the *histogram intersection kernel*, the Weisfeiler-Lehman optimal assignment kernel can be computed in time $O(hm)$, where m is the number of edges.

The Wasserstein Weisfeiler-Lehman Kernel Togninalli et al. (2019) propose a graph kernel which can be considered a "soft" matching variant between vertex sets. It compares two graphs G, G' by computing the Wasserstein distance between their depth-h Weisfeiler-Lehman label vectors $\phi(G)$ and $\phi(G')$. The applied ground distance between two depth-h Weisfeiler-Lehman labels is essentially defined in terms of their shortest-path distance in the Weisfeiler-Lehman hierarchy tree. It is shown that this kind of Wasserstein distance yields proper kernel functions. That is, for graphs G, G' and parameter $\gamma > 0$, the function

$$k(G, G') = e^{-\gamma \mathcal{W}_d(\phi(G), \phi(G'))} \tag{3.8}$$

is positive semi-definite. The authors report a worst-case complexity of $O(n^3 \log(n))$, but argue that this can be significantly reduced using approximation variants. We note that the particular Wasserstein distance above can even be solved exactly in linear time using results by Le et al. (2019).

Gradual Weisfeiler-Lehman Kernels In recent work, Bause and Kriege (2022) propose graph kernels comparing node features obtained by effectively slowing down the Weisfeiler-Lehman vertex relabeling method. Similarly to our contribution presented in Chapter 6, they identify the main drawback of the Weisfeiler-Lehman subtree kernel to be its too coarse similarity measure. In an effort to address this issue, Bause and Kriege (2022) replace the injective relabeling function of the Weisfeiler-Lehman method by more general ones. Such functions allow that two vertices with the same label in iteration i and different neighborhood multisets can still be mapped onto the same label in iteration $i + 1$. This deceleration of the vertex refinement method is effectively achieved by clustering similar vertex neighborhood multisets in each refinement step. Finally, the obtained gradual labels are utilized to define

a finer similarity measure employing either the ordinary kernel method of Eq. 3.6 or the optimal assignment kernel of Eq. 3.7. It needs to be noted, that the proposed method by Bause and Kriege is in fact very similar to the GWL* variant presented in Chapter 4, which also effectively slows down the vertex refinement process by performing a hard clustering on labels after every refinement iteration.

Filtration Curves for Graph Classification Similarly to our contribution presented in Chapter 5, O'Bray, Rieck, and Borgwardt (2021) propose an approach which applies the notion of graph filtrations for graph classification tasks. Graph filtrations are sequences of nested subgraphs which describe graphs on various levels of resolution (see Sect. 5.1.1). O'Bray, Rieck, and Borgwardt propose a general framework transforming graphs into so-called *filtration curves*, which are then used as graph descriptors. The generation process of such curves requires an edge weight function inducing filtrations and a graph descriptor function which extracts graph attributes within each filtration graph. For the edge weight function, several functions such as, for example, variants utilizing the maximum node degree, the *Ricci curvature*, or the *Heat kernel signature* are proposed. Concerning the descriptor function, the authors suggest either utilizing numbers of node labels or counting connected components. The resulting filtration curves represent graph attribute distributions over filtrations and can be used as graph representations for learning tasks on graphs. While the authors utilize random forests as the classification method in their experimental evaluations, they mention that filtration curves can also be used for graph kernels. We note that the proposed notion of discrete filtration curves essentially corresponds to that of filtration histograms introduced in Chapter 5. However, there are several distinguishing aspects between our approach and that of O'Bray, Rieck, and Borgwardt (2021). For one, we propose a fine-grained kernel function comparing filtration histograms using the Wasserstein distance. Furthermore, we focus on the case that the features correspond to Weisfeiler-Lehman labels and show that the resulting graph kernels have high expressive power.

3.5 Graph Neural Networks

Graph neural networks (GNN) have emerged as a powerful tool for learning from graph-structured data. GNNs operate by iteratively updating node representations through message passing along the edges, enabling them to capture essential structural information within graphs. This process allows GNNs to be applied to a wide range of graph learning tasks such as node and graph classification, or link prediction.

When compared to traditional methods like graph kernels, a distinguishing property of GNNs is that they work in an end-to-end manner, not requiring an explicit feature extraction step. Instead, GNNs directly learn suitable node representations. While GNNs have proven to be successful learning methods, their expressive power is nonetheless limited by the message passing scheme.

A majority of graph neural networks follows a simple neighborhood aggregation strategy. Much like the vertex relabeling of the Weisfeiler-Lehman method, GNNs update node representations by combining the current node representation with an aggregate of its neighbors'

representations. More precisely, given node representations $r_i(v), v \in V(G)$ at iteration i, a GNN layer computes a new representation of v in iteration $i+1$ by

$$r_{i+1}(v) = \textsc{Combine}(r_i(v), \textsc{Aggregate}(\{\{r_i(u) : u \in \mathcal{N}(v)\}\})) \tag{3.9}$$

where functions Update and Aggregate essentially define the GNN architecture. By using k such layers, a GNN effectively captures the k-hop neighborhood in the computations of node representations. In order to perform graph classification tasks, an additional function is required which combines the set of final node representations over all nodes. For a graph G, this function returns a graph representation

$$r(G) = \textsc{Readout}(\{\{r_k(v), v \in V(G)\}\}) \tag{3.10}$$

that aggregates all node representations of G obtained after k GNN layers.

In seminal work, Xu et al. (2019) show that such graph neural networks can be at most as expressive as the Weisfeiler-Lehman test of isomorphism. In fact, it is proven that there exist GNNs that exhibit the same expressiveness as the 1-dimensional Weisfeiler-Lehman method. More precisely, for *injective* functions Combine, Aggregate, and Readout, there exist GNNs that map any two graphs $G, G' \in \mathcal{G}$, which are distinguished by the Weisfeiler-Lehman method, to different embeddings $r(G) \neq r(G')$.

Recently, there has been a multitude of work concerned with further increasing the expressive power of GNNs and overcoming the above mentioned limitations of the Weisfeiler-Lehman test of isomorphism. For instance, Morris et al. (2019) achieve this by designing a higher-order message passing scheme based on the k-dimensional Weisfeiler-Lehman method, effectively generalizing ordinary GNNs. Another common approach is to enrich nodes with subgraph information (see, e.g., Barceló et al., 2021). Alternatively, the expressive power of GNNs can be improved by restricting the message passing to sets of subgraphs. This can, for instance, be achieved by removing single nodes (Papp et al., 2021; Cotta, Morris, and Ribeiro, 2021) or by restricting the message passing for each node to the subgraph induced by its k-hop neighborhood (Zhang and Li, 2021). In Corollary 5.2 of Chapter 5, we argue that the proposed concept of graph filtrations may also directly be applied to yield more expressive graph neural networks.

4

Graph Kernels Based on Partially Injective Homomorphisms

In Chapter 1, we outlined the need for efficient pattern embedding operators that allow for rich graph representations. Ordinarily, this embedding operator is defined by *subgraph isomorphism*. However, deciding subgraph isomorphism is an NP-complete problem and is thus often not a suitable choice for practical purposes. Motivated by this limitation, in this chapter, we propose a novel type of graph pattern embedding operator which can in fact be decided in *polynomial time*. We show that there exists a mining algorithm that enumerates tree patterns with *polynomial delay* and empirically demonstrate that the generated pattern sets are a suitable choice for graph classification purposes.

In order to define this embedding operator, we consider the relationship between the two standard graph embedding operators *graph homomorphism* and *subgraph isomorphism*. Utilizing the key observation that subgraph isomorphisms are simply *injective* homomorphisms, we introduce a unifying embedding operator which relaxes the notion of injectivity, leading to the concept of *partially injective homomorphisms*. More specifically, while subgraph isomorphisms require all node pairs of a pattern graph to be mapped onto distinct vertices in the target graph, our proposed relaxation defines partial injectivity by requiring only a *subset* of all node pairs to be mapped distinctly. By using this idea of partially injective homomorphisms, we relax the rigid conception of having the binary choice between graph homomorphism and subgraph isomorphism, allowing to *dynamically* choose the degree of injectivity in the pattern matching operator.

The goal in this chapter is to extract a set of suitable *tree* patterns for graph classification tasks. A common practice for this purpose is to consider sets of *frequent* subgraph patterns, i.e., patterns which are contained in at least a certain fraction of the target graphs. However, this approach is generally not computationally feasible due to the NP-completeness of the subgraph isomorphism problem, even when the pattern graphs are trees. Graph homomorphisms, on the other hand, can in fact be decided in polynomial time for tree patterns. It, however, generally produces less suitable frequent pattern sets for graph classification tasks. This drawback of employing graph homomorphisms as the pattern embedding operator for mining frequent trees becomes particularly apparent when considering the unlabeled case,

where every tree pattern can be embedded into any target graph which contains at least one edge. As a trade-off between expressiveness and complexity, we employ the concept of partially injective homomorphism to preserve from the rigidity of subgraph isomorphism as much as possible, while utilizing the efficiency of homomorphisms for tree patterns.

From an algorithmic point of view, partially injective homomorphisms are realized by extending pattern graphs by special edges that correspond to injectivity constraints. We show that partially injective homomorphisms are efficiently decidable if and only if ordinary homomorphism from the edge-extended pattern graph can be decided in polynomial time. In order to maximize the number of injectivity constraints while guaranteeing efficiency, we utilize the class of *bounded treewidth* graphs (Robertson and Seymour, 1986). More precisely, we extend patterns by a maximal set of injectivity constraint edges such that the resulting graphs have bounded treewidth. While there exist several graph classes from which ordinary homomorphism can be decided efficiently, bounded treewidth graphs are of particular interest for our purpose. In fact, Grohe (2007) proves that homomorphisms can be decided in polynomial time if and only if the pattern graph has bounded treewidth. Using this result, partially injective homomorphisms guarantee a *maximal degree* of injectivity while remaining efficiently decidable.

In order to extract suitable pattern sets for graph classification purposes, we propose a mining algorithm that enumerates frequent patterns w.r.t. partially injective homomorphism. As the set of injectivity constraints depends on the particular pattern at hand, the output of the mining algorithm contains not only the tree patterns, but also the injectivity constraints. A complete enumeration of all frequent patterns together with the corresponding constraint set is, however, practically infeasible. For one thing, this issue is overcome by requiring edge-extended pattern graphs to be k-trees (Rose, 1974), i.e., edge maximal graphs of treewidth at most k. Utilizing the algorithmic definition of k-trees, we arrive at a *natural refinement operator* for the corresponding pattern mining problem, allowing for an efficient frequent pattern enumeration.

In our experimental evaluation, we asses the predictive performance of the patterns obtained. We measure the accuracy achieved using kernel functions which define graph similarity in terms of co-occurring pattern graphs. Our experimental results yield two significant observations. Firstly, it can be observed that the degree of partial injectivity is directly correlated to the predictive performance. The results show that as the degree of partial injectivity increases, the predictive performance gradually approaches that achieved by ordinary subgraphs. Secondly, we found that relatively low degrees of partial injectivity suffice to obtain results close to that of ordinary subgraphs. Hence, our approach provides an attractive trade-off between complexity and predictive performance.

4.1 Partially Injective Homomorphisms

In this section, we formally define *partially injective homomorphism*, the central notion for this chapter, and discuss some of its properties. As outlined above, partially injective homomorphisms provide a natural transition from graph homomorphisms to subgraph isomorphisms. We obtain this transition by considering *partial* injectivity, i.e., by requiring the injectivity

(a) Pattern H with constraints $\mathcal{C} = \{v_2v_3\}$.

(b) Target graph G.

Figure 4.1: Transformation of pattern graph H with constraint set $\mathcal{C} = \{v_2v_3\}$ and target graph G using the reduction steps as described in Sect. 4.1.1.

constraint *not* for all vertex pairs in the pattern graph, but only for subsets. This can be formalized as follows:

Definition 4.1 (Partially Injective Homomorphism). *For graphs $H, G \in \mathcal{G}$, and constraints $\mathcal{C} \subseteq [V(H)]^2$, a* partially injective homomorphism *from H into G satisfying the injectivity constraints in $\mathcal{C}$ is a homomorphism φ from H into G such that $\varphi(u) \neq \varphi(v)$ for all $uv \in \mathcal{C}$. We write $H \xrightarrow{\mathcal{C}} G$ if such a partially injective homomorphism exists.*

We refer to the corresponding decision problem as PIHom *problem* and denote it by PIHom$(H, G, \mathcal{C})$. Notice that the set of constraints $\mathcal{C}$ is specific to the pattern graph H at hand. Hence, we consider the pairs $(H, \mathcal{C})$ and call it a PIHom *pattern*. Due to loop-freeness it suffices to consider the injectivity constraints only for unconnected vertex pairs in the pattern graph H (i.e., we can assume w.l.o.g. that $\mathcal{C} \cap E(H) = \varnothing$).

4.1.1 Reduction to Ordinary Homomorphisms

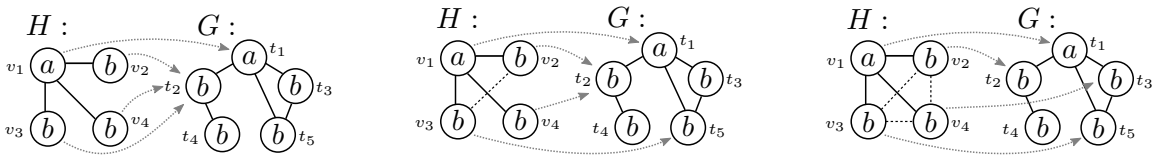
Partially injective homomorphisms can be polynomially reduced to ordinary homomorphisms by performing a sequence of transformation steps on the pattern and target graphs. This is achieved by adding auxiliary edges (referred to as *red edges*) to both graphs in the following way: For H, G, and $\mathcal{C}$ above we transform H and G into edge colored graphs by the following steps:

1. Color all (original) edges of H and G in *blue*,
2. for all $uv \in E(H) \cup \mathcal{C}$, connect u and v by a red edge, and
3. for *all* $u, v \in V(G)$ with $u \neq v$, connect u and v by a red edge.

Let the graphs obtained be denoted by $H\langle\mathcal{C}\rangle$ and $G\langle\top\rangle$. A visualization of this transformation is depicted in Fig. 4.1. Since there is a one-to-one correspondence between pairs $(H, \mathcal{C})$ and graphs $H\langle\mathcal{C}\rangle$, we sometimes don't distinguish between the two notions. Note that for reasons of clarity, in the following, we will only visualize the red edges in the pattern graphs.

As homomorphisms between colored graphs preserve also the edge colors, the proof of the claim below is immediate from the definitions.

Proposition 4.1. *For $H, G, \mathcal{C}$ and $H\langle\mathcal{C}\rangle, G\langle\top\rangle$ above, it holds that $H \xrightarrow{\mathcal{C}} G$ if and only if $H\langle\mathcal{C}\rangle \preceq_h G\langle\top\rangle$.*



(a) $\mathcal{C}_1 = \varnothing$ (b) $\mathcal{C}_2 = \{v_2v_3\}$ (c) $\mathcal{C}_3 = \top$

Figure 4.2: Examples for partially injective homomorphisms from H into G w.r.t. constraint sets $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$. Solid lines represent blue edges whereas dashed ones represent red edges.

Fig. 4.2 visualizes three partially injective homomorphisms from H into G for different choices of $\mathcal{C}$. The constraints are explicitly visualized as dashed lines. Fig. 4.2(a) corresponds to $H \xrightarrow{\varnothing} G$, i.e., to ordinary homomorphism. Due to the lack of additional injectivity constraints between vertices in H, the vertices $\{v_2, v_3, v_4\}$ may be arbitrarily mapped onto neighbors of t_1 in G. Fig. 4.2(b) depicts a partially injective homomorphism with a single additional injectivity constraint. The constraint set $\mathcal{C} = \{v_2v_3\}$ enforces that v_2 and v_3 are mapped to distinct vertices in G. The final case depicted in Fig. 4.2(c) shows a partially injective homomorphism with a maximal constraint set $\mathcal{C} = \{v_2v_3, v_2v_4, v_3v_4\} = [V(H)]^2 \backslash E(H)$. The mapping enforces partial injectivity between all vertex pairs in H and, thus, corresponds to subgraph isomorphism.

4.1.2 The Lattice of PIHom Problems

For H and G above, each set $\mathcal{C} \subseteq [V]^2 \backslash E(H)$ defines a distinct PIHom$(H, G, \mathcal{C})$ problem and the set of all PIHom patterns over H form a (complete) lattice $(\mathcal{L}_H, \preceq)$ with

$$\mathcal{L}_H = \{(H, \mathcal{C}) : \mathcal{C} \subseteq [V(H)]^2 \backslash E(H)\} \tag{4.1}$$

and with partial order $\preceq$ defined as follows:

$$\text{For all } \mathcal{C}_1, \mathcal{C}_2 \subseteq [V(H)]^2 \backslash E(H),\ (H, \mathcal{C}_1) \preceq (H, \mathcal{C}_2) \text{ iff } \mathcal{C}_1 \subseteq \mathcal{C}_2.$$

The least element $(H, \varnothing)$ of $\mathcal{L}_H$ corresponds to ordinary homomorphism from H. Similarly, the greatest element $(H, [V]^2 \backslash E(H))$ matches the case of subgraph isomorphism. For any target graph G, $(\mathcal{L}_H, \preceq)$ is closed downwards in the sense that $H \xrightarrow{\mathcal{C}_1} G$ and $\mathcal{C}_2 \subseteq \mathcal{C}_1$ implies $H \xrightarrow{\mathcal{C}_2} G$.

In this chapter, we consider lattices of PIHom *tree* patterns, i.e., when the first component in the PIHom pattern $(H, \mathcal{C})$ is a tree. When H is a tree with n vertices, the cardinality of the corresponding PIHom pattern lattice $\mathcal{L}_H$ is $2^{O(n^2)}$, i.e., *exponential* in the size of H. Fig. 4.3 illustrates such a lattice $(\mathcal{L}_H, \preceq)$ for a labeled path H of length 3. Each depicted graph corresponds to a PIHom tree pattern with a specific set of injectivity constraints.

4.2 Pattern Mining

In this section, we discuss the main ingredients concerned with the task of generating frequent trees w.r.t. partially injective homomorphism. We start by defining the problem of

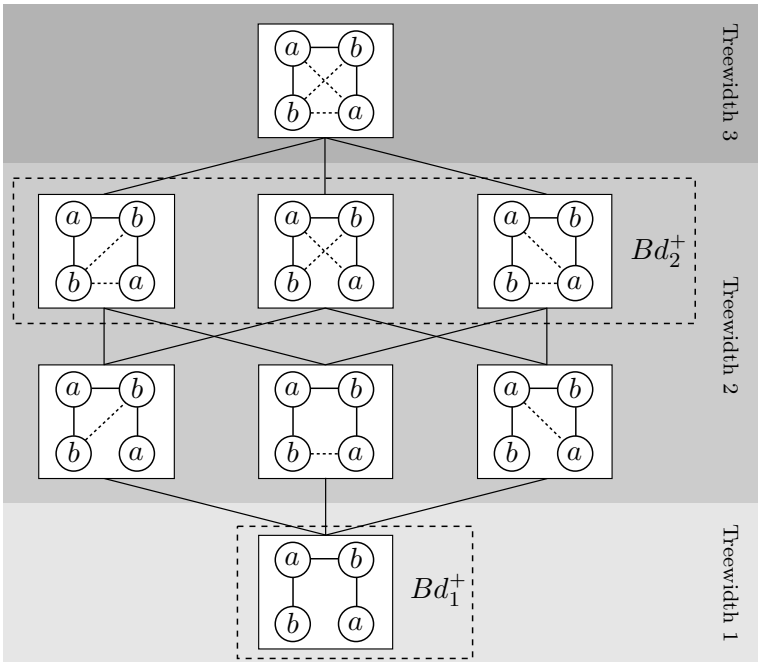


Figure 4.3: Visualization of the lattice $(\mathcal{L}_H, \preceq)$ for a labeled path H.

enumerating frequent *maximally* constrained PIHom tree patterns and introduce a corresponding *refinement operator*. Next, we show that there exists no locally finite and complete refinement operator for a natural representative set of this kind of patterns if we want to avoid redundancy. We therefore relax the problem definition and propose an efficient pattern mining algorithm tolerating certain redundancies in the output.

4.2.1 Efficiently Decidable PIHom Problems

Our key idea is to consider such PIHom problems that can *polynomially* be reduced to *efficiently* decidable ordinary homomorphism problems. For the efficiency, we consider PIHom patterns of bounded treewidth, utilizing positive complexity results on deciding homomorphisms from this graph class (Dalmau, Kolaitis, and Vardi, 2002). The restriction of patterns to *trees* (i.e., H in $(H, \mathcal{C})$ is a tree) is motivated by a very natural refinement operator as well as by the remarkable predictive performance, which is achieved by using frequent subtrees w.r.t. subgraph isomorphism (Welke, Horváth, and Wrobel, 2017).

In general, given a *tree* H and target graph G, $\text{PIHom}(H, G, \mathcal{C})$ can be decided in polynomial time if $\mathcal{C} = \varnothing$ (i.e., for ordinary homomorphism), but is NP-complete whenever $\mathcal{C} = [V(H)]^2 \backslash E(H)$ (i.e., for subgraph isomorphism). We bridge this complexity gap by considering a distinguished subset of $\mathcal{L}_H$ for which the corresponding PIHom problems are decidable in polynomial time for any target graph G. To ensure this property, we utilize the notion of treewidth (Robertson and Seymour, 1986). More precisely, for a tree H and some

constant $k > 0$, we consider the pattern set

$$\mathcal{L}_H^k = \{(H, \mathcal{C}) \in \mathcal{L}_H : H\langle\mathcal{C}\rangle \text{ has treewidth at most } k\}. \tag{4.2}$$

The following proposition is immediate from the reduction of PIHom problems to ordinary homomorphism (Proposition 4.1), together with the positive complexity result on deciding homomorphisms from graphs of bounded treewidth (Dalmau, Kolaitis, and Vardi, 2002).

Proposition 4.2. *For all $(H, \mathcal{C}) \in \mathcal{L}_H^k$ and for all target graphs G,* $\text{PIHom}(H, G, \mathcal{C})$ *can be decided in polynomial time.*

Our experiments (cf. Sect. 4.4) clearly demonstrate that, besides the structural gap between homomorphisms and subgraph isomorphisms discussed earlier, there is a large gap between the predictive performances obtained when utilizing them as pattern embedding operators for graph classification purposes. Furthermore, this gap vanishes as the number of injectivity constraints increases. Motivated by this empirical observation and the negative result formulated in Section 4.2.2 below, we will pay special attention to the *maximal* elements of $\mathcal{L}_H^k$. The intuition is to consider such constraint sets $\mathcal{C}$ for which $\text{PIHom}(H, G, \mathcal{C})$ can be efficiently decided while retaining as much partial injectivity as possible. More precisely:

Maximal elements of $\mathcal{L}_H^k$**:** A pattern $(H, \mathcal{C}) \in \mathcal{L}_H^k$ is *maximally constrained* if

(i) $H\langle\mathcal{C}\rangle$ is a complete graph whenever $|V(H)| \leqslant k + 1$, and

(ii) $H\langle\mathcal{C}\rangle$ is a k-tree whenever $|V(H)| > k + 1$.

In fact, the set of maximally constrained PIHom patterns form a *positive border* on $(\mathcal{L}_H^k, \leq)$ w.r.t. the following *interestingness predicate*: $(H, \mathcal{C}) \in \mathcal{L}_H$ is *interesting* if $H\langle\mathcal{C}\rangle$ has treewidth at most k. We denote this border by $Bd_k^+(H)$.

Fig. 4.3 depicts the complete lattice $(\mathcal{L}_H, \leq)$ for a particular pattern H. The least element in it corresponds to the PIHom pattern $(H, \varnothing)$. It is the only element having treewidth 1 and, hence, trivially forms the border Bd_1^+ for this tree pattern. While there exist a total of six distinct PIHom patterns of treewidth 2, only the ones containing two injectivity constraints are in fact maximally constrained w.r.t. treewidth $k = 2$. Adding any further constraint would increase their treewidth to 3.

4.2.2 PIHom Core Patterns: A Negative Result

In the following, we address a central issue of enumerating PIHom patterns. Using the concepts introduced above, in this section we consider the pattern language $\mathcal{L}^k$ defined by the union of the $\mathcal{L}_H^k$s over all trees H. We show that a duplicate free enumeration of elements in $\mathcal{L}^k$ is not possible when using standard frequent pattern mining methods.

Our goal is to generate a subset S of $\mathcal{L}^k$ on the basis of a graph database $\mathcal{D}$ for graph classification purposes. For this purpose, it is desirable to avoid "redundancies" among the patterns in S. Clearly, there are various ways of defining redundancies. Perhaps the most natural one is to consider two patterns equivalent if they are contained in the exact same subset of graphs in $\mathcal{D}$. We refer to two such patterns as *model equivalent*. More specifically:

MODEL EQUIVALENCE **($\equiv_{\mathbf{m}}$):** Two pattern graphs $(H_1, \mathcal{C}_1), (H_2, \mathcal{C}_2) \in \mathcal{L}^k$ are *model equivalent*, denoted $(H_1, \mathcal{C}_1) \equiv_{\mathrm{m}} (H_2, \mathcal{C}_2)$, if the equivalence $H_1 \xrightarrow{\mathcal{C}_1} G \iff H_2 \xrightarrow{\mathcal{C}_2} G$ holds for all graphs $G \in \mathcal{D}$.

While this notion of redundancy is arguably the most desirable one, it raises severe algorithmic issues. In fact, such issues even exist for the weaker definition of pattern equivalence defined by homomorphism equivalence. More precisely, consider the order $\preceq_{\mathrm{h}}$ on $\mathcal{L}^k$ defined as follows: For all $(H_1, C_1), (H_2, C_2) \in \mathcal{L}^k$, $(H_1, C_1) \preceq_{\mathrm{h}} (H_2, C_2)$ iff $H_1\langle\mathcal{C}_1\rangle \preceq_{\mathrm{h}} H_2\langle C_2\rangle$. One can easily see that $\preceq_{\mathrm{h}}$ is a preorder on $\mathcal{L}^k$. Using this definition, a set $S \subseteq \mathcal{L}^k$ is regarded as *non-redundant* if it contains no two *homomorphism equivalent* patterns, defined as follows.

HOMOMORPHISM EQUIVALENCE **($\equiv_{\mathbf{h}}$):** Two pattern graphs $(H_1, \mathcal{C}_1), (H_2, \mathcal{C}_2) \in \mathcal{L}^k$ are *homomorphism equivalent*, denoted $(H_1, \mathcal{C}_1) \equiv_{\mathrm{h}} (H_2, \mathcal{C}_2)$, iff $(H_1, C_1) \preceq_{\mathrm{h}} (H_2, C_2)$ and $(H_2, C_2) \preceq_{\mathrm{h}} (H_1, C_1)$.

Clearly, $\equiv_{\mathrm{m}}$ and $\equiv_{\mathrm{h}}$ are both equivalence relations and $\equiv_{\mathrm{h}}$ is finer than $\equiv_{\mathrm{m}}$, i.e., the partition of $\mathcal{L}^k$ induced by $\equiv_{\mathrm{h}}$ is a refinement of that induced by $\equiv_{\mathrm{m}}$. Thus, model equivalent patterns are not necessarily homomorphism equivalent, implying that $\equiv_{\mathrm{h}}$ may allow a certain amount of redundancies w.r.t. $\equiv_{\mathrm{m}}$.

A central issue of employing homomorphism as a pattern matching operator is that the equivalence classes in $\mathcal{L}^k/\equiv_{\mathrm{h}}$ may contain infinitely many patterns, due to the fact that $\preceq_{\mathrm{h}}$ is not anti-symmetric. In order to avoid duplicates, it is therefore desirable to identify at most one element from each equivalence class in $\mathcal{L}^k/\equiv_{\mathrm{h}}$. For each such $C \in \mathcal{L}^k/\equiv_{\mathrm{h}}$, one can consider the *core* of C, a canonical representative element, defined as follows: Select an arbitrary pattern $(H, \mathcal{C}) \in C$ and take the smallest subgraph $H'\langle\mathcal{C}'\rangle$ of $H\langle\mathcal{C}\rangle$ such that $(H, \mathcal{C}) \equiv_{\mathrm{h}} (H', \mathcal{C}')$. It holds that $H'\langle\mathcal{C}'\rangle$ can be computed by a greedy algorithm removing the redundant edges one by one. The properties of treewidth together with the results of Chandra and Merlin (1977) and Dalmau, Kolaitis, and Vardi (2002) imply that $(H', \mathcal{C}')$ is a core of C, it always exists and is unique modulo isomorphism independently of the choice of $(H, \mathcal{C})$, and can be calculated in time polynomial in the size of $(H, \mathcal{C})$. We denote the set of cores in $\mathcal{L}^k$ by $\mathcal{L}^k_{\mathrm{c}}$.

While subgraph isomorphism as the pattern matching operator allows for a very natural refinement operator on the pattern language, this is typically not the case for homomorphism, caused also by the difference in the anti-symmetry. Indeed, in case of subgraph isomorphism, the pattern language along with the partial order defined by subgraph isomorphism can directly be translated into an *ideal* refinement operator (assuming that all patterns are connected); just extend the pattern at hand in every possible way either by a single edge or by a single vertex connected to one of the existing vertices. In contrast, the preorder on $\mathcal{L}^k$ defined by $\preceq_{\mathrm{h}}$ does not impose such an algebraic structure that could be turned into a (natural) algorithmic definition of a refinement operator on $\mathcal{L}^k_{\mathrm{c}}$. In fact, $\mathcal{L}^k_{\mathrm{c}}$ may contain cores having infinitely many "direct" refinements, implying the following negative result:

Theorem 4.1. *For $k \geqslant 1$, there exists no finite and complete refinement operator for the preordered set $(\mathcal{L}^k_c, \preceq_h)$.*

Proof. We show the claim for $k = 1$, by noting that a similar argument can be used for all $k > 1$. Consider the case that the vertices of H are labeled by the elements of $\Sigma = \{a, b, c\}$ for all $(H, \mathcal{C}) \in \mathcal{L}_c^k$ and suppose for contradiction that there exists a finite and complete refinement operator ρ for $(\mathcal{L}_c^k, \preceq_h)$. Then, for all $(H, \mathcal{C}) \in \mathcal{L}_c^k$, there exists an $N \in \mathbb{N}$ such that $|\rho((H, \mathcal{C}))| \leqslant N$. Let $P = v_1v_2v_3$ be a path such that its vertices v_1, v_2, v_3 are labeled by a, b, b, respectively. We denote P by the string abb. Clearly, $P\langle\varnothing\rangle$ is a core. Thus, $(P, \varnothing) \in \mathcal{L}_c^k$. One can see in a similar way that $(P_\ell, \varnothing) \in \mathcal{L}_c^k$ for all $\ell \in \mathbb{N}$, where P_ℓ is the path $abbb^\ell c$. Furthermore, for all $\ell \in \mathbb{N}$ it holds that there is no $(H, \mathcal{C}) \in \mathcal{L}_c^k$ such that $P\langle\varnothing\rangle \preceq_h H\langle\mathcal{C}\rangle$ and $H\langle\mathcal{C}\rangle \preceq_h P_\ell\langle\varnothing\rangle$. By finiteness, $|\rho((P, \varnothing))| = n$ for some $n \in \mathbb{N}$. But then at least one of the elements of $\{(P_\ell, \varnothing) : \ell = 0, 1, \ldots, n\}$ is not in $\rho((P, \varnothing))$, contradicting ρ's completeness. □

The negative result formulated in Theorem 4.1 does not imply that PIHom core patterns cannot be enumerated *efficiently*. However, it indicates that traditional pattern generation paradigms based on refinement operators are not applicable to $(\mathcal{L}_c^k, \preceq_h)$. In the next section we therefore relax our problem setting and tolerate further redundancies in the output pattern set.

4.2.3 The Problem Definition

Theorem 4.1 implies that we have to consider a different pattern language in place of $\mathcal{L}_c^k$ if we want to generate the output patterns by using some algorithmically appropriate refinement operator. To achieve this goal, we revert to considering the set of *maximally constrained* patterns $\mathcal{L}_{\max}^k \subseteq \mathcal{L}^k$. More formally, we consider the patterns

$$\mathcal{L}_{\max}^k := \{(H, \mathcal{C}) \in Bd_k^+(H) : H \text{ is a tree}\}. \tag{4.3}$$

The corresponding partially injective homomorphisms obtained in this way are as close as possible to subgraph isomorphism subject to bounded treewidth, resulting in a pattern set of higher predictive performance, as shown empirically in Section 4.4. While this choice of pattern language does not lead to a redundancy free mining algorithm w.r.t. homomorphism equivalence, it allows for outputting such patterns with incremental polynomial delay. Using this definition, we consider the following pattern enumeration problem:

Frequent Maximally Constrained Tree Mining (FMCTM) Problem: *Given* a finite set $\mathcal{D}$ of graphs and integers $t, h, k > 0$, *list* all $(H, \mathcal{C}) \in \mathcal{L}_{\max}^k$ such that $|V(H)| \leqslant h$ and $\text{freq}((H, \mathcal{C}), \mathcal{D}) \geqslant t$, where $\text{freq}((H, \mathcal{C}), \mathcal{D})$ denotes the (absolute) *frequency* of the PIHom tree pattern $(H, \mathcal{C})$ in $\mathcal{D}$, i.e., $\text{freq}((H, \mathcal{C}), \mathcal{D}) = |\{G \in \mathcal{D} : H \xrightarrow{\mathcal{C}} G\}|$.

PIHom tree patterns satisfying the frequency constraint in the definition above will be referred to as *frequent* PIHom patterns, or simply frequent patterns. As mentioned earlier, one of the most important distinguishing features of the problem setting above is that the pattern matching operator is *not* static (i.e., fixed in advance), but *dynamic*, in contrast to all traditional frequent graph mining algorithms. Clearly, whether a tree pattern is frequent or not directly depends on the underlying set of constraints applied in the embedding operator. It is therefore necessary to output a frequent tree along with the injectivity constraints

Algorithm 2 Listing Frequent Maximally Constrained Patterns

input: graph dataset $\mathcal{D}$, integers $t, k, h > 0$
output: all t-frequent patterns of $\mathcal{L}^k_{\max}$ with size at least 1 and at most h

Enumerate$((H, \mathcal{C}))$:

```
1: R := Refinements((H, C), k)
2: for all (H', C') ∈ R do
3:     if |V(H')| ⩽ h ∧ (H', C') ∉ O ∧ freq((H', C'), D) ⩾ t then
4:         print (H', C') and add it to O
5:         Enumerate((H', C'))
```

Main:

```
1: O := ∅
2: Enumerate((⊥, ∅))                    // ⊥ denotes the empty graph
```

defining the (dynamic) pattern matching operator, i.e., the output is always a pair $(H, \mathcal{C})$, instead of H only.

The parameter h in the problem definition provides an upper bound on the size of the output patterns. It ensures that the algorithm solving the FMCTM problem will always terminate. It is noteworthy that without h, the output of the FMCTM problem would generally contain infinitely many frequent patterns. We also note that the output $\mathcal{O}$ of the FMCTM problem may contain frequent patterns that are homomorphism equivalent. Furthermore, the elements of $\mathcal{O}$ are not necessarily cores. In case the output is required to be a non-redundant subset of $\mathcal{L}^k_c$, after the computation of $\mathcal{O}$, one can first remove all patterns from it that are redundant w.r.t. homomorphism equivalence and then calculate the core for each pattern remaining in $\mathcal{O}$. Since all patterns in $\mathcal{O}$ have bounded treewidth, both steps can be performed in time polynomial in the size of $\mathcal{O}$.

4.2.4 The Mining Algorithm

In this section, we present our algorithm solving the FMCTM problem and prove that it is correct and enumerates the output patterns in *incremental polynomial* time.

We guarantee efficiency, i.e., incremental polynomial time (c.f. Sect 2.5.1), by considering the *partial* order $\subseteq$ on $\mathcal{L}^k_{\max}$, instead of the *preorder* $\preceq_h$, where $\subseteq$ is defined as follows: For all $(H, \mathcal{C}), (H', \mathcal{C}') \in \mathcal{L}^k_{\max}$, $(H, \mathcal{C}) \subseteq (H', \mathcal{C}')$ if and only if there exists a subgraph isomorphism from $H\langle\mathcal{C}\rangle$ into $H'\langle\mathcal{C}'\rangle$. Clearly, $\text{freq}((H, \mathcal{C}), \mathcal{D}) \geqslant \text{freq}((H', \mathcal{C}'), \mathcal{D})$ whenever $H\langle\mathcal{C}\rangle \subseteq H'\langle\mathcal{C}'\rangle$, i.e., frequency is *anti-monotonic* on the poset $(\mathcal{L}^k_{\max}, \subseteq)$. Thus, maximal PIHom tree patterns are closed downwards w.r.t. frequency. While the poset $(\mathcal{L}^k_{\max}, \subseteq)$ allows for an efficient pattern enumeration, the output may contain patterns that are homomorphism equivalent. That is, the price we have to pay for the positive complexity result is that the output may contain some redundant patterns.

Algorithm 2 is based on the recursive function Enumerate generating the output patterns in a DFS manner. Its input consists of the same parameters $\mathcal{D}$, t, h, and constant k as the

FMCTM problem. The output patterns already generated are stored in the global variable $\mathcal{O}$. The algorithm calls Enumerate with the empty pattern $(\bot, \varnothing)$, where $\bot$ denotes the empty graph (line 2 of Main). As a first step (line 1 of Enumerate), function Refinements generates the set of refinements for the input pattern $(H, \mathcal{C})$; the process governing how new candidate patterns are generated is determined by the refinement operator described below. If a newly generated candidate pattern $(H', \mathcal{C}')$ (i) fulfills the size constraint (i.e, $|V(H')| \leqslant h$), (ii) has not been generated before (i.e., $(H', \mathcal{C}') \notin \mathcal{O}$), and (iii) is t-frequent (i.e., $\text{freq}((H', \mathcal{C}'), \mathcal{D}) \geqslant t$), we print it, store it in $\mathcal{O}$, and call Enumerate recursively for this new frequent pattern (lines 3–5).

Refinement Operator Function Refinements in Algorithm 2 returns the set R of *refinements* for a pattern $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ and $k > 0$. All patterns $(H', \mathcal{C}') \in R$ are required to satisfy the following conditions:

(i) H' is a supertree of H obtained by extending H with a new vertex and edge,

(ii) $\mathcal{C} \subseteq \mathcal{C}'$, and

(iii) $(H', \mathcal{C}') \in \mathcal{L}^k_{\max}$, i.e, it is maximal w.r.t. treewidth k.

That is, trees of size n are extended into trees of size $n+1$ by condition (i). Furthermore, condition (iii) implies that $H'\langle\mathcal{C}'\rangle$ is a k-tree if $|V(H')| > k+1$; otherwise it is a complete graph.

The algorithmic characterization of k-trees (cf. Def. 2.11 in Sect. 2.4) gives rise to the following natural refinement operator on $\mathcal{L}^k_{\max}$: A pattern $(H', \mathcal{C}')$ of size $n+1$ is among the refinements of a pattern $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ of size n iff $(H', \mathcal{C}')$ can be obtained from $(H, \mathcal{C})$ in the following way: If $n = 0$ (i.e., $(H, \mathcal{C}) = (\bot, \varnothing)$), we define the refinements of $(H, \mathcal{C})$ by the set of graphs consisting of a single vertex (and no edges, as we consider loop-free graphs). Otherwise, i.e., for $n > 0$, we proceed as follows:

(i) Introduce a new vertex u.

(ii) If $n \leqslant k$, then connect u to a vertex $v \in V(H)$ and add an injectivity constraint uv' to $\mathcal{C}$ for every $v' \in V(H)\backslash\{v\}$; otherwise select a k-clique C in $H\langle\mathcal{C}\rangle$, connect u to a vertex v of C in H, and add an injectivity constraint uv' to $\mathcal{C}$ for every $v' \in V(C)\backslash\{v\}$.

An example of such a refinement step is given in Fig. 4.4. The correctness of the refinement step is stated in the following lemma.

Lemma 4.1. *Let $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ and let $(H', \mathcal{C}')$ be a pattern obtained from $(H, \mathcal{C})$ using the refinement process as defined above. Then $(H', \mathcal{C}') \in \mathcal{L}^k_{\max}$.*

Proof. The proof that H' is a tree is straightforward. Regarding the maximality w.r.t. treewidth k, the graph $H'\langle\mathcal{C}'\rangle$ of size $n+1$ which corresponds to the refined pattern $(H', \mathcal{C}')$ is always a complete graph of treewidth at most k for the case $n \leqslant k$. For $n > k$, it is always a k-tree by the algorithmic characterization of k-trees. Thus, $(H', \mathcal{C}') \in \mathcal{L}^k_{\max}$ follows as claimed. □

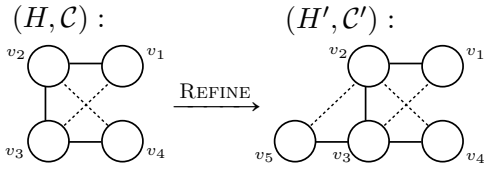


Figure 4.4: PIHom pattern $(H, \mathcal{C})$ and one of its refinements for the case $k = 2$. $(H', \mathcal{C}')$ is constructed by selecting the clique $\{v_2, v_3\}$, connecting the new vertex v_5 to v_3 by a blue edge and to v_2 by a red edge.

Due to the algorithmic description of the refinement operator, it is easy to see that the number of refinements is polynomially bounded. More precisely:

Lemma 4.2. *For any $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$, the number of refinements of $(H, \mathcal{C})$ is bounded by a polynomial of the size of H and the cardinality of the node label alphabet Σ.*

Proof. Let $n = |V(H)|$. For $n \leqslant k$, the new vertex u can connect to each of the n vertices by a blue edge. Taking into account the degree of freedom for the vertex label of u and the edge label of the newly introduced blue edge, the set of refinements of $(H, \mathcal{C})$ is bounded by $n|\Sigma|^2$. For the case $n > k$, the new vertex u can be connected by a blue edge to each of the k vertices of each k-clique of $H\langle\mathcal{C}\rangle$. As $H\langle\mathcal{C}\rangle$ is a k-tree, it has exactly $nk - k^2 + 1 \leqslant nk$ k-cliques (Kloks, 1994). Thus, the number of refinements is bounded by $nk^2|\Sigma|^2$. □

Finally, we are ready to formalize our main result which states that frequent PIHom patterns can be efficiently enumerated:

Theorem 4.2. *For any $\mathcal{D}$, t, k, and h, Algorithm 2 is correct and generates the output patterns in incremental polynomial time.*

Proof. Regarding the correctness, the soundness is immediate from Lemma 4.1 together with lines 2–3 of Alg. 2 and the completeness follows by induction on the pattern size from the algorithmic characterization of k-trees and from the anti-monotonicity of frequency.

Regarding the enumeration complexity, the number of refinements of a pattern $(H, \mathcal{C})$ is bounded by a polynomial of the combined size of H and Σ (Lemma 4.2), which, in turn, is bounded by the size of $\mathcal{D}$. As the patterns are generated in a DFS manner, the number of patterns for which the condition in line 3 has to be tested after the output of the last frequent pattern until the next one or termination is thus bounded by a polynomial of $\mathcal{D}$ and h. Regarding the complexity of the conditions of line 3, we note that the condition on the pattern size can be checked in time linear in the size of H'. Testing the membership condition $(H', \mathcal{C}') \in \mathcal{O}$ can be done in time polynomial in the combined size of H' and $\mathcal{O}$, as isomorphism between k-trees can be decided in linear time (Arvind et al., 2012). Finally, deciding whether $(H', \mathcal{C}') \in \mathcal{L}^k_{\max}$ is frequent in $\mathcal{D}$ can be decided in time polynomial in the combined size of H' and $\mathcal{D}$, as for all graphs $G \in \mathcal{D}$ it can be decided in time polynomial in the combined size of H' and G whether there exists a homomorphism from $H'\langle\mathcal{C}'\rangle$ into $G\langle\top\rangle$ (cf. Proposition 4.2). Thus, as the pattern size is bounded by h, the conditions in line 3 can be checked in time $\mathrm{poly}(h, \mathrm{size}(\mathcal{D}), \mathrm{size}(\mathcal{O}))$, i.e., in incremental polynomial time as stated. □

4.2.5 The Number of PIHom Patterns

The last section has introduced an efficient mining algorithm that enumerates frequent PIHom patterns with incremental polynomial delay. In other words, the time it takes to output two consecutive elements is bounded by a polynomial in the combined size of the input as well as previously outputted patterns. This notion commonly serves as a complexity measure for enumeration algorithms (see Sect. 2.5.1) and is particularly relevant in cases where the output set may be be of exponential or even infinite size.

Despite the bounded delay, we, nonetheless, require the amount of frequent patterns to be manageable in order to solve the FMCTM problem in practice. The output size may (to some degree) be influenced by the choices for the frequency threshold t and maximum pattern size h. However, while the output size of pattern mining algorithms is a general concern of subgraph mining problems, it is specifically problematic when considering partially injective homomorphism. More precisely, it holds that if a pattern tree H is subgraph isomorphic to a target graph G, then $H \xrightarrow{\mathcal{C}} G$ holds for any injectivity constraint set $\mathcal{C}$. In fact, for every such H, the number of frequent maximally constrained patterns $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ may even be exponential in the size of H. This observation on the output size of the FMCTM problem suggests that while the pattern enumeration can be done efficiently, the burden of complexity is somewhat shifted onto an exploding number of frequent PIHom patterns.

In order to tackle this problem, in our experimental evaluation (see Sect. 4.4), we consider a practically more feasible variant of Algorithm 2 by giving up the completeness. More precisely, we require the output to contain no two patterns $(H_1, \mathcal{C}_1)$ and $(H_2, \mathcal{C}_2)$ with $H_1 \equiv H_2$. In other words, a tree pattern may be outputted at most once. This is achieved by altering line 1 in Alg. 2 such that function Refinements returns a single random pattern from the set of all possible refinements. Furthermore, instead storing PIHom patterns in $\mathcal{O}$, it suffices to store only tree patterns in it. A consequence of this alteration is that some tree patterns which are contained in the complete output of Alg. 2 may not be outputted anymore. However, if a tree is a frequent subgraph (w.r.t. subgraph isomorphism), then it is also contained in the output of the altered version of Algorithm 2. Finally, we note that the incompleteness of the output does not compromise the soundness or the polynomial delay of Algorithm 2.

4.2.6 Pattern Embedding Computation

In this section, we briefly describe how we computed the embeddings of PIHom patterns into target graphs. We avoid an in-depth discussion on a general algorithm for deciding partially injective homomorphisms from arbitrary PIHom patterns and instead focus on the case that patterns are generated as described in Algorithm 2.

Recall from Sect. 2.4.1 that deciding homomorphisms from graphs of treewidth k involves two essential steps. The first step consists of computing a *tree-decomposition* of width k. A tree-decomposition decomposes a graph G into a tree whose vertices correspond to specific induced subgraphs of G. Subsequently, this tree-decomposition is utilized to solve the decision problem (and even homomorphism counting problem) using a *dynamic programming* approach. In the following, we provide an outline of these two steps for the case that the

graphs are PIHOM patterns enumerated by Algorithm 2.

While, for a constant k, finding a tree-decomposition of width k for a graph with treewidth at most k can be done in linear time, the known algorithms tend to be practically infeasible for $k > 3$ (Bodlaender, 1993). Fortunately, we can avoid the costly tree-decomposition computations for PIHOM patterns. This follows from the algorithmic description of the pattern refinement operator (see Sect. 4.2.4), which can be used to directly derive tree-decompositions of patterns.

Recall, that a refinement step connects the newly introduced vertex v to a set S of vertices in a given pattern $H\langle\mathcal{C}\rangle$. If $n \leqslant k$, then $S = V(H\langle\mathcal{C}\rangle)$, otherwise S is a k-clique of $H\langle\mathcal{C}\rangle$. Let $H'\langle\mathcal{C}'\rangle$ be such a refinement of $H\langle\mathcal{C}\rangle$ and assume the tree-decomposition $TD(H\langle\mathcal{C}\rangle) = (T, X, r)$ to be known. Then, a tree-decomposition of $H'\langle\mathcal{C}'\rangle$ is obtained as follows:

(i) Select a bag $B_i \in X$ with $S \subseteq B_i$,

(ii) create a new bag $B_{r'} = S \cup \{v\}$, and

(iii) connect the corresponding new node r' to node i in T.

The resulting tree-decomposition of $H'\langle\mathcal{C}'\rangle$ is given by $(T', X \cup B_{r'}, r')$ where T' is the tree that is obtained by connecting r' to T. Trivially, the tree-decomposition of a singleton pattern consists of a single bag containing only one node. Thus, using the above procedure, tree-decompositions of PIHOM patterns can easily be acquired during the pattern refinement step described in Sect. 4.2.4.

The second step for deciding (resp. counting) homomorphisms from PIHOM patterns directly utilizes the tree-decompositions. Given a pattern $H\langle\mathcal{C}\rangle$ and its tree-decomposition, we can decide (and even count) partially injective homomorphisms from $H\langle\mathcal{C}\rangle$ using a dynamic programming approach. The corresponding algorithm makes use of the observation that $TD(H\langle\mathcal{C}\rangle) = (T, X, r)$ decomposes the pattern $H\langle\mathcal{C}\rangle$ into a set of subgraphs induced by the bags in X. The key idea is to iterate over the nodes in T in a bottom up manner (w.r.t. root $r \in V(T)$) and compute sets of partially injective homomorphisms from increasingly large subgraphs of $H\langle\mathcal{C}\rangle$ into the target graph G. For this purpose, every node $j \in v(T)$ is associated with an embedding count table, denoted $\#_j$, which tracks the amounts of partially injective homomorphisms from the subgraph induced by bag B_j. Finally, whether there exists a partially injective homomorphism $H \xrightarrow{\mathcal{C}} G$ is determined by the embedding count table $\#_r$ computed for the root r of T. For a more detailed description on counting homomorphisms from graphs of bounded treewidth, we refer to Sect. 2.4.1.

While this homomorphism counting algorithm has polynomial time complexity, it is nonetheless quite often practically inefficient. This practical inefficiency is mainly rooted in the overwhelming amount of mappings from induced subgraphs of $H\langle\mathcal{C}\rangle$, which need to be checked for whether they are partially injective homomorphisms or not. Fortunately, much of this computational effort can be avoided in our pattern mining scenario. This is achieved by reusing prior embedding information instead of naively applying the counting algorithm to every pattern from scratch. More precisely, let $H\langle\mathcal{C}\rangle$ be a pattern with $TD(H\langle\mathcal{C}\rangle) = (T, X, r)$ for which the partially injective homomorphism counting algorithm into some target graph G has already been invoked and all corresponding count tables $\#_j, j \in V(T)$ have been stored.

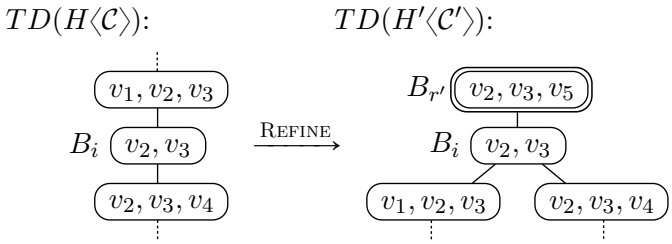


Figure 4.5: Consider a refinement $H'\langle\mathcal{C}'\rangle$ of pattern $H\langle\mathcal{C}\rangle$ introducing a new vertex v_5 which connects to vertices v_2 and v_3 (e.g, as displayed in Fig. 4.4). The tree-decomposition of $H'\langle\mathcal{C}'\rangle$ can simply be obtained by connecting a new bag $B_{r'}$ to an existing bag B_i in the tree-decomposition of $H\langle\mathcal{C}\rangle$ which contains vertices v_2, v_3.

Let $H'\langle\mathcal{C}'\rangle$ with $TD(H'\langle\mathcal{C}'\rangle) = (T', X', r')$ be a refinement of $H\langle\mathcal{C}\rangle$ introducing a new bag $B_{r'}$. We now copy the count tables $\#_j, j \in V(T)$ of $TD(H\langle\mathcal{C}\rangle)$ into $TD(H'\langle\mathcal{C}'\rangle)$. This can be done since we know that $V(T) = V(T')\backslash\{r'\}$. Next, we set the corresponding node r' as the root of T'. It is then sufficient to merely update the counts in $\#_j$ for all nodes $j \in V(T')\backslash\{r'\}$ of $TD(H'\langle\mathcal{C}'\rangle)$. This can be done by simple joins without the need to check for whether embeddings are homomorphisms or not. Finally, only for the root $r' \in V(T')$, new partially injective homomorphisms need to be computed.

An illustration of a PIHom pattern's tree-decomposition refinement can be found in Fig. 4.5. For the newly introduced vertex v_5 which connects to v_2 and v_3 by either a blue or red edge, a new bag $B_{r'} = \{v_2, v_3, v_5\}$ is added to the tree-decomposition. The corresponding node r' is connected to a node i with B_i containing v_2 and v_3. Finally, r' becomes the root of $TD(H'\langle\mathcal{C}'\rangle) = (T', X', r')$. In order to compute the embeddings from the refined graph $H'\langle\mathcal{C}'\rangle$ into some target graph G, it suffices to reevaluate the entries in the embedding count tables in bottom-up manner for all nodes in T' up to node i. Since the embedding count table $\#_i$ now contains all possible mappings for v_2, v_3, it only needs to be checked for what mappings of v_5 this can be extended.

4.3 The Kernel Function

Frequent tree patterns have proven to be a powerful graph feature in terms of their predictive capabilities (Welke, Horváth, and Wrobel, 2017). To evaluate the predictive performance of our PIHom patterns, we now define a general kernel method which compares graphs based on the set of mutual frequent patterns.

Using the above concepts, we can embed graphs into the binary feature space spanned by frequent PIHom patterns and define a graph similarity by the simple dot product in that space. More formally, let $\mathcal{O}$ be a set of PIHom patterns outputted by Alg. 2. Then for $G, G' \in \mathcal{G}$, the *frequent pattern kernel* is defined by

$$k(G, G') = |\{(H, \mathcal{C}) \in \mathcal{O} : H \xrightarrow{\mathcal{C}} G \wedge H \xrightarrow{\mathcal{C}} G'\}|. \tag{4.4}$$

Hence, the kernel defines graph similarity in terms of the number of mutually occurring frequent PIHom patterns.

4.4 Experimental Evaluation

This section is concerned with the empirical evaluation of the kernel method described in Sect. 4.3. In particular, we evaluate the predictive performance of frequent maximally constrained PIHom tree patterns and the runtime of the algorithm as outlined in Sect. 4.2.5. In the experiments below, we investigate the following three research questions:

(i) We compare the predictive performance of the patterns generated by our algorithm with that of *ordinary* frequent subtrees on different benchmark datasets. We found that the predictive performance achieved by PIHom patterns for moderate amounts of injectivity constraints already compares favorably to that of obtained by utilizing ordinary frequent subtrees.

(ii) We, furthermore, analyze how the *degree of injectivity* in PIHom patterns impacts the resulting predictive performance of the kernel method. To answer this question, we fix a set of tree patterns and consider partially injective homomorphisms with increasing degrees of injectivity, ranging from ordinary homomorphism to subgraph isomorphism. Our results clearly indicate a strong correlation between degree of injectivity and predictive performance.

(iii) Finally, we provide *runtime* measures comparing our algorithm to the graph miners GASTON (Nijssen and Kok, 2005) and FSG (Deshpande et al., 2005). We show that while these algorithms are practical only for very restricted graph types, our approach performs well on *arbitrary* graph datasets. In particular, while our implementation was generally slower on the considered molecular benchmark datasets, it clearly outperforms GASTON and FSG on artificial datasets containing only slightly more complex graph structures beyond molecular graphs.

To evaluate the predictive performance, we restrict the discussion to the molecular real-world benchmark datasets DHFR, MUTAG, NCI1, and PTC-MR. This choice is mainly of practical nature. In particular, the graphs considered are of a fairly simple structure (Horváth and Ramon, 2010), allowing for the application of state-of-the-art frequent subgraph mining systems that are generally very efficient on molecular graph data, but quickly become practically infeasible for slightly more complex graphs. Thus, the use of molecular datasets allows for a proper comparison of the predictive performance of our PIHom patterns to "gold-standard" results achieved by ordinary frequent subgraphs (obtained by utilizing subgraph isomorphism).

For all frequent pattern generation methods, we chose a frequency threshold value of 5%, which has proven to be a suitable choice in prior experiments. Furthermore, prior experiments have shown that patterns of fairly low sizes achieve the overall best predictive performances. We therefore limit the size of patterns to at most 8 vertices. In fact, in many cases the results even decrease for larger patterns.

The predictive performances are reported in terms of accuracy obtained by support vector machines (SVM) using a 5-fold cross-validation. The SVM parameter C is selected from the value set $2^i : i \in \{-12, -8, -5, -3, -1, 1, 3, 5, 8, 12\}$. We report the mean and standard deviation over 10 such cross-validation repetitions.

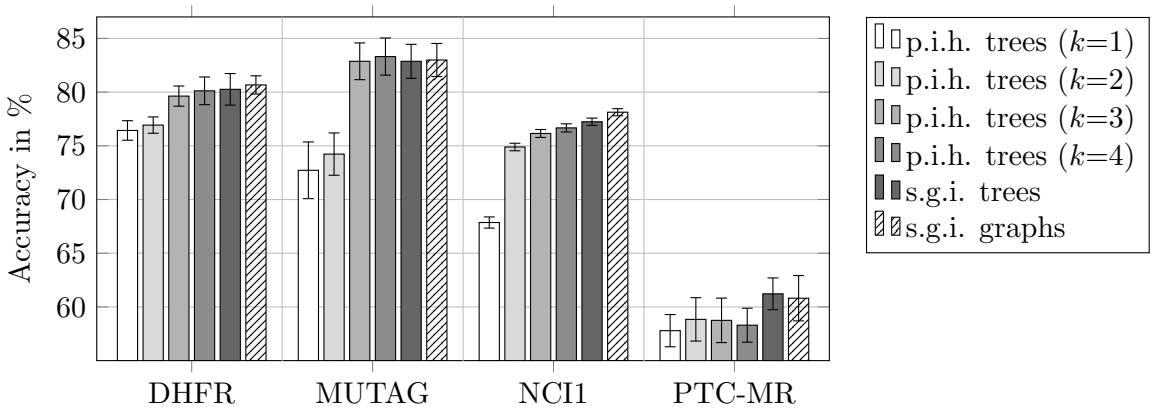


Figure 4.6: Prediction measures for different treewidth choices k in contrast to frequent subgraphs and subtrees (s.g.i.: subgraph isomorphism, p.i.h.: partially injective homomorphism).

	DHFR	MUTAG	NCI1	PTC-MR
p.i.h. trees ($k = 1$)	36975	6852	17659	13956
p.i.h. trees ($k = 2$)	25465	3551	10043	7573
p.i.h. trees ($k = 3$)	17521	2109	5969	3142
p.i.h. trees ($k = 4$)	10615	1160	3666	1701
s.i. trees	4009	421	1360	484
s.i. graphs	4061	432	1411	470

Table 4.1: Numbers of frequent patterns for each pattern type in the respective datasets.

4.4.1 Predictive Performance

In order to evaluate the predictive performance of the frequent maximally constrained PIHom patterns generated by our algorithm, we compare their predictive power to that achieved by the set of (ordinary) frequent subtrees as well as frequent subgraphs for which the embedding operator is subgraph isomorphism. For all pattern sets, we provide the accuracy values obtained by the frequent pattern kernel (see Sect. 4.3).

Fig. 4.6 shows the predictive performance for different degrees of injectivity governed by the treewidth parameter k. Apart from the dataset PTC-MR, it is apparent that increasing values of k lead to overall better accuracy values. The results on MUTAG and NCI1 specifically show that patterns mined w.r.t. homomorphism (i.e., $k = 1$) result in comparatively poor predictive performances. However, already for treewidth values as low as $k = 3$, the results are very close to those obtained by subgraphs ("s.g.i. graphs") and ordinary frequent subtrees ("s.g.i. trees"). Hence, our approach offers an attractive trade-off between complexity of the pattern embedding operator (depending on k) and predictive power.

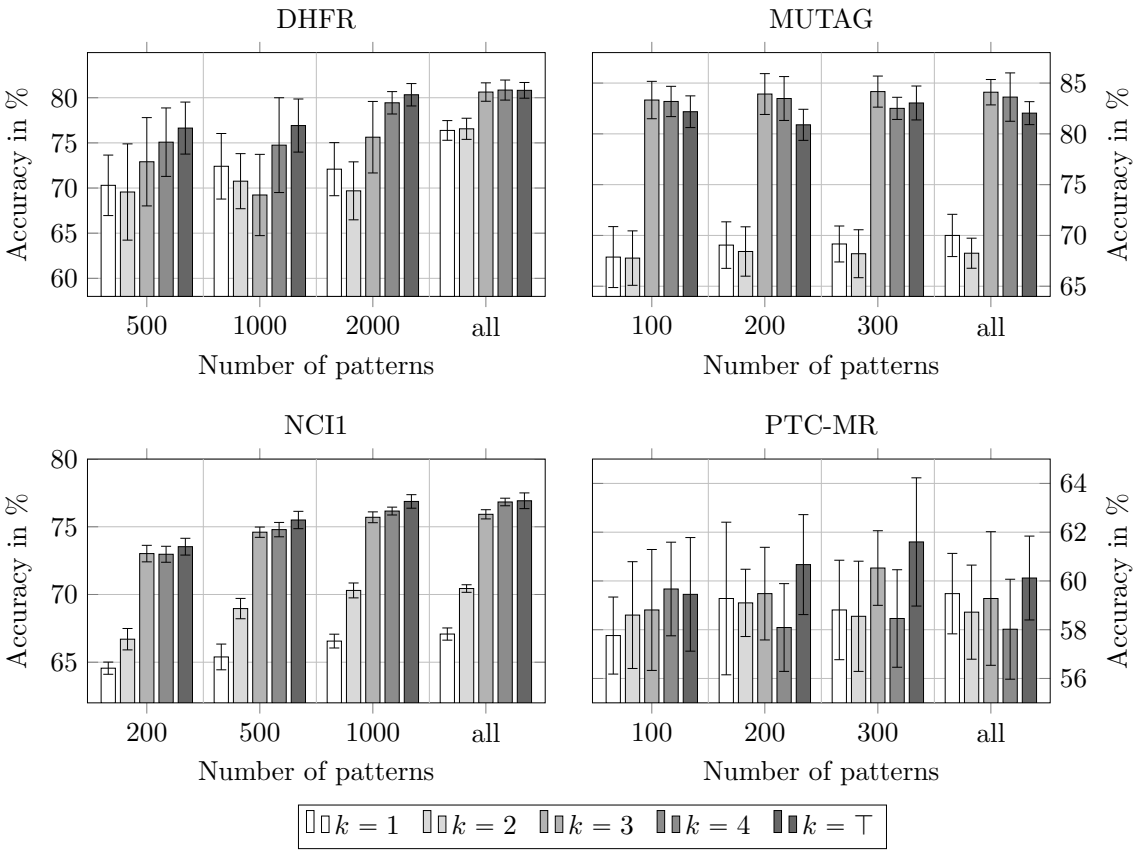


Figure 4.7: Predictive performances for different degrees of injectivity governed by treewidth k. $k = \top$ corresponds to ordinary subgraph isomorphism.

The amounts of frequent PIHom patterns returned by our algorithm as well as those of ordinary frequent subtrees and subgraphs up to size 8 can be found in Tab. 4.1. Since utilizing partially injective homomorphism for lower treewidth values k leads to less restrictive pattern embedding operators, the numbers of frequent PIHom patterns decrease with increasing values of k. We note that the sets of frequent PIHom patterns for $k \in \{2, 3, 4\}$ are subject to some degree of randomness caused by the enumeration algorithm variant described in Sect. 4.2.5. However, prior experiments showed that the numbers did not significantly differ for repeated invocations of the mining algorithm.

4.4.2 Degree of Injectivity vs. Predictive Performance

We now specifically analyze the influence of the degree of injectivity in PIHom tree patterns on the predictive performance. To exclude possible side-effects caused by different pattern sets, we first fix a set S of tree patterns for all datasets in our experiments by selecting some random subset of the frequent trees generated w.r.t. subgraph isomorphism. Then, for each

tree pattern $H \in S$, we consider a PIHom tree pattern $H_k = (H, \mathcal{C}_k)$ that is maximally constrained w.r.t. treewidth k. Since H is a tree, $\mathcal{C}_1 = \varnothing$ and hence H_1 is the least element in the lattice $(\mathcal{L}_H, \preceq)$, corresponding to ordinary homomorphism from H. Analogously, $H_\top$ contains all possible injectivity constraints and is therefore the greatest element of $(\mathcal{L}_H, \preceq)$, corresponding to ordinary subgraph isomorphism from H. In this way, we can simulate monotonically increasing degrees of injectivity in the pattern matching operator for a tree pattern H, leading from ordinary homomorphisms to subgraph isomorphisms. The degree of injectivity in the pattern matching operator is directly governed by k. For each $k \in \{1, \ldots, 4, \top\}$ we consider the feature set $S_k = \{H_k : H \in S\}$ and evaluate the predictive performance.

Figure 4.7 shows the predictive performances achieved for different degrees of injectivity governed by k on several tree pattern sets S. With PTC-MR being an exception, all other datasets show a significant difference between employing homomorphism ($k = 1$) and subgraph isomorphism ($k = \top$) as the pattern matching operators. Concerning PTC-MR, this correlation can only vaguely be observed due to the large standard deviations. The datasets DHFR, MUTAG, and NCI1, however, show a close correlation between the degree of injectivity and predictive performance. Increasing values of k in most cases yield an improvement in predictive performance. Notice that the gap between $k = 2$ and $k = 3$ is substantial for the datasets MUTAG and NCI1. For MUTAG a choice of treewidth 3 even leads to the overall best predictive performance. In summary, it can be observed that already fairly low degrees of injectivity suffice to considerably outperform the predictive power of ordinary homomorphism and approximate that of subgraph isomorphism.

4.4.3 Runtime Analysis

We measured the runtimes of our algorithm and compared them to those achieved with the graph miners GASTON (Nijssen and Kok, 2005) and FSG (Deshpande et al., 2005). GASTON is a depth-first search frequent pattern enumerator that operates by first generating paths, which are subsequently extended into trees and finally into cyclic graphs. We employed GASTON to mine the set of frequent tree patterns which can be done using a command line option. There are two variants of the tool that differ by whether or not the embedding lists are explicitly stored in memory. Thus, the *embedding lists* (EL) and the *recomputed embeddings* (RE) methods are essentially trade-offs between runtime and memory requirement. The FSG algorithm enumerates frequent patterns in a breath-first search manner. It needs to be noted, that FSG outputs all frequent *subgraphs*, and not only frequent subtrees as in all other approaches considered in this chapter.

Since frequent subgraph mining systems like GASTON (Nijssen and Kok, 2005) seem to be specifically designed to cope with graphs of simple structure (such as molecular graphs), runtime comparisons on only chemical graph datasets are not expressive enough. We therefore consider also artificial graph datasets generated according to the Erdős-Rényi random graph model. We consider several such datasets consisting of 50 graphs with an average of 25 vertices, similar to the chemical datasets. For each dataset, we consider different node-to-edge ratios to investigate the behavior of the mining algorithms on graphs of various structural complexity. Note that only connected graphs are considered in our experiments.

	DHFR	MUTAG	NCI1	PTC-MR
GASTON (EL)	0.8	0.1	2.5	0.1
GASTON (RE)	3.5	0.3	8.5	0.1
FSG	32.2	0.7	30.5	0.4
PIH Miner ($k = 3$)	202.2	5.1	229.1	7.7

Table 4.2: Runtimes (in seconds) of our algorithm in comparison to GASTON and FSG on molecular datasets for mining all patterns up to size 10.

	Unlabeled				Labeled			
Node-to-edge ratio	1 : 1	1 : 1.5	1 : 2	1 : 3	1 : 1	1 : 1.5	1 : 2	1 : 3
GASTON (EL)	2.8	54.6	-	-	1.8	20.1	1168.9	56464.6
GASTON (RE)	5.0	39.5	1163.0	31061.4	5.7	44.9	1120.2	24778.0
FSG	194.2	10584.9	10888.4	10852.5	19.9	82.8	2375.7	58816.2
PIH Miner ($k = 3$)	1.1	2.7	8.4	23.2	6.8	20.8	160.8	568.9

Table 4.3: Runtimes (in seconds) of our algorithm in comparison to GASTON and FSG on Erdős-Rényi random graphs for mining all patterns up to size 10. Experiments were conducted for unlabeled as well as labeled graphs where nodes were randomly assigned one of two colors. Cases that did not finish due to insufficient memory are marked by "-".

While molecules have roughly as many vertices as edges (i.e., ratio 1:1), we consider ratios of up to 1:3 in the artificial datasets. All experiments were performed on an Intel i7-4770 processor (4 cores) with 16GB of memory.

Table 4.2 and 4.3 show the running times for each algorithm and respective dataset. As expected, GASTON and FSG perform very well on molecular graphs, compared to our algorithm. However, for slightly more complex structures, i.e. Erdős-Rényi with node-to-edge ration 1:2, the two traditional graph miners become quickly infeasible. Our approach (referred to as *PIH Miner*) clearly outperforms FSG and GASTON on both artificial datasets for node-to-edge ratios above 1:1.5.

4.5 Summary and Concluding Remarks

As a unifying view of ordinary graph homomorphisms and subgraph isomorphisms, we proposed the concept of partially injective homomorphisms, a new kind of parameterized pattern matching operator. We defined a class of efficiently decidable partially injective homomorphisms by extending tree patterns into bounded treewidth graphs and proposed an efficient mining algorithm enumerating frequent constrained PIHom patterns that are maximal w.r.t. bounded treewidth. The experimental results showed that the predictive performance obtained using this kind of patterns is close to that of ordinary frequent subtrees (and hence, to that of subgraphs as well) on a range of real-world datasets. Furthermore, it could be ob-

served that there is a direct correlation between partial injectivity in the pattern embedding process and predictive performance of the obtained patterns, thus making partially injective homomorphism an attractive choice for an efficiently decidable pattern embedding operator.

While we utilized trees as the choice of pattern class (i.e., the element H in $(H, \mathcal{C})$ is a tree), we note that this is not a necessary restriction. In fact, our proposed method is applicable to any type of graph H as long as ordinary homomorphisms from the edge extended graph $H\langle\mathcal{C}\rangle$ can be efficiently decided.

An interesting research question is concerned with the extension of our approach to more general relational structures. As graphs can be considered very simple relational structures, it is an open question what implications our results have to inductive logic programming (Nienhuys-Cheng and Wolf, 1997). More precisely, $\text{PIHOM}(H, G, \mathcal{C})$ problems can polynomially be reduced to θ-subsumptions between DATALOG goal clauses (or equivalently, *boolean conjunctive queries*). From this reduction and our empirical results it is immediate that the predictive performance of DATALOG goal clauses as patterns may also be improved by adding injectivity constraints (i.e., literals of a distinguished binary predicate) to their bodies.

Finally, we note that our approach suffers from two drawbacks. One central issue lies within the fact that although Algorithm 2 enumerates patterns with incremental polynomial delay, it, nonetheless, generally has an exponential total runtime due to the *exponential number* of potentially frequent patterns. We, however, note that the algorithm may be terminated any time and is therefore capable of providing a non-empty output in feasible time. Furthermore, it needs to be noted that the algorithm for deciding homomorphism from graphs of bounded treewidth is often practically infeasible despite its theoretical polynomial runtime. In order to address both issues, in the next chapters, we consider a pattern extraction process which avoids both disadvantages using a conceptually different kind of constrained homomorphism as the pattern embedding operator.

5

Weisfeiler-Lehman Filtration Kernels

The previous chapter introduced a method that utilized a tree pattern embedding operator based on a specific kind of constrained homomorphism. While we showed that such partially injective homomorphisms can be decided in polynomial time for a certain class of patterns, there remain some disadvantages associated with this approach. As previously stated, the main concerns that stem from this method include the lack of practical efficiency of the decision algorithm as well as the uncontrollable amount of frequent patterns.

In an effort to address these issues, in this chapter, we make use of the popular Weisfeiler-Lehman method (Weisfeiler and Lehman, 1968). The Weisfeiler-Lehman vertex relabeling scheme was originally designed as a means to decide graph isomorphism with one-sided error. It follows a label propagation approach that iteratively compresses the label of a vertex and that of its neighbors into a new vertex label (see Sect. 2.7). These Weisfeiler-Lehman labels have become a popular node descriptor due to the efficiency of the extraction method as well as the remarkable predictive performance of this kind of pattern language (Shervashidze et al., 2011).

A most interesting aspect of the Weisfeiler-Lehman method is that the generated vertex labels correspond to *rooted trees*, known as *unfolding trees*. The Weisfeiler-Lehman vertex relabeling process implicitly constructs such unfolding trees and furthermore induces a pattern embedding operator on this kind of trees. Specifically, unfolding trees are embedded into target graphs by *locally bijective homomorphisms*, which are certain types of *constrained homomorphisms* (see Sect. 2.7.1). The determining advantage of the Weisfeiler-Lehman method is that the identification of labels, or equivalently unfolding tree patterns, can be done very efficiently. In fact, unlike the approach presented in Chapter 4, the extraction of Weisfeiler-Lehman patterns requires neither explicit computations of the embedding operator nor costly explorations of the pattern space.

A drawback of the Weisfeiler-Lehman method, however, is that it generates overly specific features. More precisely, Weisfeiler-Lehman vertex labels are often unique within a set of graphs and are thus unsuitable for graph comparisons in terms of mutually occurring features. The specificity stems from the fact that the Weisfeiler-Lehman labels (or equivalently, unfolding trees) encode k-hop neighborhoods of vertices which are embedded via locally bijective homomorphisms. Specifically, a certain unfolding tree T can be embedded into a given

target graph G by a locally bijective homomorphism only if G contains a vertex v such that T *precisely* encodes the k-hop neighborhood of v.

As this kind of pattern matching of unfolding trees is arguably too restrictive, we propose an approach which relaxes this strict pattern embedding process. This is effectively achieved by considering only *subsets* of edges in the target graphs. The key idea is that an unfolding tree T which cannot be embedded into a graph $G = (V, E)$ may nonetheless be embedded into a graph $G' = (V, E' \subseteq E)$. Fig. 5.1 shows an example of this concept. While the simple depth-1 Weisfeiler-Lehman label corresponding to a vertex with a single neighbor is not contained in G, it can nonetheless be embedded into its subgraphs G_2, G_3 and G_4.

In this chapter, we construct several such subgraphs using concepts from computational topology (Edelsbrunner and Harer, 2010). By using meaningful orders on the edges, we replace a graph by a sequence of nested subgraphs, called *filtration*. More precisely, a filtration of a graph $G = (V, E)$ is a sequence of graphs $G_1, \ldots, G_k = G$ with evolving sets of edges over the same set of vertices, i.e., $G_i = (V, E_i)$ with $E_i \subseteq E$ and $E_i \subseteq E_j$ whenever $i \leqslant j$ for $i, j \in [k]$. The graphs in such a sequence can be interpreted as *multiple levels of resolution* and describe how G is incrementally constructed by adding sets of edges. Clearly, with changing sets of edges, the node neighborhoods and thus the Weisfeiler-Lehman labels change as well. By extracting Weisfeiler-Lehman patterns over graph filtrations, we define powerful graph representations. When compared to the ordinary set of Weisfeiler-Lehman labels in a graph G, the advantages of considering graph filtrations of G are twofold: For one, considering filtration graphs $G_i, i \in [k]$ allows for the extraction of patterns that are not necessarily present in the original graph G. Secondly, instead of simply representing a graph by the set of its Weisfeiler-Lehman patterns, we can now represent it using *existence intervals* of patterns over filtrations.

Using the above concept, we present a novel graph kernel that defines graph similarity by comparing existence intervals of Weisfeiler-Lehman labels. This comparison is realized using a Wasserstein distance on histograms representing aggregated existence intervals. As one of the central contributions of this chapter, we show that this specific kind of distance measure in fact yields proper graph kernel functions, the *Weisfeiler-Lehman filtration kernels*. This type of kernel has several interesting properties. For one, we prove that it generalizes the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) while increasing its computational complexity only by a factor corresponding to the length of the filtration sequence. Furthermore, we show that for certain choices of filtrations, the kernel is in fact a *complete* graph kernel, i.e., it is capable of distinguishing all non-isomorphic graphs.

While our focus lies on considering Weisfeiler-Lehman labels as graph features, we show that our approach can just as well be applied using any other kind of graph feature. We therefore introduce two types of graph filtration kernels and prove that they generalize standard graph kernel frameworks.

In an experimental evaluation, we empirically demonstrate the favorable performance of the Weisfeiler-Lehman filtration kernel on real-world datasets compared to state-of-the-art graph kernels. We furthermore validate our theoretical results on the kernel's ability to distinguish non-isomorphic graphs, and finally report runtimes.

5.1 From Filtrations to Distances

In this section, we formally define graph filtrations and show how occurences of graph features are tracked over such sequences of graphs. Subsequently, we define a distance measure between graphs using this kind of information. While we primarily focus on the case that the graph features are Weisfeiler-Lehman labels, we introduce the following concepts in a general fashion allowing our approach to be applied to any kind of graph feature (see Sect. 5.3).

5.1.1 Feature Persistence

The idea of tracking feature occurences over filtrations originates from *persistent homology*, which refers to a method in computational topology that aims at measuring topological features at various resolution levels (Edelsbrunner and Harer, 2010). Persistent homology is applied in a wide range of topological data analysis tasks and has recently become a popular tool for analyzing topological properties in graphs (Aktas, Akbas, and Fatmaoui, 2019). In the following, we adopt some of its basic concepts and specifically fit them to describe the idea of *feature persistence*.

Intuitively speaking, feature persistence tracks the *lifespans* of graph features in evolving graphs. That is, it records the intervals during which occurrences of a specific feature appear in sequentially constructed graphs. Such a graph sequence is defined by a graph filtration which is essentially an ordered graph refinement that constructs a graph by gradually adding sets of edges. More precisely, given a graph $G = (V, E)$, a *graph filtration* $\mathcal{F}(G)$ is a sequence of graphs

$$G_1 \subseteq G_2 \subseteq \ldots \subseteq G_k = G \ , \tag{5.1}$$

where $\subseteq$ denotes the subgraph relation and $G_i = (V, E_i \subseteq E), i \in [k]$ is called a *filtration graph* of G. Hence, filtration graphs differ only in the sets of edges and describe a sequence in which the last element is the graph G itself. Without loss of generality, we assume G to be edge-weighted by a function $\omega : E(G) \to \mathbb{R}_{\geqslant 0}$ such that the filtration $\mathcal{F}(G)$ is implicated by a sequence of decreasing real values

$$\alpha_1 > \alpha_2 > \ldots > \alpha_k = 0 \ , \tag{5.2}$$

where G_i contains the set of edges with weights greater or equal α_i. In other words,

$$G_i = (V, \{e \in E : \ \omega(e) \geqslant \alpha_i\}) \ . \tag{5.3}$$

Thus, function $\mathcal{F}$ is determined by a *threshold sequence* $\mathcal{F}_\alpha = \{\alpha_1, \ldots, \alpha_k\}$. Clearly, given some filtration function $\mathcal{F}$ and isomorphic graphs G, G' (considering edge weights as well), it holds that the graphs in $\mathcal{F}(G)$ and $\mathcal{F}(G')$ must be pairwise isomorphic, i.e., $G_i \equiv G'_i$ for all $i \in [k]$.

While traditional persistent homology tracks lifespans of *topological* features such as connected components and cycles, our notion of feature persistence is concerned with *arbitrary* graph features. For the specific case of Weisfeiler-Lehman labels, feature persistence describes the set of occurrence intervals of a certain Weisfeiler-Lehman label ℓ over the sequence $\mathcal{F}(G)$. This concept can very intuitively be depicted using (discrete) persistence barcodes, as shown

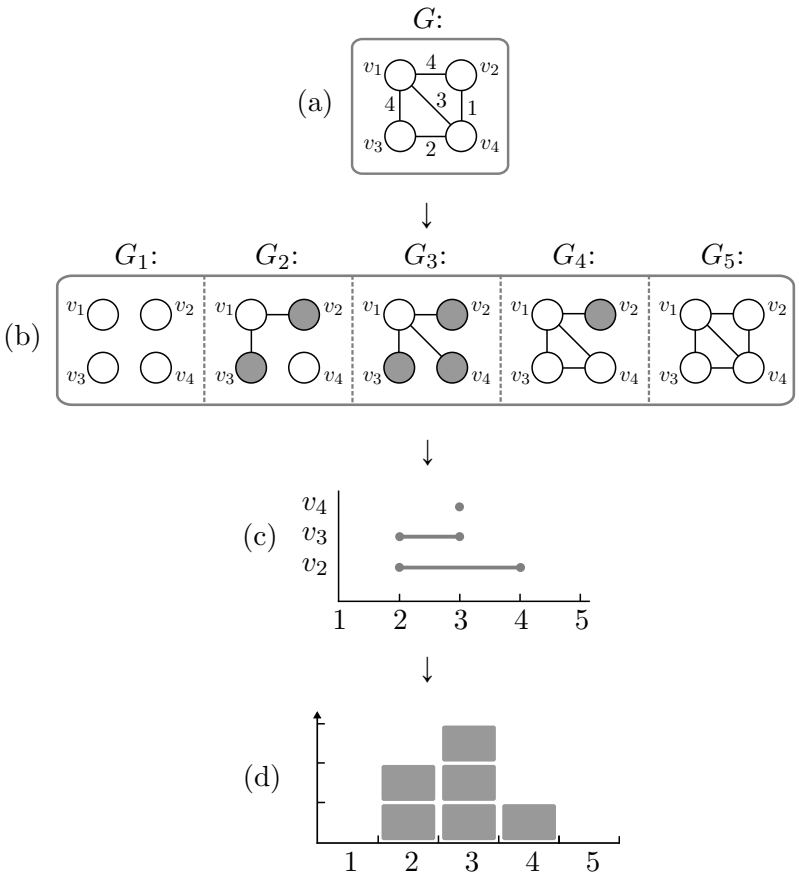


Figure 5.1: Consider the simple depth-1 Weisfeiler-Lehman label ℓ corresponding to a vertex having exactly one neighbor. Each vertex with this label ℓ is individually marked in the filtration graphs $\mathcal{F}(G)$ shown in (b). The barcode in (c) depicts the existence intervals of ℓ for each vertex in G. This information is then aggregated into a filtration histogram $\phi_\ell^{\mathcal{F}}(G)$ in (d).

in Fig. 5.1. Each bar in the barcode diagram corresponds to the lifespan of an occurrence of the very simple depth-1 Weisfeiler-Lehman label ℓ corresponding to a vertex having exactly one neighbor. Clearly, there are several vertices which obtain this specific label during the filtration. Consequently there exists a bar for each such vertex v_2, v_3 and v_4.

5.1.2 The Wasserstein Distance on Filtration Histograms

The majority of traditional graph kernels, including the Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011), defines similarity measures in terms of the number of mutual substructures. This comes down to simply comparing frequencies of features. Using the concept of feature persistence, we are able to define much finer similarity measures on graphs. We achieve this by defining a distance function on histograms which aggregate lifespans of feature occurrences. This distance measure compares graphs in terms of *when* and *for how*

long a certain feature appears in the filtration. The underlying intuition is that features occurring close to each other in the filtration sequence indicate a higher similarity than those lying farther apart. A natural choice for this distance function is the Wasserstein distance, as the aggregated feature occurrence lifespans directly translate into 1-dimensional distributions.

To define a distance measure w.r.t. *a single* feature ℓ on graphs G, G', we first aggregate the feature persistence information of G and G' into histograms. This aggregation is visualized in Fig. 5.1(d) for a graph and a feature ℓ. Such histograms accumulate all feature lifespans of a particular feature ℓ and reflect the number of feature occurrences in each filtration graph.

Definition 5.1 (Filtration Histogram). *For a graph $G \in \mathcal{G}$, length-k filtration function $\mathcal{F}$, and a feature ℓ, the function $\phi_\ell^{\mathcal{F}} : \mathcal{G} \to \mathbb{R}^k$ maps G to its* filtration histogram *which counts the number of ℓ in each filtration graph of $\mathcal{F}(G)$.*

We acknowledge, that aggregating all feature occurrence intervals into a single histogram clearly loses information on the distribution of the individual lifespans. However, while a pairwise comparison of persistence intervals has been shown to lead to valid kernels in the context of persistent homology (Reininghaus et al., 2015), our approach relies on a single histogram representation, which we show leads to very powerful kernel functions, nonetheless.

In the following, we sometimes omit the filtration function $\mathcal{F}$ in the notations if it is either irrelevant or clear from the context. That is, we simply write ϕ_ℓ instead of $\phi_\ell^{\mathcal{F}}$. Filtration histograms allow for the application of natural distance measures such as the *Wasserstein distance*. Intuitively speaking, the Wasserstein distance between such histograms describes the cost of shifting (accumulated) feature lifespans into another.

Definition 5.2 (Filtration Histogram Distance). *Given graphs $G, G' \in \mathcal{G}$, a filtration histogram mapping $\phi_\ell^{\mathcal{F}} : \mathcal{G} \to \mathbb{R}^k$ together with a distance function $d : \mathcal{F}_\alpha \times \mathcal{F}_\alpha \to \mathbb{R}$ on associated values $\mathcal{F}_\alpha = \{\alpha_1, \ldots, \alpha_k\}$, the* filtration histogram distance *is given by*

$$\mathcal{W}_d(\phi_\ell^{\mathcal{F}}(G), \phi_\ell^{\mathcal{F}}(G')) \quad . \tag{5.4}$$

Here $\mathcal{W}_d$ is the Wasserstein distance equipped with ground distance d (see Sect. 2.10). The ground distance d defines how feature occurrences at different points in the filtrations are being compared to each other. Since the values in $\mathcal{F}_\alpha$ can be viewed as points on the timeline $[\alpha_1, \alpha_k]$, a natural choice for this distance is the Euclidean distance in $\mathbb{R}^1$, i.e.,

$$d^1(\alpha_i, \alpha_j) = |\alpha_i - \alpha_j| \tag{5.5}$$

for all $\alpha_i, \alpha_j \in \mathcal{F}_\alpha$. While the Wasserstein distance has cubic time complexity in the length of filtrations in general, this reduces to a *linear* time complexity when employing d^1 on the real line as ground distance (Peyré and Cuturi, 2019, Rem. 2.30). Plugging d^1 into Eq. 5.4 yields the filtration histogram distance used throughout this chapter.

5.2 The Weisfeiler-Lehman Filtration Kernel

In this section, we define the Weisfeiler-Lehman filtration kernel which generalizes the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011). We start by defining a

base kernel function which defines graph similarity w.r.t. a specific Weisfeiler-Lehman label and describe how such kernels are *combined* to form the Weisfeiler-Lehman filtration kernel. Subsequently, we show that our kernel's runtime complexity increases that of the ordinary Weisfeiler-Lehman subtree kernel only by a *linear* factor. Finally, we discuss its expressive power and show that our kernel function is *complete* when choosing suitable filtrations.

Results in optimal transport theory give rise to proper kernel functions using the above filtration histogram distances when equipped with a suitable ground distance function (Le et al., 2019). In fact, it can be shown that utilizing the Euclidean ground distance d^1 yields positive semi-definite kernels. These kernels serve as *building blocks* for our Weisfeiler-Lehman filtration kernel. More precisely, we construct graph kernels by combining multiple base kernels κ_ℓ over the set of Weisfeiler-Lehman labels $\ell \in \mathcal{L}_h$, where $\mathcal{L}_h$ denotes set of all Weisfeiler-Lehman labels up to depth h. Each such base kernel is concerned with a single label and defines a similarity between graphs G and G' w.r.t. this particular label ℓ.

In general, Wasserstein distances require that the histograms are of equal mass. We therefore mass-normalize each histogram by dividing all of its entries by its original mass, and denote such mass-normalized filtration histograms using function $\hat{\phi}_\ell^{\mathcal{F}}$. More formally:

$$\hat{\phi}_\ell^{\mathcal{F}}(G) = \frac{1}{\left\|\phi_\ell^{\mathcal{F}}(G)\right\|_1} \phi_\ell^{\mathcal{F}}(G) \tag{5.6}$$

The base kernel on graphs G, G' w.r.t. label ℓ and value $\gamma \in \mathbb{R}_+$ is then defined over normalized histograms by

$$\kappa_\ell^{\mathcal{F}}(G, G') = e^{-\gamma \mathcal{W}_{d^1}(\hat{\phi}_\ell^{\mathcal{F}}(G), \hat{\phi}_\ell^{\mathcal{F}}(G'))} \quad . \tag{5.7}$$

This base kernel essentially "transforms" the histogram distance (Eq. 5.4) with d_1 as the ground distance into a proper kernel. It utilizes the result that the term $e^{-\gamma g(x,y)}$ is positive semi-definite for certain choices of function g (see Sect. 2.8). For a proof of the base kernel's positive semi-definiteness, we refer to Sect. 5.3.

The Weisfeiler-Lehman filtration kernel is defined as a *linear combination* of base kernels. That is, it is a sum of kernels $\kappa_\ell^{\mathcal{F}}$ over features $\ell \in \mathcal{L}_h$. Note that the mass-normalization of filtration histograms results in a loss of information on the number of label occurrences. This frequency information is often quite crucial. By introducing weights corresponding to the original histogram masses, this information loss can be in part reverted. That is, we weigh each base kernel $\kappa_\ell(G, G')$ using the original histogram masses of $\phi_\ell^{\mathcal{F}}(G)$ and $\phi_\ell^{\mathcal{F}}(G')$, i.e., $\left\|\phi_\ell^{\mathcal{F}}(G)\right\|_1$ resp. $\left\|\phi_\ell^{\mathcal{F}}(G')\right\|_1$. We can now define the kernel:

Definition 5.3 (Weisfeiler-Lehman Filtration Kernel). *Given graphs $G, G' \in \mathcal{G}$, a filtration function $\mathcal{F}$, and depth parameter h, the* Weisfeiler-Lehman filtration kernel *is given by*

$$k_{\mathcal{L}_h}^{\mathcal{F}}(G, G') = \sum_{\ell \in \mathcal{L}_h} \kappa_\ell^{\mathcal{F}}(G, G') \ \left\|\phi_\ell^{\mathcal{F}}(G)\right\|_1 \left\|\phi_\ell^{\mathcal{F}}(G')\right\|_1 . \tag{5.8}$$

A notable aspect of the Weisfeiler-Lehman filtration Kernel is that it reduces to the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) for the case $k = 1$,

i.e., when the filtration has length one. A proof of this statement as well as a proof for the positive semi-definiteness of the kernel can be found below in Sect. 5.3.

Recall that filtrations define an order in which edges are successively added until the final graph is obtained. During this sequence, neighborhoods of vertices evolve and with them their Weisfeiler-Lehman labels change. By also considering labels appearing only in filtration graphs, such labels allow to consider *partial* neighborhoods which contribute to a *finer* similarity measure. Fig. 5.1 depicts an example of a filtration where appearances of a specific Weisfeiler-Lehman label are being tracked. While the graph G does not contain this particular label, there are several occurrences of it in the filtration sequence.

The Weisfeiler-Lehman filtration kernel can be efficiently computed. In fact, it increases the complexity of the ordinary Weisfeiler-Lehman subtree kernel merely by the factor k, i.e., the length of the filtration.

Theorem 5.1. *The Weisfeiler-Lehman filtration kernel $k^{\mathcal{F}}_{\mathcal{L}_h}(G, G')$ on graphs G, G' can be computed in time $O(hkm + hkn)$, where h is the Weisfeiler-Lehman depth parameter, k is the length of filtration $\mathcal{F}$, and m, n denote the number of edges, resp nodes.*

Proof. The Weisfeiler-Lehman algorithm has complexity $O(hm)$. This algorithm needs to be computed for each of the k filtration graphs, leading to a total runtime of $O(hkm)$. For graphs G, G', let $\mathcal{L} \subseteq \mathcal{L}_h$ be the set of mutual Weisfeiler-Lehman labels appearing in both $\mathcal{F}(G)$ and $\mathcal{F}(G')$. Notice that the cardinality of $\mathcal{L}$ is bounded by hkn. Thus, for $H \in \{G, G'\}$, setting up all histograms $\phi_\ell(H), \ell \in \mathcal{L}$ can be done in time $O(hkm + hkn)$. To compute the kernel of Eq. 5.8, we need to calculate $\kappa_\ell(G, G')$ and thus $W_{d^1}(\hat{\phi}_\ell(G), \hat{\phi}_\ell(G'))$ for all $\ell \in \mathcal{L}$. For each label $\ell \in \mathcal{L}$, the distance $W_{d^1}(\hat{\phi}_\ell(G), \hat{\phi}_\ell(G'))$ can be calculated in time *linear* in the number of *non-zero* entries of both histograms $\hat{\phi}_\ell(G)$ and $\hat{\phi}_\ell(G')$ (see Sect. 2.10). Let the number of non-zero entries in $\hat{\phi}_\ell(G)$ be denoted by $c_\ell(G)$. Consequently, $W_{d^1}(\hat{\phi}_\ell(G), \hat{\phi}_\ell(G'))$ can be computed in time $O(c_\ell(G) + c_\ell(G'))$. For $H \in \{G, G'\}$, it holds that the sum of non-zero entries over all $\ell \in \mathcal{L}$ is bounded by hkn, i.e., $\sum_{\ell \in \mathcal{L}} c_\ell(H) \leqslant hkn$. Therefore, it must hold that the set of Wasserstein distances $W_{d^1}(\hat{\phi}_\ell(G), \hat{\phi}_\ell(G'))$ for all $\ell \in \mathcal{L}$ can be calculated in total time $\sum_{\ell \in \mathcal{L}'} O(c_\ell(G) + c_\ell(G')) = O(hkn)$. In summary, all steps for computing the kernel of Eq. 5.8 can be done in a total runtime of $O(hkm + hkn)$.

□

5.2.1 On the Expressive Power

We now show that tracking Weisfeiler-Lehman features over filtrations yields powerful methods in terms of expressiveness. Recall that the *expressive power* (see Sect. 2.9.2) of a method describes its ability to distinguish non-isomorphic graphs, i.e., method A is said to be "more expressive" or "more powerful" than method B if $A(G) = A(G') \Rightarrow B(G) = B(G')$ and there exist non-isomorphic graphs G, G' such that $A(G) \neq A(G')$ and $B(G) = B(G')$. Furthermore, recall that the (1-dimensional) Weisfeiler-Lehman test for isomorphism is known to be inexact even after n iterations (Cai, Fürer, and Immerman, 1992). That is, there exist pairs of non-isomorphic graphs which cannot be distinguished by the Weisfeiler-Lehman isomorphism test. However, we can show that considering Weisfeiler-Lehman labels over certain filtrations strictly increases the expressive power.

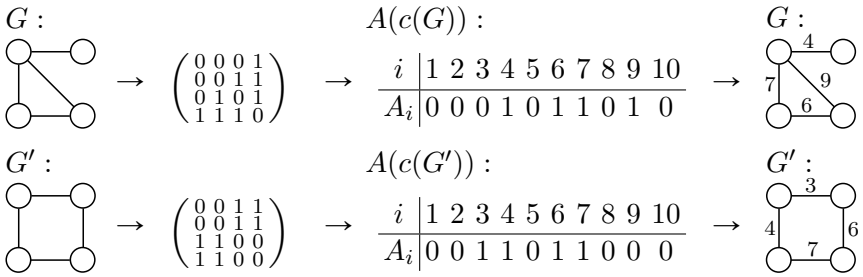


Figure 5.2: Graphs G and G' are assigned edge weights by a canonicalization process. Using these weights, the filtration graphs $G_9 = (V(G), \{e \in E(G) : w(e) \geqslant 9\})$ and $G'_9 = (V(G'), \{e \in E(G') : w(e) \geqslant 9\})$ can be distinguished using Weisfeiler-Lehman labels since G_9 has an edge and G'_9 does not. Thus, graphs G and G' cannot be isomorphic.

Theorem 5.2. *For any $h \geqslant 1$, there exists a filtration function $\mathcal{F}$ such that $\phi_\ell^{\mathcal{F}}(G) = \phi_\ell^{\mathcal{F}}(G')$ for all $\ell \in \mathcal{L}_h$ if and only if G and G' are isomorphic.*

Proof. The proof of this theorem is based on the fact that isomorphism between graphs and canonicalization of graphs are closely related concepts. A *canonical form* for a class of graphs $\mathcal{G}$ is a function $c : \mathcal{G} \to \mathcal{G}$ that maps each $G \in \mathcal{G}$ to a unique representative $c(G)$ of its isomorphism equivalence class. Correspondingly, a *canonical ordering* o_c of $G \in \mathcal{G}$ assigns the vertices of G the index of their images in $c(G)$. Hence, a check for isomorphism between G and G' can be done by simply comparing $c(G)$ and $c(G')$ for equality using the canonical vertex orderings.

Canonical vertex orderings can be obtained by defining canonical forms on adjacency matrices.[1] In the following, we obtain canonical vertex orderings by utilizing the lexicographic order over the flattened upper triangle of adjacency matrices. For graphs without vertex labels, the standard definition of adjacency matrices as binary matrices can be used. For simple labeled graphs, we can define an adjacency matrix in this context as the $n \times n$ matrix A that – for a given ordering $v_1, v_2, \ldots, v_n$ of vertices of G – contains the label of v_i at $A_{i,i}$ and a one at $A_{i,j}$ and $A_{j,i}$ if and only if $\{v_i, v_j\} \in E(G)$. We now show that canonical orderings can be used to define filtrations which allow to solve the graph isomorphism problem using depth-h Weisfeiler-Lehman labels alone, for any depth $h \geqslant 1$.

Let $\mathcal{G}$ be a class of graphs and let c be a function assigning graphs $G \in \mathcal{G}$ to their canonical form. For $G \in \mathcal{G}$, we define an edge weight function $\omega : E(G) \to \mathbb{N}$ using the canonical ordering o_c of G as follows: Let $A(c(G))$ be the flattened upper triangle of the adjacency matrix of $c(G)$. Then, for $e = \{u, v\} \in E(G)$, we set $\omega(e)$ to the index of edge $\{o_c(u), o_c(v)\}$ in $A(c(G))$. This weight function maps each edge to an integer weight greater than zero (see Fig. 5.2).

Now, if G and G' have different numbers of vertices, then the sets of Weisfeiler-Lehman labels of G and G' have different cardinalities and the above claim is trivial. Otherwise, for

[1] One folklore example of an algorithm that computes a canonical form is to fix some total order on the adjacency matrices of graphs in $\mathcal{G}$ and map each $G \in \mathcal{G}$ to the smallest permutation of its adjacency matrix with respect to that total order.

two graphs G, G' with n vertices and edge weights defined as above, we can use the natural numbers as cutoff values, resulting in filtrations $\mathcal{F}(G), \mathcal{F}(G')$ of length $\binom{n}{2} + n$. That is, the filtration $\mathcal{F}(G)$ contains a filtration graph G_i for each index $i \in [\binom{n}{2} + n]$ of the flattened upper triangular matrix $A(c(G))$. G_i contains only such edges with weights greater or equal i. If G and G' are isomorphic, then for each $i \in [\binom{n}{2} + n]$ we have that the filtration graphs G_i and G'_i are isomorphic. Hence, the sets of Weisfeiler-Lehman labels of G_i and G'_i will be identical for every Weisfeiler-Lehman iteration.

Now consider the case that G and G' are not isomorphic. Then it must be the case that their canonical forms are not equal, i.e., $c(G) \neq c(G')$. As a result, the adjacency matrices of $c(G)$ and $c(G')$ are not equal and there must be a *largest* index x where the entries of $A(c(G))$ and $A(c(G'))$ differ. Hence, G_x and G'_x are not isomorphic, as they have different numbers of edges. It is easy to see that the Weisfeiler-Lehman isomorphism test distinguishes two graphs with different numbers of edges, for any depth $h \geqslant 1$. This directly follows from the fact that in such a case, the two graphs have different sets of vertex degrees. As a result, the sets of depth-1 Weisfeiler-Lehman labels of G_x and G'_x differ, which concludes the proof. □

Note that the construction in the proof of Theorem 5.2 shows how to obtain a filtration function from a canonical form. While such a filtration is not known to be efficiently computable, it shows that Weisfeiler-Lehman labels on top of filtrations $\mathcal{F}(G), \mathcal{F}(G')$ are strictly more powerful than Weisfeiler-Lehman labels on the original graphs G, G'. In particular, it shows that filtrations and Weisfeiler-Lehman labels are "compatible" in the sense that the information provided by a graph filtration may be used beneficially by the Weisfeiler-Lehman algorithm. Together with several efficiently computable filtrations (such as the canonical form for bounded treewidth graphs (Wagner, 2011)), this supports the claim that our approach can be beneficial in practice.

As a first implication, the Weisfeiler-Lehman filtration kernel is – given a suitable filtration – strictly more expressive than the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) on the original graphs.

Corollary 5.1. *For any $h \geqslant 1$, and $n \geqslant 1$, there exists a filtration function $\mathcal{F}$ such that the kernel $k^{\mathcal{F}}_{\mathcal{L}_h}(G, G')$ on graphs $G, G' \in \mathcal{G}_n$ is* complete.

Proof. In the following, we will use the edge weight functions and filtrations from the proof of Thm. 5.2. In order to prove that the kernel

$$k^{\mathcal{F}}_{\mathcal{L}_h}(G, G') = \sum_{\ell \in \mathcal{L}_h} \kappa^{\mathcal{F}}_{\ell}(G, G') \ \left\|\phi^{\mathcal{F}}_{\ell}(G)\right\|_1 \left\|\phi^{\mathcal{F}}_{\ell}(G')\right\|_1 \tag{5.9}$$

is complete (Gärtner, Flach, and Wrobel, 2003), we need to prove that its corresponding embedding function $\varphi^{\mathcal{F}}_{\mathcal{L}_h}$ with $k^{\mathcal{F}}_{\mathcal{L}_h}(G, G') = \langle \varphi^{\mathcal{F}}_{\mathcal{L}_h}(G), \varphi^{\mathcal{F}}_{\mathcal{L}_h}(G') \rangle$ satisfies

$$\varphi^{\mathcal{F}}_{\mathcal{L}_h}(G) = \varphi^{\mathcal{F}}_{\mathcal{L}_h}(G') \Leftrightarrow G, G' \text{are isomorphic.} \tag{5.10}$$

To prove the claim in Eq. 5.10, we first consider the case that G and G' are isomorphic. Then, the filtration graphs in $\mathcal{F}(G)$ and $\mathcal{F}(G')$ must be pairwise isomorphic as well. More precisely, for a length-k filtration $\mathcal{F}$, it holds that $G_i \equiv G'_i$ for all $i \in [k]$ where G_i and G'_i are

filtrations graphs in $\mathcal{F}(G)$, resp. $\mathcal{F}(G')$. It immediately follows that the kernel embeddings of G and G' must be equivalent as well, i.e., $\varphi^{\mathcal{F}}_{\mathcal{L}_h}(G) = \varphi^{\mathcal{F}}_{\mathcal{L}_h}(G')$.

We prove the other direction in Eq. 5.10 by contradiction. Suppose that G and G' are not isomorphic. Then, by Theorem 5.2, there exists a Weisfeiler-Lehman label $\ell' \in \mathcal{L}_h$ such that $\phi^{\mathcal{F}}_{\ell'}(G) \neq \phi^{\mathcal{F}}_{\ell'}(G')$. Note that the embedding space of $k^{\mathcal{F}}_{\mathcal{L}_h}$ corresponds to the concatenation of the embedding spaces of the weighted base kernels

$$k^{\mathcal{F}}_{\ell}(G, G') = \kappa^{\mathcal{F}}_{\ell}(G, G') \ \left\|\phi^{\mathcal{F}}_{\ell}(G)\right\|_1 \left\|\phi^{\mathcal{F}}_{\ell}(G')\right\|_1 \tag{5.11}$$

for all $\ell \in \mathcal{L}_h$. This follows from the fact that $k^{\mathcal{F}}_{\mathcal{L}_h}$ is a sum of such kernels $k^{\mathcal{F}}_{\ell}, \ell \in \mathcal{L}_h$. Let $\varphi^{\mathcal{F}}_{\ell}$ be the embedding function corresponding to $k^{\mathcal{F}}_{\ell}$, and let $\widetilde{\varphi}^{\mathcal{F}}_{\ell}$ denote the embedding function corresponding to $\kappa^{\mathcal{F}}_{\ell}$. Then

$$\begin{aligned} k^{\mathcal{F}}_{\ell}(G, G') &= \kappa^{\mathcal{F}}_{\ell}(G, G') \ \left\|\phi^{\mathcal{F}}_{\ell}(G)\right\|_1 \left\|\phi^{\mathcal{F}}_{\ell}(G')\right\|_1 \\ &= \langle \widetilde{\varphi}^{\mathcal{F}}_{\ell}(G), \widetilde{\varphi}^{\mathcal{F}}_{\ell}(G') \rangle \ \left\|\phi^{\mathcal{F}}_{\ell}(G)\right\|_1 \left\|\phi^{\mathcal{F}}_{\ell}(G')\right\|_1 \\ &= \langle \widetilde{\varphi}^{\mathcal{F}}_{\ell}(G) \left\|\phi^{\mathcal{F}}_{\ell}(G)\right\|_1, \widetilde{\varphi}^{\mathcal{F}}_{\ell}(G') \left\|\phi^{\mathcal{F}}_{\ell}(G')\right\|_1 \rangle \ . \end{aligned}$$

Thus, for all $H \in \mathcal{G}$,

$$\varphi^{\mathcal{F}}_{\ell}(H) = \widetilde{\varphi}^{\mathcal{F}}_{\ell}(H) \left\|\phi^{\mathcal{F}}_{\ell}(H)\right\|_1 \ . \tag{5.12}$$

In order to prove that $\varphi^{\mathcal{F}}_{\mathcal{L}_h}(G) \neq \varphi^{\mathcal{F}}_{\mathcal{L}_h}(G')$, it suffices to show that $\varphi^{\mathcal{F}}_{\ell'}(G) \neq \varphi^{\mathcal{F}}_{\ell'}(G')$ for the distinguishing label ℓ'. Recall that for a graph H and label ℓ' it holds that $\phi^{\mathcal{F}}_{\ell'}(H) = \hat{\phi}^{\mathcal{F}}_{\ell'}(H) \left\|\phi^{\mathcal{F}}_{\ell'}(H)\right\|_1$. We distinguish two cases:

Case 1 $\left\|\phi^{\mathcal{F}}_{\ell'}(G)\right\|_1 = \left\|\phi^{\mathcal{F}}_{\ell'}(G')\right\|_1$:
In this case, it holds that the normalized filtration histograms must be unequal for the distinguishing label ℓ', i.e., $\hat{\phi}^{\mathcal{F}}_{\ell'}(G) \neq \hat{\phi}^{\mathcal{F}}_{\ell'}(G')$. It follows that $\mathcal{W}_{d^1}(\hat{\phi}^{\mathcal{F}}_{\ell'}(G), \hat{\phi}^{\mathcal{F}}_{\ell'}(G')) > 0$ while $\mathcal{W}_{d^1}(\hat{\phi}^{\mathcal{F}}_{\ell'}(G), \hat{\phi}^{\mathcal{F}}_{\ell'}(G)) = \mathcal{W}_{d^1}(\hat{\phi}^{\mathcal{F}}_{\ell'}(G'), \hat{\phi}^{\mathcal{F}}_{\ell'}(G')) = 0$, as $\mathcal{W}_{d^1}$ is a metric on normalized filtration histograms. Hence, the base kernel embeddings $\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(G)$ and $\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(G')$ cannot be identical, since $\kappa^{\mathcal{F}}_{\ell'}$ is a function and

$$\kappa^{\mathcal{F}}_{\ell'}(G, G) = \kappa^{\mathcal{F}}_{\ell'}(G', G') = 1 > \kappa^{\mathcal{F}}_{\ell'}(G, G') \ . \tag{5.13}$$

As a direct consequence of this and Eq. 5.12, we have $\varphi^{\mathcal{F}}_{\ell'}(G) \neq \varphi^{\mathcal{F}}_{\ell'}(G')$.

Case 2 $\left\|\phi^{\mathcal{F}}_{\ell'}(G)\right\|_1 \neq \left\|\phi^{\mathcal{F}}_{\ell'}(G')\right\|_1$:
First, note that the base kernel embedding $\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(H)$ has unit length for every graph $H \in \mathcal{G}$ due to the fact that $\mathcal{W}_{d^1}$ is a metric. That is:

$$\begin{aligned} \left\|\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(H)\right\|_2 &= \sqrt{\langle \widetilde{\varphi}^{\mathcal{F}}_{\ell'}(H), \widetilde{\varphi}^{\mathcal{F}}_{\ell'}(H) \rangle} \\ &= \sqrt{\kappa^{\mathcal{F}}_{\ell'}(H, H)} \\ &= \sqrt{e^{-\gamma \mathcal{W}_{d^1}(\hat{\phi}^{\mathcal{F}}_{\ell'}(H), \hat{\phi}^{\mathcal{F}}_{\ell'}(H))}} \\ &= \sqrt{e^{-\gamma 0}} = 1 \ . \end{aligned}$$

Thus, using Eq. 5.12, the following holds for label ℓ':

$$\begin{aligned}
\left\|\varphi^{\mathcal{F}}_{\ell'}(G)\right\|_2 &= \left\|\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(G)\left\|\phi^{\mathcal{F}}_{\ell'}(G)\right\|_1\right\|_2 \\
&= \left\|\widetilde{\varphi}^{\mathcal{F}}_{\ell'}(G)\right\|_2 \left\|\phi^{\mathcal{F}}_{\ell'}(G)\right\|_1 \\
&= \left\|\phi^{\mathcal{F}}_{\ell'}(G)\right\|_1 \\
&\neq \left\|\phi^{\mathcal{F}}_{\ell'}(G')\right\|_1 \\
&= \left\|\varphi^{\mathcal{F}}_{\ell'}(G')\right\|_2 \ .
\end{aligned}$$

Since the two vectors $\varphi^{\mathcal{F}}_{\ell'}(G)$ and $\varphi^{\mathcal{F}}_{\ell'}(G')$ have different lengths, they cannot be identical, which concludes our proof. □

While filtrations yielding complete kernels are not known to be efficiently computable, there are efficiently computable filtrations that result in strictly more expressive (but incomplete) Weisfeiler-Lehman filtration kernels when compared to the ordinary Weisfeiler-Lehman subtree kernel. As an example of such an efficiently computable filtration, consider the function that annotates edges by the number of triangles they belong to. Figure 5.3 shows two graphs which cannot be distinguished by the 1-dimensional Weisfeiler-Lehman test of isomorphism, but become distinguishable using this kind of filtration.[2] This concept can be extended to larger (or multiple) subgraphs, which allows for more expressive filtrations, similar to the approach of Barceló et al. (2021).

Orthogonal to the contribution of this chapter, there exists a large body of work that relates the expressive power of certain graph neural networks (GNN, see Sect. 3.5) to that of the Weisfeiler-Lehman isomorphism test (Xu et al., 2019; Morris et al., 2019). In this regard, Theorem 5.2 implies that a (neural) ensemble of graph neural networks over a suitably chosen filtration is strictly more powerful than a graph neural network on the original graph(s) alone:

Corollary 5.2. *There exist a filtration function $\mathcal{F}$ and GNN (Xu et al., 2019) $\mathcal{N}$ such that $\mathcal{N}$ can distinguish any two non-isomorphic graphs when provided with the filtration graphs corresponding to $\mathcal{F}$.*

Proof. This result directly follows from Theorem 3 in Xu et al. (2019). Let $\mathcal{N}'$ be a GNN with injective aggregator and update functions, as well as an injective graph level readout function as stated by Xu et al. (2019). For graph G, we first pass all its filtration graphs $\mathcal{F}(G)$ through $\mathcal{N}'$. We then construct $\mathcal{N}$ by adding another injective readout layer over the set of the resulting graph level readouts of the filtration graphs. Using Theorem 5.2, for two non-isomorphic graphs G, G', there exist filtration graphs G_x, G'_x that can be distinguished by their Weisfeiler-Lehman labels. Following Xu et al. (2019), $\mathcal{N}'$ maps G_x and G'_x to different embeddings. Hence, an injective readout layer over these embeddings must map G and G' to different embeddings. □

[2] It is obvious that using the proposed weights as edge labels allows the Weisfeiler-Lehman isomorphism test to distinguish these two edge labeled graphs. However, it can be shown that it suffices to consider the Weisfeiler-Lehman labels on the unlabeled filtration graphs.

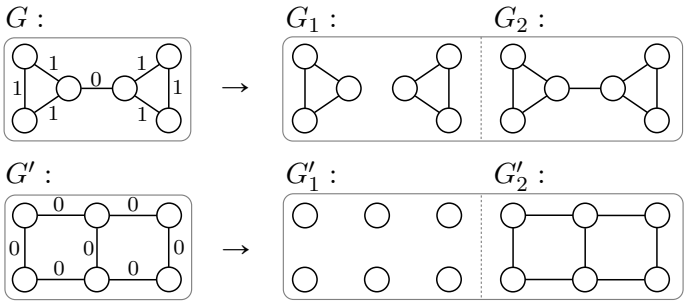


Figure 5.3: Consider the unlabeled graphs G, G' which cannot be distinguished using the Weisfeiler-Lehman isomorphism test. Adding edge weights corresponding to the number of triangles that each edge is part of yields filtration graphs G_1, G'_1, which can now be distinguished by the Weisfeiler-Lehman isomorphism test. Note that the same filtration is generated using the edge weight function ω_w^2 (see Sect. 5.4.1).

5.3 Filtration Kernels for Arbitrary Graph Features

Although the utilization of Weisfeiler-Lehman labels results in a particularly interesting filtration kernel, it is worth noting that filtration kernels are not exclusively restricted to this feature type. In fact, the concept can be applied to any kind of graph feature which yields finite graph representations. In this section, we briefly generalize the above approach and introduce the family of *graph filtration kernels*.

Instead of tracking and comparing the lifespans of Weisfeiler-Lehman labels, we may just as well consider any set $\mathcal{X}$ of graph features. In order to define suitable kernel functions, it merely needs to hold that for each graph $G \in \mathcal{G}$ and filtration $\mathcal{F}$, the number of features $X \subseteq \mathcal{X}$ contained in each filtration graph of $\mathcal{F}(G)$ is finite.

To properly define graph filtration kernels, we now quickly recap the definition of base kernels and prove their positive semi-definiteness. Recall that the filtration histogram distance $\mathcal{W}_{d^1}(\hat{\phi}_\ell^{\mathcal{F}}(G), \hat{\phi}_\ell^{\mathcal{F}}(G'))$ compares the (mass-normalized) feature ℓ distributions of graphs G and G'. A base kernel utilizes this distance function and "transforms" it into a similarity measure. More precisely:

Definition 5.4 (Base Kernel). *Given graphs $G, G' \in \mathcal{G}$, a filtration function $\mathcal{F}$, and a feature $\ell \in \mathcal{X}$, the* base kernel *is defined by*

$$\kappa_\ell^{\mathcal{F}}(G, G') = e^{-\gamma \mathcal{W}_{d^1}(\hat{\phi}_\ell^{\mathcal{F}}(G), \hat{\phi}_\ell^{\mathcal{F}}(G'))}. \tag{5.14}$$

where $\hat{\phi}_\ell^{\mathcal{F}}(G)$ is the mass-normalized filtration histogram of $\phi_\ell^{\mathcal{F}}(G)$.

Theorem 5.3. *The base kernel $\kappa_\ell^{\mathcal{F}}$ is positive semi-definite.*

Proof. The proof follows directly from results by Le et al. (2019) who show that that the Wasserstein distance $\mathcal{W}_{d^1}(\cdot, \cdot)$ using the 1-dimensional ground distance d^1 is *(conditionally) negative definite* (CND, see Def. 2.17). According to Schoenberg (1938), it holds that $k(x, y) = e^{-\gamma g(x,y)}$ is *positive semi-definite* (PSD) for all $\gamma \in \mathbb{R}_+$ if g is CND. This implies that the base kernel is PSD. □

In the following, we introduce two graph filtration kernel variants, which differ in the way they aggregate the base kernels. We show some of their properties and prove their positive semi-definiteness.

5.3.1 Linear Combination Kernel

Filtration kernels can be constructed in form of *linear combinations* of base kernels. More precisely, we define them as sums of kernels $\kappa_\ell^{\mathcal{F}}$ over features $\ell \in \mathcal{X}$.

Definition 5.5 (Filtration Kernel). *Given graphs $G, G' \in \mathcal{G}$, a filtration function $\mathcal{F}$, and a set of features $\mathcal{X}$, the* filtration kernel *is given by*

$$k_{\mathcal{X}}^{\mathcal{F}}(G, G') = \sum_{\ell \in \mathcal{X}} \kappa_\ell^{\mathcal{F}}(G, G') \ \left\|\phi_\ell^{\mathcal{F}}(G)\right\|_1 \left\|\phi_\ell^{\mathcal{F}}(G')\right\|_1 \tag{5.15}$$

where $\left\|\phi_\ell^{\mathcal{F}}(G)\right\|_1$ is the total mass of the filtration histogram $\phi_\ell^{\mathcal{F}}(G)$.

Notice the special case that a feature ℓ does not appear in the filtration sequence of G. Then, the corresponding histogram $\hat{\phi}_\ell(G)$ has zero mass and the Wasserstein distance to some non-zero mass histogram $\hat{\phi}_\ell(G')$ is not properly defined. Formally, this issue can simply be resolved by setting the value at the first histogram index of $\hat{\phi}_\ell(G)$ to 1 whenever ℓ does not appear in $\mathcal{F}(G)$. We note that the resulting Wasserstein distance $\mathcal{W}_{d^1}(\hat{\phi}_\ell(G), \hat{\phi}_\ell(G'))$ over such altered histograms is not necessarily meaningful. However, since ℓ does not appear in G, we have $\|\phi_\ell(G)\|_1 = 0$, and thus, the similarity $\kappa_\ell(G, G')$ is effectively disregarded. In fact, therefore, $\kappa_\ell(G, G')$ does not need to be computed to begin with. Thus, only such features in $\mathcal{X}$ contribute to the similarity $k_{\mathcal{X}}^{\mathcal{F}}(G, G')$ which appear in both $\mathcal{F}(G)$ *and* $\mathcal{F}(G')$. More formally:

Lemma 5.1. *For graphs G, G', a set of features $\mathcal{X}$ and a feature $\ell' \notin \mathcal{X}$, it holds that $k_{\mathcal{X}}^{\mathcal{F}}(G, G') = k_{\mathcal{X} \cup \ell'}^{\mathcal{F}}(G, G')$ if ℓ' does not appear in $\mathcal{F}(G)$ or $\mathcal{F}(G')$.*

We can now prove that the filtration kernel in Eq. 5.15 is a proper kernel function.

Theorem 5.4. *The filtration kernel $k_{\mathcal{X}}^{\mathcal{F}}$ is positive semi-definite.*

Proof. For any given feature $\ell \in \mathcal{X}$ and graphs G, G', the term $\|\phi_\ell(G)\|_1 \|\phi_\ell(G')\|_1$, which corresponds to the simple feature frequency product, is trivially PSD. Furthermore, by Theorem 5.3, $\kappa_\ell^{\mathcal{F}}(G, G')$ is PSD. Following, e.g., Schölkopf and Smola (2002), kernels are closed under (finite) addition and multiplication. Since the set of features that appear in both G and G' is finite, it follows that $k_{\mathcal{X}}^{\mathcal{F}}$ is well defined and PSD. □

The graph filtration kernel defines a general framework which generalizes a wide range of existing graph kernel methods. In the following, we outline a standard paradigm that has been used for graph kernel construction.

A majority of graph kernels essentially measure graph similarity by counting pairs of equivalent features. Examples include the Weisfeiler-Lehman subtree kernel (Shervashidze et

al., 2011) and the cyclic pattern kernel (Horváth, Gärtner, and Wrobel, 2004). Such kernels can be computed by the inner product of *explicit* feature vectors

$$\varphi(G) = [c(\ell_1(G)), c(\ell_2(G)), \ldots], \tag{5.16}$$

where $c(\ell_i(G))$ indicates the count of feature $\ell_i \in \mathcal{X}$ in G over some fixed feature domain $\mathcal{X}$. We refer to this kind of kernels as *histogram* kernels. Note that histogram kernels can equivalently be expressed as sums of feature frequency products, i.e.,

$$k_{\mathcal{X}}(G, G) = \sum_{\ell \in \mathcal{X}} c(\ell(G))\ c(\ell(G')). \tag{5.17}$$

Notice the resemblance of the histogram kernels to our graph filtration kernels. In fact, it can be shown that the histogram kernel is a special case of graph filtration kernels.

Proposition 5.1. *For filtrations of length $k = 1$, the filtration kernel $k^{\mathcal{F}}_{\mathcal{X}}$ reduces to the histogram kernel $k_{\mathcal{X}}$.*

Proof. If there exists only a single filtration graph (i.e., $k = 1$), then $\kappa_\ell(G, G') = 1$ for all features ℓ which appear in both G and G'. (Recall that features which do not appear in both graphs can be disregarded as they do not actively contribute to $k^{\mathcal{F}}_{\mathcal{X}}$.) We know that for $k = 1$ it holds that $c(\ell(G)) = \|\phi_\ell(G)\|_1$. Thus, the filtration kernel reduces to

$$k^{\mathcal{F}}_{\mathcal{X}}(G, G') = \sum_{\ell \in \mathcal{X}} \|\phi_\ell(G)\|_1 \left\|\phi_\ell(G')\right\|_1 = \sum_{\ell \in \mathcal{X}} c(\ell(G))c(\ell(G')) = k_{\mathcal{X}}(G, G').$$

□

As the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) is an example of a histogram kernel, it directly follows that our Weisfeiler-Lehman filtration kernel (see Sect. 5.2) is a direct generalization of it.

5.3.2 Product Kernel

Filtration kernels can alternatively be defined as *products* over base kernels. Analogously to the (linear combination based) kernel presented in Sect. 5.3.1, each base kernel is individually weighted to make up for the loss of information due to the mass-normalization of histograms. For the product variant, this weighting is realized by a radial basis function term measuring the similarity between original histogram masses.

Definition 5.6 (Filtration (Product) Kernel). *Given graphs G, G', a filtration function $\mathcal{F}$, a set of features $\mathcal{X}$, and a parameter $\beta \in \mathbb{R}_+$, the* filtration (product) kernel *is given by*

$$\mathring{k}^{\mathcal{F}}_{\mathcal{X}}(G, G') = \prod_{\ell \in \mathcal{X}} \kappa^{\mathcal{F}}_\ell(G, G')\ e^{-\beta\ (\|\phi^{\mathcal{F}}_\ell(G)\|_1 - \|\phi^{\mathcal{F}}_\ell(G')\|_1)^2} \tag{5.18}$$

where $\left\|\phi^{\mathcal{F}}_\ell(G)\right\|_1$ denotes the total mass of the filtration histogram $\phi^{\mathcal{F}}_\ell(G)$.

We note that, this kernel is introduced only for purposes of demonstrating the *universality* of the filtration kernel concept and should be regarded as a theoretical contribution that is independent of the kernel introduced in Sect. 5.3.1.

Theorem 5.5. *The kernel $\mathring{k}^{\mathcal{F}}_{\mathcal{X}}$ is positive semi-definite.*

Proof. The proof is to a large extend analog to that of Theorem 5.4. Theorem 5.3 shows that the base kernel $\kappa^{\mathcal{F}}_{\ell}$ is positive semi-definite (PSD). Furthermore, the RBF term $e^{-\beta\|x-y\|^2}$ with $x, y \in \mathbb{R}$ and $\beta > 0$ is known to be PSD. One difference, however, is that whereas the linear combination kernel $k^{\mathcal{F}}_{\mathcal{X}}(G, G')$ depends only on features $\ell \in \mathcal{X}$ which appear in *both* $\mathcal{F}(G)$ and $\mathcal{F}(G')$, the product kernel $\mathring{k}^{\mathcal{F}}_{\mathcal{X}}(G, G')$ is affected by features $\ell \in \mathcal{X}$ which appear in *at least one* of $\mathcal{F}(G), \mathcal{F}(G')$. Thus, the key difference between the two variants is the case where $\mathcal{F}(G)$ contains a feature ℓ which does not appear in $\mathcal{F}(G')$. Nonetheless, the number of features that appear in $\mathcal{F}(G)$ or $\mathcal{F}(G')$ remains finite. Furthermore, the kernel value is not changed by any feature ℓ' which is contained in neither of the filtrations since in this case $\kappa^{\mathcal{F}}_{\ell'}(G, G') = 1$ and $e^{-\beta\ (\|\phi_{\ell'}(G)\|_1 - \|\phi_{\ell'}(G')\|_1)^2} = 1$. It follows that $\mathring{k}^{\mathcal{F}}_{\mathcal{X}}(G, G')$ is PSD. □

Similarly to Sect. 5.3.1, the filtration product kernel has a particularly theoretic value as it generalizes the radial basis function kernel. More precisely:

Proposition 5.2. *For filtrations $\mathcal{F}$ of length $k = 1$, the kernel $\mathring{k}^{\mathcal{F}}_{\mathcal{X}}(G, G')$ reduces to the RBF kernel $e^{-\beta\|\varphi(G)-\varphi(G')\|^2}$, where $\varphi(G)$ and $\varphi(G')$ denote the feature vectors of G, resp. G' (see Eq. 5.16), and $\|\varphi(G) - \varphi(G')\|^2$ denotes the squared Euclidean distance.*

Proof. For $k = 1$, there exists only a single entry in the filtration histograms. Recall from Sect. 5.3.1 that we set the value at the first histogram index of $\hat{\phi}_{\ell}(G)$ to 1 whenever ℓ does not appear in $\mathcal{F}(G)$. As a direct consequence, all mass-normalized filtration histograms are trivially equivalent. It follows that $\kappa^{\mathcal{F}}_{\ell}(G, G') = 1$ for all $\ell \in \mathcal{X}$ and $G, G' \in \mathcal{G}$. Thus, the kernel reduces to the following:

$$\begin{aligned}
\mathring{k}^{\mathcal{F}}_{\mathcal{X}}(G, G') &= \prod_{\ell \in \mathcal{X}} e^{-\beta\ (\|\phi_{\ell}(G)\|_1 - \|\phi_{\ell}(G')\|_1)^2} \\
&= \prod_{\ell \in \mathcal{X}} e^{-\beta\ (c(\ell(G)) - c(\ell(G')))^2} \\
&= e^{-\beta \sum_{\ell \in \mathcal{X}} (c(\ell(G)) - c(\ell(G')))^2} \\
&= e^{-\beta\|\varphi(G)-\varphi(G')\|^2}.
\end{aligned}$$

□

5.4 Experimental Evaluation

In this section, we experimentally evaluate the predictive performance of our Weisfeiler-Lehman filtration kernel introduced above and compare it to several state-of-the-art graph kernels. In order to generate meaningful graph filtrations, we consider two functions that

provide edge weights for originally unweighted graphs. The results of this section can be summarized as follows:

(i) The Weisfeiler-Lehman filtration kernel *significantly outperforms* its competitor kernels on several real-world benchmark datasets.

(ii) We investigated the influence of the filtration length k on the predictive performance and found that it suffices to consider values as low as $k = 3$. This result highlights that the Weisfeiler-Lehman filtration kernel improves over the ordinary Weisfeiler-Lehman subtree kernel at the cost of very *little* computational overhead.

(iii) Using synthetic benchmark graph datasets, we practically demonstrate the *expressive power* of the Weisfeiler-Lehman filtration kernel, i.e., its capability to distinguish non-isomorphic graphs.

In all experiments, we provide the accuracies obtained by support vector machines (SVM) using a 10-fold stratified cross-validation. A grid search over sets of kernel specific parameters is used for optimal training. We perform 10 such cross-validations and report the mean and standard deviation.

For all applied methods, we used the implementation as provided by Siglidis et al. (2020) or that of the respective authors and ran grid searches using the following kernel specific parameters. For approaches employing the Weisfeiler-Lehman subtree features, we chose $h \in \{1, \ldots, 5\}$. In case of the graphlet sampling (GS) kernel, the parameters $\epsilon = 0.1$, $\delta = 0.1$ and $k \in \{3, 4, 5\}$ were applied.

5.4.1 Filtration Variants

A unique parameter of our kernels is the graph filtration. If such a filtration is not provided through expert knowledge, one can be generated using edge weights. For now, assume that we are provided with an edge weight function $\omega : E(G) \to \mathbb{R}_{\geqslant 0}$ on graphs $G \in \mathcal{G}$. Recall that a filtration is induced by the value sequence $\alpha_1 > \ldots > \alpha_k$ (c.f. Eq. 5.2). While there exist infinitely many such sequences, we generate the α_is for a given set of graphs $\mathcal{D}$ and parameter $k \in \mathbb{N}$ as follows.

1. Apply the edge weight function $\omega : E(G) \to \mathbb{R}_{\geqslant 0}$ to all edges of graphs $G \in \mathcal{D}$.
2. Extract and sort the set of all edge weights $\{\omega(e) \in E(G) : G \in \mathcal{D}\}$ in a descending order.
3. Partition the ordered sequence of values $w_1, ..., w_n$ into k consecutive subsequences.
4. Generate a length-k threshold sequence $\alpha_1, \ldots, \alpha_k$ by setting α_i as the minimum element within the i-th subsequence.

The threshold sequence $\alpha_1, \ldots, \alpha_k$ then induces a graph filtration $\mathcal{F}$ for graphs in $\mathcal{D}$. Accordingly, for a graph $G = (V, E)$, the i-th filtration graph G_i contains edges with edge weight at least α_i, i.e., $G_i = (V, \{e \in E : \ \omega(e) \geqslant \alpha_i\})$ (see Sect. 5.1.1).

Since graphs are generally not explicitly equipped with edge weights, we consider two exemplary edge weight functions ω_d and ω_w^λ that each assign weights to edges according to the edges' structural relevance w.r.t. a specific property.

ω_d The function ω_d assigns each edge $e = \{u, v\}$ a weight that is equal to the maximum degree of its incident vertices, i.e., $\omega_d(e) = max(\delta(u), \delta(v))$. Thus, edges with at least one incident vertex of high degree appear early in the graph filtrations.

ω_w^λ The function ω_w^λ considers the total number of walks of length at most λ between adjacent vertices. That is, for edge $e = \{u, v\}$, $\omega_w^\lambda(e)$ is the number of walks of length at most λ between u and v.

In the following, the Weisfeiler-Lehman filtration kernel applied on dataset graphs with edge weights calculated according to ω_d is referred to as FWL-D. Analogously, FWL-W denotes the kernel based on weights computed according to ω_w^λ. The filtration length value k is treated as a parameter of the kernel methods. We choose $k \in \{1, \ldots, 10\}$.

5.4.2 Real-World Benchmarks

Figure 5.4 compares the predictive performance of the Weisfeiler-Lehman filtration kernel to various state-of-the-art graph kernels on a range of real-world benchmark datasets. On datasets DHFR, MUTAG, NCI1, PTC-MR and IMDB-BINARY, there are only small discrepancies between our method and the best performing competitor kernels. In fact, with PTC-MR being an exception, the results of all kernels using Weisfeiler-Lehman subtree features are virtually indistinguishable. This changes for the EGO datasets. While on EGO-1, our FWL-D variant is second only to the shortest-path kernel (Borgwardt and Kriegel, 2005), it significantly outperforms all tested kernels on EGO-2, EGO-3 and EGO-4, amounting to a roughly 10% accuracy increase for the case of the latter dataset. It becomes apparent from the results of the FWL-W variant that our approach is particularly reliant on the provided edge weights, or equivalently, the filtration function. However, the results of FWL-D suggest that in case the data at hand is not equipped with meaningful edge weights, very simple edge weight functions may already suffice to significantly increase the predictive performance over state-of-the-art graph kernels.

A conceptually similar approach to the contribution proposed in this chapter was introduced in parallel research by O'Bray, Rieck, and Borgwardt (2021). We refer to Sect. 3.4 for a discussion on this graph classification method. Due to the resemblance between the two methods in obtaining filtrations, we did not include the results obtained by O'Bray, Rieck, and Borgwardt (2021) in Fig. 5.4, by noting that the accuracies are nearly identical to those obtained by our kernel. Nonetheless, their approach once more highlights the power of filtrations for graph classification purposes.

5.4.3 The Influence of the Filtration Length k

As a central aspect of our approach, the filtration clearly plays a critical role in practical applications. In order to investigate the influence of filtrations, we provide experiments for varying choices of k, i.e., the filtration length. Figure 5.5 shows results on the EGO datasets

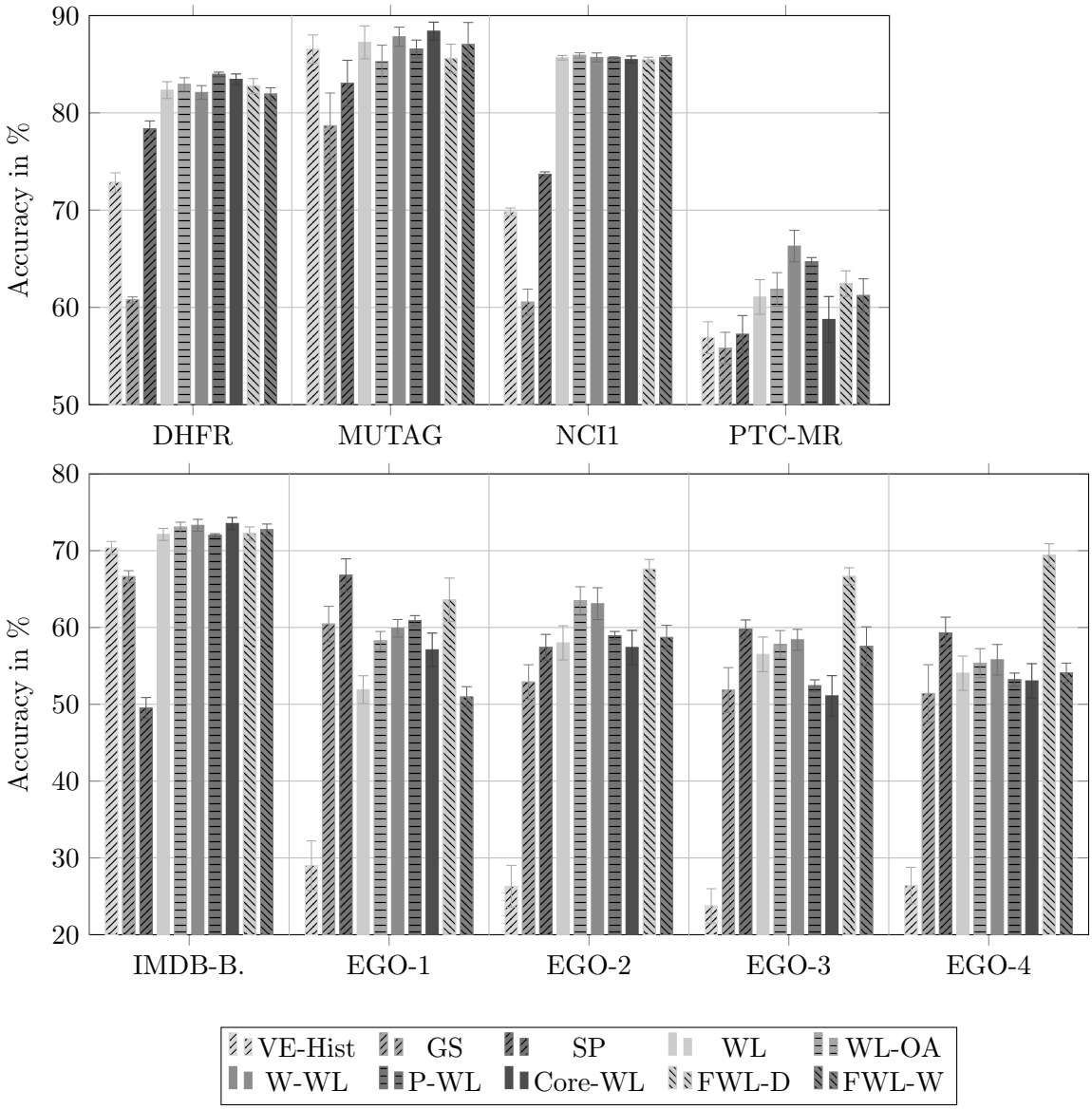


Figure 5.4: Classification accuracies and standard deviations on real-world benchmark datasets. We compare our approach to the Label Histogram (VE-Hist) kernel (see Sect. 3.4), the graphlet sampling (GS) kernel (Shervashidze et al., 2009), the shortest-path (SP) kernel (Borgwardt and Kriegel, 2005), the ordinary Weisfeiler-Lehman (WL) subtree kernel (Shervashidze et al., 2011), the Weisfeiler-Lehman optimal assignment (WL-OA) kernel (Kriege, Giscard, and Wilson, 2016), the Wasserstein Weisfeiler-Lehman (W-WL) kernel (Togninalli et al., 2019), the persistent Weisfeiler-Lehman (P-WL) method (Rieck, Bock, and Borgwardt, 2019), and the core variant of the Weisfeiler-Lehman (Core-WL) kernel (Nikolentzos et al., 2018). We employ our Weisfeiler-Lehman filtration kernel (FWL-D and FWL-W) using two different edge weight functions (see Sect. 5.4.1).

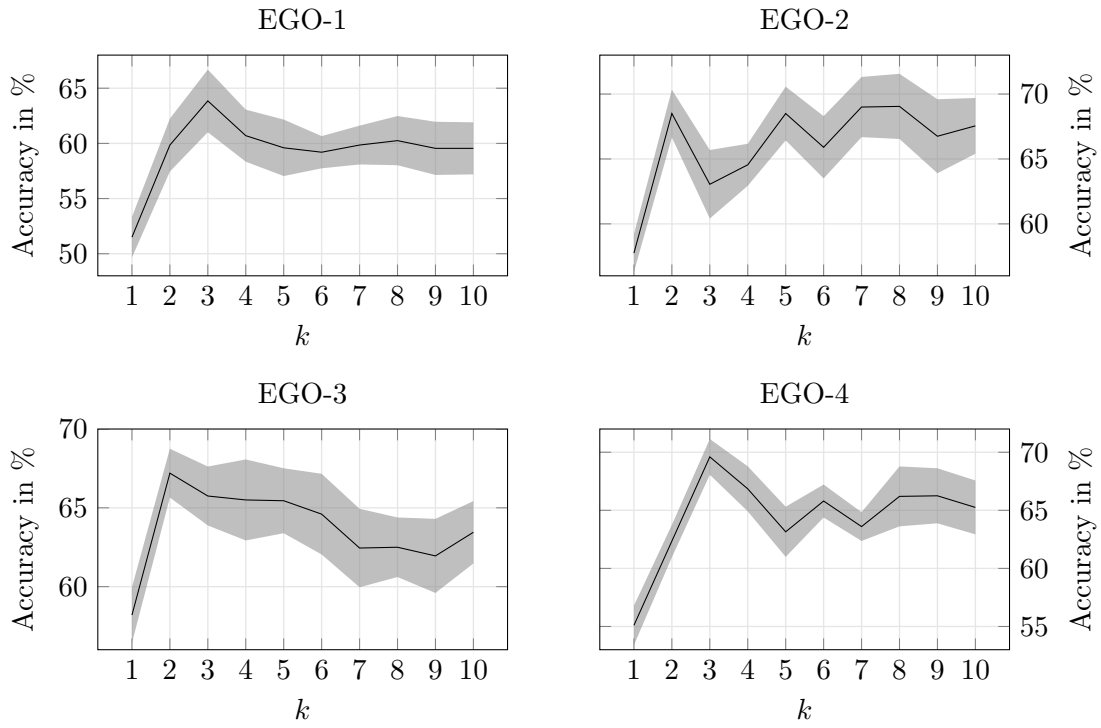
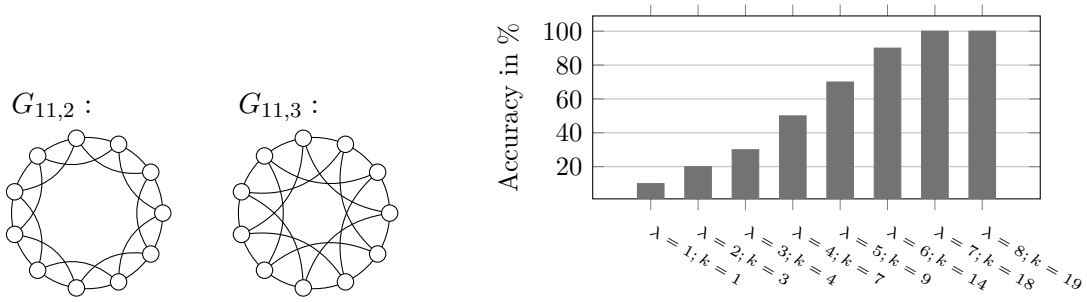


Figure 5.5: Classification accuracies and standard deviations of the FWL-D variant for different filtration lengths k.

for $k \in \{1, \ldots, 10\}$ using the FWL-D variant. We omit the FWL-W method in this analysis since the accuracies did not significantly differ for different choices for k. Recall that the case $k = 1$ is equivalent to the ordinary Weisfeiler-Lehman approach. Note that already for $k = 2$, the predictive performance significantly improves over $k = 1$ in all cases and that all datasets reach their accuracy peak at only $k = 2$ or $k = 3$. Since the kernel's complexity grows linearly with k (see Theorem 5.1), our approach improves upon the predictive performance of the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) at only a small additional computational cost.

5.4.4 Investigation of the Expressive Power

In this section, we investigate the *discriminative power* of graph filtration kernels. In particular, we consider a benchmark setup as discussed in Murphy et al. (2019), which classifies *circular skip link* (CSL) graphs. A CSL graph $G_{n,s}$ with $n > 4$ and $1 < s < \frac{n}{2}$ is a 4-regular graph where the n nodes form a cycle of length n and all pairs of nodes with path distance s on that cycle are furthermore connected by an edge. Examples for CSL graphs are depicted in Fig. 5.6(a). Following Murphy et al. (2019), for $n = 41$ and $s \in \{2, 3, 4, 5, 6, 9, 11, 12, 13, 16\}$, graphs are pairwise non-isomorphic. We construct a dataset containing 10 permuted copies of each graph. The classification task is then to assign graphs to their skip link value s. This setup is a commonly considered benchmark for measuring a model's capability to distin-



(a) Circular skip link graphs.

(b) Accuracies of FWL-W on CSL graphs.

Figure 5.6: Circular skip link graphs are regular graphs which the ordinary Weisfeiler-Lehman isomorphism test cannot distinguish. Two such exemplary graphs are shown in (a). When provided with filtrations induced by edge weights obtained by function ω_w^λ, the Weisfeiler-Lehman filtration kernel can distinguish more and more pairs of graphs with increasing values λ. The corresponding accuracies are provided in (b).

guish non-isomorphic graphs. It is particularly interesting since the 1-dimensional Weisfeiler-Lehman isomorphism test is not capable of distinguishing graphs in such a dataset.

In this experiment, we omit the FWL-D variant since the edge weights computed by ω_d did not result in increased expressive powers. Instead, we consider the FWL-W variant and do *not* limit the number of filtrations k, but allow as many filtrations as there are distinct edge weights in the dataset graphs. Thus, the number of filtration graphs is directly governed by λ, i.e., the parameter of ω_w^λ which corresponds to the maximal walk length. For example, when setting $\lambda = 4$, there exist a total of seven distinct edge weights within the CSL graph dataset. We, thus, generate $k = 7$ filtration graphs, one for each edge weight. Fig. 5.6(b) shows the predictive performances for different choices of λ (and thus k). The case $\lambda = 1$ implies $k = 1$ and therefore corresponds to the ordinary Weisfeiler-Lehman kernel. Since the Weisfeiler-Lehman method falls short of distinguishing regular graphs, the predictive performance corresponds to that of a random classifier. However, for increasing values of λ, the number of filtrations k grows as well, which results in the kernel's ability to distinguish more and more CSL graphs. Finally, for the case $\lambda = 7$, all non-isomorphic graphs can be distinguished. It is noteworthy, that these results are not entirely surprising as the considered edge weights provide the filtration kernel with increasing degrees of cyclic information. Nonetheless, the experiments highlight the power of filtration kernels and practically support Corollary 5.1.

5.4.5 Runtime Analysis

Fig. 5.7 provides the runtimes to compute the kernel matrix of our FWL-D variant. More specifically, we measured the runtime after the data has been loaded into memory. The datasets DHFR, MUTAG, NCI1, and PTC-MR contain only graphs with maximum node

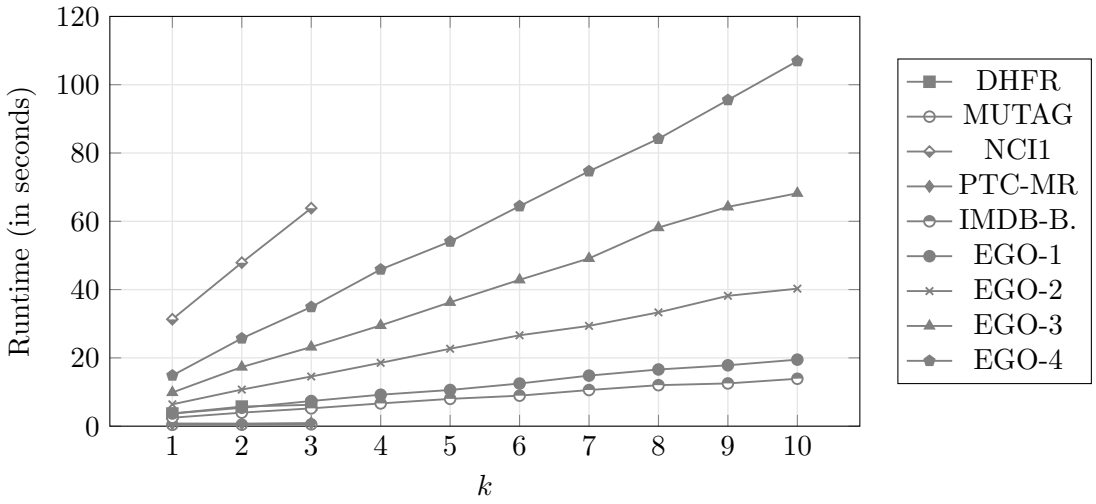


Figure 5.7: Runtime measures of the FWL-D variant for different filtration length values k.

degree 3, thus, limiting the range of the value k to 3 as well. We note that these results were obtained using an non-optimized version of our kernel and are hence not necessarily representative for its potential real-world performance. However, it is evident from the results that the runtime linearly increases with growing filtration length values k. This is consistent with our theoretical results on the runtime complexity of the kernel. To put the above numbers into context, Tab. 5.1 provides the runtime measures for all tested competitor kernels. The GS kernel was run using parameter $k = 5$. For all kernels based on the Weisfeiler-Lehman method, we picked depth parameter $h = 5$. All experiments were performed on an AMD 3900X processor (12 cores) with 64GB of memory.

5.5 Summary and Concluding Remarks

In this chapter, we introduced the Weisfeiler-Lehman filtration kernel, a novel kernel method that compares graphs at different *levels of resolution*. This was achieved utilizing the concept of graph filtrations, which define sequences of nested subgraphs. Tracking existence intervals of Weisfeiler-Lehman label occurences within such sequences allows for a comparison not only in terms of feature frequency, but also by *when* and for *how long* features appear in filtrations.

By using partial Weisfeiler-Lehman labels, i.e., labels which appear in filtration graphs only, our approach defines a *finer* similarity measure than the ordinary Weisfeiler-Lehman subtree kernel. Furthermore, the kernel allows for a meaningful *consideration of edge weights*, which is a widely disregarded aspect in the Weisfeiler-Lehman paradigm. Empirically, we have demonstrated that our proposed kernel significantly outperforms other state-of-the-art kernel methods on several real-world benchmark datasets.

We showed that the Weisfeiler-Lehman filtration kernel generalizes the ordinary Weisfeiler-Lehman subtree kernel, and for suitable choices of filtrations, it yields *strictly more powerful*

	GS	SP	WL	WL-OA	W-WL	P-WL	Core-WL
DHFR	77.17	4.34	0.41	12.39	290.32	4.27	1.28
MUTAG	20.07	0.20	0.05	0.35	9.61	0.41	0.52
NCI1	434.32	12.79	2.16	1624.24	6457.34	27.78	7.92
PTC-MR	38.64	0.30	0.08	1.65	32.49	0.68	0.21
IMDB-B.	135.89	2.39	0.51	17.53	297.06	12.55	6.65
EGO-1	21.62	15.96	0.50	4.16	103.46	18.48	5.08
EGO-2	23.38	33.27	0.92	6.14	173.11	55.65	17.18
EGO-3	24.44	60.01	1.47	7.93	260.58	119.40	40.26
EGO-4	26.58	97.18	2.16	9.85	353.57	212.05	78.98

Table 5.1: Runtime measures of competitor kernels (in seconds).

and even *complete* kernel functions. Finally, we proposed a general graph kernel framework which generalizes our approach to arbitrary types of graph features.

An interesting further research question is concerned with suitable choices for graph filtrations whenever they are not provided through expert knowledge or in form of edge weights. While we showed that it can already suffice to consider easily computable filtrations, we acknowledge that such filtrations may not exist for arbitrary graph datasets.

Finally, a particularly relevant research question is concerned with the utilization of our results to graph neural networks. Due to the close relationship of the Weisfeiler-Lehman test of isomorphism to the expressive capabilities of graph neural networks, the concept of graph filtrations can directly be used to increase the expressive power of GNNs.

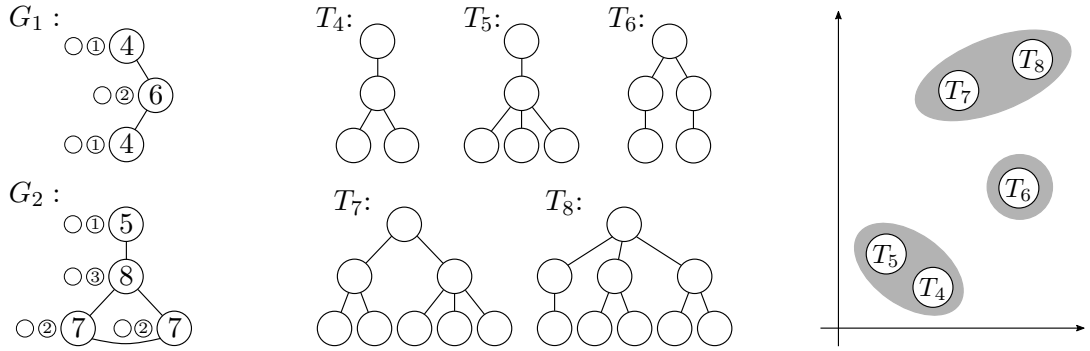
6

Generalized Weisfeiler-Lehman Kernels

In the previous chapter, we considered the popular Weisfeiler-Lehman method and introduced a graph kernel which compares graphs in terms of Weisfeiler-Lehman label distributions over sequences of subgraphs utilizing the concept of graph filtrations. Although we showed that this kernel is powerful in terms of expressivity and leads to remarkable predictive performances in practice for suitable choices of graph filtrations, its determining parameter (i.e., the graph filtration) needs to be explicitly provided. If a suitable filtration is not known in advance or cannot otherwise be induced, the Weisfeiler-Lehman filtration kernel effectively reduces to the ordinary Weisfeiler-Lehman subtree kernel (see Prop. 5.1) (Shervashidze et al., 2011).

In this chapter, we propose a different approach which addresses the drawback of the Weisfeiler-Lehman subtree kernel discussed in Chapter 5. Recall that this drawback is rooted in the "specificity" of the Weisfeiler-Lehman labels. More precisely, Weisfeiler-Lehman labels encode vertex neighborhoods which are often unique within a set of graphs. Consequently, since the Weisfeiler-Lehman subtree kernel defines similarity in terms of mutually occurring labels, this sparsity of labels may limit the effectiveness of the kernel. In other words, the kernel's primary drawback is that it is conceptually limited to comparing Weisfeiler-Lehman vertex labels w.r.t. *equality*. While this comparison is extremely well-suited for deciding graph isomorphism, which was the original problem considered by Weisfeiler and Lehman (1968), it is arguably too restrictive for defining *similarities*, in particular, graph kernels.

In this chapter, we *generalize* the Weisfeiler-Lehman subtree graph kernel by *relaxing* the strict comparison of labels. In particular, instead of distinguishing between Weisfeiler-Lehman labels by the binary valued equality relation, we propose a *natural* similarity measure to compare them on a much *finer* scale. To this end, recall that Weisfeiler-Lehman labels correspond to rooted trees, called unfolding trees (see Sect. 2.7.1). The embedding operator for this kind of pattern trees is implicitly induced by the Weisfeiler-Lehman method and corresponds to *locally bijective homomorphism*. The key idea in this chapter is to meaningfully compare Weisfeiler-Lehman labels by proposing a specifically designed distance measure between the respective unfolding trees, which provides a *semantically* adequate comparison for this kind of trees. For unfolding trees $T = T(G, v)$ and $T' = T(G', v')$, this distance between T and T' reflects how "close" the unfolding tree T is to being embeddable into G'



(a) Initially unlabeled graphs with their first two Weisfeiler-Lehman labels.

(b) Depth-2 unfolding trees corresponding to Weisfeiler-Lehman labels appearing in graphs G_1,G_2.

(c) Conceptual depiction of unfolding tree clustering.

Figure 6.1: (a) contains initially unlabeled graphs which are annotated by their first two Weisfeiler-Lehman labels, indicated by integers. These Weisfeiler-Lehman labels directly correspond to unfolding trees. In (b), we depict depth-2 unfolding trees, where label i corresponds to tree T_i. Consider unfolding trees T_4, T_5 and T_8. The tree T_4 visibly differs from T_5 by only a single vertex while it differs from T_8 by significantly more. To quantify this similarity difference between unfolding trees, we utilize the concept of tree edit distances and cluster unfolding trees w.r.t. this distance measure.

at v' by a locally bijective homomorphism. To achieve this, we propose a *modified* variant of the *tree edit distance*, which respects the essential properties of unfolding trees w.r.t. the pattern embedding operator defined by locally bijective homomorphism. Next to its semantic meaningfulness, we show that in contrast to more general tree edit distances, this distance can in fact be *efficiently* calculated.

An example motivating the approach of this chapter is depicted in Fig. 6.1(b). While unfolding tree T_4 visibly resembles T_5 much more than T_8, the ordinary Weisfeiler-Lehman subtree kernel simply treats them all as unequal and is thus unable to *quantify* the apparent difference among the pairwise similarities between these three unfolding trees. Specifically, the unfolding tree T_4 requires only a single node addition such that it becomes equivalent (i.e., isomorphic) to T_5, while it needs five node additions to be equivalent to T_8.

Using the above distance measure on unfolding trees, we are able to identify groups of *similar* Weisfeiler-Lehman labels by *clustering* (visualized in Fig. 6.1(c)). The elements within a cluster are then treated as *equal* labels. That is, we generalize the ordinary Weisfeiler-Lehman subtree kernel by regarding two unfolding trees as equivalent if they belong to the same cluster, i.e., have a *small* distance to each other. In this way, the ordinary Weisfeiler-Lehman subtree kernel becomes the *special* case, in which labels are considered equivalent if and only if they have distance zero. For partitioning the Weisfeiler-Lehman labels, we use *Wasserstein k-means* clustering (Irpino, Verde, and Carvalho, 2014). This choice is motivated by our result that the proposed adaptation of tree edit distances between unfolding trees can in fact be reformulated in terms of the Wasserstein distance.

We have empirically evaluated the predictive performance of our generalization of the Weisfeiler-Lehman subtree kernel on various real-world and synthetic datasets. The experimental results clearly show that while our more general approach does not result in an improvement on small molecular graphs, which are *sparse* and *structurally simple*, it considerably outperforms state-of-the-art graph kernels, and most importantly the *ordinary* Weisfeiler-Lehman subtree kernel, on datasets containing *dense* and *structurally diverse* graphs.

6.1 The Weisfeiler-Lehman Tree Edit Distance

In this section, we define a semantically meaningful distance on Weisfeiler-Lehman labels, give an algorithm computing this distance, and prove that it runs in polynomial time.

6.1.1 The Structure and Depth Preserving Tree Edit Distance

While the *strict* comparison of labels, or equivalently, that of unfolding trees, is advantageous for the original intention of the Weisfeiler-Lehman method, it poses a severe drawback for the Weisfeiler-Lehman subtree kernel. The reason is that comparing unfolding trees with each other by *equality* (i.e., isomorphism), is arguably too restrictive for kernel design, as in case of kernels, we are interested in defining similarities. Our typical observation is that the depth-k unfolding trees (or, equivalently, Weisfeiler-Lehman labels at iteration k) of most vertices are unique for very small values of k. In other words, the limitation of the Weisfeiler-Lehman subtree kernels is that two structurally completely different unfolding trees are treated identically to two unfolding trees which differ by only very little.

To overcome this drawback, we propose a finer label comparison by defining a new *(dis-) similarity measure* between unfolding trees that employs a *specialized* form of the well-known *tree edit distance*. On an abstract level, the tree edit distance measures the minimum amount of edit operations necessary to turn one tree into another. Calculating this distance is NP-hard in general (see, e.g., Bille, 2005). However, for our purpose it suffices to consider a restricted type of tree edit distance. We argue that this particular distance preserves essential properties of unfolding trees. Furthermore we show that, in contrast to the general case, this variant can be calculated efficiently.

Recall from Sect. 2.7.1 that depth-k unfolding trees are rooted trees that reflect the k-hop neighborhoods of vertices. For a graph G and vertex $v \in V(G)$, the depth-k unfolding tree $T_k(G, v)$ is implicitly constructed by the Weisfeiler-Lehman relabeling algorithm for node v. This unfolding tree $T = T_k(G, v)$ with root r is embedded into G by a *locally bijective homomorphism*. More precisely, there exists a homomorphism $\varphi : V(T) \to V(G)$ that maps r onto v, and for all non-leaves in $t \in V(T)$ the homomorphism φ induces a bijection between the children of t in T and the neighbors of $\varphi(t)$ in G (Dell, Grohe, and Rattan, 2018).

Considering the nature of unfolding trees, we propose a specifically designed distance measure on such trees. Intuitively speaking, the distance between depth-k unfolding trees $T = T_k(G, v)$ and $T' = T_k(G', v')$ is measured in terms of the costs that are necessary for altering T such that it can be embedded into G' at v' by a locally bijective homomorphism. We achieve this by defining a tree edit distance variant which upholds essential properties

of these trees' semantics and their embedding operator. Recall that tree edit distances can be expressed through the concept of *node maps* (see Sect. 2.6). A node map from T into T' essentially defines a partial mapping $f : V(T) \to V(T')$. To match some of the characteristics of locally bijective homomorphisms, we require this partial mapping f to be subject to the following two constraints:

1. f maps the root r of T onto the root r' of T'.
2. If f maps a node $t \in V(T)$ onto $t' \in V(T')$, then f can map the children of t only onto children of t'.

These requirements on comparing unfolding trees lead to the following definition of constrained node maps.

Definition 6.1 (Structure and Depth Preserving Mapping). *A* structure and depth preserving mapping (SDM) *between two rooted trees T and T' is a triple (π, T, T') with $\pi \subseteq V(T) \times V(T')$ satisfying*

(i) $\forall (v_1, v_1'), (v_2, v_2') \in \pi : v_1 = v_2 \iff v_1' = v_2'$, *(definite)*

(ii) $(r(T), r(T')) \in \pi$, *and* *(root preserving)*

(iii) $\forall (v, v') \in \pi : (par(v), par(v')) \in \pi$ *(structure preserving)*

where $par(v)$ with $v \in V(T)$ denotes the parent of node v in T. The set of all structure and depth preserving mappings between T and T' is denoted by $\mathrm{SDM}(T, T')$.

SDMs represent sequences of edit operations subject to the above constraints that transform trees into trees. More precisely, for an SDM (π, T, T') let $T = T_0, T_1, \ldots, T_k$ be a sequence of trees such that T_{i+1} is obtained from T_i by applying one of the following atomic transformations:

relabel: If $(v, v') \in \pi$, then replace the label of v in T_i by that of v'.

delete: If v is a leaf in T_i and it does not occur in a pair of π, then remove v from T_i.

insert: If v' is a vertex in T' which does not occur in a pair of π and for which the corresponding parent u already exists in T_i, then add a child to u with the label of v'.

The proof of the following claim is straightforward.

Proposition 6.1. *Let (π, T, T') be an* SDM *and $T = T_0, T_1, \ldots, T_k$ be a sequence of trees obtained by the above atomic transformations such that every $v \in T$ and $v' \in T'$ has been considered in exactly one transformation. Then $T_k = T'$.*

We stress that SDMs uphold essential properties of unfolding trees. In particular, they ensure that sibling relationships are preserved. That is, for any SDM (π, T, T'), nodes v_1' and v_2' are siblings in T' whenever $(v_1, v_1'), (v_2, v_2') \in \pi$ and v_1, v_2 are siblings in T. Furthermore,

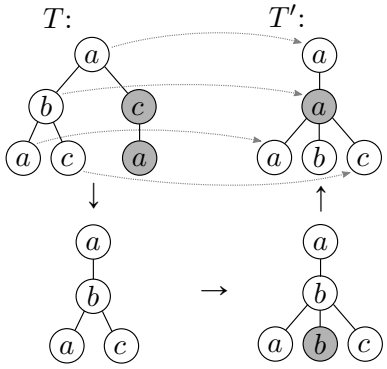


Figure 6.2: Visualization of a structure and depth preserving mapping (π, T, T') that transforms T into T'. Gray lines correspond to pairs contained in the mapping π. Furthermore, the figure provides a corresponding edit sequence where deleted, inserted, resp. relabeled nodes are marked in gray.

vertices can only be mapped onto vertices of the same depth. Recall, that our goal is to measure similarities between neighborhoods of vertices. It is therefore essential that roots are being preserved. This is guaranteed by the second constraint in Def. 6.1. Furthermore, from Def. 6.1 it follows that π maps a connected subtree of T onto a connected subtree of T'. That is, the first (resp. second) components of the pairs in π form a connected subtree of T (resp. T').

Figure 6.2 demonstrates the motivation of SDMs. The displayed mapping is a structure and depth preserving mapping from T into T' which visibly preserves the depth as well as the pairwise sibling relationships for all mapped vertices. Notice that nodes in T which are part of the mapping are relabeled. Except for one node, these relabelings are trivial, i.e., node labels are not altered. Furthermore note that, the two nodes in T which are not part of the mapping are deleted, while the node in T', which is not mapped upon, is being inserted.

Using the notions above, we are ready to define the distance between unfolding trees.

Definition 6.2 (Structure and Depth Preserving Tree Edit Distance). *Let T, T' be unfolding trees over a vertex label alphabet Σ and $\gamma : \Sigma_\perp \times \Sigma_\perp \to \mathbb{R}$ a cost function (i.e., metric), where $\Sigma_\perp = \{\Sigma \cup \perp\}$ with $\perp$ being the special blank symbol. Then the* cost *for an* SDM *(π, T, T') is given by*

$$\gamma(\pi) = \sum_{(v,v')\in\pi} \gamma(\ell(v), \ell(v')) + \sum_{v\in N} \gamma(\ell(v), \perp) + \sum_{v'\in N'} \gamma(\perp, \ell(v')) \tag{6.1}$$

where N (resp. N') are the vertices of T (resp. T') that do not occur in any pair of π. The structure and depth preserving tree edit distance *from T into T', denoted* SDTED(T, T')*, is then defined by*

$$\text{SDTED}(T, T') = \min\{\gamma(\pi) : (\pi, T, T') \in \text{SDM}(T, T')\}. \tag{6.2}$$

In other words, the cost of an SDM (π, T, T') is defined by the sum of the individual costs of relabeling, insertion, and deletion operations over all vertices of T and T', where the cost

for insertion (resp. deletion) of a vertex v is given by $\gamma(\ell(v), \bot)$ (resp. $\gamma(\bot, \ell(v))$). Finally, the structure and depth preserving tree edit distance between trees T and T' is simply the minimal cost over all possible mappings.

6.1.2 Computing Distances between Unfolding Trees

We now show that for any pair of unfolding trees T, T' of the same depth, the distance $\textsc{SdTed}(T, T')$ can be efficiently calculated in a recursive manner. The algorithm makes use of the fact that for a depth-k unfolding tree T in graph G, the subtrees below the root of T are $(k-1)$-unfolding trees of G. More precisely, let v be a child of the root $r(T)$. Then, the tree $T[v]$ (i.e., the subtree of T induced by v and its descendants) with root v is a $(k-1)$-unfolding tree appearing in G. We denote the set of all such trees below the root of T by $F(r(T))$.

Recall that an SDM between T and T' requires $r(T)$ to be mapped onto $r(T')$. Furthermore, from the properties of Definition 6.1 it follows that subtrees of T are mapped onto subtrees of T'. Thus, computing an optimal SDM from T into T' requires finding the set of optimal SDMs turning the trees below the root of T into the trees below the root of T'. In other words, we require the distances between $(k-1)$-unfolding trees in order to compute the distances between k-unfolding trees. These distances can be computed in a recursive manner by noting that 0-unfolding trees correspond to single vertices for which the costs for relabeling, insertion and deletion is given by function γ.

Alg. 3 implements this idea of computing the $\textsc{SdTed}(T, T')$ between unfolding trees T and T'. In order to allow for insertions and deletions, we pad the sets $F = F(r(T))$ and $F' = F(r(T'))$ by a sufficient number of empty trees (line 2). To find the set of optimal SDMs between F and F', we require the pairwise distances $\textsc{SdTed}(T_i, T'_j)$, with $T_i \in F$ and $T'_j \in F'$, as well as the costs for deleting and inserting trees T_i, resp. T'_j. The computation of these costs is done in line 3 of Alg. 3. The first case recursively calculates the $\textsc{SdTed}(T_i, T'_j)$ for all pairs of non-empty trees in F and F'. The second case considers the instance where the root of some subtree T_i is not part of a mapping, which implies that all vertices in T_i are deleted. A similar argument follows for the insertion of trees T'_j (third case of line 3). Then, finding an optimal SDM between T and T' can be reduced to the *minimum cost perfect bipartite matching* problem between elements in F and F' with distances as defined above (line 4). Finally, the $\textsc{SdTed}(T, T')$ is the cumulative cost of the distance between the roots of T and T' and the minimal cost perfect bipartite matching between the trees below these roots (line 5). The above considerations imply the following result:

Theorem 6.1. *Given unfolding trees T, T' of the same depth with labels from Σ and a cost function $\gamma : \Sigma_\bot \times \Sigma_\bot \to \mathbb{R}$, Alg. 3 returns $\textsc{SdTed}(T, T')$.*

As an example, consider the SDTED between unfolding trees T and T' of Fig. 6.3(a). We assume that each insertion, deletion, and relabeling operation has cost 1. Following Def. 6.1, the root of T is mapped onto the root of T'. As both vertices have the same label, the respective cost is zero (i.e. $\gamma(\ell(v_1), \ell(v'_1)) = 0$). Due to the structure preserving property of SDMs, calculating the edit costs for the remaining vertices beneath the roots comes down to matching (resp. inserting and deleting) the highlighted subtrees. It can easily be checked

Algorithm 3 COMPUTE SDTED

input: Trees T, T', cost function $\gamma : \Sigma_\perp \times \Sigma_\perp \to \mathbb{R}$
output: Structure and depth preserving tree edit distance between T and T'

SDTED(T, T'):

1: $F := F(r(T))$, $F' := F(r(T'))$
2: Pad F and F' with empty trees $T_\perp$ such that $|F| = |F'| = \delta(r(T)) + \delta(r(T'))$
3: **for all** $T_i \in F$, $T'_j \in F'$ **do**

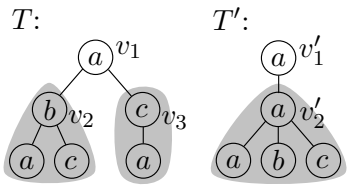
$$\delta_{ij} = \begin{cases} \text{SDTED}(T_i, T'_j) & \text{if } T_i \in F(r(T)) \text{ and } T'_j \in F(r(T')) \\ \sum_{v \in V(T_i)} \gamma(\ell(v), \perp) & \text{if } T_i \in F(r(T)) \text{ and } T'_j = T_\perp \\ \sum_{v' \in V(T'_j)} \gamma(\perp, \ell(v')) & \text{if } T_i = T_\perp \text{ and } T'_j \in F(r(T')) \\ 0 & \text{o/w .} \end{cases}$$

4: Let $S \subseteq F \times F'$ be a minimum cost perfect bipartite matching w.r.t. distances δ
5: **return** $\gamma(\ell(r(T)), \ell(r(T'))) + \sum_{(T_i, T'_j) \in S} \delta_{ij}$

that matching $T[v_2]$ with $T[v'_2]$ (which has cost 2) and thus deleting $T[v_3]$ (which has cost 2) has minimal cost over all possible matchings. The individual edit operations corresponding to this case are depicted in Fig. 6.2.

By the construction of unfolding trees, vertices closer to v in G begin to appear at smaller depths in $T_k(G, v)$. In fact, the number of occurrences in $T_k(G, v)$ of a node $u \in V(G)$ grows exponentially with k once it has appeared for the first time. This indirectly assigns higher weights to vertices closer to v in the calculation of the structure and depth preserving tree edit distance.

Notice that Algorithm 3 describes a naive implementation which in general requires an exponential number of recursion calls. However, this can be avoided using a lookup table which stores the SDTEDs between previously considered pairs of unfolding trees. Let $T = T_k(G, v)$ and $T' = T_k(G', v')$ be two depth-k unfolding trees. Clearly, for any $i \in \mathbb{N}$, the number of i-unfolding trees in G and G' is bounded by their sizes $n = |V(G)|$, resp. $n' = |V(G')|$. Thus, to compute SDTED(T, T'), we need to invoke Alg. 3 at most nn' times for every depth $i \in [k]$. Once a SDTED between two i-unfolding trees has been calculated, it can be stored in a lookup table. With such a lookup table, we require at most $nn'k$ invocations of the minimum cost perfect bipartite matching algorithm, each of complexity $O(d^3 log(d))$, where d is the maximum degree in G, G'. In summary, computing the SDTED between T and T' can be done in polynomial time.



(a) Unfolding trees with child trees below roots marked gray.

M_r	a	b	c	$\perp$
a	0	1	1	1
b	1	0	1	1
c	1	1	0	1
$\perp$	1	1	1	0

(b) Distance matrix on elements in $\Sigma_\perp$.

M_c	$T[v_2]$	$T[v_3]$	$T[v_2']$	$\perp$
$T[v_2]$	0	2	2	3
$T[v_3]$	2	0	3	2
$T[v_2']$	2	3	0	4
$\perp$	3	2	4	0

(c) Matrix containing SDTEDs between child trees.

Figure 6.3: The structure and depth preserving distance between T and T' can be formulated in terms of Wasserstein distances. Following the order on $\Sigma_\perp$ as in M_r of Fig. (b), resp. the order on child trees as in M_c of Fig. (c), the unfolding tree vectors of T and T' have the form $\mathbb{V}_r(T) = [1,0,0,0]$, $\mathbb{V}_c(T) = [1,1,0,4]$ and $\mathbb{V}_r(T') = [1,0,0,0]$, $\mathbb{V}_c(T') = [0,0,1,5]$. One can check that $\mathcal{W}_{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) = 0$ and $\mathcal{W}_{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')) = 4$, resulting in $\text{SDTED}(T, T') = \mathcal{W}_{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) + \mathcal{W}_{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')) = 4$.

6.2 The Generalized Weisfeiler-Lehman Kernel

Using the definitions and results of Sect. 6.1, we now introduce the *generalized Weisfeiler-Lehman subtree kernel* and show that it is in fact a generalization of the original Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011). Its key idea is to *relax* the rigid comparison of unfolding trees by equality (i.e., isomorphism) used in the original Weisfeiler-Lehman subtree kernel by considering the structure and depth preserving distances between unfolding trees. Using SDTED, we identify groups of similar trees by means of *hard* clustering. This ensures that similar unfolding trees are assigned to the same clusters, while dissimilar to different ones. Two unfolding trees are then regarded equivalent by the generalized Weisfeiler-Lehman subtree kernel iff they belong to the same cluster.

More precisely, for a set $\mathcal{D}$ of graphs, let Θ_i be a set of hard clusterings (i.e., partitionings) of the set of depth-i unfolding trees $\mathcal{T}^{(i)}$ appearing in the graphs in $\mathcal{D}$. We regard each element of Θ_i as a function $\rho : \mathcal{T}^{(i)} \to [k]$, where k is the number of clusters defined by ρ. Then, for any graphs $G, G' \in \mathcal{D}$ and depth parameter h, the *generalized Weisfeiler-Lehman subtree kernel* is defined by

$$k^h_{\text{GWL}}(G, G') = \sum_{i=0}^{h} \sum_{\rho \in \Theta_i} \sum_{v \in V(G)} \sum_{v' \in V(G')} \delta(\rho(T_i(G, v)), \rho(T_i(G', v'))), \tag{6.3}$$

where δ is the Dirac delta.

Theorem 6.2. *The generalized Weisfeiler-Lehman subtree kernel k^h_{GWL} is positive semi-definite.*

Proof. Since δ is trivially positive semi-definite and kernels are closed under addition (see Sect. 2.8), k^h_{GWL} is positive semi-definite and thus a proper kernel. □

Notice that k^h_{GWL} is equivalent to the original Weisfeiler-Lehman subtree kernel for the case that $\Theta_i = \{\rho_i\}$ with ρ_i defined as follows: For all $T, T' \in \mathcal{T}^{(i)}$, $\rho_i(T) = \rho_i(T')$ iff T and

T' are isomorphic (or equivalently, $\textsc{SdTed}(T, T') = 0$). Thus, our definition generalizes the ordinary Weisfeiler-Lehman subtree kernel in two ways: First, while the ordinary Weisfeiler-Lehman subtree kernel regards two unfolding trees T, T' to be equivalent iff $\textsc{SdTed}(T, T') = 0$, our definition allows $\textsc{SdTed}(T, T') \geqslant 0$ as well. Second, our definition enables more than one partitioning (or hard clustering) function.

We employ the concept of *Wasserstein k-means clustering* (Irpino, Verde, and Carvalho, 2014) as a method to partition the set of unfolding trees. This choice is motivated by several arguments. As mentioned above, the purpose of clustering is to group similar unfolding trees w.r.t. SdTed. We therefore require the clusters to be *convex* such that unfolding trees of a cluster ideally have pairwise small distance. Another requirement is to be able to control the number of clusters in order to govern the degree of relaxation. In the following, we show that the SdTed can in fact be calculated using the discrete Wasserstein distance (see Sect. 2.10). The Wasserstein distance has been the focus of comprehensive research leading to *fast* approximation methods for distance and center computations (Cuturi, 2013).

Below we address the most important ingredients of Wasserstein k-means needed for our purpose. In particular, we first discuss how unfolding trees can be represented by real-valued vectors. Subsequently, we state that the Wasserstein distance between such vectors corresponds to the SdTed of the respective unfolding trees. This representation furthermore allows for the calculation of center points using *Wasserstein barycenters*, enabling the application of Wasserstein k-means clustering approaches.

6.2.1 Unfolding Tree Vectors

In order to effectively apply Wasserstein k-means, we represent i-unfolding trees by (sparse) real-valued vectors. Recall that the structure and depth preserving tree edit distance is calculated as the *sum* of

(i) the distance between the root nodes, and

(ii) the minimum cost of the perfect bipartite matching between child trees below these roots,

as described in Alg. 3. We accordingly represent an i-unfolding tree T as a pair of vectors $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$, where

(i) $\mathbb{V}_r(T)$ encodes the root node's label $\ell(r(T))$, and

(ii) $\mathbb{V}_c(T)$ encodes the set of $(i-1)$-unfolding child trees $F(r(T))$ below the root $r(T)$.

We define $\mathbb{V}_r(T)$ as a *one-hot* vector with entry 1 at the index corresponding to the root node's label $\ell(r(T))$ and 0 everywhere else. The vector $\mathbb{V}_c(T)$ corresponds to a histogram that counts isomorphic $(i-1)$-unfolding child trees below the root. Both vectors $\mathbb{V}_r(T)$ and $\mathbb{V}_c(T)$ furthermore contain an entry which allows for insertions and deletions of nodes and subtrees.

More precisely, let $\Sigma_\perp = (l_1, \ldots, l_p, \perp)$ be the ordered set of all original vertex labels appearing in the graph dataset $\mathcal{D}$ and blank symbol $\perp$. Then, the root node label of an

unfolding tree T is represented by the vector $\mathbb{V}_r(T) = (x_1, \ldots, x_p, x_\perp)$, where

$$x_j = \begin{cases} 1 & \text{if } j \in [p] \text{ and } \ell(r(T)) = l_j \\ 0 & \text{o/w .} \end{cases}$$

Furthermore, let $\mathcal{T}^{(i-1)} = (T_1, \ldots, T_q)$ be the ordered set of all pairwise non-isomorphic $(i-1)$-unfolding trees in $\mathcal{D}$. Then, the set of child trees $F(r(T))$ below the root of T is represented by the vector $\mathbb{V}_c(T) = (x_1, ..., x_q, x_{q+1})$ with

$$x_j = \begin{cases} |\{T' \in F(r(T)) : T' \equiv T_j\}| & \text{if } j \in [q] \\ 2d - \delta(r(T)) & \text{o/w} \end{cases}$$

where d is the maximum vertex degree in graphs of $\mathcal{D}$.

We give an example of these vector representations in the description of Fig. 6.3.

6.2.2 Wasserstein Distance on Unfolding Tree Vectors

Utilizing the vector representations of unfolding trees above, we are able to reformulate the computation of the structure and depth preserving tree edit distance in terms of the Wasserstein distance. We show that the $\textsc{SdTed}(T, T')$ for i-unfolding trees T and T' can in fact be calculated as the sum of (i) the Wasserstein distance between the root node vectors $\mathbb{V}_r(T), \mathbb{V}_r(T')$ and (ii) the Wasserstein distance between the child tree histogram vectors $\mathbb{V}_c(T), \mathbb{V}_c(T')$. This reformulation requires the distance matrix M_r which represents the cost function γ on $\Sigma_\perp$, as well as the distance matrix M_c which defines the pairwise distances between $(i-1)$-unfolding trees. These matrices are constructed as follows.

For the ordered set $\Sigma_\perp = (l_1, \ldots, l_p, \perp)$ as above, let $M_r \in \mathbb{R}^{(p+1)\times(p+1)}$ be the pairwise distance matrix between labels and blank symbol $\perp = l_{p+1}$ according to $\gamma : \Sigma_\perp \times \Sigma_\perp \to \mathbb{R}$, i.e.:

$$M_r = (m_{ij})_{i,j\in[p+1]} \text{ with } m_{ij} = \gamma(l_i, l_j). \tag{6.4}$$

Analogously, for the ordered set $\mathcal{T}^{(i-1)} = (T_1, \ldots, T_q)$ as above, let $M_c \in \mathbb{R}^{(q+1)\times(q+1)}$ be the pairwise distance matrix between $(i-1)$-unfolding trees and the empty graph $T_\perp = T_{q+1}$, i.e.,

$$M_c = (m_{ij})_{i,j\in[q+1]} \text{ with } m_{ij} = \textsc{SdTed}(T_i, T_j). \tag{6.5}$$

It can be shown that for two depth-i unfolding trees T and T', the distance between their root nodes is equal to the Wasserstein distance $\mathcal{W}_{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T'))$. Furthermore, the calculation of the minimum cost perfect bipartite matching between the sets of child trees below these roots can be reduced to computing the Wasserstein distance between $\mathbb{V}_c(T)$ and $\mathbb{V}_c(T')$, i.e., $\mathcal{W}_{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. To prove this, we utilize the following lemma which follows from the *integral flow theorem*.

Lemma 6.1. *For $x, x' \in \mathbb{N}^d$ and cost matrix $C \in \mathbb{R}^{d\times d}$, there exists a transport matrix $T \in \mathcal{T}(x, x')$ with $T \in \mathbb{N}^{d\times d}$ such that $\langle T, C\rangle = \mathcal{W}_C(x, x')$.*

We refer to Sect. 2.10 for definitions on the Wasserstein distance. Lemma 6.1 implies that the minimum cost perfect bipartite matching between the sets of child trees below $r(T)$ and $r(T')$ is equivalent to $\mathcal{W}_{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. Putting all together we have:

$$\textsc{SdTed}(T, T') \;=\; \mathcal{W}_{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) + \mathcal{W}_{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')) \;. \tag{6.6}$$

An example demonstrating the reformulation of the SdTed in terms of the Wasserstein distance is given in Fig. 6.3.

6.2.3 Unfolding Tree Barycenters

The above reformulation enables the calculation of *barycenters* on sets of unfolding trees to perform Wasserstein k-means (Irpino, Verde, and Carvalho, 2014). A barycenter of a set S of unfolding trees is a point which minimizes the sum of distances to unfolding tree vectors corresponding to S. Similarly to unfolding tree vectors, this barycenter is a pair of real-valued vectors (μ_r, μ_c), where μ_r is the center of the $\mathbb{V}_r$s and μ_c of the $\mathbb{V}_c$s. Formally, the barycenter of S is a pair $(\mu_r, \mu_c) \in (\mathbb{R}^{p+1}, \mathbb{R}^{q+1})$ defined by:

$$\operatorname*{argmin}_{\mu_r, \mu_c} \sum_{T \in S} \mathcal{W}_{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}_{M_c}(\mathbb{V}_c(T), \mu_c). \tag{6.7}$$

While a barycenter, in general, does not correspond to an existing unfolding tree, the Wasserstein distance between an unfolding tree vector $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$ and a center vector $\mu = (\mu_r, \mu_c)$ can be computed nonetheless by:

$$\mathcal{W}_{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}_{M_c}(\mathbb{V}_c(T), \mu_c). \tag{6.8}$$

6.2.4 The Wasserstein k-Means Algorithm for Unfolding Trees

Using the above concepts, the Wasserstein k-means clustering algorithm can be stated in form of Lloyd's algorithm (Lloyd, 1982). For a set $\mathcal{T}^{(i)}$ of depth-i unfolding trees, we perform the following steps:

(i) Randomly select k unfolding trees of $\mathcal{T}^{(i)}$ and assign them as initial centers.

(ii) Assign each $T \in \mathcal{T}^{(i)}$ to its nearest center point (using Eq. 6.8).

(iii) Recalculate the centers of all clusters (using Eq. 6.7).

Steps (ii) and (iii) are repeated until clusters do not change anymore, i.e., the algorithm converges, or a predefined number of iterations has been reached.

6.3 A Faster Kernel Variant

For many graph datasets, the number of distinct Weisfeiler-Lehman labels, or equivalently unfolding trees, grows rapidly with increasing Weisfeiler-Lehman iterations. Although this number is bounded by the total amount of vertices, dealing with large amounts of unfolding

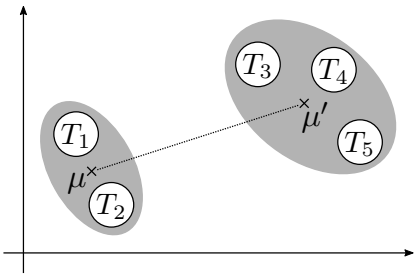


Figure 6.4: Visualization of a clustering over unfolding trees $T_1, \ldots, T_5$ and respective cluster centers μ, μ'. Instead of computing the SdTed for all pairs of unfolding trees, the kernel variant $k^h_{\mathrm{GWL}^*}$ approximates their distances by the distance between their centers. For instance, $\textsc{SdTed}(T_1, T_3)$ is approximated by the distance between $\mu = (\mu_r, \mu_c)$ and $\mu' = (\mu'_r, \mu'_c)$.

trees in the Wasserstein k-means step can become computationally expensive. We therefore propose a more practical variant of our kernel that addresses this issue by approximating distances between unfolding trees using their cluster centers. The basic concept of this approach is depicted in Fig. 6.4.

Consider the calculation of pairwise distances between unfolding trees as described in Sect. 6.1.2. That is, the distances of 0-unfolding trees are defined by the metric γ and the SdTeds for all pairs of $(i+1)$-unfolding trees are computed using distances of i-unfolding trees. To reduce the number of distinct i-unfolding trees $\mathcal{T}^{(i)}$ (or equivalently labels Σ_i), we first perform a clustering $C_1, ..., C_k$ of $\mathcal{T}^{(i)}$ with centers $\mu_1, ..., \mu_k$ as in Sect. 6.2.4. We then approximate the distance between i-unfolding trees $T \in C$ and $T' \in C'$ by the distance between their cluster centers. More precisely, we approximate $\textsc{SdTed}(T, T')$ by

$$\mathcal{W}_{M_r}(\mu_r, \mu'_r) + \mathcal{W}_{M_c}(\mu_c, \mu'_c)$$

where T and T' have been assigned to clusters with centers $\mu = (\mu_r, \mu_c)$ and $\mu' = (\mu'_r, \mu'_c)$, respectively. Subsequently, these distances are used in iteration $i+1$, greatly reducing the number of distance calculations. That is, in contrast to the computation of $k^h_{\mathrm{GWL}}(G, G')$, our approximation variant, denoted $k^h_{\mathrm{GWL}^*}(G, G')$, considers only k labels instead of $|\mathcal{T}^{(i)}|$ labels in iteration i.

The concept of clustering unfolding trees after *each* Weisfeiler-Lehman iteration and then continuing the process with representatives of clusters, can be considered slowing down the Weisfeiler-Lehman relabeling process. It ensures that vertices are split into different Weisfeiler-Lehman label classes at a later iteration when compared to the ordinary Weisfeiler-Lehman method.

6.4 Experimental Evaluation

In this section, we experimentally evaluate the predictive performance of our approach on a set of real-world as well as synthetic datasets. We note that in the following, we limit the evaluation to the approximation kernel GWL* defined in Sect. 6.3. This choice was made

due to the fact that while the original variant is well applicable to sparse graphs such as, for example, molecules, an explicit consideration of all unfolding trees may become computationally too expensive on structurally more complex graphs. Moreover, prior experiments have shown that the predictive performances of the two kernel variants were closely aligned.

To put the performance of our kernel into perspective, we compare it to several state-of-the art graph kernels. We refer to Sect. 3.4 for short descriptions on these kernels. For the sake of clarity, we omit some of the Weisfeiler-Lehman-based kernels that were previously considered in Chapter 5, by noting that their performance is indistinguishable from that achieved by other methods. The empirical results can be summarized as follows:

(i) While our approach does not improve upon existing methods on molecular datasets, it significantly *outperforms* all considered competitor kernels (except the FWL-D kernel variant of Chapter 5) on the ego-network datasets containing structurally more diverse and noisy graphs. The results strongly confirm the assessment that our generalization of the ordinary Weisfeiler-Lehman subtree kernel is beneficial on graphs that exceed graphs of simple and uniform structure.

(ii) We systematically evaluate the predictive performance of our kernel by analyzing its behavior on synthetic datasets containing graphs of different structural complexities. The results show that our approach is more robust to (structural) noise than competing kernels and makes up for the shortcomings of the ordinary Weisfeiler-Lehman subtree kernel.

We measure the prediction performance in terms of accuracy obtained by support vector machines (SVM) using a 10-fold cross-validation. If not explicitly chosen otherwise by the authors of the individual implementations, the SVM parameter C is selected from the value set $2^i : i \in \{-12, -8, -5, -3, -1, 1, 3, 5, 8, 12\}$. In each fold, a grid search is used to identify the optimal kernel parameters. We report the mean and standard deviation over 10 such cross-validation repetitions. For all applied kernels, we used the implementation as provided by Siglidis et al. (2020) or that of the respective authors and ran grid searches using the following kernel specific parameters. For approaches employing the Weisfeiler-Lehman subtree features we chose $h \in \{1, \ldots, 5\}$. In case of the graphlet sampling (GS) kernel, the parameters $\epsilon = 0.1$, $\delta = 0.1$ and $k \in \{3, 4, 5\}$ were applied. For the Weisfeiler-Lehman filtration (FWL-D) kernel, we chose $k \in \{1, ..., 10\}$. Throughout this section, we usually refer to all considered kernel methods using the abbreviations as mentioned below Fig. 6.5.

For our generalized Weisfeiler-Lehman kernel GWL* (as described in Sect. 6.3), we use depth parameter h up to 4. Concerning the costs for relabeling, deletion and insertion operations, we chose unit costs. We perform a total of 3 clusterings (i.e. $|\Theta_i| = 3$) to make up for the randomness caused by the k-means initialization step. We set the number of clusters to $k = \sqrt{|\Sigma_i|}$. This choice for k selects the number of clusters relative to the amount of Weisfeiler-Lehman labels in each iteration and significantly reduces the computational complexity of the GWL* kernel.

							Number of node labels			
Dataset	$\|D\|$	$\|C\|$	$\varnothing\|V\|$	$\varnothing\|E\|$	$\varnothing\frac{\|E\|}{\|V\|}$	Δ	$\|\Sigma_0\|$	$\|\Sigma_1\|$	$\|\Sigma_2\|$	$\|\Sigma_3\|$
DHFR	756	2	42.4	44.5	1.1	4	9	71	630	2478
MUTAG	188	2	17.9	19.8	1.1	4	7	33	174	572
NCI1	4110	2	29.9	32.3	1.1	4	37	292	4058	$23k$
PTC_MR	344	2	14.3	14.7	1.0	4	18	130	780	1987
IMDB-B.	1000	2	19.8	96.5	4.4	135	1	65	2931	3595
EGO-1	200	4	139.0	593.5	4.3	140	1	113	$21k$	$25k$
EGO-2	200	4	178.6	1444.9	8.1	180	1	141	$33k$	$35k$
EGO-3	200	4	220.0	2613.5	11.9	203	1	170	$42k$	$43k$
EGO-4	200	4	259.8	4135.8	15.9	237	1	209	$51k$	$52k$

Table 6.1: Structural information of graph benchmark datasets. $|D|$, $|C|$, and Δ, denote the number of graphs, number of classes, and maximum degree in a dataset, respectively. $|\Sigma_0|$ is the number of distinct vertex labels. $|\Sigma_1|, |\Sigma_2|, |\Sigma_3|$ are the amounts of distinct Weisfeiler-Lehman labels for depth $h = 1, 2, 3$, respectively.

6.4.1 Real-world Benchmarks

Figure 6.5 lists the classification accuracies for real-world benchmark datasets. The respective runtimes are provided in Table 6.2. On the set of molecular datasets, i.e., on DHFR, MUTAG, NCI and PTC-MR, our approach is in close range to the best performing competitor kernels. Only for NCI1, there is a noticeable performance gap. The overall results suggest that a relaxation of the Weisfeiler-Lehman kernel is not advantageous when applied to these simple molecular graphs. This may be explained by the assumption that structurally similar unfolding trees can have completely opposing chemical properties. Clustering might thus even be disadvantageous for this kind of data.

Moving on to more complex graphs, we observe a different picture. While there are no large discrepancies between our method and the best performing comparison kernels on datasets IMDB and EGO-1 (which both have an average node-to-edge ratio up to roughly $1:4$), the GWL* kernel considerably outperforms all other kernels (except our FWL-D kernel variant) on the three remaining EGO datasets. The performance gap between the GWL* kernel and the ordinary Weisfeiler-Lehman subtree kernel becomes increasingly larger with growing density and number of distinct Weisfeiler-Lehman labels in the dataset graphs, leading to a roughly 20% accuracy difference. It is noteworthy that in case of the EGO datasets, already for depth $h = 2$ nearly all unfolding trees (i.e. depth-2 unfolding trees) appear only once in the respective dataset (see Tab. 6.1). Thus, the ordinary Weisfeiler-Lehman subtree kernel is not able to profit from any structural information exceeding node degrees, as graphs share almost no i-unfolding trees for $i \geqslant 2$. In contrast, our approach clearly improves upon this limitation by relaxing the strict comparison of unfolding trees. Given these two high level experimental results, we conjecture that identifying *similar* unfolding trees instead of *identical* unfolding trees becomes more advantageous as the dataset graphs become more complex and diverse.

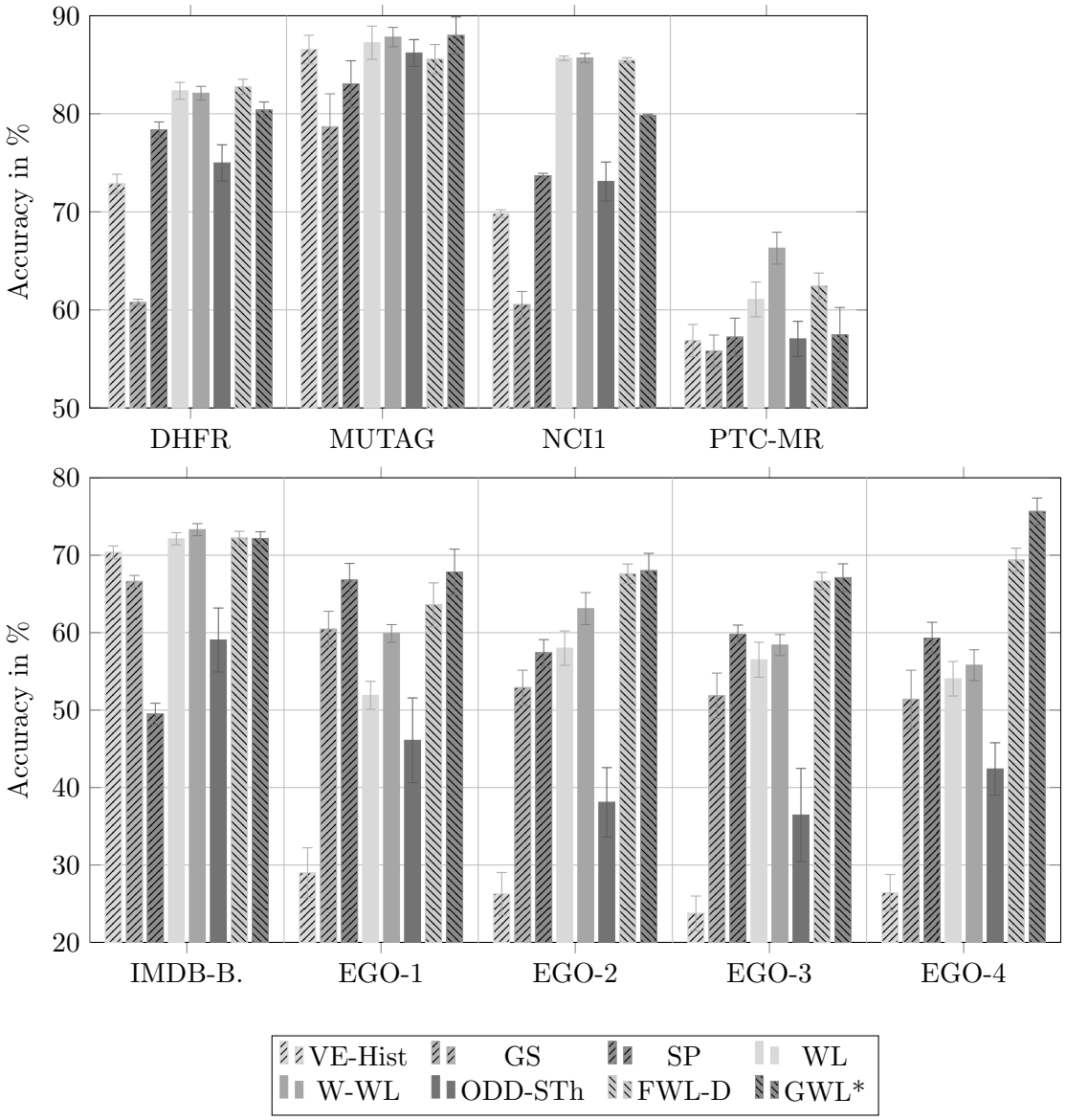


Figure 6.5: Classification accuracies and standard deviations on real-world benchmark datasets. We compare our approach to the Label Histogram (VE-Hist) kernel (see Sect. 3.4), the graphlet sampling (GS) kernel (Shervashidze et al., 2009), the shortest-path (SP) kernel (Borgwardt and Kriegel, 2005), the ordinary Weisfeiler-Lehman (WL) subtree kernel (Shervashidze et al., 2011), the Wasserstein Weisfeiler-Lehman (W-WL) kernel (Togninalli et al., 2019), the ordered decomposition DAG (ODD-STh) kernel (Martino, Navarin, and Sperduti, 2012), and the Weisfeiler-Lehman filtration (FWL-D) kernel (see Chapter 5). The approximate generalized Weisfeiler-Lehman kernel (see Sect. 6.3) is denoted by GWL*.

	GS	SP	WL	W-WL	ODD-STh	FWL-D	GWL*
DHFR	77.17	4.34	0.41	290.32	4.52	6.26	1.58
MUTAG	20.07	0.20	0.05	9.61	0.29	0.53	0.87
NCI1	434.32	12.79	2.16	6457.34	147.53	63.84	19.60
PTC-MR	38.64	0.30	0.08	32.49	0.71	0.91	2.01
IMDB-B.	135.89	2.39	0.51	297.06	4.12	5.21	7.91
EGO-1	21.62	15.96	0.50	103.46	56.29	7.36	339.85
EGO-2	23.38	33.27	0.92	173.11	145.39	14.54	1256.58
EGO-3	24.44	60.01	1.47	260.58	298.77	23.18	2449.26
EGO-4	26.58	97.18	2.16	353.57	555.43	34.90	3834.94

Table 6.2: Runtimes of kernels (in seconds) measuring the time for computing the kernel matrix. All experiments were performed on an AMD 3900X processor (12 cores) with 64GB of memory.

Following the idea of our generalized Weisfeiler-Lehman kernel, Bause and Kriege (2022) propose a conceptually similar approach. They suggest slowing down the Weisfeiler-Lehman vertex relabeling scheme using a method which closely resembles our approach outlined in Sect. 6.3, with the key distinction being the type of clustering function. We refer to Sect. 3.4 for a brief discussion on this method. Due to the resemblance between the two kernels, we do not include the results obtained by Bause and Kriege (2022) in Fig. 6.5, by noting that the obtained predictive performances are nearly identical.

6.4.2 Investigating Noise and Structural Deviation

To systematically evaluate the predictive performance of our kernel on graphs with varying structural complexity, we consider graph datasets generated by the stochastic block model (Wang and Wong, 1987). In the following, we describe their generation process and the corresponding classification task. Subsequently, we discuss the results obtained by the generalized Weisfeiler-Lehman kernel and a selected subset of competitor kernels on this kind of graphs.

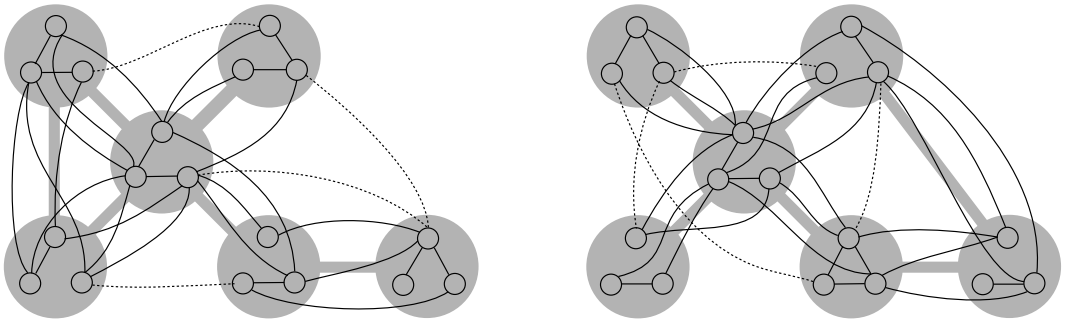
Block Model Graphs Let T be some random tree of a predefined size. Create two non-isomorphic graphs G_1 and G_2 by adding a new edge to T. [1] The graphs G_1 and G_2 represent the underlying structure of the block model graphs. A block model graph for $G \in \{G_1, G_2\}$ is generated in the following fashion: Let $c, m_x \in \mathbb{N}$ and $p \in [0, 1]$.

(i) For each $v \in V(G)$, create a set of c vertices $C_v = \{u_1, ..., u_c\}$.

(ii) For all $u \in C_v, u' \in C_{v'}$, connect u and u' by an edge with probability p if $\{v, v'\} \in E(G)$ or $v = v'$.

(iii) Connect a number of m_x prior unconnected random vertex pairs $\{u, u'\}$ by an edge.

[1] To ensure that the classification task is non-trivial, we furthermore require that G_1 and G_2 have the same multiset of vertex degrees.



(a) (b)

Figure 6.6: Graphs generated by slightly different underlying structures (depicted in gray). Each block in the underlying structure contains 3 vertices (i.e., $c = 3$). Two vertices are connected by an edge with probability p if they belong to the same block or their blocks are connected in the underlying structure. Furthermore, both graphs contain $m_x = 4$ additional noise edges (depicted by dashed lines).

We generate a number of block model graphs for both graphs G_1 and G_2. The classification task is then to predict whether a block model graph was generated w.r.t. G_1 or G_2. Fig. 6.6 depicts an example of two such generated graphs.

All datasets considered in the following evaluations were created starting with a random tree of size 16 which was extended by a single random edge resulting in graphs G_1, G_2. For each classification task, we generated 200 random graphs for each $G \in \{G_1, G_2\}$. The number of vertices c contained in a block was set to 8 in all experiments. The remaining parameters were selected as stated in Fig. 6.7. For each set of parameter choices, we generated 5 datasets and provide the mean accuracy.

Evaluation We now investigate the effect of noise and structural deviation in block model graphs on the predictive performance of our kernel. To this end, we vary the values of the parameters m_x and p. The parameter p governs the probabilities of edges within and between vertex blocks while m_x indicates the number of randomly added noise edges. Hence, they directly influence the noise and structural deviation of graphs within a class. Note that we limit our discussion to the competitor kernels SP, WL, P-WL and ODD-STh by noting that all previously considered graph kernels underperformed or were indistinguishable from of one of these methods.

Figure 6.7(a) investigates the methods' robustness to noise. Higher values of m_x increase the deviation of graphs within the same class. In this experiment, we fixed the edge probability to $p = 1.0$. For $m_x = 0$, all graphs in the database have 128 nodes and 2048 edges, and graphs of the same underlying structure are pairwise isomorphic. Thus, all kernels perfectly classify the graphs. While our method achieves 100% accuracy for all choices of m_x, the remaining kernels gradually and significantly decrease in predictive performance with increasing values for m_x. It is noteworthy that for the case of 100 noise edges, no competitor kernel but P-WL performs significantly better than random.

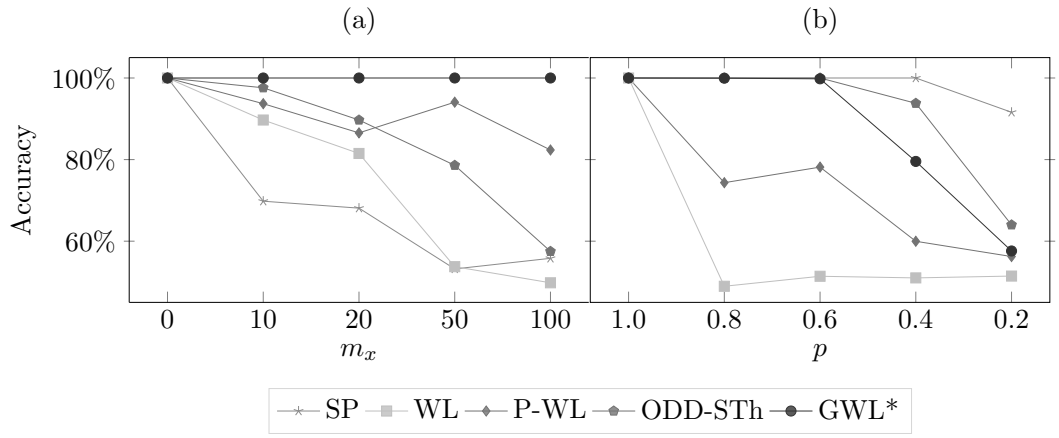


Figure 6.7: Classification accuracies for synthetic dataset evaluations. Fig. (a) analyzes the influence of different amounts of noise edges m_x (with $p = 1.0$ fixed) and Fig. (b) shows results obtained for different values of the edge probability parameter p (with $m_x = 0$ fixed). Block size c has been set to 8 for all cases.

Figure 6.7(b) analyzes the methods' ability to identifying the underlying structure using different values of the edge probability parameter p. In this experiment, we fixed the number of additional noise edges to $m_x = 0$. The lower the value p, the more the dataset graphs within a class deviate from each other, and the less of the underlying structure is being reflected. While in the trivial case $p = 1.0$ (where graphs belonging to the same class are pairwise isomorphic) all methods achieve 100% accuracy, we observe a rapid performance decline for the WL and P-WL methods for smaller values of p. In particular, WL does not perform better than random other than for the trivial case. While ODD-STh underperformed in all previous experiments, it seems that it is well suited for this kind of structural deviation.

In summary, it is apparent that kernels based on the comparison of Weisfeiler-Lehman labels by equality are less suited when structural noise distorts the graphs or when the graphs in a class structurally deviate significantly. Our method mitigates this drawback: Its ability to identify *similar* vertex neighborhoods leads to major increases in predictive performance on datasets containing *noisy* and *structurally diverse* graphs.

6.5 Summary and Concluding Remarks

We experimentally demonstrated a drawback of the original Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) caused by the rigid comparison of Weisfeiler-Lehman labels w.r.t. equality. To overcome this limitation, we introduced a generalization of this kernel, which allows for a finer similarity measure between Weisfeiler-Lehman labels. The experimental results clearly show that the proposed generalization outperforms other state-of-the-art kernels on several real-world datasets. Specifically, we demonstrated the advantage of our approach on graphs of structural complexity beyond the typically considered molecular graphs

of small pharmacological compounds.

For simplicity, we limited the discussion to the Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) which is arguably the most established member of the Weisfeiler-Lehman graph kernel family. However, we note that our approach can be applied to *all* graph kernels relying on the Weisfeiler-Lehman label propagation algorithm.

Our results raise several interesting questions for further research. One research question is to investigate the kernel parameter concerned with the number of clusters. Clearly, this value governs the degree to which the ordinary Weisfeiler-Lehman subtree kernel is being relaxed. We note that choosing appropriate values for k is a general problem of k-means approaches. In the experimental evaluation, k was chosen relative to the total number of unfolding trees. While this choice led to very impressive predictive performances on the EGO datasets, our results showed that it may also be disadvantages on datasets such as NCI1. Since our kernel variant allows for several clusterings, a possible solution is to simply consider several values for k.

Another particularly relevant question is whether the tree edit distance can directly be used as a ground distance in vertex matching kernels (see e.g., Kriege, Giscard, and Wilson, 2016 and Togninalli et al., 2019). Employing such matching kernels would eliminate the need for a hard partitioning of unfolding trees. Unfortunately, straightforward approaches such as replacing the ground distance in the Wasserstein Weisfeiler-Lehman kernel (Togninalli et al., 2019) by the tree edit distance does not yield a positive semi-definite kernel in general. However, there has been comprehensive research addressing the problem of dealing with indefinite kernels. For instance, indefinite kernel matrices may be converted to positive definite ones using spectrum transformation approaches, which, e.g., aim at flipping all negative eigenvalues to zero (Wu, Chang, and Zhang, 2005) or alter the matrix's diagonal by adding a positive term (Roth et al., 2003). We stress that our distance function SdTed is not restricted to be applied in the context of graph kernels. In fact, using SdTed as Wasserstein ground distance for computing distances between graphs, directly allows the application of other (dis-)similarity-based classifiers such as k-nearest-neighbor approaches.

Finally, it would be interesting to study other meaningful similarities between Weisfeiler-Lehman labels that allow for a *faster* calculation of minimum cost perfect bipartite matchings (or Wasserstein distances). As the cost function γ on the original vertex labels can be defined by an arbitrary metric, the application of our approach to *attributed graphs* is another natural research question.

7

Conclusion

In this chapter, we first briefly summarize the thesis and discuss some implications of our contributions. We then conclude by pointing out some directions for possible future research.

7.1 Summary and Discussion

In this thesis, we designed efficient graph kernels based on tree patterns that are identified by constrained homomorphisms. The notion of constrained homomorphisms served as the pattern embedding operator and was motivated by the conceptual relationship between graph homomorphisms and subgraph isomorphisms as well as the apparent complexity gap between them when considering tree patterns. One of the goals of this thesis was to utilize tractable instances of constrained homomorphisms to efficiently extract suitable pattern sets, leading to rich graph representations and enabling powerful similarity measures on graphs. We achieved this goal by considering two particular kinds of constrained homomorphisms: partially injective and locally bijective homomorphisms.

Although we restricted our attention to these two notions, the results of this thesis clearly demonstrate the suitability of this kind of pattern embedding operator for graph classification purposes. In particular, the positive correlation between increasingly constrained homomorphisms and improved predictive performances, as observed in Chapter 4, experimentally confirmed the significance of injectivity constraints. This insight ultimately motivated the utilization of locally bijective homomorphisms in form of the Weisfeiler-Lehman method, leading to sophisticated graph similarity measures.

The notion of partially injective homomorphism was introduced to effectively bridge the gap between graph homomorphisms and subgraph isomorphisms. In order to preserve much of the advantageous properties of subgraph isomorphisms for graph classification purposes while remaining efficiently decidable, the goal was to retain as much injectivity in the pattern matching operator as possible. This was achieved by utilizing positive complexity results on deciding homomorphisms from graphs of bounded treewidth. Considering the results by Grohe (2007), we can infer that this graph class offers a tight positive border on efficiently decidable partially injective homomorphisms. Although the concept of partially injective homomorphisms has implications beyond the contributions of this thesis, we focused on developing a mining algorithm using this kind of pattern embedding operator in order to obtain

powerful graph representations. In our experimental evaluation, it was shown that this approach offers an attractive trade-off between efficiency and predictive performance.

Subsequently, we turned our attention towards approaches that utilize locally bijective homomorphisms as the pattern embedding operator. While this kind of constrained homomorphisms is known to be efficiently decidable for tree patterns (Chaplick et al., 2015) and therefore directly enables pattern mining-based approaches as outlined in the previous paragraph, in this thesis we focused on a specific case by restricting the class of tree patterns. In particular, we considered rooted trees with all leaves having the same depth. Deciding locally bijective homomorphisms on this kind of trees can be done efficiently using the Weisfeiler-Lehman method. In comparison to the previous approach, this method offers a distinguishing advantage: its vertex relabeling scheme directly generates tree patterns (or, more precisely, encodings thereof), without the necessity for a costly mining process. However, in this thesis, we identified the drawback of this method to the "specificity" of the outputted patterns, leading to sparse graph representations and overly coarse graph similarity measures. To address this limitation, this thesis introduced two approaches to improve upon this drawback.

The first approach borrowed concepts from persistent homology which allow to view graphs at different resolution levels. We utilized this concept to extract Weisfeiler-Lehman patterns from sequences of evolving graphs, leading to fine-grained graph similarity measures. A noteworthy property that distinguishes this kind of kernels from most other graph kernels is its capability to meaningfully utilize edge weights. Although the main emphasis was on Weisfeiler-Lehman patterns, we stress that our approach can be applied to any kind of graph feature. Nonetheless, we showed that Weisfeiler-Lehman patterns are a particularly interesting choice of graph patterns as they yield specifically powerful graph kernels.

For the second approach, we proposed a relaxation of the ordinary Weisfeiler-Lehman subtree kernel (Shervashidze et al., 2011) by applying a fine-grained comparison between Weisfeiler-Lehman patterns. This (dis-)similarity measure was defined by a particularly fitted tree edit distance which provides a semantically meaningful and efficiently computable distance on such patterns. Reformulating this distance measure in terms of the Wasserstein distance allowed for a k-means clustering approach on Weisfeiler-Lehman patterns. Using these results, we defined a graph kernel which generalizes the ordinary Weisfeiler-Lehman kernel by clustering similar patterns. While our use of the proposed distance measure was limited to clustering, we argue that it is furthermore relevant to other applications such as, for example, node classification.

In an experimental assessments of these Weisfeiler-Lehman-based kernels, it became apparent that our approaches significantly outperformed competitor kernels on several datasets containing graphs that exceed the typically considered molecular graphs. This result is particularly noteworthy considering the fact that a majority of previously proposed graph kernels focuses on molecular datasets, on which almost all graph kernels perform nearly identically.

7.2 Outlook

The contents presented in this dissertation raise a number of interesting future research questions. While we already discussed open questions and directions in each chapter's conclusion,

we now turn to potential research questions that conclude from this thesis.

Although constrained homomorphisms have been a central notion of this thesis, our intention was not to conduct an extensive study on this class of pattern matching operators. Instead, we restricted our attention to two particular instances with the intention of designing suitable and efficient graph kernels. Naturally, there is an abundance of constrained homomorphisms that may serve as candidates for this purpose. Regarding locally constrained homomorphisms, it was noted that apart from the examined locally bijective homomorphisms, both locally surjective and locally injective homomorphisms can also be computed in polynomial time when the pattern class is restricted to trees (Chaplick et al., 2015). Recall that Chapter 4 points out the advantages of injectivity in pattern matching operators. Furthermore, Chapters 5 and 6 identify the drawback of the Weisfeiler-Lehman method to be the pattern graphs' "specificity" which is ultimately caused by the local surjectiveness of the embedding operator. Consequently, we conjecture that locally injective homomorphisms are a particularly promising pattern matching candidate on tree patterns for future research. An alternate strategy for devising constrained homomorphisms involves partially injective homomorphisms while considering other tractable graph classes instead of (edge-maximal) bounded treewidth graphs as considered in Chapter 4. Due to the result of Grohe (2007), it needs to be noted that all such graph classes are necessarily subclasses of bounded treewidth graphs (see Bodlaender, 1993 for a list of such classes). Nonetheless, in light of the practical inefficiency of deciding homomorphism from graphs of bounded treewidth using the standard dynamic programming approach (see Sect. 2.4.1), a natural research direction is to explore alternative graph classes for which there exist more practically feasible decision algorithms. An example are outerplanar graphs which have treewidth at most 2 and from which homomorphisms can hence be decided in polynomial time. Consequently, an interesting research question is whether partially injective homomorphisms w.r.t. outerplanar graphs can be decided fast in practice and whether the obtained patterns lead to adequate predictive performances.

Another research direction is the application of this thesis's results and concepts to graph neural network approaches. One major direction that stands out is motivated by the close relationship of the Weisfeiler-Lehman method and the message passing scheme of GNNs (see Sect. 3.5 for a brief discussion). Particularly, Corollary 5.2 in Chapter 5 shows that our results on the Weisfeiler-Lehman filtration kernels can directly be utilized to improve the expressive powers of GNNs. In fact, the generality of graph filtrations allows for arbitrarily powerful GNNs. A natural research question emerging from this corollary is whether filtration functions can be learned by a neural network instead having to be provided in advance. Thus, the concept of graph filtrations could be directly integrated into an end-to-end graph neural network approach. A conceptually different paradigm to increase the expressive power of GNNs is to enrich graphs with subgraph information. This is achieved by incorporating the embedding information of a fixed set of patterns $\mathcal{P}$ into the node representations of graphs. Several approaches have been proposed that consider, e.g., subgraph isomorphism counts (Bouritsas et al., 2023) or homomorphism counts (Barceló et al., 2021) from the patterns in $\mathcal{P}$. A natural research direction is the application of this concept to partially injective homomorphisms. More precisely, it would be interesting to study how the expressive power of subgraph-enhanced GNNs is effected when considering different degrees of partial injectivity in the embedding operator.

Bibliography

Rakesh Agrawal and Ramakrishnan Srikant (1994). "Fast Algorithms for Mining Association Rules in Large Databases". In: *International Conference on Very Large Data Bases (VLDB)*, pp. 487–499.

Martial Agueh and Guillaume Carlier (2011). "Barycenters in the Wasserstein Space". In: *SIAM J. Math. Anal.* 43.2, pp. 904–924. DOI: 10.1137/100805741.

Mehmet Emin Aktas, Esra Akbas, and Ahmed El Fatmaoui (2019). "Persistence homology of networks: methods and applications". In: *Appl. Netw. Sci.* 4.1, 61:1–61:28. DOI: 10.1007/s41109-019-0179-3.

Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski (1987). "Complexity of Finding Embeddings in a k-Tree". In: *SIAM Journal on Algebraic Discrete Methods* 8.2, pp. 277–284. DOI: 10.1137/0608024.

Vikraman Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert (2012). "The isomorphism problem for k-trees is complete for logspace". In: *Inf. Comput.* 217, pp. 1–11. DOI: 10.1016/j.ic.2012.04.002.

László Babai (2016). "Graph isomorphism in quasipolynomial time [extended abstract]". In: *Symposium on Theory of Computing (STOC)*, pp. 684–697. DOI: 10.1145/2897518.2897542.

László Babai, Paul Erdös, and Stanley M. Selkow (1980). "Random Graph Isomorphism". In: *SIAM Journal on Computing* 9.3, pp. 628–635. DOI: 10.1137/0209047.

Pablo Barceló, Floris Geerts, Juan L. Reutter, and Maksimilian Ryschkov (2021). "Graph Neural Networks with Local Graph Parameters". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 25280–25293.

Franka Bause and Nils Morten Kriege (2022). "Gradual Weisfeiler-Leman: Slow and Steady Wins the Race". In: *Learning on Graphs Conference (LoG)*, 20:1–20:18.

Christian Berg, Jens Peter Reus Christensen, and Paul Ressel (1984). "General Results on Positive and Negative Definite Matrices and Kernels". In: *Harmonic Analysis on Semi-*

groups: Theory of Positive Definite and Related Functions. New York, NY: Springer New York, pp. 66–85. DOI: 10.1007/978-1-4612-1128-0_3.

Philip Bille (2005). "A survey on tree edit distance and related problems". In: *Theor. Comput. Sci.* 337.1-3, pp. 217–239. DOI: 10.1016/j.tcs.2004.12.030.

David B. Blumenthal (2019). "New Techniques for Graph Edit Distance Computation". PhD thesis. Free University of Bozen-Bolzano.

Hans L. Bodlaender (1993). "A Tourist Guide through Treewidth". In: *Acta Cybern.* 11.1-2, pp. 1–21.

Hans L. Bodlaender and Ton Kloks (1996). "Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs". In: *Journal of Algorithms* 21.2, pp. 358–402. DOI: 10.1006/jagm.1996.0049.

Karsten M. Borgwardt, M. Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck (2020). "Graph Kernels: State-of-the-Art and Future Challenges". In: *Foundations and Trends in Machine Learning* 13.5-6. DOI: 10.1561/2200000076.

Karsten M. Borgwardt and Hans-Peter Kriegel (2005). "Shortest-Path Kernels on Graphs". In: *International Conference on Data Mining (ICDM), Proceedings*, pp. 74–81. DOI: 10.1109/ICDM.2005.132.

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik (1992). "A Training Algorithm for Optimal Margin Classifiers". In: *Conference on Computational Learning Theory (COLT), Proceedings*, pp. 144–152. DOI: 10.1145/130385.130401.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein (2023). "Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 45.1, pp. 657–668. DOI: 10.1109/TPAMI.2022.3154319.

Jin-yi Cai, Martin Fürer, and Neil Immerman (1992). "An optimal lower bound on the number of variables for graph identifications". In: *Combinatorica* 12.4, pp. 389–410. DOI: 10.1007/BF01305232.

Ashok K. Chandra and Philip M. Merlin (1977). "Optimal Implementation of Conjunctive Queries in Relational Data Bases". In: *Symposium on Theory of Computing (STOC)*, pp. 77–90. DOI: 10.1145/800105.803397.

Steven Chaplick, Jirí Fiala, Pim van 't Hof, Daniël Paulusma, and Marek Tesar (2015). "Locally constrained homomorphisms on graphs of bounded treewidth and bounded degree". In: *Theor. Comput. Sci.* 590, pp. 86–95. DOI: 10.1016/j.tcs.2015.01.028.

Yun Chi, Yirong Yang, and Richard R. Muntz (2003). "Indexing and Mining Free Trees". In: *International Conference on Data Mining (ICDM), Proceedings*, pp. 509–512. DOI: 10.1109/ICDM.2003.1250964.

Leonardo Cotta, Christopher Morris, and Bruno Ribeiro (2021). "Reconstruction for Powerful Graph Representations". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1713–1726.

Marco Cuturi (2013). "Sinkhorn Distances: Lightspeed Computation of Optimal Transport". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2292–2300.

Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi (2002). "Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics". In: *Principles and Practice of Constraint Programming (CP), Proceedings*, pp. 310–326. DOI: 10.1007/3-540-46135-3_21.

Brian A. Davey and Hilary A. Priestley (2002). *Introduction to Lattices and Order, Second Edition*. Cambridge University Press. DOI: 10.1017/CBO9780511809088.

Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch (1991). "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of Medicinal Chemistry* 34.2, pp. 786–797. DOI: 10.1021/jm00106a046.

Holger Dell, Martin Grohe, and Gaurav Rattan (2018). "Lovász Meets Weisfeiler and Leman". In: *International Colloquium on Automata, Languages, and Programming, (ICALP)*, 40:1–40:14. DOI: 10.4230/LIPIcs.ICALP.2018.40.

Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis (2005). "Frequent Substructure-Based Approaches for Classifying Chemical Compounds". In: *IEEE Trans. Knowl. Data Eng.* 17.8, pp. 1036–1050. DOI: 10.1109/TKDE.2005.127.

Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos (2002). "Counting H-colorings of partial k-trees". In: *Theor. Comput. Sci.* 281.1-2, pp. 291–309. DOI: 10.1016/S0304-3975(02)00017-8.

Reinhard Diestel (2012). *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer.

Anton Dries and Siegfried Nijssen (2012). "Mining Patterns in Networks using Homomorphism". In: *SIAM International Conference on Data Mining (SDM)*, pp. 260–271. DOI: 10.1137/1.9781611972825.23.

Herbert Edelsbrunner and John Harer (2010). *Computational Topology - an Introduction*. American Mathematical Society.

Jirí Fiala and Jan Kratochvíl (2008). "Locally constrained graph homomorphisms - structure, complexity, and applications". In: *Comput. Sci. Rev.* 2.2, pp. 97–111. DOI: 10.1016/j.cosrev.2008.06.001.

Jacob Focke, Leslie Ann Goldberg, and Stanislav Zivný (2019). "The Complexity of Counting Surjective Homomorphisms and Compactions". In: *SIAM J. Discret. Math.* 33.2, pp. 1006–1043. DOI: 10.1137/17M1153182.

Michael Garey and David S. Johnson (1979). *Computers and Intractability, A Guide to the Theory of NP-Completeness*. 1st ed. New York: W. H. Freeman.

Thomas Gärtner, Peter A. Flach, and Stefan Wrobel (2003). "On Graph Kernels: Hardness Results and Efficient Alternatives". In: *Computational Learning Theory and Kernel Machines (COLT/Kernel)*, pp. 129–143. DOI: 10.1007/978-3-540-45167-9_11.

Lise Getoor and Christopher P. Diehl (2005). "Link mining: a survey". In: *SIGKDD Explor.* 7.2, pp. 3–12. DOI: 10.1145/1117454.1117456.

Martin Grohe (2007). "The complexity of homomorphism and constraint satisfaction problems seen from the other side". In: *Journal of the ACM* 54.1, 1:1–1:24. DOI: 10.1145/1206035.1206036.

David Haussler (July 1999). *Convolution Kernels on Discrete Structures*. Tech. rep. UCSC-CRL-99-10. Univerisity of California - Santa Cruz.

Pavol Hell and Jaroslav Nesetril (2004). *Graphs and homomorphisms*. Vol. 28. Oxford lecture series in mathematics and its applications. Oxford University Press.

Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan (2001). "The Predictive Toxicology Challenge 2000-2001". In: *Bioinformatics* 17.1, pp. 107–108. DOI: 10.1093/bioinformatics/17.1.107.

Tamás Horváth, Thomas Gärtner, and Stefan Wrobel (2004). "Cyclic pattern kernels for predictive graph mining". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 158–167. DOI: 10.1145/1014052.1014072.

Tamás Horváth and Jan Ramon (2010). "Efficient frequent connected subgraph mining in graphs of bounded tree-width". In: *Theoretical Computer Science* 411.31-33, pp. 2784–2797. DOI: 10.1016/j.tcs.2010.03.030.

Tamás Horváth and György Turán (2001). "Learning logic programs with structured background knowledge". In: *Artificial Intelligence* 128.1-2, pp. 31–97. DOI: 10.1016/S0004-3702(01)00062-5.

Antonio Irpino, Rosanna Verde, and Francisco de A. T. de Carvalho (2014). "Dynamic clustering of histogram data based on adaptive squared Wasserstein distances". In: *Expert Syst. Appl.* 41.7, pp. 3351–3366. DOI: 10.1016/j.eswa.2013.12.001.

Chuntao Jiang, Frans Coenen, and Michele Zito (2013). "A survey of frequent subgraph mining algorithms". In: *Knowl. Eng. Rev.* 28.1, pp. 75–105. DOI: 10.1017/S0269888912000331.

David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis (1988). "On Generating All Maximal Independent Sets". In: *Inf. Process. Lett.* 27.3, pp. 119–123. DOI: 10.1016/0020-0190(88)90065-8.

Leonid V. Kantorovich (1960). "Mathematical Methods of Organizing and Planning Production". In: *Management Science* 6, pp. 366–422.

Sandra Kiefer (2020). "Power and limits of the Weisfeiler-Leman algorithm". PhD thesis. RWTH Aachen University.

Sandra Kiefer and Daniel Neuen (2019). "The Power of the Weisfeiler-Leman Algorithm to Decompose Graphs". In: *Mathematical Foundations of Computer Science, (MFCS)*. Vol. 138, 45:1–45:15. DOI: 10.4230/LIPIcs.MFCS.2019.45.

Ton Kloks (1994). *Treewidth - Computations and Approximations*. Electronic Edition. Berlin, Heidelberg: Springer-Verlag, pp. 1–23.

Nils M. Kriege, Pierre-Louis Giscard, and Richard C. Wilson (2016). "On Valid Optimal Assignment Kernels and Applications to Graph Classification". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1615–1623.

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris (2020). "A survey on graph kernels". In: *Applied Network Science* 5.1, p. 6. DOI: 10.1007/s41109-019-0195-3.

Nils M. Kriege, Marion Neumann, Christopher Morris, Kristian Kersting, and Petra Mutzel (2019). "A unifying view of explicit and implicit feature maps of graph kernels". In: *Data Min. Knowl. Discov.* 33.6, pp. 1505–1547. DOI: 10.1007/s10618-019-00652-0.

H. W. Kuhn (1955). "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1-2, pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109.

Michihiro Kuramochi and George Karypis (2004). "An Efficient Algorithm for Discovering Frequent Subgraphs". In: *IEEE Trans. Knowl. Data Eng.* 16.9, pp. 1038–1051. DOI: 10.1109/TKDE.2004.33.

Patrick R. J. van der Laag and Shan-Hwei Nienhuys-Cheng (1998). "Completeness and Properness of Refinement Operators in Inductive Logic Programming". In: *J. Log. Program.* 34.3, pp. 201–225. DOI: 10.1016/S0743-1066(97)00077-0.

Tam Le, Makoto Yamada, Kenji Fukumizu, and Marco Cuturi (2019). "Tree-Sliced Variants of Wasserstein Distances". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 12283–12294.

R. M. R. Lewis (2016). *A Guide to Graph Colouring - Algorithms and Applications*. Springer. DOI: 10.1007/978-3-319-25730-3.

Stuart P. Lloyd (1982). "Least squares quantization in PCM". In: *IEEE Trans. Inf. Theory* 28.2, pp. 129–136. DOI: 10.1109/TIT.1982.1056489.

Yangjing Long (2014). “Graph Relations and Constrained Homomorphism Partial Orders”. PhD thesis. University of Leipzig.

László Miklós Lovász (1967). “Operations with structures”. In: *Acta Mathematica Academiae Scientiarum Hungarica* 18, pp. 321–328.

Giovanni Da San Martino, Nicolò Navarin, and Alessandro Sperduti (2012). “A Tree-Based Kernel for Graphs”. In: *SIAM International Conference on Data Mining (SDM)*, pp. 975–986. DOI: 10.1137/1.9781611972825.84.

Dániel Marx and Michal Pilipczuk (2014). “Everything you always wanted to know about the parameterized complexity of Subgraph Isomorphism (but were afraid to ask)”. In: *Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 542–553. DOI: 10.4230/LIPIcs.STACS.2014.542.

David W Matula (1968). “An algorithm for subtree identification”. In: *Siam Rev* 10, pp. 273–274.

David W. Matula (1978). “Subtree Isomorphism in O(n5/2)”. In: *Algorithmic Aspects of Combinatorics*, pp. 91–106. DOI: https://doi.org/10.1016/S0167-5060(08)70324-8.

Brendan D. McKay and Adolfo Piperno (2014). “Practical graph isomorphism, II”. In: *J. Symb. Comput.* 60, pp. 94–112. DOI: 10.1016/j.jsc.2013.09.003.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann (2020). “TUDataset: A collection of benchmark datasets for learning with graphs”. In: *CoRR* abs/2007.08663.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). “Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks”. In: *AAAI Conference on Artificial Intelligence, (AAAI)*, pp. 4602–4609. DOI: 10.1609/aaai.v33i01.33014602.

Aida Mrzic, Pieter Meysman, Wout Bittremieux, Pieter Moris, Boris Cule, Bart Goethals, and Kris Laukens (2018). “Grasping frequent subgraph mining for bioinformatics applications”. In: *BioData Min.* 11.1, 20:1–20:24. DOI: 10.1186/s13040-018-0181-9.

Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak A. Rao, and Bruno Ribeiro (2019). “Relational Pooling for Graph Representations”. In: *International Conference on Machine Learning (ICML), Proceedings*, pp. 4663–4673.

Shan-Hwei Nienhuys-Cheng and Ronald de Wolf (1997). *Foundations of Inductive Logic Programming*. Vol. 1228. Lecture Notes in Computer Science. Springer. DOI: 10.1007/3-540-62927-0.

Siegfried Nijssen and Joost N. Kok (2004). “Frequent graph mining and its application to molecular databases”. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, pp. 4571–4577. DOI: 10.1109/ICSMC.2004.1401252.

— (2005). "The Gaston Tool for Frequent Subgraph Mining". In: *Electronic Notes in Theoretical Computer Science* 127.1, pp. 77–87. DOI: https://doi.org/10.1016/j.entcs.2004.12.039.

Giannis Nikolentzos, Polykarpos Meladianos, Stratis Limnios, and Michalis Vazirgiannis (2018). "A Degeneracy Framework for Graph Similarity". In: *International Joint Conference on Artificial Intelligence, (IJCAI), Proceedings*, pp. 2595–2601. DOI: 10.24963/ijcai.2018/360.

Leslie O'Bray, Bastian Rieck, and Karsten M. Borgwardt (2021). "Filtration Curves for Graph Representation". In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Proceedings*, pp. 1267–1275. DOI: 10.1145/3447548.3467442.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer (2021). "DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 21997–22009.

Gabriel Peyré and Marco Cuturi (2019). "Computational Optimal Transport". In: *Foundations and Trends in Machine Learning* 11.5-6, pp. 355–607. DOI: 10.1561/2200000073.

Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt (2015). "A stable multi-scale kernel for topological machine learning". In: *IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*, pp. 4741–4748. DOI: 10.1109/CVPR.2015.7299106.

Bastian Rieck, Christian Bock, and Karsten M. Borgwardt (2019). "A Persistent Weisfeiler-Lehman Procedure for Graph Classification". In: *International Conference on Machine Learning (ICML), Proceedings*, pp. 5448–5458.

Neil Robertson and Paul D. Seymour (1986). "Graph Minors. II. Algorithmic Aspects of Tree-Width". In: *J. Algorithms* 7.3, pp. 309–322. DOI: 10.1016/0196-6774(86)90023-4.

Donald J. Rose (1974). "On simple characterizations of k-trees". In: *Discrete Mathematics* 7.3-4, pp. 317–322.

Marc Roth (2021). "Parameterized Counting of Partially Injective Homomorphisms". In: *Algorithmica* 83.6, pp. 1829–1860. DOI: 10.1007/s00453-021-00805-y.

V. Roth, J. Laub, M. Kawanabe, and J.M. Buhmann (2003). "Optimal cluster preserving embedding of nonmetric proximity data". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.12, pp. 1540–1551. DOI: 10.1109/TPAMI.2003.1251147.

I. J. Schoenberg (1938). "Metric Spaces and Positive Definite Functions". In: *Transactions of the American Mathematical Society* 44.3, pp. 522–536. DOI: 10.2307/1989894.

Bernhard Schölkopf and Alexander Johannes Smola (2002). *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. MIT Press.

Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel (2018). "Mining Tree Patterns with Partially Injective Homomorphisms". In: *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pp. 585–601. DOI: 10.1007/978-3-030-10928-8_35.

— (2022). "A generalized Weisfeiler-Lehman graph kernel". In: *Machine Learning* 111.7, pp. 2601–2629. DOI: 10.1007/s10994-022-06131-w.

Till Hendrik Schulz, Pascal Welke, and Stefan Wrobel (2022). "Graph Filtration Kernels". In: *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 8196–8203. DOI: 10.1609/aaai.v36i8.20793.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). "Weisfeiler-Lehman Graph Kernels". In: *Jorunal of Machine Learning Research* 12, pp. 2539–2561. DOI: 10.5555/1953048.2078187.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt (2009). "Efficient graphlet kernels for large graph comparison". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 488–495.

Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis (2020). "GraKeL: A Graph Kernel Library in Python". In: *Journal of Machine Learning Research* 21.54, pp. 1–5.

Jeffrey J. Sutherland, Lee A. O'Brien, and Donald F. Weaver (2003). "Spline-Fitting with a Genetic Algorithm: A Method for Developing Classification Structure-Activity Relationships". In: *J. Chem. Inf. Comput. Sci.* 43.6, pp. 1906–1915. DOI: 10.1021/ci034143r.

Matteo Togninalli, M. Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten M. Borgwardt (2019). "Wasserstein Weisfeiler-Lehman Graph Kernels". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6436–6446.

S. V. N. Vishwanathan and Alexander J. Smola (2002). "Fast Kernels for String and Tree Matching". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 569–576.

S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt (2010). "Graph Kernels". In: *Journal of Machine Learning Research* 11.40, pp. 1201–1242.

Fabian Wagner (2011). "Graphs of Bounded Treewidth Can Be Canonized in AC^1". In: *Computer Science - Theory and Applications. Proceedings*, pp. 209–222. DOI: 10.1007/978-3-642-20712-9_16.

Nikil Wale and George Karypis (2006). "Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification". In: *International Conference on Data Mining (ICDM), Proceedings*, pp. 678–689. DOI: 10.1109/ICDM.2006.39.

Yuchung J. Wang and George Y. Wong (1987). "Stochastic Blockmodels for Directed Graphs". In: *Journal of the American Statistical Association* 82.397, pp. 8–19.

Boris Weisfeiler and A. A. Lehman (1968). "A reduction of a graph to a canonical form and an algebra arising during this reduction". In: *Nauchno-Technicheskaya Informatsiya* 2.1, pp. 12–16.

Pascal Welke, Tamás Horváth, and Stefan Wrobel (2017). "Probabilistic Frequent Subtrees for Efficient Graph Classification and Retrieval". In: *Machine Learning*. DOI: 10.1007/s10994-017-5688-7.

Gang Wu, Edward Y. Chang, and Zhihua Zhang (2005). "An analysis of transformation on non-positive semidefinite similarity matrix for kernel machines". In: *International Conference on Machine Learning (ICML), Proceedings*.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations (ICLR)*.

Xifeng Yan and Jiawei Han (2002). "gSpan: Graph-Based Substructure Pattern Mining". In: *International Conference on Data Mining (ICDM)*, pp. 721–724. DOI: 10.1109/ICDM.2002.1184038.

Pinar Yanardag and S. V. N. Vishwanathan (2015). "Deep Graph Kernels". In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1365–1374. DOI: 10.1145/2783258.2783417.

Kaizhong Zhang, Richard Statman, and Dennis E. Shasha (1992). "On the Editing Distance Between Unordered Labeled Trees". In: *Inf. Process. Lett.* 42.3, pp. 133–139. DOI: 10.1016/0020-0190(92)90136-J.

Muhan Zhang and Pan Li (2021). "Nested Graph Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 15734–15747.