Scalable Graph Kernels

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Nino Shervashidze, M.Sc.

aus Tbilisi

Tübingen

2012

Tag der mündlichen Qualifikation:	18.06.2012
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Karsten Borgwardt
2. Berichterstatter:	Prof. Dr. Bernhard Schölkopf

Zusammenfassung

Daten in Form von Graphen spielen in immer mehr Bereichen der Wissenschaft und der Wirtschaft eine Rolle, wie in der Analyse sozialer Netzwerke, in der Molekularbiologie, in der Chemie, in der Computervision und in der Programmverifikation. Um diese Daten zu nutzen, braucht man Methoden zur Datenanalyse und zum Maschinenlernen, die in der Lage sind, große Datenmengen effizient zu verarbeiten. Um die Algorithmen des Maschinenlernens erfolgreich auf Graphen anzuwenden, benötigt man Verfahren, um Graphen effizient vergleichen oder repräsentieren zu können. Standardlösungen für diese Probleme sind entweder NP-schwer, nicht expressiv genug, oder schwierig auf das jeweilige Problem anzuwenden.

Graphkerne haben im letzten Jahrzehnt im Bereich Maschinenlernen viel Aufmerksamkeit auf sich gezogen, da sie einen vielversprechenden Lösungsansatz für die obengenannten Probleme darstellen. Trotz signifikanter Fortschritte im Bereich der Graphkerne in den letzten Jahren reichen bekannte Graphkerne nicht für die gegenwärtigen Anforderungen im Maschinenlernen aus, wenn die zu untersuchenden Graphen sehr groß oder gelabelt sind. Selbst die effizientesten Kerne erfordern eine Laufzeit von $O(n^3)$, um ein Paar Graphen mit n Knoten zu vergleichen oder um Knoten- und Kantenlabels zu berücksichtigen. Unser wichtigstes Ziel in dieser Dissertation ist daher die Entwicklung effizienter und expressiver Kerne für das Maschinenlernen auf Graphen.

Zuerst konzentrieren wir uns auf die Entwicklung allgemeiner Graphkerne, die auf Graphen mit oder ohne Labels angewendet werden können. Unsere Hauptbeiträge sind dabei die Folgenden:

Erstens beschleunigen wir die exakte Berechnung von Graphlet-Kernen von $O(n^k)$ bis $O(nd^{k-1})$ für ein Paar Graphen, wobei n die Größe des Graphen ist, k die Größe der angewandten Graphlets, und d der maximale Grad der gegebenen Graphen.

Zweitens definieren wir einen neuen Kern für Graphen, den Weisfeiler-Lehman-Unterbaumkern, der der erste Graphkern ist, der linear in der Anzahl der Kanten in dem gegebenen Graphensatz skaliert. In unseren Experimenten auf Vergleichsdatensätzen aus der Chemoinformatik und der Bioinformatik skaliert der Weisfeiler-Lehman-Unterbaumkern bis zu großen Graphen und übertrifft alle bekannten Graphkerne an Geschwindigkeit mit vergleichbarer oder bessere Vorhersagegenauigkeit bei der Graphenklassifizierung.

Drittens verallgemeinen wir den Weisfeiler-Lehman-Unterbaumkern zu einer Kernfamilie, die viele bekannte Graphkerne als Spezialfälle beinhaltet. Diese Verallgemeinerung ermöglicht es bekannten Graphkerne, mehr Informationen über die Topologie der Graphen zu berücksichtigen.

Im letzten Teil dieser Dissertation präsentieren wir zwei Anwendungsbeispiele: Basierend auf den vorherigen Beiträgen schlagen wir spezialisierte Kerne für Pixelklassifizierung in Fernerkundungsbildern und Graphkerne für die Vorhersage von chemischen Verschiebungen in der strukturellen Bioinformatik vor. Unsere Kerne ermöglichen es erstmals, in diesen Anwendungen die reichhaltige Graphenstruktur beim Lernen zu nutzen.

Die Weisfeiler-Lehman-Kerne, die wir hier vorschlagen, ermöglichen es Graphkerne, auf große und gelabelte Graphen zu skalieren. Sie erlauben die Nutzung von Graphkernen in zahlreichen Anwendungsgebieten, die sich mit Graphen beschäftigen, dessen Größe und Labels mit bekannten Graphkernen vorher nicht verarbeitet werden konnten.

Abstract

Graph-structured data are becoming more and more abundant in many fields of science and engineering, such as social network analysis, molecular biology, chemistry, computer vision, or program verification. To exploit these data, one needs data analysis and machine learning methods that are able to efficiently handle large-scale graph data sets. Successfully applying machine learning and data analysis methods to graphs requires the ability to efficiently compare and represent graphs. Standard solutions to these problems are either NP-hard, not expressive enough, or difficult to adapt to a problem at hand.

Graph kernels have attracted considerable interest in the machine learning community in the last decade as a promising solution to the above-mentioned issues. Despite significant progress in the design and improvement of graph kernels in the past few years, existing graph kernels do not measure up to the current needs of machine learning on large, labeled graphs: Even the most efficient existing kernels need $O(n^3)$ runtime to compare a pair of graphs with n nodes, or cannot take into account node and edge labels. Our primary goal in this thesis is the design of efficient and expressive kernels for machine learning on graphs.

We first focus on the design of generic graph kernels that can be applied to graphs with or without labels. Our main contributions to this end are the following:

First, we speed up the exact computation of graphlet kernels from $O(n^k)$ to $O(nd^{k-1})$ for a pair of graphs, where n is the size of the graphs, k is the size of considered graphlets, and d is the maximum degree in the given graphs.

Second, we define a new kernel on graphs, the Weisfeiler-Lehman subtree kernel, which is the first graph kernel scaling linearly in the number of edges in the given graph set. In our experiments on benchmark graph data sets from chemoinformatics and bioinformatics, the Weisfeiler-Lehman subtree kernel gracefully scales up to large graphs, outperforms all existing graph kernels in speed, and yields highly competitive performance in graph classification.

Third, we generalize the Weisfeiler-Lehman subtree kernel to a family of kernels that includes many known graph kernels as special cases. This generalization enables existing graph kernels to take into account more information about the graph topology, and thereby become more expressive.

In the last part of this thesis, we present two examples of applications: Based on our previous contributions, we propose specialized node kernels for pixel classification in remote sensing images, and graph kernels for chemical shift prediction in structural bioinformatics. Our kernels make it possible for the first time to take advantage of the

rich graph structure in these applications.

The Weisfeiler-Lehman kernels we propose here now allow graph kernels to scale to large, labeled graphs. They open the door to manifold applications of graph kernels in numerous domains which deal with graphs whose size and attributes could not be handled by graph kernels before.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor, Prof. Karsten Borgwardt, for his guidance and his invaluable, continuous support, encouragements, and availability during my PhD. I also thank him for taking time to give me insightful comments during the writing of this thesis.

This thesis would not have come into being without Prof. Bernhard Schölkopf accepting me as a PhD student in his department. Being part of his group has been a great privilege: I could hardly imagine a more stimulating, open, and inspiring scientific environment for doing a PhD in machine learning. I would also like to thank Prof. Schölkopf for his interest in my work, and for his support.

I am grateful to Prof. Detlef Weigel for also accepting me in the Molecular Biology department of the MPI for Developmental Biology, and for his advice as a member of my PhD advisory committee with K. Borgwardt and B. Schölkopf. Being a student at this institute has been a wonderful opportunity for me to learn about cutting-edge research in many fields of biology and to develop the ability to present my work to researchers in a completely different field.

I thank Prof. Oliver Kohlbacher and Prof. Daniel Huson for agreeing to be part of my PhD defense committee.

I would like to thank my collaborators for many fruitful discussions. I especially thank Gustavo Camps-Valls, Michael Habeck, Risi Kondor, Kurt Mehlhorn, Pascal Schweitzer, and Alex Smola. I am grateful to Alex Smola for offering me a very challenging and rewarding internship at Yahoo! Research. I thank Michael Habeck who proofread a part of this thesis.

I thank my fellow students and postdocs, whether it be for discussions and feedback, help in everyday matters, good time spent together, or all of this combined. Among them: Chloé Azencott, Dominik Grimm, Theofanis Karaletsos, Karin Klotzbücher, Christoph Lippert, Jonas Peters, Barbara Rakitsch, Oliver Stegle, and many others. I also thank present and former members of Bernhard Schölkopf's group, including Dominik Janzing, Ulrike von Luxburg, Suvrit Sra and Koji Tsuda, for many insteresting discussions and constructive feedback.

During my PhD studies, I have benefited from the help of many administrative and technical staff members. I especially thank Sabrina Rehbaum for all her work to arrange my administrative complications, and Sebastian Stark for his excellent technical support.

I acknowledge the Max Planck Society (Max-Planck-Gesellschaft) and the German Research Foundation (Deutsche Forschungsgemeinschaft) who funded my doctoral work.

I thank Nicolas Stefanovitch and Philipp Drewe who proofread large parts of this thesis and gave very valuable feedback, and Judith Pfeiffer and Barbara Rakitsch who helped me with the German abstract.

I would also like to express my gratitude to many individuals who taught or advised me before the start of my PhD. I have been extremely lucky to have many inspiring and supportive teachers, including Jean-Michel Daube, Nana Kemashvili, Merab Odzelashvili, Monique Slodzian, and Harald Wertz. I would like to especially thank Konstantin Tsiskaridze, who showed me the beauty of mathematics. Many thanks to Massih-Reza Amini for patiently answering my numerous questions, supporting and encouraging me during my Master's.

I am grateful to my partners in music in Tübingen, Jakob Zscheischler and Christiane Nüsslein-Volhard, as well as to Sieglinde and Cristobal Curio and Sabine Grubmüller who in various ways supported and helped me maintain my musical activity during my PhD studies.

I cordially thank my friends, in Tbilisi, Paris, Tübingen or elsewhere in the world, and my family for their support over the last years. I want to especially thank Philipp, Cristina, Elisabeth, Ruthi, Benedict, Yu-Hua and Judith for making my time in Tübingen particularly enjoyable. Special thanks to Nicolas who has been with me at every moment despite the many kilometers that have often separated us.

Finally, I would like to thank my parents, Guranda Datashvili and Tengiz Shervashidze, who gave me the best examples of thirst for knowledge, passion for one's work, honesty, and much more.

Contents

1. Introduction

1.1. Motivation

With the ever-accelerating development of technology, it is now possible to perform more experiments and record more results and observations than it has been possible ever before. Tremendous amounts of data are being collected by scientists, engineers, companies in the hope of using them to better understand the world, to design better technology, or to maximize customer satisfaction. To achieve these goals, one needs to be able to infer knowledge from large amounts of data. The design and theory of algorithms inferring general rules based on observed data is the primary subject of machine learning.

The nature of the data is arbitrary and domain-dependent: While certain types of observations are naturally represented as numbers, vectors in an Euclidean space or matrices, others may need more sophisticated representation depending on the inference task at hand. For example, if we wish to infer a rule that determines the topic of a news item based on the frequencies of individual words in it, the word frequencies in the news item can be easily represented as a vector of the same dimensionality as the dictionary. However, if the task is to infer a rule that determines the activity of a molecule given its structure, we have to take into consideration the molecular structure, whose natural representation is neither a scalar, nor an array of scalars. Molecular structures, and many other types of data are naturally modeled with the help of graphs.

Graphs are a general, powerful and flexible means of representing objects and concepts. A graph represents a set of relations, called edges, between elements of another set, called nodes. Typically, nodes in a graph represent entities, and edges express relations between the entities. For instance, a social network can be represented as a graph by letting the set of nodes be the set of individuals in the network, and the set of edges be the set of pairs of individuals that know each other; A molecule can be modeled as a graph by taking the set of its atoms as nodes, and the set of pairs of atoms connected with bonds as edges. The information contained in a graph with respect to the object of interest can be of two sorts. On the one hand, the object may be represented as a graph of its parts or properties linked to each other: This is the case, for instance, if the object of interest is a molecule, and it is represented as a graph of its atoms. On the other hand, the object of interest may be represented as a node in the graph, thereby being characterized by its relations to other objects: This would be the case if we were interested in individual atoms in the graph describing a molecule. Hence, graphs offer the flexibility of representing an object both as a system described by its

internal structure, and as a component of the environment surrounding it.

Applications involving graph representations are ubiquitous and their number has largely increased in the last decades. We list below examples of such fields of application.

Chemoinformatics Chemistry is one of the traditional application domains of graph representations [Gasteiger and Engel, 2003]. Finding chemical compounds with a specific property or activity is a common problem in chemistry and pharmacology. Experimental screening of molecules with unknown activity is often expensive, tedious and time-consuming. Therefore it is desirable to be able to use computational methods to find a small set of potentially interesting candidate molecules for a given property or activity, which will later be tested experimentally. To this end, chemical compounds are modeled as graphs by representing atoms as vertices and bonds as edges. In this computational analysis, often referred to as QSAR or QSPR (quantitative structure-activity or structure-property relationship) analysis, a common assumption is that molecules with similar structure have similar functional properties. The problem of measuring the similarity of graphs is therefore a central problem in chemoinformatics.

Bioinformatics Bioinformatics is a more recent, but at least as important source of graph-structured data [Junker and Schreiber, 2008]. Advances in technology in the last decades have made it less expensive, difficult and time-consuming to sequence genomes, measure gene expression levels or detect biomolecular interactions. These measurements of various kinds give rise to various types of graph data, such as protein and gene co-expression networks, protein-protein or protein-DNA interaction networks, regulatory networks, metabolic networks, or protein structures. Numerous interesting questions can be asked about these graphs, such as in which way protein structure determines its function, how regulatory networks influence a phenotype, or whether we can predict if two proteins interact based on their structure. Nowadays, biological graphs are often incomplete, can have complex or high-dimensional node labels, and tend to have a low signal-to-noise ratio. For these reasons, machine learning for biological data constitutes a challenging and broad application area for graph comparison and representation methods.

Computer vision and biomedical imaging Research in image processing, analysis and understanding, or computer vision, has been very active in the past decades. Graphs are often used for representing images in this field. These graphs can be of different nature, ranging from grids of pixels [e.g., Boykov and Jolly, 2001] to adjacency graphs of regions [Rosenfeld, 1974], colors [Matas et al., 1995], or segmented parts [Harchaoui and Bach, 2007] in an image. Graphs are also used increasingly often in biomedical image analysis, where network structures, extracted from images as high-level models of organs, can potentially be used in diagnostics or health monitoring [Feragen et al.,

2011].

Program verification and testing Recently, there has been a surge of interest in using machine learning to detect bugs or malicious behavior in software [Jiang and Su, 2007, Baskiotis and Sebag, 2008]. A large part of this research is based on the analysis of program call graphs and control flow graphs.

Social networks Social network analysis [Wasserman and Faust, 1994, Carrington et al., 2005] is another important application domain of machine learning on graphs. Last years have seen a large increase in the popularity of social networking websites: Each of these social networks is in fact a large graph. In the simplest model of a social network, vertices represent users and links indicate a relationship between users. However, one can also include products, news items or other entities as different types of vertices. Common questions on social networks are, for instance, whether one can infer interests or tastes of a person based on the information about his or her circle of friends and acquaintances, whether two people who do not yet know each other are likely to form bonds, or whether social networks in general have a common, quantifiable structure.

Web data analysis The Web as a whole is a major source of large graphs. The Internet itself is a large network of computers. Moreover, with more and more people using electronic means of communication, the number and size of email exchange graphs, social networks, and instant communication networks has been constantly growing in the last decades. As advertisement constitutes the principal source of income for internet companies, recent years have witnessed a large increase in industrial research aimed at understanding how to advertise to maximize individual users' attention. A standard approach is to identify advertisements that are popular with certain users and present them to users with a similar profile. This similarity is often defined based on the connections of the users in a communication network. Furthermore, research in targeted advertisement often relies on the analysis of the graph of user search queries and sponsored advertisements, so-called *query-ad click graphs* [Anastasakos et al., 2009]. Another type of click graphs, between queries and search results, arise in web search [Craswell and Szummer, 2007].

While graphs offer a great expressivity of representation, this expressivity comes at a price: Many operations that are trivial for simpler data structures such as arrays or matrices, such as comparing two structures to each other, become difficult for graphs. Even for the seemingly simple problem of deciding whether two graphs equal each other there is no polynomial-time algorithm known [Garey and Johnson, 1979, Chapter 7].

In machine learning, many of the most powerful techniques apply to points in Euclidean spaces or require the ability to assess the similarity of points. For this reason, the bulk of the research on graphs and machine learning tries to faithfully represent graphs in an Euclidean or more general vector spaces. The degree of this faithfulness may depend on the needs of a particular application. However, the design of general representation and comparison methods for graphs that are flexible enough to be tailored to particular applications is still essential.

1.2. Graph theory basics and notation

In this section, we summarize key concepts from graph theory and establish our notation. Clarifying these notions and notation is necessary to follow the remainder of this thesis. Most of our graph-theoretic terminology closely follows that of the monograph by Diestel [2010].

1.2.1. Graphs, subgraphs, isomorphism

A *graph* is a pair $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is an ordered set of n *vertices* (or *nodes*), and $E \subseteq V \times V$ is the set of *edges*. The size of a graph is defined as the size of the set V, here n.

A graph $G = (V, E)$ is said to be *undirected* if $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$ for all $v_i, v_j \in V$; otherwise it is *directed*. Although many of our techniques are applicable to both directed and undirected graphs, for ease of exposition, we will exclusively deal with undirected graphs in this thesis.

Any edge (v_i, v_i) is called a *loop*. In a general graph two vertices v_i and v_j may be connected by more than one edge. A *simple graph* is a graph with no loops or multiple edges. In this thesis we focus on simple graphs.

A simple graph can equivalently be represented by an *adjacency matrix* A of size $n \times n$. The (i, j)-th entry of A is 1 if an edge (v_i, v_j) exists and zero otherwise. The adjacency matrix of an undirected graph is symmetric.

A graph can have labels on nodes and/or edges. In this thesis, we denote labeled graphs with a triplet (V, E, ℓ), where $\ell : X \mapsto \Sigma$ is a function assigning a label from an alphabet Σ to each element of the set X, which can equal V, E, or $V \cup E$ depending on whether only nodes, only edges, or both are labeled.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* (denoted by $G \simeq G'$) if there exists a bijective mapping $f : V \to V'$ (called an *isomorphism*) such that $(v_i, v_j) \in E$ if and only if $(f(v_i), f(v_j)) \in E'$. If $G = G'$, f is called an *automorphism*. For labeled graphs $G = (V, E, \ell)$ and $G' = (V', E', \ell')$, the isomorphism (respectively, automorphism) function also has to satisfy $\ell(v_i) = \ell'(f(v_i))$ for each $v_i \in V$.

A function that takes a graph as an argument is called a *graph invariant* if it assigns equal values to isomorphic graphs. The number of vertices and the number of edges, for

instance, are graph invariants.

Given graphs $G = (V, E)$ and $G' = (V', E')$, we say that G' is a *subgraph* of G, denoted by $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. If $G' \subseteq G$ and, in addition, E' contains all edges $(u, v) \in E$ such that $u, v \in V'$, then we say that G' is an *induced subgraph* of G and we denote this $G' \sqsubseteq G$.

1.2.2. Neighborhood and degree

Two vertices v_i, v_j of G are *adjacent*, or *neighbors*, if $(v_i, v_j) \in E$. The *neighborhood* $\mathcal{N}(v)$ of a node v is the set of nodes to which v is adjacent, that is $\{v_i | (v, v_i) \in E\}$. All edges of the form $(v, v_i) \in E$ are said to be *incident* with the vertex v.

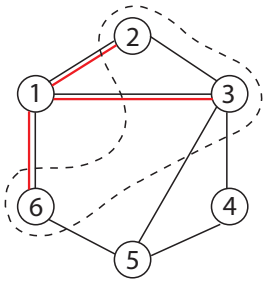

Figure 1.1.: The dashed line encompasses the neighbors of node 1. The degree of node 1 is three.

The *degree* $d(v)$ of a node v is the number of edges incident with v, which for a simple graph amounts to the cardinality of $\mathcal{N}(v)$.

1.2.3. Paths, walks, cycles, subtrees, subtree patterns

A *walk* is a sequence of nodes in a graph, in which consecutive nodes are connected by an edge. A *path* is a walk that consists of distinct nodes only, and a *cycle* is a closed path (see Figure 1.2 for illustration).

A graph G is called *connected* if any two of its nodes are linked by a path in G. A connected subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is called a *connected component* of G if no edge exists between any pair of nodes $v' \in V'$ and $v \in V \setminus V'$. The *distance* between two nodes v and v' in G is the length of the shortest path from v to v' in G, or ∞ if such a path does not exist.

A *(rooted) subtree* is a subgraph of a graph, which has no cycles, but a designated root node. A subtree of G can thus be seen as a connected subset of distinct nodes of G with an underlying tree structure. The height of a subtree is the maximum distance

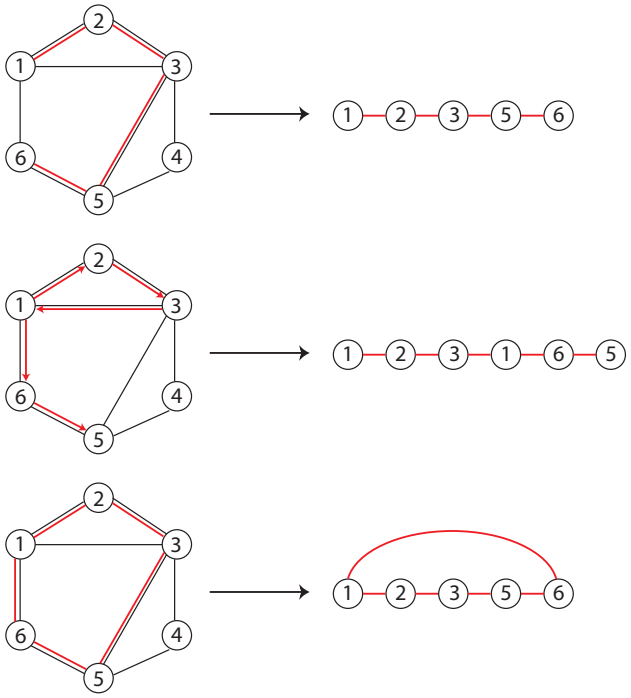

Figure 1.2.: A path, a walk and a cycle.

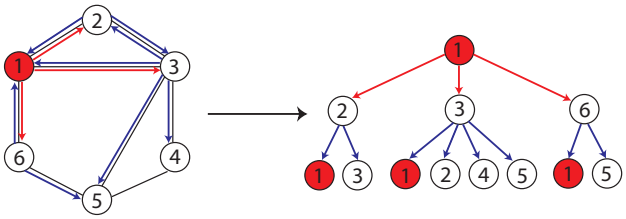

Figure 1.3.: A subtree pattern of height 2 rooted at the node 1. Note the repetitions of nodes in the unfolded subtree pattern on the right.

between the root and any other node in the subtree. Just as the notion of walk extends the notion of path by allowing nodes to be equal, the notion of subtrees can be extended to *subtree patterns* [also called *tree-walks*, Bach, 2008], which can have nodes that are equal (see Figure 1.3). These repetitions of the same node are then treated as distinct nodes, such that the pattern is still a cycle-free tree.

1.3. Review of graph comparison methods

The central problems that we deal with in this thesis are comparison and representation of graphs. Let us first formally define what we exactly mean by graph comparison.

Definition 1.3.1 (Graph comparison problem) *Given the set $\mathcal{G}$ of all possible graphs, the problem of graph comparison is defined as finding a function*

$$k : \mathcal{G} \times \mathcal{G} \mapsto \mathbb{R},$$

such that $k(G, G')$ for $G, G' \in \mathcal{G}$ quantifies the similarity of G and G'.

There exists extensive literature in computer science on the problem of graph comparison. This section reviews classical approaches to this problem.

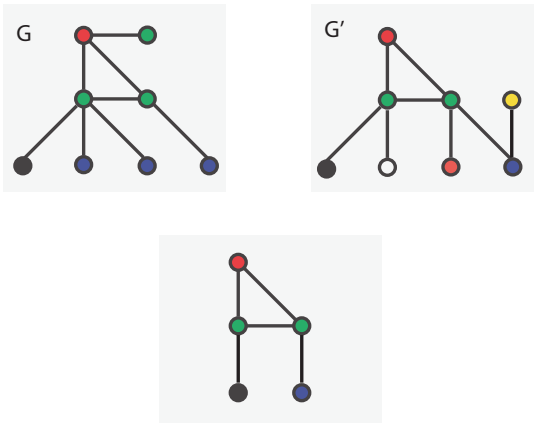

Figure 1.4.: The maximum common subgraph of graphs G and G'.

1.3.1. Isomorphism-based approaches

There exist many graph similarity measures based on graph isomorphism or related concepts such as subgraph isomorphism or the largest common subgraph. Possibly the most natural measure of similarity of graphs is to check whether the graphs are topologically identical, i.e., isomorphic. This gives rise to a binary similarity measure, which equals 1 if the graphs are isomorphic, and 0 otherwise. Despite the idea of checking graph isomorphism being so intuitive, no efficient algorithms are known for it. The graph isomorphism problem is in NP, but has been neither proven NP-complete nor found to be solved by a polynomial-time algorithm [Garey and Johnson, 1979, Chapter 7].

Subgraph isomorphism checking is the analogue of graph isomorphism checking in a setting in which the two graphs have different sizes. Unlike the graph isomorphism problem, the problem of subgraph isomorphism has been proven to be NP-complete [Garey and Johnson, 1979, Section 3.2.1].

A slightly less restrictive measure of similarity can be defined based on the size of the largest common subgraph in two graphs (see Figure 1.4 for illustration). However, unfortunately the problem of finding the largest common subgraph of two graphs is NP-hard [Garey and Johnson, 1979, Section 3.3].

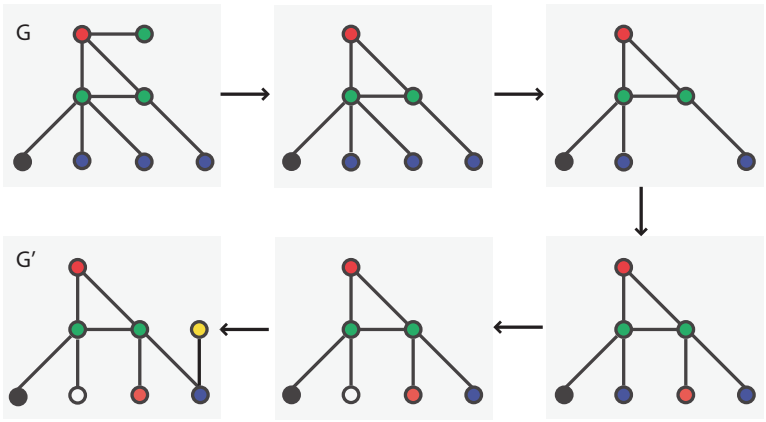

Figure 1.5.: A sequence of node deletion, node addition and node relabeling operations transforming the graph G into G'. Supposing that the cost of each operation is 1, this sequence of operations is minimal and the edit distance between G and G' equals 5.

1.3.2. Graph edit distances

Besides being computationally expensive or even intractable, similarity measures based on graph isomorphism and its variants are too restrictive in the sense that graphs have to be exactly identical or contain large identical subgraphs in order to be deemed similar by these measures. More flexible similarity measures, based on inexact matching of graphs, have been proposed in the literature. Graph comparison methods based on graph edit distances [Sanfeliu and Fu, 1983, Bunke and Allermann, 1983, Messmer and Bunke, 1998, Neuhaus and Bunke, 2005, and references therein] are a prominent example of this class. According to these methods, two graphs G and G' are similar if one can be transformed into the other using a small number of simple operations such as node

addition or deletion, edge addition or deletion, and node or edge relabeling in the case of labeled graphs (see Figure 1.5). These operations can have different costs. The edit distance between G and G' is defined as the minimum cost of transforming G into G'. Unfortunately, finding optimal costs for a particular application is a hard problem.

In summary, graph edit distances are expressive similarity measures respecting the topology, as well as node and edge labels of graphs. Unfortunately however, they are hard to parameterize and involve solving NP-complete problems as intermediate steps.

1.4. Review of graph representation methods

Analogously to Section 1.3, we start by defining our notion of graph representation.

Definition 1.4.1 (Graph representation problem) *Given the set $\mathcal{G}$ of all possible graphs, the problem of graph representation is defined as finding a function*

$$\phi : \mathcal{G} \mapsto \mathbb{R}^p,$$

$p \in \mathbb{N}$, such that $\phi(G)$ for any $G \in \mathcal{G}$ captures the structure of G.

Graph representation and graph comparison are related problems: Indeed, if it was possible to efficiently and faithfully summarize any graph structure as a feature vector, then comparing graphs would not be difficult either. This section presents three most prominent families of approaches to graph representation. The goal of these approaches is to efficiently compute a vector representation of the graph topology. All of these methods can be straightforwardly used to design similarity measures for graphs.

1.4.1. Topological descriptors

Interest in vector representations for graphs was first fuelled by research in chemoinformatics [Gasteiger and Engel, 2003]: In this field, as we mentioned in Section 1.1, a standard way to represent a molecule is by modeling it as a graph where nodes correspond to atoms and edges to bonds. A *topological descriptor* (also called topological index, connectivity index) is a number or an array of numbers, describing the topology of a molecular graph. These vectorial descriptions for graphs are usually graph invariants, and are used in a variety of chemoinformatics tasks as a proxy for molecular structures. These tasks include retrieval of query structures from a large data set of molecular graphs [Gasteiger and Engel, 2003, Chapter 6], and quantitative structure-activity and structure-property relationship (*QSAR* and *QSPR*) analyses, where one seeks to find correlations between the structure of a molecule and its chemical activity or property. Motivated by these central problems in chemoinformatics, numerous molecular descriptors have been developed since the 1950s [see the textbook by Todeschini and Consonni, 2000, for an extensive reference on this subject].

Commonly used topological descriptors include Wiener, Morgan and Zagreb indices [Wiener, 1947, Morgan, 1965, Gutman and Trinajstić, 1971], which we present below. The Wiener index [Wiener, 1947] corresponds to the sum of all shortest paths in a graph.

Definition 1.4.2 (Wiener index) *Given a graph $G = (V, E)$, the Wiener index (Wiener number) is defined as*

$$W(G) = \sum_{v_i \in V} \sum_{v_j \in V} D_{ij},$$

where D_{ij} is the shortest path from v_i to v_j.

The Morgan index [Morgan, 1965] is defined via the recursive procedure below.

Definition 1.4.3 (Morgan index) *Given a graph $G = (V, E)$, the Morgan index of order k for node v in G is defined as*

$$M_k(G, v) = \begin{cases} 1 & \text{if } k = 0, \\ \sum_{v' \in \mathcal{N}(v)} M_{k-1}(v') & \text{otherwise.} \end{cases} \tag{1.1}$$

It is not hard to notice that the Morgan index of order k for a node v is the number of walks of length k starting at v in G.

Definition 1.4.4 (Zagreb index) *Given a graph $G = (V, E)$, the first and second Zagreb indices are respectively defined as*

$$Z_1(G) = \sum_{v \in V} (d(v))^2,$$

and

$$Z_2(G) = \sum_{(v,v') \in E} d(v)d(v'),$$

where $d(v)$ is the degree of the node v.

Note that all of these topological descriptors are graph invariants: That is, if two graphs G and G' are isomorphic, then their respective topological descriptors will be identical. However, the converse is not true in general. In fact, designing a topological descriptor whose identity for two graphs would imply isomorphism would be equivalent to solving the graph isomorphism problem, for which there is no polynomial-time algorithm known. Moreover, we here encounter a similar obstacle to that of finding optimal costs for edit operations in graph edit distances: While individual descriptors may work well for a given application, it is difficult to decide in advance which ones to pick for another application. Given the great multitude of existing descriptors, this choice is made even harder.

1.4.2. Frequent pattern mining

Frequent pattern mining, or frequent subgraph mining, is a subfield of data mining, and is mainly concerned with the extraction of frequent subgraphs in a set of graphs.

Graph representation using frequent pattern mining can be done as follows: During the mining procedure, it is recorded for each selected subgraph g_i how many times it occurs in each graph G. At the end, G is represented as the vector

$$(c_1(G), \ldots, c_p(G)),$$

where $c_i(G)$ denotes the number of occurrences of g_i in G, and p is the number of mined substructures.

A large part of the frequent pattern mining algorithms is based on the so-called *Apriori* property [Agrawal and Srikant, 1994]. This notion stems from frequent item set mining literature, and states that if a set of items S is frequent in a collection of item sets, then any subset of S will also be frequent. This observation naturally extends to graphs: If a graph H occurs frequently in a set of graphs as a substructure, then any subgraph of H will occur at least as frequently as H. A prominent example of Apriori-based subgraph mining approaches is the AGM algorithm [Inokuchi et al., 2000].

Pattern-growth approaches, first introduced by Han et al. [2000], constitute the other important class of frequent subgraph mining methods. These algorithms start with smaller patterns and extend them until they are not frequent any more. Prominent examples of such approaches include gSpan [Yan and Han, 2002] and Gaston [Nijssen and Kok, 2004]. In gSpan, Yan and Han [2002] introduced a lexicographic order for labeled graphs based on depth-first search (DFS), which allows to map each graph to a unique canonical label called the *minimum DFS code*. The search space of gSpan is organized as a tree of DFS codes, where child nodes of a DFS code are obtained through a way of extending patterns, called *rightmost extension*. This allows gSpan to avoid examining the same substructures twice while searching for frequent subgraphs.

While all above-mentioned approaches serve as a pre-processing step for subsequent machine learning tasks, there exist methods that combine pattern mining and learning. Examples of such methods include graph classification algorithms gBoost [Saigo et al., 2009] and CORK [Thoma et al., 2009]. gBoost is a boosting method for graph data that uses linear programming and column generation techniques. CORK makes use of a submodular quality criterion that allows to greedily pick frequent subgraphs and still provide guarantees on the quality of the selected set of subgraphs. Note that the mining module of both algorithms is derived from gSpan [Yan and Han, 2002].

Frequent subgraph mining methods have been shown to be useful in learning tasks on graphs of limited size, such as small molecule classification. However, the major problem with these approaches is the exponential size of the substructure search space: Despite clever branch-and-bound and other pruning strategies used to accelerate the search, they

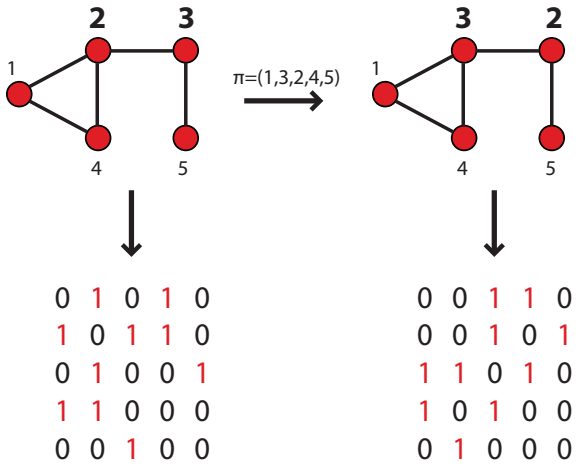

Figure 1.6.: If we exchange the node IDs 2 and 3 in the graph on the left, the corresponding adjacency matrix is reshuffled. The graph representation has to be invariant with respect to this reshuffling.

have to enumerate all subgraphs in a graph in the worst case, which is intractable for graphs of more than around 20 nodes.

1.4.3. Group-theoretic approaches

Group-theoretic methods offer a cardinally different way of looking at the problem of graph representation. Algebraic approaches to graph classification have first been proposed in the 1990s by Shawe-Taylor [1993], but have not received much attention until lately. Recent approaches from this class, the skew spectrum [Kondor and Borgwardt, 2008] and the graphlet spectrum [Kondor et al., 2009] can be used to define similarity measures on graphs that are computable in polynomial time.

Given a graph G represented by its adjacency matrix A, the goal of these approaches is to represent G in a Euclidean space, while retaining as much information as possible about the graph structure. Importantly, we have to design the features in such a way that they do not depend on the numbering of the vertices (see Figure 1.4.3 for illustration). In other words, the features have to be permutation-invariant. This is achieved by using results from non-commutative harmonic analysis and tools from representation theory of $\mathbb{S}_n$.

Before we start describing these approaches, let us first introduce some notation and terminology. Let G be a graph of n vertices, represented by its adjacency matrix, $A \in \mathbb{R}^{n \times n}$. $\mathbb{S}_n$ denotes the *symmetric group* of order n. Given a function $f : \mathbb{S}_n \mapsto \mathbb{R}$,

the group structure suggests defining the *left–translate* of f by $\pi \in \mathbb{S}_n$ as $f^\pi : \mathbb{S}_n \mapsto \mathbb{R}$, $\quad f^\pi(\sigma) = f(\pi^{-1}\sigma)$.

In terms of any complete set of inequivalent irreducible representations $\{\rho_\lambda\}\lambda \vdash n$ of $\mathbb{S}_n$, the *Fourier transform* of a function $f : \mathbb{S}_n \mapsto \mathbb{R}$ is defined as the sequence of matrices

$$\hat{f}(\lambda) = \sum_{\sigma \in \mathbb{S}_n} f(\sigma)\, \rho_\lambda(\sigma), \quad \lambda \vdash n.$$

Of the several properties of ordinary Fourier transformation inherited by such generalized Fourier transforms, we are particularly interested in the *translation theorem*, which, coupled with the unitarity of $\rho_\lambda(\pi)$, tells us that the matrices $\hat{a}(\lambda) = \hat{f}(\lambda)^\dagger \cdot \hat{f}(\lambda)$, $\lambda \vdash n$, are translation invariant.

Kondor and Borgwardt [2008] show that if we encode the adjacency matrix in the function

$$f_A(\sigma) = A_{\sigma(n),\sigma(n-1)},$$

then permuting the vertices of G by π transforms f_A exactly into $(f_A)^\pi$. In other words, renumbering the nodes in the graph causes the corresponding f_A to "translate to the left" by π.

A common alternative to the algebraic approaches to graph representation is to characterize graphs in terms of the frequency or position of certain elementary subgraphs embedded within them. Depending on the context these small subgraphs are usually called *graphlets* or *motifs*. In the graphlet spectrum, we [Kondor et al., 2009] extend the function f_A to take into account such labeled graphlets (motifs) instead of just unlabeled edges. Given a graphlet g of $k \ll n$ vertices whose adjacency matrix we denote with the same letter g and vertices $\{v_1, \ldots, v_k\}$ in G, the *indicator*

$$\mu_g(\{v_1, \ldots, v_k\}) = \begin{cases} 1 & \text{if } \; g_{i,j} \leq A_{v_i,v_j} \quad \forall\, i,j, \\ 0 & \text{otherwise,} \end{cases} \tag{1.2}$$

captures whether g is a subgraph of G at position $\{v_1, \ldots, v_k\}$. If we replace "$\leq$" by "$=$" in (1.2), then the corresponding indicator μ_g^{ind} captures whether g is an induced subgraph at the same position. The fundamental observation leading to the extension f_A from the skew spectrum of graphs to taking into account larger graphlets is that f_A can be re-written as

$$f_A(\sigma) = \mu_e(\sigma(n), \sigma(n-1)),$$

where e stands for the elementary graphlet of two vertices and a single directed edge. In other words, f_A encodes where the edge e occurs in G as a subgraph. It is straightforward to extend this idea to larger graphlets by setting

$$f_{A,g}(\sigma) = \mu_g(\sigma(n), \sigma(n-1), \ldots, \sigma(n-k+1)), \tag{1.3}$$

or $f_{A,g}(\sigma) = \mu_g^{\text{ind}}(\sigma(n), \sigma(n-1), \ldots, \sigma(n-k+1))$, depending whether we are interested in non-induced or induced subgraphs. Crucially, $f_{A,g}$ will still obey the same transformation property as f_A did: Since if μ_g^π is the indicator of the permuted adjacency matrix A^π, we have

$$\mu_g^\pi(\pi(v_1), \pi(v_2), \ldots, \pi(v_k)) = \mu_g(v_1, v_2, \ldots, v_k), \tag{1.4}$$

hence

$$\mu_g^\pi(v_1, \ldots, v_k) = \mu_g(\pi^{-1}(v_1), \ldots, \pi^{-1}(v_k)),$$

and therefore

$$f_{A^\pi,g}(\sigma) = \mu_g(\pi^{-1}\sigma(n), \ldots, \pi^{-1}\sigma(n-k+1)) = f_{A,g}(\pi^{-1}\sigma) = (f_{A,g})^\pi(\sigma). \tag{1.5}$$

Being able to express a function on a reshuffled adjacency matrix as a left-translate of the original function means that it is possible to invoke the machinery of power spectra, skew spectra, etc., to derive graph invariants. While the graph invariants of the skew spectrum of graphs [Kondor and Borgwardt, 2008] will be sensitive to the presence of individual unlabeled edges in a graph, we can take into account larger, labeled subgraphs in G in the case of the graphlet spectrum [Kondor et al., 2009]. On the other hand, the skew spectrum is a higher-order and more expressive invariant than the power spectrum used in the graphlet spectrum. An attractive feature of the graphlet spectrum approach is that given a small library $\{g_1, \ldots, g_m\}$ of graphlets, it is possible to compute a separate f_{A,g_i} function for each graphlet, and then form invariants from all possible combinations of these functions, capturing information about the relative position of different types of subgraphs as well as different subgraphs of the same type. Since in this case second-order invariants already yield a rich set of features, one can forgo computing higher order, more expensive invariants, such as the skew spectrum.

It has been shown empirically, that on graphs of up to a hundred vertices the skew and graphlet spectra for graphs have produced competitive results with the state of the art. However, for many applications, the computation cost scaling cubically in n is still too expensive. Moreover, the skew spectrum is restricted to unlabeled graphs, while the graphlet spectrum can be difficult to parameterize on general labeled graphs, as one may not know in advance which graphlets $\{g_1, \ldots, g_m\}$ will suit a particular application.

1.5. Review of graph kernels

As we have seen in Sections 1.3 and 1.4, traditional graph comparison and representation approaches either suffer from exponential runtime in the worst case, are not expressive enough, ignore node labels, or are hard to parameterize. Graph kernels, that have recently evolved into a rapidly developing branch of learning on structured data, try to address all these problems. These constructs, that can be viewed as a means of both representation and comparison, respect and exploit graph topology, but restrict

themselves to substructures of graphs that allow the kernel computation in polynomial time. Graph kernels bridge the gap between graph-structured data and a large spectrum of machine learning algorithms called kernel methods [Schölkopf and Smola, 2002], that include algorithms such as support vector machines, kernel regression, or kernel PCA.

Before we start to review graph kernels, we give a short introduction to kernels in machine learning. The reader is referred to the textbooks by Schölkopf and Smola [2002] and Shawe-Taylor and Cristianini [2004] for a complete and in-depth treatment of kernel methods.

1.5.1. Kernels in machine learning

Informally, a (positive semidefinite) kernel is a function of two objects that quantifies their similarity. Mathematically, it corresponds to an inner product[1]. In this section, we will first introduce kernels, and later give one example of their application in machine learning by putting them in the context of binary classification and support vector machines.

1.5.1.1. Positive semidefinite kernels

We borrow the definition of positive semidefinite kernels from Schölkopf et al. [2004]:

Definition 1.5.1 ((Positive semidefinite) kernel) *Let $\mathcal{X}$ be a nonempty set. A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called a positive semidefinite kernel if and only if it is symmetric, that is, $k(x, x') = k(x', x)$ for any $x, x' \in \mathcal{X}$, and positive semidefinite, that is,*

$$\sum_{i=1}^{N} \sum_{j=1}^{N} c_i c_j k(x_i, x_j) \geq 0$$

for any $N \in \mathbb{N}$, any choice of objects $x_1, \ldots, x_N \in \mathcal{X}$, and any choice of real numbers $c_1, \ldots, c_N \in \mathbb{R}$.

Note that in machine learning literature, positive semidefinite kernels are often called positive definite kernels or just kernels. In the remainder of this thesis, we will refer to them as *kernels*.

Why are kernels interesting? A short answer is that they are interesting because they can sometimes provide a way of efficiently computing inner products in high-dimensional spaces, which is essential for many machine learning algorithms (we will see an example in the next section). We explain this in more detail below.

[1] Note that there exist kernels that are not positive semidefinite and that can still be useful in some cases, for example as dissimilarity measures. They are, however, out of the scope of this thesis. Interested readers are invited to refer to [Schölkopf and Smola, 2002, Section 2.4] for details on this subject.

The key property of kernels that links them with inner products is the following [Mercer, 1909, Aronszajn, 1950, Aizerman et al., 1964]: For every positive semidefinite kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, there exists a reproducing kernel Hilbert space[2] (RKHS) $\mathcal{H}$ and a mapping $\phi : \mathcal{X} \mapsto \mathcal{H}$ such that for every $x, x' \in \mathcal{X}$, $k(x, x')$ corresponds to an inner product of $\phi(x)$ and $\phi(x')$:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle.$$

The converse also holds: Given a nonempty set $\mathcal{X}$, a reproducing kernel Hilbert space $\mathcal{H}$, and a mapping $\phi : \mathcal{X} \mapsto \mathcal{H}$, every function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ given by $k(x, x') = \langle \phi(x), \phi(x') \rangle$ is a positive semidefinite kernel. In machine learning, $\mathcal{H}$ is usually called a *feature space*, ϕ the corresponding *feature map*, and $\phi(x)$ the *feature representation* of x.

In fact, defining a feature map is a straightforward way towards defining a kernel. As a simple example, consider $\mathcal{X} = \mathbb{R}^p$ for some p, $\mathcal{H} = \mathcal{X}$, and ϕ equal to identity: The resulting

$$k(x, x') = \langle \phi(x), \phi(x') \rangle = \langle x, x' \rangle,$$

which is a simple inner product, is a valid kernel, called the *linear kernel*. As another simple example, let $\mathcal{X}$ be the set of all possible graphs, let $\mathcal{H}$ be $\mathbb{R}^2$ and let $\phi(G) = (|V|, |E|)$ for all $G = (V, E) \in \mathcal{X}$. The resulting function comparing graphs,

$$k(G, G') = \langle (|V|, |E|), (|V'|, |E'|) \rangle,$$

is not very elaborate or expressive, but it is a valid kernel.

In the previous paragraph we gave an intuition of one way of designing a kernel function:

1. Choose $\mathcal{H}$,
2. choose ϕ,
3. set $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

While a perfectly legitimate way of defining a kernel, this procedure does not bring anything new to the computation of $\langle \phi(x), \phi(x') \rangle$: To compute $k(x, x')$, one has to perform multiplications and a summation in $\mathcal{H}$, a procedure with the time complexity proportional to the dimensionality of $\mathcal{H}$. One important reason why kernels are popular in machine learning is that for some kernels it is possible to compute $k(x, x')$ without having to perform any operation in $\mathcal{H}$. This property can be extremely helpful, as it allows us to consider data representations in very high or even infinite-dimensional spaces $\mathcal{H}$. We illustrate this statement with two examples of prominent kernels, the *polynomial* and the *Gaussian radial basis function (RBF)* kernels on $\mathbb{R}^p \times \mathbb{R}^p$.

[2] For readers not familiar with the notion of RKHS: For the purposes of this section, an RKHS can be thought of as a generalization of an Euclidean space to any, possibly infinite, number of dimensions. For more detail please refer to Appendix A.

Polynomial kernel This kernel is given by

$$k(x, x') = (\langle x, x' \rangle + c)^d,$$

where $d \in \mathbb{N}$ and $c \geq 0$ are parameters. The dimensionality of the feature space corresponding to k equals the number of monomials of degree equal to or less than d in p variables, which is $\binom{d+p}{d}$. Consider, for example, $p = 100$ and $d = 4$: While computing $k(x, x')$ takes 202 operations in total (102 multiplications and 100 additions), performing this computation using feature representations $\phi(x)$ and $\phi(x')$ would cost $2\binom{104}{4} - 1 = 2 \times 4598126 - 1 = 9196251$ operations. Thus, the polynomial kernel allows us to compute an inner product in $\mathbb{R}^{4598126}$ in 202 operations instead of more than 9 million with direct computation. This is a tremendous gain in computation time, which becomes even more dramatic for larger values of p and d.

Gaussian RBF kernel This kernel has the form

$$k(x, x') = \exp\Big(- \frac{\|x - x'\|^2}{2\sigma^2} \Big), \tag{1.6}$$

where $\sigma > 0$. A particularity of this widely used kernel is that its corresponding RKHS is infinite-dimensional [Schölkopf and Smola, 2002, Section 2.3]. Representing data points in this space explicitly would be impossible, but thanks to kernels we can compute inner products in it.

How to design positive semidefinite kernels without first choosing an explicit $\mathcal{H}$ and ϕ? There is no simple answer to this question, however, the set of positive semidefinite kernels possesses several closure properties that can be exploited in kernel design. We list below some prominent examples of these properties:

- if k_1 and k_2 are kernels, and $\alpha_1, \alpha_2 \geq 0$, then $\alpha_1 k_1 + \alpha_2 k_2$ is a kernel;
- if $k_1, k_2, \ldots$ are kernels, and $k(x, x') = \lim_{n \to \infty} k_n(x, x')$ exists for all x, x', then k is a kernel;
- if k_1 and k_2 are kernels, then $k_1 k_2$, defined by $k_1 k_2(x, x') = k_1(x, x') k_2(x, x')$ is a kernel.

For a more extensive list of closure properties we invite the reader to consult the textbook by Schölkopf and Smola [2002, Section 13.1].

In addition to making it possible to compute inner products in very high or infinite-dimensional spaces, kernels allow us to work with nonvectorial data [Schölkopf, 1997]. In fact, if we define a symmetric similarity measure for arbitrary objects and prove that it is positive semidefinite, this automatically provides a vectorial representation of these objects in the feature space.

Let us precise the link between kernels and similarity measures: In the machine learning community in general, and in this thesis in particular, inner products are thought of as measures of similarity. However, this intuition is not always self-evident: In fact, the notion of similarity is more often associated with distance. If a certain notion of distance between two objects is small, then they are similar in the sense of this notion. Sometimes these two intuitions coincide. This is the case for the Gaussian RBF kernel from above (1.6): it is a decreasing function of Euclidean distance, and at the same time it is an inner product by virtue of being a kernel. More generally, it is not hard to show that for any (positive semidefinite) kernel k on any $\mathcal{X}$, we have:

$$k(x, x') = \frac{||\phi(x)||^2 + ||\phi(x')||^2 - ||\phi(x) - \phi(x')||^2}{2},$$

where $||z|| = \langle z, z \rangle$. As Schölkopf et al. [2004, Chapter 1] put it, $k(x, x')$ measures the similarity between x and x' as the opposite of the square distance between their images $\phi(x)$ and $\phi(x')$ in the feature space, up to the terms $||\phi(x)||^2$ and $||\phi(x')||^2$. If all points have the same length in the feature space ($||\phi(x)||^2$ is constant for all $x \in \mathcal{X}$), then the kernel is simply a decreasing measure of the distance in the feature space.

To summarize the key points of this section: Being inner products by nature, kernels can be thought of as measures of similarity. As every kernel has an associated feature map, they can also be regarded as means of representation. For some kernels, it is computationally intractable or impossible to make this representation in the feature space explicit. However, it is still possible to compute the inner product in the feature space thanks to the kernel function. Moreover, a key advantage of kernels is that they can be defined for any type of data.

1.5.1.2. Binary classification with support vector machines

Binary classification is a fundamental problem in machine learning. It can be informally described as follows: Given two classes of objects, infer a rule to assign a new, previously unseen object to one of the two classes. More formally, we are given a set of pairs

$$(x_1, y_1), \ldots, (x_N, y_N) \in \mathcal{X} \times \mathcal{Y},$$

where $\mathcal{X}$ denotes just a set of objects, that we also call *examples, patterns,* or *data points*, and $\mathcal{Y} = \{-1, +1\}$. $\mathcal{X}$ can be anything from a set of living organisms, a set of climatic conditions, or a set of protein structures to a set of real vectors of p components. For a given $x_i \in \mathcal{X}$, y_i indicates the class that x_i belongs to. The y_i are usually called *class labels*. The goal is to find (*learn*) a function $f : \mathcal{X} \mapsto \mathcal{Y}$ that can be used to predict the class labels of unseen data examples.

The *Support Vector Machine* (SVM) classification algorithm takes its origins from the 1960s [Vapnik and Lerner, 1963] and was formulated in its present form in the 1990s in

the works of Boser et al. [1992], Cortes and Vapnik [1995] and Vapnik [1995]. Support vector machines soon gained wide recognition due to their solid statistical foundations and the convex and well-behaved nature of their associated optimization problems. For a technical, thorough and up-to-date treatment of SVMs we refer the reader to the textbook by Steinwart and Christmann [2008].

The binary SVM classification problem is formulated in terms of input data of the form $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathcal{H} \times \{-1, +1\}$, where $\mathcal{H}$ is a reproducing kernel Hilbert space. The change with respect to the general binary classification problem above is that the data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are elements of an RKHS and not of an arbitrary set. In fact, without loss of generality, we can imagine every $\mathbf{x}_i$ as some vectorial description of the object x_i. This includes the case where $\mathcal{X}$ is itself a reproducing kernel Hilbert space, as we only require from $\mathcal{X}$ to be a nonempty set. Individual components of vectors $\mathbf{x}_i$ are referred to as *features*, or *variables*. The goal of the SVMs is to find a hyperplane that separates the two classes with the largest *margin*, that is, the largest minimum distance between a data point and the hyperplane (see Figure 1.7 for illustration). Mathematically, this corresponds to the following optimization problem:

$$\begin{aligned} \underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}}{\text{minimize}} \quad & \frac{1}{2}||\,\mathbf{w}\,||^2 \\ \text{subject to} \quad & y_i(\langle \mathbf{x}_i, w\rangle + b) \geq 1 \text{ for all } i = 1, \ldots, N. \end{aligned} \tag{1.7}$$

The problem (1.7) is often solved using its Lagrangian dual:

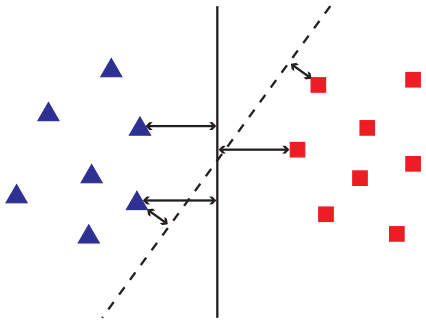

Figure 1.7.: The red squares and the blue triangles represent the two classes to be separated in $\mathbb{R}^2$, and the dashed and solid lines depict two candidate hyperplanes. The solid hyperplane has a much larger margin compared to the dashed one. The data points whose distance to a hyperplane equals the margin are called the *support vectors* of that hyperplane.

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to} \quad & \alpha_i \geq 0 \text{ for all } i = 1, \ldots, N, \\ \text{and} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{1.8}$$

If the positive and negative data points are separable by a hyperplane (as they are in Figure 1.7), or *linearly separable*, solving the Problem (1.7) yields the optimal hyperplane, similar to the solid line in Figure 1.7. However, input data are not always linearly separable. Cortes and Vapnik [1995] proposed a *soft-margin* version of SVMs that can successfully handle cases where positive and negative examples are "almost" linearly separable (see Figure 1.8 for illustration). The primal and dual for this variant, *C-SVM*, are given below.

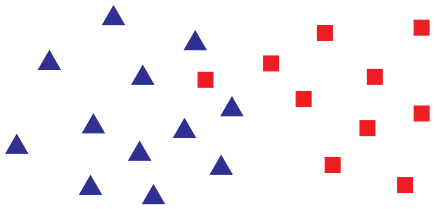

Figure 1.8.: "Almost" linearly separable classes

$$\begin{aligned} \underset{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}}{\text{minimize}} \quad & \frac{1}{2} || \mathbf{w} ||^2 + \frac{C}{N} \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{x}_i, w \rangle + b) \geq 1 - \xi_i \text{ for all } i = 1, \ldots, N, \\ \text{and} \quad & \xi_i \geq 0 \text{ for all } i = 1, \ldots, N. \end{aligned} \tag{1.9}$$

Here, ξ_i are the so-called *slack* variables, that measure how strongly the data point i violates the constraint of being correctly classified with the margin, $y_i(\langle \mathbf{x}_i, w \rangle + b) \geq 1$. The parameter C regulates the trade-off between maximizing the margin and minimizing

the number of points that are inside the margin or misclassified.

$$\begin{aligned} \underset{\boldsymbol{\alpha}\in\mathbb{R}^N}{\text{minimize}} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to} \quad & 0 \leq \alpha_i \leq \frac{C}{N} \text{ for all } i = 1, \ldots, N, \\ \text{and} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{1.10}$$

C-SVMs and a more recent soft-margin SVM called ν-*SVM* [Schölkopf et al., 2000] solve

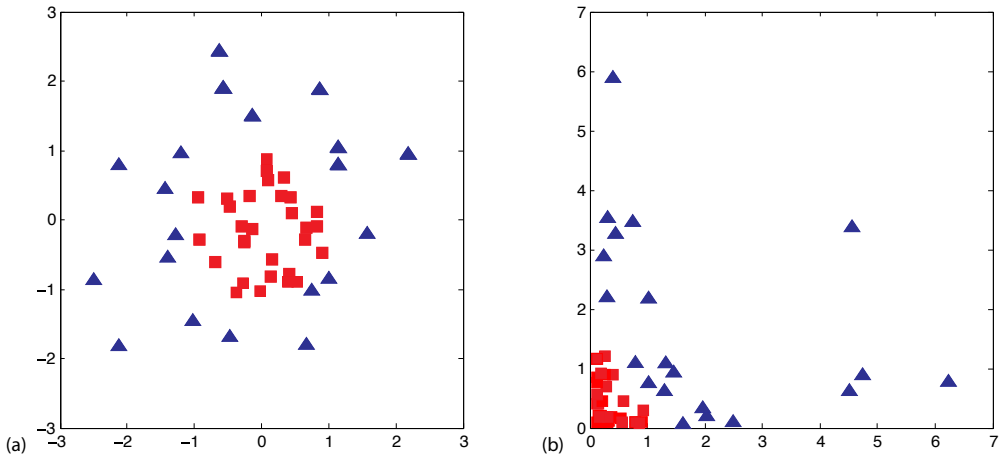

Figure 1.9.: The red squares and blue triangles are two different classes. On the left, in the original space $\mathcal{X} = \mathbb{R}^2$, the two classes are not linearly separable. On the right, the new feature representation $\phi(x) = (x_1^2, x_2^2)$, although in the same space $\mathcal{H} = \mathbb{R}^2$, makes them separable by a hyperplane.

the problem of finding an optimal separating hyperplane while allowing a small number of misclassified points. Nonetheless, there are cases where the data are inherently not separable by a hyperplane: No matter which hyperplane we pick, we will misclassify a large number of points (see Figure 1.9, (a), for illustration). In this type of cases, it is sometimes possible to redefine the representation of data points in such a way that they become linearly separable (Figure 1.9). To make a parallel with everyday life, changing the representation of data is comparable with looking at a set of objects from a different angle. As we see from Figure 1.9, it is possible to handle a highly non-linear classification problem with a linear classification algorithm, and still take advantage of its statistical

soundness and ability to be optimized efficiently. This is possible, provided that we can re-represent the data in a space where they become linearly separable. Intuitively, higher-dimensional RKHSs will be more likely to render input data linearly separable.

The key observation to make here is that the SVM dual only makes use of the input data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in the form of inner products (see Problems (1.8) and (1.10)). Therefore it does not matter in which space these inner products will be taken, provided that they are valid inner products. The RKHS $\mathcal{H}$ can be replaced by any other RKHS $\mathcal{H}'$. This is where positive semidefinite kernels come into play: Given a kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, we can replace every occurrence of $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ by $k(x_i, x_j)$. Kernels, combined with SVMs, make it possible to perform linear classification in spaces of very high or infinite dimension. Moreover, as kernels can be defined for any type of data, support vector classification can also be carried out for any type of data.

One question that arises is whether we can plug into an SVM a measure of similarity that is not an inner product (not symmetric, not positive semidefinite or neither). While such non-positive semidefinite kernels have already been used in machine learning (e.g., the so-called *sigmoid* kernels for points in $\mathbb{R}^p$), if we proceed in this way, the convexity of the SVM optimization problem is no longer guaranteed and the Problem (1.10) may not even have a solution [Burges, 1998].

SVMs, while an important class of methods using kernels, are only one member a large family of learning algorithms, called *kernel methods*, that are able to operate on input data in the form of inner products without requiring access to actual examples. Examples of kernel methods include support vector regression, Gaussian processes for regression, kernel principal component analysis and kernelized statistical independence testing [see Hofmann et al., 2008, for a recent review of kernel algorithms].

1.5.2. Graph kernel basics

Graph kernels are instances of the family of so-called *R-convolution kernels* by Haussler [1999]. R-convolution is a generic way of defining kernels on discrete compound objects by comparing all pairs of decompositions thereof. Therefore, a new type of decomposition of a graph results in a new graph kernel.

Given a decomposition relation R that decomposes a graph into any of its subgraphs and the remaining part of the graph, the associated R-convolution kernel will compare all subgraphs in two graphs. However, this *all subgraphs* kernel is at least as hard to compute as deciding if graphs are isomorphic [Gärtner et al., 2003]. Therefore one usually restricts graph kernels to compare only specific types of subgraphs that are computable in polynomial runtime.

Certain graph kernels are also generalized by *rational kernels* by Cortes et al. [2004], that compare weighted automata to each other. While they generalize some prominent graph kernels such as kernels based on random walks, they assume that the set of possible labels on nodes and edges is finite, and therefore do not subsume kernels that take into

account continuous labels [Vishwanathan et al., 2010].

Note that in machine learning the term *graph kernel* occasionally refers to a kernel between nodes of one large graph, which we call a *node kernel*. Throughout this thesis, *graph kernel* will denote a kernel that compares graphs to each other.

Several different graph kernels have been defined in machine learning which can be categorized into three classes: graph kernels based on walks [Kashima et al., 2003, Gärtner et al., 2003] and paths [Borgwardt and Kriegel, 2005], graph kernels based on limited-size subgraphs [Horváth et al., 2004], and graph kernels based on subtree patterns [Ramon and Gärtner, 2003, Mahé and Vert, 2009]. We review each of these classes below.

1.5.3. Kernels based on walks and paths

The first class, graph kernels on walks and paths, compute the number of matching pairs of random walks (respectively, paths) in two graphs. The standard formulation of the random walk kernel by Gärtner et al. [2003] is computable in $O(n^6)$ for a pair of graphs. However, the same problem can be stated in terms of Kronecker products that can be exploited to bring down the runtime complexity to $O(n^3)$ [Vishwanathan et al., 2010]. While this constitutes an important gain in efficiency that allows to compute the random walk kernel orders of magnitude faster, the complexity of $O(n^3)$ is still too high for many applications. Besides computation time, random walks suffer from two more problems: *tottering* [Mahé et al., 2004], and *halting* [Borgwardt, 2007].

The problem of tottering originates from the fact that walks, by definition, allow for repetitions of nodes and edges. Consequently, a single shared edge can potentially contribute infinitely many times to the kernel value between two graphs. The same can be said of shared cycles or paths. Therefore, the kernel value between two graphs that do not share many structural elements may be unduly inflated.

Halting is a different problem: Random walk kernels, as we have seen, count the number of *all* matching pairs of walks. However, the number of walks in a graph with at least one edge is infinite. Therefore one usually imposes a decaying factor λ on the length of walks, that down-weights longer walks. As a result, it is made possible to compare graphs with respect to all walks, but at the expense of the kernel value being almost exclusively determined by short walks. This problem is referred to as "halting".

Due to the above-mentioned problems and motivated by applications, there have been proposed numerous extensions and variants of random walk kernels: For a computer vision application, Harchaoui and Bach [2007] have proposed a dynamic programming-based approach to speed up the computation of the random walk kernel, but at the cost of considering walks of fixed size. Suard et al. [2005] and Vert et al. [2009] present other applications of random walk kernels in computer vision. Mahé et al. [2004] have proposed extensions of marginalized graph kernels [Kashima et al., 2003] for a chemoinformatics application: Here the authors relabel vertices of graphs using the Morgan index [Morgan, 1965] (defined in Section 1.4.1 of this thesis), which increases the specificity of labels by

augmenting them with information on the number of walks starting at a node, and thereby also helps reduce the runtime, as fewer vertices will match.

The shortest path kernel by Borgwardt and Kriegel [2005] compares shortest path lengths between pairs of nodes with matching source and sink labels in two graphs. The runtime of this kernel scales as $O(n^4)$. Ralaivola et al. [2005] propose specialized graph kernels for chemoinformatics that are efficient to compute on small molecules with an average degree of 2 or 3: They are based on molecular descriptors, and count labeled paths of length p that can be retrieved by depth-first search from each vertex. To compute a kernel on the resulting feature vectors, they use similarity measures derived from Tanimoto and Jaccard coefficients [Fligner et al., 2002].

1.5.4. Kernels based on small subgraphs

The second class, graph kernels based on limited-size subgraphs, includes kernels based on so-called *graphlets*, which represent graphs as distributions of all types of subgraphs of size $k \in \{3, 4, 5\}$. These kernels generally do not take into account node labels, and their naive computation for a pair of graphs has time complexity $O(n^{2k})$. Borgwardt et al. [2007] introduced sampling schemes to approximate the graphlet distribution. Chapter 2 of this thesis discusses these kernels in detail. Cyclic pattern kernels [Horváth et al., 2004] count pairs of matching cyclic patterns in two graphs. Computing this kernel for a general graph is unfortunately NP-hard, however there exist special cases where the kernel can be efficiently computed. The kernel, recently proposed by Costa and De Grave [2010], can also be classified in this category: It counts identical pairs of rooted subgraphs containing nodes up to a certain distance from the root, the roots of which are located at a certain distance from each other, in two graphs.

1.5.5. Kernels based on subtree patterns

The first kernel from the third class, subtree kernels, was defined by Ramon and Gärtner [2003]. Intuitively, to compare graphs G and G', this kernel iteratively compares all matchings between neighbors of two nodes v from G and v' from G'. In other words, for all pairs of nodes v from G and v' from G', it counts all pairs of matching substructures in subtree patterns rooted at v and v'. This kernel is discussed in detail in Section 3.2.2 of this thesis. While taking into account expressive subtree patterns, this kernel is unfortunately expensive to compute: The runtime complexity of the subtree kernel for a set of N graphs is $O(N^2 n^2 h\ 4^d)$, where n is the number of nodes, h is the maximum height of the subtree patterns considered, and d is the maximum degree in the graph data set.

The subtree kernels by Mahé and Vert [2009] and Bach [2008] refine the Ramon-Gärtner kernel for applications in chemoinformatics and hand-written digit recognition. Both Mahé and Vert [2009] and Bach [2008] propose to consider α-ary subtrees with at

most α children per node. This restricts the set of matchings to matchings of up to α nodes, but the runtime complexity is still exponential in this parameter α, which both papers describe as feasible on small graphs with many distinct node labels.

1.5.6. Other graph kernels

There exist graph kernels based on graph edit distances, discussed in Section 1.3.2 [Neuhaus and Bunke, 2006, 2007]. However, they are not positive semidefinite in general.

Another type of graph similarity measures, optimal assignment kernels [Fröhlich et al., 2005], arise from finding the best match between substructures of graphs. Unfortunately, these kernels are not positive semidefinite either [Vert, 2008].

1.5.7. Discussion

In this section, we presented an overview of the state of the art in graph kernels. Graph kernels are an attractive research direction with a view to designing efficient graph comparison and representation methods. First, as we pointed out in Section 1.5.1, any positive semidefinite kernel for graphs has the form $k(G, G') = \langle \phi(G), \phi(G') \rangle$, where $\phi(G)$ is the feature representation of G in a reproducing kernel Hilbert space: By choosing k (and therefore choosing ϕ, or vice versa), we respond to both graph comparison and graph representation problems given in Sections 1.3 and 1.4. Second, some kernels are able to efficiently compute inner products in very high or infinite-dimensional spaces: This can potentially allow us to design graph kernels that take into account rich structural and label information contained in graphs. Third, graph kernels make it possible to apply a large spectrum of learning algorithms to graph-structured data.

However, the potential of graph kernels has by no means been exhausted by the existing kernels for graphs. Efficiency is one of the most important issues: Notwithstanding important achievements in speeding up random walk graph kernels, there is no graph kernel that scales better than $O(n^3)$ for a pair of labeled graphs of n nodes. Moreover, random walk kernels suffer from tottering and halting due to the intrinsic properties of random walks (Section 1.5.3). Existing subtree pattern kernels are limited to small graphs of 20-30 nodes because of their high computational cost (Section 1.5.5), and optimal assignment kernels are not positive semidefinite (Section 1.5.6).

1.6. Contributions of this thesis

Our goal in this thesis was to design scalable kernels for efficiently comparing and representing large, labeled graphs for machine learning on graph-structured data.

The motivation for this objective comes from numerous application domains of machine learning where graph-structured data are becoming increasingly abundant, large-scale and diverse (see Section 1.1), and from the need for efficient graph comparison and

representation. As we have seen in Sections 1.3 and 1.4, most of the state-of-the-art graph comparison and representation methods are NP-hard, or lack expressivity. There exist polynomial-time alternatives, such as some topological descriptors (Section 1.4.1) and the algebraic approaches based on representative labeled graphlets, which we recently proposed [Kondor et al., 2009, and Section 1.4.3]; However, it may be difficult to decide which descriptors, respectively, graphlets, to use for a problem at hand. Graph kernels are a promising research direction: Being kernels, they have the potential to efficiently compare graphs in high-dimensional spaces that can accommodate large amounts of information about graphs, and to act as a bridge between graph-structured data and a large family of machine learning algorithms (see Section 1.5). However, as we argued in Section 1.5.7, despite significant advances in the field in recent years [Borgwardt, 2007, Vishwanathan et al., 2010], existing graph kernels do not fully respond to the current, increasingly demanding needs of machine learning on graph data.

1.6.1. Efficiently counting graphlets

As we have seen in Section 1.5, recent years have witnessed important steps forward in the design and improvement of graph kernels, including the speedup of random walk kernels from $O(n^6)$ to $O(n^3)$ for a pair of graphs of n nodes, and the development of shortest path, cyclic and subtree pattern kernels.

Unfortunately, none of these kernels have a better runtime than $O(n^3)$. This runtime is acceptable when comparing graphs of a few dozens of nodes. However, with the proliferation of large-scale graph data in different applications (see Section 1.1), it is becoming more and more pressing to design graph kernels that scale to large graphs of hundreds and thousands of nodes.

Chapter 2 presents our first contribution: In this work, we compare graphs by counting *graphlets*, or subgraphs with k nodes where $k \in \{3, 4, 5\}$. The choice of graphlets as representative substructures is motivated by the graph reconstruction conjecture [Kelly, 1957] and recent studies arguing about their importance in bioinformatics applications [Przulj, 2007, Milenkovic et al., 2010]. Moreover, being explicitly based on distributions of subgraphs of fixed size, our kernels do not suffer from tottering and halting, problems that plague the random walk kernel (see Section 1.5.3). This further strengthens our motivation to study graphlet kernels.

Exhaustive enumeration of all graphlets being prohibitively expensive, Borgwardt et al. [2007] have introduced a sampling scheme that significantly improved the computation runtime. However, this gain in speed is achieved to the detriment of precision. We develop another theoretically grounded speedup scheme that computes the graphlet distribution exactly, without approximation, in $O(nd^{k-1})$, where d is the maximum degree of the graph. As most real-world graphs are generally sparse with low maximum degree, our algorithms are expected to require much lower runtime than $O(n^{2k})$ on average. In our experimental evaluation, graphlet kernels allow us to efficiently compare

large graphs that could not be tackled by existing graph kernels, and the exact computation of the graphlet distribution systematically leads to improved results with respect to sampling.

1.6.2. Weisfeiler-Lehman graph kernels

While the efficient computation strategies for random walks [Vishwanathan et al., 2010] and graphlet kernels from the previous section yield an important gain in speed for unlabeled graphs with up to hundreds of nodes, it has been a general limitation of all the state-of-the-art graph kernels that they scale poorly to large, labeled graphs with more than a hundred nodes: In the worst case, none of them can scale better than $O(n^3)$. The efficient comparison of large, labeled graphs has remained an unsolved challenge for almost a decade.

In Chapter 3 we define a fast subtree kernel on graphs, the Weisfeiler-Lehman subtree kernel, that combines scalability with the ability to deal with node labels. It is competitive with state-of-the-art kernels on several classification benchmark data sets in terms of accuracy, and outperforms them significantly in terms of runtime on large graphs. This new kernel opens the door to applications of graph kernels on large graphs, for instance, protein function prediction via detailed graph models of protein structure on the amino acid level, or on gene networks for phenotype prediction in bioinformatics.

In Section 3.3, we generalize our subtree kernel to a family that encompasses many previously known kernels for graphs. In Sections 3.6 and 3.7 we present two other directions for extending the Weisfeiler-Lehman subtree kernel: The first deals with the inexact matching of subtree patterns, and the second addresses the problem of comparing nodes within a graph.

1.6.3. Applications in remote sensing and structural biology

For decades, graph comparison applications have been almost exclusively limited to the comparison of small molecules in chemoinformatics because of the lack of scalable algorithms for graph comparison and representation. Our efficient graph kernels open the way to new applications involving the comparison of large, labeled graphs.

In Chapter 4, we present two such applications. The first application domain that we consider is remote sensing image classification: Remote sensing images are images of earth acquired from airborne and satellite sensors, in which pixels are usually high-dimensional, and the task is to correctly segment the image into a thematic representation, given a low number of correctly labeled pixels. Due to the lack of methods that can efficiently take into account the structure of the underlying large pixel graph, most state-of-the-art classifiers for remote sensing images are spectral-based (pixel-based) and thus do not use the intuition that neighboring pixels should tend to belong to the same class. In Section 4.1, we propose a graph kernel-based approach that takes into account

both spectral and spatial information of the pixels at different resolutions, and yields competitive accuracy compared to state-of-the-art remote sensing image classification methods.

In Section 4.2 we apply graph kernels to the problem of predicting *chemical shifts* based on the protein structure. Chemical shifts are shifts in signal frequency of atomic nuclei in the protein due to local chemical structure. They are obtained using Protein Nuclear Magnetic Resonance (NMR) spectroscopy, and the task is to correctly assign peaks in this spectrum of chemical shifts to the corresponding nuclei (that is, those which exhibit these chemical shifts) in the protein. Providing good solutions to this problem would allow us, for instance, to understand how certain drugs alter the function of certain proteins. Most state-of-the-art methods for protein function prediction are based on physico-chemical properties of the neighborhoods of nuclei, these neighborhoods being defined as the amino-acid containing the nucleus and its neighboring amino-acids in the protein sequence. However, an atom in a protein can be chemically influenced by another atom far away according to the protein sequence, but close in the 3D structure of the protein. In other words, while a graph representation of the local 3D structure around a nucleus would be more natural, many existing methods use a poorer representation based on the sequence. Other methods that do exploit the whole neighborhood of nuclei rely on parametric models of chemical shifts which may not be precise enough. Unlike existing approaches, we represent local chemical neighborhoods of nuclei as weighted labeled graphs, taking into account every atom in a certain radius around the nucleus. We hope that this ongoing work with promising preliminary results will lead to a useful tool for predicting chemical shifts.

1.6.4. Published work appearing in this thesis

This thesis contains material, in abbreviated, modified or extended form, from several published articles. We list these publications below, indicating the corresponding sections of this manuscript.

- N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2009.

  Chapter 2 of this thesis is an extended version of this article.

- I. R. Kondor, N. Shervashidze, and K. M. Borgwardt. The graphlet spectrum. In *Proceedings of the International Conference on Machine Learning*, pages 529–536, 2009.

  Section 1.4.3 of Chapter 1 uses material from this work.

- N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*,

pages 1660–1668, 2009.

Section 3.2 of Chapter 3 is based on the work presented in this paper.

- N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

  This article gave rise to the bulk of Chapter 3 of this thesis, as well as Sections 1.5.2-1.5.6 of Chapter 1.

- G. Camps-Valls, N. Shervashidze, and K. M. Borgwardt. Spatio-spectral remote sensing image classification with graph kernels. *IEEE Geoscience and Remote Sensing Letters*, 7(4):741 –745, 2010.

  Section 4.1 of Chapter 4 closely follows this article.

Certain parts of this thesis are based on unpublished work done in collaboration with other researchers. Section 3.6 in Chapter 3 is based on my unpublished research with Alexander J. Smola and Karsten M. Borgwardt. The material presented in Section 3.7 is based on unpublished work with Hyokun Yun, Alexander J. Smola and Karsten M. Borgwardt. The ongoing work presented in Section 4.2 of Chapter 4 is being done in collaboration with Michael Habeck.

2. The graphlet kernels

This chapter presents our first contribution to the design of efficient graph kernels for large graph comparison. We here represent graphs by counting *graphlets*, that is, subgraphs of k nodes where $k \in \{3, 4, 5\}$[1].

One may question the choice of graphlets as representative substructures: A natural approach to speeding up graph kernels would have been to try to further accelerate the computation of the random walk kernel, the existing graph kernel with the most scalable runtime. However, random walk kernels have inherent problems of tottering and halting, discussed in Section 1.5.3: It is not obvious how to get rid of these problems while still considering walks as basic substructures, as they are caused by the very nature of walks. Graphlets do not suffer from these problems as no node or edge is repeated in a graphlet. This choice is further motivated by the graph reconstruction conjecture [Kelly, 1957], that states that any graph of size n can be reconstructed from the set of all its subgraphs of $n-1$ nodes [Borgwardt, 2007]. We define the graphlet kernels in Section 2.1.

Exhaustive enumeration of graphlets is expensive, scaling as $O(n^k)$ where n is the number of nodes in the graph and k the size of the graphlets. Borgwardt et al. [2007] have proposed a sampling scheme for estimating the graphlet distribution, showing that sampling a fixed number of graphlets suffices to bound the deviation of the empirical estimate of the graphlet distribution from the true distribution (Section 2.2). However, while the obtained bounds hold for any graph, the notion of deviation used in this sampling scheme is not adequate when graphs are sparse. We here show that for graphs of degree bounded by d, the *exact* number of all graphlets of size k can be determined in time $O(nd^{k-1})$ for $k \in \{3, 4, 5\}$ (Section 2.3). Note that large, real-world graphs are often sparse with $d \ll n$. Finally, we experimentally show that graphlet kernels allow us to compute graph kernels on graphs of sizes that are beyond the scope of the state of the art, and that our exact counting scheme systematically gives better performance than sampling in graph classification (Section 2.4).

2.1. The graphlet kernel

Let $\mathcal{G}_k = \{g_1, \ldots, g_{n_k}\}$ be the set of graphlets of size k and G be a graph of size n. Let us define $c_i^k(G)$ as the number of occurrences of the graphlet g_i with k nodes in G, that is, the

[1] As our main concern is efficiency, we do not consider graphlets of size $k \geq 6$ in this work. In fact, nor have they ever been considered in the literature.

cardinality of the set of induced subgraphs of G isomorphic to g_i, $\{H|H \sqsubseteq G, H \simeq g_i\}$. Define $c^k(G)$ as the vector $(c_1^k(G), \ldots, c_{n_k}^k(G))$. We call $c^k(G)$ the k-spectrum of G. This statistic will serve as the foundation for the graphlet kernels. Note that whenever the value of k is clear from the context, we will omit the superscript k in $c_i^k(G)$ and use $c_i(G)$ instead to avoid cluttering the notation.

Definition 2.1.1 (Graphlet kernel) *Given k and two graphs G and G' of size $n \geq k$, the graphlet kernel K_k is defined as*

$$K_k(G, G') = c^k(G)^\top c^k(G'). \tag{2.1}$$

In order to account for differences in the sizes of the graphs, which can greatly skew the counts $c^k(G)$, we normalize the counts to vectors of frequencies. For a graph G of size n, we set

$$f^k(G) = \frac{1}{\binom{n}{k}} c^k(G),$$

and work with the following normalized variant of (2.1):

$$K_k(G, G') = f^k(G)^\top f^k(G'). \tag{2.2}$$

Note that there are many other possibilities of defining a kernel using $c^k(G)$. Essentially, $c^k(G)$ is a feature vector representing G by the counts of graphs of size k that it contains. Once we have this representation, we can use any kernel or non-kernel technique to learn on graphs.

Since there are $\binom{n}{k}$ subgraphs of size k in a graph, computing $c^k(G)$ for each graph of size n requires $O(n^k)$ effort. Once all the $c^k(G)$ vectors are computed, then computing (2.2) requires essentially $O(1)$ work. In the sequel we will show how the $O(n^k)$ preprocessing step can be made efficient.

Clearly, if $G \simeq G'$, then $c^k(G) = c^k(G')$. But is the reverse true? It has been shown that when $n = k+1$ and $n \leq 11$, equality of k-spectra implies isomorphism [Kelly, 1957, McKay, 1997]. For $n > 11$, it is still a conjecture whether a graph of size n can be reconstructed from its subgraphs of size $n-1$.

Note that while the term *graphlet kernel* was originally used by Borgwardt [2007] in the same sense as we do in this thesis, more recently it has also been employed to denote a kernel comparing nodes in a network using counts of connected graphlets adjacent with a node [Vacic et al., 2010].

2.2. Sampling graphlets

We here detail the sampling approach to the estimation of graphlet distributions, proposed by Borgwardt et al. [2007].

To compute the graphlet kernel exactly, one needs to count all graphlets of size k in the input graphs. A graph with n nodes has $\binom{n}{k} = O(n^k)$ graphlets, whose straightforward enumeration may be expensive. Sampling is an extensively studied technique for estimating distributions. It relies on the fact that if a sufficient number of random samples is drawn, then the underlying distribution can be well approximated by the empirical distribution. The number of samples necessary to guarantee with a certain confidence that the empirical distribution will not deviate from the true distribution more than a given ϵ is called the sample complexity.

Subgraph sampling methods for graphs have been widely studied before in the bioinformatics literature [Kashtan et al., 2004, Wernicke, 2005, Przulj, 2007]. These approaches are unfortunately not satisfactory, as they are ad hoc and do not provide any guarantees on the divergence between the empirical and the true distributions. Borgwardt et al. [2007] derive sample complexity bounds for estimating graphlet distributions by adapting results by Weissman et al. [2003], who proved distribution-dependent bounds for the ℓ_1 deviation between the true and the empirical distributions.

2.2.1. Bounding the sample complexity

Let $\mathcal{A}$ denote the finite set $\{1, 2, \ldots, a\}$, where $a \in \mathbb{N}$. For two probability distributions P and Q on $\mathcal{A}$, the ℓ_1 distance between P and Q is defined as

$$\|P - Q\|_1 := \sum_{i=1}^{a} |P(i) - Q(i)|. \tag{2.3}$$

Let $X^m = X_1, \ldots, X_m$ be independent identically distributed random variables drawn from some distribution P (denoted as $X_j \sim P$). The empirical estimate of P is defined as

$$\hat{P}_{X^m}(i) = \frac{1}{m} \sum_{j=1}^{m} \delta(X_j = i),$$

where $i \in \mathcal{A}$, and $\delta(\cdot)$ denotes the indicator function: $\delta(X_j = i)$ equals 1 if $X_j = i$ and 0 otherwise.

Weissman et al. [2003] prove the theorem below (Theorem 2.2.1), which, by Corollary 2.2.2, allows us to determine the number of samples that is sufficient for the probability of the ℓ_1 distance being larger than a given ϵ to be smaller than a given δ.

Theorem 2.2.1 *Let P be a probability distribution on the finite set $\mathcal{A} = \{1, \ldots, a\}$. Let $X^m = X_1, \ldots, X_m$ be independent identically distributed random variables distributed according to P. Let $\hat{P}_{X^m}$ be the empirical distribution on $\mathcal{A}$. Then, for all $\epsilon > 0$,*

$$Pr\left\{\|P - \hat{P}_{X^m}\|_1 \geq \epsilon\right\} \leq (2^a - 2)e^{-m\phi(\pi_P)\epsilon^2/4}, \tag{2.4}$$

where $\pi(P) = \max_{A \subseteq \mathcal{A}} \min(P(A), 1 - p(A))$*, and for* $p \in [0, 1/2)$*,* $\phi(p) = \frac{1}{1-2p} \log(\frac{1-p}{p})$*.*

The next corollary follows from Theorem 2.2.1.

Corollary 2.2.2 *Let* P *be a probability distribution on the finite set* $\mathcal{A} = \{1, \ldots, a\}$*. Let* $X = \{X_j\}_{j=1}^{m}$*, with* $X_j \sim P$*. For a given* $\epsilon > 0$ *and* $\delta > 0$*,*

$$m = \left\lceil \frac{2\left(a \log(2) + \log\left(\frac{1}{\delta}\right)\right)}{\epsilon^2} \right\rceil \tag{2.5}$$

samples suffice to ensure that $Pr\left\{\|P - \hat{P}_{X^m}\|_1 \geq \epsilon\right\} \leq \delta$*.*

Borgwardt et al. [2007] apply Corollary 2.2.2 to graphlet distribution estimation by setting $\mathcal{A}$ to be the set of all graphlets of size k, m to be the number of graphlets randomly sampled from the graph, and by assuming that they are distributed according to an unknown distribution P. Setting m according to (2.5) ensures with confidence $1 - \delta$ that the empirical distribution $\hat{P}_{X^m}$ will not deviate more than ϵ from the true distribution P (in the sense of ℓ_1 distance). An attractive feature of this approach is that the sample complexity (2.5) does not depend on graph size, but only on the size of graphlets we are interested in.

The number of distinct unlabeled graphlets of sizes $k = 3$, 4, and 5 are respectively $n_k = 4$, 11, and 34 (see Figures 2.2, 2.4, and B.1). For example, if we take the case $k = 4$, and fix both ϵ and δ to 0.05, then the number of samples verifying (2.5) equals 8,497. If we decrease ϵ and δ to 0.01, the number of samples grows to 244,596.

2.3. Bounded degree graphs

While sampling allows us to deal with graphs on which the exhaustive enumeration of all graphlets is infeasible, in pratice, there is a large fraction of graphs on which exact counting can be performed efficiently: the class of graphs with a bounded degree d.

There is another important reason why it is interesting to exactly count graphlets instead of sampling them: As we mostly deal with sparse graphs in real-world applications, the larger the graphs grow, the sparser they become and the more the number of graphlets with no edges is expected to dominate the number of graphlets of all other types. This leads to greatly skewed graphlet distributions. In these cases, the ℓ_1 deviation is no more an accurate measure of the quality of the empirical distribution, as the frequencies of connected graphlets are too small compared to the frequencies of disconnected graphlets. To account for this, one may strongly decrease the values of ϵ and δ to encourage the accurate estimation of the proportions of connected graphlets. This, in turn, results in greater sample complexity, which means that sampling can potentially take even longer than exact counting of graphlets.

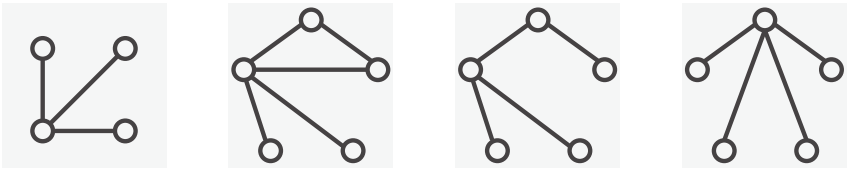

Figure 2.1.: Connected graphlets of size $k \in \{3, 4, 5\}$ which do not contain a path of length $k - 1$.

In other words, while the graph size does not directly influence the sample complexity (2.5), it does indirectly affect the number of samples needed to accurately estimate the graphlet distribution, as the parameters ϵ and δ that are precise enough for flatter graphlet distributions in small graphs do not suffice to accurately approximate the skewed graphlet distributions in large graphs. We empirically study this effect in our experiments in Section 2.4.2.

We present two algorithms for efficiently counting graphlets in graphs of low degree: one for counting *all connected graphlets*, and one for counting *all graphlets*.

2.3.1. Enumerating all connected graphlets

We assume that our graphs are given in standard adjacency list representation. As a preprocessing step, we construct a data structure that supports edge existence verification in time $O(1)$. Given vertices u and v, the data structure checks whether $(u, v) \in E$. We may either use the adjacency matrix or a hashing scheme [Mehlhorn and Sanders, 2008, Chapter 4]. Observe that the adjacency matrix can be constructed in time $O(|E|)$ if one uses an implicit initialization scheme [Mehlhorn and Sanders, 2008, Exercise 3.16]. With such a data structure one can determine which graphlet is induced by a path of length k in time $O(k^2)$.

Theorem 2.3.1 *Let G be a graph, and let d denote its maximum degree. Then all connected graphlets of G of size $k \in \{3, 4, 5\}$ can be enumerated in $O(nd^{k-1})$ time.*

Proof We assume that given k nodes in a graph, we can look up the graphlet they form in constant time. Graphlets of size k can be divided into two classes: graphlets that contain a simple path of length $k - 1$, and graphlets that do not contain such a path.

By a simple depth first search (DFS), all paths of length $k - 1$ originating from a node v can be enumerated in $O(d^{k-1})$ time. Counting the graphlets of size k induced by these paths requires $O(d^{k-1})$ effort; the overall complexity for a graph with n nodes is therefore $O(nd^{k-1})$. Note that the same graphlet may be induced by more than one

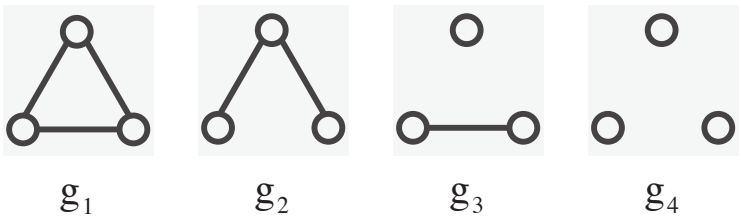

Figure 2.2.: All graphlets of size 3.

path of length $k-1$, and hence may be counted multiple times. To account for this, we need to divide the final counts by the number of paths of length $k-1$ per graphlet.

The connected graphlets that do not contain a path of length $k-1$ can also be enumerated efficiently for $k \in \{3,4,5\}$. For $k=3$, there is no connected graphlet that does not contain at least one path of length 2.

For $k=4$, there is only one connected graphlet that does not contain a path of length 3 (see I in Figure 2.1). Let us call this graphlet a 3-star. Let d_i denote the degree of node v_i. We look up the $\binom{d_i}{3}$ neighbor triplets of v_i, and check if they induce the graphlet we are interested in. The time complexity per node is $O(d^3)$, and for the entire graph is $O(nd^3)$.

For $k=5$, there are 3 connected graphlets with no path of length 4 (see II to IV in Figure 2.1). To compute the frequency of their occurrence we note that all contain the 3-star as a subgraph. So we first enumerate all occurrences of the 3-star, and then check the neighbors of each node in the 3-star to see whether they induce the graphlets in question. This step has a complexity of $O(d)$ per graphlet, bringing the overall complexity of the method to $O(nd^4)$, as claimed. ■

2.3.2. Counting all graphlets

Here we show that all graphlets of size 3, 4 and 5 can be counted efficiently in graphs with a bounded degree.

Theorem 2.3.2 *Let G be a graph, and let d denote its maximum degree. For a fixed node v_1 in G, we can count all subgraphs of size 3, 4, and 5 containing v_1 in time $O(d^2)$, $O(d^3)$, and $O(d^4)$ respectively.*

Proof Let us first consider counting graphlets of size 3. Modulo isomorphism there are 4 types of such graphlets (see Figure 2.2). We first count subgraphs with at least one edge and then obtain the number of graphlets with no edge, $c_4^3(G)$, by subtracting

$c_1^3(G)+c_2^3(G)+c_3^3(G)$ from $\binom{n}{3}$, which is the number of all triplets of nodes in the graph. In the following, we will systematically omit the superscript of $c_i^k(G)$ whenever the value of k is clear from the context.

For each pair of nodes (v_1, v_2) connected by an edge, we have to distinguish four cases for the third node v_3: $v_3 \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)$, $v_3 \in \mathcal{N}(v_1) \setminus (\mathcal{N}(v_2) \cup \{v_2\})$, $v_3 \in \mathcal{N}(v_2) \setminus (\mathcal{N}(v_1) \cup \{v_1\})$, $v_3 \notin \mathcal{N}(v_1) \cup \mathcal{N}(v_2)$.

It is easy to see that the subgraph spanned by v_1, v_2 and v_3 in the first case corresponds to the graphlet with 3 edges, in the second and third cases it corresponds to the graphlet with 2 edges, and in the fourth to the graphlet with 1 edge. The pseudocode for this procedure is given in Algorithm 1, illustrated by Figure 2.3.

Algorithm 1 Count graphlets of 3 nodes in a graph $G = (V, E)$

1: $c_i(G) \leftarrow 0, i \in \{1, 2, 3\}$
2: **for all** edges $(v_1, v_2) \in E$ (illustrated by black nodes in Figure 2.3) **do**
3: $\quad c_1(G) \leftarrow c_1(G) + |\mathcal{N}(v_1) \cap \mathcal{N}(v_2)|$ ▷ red nodes
4: $\quad c_2(G) \leftarrow c_2(G) + |\mathcal{N}(v_1) \setminus (\mathcal{N}(v_2) \cup \{v_2\})|$ ▷ green nodes
5: $\quad c_2(G) \leftarrow c_2(G) + |\mathcal{N}(v_2) \setminus (\mathcal{N}(v_1) \cup \{v_1\})|$ ▷ green nodes
6: $\quad c_3(G) \leftarrow c_3(G) + (n - |\mathcal{N}(v_1) \cup \mathcal{N}(v_2)|)$ ▷ white nodes
7: **end for**
8: $c_1(G) \leftarrow c_1(G)/6$ ▷ as g_1 has 3 undirected edges
9: $c_2(G) \leftarrow c_2(G)/4$ ▷ as g_2 has 2 undirected edges
10: $c_3(G) \leftarrow c_3(G)/2$ ▷ as g_3 has 1 undirected edge
11: $c_4(G) \leftarrow \binom{n}{3} - [c_1(G) + c_2(G) + c_3(G)]$

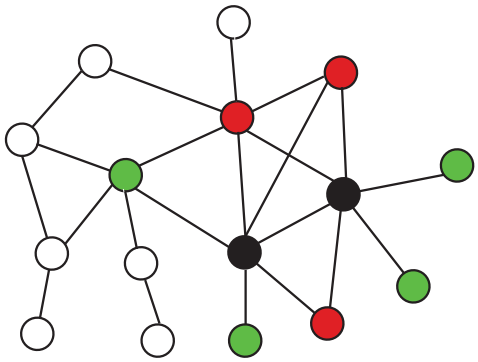

Figure 2.3.: Counting graphlets of size 3.

Enumerating all pairs of edges originating at v_1 is a $O(d)$ effort. For each pair (v_1, v_2), determining the cardinality of the sets $\mathcal{N}(v_1) \cap \mathcal{N}(v_2)$, $\mathcal{N}(v_1) \setminus (\mathcal{N}(v_2) \cup \{v_2\})$ and

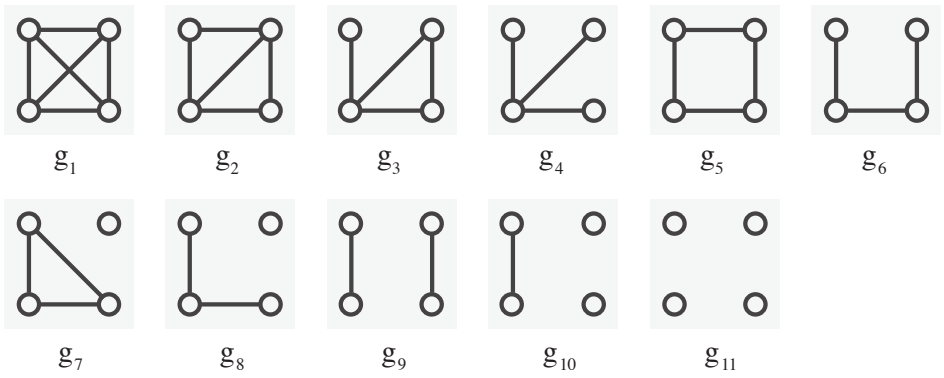

Figure 2.4.: All graphlets of size 4.

$\mathcal{N}(v_2) \setminus (\mathcal{N}(v_1) \cup \{v_1\})$ has $O(d)$ time complexity as well. As to the cardinality of the set $V \setminus (\mathcal{N}(v_1) \cup \mathcal{N}(v_2))$, it can be easily computed by observing that it is equal to $n - |\mathcal{N}(v_1) \cup \mathcal{N}(v_2)|$. This leads to the overall complexity of $O(d^2)$.

Note that counting graphlets in the proposed way would imply counting them twice as many times as the number of edges they contain. To deal with this, we need to divide the final counts by twice the number of edges per graphlet.

We now consider graphlets of size 4.

Modulo isomorphism there are 11 graphlets of size 4 (see Figure 2.4). As in the previous case, we will first count all graphlets which contain at least one edge.

Assume we want to count subgraphs containing an edge (v_1, v_2). As before, for v_2 there are $|\mathcal{N}(v_1)|$ choices and for each pair (v_1, v_2) we have 4 cases for the third node v_3: $v_3 \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)$, $v_3 \in \mathcal{N}(v_1) \setminus (\mathcal{N}(v_2) \cup \{v_2\})$, $v_3 \in \mathcal{N}(v_2) \setminus (\mathcal{N}(v_1) \cup \{v_1\})$, and $v_3 \notin \mathcal{N}(v_1) \cup \mathcal{N}(v_2)$.

v_3 from the first three cases can be enumerated in $O(d)$. And for each triplet (v_1, v_2, v_3) we can count subgraphs of size 4 containing this triplet in $O(d)$, as we can compute cardinalities of all intersections of $\mathcal{N}(v_1)$, $\mathcal{N}(v_2)$ and $\mathcal{N}(v_3)$ in $O(d)$.

As to v_3 in the fourth case, there are 2 types of graphlets which arise in this case and do not arise in previous cases: graphlets g_9 and g_{10}. For fixed v_1 and v_2 it is possible to obtain the number of graphlets of type g_9 by counting the number of edges not adjacent to any of the nodes $v \in \mathcal{N}(v_1) \cup \mathcal{N}(v_2)$. This quantity is equal to $m + 1 - |\mathcal{N}(v_1)| - |\mathcal{N}(v_2)| - K$, where m is the number of edges in the graph which can be precomputed, $|\mathcal{N}(v_1)| + |\mathcal{N}(v_2)| - 1$ is the number of edges adjacent to v_1 or v_2 and K is the number of edges adjacent to nodes in $(\mathcal{N}(v_1) \cup \mathcal{N}(v_2)) \setminus \{v_1, v_2\}$. The latter equals to the number of previously counted graphlets with the same v_1 and v_2, where v_3 and v_4 are connected (i.e., where $v_4 \in \mathcal{N}(v_3)$). Once $c_9(G)$ is computed, we obtain $c_{10}(G)$ by subtracting $c_9(G)$

from $\binom{n-|\mathcal{N}(v_1)\cup\mathcal{N}(v_2)|}{2}$, which is the number of pairs of nodes outside $\mathcal{N}(v_1) \cup \mathcal{N}(v_2)$. The fourth case does not increase the runtime complexity of previous cases and remains $O(d^3)$ per v_1.

At last, $c_{11}(G) = \binom{n}{4} - \sum_{i=1}^{10} c_i(G)$.

Note that, as in the case of graphlets with 3 nodes, we will count each graphlet at least twice as many times as the number of edges it contains. Additionally, in case v_3 and v_4 are both neighbors of v_1 or v_2 (that is, all configurations except $v_4 \in \mathcal{N}(v_3)\backslash(\mathcal{N}(v_1)\cup\mathcal{N}(v_2))$), the graphlet spanned by these four nodes will be counted twice per fixed (v_1, v_2). To avoid this, we divide the counts of these graphlets by 2.

For the proof for graphlets of size 5, please refer to Appendix B.

■

data set	size	classes	# nodes	# edges
MUTAG	188	2 (125 vs. 63)	17.7	38.9
PTC	344	2 (192 vs. 152)	26.7	50.7
ENZYMES	600	6 (100 each)	32.6	124.3
D&D	1178	2 (691 vs. 587)	284.4	1921.6

Table 2.1.: Statistics on used data sets.

2.4. Experiments

To evaluate our approaches, we perform two sets of experiments. First we compare the graphlet kernel to other state-of-the-art graph kernels on unlabeled graph benchmark data sets (Section 2.4.1). Next, we experimentally verify our argument from Section 2.3 that the sampling approach of Borgwardt et al. [2007] suffers in sparse graphs (Section 2.4.2).

2.4.1. Graph classification

We here evaluate the performance of the graphlet kernel and compare it with state-of-the-art graph kernels in terms of runtime, scalability, and prediction accuracy. Our baseline comparison methods are the classic random walk kernel of Gärtner et al. [2003], Kashima et al. [2004] and Vishwanathan et al. [2010], that counts common walks in two graphs, and the shortest path kernel of Borgwardt and Kriegel [2005], that compares shortest path lengths in two graphs. Both these kernels work on generic graphs, and are shown to perform competitively in their respective publications. For the random walk kernel we uniformly set the decay factor $\lambda = 10^{-4}$. For the shortest path kernel we used the Dirac δ kernel to compare shortest-path distances.

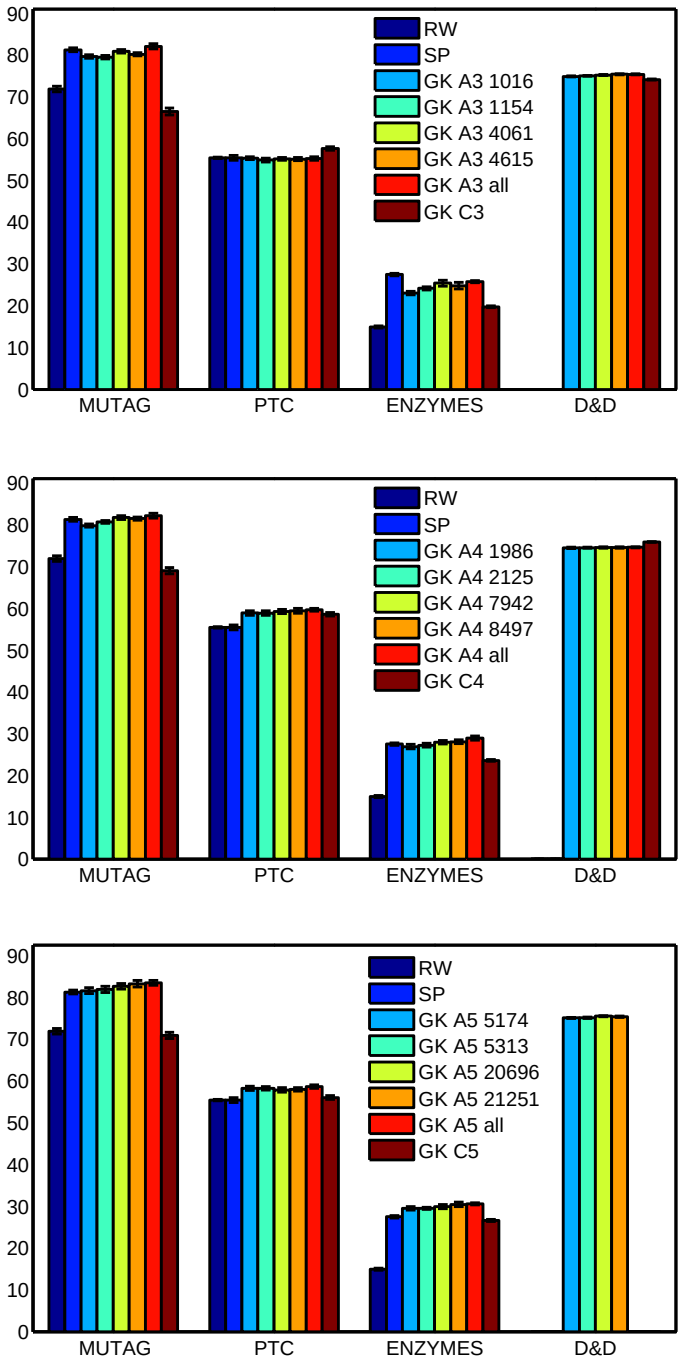

Figure 2.5.: Prediction accuracy on unlabeled graph classification benchmark data sets. RW and SP denote the random walk and the shortest path kernels. GK Ak all and GK Ck respectively denote the graphlet kernel with exact counting of all and only connected graphlets. GK Ak m denotes the graphlet kernel computed using m samples of size k graphlets. We here only report results for kernels whose computation took less than 24 hours.

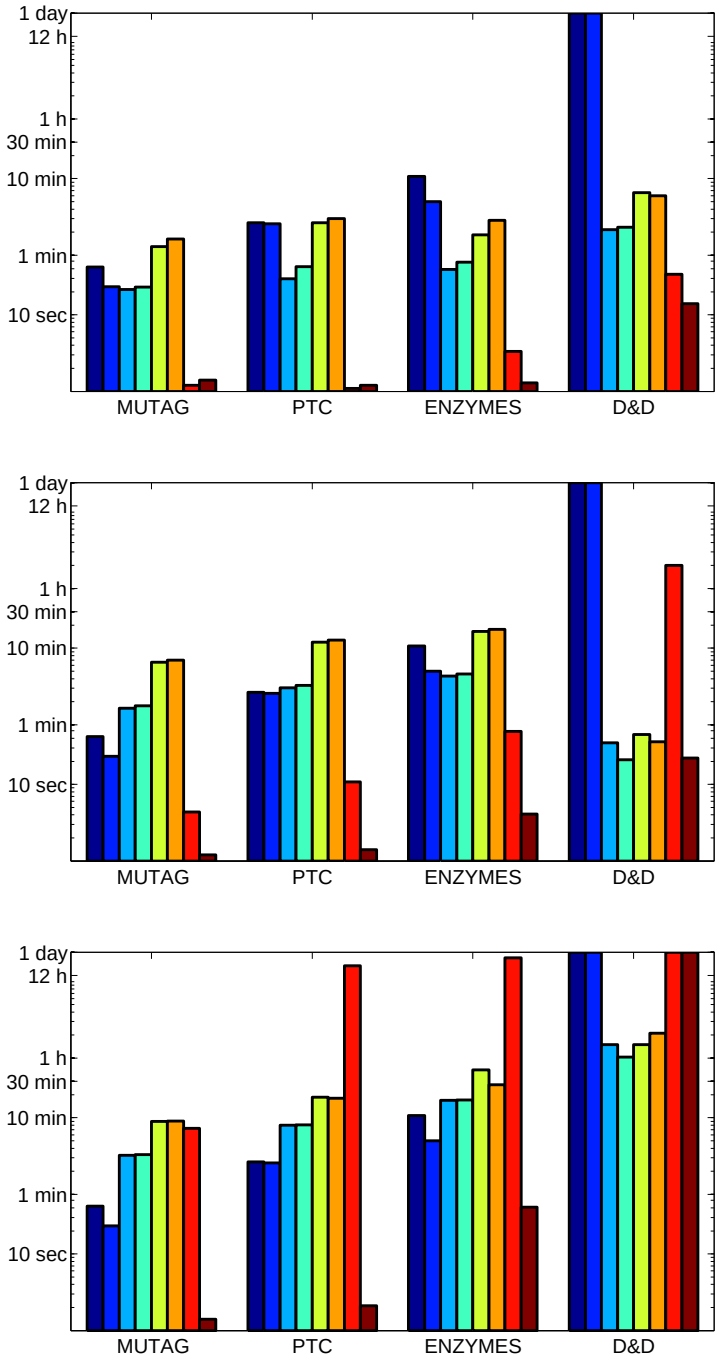

Figure 2.6.: CPU runtime for kernel computation on unlabeled graph classification benchmark data sets (implemented in Matlab). The color codes in each of these plots are the same as in the corresponding plots in Figure 2.5.

Data sets We perform experiments on three different well-known, publicly available data sets, namely MUTAG, PTC, and ENZYMES. We also test our kernels on a large protein function prediction data set from Dobson and Doig [2003], which we will refer to as D&D. Table 2.1 provides a summary of data sets used in our experiments.

Experimental setup We test different variants of graphlet kernels by varying the graphlet sizes $k \in \{3, 4, 5\}$, the types of graphlets we consider (connected vs. all), and the sample size for sampling-based kernels (different values of precision, ϵ, and confidence, δ).

To compute the 3-graphlet kernel based on sampling, we drew samples of 1016 graphlets (corresponding to $\epsilon = 0.1$, $\delta = 0.1$), 1154 graphlets ($\epsilon = 0.1$, $\delta = 0.05$), 4061 graphlets ($\epsilon = 0.05$, $\delta = 0.1$) and 4615 graphlets ($\epsilon = 0.05$, $\delta = 0.05$). Analogously, for kernels based on graphlets of size 4 and 5 we used four sample sizes according to the same values of ϵ and δ: {1986, 2125, 7942, 8497} and {5174, 5313, 20696, 21251}, respectively.

We use a binary C-Support Vector Machine (SVM) to evaluate the accuracy in graph classification yielded by our kernels. We perform 10-fold cross validation, and for each fold we independently tune the value of C, the SVM regularization parameter, by considering the training data from that fold. This process is averaged over 10 random splits of the data. We report classification accuracies and runtimes for kernel matrix computation in Figures 2.5 and 2.6.

Results On MUTAG, PTC and ENZYMES, modeled as unlabeled graphs, graphlet kernels based on exactly counting all graphlets reached the highest accuracy. On D&D it is either the graphlet kernel counting all graphlets or the one counting connected graphlets that achieve the best accuracy. Graphlet kernels based on sampling yield systematically worse but comparable results. The classification accuracies they reach are comparable to that of the shortest path kernel on MUTAG and ENZYMES and better on PTC. In all cases, all graphlet kernels comprehensively outperform random walk kernels.

In terms of runtime, 4 and 5-node graphlet sampling and 5-node graphlet counting are expensive and slower than the shortest path and the random walk kernels on small data sets such as MUTAG and PTC. As graph size increases (ENZYMES), graphlet sampling gets more competitive: Sampling 1986 and 2125 graphlets of size 4 on ENZYMES is already faster than computing shortest path and random walk kernels. On D&D, none of the latter kernels finishes computation within 24 hours, nor does the counting of all 5-node graphlets. The graphlet kernels based on sampling manage to compute kernel matrices on D&D in less than 2 hours and 7 minutes for 5-node graphlets.

Kernels based on counting graphlets in bounded degree graphs are fast to compute for MUTAG, PTC, ENZYMES, but less so for D&D. This is due to the fact that the first three data sets have a much lower maximum degree than D&D. In terms of accu-

racy, on PTC and D&D they are comparable with other kernels, while on MUTAG and ENZYMES they perform worse. Disconnected graphlets seem to be essential for correct classification on these data sets.

We note in the passing that even though our graphlet kernels do not exploit any domain knowledge and operate on simple unlabeled graph models of proteins, on the D&D data set the classification accuracy they obtain is comparable with published work that uses heavily annotated vector or graph models of proteins [Dobson and Doig, 2003, Borgwardt et al., 2005].

2.4.2. Sparse graphs and ℓ_1 deviation

In this section, we empirically study the behavior of the sampling approach from Section 2.2 with respect to the sparsity of graphs. Our hypothesis is that the estimation quality of the distribution of connected graphlets decreases as graphs become sparse, even if the ℓ_1 deviation of the empirical distribution from the true distribution is guaranteed with high confidence to be smaller than a given ϵ independently from the size of the graphs.

Experimental setup We assessed the ℓ_1 distances between the empirical and true graphlet distributions (on all and on only connected graphlets) on randomly generated graphs of different density levels. For a graph of size n, its density is defined as the number of edges of the graph divided by $n(n-1)/2$. We varied the graph density parameter c in $\{0.05, 0.1, \ldots, 0.5\}$. For each value of c, we generated 10 random graphs of density c, each of size 100. For graphlet sizes $k = 3$ and 4, we evaluated the true and empirical graphlet distributions on these graphs. The empirical distributions were estimated with the sample size corresponding to $\epsilon = 0.1$ and $\delta = 0.1$, that is, 1016 and 1986 for $k = 3$ and $k = 4$ respectively.

The true and empirical distributions on connected graphlets were obtained by renormalizing the components corresponding to connected graphlets in the complete graphlet distributions.

For each value of graph density c, we report the mean and the standard deviation of the 10 ℓ_1 distances corresponding to 10 graphs with density c. We present our experimental results in Figure 2.7.

Results First, we observe that both for $k = 3$ and $k = 4$, the overall ℓ_1 deviation is well under $\epsilon = 0.1$ no matter how sparse the graphs are. This agrees with the theory discussed in Section 2.2. However, when we consider the distributions restricted to only connected graphlets, we clearly see a deterioration in the distribution approximation quality as graphs become sparse. This deterioration is significantly more dramatic with larger graphlets. In the case of graphlets of size 4, we see that for sparse graphs the ℓ_1 deviation between the empirical and true distributions on connected graphlets can be more than 20 times worse compared to the overall deviation.

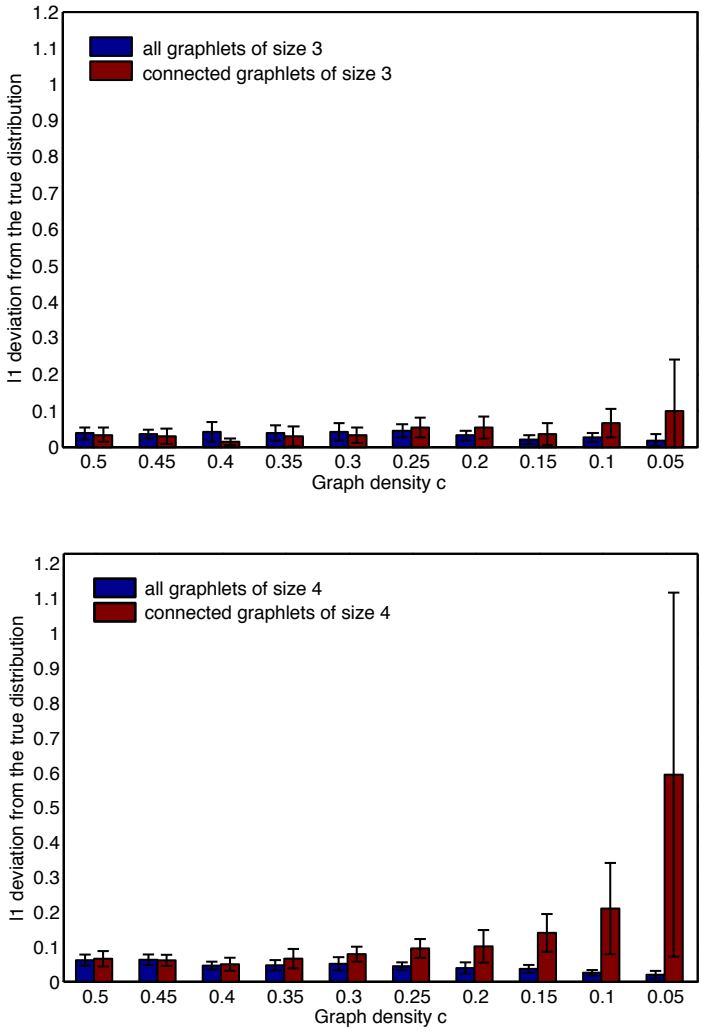

Figure 2.7.: The quality of empirical distributions on all graphlets and on connected graphlets.

In the light of these observations, we can reinterpret the experimental results in Figure 2.5 from the previous section. In all three plots in Figure 2.5, for data sets MUTAG, PTC, and ENZYMES we observe that classification accuracy improves with the sample size. However, on D&D we see almost no difference in classification accuracy when considering different sample sizes. This is likely to be caused by the sparsity of the large

graphs in D&D: In fact, the average density of graphs in MUTAG, PTC and ENZYMES is of 0.14, 0.12 and 0.16 respectively, while it only amounts to 0.03 in D&D.

To summarize, the sampling approach presented in Section 2.2 is well suited for graphs that are sufficiently dense for ℓ_1 distance to be an appropriate quality measure of the empirical distribution of graphlets. However, most large real-world graphs are too sparse for this condition to hold. In contrast, the exact counting approach that we propose is most efficient when the graphs are sparse. Therefore these two approaches can be viewed as complementary to each other.

2.5. Summary

In this chapter, we proposed efficiently computable kernels for unlabeled graphs, based on counting subgraphs of limited size (or graphlets) in a graph. Straightforward enumeration of graphlets being prohibitively expensive, Borgwardt et al. [2007] had proposed a sampling scheme to approximate graphlet distributions with a given precision and confidence. We here presented efficient algorithms for exactly counting graphlets. We experimentally showed that when graphs are sparse, the measure of precision used by Borgwardt et al. [2007] does not capture well the approximation quality of the distribution of connected graphlets. Our new algorithms for exact counting, in contrast, exploit the sparsity of graphs.

In our graph classification experiments, graphlet kernels scale to large graphs that could not be tackled by existing kernels for unlabeled graphs, and yield competitive classification accuracy with the state of the art. Our methods for representing unlabeled graphs with their graphlet distributions are not limited to being used in graph kernels, but can be applied in a variety of problems in learning on graph-structured data. A challenge for future research would be to take into account node and edge labels of graphs when counting graphlets.

3. Weisfeiler-Lehman kernels

Chapter 2 presented efficient sampling and counting schemes for graphlets, which enable us to compare large unlabeled graphs. However, it is not straightforward to extend them to labeled graphs, especially when the label alphabet is large. In fact, for a given label alphabet Σ, the number of graphlets of size k is $O(n_k|\Sigma|^k)$, where n_k is the number of distinct unlabeled graphlets of size k. In addition, when sampling labeled graphlets, the sampling complexity will depend on a quantity in $O(n_k|\Sigma|^k)$ instead of n_k, which may result in prohibitive runtimes if we require reasonable guarantees on the quality of the empirical distribution, and if $k > 3$. To sum up, we believe that labeled graphlet comparison is tractable for small alphabets of labels and small k such as $k = 3$. For larger graphlets, it may only be feasible for small label alphabets. As we discussed in Section 1.5.7, nor are other graph kernels able to exploit label information and scale up to large graphs at the same time.

In this chapter, we develop a fast subtree kernel that scales up to large, labeled graphs (Section 3.2). It uses ideas from the Weisfeiler-Lehman test of isomorphism [Weisfeiler and Lehman, 1968], that we introduce in Section 3.1. The computational complexity of our subtree kernel is linear in the number of edges and does not depend on the alphabet size. We study the link of our proposed kernel with the classic subtree kernel by Ramon and Gärtner [2003] in Section 3.2.3. In Sections 3.3 and 3.4 we present a generalization of our subtree kernel that includes all graph kernels that consider node labels, and some of its instances. In Section 3.5 we experimentally compare our new kernel with the state of the art graph kernels. We present two extensions of our fast subtree kernel in Sections 3.6 and 3.7.

3.1. The Weisfeiler-Lehman test of isomorphism

Our graph kernels use concepts from the Weisfeiler-Lehman test of isomorphism [Weisfeiler and Lehman, 1968], more specifically its 1-dimensional variant, also known as "naive vertex refinement". Assume we are given two graphs G and G' and we would like to test whether they are isomorphic. The 1-dimensional Weisfeiler-Lehman test proceeds in iterations, which we index by i and which comprise the steps given in Algorithm 2.

The key idea of the algorithm is to augment the node labels by the sorted set of node labels of neighboring nodes, and compress these augmented labels into new, short labels. These steps are then repeated until the node label sets of G and G' differ, or the number of iterations reaches n. See Figure 3.1, a-d, for an illustration of these steps

(note however, that the two graphs in the figure would directly be identified as non-isomorphic by the Weisfeiler-Lehman test, as their label sets are already different in the beginning).

Sorting the set of multisets allows for a straightforward definition and implementation of f for the compression of labels in step 4: we keep a counter variable for f that records the number of distinct strings that f has compressed before. f assigns the current value of this counter to a string if an identical string has been compressed before, but when we encounter a new string, we increment the counter by one and f assigns its value to the new string. The sorted order of the set of multisets guarantees that all identical strings are mapped to the same number, because they occur in a consecutive block. However, note that the sorting of the set of multisets is not required for defining f. Any other injective mapping will give equivalent results. The alphabet Σ has to be sufficiently large for f to be injective. For two graphs, $|\Sigma| = 2n$ suffices.

The Weisfeiler-Lehman algorithm terminates after step 4 of iteration i if $\{l_i(v)|v \in V\} \neq \{l_i(v')|\ v' \in V'\}$, that is, if the sets of newly created labels are not identical in G and G'. The graphs are then not isomorphic. If the sets are identical after n iterations, it means that either G and G' are isomorphic, or the algorithm has not been able to determine that they are not isomorphic [see Cai et al., 1992, for examples of graphs that cannot be distinguished by this algorithm or its higher-dimensional variants]. As a side note, we mention that the 1-dimensional Weisfeiler-Lehman algorithm has been shown to be a valid isomorphism test for almost all graphs [Babai and Kucera, 1979].

Note that in Algorithm 2 we used the same node labeling functions $\ell, l_0, \ldots, l_h$ for both G and G' in order not to overload the notation. We will continue using this notation throughout this chapter and assume without loss of generality that the domain of these functions $\ell, l_0, \ldots, l_h$ is the set of all nodes in our data set of graphs, which corresponds to $V \cup V'$ in the case of Algorithm 2.

Complexity The runtime complexity of the 1-dimensional Weisfeiler-Lehman algorithm with h iterations is $O(hm)$. Defining the multisets in step 1 for all nodes is an $O(m)$ operation. Sorting each multiset is an $O(m)$ operation for all nodes. This efficiency can be achieved by using counting sort, which is an instance of bucket sort, due to the limited range of the elements of the multiset. The elements of each multiset are a subset of $\{f(s_i(v))|v \in V\}$. For a fixed i, the cardinality of this set is upper-bounded by n, which means that we can sort all multisets in $O(m)$ by the following procedure: We assign the elements of all multisets to their corresponding buckets, recording which multiset they came from. By reading through all buckets in ascending order, we can then extract the sorted multisets for all nodes in a graph (see the pseudocode in Algorithm 3). The runtime is $O(m)$ as there are $O(m)$ elements in the multisets of a graph in iteration i. Sorting the resulting strings is of time complexity $O(m)$ via radix sort [see Mehlhorn, 1984, Vol. 1, Section II.2.1]. The label compression requires one pass over all strings and

Algorithm 2 One iteration of the 1-dim. Weisfeiler-Lehman test of graph isomorphism

1: Multiset-label determination
- If $i = 0$, set $l_0(v) \leftarrow \ell(v)$ [1] and $i \leftarrow 1$.
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G and G' which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

3: Label compression
- Sort all of the strings $s_i(v)$ for all v from G and G' in ascending order.
- Map each string $s_i(v)$ to a new compressed label, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

their characters, that is $O(m)$. Hence all these steps result in a total runtime of $O(hm)$ for h iterations.

Algorithm 3 Sorting each multiset at iteration i

```
1: for all graph G do
2:    for all node u in graph G do
3:       for all node v in N(u) do
4:          append the pair (G, u)[2] to bucket l_{i-1}(v)
5:       end for
6:       s_i(u) ← l_{i-1}(u)
7:    end for
8: end for
9: for k = 1 → |Σ| do
10:   for all (G, v) in bucket k do
11:      append k to s_i(v) in G
12:   end for
13: end for
```

[1] For unlabeled graphs, node labels $l_0(v)$ can be initialized with letters corresponding one to one to node degrees $|\mathcal{N}(v)|$.

[2] By G in the pair (G, u) we only mean the identifier of the graph G in the graph data set, not the whole data structure containing its nodes and edges.

Link with subtree patterns Note that the compressed labels $l_i(v)$ correspond to subtree patterns of height i rooted at v (see Figure 1.3 for an illustration of subtree patterns).

3.2. The Weisfeiler-Lehman subtree kernel

Based on the Weisfeiler-Lehman relabeling, presented in Algorithm 2, we define the Weisfeiler-Lehman subtree kernel below.

Definition 3.2.1 *Let G and G' be graphs. Define $\Sigma_i \subseteq \Sigma$ as the set of letters that occur as node labels at least once in G or G' at the end of the i-th iteration of the Weisfeiler-Lehman algorithm. Let Σ_0 be the set of original node labels of G and G'. Assume all Σ_i are pairwise disjoint. Without loss of generality, assume that every $\Sigma_i = \{\sigma_{i1}, \ldots, \sigma_{i|\Sigma_i|}\}$ is ordered. Define a map $c_i : \{G, G'\} \times \Sigma_i \to \mathbb{N}$ such that $c_i(G, \sigma_{ij})$ is the number of occurrences of the letter σ_{ij} in the graph G.*

The Weisfeiler-Lehman subtree kernel on two graphs G and G' with h iterations is defined as

$$k^{(h)}_{WLsubtree}(G, G') = \langle \phi^{(h)}_{WLsubtree}(G), \phi^{(h)}_{WLsubtree}(G') \rangle, \tag{3.1}$$

where

$$\phi^{(h)}_{WLsubtree}(G) = (c_0(G, \sigma_{01}), \ldots, c_0(G, \sigma_{0|\Sigma_0|}), \ldots, c_h(G, \sigma_{h1}), \ldots, c_h(G, \sigma_{h|\Sigma_h|})),$$

and

$$\phi^{(h)}_{WLsubtree}(G') = (c_0(G', \sigma_{01}), \ldots, c_0(G', \sigma_{0|\Sigma_0|}), \ldots, c_h(G', \sigma_{h1}), \ldots, c_h(G', \sigma_{h|\Sigma_h|})).$$

That is, the Weisfeiler-Lehman subtree kernel counts common *original and compressed labels* in two graphs. See Figure 3.1 for an illustration.

Theorem 3.2.2 *The Weisfeiler-Lehman subtree kernel on a pair of graphs G and G' can be computed in time $O(hm)$.*

Proof This follows directly from the definition of the Weisfeiler-Lehman subtree kernel and the runtime complexity of the Weisfeiler-Lehman test, as described in Section 3.1. ■

While based on subtree patterns that, as random walks, contain repetitions of nodes and edges, the Weisfeiler-Lehman subtree kernel is not prone to tottering thanks to the rich structure of the subtree patterns resulting from the Weisfeiler-Lehman relabeling. Tottering in random walk kernels stems from the following fact: Given a walk of length i in a graph, extending it to a walk of length $i + 1$ may cover the same subgraph that

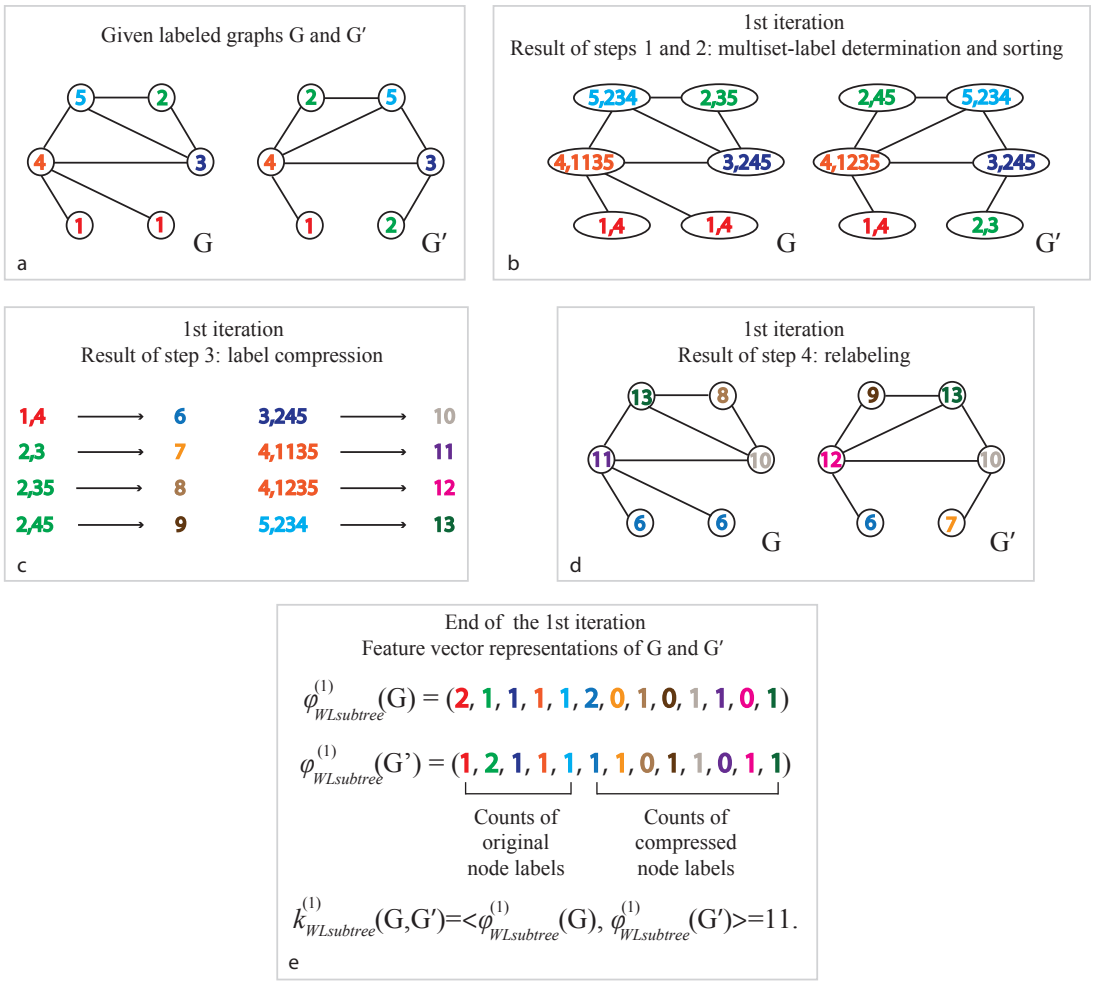

Figure 3.1.: Illustration of the computation of the Weisfeiler-Lehman subtree kernel with $h = 1$ for two graphs. Here $\{1, 2, \ldots, 13\} \in \Sigma$ are considered as letters. Note that compressed labels denote subtree patterns: For instance, if a node has label 8, this means that there is a subtree pattern of height 1 rooted at this node, where the root has label 2 and its neighbors have labels 3 and 5.

was covered by the walk of length i [3]. As the extension of a walk of length i to length $i + 1$ amounts to adding to it at most one node and one edge previously not covered

[3] We say that a walk (or a subtree pattern) covers a subgraph $G' = (V', E')$ of a graph G if V' and E' are the sets of nodes and edges contained in the walk (respectively, in the subtree pattern).

by the walk, the probability of tottering is high when we consider walks. In the case of the Weisfeiler-Lehman subtree kernel, tottering would be equivalent to the equality of subgraphs covered by subtree patterns of heights i and $i+1$ rooted at a node v. As we here extend subtree patterns of height i to height $i+1$ by considering all neighbors of all nodes at distance i from the root, this equality can only occur if all $O(d^{i+1})$ of these nodes are already covered by the subtree pattern of height i (where d is the maximum degree of the graph). This can only happen if the longest path between v and any other node in the connected component containing v is at most i. To summarize, while any walk of any length $i \geq 1$ is likely to totter in any undirected graph with at least one edge, a subtree pattern rooted at a node v will totter only if its height is not smaller than the longest path between v and any other node in the connected component containing v. For reasonable values of i, this is unlikely to happen often in graphs that are not mere collections of small connected components.

3.2.1. Computing the Weisfeiler-Lehman subtree kernel on many graphs

To compute the Weisfeiler-Lehman subtree kernel on N graphs, we propose Algorithm 3.2.1, which improves over the naive, N^2-fold application of the kernel from Definition 3.2.1. We now process all N graphs simultaneously and conduct the steps given in Algorithm 3.2.1 on each graph G in each of the h iterations.

Algorithm 4 One iteration of the Weisfeiler-Lehman subtree kernel computation on N graphs

1: Multiset-label determination
- Assign a multiset-label $M_i(v)$ to each node v in G which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$.

3: Label compression
- Map each string $s_i(v)$ to a compressed label using a hash function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G.

As before, Σ is assumed to be sufficiently large to allow f to be injective. In the case of N graphs and h iterations, a Σ of size $Nn(h+1)$ suffices.

One way of implementing f is to sort all neighborhood strings using radix sort, as done in step 4 in Algorithm 2. The resulting complexity of this step would be linear in the sum of the size of the current alphabet and the total length of strings, that is

$O(Nn + Nm) = O(Nm)$. An alternative implementation of f would be by means of a perfect hash function.

Theorem 3.2.3 *For N graphs, the Weisfeiler-Lehman subtree kernel with h iterations on all pairs of these graphs can be computed in $O(Nhm + N^2hn)$.*

Proof Naive application of the kernel from Definition 3.2.1 for computing an $N \times N$ kernel matrix would require a runtime of $O(N^2hm)$. One can improve upon this runtime complexity by computing $\phi^{(h)}_{WLsubtree}$ explicitly for each graph and only then taking pairwise inner products.

Step 1, the multiset-label determination, still requires $O(Nm)$. Step 2, the sorting of the elements in each multiset, can be done via a joint bucket sort (counting sort) of all strings, requiring $O(Nn + Nm)$ time.

The effort of computing $\phi^{(h)}_{WLsubtree}$ on all N graphs in h iterations is then $O(Nhm)$, assuming that $m > n$. To get all pairwise kernel values, we have to multiply all feature vectors, which requires a runtime of $O(N^2hn)$, as each graph G has at most hn non-zero entries in $\phi^{(h)}_{WLsubtree}(G)$. In Section 3.5.1, we empirically show that the first term Nhm dominates the overall runtime in practice. ■

While our Weisfeiler-Lehman subtree kernel matches neighborhoods of nodes in a graph exactly, one could also think of other strategies of comparing node neighborhoods, and still retain the favourable runtime of our graph kernel. In research that was published in parallel to ours, Hido and Kashima [2009] present such an alternative kernel based on node neighborhoods, which uses hash functions and logical operations on bit-representations of node labels and which also scales linearly in the number of edges. The Morgan index [Morgan, 1965] is another way of summarizing information contained in the neighborhood of a node, and has been used by Mahé et al. [2004] in the context of graph kernels.

3.2.2. The Ramon-Gärtner subtree kernel

The first subtree kernel on graphs was defined by Ramon and Gärtner [2003]. The Ramon-Gärtner subtree kernel with subtree height h compares all pairs of nodes from graphs $G = (V, E, \ell)$ and $G' = (V', E', \ell)$ by iteratively comparing their neighborhoods:

$$k^{(h)}_{RG}(G, G') = \sum_{v \in V} \sum_{v' \in V'} k_{RG,h}(v, v'),$$

where

$$k_{RG,h}(v, v') = \begin{cases} \delta(\ell(v), \ell(v')), & \text{if } h = 0 \\ \lambda_v \lambda_{v'} \delta(\ell(v), \ell(v')) \sum_{R \in \mathcal{M}(v,v')} \prod_{(w,w') \in R} k_{RG,h-1}(w, w'), & \text{if } h > 0, \end{cases}$$

δ is an indicator function that equals 1 if its arguments are equal, 0 otherwise, λ_v and $\lambda_{v'}$ are weights associated with nodes v and v', and

$$\mathcal{M}(v,v') = \Big\{ R \subseteq \mathcal{N}(v) \times \mathcal{N}(v') \Big| (\forall (u,u'),(w,w') \in R : u = w \Leftrightarrow u' = w') \\ \wedge (\forall (u,u') \in R : \ell(u) = \ell(u')) \Big\}. \tag{3.2}$$

Said differently, $\mathcal{M}(v,v')$ is the set of exact matchings of subsets of the neighborhoods of v and v'. Each element R of $\mathcal{M}(v,v')$ is a set of pairs of nodes from the neighborhoods of $v \in V$ and $v' \in V'$ such that nodes in each pair have identical labels and no node is contained in more than one pair. Thus, intuitively, k_{RG} iteratively considers all matchings $\mathcal{M}(v,v')$ between neighbors of two identically labeled nodes v from G and v' from G'. Taking the parameters λ_v and $\lambda_{v'}$ equal to a single parameter λ results in weighting each pattern by λ raised to the power of the number of nodes in the pattern.

The runtime complexity of the subtree kernel for a pair of graphs is $O(n^2 h 4^d)$, including a comparison of all pairs of nodes (n^2), and a pairwise comparison of all matchings in their neighborhoods in $O(4^d)$, which is repeated in h iterations. h is a multiplicative factor, not an exponent, since one can implement the subtree kernel via dynamic programming, starting with k_1 and computing k_h from k_{h-1}. For a data set of N graphs, the resulting runtime complexity is then in $O(N^2 n^2 h 4^d)$.

3.2.3. Link to the Weisfeiler-Lehman subtree kernel

The Weisfeiler-Lehman subtree kernel can be defined in a recursive fashion which elucidates its relation to the Ramon-Gärtner kernel.

Theorem 3.2.4 *The kernel $k_{rec}^{(h)}$ defined as*

$$k_{rec}^{(h)}(G,G') = \sum_{i=0}^{h} \sum_{v \in V} \sum_{v' \in V'} k_{rec,i}(v,v'), \tag{3.3}$$

where

$$k_{rec,i}(v,v') = \\ \begin{cases} \delta(\ell(v),\ell(v')), & \text{if } i = 0 \\ k_{rec,i-1}(v,v') \max_{R \in \mathcal{M}(v,v')} \prod_{(w,w') \in R} k_{rec,i-1}(w,w'), & \text{if } i > 0 \text{ and } \mathcal{M} \neq \emptyset \\ 0, & \text{if } i > 0 \text{ and } \mathcal{M} = \emptyset, \end{cases} \tag{3.4}$$

δ is the indicator function again, and

$$\mathcal{M}(v,v') = \Big\{ R \subseteq \mathcal{N}(v) \times \mathcal{N}(v') \Big| \; |R| = |\mathcal{N}(v)| = |\mathcal{N}(v')| \\ \wedge (\forall (u,u'),(w,w') \in R : u = w \Leftrightarrow u' = w') \wedge (\forall (u,u') \in R : \ell(u) = \ell(u')) \Big\}, \tag{3.5}$$

is equivalent to the Weisfeiler-Lehman subtree kernel $k^{(h)}_{WLsubtree}$.

In other words, $\mathcal{M}(v, v')$ is the set of exact matchings of the neighborhoods of v and v'. It is nonempty only in the case where the neighborhoods of v and v' have exactly the same size and the multisets of labels of their neighbors $\{\ell(u)|u \in \mathcal{N}(v)\}$ and $\{\ell(u')|u' \in \mathcal{N}(v')\}$ are identical. Note that $k_{rec,i}(v, v')$ only takes binary values: it evaluates to 1 if the subtree patterns of height i rooted at v and v' are identical, and to 0 otherwise.

Proof We prove this theorem by induction over h.

Induction initialisation $h = 0$:

$$k^{(0)}_{WLsubtree} = \langle \phi^{(0)}_{WLsubtree}(G), \phi^{(0)}_{WLsubtree}(G) \rangle = \sum_{j=1}^{|\Sigma_0|} c_0(G, \sigma_{0j}) c_0(G', \sigma_{0j}) =$$

$$= \sum_{v \in V} \sum_{v' \in V'} \delta(\ell(v), \ell(v')) = k^{(0)}_{rec},$$

where Σ_0 is the initial alphabet of node labels and $c_0(G, \sigma_{0j})$ is the number of occurrences of the letter σ_{0j} as a node label in G. The equality follows from the definitions of $k^{(h)}_{rec}$ and $k^{(h)}_{WLsubtree}$.

Induction step $h \to h + 1$: Assume that $k^{(h)}_{WLsubtree} = k^{(h)}_{rec}$. Then

$$k^{(h+1)}_{rec} = \sum_{v \in V} \sum_{v' \in V'} k_{rec,h+1}(v, v') + \sum_{i=0}^{h} \sum_{v \in V} \sum_{v' \in V'} k_{rec,i}(v, v') = \tag{3.6}$$

$$= \sum_{j=1}^{|\Sigma_{h+1}|} c_{h+1}(G, \sigma_{h+1,j}) c_{h+1}(G', \sigma_{h+1,j}) + k^{(h)}_{WLsubtree} = k^{(h+1)}_{WLsubtree}, \tag{3.7}$$

where the equality of (3.6) and (3.7) follows from the fact that $k_{rec,h+1}(v, v') = 1$ if and only if the labels and neighborhoods of v and v' are identical, that is, if $f(s_{h+1}(v)) = f(s_{h+1}(v'))$. ■

Theorem 3.2.4 highlights the following differences between the Weisfeiler-Lehman and the Ramon-Gärtner subtree kernels: In Equation (3.3), Weisfeiler-Lehman considers all subtrees up to height h, whereas the Ramon-Gärtner kernel looks at subtrees of exactly height h. In Equations (3.4) and (3.5), the Weisfeiler-Lehman subtree kernel checks whether the neighborhoods of v and v' match exactly, while the Ramon-Gärtner kernel considers all pairs of matching subsets of the neighborhoods of v and v' in Equation (3.2). In our experiments, we examine the empirical differences between these two kernels in terms of runtime and prediction accuracy on classification benchmark data sets (Section 3.5.2).

3.3. The general Weisfeiler-Lehman kernels

In this section, we define the Weisfeiler-Lehman graph sequence and the general graph kernels based on them. We then (in Section 3.4) prove that this kernel generalizes the Weisfeiler-Lehman subtree kernel (Section 3.2), and present two more instances of this kernel, the Weisfeiler-Lehman edge kernel (Section 3.4.2), and the Weisfeiler-Lehman shortest path kernel (Section 3.4.3).

In each iteration i of the Weisfeiler-Lehman algorithm (see Algorithm 2), we get a new labeling $l_i(v)$ for all nodes v. Recall that this labeling is concordant in G and G', meaning that if nodes in G and G' have identical multiset labels, and only in this case, they will get identical new labels. Therefore, we can imagine one iteration of Weisfeiler-Lehman relabeling as a function $r((V, E, l_i)) = (V, E, l_{i+1})$ that transforms all graphs in the same manner. Note that r depends on the set of graphs that we consider.

Definition 3.3.1 *Define the* Weisfeiler-Lehman graph *at height i of the graph $G = (V, E, \ell) = (V, E, l_0)$ as the graph $G_i = (V, E, l_i)$. We call the sequence of Weisfeiler-Lehman graphs*

$$\{G_0, G_1, \ldots, G_h\} = \{(V, E, l_0), (V, E, l_1), \ldots, (V, E, l_h)\},$$

where $G_0 = G$ and $l_0 = \ell$, the Weisfeiler-Lehman sequence *up to height h of G.*

G_0 is the original graph, $G_1 = r(G_0)$ is the graph resulting from the first relabeling, and so on. Note that neither V, nor E ever change in this sequence, but we define it as a sequence of graphs rather than a sequence of labeling functions for the sake of clarity of definitions that follow.

Definition 3.3.2 *Let k be any kernel for graphs, that we call the* base kernel. *Then the Weisfeiler-Lehman kernel with h iterations with the base kernel k is defined as*

$$k_{WL}^{(h)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \ldots + k(G_h, G'_h), \tag{3.8}$$

where h is the number of Weisfeiler-Lehman iterations and $\{G_0, \ldots, G_h\}$ and $\{G'_0, \ldots, G'_h\}$ are the Weisfeiler-Lehman sequences of G and G' respectively.

Theorem 3.3.3 *Let the base kernel k be any positive semidefinite kernel on graphs. Then the corresponding Weisfeiler-Lehman kernel $k_{WL}^{(h)}$ is positive semidefinite.*

Proof Let ϕ be the feature mapping corresponding to the kernel k:

$$k(G_i, G'_i) = \langle \phi(G_i), \phi(G'_i) \rangle.$$

We have

$$k(G_i, G'_i) = k(r^i(G), r^i(G')) = \langle \phi(r^i(G)), \phi(r^i(G')) \rangle.$$

Let us define the feature mapping $\psi(G)$ as $\phi(r^i(G))$. Then we have

$$k(G_i, G_i') = \langle \psi(G), \psi(G') \rangle,$$

hence k is a kernel on G and G' and $k_{WL}^{(h)}$ is positive semidefinite as a sum of positive semidefinite kernels.

■

This definition provides a framework for applying all graph kernels that take into account categorical node labels to different levels of node-labeling of graphs, from the original labeling to more and more fine-grained labelings for growing h. This enriches the set of extracted features. For example, while the shortest path kernel compares shortest path lengths between identically labeled source and sink nodes on the original graphs, it will compare shortest path lengths between the roots of identical subtree patterns of height 1 on Weisfeiler-Lehman graphs with $h = 1$.

For some base kernels one may be able to exploit the fact that the graph structure does not change over the Weisfeiler-Lehman sequence to do some computations only once instead of repeating it h times. One example of such a base kernel is the shortest path kernel: As shortest path lengths in a graph G are the same as shortest path lengths in corresponding Weisfeiler-Lehman graphs G_i, we can precompute them. One should bear in mind that for graph kernels k that depend on the size of the alphabet of node labels, computing $k(G_i, G_i')$ will accordingly get increasingly expensive, or, in some cases, cheaper, as a function of growing i.

Note that it is possible to put nonnegative real weights α_i on $k(G_i, G_i')$, $i = \{0, 1, \ldots, h\}$, to obtain a more general definition of the Weisfeiler-Lehman kernel:

$$k_{WL}^{(h)}(G, G') = \alpha_0 k(G_0, G_0') + \alpha_1 k(G_1, G_1') + \ldots + \alpha_h k(G_h, G_h').$$

In this case, $k_{WL}^{(h)}$ will still be positive semidefinite, as a positive linear combination of positive semidefinite kernels.

Note on computing Weisfeiler-Lehman kernels in practice In the inductive learning setting, we compute the kernel on the training set of graphs. For any test graph that we subsequently need to classify, we have to map it to the feature space spanned by original and compressed labels occurred in the training set. For this purpose, we will need to maintain record of the data structures that hold the mappings $l_i(v) := f(s_i(v))$ for each iteration i and each distinct $s_i(v)$. This requires $O(Nmh)$ memory in the worst case.

In contrast, in the transductive setting, where the test set is already known, we can compute the kernel matrix on the whole data set (training and test set) without having to keep the mappings mentioned above.

3.4. Special cases of the Weisfeiler-Lehman kernel

In this section, we present instances of the general Weisfeiler-Lehman kernel.

3.4.1. The Weisfeiler-Lehman subtree kernel

The following theorem shows that the Weisfeiler-Lehman subtree kernel (3.1) from Section 3.2 is indeed a special case of the general Weisfeiler-Lehman kernel (3.8).

Theorem 3.4.1 *Let the base kernel k be a function counting pairs of matching node labels in two graphs:*

$$k(G, G') = \sum_{v \in V} \sum_{v' \in V'} \delta(\ell(v), \ell(v')),$$

where δ is the Dirac kernel, that is, it is 1 when its arguments are equal and 0 otherwise. Then $k^{(h)}_{WL}(G, G') = k^{(h)}_{WLsubtree}(G, G')$ for all G, G'.

Proof It is easy to notice that for each $i \in \{0, 1, \ldots, h\}$ we have

$$\sum_{v \in V} \sum_{v' \in V'} \delta(l_i(v), l'_i(v')) = \sum_{j=1}^{|\Sigma_i|} c_i(G, \sigma_{ij}) c_i(G', \sigma_{ij}).$$

Adding up these sums for all $i \in \{0, 1, \ldots, h\}$ gives us $k^{(h)}_{WL}(G, G') = k^{(h)}_{WLsubtree}(G, G')$. ■

3.4.2. The Weisfeiler-Lehman edge kernel

The Weisfeiler-Lehman edge kernel is another instance of the Weisfeiler-Lehman kernel framework. In the case of graphs with unweighted edges, we consider the base kernel that counts matching pairs of edges with identically labeled endpoints (incident nodes) in two graphs. In other words, the base kernel is defined as

$$k_E = \langle \phi_E(G), \phi_E(G') \rangle,$$

where $\phi_E(G)$ is a vector of numbers of occurrences of pairs (a, b), $a, b \in \Sigma$, which represent ordered labels of endpoints of an edge in G. Denoting (a, b) and (a', b') the ordered labels of endpoints of edges e and e' respectively, and δ the Dirac kernel, k_E can equivalently be expressed as $\sum_{e \in E} \sum_{e' \in E'} \delta(a, a')\delta(b, b')$. If the edges are weighted by a function w that assigns weights, the base kernel k_E can be defined as $\sum_{e \in E} \sum_{e' \in E'} \delta(a, a')\delta(b, b')k_w(w(e), w(e'))$, where k_w is a kernel comparing edge weights.

Following (3.8), we have

$$k^{(h)}_{WL\ edge} = k_E(G_0, G'_0) + k_E(G_1, G'_1) + \ldots + k_E(G_h, G'_h).$$

Note on computational complexity If the edges are not weighted or labeled, the number of possible edge features in each iteration equals the number of distinct ordered pairs (a, b), that is, $\frac{|\Sigma_i|(|\Sigma_i|+1)}{2}$. It is easy to notice by looking at the Algorithm 2 that for each $i \in \{0, \ldots, h-1\}$, we have $|\Sigma_i| \leq |\Sigma_{i+1}|$. Therefore, if we compute the edge kernel by first explicitly computing $\phi_E(G)$ for each G in the data set, the computation will become increasingly expensive in each iteration i of the Weisfeiler-Lehman relabeling.

If edges are weighted and we use any general kernel to compare their weights, computing the feature map explicitly may not be possible or practical any more. In this case, the kernel can be computed by comparing edges pairwise in each pair of graphs. Assuming that the kernel on a pair of weights can be computed in $O(1)$, this results in $O(N^2m^2)$ operations per Weisfeiler-Lehman iteration.

Computing the feature map explicitly can also become problematic if the alphabet size gets prohibitively large. In this case, one can either compute the kernel via pairwise comparisons of edges in each pair of graphs as above ($O(N^2m^2)$ per iteration), or via the construction of the explicit feature map for each pair of graphs separately, potentially yielding smaller alphabets Σ_i than considering the whole data set of N graphs at once.

3.4.3. The Weisfeiler-Lehman shortest path kernel

Another example of the general Weisfeiler-Lehman kernels that we consider is the Weisfeiler-Lehman shortest path kernel. Here we use a node-labeled shortest path kernel [Borgwardt and Kriegel, 2005] as the base kernel.

In the particular case of graphs with unweighted edges, we consider the base kernel k_{SP} of the form $k_{SP}(G, G') = \langle \phi_{SP}(G), \phi_{SP}(G') \rangle$, where $\phi_{SP}(G)$ (resp. $\phi_{SP}(G')$) is a vector whose components are numbers of occurrences of triplets of the form (a, b, p) in G (resp. G'), where $a, b \in \Sigma$ are ordered endpoint labels of a shortest path and $p \in \mathbb{N}_0$ is the shortest path length.

According to (3.8), we have

$$k^{(h)}_{WL\ shortest\ path} = k_{SP}(G_0, G'_0) + k_{SP}(G_1, G'_1) + \ldots + k_{SP}(G_h, G'_h).$$

Note on computational complexity Computing shortest paths between all pairs of nodes in a graph can be done in $O(n^3)$ using the Floyd-Warshall algorithm. Consequently, for N graphs, the complexity is of $O(Nn^3)$. This step does not have to be repeated for every Weisfeiler-Lehman iteration, as the topology of a graph does not change along the Weisfeiler-Lehman sequence. In case edges are not weighted, shortest paths are determined in terms of geodesic distance and path lengths are integers. Denote the number of distinct shortest path lengths occurring in the data set of graphs as P.

Let us first consider the Dirac (δ) kernel on the shortest path lengths, which means that the similarity of two paths in two graphs equals 1 if they have exactly the same length and identically labeled endpoints and 0 otherwise. Then, in iteration i of the

Weisfeiler-Lehman relabeling, we can bound the number of features, triplets (a, b, p) where $a, b \in |\Sigma_i|$ are ordered start and end node labels and $p \in \mathbb{N}_0$ the shortest path length, by $\frac{|\Sigma_i|(|\Sigma_i|+1)}{2}P$. As $|\Sigma_i| \leq |\Sigma_{i+1}|$ for each $i \in \{0, \ldots, h-1\}$, if we compute the shortest path kernel by first explicitly computing $\phi_{SP}(G)$ for each G in the data set, the computation will get increasingly expensive in each iteration, as in the case of edge kernels (Section 3.4.2).

Similarly to the Weisfeiler-Lehman edge kernel, in a more general setting where we do not assume that edges are unweighted and use any kernel (not necessarily the Dirac kernel) on shortest path lengths, or if the alphabet size gets prohibitively large, computing the feature map explicitly may become impossible or difficult. In this case, we can compute the kernel by comparing shortest path lengths pairwise in two graphs. Therefore, the runtime of computing $k_{SP}(G_i, G'_i)$ will not depend on i any more. It will scale as $O(n^4)$ for each pair of graphs as we have to compare all pairs of the $O(n^2)$ shortest path lengths, and $O(N^2 n^4)$ for the whole data set.

3.4.4. Other Weisfeiler-Lehman kernels

In a similar fashion, we can plug other base graph kernels into our Weisfeiler-Lehman graph kernel framework. As node labels are the only aspect that differentiate Weisfeiler-Lehman graphs at different *resolutions* (determined by the number of iterations), a clear requirement that the base kernel has to satisfy for the Weisfeiler-Lehman kernel to make sense is to exploit the labels on nodes. A non-exhaustive list of possible base kernels not mentioned in previous sections includes the labeled version of the graphlet kernel [Shervashidze et al., 2009], the random walk kernel [Gärtner et al., 2003, Vishwanathan et al., 2010], and the subtree kernel by Ramon and Gärtner [2003].

3.5. Experiments

In this section, we first empirically study the runtime behavior of the Weisfeiler-Lehman subtree kernel on synthetic graphs (Section 3.5.1). Next, we compare the Weisfeiler-Lehman subtree kernel, the Weisfeiler-Lehman edge kernel, and the Weisfeiler-Lehman shortest path kernel to state-of-the-art graph kernels in terms of kernel computation runtime and classification accuracy on graph benchmark data sets (Section 3.5.2).

3.5.1. Runtime behavior of Weisfeiler-Lehman subtree kernel

Methods We empirically compared the runtime behavior of our two variants of the Weisfeiler-Lehman subtree (WL) kernel. The first variant computes kernel values pairwise in $O(N^2hm)$. The second variant computes the kernel values in $O(Nhm + N^2hn)$ on the data set simultaneously. We will refer to the former variant as the "pairwise" WL, and the latter as "global" WL.

Experimental setup We assessed the behavior on randomly generated graphs with respect to four parameters: data set size N, graph size n, subtree height h and graph density c. The density of an undirected graph of n nodes without self-loops is defined as the number of its edges divided by $n(n-1)/2$, the maximal number of edges. We kept 3 out of 4 parameters fixed at their default values and varied the fourth parameter. The default values we used were 10 for N, 100 for n, 4 for h and 0.4 for the graph density c. In more detail, we varied N in range $\{10, 100, 1000\}$, n in $\{100, 200, \ldots, 1000\}$, h in $\{2, 4, 8\}$ and c in $\{0.1, 0.2, \ldots, 0.9\}$.

For each individual experiment, we generated N graphs with n nodes, and inserted edges randomly until the number of edges reached $\lfloor cn(n-1)/2 \rfloor$. We then computed the pairwise and the global WL kernel on these synthetic graphs. We report CPU runtimes in seconds in Figure 3.2, as measured in Matlab R2008a on an Apple MacPro with 3.0GHz Intel 8-Core with 16GB RAM.

Results Empirically, we observe that the pairwise kernel scales quadratically with data set size N. Interestingly, the global kernel scales linearly with N for the considered range of N. The N^2 sparse vector multiplications that have to be performed for kernel computation with global WL do not dominate runtime here. This result on synthetic data indicates that the global WL kernel has attractive scalability properties for large data sets.

When varying the number of nodes n per graph, we observe that the runtime of both WL kernels scales quadratically with n, and the global WL is much faster than the pairwise WL for large graphs. This agrees with the fact that our kernels scale linearly with the number of edges per graph, m, which is $0.4\frac{n(n-1)}{2}$ in this experiment.

We observe a different picture for the height h of the subtree patterns. The runtime of both kernels grows linearly with h, but the global WL is more efficient in terms of runtime.

Varying the graph density c, both methods show again a linearly increasing runtime, although the runtime of the global WL kernel is much lower than the runtime of the pairwise WL.

Across all different graph properties, the global WL kernel from Section 3.2.1 requires less runtime than the pairwise WL kernel from Section 3.2. Hence the global WL kernel is the variant of our Weisfeiler-Lehman subtree kernel that we use on the following graph classification tasks.

3.5.2. Graph classification

We compared the performance of the WL subtree kernel, the WL edge kernel and the WL shortest path kernel to several other state-of-the-art graph kernels in terms of runtime and classification accuracy on graph benchmark data sets.

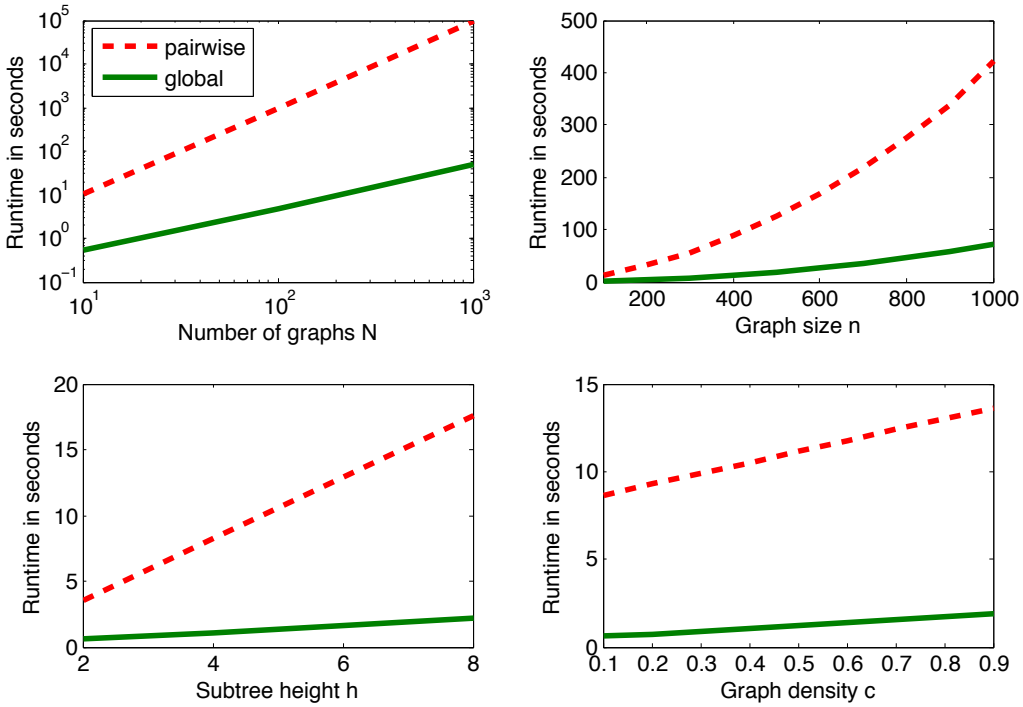

Figure 3.2.: Runtime in seconds for kernel matrix computation on synthetic graphs using the pairwise (red, dashed) and the global (green, solid) computation schemes for the Weisfeiler-Lehman subtree kernel (Default values: data set size $N = 10$, graph size $n = 100$, subtree height $h = 4$, graph density $c = 0.4$).

Data sets We employed the following data sets in our experiments: MUTAG, NCI1, NCI109, ENZYMES and D&D. MUTAG [Debnath et al., 1991] is a data set of 188 mutagenic aromatic and heteroaromatic nitro compounds labeled according to whether or not they have a mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium*. NCI1 and NCI109 represent two balanced subsets of data sets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively [Wale and Karypis, 2006, and `http://pubchem.ncbi.nlm.nih.gov`]. ENZYMES is a data set of protein tertiary structures obtained from [Borgwardt et al., 2005] consisting of 600 enzymes from the BRENDA enzyme database [Schomburg et al., 2004]. In this case the task is to correctly assign each enzyme to one of the 6 EC top-level classes. D&D is a data set of 1178 protein structures [Dobson and Doig, 2003]. Each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 ångströms apart. The prediction task is to classify the protein structures into enzymes and non-enzymes. Note that nodes are labeled in all data sets. Figure 3.3 shows the distributions of node numbers, edge numbers, and degrees in these data sets.

All of these data sets, as well as Matlab scripts for computing kernels used in our experiments, can be downloaded from `http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/WL/`.

Experimental setup On these data sets, we compared our Weisfeiler-Lehman subtree, Weisfeiler-Lehman edge, and Weisfeiler-Lehman shortest path kernels to the Ramon-Gärtner kernel ($\lambda = 1$), as well as to several state-of-the-art graph kernels for large graphs. Due to the large number of graph kernels in the literature, we could not compare to every single graph kernel, but to representative instances of the major families of graph kernels.

From the family of kernels based on walks, we compared our new kernels to the fast geometric random walk kernel by Vishwanathan et al. [2010] that counts common labeled walks, and to the p-random walk kernel that compares random walks up to length p in two graphs (a special case of random walk kernels [Kashima et al., 2003, Gärtner et al., 2003]).

From the family of kernels based on limited-size subgraphs, we chose an extension of the graphlet kernel from Chapter 2 that counts common induced labeled connected subgraphs of size 3.

From the family of kernels based on paths, we compared to the shortest path kernel by Borgwardt and Kriegel [2005] that counts pairs of labeled nodes with identical shortest path length.

Note that whenever possible, we used fast computation schemes based on explicitly computing the feature map (similar to that in Algorithm 3.2.1) before taking the inner product, in order to speed up kernel computation. In particular, we used this technique

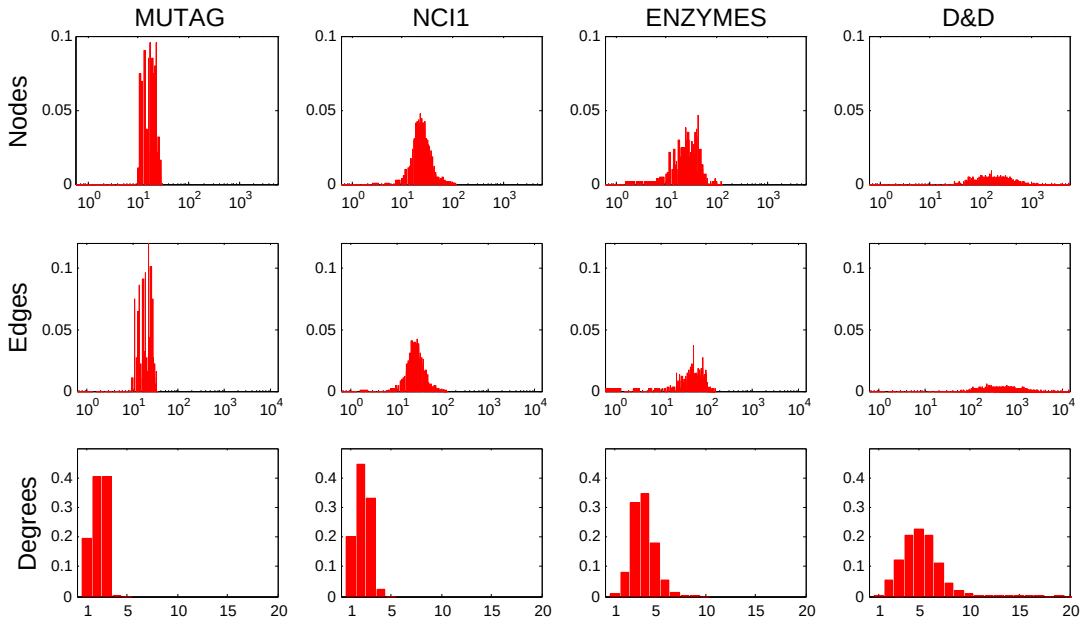

Figure 3.3.: The rows illustrate the distributions of node number, edge number, and degree in data sets MUTAG, NCI1, ENZYMES and D&D. We omitted NCI109, as its node number, edge number, and degree distributions are similar to those of NCI1.

for computing shortest path and graphlet kernels. For connected 3-node graphlet kernels it is rather intuitive to imagine the explicit feature map: First, we have only 4 types of different graphlets with 3 nodes. Second, for each type of graphlet we can determine the number of possible labelings of the three nodes as a function of the size of the node label alphabet. In the case of the shortest path kernel, the explicit feature map may or may not exist. In our experiments, as edges were not weighted, we used the number of edges in a path as a measure of its length. Moreover, we used the Dirac kernel on shortest path distances. This allowed us to explicitly compute the feature map corresponding to the shortest path kernel for each graph in all data sets. We were able to compute the explicit feature maps corresponding to the WL edge and WL shortest path up to and including $h = 3$ and $h = 2$ respectively on all data sets except the largest one, D&D (which also has the largest original node label alphabet), because of the large number of compressed labels. In the case of this data set, we used the pairwise edge (resp. shortest path) comparison scheme described in Sections 3.4.2 and 3.4.3.

We performed 10-fold cross-validation of C-Support Vector Machine Classification using LIBSVM [Chang and Lin, 2001], using 9 folds for training and 1 for testing. All parameters of the SVM were optimized on the training data set only. To exclude random effects of fold assignments, we repeated the whole experiment 10 times. We report average prediction accuracies and standard deviations in Figure 3.4, (a).

We chose h for our Weisfeiler-Lehman subtree kernel by cross-validation on the training data set for $h \in \{0, 1, \ldots, 10\}$, which means that we computed 11 different WL subtree kernel matrices in each experiment. In the case of the WL edge and WL shortest path kernels, h was chosen by cross-validation for $h \in \{0, 1, 2, 3\}$ and $h \in \{0, 1, 2\}$ respectively. We reported the total runtime of these computations (*not* the average per kernel matrix).

Note that all kernel matrices in Figure 3.4, (b) which needed more than 3 days to be computed on one machine were computed on a cluster by distributing different blocks of the kernel matrix to be computed to different nodes. The reported runtime is the sum of the runtimes required to obtain each block.

Proceeding in the same fashion as in the case of the Weisfeiler-Lehman subtree kernel, we computed the Ramon-Gärtner subtree and Weisfeiler-Lehman shortest path kernels for $h \in \{0, 1, 2\}$ and the p-random walk kernel for $p \in \{1, \ldots, 10\}$. We computed the random walk kernel for λ chosen from the set $\{10^{-2}, 10^{-3}, \ldots, 10^{-6}\}$ for smaller data sets and did not observe a large variation in the resulting accuracy. For this reason and because of the relatively high runtime needed to compute this kernel on larger data sets (see Figure 3.4, (b)), we set λ as the largest power of 10 smaller than the inverse of the squared maximum degree in the data set.

Results In terms of runtime, the Weisfeiler-Lehman subtree kernel could easily scale up even to graphs with thousands of nodes. On D&D, subtree-patterns of height up to 10 were computed in 11 minutes, while no other comparison method could handle this data set in less than half an hour. The shortest path kernel, the WL edge kernel and the WL shortest path kernel were competitive to the WL subtree kernel on smaller graphs (MUTAG, NCI1, NCI109, ENZYMES), but on D&D their runtime degenerated to more than 23 hours for the shortest path kernel, to 3 days for the WL edge kernel, and to more than a year for the WL shortest path kernel. The Ramon and Gärtner kernel was computable on MUTAG in approximately 40 minutes, but it finished computation in more than a month on ENZYMES and the computation took even longer time on larger data sets. The random walk kernel was competitive on MUTAG and ENZYMES in terms of runtime, but took more than a week on each of the NCI data sets and more than a month on D&D. The fact that the random walk kernel was competitive on the smallest of our data sets, MUTAG, is not surprising, as on this data set one could also afford using kernels with exponential runtime, such as the all paths kernel [Gärtner et al., 2003]. The graphlet kernel was faster than our WL subtree kernel on MUTAG and the NCI data sets, and about a factor of 3 slower on D&D. However, this efficiency came at

a price, as the kernel based on 3-graphlets turned out to lead to poor accuracy levels on four data sets.

On NCI1, NCI109, ENZYMES and D&D, the kernels from the Weisfeiler-Lehman framework reached the highest accuracy. While on NCI1, NCI109, and D&D the results of all three WL kernels were competitive with each other, on ENZYMES the WL shortest path kernel dramatically improved over the other two WL kernels. On D&D the shortest path and graphlet kernels yielded similarly good results, while on NCI1 and NCI109 the Weisfeiler-Lehman subtree kernel improved by more than 8% the best accuracy attained by other methods. On MUTAG, the WL kernels reached the third, the fourth and the fifth best accuracy levels among all methods considered.

The labeled 3-graphlet kernel achieved low accuracy levels, except on D&D. The random walk and the p-random walk kernels, as well as the Ramon-Gärtner kernel, were less competitive to kernels that performed the best on data sets other than MUTAG.

It is worth mentioning that in the case of WL edge and WL shortest path kernels, the values 2 and 3 of h were almost always chosen by the cross-validation procedure, meaning that the kernels comparing edges and shortest paths on Weisfeiler-Lehman graphs of positive height systematically improved the accuracy of the base kernel (corresponding to $h = 0$).

To summarize, the WL subtree kernel turned out to be competitive in terms of runtime on all smaller data sets, fastest on the large protein data set, and its accuracy levels were competitive on all data sets. The WL edge kernel performed slightly better than the WL subtree kernel on three out of five data sets in terms of accuracy. The WL shortest path kernel achieved the highest accuracy level on two out of five data sets, and was competitive on the remaining data sets.

3.6. Weisfeiler-Lehman kernels with approximate matching of subgraphs

There is a central assumption in the Weisfeiler-Lehman subtree kernel which may affect its performance in certain applications: the concept of *exact matching* of node neighborhoods.

To explain what we mean by exact matching, consider Figure 3.1, b,c. Let us take the example of the multiset-labels 4,1135 and 4,1235: Intuitively, they are not completely dissimilar, as they represent subtree patterns that only differ by one leaf node, the other three and the root being identical. However, in Figure 3.1, c, 4,1135 and 4,1235 are relabeled to different short labels, 11 and 12, which means that the similarity between 4,1135 and 4,1235 will be forgotten and never be taken into account. Only identical subtree patterns in G and G' contribute to the similarity score of G and G'.

In application domains where edges and nodes and their labels are noisy or false measurements occur, hardly any neighborhoods may match exactly. Therefore it seems

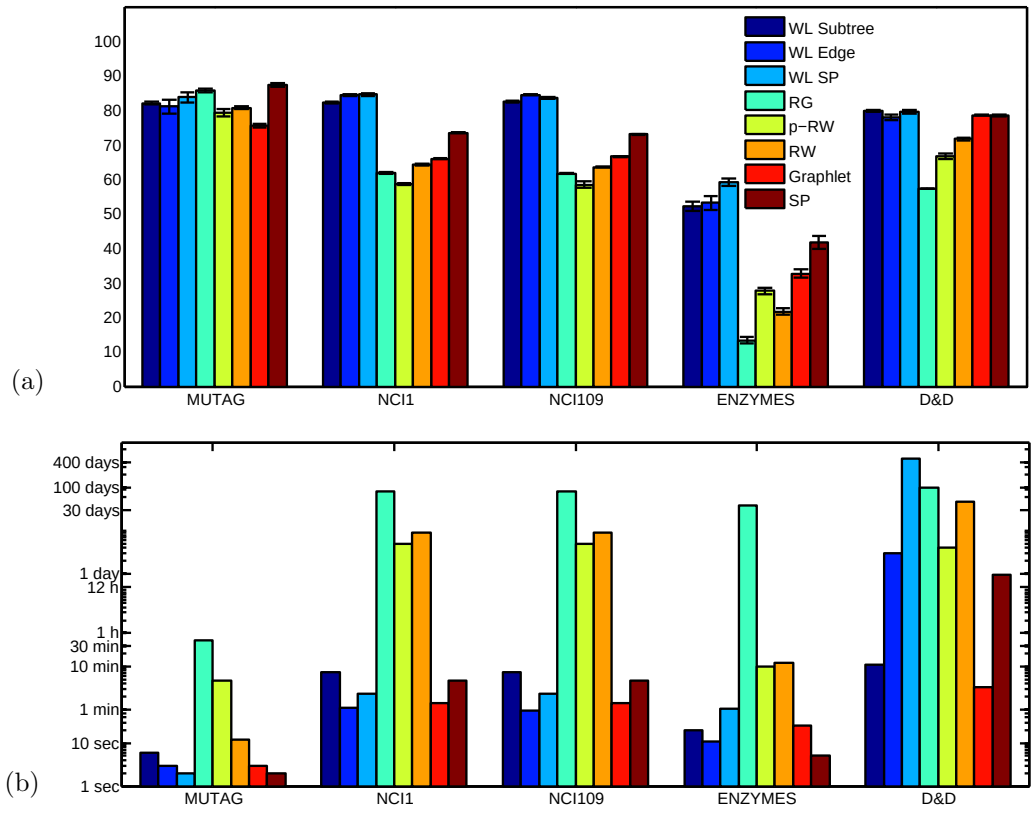

Figure 3.4.: (a) Prediction accuracy (± standard deviation) on graph classification benchmark data sets.
(b) CPU runtime for kernel computation on graph classification benchmark data sets.

attractive to define a variant of the Weisfeiler-Lehman subtree kernel that uses approximate matching of subtree patterns.

In this section, we seek to define an efficient and scalable subtree kernel that allows approximate matching of neighborhoods. One way to perform approximate matching is to compute similarities between multisets of labels of neighbors, $\{l_i(u)|u \in \mathcal{N}(v)\}$ and $\{l_i(u')|u' \in \mathcal{N}(v')\}$ for each $v \in G$ and $v' \in G'$ and each G and G', before compressing the multiset-labels. The next section presents a measure of similarity for sets, the *Jaccard coefficient*, and a way of computing it efficiently using permutations of the alphabet of labels.

3.6.1. Efficiently comparing sets and multisets

Jaccard coefficient The Jaccard coefficient [Gower, 1971] is a natural measure of similarity for sets and is defined as follows: Given two sets A and B,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{3.9}$$

that is, it represents the cardinality of the intersection of the two sets over the cardinality of the union of the two sets (see Figure 3.5 for illustration). $J(A, B)$ is a positive

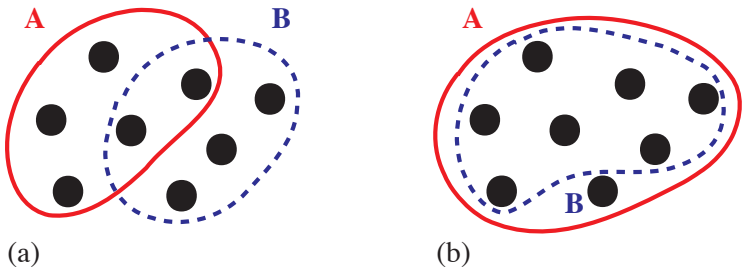

Figure 3.5.: The Jaccard coefficient for two examples of sets A and B.
(a): $J(A, B) = \frac{2}{8} = 0.25$, (b): $J(A, B) = \frac{7}{8} = 0.875$

semidefinite kernel, as shown in a classic result by Gower [1971]. It has been generalized into the so-called MinMax kernel for comparing molecular compounds by Ralaivola et al. [2005].

Shingling To compute the Jaccard coefficient one needs to read all elements in A and B once (if they are sorted). This single pass can become prohibitively expensive if A and B are of large cardinality. To avoid this pass over all elements in A and B, we adopt an approximation strategy following the *shingling* algorithm given by Gibson et al. [2005].

Shingling was originally developed to measure the similarity of web pages [Broder et al., 1997, Broder, 1998] and takes its name from a feature extraction technique based on overlapping windows of words (as windows of words overlap like shingles on a roof). While the aspect of the algorithm that we are interested in does not have anything to do with this feature extraction scheme, we still use the name *shingling* to refer to it. The key idea is, given a subset A of a universe X of elements, to generate a constant-size fingerprint such that the two subsets A and B can be compared by simply comparing their fingerprints [Gibson et al., 2005]. If π is a random permutation of the elements in the ordered universe X from which A and B are drawn, then it follows that [Broder et al., 2000]

$$Pr\{\min_{a\in A}\pi(a) = \min_{b\in B}\pi(b)\} = \frac{|A\cap B|}{|A\cup B|}. \tag{3.10}$$

This means that the probability that the smallest elements of $\{\pi(a)|a\in A\}$ and $\{\pi(b)|b\in B\}$ coincide corresponds exactly to the similarity of the two sets according to the Jaccard coefficient (see an example in Figure 3.6). Using this observation, one can sample from

A = {1, 2, 3}, B={1, 3, 4}, X={1, 2, 3, 4}

1 2 3 4	2 1 3 4	3 1 2 4	4 1 2 3
1 2 4 3	2 1 4 3	3 1 4 2	4 1 3 2
1 3 2 4	2 3 1 4	3 2 1 4	4 2 1 3
1 3 4 2	2 3 4 1	3 2 4 1	4 2 3 1
1 4 2 3	2 4 1 3	3 4 1 2	4 3 1 2
1 4 3 2	2 4 3 1	3 4 2 1	4 3 2 1

- where the condition holds (1/2 of the cases, which equals J(A,B))

Figure 3.6.: Jaccard coefficient and permutations: The permutations of the set X that satisfy $\min_{a\in A}\pi(a) = \min_{b\in B}\pi(b)$ from (3.10) are shaded. Their proportion over all permutations of X is $\frac{1}{2}$ which equals $J(A,B)$.

the space of permutations π and test whether the equality $\min_{a\in A}\pi(a) = \min_{b\in B}\pi(b)$ holds. To this end, we define a *fingerprint* of a set A with respect to a permutation π as $\operatorname{argmin}_{a\in A}\pi(a)$; The similarity of two sets is then estimated to be the fraction of components in their fingerprints that are identical. This idea can be extended to all subsets of size s of A instead of single elements of A by taking the minimum over these after permutation, that is, taking as a fingerprint the set of s elements of A that correspond to s smallest values in $\{\pi(a)|a\in A\}$. To express the dependence on s and c, such an algorithm is referred to as a (s,c) shingling algorithm.

Jaccard coefficient for multisets Now we have a measure of similarity for sets, as well as an efficient algorithm for approximating it. However, the sets we wish to compare in our problem, which are the sets of labels of neighbors of nodes, are in fact multisets. In this section we extend the notion of Jaccard coefficient to multisets.

We start by defining the following extension to the indicator function. For a given set X and a multiset A with elements from X we extend the indicator function from

$$\mathbf{1}_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise,} \end{cases}$$

to

$$\mathbf{1}_A^{\mathrm{MS}}(x) = \begin{cases} k & \text{if } x \text{ occurs } k \text{ times in } A, \\ 0 & \text{otherwise.} \end{cases}$$

This leads to a variant of the Jaccard kernel for multisets:

Definition 3.6.1 (Multiset Jaccard Coefficient) *Let X be a set and A, B multisets of elements from X. Let us introduce the following set operations:*

$$|A \cup B|_{\mathrm{MS}} = \sum_{x \in X} \max(\mathbf{1}_A^{\mathrm{MS}}(x), \mathbf{1}_B^{\mathrm{MS}}(x)),$$
$$|A \cap B|_{\mathrm{MS}} = \sum_{x \in X} \min(\mathbf{1}_A^{\mathrm{MS}}(x), \mathbf{1}_B^{\mathrm{MS}}(x)).$$

Then the Multiset Jaccard coefficient is defined as

$$J_{\mathrm{MS}}(A, B) = \frac{|A \cap B|_{\mathrm{MS}}}{|A \cup B|_{\mathrm{MS}}}.$$

We now proceed to showing that $J_{\mathrm{MS}}(A, B)$ corresponds to the Jaccard coefficient (3.9) on transformed A and B and hence is a kernel.

We introduce the following transformation T of the arguments of J_{MS}: Given a set A, T maps any element $x \in A$ occurring n times in A into the set of pairs $\{(x, 1), \ldots, (x, n)\}$ and takes the union over all $x \in A$. It follows immediately that

$$|A \cap B|_{\mathrm{MS}} = |T(A) \cap T(B)| \text{ and } |A \cup B|_{\mathrm{MS}} = |T(A) \cup T(B)|,$$

and consequently $J_{\mathrm{MS}}(A, B) = J(T(A), T(B))$. As J is a kernel, it follows that J_{MS} is a kernel as well.

Hence, to approximate the multiset Jaccard coefficient on the multisets A and B, we approximate the Jaccard coefficient on the corresponding sets A^* and B^*, which treat reoccurring elements from A and B as distinct characters. In practice, that means that we need to go over the sets of neighbors of each node in the graph once in $O(m)$ in order to apply the transformation T to the multisets of labels of neighbors of nodes.

3.6.2. Shingled Weisfeiler-Lehman subtree kernel

Now we have all ingredients to define a kernel with approximate subtree pattern matching. The pseudocode for computing the shingled subtree kernel is given in Algorithm 5. The labels are initialized as before: $l_0(v) = \ell(v)$ for all v in all graphs.

Algorithm 5 Shingled WL subtree kernel on N graphs, i-th iteration, $i > 0$

1: Multiset-label determination
- Assign a multiset label $M_i(v)$ to each node v in G which consists of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.

3: Creating fingerprints
- Permute the alphabet Σ_{i-1} and sort the strings $s_i(v)$ accordingly.
- Remember the first s letters of each string as a *fingerprint*.
- Repeat this permutation c times to extract c fingerprints per node.

4: Label compression
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.
- Map each string $s_i(v)$ to a compressed label using a hash function $f : \Sigma^* \to \Sigma$

5: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G.

The shingling Weisfeiler-Lehman (WL) subtree kernel proceeds in h iterations as the exact WL subtree kernel. In the latter case, each of these iterations comprises the following steps: i) for each node, record its neighbors, ii) for each node, sort its neighbors according to their node label, iii) compress the node labels, iv) relabel and update the kernel.

In shingling WL, we switch this order: We first update the kernel, then compress the labels. In order to update the kernel matrix entries, we approximate the Jaccard coefficient between the sets of neighbors of all pairs of nodes via shingling. We permute the neighborhood strings c times and then check for all pairs of nodes, whether their fingerprints of s letters coincide. We next give the formal definition of the kernel computed by Algorithm 5.

3.6.2.1. Kernel definition

Denote by $\pi_j^{(i)}$ a permutation of the compressed labels of all graphs after iteration i of WL. Assume that we perform c such permutations $\{\pi_1^{(i)}, \ldots, \pi_c^{(i)}\}$. Then we can define the following kernel on neighborhood strings which approximates the Jaccard coefficient.

Definition 3.6.2 ((c,s)-approximating Jaccard kernel) *Let c denote the number of permutations, and s the length of fingerprints that we compare. Let $s_i(v)$ be the multiset label as in the step 2 of Algorithm 5, at the i-th iteration. Slightly abusing notation, let $\pi(s_i(v))$ denote a transformation of $s_i(v)$ where all of its letters x are replaced by $\pi(x)$. Let δ_s be a Dirac kernel on strings which equals 1 if the leading substrings of length s match and 0 otherwise. Then the (c,s)-approximating Jaccard kernel is defined as*

$$k_J(s_i(v), s_i(v')) = \sum_{j=1}^{c} \delta_s(\pi_j^{(i)}(s_i(v)), \pi_j^{(i)}(s_i(v'))).$$

We can now define a corresponding graph kernel which computes the Jaccard kernel on all pairs of nodes from two graphs:

Definition 3.6.3 (Shingling graph kernel) *For two graphs $G = (V, E, \ell)$ and $G' = (V', E', \ell)$ the shingling graph kernel is defined as*

$$k_S(G, G') = \sum_{i=1}^{h} \sum_{v \in V} \sum_{v' \in V'} \delta(l_i(v), l_i(v')) k_J(s_i(v), s_i(v')). \tag{3.11}$$

h is the number of iterations of the Weisfeiler-Lehman algorithm, and δ is the Dirac kernel that equals 1 if its two arguments are identical and 0 otherwise.

Theorem 3.6.4 *k_S is a positive semidefinite kernel.*

Proof k_J is a sum over Dirac δ kernels, hence it is positive semidefinite. k_S is a sum over products of a Dirac kernel and k_J terms, hence k_S is also positive semidefinite. Finally, this also holds in expectation over all permutations, since again, it is a positive linear combination of positive semidefinite kernels. ■

3.6.2.2. Runtime analysis

How can we efficiently compute the shingling kernel on a data set of N graphs? Naively, one would compare all pairs of graphs ($O(N^2)$), and for each of these comparisons, we would compare all pairs of n^2 nodes from the two graphs, resulting in a total runtime of $O(N^2 n^2 dhc)$ in h iterations, for c permutations and for a maximum degree of d per graph.

However, we can drastically reduce this complexity in practice by hashing. Instead of comparing all pairs of fingerprints, we hash the first s letters of each fingerprint after each of the c iterations. We can then use the hash values of a graph G as a feature vector representation of G. The shingling kernel can then be computed by taking the

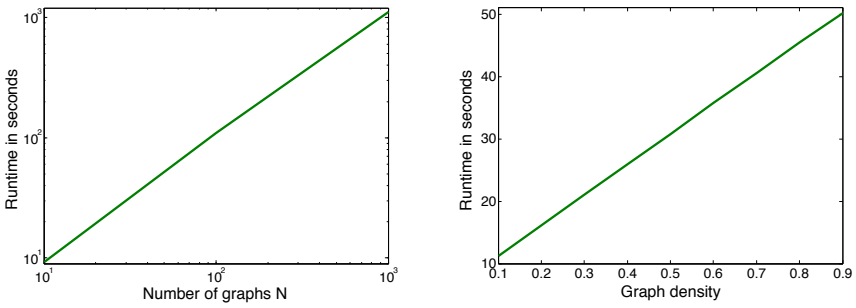

Figure 3.7.: Runtime behavior of WL on synthetic graphs as a function of data set size (left) and graph density (right).

inner product of the feature vectors of two graphs. This reduces the runtime effort to $O(Nhmc + N^2hmc)$, where the first term is the cost for computing the feature vector representation of each of the N graphs with m edges in h iterations and c permutations. The second term is the effort of computing the kernel matrix entries from the N^2 pairs of feature vectors, which contain at most hnc non-zero entries each. As we will confirm in our experiments, the latter step, the sparse vector multiplication does not dominate the runtime, such that the empirical runtime scales as $O(Nhmc)$.

3.6.3. Experiments

In our experiments, we compare the runtime and accuracy of our approximate subtree kernel to that of the original Weisfeiler-Lehman subtree kernel on real and synthetic data sets.

3.6.3.1. Runtime on synthetic data sets

As a first experiment, we assessed the runtime behavior of the shingling WL kernel, in particular with respect to the data set size and the number of edges per graph.

Experimental setup We examined the runtime behavior of shingling WL on synthetic graphs with respect to two parameters: data set size N and graph density δ. The density of an undirected graph of n nodes without self-loops is defined as the number of its edges divided by $n(n-1)/2$, the largest possible number of edges. For a fixed value of n, the density is proportional to the number of edges.

We kept one of the two parameters fixed at its default value while varying the other. The default values we used were $N = 10$ and $\delta = 0.4$; we varied N in range $\{10, 100, 1000\}$ and δ in $\{0.1, 0.2, \ldots, 0.9\}$. We set the size of the graphs n to 100, the height of the

	Data set	ENZYMES	PROTEINS	D&D
Number of graphs		600	1113	1178
Maximum degree		9	25	19
Average number of nodes		32.6	39.1	284.3
WL kernel (exact)	Accuracy (%)	55.00±2.98	74.77±1.61	78.54±1.31
WL kernel (exact)	Runtime	0'26"	1'03"	7'04"
Shingling kernel (approximate)	Accuracy (%)	57.20±1.86	76.57±0.69	81.54±1.54
Shingling kernel (approximate)	Runtime	11'29"	26'26"	3h01'7"

Table 3.1.: Runtime and classification accuracy on three bioinformatics data sets.

subtree patterns $h = 2$, the number of permutations $c = 10$, and the fingerprint size $s = 1$.

For each individual experiment, we generated N graphs with n nodes, and inserted edges randomly until the number of edges reached $\lfloor \delta n(n-1)/2 \rfloor$. We then computed the shingling WL kernel on these synthetic graphs. We report CPU runtimes in seconds in Figure 3.7, as measured in Matlab R2008a on an Apple MacPro.

Results In Figure 3.7, we clearly observe that the shingling WL scales linearly in the number of graphs N in the data set. As stated in Section 3.6.2.2, the empirical runtime hence depends on the actual Weisfeiler-Lehman iterations (theoretical runtime $O(Nhmc)$), not the effort of computing the kernel matrix from all feature vectors in the end (theoretical runtime $O(N^2hmc)$). It is also apparent that the runtime scales linearly with graph density, and hence the number of edges, as expected from the theoretical runtime complexity. Both these results indicate that the shingling WL kernel exhibits attractive scalability properties.

3.6.3.2. Graph classification on real-world data sets

Next, we performed graph classification on real-world data sets from bioinformatics.

Data sets D&D is a set of 1178 protein structures [Dobson and Doig, 2003] represented as graphs where each amino acid corresponds to a node, and an edge is drawn between two nodes if the distance between them is less than 6 ångströms. The prediction task is to classify the protein structures into enzymes and non-enzymes. PROTEINS is a data set that is based on a subset of 1113 proteins from D&D. Again, each protein is represented by a graph, but now each node is a secondary structure element of the protein and edges represent spatial neighborhood between these elements. The prediction task is the same as on D&D, that is, to classify the protein structures into enzymes and non-enzymes. The graphs in ENZYMES also represent protein structures in the same form as

in PROTEINS. ENZYMES includes 6 classes with 100 enzymes each from the top level of the enzyme commission (EC) enzyme hierarchy. The prediction task is to correctly predict the membership to each of these 6 classes in one-against-the-rest classification.

We provide statistics on the size, average node number and degree of these three data sets in Table 3.1.

Experimental setup We compared the new shingling Weisfeiler-Lehman kernel to the exact matching WL subtree kernel from Section 3.2. The default settings for fingerprint size and number of permutations were $s = 2$ and $c = 25$, respectively.

We performed 10-fold cross-validation of C-Support Vector Machine Classification, using 9 folds for training and 1 for testing. All parameters of the SVM were optimized on the training data set only. We optimized $h \in \{1, \ldots, 5\}$ for the shingling and the approximate WL kernel by 2-fold cross-validation on the training data set. To exclude random effects of fold assignments, we repeated the whole experiment 10 times. We report average prediction accuracies and standard errors in Table 3.1.

Results The approximate Weisfeiler-Lehman algorithm achieves slight improvements in accuracy over the exact Weisfeiler-Lehmann kernel on all three data sets. The largest improvement is achieved on the largest data set, the D&D data set, where the approximate WL kernel improves accuracy by 3%.

This improvement in accuracy is accompanied by a increase in runtime which is proportional to the number of permutations performed. This indicates that the exact WL kernel is the method of choice when runtime matters most, while the approximate WL kernel should be preferred when better accuracy is more important than computational runtime.

Number of permutations On ENZYMES, we explore the impact of varying the number of permutations c on runtime and accuracy. We kept all other kernel parameters fixed ($s = 2$, $h = 4$) and varied c in $\{10, 20, 50, 100\}$. We report runtimes classification accuracies in Table 3.2.

We observe that – as expected from the theoretical runtime complexity – the runtime grows linearly with c. The classification accuracy improves significantly when going from $c = 10$ to $c = 100$, while $c = 20$ and $c = 50$ do not lead to more than slight improvements over the accuracy reached with ten permutations.

Fingerprint size We also explored the effect of changing fingerprint size on the results on ENZYMES. We kept all other parameters fixed ($c = 100$, $h = 4$), and report accuracy and runtime in for $s \in \{1, 2, 3, 4\}$ in Table 3.3.

We observe a slight increase in runtime with growing fingerprint size. The classification accuracy reaches its maximum for $s = 2$ with at 57.07%, the other three fingerprint sizes

c	10	20	50	100
Accuracy (%)	54.56±1.07	55.76±0.86	54.67±0.79	57.07±0.57
Runtime	5'13"	8'54"	20'06"	39'43"

Table 3.2.: Impact of varying number of permutations c on the ENZYMES data set ($s = 2$, $h = 4$).

s	1	2	3	4
Accuracy (%)	55.23±0.65	57.07±0.57	55.43±1.40	54.20±1.57
Runtime	39'7"	39'43"	40'40"	40'50"

Table 3.3.: Impact of varying fingerprint size s on the ENZYMES data set ($c = 100$, $h = 4$).

lead to similar results around 54%.

3.6.4. Discussion

In this section, we presented an efficient and scalable graph kernel for approximately matching subtree patterns in two graphs. It is based upon the Weisfeiler-Lehman algorithm [Weisfeiler and Lehman, 1968], the Weisfeiler-Lehman subtree kernel from Section 3.2, and the efficient shingling approximation of the Jaccard coefficient [Gower, 1971]. It allows us to compare N graphs in a runtime of $O(Nhmc)$, where h is the number of iterations of Weisfeiler-Lehman, m the number of edges per graph, and c the number of permutations used to approximate the Jaccard coefficient. This runtime behavior which is linear in terms of all important kernel parameters and graph properties, enables scalable graph kernel computation and approximate matching, which could not be achieved simulatenously before.

However, this efficiency comes at a cost: While we do take into account partial similarities of neighborhood multisets before relabeling the graphs, the relabeling is still done according to the classic Weisfeiler-Lehman scheme from Algorithm 2 in Section 3.1. Therefore, some partial similarities are already lost in the second iteration, and even more disappear in subsequent iterations. This may be one reason why we do not observe an important boost in the expressivity in our experiments in Section 3.6.3.

3.7. Scalable node kernels

As another direction to explore in exploiting the efficiency and expressivity of the Weisfeiler-Lehman subtree kernel (Section 3.2), we consider using it to compare nodes

in a network. In other words, we would like to design a node kernel based on the Weisfeiler-Lehman subtree kernel.

Node classification in a network is a problem that has received considerable attention over recent years. In fact, in many large networks, such as social networks or protein interaction networks, a small number of nodes is annotated with class labels, which one would like to propagate to the whole network, or even to another, unlabeled network. Computational and space efficiency are key challenges in this task, as real-world networks often consist of $10^6 - 10^9$ nodes. In this section, we present a new algorithm for efficiently extracting expressive features for individual nodes in a large network, based on the network structure and, if available, the node attributes. Its runtime is only linear in the number of the edges.

3.7.1. Classification of nodes in a graph

Recent years have seen a rapid increase in the availability of various large networks, from social networks to web graphs to biological networks such as protein or gene interaction networks. When studying these networks, one central question that often arises is the following: Can we, given a set of class-labeled nodes in a network, infer class labels of other nodes based on the structure of the network?

There have been extensive studies addressing this question (interested reader can refer to Bhagat et al. [2011] for a recent survey), that can be broadly divided into two classes: The first class comprises learning algorithms exploiting the graph structure [Chakrabarti et al., 1998, Neville and Jensen, 2000]. Here, a subset of the nodes in the network is annotated with a class label, and this information is propagated to the rest of the network. Note that semi-supervised classification on graphs also exploits graph structure, however, one should make an important distinction between graph-based semi-supervised learning, and node labeling. In semi-supervised learning setting, usually the network is constructed based on the similarity of nodes [Zhu et al., 2003, Zhou et al., 2004, Gärtner et al., 2006, Abernethy et al., 2008, Fergus et al., 2009]; In contrast, in the setting which we are interested in the graph is an additional feature of the data, and having a link between two nodes does not necessarily mean that they are similar. A general property of the algorithms of semi-supervised learning type is that they either require a connected graph or labeled nodes in every component of the graph. Information cannot be propagated across graphs or across connected components in a single graph.

The second class consists of two-step procedures. They first extract node features based on the graph structure to compute pairwise similarity scores of nodes and then use any standard learning algorithm on these features. Node kernels, which compute kernel values between all pairs of nodes in a graph, are instances of this second class [Smola and Kondor, 2003, Fouss et al., 2006]. This family of methods is prone to suffering from runtime and memory problems: Computing the features and similarity scores usually

require at least taking powers of the adjacency matrix or the Laplacian of the graph. These operations are cubic in the number of nodes. Storing the similarity matrix is challenging, as it is typically a large dense matrix of size $n \times n$ for large n. This problem can be even more severe if nodes have attributes that one would like to take into account.

Note in the passing that we distinguish "node class label" and "node attributes", the former being a class label of a node, the latter a feature of the node. In this section, as in most of this thesis, "label" will usually mean the latter.

In this work, we propose a new feature generation method (or representation method in the sense of Section 1.4) for nodes of arbitrary networks, that addresses all three of the aforementioned problems. Namely, it can be used even for predicting labels for nodes from an entirely unlabeled connected component; Second, it can naturally handle node attributes, without incurring any additional computational cost; Finally, the algorithm runs in time linear in the number of edges in the network, ensuring scalability to large networks.

3.7.2. Technical background

In this section, we clarify notions that will help us introduce our node representation algorithms in the next section.

As in the previous sections of this chapter, we will rely on Algorithm 2 in Section 3.1 as the basis for our new algorithms. In Sections 3.2 and 3.3 we have proposed kernels for comparing *graphs* to each other based on the Weisfeiler-Lehman relabeling procedure. In this section, in contrast, we will exploit this algorithm for efficiently extracting features for comparing *nodes in one network*.

3.7.2.1. Locality sensitive hashing

In this section we recall locality sensitive hashing (LSH) [Indyk and Motwani, 1998], which will later serve as an ingredient for designing new feature extraction algorithms for nodes in a network. We follow the definitions given by Charikar [2002].

Definition 3.7.1 (Locality sensitive hashing) *A locality sensitive hashing scheme is a distribution on a family $\mathcal{F}$ of hash functions operating on a collection of objects, such that for two objects x, y,*

$$Pr_{h \in \mathcal{F}}[h(x) = h(y)] = sim(x, y),$$

where $sim(x, y)$ is some similarity function defined on the collection of objects.

If we choose a random vector $r \sim \mathcal{N}(\mathbf{0}, I)$ from the multivariate Gaussian distribution, and define the hash function h_r as

$$h_r(u) = \begin{cases} 1 & \text{if } \langle r, u \rangle \geq 0, \\ 0 & \text{if } \langle r, u \rangle < 0, \end{cases} \tag{3.12}$$

then for vectors u and v,

$$Pr[h_r(u) = h_r(v)] = 1 - \frac{\theta(u,v)}{\pi}, \tag{3.13}$$

where $\theta(u,v)$ is the angle between vectors u and v. In other words, if the angle between two vectors is small, then with high probability they will lie on the same side of the hyperplane defined by r [Goemans and Williamson, 1995, Charikar, 2002].

3.7.3. Weisfeiler-Lehman node feature maps

In this section we introduce two new node feature maps inspired by the Weisfeiler-Lehman relabeling procedure from Section 3.1. Our goal is, for a network (V, E, ℓ), to come up with an expressive and efficiently computable feature mapping $\phi(v), v \in V$.

3.7.3.1. Exact WL node feature map

Here we describe a natural modification of the Weisfeiler-Lehman subtree kernel to obtain a feature mapping for individual nodes. To do this, we will simply associate with every node its original and compressed labels in iterations $\{1, \ldots, h\}$. The following definition formalizes this idea.

Definition 3.7.2 (Exact WL node feature map) *Define $\Sigma_i \subseteq \Sigma$ as the set of letters that occur as node labels at least once in G at the end of the i-th iteration of the Weisfeiler-Lehman algorithm. Let Σ_0 be the set of original node labels of G. Assume all Σ_i are pairwise disjoint. Without loss of generality, assume that every $\Sigma_i = \{\sigma_{i1}, \ldots, \sigma_{i|\Sigma_i|}\}$ is ordered. Let $\delta(x,y) = 1$ if $x = y$, and 0 otherwise, for $x, y \in \Sigma$.*

The exact Weisfeiler-Lehman feature mapping for a node u in G with h iterations is defined as:

$$\phi^{(h)}_{exact}(u) = (\delta(l_0(u), \sigma_{01}), \ldots, \delta(l_0(u), \sigma_{0|\Sigma_0|}), \ldots, \delta(l_h(u), \sigma_{h1}), \ldots, \delta(l_h(u), \sigma_{h|\Sigma_h|})). \tag{3.14}$$

Clearly, $\phi^{(h)}_{exact}(v)$ is binary and has only $h+1$ non-zero entries, and can therefore be stored in a memory-efficient way as a sparse array.

Runtime analysis Computing this feature vector representation for all nodes in a network with m edges and with h iterations of the Weisfeiler-Lehman algorithm requires a runtime of $O(h\ m)$, which follows directly from the runtime of the Weisfeiler-Lehman algorithm.

3.7.3.2. LSH-WL node feature map

Of course, the exact WL feature map is restrictive in several aspects. In particular, it will miss to take into account any kind of partial similarity between nodes and their neighborhoods. Based on this feature map, one can only decide whether two nodes have exactly matching neighborhoods of distance up to h. As soon as our knowledge of the network is noisy or incomplete, as it is often the case in biological networks for instance, this exact matching of neighborhoods will hardly ever occur. Even when the network is not incomplete or noisy, neighborhoods may not match exactly due to the inherent nature of the network. For example, in a social network, if a person a is connected to 15 architects and 10 interior designers, and b to 16 architects and 8 interior designers, the similarity of a and b in terms of their connections is intuitively not very different from the situation where b would also have 15 architects and 10 interior designers as acquaintances.

To address these problems, we here introduce a more complex version of the Weisfeiler-Lehman node feature map based on locality sensitive hashing [Indyk and Motwani, 1998, Gionis et al., 1999, Charikar, 2002], as described in Section 3.7.2.1. First, we describe the modified relabeling procedure (for one graph G) in Algorithm 6.

Algorithm 6 LSH-WL relabeling procedure (i-th iteration)

1: Multiset-label determination
- For $i = 0$, set $l_0(v) \leftarrow \ell(v)$ and $i \leftarrow 1$.
- For $i > 0$, assign a multiset-label $M_i(v)$ to each node v in G which consists of the multiset of pairs $\{(l_{i-1}(v), l_{i-1}(u)) | u \in \mathcal{N}(v)\}$. [4]

2: Approximate label compression
- Compute a histogram $x_i(v)$ of node label pair frequencies in $M_i(v)$. [5]
- Hash vectors $x_i(v)$ for all v from G to bitstrings $s_i(v)$, using LSH with k hyperplanes (taking k small to explicitly allow potentially many collisions).
- Compress bitstrings $s_i(v)$ for all v from G to new labels, using a function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

3: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G.

More informally, the relabeling procedure in Algorithm 2 (for $i > 0$) consists of 3 major steps, which proceed as follows:

1. For each node, form a multiset of edge labels. For instance, if a node v has label 2 and has 3 neighbors with labels 3, 3, and 2, then $M_i(v) = \{(2,3),(2,3),(2,2)\}$.

[4] Note that here we directly incorporate the label of the root node in the multiset label.

[5] $x_i(v)$ is a sparse vector $(c[(\sigma_{i1}, \sigma_{i1}), M_i(v)], c[(\sigma_{i1}, \sigma_{i2}), M_i(v)], \ldots, c[(\sigma_{i|\Sigma_i|}, \sigma_{i|\Sigma_i|}), M_i(v)])$ of dimension $|\Sigma_0|^2$, where $c[a, A]$ counts the number of occurrences of an element a in a set A.

2. First, form a histogram over label pairs in $M_i(v)$ for all v. The $x_i(v)$ corresponding to v from our example would have 2 nonzero entries - the component indexed by label pair $(2,3)$ would equal 2, and the one indexed by $(2,2)$ would be 1. $x_i(v)$ is a vector in $\mathrm{R}^{|\Sigma_i|^2}$ and can be hashed by LSH to a bitstring of length k. Then hash these bitstrings to short labels.

3. Relabel the graph.

Based on the relabeling procedure given in Algorithm 2, we define the new feature map as follows.

Definition 3.7.3 (LSH-WL node feature map) *Define $\Sigma_i \subseteq \Sigma$ as the set of letters that occur as node labels at least once in G at the end of the i-th iteration of the Weisfeiler-Lehman algorithm. Let Σ_0 be the set of original node labels of G. Assume all Σ_i are pairwise disjoint. Without loss of generality, assume that every $\Sigma_i = \{\sigma_{i1}, \ldots, \sigma_{i|\Sigma_i|}\}$ is ordered. Let $x_i(v)$ be defined as in Algorithm 2. Let $\delta(x,y) = 1$ if $x = y$, and 0 otherwise, for $x, y \in \Sigma$.*

The LSH-WL feature mapping for a node v in G with h iterations is defined as:

$$\phi^{(h)}_{LSH-WL}(v) = \begin{cases} (\delta(l_0(v), \sigma_{01}), \ldots, \delta(l_0(v), \sigma_{0|\Sigma_0|})) & \text{if } h = 0, \\ (x_1(v), \ldots, x_h(v)) & \text{if } h > 0. \end{cases} \tag{3.15}$$

Note that $\phi^{(h)}_{LSH-WL}(v)$ is either a vector of length $|\Sigma_0|$, or a concatenation of h vectors with dimension $|\Sigma_i|^2$ each. However, in the first case only one component of $\phi^{(h)}_{LSH-WL}(v)$ is nonzero, and in the second, at most $|N(v)|$ components of each $x_j(v)$, $j \in \{1, \ldots, h\}$ are different from zero. Therefore $\phi^{(h)}_{LSH-WL}(v)$ is sparse and can be dealt with efficiently.

The feature vector of a node v will, for each iteration, represent the original and compressed labels of v, and the histograms of labels of its neighbors over different iterations. Like this, if we use linear SVM classification in combination with the exctracted features, two nodes will be deemed similar if they share labels over many iterations, and when they do share labels, their similarity will be boosted if their neighborhoods also share many labels. The intuition behind picking a low number k of random hyperplanes for hashing the label histograms of neighborhoods is that we would like to be rather liberal in compressing neighborhood histograms – if they are not the same, but similar, we still would like to hash them to the same bucket. This intuition is reflected in the following theorem.

Theorem 3.7.4 *Let $\{r_1, \ldots, r_k\}$ be the set of independently sampled hyperplanes, used to locality sensitive hash node neighborhood histograms. If in the i-th iteration of Algorithm 2 nodes u and v have the same label and different neighborhoods, then the probability that they will have the same label in the $(i+1)$-th iteration equals $\left(1 - \frac{\theta(x_i(u), x_i(v))}{\pi}\right)^k$, where $\theta(x_i(u), x_i(v))$ is the angle between the neighborhood histograms of u and v.*

Proof We have $l_i(u) = l_i(v)$ and $x_i(u) \neq x_i(v)$. From Equation (3.13) and from the r_j being independent of each other, it directly follows

$$Pr[l_{i+1}(u) = l_{i+1}(v)] = Pr[h_{r_1}(u) = h_{r_1}(v)] \cdot \ldots \cdot Pr[h_{r_k}(u) = h_{r_k}(v)] \tag{3.16}$$

$$= \left(1 - \frac{\theta(x_i(u), x_i(v))}{\pi}\right)^k. \tag{3.17}$$

■

As $x_i(u) \neq x_i(v)$, we have $\left(1 - \frac{\theta(x_i(u),x_i(v))}{\pi}\right) < 1$, and therefore its powers decrease exponentially in k.

Note that for using the feature map from Definition 3.7.3 in an inductive setting, that is, where the test set is not known before learning, one will have to keep track of the random hyperplanes $\{r_1, \ldots, r_k\}$.

Theorem 3.7.5 *For a graph G of m edges, for h Weisfeiler-Lehman iterations, and for k random hyperplanes used in the locality sensitive hashing, the LSH-WL features can be extracted from G in $O(hmk)$.*

Proof There are two additional steps that we have to perform in the LSH-WL feature map computation compared to the exact WL map. First, in Step 2, we have to compute a histogram of label pairs in the neighborhood of each node. This step is linear in the number of edges, that is, $O(m)$. Second, in Step 3, we have to hash these histograms via locality sensitive hashing into k bits. For computing each of these bits, we have to compute an inner product $\langle x_i(v), r_j \rangle$, $j \in \{1, \ldots, k\}$. As $x_i(v)$ has non-zero entries only for neighbors of v, we can compute these inner products for all nodes n in runtime $O(m)$ per bit, hence $O(km)$ in total. Putting all these steps together, the runtime of this modified Weisfeiler-Lehman scheme is now $O(hmk)$. ■

If k and h are constants, the complexity of $O(hmk)$ is linear in m. If k and h are slowly growing functions of m, it is essentially linear in m.

3.7.4. Experiments

In our experiments, we evaluate our feature extraction methods on several large networks in terms of their runtime, and in terms of the classification accuracy that an SVM can achieve based on these features.

Data sets We conducted experiments on three publicly available data sets. The first, Blogosphere (downloaded from `http://www-personal.umich.edu/~mejn/netdata/`, collected by Adamic and Glance [2005]), consists of 1,490 nodes and 16,715 undirected edges. The nodes represent political blogs, and edges are based on the incoming and outgoing links on these blogs around the time of the 2004 presidential election in the

United States. The nodes have been manually labeled as belonging to the class 0 - left or liberal, or 1 - right or conservative. The task here is to correctly classify the nodes into these two classes, based on the network structure.

The second set of data sets comes from the DREAM5 network inference challenge, obtained from `http://wiki.c2b2.columbia.edu/dream/index.php/D5c4`. The goal of this challenge is to reverse-engineer gene regulatory networks from gene expression information. The challenge consists of 4 tasks: In each task, participants are given a microarray compendium and are asked to infer the structure of the underlying transcriptional regulatory network. Our goal is different: We are interested in predicting whether a gene is a transcription factor or not based on the transcriptional regulatory network. We use ground truth networks from tasks for which such a ground truth is available, that is, tasks 1, 3, and 4. The networks 1, 3, and 4 consist of 1,565 nodes and 3,996 undirected edges, 1,081 nodes and 2,055 edges, and 1,994 nodes and 3,935 edges respectively. We obtained these data with the kind permission from Prof. H.-J. Thiesen, Institute of Immunology, University of Rostock, Germany [Lorenz et al., 2009].

The third, Webspam-UK2007 data set, obtained from `http://barcelona.research.yahoo.net/webspam/datasets/` is based on a crawl of the .uk domain, done on May 2007. Its nodes, which represent hosts, were labeled as spam or non-spam by a group of volunteers. The network includes 114,529 hosts out of which 6,479 are labeled, and 1.7 million undirected edges. The task is to determine whether a host is spam or non-spam based on the network information. Note that in this task we use the totality of the network to derive features for nodes, but we only train and predict on labeled nodes.

Methods In this section, we have introduced the exact WL node feature map in Definition 3.7.2 (that we will call EXACT) and its extension in Definition 3.7.3 (LSH WL). LSH WL differs from EXACT in two ways: First, we explicitly represent label histograms of neighborhoods; Second, we deliberately use a coarse compression method for relabeling nodes based on their neighborhoods. We examine the effect of each of these modifications on the performance. We call the first extension EXACT+NH, and the second EXACT+LSH. Due to the random nature of LSH, we repeat the algorithm several times to get a reliable estimate of the quality of extracted features. The number of repetitions is considered to be one of the parameters of the algorithm.

We also compare our algorithms with two state-of-the-art node kernels, the diffusion kernel and the regularized Laplacian kernel [Smola and Kondor, 2003]. We could run them on all of our data sets except Webspam, where the matrix operations exceeded our memory and runtime resources. These similarity measures can only be applied to large-scale graphs via special transductive classifiers as used by Gärtner et al. [2006] or Abernethy et al. [2008]. We also ran a simple baseline, a linear kernel using lines of the adjacency matrix as feature vectors. This kernel, that we will call A^2, counts the number of neighbors two nodes share, and is a common criterion for friend recommendation used

Parameter	EXACT	EXACT+NH	EXACT+LSH	LSH WL
Regularization Parameter C	$\{0.01, 1, 100\}$	$\{0.01, 1, 100\}$	$\{0.01, 1, 100\}$	$\{0.01, 1, 100\}$
Number of WL iterations	$\{1, 2, 4, 6\}$	$\{1, 2, 4, 6\}$	$\{1, 2, 4, 6\}$	$\{1, 2, 4, 6\}$
Number of Repeats	-	-	$\{1, 5, 10\}$	$\{1, 5, 10\}$
# Hyperplanes for Compression k	-	-	$\{1, 2, 4\}$	$\{1, 2, 4\}$

Table 3.4.: Range of parameters being considered in optimization.

Data set	DREAM5-1	DREAM5-3	DREAM5-4	Blogosphere	WEBSPAM	ℓ_2
Diffusion	0.92 ± 0.00	0.72 ± 0.04	0.74 ± 0.07	0.96 ± 0.00	-	0.28
Laplacian	0.83 ± 0.01	0.60 ± 0.01	0.71 ± 0.02	0.96 ± 0.01	-	0.41
A^2	0.99 ± 0.00	0.83 ± 0.01	0.71 ± 0.01	0.96 ± 0.00	0.66 ± 0.01	0.13
EXACT	0.96 ± 0.02	0.90 ± 0.06	0.80 ± 0.05	0.52 ± 0.10	0.50 ± 0.01	0.47
EXACT+NH	1.00 ± 0.01	0.92 ± 0.05	0.77 ± 0.07	0.95 ± 0.02	0.66 ± 0.03	0.03
EXACT+LSH	1.00 ± 0.01	0.91 ± 0.05	0.78 ± 0.07	0.94 ± 0.02	0.67 ± 0.01	0.03
LSH WL	1.00 ± 0.01	0.92 ± 0.05	0.75 ± 0.08	0.95 ± 0.02	0.67 ± 0.01	0.05

Table 3.5.: Classification performance of node feature extraction methods.

by social networking websites.

Experimental setup We performed stratified 10-fold cross-validation for C-Support Vector Machine (SVM) Classification, using 9 folds for training and the remaining one for testing. All parameters including parameters for feature extraction and regularization parameter for SVM were optimized only on the training data set. The range of feature extraction and SVM parameters considered in the grid search is shown in Table 3.4. To exclude random effects of fold assignments, we repeated the whole experiment 10 times. We report average areas under the receiver operating characteristic curve (ROC AUC) for all data sets.

We also report the CPU runtime for feature extraction in seconds. Since the runtime of each algorithm depends on the parameters used, we chose the most time-consuming parameter setting from the possible combinations of parameter values in Table 3.4 for each algorithm for a fair comparison (WL iterations – 6, number of repetitions – 10, number of hyperplanes k – 4).

Results Average areas under the ROC curve (AUC) for classification and CPU runtimes for feature extraction are given respectively in Tables 3.5 and 3.6.

In terms of classification performance, LSH WL improves over EXACT on 4 out of 5 data sets, the gain in AUC being larger than 0.15 in two cases. The average AUC score of the LSH WL feature map is larger than that of the diffusion and regularized Laplacian kernels on the DREAM5 data sets. The same holds in comparison to A^2 on

Data set	DREAM5-1	DREAM5-3	DREAM5-4	Blogosphere	WEBSPAM
# nodes	1,565	1,081	1,994	1,490	114,529
# edges	3,996	2,055	3,935	16,715	1,688,827
Diffusion	17”	7.9”	28”	10”	-
Laplacian	12”	6.0”	11”	3.2”	-
A^2	0.02”	0.01”	0.01”	0.15”	1.0”
EXACT	0.06”	0.03”	0.06”	0.23”	25”
EXACT+NH	0.09”	0.04”	0.09”	0.46”	49”
EXACT+LSH	0.05”	0.03”	0.06”	0.11”	10”
LSH WL	0.20”	0.11”	0.20”	0.42”	45”

Table 3.6.: CPU runtime for feature extraction.

Webspam and all three DREAM5 data sets.

We also report the ℓ_2 distance to the best method on each data set for each of the seven methods used in Table 3.5. This criterion rewards methods that achieve AUC scores close to the best result across all data sets. Our three extensions of the exact WL are the three best performing methods according to this criterion, followed by A^2. The exact WL strongly fluctuates in its classification performance.

Interestingly, each of the two extensions of the exact-WL seem to improve classification accuracy on their own and reach results close to or even slightly better than that of the LSH-WL kernel. Hence they are competitive alternatives to using the LSH WL feature map on all of these tasks, and one cannot attribute the improvement compared to the exact WL to one or the other alone.

In terms of runtime per kernel matrix/feature map, our algorithms scale up easily to networks with more than one hundred thousand nodes. On Webspam, feature extraction for 6,479 nodes (while taking into account the whole network) took 10-49 seconds, while for kernel-based methods it is even problematic to keep the kernel matrix in real memory. A^2 is competitive in terms of speed with our feature extraction methods on all data sets and has a further advantage of not needing any parameters, but its performance in classification is significantly worse on 2 out of 5 data sets.

Among our four feature extraction schemes, it is noteworthy that the two LSH-based random algorithms (EXACT+LSH and LSH WL) do not take longer than the two deterministic algorithms (EXACT and EXACT+NH), although we repeat the randomized algorithms ten times to reduce random effects. The reason for this is the following: In deterministic algorithms, we have to maintain the hash table of strings $s_i(v)$ for all nodes v to relabel them. Since the number of elements stored in the table (i.e., the size of the alphabet $|\Sigma_i|$) approaches n as the iteration of the algorithm increases, the cost of maintaining the hash table grows as well. In LSH-based algorithms, however,

we do not need to look up a hash table to relabel nodes: The random projection procedure in LSH-based algorithms gives us the k-bit representation of hashed labels directly. Such an advantage is especially useful as the graph gets larger: In the WEBSPAM data set, the LSH-based algorithms EXACT+LSH and LSH WL are even faster than their deterministic counterparts EXACT and EXACT+NH, despite being repeated 10 times.

3.7.5. Discussion

In this section, we have defined an algorithm for rapid feature extraction for nodes in large networks. The algorithm has three advantages: First, it does not require the input graph ro be connected, and can hence be applied to graphs with several components or sets of graphs, in inductive or transductive settings. Second, it is directly applicable to graphs with categorical node attributes, while other feature extraction methods or similarity measures on nodes cannot take node attributes into account, or have to be severely redesigned to acquire this ability. Third, the method is highly scalable, and scales only linearly in the number of edges in the network.

Still, in our experiments, the feature extraction approaches presented in this section do not systematically outperform simple baselines with a high confidence. As the features are fast and easy to compute, an interesting question for future research will be to investigate if they can complement and improve existing methods for learning on large graphs.

3.8. Summary

In this chapter, we proposed the Weisfeiler-Lehman subtree kernel, the first kernel for both labeled and unlabeled graphs whose computation scales linearly in the number of edges in the graph data set. We then defined a general procedure for constructing graph kernels on graphs with unlabeled or categorically labeled nodes. This definition generalizes all existing graph kernels that deal with categorical labels on nodes. Our kernels outperform state-of-the-art kernels in terms of runtime, even the recently developed efficient computation schemes for random walks and graphlets. Moreover, they are highly competitive with existing graph comparison and representation methods in terms of classification accuracy, which demonstrates their expressivity. Many applications of machine learning on graph-structured data that could not be addressed by existing graph kernels can now benefit from the scalability and the ability to deal with node labels of our kernels.

We also considered two extensions of the Weisfeiler-Lehman subtree kernel, one for approximately matching node neighborhoods, and another for comparing nodes in a graph. Despite promising steps that we propose in the first extension, so far there is no strong empirical evidence in favor of our proposed solution to the problem of inexact matching of neighborhoods. It may be fruitful to approach the problem from a

different perspective, for instance, by modifying the hash function that relabels subtree patterns. As for the second extension, it would be interesting to study the behavior of our proposed node feature extraction scheme on other large-scale real-world graphs, alone or in combination with other approaches. Another exciting problem for future research is the efficient and expressive representation and comparison of graphs with complex labels, such as real numbers, high-dimensional vectors or strings, and real edge weights.

4. Applications

As we have seen in Section 1.1, there exist plenty of applications of machine learning on graph-structured data. However, in many applications, practitioners try to get round using graphs to model data because of the insufficient efficiency or expressivity of existing graph comparison and representation methods. This is unfortunate, as in these applications graphs are often the most natural representation of data.

In this chapter, we present applications of our previous contributions to problems from two different domains, remote sensing and computational structural biology, that benefit from using graphs as basic data structures. In the first case we develop an expressive node kernel that compares pixels in an image. In the second, we design a graph kernel that quantifies the similarity of local neighborhoods of atomic nuclei in proteins.

4.1. Node kernels for remote sensing image classification

In its most general sense, the term *remote sensing* refers to the acquisition of data without direct contact with the object of interest. It is however most frequently used to denote the acquisition and analysis of signals from earth sensed from aircraft and satellites. This is also the meaning that we will use throughout this section. *Remote sensing images* are usually acquired via *multispectral* or *hyperspectral* sensors; the size of these images is usually large, and each pixel is high-dimensional due to the large number of spectral channels. Remote sensing image classification refers to the following classification problem: Given a remotely sensed image with a small number of pixels annotated with class labels denoting different types of land cover (e.g., water, farmland, desert, or trees), learn how to classify the remaining pixels. Classifying all pixels results in a segmentation of the image, called the *thematic map* or the *classification map* of the land cover captured on the image. Developing efficient algorithms that accurately classify pixels is a crucial step towards the analysis of large amounts of available remote sensing data. Such algorithms would allow environmental and geoscientists to better detect spreads of pollution, better understand the dynamics of crop epidemics or vegetation growth, or discover new water sources, to name just a few applications.

We present the current state of the art in machine learning for remote sensing image classification in the next section. In Section 4.1.2, we present our proposed node kernel and study its relationship to other kernels for remote sensing image classification. In Section 4.1.3 we describe the remote sensing images that we classify in our experiments

and empirically compare our proposed node kernel to the state-of-the-art remote sensing image classifiers. We summarize our findings in Section 4.1.4.

4.1.1. Remote sensing image classification

Remote sensing image classification is a challenging problem. This is due to a combination of several factors, including the high dimensionality of each pixel, noise in the data, the large number of pixels, the typically low number of annotated pixels (that is, training examples), and the spatial interdepedence of pixel values (also called *spectral signatures* or *spectral responses*) [Lillesand et al., 2004].

The first two factors motivate the use of robust classifiers that can deal with noise and high dimensionality. The robustness is typically obtained via regularized classifiers that not only try to correctly classify the training points, but also control the complexity of the classification function, favoring simpler, more robust functions. SVMs, and more generally kernel methods, are a prominent family of methods with these properties. Kernel methods have been successfully used in pixel-based (spectral-based) classification, and are among the state-of-the-art remote sensing image classifiers [Camps-Valls and Bruzzone, 2005, 2009].

Despite the good performance of these methods with respect to previous related work, the obtained classification maps are often very noisy. This is especially the case when dealing with images acquired by very high resolution (VHR) sensors, or when classifying images acquired over regions where spatial homogeneity can not be assumed (e.g., urban areas). In fact, as we pointed out earlier, pixel values are spatially dependent: An image is *not* a mere collection of independent and identically distributed (*i.i.d.*) pixels, as assumed in pixel-based image classification, but it is a *structured* domain. Intuitively, spatially close pixels are likely to belong to the same class. Hence, the classifier should not solely rely on spectral features, but also take the spatial information into account. Classifiers that use both spectral and spatial information contained in the image are referred to as *spatio-spectral* classification methods.

The field of spatio-spectral image classification is very active, especially with the advent of very high resolution (VHR) sensors. Because of their high resolution, images generated by VHR sensors require particularly robust techniques to deal with the high correlation between spectral responses of neighboring pixels [Plaza et al., 2009]. Spatio-spectral methods in the literature can be categorized into two basic families: feature extraction (or preprocessing) techniques, and filtering (or post-classification) methods. In the first case, the spectral signature of a given pixel is combined with spectral signatures of its spatially neighboring pixels through window-based approaches, and the resulting feature vectors are then classified. This class of methods includes morphological filtering [Soille, 2003, Benediktsson et al., 2003, 2005], geometrical features [Inglada, 2007], and Markov random fields [Dubes and Jain, 1989, Jackson and Landgrebe, 2002]. These techniques yield good results in general but several critical parameters need to

be tuned, such as the scale and the extent of the spatial relations. Moreover, the quality of the feature extraction technique may hamper the representation capabilities of the classifier. The second class consists of approaches that essentially perform spatial smoothing of a pixel-based classification map, and are often called *post-regularization* methods. These approaches are mainly based on morphological operators, such as the majority voting scheme [Tomas, 1980, Ton et al., 1991, Solaiman et al., 1998, Zhang, 2001]. The generally low improvement in accuracy of these methods is due to their strong dependence on the performance of the particular classifier used and the filter design. For these reasons, considering spectral and spatial information jointly in the classifier training generally yields better results.

Existing spatio-spectral SVM classifiers usually perform a local spatial feature extraction. The spatial feature vector is then concatenated at a pixel level with the spectral signature and subsequently used for classification. The main problem with this approach is that the *curse of dimensionality* problem is worsened as the feature extraction process is repeated for each spectral channel. The latter problem has been alleviated with the introduction of composite kernels by Camps-Valls et al. [2006] which combine dedicated kernels for the spectral and spatial information. This framework has been recently extended to deal with convex combinations of kernels through *multiple kernel learning*. In both cases, however, the methodology still relies on performing an *ad hoc* spatial feature extraction before kernel computation, typically limited to second-order statistics or morphological operators [Camps-Valls et al., 2008, Marconcini et al., 2009, Tuia et al., 2010].

In the next section, we present an alternative approach for spatio-spectral classification with SVMs which alleviates the aforementioned problems. Our approach has several advantages over previous work: First, it takes into account higher order dependences in the neighborhood of the pixels than just pairwise relations. Second, it computes a single kernel that simultaneously captures both spectral and spatial similarities. Third, the proposed kernel generalizes the composite kernel approach [Camps-Valls et al., 2006] by computing similarities in a proper feature space. We later illustrate its performance in multi- and hyperspectral images of different spatial and spectral resolutions.

4.1.2. Expressive spatio-spectral kernels for pixel comparison

This section presents a node kernel for comparing pixels in a remote sensing image, represented as a graph of pixels. This well-founded approach guarantees that all relations of spectral signatures in a spatial neighborhood, beyond pairwise sample relations, are taken into consideration. Before we introduce our kernel, we point out the weaknesses of existing spatio-spectral kernels.

4.1.2.1. Limitations of existing spatio-spectral kernels

Camps-Valls et al. [2006] presented a family of various kernel-based approaches for remote sensing image classification. In their work, kernels are essentially constructed through a weighted summation of dedicated kernels on spatial and spectral features. This approach suffers from several limitations: First, as other methods that rely on a preprocessing step, composite kernels are limited by the quality of the feature extraction process and thus cannot learn all higher order similarities between neighboring samples directly. For example, using the mean or variance of the spatially neighboring pixels as a feature for classification may be useful, yet limited as only first- and second-order statistics are considered. Second, by merely looking at the similarity between pixels (or between pixels and means of their neighbors), one is implicitly assuming that the neighbors of two similar-enough pixels should be also similar. This is not necessarily true in remote sensing image processing, where data may lie in complex manifolds [Bachmann et al., 2005]. Finally, the computational cost of the method is high because a different parameter has to be tuned for each kernel in the composition.

We propose in the next section a new kernel that addresses these limitations.

4.1.2.2. Proposed node kernel

To define our node kernel, we employ the graph-theoretic notation introduced in Section 1.2: An undirected unweighted graph $G = (E, V)$ is comprised of its set of nodes, $V = \{v_1, \ldots, v_n\}$, and its set of edges, $E \subseteq V \times V$. Two nodes v_i, v_j of G are neighbors, if $(v_i, v_j) \in E$, and the neighborhood $\mathcal{N}(v)$ of a node v is the set of all its neighbors, that is $\{v_i | (v, v_i) \in E\}$. For each $v_i, v_j \in V$, $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$. The adjacency matrix A of G is a symmetric matrix of size $n \times n$, where the (i, j)-th entry equals 1 if an edge (v_i, v_j) exists and 0 otherwise.

A node kernel, given a graph G, is a function

$$k(v_i, v_j) = \langle \phi(v_i), \phi(v_j) \rangle,$$

that takes into account the topology of the graph G. See Figure 4.1 for a simple illustration of a graph and a possible feature map ϕ on its nodes.

Most prominent examples of kernels on nodes in a graph include the diffusion kernel [Kondor and Lafferty, 2002], p-step random walk kernel, regularized Laplacian kernel or inverse cosine kernel [Smola and Kondor, 2003]. These kernels are all based on the *graph Laplacian*, that is, the matrix $L = D - A$, where A is the adjacency matrix of the graph and D is an $n \times n$ diagonal matrix with $D_{ii} = \sum_j A_{ij}$. The graph Laplacian has also been used in remote sensing image classification previously [Camps-Valls et al., 2007, Gómez-Chova et al., 2008].

The computation time of most existing node kernels scales as $O(n^3)$, where n is the number of nodes in the graph, because it involves a multiplication or inversion of an

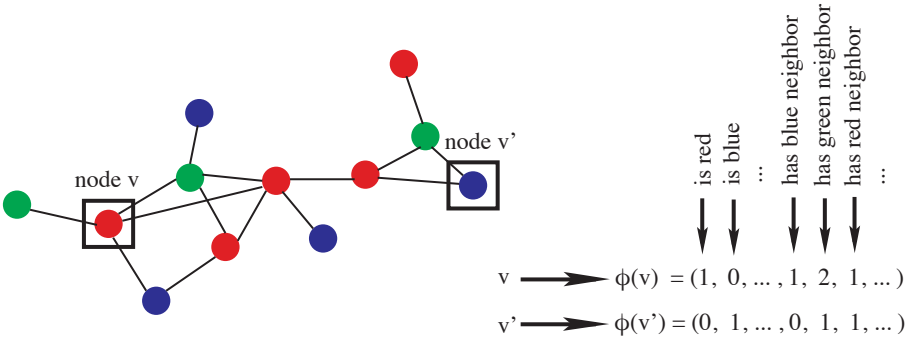

Figure 4.1.: A toy graph with data points as nodes (left) and an associated example feature map (right). Each data point is mapped into a vector containing entries that describe both the nodes and their neighborhood relations. In this example the mapping is explicit, but in general it can be implicit and infinite-dimensional.

adjacency matrix or a graph Laplacian. Such a complexity is prohibitively expensive when n is large. We here propose a scalable node kernel, which we later tailor to our application.

Definition 4.1.1 *Given a graph G and a kernel k_0 on its nodes (for instance, a kernel that checks for the identity of node labels) that we call the* base kernel*, the recursive node kernel of depth δ is defined as*

$$k_\delta(v, v') = \sum_{u \in \mathcal{N}(v)} \sum_{u' \in \mathcal{N}(v')} k_{\delta-1}(u, u'). \tag{4.1}$$

The recursive node kernel in Equation (4.1) can be specialized for image processing. In this setting, nodes become pixels, and the notion of vicinity in the graph corresponds to the spatial vicinity of the pixels. The proposed kernel recursively computes similarity between nodes: At each step δ, the kernel function (similarity) is computed first between spatial neighbors, and then between the closest neighbors in the previous kernel matrix. In this way, the kernel has a multi-scale structure depending on the so-called *data structural depth* parameter δ. For pixels $\mathbf{x}_v$ and $\mathbf{x}_{v'}$ (recall that pixels are represented by their spectral signature vectors), the kernel is defined as

$$k_\delta(\mathbf{x}_v, \mathbf{x}_{v'}) = \frac{1}{w^4} \sum_{m=1}^{w^2} \sum_{n=1}^{w^2} k_{\delta-1}(\mathbf{x}_m^v, \mathbf{x}_n^{v'}), \tag{4.2}$$

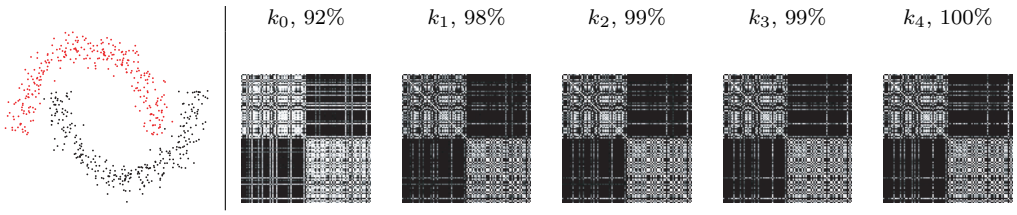

Figure 4.2.: Illustration of the node kernel in the "two moons" toy example. Kernels obtained for different $\delta = \{0, \ldots, 4\}$ are represented, along with the 10-fold cross-validation accuracy (in brackets) when using 50 training points per class in an SVM with Gaussian RBF kernel. Increasing the depth parameter δ gives rise to better results and smoother kernel matrices.

where $\mathbf{x}_m^v \in \Omega_v$ and $\mathbf{x}_n^{v'} \in \Omega_{v'}$, Ω_v and $\Omega_{v'}$ being spatial windows of width and height w around the pixels $\mathbf{x}_v$ and $\mathbf{x}_{v'}$. The base kernel k_0 is computed with all pairs of pixels in Ω_v and $\Omega_{v'}$.

Figure 4.2 illustrates the behaviour of the proposed node kernel in a 2-dimensional toy example. The kernel matrices for different depths k_δ are represented, along with the obtained 10-fold cross-validation accuracy when using 50 data points per class. The first kernel k_0 represents the sample-based classification, without including neighborhood relations. After the first iteration, a noticeable gain in accuracy is reached. The gain is only slightly improved with more iterations as the notion of vicinity in this example is based on Euclidean distance and is well-defined.

4.1.2.3. Relations to other remote sensing image classification methods

Our proposed method has direct connections to other kernel approaches in remote sensing image classification. For example, it can be easily shown that if $w = 1$, the proposed node kernel reduces to the spectral approach since in this case $\mathbf{x}_m^v = \mathbf{x}_v$. It also generalizes the cross-information kernel by Camps-Valls et al. [2006], as not only pairs of individual pixels are considered but all neighbors in a spatial window. Moreover, our node kernel generalizes the mean map kernel [Gómez-Chova et al., 2010]: Rather than computing average distances, the kernel estimates higher order deviations of all neighboring data points in the feature space. In fact, the notion of distance in our proposed method is based not only on the labeled training points but also on their spatial neighbors: This resembles kernel deformation methods used in semi-supervised learning, such as bagged kernels [Tuia and Camps-Valls, 2009], Laplacian SVM [Gómez-Chova et al., 2008] or label propagation methods [Camps-Valls et al., 2007].

4.1.2.4. Complexity

Our proposed kernel can capture all higher order spatial relations between pixels in a spatial neighborhood. Its main drawback is that the computational cost increases with the window size in $O(w^4)$, which can be prohibitive for large window sizes. In practice, however, one starts by building the largest training kernel possible and derive the lower scales from it through properly indexing kernel entries, without needing to recompute the whole kernel matrix. For the test phase, the computational cost is linear in the number of test pixels. An interesting advantage is that, once the kernel is computed (which is fast and simple for a low number of training samples), the training cost is the same as that of the standard SVM since no additional free parameters are included as occurred in the composite kernels framework. Finally, note that for the interesting case of VHR images, window sizes should not be very large to efficiently deal with spatial details.

4.1.2.5. Relation of the node kernel to the Morgan index

In this section we remark that the node kernel in Definition 4.1.1 is related to the topological index by Morgan [1965], discussed in Section 1.4.1. To make this relation explicit, we first reformulate the node kernel k_δ (4.1).

Proposition 4.1.2 *If we denote as ϕ_0 the feature map corresponding to the base kernel k_0, and define ϕ_δ as*

$$\phi_\delta(v) = \begin{cases} \phi_0(v) & \text{if } \delta = 0, \\ \sum_{v' \in \mathcal{N}(v)} \phi_{\delta-1}(v') & \text{otherwise,} \end{cases} \tag{4.3}$$

then $k_\delta(v, v') = \langle \phi_\delta(v), \phi_\delta(v') \rangle$ for any pair of nodes v and v' in a graph G and any $\delta \in \mathbb{N}$.

Proof We prove this proposition by induction over δ.

Induction initialisation $\delta = 0$: By definition of ϕ_0, $k_0(v, v') = \langle \phi_0(v), \phi_0(v') \rangle$.

Induction step $\delta \to \delta + 1$: Assume that $k_\delta(v, v') = \langle \phi_\delta(v), \phi_\delta(v') \rangle$. Then

$$\begin{aligned} k_{\delta+1}(v, v') &= \sum_{u \in \mathcal{N}(v)} \sum_{u' \in \mathcal{N}(v')} k_\delta(u, u') = \sum_{u \in \mathcal{N}(v)} \sum_{u' \in \mathcal{N}(v')} \langle \phi_\delta(u), \phi_\delta(u') \rangle \\ &= \Big\langle \sum_{u \in \mathcal{N}(v)} \phi_\delta(u), \sum_{u' \in \mathcal{N}(v')} \phi_\delta(u') \Big\rangle = \langle \phi_{\delta+1}(v), \phi_{\delta+1}(v') \rangle. \end{aligned}$$

■

We have showed above that ϕ_δ is the feature map corresponding to the kernel k_δ from Definition 4.1.1. Observe the similarity between ϕ_δ (4.3) and the Morgan index (1.1, Section 1.4.1): If we set $\phi_0(v)$ to 1 for all v in the definition of ϕ_δ, then they are equivalent up to notation. Thus, the feature map corresponding to our recursive node kernel (4.1) is a generalization of the Morgan index. If the map ϕ_0 is explicit and not prohibitively high-dimensional, then exploiting this relation may speed up the computation of the recursive kernel from (4.1). For a graph $G = (V, E)$ of n nodes, this can be done by first computing $\phi_\delta(v)$ for every v in $O(p\ \delta|E|)$, where p is the dimensionality of the feature vectors $\phi_0(v)$, and then taking the inner products in $O(n^2 p)$. The complexity of the recursive computation of k_δ consists of the cost of computing the kernel k_0, and $O(\delta|E|^2)$, the cost of computing $k_1, \ldots, k_\delta$. Depending on the problem at hand and the choice of k_0, one or the other computation scheme may be more efficient.

4.1.3. Experiments

In this section, we empirically evaluate our new node kernel.

4.1.3.1. Data collection

We consider two different multispectral and hyperspectral images in our experiments.

- *AVIRIS Indian Pines.* The first experiment deals with the standard AVIRIS image taken over North-West Indiana's Indian Pine test site in June 1992. Discriminating among the major crops can be difficult (in particular, given the moderate spatial resolution of 20 meters), which has made this scene a challenging benchmark to validate classification accuracy of hyperspectral imaging algorithms. The calibrated data is available online (along with detailed ground-truth information) from `http://dynamo.ecn.purdue.edu/~biehl/`. We used the whole scene, consisting of the full 145×145 pixels, which contains 16 classes, ranging in size from $20 - 2468$ pixels, and thus constituting a very challenging problem. Even though 20 noisy bands covering the region of water absorption are typically removed, we decided to keep them here to assess methods for robustness to noise.

- *DAISEX-1999 data set*: This data set consists of labeled pixels of 6 different hyperspectral images (700×670 pixels) acquired with the 128-bands HyMap airborne spectrometer during the DAISEX-99 campaign. This instrument provides 128 bands across the reflective solar wavelength region of 0.4μm-2.5μm with contiguous spectral coverage (except in the atmospheric water vapour absorptions bands), bandwidths around 16 nm, very high signal to noise ratio, and a relatively high spatial resolution of 5m.

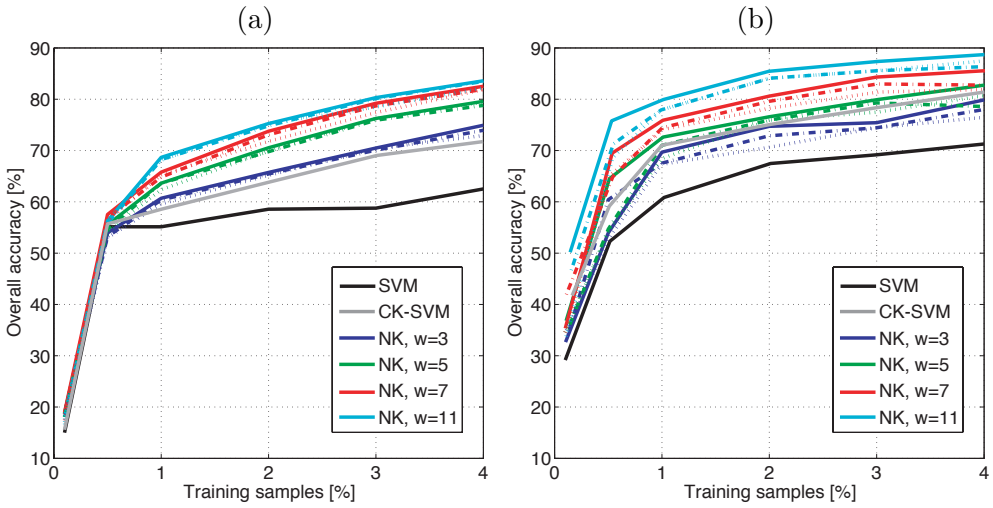

Figure 4.3.: Results for different remote sensing image classification problems: (a) AVIRIS Indian Pines, and (b) Hymap image. The overall accuracy is shown for different amounts of training data, window sizes w, and neighborhood depths $\delta = 1$ (dotted lines), $\delta = 2$ (dashed lines), and $\delta = 3$ (solid lines).

4.1.3.2. Model development and free parameter selection

We used different rates of randomly selected training samples, $N = \{0.1, 1, 3, 5\}\%$. In all cases, we used the Gaussian RBF kernel with parameter σ for the SVM classifiers. The classifiers were trained with 10-fold cross-validation on the training set, using the following parameters: $\sigma \in \{0.1, 0.25, 0.5, 1, 2, 3\}/N$, the regularization parameter $C \in \{10^0, \ldots, 10^2\}$, window size $w \in \{3, 5, 7, 9\}$, and different depths of the neighborhood parameter, $\delta \in \{1, 2, 3\}$. A *one-vs-one* multi-class classification scheme was adopted. We compare three kernels for SVMs: The kernel based on the spectral signature, a contextual kernel (CK) that uses stacked spectral and morphological features (specifically, a total of 25 spatial features were extracted by opening and closing morphological filters from specific bands), and our node kernel (NK). Before training, data were normalized to have zero mean and unit variance.

4.1.3.3. Numerical comparison

Figure 4.3 shows the results for the test set formed by all labeled data in the image, for the different images considered, and for different rates of training data, window sizes, w, and neighboring depths, δ. Several conclusions can be made: First, all node kernels, regardless of w or δ, outperform the standard spectral kernel. Second, as w increases,

results are generally improved but they typically saturate for $w \approx 7$. For larger values of w the performance is expected to decrease as the classification map would be over-smoothed. Third, the advantage of our proposed approach is clearly noticeable in the Indian Pines image (between $10-15\%$ gain in accuracy), mainly due to the high spectral resolution and spatial homogeneity of classes. In the particular case of the HyMap high resolution image, the gain in accuracy over the spectral SVM is between $16-18\%$. The SVM with the contextual kernel (CK-SVM) provides better results than the node kernel with $w = 3$, but with higher window sizes, our new kernel largely outperforms the contextual kernel. Note that the kernel depth δ improves the results in both cases, but the improvement mainly comes from the proper definition of the spatial extent. Classes are spatially and spectrally homogeneous and hence incorporating complex neighboring relations only improves results noticeably in high spatial resolution scenarios.

4.1.3.4. Statistical comparison

To further compare kernels, we have additionally performed a statistical analysis of the differences (in the test set) between all the considered spectral and node kernels with different w and δ. The comparison was done through the McNemar's test [Lapin, 1998, Foody, 2004], which is based on the standardized normal test statistic[1]. The test can be used not only to assess whether statistical differences between methods exist or not, but also to quantify the statistical gain of one classifier versus the others. The difference in accuracy between two classifiers is said to be statistically significant if $|z| > 1.96$, and the sign of z indicates which is the more accurate classifier. For the two considered images, the test allowed us to draw several conclusions: First, there are significant statistical differences between the spectral and spatial kernels ($|z| > 20$); Second, we observed significant statistical differences between different δ values ($|z| > 3.1$) only for the higher spatial resolution HyMap image; Third, we found that as the rate of training examples increases, the z-score decreases. All these conclusions match the numerical results shown before. To summarize, in both situations, the proposed spatio-spectral node kernel is significantly different (and in most of the cases better) than the standard kernels.

4.1.3.5. Visual comparison

Figure 4.4 shows the classification maps obtained with standard SVM and the proposed node kernel for 1% of training samples, $w = 11$ and $\delta = 3$, in a representative realization. Numerical results are confirmed by the visual inspection of the classification maps. For the case of the AVIRIS Indian Pines scene, the supervised spectral-based SVM obtains poor results, mainly due to the low number of training samples and high number of spectral bands and classes. The node kernel dramatically improves the accuracy and also

[1] Computing the z-score of classifiers c_1 and c_2 reduces to $z = (f_{12} - f_{21})/\sqrt{f_{12} + f_{21}}$, where f_{12} represents the number of samples correctly classified by c_1 and incorrectly classified by c_2.

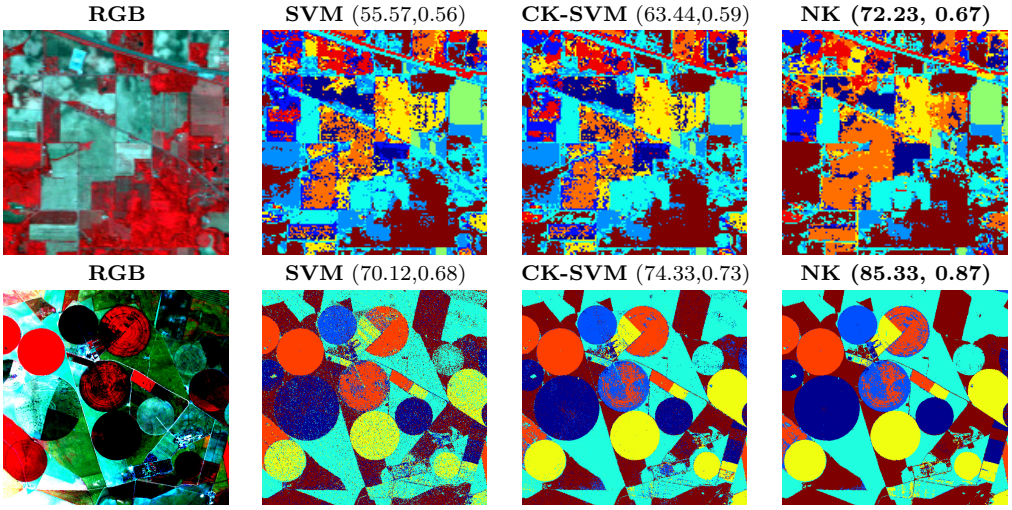

Figure 4.4.: RGB composition and classification maps for the best SVM, SVM with the contextual kernel (CK-SVM), and node kernel SVM for (top) AVIRIS Indian Pines and (bottom) HyMap images. Overall accuracy and kappa statistic are given in brackets.

yields a more spatially regularized classification map. See for instance the high spatial homogeneity coherence on different crop fields, such as the large soybean area in the center (orange in the image). The same is true for class "Soybeans-notill" (in blue, east side of the scene), which is better learned by the node kernel. Interestingly, improvement is typically observed for spectrally very similar classes (such as "Soybeans" subclasses), which suggests that the spatio-spectral information helps in identifying subtle, but critical differences. For the case of the HyMap image, more spatially coherent classification maps are obtained with node kernels than with both the standard (pixel-based) and the contextual SVM.

4.1.4. Summary

In this section, we have presented a kernel on graph nodes for spatio-spectral remote sensing image classification. Our novel kernel considers spatial neighborhoods of all pixels to compute pairwise similarities in a high-dimensional feature space. Our node kernel is a powerful alternative to existing pixel comparison and representation methods and generalizes previous approaches on kernel-based spatio-spectral classification. Since in spatio-spectral classification one is interested in computing all possible high-order neighborhood relations, we here proposed to directly work with the pixels embedded in a graph-based feature space. This has the advantage of getting rid of a spatial prepro-

cessing step, and of computing the similarity among the labeled pixels and those in their neighborhoods at different scales. Future work will explore automatic ways of selecting the window size, relating it to the spatial resolution of the image, and the neighborhood depth, related to the spatial arrangement of classes in the scene.

4.2. Chemical shift prediction using graph kernels

As another application, we present our ongoing work on chemical shift prediction. Analogously to the previous section, we start by providing a brief background on chemical shift prediction and nuclear magnetic resonance (NMR) spectroscopy for proteins.

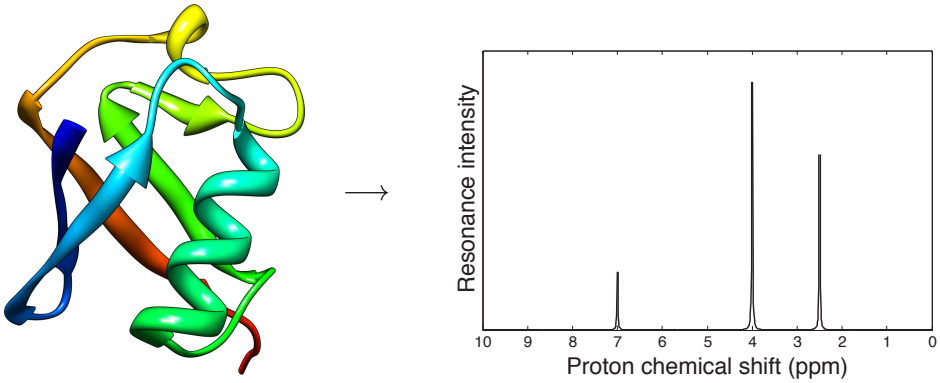

Figure 4.5.: A protein gives rise to a spectrum corresponding to a specific nucleus type, here hydrogen (or sets of nuclei for multidimensional spectra). A graphic of such a spectrum is shown on the right. Peak locations on the X axis represent chemical shifts of hydrogen nuclei, and the corresponding values on the Y axis depict their resonance intensities. Ideally, there should be as many peaks as there are hydrogen nuclei in the protein, however sometimes they overlap with each other. This problem is alleviated by multidimensional spectra, which are most common nowadays. The graphic on the left was generated by Michael Habeck using the UCSF Chimera package [Pettersen et al., 2004].

4.2.1. Chemical shift prediction

Protein NMR spectrosopy is a field of structural biology concerned with the design and development of NMR techniques that help obtain information about the structure and dynamics of proteins [refer to Roberts and Lian, 2011, for a recent textbook on protein

NMR]. The general procedure of protein NMR spectroscopy is organized as follows: First, a magnetic field is applied to a sample of the protein of interest. Every nucleus in the protein responds to this field with its resonance frequency and a certain intensity. This results in associating the protein with a one- or multidimensional spectrum of resonance frequencies of the nuclei in this protein. These frequencies are expressed via *chemical shifts*: A chemical shift of a nucleus is defined as a shift in its resonance frequency, relative to the resonance frequency of the same type of nucleus in the reference substance[2], and divided by the operating frequency of the spectrometer. As the frequencies of nuclei are expressed in hertz, and the operating frequency in megahertz, chemical shifts are expressed in *parts per million*, or ppm (see Figure 4.5). Thus, chemical shift is a convenient proxy to the shift in the resonance frequency of a nucleus with respect to the standard that does not depend on the applied magnetic field. Chemical shifts are due to the local chemical structure of nuclei. Therefore, the chemical shift is a function of a nucleus and its neighborhood, and can be used to obtain information about this neighborhood. The next step in protein NMR is resonance assignment, that is, association of peaks in the spectrum to the corresponding individual nuclei in the protein.

Chemical shift prediction may be seen as exactly the opposite of the latter task: It consists in predicting the shifts based on the local neighborhood of nuclei. To illustrate why this is useful, consider the following example: Given a protein, whose structure is already known, and a ligand, we would like to find out where the ligand binds the protein. This is a common problem in drug design, where the ligand is a drug, the protein is a target, and the question of interest is to find out how the drug acts on the target. Solving the structure of the protein together with the ligand is often an expensive and tedious task, while it is generally easy to measure its chemical shifts via NMR spectroscopy. If we compare the spectra of the protein in its pure state and together with the ligand, we will usually observe a slight difference (see Figure 4.6 for illustration). The chemical shifts that will have changed will correspond to the nuclei in the amino acids that the ligand directly affects. However, we also need to know which amino acid corresponds to which peak in the spectrum. If we are able to reliably predict chemical shifts, then we can align the predicted chemical shifts with the true chemical shifts, and thereby map the chemical shifts to their corresponding locations in the protein structure. Chemical shift prediction is useful as well in Bayesian techniques for predicting protein structure from chemical shifts.

Several methods have been proposed that predict chemical shifts from structure. Most prominent approaches include ProShift [Meiler, 2003], ShiftX [Neal et al., 2003], Sparta [Shen and Bax, 2007], CamShift [Kohlhoff et al., 2009], and ShiftX2 [Han et al., 2011]. To predict the shift of a nucleus, all of these methods except ShiftX and ShiftX2 rely on various types of information contained in the amino acid surrounding the nucleus and

[2]Standard chemical shifts are usually defined as the chemical shifts of tetramethylsilane (TMS).

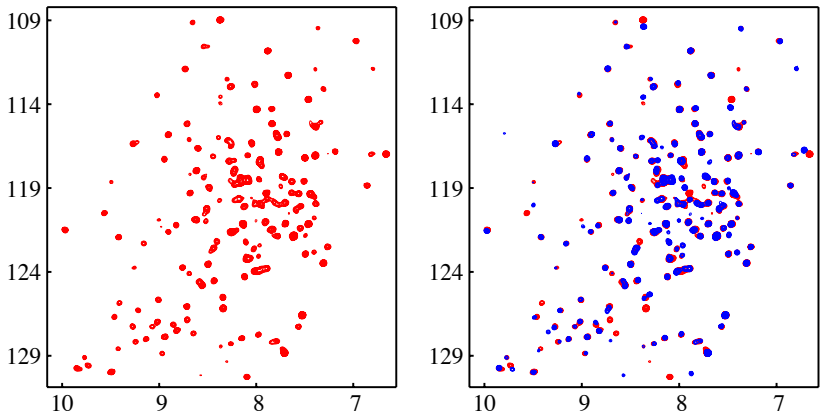

Figure 4.6.: On the left, a 2-dimensional spectrum corresponding to a protein in its pure state. Dots denote intensity peaks at the intersections of frequencies on x and y axes. On the right, the spectrum corresponding to the same protein, but with a ligand: The new, blue peaks of the protein with the ligand largely coincide with the old, red ones, but the differences give us information about the location of the ligand binding site. The figure was obtained from and is used with the kind permission of Remco Sprangers.

its neighboring amino acids according to the protein *sequence*. However, the chemical shift of a nucleus can be influenced by other nuclei far away according to the sequence, but close in the folded protein. ShiftX and ShiftX2 use information from the whole 3-dimensional neighborhood of a nucleus, but use a physical parametric model of chemical shifts which may not be precise enough. Moreover, none of these methods output the variance of the prediction. Knowing how confident one is about a prediction can however be helpful.

4.2.2. Graph kernels for neighborhoods of nuclei

Our goal in this work is to predict chemical shifts of nuclei while taking into account their complete neighborhood, to do this is in a flexible manner without relying on a predefined physical model, and to provide a way to quantify the uncertainty of our predictions.

In our approach, chemical shift prediction takes the form of a regression problem on graphs. We are given a set of proteins, and in each protein there is a set of nuclei for which the corresponding chemical shifts are available (obtained by experimental methods). The goal is to learn a function, which for each nucleus gives its corresponding chemical shift.

This function can then be used as a predictor for nuclei that do not have their chemical shift assigned. It is easy to distinguish chemical shifts associated with different types of nuclei, as their resonance frequency ranges strongly differ. Therefore predicting chemical shifts of different types of nuclei can be considered as different problems. We describe each nucleus of interest as a labeled graph representing its neighborhood of a certain radius (the construction of graphs will be detailed in the next section). We then compute a kernel between these graphs, and perform Gaussian process regression using this kernel.

Note that for applying graph kernels to the chemical shift prediction problem we do not need more structural information than other chemical shift prediction methods. Nonetheless, we believe that by modeling atom neighborhoods as graphs we can exploit the available structural information better than the other methods.

4.2.2.1. Construction of graphs

The neighborhood graphs of nuclei are constructed as follows: For each atom a for which we would like learn to predict chemical shifts, namely, atoms of type C$'$, Cα, Cβ, N, Hα, and H$_N$, we extract all atoms within a radius of several ångströms (3-6 in our experiments) around the nucleus. The range of values considered for the radius is based on prior knowledge on how far-reaching the chemical interactions influencing chemical shifts can be. The nuclei of these atoms play the role of nodes of the neighborhood graph of a. Edges in this graph are determined by thresholding the physical distance between nodes. They can also be designed to reflect covalent bonding between atoms. Nodes are labeled by their corresponding atom type: $l_0(p)$ denotes the label of node p. Moreover, in each graph, we keep track of the distances between nodes that we get from the protein structure: d_{pq} for nuclei p and q.

4.2.2.2. Proposed graph kernel

We propose to compute the similarity between two neighborhood graphs $G = (V, E)$ and $G' = (V', E')$ via the kernel function

$$k(G, G') = \sum_{i=0}^{h} w_i k_i(G, G'),$$

where

$$k_i(G, G') = \sum_{p,q \in V^2} \sum_{r,s \in V'^2} \delta(l_i(p), l_i(r))\ \delta(l_i(q), l_i(s))\ k_{dist}(d_{pq}, d_{rs}).$$

Here δ is the Dirac kernel ($\delta(x, y) = 1$ if $x = y$ and 0 otherwise), k_{dist} is a kernel comparing two distances (for instance, a Gaussian RBF kernel), and $l_0(v)$ is the atom type of the node v. Further node labeling functions, l_i for $i = 1, \ldots, h$, are determined by the Weisfeiler-Lehman relabeling from Section 3.1, Algorithm 2.

4.2.2.3. Regression method

We use Gaussian process regression to learn the kernel weights w_i and predict chemical shifts. Gaussian processes for regression are a powerful kernel method that view the function to be learned as a Gaussian distribution of as many dimensions as there are data points, and whose covariance matrix is given by the kernel matrix between these points [see the textbook by Rasmussen and Williams, 2005, for an extensive treatment of Gaussian processes in machine learning]. One of the advantages of Gaussian process regression is that its prediction for a data point is not a single number, but a one-dimensional Gaussian distribution. If this distribution is narrow for a given data point, then the regression model is highly confident about the predicted value for this point. The more spread out the distribution, the less we should trust its mean as our prediction. This key property motivates our choice of Gaussian process regression as the learning method in this problem. As predicted chemical shifts will have to be aligned with true chemical shifts (as discussed earlier in this section), knowing which predictions are more likely to be accurate can facilitate this alignment.

4.2.3. Experiments

In this section, we present our preliminary experimental results on a benchmark chemical shift prediction data set.

Data As a training set, we used 25 randomly chosen proteins from the VASCO data set [Rieping and Vranken, 2010], available at `http://www.ebi.ac.uk/pdbe-apps/nmr/vasco/main.html`. We predicted on the so-called "7 proteins" data set from Kohlhoff et al. [2009].

Experimental setup For each type of atoms considered, we extracted neighborhood graphs of radii of $3, 4, 5$ and 6 ångströms. We created edges between a pair of nodes if the distance in ångströms between them was shorter than $1.5, 1.7, 2$. As k_{dist}, we considered the Gaussian RBF kernel (see Section 1.5.1, (1.6)) with parameter σ in $\{2^{-6}, 2^{-4}, \ldots, 2^{6}\}$. We computed kernels for each combination of parameters, and chose the best ones for each task (C', $C\alpha$, $C\beta$, N, $H\alpha$ and H_N) by cross-validation on the training set. The number of Weisfeiler-Lehman relabelings, h, was set to 3. To perform regression, we used a publicly available Gaussian process toolbox for Python, developed by Stegle et al. [2011].

Note that throughout our experiments we use secondary chemical shifts, that is, the differences between the measured chemical shifts and the so-called random coil chemical shifts. This is a standard normalization technique used by all chemical shift prediction methods.

Preliminary results A standard measure of prediction quality used for the chemical shift prediction problem, and many regression problems in general, is the root mean square error. If $y_1, \ldots, y_N$ are the values to be predicted, and $\tilde{y}_1, \ldots, \tilde{y}_N$ the corresponding predictions, the root mean square error is defined as

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}.$$

We compare our results with results reported by Kohlhoff et al. [2009] on the "7 proteins" data set.

Nucleus	N	H_N	Hα	Cα	Cβ	C$'$
Our method	3.20	0.55	0.34	1.29	1.53	1.20
CamShift	2.78	0.56	0.26	1.22	1.19	1.12
SPARTA	2.66	0.53	0.26	1.03	1.07	1.05
ShiftX	2.66	0.55	0.28	1.14	1.25	1.19

Table 4.1.: Preliminary results (RMSE) on the "7 proteins" data set

As we can see from table 4.1, the performance of our method in its present state on the "7 proteins" data set is sometimes comparable with, but sometimes worse than the state of the art. However, we strongly believe that the performance of our approach can be significantly improved by taking into account more information already exploited by other algorithms. There exist at least four types of relevant information that we can further incorporate to improve our method: First, we have so far not considered the covalent bonding information into our method. This information is however relevant for chemical shift prediction and is used by most existing approaches. The same can be said of the type of the amino acid enclosing each nucleus of interest. Moreover, it has been shown that the chemical shift of a nucleus also depends on the type of secondary structure containing the nucleus [see e.g., Mielke and Krishnan, 2009]. Finally, one can also exploit structural information on the nuclei in the side chains. As all of these features can be obtained from the protein structures, we do not need additional data to incorporate these features into our model. It may also be helpful to predict chemical shifts for different atom types jointly via multidimensional regression, taking advantage of existing correlations between these tasks.

It is worth mentioning that in our experiments, the learned kernel weights w_i were systematically significantly higher for $i > 1$ than for $i \leq 1$ for all tasks except for predicting shifts of H_N nuclei. This indicates that the Weisfeiler-Lehman relabeling is indeed useful in this problem.

Moreover, we observed that for each of the 6 considered tasks a particular set of parameters was systematically selected over different random splits of the data into

Nucleus	N	H_N	Hα	Cα	Cβ	C$'$
Neighborhood radius	4	3	3, 4	4	3	4
Edge threshold	1.7	1.5	1.7	1.7	1.7	1.7
σ of the RBF kernel	2^{-6}	1	2^{-2}	2^{-2}	2^{-2}	1

Table 4.2.: Optimal parameters of our method found by cross-validation for different chemical shift prediction tasks

training and test sets (chosen parameters are reported in Table 4.2). We can therefore fix these parameters for each task, which will result in a much lower memory usage for storing kernel matrices. This saving of memory, in turn, will allow us to consider more training data.

4.2.4. Summary

In this section, we proposed a graph kernel-based method for predicting chemical shifts of nuclei in proteins from the local neighborhoods of these nuclei in the protein structure. In contrast with other existing approaches, we, first, take into account the complete 3-dimensional neighborhood of a nucleus in a given radius instead of focusing on information from neighboring amino acids according to the protein sequence. Second, we do not restrict our method to a particular physical parametric model. Third, we are able to quantify the uncertainty of our predictions.

While our preliminary experimental results do not yet outperform the state of the art, they are promising, as to date we have used little information in our approach compared to existing methods. As discussed above, exploiting other types of available information is expected to make the prediction accuracy of our method competitive with that of existing techniques. We believe that this will be the case, and we plan to further develop our approach in the near future.

5. Conclusions and outlook

In this thesis, we have studied the problem of scalable graph comparison and representation for machine learning on large, labeled graphs.

The problems of graph comparison and representation have a long and rich history in computer science research, yet traditional solutions do not suffice to address the more and more demanding needs of current machine learning problems on graphs: They are either not expressive enough, difficult to parameterize for a given problem, or intractable to compute in the worst case. This is due to several factors: First of all, as we have stressed in this thesis, graph comparison and representation are hard problems. Second, the bulk of research in graph comparison until the end of the 20th century has been concentrated on graph and subgraph isomorphism, less on flexible measures of graph similarity. Moreover, the comparison and representation of large graphs had not become truly pressing until around fifteen years ago, when the trend of data-intensive research started and efficient machine learning algorithms were developed.

We tackle the problem of graph comparison and representation with graph kernels. Graph kernels are an attractive choice for several reasons: First, they provide both similarity measures and representations for graphs. Second, they have the potential to efficiently compute expressive similarities in high-dimensional spaces. Third, they make it possible to readily apply a wide range of machine learning algorithms to graphs. Research on graph kernels has been an active area of study in machine learning on graph-structured data in the last decade, resulting in significant developments in the design of efficient and expressive graph kernels. Nonetheless, the most efficient existing graph kernels, kernels based on random walks, scale as $O(n^3)$ which is prohibitive for the size of graphs in current applications of machine learning on graphs. Moreover, random walk kernels are hindered by two problems inherent to walks, tottering and halting. Graphlet kernels were proposed as an efficient alternative to random walk kernels that does not suffer from these problems [Borgwardt, 2007]. They are based on the distribution of small subgraphs of size 3, 4, and 5 in graphs. Explicit enumeration of graphlets being computationally expensive, Borgwardt et al. [2007] proposed a sampling approach with guarantees on the ℓ_1 deviation of the empirical from the true distribution. This allowed to compare graphs whose size had been an unsurmountable obstacle for the existing graph kernels.

In Chapter 2, we argue that while the sampling approach is effective on dense graphs, its quality criterion, the ℓ_1 distance between the empirical and true distributions, is not adequate for sparse graphs. In fact, in sparse graphs connected subgraphs are rare,

and having a guarantee on the ℓ_1 deviation of the entire graphlet distribution does not mean that the distribution restricted to connected subgraphs will be accurately approximated. We then propose efficient algorithms for exact counting of all connected and non-connected graphlets of size up to 5. Our algorithms scale as $O(nd^{k-1})$ for graphs of size n with maximum degree d and graphlets of size k, as opposed to the complexity of $O(n^k)$ needed for the straightforward enumeration of graphlets of size k. As our new algorithms are most efficient for sparse graphs with $d \ll n$, exactly in the setting where random sampling suffers, these two approaches can be regarded as complementary to each other in efficiently comparing large graphs.

While graphlet kernels enable us to efficiently compare and represent unlabeled graphs, the comparison and representation of large, labeled graphs remains a challenge. In the worst case, no existing graph kernel scales better than $O(n^3)$ for a pair of labeled graphs of size n.

We show in Chapter 3 that it is possible to efficiently compare unlabeled or categorically labeled graphs of arbitrary size: The Weisfeiler-Lehman subtree kernel that we propose here scales linearly in the number of edges in given graphs. Note that merely reading a graph scales in the same way. A further advantage of our kernel is that while taking an inner product in the rich space of all subtree patterns up to a given height in the given graph set, we are still able to get a sparse explicit representation of each graph in the feature space. When the number of graphs is very large, such as many thousands or more, this property proves very useful as it avoids the need of storing a large kernel matrix which may not even fit in memory. Furthermore, we generalize the Weisfeiler-Lehman subtree kernel to a family of kernels for graphs with categorically labeled nodes. Our generalization makes existing graph kernels more expressive by enabling them to better capture the graph topology. Finally, we present two extensions of the Weisfeiler-Lehman subtree kernel: One for flexible matching of subtree patterns, and one for extracting topological features for comparing nodes within one large graph.

In Chapter 4, we present applications of our contributions in two different areas. First, we propose a kernel comparing pixels for remote sensing image classification. The kernel captures high order neighborhood relations in the input image and achieves competitive classification accuracy on standard remote sensing image classification tasks. Next, we present our ongoing work on using graph kernels on neighborhoods of nuclei to predict chemical shifts of nuclei based on protein structure. Unlike many existing approaches, our method takes into account the complete 3-dimensional neighborhood of nuclei and compares them using a flexible similarity measure.

We believe that our contributions have the potential to prove useful in many other applications where graph representations have not yet been considered because of the lack of scalable graph comparison methods.

We conclude this thesis with an outlook into further research in scalable machine learning on graphs. We start by presenting potential extensions of the contributions of this thesis (Section 5.1), and subsequently discuss some new directions in learning on graph data (Section 5.2).

5.1. Graph comparison and representation

Inexact matching of subtree patterns The first natural research direction that we have already started exploring in this thesis is graph comparison via inexact matching of subtree patterns while retaining the attractive, linear runtime of the Weisfeiler-Lehman subtree kernel. Solving this problem would benefit application fields where graphs tend to be noisy or incomplete, or too large for node neighborhoods to match exactly. Many graph problems in molecular biology fall into this category [see e.g., Reguly et al., 2006]. As the exact matching of subtree patterns plays a key role in the efficiency of the Weisfeiler-Lehman algorithm, this problem is indeed challenging. We proposed an efficient solution, based on approximating the Jaccard coefficient between the sets of labels of neighbors of nodes. However, we did not observe significant improvement over the standard Weisfeiler-Lehman subtree kernel on benchmark graph classification tasks, even when using exact instead of approximate Jaccard coefficients. One reason for this may be the fact that we continue using the standard Weisfeiler-Lehman relabeling from Algorithm 2 in Section 3.1. A strategy that may prove useful is to learn a non-perfect hash function for subtree patterns appropriate for a problem at hand, and use it instead. There exist several recent efficient methods for learning hash functions [e.g., Baluja and Covell, 2008, Kulis and Darrell, 2009] that could be considered to this end.

Complex node and edge labels A related, but different problem is that of efficiently comparing graphs with complex labels on nodes or edges. By complex labels we mean, for instance, real numbers, vectors, or strings. Even discrete labels can be regarded as complex, if they are not categorical: For example, if labels correspond to ratings on some discrete scale, then 4 and 5 resemble each other more than 2 and 5. Currently, comparing a pair of this type of graphs in the simplest and least expressive fashion already results in more than quadratic runtime in the number of nodes. Many applications would benefit from efficient solutions to these problems. For instance, document classification in semantic learning, where a document is represented as a graph, node labels are words, edge labels describe relations between words, and there exists structure on both word and edge label dictionaries [Rettinger et al., 2012]; or biological network analysis, where nodes are often labeled with many different types of information [Lee et al., 2008]. As we now have efficient methods for comparing and representing graphs with categorical labels, one way to address continuous node and edge labels would be to first cluster them into categorical labels while retaining as much information as possible. To achieve

this goal, one could explore recently developed discretization algorithms in data mining [e.g., Sugiyama and Yamamoto, 2011]. If labels constitute p-dimensional vectors of categorical variables, one can learn different kernels for each component and combine them, for instance, by multiple kernel learning [Lanckriet et al., 2004].

Moreover, we deem interesting several questions regarding learning on graph-structured data in general. We present them in the next section.

5.2. Feature selection on graphs

In data analysis methods for experimental sciences, the question of *interpretability* is often important. As an example, consider classifying molecules into mutagenic and non-mutagenic groups in chemoinformatics: Here, it often does not suffice to predict that a given molecule should be mutagenic, but it is as important to identify the (preferably small number of) substructures of the molecule that make it likely to be so. Algorithms that seek to select a subset of relevant variables to predict an output variable are called *feature selection* methods [Guyon and Elisseeff, 2003]. An important special case of the feature selection problem arises in the setting where features are combined linearly and one seeks to predict the output variable using only a small subset of features. This setting is referred to as *sparse linear* modeling. *Sparsity*, or *parsimony*, is a principle stating that when comparing different hypotheses that explain an event equally well, one should favor the one making the least assumptions, in other words, the simplest one. For example, if we find that the prediction of mutagenicity of molecules can be done equally accurately whether we consider all substructures of molecules or only a small subset of them, then according to the principle of parsimony we should predict using the small subset of substructures.

Feature selection on graphs is not a new problem. Most methods in the field of pattern mining, discussed in Section 1.4.2, are in fact concerned with the discovery of interpretable substructures that are relevant in one learning task or another. Sparsity in feature selection on graphs has also been explicitly addressed, for instance, in gBoost by [Saigo et al., 2009], or in our work prior to this thesis [Shervashidze and Tsuda, 2008].

Compound subgraphs One interesting problem in learning on graph-structured data is feature selection on graphs by considering relations between subgraphs. As we have seen in this thesis, most representation methods for graphs in machine learning rely on particular types of substructures, such as random walks or subtree patterns. When performing feature selection for graph classification, a graph is represented (explicitly or implicitly) as a bag of these substructures, and the learning algorithm then tries to find substructures that are important in predicting the variable of interest. The fact that these substructures often overlap is usually not taken into account by the learning algorithm. From the interpretability point of view and according to the principle of

parsimony, it would be desirable to be able to consider the relations between the variables and encourage the learning algorithm to select few groups of features forming larger substructures rather than many small substructures without any relation to each other.

Structured sparse learning with graphs This is a particular instance of a more general problem: Consider a learning problem where variables (features) are represented by vertices in a graph whose edges describe relations between variables. To predict the variable of interest (output variable), we would like to select a small number of groups of variables forming connected subgraphs in the given graph, without specifying the candidate groups in advance. This problem is part of the structured sparsity framework [Jenatton et al., 2011], and is unfortunately intractable in its most general form [Mairal and Yu, 2011]. Its special cases include the setting where the graph between variables is a chain graph, or a grid. While efficient optimization algorithms were developed for special cases where groups are sets of consecutive variables in a vector, rectangular patterns in a grid [Jacob et al., 2009, Jenatton et al., 2011], or paths in a directed acyclic graph [Mairal and Yu, 2011], it is still unclear how to efficiently select features using a sparsity-inducing norm over more general connected subgraphs of a general graph. As a first step, one could consider using efficient enumeration schemes for different classes of subgraphs developed in the field of graph kernels, and in particular in this thesis, to approach this problem.

Solving this problem would be of both theoretical and practical interest: In terms of theory, an efficient optimization scheme for general graphs would unify the optimization techniques and theory developed for chains, grids and paths [Jacob et al., 2009, Jenatton et al., 2011, Mairal and Yu, 2011]. In terms of practical applications, this would, for instance, allow to detect gene pathways involved in a phenotype, given a gene network, gene expression levels and the phenotype for several individuals.

A. Reproducing kernel Hilbert spaces

This appendix is meant to clarify the notion of reproducing kernel Hilbert spaces step by step using basic definitions in algebra and functional analysis. We assume that the reader is familiar with vector spaces and notions such as basis, span and dimension. The material given here is based on the textbooks by Schölkopf and Smola [2002] and Young [1988].

Reproducing kernel Hilbert spaces are Hilbert spaces, and Hilbert spaces are complete inner product spaces: To define completeness and inner product space, we first need to clarify some more basic notions. We start this section by introducing bilinear forms, norms and inner products.

Definition A.0.1 (Bilinear form) *A* bilinear form *on a vector space $\mathcal{H}$ is a function*

$$Q : \mathcal{H} \times \mathcal{H} \mapsto \mathbb{R}$$

such that for all $x, x', x'' \in \mathcal{H}$ and all $c, c' \in \mathbb{R}$, we have

$$Q((cx + c'x'), x'') = cQ(x, x'') + c'Q(x', x''),$$
$$Q(x, (cx' + c'x'')) = cQ(x, x') + c'Q(x, x'').$$

If Q also satisfies

$$Q(x, x') = Q(x', x)$$

for all $x, x' \in \mathcal{H}$, it is called symmetric.

Definition A.0.2 (Inner product (dot product)) *A dot product on a vector space $\mathcal{H}$ is a symmetric bilinear form,*

$$\langle ., . \rangle : \mathcal{H} \times \mathcal{H} \mapsto \mathbb{R},$$

that is strictly positive definite, that is, for all $x \in \mathcal{H}$, $\langle x, x \rangle \geq 0$, and $\langle x, x \rangle = 0$ only for $x = 0$.

Definition A.0.3 (Norm) *A function $||.|| : \mathcal{H} \mapsto \mathbb{R}_0^+$ that for all $x, x' \in \mathcal{H}$ and $c \in \mathbb{R}$ satisfies*

$$||x + x'|| \leq ||x|| + ||x'||,$$
$$||cx|| = c||x||,$$
$$||x|| > 0 \text{ if } x \neq 0,$$

is called a norm *on $\mathcal{H}$.*

Definition A.0.4 (Normed space) *A* normed space *is a vector space endowed with a norm.*

We can now introduce an inner product space.

Definition A.0.5 (Inner product space) *An* inner product space *(or* pre-Hilbert space*) is a vector space endowed with an inner product. Any inner product defines a corresponding* norm *via*

$$||x|| = \sqrt{\langle x, x \rangle},$$

therefore any inner product space is also a normed space*.*

To define the notion of completeness, we need definitions of metric space, Cauchy sequence and its convergence.

Definition A.0.6 (Metric space) *A metric space (M, d) is a set M with a metric $d : M \times M \mapsto \mathbb{R}$ such that*

$$\begin{aligned} d(x, x') &\geq 0, \\ d(x, x') &= 0 \Leftrightarrow x = x', \\ d(x, x') &= d(x', x), \\ d(x, x'') &\leq d(x, x') + d(x', x''). \end{aligned}$$

Note that any normed space $\mathcal{H}$ is also a metric space, if we define $d(x, x')$ as $||x - x'||$ for all $x, x' \in \mathcal{H}$.

Definition A.0.7 (Cauchy sequence, completeness, dense subset) *Let (M, d) be a metric space. A sequence $(x_k)_{k \in \mathbb{N}}$ in M is a* Cauchy sequence *if, for every $\epsilon > 0$, there exists an integer k_0 such that $k, l \geq k_0$ implies that $d(x_k, x_l) < \epsilon$.*

A Cauchy sequence is said to converge *to a point $x \in M$ if $d(x_k, x) \to 0$ as $k \to \infty$.*

A metric space (M, d) is complete *if all Cauchy sequences in M converge to a point in M.*

A subset of M, M', is called dense *in M, if every element in M is a limit of a Cauchy sequence in M'. Equivalently, M is the* completion *of the set M'.*

Definition A.0.8 (Hilbert space) *A* Hilbert space *is a* complete inner product space*.*

Now we can finally turn our attention to reproducing kernel Hilbert spaces. Schölkopf and Smola [2002] define them as follows:

Definition A.0.9 (Reproducing kernel Hilbert space) *Let $\mathcal{X}$ be a nonempty set and $\mathcal{H}$ a Hilbert space of functions $f : \mathcal{X} \mapsto \mathbb{R}$. Then $\mathcal{H}$ is called a* reproducing kernel Hilbert space *endowed with an inner product $\langle ., . \rangle$ and the norm $||f|| = \sqrt{\langle f, f \rangle}$ if there exists a function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ with properties:*

1. *k has the* reproducing property*, that is,*

$$\langle f, k(x,.)\rangle = f(x) \text{ for all } f \in \mathcal{H};$$

In particular,

$$\langle k(x,.), k(x',.)\rangle.$$

2. *k spans $\mathcal{H}$, i.e., $\mathcal{H}$ is the completion of the space spanned by $\{k(x,.)|x \in \mathcal{X}\}$.*

B. Counting all graphlets of size 5

In this appendix, we prove the Theorem 2.3.2 for graphlets of size 5.

Theorem *Let G be a graph, and let d denote its maximum degree. For a fixed node v_1 in G, we can count all subgraphs of size 5 containing v_1 in time $O(d^4)$.*

Proof There are 34 graphlets of 5 nodes modulo isomorphism (see Figure B.1).

Theorem 2.3.1 shows how to count connected graphlets of 5 nodes. The size distribution of the connected components of the non-connected graphlets on five nodes is $(4,1)$, $(3,2)$, $(3,1,1)$, $(2,2,1)$, $(2,1,1,1)$ or $(1,1,1,1,1)$ (see $g_{22} - g_{34}$ in Figure B.1). We will consider these cases in this order.

Let $\{v_1, v_2, v_3, v_4\}$ be the four nodes of a connected graphlet of size 4. Such graphlets can be enumerated in $O(nd^3)$. For each such graphlet $\cup_{i=1}^{4} \mathcal{N}(v_i)$ is the set of nodes where the fifth node is not allowed to come from. The cardinality of this set can be determined in $O(d)$ time.

Let us now consider graphlets with a connected component of size 3 (graphlets $g_{28} - g_{31}$). Let $\{v_1, v_2, v_3\}$ be the three nodes in this component. The connected graphlets of size 3 can be enumerated in $O(nd^2)$. Let S be the set of nodes $\cup_{i=1}^{3} \mathcal{N}(v_i)$, and $E(S)$ the set of edges $\{(u,v) \in E | u \in S \text{ or } v \in S \text{ or both}\}$. There are $O(d)$ nodes in S, and $O(d^2)$ edges in $E(S)$. The fourth and fifth nodes have to be chosen from $V \setminus S$. If we denote m the total number of edges in G, we will obtain a graphlet with connected component size distribution $(3,2)$ in $m - |E(S)|$ cases, and $(3,1,1)$ in $\binom{n-|S|}{2} - (m - |E(S)|)$ cases. $|E(S)|$ can be obtained in $O(d^2)$, making the overall complexity for counting graphlets $g_{28} - g_{31}$ $O(nd^4)$.

We now turn to graphlets with the connected component distribution $(2,2,1)$, $(2,3)$, or $(2,1,1,1)$. Graphlets g_{28}, g_{29}, g_{32} and g_{33} fall into this category. Let $e = (v_1, v_2)$ be the 2-node connected component (i.e., edge) of such a graphlet. For an edge $e = (v_1, v_2)$, we let S_e be the set $\mathcal{N}(v_1) \cup \mathcal{N}(v_2)$, and $E(S_e)$ the set of edges with at least one endpoint in S_e. Observe that for a given edge e, the numbers of occurrences of graphlets of type g_{28}, g_{29}, g_{32} or g_{33} containing the edge e sums to $\binom{|V \setminus S|}{3}$. We have showed in the previous paragraph how to count the occurrences of graphlets consisting of two connected components of sizes 2 and 3 (graphlets g_{28} and g_{29}). We will show here how

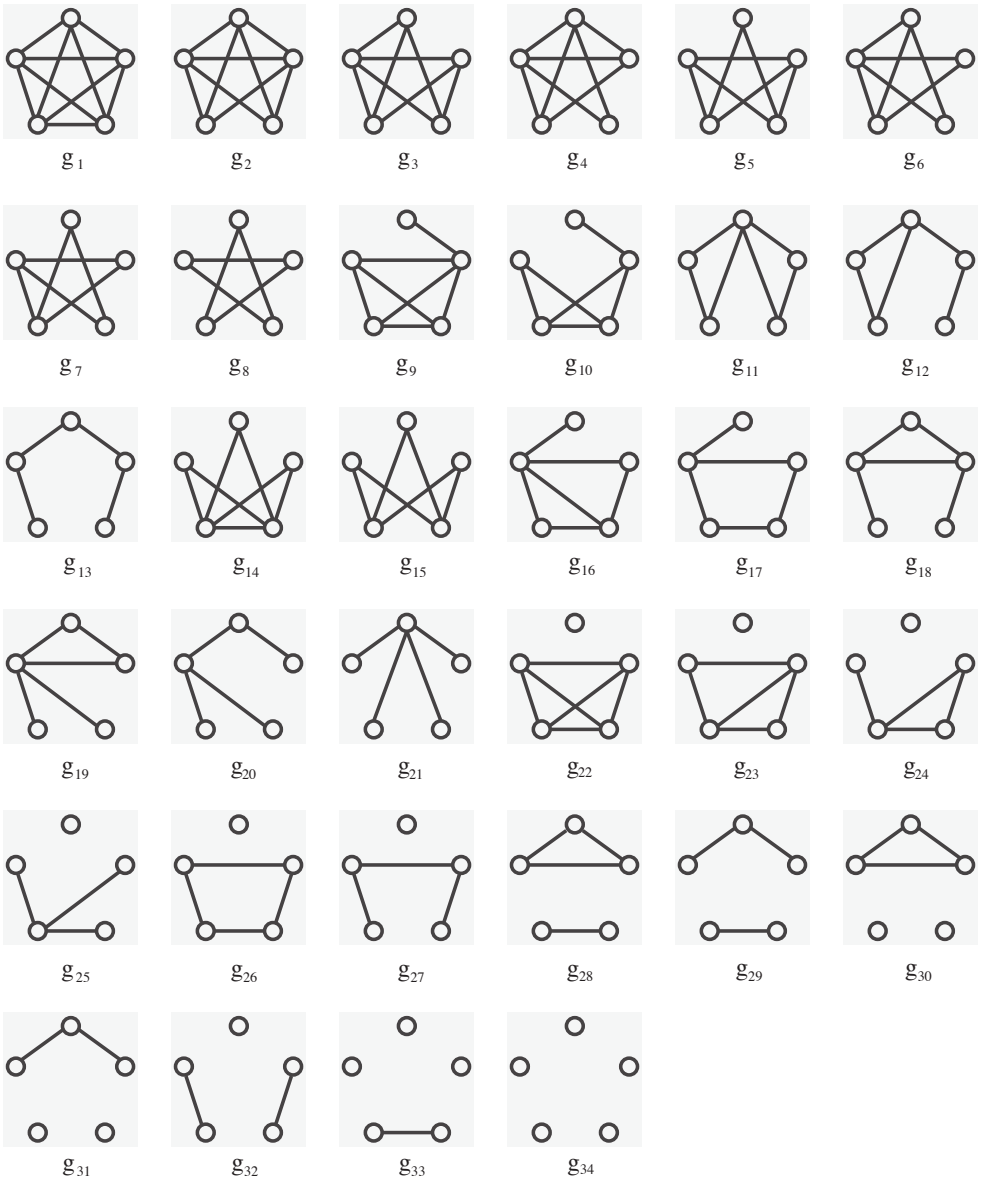

Figure B.1.: All graphlets of size 5.

to count graphlets of type g_{32}, and get the number of graphlets of type g_{33} by

$$c_{33}(G) = \sum_{e \in E} \binom{n - |S_e|}{3} - [c_{28}(G) + c_{29}(G) + 2c_{32}(G)].$$

Note that $c_{32}(G)$ has to be subtracted twice as g_{32} has two isolated edges. Given an edge e, we need to choose the remaining three nodes in $V \setminus S_e$. If we knew for each edge $e' = (v_3, v_4) \in E \setminus E(S_e)$ the cardinality of the set $S'_{e'} = (\mathcal{N}(v_3) \cup \mathcal{N}(v_4)) \setminus S_e$ (number of nodes in the neighborhood of v_3 and v_4 in the graph induced by nodes in $V \setminus S_e$), we could obtain the number of graphlets of type g_{32} containing e by the sum $\sum_{e' \in E \setminus E(S_e)} (n - |S'_{e'}|)$. This quantity can be obtained in the following fashion: First, for each integer $c \leq 2d - 1$ precompute the number m_c of edges e' such that $|S_{e'}| = c$. This can be done in $O(nd^2)$ time. For each edge $e = (v_1, v_2)$, iterate over edges e' in $E(S_e)$ and set $m_{|S_{e'}|,e}$ to $m_{|S_{e'}|} - 1$. For e, we can choose another edge outside S_e and an isolated node in $V \setminus S_e$ in

$$\sum_{c=2}^{2d-1} m_{c,e}(n - |S_e| - c)$$

different ways. At last, we set

$$c_{34}(G) = \binom{n}{5} - \sum_{i=1}^{33} c_i(G).$$

■

List of Figures

List of Tables

Bibliography

J. Abernethy, O. Chapelle, and C. Castillo. Webspam identification through content and hyperlinks. In *Proceedings of the International Workshop on Adversarial Information Retrieval on the Web*, 2008.

L. Adamic and N. Glance. The political blogosphere and the 2004 U.S. election. In *Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem*, 2005.

R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

A. Aizerman, E. M. Braverman, and L. I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

T. Anastasakos, D. Hillard, S. Kshetramade, and H. Raghavan. A collaborative filtering approach to ad recommendation using the query-ad click graph. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1927–1930, 2009.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68, 1950.

L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Proceedings Symposium on Foundations of Computer Science*, pages 39–46, 1979.

F. R. Bach. Graph kernels between point clouds. In *Proceedings of the International Conference on Machine Learning*, pages 25–32, 2008.

C. M. Bachmann, T. L. Ainsworth, and R. A. Fusina. Exploiting manifold geometry in hyperspectral imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 43(3): 441–454, 2005.

S. Baluja and M. Covell. Learning to hash: forgiving hash functions and applications. *Data Min. Knowl. Discov.*, 17(3):402–430, 2008.

N. Baskiotis and M. Sebag. Structural statistical software testing with active learning in a graph. In *Proceedings of the 17th international conference on Inductive logic programming*, pages 49–62, 2008.

J. A. Benediktsson, M. Pesaresi, and K. Arnason. Classification and feature extraction for remote sensing images from urban areas based on morphological transformations. *IEEE Transactions on Geoscience and Remote Sensing*, 41(9):1940–1949, 2003.

J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson. Classification of hyperspectral data from urban areas based on extended morphological profiles. *IEEE Transactions on Geoscience and Remote Sensing*, 42(3):480–491, 2005.

S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social network data analytics*, pages 115–148. Springer, 2011.

K. M. Borgwardt. *Graph kernels*. PhD thesis, Ludwig-Maximilians University, Munich, 2007.

K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the International Conference on Data Mining*, pages 74–81, 2005.

K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(Suppl 1):i47–i56, Jun 2005.

K. M. Borgwardt, T. Petri, S. V. N. Vishwanathan, and H.-P. Kriegel. An efficient sampling scheme for comparison of large graphs. In *Mining and Learning with Graphs*, 2007.

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Annual Conference on Computational Learning Theory*, pages 144–152, 1992.

Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *Proceedings of the International Conference on Computer Vision*, pages 105–112, 2001.

A. Z. Broder. On the resemblance and containment of documents. In *SEQS: Sequences 91.*, 1998.

A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, 1997.

A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253, 1983.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

G. Camps-Valls and L. Bruzzone. Kernel-based methods for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1351–1362, 2005.

G. Camps-Valls and L. Bruzzone, editors. *Kernel methods for Remote Sensing Data Analysis*. Wiley & Sons, UK, 2009.

G. Camps-Valls, L. Gómez-Chova, J. Muñoz-Marí, J. Vila-Francés, and J. Calpe-Maravilla. Composite kernels for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 3(1):93–97, 2006.

G. Camps-Valls, T. Bandos, and D. Zhou. Semisupervised graph-based hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 45(10): 3044–3054, 2007.

G. Camps-Valls, L. Gómez-Chova, M. Muñoz Marí, J.and Martínez-Ramón, and J. L. Rojo-Álvarez. Kernel-based framework for multi-temporal and multi-source remote sensing data classification and change detection. *IEEE Transactions on Geoscience and Remote Sensing*, 46(6):1822–1835, 2008.

P. J. Carrington, J. Scott, and L. Wasserman. *Models and methods in social network analysis*. Structural analysis in the social sciences. Cambridge University Press, 2005.

S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. *SIGMOD Record*, 27:307–318, 1998.

C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

C. Cortes, P. Haffner, and M. Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062, 2004.

F. Costa and K. De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the International Conference on Machine Learning*, pages 255–262, 2010.

N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 239–246, 2007.

A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34:786–797, 1991.

R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer-Verlag, Heidelberg, fourth edition, 2010.

P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, Jul 2003.

R. C. Dubes and A. K. Jain. Random field models in image analysis. *Journal of Applied Statistics*, 16(2):131–163, 1989.

A. Feragen, P. Lo, V. Gorbunova, M. Nielsen, A. Dirksen, F. Lauze, and M. de Bruijne. An airway tree-shape model for geodesic airway branch labeling. In *Mathematical Foundations of Computational Anatomy*, 2011.

R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised Learning in Gigantic Image Collections. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2009.

M. Fligner, J. Verducci, and P. Blower. A modification of the Jaccard/Tanimoto similarity index for diverse selection of chemical compounds using binary strings. *Technometrics*, 44:110–119, 2002.

G. M. Foody. Thematic map comparison: Evaluating the statistical significance of differences in classification accuracy. *Photogrammetric Engineering & Remote Sensing*, 70(5):627–633, May 2004.

F. Fouss, L. Yen, A. Pirotte, and M. Saerens. An experimental investigation of graph kernels on a collaborative recommendation task. In *Proceedings of the International Conference on Data Mining*, pages 863–868, 2006.

H. Fröhlich, J. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the International Conference on Machine Learning*, pages 225–232, Bonn, Germany, 2005.

M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the Annual Conference on Computational Learning Theory*, pages 129–143, 2003.

T. Gärtner, Q. V. Le, S. Burton, A. J. Smola, and S. V. N. Vishwanathan. Large-scale multiclass transduction. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2006.

J. Gasteiger and T. Engel, editors. *Chemoinformatics. A Textbook*. Wiley-VCH, 2003.

D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, pages 721–732, 2005.

A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th VLDB Conference*, pages 518–529, 1999.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

L. Gómez-Chova, G. Camps-Valls, J. Muñoz-Marí, and J. Calpe-Maravilla. Semisupervised image classification with Laplacian support vector machines. *IEEE Geoscience and Remote Sensing Letters*, 5(3):336–340, 2008.

L. Gómez-Chova, G. Camps-Valls, L. Lorenzo Bruzzone, and J. Calpe-Maravilla. Mean map kernel methods for semisupervised cloud classification. *IEEE Transactions on Geoscience and Remote Sensing*, 48(1-1):207–220, 2010.

J. C. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–871, 1971.

I. Gutman and N. Trinajstić. Total π-electron energy of alternant hydrocarbons. *Chemical Physics Letters*, 17:535–538, 1971.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learnng Research*, 3:1157–1182, 2003.

B. Han, Y. Liu, S. W. Ginzinger, and D. S. Wishart. SHIFTX2: significantly improved protein chemical shift prediction. *Journal of biomolecular NMR*, 50(1):43–57, 2011.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 1–12, 2000.

Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, UC Santa Cruz, 1999.

S. Hido and H. Kashima. A linear-time graph kernel. In *Proceedings of the International Conference on Data Mining*, pages 179–188, 2009.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.

T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 158–167, 2004.

P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the* 30^{th} *Symposium on Theory of Computing*, pages 604–613, 1998.

J. Inglada. Automatic recognition of man-made objects in high resolution optical remote sensing images by SVM classification of geometric image features. *ISPRS Journal of Photogrammetry Rem. Sens.*, 62:236–248, 2007.

A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *Principles of Data Mining and Knowledge Discovery*, 73(2):13–23, 2000.

Q. Jackson and D. Landgrebe. Adaptive bayesian contextual classification based on Markov random fields. *IEEE Transactions on Geoscience and Remote Sensing*, 40 (11):2454–2463, 2002.

L. Jacob, G. Obozinski, and J.-P. Vert. Group lasso with overlap and graph lasso. In *Proceedings of the International Conference on Machine Learning*, pages 433–440, 2009.

R. Jenatton, J.-Y. Audibert, and F. Bach. Structured variable selection with sparsity-inducing norms. *Journal of Machine Learning Research*, 12:2777–2824, November 2011.

L. Jiang and Z. Su. Context-aware statistical debugging: from bug predictors to faulty control flow paths. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 184–193, 2007.

B. H. Junker and F. Schreiber. *Analysis of Biological Networks*. Wiley Series on Bioinformatics: Computational Techniques and Engineering. Wiley-Interscience, 2008.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the International Conference on Machine Learning*, 2003.

H. Kashima, K. Tsuda, and A. Inokuchi. Kernels for graphs. In B. Schölkopf, K. Tsuda, and J.P. Vert, editors, *Kernel methods in computational biology*, pages 155–170. MIT Press, 2004.

N. Kashtan, S. Itzkovitz, R. Milo, and U. Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20 (11):1746–1758, 2004.

P. J. Kelly. A congruence theorem for trees. *Pacific Journal of Mathematics*, 7(1): 961–968, 1957.

K. J. Kohlhoff, P. Robustelli, A. Cavalli, X. Salvatella, and M. Vendruscolo. Fast and accurate predictions of protein nmr chemical shifts from interatomic distances. *Journal of American Chemical Society*, 131(39):1389413895, 2009.

I. R. Kondor and K. M. Borgwardt. The skew spectrum of graphs. In *Proceedings of the International Conference on Machine Learning*, pages 496–503, 2008.

I. R. Kondor and J. D. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *Proceedings of the International Conference on Machine Learning*, pages 315–322, 2002.

I. R. Kondor, N. Shervashidze, and K. M. Borgwardt. The graphlet spectrum. In *Proceedings of the International Conference on Machine Learning*, pages 529–536, 2009.

B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Proceedings of the Conference on Advances in Neural Information Processing Systems*, pages 1042–1050, 2009.

G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

L. L. Lapin. *Probability and Statistics for Modern Engineering*. Waveland Press, 1998.

I. Lee, B. Lehner, C. Crombie, W. Wong, A. G. Fraser, and E. M. Marcotte. A single gene network accurately predicts phenotypic effects of gene perturbation in Caenorhabditis elegans. *Nature genetics*, 40(2):181–188, 2008.

T. M. Lillesand, R. W. Kiefer, and J. W. Chipman. *Remote Sensing and Image Interpretation*. John Wiley, New York, 5th edition, 2004.

P. Lorenz, M. Kreutzer, J. Zerweck, M. Schutkowski, and H. J. Thiesen. Probing the epitope signatures of igg antibodies in human serum from patients with autoimmune disease. *Methods in Molecular Biology*, 2009.

P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, 2009.

P. Mahé, N. Ueda, T. Akutsu, J. Perret, and J. Vert. Extensions of marginalized graph kernels. In *Proceedings of the International Conference on Machine Learning*, pages 552–559, Alberta, Canada, 2004.

J. Mairal and B. Yu. Path coding penalties for directed acyclic graphs. In *NIPS workshop on optimization for machine learning*, 2011.

M. Marconcini, G. Camps-Valls, and L. Bruzzone. A composite semisupervised SVM for classification of hyperspectral images. *IEEE Geoscience and Remote Sensing Letters*, 6(2):234–238, 2009.

J. Matas, R. Marik, and J. Kittler. The color adjacency graph representation of multi-coloured objects. Technical Report VSSP-TR-1/95, University of Surrey, 1995.

B. McKay. Small graphs are reconstructible. *Australasian Journal of Combinatorics*, 15:123–126, 1997.

K. Mehlhorn. *Data Structures and Efficient Algorithms*. Springer, 1984.

K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, Berlin, 2008.

J. Meiler. PROSHIFT: protein chemical shift prediction using artificial neural networks. *Journal of Biomolecular NMR*, 26(1):25–37, 2003.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society*, A 209(441-458): 415–446, 1909.

B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5): 493–504, 1998.

S. P. Mielke and V. V. Krishnan. Characterization of protein secondary structure from NMR chemical shifts. *Progress in nuclear magnetic resonance spectroscopy*, 54(3-4): 141–165, 2009.

T. Milenkovic, W. L. Ng, W. Hayes, and N. Przulj. Optimal network alignment with graphlet degree vectors. *Cancer Informatics*, 9, 2010.

H. L. Morgan. The generation of unique machine description for chemical structures - a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965.

S. Neal, A. M. Nip, H. Zhang, and D. S. Wishart. Rapid and accurate calculation of protein 1H, 13C and 15N chemical shifts. *Journal of biomolecular NMR*, 26:215–240, 2003.

M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3): 503–514, 2005.

M. Neuhaus and H. Bunke. Edit distance-based kernel functions for structural pattern classification. *Pattern Recognition*, 39(10):1852–1863, 2006.

M. Neuhaus and H. Bunke. *Bridging the Gap Between Graph Edit Distance and Kernel Machines*. World Scientific Publishing Co., Inc., 2007.

J. Neville and D. Jensen. Iterative classification in relational data. In *Proceedings of the AAAI Workshop Learning Statistical Models from Relational Data*, 2000.

S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 647–652, 2004.

E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin. UCSF Chimera-a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.

A. Plaza, J. A. Benediktsson, J. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, A. Gualtieri, and J. C. Tilton. Recent advances in techniques for hyperspectral image processing. *Remote Sensing of Environment*, 113(S1):110–122, 2009.

N. Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007.

L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD'03)*, 2003.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2005.

T. Reguly, A. Breitkreutz, L. Boucher, B.-J. Breitkreutz, G. C. Hon, C. L. Myers, A. Parsons, H. Friesen, R. Oughtred, A. Tong, C. Stark, Y. Ho, D. Botstein, B. Andrews, C. Boone, O. G. Troyanskya, T. Ideker, K. Dolinski, N. N. Batada, and M. Tyers. Comprehensive curation and analysis of global interaction networks in saccharomyces cerevisiae. *Journal of Biology*, 5:11, 2006.

A. Rettinger, U. Lösch, V. Tresp, C. d'Amato, and N. Fanizzi. Mining the semantic web - statistical learning for next generation knowledge bases. *Data Mining and Knowledge Discovery*, 2012.

W. Rieping and W. F. Vranken. Validation of archived chemical shifts through atomic coordinates. *Proteins: Structure, Function, and Bioinformatics*, 78(11):2482–2489, 2010.

G. Roberts and L.Y. Lian. *Protein NMR Spectroscopy: Practical Techniques and Applications*. John Wiley & Sons, 2011.

A. Rosenfeld. Adjacency in digital pictures. *Information and Control*, 26(1):24–33, 1974.

H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. gboost: a mathematical programming approach to graph classification and regression. *Machine Learning*, 75 (1):69–89, 2009.

A. Sanfeliu and K.-S. Fu. A Distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, 13(3): 353–362, 1983.

B. Schölkopf. *Support vector learning*. Oldenbourg Verlag, Munich, 1997.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, 2000.

B. Schölkopf, K. Tsuda, and J.-P. Vert. *Kernel methods in computational biology*. MIT press, 2004.

I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32D:431–433, 2004.

J. Shawe-Taylor. Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, 4:816–826, 1993.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

Y. Shen and A. Bax. Protein backbone chemical shifts predicted from searching a database for torsion angle and sequence homology. *Journal of biomolecular NMR*, 38: 289–302, 2007.

N. Shervashidze and K. Tsuda. Logistic regression for graph classification. In *NIPS workshop on Structured Input and Structured Output*, 2008.

N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In D. van Dyk and M. Welling, editors, *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 488–495, 2009.

A. J. Smola and I. R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, editors, *Proceedings of the Annual Conference on Computational Learning Theory*, 2003.

P. Soille. *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 2003.

B. Solaiman, R. Koffi, M. Mouchot, and A. Hillion. An information fusion method for multispectral image classification postprocessing. *IEEE Transactions on Geoscience and Remote Sensing*, 36(2):395–406, 1998.

O. Stegle, N. Fusi, and M. Zwiessele. Gaussian process toolbox for python, 2011. URL `https://github.com/PMBio/pygp`.

I. Steinwart and A. Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008.

F. Suard, V. Guigue, A. Rakotomamonjy, and A. Benshrair. Pedestrian detection using stereo-vision and graph kernels. In *IEEE Symposium on Intelligent Vehicles*, 2005.

M. Sugiyama and A. Yamamoto. A fast and flexible clustering algorithm using binary discretization. In *Proceedings of the International Conference on Data Mining*, pages 1212–1217, 2011.

M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *Proceedings of the SIAM International Conference on Data Mining*, 2009.

R. Todeschini and V. Consonni. *Handbook of molecular descriptors*. Wiley-VCH, 2000.

I. Tomas. Spatial postprocessing of spectrally classified Landsat data. *Photogrammetic Engineering and Remote Sensing*, 46:1201–1206, 1980.

J. Ton, J. Sticklen, and A. Jain. Knowledge-based segmentation of Landsat images. *IEEE Transactions on Geoscience and Remote Sensing*, 29:222–231, 1991.

D. Tuia and G. Camps-Valls. Semisupervised remote sensing image classification with cluster kernels. *IEEE Geoscience and Remote Sensing Letters*, 6(2):224–228, 2009.

D. Tuia, F. Ratle, A. Pozdnoukhov, and G. Camps-Valls. Multisource composite kernels for urban image classification. *IEEE Geoscience and Remote Sensing Letters*, 7(1): 88–92, 2010.

V. Vacic, L. M. Iakoucheva, S. Lonardi, and P. Radivojac. Graphlet kernels for prediction of functional residues in protein structures. *Journal of Computational Biology*, 17(1): 55–72, 2010.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

J.-P. Vert. The optimal assignment kernel is not positive definite. *CoRR*, abs/0801.4061, 2008.

J.-P. Vert, T. Matsui, S. Satoh, and Y. Uchiyama. High-level feature extraction using svm with walk-based graph kernel. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1121–1124, 2009.

S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the International Conference on Data Mining*, pages 678–689, Hong Kong, 2006.

S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Structural analysis in the social sciences. Cambridge University Press, 1 edition, 1994.

B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, Ser. 2*, 9, 1968.

T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. J. Weinberger. Inequalities for the l_1 deviation of the empirical distribution. Technical Report HPL-2003-97(R.1), HP Labs, Palo Alto, 2003.

S. Wernicke. A faster algorithm for detecting network motifs. In *Proceedings of the workshop on algorithms in bioinformatics*, pages 165–177, 2005.

H. Wiener. Structural determination of paraffin boiling points. *Journal of American Chemical Society*, 69(1):17–20, 1947.

X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724, 2002.

N. Young. *An Introduction to Hilbert Space*. Cambridge Mathematical Textbooks. Cambridge University Press, 1988.

Y. Zhang. Detection of urban housing development by fusing multisensor satellite data and performing spatial feature post-classification. *International Journal of Remote Sensing*, 22(17):3339–3355, 2001.

D. Zhou, O. Bousquet, T. N. Lal, Jason W., and B. Schölkopf. Learning with local and global consistency. In *Proceedings of the Conference on Advances in Neural Information Processing Systems*, 2004.

X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the International Conference on Machine Learning*, pages 912–919, 2003.