GRAPH KERNELS

Karsten Michael Borgwardt

München 2007

GRAPH KERNELS

Karsten Michael Borgwardt

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Karsten Michael Borgwardt
aus Kaiserslautern

München, den 22.05.2007

Erstgutachter: Prof. Dr. Hans-Peter Kriegel
Zweitgutachter: Prof. Dr. Bernhard Schölkopf
Tag der mündlichen Prüfung: 05.07.2007

Contents

Acknowledgments

Many individuals and institutions contributed in many different ways to the completion of this thesis. I am deeply grateful for their support, and thankful for the unique chances this support offered me.

Prof. Hans-Peter Kriegel financed my research assistant position and my numerous trips to conferences. He also encouraged me to give a lecture on kernels in the second year of my PhD studies. With his decades of experience, he has been a guide and helpful source of advice during this time. I am greatly thankful for all that, and for his wise support over the last 2 years.

Alexander Smola and SVN "Vishy" Vishwanthan, although located at the other end of the world, were teachers of mine during this time. It has been a unique chance for me to learn from their scientific experience, their vast knowledge base and their never-ending pursuit of scientific discovery. Special thanks to Alex and NICTA for funding my trip to Australia in September 2006.

My research has profited a lot from interacting with some of the best researchers in my field. I am thankful to all of them: Arthur Gretton, Hans-Peter Kriegel, Quoc V. Le, Cheng Soon Ong, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Le Song, Xifeng Yan and SVN Vishwanathan. Prof. Bernhard Schölkopf also kindly agreed to act as second examiner of this thesis.

I will remember the good collaboration with my colleagues, both in teaching and research: Elke Achtert, Johannes Aßfalg, Stefan Brecheisen, Peer Kröger, Peter Kunath, Christian Mahrt, Alexey Pryakhin, Matthias Renz, Matthias Schubert, Steffi Wanka, Arthur Zimek, and Prof. Christian Böhm. I would also like to thank our chair secretary, Susanne Grienberger, and our technician, Franz Krojer, for keeping our group and our hardware equipment organized and running during my PhD studies.

I enjoyed the enthusiasm for science shown by the students I directly supervised during my PhD. I am proud of Sebastian Böttger, Christian Hübler, Nina Meyer, Tobias Petri, Marisa Thoma, Bianca and Peter Wackersreuther who all managed to produce publication-quality level results in their student projects and theses. I am happy to have supervised these dedicated students.

Apart from individuals, I would also like to thank two institutions for their support: the Stiftung Maximilianeum that offered me board and lodging during my undergraduate studies, and the Studienstiftung des deutschen Volkes that accepted me both during my undergraduate and my PhD studies as its scholar.

I am grateful to SVN "Vishy" Vishwanathan and Quoc V. Le for proofreading parts of

this manuscript.

More than to anyone else, I owe to the love and support of my family: My mother Doris, my father Karl Heinz, my brother Steffen, my grandparents, and my girlfriend Ruth. Despite all graph kernels, you are the best part of my life.

Zusammenfassung

Data Mining und Maschinelles Lernen befinden sich inmitten einer "strukturierten Revolution". Nach Jahrzehnten, in denen unabhängige und gleichverteilte Daten im Zentrum des Interesses standen, wenden sich viele Forscher nun Problemen zu, in denen Daten Sammlungen von Objekten darstellen, die miteinander in Beziehungen stehen, oder durch einen komplexen Graphen miteinander verbunden sind. [Übersetzt aus dem Englischen, aus dem Call for Papers der Tagung Mining and Learning on Graphs (MLG'07)*]*

Da ständig neue Daten in Form von Graphen erzeugt werden, sind Lernen und Data Mining auf Graphen zu einer wichtigen Herausforderung in Anwendungsgebieten wie der Molekularbiologie, dem Telekommunikationswesen, der Chemoinformatik und der Analyse sozialer Netzwerke geworden. Die zentrale algorithmische Frage in diesen Bereichen, der Vergleich von Graphen, hat daher in der jüngsten Vergangenheit viel Interesse auf sich gezogen. Bedauerlicherweise sind die vorhandenen Verfahren langsam, ignorieren wichtige topologische Informationen, oder sind schwer zu parametrisieren.

Graph-Kerne wurden als ein theoretisch-fundierter und vielversprechender neuer Ansatz zum Vergleich von Graphen vorgeschlagen. Ihre Attraktivität liegt darin begründet, dass durch das Definieren eines Kerns auf Graphen eine ganze Familie von Lern- und Mining-Algorithmen auf Graphen anwendbar wird. Diese Graph-Kerne müssen sowohl die Topologie als auch die Attribute der Knoten und Kanten der Graphen berücksichtigen, und gleichzeitig sollen sie effizient zu berechnen sein. Die vorhandenen Graph-Kerne werden diesen Anforderungen keineswegs gerecht: sie vernachlässigen wichtige Teile der Struktur der Graphen, leiden unter Laufzeitproblemen und können nicht auf große Graphen angewendet werden. Das vorrangige Ziel dieser Arbeit war es daher, effizientes Lernen und Data Mining mittels Graph-Kernen zu ermöglichen.

In der ersten Hälfte dieser Arbeit untersuchen wir die Nachteile moderner Graph-Kerne. Anschließend schlagen wir Lösungen vor, um diese Schwächen zu überwinden. Höhepunkte unserer Forschung sind

- die Beschleunigung des klassischen Graph-Kerns basierend auf Random-Walks, auf theoretischer Ebene von $O(n^6)$ auf $O(n^3)$ (wobei n die Anzahl der Knoten im größeren der beiden Graphen ist) und auf experimenteller Ebene um bis zu das Tausendfache,

- die Definition neuer Graph-Kerne basierend auf kürzesten Pfaden, die in unseren Experimenten schneller als Random-Walk-Kerne sind und höhere Klassifikationsge-

nauigkeiten erreichen,

- die Entwicklung von Graph-Kernen, die die Häufigkeit kleiner Subgraphen in einem großen Graphen schätzen, und die auf Graphen arbeiten, die aufgrund ihrer Größe bisher nicht von Graph-Kernen bearbeitet werden konnten.

In der zweiten Hälfte dieser Arbeit stellen wir algorithmische Lösungen für zwei neuartige Probleme im Graph-Mining vor. Als Erstes definieren wir einen Zwei-Stichproben-Test für Graphen. Wenn zwei Graphen gegeben sind, lässt uns dieser Test entscheiden, ob diese Graphen mit hoher Wahrscheinlichkeit aus derselben zugrundeliegenden Verteilung hervorgegangen sind. Um dieses Zwei-Stichproben-Problem zu lösen, definieren wir einen kern-basierten statistischen Test. Dieser führt in Verbindung mit Graph-Kernen zum ersten bekannten Zwei-Stichproben-Test auf Graphen.

Als Zweites schlagen wir einen theoretisch-fundierten Ansatz vor, um überwachte Feature-Selektion auf Graphen zu betreiben. Genau wie die Feature-Selektion auf Vektoren zielt die Feature-Selektion auf Graphen darauf ab, Features zu finden, die mit der Klassenzugehörigkeit eines Graphen korrelieren. In einem ersten Schritt definieren wir eine Familie von überwachten Feature-Selektions-Algorithmen, die auf Kernen und dem Hilbert-Schmidt Unabhängigkeitskriterium beruhen. Dann zeigen wir, wie man dieses Prinzip der Feature-Selektion auf Graphen erweitern kann, und wie man es mit gSpan, dem modernsten Verfahren zur Suche von häufigen Subgraphen, kombinieren kann. Auf mehreren Vergleichsdatensätzen gelingt es unserem Verfahren, unter den Tausenden und Millionen von Features, die gSpan findet, eine kleine informative Untermenge von Dutzenden von Features auszuwählen. In unseren Experimenten werden mit diesen Features durchweg höhere Klassifikationsgenauigkeiten erreicht als mit Features, die andere Feature-Selektions-Algorithmen auf denselben Datensätzen bevorzugen.

Im Rahmen der Entwicklung dieser Verfahren müssen wir mehrere Probleme lösen, die für sich selbst genommen ebenfalls Beiträge dieser Arbeit darstellen:

- Wir vereinigen beide Varianten der Random-Walk-Graph-Kerne, die in der Literatur beschrieben sind, in einer Formel.

- Wir zeigen den ersten theoretischen Zusammenhang zwischen Graph-Kernen und topologischen Deskriptoren aus der Chemoinformatik auf.

- Wir bestimmen die Stichprobengröße, die erforderlich ist, um die Häufigkeit bestimmter Subgraphen innerhalb eines großen Graphen mit einem festgelegten Präzisions- und Konfidenzlevel zu ermitteln. Dieses Verfahren kann zur Lösung von wichtigen Problemen im Data Mining und in der Bioinformatik beitragen.

Drei Zweige der Informatik profitieren von unseren Ergebnissen: das Data Mining, das Maschinelle Lernen und die Bioinformatik. Im Data Mining ermöglichen unsere effizienten Graph-Kerne nun die Anwendung der großen Familie von Kern-Verfahren auf Probleme im Graph-Mining. Dem Maschinellen Lernen bieten wir die Gelegenheit, fundierte theoretische Ergebnisse im Lernen auf Graphen in nützliche Anwendungen umzusetzen. Der

Bioinformatik steht nun ein ganzes Arsenal an Kern-Verfahren und Kern-Funktionen auf Graphen zur Verfügung, um biologische Netzwerke und Proteinstrukturen zu vergleichen. Neben diesen können auch weitere Wissenschaftszweige Nutzen aus unseren Ergebnissen ziehen, da unsere Verfahren allgemein einsetzbar und nicht auf eine spezielle Art von Anwendung eingeschränkt sind.

Abstract

Data Mining and Machine Learning are in the midst of a "structured revolution". After many decades of focusing on independent and identically-distributed (iid) examples, many researchers are now studying problems in which examples consist of collections of interrelated entities or are linked together into complex graphs. [From Mining and Learning on Graphs (MLG'07)*: Call for Papers]*

As new graph structured data is constantly being generated, learning and data mining on graphs have become a challenge in application areas such as molecular biology, telecommunications, chemoinformatics, and social network analysis. The central algorithmic problem in these areas, measuring similarity of graphs, has therefore received extensive attention in the recent past. Unfortunately, existing approaches are slow, lacking in expressivity, or hard to parameterize.

Graph kernels have recently been proposed as a theoretically sound and promising approach to the problem of graph comparison. Their attractivity stems from the fact that by defining a kernel on graphs, a whole family of data mining and machine learning algorithms becomes applicable to graphs.

These kernels on graphs must respect both the information represented by the topology and the node and edge labels of the graphs, while being efficient to compute. Existing methods fall woefully short; they miss out on important topological information, are plagued by runtime issues, and do not scale to large graphs. Hence the primary goal of this thesis is to make learning and data mining with graph kernels feasible.

In the first half of this thesis, we review and analyze the shortcomings of state-of-the-art graph kernels. We then propose solutions to overcome these weaknesses. As highlights of our research, we

- speed up the classic random walk graph kernel from $O(n^6)$ to $O(n^3)$, where n is the number of nodes in the larger graph, and by a factor of up to 1,000 in CPU runtime, by extending concepts from Linear Algebra to Reproducing Kernel Hilbert Spaces,
- define novel graph kernels based on shortest paths that avoid tottering and outperform random walk kernels in accuracy,
- define novel graph kernels that estimate the frequency of small subgraphs within a large graph and that work on large graphs hitherto not handled by existing graph kernels.

In the second half of this thesis, we present algorithmic solutions to two novel problems in graph mining. First, we define a two-sample test on graphs. Given two sets of graphs, or a pair of graphs, this test lets us decide whether these graphs are likely to originate from the same underlying distribution. To solve this so-called two-sample-problem, we define the first kernel-based two-sample test. Combined with graph kernels, this results in the first two-sample test on graphs described in the literature.

Second, we propose a principled approach to supervised feature selection on graphs. As in feature selection on vectors, feature selection on graphs aims at finding features that are correlated with the class membership of a graph. Towards this goal, we first define a family of supervised feature selection algorithms based on kernels and the Hilbert-Schmidt Independence Criterion. We then show how to extend this principle of feature selection to graphs, and how to combine it with gSpan, the state-of-the-art method for frequent subgraph mining. On several benchmark datasets, our novel procedure manages to select a small subset of dozens of informative features among thousands and millions of subgraphs detected by gSpan. In classification experiments, the features selected by our method outperform those chosen by other feature selectors in terms of classification accuracy.

Along the way, we also solve several problems that can be deemed contributions in their own right:

- We define a unifying framework for describing both variants of random walk graph kernels proposed in the literature.
- We present the first theoretical connection between graph kernels and molecular descriptors from chemoinformatics.
- We show how to determine sample sizes for estimating the frequency of certain subgraphs within a large graph with a given precision and confidence, which promises to be a key to the solution of important problems in data mining and bioinformatics.

Three branches of computer science immediately benefit from our findings: data mining, machine learning, and bioinformatics. For data mining, our efficient graph kernels allow us to bring to bear the large family of kernel methods to mining problems on real-world graph data. For machine learning, we open the door to extend strong theoretical results on learning on graphs into useful practical applications. For bioinformatics, we make a number of principled kernel methods and efficient kernel functions available for biological network comparison, and structural comparisons of proteins. Apart from these three areas, other fields may also benefit from our findings, as our algorithms are general in nature and not restricted to a particular type of application.

Chapter 1

Introduction: Why Graph Kernels?

1.1 Motivation

Graphs are universal data structures. This claim can be justified both from a philosophical and an algorithmic point of view.

In general, a graph models a network of relationships between objects. This is interesting for two reasons: First, from a system-wide perspective, a graph represents a system and the interactions between its components. Second, from a component-centered point of view, a graph describes all relationships that link this object to the rest of the system. The philosophical relevance stems from the fact that one may argue that all real-world objects may be described either as a network of interactions of its subcomponents, or as components of a larger network. Interestingly, even philosophers argue that a graph is the best way of describing the world as a mathematical structure [Dipert, 1997].

From an algorithmic perspective, graphs are the most general data structures, as all common data types are simple instances of graphs. To name a few among many examples: A scalar can be modeled as a graph with one single node labeled by the value of this scalar. Vectors and matrices can be modeled as graphs, with one node per entry and edges between consecutive components within a vector and matrix, respectively. A time series of vectors can be represented as a graph that contains one node per time step, and consecutive steps are linked by an edge. A string is a graph in which each node represents one character, and consecutive characters are connected by an edge.

Given their generality, the natural question to ask is: Why have graphs not been the common data structure in computer science for decades? The answer is simple: Their comparison is computationally expensive. Graphs are prisoners of their own their flexibility.

On the one hand, graphs are very flexible, as they allow to compare objects of arbitrary sizes to each other. Distance functions on feature vectors are more restrictive, as they require two objects to be of equal dimension. On the other hand, for vectors, the Euclidean metric serves as a *gold standard* among all distance functions, *i.e.,* it is widely accepted and used, and can be computed efficiently. But there is no such universally accepted metric on graphs, which could be computed efficiently. The problem here is that in order to identify common parts of two graphs, we have to consider all their subgraphs. Unfortunately, in a graph with n nodes, there are always 2^n possible subsets of nodes. Hence our search space is exponential in the size of the graphs. In an excellent review, Bunke [Bunke, 2003]

summarizes this problem as follows: '[...] computing the distances of a pair of objects[...] is linear in the number of data items in the case of feature vectors, quadratic in case of strings, and exponential for graphs".

In order to overcome the curse of exponential search space, traditionally, data mining and statistical machine learning have sacrificed the universality of graph models. Instead, research in these areas concentrated on methods for feature vectors, as these can be dealt with much more efficiently. Whenever possible, feature vector models were employed instead of graph models, and even in application domains where graphs are the natural choice of data structure, attempts were made to transform the graphs into feature vectors. As a result, after initial enthusiasm induced by the apparent universality of graphs as data structures, graphs have been practically left unused for a long period of time, due to the expensiveness of their analysis [Conte et al., 2004].

1.1.1 Graph Models in Applications

Given the abundance of methods for feature vectors in data mining and the high computational cost of graph-based techniques, the natural question to ask is: Why is it necessary to employ graph models at all? Are graph models merely of academic interest? In fact, graph models are necessary and of general interest, as efficient feature vector representations cannot preserve the rich topological information represented by a graph.

Despite all computational difficulties, two factors have turned the tide in favor of graph-based data mining over recent years: First, new generations of computers are increasingly able to deal with large graph problems. Second, over the last decade, graph-structured data has increasingly started to emerge in various application areas, ranging from bioinformatics to social network analysis, and fostered by the generation of data in biology, and the enormous growth of the Internet. In these different domains, graphs are the natural data structure to model networks, which represent systems and structures. We will provide a short summary of these fields of application for graphs in the following.

Chemoinformatics Traditionally, graphs have been used to model molecular compounds in chemistry [Gasteiger and Engel, 2003]. Chemoinformatics aims at predicting characteristics of molecules from their graph structures, e.g. toxicity, or effectiveness as a drug. Most traditional benchmark datasets for graph mining algorithms originate from this domain, including MUTAG [Debnath et al., 1991] and PTC [Toivonen et al., 2003]. We will describe these datasets in more detail in Section 2.1.4.

Bioinformatics A major reason for the growing interest in graph-structured data is the advent of large volumes of structured data in molecular biology. This structured data comprises graph models of molecular structures, from RNA to proteins [Berman et al., 2000], and of networks, which include protein-protein interaction networks [Xenarios et al., 2002], metabolic networks [Kanehisa et al., 2004], regulatory networks [Davidson et al., 2002], and phylogenetic networks [Huson and Bryant, 2006]. Bioinformatics seeks to establish the functions of these networks and structures.

Currently, the most successful approach towards function prediction of structures is based on similarity search among structures with known function. For instance, if we want

to predict the function of a new protein structure, we compare its structure to a database of functionally annotated protein structures. The protein is then predicted to exert the function of the (group of) protein(s) which it is most similar to. This concept is supported by models of evolution: Proteins that have similar topological structures are more likely to share a common ancestor, and are hence more likely to carry out the same biochemical function [Whisstock and Lesk, 2003].

Social Network Analysis Another important source of graph structured data is social network analysis [Wasserman and Faust, 1995]. In social networks, nodes represent individuals and edges represent interaction between them. The analysis of these networks is both of scientific and commercial interest. On the one hand, psychologists want to study the complex social dynamics between humans, and biologists want to uncover the social rules in a group of animals. On the other hand, industries want to analyze these networks for marketing purposes. Detecting influential individuals in a group of people, often referred to as 'key-players' or 'trend-setters', is relevant for marketing, as companies could then focus their advertising efforts on persons known to influence the behavior of a larger group of people. In addition, telecommunication and Internet surfing logs provide a vast source of social networks, which can be used for mining tasks ranging from telecommunication network optimization to automated recommender systems.

Internet, HTML, XML A fourth application area for graph models is the Internet which is a network and hence a graph itself. HTML documents are nodes in this network, and hyperlinks connect these nodes. In fact, Google exploits this link structure of the Internet in its famous `PageRank` algorithm [Page et al., 1998] for ranking websites. Furthermore, semi-structured data in form of XML documents is becoming very popular in the database community and in industry. The natural mathematical structure to describe semi-structured data is a graph. As the W3 Consortium puts it: "The main structure of an XML document is tree-like, and most of the lexical structure is devoted to defining that tree, but there is also a way to make connections between arbitrary nodes in a tree" [World Wide Web Consortium (W3C), 2005]. Consequently, XML documents should be regarded as graphs. Various tasks of data manipulation and data analysis can be performed on this graph representation, ranging from basic operations such as querying [Deutsch et al., 1999] to advanced problems such as duplicate detection [Weis and Naumann, 2005].

Benefits of Using Graphs Why is it necessary to represent objects as graphs in these domains? Because all these domains describe systems that consist of interacting substructures. For instance, a social network is a group of interacting individuals. A protein interaction network is a group of interacting molecules. A molecular structure is a group of interacting atoms. The Internet is a network of interlinked websites.

By choosing a graph model, we can store each substructure and its interactions with other substructures. Why is it not possible to represent the same information in a feature vector model? One could think of two ways to do so: First, one could represent each node in a graph as a feature vector that contains a list of its neighbors in the graph. What

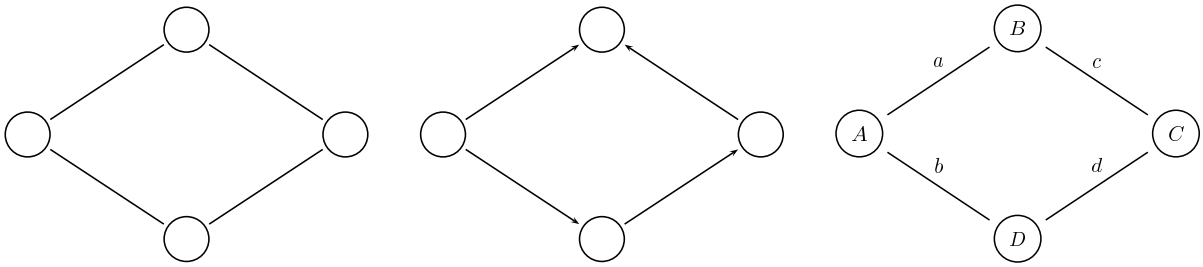

Figure 1.1: Directed, undirected and labeled graphs. **Left**: Undirected graph. **Center**: Directed graph. **Right**: Labeled (undirected) graph.

we would end up with is an adjacency list - which is one way of storing a graph. Second, we could represent each node by a feature vector whose i-th component is 1 if the node is connected to the i-th node in the graph, 0 otherwise. The set of these vectors would be merely the set of columns of the adjacency matrix of a graph - which is another way of storing a graph. As we can see from these two examples, representing a graph by feature vectors that require less memory, but preserve the same topological information seems to be a difficult task. In fact, this has been a central challenge in chemoinformatics over past decades, and a general solution has not been achieved. This is reflected in the fact that the handbook of molecular descriptors [Todeschini and Consonni, 2000] lists several hundreds of feature vector descriptions of graphs.

1.1.2 Bridging Statistical and Structural Pattern Recognition

On a more abstract level, this feature vector representation problem from chemoinformatics is also a major challenge for the field of pattern recognition, data mining and machine learning. The question boils down to: How can we extend the arsenal of efficient mining algorithms on feature vectors to graphs? How can we bridge the gap between statistical pattern recognition on vectors and structural pattern recognition on graphs?

In this thesis, we will elucidate how graph kernels can help to solve this problem.

1.2 Primer on Graph Theory

1.2.1 Directed, Undirected and Labeled Graphs

To understand why graph kernels are important, and in which aspects they can be improved, we will need a primer on graph theory. The purpose of this section is to define terminology and notation for the remainder of this thesis, and to provide the definitions from graph theory that are necessary to follow our line of reasoning [Diestel, 2006].

In its most general form, a graph is a set of nodes connected by edges.

Definition 1 (Graph) *A graph is a pair $G = (V, E)$ of sets of nodes (or vertices) V and edges E, where each edge connects a pair of nodes,* i.e., *$E \subseteq V \times V$. In general, $V(G)$ refers to the set of nodes of graph G, and $E(G)$ refers to the edges of graph G.*

If we assign labels to nodes and edges in a graph, we obtain a labeled graph.

Definition 2 (Labeled Graph) *A labeled graph is a triple $G(V, E, \mathcal{L})$ where (V, E) is a graph, and $\mathcal{L} : V \cup E \to \mathcal{Z}$ is a mapping from the set of nodes V and edges E to the set of node and edge labels $\mathcal{Z}$.*

A graph with labels on its nodes is called *node-labeled*, a graph with labels on edges is called *edge-labeled*. Sometimes *attributes* and *attributed graph* are used as synonyms for *labels* and *labeled graph*, respectively. An example of a labeled graph is depicted in Figure 1.1 (right).

Depending on whether we assign directions to edges, the resulting graph is directed or undirected.

Definition 3 (Directed and Undirected Graph) *Given a graph $G = (V, E)$. If we assign directions to edges such that edge $(v_i, v_j) \neq$ edge (v_j, v_i) for $v_i, v_j \in V$, then G is called a directed graph. G is an undirected graph if*

$$\forall v_i, v_j \in V : (v_i, v_j) \in E \Leftrightarrow (v_j, v_i) \in E \tag{1.1}$$

Figure 1.1 (left) gives an example of an undirected graph, Figure 1.1 (Center) an example of a directed graph. Throughout this thesis, we will assume that we are dealing with undirected graphs. Our results can be directly extended to directed graphs though.

The number of nodes of a graph $G = (V, E)$ is the *graph size*, written $|V|$ or $|V(G)|$. We will denote the graph size as n in this thesis. G is *finite* if its number of nodes is finite; otherwise, it is *infinite*. Graphs considered in this thesis are finite. We call G' *smaller than* G if $|V(G')| < |V(G)|$, and G' *larger than* G if $|V(G')| > |V(G)|$. The number of edges of G is denoted by $|E|$ or $|E(G)|$.

1.2.2 Neighborship in a Graph

Two nodes v_i and v_j in a graph G are adjacent, or neighbors, if (v_i, v_j) is an edge of G. Two edges $e_i \neq e_j$ are adjacent if they have a node in common. If all the nodes of G are pairwise adjacent, then G is complete. This neighborship information on all pairs of nodes in a graph is commonly represented by an adjacency matrix.

Definition 4 (Adjacency Matrix) *The adjacency matrix $A = (A_{ij})_{n \times n}$ of graph $G = (V, E)$ is defined by*

$$A_{ij} := \begin{cases} 1 & \text{if } (v_i, v_j) \in E, \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

where v_i and v_j are nodes from G.

The number of neighbors of a node is closely connected to its *degree*.

Definition 5 (Degree of a Node) *The degree $d_G(v_i)$ of a node v_i in $G = (V, E)$ is the number of edges at v_i:*

$$d_G(v_i) := |\{v_j | (v_i, v_j) \in E\}|$$

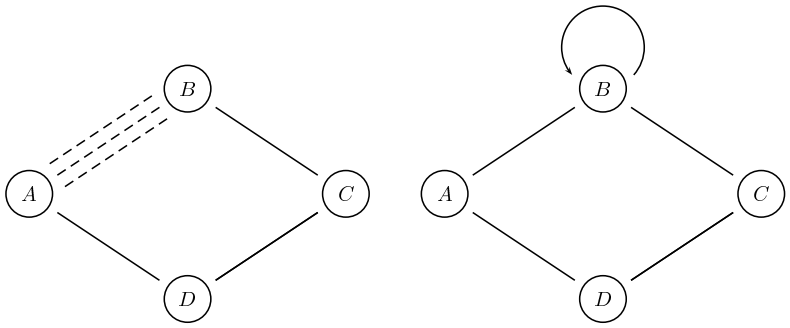

Figure 1.2: Self-loops and multiple edges. **Left**: Graph with multiple edges. **Right**: Graph with self-loop.

In an undirected graph, this is equal to the number of neighbors of v_i, where $\delta(v_i) := \{v_j | (v_i, v_j) \in E\}$ is the set of neighbors of node v_i. A node without neighbors is isolated. The number $\Delta_{min}(G) := \min\{d_G(v) | v \in V\}$ is the minimum degree of G, the number $\Delta_{max}(G) := \max\{d_G(v) | v \in V\}$ its maximum degree. If all the nodes of G have the same degree k, then G is k-regular, or simply regular. The number

$$d_G(G) := \frac{1}{|V|} \sum_{v \in V} d_G(v) \tag{1.3}$$

is the average degree of G.

Pairwise non-adjacent pairs of nodes or edges are called *independent*. More formally, a set of nodes or edges is independent if none of its elements are adjacent. A *self-loop* is an edge (v, v) with two identical ends. A graph contains *multiple edges* if there may be more than one edge between two nodes v_i and v_j. In Figure 1.2 (left), there are multiple edges between nodes "A" and "B". In Figure 1.2 (right), there is a self-loop at "B". In this thesis, we are considering graphs without self-loops and multiple edges.

Definition 6 (Walk, Path, Cycle) *A walk w (of length $\ell - 1$) in a graph G is a non-empty alternating sequence $(v_1, e_1, v_2, e_2, \ldots, e_{\ell-1}, v_\ell)$ of nodes and edges in G such that $e_i = \{v_i, v_{i+1}\}$ for all $1 \leq i \leq \ell - 1$. If $v_1 = v_\ell$, the walk is closed. If the nodes in w are all distinct, it defines a path p in G, denoted $(v_1, v_2, \ldots, v_\ell)$. If $v_1 = v_\ell$, then p is a cycle.*

Note that in the literature, paths are sometimes referred to as *simple* or *unique* paths, and walks are then called *paths*. A *Hamilton path* is a path that visits every node in a graph exactly once. An *Euler path* is a path that visits every edge in a graph exactly once. A graph G is called *connected* if any two of its nodes are linked by a path in G; otherwise G is referred to as 'not connected' or 'disconnected'.

1.2.3 Graph Isomorphism and Subgraph Isomorphism

To check if two graphs are identical, we cannot simply compare their adjacency matrices, as the adjacency matrix changes when we reorder the nodes. Hence a concept of its own, namely *isomorphism*, is required to define identity among graphs.

Definition 7 (Isomorphism) *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We call G and G' isomorphic, and write $G \simeq G'$, if there exists a bijection $f : V \to V'$ with $(v, v') \in E \Leftrightarrow (f(v), f(v')) \in E'$ for all $v, v' \in V$. Such a map f is called an isomorphism.*

The *graph isomorphism problem* is the problem of deciding whether two graphs are isomorphic. An isomorphism of a graph with itself is called an *automorphism*.

In terms of set operations, isomorphism of graphs corresponds to equality of sets. To define a concept analogous to the subset relation, we have to define the concept of a *subgraph* first.

Definition 8 (Subgraph, Induced Subgraph, Clique) *Graph $G' = (V', E')$ is a subgraph of graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq ((V' \times V') \cap E)$, denoted by $G' \sqsubseteq G$. G is then a supergraph of G'. If $|V(G')| < |V(G)|$ or $|E(G')| < |E(G)|$, then G' is a strict subgraph of G, denoted $G' \sqsubset G$. If additionally $E' = ((V' \times V') \cap E)$, then G' is called an induced subgraph of G. A complete subgraph is referred to as a clique.*

Deciding whether a graph is isomorphic to a subgraph of another graph is the *subgraph isomorphism problem*. To tackle such isomorphism problems, graphs are often transfered into vectorial representations, called *graph invariants*.

Definition 9 (Graph Invariant) *Let $\sigma : \mathcal{G} \to \mathbb{R}^d$ with $d \geq 1$ be a mapping from the space of graphs $\mathcal{G}$ to $\mathbb{R}^d$. If $G \simeq G' \Rightarrow \sigma(G) = \sigma(G')$, then σ is called a graph invariant.*

For instance, graph size is a graph invariant. In this context, we are often interested in subgraphs that are *maximal* or *maximum* with respect to such a graph invariant.

Definition 10 (Maximal and Maximum Subgraph) *A subgraph G' of G is maximal with respect to a graph invariant $\xi(G')$ if there is no supergraph G'' of G' in G with $\xi(G'') > \xi(G')$:*

$$\neg\exists\, G'' \sqsubseteq G : (\xi(G') < \xi(G'') \wedge G' \sqsubset G'') \tag{1.4}$$

A subgraph G' of G is maximum with respect to a graph invariant $\xi(G')$ if there is no subgraph G'' of G with $\xi(G'') > \xi(G')$:

$$\neg\exists\, G'' \sqsubseteq G : \xi(G') < \xi(G'') \tag{1.5}$$

We use this notation and terminology from graph theory throughout the remainder of this thesis, unless explicitly stated otherwise.

Besides concepts from graph theory, we will use concepts from linear algebra, functional analysis, probability theory and statistics in this thesis. We assume that the reader is familiar with basic definitions from these domains. For readers who feel not familiar with these domains, we have added primers on functional analysis, and probability theory and statistics in Appendix A.1 and Appendix A.2.

1.3 Review on Alternative Approaches to Graph Comparison

The central problem we tackle in this thesis is to measure similarity between graphs. We will refer to this problem as the graph comparison problem.

Definition 11 (Graph Comparison Problem) *Given two graphs G and G' from the space of graphs $\mathcal{G}$. The graph comparison problem is to find a function*

$$s : \mathcal{G} \times \mathcal{G} \to \mathbb{R} \tag{1.6}$$

such that $s(G, G')$ quantifies the similarity (or dissimilarity) of G and G'.

Note that in the literature, this problem is often referred to as *graph matching*. There is a subtle difference though: While *graph matching* wants to identify corresponding regions in two graphs, *graph comparison* aims at finding a score for the overall similarity of two graphs. Graph matching algorithms often lend themselves easily towards defining an associated similarity score, but graph comparison methods cannot necessarily be employed for graph matching.

The problem of graph comparison has been the topic of numerous studies in computer science [Bunke, 2000]. In this section, we will summarize and review the traditional algorithmic approaches to graph comparison. This field of research can be divided into three categories: similarity measures based on graph isomorphism, inexact matching algorithms, and topological descriptors. We will review these three branches in the following, and focus on their underlying theory. For an in-depth treatment of individual algorithms to graph comparison, we refer the interested reader to [Conte et al., 2004].

1.3.1 Similarity Measures based on Graph Isomorphism

A large family of similarity measures on graphs have been defined based upon the concept of graph isomorphism or variants thereof, which we will describe in the following.

Graph Isomorphism

An intuitive similarity measure on graphs is to check them for topological identity, *i.e.,* for isomorphism. This would give us a basic similarity measure, which is 1 for isomorphic, and 0 for non-isomorphic graphs. Unfortunately, no polynomial runtime algorithm is known for this problem of graph isomorphism [Garey and Johnson, 1979]. Note as a side remark, that graph isomorphism is obviously in NP, but has not yet been proved to either belong to P or to be NP-complete. Intuitively, it is easy to see that when checking two graphs G and G' for isomorphism, one has to consider all permutations of nodes from G' and check if any of the permutations is identical to G.

All graph invariants of two graphs have to be identical in order for the two graphs to be isomorphic. Therefore in practice, simple tests often suffice to establish that two graphs are not isomorphic. For instance, if two graphs have different numbers of nodes or edges, they cannot be isomorphic. But, if two graphs are of identical size, one has to resort to graph invariants that are more expensive to compute, such as shortest path lengths which requires runtime cubic in the number of nodes. In fact, the most efficient way to find

out quickly if two graphs are not isomorphic seems to be to compute a whole series of graph invariants of increasing computational complexity: if the graphs differ in even one invariant, they cannot be isomorphic any more. `nauty` [McKay, 1984], the world's fastest isomorphism testing program, is based on this approach. The problem remains, however, that it is still very hard to decide isomorphism for two graphs that are very similar. On these, the isomorphism problem can only be decided by invariants that are exponentially expensive to compute.

Subgraph Isomorphism

If two graphs are of different sizes, they are obviously not isomorphic. But the smaller graph G' might still be similar to G if G' is a subgraph of G. To uncover this relationship, we have to solve the subgraph isomorphism problem. Unfortunately, this problem is known to be NP-complete [Garey and Johnson, 1979], and is not practically feasible on large graphs.

Why is this problem harder than graph isomorphism? Because we not only have to check which permutation of G' is identical to G as before, but we have to find out if any permutation of G' is identical to any of the subgraphs of G. In short, for isomorphism checking, we have to consider all permutations of G', while for subgraph isomorphism checking, we have to check all permutations of G' and all subsets of G (of the size of G'). Note that the isomorphism problem is one instance of the subgraph isomorphism problem, where $|V(G)| = |V(G')|$ and $|E(G)| = |E(G')|$.

A setback of both graph and subgraph isomorphism is that they do not care about partial similarities of two graphs. Graphs must be topologically equivalent, or contained in each other, to be deemed similar. This is a serious limitation of isomorphism-based similarity measures of graphs.

Maximum Common Subgraph

A related measure of similarity deems two graphs similar if they share a large common subgraph. This leads to the concept of a *maximum common subgraph* [Neuhaus, 2006]:

Definition 12 (Maximum Common Subgraph, mcs) *Let G and G' be graphs. A graph G_{sub} is called a common subgraph of G and G' if G_{sub} is a subgraph of G and of G'. G_{sub} is a maximum common subgraph (mcs) if there exists no other common subgraph of G and G' with more nodes.*

In general, the maximum common subgraph needs not be unique, *i.e.,* there may be more than one maximum common subgraphs of identical size.

Turning the idea of using the maximum common subgraph upside-down, one might also think of the following measure of graph similarity: G and G' are similar if they are both subgraphs of a "small" supergraph G_{super}. The smaller the size of G_{super}, the more similar G and G' are. This leads to the concept of a minimum common supergraph.

Definition 13 (Minimum Common Supergraph, MCS) *Let G and G' be graphs. A graph G_{super} is called common supergraph of G and G' if there exist subgraph isomorphisms from G to G_{super} and from G' to G_{super}. A common supergraph of G and G' is called*

minimum common supergraph (MCS) if there exists no other common supergraph of G and G' with fewer nodes than G_{super}.

The computation of the minimum common supergraph can be reduced to computing a maximum common subgraph [Bunke et al., 2000]. While the size of the maximum common subgraph and the minimum common supergraph represent a measure of similarity, they can also be applied to define distances on graphs. For instance, Bunke and Shearer [Bunke and Shearer, 1998] define a distance that is proportional to the size of the maximum common subgraph compared to that of the larger of the two graphs:

$$d_1(G, G') = 1 - \frac{|mcs(G, G')|}{max(|G|, |G'|)} \tag{1.7}$$

In another approach, the difference of the sizes of the minimum common supergraph and the maximum common subgraph is evaluated, resulting in a distance metric defined as [Fernández and Valiente, 2001]:

$$d_2(G, G') = |MCS(G, G')| - |mcs(G, G')| \tag{1.8}$$

Maximal Common Subgraphs in Two Graphs

Even the maximum common subgraph is not necessarily a good measure of similarity. There may be graphs that share many subgraphs that are rather small, but which do not include even one large common subgraph. Such graphs would be deemed dissimilar by a similarity measure based on the maximum common subgraph.

An approach that would account for such frequent local similarities is counting maximal common subgraphs. Obviously, this procedure is NP-hard, as it requires repeated subgraph isomorphism checking. But, rather efficient algorithms have been proposed for this task, which transform the problem of finding maximum common subgraphs into finding all cliques in a product graph [Koch, 2001]. The classic branch-and-bound algorithm by Bron and Kerbosch [Bron and Kerbosch, 1973] is then applied to enumerate all cliques in this product graph.

While this is a widely used technique for graph comparison in bioinformatics [Liang et al., 2006], it faces enormous runtime problems when the size of the product graph exceeds more than several hundreds of nodes. For instance, suppose we want to compare two graphs of size 24. This results in a product graph of roughly 600 nodes. Ina Koch [Koch, 2001] reports that Bron-Kerbosch on a product graph of this size requires more than 3 hours.

Discussion

Graph isomorphism is rarely used in practice, because few graphs completely match in real-world applications [Conte et al., 2004]. A major reason for this is experimental noise, which in the case of graphs, may lead to extra or missing edges and nodes. In contrast, subgraph isomorphism methods have been applied successfully in many contexts, despite the fact that they are computationally more expensive than graph isomorphism. Maximum common subgraph methods seem intuitively attractive and have received attention recently, but are so far only applicable on graphs with very few nodes.

To summarize, the class of similarity measures based on graph isomorphism, subgraph isomorphism and common subgraphs are the methods of choice when dealing with small graphs with few nodes. As network size increases, the underlying exponential size of the subgraph isomorphism problem renders the computation impractical.

1.3.2 Inexact Matching Algorithms

The second major family of graph similarity measures does not enforce strict matching of graphs and their subgraphs. These inexact matching algorithms measure the discrepancy of two graphs in terms of a cost function or edit distance to transform one graph into the other.

From an application point of view, these error-tolerant matching algorithms seem attractive, because real-world objects are often corrupted by noise. Therefore it is necessary to integrate some degree of error tolerance into the graph matching process.

The most powerful concept within the category of error-tolerant graph matching is graph edit distance [Bunke and Allermann, 1983, Bunke, 2003]. In its most general form, a graph edit operation is either a deletion, insertion, or substitution (*i.e.,* label change). Edit operations can be applied to nodes as well as to edges. By means of edit operations differences between two graphs are modeled. In order to enhance the modeling capabilities, often a cost is assigned to each edit operation. The costs are real nonnegative numbers. They have to be chosen based on domain knowledge. Typically, the more likely a certain distortion is to occur the lower is its cost. The edit distance, $d(G, G')$, of two graphs is defined to be the minimum cost c incurred over all sequences S of edit operations that transform graph G into G'. Formally,

$$d(G, G') = min_S\{c(S)|S \text{ is a sequence of edit operations that transform } G \text{ into } G'\} \tag{1.9}$$

Obviously, if $G = G'$, then $d(G, G') = 0$, and the more G and G' differ, the larger is $d(G, G')$.

Discussion Inexact matching algorithms in general, and edit distances in particular, are very expressive measures of graph similarity. Differences between graphs can be penalized on different levels (nodes, edges, labels) and with different weights. This leads to a powerful measure of similarity that can be tailored to the needs of a specific application domain.

However, graph edit distances are plagued by a few problems. It is often difficult to find the appropriate penalty costs for individual edit operations. In other words, graph edit distances are hard to parameterize. Furthermore, finding the minimal edit distance is NP-hard, as subgraph isomorphism and maximum common subgraph can be shown to be instances of the edit distance problem [Bunke, 1999]. In short, while a powerful measure of similarity, edit distances pose a major computational challenge. Ongoing research is exploring various ways of making both parameterization and computation of edit distances more efficient [Neuhaus, 2006, Riesen et al., 2006, Justice and Hero, 2006].

1.3.3 Similarity Measures based on Topological Descriptors

A major reason why graph comparison, learning on graphs, and graph mining are so difficult and expensive is the complex structure of graphs which does not lend itself to a simple feature vector representation. The third family of similarity measures for graph comparison aims at finding feature vector representations of graphs that summarize graph topology efficiently. These feature vector descriptions of graph topology are often referred to as *topological descriptors*. The goal is to find vector-representations of graphs such that comparing these vectors gives a good indication of graph similarity. One popular category of these vector representations is based on spectral graph theory [Chung-Graham, 1997].

The roots of encoding graphs as scalars lie in the field of chemoinformatics. A long-standing challenge in this area is to answer queries on large databases of molecular graphs. For this purpose, hundreds and thousands of different molecular (topological) descriptors were invented, as reflected by extensive handbooks on this topic [Todeschini and Consonni, 2000]. A prominent example is the Wiener Index [Wiener, 1947], defined as the sum over all shortest paths in a graph.

Definition 14 (Wiener Index) *Let $G = (V, E)$ be a graph. Then the Wiener Index $W(G)$ of G is defined as*

$$W(G) = \sum_{v_i \in G} \sum_{v_j \in G} d(v_i, v_j), \tag{1.10}$$

where $d(v_i, v_j)$ is defined as the length of the shortest path between nodes v_i and v_j from G.

Clearly, this index is identical for isomorphic graphs. Hence the Wiener Index, and all topological descriptors (that do not include node labels) represent graph invariants (see Definition 9). The problem is that the reverse, identical topological descriptors imply isomorphism, does not hold in general. If this is the case, then we call this topological descriptor a complete graph invariant [Koebler and Verbitsky, 2006]. All known complete graph invariants require exponential runtime though, as their computation is equivalent to solving the graph isomorphism problem.

Discussion Topological descriptors do not remove the burden of runtime complexity from graph comparison. While it seems easy and attractive to compare scalars to get a measure of graph similarity, one should not forget that the computation of many of these topological indices may require exponential runtime. Furthermore, the vast number of topological descriptors that have been defined reflect both an advantage and a disadvantage of this concept: On the one hand, this huge number of variants clearly indicates that topological descriptors provide a good approximate measure of graph similarity. On the other hand, this multitude of variations on the same topic also points at a major weakness of topological descriptors. None of them is general enough to work well across all different application tasks. It seems that every application requires its own topological descriptor to achieve good results. Choosing the right one for the particular application at hand is the major challenge in practice, and is similar to the problem of picking the right cost function for edit distances, as outlined in Section 1.3.2.

1.3.4 Recent Trends in Graph Comparison

Due to the inherent problems in traditional approaches to graph comparison, machine learning, pattern recognition and data mining have started to take new roads towards this problem in the recent past. As we have mentioned in Section 1.3.2, one current focus in pattern recognition is the automatic learning of edit distance parameters [Neuhaus and Bunke, 2005, Neuhaus and Bunke, 2007]. Machine learning has begun to explore the usage of graphical models for graph matching [Caelli and Caetano, 2005]. An alternative strategy has been adopted in data mining: Efficient branch and bound algorithms have been developed to enumerate frequent subgraphs in a set of graphs, and two graphs are then deemed the more similar, the more of these frequent subgraphs they share [Kramer et al., 2001, Deshpande et al., 2005, Cheng et al., 2007] (see Section 4.2).

While these new approaches show promising results in applications, none of these methods can avoid the same problems encountered in the classic approaches: either the runtime degenerates for large graphs, or one has to resort to simplified representations of graphs that ignore part of their topological information.

Graph kernels are one of the most recent approaches to graph comparison. Interestingly, graph kernels employ concepts from all three traditional branches of graph comparison: they measure similarity in terms of isomorphic substructures of graphs, they allow for inexact matching of nodes, edges, and labels, and they treat graphs as vectors in a Hilbert space of graph features. Graph kernels are the topic of this thesis, and we will review them in detail in the following section.

1.4 Review on Graph Kernels

All major techniques for comparing graphs described in Section 1.3 suffer from exponential runtime in the worst case. The open question is whether there are fast polynomial alternatives that still provide an expressive measure of similarity on graphs: We will show next that graph kernels are an answer to this problem.

To understand the contribution of graph kernels to the field of graph comparison, we first have to define what a kernel is. Afterwards, we will show how kernels can be defined on structured data in general, and on graphs in particular.

1.4.1 Primer on Kernels

As a start, we will describe the historical development of kernels from ingredients of the Support Vector Machine to the underlying principle of a large family of learning algorithms. For a more extensive treatment we refer the reader to [Schölkopf and Smola, 2002], and the references therein.

Kernels in Support Vector Machines

Traditionally, Support Vector Machines (SVMs) deal with the following binary classification problem (although Multiclass-SVMs have been developed over recent years [Tsochantaridis et al., 2005]): Given a set of training objects associated with class labels $\{\mathbf{x}_i, y_i\}_{i=1}^m$, $\mathbf{x}_i \in \mathcal{X} = \mathbb{R}^d$ with $d \in \mathbb{N}$, $y_i \in \mathcal{Y} = \{\pm 1\}$, the task is to learn a classifier $f : \mathcal{X} \to \mathcal{Y}$ that predicts the labels of unclassified data objects.

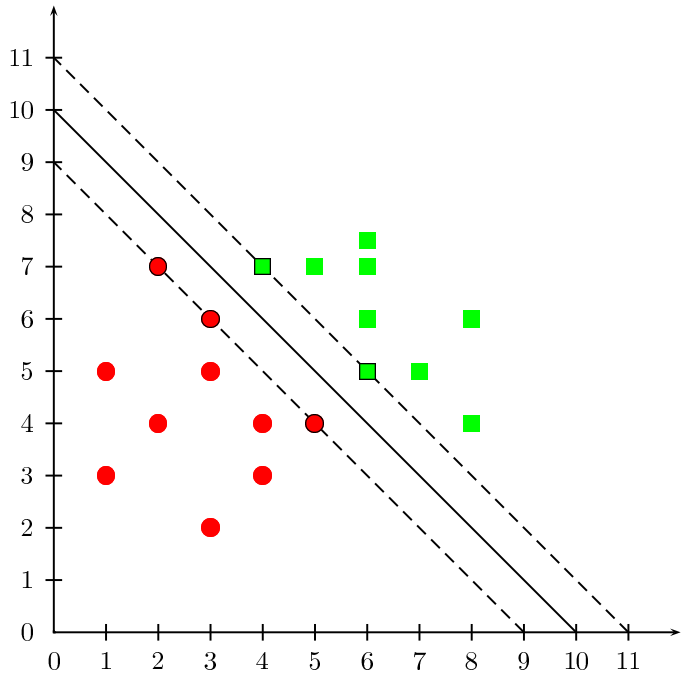

Figure 1.3: Toy example: Binary classification problem with maximum margin hyperplane. Hyperplane (straight line) separating two classes of input data (dots and squares). Data points located on the margin (dashed line) are support vectors.

Step 1: Maximizing the Margin

Large margin methods try to solve this question by introducing a hyperplane between class $y = 1$ and class $y = -1$. Depending on the location of $\mathbf{x}_i$ with respect to the hyperplane, y_i is predicted to be 1 or -1, respectively.

Let us first assume that such a hyperplane exists that correctly separates both classes. Then infinitely many of these hyperplanes exist, parameterized by $(\mathbf{w}, b)$ with $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ which can be written as $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$, where $\langle \mathbf{w}, \mathbf{x} \rangle$ denotes the dot product between vectors $\mathbf{w}$ and $\mathbf{x}$. These hyperplanes satisfy

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0, \ \forall i \in \{1, 2, \ldots, m\}, \tag{1.11}$$

and these hyperplanes correspond to decision functions

$$f(\mathbf{x}) = sgn(\langle \mathbf{w}, \mathbf{x} \rangle + b), \tag{1.12}$$

where $f(\mathbf{x})$ is the (predicted) class label of data point $\mathbf{x}$. Among these hyperplanes a unique optimal hyperplane can be chosen which maximizes the *margin* (see Figure 1.3), *i.e.,* the minimum distance between the hyperplane and the nearest data points from both classes [Vapnik and Lerner, 1963].

Linear Hard-Margin Formulation An equivalent formulation of this optimization problem is

$$\begin{array}{ll} \underset{\mathbf{w},b}{\text{minimize}} & \frac{1}{2}\| \mathbf{w} \|^2 \\ \text{subject to} & y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1 \text{ for all } i \in \{1, 2, \ldots, m\}. \end{array} \tag{1.13}$$

where $\frac{1}{2}\| \mathbf{w} \|^2$ is referred to as the *objective function*.

The standard optimization technique for such problems is to formulate the Lagrangian and to solve the resulting dual problem:

$$\begin{array}{ll} \underset{\alpha}{\text{maximize}} & -\frac{1}{2}\alpha^\top H \alpha + \sum_{i=1}^{m} \alpha_i \\ \text{subject to} & \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } \alpha_i \geq 0 \text{ for all } i \in \{1, 2, \ldots, m\}, \end{array} \tag{1.14}$$

where $H \in \mathbb{R}^{m \times m}$ with $H_{ij} := y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \, \mathbf{x}_i \tag{1.15}$$

Interestingly, the solution vector has an expansion in terms of the training examples.

Two observations in these equations are fundamental for Support Vector Machine classification. First, the dual problem involves the input data points purely in form of dot products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Second, α_i's are non-zero exclusively for those data points $\mathbf{x}_i$ that satisfy the primal constraints $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ with equality. These $\mathbf{x}_i$ are the points on the margin. The hyperplane is defined by these points, as their corresponding α_i's are non-zero, *i.e.,* only these $\mathbf{x}_i$'s are *supporting* the hyperplane; they are the *support vectors*, from which the algorithm inherits its name.

Step 2: Allowing for Margin Errors

Soft Hard-Margin Formulation In most cases it is illusory to assume that there exists a hyperplane in input space that correctly separates two classes. In fact, usually it is impossible to find such a hyperplane because of noise that tends to occur close to the boundary [Duda et al., 2001]. For this reason, soft-margin SVMs have been developed as an alternative to hard-margin SVMs. While hard-margin SVMs force the condition $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ to hold, the soft-margin SVMs allow for some misclassified training points. The goal is to improve the generalization performance of the SVM, *i.e.,* its performance on test samples different from the training set.

C-Support Vector Machines The earliest version of soft-margin SVMs that allow for some training errors are C-Support Vector Machines (C-SVM). They introduce non-negative slack variables ξ [Bennett and Mangasarian, 1993, Cortes and Vapnik, 1995] and a penalty factor C into the primal optimization problem (1.13).

The primal problem changes into

$$\begin{array}{ll} \underset{\mathbf{w},b,\xi}{\text{minimize}} & \frac{1}{2}\| \mathbf{w} \|^2 + C \sum_{i=1}^{m} \xi_i \\ \text{subject to} & y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i \text{ for all } i \in \{1, 2, \ldots, m\}, \end{array} \tag{1.16}$$

The slack variable ξ_i relaxes the condition $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ at penalty $C * \xi_i$. The C-SVM hence allows for *margin errors*, penalizing them proportional to their violation of the condition $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$. Margin errors are those training data points $\mathbf{x}_i$ for which $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) < 1$, *i.e.,* they are lying within the margin or are misclassified.

The dual to (1.16) is

$$\begin{array}{ll} \underset{\alpha}{\text{maximize}} & -\frac{1}{2}\alpha^\top H \alpha + \sum_{i=1}^{m} \alpha_i \\ \text{subject to} & \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } C \geq \alpha_i \geq 0 \text{ for all } i \in \{1, 2, \ldots, m\}. \end{array} \tag{1.17}$$

Thus C determines the tradeoff between two competing goals: maximizing the margin and minimizing the training error. While contributing to a better generalization performance, the C-SVM have one practical disadvantage: C is a rather unintuitive parameter and there is no *a priori* way to select it. For this reason, an alternative soft-margin SVM, the so-called ν-SVM was proposed to overcome this problem [Schölkopf et al., 2000].

ν-Support Vector Machine Introducing the parameter ν, the soft margin optimization problem is rewritten as:

$$\begin{array}{ll} \underset{\mathbf{w},b,\xi,\rho}{\text{minimize}} & \frac{1}{2}\| \mathbf{w} \|^2 - \nu\rho + \frac{1}{m} \sum_{i=1}^{m} \xi_i \\ \text{subject to} & y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq \rho - \xi_i \text{ for all } i \in \{1, 2, \ldots, m\} \\ \text{and} & \xi_i \geq 0, \quad \rho \geq 0. \end{array} \tag{1.18}$$

This can be transfered into the corresponding dual:

$$\begin{array}{ll} \underset{\alpha}{\text{maximize}} & -\frac{1}{2}\alpha^\top H \alpha \\ \text{subject to} & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \frac{1}{m} \\ & \sum_{i=1}^{m} \alpha_i \geq \nu. \end{array}$$

ν has a much more concrete interpretation than C, as can be seen from the following lemma [Schölkopf et al., 2000].

Theorem 15 *Suppose we run ν-SVM on some data with the result that $\rho > 0$, then*

- *ν is an upper bound on the fraction of margin errors,*
- *ν is a lower bound on the fraction of support vectors.*

Step 3: Moving the Problem to Feature Space

Kernel Trick Still, even soft-margin classifiers cannot solve every classification problem. Just imagine the following 2-d example: All positive data points lie within a circle, all negative data points outside (see Figure 1.4). How to introduce a hyperplane that shows good generalization performance in this case?

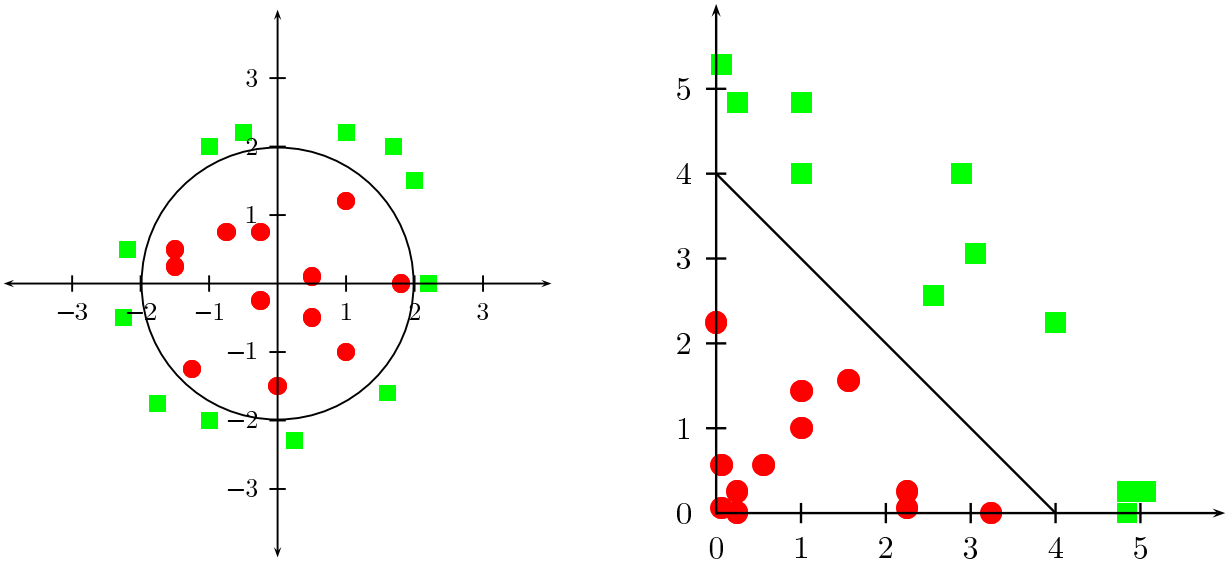

Figure 1.4: Toy example illustrating kernel trick: Mapping a circle into feature space: data point distribution in input space (**Left**) and feature space (**Right**). By transformation from input space to feature space, dots and squares become linearly separable. In addition, all operations in feature space can be performed by evaluating a kernel function on the data objects in input space.

The trick to overcome these sorts of problems is to map the input points into a (usually higher-dimensional) feature space $\mathcal{H}$. The idea is to find a non-linear mapping $\phi : \mathbb{R}^d \to \mathcal{H}$, such that in $\mathcal{H}$, we can still use our previous SVM formulation, simply by replacing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. Recall what we said earlier: Data points in the dual hyperplane optimization problems occur only within dot products; if we map $\mathbf{x}_i$ and $\mathbf{x}_j$ to $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$, respectively, then we just have to deal with $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ instead. If we define a *kernel function* k with the following property

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle, \tag{1.19}$$

we obtain decision functions of the form

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle + b \right) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \right), \tag{1.20}$$

and the following quadratic problem (for the hard-margin case):

$$\begin{aligned} \underset{\alpha \in \mathbb{R}^m}{\text{maximize}} \qquad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} \qquad & \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \end{aligned}$$

This means nothing less than that we move our classification problem into a higher-dimensional space $\mathcal{H}$ and solve it even without explicitly computing the mapping ϕ to $\mathcal{H}$. This is commonly known as the famous *kernel trick*.

Kernel Functions

Positive Definiteness Which class of functions are eligible as kernel functions? To answer this question in short, we have to clarify three definitions first [Schölkopf and Smola, 2002]:

Definition 16 (Gram Matrix) *Given a function $k : \mathcal{X}^2 \to \mathbb{K}$ (where $\mathbb{K} = \mathbb{C}$ or $\mathbb{K} = \mathbb{R}$) and patterns $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$, the $m \times m$ matrix K with elements*

$$K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.21}$$

is called the Gram matrix *(or* kernel matrix*) of k with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_m$.*

Later on, we will refer to Gram matrices as kernel matrices.

Definition 17 (Positive Definite Matrix) *A complex $m \times m$ matrix K satisfying*

$$\sum_{i,j=1}^{m} c_i \bar{c}_j K_{ij} \geq 0 \tag{1.22}$$

for all $c_i \in \mathbb{C}$ is called positive definite[1].

Similarly, a real symmetric $m \times m$ matrix K satisfying condition 1.22 for all $c_i \in \mathbb{R}$ is called positive definite.

Note that a symmetric matrix is positive definite if and only if all its eigenvalues are nonnegative.

Definition 18 (Positive Definite Kernel) *Let $\mathcal{X}$ be a nonempty set. A function k on $\mathcal{X} \times \mathcal{X}$ which for all $m \in \mathbb{N}$ and all $\mathbf{x}_1 \ldots, \mathbf{x}_m \in \mathcal{X}$ gives rise to a positive definite Gram matrix is called a* positive definite kernel*, or short* kernel*.*

[1] In mathematics, this matrix is called a positive *semi*definite matrix. In machine learning, the "semi" is usually omitted for brevity. In this thesis, we kept to this machine learning notation.

Given these definitions, we can state the following about the choice of k: if k is a positive definite kernel function, then we can construct a feature space in which k is the dot product. More precisely, we can construct a Hilbert space $\mathcal{H}$ with

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle. \tag{1.23}$$

A Hilbert space is a dot product space, which is also complete with respect to the corresponding norm; that is, any Cauchy sequence of points converges to a point in the space [Burges, 1998]. The Hilbert space associated with a kernel is referred to as a *Reproducing Kernel Hilbert Space* (RKHS). It can be shown by means of functional analysis that every kernel function is associated with a RKHS and that every RKHS is associated with a kernel function.

Kernel Design The class of positive definite kernel functions has attractive closure properties that ease the design of new kernel functions by combining known ones. Two of the most prominent of these properties are that linear combinations and point-wise products of kernels are themselves positive definite kernels:

- 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 and k_2 are kernels, then $k_1 k_2$, defined by $(k_1 k_2)(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$, is a kernel.

These rules can be used to combine known kernels in order to create new kernel functions. Among the most famous kernel functions are the
delta kernel

$$k(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{x}', \\ 0 & \text{otherwise} \end{cases}$$

the *polynomial* kernel

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

the *Gaussian radial basis function (RBF)* kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\, \sigma^2} \right),$$

and the *Brownian bridge* kernel

$$k(\mathbf{x}, \mathbf{x}') = \max(0, c - k| \mathbf{x} - \mathbf{x}' |).$$

with $d \in \mathbb{N}$ and $c, k, \sigma \in \mathbb{R}$ and $\mathbf{x}, \mathbf{x}' \in \mathcal{X} \subset \mathbb{R}^N$. For $d = 1$ and $c = 0$, the polynomial kernel is also referred to as the *linear* kernel. Starting from this set and exploiting the characteristics of positive definite kernels, a whole battery of kernel functions can be developed.

Kernels Methods

A further key advantage of kernel methods is that they can be applied to non-vectorial data, as first realized by [Schölkopf, 1997]. In contrast to our initial assumption that $\mathcal{X} = \mathbb{R}^d$, $\mathcal{X}$ can also represent any structured domain, such as the space of strings or graphs. In this case, all kernel methods remain applicable, as long as we can find a mapping $\phi : \mathcal{X} \to \mathcal{H}$, where $\mathcal{H}$ is a RKHS. A thrilling consequence of the *kernel trick* is that we do not even have to determine this mapping ϕ explicitly. Finding a kernel function $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ on pairs of objects from $\mathcal{X}$ is completely sufficient. As a result, we can compare structured data via kernels without even explicitly constructing the feature space $\mathcal{H}$. This finding has had a huge scientific impact over recent years, and defining kernel functions for structured data has become a hot topic in machine learning [Gärtner, 2003], and in bioinformatics [Schölkopf et al., 2004].

1.4.2 Primer on Graph Kernels

Kernels on structured data almost exclusively belong to one single class of kernels: R-convolution kernels as defined in a seminal paper by Haussler [Haussler, 1999].

R-Convolution Kernels R-convolution kernels provide a generic way to construct kernels for discrete compound objects. Let $x \in \mathcal{X}$ be such an object, and $\boldsymbol{x} := (x_1, x_2, \ldots, x_D)$ denote a decomposition of x, with each $x_i \in \mathcal{X}_i$. We can define a boolean predicate

$$R : \boldsymbol{\mathcal{X}} \times \mathcal{X} \to \{\textsc{True}, \textsc{False}\}, \tag{1.24}$$

where $\boldsymbol{\mathcal{X}} := \mathcal{X}_1 \times \ldots \times \mathcal{X}_D$ and $R(\boldsymbol{x}, x)$ is True whenever $\boldsymbol{x}$ is a valid decomposition of x. This allows us to consider the set of all valid decompositions of an object:

$$R^{-1}(x) := \{\boldsymbol{x} | R(\boldsymbol{x}, x) = \textsc{True}\}. \tag{1.25}$$

Like [Haussler, 1999] we assume that $R^{-1}(x)$ is countable. We define the R-convolution $\star$ of the kernels $\kappa_1, \kappa_2, \ldots, \kappa_D$ with $\kappa_i : \mathcal{X}_i \times \mathcal{X}_i \to \mathbb{R}$ to be

$$\begin{aligned} k(x, x') = \kappa_1 \star \kappa_2 \star \ldots \star \kappa_D(x, x') \;&:= \\ &:= \sum_{\substack{\boldsymbol{x} \in R^{-1}(x) \\ \boldsymbol{x}' \in R^{-1}(x')}} \mu(\boldsymbol{x}, \boldsymbol{x}') \prod_{i=1}^{D} \kappa_i(x_i, x_i'), \end{aligned} \tag{1.26}$$

where μ is a finite measure on $\boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{X}}$ which ensures that the above sum converges.[2] [Haussler, 1999] showed that $k(x, x')$ is positive semi-definite and hence admissible as a kernel [Schölkopf and Smola, 2002], provided that all the individual κ_i are. The deliberate vagueness of this setup regard to the nature of the underlying decomposition leads to a rich framework: Many different kernels can be obtained by simply changing the decomposition.

[2] [Haussler, 1999] implicitly assumed this sum to be well-defined, and hence did not use a measure μ in his definition.

In this thesis, we are interested in kernels between two graphs. We will refer to those as *graph kernels*. Note that in the literature, the term *graph kernel* is sometimes used to describe kernels between two nodes in one single graph. Although we are exploring the connection between these two concepts in ongoing research [Vishwanathan et al., 2007b], in this thesis, we exclusively use the term *graph kernel* for kernel functions comparing two graphs to each other.

The natural and most general R-convolution on graphs would decompose two each graphs G and G' into all of their subgraphs and compare them pairwise. This all-subgraphs kernel is defined as

Definition 19 (All-Subgraphs Kernel) *Let G and G' be two graphs. Then the all-subgraphs kernel on G and G' is defined as*

$$k_{subgraph}(G, G') = \sum_{S \sqsubseteq G} \sum_{S' \sqsubseteq G'} k_{isomorphism}(S, S'), \tag{1.27}$$

where

$$k_{isomorphism}(S, S') = \begin{cases} 1 & \text{if } S \simeq S', \\ 0 & \text{otherwise.} \end{cases} \tag{1.28}$$

In an early paper on graph kernels, [Gärtner et al., 2003] show that the problem of computing this all-subgraphs kernel based on all subgraphs is NP-hard. Their proof is founded in the fact that computing the all-subgraphs kernel is as hard as deciding subgraph isomorphism. This can be easily seen as follows. Given a subgraph S from G. If there is a subgraph S' from G' such that $k_{isomorphism}(S, S') = 1$, then S is a subgraph of G'. Hence we have to solve subgraph isomorphism problems when computing $k_{isomorphism}$, which are known to be NP-hard.

Random Walk Kernels As an alternative to the all-subgraphs kernel, two types of graph kernels based on walks have been defined in the literature: the product graph kernels of [Gärtner et al., 2003], and the marginalized kernels on graphs of [Kashima et al., 2003]. We will review the definitions of these random walk kernels in the following. For the sake of clearer presentation, we assume without loss of generality that all graphs have identical size n in the following. The results clearly hold even when this condition is not met.

Product Graph Kernel [Gärtner et al., 2003] propose the a random walk kernel counting common walks in two graphs. For this purpose, they employ a type of graph product, the direct product graph, also referred to as tensor or categorical product [Imrich and Klavzar, 2000].

Definition 20 *The direct product of two graphs $G = (V, E, \mathcal{L})$ and $G' = (V', E', \mathcal{L}')$ shall be denoted as $G_\times = G \times G'$. The node and edge set of the direct product graph are*

respectively defined as:

$$\begin{aligned} V_\times =& \{(v_i, v'_{i'}) : v_i \in V \wedge v'_{i'} \in V' \wedge \mathcal{L}(v_i) = \mathcal{L}'(v_{i'})\} \\ E_\times =& \{((v_i, v'_{i'}), (v_j, v'_{j'})) \in V_\times \times V_\times : \\ & (v_i, v_j) \in E \wedge (v'_{i'}, v'_{j'}) \in E' \wedge (\mathcal{L}(v_i, v_j) = \mathcal{L}'(v'_{i'}, v'_{j'}))\} \end{aligned} \tag{1.29}$$

Using this product graph, they define the random walk kernel as follows.

Definition 21 *Let G and G' be two graphs, let $A_\times$ denote the adjacency matrix of their product graph $G_\times$, and let $V_\times$ denote the node set of the product graph $G_\times$. With a sequence of weights $\lambda = \lambda_0, \lambda_1, \ldots (\lambda_i \in \mathbb{R}; \lambda_i \geq 0$ for all $i \in \mathbb{N})$ the product graph kernel is defined as*

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} [\sum_{k=0}^{\infty} \lambda_k A_\times^k]_{ij} \tag{1.30}$$

if the limit exists.

The limit of $k(G, G')$ can be computed rather efficiently for two particular choices of λ: the geometric series and the exponential series.

Setting $\lambda_k = \lambda^k$, *i.e.,* to a geometric series, we obtain the *geometric random walk kernel*

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} [\sum_{k=0}^{\infty} \lambda^k A_\times^k]_{ij} = \sum_{i,j=1}^{|V_\times|} [(I - \lambda A_\times)^{-1}]_{ij} \tag{1.31}$$

if $\lambda < \frac{1}{a}$, where $a \geq \Delta_{max}(G_\times)$, the maximum degree of a node in the product graph.

Similarly, setting $\lambda_k = \frac{\beta^k}{k!}$, *i.e.,* to an exponential series, we obtain the *exponential random walk kernel*

$$k_\times(G, G') = \sum_{i,j=1}^{|V_\times|} [\sum_{k=0}^{\infty} \frac{(\beta A_\times)^k}{k!}]_{ij} = \sum_{i,j=1}^{|V_\times|} [e^{\beta A_\times}]_{ij} \tag{1.32}$$

Both these kernel require $O(n^6)$ runtime, which can be seen as follows: The geometric random walk requires inversion of an $n^2 \times n^2$ matrix $(I - \lambda A_\times)$. This is an effort cubic in the size of the matrix, hence $O(n^6)$. For the exponential random walk kernel, matrix diagonalization of the $n^2 \times n^2$ matrix $A_\times$ is necessary to compute $e^{\beta A_\times}$, which is again an operation with runtime cubic in the size of the matrix.

Marginalized Graph Kernels Though motivated differently, the marginalized graph kernels of [Kashima et al., 2003] are closely related. Their kernel is defined as the expectation of a kernel over all pairs of label sequences from two graphs

For extracting features from graph $G = (V, E, \mathcal{L})$, a set of label sequences is produced by performing a random walk. At the first step, $v_1 \in V$ is sampled from an initial probability

distribution $p_s(v_1)$ over all nodes in V. Subsequently, at the i-th step, the next node $v_i \in V$ is sampled subject to a transition probability $p_t(v_i|v_{i-1})$, or the random walk ends with probability $p_q(v_{i-1})$:

$$\sum_{v_i=1}^{|V|} p_t(v_i|v_{i-1}) + p_q(v_{i-1}) = 1 \tag{1.33}$$

Each random walk generates a sequence of nodes $w = (v_1, v_2, ..., v_\ell)$, where ℓ is the length of w (possibly infinite).

The probability for the walk w is described as

$$p(w|G) = p_s(v_1) \prod_{i=2}^{\ell} p_t(v_i|v_{i-1}) p_q(v_\ell). \tag{1.34}$$

Associated with a walk w, we obtain a sequence of labels

$$h_w = (\mathcal{L}(v_1), \mathcal{L}(v_1, v_2), \mathcal{L}(v_2), \ldots, \mathcal{L}(v_\ell)) = (h_1, h_2, \ldots, h_{2\ell-1}), \tag{1.35}$$

which is an alternating label sequence of node labels and edge labels from the space of labels $\mathcal{Z}$:

$$h_w = (h_1, h_2, \ldots, h_{2\ell-1}) \in \mathcal{Z}^{2\ell-1}. \tag{1.36}$$

The probability for the label sequence h is equal to the sum of the probabilities of all walks w emitting a label sequence h_w identical to h,

$$p(h|G) = \sum_w \delta(h = h_w) \left\{ p_s(v_1) \prod_{i=2}^{\ell} (p_t(v_i|v_{i-1}) p_q(v_l)) \right\} \tag{1.37}$$

where δ is a function that returns 1 if its argument holds, 0 otherwise.

[Kashima et al., 2003] then define a kernel k_z between two label sequences h and h'. Assuming that k_v is a nonnegative kernel on nodes, and k_e is a nonnegative kernel on edges, then the kernel for label sequences is defined as the product of label kernels when the lengths of two sequences are identical ($\ell = \ell'$):

$$k_z(h, h') = k_v(h_1, h'_1) \prod_{i=2}^{\ell} k_e(h_{2i-2}, h'_{2i-2}) k_v(h_{2i-1}, h'_{2i-1}) \tag{1.38}$$

The label sequence graph kernel is then defined as the expectation of k_z over all possible h and h'

$$k(G, G') = \sum_h \sum_{h'} k_z(h, h') p(h|G) p(h'|G'). \tag{1.39}$$

In terms of R-convolution, the decomposition corresponding to this graph kernel is the set of all possible label sequences generated by a random walk.

The runtime of the marginalized graph kernel $k(G, G')$ is easiest to check if we transform the above equations into matrix notation. For this purpose we define two matrices S and Q of size $n \times n$. Let S be defined as

$$S_{ij} = p_s(v_i)p'_s(v'_j)k_v(\mathcal{L}(v_i), \mathcal{L}'(v'_j)). \tag{1.40}$$

and Q as

$$Q_{ij} = p_q(v_i)p'_q(v'_j) \tag{1.41}$$

.

Furthermore, let T be a $n^2 \times n^2$ transition matrix:

$$T_{(i-1)*n+j,(i'-1)*n+j'} = p_t(v_j|v_i)p'_t(v'_{j'}|v'_{i'})k_v(\mathcal{L}(v_j), \mathcal{L}'(v'_{j'}))k_e(\mathcal{L}(v_i, v_j), \mathcal{L}'(v'_{i'}, v'_{j'})); \tag{1.42}$$

The matrix form of the kernel in terms of these three matrices is then [Kashima et al., 2003]

$$k(G, G') = ((I - T)^{-1} \operatorname{vec}(Q))' \operatorname{vec}(S) = \operatorname{vec}(Q)'(I - T)^{-1} \operatorname{vec}(S) \tag{1.43}$$

where the vec operator flattens an $n \times n$ matrix into an $n^2 \times 1$ vector and I is the identity matrix of size $n^2 \times n^2$. We observe that the computation of the marginalized kernel requires an inversion of an $n^2 \times n^2$ matrix. Like the random walk kernel, the runtime of the marginalized kernel on graphs is hence in $O(n^6)$.

Note the similarity between equation (1.43) and equation (1.31), *i.e.,* the definitions of the random walk kernel and the marginalized kernel on graphs. This similarity is not by chance. In Section 2.1, we will show that both these graph kernels are instances of a common unifying framework for walk-based kernels on graphs.

Discussion Graph kernels based on random walks intuitively seem to be a good measure of similarity on graphs, as they take the whole structure of the graph into account, but require polynomial runtime only. However, these kernels suffer from several weaknesses, which we will describe in the following.

Bad News: The Runtime Complexity Random walk kernels were developed as an alternative to the NP-hard subgraph kernel. So do these $O(n^6)$ graph kernels save the day? Unfortunately, although being polynomial, n^6 is a huge computational effort. For small graphs, n^6 operations (neglecting constant factors) are even more than 2^n operations, as you can see from Figure 1.5. Hence for graphs with less than 30 nodes, n^6 is slower than 2^n. Interestingly, the average node number for typical benchmark datasets frequently used in graph mining is less than 30 (MUTAG 17.7, PTC 26.7)!

This high computational runtime severely limits the applicability of random walk graph kernels on real-world data. It is not efficient enough for dealing with large datasets of graphs, and does not scale up to large graphs with many nodes. As our first contribution in this thesis, we show how to speed up the random walk kernel to $O(n^3)$ in Section 2.1.

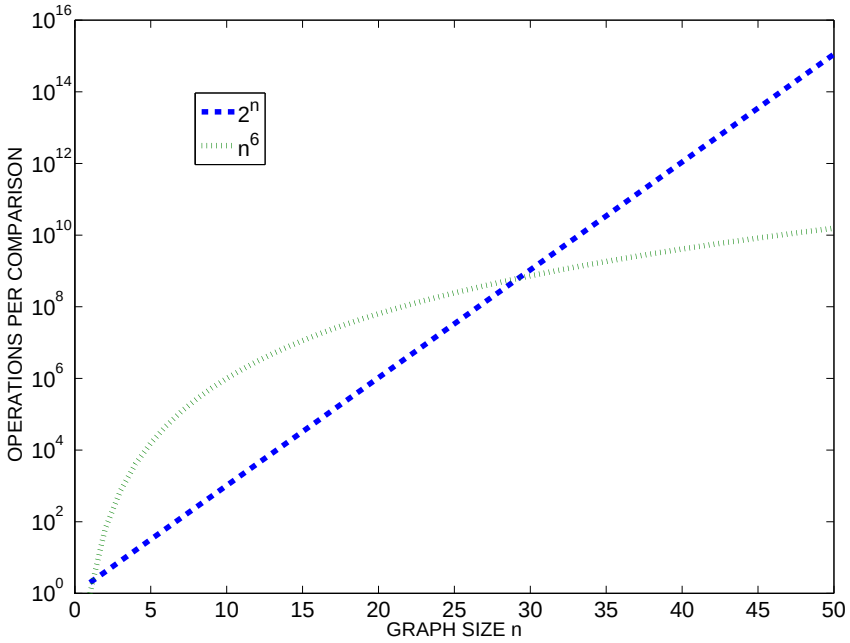

Figure 1.5: Runtime versus graph size n for two algorithms requiring n^6 and 2^n operations.

Tottering In addition to lack of efficiency, walk kernels suffer from a phenomenon called *tottering* [Mahé et al., 2004]. Walks allow for repetitions of nodes and edges, which means that the same node or edge is counted repeatedly in a similarity measure based on walks. In an undirected graph, a random walk may even start tottering between the same two nodes in the product graph, leading to an artificially high similarity score, which is caused by one single common edge in two graphs. Furthermore, a random walk on any cycle in the graph can in principle be infinitely long, and drastically increase the similarity score, although the structural similarity between two graphs is minor.

Halting Walk kernels show a second weakness. The decaying factor λ down-weights longer walks, which makes short walks dominate the similarity score. We describe this problem — which we refer to as "halting" — in more detail in Section 2.1.5. Approaches to overcome both halting and tottering are the topic of Section 2.2 and Section 2.3.

Due to the shortcomings of random walk kernels, extensions of these and alternative kernels have been defined in the literature. We will summarize these next.

Extensions of Marginalized Graph Kernels Mahe et al. [Mahé et al., 2004] designed two extensions of marginalized kernels to overcome a) the problem of tottering and b) their computational expensiveness. Both these extensions are particularly relevant for chemoinformatics applications.

The first extension is to relabel each node automatically in order to insert information about the neighborhood of each node in its label via the so-called *Morgan Index*. This has

both an effect in terms of feature relevance, because label paths contain information about their neighborhood as well, and computation time, because the number of identically labeled paths significantly decreases. This speed-up effect is successfully shown on real-world datasets. However, this node label enrichment could only slightly improve classification accuracy.

Second, they show how to modify the random walk model in order to remove tottering between 2 nodes (but not on cycles of longer length). This removal of length-2 tottering did not improve classification performance uniformly.

Subtree-Pattern Kernels As an alternative to walk kernels on graphs, graph kernels comparing subtree-patterns were defined in [Ramon and Gärtner, 2003]. Intuitively, this kernel considers all pairs of nodes V from G and V' from G' and iteratively compares their neighborhoods. 'Subtree-pattern' refers to the fact that this kernel counts subtree-like structures in two graphs. In contrast to the strict definition of trees, subtree-patterns may include several copies of the same node or edge. Hence they are not necessarily isomorphic to subgraphs of G or G', let alone subtrees of G and G'. To be able to regard these patterns as trees, [Ramon and Gärtner, 2003] treat copies of identical nodes and edges as if they were distinct nodes and edges.

More formally, let $G(V, E)$ and $G'(V', E')$ be two graphs. The idea of the subtree-pattern kernel $k_{v,v',h}$ is to count pairs of identical subtree-patterns in G and G' with height less than or equal to h, with the first one rooted at $v \in V(G)$ and the second one rooted at $v' \in V(G')$. Now, if $h = 1$ and $\mathcal{L}(v) = \mathcal{L}'(v')$ we have $k_{v,v',h} = 1$. If $h = 1$ and $\mathcal{L}(v) \neq \mathcal{L}'(v')$ we have $k_{v,v',h} = 0$. For $h > 1$, one can compute $k_{v,v',h}$ as follows:

- Let $M_{v,v'}$ be the set of all matchings from the set $\delta(v)$ of neighbors of v to the set $\delta(v')$ of neighbors of v', *i.e.,*

$$M_{v,v'} = \{R \subseteq \delta(v) \times \delta(v') | (\forall (v_i, v_i'), (v_j, v_j') \in R : v_i = v_i' \Leftrightarrow v_j = v_j') \wedge (\forall (v_k, v_k') \in R : \mathcal{L}(v_k) = \mathcal{L}'(v_k'))\} \tag{1.44}$$

- Compute

$$k_{v,v',h} = \lambda_v \lambda_{v'} \sum_{R \in M_{v,v'}} \prod_{(v,v') \in R} k_{v,v',h-1} \tag{1.45}$$

  Here λ_v and $\lambda_{v'}$ are positive values smaller than 1 to cause higher trees to have a smaller weight in the overall sum.

Given two graphs $G(V, E)$, $G'(V', E')$, then the subtree-pattern kernel of G and G' is given by

$$k_{tree,h}(G, G') = \sum_{v \in V} \sum_{v' \in V'} k_{v,v',h}. \tag{1.46}$$

As the walk kernel, the subtree pattern kernel suffers from tottering. Due to the more complex patterns it examines, its runtime is even worse than that of the random walk kernel. It grows exponentially with the height h of the subtree-patterns considered.

Cyclic Pattern Kernels [Horvath et al., 2004] decompose a graph into cyclic patterns, then count the number of common cyclic patterns which occur in both graphs. Their kernel is plagued by computational issues; in fact they show that computing the cyclic pattern kernel on a general graph is NP-hard. They consequently restrict their attention to practical problem classes where the number of simple cycles is bounded.

Fingerprint and Depth First Search Kernels [Ralaivola et al., 2005] define graph kernels based on molecular fingerprints and length-d paths from depth-first search. These kernels are tailored for applications in chemical informatics, and exploit the small size and low average degree of these molecular graphs.

Optimal Assignment Kernels In the aforementioned graph kernels, R-convolution often boils down to an all-pairs comparison of substructures from two composite objects. Intuitively, finding a best match, an optimal assignment between the substructures from G and G' would be more attractive than an all-pairs comparison. In this spirit, [Fröhlich et al., 2005] define an optimal assignment kernel on composite objects that include graphs as a special instance.

Definition 22 (Optimal Assignment Kernel) *Let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be some non-negative, symmetric and positive definite kernel. Assume that x and x' are two composite objects that have been decomposed into their parts $\boldsymbol{x} := (x_1, x_2, \ldots, x_{|\boldsymbol{x}|})$ and $\boldsymbol{x}' := (x'_1, x'_2, \ldots, x'_{|\boldsymbol{x}'|})$. Let $\Pi(\boldsymbol{x})$ denote all possible permutations of $\boldsymbol{x}$, and analogously $\Pi(\boldsymbol{x}')$ all possible permutations of $\boldsymbol{x}'$.*

Then $k_A : \mathcal{X} \times \mathcal{X} \to R$ with

$$k_A(x, x') := \begin{cases} \max_{\pi \in \Pi(\boldsymbol{x}')} \sum_{i=1}^{|\boldsymbol{x}|} \kappa(x_i, x'_{\pi(i)}) & \text{if } |\boldsymbol{x}'| > |\boldsymbol{x}|, \\ \max_{\pi \in \Pi(\boldsymbol{x})} \sum_{j=1}^{|\boldsymbol{x}'|} \kappa(x_{\pi(j)}, x'_j) & \text{otherwise} \end{cases} \tag{1.47}$$

is called an optimal assignment kernel.

While based on a nice idea, the optimal assignment kernel is unfortunately not positive definite [Vishwanathan et al., 2007b], seriously limiting its use in SVMs and other kernel methods.

Other Graph Kernels Two more types of graph kernels have been described in the literature: Graph edit distance kernels that employ results of a graph edit distance to give extra weight to matching vertices [Neuhaus, 2006], and weighted decomposition kernels that decompose a graph into small subparts and reward similarity of these subparts with different weights [Menchetti et al., 2005]. However, while the former fails to be positive definite, the latter can only deal with highly simplified representations of graphs efficiently.

Quality Criteria in Graph Kernel Design From our review on the state-of-the-art in graph kernels, it becomes apparent that all current graph kernels suffer from different kinds of weaknesses. The open question remains: How to design a 'good' graph kernel? The definition of 'good' is the key to answering this question. Here we try to define several central requirements graph kernel design has to fulfill to yield a good graph kernel. A good

graph kernel that is theoretically sound and widely applicable should show the following characteristics:

- **positive definiteness**. A valid kernel function guarantees a global optimal solution when this graph kernel is employed within a convex optimization problem, as in SVMs.

- **not restricted to special class of graphs**. While kernels that are specialized to certain classes of graphs may be helpful in some applications, it is much more attractive to define a graph kernel that is generally applicable. This way, one needs not worry if the graph kernel is applicable to the particular problem at hand.

- **efficient to compute**. In practice it is not only desirable to theoretically define a kernel on graphs, but also to guarantee that it is fast to compute and has a low theoretical runtime complexity. A graph kernel needs to be efficient to compute, because otherwise one may also employ one of the many expensive graph matching and graph isomorphism approaches from Section 1.3, and then apply a kernel on the similarity scores obtained by these approaches.

- **expressive**. A graph kernel has to represent an expressive, non-trivial measure of similarity on graphs. It has to compare features or subgraphs of two graphs that allow to really tell if the topology and/or node and edge labels of two graphs are similar.

Some of these goals may be at loggerheads. Graph kernels for special classes of graphs, for example trees, can be computed highly efficiently, requiring quadratic [Lodhi et al., 2002] or with canonical ordering even linear runtime [Vishwanathan and Smola, 2004]. These kernels, however, cannot be applied to graphs in general. The graph kernels proposed in [Neuhaus, 2006] are expressive measures of similarity on graphs, but they lack validity, *i.e.,* they are not positive definite. The all-subgraphs kernel [Gärtner et al., 2003] is extremely expressive, as it considers all pairs of common subgraphs from two graphs, but its computation is NP-hard.

For all these reasons, one central challenge in this thesis was the development of graph kernels that overcome the limitations of current graph kernels.

1.5 Contributions of this Thesis

The goal of this thesis was to define fast graph kernel functions and novel kernel methods for solving graph problems in data mining and bioinformatics.

Our interest in graph kernels derives from the vast number of applications in which graph data started to emerge over recent years (see Section 1.1.1), and the fact that current similarity measures on graphs are either NP-hard to compute, or resort to heuristics or simplified representations of graphs (see Section 1.3). Furthermore, graph kernels benefit extremely from two characteristics shared by all kernel functions:

- First, we do not need to know the mapping ϕ to feature space explicitly. As a consequence, we can compare graphs in feature spaces without explicitly computing these graph features, unlike all approaches using topological descriptors which require this explicit determination. Note that we can use this trick on different levels of a graphs, *i.e.,* on its topology, and on its node and edge labels.

- Second, graph kernels offer a powerful method to extend statistical machine learning and data mining to graphs. By defining a kernel on graphs, a whole battery of kernel methods and kernel-based algorithms becomes applicable to graphs. Hence graph kernels could bridge the gap between statistical pattern recognition on vectors and structural graph pattern recognition on graphs.

Still, as we have seen in Section 1.4.2, defining a 'good' graph kernel is extremely difficult. Known graph kernels are either not efficient to compute, restricted to a subclass of graphs, not positive definite or not an expressive measure of similarity on graphs. None of the existing graph kernel fulfills all our four quality criteria.

1.5.1 Fast Graph Kernels

Above all, efficiency is a major bottleneck of state-of-the-art graphs kernels. As mentioned in Section 1.4.2, all-subgraphs kernels are NP-hard to compute, and classic random walk kernels require a runtime of $O(n^6)$ where n is the number of nodes in the larger of the two graphs. While polynomial, this runtime is too expensive even for large graphs and large datasets of small graphs. In fact, instead of computing such an expensive graph kernel, one could extract features from a graph to obtain a feature vector representation of this graph, and to then apply known feature vector methods from pattern recognition to these vectors. Hence the unique advantages of graph kernels, their ability to bridge feature vector and graph-based learning, can only be exploited if they are efficient to compute.

In Section 2.1, we extend concepts from linear algebra to Reproducing Kernel Hilbert Spaces, to speed up the computation of the classic geometric random walk kernel. The speed-up reduces the computational effort to $O(n^3)$, and leads to a more than 1,000 times lower CPU runtime on real-world datasets. In Section 2.2, we define a graph kernel based on shortest paths which does not suffer from tottering. It is a positive definite and expressive measure for graph similarity, which is computable in $O(n^4)$ and applicable to a large class of graphs. In Section 2.3, we develop graph kernels for comparing large graphs with hundreds and thousands of nodes. The underlying idea, motivated by the graph reconstruction conjecture, is to sample small subgraphs from large graphs, and to approximate the distribution of these small subgraphs within the large graph. In this manner, we are able to compute graph kernels on graphs that are too large for state-of-the-art methods.

1.5.2 Two-Sample Test on Graphs

Besides the traditional learning tasks such as classification and clustering, there are other learning problems that could not be solved on graphs so far, for instance, two-sample tests. Two-sample tests try to answer the question whether two samples — in our case two sets of graphs — are likely to originate from the same underlying distribution. This question is of

interest in data integration, especially when fusing graph data that originate from different sources, e.g. different laboratories or databases. While two-sample tests for vectors have been a research topic in statistics for decades, no such test has been developed for graphs.

In Chapter 3, we develop a two-sample test for graphs. We proceed in three steps. First, we present the first two-sample test based on kernels in Section 3.1. Second, in Section 3.2.1 we show how this kernelized two-sample test based on a test statistic called Maximum Mean Discrepancy can be applied to sets of graphs by employing a graph kernel. Third, in Section 3.2.2, we extend the concept of Maximum Mean Discrepancy from comparing sets of graphs to comparing pairs of graphs. This way, we yield the first statistical test of graph similarity described in the literature.

1.5.3 Efficient Feature Selection on Graphs

In Chapter 4, we tackle the problem of feature selection on graphs. While constructing an accurate classifier for assigning graphs to different categories is an interesting task, it is equally relevant to understand which features of graphs are most correlated to its class membership. No principled approach to this problem of feature selection on graphs has been proposed so far.

In this chapter, we present an efficient procedure for feature selection on graphs. In Section 4.1, we define a family of kernel-based feature selection algorithms. They employ the Hilbert-Schmidt Independence Criterion ($HSIC$) [Gretton et al., 2005] for measuring dependence between data objects and their class labels. In Section 4.2, we extend this principle to feature selection on graphs. We show that for one particular choice of kernel, HSIC-based feature selection can cope with the huge search space encountered in feature selection on graphs. We successfully combine our feature selector with the state-of-the-art method for frequent graph mining, gSpan [Yan and Han, 2002], and manage to select an informative subset of a few dozens of features from the thousands and millions of features found by gSpan.

1.5.4 Applications in Data Mining and Bioinformatics

All our findings have immediate applications in data mining and bioinformatics. These reach from biological network comparison to efficient frequent subgraph mining. We have already explored several of these:

- Protein function prediction via graph kernels [Borgwardt et al., 2005]
- Protein interaction network comparison via graph kernels [Borgwardt et al., 2007c]
- Integration of structured data and automatic ontology matching via Maximum Mean Discrepancy [Borgwardt et al., 2006]
- Gene selection from microarray data via HSIC [Song et al., 2007a]

Apart from these, the new algorithmic and statistical concepts we define as part of our novel graph kernels and kernel methods may contribute to the development of new machine learning and data mining algorithms.

Fast Graph Kernels	
Shortest-Path Kernels	ICDM 2005 [Borgwardt and Kriegel, 2005]
Fast Computation of Random Walk Kernels	NIPS 2006 [Vishwanathan et al., 2007a],
Graphlet Kernels	*under preparation* [Borgwardt et al., 2007a]
Kernel Methods for Novel Problems on Graphs	
Kernel Method for Two-Sample Problem	NIPS 2006 [Gretton et al., 2007a]
Feature Selection using HSIC	ICML 2007 [Song et al., 2007c]
Feature Selection on Graphs	*under preparation* [Borgwardt et al., 2007b]
Applications in Bioinformatics	
Protein Function Prediction	ISMB 2005 [Borgwardt et al., 2005]
Protein Interaction Network Comparison	PSB 2007 [Borgwardt et al., 2007c]
Data Integration in Bioinformatics	ISMB 2006 [Borgwardt et al., 2006]
Gene Selection from Microarray Data	ISMB 2007 [Song et al., 2007a]

Table 1.1: Contributions of this thesis and accompanying publications.

In Chapter 5, we summarize both our work on applications in bioinformatics and give an outlook to future challenges and chances for our graph kernels and kernel methods in the field of algorithms and bioinformatics.

We summarize all our contributions with their accompanying publications in Table 1.1.

Chapter 2

Fast Graph Kernel Functions

As we have stressed before, the key challenge in graph kernel design is to define positive definite kernels that are both an expressive measure of similarity for graphs and that are efficient to compute, and not restricted to a subclass of graphs. We have also explained that random walk kernels suffer from several shortcomings: above all high computational runtime and tottering, limiting their efficiency, scalability and expressivity.

In this chapter, we overcome these limitations step by step. In Section 2.1, we employ techniques from numerical algebra to speed up the classic random walk kernels [Gärtner et al., 2003, Kashima et al., 2003] to $O(n^3)$ in theoretical runtime, and by up to a factor of 1,000 in CPU runtime. In this manner, we make the classic random walk kernel more efficient.

In Section 2.2, we define a novel graph kernel based on shortest path distances. It outperforms the random walk kernel in experimental runtime and it avoids tottering, thus improving upon the random walk kernel both in terms of efficiency and expressivity.

To scale graph kernels up to large graphs with hundreds and thousands of nodes, we propose a second class of novel graph kernels in Section 2.3. These *graphlet kernels* count common subgraphs with 4 nodes in two graphs, without tottering. We establish an efficient sampling scheme for estimating the distribution of these small subgraphs within a given graph. As a further improvement over the random walk kernel, these graphlet kernels are efficient, expressive and even scalable to large graphs, and can tackle problems on graph sizes that were beyond the scope of graph kernels so far.

2.1 Fast Computation of Random Walk Graph Kernels

In this section, we speed up the classic random walk kernel. Towards this end, we extend common concepts from linear algebra to Reproducing Kernel Hilbert Spaces (RKHS), and use these extensions to define a unifying framework for random walk kernels including those of [Gärtner et al., 2003] and [Kashima et al., 2003]. We show that computing many random walk graph kernels can be reduced to the problem of solving a large linear system, which can then be solved efficiently by a variety of methods which exploit the structure of the problem. In this fashion, we are able to speed up the computation of the classic random walk kernel.

2.1.1 Extending Linear Algebra to RKHS

Let $\phi : \mathcal{X} \to \mathcal{H}$ denote the feature map from an input space $\mathcal{X}$ to the RKHS $\mathcal{H}$ associated with the kernel $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$. Given an n by m matrix $X \in \mathcal{X}^{n \times m}$ of elements $X_{ij} \in \mathcal{X}$, we extend ϕ to matrix arguments by defining $\Phi : \mathcal{X}^{n \times m} \to \mathcal{H}^{n \times m}$ via $[\Phi(X)]_{ij} := \phi(X_{ij})$. We can now borrow concepts from tensor calculus to extend certain linear algebra operations to $\mathcal{H}$:

Definition 23 *Let $A \in \mathcal{X}^{n \times m}$, $B \in \mathcal{X}^{m \times p}$, and $C \in \mathbb{R}^{m \times p}$. The matrix products $\Phi(A)\Phi(B) \in \mathbb{R}^{n \times p}$ and $\Phi(A)\, C \in \mathcal{H}^{n \times p}$ are*

$$[\Phi(A)\Phi(B)]_{ik} := \sum_j \langle \phi(A_{ij}), \phi(B_{jk}) \rangle_{\mathcal{H}} \qquad \text{and} \qquad [\Phi(A)\, C]_{ik} := \sum_j \phi(A_{ij})\, C_{jk}.$$

Given $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{p \times q}$ the Kronecker product $A \otimes B \in \mathbb{R}^{np \times mq}$ and vec operator are defined as

$$A \otimes B := \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1m}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nm}B \end{bmatrix}, \quad \operatorname{vec}(A) := \begin{bmatrix} A_{*1} \\ \vdots \\ A_{*m} \end{bmatrix}, \tag{2.1}$$

where A_{*j} denotes the j-th column of A. They are linked by the well-known property [Golub and Van Loan, 1996]:

$$\operatorname{vec}(ABC) = (C^\top \otimes A) \operatorname{vec}(B). \tag{2.2}$$

Definition 24 *Let $A \in \mathcal{X}^{n \times m}$ and $B \in \mathcal{X}^{p \times q}$. The Kronecker product $\Phi(A) \otimes \Phi(B) \in \mathbb{R}^{np \times mq}$ is*

$$[\Phi(A) \otimes \Phi(B)]_{(i-1)p+k,(j-1)q+l} := \langle \phi(A_{ij}), \phi(B_{kl}) \rangle_{\mathcal{H}}. \tag{2.3}$$

It is easily shown that the above extensions to RKHS obey an analogue of (2.2):

Lemma 1 *If $A \in \mathcal{X}^{n \times m}$, $B \in \mathbb{R}^{m \times p}$, and $C \in \mathcal{X}^{p \times q}$, then*

$$\operatorname{vec}(\Phi(A)\, B\, \Phi(C)) = (\Phi(C)^\top \otimes \Phi(A)) \operatorname{vec}(B). \tag{2.4}$$

If $p = q = n = m$, direct computation of the right hand side of (2.4) requires $O(n^4)$ kernel evaluations. For an arbitrary kernel the left hand side also requires a similar effort. But, if the RKHS $\mathcal{H}$ is isomorphic to $\mathbb{R}^r$, in other words the feature map $\phi(\cdot) \in \mathbb{R}^r$, the left hand side of (2.4) is easily computed in $O(n^3 r)$ operations. Our efficient computation schemes described in Subsection 2.1.3 will exploit this observation.

2.1.2 Random Walk Kernels

As summarized in Section 1.4.2, random walk kernels on graphs are based on a simple idea: Given a pair of graphs, perform a random walk on both of them and *count* the number of *matching* walks [Gärtner et al., 2003, Kashima et al., 2003]. These kernels mainly differ in the way the similarity between random walks is computed. For instance, the product graph kernel by [Gärtner et al., 2003] counts the number of nodes in the random walk which have the same label. They also include a decaying factor to ensure convergence. The marginalized graph kernels by [Kashima et al., 2003] use a kernel defined on nodes and edges in order to compute similarity between random walks, and define an initial probability distribution over nodes in order to ensure convergence. In this section we present a unifying framework which includes the above mentioned kernels as special cases.

Notation

We need some additional notation to present our schemes for fast graph kernel computation. We use $\mathbf{e}_i$ to denote the i-th standard basis (*i.e.,* a vector of all zeros with the i-th entry set to one), $\mathbf{1}$ to denote a vector with all entries set to one, $\mathbf{0}$ to denote the vector of all zeros, and I to denote the identity matrix. When it is clear from context we will not mention the dimensions of these vectors and matrices.

Recall that the (unnormalized) adjacency matrix of a graph $G = (V, E)$ is an $n \times n$ real matrix A with $A_{ij} = 1$ if $(v_i, v_j) \in E$, and 0 otherwise. If G is weighted then A can contain non-negative entries other than zeros and ones, *i.e.,* $A_{ij} \in (0, \infty)$ if $(v_i, v_j) \in E$ and zero otherwise. Let D be an $n \times n$ diagonal matrix with entries $D_{ii} = \sum_j A_{ij}$. The matrix $P := AD^{-1}$ is then called the normalized adjacency matrix.

Recall from Definition 6 that a *walk* w in a graph G is a non-empty alternating sequence $(v_1, e_1, v_2, e_2, \ldots, e_{\ell-1}, v_\ell)$ of nodes and edges in G such that $e_i = \{v_i, v_{i+1}\}$ for all $1 \leq i \leq \ell - 1$. The length of a walk is equal to the number of edges encountered during the walk (here: $\ell - 1$). A random walk is a walk where $\mathbb{P}(w_{i+1}|w_1, \ldots w_i) = P_{w_i, w_{i+1}}$, *i.e.,* the probability at w_i of picking w_{i+1} next is directly proportional to the weight of the edge $(v_{w_i}, v_{w_{i+1}})$. The ℓ-th power of the transition matrix P describes the probability of ℓ-length walks. In other words, $[P^\ell]_{ij}$ denotes the probability of a transition from node v_i to node v_j via a walk of length ℓ. We use this intuition to define random walk kernels on graphs.

Let $\mathcal{X} \subset \mathcal{Z}$ be a set of labels which includes the special label ϵ. Every edge-labeled graph G is associated with a label matrix $L \in \mathcal{X}^{n \times n}$, such that $L_{ij} = \epsilon$ iff $(v_i, v_j) \notin E$, in other words only those edges which are present in the graph get a non-ϵ label. Let $\mathcal{H}$ be the RKHS endowed with the kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and let $\phi : \mathcal{X} \to \mathcal{H}$ denote the corresponding feature map which maps ϵ to the zero element of $\mathcal{H}$. We use $\Phi(L)$ to denote

the feature matrix of G. For ease of exposition we do not consider labels on nodes here, though our results hold for that case as well. In the remainder of this section, we use the term labeled graph to denote an edge-labeled graph.

Product Graphs

Given two graphs $G(V, E)$ of size n and $G'(V', E')$ of size n', the product graph $G_\times(V_\times, E_\times)$ is a graph with nn' nodes, each representing a pair of nodes from G and G', respectively (see Definition 20). An edge exists in $E_\times$ iff the corresponding nodes are adjacent in both G and G'. Thus

$$V_\times = \{(v_i, v'_{i'}) : v_i \in V \wedge v'_{i'} \in V'\}, \tag{2.5}$$
$$E_\times = \{((v_i,v'_{i'}), (v_j,v'_{j'})) : (v_i, v_j) \in E \wedge (v'_{i'}, v'_{j'}) \in E'\}. \tag{2.6}$$

If A and A' are the adjacency matrices of G and G', respectively, the adjacency matrix of the product graph $G_\times$ is $A_\times = A \otimes A'$. An edge exists in the product graph iff an edge exits in both G and G', therefore performing a simultaneous random walk on G and G' is equivalent to performing a random walk on the product graph [Harary, 1969].

Let p and p' denote initial probability distributions over nodes of G and G'. Then the initial probability distribution $p_\times$ of the product graph is $p_\times := p \otimes p'$. Likewise, if q and q' denote stopping probabilities (*i.e.,* the probability that a random walk ends at a given node), the stopping probability $q_\times$ of the product graph is $q_\times := q \otimes q'$.

If G and G' are edge-labeled, we can associate a weight matrix $W_\times \in \mathbb{R}^{nn' \times nn'}$ with $G_\times$, using our Kronecker product in RKHS (Definition 24): $W_\times = \Phi(L) \otimes \Phi(L')$. As a consequence of the definition of $\Phi(L)$ and $\Phi(L')$, the entries of $W_\times$ are non-zero only if the corresponding edge exists in the product graph. The weight matrix is closely related to the adjacency matrix: assume that $\mathcal{H} = \mathbb{R}$ endowed with the usual dot product, and $\phi(L_{ij}) = 1$ if $(v_i, v_j) \in E$ or zero otherwise. Then $\Phi(L) = A$ and $\Phi(L') = A'$, and consequently $W_\times = A_\times$, *i.e.,* the weight matrix is identical to the adjacency matrix of the product graph.

To extend the above discussion, assume that $\mathcal{H} = \mathbb{R}^d$ endowed with the usual dot product, and that there are d distinct edge labels $\{1, 2, \ldots, d\}$. For each edge $(v_i, v_j) \in E$ we have $\phi(L_{ij}) = \mathbf{e}_l$ if the edge (v_i, v_j) is labeled l. All other entries of $\Phi(L)$ are set to $\mathbf{0}$. κ is therefore a delta kernel, *i.e.,* its value between any two edges is one iff the labels on the edges match, and zero otherwise. The weight matrix $W_\times$ has a non-zero entry iff an edge exists in the product graph and the corresponding edges in G and G' have the same label. Let ${}^l\!A$ denote the adjacency matrix of the graph filtered by the label l, *i.e.,* ${}^l\!A_{ij} = A_{ij}$ if $L_{ij} = l$ and zero otherwise. Some simple algebra shows that the weight matrix of the product graph can be written as

$$W_\times = \sum_{l=1}^{d} {}^l\!A \otimes {}^l\!A'. \tag{2.7}$$

Kernel Definition

Performing a random walk on the product graph $G_\times$ is equivalent to performing a simultaneous random walk on the graphs G and G' [Harary, 1969]. Therefore, the $((i-1)n+j,(i'-1)n'+j')$-th entry of $A_\times^k$ represents the probability of simultaneous k length random walks on G (starting from node v_i and ending in node v_j) and G' (starting from node $v'_{i'}$ and ending in node $v'_{j'}$). The entries of $W_\times$ represent similarity between edges. The $((i-1)n+j,(i'-1)n'+j')$-th entry of $W_\times^k$ represents the similarity between simultaneous k length random walks on G and G' measured via the kernel function κ.

Given the weight matrix $W_\times$, initial and stopping probability distributions $p_\times$ and $q_\times$, and an appropriately chosen discrete measure μ, we can define a random walk kernel on G and G' as

$$k(G,G') := \sum_{k=0}^{\infty} \mu(k)\, q_\times^\top W_\times^k p_\times. \tag{2.8}$$

In order to show that (2.8) is a valid Mercer kernel we need the following technical lemma.

Lemma 2 $\forall\ k \in \mathbb{N}_0 :\quad W_\times^k p_\times = \mathrm{vec}[\Phi(L')^k p'\,(\Phi(L)^k p)^\top]$.

Proof By induction over k. Base case: $k=0$. Since $\Phi(L')^0 = \Phi(L)^0 = I$, using (2.2) we can write

$$W_\times^0 p_\times = p_\times = (p \otimes p')\,\mathrm{vec}(1) = \mathrm{vec}(p'\,1\,p^\top) = \mathrm{vec}[\Phi(L')^0 p'\,(\Phi(L)^0 p)^\top].$$

Induction from k to $k+1$: Using Lemma 1 we obtain

$$\begin{aligned} W_\times^{k+1} p_\times &= W_\times W_\times^k p_\times = (\Phi(L)\otimes\Phi(L'))\,\mathrm{vec}[\Phi(L')^k p'\,(\Phi(L)^k p)^\top] \\ &= \mathrm{vec}[\Phi(L')\Phi(L')^k p'\,(\Phi(L)^k p)^\top \Phi(L)^\top] = \mathrm{vec}[\Phi(L')^{k+1} p'\,(\Phi(L)^{k+1} p)^\top]. \end{aligned}$$

■

Lemma 3 *If the measure $\mu(k)$ is such that* (2.8) *converges, then it defines a valid Mercer kernel.*

Proof Using Lemmas 1 and 2 we can write

$$\begin{aligned} q_\times^\top W_\times^k p_\times &= (q\otimes q')\,\mathrm{vec}[\Phi(L')^k p'\,(\Phi(L)^k p)^\top] = \mathrm{vec}[q'^\top \Phi(L')^k p'\,(\Phi(L)^k p)^\top q] \\ &= \underbrace{(q^\top \Phi(L)^k p)^\top}_{\psi_k(G)^\top}\,\underbrace{(q'^\top \Phi(L')^k p')}_{\psi_k(G')}. \end{aligned}$$

Each individual term of (2.8) equals $\psi_k(G)^\top \psi_k(G')$ for some function ψ_k, and is therefore a valid kernel. The lemma follows since a convex combination of kernels is itself a valid kernel, if we choose $\mu(k)$ to be nonnegative. ■

Special Cases

A popular choice to ensure convergence of (2.8) is to assume $\mu(k) = \lambda^k$ for some $\lambda > 0$. If λ is sufficiently small[1] then (2.8) is well defined, and we can write

$$k(G, G') = \sum_{k=0}^{\infty} \lambda^k q_\times^\top W_\times^k p_\times = q_\times^\top (I - \lambda W_\times)^{-1} p_\times. \tag{2.9}$$

Marginalized Graph Kernels As we have seen in Section 1.4, [Kashima et al., 2003] use marginalization and probabilities of random walks to define kernels on graphs. Given transition probability matrices P and P' associated with graphs G and G' respectively, their kernel can be written as (see Eq. 1.19, [Kashima et al., 2003])

$$k(G, G') = q_\times^\top (I - T_\times)^{-1} p_\times, \tag{2.10}$$

where $T_\times := (\mathrm{vec}(P)\,\mathrm{vec}(P')^\top) \odot (\Phi(L) \otimes \Phi(L'))$, using $\odot$ to denote element-wise (Hadamard) multiplication. The edge kernel $\hat{\kappa}(L_{ij}, L'_{i'j'}) := P_{ij} P'_{i'j'} \kappa(L_{ij}, L'_{i,j'})$ with $\lambda = 1$ recovers (2.9).

Product Graph Kernels [Gärtner et al., 2003] use the adjacency matrix of the product graph to define the so-called geometric kernel

$$k(G, G') = \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \lambda^k [A_\times^k]_{ij}. \tag{2.11}$$

To recover their kernel in our framework, assume an uniform distribution over the nodes of G and G', *i.e.,* set $p = q = 1/n$ and $p' = q' = 1/n'$. The initial as well as final probability distribution over nodes of $G_\times$ is given by $p_\times = q_\times = \mathbf{1}/(nn')$. Setting $\Phi(L) := A$, and hence $\Phi(L') = A'$ and $W_\times = A_\times$, we can rewrite (2.8) to obtain

$$k(G, G') = \sum_{k=0}^{\infty} \lambda^k q_\times^\top A_\times^k p_\times = \frac{1}{n^2 n'^2} \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \lambda^k [A_\times^k]_{ij},$$

which recovers (2.11) to within a constant factor.

2.1.3 Efficient Computation

In this subsection we show that iterative methods, including those based on Sylvester equations, conjugate gradients, and fixed-point iterations, can be used to greatly speed up the computation of (2.9).

Sylvester Equation Methods

Consider the following equation, commonly known as the Sylvester or Lyapunov equation:

$$X = SXT + X_0. \tag{2.12}$$

[1] The values of λ which ensure convergence depends on the spectrum of $W_\times$.

Here, $S, T, X_0 \in \mathbb{R}^{n \times n}$ are given and we need for solve for $X \in \mathbb{R}^{n \times n}$. These equations can be readily solved in $O(n^3)$ time with freely available code [Gardiner et al., 1992], *e.g.* Matlab's `dlyap` method. The generalized Sylvester equation

$$X = \sum_{i=1}^{d} S_i X T_i + X_0 \tag{2.13}$$

can also be solved efficiently, albeit at a slightly higher computational cost of $O(dn^3)$.

We now show that if the weight matrix $W_\times$ can be written as (2.7) then the problem of computing the graph kernel (2.9) can be reduced to the problem of solving the following Sylvester equation:

$$X = \sum_{i=1}^{d} {}^{i}\!A' \lambda X \, {}^{i}\!A^\top + X_0, \tag{2.14}$$

where $\mathrm{vec}(X_0) = p_\times$. We begin by *flattening* the above equation:

$$\mathrm{vec}(X) = \lambda \sum_{i=1}^{d} \mathrm{vec}({}^{i}\!A' X \, {}^{i}\!A^\top) + p_\times. \tag{2.15}$$

Using Lemma 1 we can rewrite (2.15) as

$$(I - \lambda \sum_{i=1}^{d} {}^{i}\!A \otimes {}^{i}\!A') \, \mathrm{vec}(X) = p_\times, \tag{2.16}$$

use (2.7), and solve for $\mathrm{vec}(X)$:

$$\mathrm{vec}(X) = (I - \lambda W_\times)^{-1} p_\times. \tag{2.17}$$

Multiplying both sides by $q_\times^\top$ yields

$$q_\times^\top \mathrm{vec}(X) = q_\times^\top (I - \lambda W_\times)^{-1} p_\times. \tag{2.18}$$

The right-hand side of (2.18) is the graph kernel (2.9). Given the solution X of the Sylvester equation (2.14), the graph kernel can be obtained as $q_\times^\top \mathrm{vec}(X)$ in $O(n^2)$ time. Since solving the generalized Sylvester equation takes $O(dn^3)$ time, computing the graph kernel in this fashion is significantly faster than the $O(n^6)$ time required by the direct approach.

Where the number of labels d is large, the computational cost may be reduced further by computing matrices S and T such that $W_\times \approx S \otimes T$. We then simply solve the simple Sylvester equation (2.12) involving these matrices. Finding the nearest Kronecker product approximating a matrix such as $W_\times$ is a well-studied problem in numerical linear algebra and efficient algorithms which exploit sparsity of $W_\times$ are readily available [Van Loan, 2000].

Conjugate Gradient Methods

Given a matrix M and a vector b, conjugate gradient (CG) methods solve the system of equations $Mx = b$ efficiently [Nocedal and Wright, 1999]. While they are designed for symmetric positive semi-definite matrices, CG solvers can also be used to solve other linear systems efficiently. They are particularly efficient if the matrix is rank deficient, or has a small *effective rank*, *i.e.,* number of distinct eigenvalues. Furthermore, if computing matrix-vector products is cheap — because M is sparse, for instance — the CG solver can be sped up significantly [Nocedal and Wright, 1999]. Specifically, if computing Mv for an arbitrary vector v requires $O(k)$ time, and the effective rank of the matrix is m, then a CG solver requires only $O(mk)$ time to solve $Mx = b$.

The graph kernel (2.9) can be computed by a two-step procedure: First we solve the linear system

$$(I - \lambda W_\times)\, x = p_\times, \tag{2.19}$$

for x, then we compute $q_\times^\top x$. We now focus on efficient ways to solve (2.19) with a CG solver. Recall that if G and G' contain n nodes each then $W_\times$ is a $n^2 \times n^2$ matrix. Directly computing the matrix-vector product $W_\times r$, requires $O(n^4)$ time. Key to our speed-ups is the ability to exploit Lemma 1 to compute this matrix-vector product more efficiently: Recall that $W_\times = \Phi(L) \otimes \Phi(L')$. Letting $r = \mathrm{vec}(R)$, we can use Lemma 1 to write

$$W_\times r = (\Phi(L) \otimes \Phi(L'))\,\mathrm{vec}(R) = \mathrm{vec}(\Phi(L')R\,\Phi(L)^\top). \tag{2.20}$$

If $\phi(\cdot) \in \mathbb{R}^s$ for some s, then the above matrix-vector product can be computed in $O(n^3 s)$ time. If $\Phi(L)$ and $\Phi(L')$ are sparse, however, then $\Phi(L')R\,\Phi(L)^\top$ can be computed yet more efficiently: if there are $O(n)$ non-ϵ entries in $\Phi(L)$ and $\Phi(L')$, then computing (2.20) requires only $O(n^2)$ time.

Fixed-Point Iterations

Fixed-point methods begin by rewriting (2.19) as

$$x = p_\times + \lambda W_\times x. \tag{2.21}$$

Now, solving for x is equivalent to finding a fixed point of the above iteration [Nocedal and Wright, 1999]. Letting x_t denote the value of x at iteration t, we set $x_0 := p_\times$, then compute

$$x_{t+1} = p_\times + \lambda W_\times x_t \tag{2.22}$$

repeatedly until $||x_{t+1} - x_t|| < \varepsilon$, where $||\cdot||$ denotes the Euclidean norm and ε some pre-defined tolerance. This is guaranteed to converge if all eigenvalues of $\lambda W_\times$ lie inside the unit disk; this can be ensured by setting $\lambda < 1/\xi_{\max}$, where $\xi_{\max}$ is the largest-magnitude eigenvalue of $W_\times$.

The above is closely related to the power method used to compute the largest eigenvalue of a matrix [Golub and Van Loan, 1996]; efficient preconditioners can also be used

to speed up convergence [Golub and Van Loan, 1996]. Since each iteration of (2.22) involves computation of the matrix-vector product $W_\times x_t$, all speed-ups for computing the matrix-vector product as discussed for conjugate gradient methods are applicable here. In particular, we exploit the fact that $W_\times$ is a sum of Kronecker products to reduce the worst-case time complexity to $O(n^3)$ in our experiments, in contrast to [Kashima et al., 2003] who computed the matrix-vector product explicitly.

2.1.4 Experiments

We present two sets of experiments. First, we work with randomly generated graphs and study the scaling behavior of our algorithms. Second, we assess the practical impact of our algorithmic improvement by comparing the time taken to compute graph kernels on four real-world datasets.

For all our experiments our baseline comparator is the direct approach of [Gärtner et al., 2003]. All code was written in MATLAB Release 14, and experiments run on a 2.6 GHz Intel Pentium 4 PC with 2 GB of main memory running Suse Linux. The Matlab function `dlyap` was used to solve the Sylvester equation.

By default, we used a value of $\lambda = 0.001$, and set the tolerance for both CG solver and fixed-point iteration to 10^{-6} for all our experiments. We used Lemma 1 to speed up matrix-vector multiplication for both CG and fixed-point methods (see Section 2.1.3). Since all our methods are exact and produce the same kernel values (to numerical precision), we only report their runtimes below. Classification accuracies on these datasets will be reported in Sections 2.2 and 2.3, when we compare the performance of the random walk to that of other graph kernels.

Synthetic Datasets

The aim here is to study the scaling behavior of our algorithms on graphs of different sizes and different node degrees. We generated two sets of graphs: for the first set, SET-1, we begin with an empty graph of size 2^k, $k = 1, 2, \ldots, 10$, and randomly insert edges until the average degree of each node is at least 2. For each k we repeat the process 10 times and generate 10 graphs of size 2^k. The time required to compute the 10×10 kernel matrix for each value of k is depicted in Figure 2.1 (top). As expected, the direct approach scales as $O(n^6)$, solving the Sylvester equation (SYLV) as $O(n^3)$, while the conjugate gradient (CG) and fixed-point iteration (FP) approaches scale sub-cubically. Furthermore, note that the direct approach could not handle graphs of size greater than 2^7 even after two days of computation.

We also examined the impact of Lemma 1 on enhancing the runtime performance of the fixed-point iteration approach as originally proposed by [Kashima et al., 2003]. For this experiment, we again use graphs from SET-1 and computed the 10×10 kernel matrix, once using the original fixed-point iteration, and once using fixed-point iteration enhanced by Lemma 1. Results are illustrated in Figure 2.1 (bottom). As expected, our approach is often 10 times or more faster than the original fixed-point iteration, especially on larger graphs.

The second set of randomly generated graphs is called SET-2. Here, we fixed the size of

the graph at $2^{10} = 1024$, and randomly inserted edges until the average number of non-zero entries in the adjacency matrix is at least $x\%$, where $x = 10, 20, \ldots, 100$. For each x, we generate 10 such graphs and compute the 10×10 kernel matrix. We employed the direct approach, fixed-point iteration with vec-trick and without vec-trick, the conjugate gradient (CG) and the Sylvester equation approach (SYLV). Both the direct approach and the fixed-point iteration without vec-trick produced "out-of-memory" errors in all repetitions of this experiment. They cannot handle graphs of this size as they try to explicitly compute the weight matrix $W_\times$. In contrast, our three approaches to fast graph kernel computation enhanced by Lemma (1) can deal with graphs of this size. Results for these three methods are shown in Figure 2.2. As can be seen, the runtime of fixed-point iteration and conjugate gradient is filling-degree dependent, while that of the Sylvester equation is not. The reason might be that the former are able to exploit sparsity of the weight matrix $W_\times$, while the latter is not.

Real-World Datasets

We tested the practical feasibility of the presented techniques on four real-world datasets: two datasets of molecular compounds (MUTAG and PTC), and two datasets with hundreds of graphs describing protein tertiary structure (Protein and Enzyme). Graph kernels provide useful measures of similarity for all these graphs. We provide more details on these datasets and the associated learning task in the following.

Chemical Molecules Activity of chemical molecules can be predicted to some degree by comparison of their three-dimensional structure. We employed graph kernels to measure similarity between molecules from the MUTAG [Debnath et al., 1991] and the PTC dataset [Toivonen et al., 2003]. The average number of nodes per graph is 17.72 and 26.70, respectively, and the average number of edges is 38.76 and 52.06, respectively.

The MUTAG dataset [Debnath et al., 1991] consists of 230 mutagenic aromatic and heteroaromatic nitro compounds. Each of these molecules is known to possess a mutagenic effect in gram-negative bacterium *Salmonella typhimurium* or not. The classification task is to predict whether a given molecule exerts a mutagenic effect.

Each molecule is modeled as a graph, with the nodes representing atoms and the edges representing bonds between the atoms. Furthermore, we label each node with its atom type. A graph $G = (V, E)$ is derived for each molecule by representing each atom as a node in V. We assign the atom type as a non-unique label to each $v \in V$. An undirected edge is inserted if a bond exists between two atoms thus yielding a 3D structural representation of the molecule.

The Predictive Toxicology Challenge (PTC) dataset by [Toivonen et al., 2003] contains 417 chemical compounds which are tested for cancerogenicity in mice and rats. The classification task is to predict the cancerogenicity of compounds. As for MUTAG, each compound is represented as a graph, whose nodes are atoms and whose edges are bonds.

Large Protein Graph Dataset A fundamental first step in protein function prediction entails classifying proteins into enzymes and non-enzymes, then further assigning enzymes to one of the six top-level classes of the EC (Enzyme Commission) hierarchy. Towards this

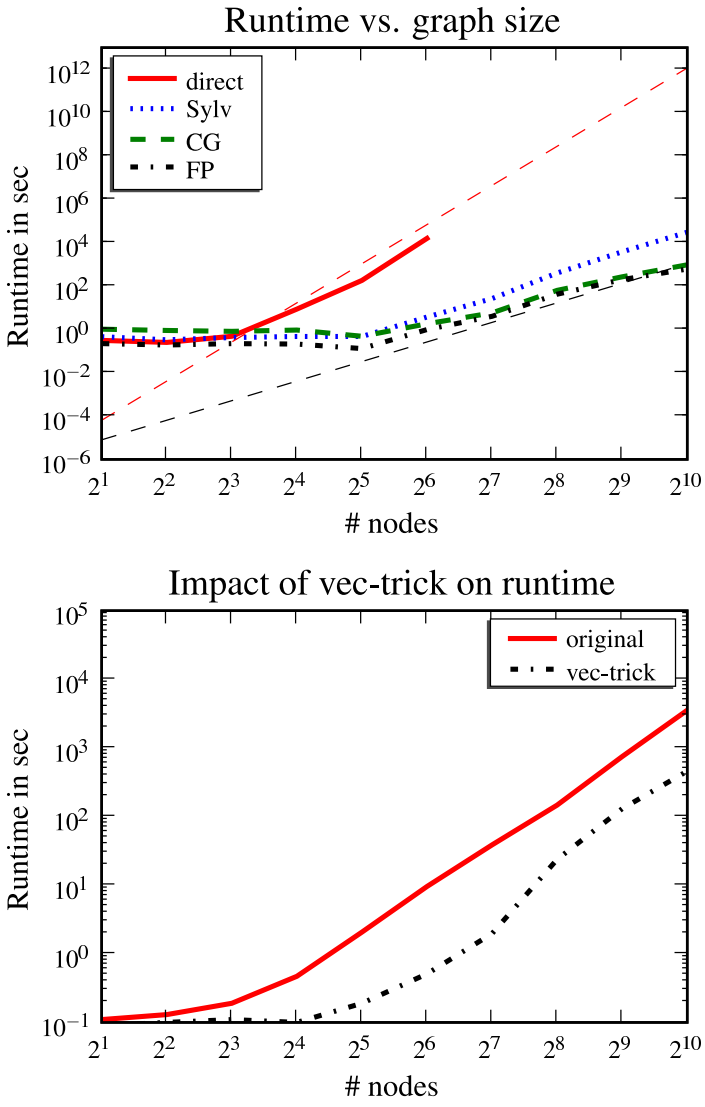

Figure 2.1: Time to compute a 10×10 kernel matrix on SET-1 plotted as a function of the size of graphs (# nodes). **Top**: We compare the Sylvester equation (Sylv), conjugate gradient (CG), and fixed-point iteration (FP) approaches to the direct approach (direct). The dashed thin red line indicates $O(n^6)$ scaling, while the dashed thin black line indicates $O(n^3)$ scaling. **Bottom**: We compare the runtime of the original fixed-point iteration (original) and that of the fixed-point iteration enhanced with Lemma 1 (vec-trick).

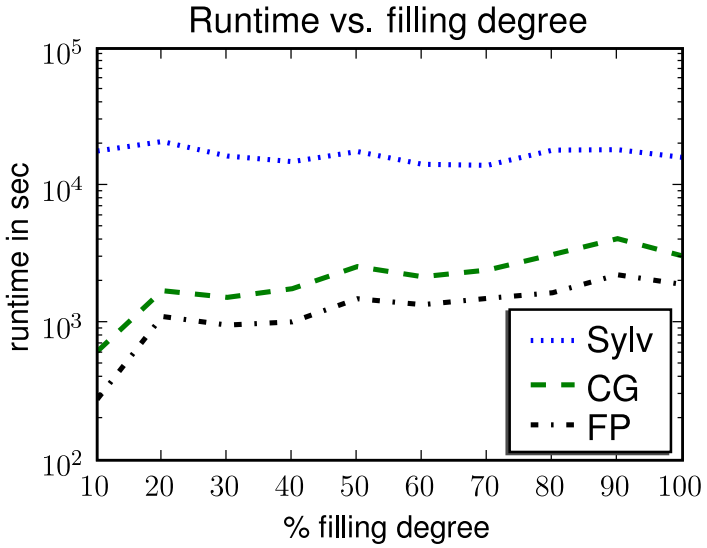

Figure 2.2: Time to compute a 10×10 kernel matrix on SET-2 with 1024 nodes vs. filling degree of adjacency matrix. We compare the Sylvester equation (Sylv), conjugate gradient (CG), and fixed-point iteration (FP) approaches.

end, [Borgwardt et al., 2005] modeled a dataset of 1128 proteins as graphs in which nodes represent secondary structure elements, and edges neighborhood within the 3D structure or along the amino acid chain. Comparing these graphs via a modified random walk kernel and classifying them via a Support Vector Machine (SVM) led to function prediction accuracies competitive with state-of-the-art approaches [Borgwardt et al., 2005].

We used [Borgwardt et al., 2005]'s data to test the efficacy of our methods on a large dataset. The average number of nodes and edges per graph in this data is 38.57 and 143.75, respectively. We used a single label on the edges, and the delta kernel to define similarity between edges.

Large Enzyme Graph Dataset We repeated the above experiment on an enzyme graph dataset also from [Borgwardt et al., 2005]. This dataset contains 600 graphs, containing 32.63 nodes and 124.27 edges on average. Graphs in this dataset represent enzymes from the BRENDA enzyme database [Schomburg et al., 2004a]. The biological challenge on this data is to correctly assign the enzymes to one of the EC top level classes.

Unlabeled Graphs

In a first series of experiments, we compared graph topology only on our 4 datasets, *i.e.,* without considering node and edge labels. We report the time taken to compute the full graph kernel matrix for various sizes (number of graphs) in Table 2.1.4 and show the results for computing a 100×100 sub-matrix in Figure 2.3 (left).

On unlabeled graphs, conjugate gradient and fixed-point iteration — sped up via our Lemma 1 — are consistently about two orders of magnitude faster than the conventional

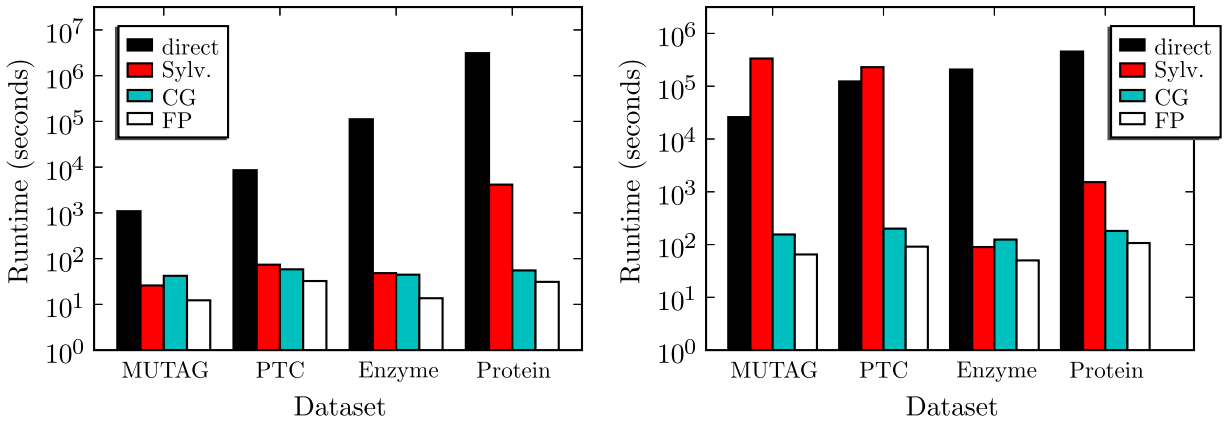

Figure 2.3: Time (in seconds on a log-scale) to compute 100×100 kernel matrix for unlabeled (**Left**) *resp.* labeled (**Right**) graphs from several datasets. Compare the conventional direct method (black) to our fast Sylvester equation, conjugate gradient (CG), and fixed-point iteration (FP) approaches.

dataset	MUTAG		PTC		Enzyme		Protein	
nodes/graph	17.7		26.7		32.6		38.6	
edges/node	2.2		1.9		3.8		3.7	
#graphs	100	230	100	417	100	600	100	1128
Direct	18'09"	104'31"	142'53"	41h*	31h*	46.5d*	36d*	12.5y*
Sylvester	25.9"	2'16"	73.8"	19'30"	48.3"	36'43"	69'15"	6.1d*
Conjugate	42.1"	4'04"	58.4"	19'27"	44.6"	34'58"	55.3"	97'13"
Fixed-Point	12.3"	1'09"	32.4"	5'59"	13.6"	15'23"	31.1"	40'58"

Table 2.1: Time to compute random walk kernel matrix for given number of unlabeled graphs from various datasets (∗: Extrapolated; run did not finish in time available.) .

direct method. The Sylvester approach is very competitive on smaller graphs (outperforming CG on MUTAG) but slows down with increasing number of nodes per graph; this is because we could not incorporate Lemma 1 into Matlab's black-box `dlyap` solver. Even so, the Sylvester approach still greatly outperforms the direct method.

Labeled Graphs

In a second series of experiments, we compared graphs with node and edge labels. On our two protein datasets we employed a linear kernel to measure similarity between edge labels representing distances (in Å) between secondary structure elements. On our two chemical datasets we used a delta kernel to compare edge labels reflecting types of bonds in molecules. We report results in Table 2.2 and Figure 2.3 (right).

On labeled graphs, our three methods outperform the direct approach by about a factor of 1000 when using the linear kernel. In the experiments with the delta kernel, conjugate

kernel	delta				linear			
dataset	MUTAG		PTC		Enzyme		Protein	
#graphs	100	230	100	417	100	600	100	1128
Direct	7.2h	1.6d*	1.4d*	25d*	2.4d*	86d*	5.3d*	18y*
Sylvester	3.9d*	21d*	2.7d*	46d*	89.8"	53'55"	25'24"	2.3d*
Conjugate	2'35"	13'46"	3'20"	53'31"	124.4"	71'28"	3'01"	4.1h
Fixed-Point	1'05"	6'09"	1'31"	26'52"	50.1"	35'24"	1'47"	1.9h

Table 2.2: Time to compute random walk kernel matrix for given number of labeled graphs from various datasets ($*$: Extrapolated; run did not finish in time available).

gradient and fixed-point iteration are still at least two orders of magnitude faster. Since we did not have access to a generalized Sylvester equation (2.13) solver, we had to use a Kronecker product approximation [Van Loan, 2000] which dramatically slowed down the Sylvester equation approach.

2.1.5 Summary

We have shown that computing random walk graph kernels is essentially equivalent to solving a large linear system. We have extended a well-known identity for Kronecker products which allows us to exploit the structure inherent in this problem. From this we have derived three efficient techniques to solve the linear system, employing either Sylvester equations, conjugate gradients, or fixed-point iterations. Experiments on synthetic and real-world datasets have shown our methods to be scalable and fast, in some instances outperforming the conventional approach by more than three orders of magnitude.

Even though the Sylvester equation method has a worst-case complexity of $O(n^3)$, the conjugate gradient and fixed-point methods tend to be faster on all our datasets. This is because computing matrix-vector products via Lemma 1 is quite efficient when the graphs are sparse, so that the feature matrices $\Phi(L)$ and $\Phi(L')$ contain only $O(n)$ non-ϵ entries. Matlab's black-box `dlyap` solver is unable to exploit this sparsity.

In this section, we have overcome one limitation of graph kernels based on random walks, the lack of efficiency. Two more weaknesses remain: Tottering and the impairing influence of the decaying factor λ. While the former has been described in discussed in the literature [Mahé et al., 2004], the second one has not received any attention yet.

This second weakness of random walks could be described as "halting": As walks allow for repetitions of nodes, the number of walks in a graph is infinitely large. We need a series of decaying factors $\lambda_0 > \lambda_1 > \lambda_2 > \dots$ that downweight longer walks, to make the series $\sum_{k=0}^{\infty} \lambda_k A_\times^k$ converge. For the geometric random walk kernels, λ has to be less than $\Delta_{max}(G_\times)^{-1}$. The effect of this decaying factor is that longer walks are completely neglected compared to shorter walks. Even walks of length 1 get $1/\lambda_1$ more weight than walks of length 2! The result is that the larger the maximum degree of a node in the product graph, the more walks of length 1 dominate the walk kernel value, and the contributions of longer walks tend towards zero. For graphs with large maximum degree, this means

that the random walk kernel converges to a graph kernel that compares length 1 walks only - which is an all-edges-comparison between two graphs, which is a naive measure of similarity. Note that the stronger the effect of halting, the weaker that of tottering, and vice versa. The reason is that if we are not walking, there can be no tottering, and if we are walking (or even tottering), there is no halting.

We will tackle both halting and tottering in the following sections by defining novel classes of graph kernels. Towards this end, we will explore the usage of paths instead of walks in graph kernels.

2.2 Graph Kernels based on Shortest Path Distances

Graph kernels using walks suffer from tottering and halting. Unlike walks, paths do not allow for repetitions of nodes, and therefore there can be no tottering on paths. For the same reason, paths cannot be of infinite length, and consequently we do not need a decaying factor that could cause halting.

Hence if we restricted ourselves to paths instead of walks, and defined a kernel on graphs that compares paths, we could avoid tottering and halting. Exactly this is the scope of this section.

2.2.1 Graph Kernels on All Paths

We start by defining a kernel that compares all paths in two graphs.

Definition 4 (All-Paths Kernel) *Given two graphs G and G'. Let $P(G)$ and $P(G')$ be the set of all paths in graph G and G', respectively. Let k_{path} be a positive definite kernel on two paths, defined as the product of kernels on edges and nodes along the paths. We then define an all-paths kernel $k_{all\,paths}$ as*

$$k_{all\,paths}(G, G') = \sum_{p \in P(G)} \sum_{p' \in P(G')} k_{path}(p, p'),$$

i.e., *we define the all-paths kernel as the sum over all kernels on pairs of paths from G and G'.*

In the following lemma, we prove that the all-paths kernel is a valid kernel.

Lemma 5 *The all-paths kernel is positive definite.*

Proof We define a relation $R(p, G \setminus \{p\}, G)$, where p is a path, $G \setminus \{p\}$ is a set of edges and nodes, and G is a graph. $R(p, G \setminus \{p\}, G) = 1$ iff $G \setminus \{p\}$ is the set of nodes and edges that remain when removing all edges and nodes in p from G.

$R^{-1}(G)$ is then the set of all possible decompositions of graph G via R into p and $G \setminus \{p\}$. R is finite, as there is only a finite number of paths in a graph, since their length is upper bounded by the number of edges. We define a kernel k_{path} on paths as a product of kernels on nodes and edges in these paths; this is a positive definite tensor product kernel [Schölkopf and Smola, 2002]. We also define a trivial set kernel $k_{one} = 1$ for all pairs of sets of nodes and edges.

We can then define an all-paths kernel as a positive definite R-convolution kernel [Haussler, 1999]:

$$\begin{aligned} k_{all\,paths}(G, G') &= \sum_{(p, G\setminus\{p\}) = R^{-1}(G)} \sum_{(p', G'\setminus\{p'\}) = R^{-1}(G')} k_{path}(p, p') * k_{one}(G \setminus \{p\}, G' \setminus \{p'\}) = \\ &= \sum_{p \in P(G)} \sum_{p' \in P(G')} k_{path}(p, p') \end{aligned} \tag{2.23}$$

with $P(G)$ and $P(G')$ as the set of all paths in G and G', respectively. ■

The computation of this kernel, however, is NP-hard, as we will prove in the following.

Lemma 6 *Computing the all-paths kernel is NP-hard.*

Proof We show this result by proving that finding all paths in a graph is NP-hard. If determining the set of all paths in a graph was not NP-hard, one could determine whether a graph has a Hamilton path or not by checking whether a path exists with length $n - 1$. This problem, however, is known to be NP-complete [Jungnickel, 1994]. Consequently, determining the set of all paths is NP-hard and therefore the computation of the all-paths kernel is NP-hard. ■

In [Gärtner et al., 2003] it is shown that computing kernels based on subgraphs is NP-hard. Although we are restricting ourselves to a small subset of all subgraphs, namely to paths, kernel computation is still NP-hard in our case.

2.2.2 Graphs Kernels on Shortest Paths

While determining all paths is NP-hard, finding special subsets of paths is not necessarily. Determining longest paths in a graph is again NP-hard, as it would allow to decide whether a graph contains a Hamilton path or not. Computing shortest paths in a graph, however, is a problem solvable in polynomial time. Prominent algorithms such as Dijkstra (for shortest paths from one source node) [Dijkstra, 1959] or Floyd-Warshall [Floyd, 1962, Warshall, 1962] (for all pairs of nodes) allow to determine shortest distances in $O(m + n * \log n)$ [Fredman and Tarjan, 1987] and $O(n^3)$ time, respectively, where n is the number of nodes and m the number of edges.

However, a potential problem lies in the fact that shortest paths are not unique. Obviously, there may be more than one shortest path between two nodes in a graph. Nevertheless, the shortest distance between those nodes is unique, as all shortest paths between two nodes must be of identical length. If one of these paths were shorter than the others, the other paths could not be "truly shortest" paths. For this reason, we employ the 3 unique characteristics of shortest paths to compute a kernel on them: the start node, the distance, and the end node of the shortest path.

2.2.3 Graphs Kernels on Shortest Path Distances

In algorithmic graph theory, the information about the endnodes and the length of shortest paths is commonly represented by a matrix called the *shortest path distance matrix*.

Definition 7 (Shortest Path Distance Matrix) *Let $G = (V, E)$ be a graph of size $|G| = n$. Let $d(v_i, v_j)$ be the length of the shortest path between v_i and v_j. The shortest path matrix D of G is then a $n \times n$ matrix defined as*

$$D_{ij} = \begin{cases} d(v_i, v_j) & \text{if } v_i \text{ and } v_j \text{ are connected,} \\ \infty & \text{otherwise} \end{cases} \tag{2.24}$$

For defining a graph kernel comparing all pairs of shortest paths from two graphs G and G', we have to compare all pairs of entries from D and D' that are finite (as only finite entries in S and S' indicate the existence of shortest paths). This can be achieved most easily if we think of D and D' as adjacency matrices defining corresponding graphs, the *shortest-path graphs*.

Definition 8 (Shortest-Path Graph) *Let $G = (V, E)$ be a graph, and let D be its shortest path distance matrix. Then the shortest-path graph S of G has the same set of nodes V as G, and its set of edges is defined via the adjacency matrix $A(S)$*

$$A(S)_{ij} = \begin{cases} 1 & \text{if } D(v_i, v_j) < \infty, \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

where $D(v_i, v_j)$ is the edge label of edge (v_i, v_j) in S.

Hence a shortest-paths graph S contains the same set of nodes as the original graph G. Unlike in the input graph, there exists an edge between all nodes in S which are connected by a walk in G. Every edge in S between nodes v_i and v_j is labeled by the shortest distance between these two nodes in G.

Based on this concept of a shortest-path graph, we are now in a position to present our graph kernel on shortest-path distances. The essential first step in its computation is to transform the original graphs into shortest-paths graphs. Any algorithm which solves the all-pairs-shortest-paths problem can be applied to determine all shortest distances in G, which then become edge labels in S. We propose to use Floyd's algorithm (see Algorithm 1). This algorithm has a runtime of $\mathrm{O}(n^3)$, is applicable to graphs with negative edge weights, but must not contain negative-weighted cycles. Furthermore, it is easy to implement. In the following, we will refer to the process of transforming a graph G into S via Floyd's algorithm as *Floyd-transformation*.

After Floyd-transformation of our input graphs, we can now define a shortest-path kernel.

Definition 9 (Shortest-path graph kernel) *Let G and G' be two graphs that are Floyd-transformed into S and S'. We can then define our shortest-path graph kernel on $S = (V, E)$ and $S' = (V', E')$ as*

$$k_{\textit{shortest paths}}(S, S') = \sum_{e \in E} \sum_{e' \in E'} k^1_{walk}(e, e'), \tag{2.26}$$

where k^1_{walk} is a positive definite kernel walks of length 1, i.e., *a kernel on edges.*

In the following, we will prove the validity of our shortest-path kernel.

Lemma 10 *The shortest-path graph kernel is positive definite.*

Algorithm 1 Pseudocode for Floyd-Warshall's algorithm [Floyd, 1962] for determining all-pairs shortest paths.

Input: Graph G with n nodes, adjacency matrix A, and edge weights w

```
for i := 1 to n
    for j := 1 to n
        if ((A[i,j] == 1) and i ≠ j
            D[i,j] = w[i,j];
        else
            if ( i == j)
                D[i,j] = 0;
            else
                D[i,j] = ∞;
            end
        end
    end
end
for k := 1 to n
    for i := 1 to n
        for j := 1 to n
            if (D[i,k] + D[k,j] < D[i,j])
                D[i,j] := D[i,k] + D[k,j];
            end
        end
    end
end
```

Output: Shortest path distance matrix D

Proof The shortest-path kernel is simply a walk kernel run on a Floyd-transformed graph considering walks of length 1 only. We follow the proofs in [Kashima et al., 2003] and [Borgwardt et al., 2005]. First, we choose a positive definite kernel on nodes and a positive definite kernel on edges. We then define a kernel on pairs of walks of length 1, $k^{(1)}_{walk}$, as the product of kernels on nodes and edges encountered along the walk. As a tensor product of node and edge kernels [Schölkopf and Smola, 2002], $k^{(1)}_{walk}$ is positive definite. We then zero-extend $k^{(1)}_{walk}$ to the whole set of pairs of walks, setting kernel values for all walks with length $\neq 1$ to zero. This zero-extension preserves positive definiteness [Haussler, 1999]. The positive definiteness of the shortest-path kernel follows directly from its definition as a convolution kernel, proven to be positive definite by [Haussler, 1999]. ■

Runtime Complexity The shortest-path kernel avoids tottering and halting, yet it remains an interesting question how it compares to the known random walk kernels in terms of runtime complexity.

The shortest-path kernel requires a Floyd-transformation which can be done in $O(n^3)$ when using the Floyd-Warshall algorithm. The number of edges in the transformed graph is n^2, if the original graph is connected. Pairwise comparison of all edges in both transformed graphs is then necessary to determine the kernel value. We have to consider $n^2 * n^2$ pairs of edges, resulting in a total runtime of $O(n^4)$.

Equal Length Shortest-Path Kernel

Label enrichment — in the spirit of [Mahé et al., 2004] — can also be applied to our Floyd-transformed graphs to speed up kernel computation. Both edges and nodes can be enriched by additional attributes. When performing the Floyd-Warshall algorithm, one is usually interested in the shortest distance between all nodes. However, if we store information about the shortest paths, *i.e.,* the number of edges or the average edge length in these shortest paths, then we can exploit this extra information to reduce computational cost. For instance, this can be achieved by setting kernels to zero for all pairs of shortest paths whose number of edges is not identical, *i.e.,*

$$k_{steps}(p, p') = \begin{cases} 1 & \text{if steps}(p) = \text{steps}(p'), \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

where p and p' are shortest paths and $steps(p)$ and $steps(p')$ are the number of edges in path p and p', respectively. If the steps kernel is zero for a pair of paths, we do not have to evaluate the node and edge kernel.

Note again that shortest paths need not be unique. Thus some extra criterion might be required to select one particular path out of a set of shortest paths. For instance, one could decide only to consider the shortest paths with minimum number of edges for computing k_{steps}.

k Shortest-Path Kernel

Even more valuable information for our kernel could be to know not just the shortest path between two nodes, but the k shortest paths. For each of the k shortest paths, one edge

could then be created in the Floyd-transformed graph. Note that in this case — unlike our general convention in this thesis — we would be dealing with graphs *with* multiple edges, *i.e.,* several edges between the same pair of nodes.

Finding k shortest walks and paths in a graph is a well-studied topic in graph theory and applied sciences [Yen, 1971, Lawler, 1972]. Many of the algorithms proposed for solving this problem, however, determine k shortest walks, not k shortest paths. Applying these algorithms would reintroduce the problem of tottering into our path-based kernel. It is therefore essential to chose an algorithm for finding "k loopless shortest paths" in a graph, as this is the term commonly used in the literature. Such algorithms have been proposed over 30 years ago [Yen, 1971, Lawler, 1972] and any of those can be run on our input graphs, as long as there are no cycles in our graphs with negative weights. The setback of this method is the increased runtime complexity for determining k shortest loopless paths. Yen's algorithm in [Yen, 1971] requires $O(kn(m + n \log n))$ time complexity for finding k shortest loopless paths between a pair of nodes, where n is the number of nodes and m is the number of edges. Consequently, theoretical complexity would be $O(kn^5)$ for determining k shortest loopless paths for all pairs of nodes in a fully connected graph and pairwise comparison of all k shortest paths in two graphs would be of complexity $O((kn^2) * (kn^2)) = O(k^2n^4)$. As a result, the preprocessing step has a higher runtime complexity than the kernel computation in this case.

A simple way to determine k shortest disjunct paths between two nodes, where no pair of paths shares any identical edge, is to iteratively apply Dijkstra's algorithm to the same graph and to remove all edges that belong to the currently shortest path. Still, this procedure would be of runtime complexity $O(n^2k(m + n \log n))$, which could become $O(kn^4)$ in a fully connected graph.

2.2.4 Link to Wiener Index

We have stressed before that graph kernels try to tackle the graph comparison problem that has been in the focus of several fields of research for decades, most prominently chemoinformatics. Is there any link between the approaches described in the chemoinformatics literature and the recent advances in graph kernels? For our novel shortest-path kernel, we were able to establish the first connection between molecular descriptors and graph kernels. As we will prove in the following, graph comparison via the Wiener Index [Wiener, 1947] is an instance of the shortest-path kernel. In other terms, the shortest-path kernel is a generalization of the Wiener Index.

Recall the definition of the Wiener Index as given in Section 1.3.3.

Definition 11 (Wiener Index) *Let $G = (V, E)$ be a graph. Then the Wiener Index $W(G)$ of G is defined as*

$$W(G) = \sum_{v_i \in G} \sum_{v_j \in G} d(v_i, v_j), \tag{2.28}$$

where $d(v_i, v_j)$ is defined as the length of the shortest path between nodes v_i and v_j from G.

Now assume that we are given two graphs G and G'. Then we compute the product of their Wiener Indices $W(G)$ and $W(G')$ as

$$\begin{aligned} W(G) * W(G') &= (\sum_{v_i \in G} \sum_{v_j \in G} d(v_i, v_j))(\sum_{v'_k \in G'} \sum_{v'_l \in G'} d(v'_k, v'_l)) = \\ &= \sum_{v_i \in G} \sum_{v_j \in G} \sum_{v_k \in G'} \sum_{v_l \in G'} d(v_i, v_j) d(v'_k, v'_l) = \\ &= \sum_{p \in P(G)} \sum_{p' \in P(G')} l(p) l(p') \\ &= \sum_{p \in P(G)} \sum_{p' \in P(G')} k_{linear}(l(p), l(p')) \end{aligned} \tag{2.29}$$

where $P(G)$ and $P(G')$ is the set of shortest paths in G and G', respectively, and $l(p)$ and $l(p')$ and are the lengths of shortest paths p and p', respectively.

Equation 2.29 shows that the product of two Wiener Indices is the same as a shortest-path kernel in which $k^{(1)}_{walk}$ is a linear kernel on shortest path distances. By picking another type of kernel for $k^{(1)}_{walk}$, the shortest-path kernel allows to design a similarity measure different from the Wiener Index. Hence shortest-path kernels provide a rich family of similarity measures on graphs that include similarity between Wiener Indices as one special instance.

2.2.5 Experiments

We performed two sets of experiments to experimentally evaluate our novel class of graph kernels. In the first experiment, we assessed the classification accuracy of several variants of the shortest-path kernel and of random walk kernels that suffer from tottering. In the second experiment, we assessed runtime and classification accuracy of the shortest-path kernel and the fast random walk kernel from Section 2.1 on three graph classification benchmarks.

Experiment 1: The Impact of Tottering on Classification Accuracy

To evaluate the practical performance of our shortest-path graph kernel, we chose a classification task from bioinformatics[Borgwardt et al., 2005]. 540 proteins, 90 per class, should be classified into 6 distinct functional classes in 10-fold cross-validation, solely based on protein structure information.

We obtained the protein structures from the Protein Data Bank [Berman et al., 2000] and their corresponding enzyme class labels from the BRENDA enzyme database [Schomburg et al., 2004b]. We randomly choose 90 proteins from each of the 6 enzyme EC hierarchy top level classes. We translated these protein structures into graph models in which the secondary structure elements of a protein represent the nodes.

Every node is connected to its three nearest neighbors in space. As a simplification, distances between secondary structure elements are calculated as distances between their spatial centers. Edges are labeled by the distance they represent in $\mathring{A}$. Nodes bear labels representing their type, namely helix, sheet or loop, and their length in amino acids.

On these graph models of proteins, we ran random walk kernels and shortest-path kernels. As we wanted to check the impact of tottering on the performance of the random walk kernels, we had to ensure by our choice of λ that walks of length > 1 and hence tottering would be captured by the random walk kernel, and not be blurred by halting. For this reason, we set $\lambda = 1$, but computed only walks up to a certain length k. We performed tests for k in the range from 4 to 7. This way, longer and shorter walks receive the same weight, and the random walk kernel cannot degenerate to an all-edges-comparison due to halting. However, it might suffer from tottering, which is the phenomenon we are interested in in this experiment.

We also employed our shortest-path kernel and the equal length shortest-path kernel on the same data. Furthermore, we ran a 2 shortest-paths kernel determining the 2 shortest disjunct paths between nodes via Dijkstra's algorithm.

All graph kernels use the same set of node and edge kernels. Types of two nodes v and v' are compared via a delta kernel, *i.e.,*

$$k_{type}(v, v') = \begin{cases} 1 & \text{if } type(v) = type(v'), \\ 0 & \text{otherwise} \end{cases}$$

The length attribute of nodes are compared via a Brownian bridge kernel, *i.e.,*

$$k_{length}(v, v') = max(0, c - |length(v) - length(v')|).$$

The same Brownian bridge kernel is applied to edges to measure their difference in length. c is set to 3 for nodes and to 2 for edges via cross-validation as in [Borgwardt et al., 2005].

After calculating all graph kernel matrices mentioned above, we predicted enzyme class membership in 10-fold cross-validation for 540 proteins. We performed "one-class vs. rest" Support Vector Machine classification and repeated this for all six EC top level classes. We report results as averages across all EC classes in Table 2.[2]

Results The shortest-path kernels outperform all walk kernels with an accuracy of at least 93.33%. The accuracy level of the worst shortest-path kernel on 540 proteins is statistically significantly higher than that of the best random walk kernel, which uses walks of up to length 4 (one-sided Welch t-test with 95% confidence level). As a result, considering shortest paths instead of walks increases classification accuracy significantly in our first experiment.

Among the walk kernels, classification is decreasing with the length of the walks under study. This is an indicator that the longer the walks are that we examine, the more numerous walks created by tottering get. With an increasing number of tottering walks, classification accuracy decreases. This is consistent with results reported by [Mahé et al., 2004].

Among the shortest-path kernels, the 2 shortest-path kernels perform slightly better than the equal length shortest-path kernel and the standard shortest-path kernel. However,

[2]Our graph kernel was implemented in MATLAB, release 13. We used a Linux Debian workstation with 3 GHz Intel CPUs for our experiments. We employed the SVM package SVLAB.

kernel type	accuracy
2 shortest paths	94.44 ± 0.80
e.l. shortest paths	93.52 ± 0.93
shortest paths	93.33 ± 1.02
walks up to length 4	89.63 ± 0.73
walks up to length 5	88.89 ± 0.63
walks up to length 6	88.15 ± 0.53
walks up to length 7	87.96 ± 0.56

Table 2.3: Walk kernel vs. shortest-path kernel. Prediction accuracy (± standard error) on 540 proteins from 6 EC classes in 10-fold cross-validation (st. dev. = standard deviation, e.l. = equal length).

the differences in accuracy between the different types of shortest-path kernels are not significant on our test set.

Experiment 2: Accuracy and Runtime on Benchmarks

In a second series of experiments, we compared our shortest-path kernel to the classic random walk kernel in terms of runtime and classification accuracy. We employed 3 benchmarks datasets: MUTAG, PTC, and Enzyme, as described in Section 2.1.4. Note that only subsets of MUTAG and PTC are commonly used for classification benchmarking, and we keep to this standard. For PTC, we used the cancerogenicity results from Male Rats (MR). We summarize statistics of the three datasets in Table 2.4.

We ran a geometric random walk kernel with $\lambda = 10^{-3}$, and an equal length shortest-path kernel on these 3 classification tasks. To evaluate their performance, we tested their prediction accuracy on independent evaluation sets which we obtained as follows. We split the datasets into 10 folds of identical size. We then split 9 of these folds again into 10 parts, trained an C-SVM (implemented by LIBSVM [Chang and Lin, 2001]) on 9 parts, and predicted on the 10th part. We repeated this training and prediction procedure for $C \in \{10^{-7}, 10^{-6}, \ldots, 10^{7}\}$, and determined the C reaching maximum prediction accuracy on the 10th part. We then trained an SVM with this best C on all 9 folds (= 10 parts), and predicted on the 10th fold, which acts as an independent evaluation set. We repeated the whole procedure 10 times such that each fold acts as independent evaluation set exactly once.

We repeated the whole experiment 10 times to avoid random effects resulting from random splitting of the dataset into 10 folds. We ran the complete series of experiments once ignoring node labels, once considering node labels. We report prediction accuracy for labeled and unlabeled graphs in Table 2.5 and associated runtimes in Table 2.6.

Results The shortest-path kernel comprehensively outperforms the random walk kernel in all our experiments on MUTAG and Enzyme. Differences in accuracy are large, ranging from roughly 5% on MUTAG with node labels to $\sim$ 15% on Enzyme with node labels.

dataset	instances	classes	# nodes	# edges	# distinct node labels
MUTAG	188	2 (125 vs. 63)	17.7	38.9	7
PTC	344	2 (192 vs. 152)	26.7	50.7	22
Enzyme	600	6 (100 each)	32.6	124.3	3

Table 2.4: Statistics on classification benchmark datasets.

graphs	unlabeled		labeled	
kernel	RW	SP	RW	SP
MUTAG	71.89 ± 0.66	81.28 ± 0.45	78.94 ± 0.65	83.94 ± 0.69
PTC	55.44 ± 0.15	55.44 ± 0.61	59.82 ± 0.74	59.09 ± 0.66
Enzyme	14.97 ± 0.28	27.53 ± 0.29	24.76 ± 0.38	40.19 ± 0.62

Table 2.5: Classification accuracy (± standard error) of random walk kernel (RW) and shortest-path kernel (SP) on real world datasets with and without node labels (averaged over 10 repetitions).

graphs	unlabeled		labeled	
kernel	RW	SP	RW	SP
MUTAG	42.3”	23.2”	2’24”	2’12”
PTC	2’39”	2’35”	13’7”	14’53”
Enzyme	10’45”	6’1”	46’55”	30’8”

Table 2.6: Runtime of random walk kernel (RW) and shortest-path kernel (SP) on real world datasets with and without node labels.

Note that we are using $1\,vs\,1$ classification for the balanced 6-class problem on Enzyme. A naive classifier that puts all enzymes into the same class would reach 16.67% accuracy on this dataset.

On PTC, both approaches give rather bad results that do not differ significantly. This is not very surprising, as PTC is known to be hard to separate[Toivonen et al., 2003].

In terms of runtime, the shortest-path kernel is faster than the random walk kernel in 5 out of 6 trials. On the largest dataset, Enzyme, the shortest-path kernel requires only 2/3 of the runtime of the random walk kernel. Only on PTC with labels, the random walk is 2 minutes faster than the shortest-path kernel.

2.2.6 Summary

We have defined graph kernels based on shortest path distances, whose runtime is polynomial in the size of the graphs and which are positive definite and retain expressivity while avoiding the phenomena of "tottering" and "halting". In experiments on benchmark datasets, their prediction accuracy always improved upon that of random walk kernels.

The shortest-path kernels prevent tottering. It is not possible that the same edge appears twice in the same shortest path, as this would violate the definition of a path. Subsequently, artifically high similarity scores caused by repeated visiting of the same cycle of nodes are prohibited in our graph kernel.

Our novel kernel also avoids "halting". We are looking at paths, which — in contrast to walks — cannot be of infinite length. Hence we do not have to employ a decaying factor that could cause halting. Even better, as we do not need a decaying factor, our graph kernel on shortest path distances is parameter-free.

The shortest-path kernel as described in this section is applicable to all graphs on which Floyd-Warshall can be performed. Floyd-Warshall requires that cycles with negative weight do not exist. If edge labels represent distances, which is the case in most molecular classification tasks, this condition generally holds.

As all and longest paths are NP-hard to compute, our graph kernel uses shortest paths. As shown in our experiments, shortest distances between nodes are a characteristic of graphs which is essential for graph classification in many applications such as chemoinformatics and bioinformatics. Problems could arise for the shortest-path kernel if paths other than shortest are most important in a particular application domain, as it discards all information represented by edges that are not part of a shortest path.

Concerning runtime, the shortest-path kernel is even faster than the sped-up random walk graph kernel in 5 out of 6 runs, although its runtime is in $O(n^4)$, while the random walk is in $O(n^3)$. The reason might be that the shortest-path kernel requires the computation of all shortest paths only once per graph. These shortest path lengths can then be reused for each kernel computation (note that in this fashion, we are explicitly computing the feature space). Furthermore, the $O(n^4)$ effort for comparing all pairs of shortest paths includes only one pairwise comparison of all these distances, while the $O(n^3)$ random walk requires a series of matrix multiplication that are all in $O(n^3)$. Hence lower constants in the runtime of the shortest-path kernel are likely to be the reason for its superior runtime performance.

Still, there is one main concern regarding the shortest-path kernel when applied to large graphs with hundreds and thousands of nodes: Large graphs are usually sparse, but the shortest-path kernel will turn these sparse graphs into dense shortest-path graphs that may lead to enormous memory and runtime problems. While the shortest-path kernel is efficient (on the small and medium-size graphs in our experiments), avoids tottering and halting, and is an expressive measure of graph similarity, it is probably not scalable to very large graphs. Thus the goal of the next section will be to define a graph kernel that scales up to very large graphs.

2.3 Graphlet Kernels for Large Graph Comparison

In Section 2.1 and Section 2.2, we have defined new approaches to graph kernel computation that led to a significant gain in efficiency. While the traditional random walk required a runtime of $O(n^6)$, our fast random walk kernels can be computed in $O(n^3)$ and our shortest-path kernels in $O(n^4)$. In our experiments, the computation of both kernels was up to 1,000 times faster than that of previous state-of-the-art approaches.

Unfortunately, even the fastest among these efficient graph kernels have a theoretical runtime of $O(n^3)$. While $O(n^3)$ might be a feasible and attractive runtime when dealing with the standard benchmark datasets from graph mining with less than 30 nodes on average, it is expensive when comparing large graphs with hundreds and thousands of nodes. Such large graphs may represent large groups of people, a detailed atom-level model of a protein, or an interaction network including all protein interactions in a species. It is desirable to develop graph kernels that can cope with such huge graphs.

Apart from this scalability issue, and even apart from the four criteria for graph kernel design defined earlier, graph kernels suffer from one common characteristic so far: The choice of subgraphs is completely ad-hoc. The motivation why to pick random walks, cyclic patterns or shortest paths originates mainly from runtime and expressiveness considerations. However, there is no theoretical justification, let alone a proof why certain types of substructures should reflect graph similarity better than others.

In this chapter, we tackle both problems aforementioned. Motivated by the matrix reconstruction theorem and the graph reconstruction conjecture, we argue that comparing subgraphs with 4 nodes provides an expressive measure of similarity on graphs. A graph kernel based on comparing all size-4 subgraphs, however requires a runtime of $O(n^8)$ when naively implemented. We therefore first design several algorithmic tricks to speed up computation, and second we develop a sampling scheme that allows us to approximate the distribution of size-4 subgraphs within a specified level of confidence and precision while sampling a constant number of these size 4-subgraphs. Our novel graph kernel outperforms existing ones, and is able to deal with graphs that were too large for graph kernels before.

2.3.1 Graph Reconstruction

We start our exposition by summarizing the graph reconstruction conjecture and the matrix reconstruction theorem, and conclude this section by building a bridge between these reconstruction ideas and graph kernels.

Reconstruction of Graphs

Graph reconstruction is a classic open problem in graph theory [Kelly, 1957, Hemminger, 1969]: Let $G = (V, E)$ be a undirected graph of size n. For each $v \in V$, let G_v denote a node-deleted subgraph of G, *i.e.,* the graph obtained by deleting node v and all the edges incident on it from G. Can G be reconstructed, up to an isomorphism, from its set of node-deleted subgraphs $\{G_v\}_{v \in V}$? Intuitively, one asks: Given a graph G on n nodes, is G determined uniquely up to an isomorphism by its subgraphs of size $n-1$? Put differently, are there two non-isomorphic graphs with identical $n-1$ sized subgraphs?

Kelly [Kelly, 1957] proved the following theorem: Let $G = (V, E)$ and $G' = (V', E')$ be

trees and $g : V \to V'$ be an isomorphism function such that G_v is isomorphic to $G'_{g(v)}$ for all $v \in V$, then G is isomorphic to G'. He conjectured that the following theorem is true for arbitrary graphs:

Theorem 12 (Graph Reconstruction Conjecture) *Let G and G' be graphs of size greater than 2 and $g : V \to V'$ be an isomorphism function such that G_v is isomorphic to $G'_{g(v)}$ for all $v \in V$. Then G is isomorphic to G'.*

Kelly [Kelly, 1957] verified his conjecture by enumeration of all possible graphs for $2 < n \leq 6$, which was later extended to $2 < n \leq 11$ by [McKay, 1997]. Special classes of graphs such as regular graphs, and disconnected graphs have also been shown to be reconstructible [Kelly, 1957]. The general case, however, remains a conjecture. It is widely believed though, that if a counterexample to the graph reconstruction problem exists, then it will be of size $n \gg 11$ [McKay, 1997].

Reconstruction of Matrices

While graph reconstruction remains a conjecture for general graphs, reconstruction of matrices has been resolved [Manvel and Stockmeyer, 1971]. We need some terminology to make this result clearer. Let M be any $n \times n$ matrix. We call the submatrix obtained by deletion of its k-th row and k-th column the k-th principal minor, and denote it as M_k. The following theorem due to [Manvel and Stockmeyer, 1971] asserts that the principal minors determine the matrix:

Theorem 13 *Any $n \times n$ matrix M with $n \geq 5$ can be reconstructed from its list of principal minors $\{M_1, \ldots, M_n\}$.*

The adjacency matrix of a graph is not invariant to reordering of the nodes, but, if the graph is node ordered then its adjacency matrix is unique. For such graphs, the following corollary is particularly relevant:

Corollary 14 *Any graph $G = (V, E)$ of size $n \geq 5$ whose nodes are ordered as $v_1, \ldots, v_n$ can be reconstructed from its set of maximal subgraphs $\{G_{v_1}, \ldots, G_{v_n}\}$, if their nodes are ordered in the same order as those of G.*

The condition that the nodes of all node-deleted subgraphs of G have to be ordered in the same way as those of G implies that the nodes of G must be sorted according to a *global canonical vertex ordering*. We will explain what we mean by a *global canonical vertex ordering* in the following. For this purpose, we first have to clarify two concepts: *complete graph invariant* (see Section 1.3.3) and *canonical form*.

A function f of a graph is called a *complete graph invariant* if $G \simeq G'$ is equivalent to $f(G) = f(G')$. If, in addition, $f(G)$ is a graph isomorphic to G, then f is called a *canonical form* for graphs [Koebler and Verbitsky, 2006, Gurevich, 2001]. [Gurevich, 2001] showed that graphs have a polynomial-time computable canonical form if, and only if, they have a polynomial-time computable complete invariant.

Recall that a vertex ordering π maps every node of a graph to a unique number in $\{1, \ldots, n\}$. If π is invariant to isomorphism, then it defines a complete graph invariant. With some abuse of terminology, in the sequel, we will refer to such a vertex ordering as *canonical*. This is justified because π can indeed be used to define a canonical form for graphs. Every vertex ordering also induces a vertex ordering on the subgraphs. This is because every subset of an ordered set is also ordered. We denote this induced ordering by π_G. Note that even if π is canonical, it does not guarantee that the induced vertex orderings π_G are canonical. If every induced vertex ordering of π is also canonical, then π is said to be globally canonical.

Unfortunately, computing a global canonical vertex ordering is a NP-hard problem because given a solution to this problem, one can solve subgraph isomorphism – a NP-complete problem – in polynomial time [Garey and Johnson, 1979]. Nevertheless, for many graphs of practical importance, it is easy to compute a global canonical vertex ordering, especially in the field of databases. If we are dealing with a graph whose nodes are distinct objects from the same database, then we can order the nodes in this graph according to their keys in the database. Ordering via database keys obviously results in a global canonical vertex ordering.

Graph Similarity via Graph Reconstruction

Why is the graph reconstruction conjecture interesting for graph kernels? Because it deals with a question that is implicitly asked when designing graph kernels: Which substructures of a graph determine a graph up to isomorphism? If the graph reconstruction conjecture were true, this question could be answered: A graph is determined uniquely up to isomorphism by its size-$(n-1)$ subgraphs. Although the conjecture has not been proven in general, it has been shown for certain classes of graphs, in particular for graphs with global canonical vertex ordering. We will exploit these results to define novel graph kernels in the following.

2.3.2 Graph Kernels based on Graph Reconstruction

In this section, we define graph kernels based on the idea of decomposing a graph of size n recursively into its subgraphs of size k. We will refer to these subgraphs as k minors, as formalized in the following definition.

Definition 15 (k Minors) *Let M be a $n \times n$ matrix. The set of all size-k sub-matrices of M obtained by deleting $n-k$ rows and corresponding columns of M is called the k minors of M. Analogously, given a graph G of size n, the set of all size-k graphs obtained by deleting $n-k$ nodes from G is called the k minors of G.*

Definition 16 (Principal Minors) *Let M be a $n \times n$ matrix. The set of all $(n-1)$ minors of M is called the set of principal minors. Analogously, given a graph G of size n, the set of all $n-1$ minors is called the principal minors of G.*

In the sequel we will be concerned with 4 minors. Therefore, we study some of their properties now. For undirected graphs, the entries in the upper triangular submatrix

of the adjacency matrix completely determine the graph (Recall that we are considering graphs without multiple edges and without self-loops). In the case of graphs of size 4, this submatrix contains 6 entries, each of which could either be 0 or 1 depending on the presence or absence of the corresponding edge. Therefore, there are $2^6 = 64$ different types of graphs of size 4. We refer to these 64 graphs as a *graphlets* [Przulj, 2007], and denote them as $\mathcal{G}_4 = \{graphlet(1), \ldots, graphlet(64)\}$. Corresponding to these 64 graphlets one can also compute a matrix $P \in \{0,1\}^{64\times 64}$ whose entries are defined as:

$$P_{ij} = \begin{cases} 1 & \text{if } graphlet(i) \simeq graphlet(j), \\ 0 & \text{otherwise.} \end{cases} \tag{2.30}$$

P precomputes the isomorphism relationship between graphlets.

Recursive Graph Comparison

Graph reconstruction tries to establish isomorphism between graphs by checking their principal minors for isomorphism. Along the same lines, we define a graph kernel to measure similarity between graphs by comparing their principal minors. Motivated by the matrix reconstruction theorem, we recursively iterate this procedure down to subgraphs of size 4, resulting in a graph kernel based on graphlets.

Definition 17 (Graphlet Kernel) *Given two graphs G and G' of size $n \geq 4$, let $\mathcal{M}$ and $\mathcal{M}'$ denote the set of principal minors of G and G' respectively. The recursive graph kernel, k_n, based on principal minors is defined as*

$$k_n(G, G') = \begin{cases} \frac{1}{n^2} \sum_{S\in\mathcal{M}, S'\in\mathcal{M}'} k_{n-1}(S, S') & \text{if } n > 4, \\ \delta(G \simeq G') & \text{if } n = 4 \end{cases} \tag{2.31}$$

where $\delta(G \simeq G')$ is 1 if G and G' are isomorphic, and 0 otherwise. Now the graphlet kernel is defined as

$$k(G, G') := k_n(G, G'). \tag{2.32}$$

Lemma 18 *The graphlet kernel is positive semi-definite.*

Proof The proof is by induction. Clearly, $k_4(G, G') := \delta(G \simeq G')$ is a valid positive semi-definite kernel [Schölkopf and Smola, 2002]. For any $n \geq j > 4$ let $k_{j-1}(S, S')$ be a valid kernel. Since the class of positive semi-definite kernels is closed under addition and multiplication by a positive constant, it follows that $k_j(G, G')$ is a valid positive semi-definite kernel. ■

It is easy to see that the above kernel simply compares the 4 minors in both G and G', and hence can be computed non-recursively. This intuition is formalized below.

Lemma 19 *Let $\mathcal{M}_4$ and $\mathcal{M}'_4$ denote the set of 4 minors of G and G' respectively. The graphlet kernel can be computed without recursion as*

$$k(G,G') = k_n(G,G') = \sum_{S\in\mathcal{M}_4}\sum_{S'\in\mathcal{M}'_4} \delta(S \simeq S'). \tag{2.33}$$

Equivalently,

$$k(G,G') = k_n(G,G') = \sum_{S,S'\in\mathcal{G}_4} \#(S \sqsubseteq G)\,\#(S' \sqsubseteq G')\,\delta(S \simeq S'), \tag{2.34}$$

where $\#(S \sqsubseteq G)$ is the number of occurrences of S in G, and $\#(S' \sqsubseteq G')$ the number of occurrences of S' in G'.

Proof Clearly (2.33) is true for graphs of size 4. For $n > 4$ it follows by unrolling the recursion and noting that there are n minors of size $n-1$, $n-1$ minors of size $n-2$ and so on.

To see (2.34) note that $\mathcal{M}_4$ and $\mathcal{M}'_4$ are multisets of elements from the graphlet set $\mathcal{G}_4$, with each graphlet S or S' occurring $\#(S \sqsubseteq G)$ or $\#(S' \sqsubseteq G')$ times respectively.

■

Since there are $\binom{n}{4}$, *i.e.,* $O(n^4)$ 4 minors in a graph, the following corollary is immediate.

Corollary 20 *Let c denote the time required to perform an isomorphism check on two graphs of size 4. While a naive, recursive implementation of the recursive graph kernel requires $O(n^{2n}c)$ runtime, the runtime can be reduced to $O(n^8c)$ via the non-recursive formula,* (2.33).

While we reduce runtime from exponential to polynomial in the size of the graphs by Corollary 20, the n^8 factor still represents a major problem in real-world applications. The expensive step is the pairwise comparison of the 4 minors of both graphs. Note however that if one needs to compute the pairwise kernel on a database of m graphs, then the $O(n^4)$ work per graph can be amortized by employing the following scheme: Precompute all the 4 minors of the graph, check for isomorphisms to any of the 64 graphlets, and store their frequency of occurrence. Overall, this requires $O(mn^4c)$ effort. Modulo isomorphism, there are only 11 distinct graphs of size 4. Therefore, computing each individual entry of the kernel matrix requires $O(1)$ effort. The total cost of computing the $m \times m$ kernel matrix therefore reduces from $O(m^2n^8c)$ to $O(mn^4c + m^2)$. Typically, $m \leq n^4$ and therefore the overall time complexity is dominated by the mn^4c term. In the following, we will first describe an efficient scheme to perform the isomorphism checks efficiently, and then we will show how to avoid the n^4 term by an efficient sampling scheme that drastically speeds up the preprocessing step.

2.3.3 Efficiently Checking Graph Isomorphism

In this section, we describe various *tricks of trade* that can be used to speed up isomorphism checking on small graphs. Since we are dealing with small sized subgraphs we can determine isomorphism classes among them, and explicitly precompute isomorphism relationships.

Unlabeled Graphs

Given a graph G we define a 64 dimensional vector f_G whose i-th component corresponds to the frequency of occurrence of $graphlet(i)$ in graph G. By exploiting the matrix P and frequency vector f_G we can rewrite (2.34) as follows.

Definition 21 (Kernel from Frequency Vectors) *Given two graphs G and G', and their frequency vectors f_G and $f_{G'}$, we can compute the graphlet kernel as*

$$k(G, G') = f_G^\top P f_{G'}. \tag{2.35}$$

In short, this means that we have to precompute the permutation matrix $\mathcal{P}$ in a one-time-effort. To compute a graph kernel matrix on a set of graphs, we have to determine the frequency vector of each graph by enumerating its graphlets. To obtain the graph kernel value for two graphs, we multiply their frequency vectors to the permutation matrix.

Accounting for Differences in Graph Size In Equation (2.35), a weakness of R-convolution kernels becomes apparent. R-convolution kernels compare all decompositions of two objects pairwise. As the number of decompositions is usually directly proportional to the size of the objects, graph kernel values increase for larger objects. In our setting, the graph kernel value directly depends on the absolute graphlet frequencies in two graphs. As a consequence, the larger the two graphs or one of the two graphs, the larger their kernel value if they are similar. To compensate for this problem, we may work with relative frequencies instead of absolute frequencies of 4 minors.

Definition 22 (Graphlet Distribution Vector) *Given a graph G. We define the relative frequency vector or graphlet distribution vector $D(G)$ as*

$$D(G)_i = \frac{\#occurrences\ of\ graphlet(i)\ in\ G}{\#all\ graphlets\ in\ G} \tag{2.36}$$

where $D(G)_i$ is the i-th component of $D(G)$, $1 \leq i \leq 64$, and $graphlet(i)$ is the i-th graphlet class.

This leads directly to a kernel on relative graphlet frequencies, which reflect the distribution of graphlets across the 64 graphlet classes.

Definition 23 (Kernel on Graphlet Distributions) *Given two graphs G and G', and their graphlet distribution vectors $D(G)$ and $D(G')$, we can compute the graphlet kernel k as*

$$k(G, G') = D(G)^\top P D(G'). \tag{2.37}$$

Labeled Graphs

If we are working with graphs with node labels, graph kernel computation becomes more difficult, but can still be performed efficiently. The additional complexity derives from the fact that two 4 minors need not only be isomorphic now, but their corresponding nodes have to bear identical node labels as well. In terms of graph theory, we are now dealing with isomorphisms that do preserve both topology and node labels of the graph.

As a consequence, we cannot use one single permutation matrix to get graph kernel values as in Equation (2.35). In principle, we need one permutation matrix for each set of node labels that a 4 minor could bear. Even if we assume that node labels are discrete and elements of finite alphabet Σ of size $|\Sigma|$, we have to deal with $|\Sigma|^4$ different sets of node labels. This looks like a hopeless endeavor.

Still, we can reduce the computational burden if we exploit special 'structure' within the node labels of 4 minors. Our approach is to categorize sets of node labels into different equivalence classes. Two node sets belong to the same equivalence class if they contain the same number of distinct node labels and these node labels occur with the same frequency. We will formalize this intuition in the following lemma.

Lemma 24 (Equivalence Classes of 4 Minor Labels) *Let $\gamma = (v_1, v_2, v_3, v_4)$ be a 4 minor with nodes v_1, v_2, v_3 and v_4. Let $\mathcal{L}(\gamma) = (\mathcal{L}(v_1), \mathcal{L}(v_2), \mathcal{L}(v_3), \mathcal{L}(v_4))$ denote its set of node labels, sorted according to some arbitrary order, such that nodes with identical label appear in consecutive blocks. If we now count identical node labels and their frequencies in $\mathcal{L}(\gamma)$, then $\mathcal{L}(\gamma)$ belongs to one of the following 8 equivalence classes (EC_k with $k \in \{1, \ldots, 8\}$) of 4 minor labels:*

1. *class (1-1-1-1), where $\mathcal{L}(v_1), \mathcal{L}(v_2), \mathcal{L}(v_3), \mathcal{L}(v_4)$ are pairwise non-identical,*
2. *class (1-1-2), where $\mathcal{L}(v_1), \mathcal{L}(v_2), \mathcal{L}(v_3)$ are pairwise non-identical, but $\mathcal{L}(v_3) = \mathcal{L}(v_4)$,*
3. *class (1-2-1), where $\mathcal{L}(v_1), \mathcal{L}(v_2), \mathcal{L}(v_4)$ are pairwise non-identical, but $\mathcal{L}(v_2) = \mathcal{L}(v_3)$,*
4. *class (1-3), where $\mathcal{L}(v_1), \mathcal{L}(v_2)$ are pairwise non-identical, but $\mathcal{L}(v_2) = \mathcal{L}(v_3) = \mathcal{L}(v_4)$,*
5. *class (2-1-1), where $\mathcal{L}(v_1), \mathcal{L}(v_3), \mathcal{L}(v_4)$ are pairwise non-identical, but $\mathcal{L}(v_1) = \mathcal{L}(v_2)$,*
6. *class (2-2), where $\mathcal{L}(v_1), \mathcal{L}(v_3)$ are pairwise non-identical, but $\mathcal{L}(v_1) = \mathcal{L}(v_2)$ and $\mathcal{L}(v_3) = \mathcal{L}(v_4)$,*
7. *class (3-1), where $\mathcal{L}(v_1), \mathcal{L}(v_4)$ are pairwise non-identical, but $\mathcal{L}(v_1) = \mathcal{L}(v_2) = \mathcal{L}(v_3)$,*
8. *class (4), where $\mathcal{L}(v_1), \mathcal{L}(v_2), \mathcal{L}(v_3), \mathcal{L}(v_4)$ are all pairwise identical.*

For each of these equivalence classes EC_k with $k \in \{1, \ldots, 8\}$, we precompute one permutation matrix $\mathcal{P}_k$. To check isomorphism of two 4 minors γ and γ', we sort them according to the same order, determine their graphlet classes $graphlet(\gamma) = i$ and $graphlet(\gamma') = j$, check their sets of node labels for identity, determine their equivalence class EC_k, and check isomorphism by looking up $\mathcal{P}_{k(i,j)}$. All these steps can be performed efficiently by looking up precomputed hash tables or precomputed permutation matrices.

2.3.4 Sampling from Graphs

In order to compute our kernel exactly one needs to exhaustively enumerate all graphlets of size 4 in the input graphs. Suppose a given graph has n nodes, then there are $\binom{n}{4}$ or equivalently $O(n^4)$ graphlets. If the graphs are small then this is feasible, but on large graphs (with n of the order of hundreds or thousands or more) runtime will degenerate. In this case one needs to resort to sampling. The idea is very simple: Randomly select sets of 4 nodes from the graph and observe the empirical distribution of graphlets induced by these nodes. The hope is that if sufficient number of random samples are drawn, then the empirical distribution is *sufficiently close* to the actual distribution of graphlets in the graph. The number of samples needed to achieve a given confidence with a small probability of error is called the sample complexity.

This approach is not new; the problem of sampling subgraphs from graphs has been widely studied in the bio-informatics literature [Przulj, 2007, Przulj et al., 2006, Kashtan et al., 2004, Wernicke, 2005]. Unfortunately, the algorithms proposed there are rather ad-hoc and do not provide any bounds on sample complexity. Recently, [Weissman et al., 2003] proved distribution dependent bounds for the L_1 deviation between the true and the empirical distributions. We adapt their results and derive sample complexity bounds which are much stronger than any previously known result for this problem.

Sample Complexity Bound

Let $\mathcal{A} = \{1, 2, \ldots, a\}$ denote a finite set of elements. For two probability distributions P and Q on $\mathcal{A}$, the L_1 distance between P and Q is defined as

$$||P - Q||_1 := \sum_{i=1}^{a} |P(i) - Q(i)|. \tag{2.38}$$

Given a multiset $X := \{X_j\}_{j=1}^m$ of independent identically distributed (i.i.d.) random variables X_j drawn from some distribution D (denoted as $X_j \sim D$), the empirical estimate of D is defined as

$$\hat{D}^m(i) = \frac{1}{m} \sum_{j=1}^{m} \delta(X_j = i), \tag{2.39}$$

where $\delta(\cdot)$ denotes the indicator function; $\delta(X_j = i) = 1$ if $X_j = i$ and zero otherwise. For $p \in [0, 1/2)$, define

$$\psi(p) = \frac{1}{1 - 2p} \log \frac{1 - p}{p}, \tag{2.40}$$

and set $\psi(1/2) = 2$. Note that $\psi(p) \geq 2$ for all valid p. Furthermore, for a probability distribution D on $\mathcal{A}$ define:

$$D(S) := \sum_{i \in S} D(i) \text{ for all } S \subseteq \mathcal{A} \text{ and} \tag{2.41}$$

$$\pi_D := \max_{S \subseteq \mathcal{A}} \min\{D(S), 1 - D(S)\}. \tag{2.42}$$

Theorem 25 *[Weissman et al., 2003] Let D be a probability distribution on the finite set $\mathcal{A} = \{1, \ldots, a\}$. Let $X := \{X_j\}_{j=1}^m$, with $X_j \sim D$. Then for all $\epsilon > 0$*

$$P\left\{||D - \hat{D}^m||_1 \geq \epsilon\right\} \leq (2^a - 2)e^{-m\psi(\pi_D)\epsilon^2/4}. \tag{2.43}$$

The following corollary is straightforward:

Corollary 26 *Let D, $\mathcal{A}$, and X as above. For a given $\epsilon > 0$ and $\delta > 0$, at least*

$$m \geq \frac{4\left(\log 2 \cdot a + \log\left(\frac{1}{\delta}\right)\right)}{\psi(\pi_D)\epsilon^2} \tag{2.44}$$

samples are required to ensure that $P\left\{||D - \hat{D}^m||_1 \geq \epsilon\right\} \leq \delta$.

By observing that $\psi(\pi_D) \geq 2$, one can eliminate the distribution dependent term in the above corollary to obtain:

Corollary 27 *Let D, $\mathcal{A}$, and X as above. For a given $\epsilon > 0$ and $\delta > 0$, at least*

$$m \geq \frac{2\left(\log 2 \cdot a + \log\left(\frac{1}{\delta}\right)\right)}{\epsilon^2} \tag{2.45}$$

samples are required to ensure that $P\left\{||D - \hat{D}^m||_1 \geq \epsilon\right\} \leq \delta$.

Implications of the Bound

In order to apply Corollary 27 to our problem we set $\mathcal{A}$ to be the set of all graphlets of size 4 and assume that they are distributed according to a unknown distribution D. Furthermore, let m be the number of graphlets randomly sampled from the graph. Then (2.45) gives the number of samples needed to ensure that the empirical distribution $\hat{D}^m$ is at most ϵ distance away from the true distribution D with confidence $1 - \delta$.

The bound has a number of desirable properties. First of all, notice that (2.44) is independent of n, the size of the graph. What this means in practice is that our sampling algorithm is highly scalable and works even for very large graphs. Secondly, notice that our sample complexity bound only has an additive dependence on a, the size of the set over which the distribution is defined.

When dealing with unlabeled graphs, there are a total of 64 possible graphlets of size 4. But, modulo isomorphism, there are only 11 distinct graphlets [Przulj, 2007]. Finally, if we set $\epsilon = 0.05$ and $\delta = 0.05$, then our bound implies that we only need to sample $8,497$ graphlets from a graph. If we decrease ϵ to 0.01 and δ to 0.01, then this number increases to $244,596$.

When considering labeled graphs, the total number of possible graphlets increases, as graphlets are now defined both by their topology and their node labels. If labels are chosen from an alphabet Σ with size $|\Sigma|$, then $a > \binom{|\Sigma|}{4} * 11$, as we can clearly label each graphlet

with 4 (distinct) node labels from Σ. As a consequence, a is in $O(|\Sigma|^4)$ for labeled graphs. Hence our sample size is still independent from graph size n, but growing with the size of the node label alphabet Σ to the power of 4. Recalling that isomorphism checking on labeled graphs is also much more involved than on unlabeled graphs, it becomes apparent that both our speed up techniques are faster and easier to implement on unlabeled than on labeled graphs, as labeled graphs require larger sample sizes and more involved isomorphism checks. Speeding up the efficiency of our sampling scheme for labeled graphs is a topic of ongoing research.

2.3.5 Experiments

In this section, we evaluate the performance of our novel graph kernel based on enumerating or sampling graphlets. We are interested in how it compares to our fast random walk graph kernel from Section 2.1, and the shortest-path kernel from Section 2.2 in terms of runtime and classification accuracy.

For this purpose we evaluated the graphlet kernel on the same three datasets for which we had previously established results for random walk and shortest-path kernels: MUTAG, PTC and Enzyme. We represent the objects in these datasets as unlabeled graphs. Furthermore, we employ graph kernels on a protein function prediction task from [Dobson and Doig, 2003a], which we describe next.

Dobson and Doig 2003 (D & D)

This dataset comprises 1178 proteins, including 691 enzymes and 587 non-enzymes. The classification task is to predict for each of the proteins whether it belongs to the class of enzymes or not. We turn this problem into a graph classification problem by describing the structure of each protein by a graph.

Nodes represent amino acids, and two nodes are linked by an edge if they are less than 6 Å apart. The average number of nodes per protein structure graph model is 284.4 and the average edge number is 1921.6.

Applying graph kernels to such detailed models is particularly challenging. In 2005, we concluded that state-of-the-art graph kernels could not tackle graph classification problems with graphs of this size [Borgwardt et al., 2005]. It is of particular interest to us if our novel graph kernels and sampling scheme allows to extend the applicability of graph kernels to these detailed models.

Experimental Settings

We compute the graphlet kernel (GK) for different sample sizes corresponding to a prespecified level of confidence and precision. As there are only 11 distinct subgraphs of size 4 modulo isomorphism, a equals 11.

- **GK 1986** sampling $m = 1986$ graphlets, which corresponds to a precision level of $\epsilon = 0.1$ and a confidence parameter of $\delta = 0.1$,
- **GK 2125** sampling $m = 2125$ graphlets, which corresponds to a precision level of $\epsilon = 0.1$ and a confidence parameter of $\delta = 0.05$,

kernel	MUTAG	PTC	Enzyme	D & D
RW	71.89 ± 0.66	55.44 ± 0.15	14.97 ± 0.28	> 1 day
SP	81.28 ± 0.45	55.44 ± 0.61	27.53 ± 0.29	> 1 day
GK 1986	80.42 ± 0.23	59.09 ± 0.11	27.24 ± 0.17	74.51 ± 0.13
GK 2125	80.69 ± 0.31	58.86 ± 0.21	27.62 ± 0.42	74.55 ± 0.15
GK 7942	81.57 ± 0.41	59.06 ± 0.13	28.13 ± 0.24	74.67 ± 0.08
GK 8497	81.89 ± 0.23	59.38 ± 0.22	27.32 ± 0.17	74.46 ± 0.07
GK all	82.17 ± 0.58	59.65 ± 0.31	28.95 ± 0.50	> 1 day

Table 2.7: Classification accuracy on graph benchmark datasets (RW = random walk kernel, SP = shortest-path kernel, GK m = graphlet kernel sampling m graphlets, '> 1 day' means computation did not finish within 24 hours).

kernel	MUTAG	PTC	Enzyme	D & D
RW	42.3"	2' 39"	10' 45"	> 1 day
SP	23.2"	2' 35"	5' 1"	> 1 day
GK 1986	1' 39"	3' 2"	4' 20"	11' 35"
GK 2125	1' 46"	3' 16"	4' 36"	12' 21"
GK 7942	6' 33"	12' 3"	16' 35"	42' 45"
GK 8497	6' 57"	12' 49"	17' 38"	45' 36"
GK all	3' 37"	2h 56' 26"	4h 21' 29"	> 1 day

Table 2.8: Runtime for kernel matrix computation on graph benchmark datasets (RW = random walk kernel, SP = shortest-path kernel, GK m = graphlet kernel sampling m graphlets, '> 1 day' means computation did not finish within 24 hours).

- **GK 7942** sampling $m = 7942$ graphlets, which corresponds to a precision level of $\epsilon = 0.05$ and a confidence parameter of $\delta = 0.1$,
- **GK 8497** sampling $m = 8497$ graphlets, which corresponds to a precision level of $\epsilon = 0.05$ and a confidence parameter of $\delta = 0.05$,
- **GK all**, meaning we enumerated all graphlets exhaustively.

As before, we use an independent evaluation scheme: Splitting the dataset into 10 folds, optimizing parameters of an SVM on 9 folds, then predicting on the 10th fold which acts as an independent evaluation set, and repeating the whole procedure until each fold has been the independent evaluation set exactly once. We report classification accuracies for our graphlet kernels in Table 2.7 and runtimes for kernel matrix computation in Table 2.8. For comparison, we also list the results and runtimes for the random walk kernel and the shortest-path kernel from Section 2.2, obtained on the same datasets and using the same experimental protocol.

Results

On MUTAG, PTC and Enzyme, the graphlet kernel enumerating all graphlets (GK all) reached the highest accuracy, even outperforming the shortest-path kernel. The GK kernels based on sampling instead of enumeration yield similarly good results. The classification accuracies they reach are only slightly worse than those of the exhaustive enumeration, competitive with (MUTAG, Enzyme) or even better (PTC) than the shortest-path kernel, and comprehensively better than those of the random walk kernel.

In terms of runtime, graphlet enumeration and graphlet sampling are expensive and slower than the shortest-path and the random walk kernel on small datasets such as MUTAG and PTC. As graph size increases (Enzyme), graphlet sampling gets more competitive. Sampling 1986 and 2125 graphlets on Enzyme is already faster than computing shortest-path and random walk kernel. On D & D, none of the latter kernels finishes computation within 24 hours, nor does the exhaustive enumeration of all graphlets. The 4 graphlet kernels based on sampling manage to compute a kernel matrix on D & D in less than an hour. GK 1986 even completes this task in 11 minutes and 35 seconds.

As an interesting aside, note that the classification accuracy that the graphlet kernels reach on D & D is highly competitive with those known from the literature [Dobson and Doig, 2003a, Borgwardt et al., 2005] which use heavily annotated vector or graph models of proteins. In contrast, our graphlet kernels here operate on simple unlabeled graph models of proteins.

2.3.6 Summary

In this section, motivated by the matrix reconstruction theorem and the graph reconstruction conjecture, we have defined a graph kernel counting common size-4 subgraphs, so-called *graphlets*, in two graphs. Kernel computation involves 2 expensive steps: enumeration of all graphlets in each graph and pairwise isomorphism checks on these graphlets. The latter step can be performed efficiently by exploiting the limited size of graphlets and by precomputing isomorphism groups among them. We speed up the former step by an efficient sampling scheme that allows us to estimate the distribution over graphlet isomorphism classes by sampling a constant number of graphlets. Both these methods allow us to apply our novel kernel to graph sizes that no other graph kernel could handle so far.

In our experimental evaluation on unlabeled graphs, the novel graphlet kernel reached excellent results, constantly reaching high levels of classification accuracy, and getting more competitive in runtime performance as graph size increases. Future work will look into ways of reducing the sample size required for labeled graphs, and on speeding up isomorphism checks on labeled graphs.

To conclude, in this chapter, we have sped up the random walk graph kernel to $O(n^3)$, defined a novel kernel on shortest paths that is efficient, avoids tottering and halting and is an expressive measure of graph similarity. In the last section, we have defined a graph kernel based on sampling small subgraphs from the input graphs that is also efficient, avoids tottering and halting, an expressive measure of graph similarity, and in addition, scales up to very large graphs hitherto not handled by graph kernels.

Chapter 3

Two-Sample Tests on Graphs

While we have enhanced the efficiency of graph kernels so far, we have not tackled another problem: Graph kernel values per se are a rather unintuitive measure of similarity on graphs. When comparing two graphs or when comparing two sets of graphs, the (average) graph kernel value will be large, if the graphs or the sets of graphs are very similar, and small otherwise. But how to judge what is small and what is large in terms of graph kernel values?

Ideally, we would employ a statistical test to decide whether graph similarity is significant. Little attention has been paid to the question if the similarity of graphs is *statistically significant*. Even the question itself is problematic: What does it mean that the similarity of two graphs is statistically significant?

For set of graphs, this question can be answered more easily than for pairs of graphs. Given two sets of graphs, we can regard each of these sets as a sample from an underlying distribution of graphs. We then have to define a statistical test to decide whether the underlying distributions of two samples are identical; this is known as the two-sample-problem, and an associated test is called a two-sample test. Unfortunately, no two-sample test for graphs is known from the literature.

In this chapter, we define the first two-sample test that is applicable to sets of graphs, as it is based on a test statistic whose empirical estimate can be expressed in terms of kernels. In Section 3.1, we present this test statistic, the Maximum Mean Discrepancy (MMD) and its associated two-sample tests, and evaluate its performance on classic feature vector data. In Section 3.2.1 we explain how the two-sample tests based on MMD can be applied to sets of graphs, and evaluate it on two datasets of protein structures represented as graphs. We then show that MMD can even be applied to define a statistical test of graph similarity on pairs of graph instances in Section 3.2.2, and employ it to measure similarity between protein-protein-interaction networks of different species.

A note to the reader: Our presented method uses several concepts and results from functional analysis and statistics. If you do not feel familiar with these domains, we recommend to read the primers on these topics in Appendix A.1 and Appendix A.2 of this thesis, before continuing with this chapter. To make the presentation easier to follow, we have also moved three long proofs from this chapter to a separate Appendix B.

3.1 Maximum Mean Discrepancy

In this section, we address the problem of comparing samples from two probability distributions, by proposing a statistical test of the hypothesis that these distributions are different (this is called the two-sample or homogeneity problem). This test has application in a variety of areas. In bioinformatics, it is of interest to compare microarray data from different tissue types, either to determine whether two subtypes of cancer may be treated as statistically indistinguishable from a diagnosis perspective, or to detect differences in healthy and cancerous tissue. In database attribute matching, it is desirable to merge databases containing multiple fields, where it is not known in advance which fields correspond: the fields are matched by maximizing the similarity in the distributions of their entries.

We propose to test whether distributions p and q are different on the basis of samples drawn from each of them, by finding a smooth function which is large on the points drawn from p, and small (as negative as possible) on the points from q. We use as our test statistic the difference between the mean function values on the two samples; when this is large, the samples are likely from different distributions. We call this statistic the Maximum Mean Discrepancy (MMD).

Clearly the quality of MMD as a statistic depends heavily on the class $\mathcal{F}$ of smooth functions that define it. On one hand, $\mathcal{F}$ must be "rich enough" so that the population MMD vanishes if and only if $p = q$. On the other hand, for the test to be consistent, $\mathcal{F}$ needs to be "restrictive" enough for the empirical estimate of MMD to converge quickly to its expectation as the sample size increases. We shall use the unit balls in universal Reproducing Kernel Hilbert Spaces [Steinwart, 2002] as our function class, since these will be shown to satisfy both of the foregoing properties. On a more practical note, MMD is cheap to compute: given m_1 points sampled from p and m_2 from q, the cost is $O(m_1+m_2)^2$ time.

We define two non-parametric statistical tests based on MMD. The first, which uses distribution-independent uniform convergence bounds, provides finite sample guarantees of test performance, at the expense of being conservative in detecting differences between p and q. The second test is based on the asymptotic distribution of MMD, and is in practice more sensitive to differences in distribution at small sample sizes.

We begin our presentation in Section 3.1.1 with a formal definition of the MMD, and a proof that the population MMD is zero if and only if $p = q$ when $\mathcal{F}$ is the unit ball of a universal RKHS. We also give an overview of hypothesis testing as it applies to the two-sample-problem, and review previous approaches in Section 3.1.2. In Section 3.1.3, we provide a bound on the deviation between the population and empirical MMD, as a function of the Rademacher averages of $\mathcal{F}$ with respect to p and q. This leads to a first hypothesis test. We take a different approach in Section 3.1.4, where we use the asymptotic distribution of an unbiased estimate of the squared MMD as the basis for a second test. Finally, in Section 3.1.5, we demonstrate the performance of our method on problems from neuroscience, bioinformatics, and attribute matching using the Hungarian marriage approach. Our approach performs well on high-dimensional data with low sample size. In

addition, we will show in Section 3.2 that we are able to successfully apply our test to graph data, for which no alternative tests exist.

3.1.1 The Two-Sample-Problem

We present the two-sample-problem in Section 3.1.1, and introduce the MMD test statistic, proving that it is zero only when the two distributions being tested are identical. In Section 3.1.2, we give a brief background to statistical hypothesis testing, and describe prior approaches to the two-sample-problem in the multivariate domain.

Maximum Mean Discrepancy

Our goal is to formulate a statistical test that answers the following question:

Problem 1 *Let p and q be distributions defined on a domain $\mathcal{X}$. Given observations $X := \{x_1, \ldots, x_{m_1}\}$ and $Y := \{y_1, \ldots, y_{m_2}\}$, drawn independently and identically distributed (i.i.d.) from p and q respectively, does $p \neq q$?*

To start with, we wish to determine a criterion that, in the population setting, takes on a unique and distinctive value only when $p = q$. It will be defined based on Lemma 9.3.2 of [Dudley, 2002].

Lemma 28 *Let $(\mathcal{X}, d)$ be a separable metric space, and let p, q be two Borel probability measures defined on $\mathcal{X}$. Then $p = q$ if and only if $\mathbf{E}_p[f(x)] = \mathbf{E}_q[f(x)]$ for all $f \in C(\mathcal{X})$, where $C(\mathcal{X})$ is the space of continuous bounded functions on $\mathcal{X}$.*

Although $C(\mathcal{X})$ in principle allows us to identify $p = q$ uniquely, it is not practical to work with such a rich function class in the finite sample setting. We thus define a more general class of statistic, for as yet unspecified function classes $\mathcal{F}$, to measure the disparity between p and q, as proposed by [Fortet and Mourier, 1953].

Definition 29 (Maximum Mean Discrepancy) *Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ and let p, q, X, Y be defined as above. Then we define the Maximum Mean Discrepancy (MMD) and its empirical estimate as*

$$\mathrm{MMD}(\mathcal{F}, p, q) := \sup_{f \in \mathcal{F}} \left(\mathbf{E}_{x \sim p}[f(x)] - \mathbf{E}_{y \sim q}[f(y)] \right), \tag{3.1}$$

$$\mathrm{MMD}(\mathcal{F}, X, Y) := \sup_{f \in \mathcal{F}} \left(\frac{1}{m_1} \sum_{i=1}^{m_1} f(x_i) - \frac{1}{m_2} \sum_{i=1}^{m_2} f(y_i) \right). \tag{3.2}$$

We must now identify a function class that is rich enough to uniquely identify whether $p = q$, yet restrictive enough to provide useful finite sample estimates (the latter property will be established in subsequent sections).

We have a large degree of freedom in selecting $\mathcal{F}$. The function class determines our prior knowledge as to where we expect p and q to differ most. Also, $\mathcal{F}$ will be determined by the problems we wish to solve given the datasets X and Y: for instance, if we are only

interested in linear estimates of the data afterwards, it will suffice to ensure that X and Y agree within the class of bounded linear functions.

A large class of functions used in Machine Learning can be described by Banach spaces $\mathcal{B}$. Consequently we will select $\mathcal{F}$ to be the unit ball in $\mathcal{B}$, *i.e.,* $\mathcal{F} = \{f | \|f\|_{\mathcal{B}} \leq 1 \text{ and } f \in \mathcal{B}\}$. If $\mathcal{B}$ is dense in $C(\mathcal{X})$ we have the following theorem (proved in Appendix B):

Theorem 30 *Denote by $\mathcal{B}$ a Banach space which is dense in $C(\mathcal{X})$ and let $\mathcal{F}$ be a unit ball in a $\mathcal{B}$. Then* $\mathrm{MMD}(\mathcal{F}, p, q) = 0$ *if and only if* $p = q$.

We next express the MMD in a more easily computable form. For this purpose denote by $\mathcal{B}^*$ the dual space of $\mathcal{B}$, and let $\phi(x)$ be the evaluation functionals in $\mathcal{B}$. They are defined by $f(x) =: \langle f, \phi(x) \rangle$. This allows us to find a more concise expression for $\mathrm{MMD}(\mathcal{F}, p, q)$ and $\mathrm{MMD}(\mathcal{F}, X, Y)$.

Theorem 31 *Let $\mathcal{B}$ be a Banach space of functions on $\mathcal{X}$ and denote by $\phi(x) \in \mathcal{B}^*$ the evaluation functionals on $\mathcal{B}$. Let $\mathcal{F}$ be the unit ball in $\mathcal{B}$. Moreover, let*

$$\mu[p] := \mathbf{E}_{x \sim p}[\phi(x)] \quad \text{and} \quad \mu[X] := \frac{1}{|X|} \sum_{x \in X} \phi(x). \tag{3.3}$$

Then $\mathrm{MMD}(\mathcal{F}, p, q) = \|\mu[p] - \mu[q]\|$ *and* $\mathrm{MMD}(\mathcal{F}, X, Y) = \|\mu[X] - \mu[Y]\|$.

Proof [Theorem 31] By construction we can express $\mathbf{E}_{x \sim p}[f(x)] = \mathbf{E}_{x \sim p}[\langle f, \phi(x) \rangle] = \langle \mu[p], f \rangle$. Hence

$$\mathrm{MMD}(\mathcal{F}, p, q) = \sup_{\|f\| \leq 1} \langle \mu[p] - \mu[q], f \rangle = \|\mu[p] - \mu[q]\|_{\mathcal{B}^*}. \tag{3.4}$$

The first equality follows from the linearity of the expectation, the second one follows from the definition of the dual norm. An analogous derivation proves the second claim regarding $\mathrm{MMD}(\mathcal{F}, X, Y)$. ■

A sufficient condition for the existence of $\mu[p]$ is that $\|\phi(x)\| \leq C$ for some $C \in \mathbb{R}$ and for all $x \in \mathcal{X}$. In other words, the evaluation operator needs to be bounded.

The next lemma will prove a result that is at the core of our novel approach. It establishes that under certain conditions, we can establish a one-to-one correspondence between a distribution p and its expectation in feature space $\mu[p]$.

Lemma 32 *Denote by $\mathcal{P}(\mathcal{X})$ the set of distributions on $\mathcal{X}$. The operator $\mu[p]$ is linear in p. The set $\mathcal{M} := \{\mu[p]$ where $p \in \mathcal{P}(\mathcal{X})\}$, often referred to as the marginal polytope, is convex. If $\mathcal{B}$ is dense in $C(\mathcal{X})$ then $\mu : \mathcal{P}(\mathcal{X}) \to \mathcal{M}$ is bijective.*

Proof [Lemma 32] The expectation is a linear operation in p, hence $\mu[p]$ is linear. Since $\mathcal{P}(\mathcal{X})$ is convex, also its image $\mathcal{M}$ under μ must be convex. Finally, by construction $\mu : \mathcal{P}(\mathcal{X}) \to \mathcal{M}$ is surjective.

What remains to show is injectivity for $\mathcal{B}$ dense in $C(\mathcal{X})$: by Theorem 30 $\mathrm{MMD}(\mathcal{F}, p, q)$ only vanishes for $p = q$. By Theorem 31 we can express MMD in terms of the means $\mu[p]$ and $\mu[q]$. Hence $\mu[p] = \mu[q]$ immediately implies $p = q$. ■

This means that $\mathrm{MMD}(\mathcal{F}, p, q)$ defines a *metric* on the space of probability distributions, induced by the Banach space $\mathcal{B}$. As we shall see, it is often easier to compute distances in this metric, as it will not require density estimation as an intermediate step.

Reproducing Kernel Hilbert Spaces

If $\mathcal{B}$ is a Reproducing Kernel Hilbert Space $\mathcal{H}$ many of the aforementioned quantities can be computed very efficiently. We will henceforth use $\mathcal{F}$ only to denote unit balls in $\mathcal{H}$. Moreover, we will refer to $\mathcal{H}$ as universal, whenever $\mathcal{H}$, defined on a compact metric space $\mathcal{X}$ and with associated kernel $k : \mathcal{X}^2 \to \mathbb{R}$, is dense in $C(\mathcal{X})$ with respect to the L_∞ norm. It is shown in [Steinwart, 2002] that Gaussian and Laplace kernels are universal. As a specialization of Theorem 30 we immediately have the following result:

Theorem 33 *Let $\mathcal{F}$ be a unit ball in a universal RKHS $\mathcal{H}$, defined on the compact metric space $\mathcal{X}$, with associated kernel $k(\cdot, \cdot)$. Then $\mathrm{MMD}(\mathcal{F}, p, q) = 0$ if and only if $p = q$.*

We obtain an improved condition for the existence of $\mu[p]$ via $\|\mu[p]\|^2 = \mathbf{E}_{x,x' \sim p}\left[k(x, x')\right] < \infty$. Here x, x' are drawn independently from p. Exploiting Theorem 31 we are able to obtain a more accessible formulation for $\mathrm{MMD}(\mathcal{F}, p, q)$ and $\mathrm{MMD}(\mathcal{F}, X, Y)$.

Theorem 34 *Let $\mathcal{F}$ be a unit ball in a RKHS $\mathcal{H}$ with kernel k then $\mathrm{MMD}(\mathcal{F}, p, q)$ and $\mathrm{MMD}(\mathcal{F}, X, Y)$ can be computed as follows:*

$$\mathrm{MMD}(\mathcal{F}, p, q) = \left[\mathbf{E}_{x,x' \sim p}\left[k(x, x')\right] - 2\mathbf{E}_{x \sim p, x' \sim q}\left[k(x, x')\right] + \mathbf{E}_{x,x' \sim q}\left[k(x, x')\right]\right]^{\frac{1}{2}} \tag{3.5}$$

$$\mathrm{MMD}(\mathcal{F}, X, Y) = \left[\frac{1}{m_1^2}\sum_{i,j=1}^{m_1} k(x_i, x_j) - \frac{2}{m_1 m_2}\sum_{i,j=1}^{m_1, m_2} k(x_i, y_j) + \frac{1}{m_2^2}\sum_{i,j=1}^{m_2} k(y_i, y_j)\right]^{\frac{1}{2}} \tag{3.6}$$

Proof [Theorem 34] We only prove (3.5), as the proof of (3.6) is completely analogous. By virtue of Theorem 31 we have

$$\begin{aligned}\mathrm{MMD}(\mathcal{F}, p, q)^2 &= \|\mu[p] - \mu[q]\|_{\mathcal{H}}^2 \\ &= \left\langle \mathbf{E}_{x \sim p}\left[\phi(x)\right] - \mathbf{E}_{x \sim q}\left[\phi(x)\right], \mathbf{E}_{x' \sim p}\left[\phi(x')\right] - \mathbf{E}_{x' \sim q}\left[\phi(x')\right]\right\rangle\end{aligned} \tag{3.7}$$

In an RKHS we have $\langle \phi(x), \phi(x') \rangle = k(x, x')$. Plugging this into (3.7) and pulling the expectations out of the inner product proves the claim. ■

Eq. (3.6) provides us with a test statistic for $p = q$. We shall see in Section 3.1.3 that this estimate is biased, although it is straightforward to upper bound the bias (we give an unbiased estimate, and an associated test, in Section 3.1.4). Intuitively we expect $\mathrm{MMD}(\mathcal{F}, X, Y)$ to be small if $p = q$, and the quantity to be large if the distributions are

far apart. Also, since $\mathrm{MMD}(\mathcal{F}, X, Y) \geq 0$ we intuitively expect the discrepancy measure to be positive, even when the underlying distributions $p = q$ agree. Note that it costs $O((m_1 + m_2)^2)$ time to compute the statistic.

3.1.2 Background Material

Statistical Hypothesis Testing

We start by describing the framework of *statistical hypothesis testing* as it applies in the present context, following [Casella and Berger, 2002, Chapter 8].

Definition 35 (Two-Sample Test) *Given i.i.d. samples $X \sim p$ of size m_1 and $Y \sim q$ of size m_2, the statistical test, $\mathcal{D}(X, Y) : \mathcal{X}^{m_1} \times \mathcal{X}^{m_2} \mapsto \{0, 1\}$ is used to distinguish between the null hypothesis $\mathcal{H}_0 : p = q$ and the alternative hypothesis $\mathcal{H}_1 : p \neq q$.*

This is achieved by comparing the test statistic, in our case $\mathrm{MMD}(\mathcal{F}, X, Y)$, with a particular threshold: if the threshold is exceeded, then the test rejects the null hypothesis (bearing in mind that a zero population MMD indicates $p = q$). The acceptance region of the test is thus defined as any real number below the threshold. Since the test is based on finite samples, it is possible that an incorrect answer will be returned, a so-called Type I error or Type II error.

Definition 36 (Type I and Type II errors) *Let X, Y, p, q and $\mathcal{D}$ be defined as in Definition 35. We define the Type I error of $\mathcal{D}$ as the probability of $\mathcal{D}$ rejecting the null hypothesis $p = q$ based on the observed samples X and Y, despite the null hypothesis having generated the data. Conversely, the Type II error of $\mathcal{D}$ is the probability of accepting the null hypothesis $p = q$ despite the underlying distributions being different.*

The level α of a test is an upper bound on the Type I error: this is a design parameter of the test, and is used to set the threshold to which we compare the test statistic (finding the test threshold for a given α is the topic of Sections 3.1.3 and 3.1.4). A consistent test achieves a level α, and a Type II error of zero, in the large sample limit. We will see that both of the tests proposed in this section are consistent.

Two-Sample Tests on Multivariate Data

We next give a brief overview of previous approaches to the two sample problem for multivariate data.

Multivariate t-test Various empirical methods have been proposed to determine whether two distributions are different. The first test we consider, and the simplest, is a multivariate generalization of the t-test [Hotelling, 1951], which assumes both distributions are multivariate Gaussian with unknown, identical covariance structure. This test is not model-free in the sense of MMD (and the tests described below) — indeed, it is easy to construct examples in which it fails completely.

Friedman and Rafsky A generalisation of the Wald-Wolfowitz runs test to the multivariate domain was proposed and analysed in [Friedman and Rafsky, 1979, Henze and Penrose, 1999], which involves counting the number of edges in the minimum spanning tree over the aggregated data that connect points in X to points in Y. The resulting test relies on the asymptotic normality of the test statistic, and this quantity is not distribution-free under the null hypothesis for finite samples (it depends on p and q). The computational cost of this method using Kruskal's algorithm is $O((m_1 + m_2)^2 \log(m_1 + m_2))$, although more modern methods improve on the $\log(m_1 + m_2)$ term (see [Chazelle, 2000]; note also that [Friedman and Rafsky, 1979] state that calculating the matrix of distances, which costs $O((m_1 + m_2)^2)$, dominates their computing time; this may not be the case for large sample sizes, however). Two possible generalisations of the Kolmogorov-Smirnov test to the multivariate case were studied in [Bickel, 1969, Friedman and Rafsky, 1979]. The approach of Friedman and Rafsky in this case again requires a minimal spanning tree, and has a similar cost to their multivariate runs test.

Rosenbaum A more recent multivariate test was introduced by [Rosenbaum, 2005]. This entails computing the minimum distance non-bipartite matching over the aggregate data, and using the number of pairs containing a sample from both X and Y as a test statistic. The resulting statistic is distribution-free under the null hypothesis at finite sample sizes, in which respect it is superior to the Friedman-Rafsky test; on the other hand, it costs $O((m_1 + m_2)^3)$ to compute.

Hall and Tajvidi Another distribution-free test was proposed by [Hall and Tajvidi, 2002]: for each point from p, it requires computing the closest points in the aggregated data, and counting how many of these are from q (the procedure is repeated for each point from q with respect to points from p). As we shall see in our experimental comparisons, the test statistic is costly to compute; [Hall and Tajvidi, 2002] consider only tens of points in their experiments.

Biau and Gyorfi Yet another approach is to use some distance (e.g. L_1 or L_2) between Parzen window estimates of the densities as a test statistic [Anderson et al., 1994, Biau and Gyorfi, 2005], based on the asymptotic distribution of this distance given $p = q$. When the L_2 norm is used, the test statistic is related to those we present here, although it is arrived at from a different perspective. The L_1 approach of [Biau and Gyorfi, 2005] requires the space to be partitioned into a grid of bins, which becomes difficult or impossible for high-dimensional problems. Hence we use this test only for low-dimensional problems in our experiments.

3.1.3 A Test based on Uniform Convergence Bounds

In this section, we establish two properties of the MMD. First, we show that regardless of whether or not $p = q$, the empirical MMD converges in probability at rate $1/\sqrt{m_1 + m_2}$ to its population value. This establishes the consistency of statistical tests based on MMD. Second, we give probabilistic bounds for large deviations of the empirical MMD in the case $p = q$. These bounds lead directly to a threshold for our first hypothesis test.

We begin our discussion of the convergence of $\mathrm{MMD}(\mathcal{F}, X, Y)$ to $\mathrm{MMD}(\mathcal{F}, p, q)$. The

following theorem is proved in Appendix B.

Theorem 37 *Let p, q, X, Y be defined as in Problem 1, and assume $|k(x, y)| \leq K$. Then*

$$\Pr\left\{|\mathrm{MMD}(\mathcal{F}, X, Y) - \mathrm{MMD}(\mathcal{F}, p, q)| > 2\left((K/m_1)^{\frac{1}{2}} + (K/m_2)^{\frac{1}{2}}\right) + \epsilon\right\} \leq 2\exp\left(\frac{-\epsilon^2 m_1 m_2}{2K(m_1+m_2)}\right).$$

Our next goal is to refine this result in a way that allows us to define a test threshold under the null hypothesis $p = q$. Under this circumstance, the constants in the exponent are slightly improved.

Theorem 38 *Under the conditions of Theorem 37 where additionally $p = q$ and $m = m_1 = m_2$,*

$$\mathrm{MMD}(\mathcal{F}, X, Y) > \underbrace{m^{-\frac{1}{2}}\sqrt{2\mathbf{E}_p\left[k(x,x) - k(x,x')\right]}}_{B_1(\mathcal{F},p)} + \epsilon > \underbrace{2(K/m)^{1/2}}_{B_2(\mathcal{F},p)} + \epsilon,$$

both with probability less than $\exp\left(-\frac{\epsilon^2 m}{4K}\right)$ *(see Appendix B for the proof).*

In this theorem, we illustrate two possible bounds $B_1(\mathcal{F}, p)$ and $B_2(\mathcal{F}, p)$ on the bias in the empirical estimate (3.6). The first inequality is interesting inasmuch as it provides a link between the bias bound $B_1(\mathcal{F}, p)$ and kernel size (for instance, if we were to use a Gaussian kernel with large σ, then $k(x, x)$ and $k(x, x')$ would likely be close, and the bias small). In the context of testing, however, we would need to provide an additional bound to show convergence of an empirical estimate of $B_1(\mathcal{F}, p)$ to its population equivalent.

Lemma 39 *A hypothesis test of level α for the null hypothesis $p = q$ (which is equivalent to $\mathrm{MMD}(\mathcal{F}, p, q) = 0$) has the acceptance region*

$$\mathrm{MMD}(\mathcal{F}, X, Y) < 2\sqrt{K/m}\left(1 + \sqrt{\log \alpha^{-1}}\right). \tag{3.8}$$

We emphasize that Theorem 37 guarantees the consistency of the test, and that the Type II error probability decreases to zero at rate $1/\sqrt{m}$ (assuming $m = m_1 = m_2$). To put this convergence rate in perspective, consider a test of whether two normal distributions have equal means, given they have unknown but equal variance [Casella and Berger, 2002, Exercise 8.41]. In this case, the test statistic has a Student-t distribution with $n + m - 2$ degrees of freedom, and its error probability converges at the same rate as our test.

It is worth noting that it is possible to obtain bounds for the deviation between expectations $\mu[p]$ and the empirical means $\mu[X]$ in a completely analogous fashion. In fact, the proof requires symmetrization by means of a *ghost sample*, *i.e.,* a second set of observations drawn from the same distribution.

3.1.4 An Unbiased Test Based on the Asymptotic Distribution of the U-Statistic

We now propose a second test, which is based on the asymptotic distribution of an unbiased estimate of MMD^2. We begin by defining this test statistic.

Lemma 40 *Given x and x' independent random variables with distribution p, and y and y' independent random variables with distribution q, the population* MMD^2 *is*

$$\mathrm{MMD}^2(\mathcal{F}, p, q) = \mathbf{E}_{x,x'\sim p}\left[k(x, x')\right] - 2\mathbf{E}_{x\sim p, y\sim q}\left[k(x, y)\right] + \mathbf{E}_{y,y'\sim q}\left[k(y, y')\right] \tag{3.9}$$

*Let $Z := (z_1, \ldots, z_m)$ be m i.i.d. random variables, where $z_i := (x_i, y_i)$ (*i.e., *we assume $m = m_1 = m_2$). An* unbiased empirical estimate of MMD^2 is

$$\mathrm{MMD}_u^2(\mathcal{F}, X, Y) = \frac{1}{(m)(m-1)} \sum_{i\neq j}^{m} h(z_i, z_j), \tag{3.10}$$

which is a one-sample U-statistic with $h(z_i, z_j) := k(x_i, x_j) + k(y_i, y_j) - k(x_i, y_j) - k(x_j, y_i)$.

Proof [Lemma 40]
By Theorem 31 we know that $\mathrm{MMD}(\mathcal{F}, p, q)$ is given by $\|\mu[p] - \mu[q]\|_{\mathcal{H}}$. Exploiting the fact that we are dealing with a Hilbert space yields:

$$\begin{aligned}\|\mu[p] - \mu[q]\|_{\mathcal{H}}^2 &= \langle \mu[p], \mu[p]\rangle - 2\langle \mu[p], \mu[q]\rangle + \langle \mu[q], \mu[q]\rangle \\ &= \mathbf{E}_{x,x'\sim p}\langle \phi(x), \phi(x')\rangle - 2\mathbf{E}_{x\sim p, y\sim q}\langle \phi(x), \phi(y)\rangle + \mathbf{E}_{y,y'\sim q}\langle \phi(y), \phi(y')\rangle .\end{aligned}$$

To complete the proof we use that $\langle \phi(x), \phi(x')\rangle = k(x, x')$. This proves the first claim. The second claim is completely analogous, *i.e.,* $\mathrm{MMD}(\mathcal{F}, X, Y) = \|\mu[X] - \mu[Y]\|_{\mathcal{H}}$, only that now we need to replace expectations by empirical averages. ■

The empirical statistic is an unbiased estimate of MMD^2, although it does not have minimum variance [Serfling, 1980, Section 5.1.4].

The asymptotic distribution of this test statistic under $\mathcal{H}_1$ is given by [Serfling, 1980, Section 5.5.1], and the distribution under $\mathcal{H}_0$ follows from [Serfling, 1980, Section 5.5.2] and [Anderson et al., 1994, Appendix].

Theorem 41 *We assume $\mathbf{E}\left(h^2\right) < \infty$. Under $\mathcal{H}_1$, MMD_u^2 converges in distribution (defined e.g. by [Grimmet and Stirzaker, 2001, Section 7.2]) to a Gaussian according to*

$$m^{\frac{1}{2}}\left(\mathrm{MMD}_u^2 - \mathrm{MMD}^2(\mathcal{F}, p, q)\right) \xrightarrow{D} \mathcal{N}\left(0, \sigma_u^2\right),$$

where $\sigma_u^2 = 4\left(\mathbf{E}_z\left[(\mathbf{E}_{z'}h(z, z'))^2\right] - \left[\mathbf{E}_{z,z'}(h(z, z'))\right]^2\right)$, uniformly at rate $1/\sqrt{m}$ [Serfling, 1980, Theorem B, p. 193]. Under $\mathcal{H}_0$, the U-statistic is degenerate, meaning $\mathbf{E}_{z'}h(z, z') = 0$. In this case, MMD_u^2 *converges in distribution according to*

$$m\mathrm{MMD}_u^2 \xrightarrow{D} \sum_{l=1}^{\infty} \lambda_l\left[z_l^2 - 2\right], \tag{3.11}$$

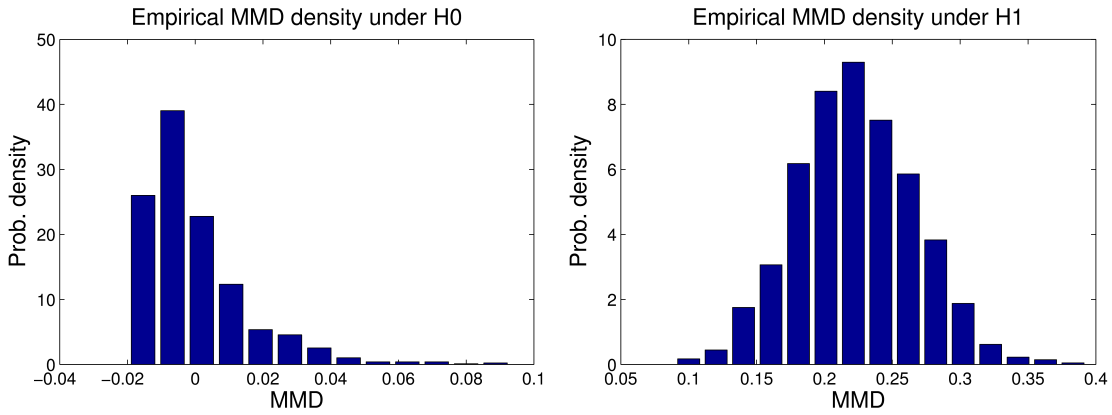

Figure 3.1: **Left:** Empirical distribution of the MMD under $\mathcal{H}_0$, with p and q both Gaussians with unit standard deviation, using 50 samples from each. **Right:** Empirical distribution of the MMD under $\mathcal{H}_1$, with p a Laplace distribution with unit standard deviation, and q a Laplace distribution with standard deviation $3\sqrt{2}$, using 100 samples from each. In both cases, the histograms were obtained by computing 2000 independent instances of the MMD.

where $z_l \sim \mathcal{N}(0, 2)$ *i.i.d.,* λ_i *are the solutions to the eigenvalue equation*

$$\int_{\mathcal{X}} \tilde{k}(x, x')\psi_i(x)dp(x) = \lambda_i\psi_i(x'),$$

and $\tilde{k}(x_i, x_j) := k(x_i, x_j) - \mathbf{E}_x k(x_i, x) - \mathbf{E}_x k(x, x_j) + \mathbf{E}_{x,x'} k(x, x')$ *is the centered RKHS kernel.*

We illustrate the MMD density under both the null and alternative hypotheses by approximating it empirically for both $p = q$ and $p \neq q$. Results are plotted in Figure 3.1.

Our goal is to determine whether the empirical test statistic MMD_u^2 is so large as to be outside the $1 - \alpha$ quantile of the null distribution in (3.11) (consistency of the resulting test is guaranteed by the form of the distribution under $\mathcal{H}_1$). One way to estimate this quantile is using the bootstrap on the aggregated data, following [Arcones and Giné, 1992].

Alternatively, we may approximate the null distribution by fitting Pearson curves to its first four moments [Johnson et al., 1994, Section 18.8]. Taking advantage of the degeneracy of the U-statistic, we obtain [Gretton et al., 2007b]

$$\mathbf{E}\left(\left[\mathrm{MMD}_u^2\right]^2\right) = \frac{2}{m(m-1)}\mathbf{E}_{z,z'}\left[h^2(z, z')\right]$$

and

$$\mathbf{E}\left(\left[\mathrm{MMD}_u^2\right]^3\right) = \frac{8(m-2)}{m^2(m-1)^2}\mathbf{E}_{z,z'}\left[h(z, z')\mathbf{E}_{z''}\left(h(z, z'')h(z', z'')\right)\right] + O(m^{-4}). \tag{3.12}$$

The fourth moment $\mathbf{E}\left(\left[\mathrm{MMD}_u^2\right]^4\right)$ is not computed, since it is both very small ($O(m^{-4})$) and expensive to calculate ($O(m^4)$). Instead, we replace the kurtosis with its lower bound $\mathrm{kurt}\left(\mathrm{MMD}_u^2\right) \geq \left(\mathrm{skew}\left(\mathrm{MMD}_u^2\right)\right)^2 + 1$.

3.1.5 Experiments

We conducted distribution comparisons using our MMD-based tests on datasets from three real-world domains: database applications, bioinformatics, and neurobiology. We investigated the uniform convergence approach (MMD), the asymptotic approach with bootstrap (MMD_u^2 B), and the asymptotic approach with moment matching to Pearson curves (MMD_u^2 M). We also compared against several alternatives from the literature (where applicable): the multivariate t-test, the Friedman-Rafsky Kolmogorov-Smirnov generalization (*Smir*), the Friedman-Rafsky Wald-Wolfowitz generalization (*Wolf*), the Biau-Györfi test (*Biau*), and the Hall-Tajvidi test (*Hall*). Note that we do not apply the Biau-Györfi test to high-dimensional problems (see end of Section 3.1.1).

An important issue in the practical application of the MMD-based tests is the selection of the kernel parameters. We illustrate this with a Gaussian RBF kernel, where we must choose the kernel width σ. The empirical MMD is zero both for kernel size $\sigma = 0$ (where the aggregate Gram matrix over X and Y is a unit matrix), and also approaches zero as $\sigma \to \infty$ (where the aggregate Gram matrix becomes uniformly constant). We set σ to be the median distance between points in the aggregate sample, as a compromise between these two extremes: this remains a heuristic, however, and the optimum choice of kernel size is an ongoing area of research.

Data Integration As a first application of MMD, we performed distribution testing for data integration: the objective is to aggregate two datasets into a single sample, with the understanding that both original samples are generated from the same distribution. Clearly, it is important to check this last condition before proceeding, or an analysis could detect patterns in the new dataset that are caused by combining the two different source distributions, and not by real-world phenomena. We chose several real-world settings to perform this task: we compared microarray data from normal and tumor tissues (Health status), microarray data from different subtypes of cancer (Subtype), and local field potential (LFP) electrode recordings from the Macaque primary visual cortex (V1) with and without spike events (Neural Data I and II). In all cases, the two data sets have different statistical properties, but the detection of these differences is made difficult by the high data dimensionality.

We applied our tests to these datasets in the following fashion. Given two datasets A and B, we either chose one sample from A and the other from B (*attributes = different*); or both samples from either A or B (*attributes = same*). We then repeated this process up to 1200 times. Results are reported in Table 3.1. Our asymptotic tests perform better than all competitors besides *Wolf*: in the latter case, we have greater Type II error for one neural dataset, lower Type II error on the Health Status data (which has very high dimension and low sample size), and identical (error-free) performance on the remaining examples. We note that the Type I error of the bootstrap test on the Subtype dataset is far from its design value of 0.05, indicating that the Pearson curves provide a better threshold estimate for these low sample sizes. For the remaining datasets, the Type I errors of the Pearson and Bootstrap approximations are close. Thus, for larger datasets, the bootstrap is to be preferred, since it costs $O(m^2)$, compared with a cost of $O(m^3)$ for Pearson (due to the

cost of computing (3.12)). Finally, the uniform convergence-based test is too conservative, finding differences in distribution only for the data with largest sample size.

Dataset	Attr.	MMD	MMD_u^2 B	MMD_u^2 M	t-test	Wolf	Smir	Hall
Neural Data I	Same	100.0	96.5	96.5	100.0	97.0	95.0	96.0
	Different	50.0	**0.0**	**0.0**	42.0	**0.0**	10.0	49.0
Neural Data II	Same	100.0	94.6	95.2	100.0	95.0	94.5	96.0
	Different	100.0	3.3	3.4	100.0	**0.8**	31.8	5.9
Health status	Same	100.0	95.5	94.4	100.0	94.7	96.1	95.6
	Different	100.0	1.0	**0.8**	100.0	2.8	44.0	35.7
Subtype	Same	100.0	99.1	96.4	100.0	94.6	97.3	96.5
	Different	100.0	**0.0**	**0.0**	100.0	**0.0**	28.4	0.2

Table 3.1: Distribution testing for data integration on multivariate data. Numbers indicate the percentage of repetitions for which the null hypothesis (p=q) was accepted, given $\alpha = 0.05$. Sample size (dimension; repetitions of experiment): Neural I 4000 (63; 100) ; Neural II 1000 (100; 1200); Health Status 25 (12,600; 1000); Subtype 25 (2,118; 1000).

Attribute Matching Our second series of experiments addresses automatic attribute matching. Given two databases, we want to detect corresponding attributes in the schemas of these databases, based on their data-content (as a simple example, two databases might have respective fields Wage and Salary, which are assumed to be observed via a subsampling of a particular population, and we wish to automatically determine that both Wage and Salary denote to the same underlying attribute). We use a two-sample test on pairs of attributes from two databases to find corresponding pairs.[1] This procedure is also called *table matching* for tables from different databases. We performed attribute matching as follows: first, the dataset D was split into two halves A and B. Each of the n attributes in A (and B, resp.) was then represented by its instances in A (resp. B). We then tested all pairs of attributes from A and from B against each other, to find the optimal assignment of attributes $A_1, \ldots, A_n$ from A to attributes $B_1, \ldots, B_n$ from B. We assumed that A and B contain the same number of attributes.

As a naive approach, one could assume that any possible pair of attributes might correspond, and thus that every attribute of A needs to be tested against all the attributes of B to find the optimal match. We report results for this naive approach, aggregated over all pairs of possible attribute matches, in Table 3.2. We used three datasets: the census income dataset from the UCI KDD archive (CNUM), the protein homology dataset from the 2004 KDD Cup (BIO) [Caruana and Joachims, 2004], and the forest dataset from the UCI ML archive [Blake and Merz, 1998]. For the final dataset, we performed univariate matching of attributes (FOREST) and multivariate matching of tables (FOREST10D) from two different databases, where each table represents one type of forest. Both our

[1]Note that corresponding attributes may have different distributions in real-world databases. Hence, schema matching cannot solely rely on distribution testing. Advanced approaches to schema matching using MMD as one key statistical test are a topic of ongoing research.

asymptotic MMD_u^2-based tests perform as well as or better than the alternatives, notably for CNUM, where the advantage of MMD_u^2 is large. Unlike in Table 3.1, the next best alternatives are not consistently the same across all data: e.g. in BIO they are *Wolf* or *Hall*, whereas in FOREST they are *Smir*, *Biau*, or the t-test. Thus, MMD_u^2 appears to perform more consistently across the multiple datasets. The Friedman-Rafsky tests do not always return a Type I error close to the design parameter: for instance, *Wolf* has a Type I error of 9.7% on the BIO dataset (on these data, MMD_u^2 has the joint best Type II error without compromising the designed Type I performance). Finally, our uniform convergence approach performs much better than in Table 3.1, although surprisingly it fails to detect differences in FOREST10D.

A more principled approach to attribute matching is also possible. Assume that $\phi(A) = (\phi_1(A_1), \phi_2(A_2), ..., \phi_n(A_n))$: in other words, the kernel decomposes into kernels on the individual attributes of A (and also decomposes this way on the attributes of B). In this case, MMD^2 can be written $\sum_{i=1}^{n} \|\mu_i(A_i) - \mu_i(B_i)\|^2$, where we sum over the MMD terms on each of the attributes. Our goal of optimally assigning attributes from B to attributes of A via MMD is equivalent to finding the optimal permutation π of attributes of B that minimizes $\sum_{i=1}^{n} \|\mu_i(A_i) - \mu_i(B_{\pi(i)})\|^2$. If we define $C_{ij} = \|\mu_i(A_i) - \mu_i(B_j)\|^2$, then this is the same as minimizing the sum over $C_{i,\pi(i)}$. This is the linear assignment problem, which costs $O(n^3)$ time using the Hungarian method [Kuhn, 1955].

Dataset	Attr.	MMD	MMD_u^2 B	MMD_u^2 M	t-test	Wolf	Smir	Hall	Biau
BIO	Same	100.0	93.8	94.8	95.2	90.3	95.8	95.3	99.3
	Different	20.0	**17.2**	17.6	36.2	**17.2**	18.6	17.9	42.1
FOREST	Same	100.0	96.4	96.0	97.4	94.6	99.8	95.5	100.0
	Different	4.9	**0.0**	**0.0**	0.2	3.8	**0.0**	50.1	**0.0**
CNUM	Same	100.0	94.5	93.8	94.0	98.4	97.5	91.2	98.5
	Different	15.2	2.7	**2.5**	19.17	22.5	11.6	79.1	50.5
FOREST10D	Same	100.0	94.0	94.0	100.0	93.5	96.5	97.0	100.0
	Different	100.0	**0.0**	**0.0**	**0.0**	**0.0**	1.0	72.0	100.0

Table 3.2: Naive attribute matching on univariate (BIO, FOREST, CNUM) and multivariate data (FOREST10D). Numbers indicate the percentage of accepted null hypothesis (p=q) pooled over attributes. $\alpha = 0.05$. Sample size (dimension; attributes; repetitions of experiment): BIO 377 (1; 6; 100); FOREST 538 (1; 10; 100); CNUM 386 (1; 13; 100); FOREST10D 1000 (10; 2; 100).

We tested this 'Hungarian approach' to attribute matching via MMD_u^2 B on three univariate datasets (BIO, CNUM, FOREST) and for table matching on a fourth (FOREST10D). Results are shown in Table 3.3. Besides BIO, MMD_u^2 B reached at least 99.8% accuracy on all datasets.

3.1.6 Summary

In this chapter, we have established three simple multivariate tests for comparing two distributions p and q, based on samples of size m_1 and m_2 from these respective distributions.

Dataset	Data type	No. attributes	Sample size	Repetitions	% correct matches
BIO	univariate	6	377	100	90.0
CNUM	univariate	13	386	100	99.8
FOREST	univariate	10	538	100	100.0
FOREST10D	multivariate	2	1000	100	100.0

Table 3.3: Hungarian Method for attribute matching via MMD_u^2 B on univariate (BIO, CNUM, FOREST), multivariate (FOREST10D) ($\alpha = 0.05$; '% correct matches' is the percentage of the correct attribute matches detected over all repetitions).

The test statistics are based on the maximum deviation of the expectation of a function evaluated on each of the random variables, taken over a sufficiently rich function class, which also allows us to express the empirical estimates of our test statistic in terms of kernels. We do not require density estimates as an intermediate step. Two of our tests provide error guarantees that are exact and distribution-free for finite sample sizes, as with [Rosenbaum, 2005]. We also give a third test based on the asymptotic normality of the associated test statistic (as in the tests of [Friedman and Rafsky, 1979, Anderson et al., 1994]). All three tests can be computed in $O((m_1 + m_2)^2)$, which is faster than the approaches of [Rosenbaum, 2005, Friedman and Rafsky, 1979].

3.2 Graph Similarity via Maximum Mean Discrepancy

As the empirical estimate of MMD and the acceptance threshold for its associated two-sample tests can be expressed in terms of kernels, we can combine both with a graph kernel to obtain the first two-sample test for sets of graphs described in the literature. In this section, we explore this application of our two-sample tests to sets of graphs, and extend them to pairs of graphs.

3.2.1 Two-Sample Test on Sets of Graphs

Given two sets of graphs X and Y, each of size m (assuming $m = m_1 = m_2$), from distributions p and q, and a universal graph kernel k, we can estimate MMD_u^2 via Lemma 40 and employ the asymptotic test from Section 3.1.4 in order to decide whether to reject the null hypothesis $p = q$. As an alternative to the asymptotic test, we could employ the biased estimate from Theorem 34, and the statistical test based on uniform convergence bounds from Section 3.1.3.

However, there are two open questions in this context: Which of the existing graph kernels is universal in the sense of [Steinwart, 2002]? If there are none, or none that are efficient to compute, can we still employ MMD on sets of graphs using a non-universal kernel? We will consider these two questions in the following.

Universal Kernels on Graphs

While many examples of universal kernels on compact subsets of $\mathbb{R}^d$ are known [Steinwart, 2002], little attention has been given to finite domains. It turns out that the issue is considerably easier in this case: the weaker notion of *strict positive definiteness* (kernels inducing nonsingular Gram matrices ($K_{ij} = k(x_i, x_j)$) for arbitrary sets of distinct

points x_i) ensures that every function on a discrete domain $\mathcal{X} = \{x_1, \ldots, x_m\}$ lies in the corresponding RKHS, and hence that the kernel is universal. To see this, let $f \in \mathbb{R}^m$ be an arbitrary function on $\mathcal{X}$. Then $\alpha = K^{-1}f$ ensures that the function $f = \sum_j \alpha_j k(., x_j)$ satisfies $f(x_i) = f_i$ for all i.

While there are strictly positive definite kernels on strings [Borgwardt et al., 2006], for graphs unfortunately no such strictly positive definite kernels exist which are efficiently computable. Note first that it is necessary for strict positive definiteness that $\phi(x)$ be injective, for otherwise we would have $\phi(x) = \phi(x')$ for some $x \neq x'$, implying that the kernel matrix obtained from $X = \{x, x'\}$ is singular. However, as [Gärtner et al., 2003] show, an injective $\phi(x)$ allows one to match graphs by computing $\|\phi(x) - \phi(x')\|^2 = k(x, x) + k(x', x') - 2k(x, x')$. In Section 1.4, we have seen that the corresponding all-subgraphs kernel is NP-hard to compute, and hence impractical in real-world applications. Due to these efficiency problems, let us discuss the consequences of employing a non-universal kernel with MMD next.

MMD and Non-Universal Kernels

So far, we have focused on the case of universal kernels, as MMD using universal kernels is a test for identity of arbitrary Borel probability distributions.

However, note that for instance in pattern recognition, there might well be situations where the best kernel for a given problem is not universal. In fact, the kernel corresponds to the choice of a prior, and thus using a kernel which does *not* afford approximations of arbitrary continuous functions can be very useful — provided that the functions it does approximate are known to be solutions of the given problem.

The situation is similar for MMD. Consider the following example: suppose we knew that the two distributions we are testing are both Gaussians (with unknown mean vectors and covariance matrices). Since the empirical means of products of input variables up to order two are sufficient statistics for the family of Gaussians, we should thus work in an RKHS spanned by products of order up to two — any higher order products contain no information about the underlying Gaussians and can therefore mislead us. It is straightforward to see that for $c > 0$, the polynomial kernel $k(x, x') = (\langle x, x' \rangle + c)^2$ does the job: it equals

$$\sum_{i,j=1}^{d} x_i x_j x'_i x'_j + 2c \sum_{i=1}^{d} x_i x'_i + c^2 = \langle \phi(x), \phi(x') \rangle \,,$$

where $\phi(x) = (c, \sqrt{2c}x_1, \ldots, \sqrt{2c}x_d, x_i x_j | i, j = 1, \ldots, d)^\top$. If we want to test for differences in higher order moments, we use a higher order kernel[2] $k(x, x') = (\langle x, x' \rangle + c)^p$. To get a test for comparing two arbitrary distributions, we need to compare all moments, which is precisely what we do when we consider the infinite-dimensional RKHS associated with a universal kernel.

Based on these considerations and to keep computation practical, we resort to our graph kernels from Section 2 and from [Borgwardt et al., 2005] that are more efficient to

[2]Kernels with infinite-dimensional RKHS can be viewed as a nonparametric generalization where we have infinitely many sufficient statistics.

compute and provide useful measures of similarity on graphs, as demonstrated in several experiments.

Combining MMD with graph kernels, we are now in a position to compare two sets of graphs and to decide whether they are likely to originate from the same distribution based on a significance level α. Recall that the design parameter α is the probability of erroneously concluding that two sets of graphs follow different distributions, albeit they are drawn from the same distribution.

Experiments

We can employ this type of statistical test for the similarity of sets of graphs to find corresponding groups of graphs in two databases. Problems of this kind may arise in data integration, when two collections of graph-structured data shall be matched. We explore this application in our subsequent experimental evaluation. As we found the uniform convergence-based test to be very conservative in our experimental evaluation in Section 3.1.5, we used the asymptotic test that showed superior performance on small sample sizes for our experiments.

To evaluate MMD on graph data, we obtained two datasets of protein graphs (Protein and Enzyme) and used the random walk graph kernel for proteins from [Borgwardt et al., 2005] for table matching via the Hungarian method (the other tests were not applicable to this graph data). The challenge here is to match tables representing one functional class of proteins (or enzymes) from dataset A to the corresponding tables (functional classes) in B.

Enzyme Graph Data

In more detail, we study the following scenario: Two researchers have each dealt with 300 enzyme protein structures. These two sets of 300 proteins are disjunct, *i.e.,* there is no protein studied by both researchers. They have assigned the proteins to six different classes according to their enzyme activity. However, both have used different protein function classification schemas for these classes and are not sure which of these classes correspond to each other.

To find corresponding classes, MMD can be employed. We obtained 600 proteins modeled as graphs from [Borgwardt et al., 2005], and randomly split these into two subsets A and B of 300 proteins each, such that 50 enzymes in each subset belong to each of the six EC top level classes (EC1 to EC6). We then computed MMD for all pairs of the six EC classes from subset A and subset B to check if the null hypothesis is rejected or accepted. To compute MMD, we employed the protein random walk kernel function for protein graphs, following [Borgwardt et al., 2005]. This random walk kernel measures similarity between two graphs by counting matching walks in two graphs.

We compared all pairs of classes via MMD_u^2 B, and repeated the experiment 100 times. Note that a comparison to competing statistical tests is unnecessary, as — to the best of our knowledge — no other distribution test for structured data exists.

We report results in Table 3.4. For a significance level of $\alpha = 0.05$, MMD rejected the null hypothesis that both samples are from the same distribution whenever enzymes from

two different EC classes were compared. When enzymes from the same EC classes were compared, MMD accepted the null hypothesis. MMD thus achieves error-free data-based schema matching here.

Protein Graph Data

We consider a second schema matching problem on complex data which is motivated by bioinformatics: If two protein databases are merged, we want to automatically find out which tables represent enzymes and which do not represent enzymes. We assume that these molecules are represented as graphs in both databases.

We repeat the above experiments for graph representations of 1128 proteins, 665 of which are enzymes and 463 of which are non-enzymes. This time we consider 200 graphs per sample, *i.e.,* two samples of 200 protein graphs are compared via the protein random walk kernel from above. Again, we compare samples from the same class (both enzymes or both non-enzymes), or samples from different classes (one enzymes, one non-enzymes) via MMD.

As on the enzyme dataset, MMD_u^2 B made no errors. Results are shown in Table 3.4.

Dataset	Data type	No. attributes	Sample size	Repetitions	% correct matches
Enzyme	graph	6	50	50	100.0
Protein	graph	2	200	50	100.0

Table 3.4: Matching database tables via MMD_u^2 B on graph data (Enzyme, Protein) ($\alpha = 0.05$; '% correct matches' is the percentage of the correct attribute matches detected over all repetitions).

3.2.2 Two-Sample Test on Pairs of Graphs

After defining a statistical test for graph similarity on sets of graphs, one question remains unanswered: Can we also employ MMD to define a two-sample test on pairs of graphs? The answer is yes, and in this section we will show why.

In contrast to Section 3.2, we now define X and Y to represent two graphs G and G', not two sets of graphs. MMD requires X and Y to be i.i.d. samples from two underlying distributions p and q. Hence the decisive question is: How to represent graphs as i.i.d. samples?

Actually, we have already dealt with this problem: Recall our graphlet sampling scheme from Section 2.3. There, each graph G is described by a sample of graphlets $X = \{x_1, \ldots, x_{m_1}\}$. These are i.i.d. drawn from G. Analogously G' is a graph and $Y = \{y_1, \ldots, y_{m_2}\}$ is a sample of graphlets from G'. In other terms, in the graphlet sampling framework, each graph is a distribution of graphlets, and we sample graphlets from that distribution. We can now apply MMD to samples X and Y of graphlets from two graphs G and G', and we can decide via MMD if these graphlet samples are likely to originate from the same underlying graphlet distribution, which represents a graph in this setting. The natural choice of kernel for this application of MMD to graphs is to employ an isomorphism kernel on the graphlets. This isomorphism kernel is 1 if two graphlets are isomorphic, zero otherwise (see Equation (1.28)).

To summarize, we employ MMD on samples of graphlets to define a statistical test of similarity for a pair of graphs. We choose graphlets and isomorphism kernel because of their excellent experimental performance when comparing graph topologies in Section 2.3.5. Note that we can also sample graph substructures other than graphlets and kernels other than the isomorphism kernel of course, and apply the same scheme to them.

3.2.3 Experiments

species	yeast	fly	human	worm
yeast	0	10	10	3
fly	—	0	10	10
human	—	—	0	10
worm	—	—	—	0

Table 3.5: Two-sample tests via MMD on pairs of protein interaction networks. Number indicate how often MMD rejected the null hypothesis ($p = q$) in 10 repetitions of each comparison. Statistics on PPI networks: yeast (2401 nodes, 11000 edges), fly (4602 nodes, 4637 edges), human (1753 nodes, 3113 edges), worm (1356 nodes, 1983 edges).

To test similarity of two graphs via MMD, we choose a task from bioinformatics. There, several studies have dealt with the topic of finding so-called motifs, i.e. small frequent subgraphs, within protein interaction network [Przulj, 2007]. Biologists are interested in whether some of these motifs are more conserved than others, and if certain motifs are more conserved in one species than another species [Shen-Orr et al., 2002, Wuchty et al., 2003, Lee et al., 2006]. In our experiment, we wanted to find out if we can distinguish samples if two sets of graphlets, i.e. motifs, have been drawn from the same or from different species.

We obtained protein-protein interaction (PPI) networks of four different species from [Przulj, 2007]: worm (*C. elegans*) [Li et al., 2004], human (*H. sapiens*) [Zanzoni et al., 2002], fly (*D. melanogaster*), [Giot et al., 2003], and yeast (*S. cerevisiae*) [von Mering et al., 2002]. Represented as unlabeled graphs, these 4 networks formed the dataset for our experiment. We report sizes and number of edges of these graphs in Table 3.5.

Setting precision parameter $\epsilon = 0.1$ and confidence parameter $\delta = 0.1$, we sampled $m = 1847$ graphlets from each graph, *i.e.,* subgraphs with 4 nodes. We only considered graphlets with at least 1 edge ($a = 10$). For each comparison of two PPIs, we sampled 1847 graphlets from each graph. We then compared all pairs of graphs via MMD. We set $\alpha = 0.05$ for these test runs and used bootstrapping for determining the threshold for MMD_u^2 B. We repeated this whole experiment 10 times, resulting in 10 decisions per pair of interaction networks.

We present results in Table 3.5. When samples were drawn from the same graph, MMD made no Type I error in 40 decisions. When samples were drawn from different graphs, MMD produced 7 Type II errors in 60 decisions.

MMD had no difficulties in distinguishing whether graphlets had been drawn from yeast, fly or human, but it failed to tell apart samples that were originated from yeast and worm in 7 out of 10 repetitions, indicating that the frequencies of graphlets in these species are similar to each other.

3.2.4 Summary

In this chapter, we have — to the best of our knowledge — developed the first two-sample test that is applicable to graphs. We proceeded in three steps: First, in Section 3.1, we defined a test statistic called Maximum Mean Discrepancy and we developed associated statistical tests for the two-sample problem. Second, in Section 3.2.1, we exploited the fact that Maximum Mean Discrepancy can be expressed in terms of kernels, and applied our statistical test to sets of graphs for matching tables from different databases. Third, in Section 3.2.2, we extended our test to pairs of graphs, by representing each graph as a sample of its subgraphs of limited size. We then applied our method to compare protein-protein-interaction networks based on samples of motifs, *i.e.,* small subgraphs from these networks.

To summarize, Maximum Mean Discrepancy allows us to tackle two-sample problems on graphs for which no alternative approach exists. Unfolding the full potential of our novel two-sample tests in applications will be one topic of our future research.

Chapter 4

Feature Selection on Graphs

Up to this point, our goal was to find out if two graphs are similar. The natural follow-on question to ask is: *Why* are two graphs similar? This leads directly from the problem of classification to the problem of feature selection.

Classification and feature selection are often accompanying tasks. On the one hand, one wants to build a classifier that is able to correctly predict class memberships of unlabeled data objects. On the other hand, one wants to select the features of an object that are most correlated to its class membership.

A multitude of papers has dealt with feature selection from objects that are represented by feature vectors [Guyon and Elisseeff, 2003]. Selecting features is equivalent to choosing a set of components or dimensions from these vectors.

Feature selection on graphs, however, has received very little attention. The main reason is that the number of features of a graph, namely its subgraphs, grows exponentially with the number of its nodes. Hence it is computationally expensive to consider all features of a graph and to then perform feature selection.

Subgraph selection has been tackled in a different branch of graph mining though, in frequent subgraph mining. The task of frequent subgraph mining can be described as follows: Given a database D of m graphs, determine all subgraphs S that have embeddings in at least t of the m graphs in D. Hence in this definition, a graph is deemed frequent if it is a subgraph of at least t of the graphs in D. Efficient algorithms exist for mining all frequent subgraphs from a database of graphs [Yan and Han, 2002], but obviously the problem is NP-hard, as one has to repeatedly perform subgraph isomorphism checks.

Recently, these frequent subgraph mining algorithms have been applied to graph classification tasks, *i.e.,* to find frequent subgraphs in datasets consisting of different classes of graphs. In this setting, the major drawback of these frequent subgraph tools is that frequency alone is not a good measure of discriminative power. Both highly frequent and very rare subgraphs may be rather useless for distinguishing different classes of graphs. Setting the frequency threshold t very high may result in finding only high-frequency patterns that are abundant in all classes. Choosing a low frequency threshold t will cause these algorithms to enumerate thousands and millions of frequent subgraphs, without providing information on the relevance of these frequent subgraphs for the class membership of a graph: Hence frequent subgraph mining algorithms would benefit from a feature selector

that finds the most informative frequent subgraphs within their solution set.

In this chapter, we tackle the problem of defining a feature selector for frequent subgraph mining. Towards this end, we define a feature selection algorithm that is purely based on kernels on both the data objects and their class labels (Section 4.1). Its core idea is to measure dependence between data and class labels via the Hilbert-Schmidt Independence Criterion (HSIC), and to greedily maximize this dependence for a subset of features. We show that the backward feature selection variant of kernel-based feature selector is competitive with state-of-the-art approaches on vectorial data in our first experimental evaluation. We then exploit the fact that kernel-based HSIC feature selection can be directly transfered to feature selection on graphs (Section 4.2). Due to the computational expensiveness of considering *all* subgraphs, we focus on feature selection among *frequent* subgraphs via HSIC, and propose an efficient forward feature selection algorithm for this problem (Section 4.2.3). Our second experimental evaluation shows that our novel approach to feature selection among frequent subgraphs selects fewer features than state-of-the-art approaches, and at the same time, these features lead to higher accuracies in classification experiments than those selected by competitor methods.

4.1 A Dependence based Approach to Feature Selection

4.1.1 The Problem of Feature Selection

In supervised learning problems, we are typically given m data points $x \in \mathcal{X}$ and their labels $y \in \mathcal{Y}$. The task is to find a functional dependence between x and y, $f : x \longmapsto y$, subject to certain optimality conditions. Representative tasks include binary classification, multi-class classification, regression and ranking. We often want to reduce the dimension of the data (the number of features) before the actual learning [Guyon and Elisseeff, 2003]; a larger number of features can be associated with higher data collection cost, more difficulty in model interpretation, higher computational cost for the classifier, and decreased generalisation ability. It is therefore important to select an informative feature subset.

The problem of supervised feature selection can be cast as a combinatorial optimization problem. We have a full set of features, denoted $\mathcal{S}$ (each element in $\mathcal{S}$ corresponds to one dimension of the data). We use these features to predict a particular outcome, for instance the presence of cancer: clearly, only a subset $\mathcal{T}$ of features will be relevant. Suppose the relevance of a feature subset to the outcome is quantified by $\mathcal{Q}(\mathcal{T})$, and is computed by restricting the data to the dimensions in $\mathcal{T}$. Feature selection can then be formulated as

$$\mathcal{T}_0 = \arg\max_{\mathcal{T}\subseteq\mathcal{S}} \ \mathcal{Q}(\mathcal{T}) \qquad \text{subject to} \quad |\mathcal{T}| \leq \theta, \tag{4.1}$$

where $|\cdot|$ computes the cardinality of a set and θ upper bounds the number of selected features. Two important aspects of problem (4.1) are the choice of the criterion $\mathcal{Q}(\mathcal{T})$ and the selection algorithm.

Feature Selection Criterion. The choice of $\mathcal{Q}(\mathcal{T})$ should respect the underlying supervised learning tasks — estimate function dependence f from training data and guarantee f predicts well on test data. Therefore, good criteria should satisfy two conditions:

- **I**: $\mathcal{Q}(\mathcal{T})$ is capable of detecting any prominent (nonlinear as well as linear) functional dependence between the data and the labels.
- **II**: $\mathcal{Q}(\mathcal{T})$ is concentrated with respect to the underlying measure. This guarantees with high probability that the detected functional dependence is preserved in test data.

While many criteria have been explored, few take these two conditions explicitly into account. Examples include the leave-one-out error bound of SVM [Weston et al., 2000] and the mutual information [Koller and Sahami, 1996]. Although the latter has good theoretical justification, it requires density estimation, which is problematic for high-dimensional and continuous variables. We sidestep these problems by employing a mutual-information *like* quantity — the Hilbert Schmidt Independence Criterion (HSIC) [Gretton et al., 2005]. HSIC uses kernels for measuring dependence, which makes it attractive for our ultimate goal of feature selection on graphs (see Section 4.2). Furthermore, HSIC does not require density estimation. HSIC also has good uniform convergence guarantees. As we show in subsection 4.1.2, HSIC satisfies conditions **I** and **II**, required for $\mathcal{Q}(\mathcal{T})$.

Feature Selection Algorithm. Finding a global optimum for (4.1) is in general NP-hard [Weston et al., 2003]. Many algorithms transform (4.1) into a continuous problem

by introducing weights on the dimensions [Weston et al., 2000, Bradley and Mangasarian, 1998, Weston et al., 2003]. These methods perform well for linearly separable problems. For nonlinear problems, however, the optimization usually becomes non-convex and a local optimum does not necessarily provide good features. Greedy approaches, forward selection and backward elimination, are often used to tackle problem (4.1) directly. Forward selection tries to increase $\mathcal{Q}(\mathcal{T})$ as much as possible for each inclusion of features, and backward elimination tries to achieve this for each deletion of features [Guyon et al., 2002]. Although forward selection is computationally more efficient, backward elimination provides better features in general since the features are assessed within the context of all others.

BAHSIC. In principle, HSIC can be employed using either the forwards or backwards strategy, or a mix of strategies. However, in this section, we will focus on a backward elimination algorithm. Our initial experiments (not reported) showed that backward elimination outperforms forward selection for HSIC on vectorial data, albeit being more runtime intensive. Still, forward selection will be the focus of Section 4.2.3, when we tackle the actual goal of feature selection on graphs, because its better runtime performance is a huge advantage on graphs.

Backward elimination using HSIC (BAHSIC) is a filter method for feature selection. It selects features independent of a particular classifier. Such decoupling not only facilitates subsequent feature interpretation but also speeds up the computation over wrapper and embedded methods.

Furthermore, BAHSIC is directly applicable to binary, multiclass, and regression problems. Most other feature selection methods are only formulated either for binary classification or regression. Multi-class extension of these methods is usually accomplished using a one-versus-the-rest strategy. Still fewer methods handle classification and regression cases at the same time. BAHSIC, on the other hand, accommodates all these cases in a principled way: by choosing different kernels, BAHSIC also subsumes many existing methods as special cases. Such versatility of BAHSIC originates from the generality of HSIC. Therefore, we begin our exposition with an introduction of HSIC.

4.1.2 Measures of Dependence

We define $\mathcal{X}$ and $\mathcal{Y}$ broadly as two domains from which we draw samples $(x, y) \sim Pr_{xy}$: these may be real valued, vector valued, class labels, strings, graphs, and so on. We define a (possibly nonlinear) mapping $\phi(x) \in \mathcal{G}$ from each $x \in \mathcal{X}$ to a feature space $\mathcal{G}$, such that the inner product between the features is given by a kernel function $k(x, x') := \langle \phi(x), \phi(x') \rangle$: $\mathcal{G}$ is called a Reproducing Kernel Hilbert Space (RKHS). Likewise, let $\mathcal{H}$ be a second RKHS on $\mathcal{Y}$ with kernel $l(\cdot, \cdot)$ and feature map $\psi(y)$. We may now define a cross-covariance operator between these feature maps, in accordance with [Baker, 1973, Fukumizu et al., 2004]: this is a linear operator $\mathcal{C}_{xy} : \mathcal{H} \longmapsto \mathcal{G}$ such that

$$\mathcal{C}_{xy} = \mathsf{E}_{xy}[(\phi(x) - \mu_x) \otimes (\psi(y) - \mu_y)], \tag{4.2}$$

where $\otimes$ is the tensor product. The square of the Hilbert-Schmidt norm of the cross-covariance operator (HSIC), $\|\mathcal{C}_{xy}\|^2_{\mathrm{HS}}$, is then used as our feature selection criterion $\mathcal{Q}(\mathcal{T})$.

[Gretton et al., 2005] show that HSIC can be expressed in terms of kernels as

$$\begin{aligned}&\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy}) = \| \mathcal{C}_{xy} \|^2_{\mathcal{H}} = \\ &= \mathsf{E}_{xx'yy'}[k(x,x')l(y,y')] + \mathsf{E}_{xx'}[k(x,x')]\mathsf{E}_{yy'}[l(y,y')] - 2\mathsf{E}_{xy}[\mathsf{E}_{x'}[k(x,x')]\mathsf{E}_{y'}[l(y,y')]].\end{aligned} \tag{4.3}$$

Previous work used HSIC to *measure* (in)dependence between two sets of random variables [Gretton et al., 2005]. Here we use it to *select* a subset $\mathcal{T}$ from the full set of random variables $\mathcal{S}$. We now describe further properties of HSIC which support its use as a feature selection criterion.

Property (I) [Gretton et al., 2005, Theorem 4] show that whenever $\mathcal{G}, \mathcal{H}$ are RKHSs with universal kernels k, l on respective compact domains $\mathcal{X}$ and $\mathcal{Y}$ in the sense of [Steinwart, 2002] (see Section 3.1.1), then $\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy}) = 0$ if and only if x and y are independent. In terms of feature selection, a universal kernel such as the Gaussian RBF kernel or the Laplace kernel permits HSIC to detect any dependence between $\mathcal{X}$ and $\mathcal{Y}$. HSIC is zero only if features and labels are independent. Clearly we want to reach the opposite result, namely strong dependence between features and class labels. Hence we try to select features that maximize HSIC.

In fact, non-universal kernels can also be used for HSIC, although they may not guarantee that all dependencies are detected. Different kernels incorporate distinctive prior knowledge into the dependence estimation, and they focus HSIC on dependence of a certain type. For instance, a linear kernel requires HSIC to seek only second order dependence. Clearly HSIC is capable of finding and exploiting dependence of a much more general nature by kernels on strings, trees, dynamical systems, and — and most interesting to us — graphs.

Property (II) Given a sample $Z = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of size m drawn from $\Pr_{xy}$, HSIC has an unbiased empirical estimate,

$$\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z) = \tfrac{1}{m(m-3)}[\mathsf{Tr}(\mathbf{KL}) + \tfrac{1}{(m-1)(m-2)}\mathbf{1}^\mathsf{T}\mathbf{K}\mathbf{1}\mathbf{1}^\mathsf{T}\mathbf{L}\mathbf{1} - \tfrac{2}{m-2}\mathbf{1}^\mathsf{T}\mathbf{K}\mathbf{L}\mathbf{1}], \tag{4.4}$$

where $\mathbf{K}$ and $\mathbf{L}$ are computed as $\mathbf{K}_{ij} = (1-\delta_{ij})k(x_i, x_j)$ and $\mathbf{L}_{ij} = (1-\delta_{ij})l(y_i, y_j)$. Note that the diagonal entries of $\mathbf{K}$ and $\mathbf{L}$ are set to zero.

The following theorem formally states that the empirical HSIC is unbiased. This property is by contrast with the mutual information, which requires sophisticated bias correction strategies (e.g. [Nemenman et al., 2002]).

Theorem 42 (HSIC is Unbiased) *Let E_Z denote the expectation taken over m independent observations (x_i, y_i) drawn from $\Pr_{xy}$. Then*

$$\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy}) = \mathsf{E}_Z\left[\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z)\right]. \tag{4.5}$$

Proof [**Theorem 42**] Recall that $\mathbf{K}_{ii} = \mathbf{L}_{ii} = 0$. We prove the claim by constructing unbiased estimators for each term in (4.3). Note that we have three types of expectations, namely $\mathsf{E}_{xy}\mathsf{E}_{x'y'}$, a partially decoupled expectation $\mathsf{E}_{xy}\mathsf{E}_{x'}\mathsf{E}_{y'}$, and $\mathsf{E}_x\mathsf{E}_y\mathsf{E}_{x'}\mathsf{E}_{y'}$, which takes all four expectations independently.

If we want to replace the expectations by empirical averages, we need to take care to avoid using the same discrete indices more than once for independent random variables. In other words, when taking expectations over n independent random variables, we need n-tuples of indices where each index occurs exactly once. The sets $\mathbf{i}_n^m$ satisfy this property, where $\mathbf{i}_n^m$ denotes the set of all n-tuples drawn without replacement from $\{1,\ldots,m\}$. Their cardinalities are given by the Pochhammer coefficients $(m)_n = \frac{m!}{(m-n)!}$. Jointly drawn random variables, on the other hand, share the same index. We have

$$\begin{aligned}\mathsf{E}_{xy}\mathsf{E}_{x'y'}\left[k(x,x')l(y,y')\right] =& \mathsf{E}_Z\left[(m)_2^{-1}\sum_{(i,j)\in\mathbf{i}_2^m}\mathbf{K}_{ij}\mathbf{L}_{ij}\right]\\ =& \mathsf{E}_Z\left[(m)_2^{-1}\mathsf{Tr}\mathbf{KL}\right].\end{aligned}$$

In the case of the expectation over three independent terms $\mathsf{E}_{xy}\mathsf{E}_{x'}\mathsf{E}_{y'}$ we obtain

$$\mathsf{E}_Z\left[(m)_3^{-1}\sum_{(i,j,q)\in\mathbf{i}_3^m}\mathbf{K}_{ij}\mathbf{L}_{iq}\right] = \mathsf{E}_Z\left[(m)_3^{-1}\mathbf{1}^\top\mathbf{KL1} - \mathsf{Tr}\mathbf{KL}\right].$$

For four independent random variables $\mathsf{E}_x\mathsf{E}_y\mathsf{E}_{x'}\mathsf{E}_{y'}$,

$$\begin{aligned}&\mathsf{E}_Z\left[(m)_4^{-1}\sum_{(i,j,q,r)\in\mathbf{i}_4^m}\mathbf{K}_{ij}\mathbf{L}_{qr}\right]\\ =&\mathsf{E}_Z\left[(m)_4^{-1}\left(\mathbf{1}^\top\mathbf{K}\mathbf{1}\mathbf{1}^\top\mathbf{L1} - 4\mathbf{1}^\top\mathbf{KL1} + 2\mathsf{Tr}\mathbf{KL}\right)\right].\end{aligned}$$

To obtain an expression for HSIC we only need to take linear combinations using (4.3). Collecting terms related to $\mathsf{Tr}\mathbf{KL}$, $\mathbf{1}^\top\mathbf{KL1}$, and $\mathbf{1}^\top\mathbf{K}\mathbf{1}\mathbf{1}^\top\mathbf{L1}$ yields

$$\begin{aligned}&\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy})\\ &= \tfrac{1}{m(m-3)}\mathsf{E}_Z\left[\mathsf{Tr}\mathbf{KL} + \tfrac{\mathbf{1}^\top\mathbf{K}\mathbf{1}\mathbf{1}^\top\mathbf{L1}}{(m-1)(m-2)} - \tfrac{2}{m-2}\mathbf{1}^\top\mathbf{KL1}\right].\end{aligned}$$

This is the expected value of $\mathrm{HSIC}[\mathcal{G},\mathcal{H},Z]$.

■

U-Statistics. The estimator in (4.4) can be alternatively formulated using U-statistics

$$\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z) = (m)_4^{-1}\sum_{(i,j,q,r)\in\mathbf{i}_4^m}^{m} h(i,j,q,r), \tag{4.6}$$

where $\mathbf{i}_r^m$ denotes the set of all r-tuples drawn without replacement from $\{1,\ldots,m\}$, and $(m)_n$ are the Pochhammer coefficients.

The kernel h of the U-statistic is defined by

$$\begin{aligned} h(i,j,q,r) =& \tfrac{1}{6}[\mathbf{K}_{ij}(\mathbf{L}_{ij}+\mathbf{L}_{qr}) + \mathbf{K}_{jq}(\mathbf{L}_{jq}+\mathbf{L}_{ir}) \\ &+ \mathbf{K}_{iq}(\mathbf{L}_{iq}+\mathbf{L}_{jr}) + \mathbf{K}_{ir}(\mathbf{L}_{ir}+\mathbf{L}_{jq}) \\ &+ \mathbf{K}_{jr}(\mathbf{L}_{jr}+\mathbf{L}_{qi}) + \mathbf{K}_{qr}(\mathbf{L}_{qr}+\mathbf{L}_{ij})] \\ &- \tfrac{1}{12}\sum_{(t,u,v)}^{(i,j,q,r)} \mathbf{K}_{tu}[\mathbf{L}_{tv}+\mathbf{L}_{uv}] \end{aligned} \tag{4.7}$$

Note that the sum in (4.7) represents all ordered triples (t,u,v) selected without replacement from (i,j,q,r). This can be seen by direct calculation.

We now show that $\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z)$ is concentrated. Furthermore, its convergence in probability to $\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy})$ occurs with rate $1/\sqrt{m}$ which is a slight improvement over the convergence of the biased estimator by [Gretton et al., 2005].

Theorem 43 (HSIC is Concentrated) *Assume k,l are bounded almost everywhere by 1, and are non-negative. Then for $m>1$ and all $\delta>0$, with probability at least $1-\delta$ for all $\Pr_{xy}$*

$$|\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z) - \mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy})| \leq 4.72\sqrt{\log(2/\delta)/m}$$

By virtue of (4.6) we see immediately that HSIC is a U-statistic of order 4, where each term is bounded in $[-2/3, 8/3]$. Applying Hoeffding's bound as in [Gretton et al., 2005] proves the result.

These two theorems imply the empirical HSIC closely reflects its population counterpart. This means the same features should consistently be selected to achieve high dependence if the data are repeatedly drawn from the same distribution.

Asymptotic Normality. It follows from [Serfling, 1980] that under the assumptions $\mathsf{E}(h^2)<\infty$ and that the data and labels are not independent, the empirical HSIC converges in distribution to a Gaussian random variable with mean $\mathrm{HSIC}(\mathcal{G},\mathcal{H},\Pr_{xy})$ and variance

$$\begin{aligned} \sigma^2_{\mathrm{HSIC}} &= \tfrac{16}{m}\left(R - \mathrm{HSIC}^2\right) \text{ where} \\ R &= \tfrac{1}{m}\sum_{i=1}^{m}\Big((m)_3^{-1}\sum_{(j,q,r)\in \mathbf{i}_3^m\setminus\{i\}} h(i,j,q,r)\Big)^2. \end{aligned} \tag{4.8}$$

The asymptotic normality and the variance allow us to formulate statistics for a significance test. This is useful because it may provide an assessment of the functional dependence between the selected features and the labels.

Simple Computation. Note that $\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z)$ is simple to compute, since only the kernel matrices $\mathbf{K}$ and $\mathbf{L}$ are needed, and no density estimation is involved. For feature selection, the kernel matrix on labels $\mathbf{L}$ is fixed through the whole process. It can be precomputed and stored for speedup if needed. Note also that $\mathrm{HSIC}(\mathcal{G},\mathcal{H},Z)$ does *not* need any explicit regularization parameter. This is encapsulated in the choice of the kernels.

4.1.3 Feature Selection via HSIC

Having defined our feature selection criterion, we now describe an algorithm that conducts feature selection on the basis of this dependence measure. The strategy is to greedily select features that maximize HSIC, *i.e.,* the dependence between features and labels.

Using HSIC, we can perform both backward (BAHSIC) and forward (FOHSIC) selection of the features. In particular, when we use a linear kernel on the data (there is no such requirement for the labels), forward selection and backward selection are equivalent: the objective function decomposes into individual coordinates, and thus feature selection can be done without recursion in one go. Although forward selection is computationally more efficient, backward elimination in general yields better features, since the quality of the features is assessed within the context of all other features. Hence we present the backward elimination version of our algorithm here (a forward greedy feature selection algorithm based on HSIC is presented in Section 4.2.3).

BAHSIC appends the features from $\mathcal{S}$ to the end of a list $\mathcal{S}^\dagger$ so that the elements towards the end of $\mathcal{S}^\dagger$ have higher relevance to the learning task. The feature selection problem in (4.1) can be solved by simply taking the last t elements from $\mathcal{S}^\dagger$. Our algorithm produces $\mathcal{S}^\dagger$ recursively, eliminating the least relevant features from $\mathcal{S}$ and adding them to the end of $\mathcal{S}^\dagger$ at each iteration. For convenience, we also denote HSIC as $\mathrm{HSIC}(\sigma, \mathcal{S})$, where $\mathcal{S}$ are the features used in computing the data kernel matrix $\mathbf{K}$, and σ is the parameter for the data kernel (for instance, this might be the width of a Gaussian kernel $k(x, x') = \exp(-\sigma \left\|x - x'\right\|^2)$).

Algorithm 2 BAHSIC

Input: The full set of features $\mathcal{S}$

1: $\mathcal{S}^\dagger \leftarrow \varnothing$
2: **repeat**
3: $\quad\sigma \leftarrow \Xi$
4: $\quad\mathcal{I} \leftarrow \arg\max_{\mathcal{I}} \ \sum_{j\in\mathcal{I}} \mathrm{HSIC}(\sigma, \mathcal{S}\setminus\{j\}), \quad \mathcal{I} \subset \mathcal{S}$
5: $\quad\mathcal{S} \leftarrow \mathcal{S} \setminus \mathcal{I}$
6: $\quad\mathcal{S}^\dagger \leftarrow \mathcal{S}^\dagger \cup \mathcal{I}$
7: **until** $\mathcal{S} = \varnothing$

Output: An ordered set of features $\mathcal{S}^\dagger$

Step 3 of the algorithm denotes a policy for adapting the kernel parameters, e.g. by optimizing over the possible parameter choices. In our experiments, we typically normalize each feature separately to zero mean and unit variance, and adapt the parameter for a Gaussian kernel by setting σ to $1/(2d)$, where $d = |\,\mathcal{S}\,| - 1$. If we have prior knowledge about the type of nonlinearity, we can use a kernel with fixed parameters for BAHSIC. In this case, step 3 can be omitted.

Step 4 of the algorithm is concerned with the selection of a set $\mathcal{I}$ of features to eliminate.

While one could choose a single element of $\mathcal{S}$, this would be inefficient when there are a large number of irrelevant features. On the other hand, removing too many features at once risks the loss of relevant features. In our experiments, we found a good compromise between speed and feature quality was to remove 10% of the current features at each iteration.

4.1.4 Connections to Other Approaches

We now explore connections to other feature selectors. For binary classification, an alternative criterion for selecting features is to check whether the distributions $\Pr(x|y=1)$ and $\Pr(x|y=-1)$ differ. For this purpose one could use Maximum Mean Discrepancy (MMD), as presented in Section 3.1.1 of this thesis. Likewise, one could use Kernel Target Alignment (KTA) [Cristianini et al., 2003] to test directly whether there exists any correlation between data and labels.

Let us consider the output kernel $l(y,y') = \rho(y)\rho(y')$, where $\rho(1) = m_+^{-1}$ and $\rho(-1) = -m_-^{-1}$, and m_+ and m_- are the numbers of positive and negative samples, respectively. With this kernel choice, we show that MMD and KTA are closely related to HSIC.

KTA has been used for feature selection. Formally it is defined as $\mathsf{Tr}\mathbf{KL}/\|\mathbf{K}\|\|\mathbf{L}\|$. For computational convenience the normalization is often omitted in practice [Neumann et al., 2005], which leaves us with $\mathsf{Tr}\mathbf{KL}$. We discuss this unnormalized variant below.

Theorem 44 (Connection to MMD and KTA) *Given a sample $Z = \{(x_1,y_1),\ldots,(x_m,y_m)\}$ of size m. Assume the kernel $k(x,x')$ for the data is bounded and the kernel for the labels is $l(y,y') = \rho(y)\rho(y')$. Then*

$$\left|\mathrm{HSIC} - (m-1)^{-2}\mathrm{MMD}\right| = O(m^{-1})$$
$$\left|\mathrm{HSIC} - (m-1)^{-2}\mathrm{KTA}\right| = O(m^{-1}).$$

Proof [**Theorem 44**] We first relate a biased estimator of HSIC to the biased estimator of MMD. The former is given by

$$\tfrac{1}{(m-1)^2}\mathsf{Tr}\mathbf{KHLH} \text{ where } \mathbf{H} = \mathbf{I} - m^{-1}\mathbf{1}\mathbf{1}^\top$$

and the bias is bounded by $O(m^{-1})$, as shown by [Gretton et al., 2005]. An estimator of MMD with bias $O(m^{-1})$ is

$$\mathrm{MMD}^2(\mathcal{F},Z) = \frac{1}{m_+^2}\sum_{i,j}^{m_+} k(\mathbf{x}_i,\mathbf{x}_j) + \frac{1}{m_-^2}\sum_{i,j}^{m_-} k(\mathbf{x}_i,\mathbf{x}_j) - \frac{2}{m_+ m_-}\sum_{i}^{m_+}\sum_{j}^{m_-} k(\mathbf{x}_i,\mathbf{x}_j) = \mathsf{Tr}\mathbf{KL}$$

If we choose $l(y,y') = \rho(y)\rho(y')$ with $\rho(1) = m_+^{-1}$ and $\rho(-1) = m_-^{-1}$, we can see that $\mathbf{L1} = 0$. In this case $\mathsf{Tr}\mathbf{KHLH} = \mathsf{Tr}\mathbf{KL}$, which shows that the biased estimators of MMD and HSIC are identical up to a constant factor. Since the bias of $\mathsf{Tr}\mathbf{KHLH}$ is $O(m^{-1})$, this implies the same bias for the MMD estimate. Note that Z has slightly different interpretations in MMD and HSIC: MMD is treating the two classes in Z as distinct samples and computes the discrepancy of their means in feature space. HSIC, however, measures dependence

between the features and their class labels, across classes and treating Z as one single sample.

To see the same result for Kernel Target Alignment, note that for equal class size the normalizations with regard to m_+ and m_- become irrelevant, which yields the corresponding MMD term. ■

Theorem 44 means that HSIC converges to $(m-1)^{-2}$MMD and $(m-1)^{-2}$KTA with rate $O(m^{-1})$, hence in some cases in binary classification, selecting features that maximize HSIC also maximizes MMD and KTA. Note that in general (multiclass, regression, or generic binary classification) this connection does not hold.

4.1.5 Variants of BAHSIC

New variants can be readily derived from our framework by combining the two building blocks of BAHSIC: a kernel on the data and another one on the labels. Here we provide three examples using a Gaussian RBF kernel $k(x, x') = \exp(-\sigma\|x - x'\|^2)$ on the data, while varying the kernels on the labels. This provides us with feature selectors for the following problems:

Binary classification (BIN) We set m_+^{-1} as the label for positive class members, and m_-^{-1} for negative class members. We then apply a linear kernel.

Multiclass classification (MUL) We apply a linear kernel on the labels using the label vectors below, as described in a 3-class example. Here m_i is the number of samples in class i and $\mathbf{1}_{m_i}$ denotes a vector of all ones with length m_i.

$$\mathbf{Y} = \begin{pmatrix} \frac{\mathbf{1}_{m_1}}{m_1} & \frac{\mathbf{1}_{m_1}}{m_2-m} & \frac{\mathbf{1}_{m_1}}{m_3-m} \\ \frac{\mathbf{1}_{m_2}}{m_1-m} & \frac{\mathbf{1}_{m_2}}{m_2} & \frac{\mathbf{1}_{m_2}}{m_3-m} \\ \frac{\mathbf{1}_{m_3}}{m_1-m} & \frac{\mathbf{1}_{m_3}}{m_2-m} & \frac{\mathbf{1}_{m_3}}{m_3} \end{pmatrix}_{m\times 3} . \tag{4.9}$$

Regression problem (REG) A Gaussian RBF kernel is also used on the labels. For convenience the kernel width σ is fixed as the median distance between points in the sample [Schölkopf and Smola, 2002].

For the above variants a further speedup of BAHSIC is possible by updating the entries of the kernel matrix incrementally, since we are using an RBF kernel. We use the fact that $\|x - x'\|^2 = \sum_j \|x_j - x'_j\|^2$. Hence $\|x - x'\|^2$ needs to be computed only once. Subsequent updates are effected by subtracting $\|x_i - x'_i\|^2$ (Here subscript indices correspond to dimensions).

4.1.6 Experiments

We conducted experiments on: (*i*) artificial datasets illustrating the properties of BAHSIC; (*ii*) real-world datasets that compare BAHSIC with other methods.

Artificial Datasets

Comparison to Filter Methods We use 3 artificial datasets, as illustrated in Figure 4.1, to compare BAHSIC with other filter methods:

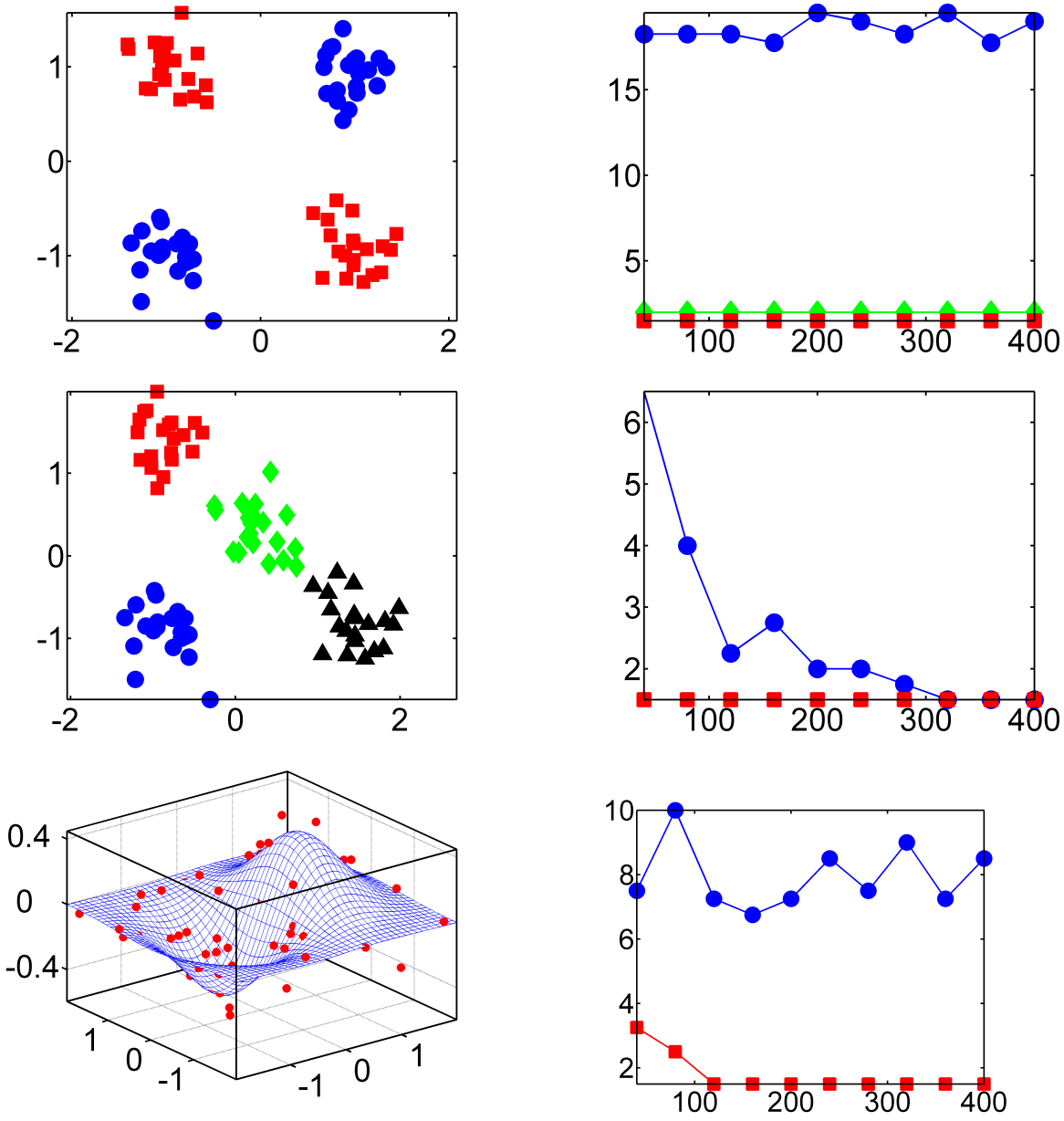

Figure 4.1: Artificial datasets and the performance of different methods when varying the number of observations. **Left column, top to bottom:** Binary, multiclass, and regression data. Different classes are encoded with different colors. **Right column:** Median rank (y-axis) of the two relevant features as a function of sample size (x-axis) for the corresponding datasets in the left column (Pearson's correlation (blue circle), RELIEF (green triangle), BAHSIC (red square). Note that RELIEF only works for binary classification.).

- Binary XOR data, where samples belonging to the same class have multimodal distributions.
- Multiclass data, where there are 4 classes but 3 of them are collinear.
- Nonlinear regression data, where the label is related to the first 2 dimension of the data by $y = x_1 \exp(-x_1^2 - x_2^2) + \epsilon$. Here ϵ denotes additive normal noise.

Each dataset has 22 dimensions: the first two dimensions are nonlinearly interacting features and the rest are just Gaussian noise.

We compare the three variants of BAHSIC (BIN, MUL and REG) to two other filter methods, namely Pearson's correlation and RELIEF [Kira and Rendell, 1992]. Note that we will mainly focus on comparison with Pearson's correlation, since it is applicable to both classification and regression problems (the original RELIEF works only for binary problem). We aim to show that when nonlinear dependencies exist in the data, BAHSIC is very competent in finding them.

We instantiate the artificial datasets over a range of sample sizes (from 40 to 400), and plot the median rank, produced by various methods, for the first two dimensions of the data. All numbers in Figure 4.1 are averaged over 10 runs. In all cases, BAHSIC shows good performance. More specifically, we observe that

- On the XOR problem, both BIN and RELIEF correctly select the first two dimensions of the data even for small sample sizes. Pearson's correlation, however, fails. This is because the latter evaluates the goodness of each feature independently. Hence it is unable to capture nonlinear interaction between features.
- In the multiclass problem, MUL selects the correct features irrespective of the size of the samples. Pearson's correlation only works for large sample size. The collinearity of 3 classes provides linear correlation between the data and the labels, but due to the interference of the fourth class such correlation is picked up by Pearson's correlation only for large sample size.
- For regression, the performance of Pearson's correlation is just slightly better than random. REG quickly converges to the correct answer as the sample size increases.

While this does not prove that BAHSIC is always better than other methods in practice, it illustrates that when nonlinearity exists, BAHSIC is able to detect it. This is obviously useful in a real-world situations. The second advantage of BAHSIC is that it is readily applicable to both classification and regression problems, by simply choosing a different kernel.

Embedded and Wrapper Methods In this experiment, we show that the performance of BAHSIC can be comparable to embedded and wrapper methods. We use the artificial data described by [Weston et al., 2000] to compare BAHSIC to 4 embedded and wrapper feature selection methods implemented in the Spider[1] Toolbox: namely FSV

[1] http://www.kyb.tuebingen.mpg.de/bs/people/spider

Method	BAHSIC	FSV	L0	R2W2	RFE
WL-3	**0.0±0.0**	*2.0±2.0*	**0.0±0.0**	**0.0±0.0**	**0.0±0.0**
WN-2	**1.0±1.0**	58.0±5.3	*2.0±1.3*	54.0±6.5	*2.0±1.3*

Table 4.1: Classification error (%) after selecting features using BAHSIC and other methods.

[Bradley and Mangasarian, 1998], L0-norm SVM [Weston et al., 2003], R2W2 [Weston et al., 2000] and SVM-RFE [Guyon et al., 2002].

For the linear problem, 25 data points of 202 dimensions are generated for each class, of which only the first six dimensions are relevant for classification (WL-3). Six features are selected and classified using linear SVM. For the nonlinear problem, 50 data points of 52 dimensions are generated for each class, and only the first two dimensions are relevant (WN-2). Two features are selected and an SVM with polynomial kernel of degree 2 is used for the classification. All results presented in Table 4.1 are obtained using 10-fold cross-validation. It can be seen that BAHSIC compares favourably to embedded and wrapper methods in small sample size and nonlinear problems.

Method	BAHSIC	Pearson	RELIEF	FSV	L0	R2W2	RFE
Covertype (b)	**36.3±1.6**	43.3±1.6	41.0±1.6	44.2±1.4	45.6±0.5	45.9±0.3	41.1±1.2
Ionosphere (b)	**11.8±1.5**	22.5±1.8	14.1±2.3	15.7±1.7	35.9±0.5*	12.8±1.4	30.0±1.0*
Sonar (b)	28.2±2.9	26.9±2.4	**22.7±2.7**	28.8±1.9	40.4±1.8*	33.7±2.30	26.3±3.9
Satimage (m)	**20.1±1.9**	53.9±6.7	-	-	25.0±1.0	-	22.6±1.2
Segment (m)	24.5±1.2	**5.3±3.1**	-	-	71.2±6.7*	-	31.2±0.6
Vehicle (m)	35.4±1.9	35.5±6.8	-	-	44.2±1.7	-	42.8±1.7
Housing (r)	**19.1±2.7**	27.5±3.0	-	-	-	-	-
Bodyfat (r)	9.4±3.1	9.3±3.3	-	-	-	-	-
Abalone (r)	55.1±2.7	54.2±3.3	-	-	-	-	-

Table 4.2: Classification error (%) or percentage of variance *not*-explained (%). Best results are shown in boldface, as long as the advantage is statistically significant (left-tailed t-test with level 0.05). b: binary problem; m: multiclass problem; r: regression problem. -: not applicable.

Real-World Datasets

We now discuss results on various real-world datasets taken from the UCI repository and Statlib. We chose the covertype, ionosphere, and sonar datasets for binary classification; satimage, segment, and vehicle for multiclass classification; and housing, bodyfat, and abalone for regression. We reduced the size of some datasets to smaller than 1000 by a balanced random sample of data. These reduced datasets enable us to compare with wrapper and embedded methods in reasonable time.

We report the performance of an SVM using an RBF kernel on a feature subset of size 5 using 10-fold cross-validation. These 5 features were selected per fold using different methods. On classification datasets, we measured the performance using the balanced error-rate metric, and on regression datasets we used the percentage of variance *not*-explained (also known as $1 - r^2$).

The results of the experiments are summarised in Table 4.1.6. Here we see that the performance of BAHSIC competes favourably across all three types of problems. Note these embedded and wrapper methods are not applicable for regression problems. Even for classification, however, they generally do not perform as well as simple filter methods such as BAHSIC or RELIEF (especially for the numbers marked with *). While Pearson's correlation also has broad applicability, it does not perform as well as BAHSIC in general.

4.1.7 Summary

In this section, we have proposed a backward elimination procedure for feature selection using the Hilbert-Schmidt Independence Criterion (HSIC). The idea behind the resulting algorithm, BAHSIC, is to choose the feature subset that maximizes the dependence between the data and labels. With this interpretation, BAHSIC provides a unified feature selection framework for any form of supervised learning. The absence of bias and good convergence properties of the empirical HSIC estimate provide a strong theoretical justification for using HSIC in this context. Although BAHSIC is a filter method, it still demonstrates good performance compared with more specialised methods in both artificial and real-world data.

4.2 Feature Selection among Frequent Subgraphs

BAHSIC offers us a theoretically justified and practically successful feature selection method based on dependence maximization and kernels. The fact that BAHSIC is a kernel method also means that we could directly extend it to graphs. However, there is a huge rub, the large number of features included in a graph model. If we want to apply BAHSIC to graphs we have to find the subgraph whose removal lowers dependence between graphs and their class labels least in each iteration. However, even the effort of enumerating these subgraphs is NP-hard. This also means that in each iteration we have to recompute a graph kernel matrix exponentially often, once for each subgraph. Despite all advances in graph kernel computation efficiency, this seems an rather hopeless endeavor concerning computational runtime.

Feature Selection Needs Frequent Subgraph Mining However, the core of this problem lies in the fact that we are enumerating all subgraphs. Is this really necessary? It seems plausible to exclude certain subgraphs, e.g. those that appear only in a negligible fraction of the graphs in the dataset. In other terms, to consider only subgraphs that appear in at least t of the graphs in the dataset. This is exactly the definition of a *frequent subgraph* in graph mining. Efficient algorithms have been developed for frequent subgraph mining, most prominently gSpan [Yan and Han, 2002]. gSpan uses elegant data structures and branch-and-bound search strategies to lower the computational burden (but of course, it cannot avoid the problem of repeated isomorphism checking which is NP-hard). Hence for feature selection on graphs, restricting ourselves to frequent subgraphs seems attractive.

Frequent Subgraph Mining Needs Feature Selection Still, frequent subgraphs are not necessarily those that help to distinguish different classes of graphs. High-frequency patterns may appear in all graphs and exert little discriminative power. That is the reason why t is often set usually set very low in frequent subgraph mining for classification tasks. The resulting low-frequency patterns, however, may be so rare that they do not help to tell apart classes either, or they may even represent noise. Furthermore, for small choices of t, frequent subgraph mining detects thousands and millions of subgraphs. The sheer amount of these frequent subgraphs makes it difficult to identify the most informative ones for class membership. Even worse, the frequent subgraphs are highly correlated, as they may be subgraphs of each other. As a consequence, frequent subgraph mining would benefit from an algorithm that allows efficient feature selection among its frequent subgraph patterns and takes correlation between features into account.

In this section, we will a present an algorithm that uses BAHSIC-like feature selection on frequent subgraphs detected by gSpan. It is the first principled feature selection technique for graphs and an extension of gSpan that reduces gSpan's vast solution set to a small set of highly informative subgraph features.

Frequent Subgraph Selection for Graph Classification

In previous sections, we were designing graph kernels for tackling classification problems on datasets of graphs. The graph kernels we considered so far compare graphs by a pairwise comparison of *all* their substructures. The reasoning behind this approach is that by

looking at all substructures, we are more likely to find the features that are associated with class membership.

However, considering all features, *i.e.,* all subgraphs, in a graph is computationally expensive, as enumeration of all subgraphs is NP-hard. Furthermore, their number grows exponentially with the size of a graph and with the number of its edges. Researchers on graph kernels and on frequent subgraph mining have taken two different roads to overcome this problem: On the one hand, state-of-the-art graph kernels, as reviewed in Section 1.4 and defined in Section 2, restrict their comparison of graphs to substructures that can be computed in polynomial time. On the other hand, frequent subgraph mining is considering only those substructures that are frequent within a given dataset for comparison [Kramer et al., 2001, Deshpande et al., 2005, Cheng et al., 2007]. [2]

A major setback of these frequent subgraph based methods is that they tend to detect vast numbers of frequent subgraphs, because the frequency threshold has to be set very low to discover discriminative subgraphs. This vast number of features poses three new challenges.

1. **Redundancy**: Most frequent substructures are only slightly different from each other in structure and are highly correlated with the class labels.

2. **Significance**: While low-support features are not representative of the graphs, frequent subgraphs are not necessarily useful for classification. Statistical significance rather than frequency of the graph patterns should be used for evaluation of their discriminative power.

3. **Efficiency**: Very frequent subgraphs are not useful since they are not discriminative between classes. Therefore, frequent subgraph based classification usually sets up a pretty low frequency threshold, resulting in thousands or even millions of features. Given such a tremendous number of features, a complicated feature selection mechanism is likely to fail.

Consequently, we need an efficient algorithm to select informative, discriminative features among a large number of frequent subgraphs. In earlier work [Cheng et al., 2007], Cheng et al. adopted a heuristic approach to this problem and demonstrated that it could outperform methods using low-frequency features.

In this section, we will define an efficient and principled approach to feature selection among frequent subgraphs generated by gSpan [Yan and Han, 2002], a the state-of-the-art tool for frequent subgraph mining. In order to select the subgraphs that are most discriminative for classification, we will employ the kernel-based feature selection algorithms based on the Hilbert-Schmidt Independence Criterion (HSIC), as presented in Section 4.1.

Unlike its predecessors which use ad-hoc strategies for feature selection, we define principled backward and forward feature selection methods based on the Hilbert-Schmidt Independence Criterion, and its associated kernel functions. Specifically, we show that forward

[2] Interestingly, even the latter could be deemed R-convolution kernels, and hence graph kernels. The only difference to state-of-the-art graph kernels is that the decomposition R of these kernels would be NP-hard to compute, as they are determining frequent subgraphs.

feature selection with one particular kernel leads to a feature selector that is efficient even on vast numbers of subgraph features: it uses an intuitive selection criterion solely based on the frequency of subgraphs in different classes. Furthermore, it achieves higher classification accuracy than the competitors in our experimental evaluation.

4.2.1 Preliminaries

In the following we will first define some additional notation, then the optimization problem we want to tackle and finally we will review gSpan, the frequent subgraph mining approach our feature selector builds upon.

Notation We begin with some additional notation. We are given a dataset $D = \{(G_1, y_i), \ldots, (G_m, y_m)\}$ of graphs that each belong to one of two classes A and B: $G_i \in A$ if $y = 1$, or $G_i \in B$ of $y = -1$. Let $|A| = |B| = \frac{1}{2}m$ be of identical size. Our presentation also applies to the unbalanced case ($|A| \neq |B|$), but for the sake of clarity of presentation, we present the simple balanced case here, and summarize the general unbalanced case in Section 4.2.3.

Given a graph database D, $|D_G|$ is the number of graphs in D where G is a subgraph. $|D_G|$ is called the *(absolute) support*, denoted by $support(G)$. A graph G is *frequent* if its support is no less than a minimum support threshold, min_sup. As one can see, the frequent graph is a relative concept: whether a graph is frequent depends on the value of min_sup.

Combinatorial Optimization Problem

As feature selection in general (see Section 4.1.1), the problem of feature selection among frequent subgraphs can be cast as a combinatorial optimization problem. We denote by $\mathcal{S}$ the full set of features, which in our case corresponds to the frequent subgraphs generated by gSpan. We use these features to predict class membership of individual graph instances: clearly, only a subset $\mathcal{T}$ of features will be relevant. Supported by the statistical properties of $HSIC$ (see Section 4.1.2), we propose to quantify the relevance of a set of frequent subgraphs for class membership by $HSIC(\mathcal{T})$. By slight abuse of notation, we will use (sets of) features as arguments of $HSIC$, not RKHSs and probability distributions as in Section 4.1. This shall reflect the fact that HSIC is measuring the quality of features in the current setting. $HSIC(\mathcal{T})$ is computed by restricting the graphs to the features in $\mathcal{T}$. Feature selection can then be formulated as:

$$\mathcal{T}_0 = \arg\max_{\mathcal{T} \subseteq \mathcal{S}} \; HSIC(\mathcal{T}) \qquad \text{s.t.} \quad |\mathcal{T}| \leq \theta \tag{4.10}$$

where $|\cdot|$ computes the cardinality of a set and θ upper bounds the number of selected features. Unfortunately, solving this problem optimally requires us to search all possible subsets of features (up to cardinality θ) exhaustively. Thus, we have to resort to greedy alternatives, as in Section 4.1.

gSpan

As considering *all* subgraphs is prohibitively expensive, we focus on feature selection on *frequent* subgraphs. However, before we can select these frequent subgraphs, we have to find

them. We employ gSpan for this purpose, the state-of-the-art tool for frequent subgraph mining. We summarize gSpan's main concepts in the following.

The discovery of frequent graphs usually consists of two steps. In the first step, we generate frequent subgraph candidates, while in the second step, we check the frequency of each candidate. The second step involves a subgraph isomorphism test, which is NP-complete. Fortunately, efficient isomorphism testing algorithms have been developed, making such testing affordable in practice. Most studies of frequent subgraph discovery pay attention to the first step; that is, how to generate as few frequent subgraph candidates as possible, and as fast as possible.

The initial frequent graph mining algorithms, such as AGM [Inokuchi et al., 2000], FSG [Kuramochi and Karypis, 2001] and the path-join algorithm [Vanetik et al., 2002], share similar characteristics with the Apriori-based itemset mining [Agrawal and Srikant, 1994]. All of them require a join operation to merge two (or more) frequent substructures into one larger substructure candidate. To avoid this overhead, non-Apriori-based algorithms such as gSpan [Yan and Han, 2002], MoFa [Borgelt and Berthold, 2002], FFSM [Huan et al., 2003], and Gaston [Nijssen and Kok, 2004] adopt the pattern-growth methodology, which attempts to extend graphs from a single subgraph directly. For each discovered subgraph S, these methods add new edges recursively until all the frequent supergraphs of S are discovered. The recursion stops once no frequent graph can be generated any more.

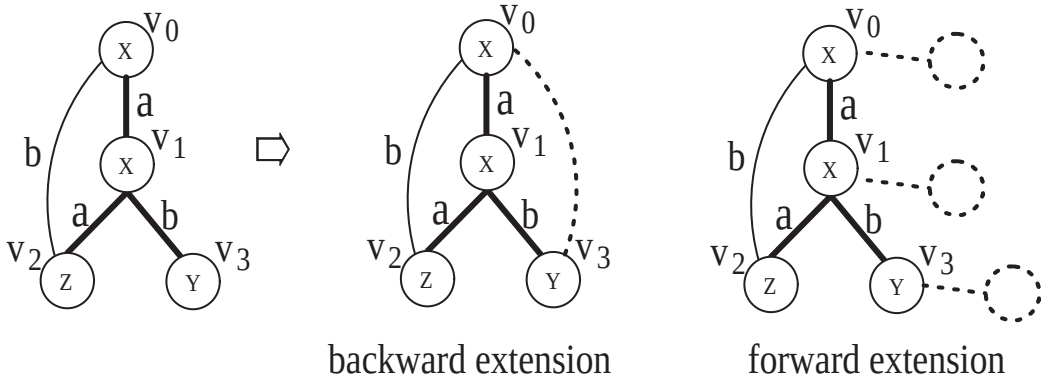

Figure 4.2: gSpan: Rightmost Extension.

gSpan introduced a sophisticated extension method, which is built on depth first search (DFS) tree. Given a graph S and a DFS tree TR, we call the starting node in TR, v_0, the *root*, and the last visited node, v_n, the *rightmost node*. The straight path from v_0 to v_n is called the *rightmost path*. Figure 4.2 shows an example. The darkened edges form a DFS tree. The nodes are discovered in the order v_0, v_1, v_2, v_3. The node v_3 is the rightmost node. The rightmost path is $v_0 \sim v_1 \sim v_3$.

The new method, called rightmost extension, restricts the extension of new edges in a graph as follows: Given a graph S and a DFS tree TR, a new edge e can be added between the rightmost node and other nodes on the rightmost path (*backward extension*); or it can

introduce a new node and connect to nodes on the rightmost path (*forward extension*). If we want to extend the graph in Figure 4.2, the backward extension candidate can be (v_3, v_0). The forward extension candidates can be edges extending from v_3, v_1, or v_0 with a new node introduced. Since there could be multiple DFS trees for one graph, gSpan establishes a set of rules to select one of them as representative so that the backward and forward extensions will only take place on one DFS tree.

Overall, new edges are only added to the nodes along the rightmost path. With this restricted extension, gSpan reduces the generation of the same graphs. However, it still guarantees the completeness of enumerating all frequent subgraphs. For a detailed description of gSpan, see [Yan and Han, 2002]. Algorithm 3 outlines the pseudocode of gSpan. $S \diamond_r e$ means that an edge is extended from graph S using backward or forward extension. $S \neq dfs(S)$ check whether S has been discovered before, where $dfs(S)$ is the canonical form of graph S [Yan and Han, 2002].

Algorithm 3 gSpan(S, D, min_sup, $\mathcal{S}$)

Input: A feature subgraph S, a graph dataset D, and min_sup.

1: **if** $S \neq dfs(S)$, **then**
2: **return**;
3: insert S into $\mathcal{S}$;
4: set C to $\varnothing$;
5: scan D once, find all the edges e such that S can be
rightmost extended to $S \diamond_r e$;
insert $S \diamond_r e$ into C and count its frequency;
6: **for each** frequent $S \diamond_r e$ in C **do**
7: Call gSpan($S \diamond_r e$, D, min_sup, $\mathcal{S}$);
8: **return**;

Output: The frequent graph set $\mathcal{S}$.

4.2.2 Backward Feature Elimination via HSIC

After summarizing essential prerequisites of our approach, we want to find a solution to the optimization problem in (4.10) next. While exhaustive search over all subsets of $\mathcal{S}$ will give us the optimal answer, this approach is not computationally feasible, as the search space grows exponentially with the size of $\mathcal{S}$. As before, we therefore consider two common greedy approaches to feature selection instead: backward elimination and forward selection. In the present section, we will show how to perform backward elimination using HSIC and graph kernels, and analyze whether it is feasible in practice. In the next section, we design a forward selection algorithm that works even more efficiently on large datasets.

A greedy approach to finding an approximate solution to problem (4.10) would be to apply backward feature elimination to graphs: check for each subgraph $S \in \mathcal{S}$ whether its

removal from the dataset lowers the HSIC. If it lowers HSIC, then the dependence between graphs and their class labels is lowered by the removal; hence this subgraph is informative for classification. If HSIC stays roughly unchanged, this subgraph is rather uninformative for distinguishing the two classes. BAHSIC iteratively removes the feature or set of features whose removal reduces HSIC least.

The interesting question is: How to remove a feature, *i.e.,* a subgraph, from a graph? If you represent each graph as an indicator vector, in which the d-th component indicates whether this graph contains the d-th frequent subgraph, then removing a feature is the same as removing one component from all these indicator vectors. Then, however, you run into a problem: by removing a subgraph feature S from a graph you also remove all its subgraphs. If you only remove the component representing S from the indicator vectors, but do not change those representing its subgraphs, then you ignore this relevant fact completely.

Here we propose an approach to overcome this problem. Instead of representing each graph by an indicator vector, we really represent it as a graph. If we want to remove a subgraph S, we delete it from all graphs in which it occurs. This means that we delete all edges in this subgraph S in all graph instances in D. In other terms, we only keep the "complement graphs of D" with respect to S. We will denote these by $D \setminus S$ in the following.

To compute HSIC on the graphs $D \setminus S$ and on the class labels $\mathcal{Y}$, we need a graph kernel for the graphs, and a vector kernel for the class labels. The natural choice for the class labels is to check them for identity. This can be achieved via a so-called delta kernel l defined on a pair of graphs G_i and G_j:

$$l(G_i, G_j) = \begin{cases} 1 & \text{if } y_i = y_j \ , \\ 0 & \text{otherwise} \end{cases}$$

where y_i and y_j are the class labels of graphs G_i and G_j, respectively.

For the graphs, we can pick any of the graph kernels defined in the literature, as reviewed in Section 1.4, or as proposed in Section 2.1 of this thesis.

These will measure topological similarity between the graphs in $D \setminus S$; or more intuitively, how similar the graphs in D are if we ignore all subgraphs S. However, there is a problem: assume we have $|\,\mathcal{S}\,|$ frequent subgraphs and m graphs. Then we have to compute $|\,\mathcal{S}\,|$ graph kernel matrices for m graphs, because for each frequent subgraph S, the set of complement graphs $D \setminus S$ changes. This procedure has to be repeated recursively, until we only have a certain pre-determined number of subgraphs θ left.

This backward elimination causes huge runtime problems because graph kernels are not fast enough for thousands of graphs. Assume that we are dealing with $10,000$ frequent subgraphs on a dataset of 100 graphs. In the first iteration, we have to compute a kernel matrix on D for all $10,000$ frequent subgraphs. Even our fast graph kernels take roughly 1 minute (see Section 2) for comparing 100 graphs. Hence computing one 100×100 graph kernel matrix for each of $10,000$ subgraphs will roughly require $10,000$ minutes - which means we need one week for the first iteration of our backward elimination algorithm! Hence we have to define a feature selection approach that avoids these costly computations.

We draw two conclusions from this observation: first, we have to avoid recomputing graph kernel matrices for each of the numerous frequent subgraphs. Second, forward selection might be more attractive than backward elimination in this setting, as the number of features is very large in our problem.

4.2.3 Forward Feature Selection via HSIX

Since backward elimination and standard graph kernels are too expensive, we next define a forward approach to frequent subgraph feature selection that uses a fast kernel on graphs. We have two main goals: a) to design a kernel that can be evaluated extremely quickly, b) to design a kernel such that HSIC combined with this kernel is an intuitive measure for dependence between graphs and their class labels.

Challenges in Forward Selection

Forward selection starts by determining the frequent subgraph, *i.e.,* the feature, with maximum HSIC score.

$$\max_{S \in \mathcal{S}} HSIC(S) \tag{4.11}$$

where HSIC(S) denotes the HSIC value for subgraph S, which is computed as follows: As shown in the section 4.1, an empirical estimate of HSIC can be computed in terms of a kernel matrix on graphs $\mathbf{K}$ and a kernel matrix on class labels $\mathbf{L}$. If we compute HSIC for one subgraph S, then $\mathbf{L}$ remains unchanged. But for computing $\mathbf{K}$ on the graphs from D, we now consider one single feature, namely only subgraph S and no other subgraph. As usually in forward feature selection, in the first iteration, we evaluate our feature selection criterion HSIC for each feature individually.

If objects are vectors, this means that we consider one component of the vectors only. If objects are graphs and features are subgraphs, as in our case, then we represent each graph by one subgraph feature S only. This means that we check for each graph in D if it contains S. We remove all edges from each graph except for those that are part of a subgraph isomorphic to S. After this "reduction" of the dataset D, we have to compute a graph kernel matrix on the remaining graphs. If we employ a graph kernel from the literature for this task, we will run into runtime problems again: We have to compute one graph kernel matrix for each subgraph feature S, which is beyond the scope of state-of-the-art graph kernels when dealing with tens of thousands of subgraphs.

However, there is one particular kernel for comparing graphs, which is simple, yet intuitive, and — combined with gSpan — efficient enough for HSIC computations even on thousands and millions of subgraphs. We will define and describe this kernel in the next section.

HSIC as a Frequency Criterion

A biased empirical estimator for HSIC in terms of two $m \times m$ kernel matrices $\mathbf{K}$ and $\mathbf{L}$ on features and labels can be obtained as [Gretton et al., 2005][3]

$$(m-1)^{-2}\mathrm{Tr}(\mathbf{KHLH}) = (m-1)^{-2}\sum_{i=1}^{m}\sum_{j=1}^{m}\mathbf{K}_{ij}[\mathbf{HLH}]_{ij},$$

where $\mathbf{H}_{ij} = \delta_{ij} - m^{-1}$ centers the kernel matrices $\mathbf{K}$ and $\mathbf{L}$, and $\mathbf{K}_{ij}$ and $[\mathbf{HLH}]_{ij}$ is the entry in row i and column j in $\mathbf{K}$ and $\mathbf{HLH}$, respectively.

The delta kernel matrix $\mathbf{L}$ on the class labels has to be evaluated only once, therefore it is not that decisive for runtime. Hence we decide to employ the delta kernel l described before, that checks class labels of two graphs G_i and G_j for identity.

$$l(G_i, G_j) = \begin{cases} 1 & \text{if } y_i = y_j \\ 0 & \text{otherwise} \end{cases}.$$

As $\mathbf{H}$ and $\mathbf{L}$ are fixed, we can precompute their product, which is also constant across repeated evaluations of HSIC. Straightforward matrix multiplication then tells us that

$$\mathbf{HLH}_{ij} = \begin{cases} 0.5 & \text{if } y_i = y_j \\ -0.5 & \text{otherwise} \end{cases}, \tag{4.12}$$

where y_i and y_j are the class labels of graphs G_i and G_j, respectively.

We have to be able to evaluate the kernel matrix $\mathbf{K}$ very efficiently, as we need HSIC values for each of our vast amount of subgraph features. We have repeatedly stressed that a graph kernel that operates on graph structures will be too slow for this task. For this reason, we suggest to represent each graph by an indicator vector of length $\mathcal{S}$:

Definition 45 (Indicator Vector) *Given a graph G from a dataset D and a set of frequent subgraph features $\mathcal{S}$ discovered by gSpan. We then define an indicator vector $v(G)$ as*

$$v(G)_d = \begin{cases} 1 & \textit{if } S_d \sqsubseteq G, \\ 0 & \textit{otherwise} \end{cases} \tag{4.13}$$

where $v(G)_d$ is the d-th component of $v(G)$ and S_d is the d-th subgraph feature in $\mathcal{S}$. Alternatively, we will refer to $v(G)_d$ as $v_{S_d}(G)$.

To compare two graphs, we now employ a linear kernel d on their indicator vectors:

$$k(G_i, G_j) = \langle v(G_i), v(G_j) \rangle \tag{4.14}$$

[3] We will neglect the constant factor $(m-1)^2$ in HSIC in the rest of this section, as it does not affect the solution.

Note that if gSpan would not precompute the indicator vectors for us, computing this seemingly simple kernel would be extremely expensive, as the indicator vectors themselves are NP-hard to compute.

In the first iteration of forward selection, we look at each subgraph feature S_d individually. Hence we only consider the d-th entry of the indicator vector, *i.e.,* $v_{S_d}(G)$ for all $G \in D$.

Then the linear kernel on these 1-dimensional vectors can then be written as:

$$\begin{aligned} k(v_{S_d}(G_i), v_{S_d}(G_j)) &= v_{S_d}(G_i) * v_{S_d}(G_j) = \\ &= \begin{cases} 1 & \text{if } S_d \sqsubseteq G_i \text{ and } S_d \sqsubseteq G_j, \\ 0 & \text{otherwise} \end{cases} \\ &= k_{S_d}(G_i, G_j), \end{aligned} \tag{4.15}$$

where the term in the last line is introduced for notational convenience.

Now we can obtain the HSIC for one single subgraph feature S_d, denoted HSIC(S_d), as follows:

$$\text{HSIC}(S_d) = \text{Tr}(\mathbf{KHLH}) = \tag{4.16}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{K}_{ij} [\mathbf{HLH}]_{ij} \tag{4.17}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{k}(v_{S_d}(G_i), v_{S_d}(G_j)) [\mathbf{HLH}]_{ij} \tag{4.18}$$

$$= \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{k}_{\mathbf{S_d}}(G_i, G_j) [\mathbf{HLH}]_{ij} \tag{4.19}$$

where G_i and G_j are graphs from D. Due to Equations (4.12) and (4.19), we can now show the following theorem:

Theorem 46 *Let S_d, D, class A, and class B be defined as before. Let a_{S_d} be the number of graphs in class A that contain S_d as a subgraph. Let b_{S_d} be the number of graphs in class B that contain S_d as a subgraph. Then* HSIC(S_d) *can be computed as*

$$\text{HSIC}(S_d) = 0.5 a_{S_d}^2 + 0.5 b_{S_d}^2 - a_{S_d} b_{S_d} \tag{4.20}$$

$$= 0.5 (a_{S_d} - b_{S_d})^2 \tag{4.21}$$

Proof A summand $\mathbf{k}_{\mathbf{S_d}}(G_i, G_j)[\mathbf{HLH}]_{ij}$ from Equation 4.19 can only be 1 (and not 0), if G_i and G_j both have S_d as a subgraph. There are $(a_{S_d} + b_{S_d})^2 = a_{S_d}^2 + 2 a_{S_d} b_{S_d} + b_{S_d}^2$ pairs of graphs that both contain S_d. Due to Equation (4.12), pairs of graphs from the same class (either both A or both B) get a weight of 0.5, while pairs of graphs from different classes get a weight of -0.5. It can thus be seen from Equation (4.19) that HSIC(S_d) sums up to $0.5 a_{S_d}^2 - a_{S_d} b_{S_d} + 0.5 b_{S_d}^2 = 0.5 (a_{S_d} - b_{S_d})^2$. ■

To summarize, by computing $(a_{S_d} - b_{S_d})^2$ (dropping the constant factor 0.5) we get the HSIC value for one frequent subgraph S. Hence we have reached the two goals of this section: First, our kernel can be computed efficiently, as it only checks for co-occurrence of subgraphs in two graph instances. Second, HSIC combined with this kernel boils down to a frequency-based criterion for subgraph feature selection.

HSIC-based Correlation Scores for Sets of Subgraphs

Now we know how to get one HSIC value per frequent subgraph efficiently. But how to select an informative *set* of several subgraph features?

The top x % of individually highest scoring subgraphs is not necessarily a good choice because

- they might occur in the same graphs from D,
- they might be subgraphs of each other,

and as a consequence, they might miss out on the same set of graphs. Hence the combination of two top-scoring subgraph features may not be more informative than the single features. In other terms, the top scoring subgraphs might not 'complement' each other very well.

Unfortunately, HSIC with the linear kernel on indicator vector as defined above suffers from these problems. Assume that we are trying to select pairs of subgraph features S and S'. Hence we are now looking at the pair of entries in the indicator vectors that represent subgraph features S and S'. We denote the linear kernel on these vectors of length 2 by $k_{S \vee S'}$.

Then HSIC for the combination of $S \vee S'$ is defined as:

$$\begin{aligned} \mathrm{HSIC}(S \vee S') &= \sum_{i=1}^{m} \sum_{j=1}^{m} \mathbf{k}_{\mathbf{S} \vee \mathbf{S}'}(G_i, G_j)[\mathbf{HLH}]_{ij} && (4.22) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} (\mathbf{k}_{\mathbf{S}}(G_i, G_j) + \mathbf{k}_{\mathbf{S}'}(G_i, G_j))[\mathbf{HLH}]_{ij} && (4.23) \\ &= \mathrm{HSIC}(S) + \mathrm{HSIC}(S') && (4.24) \end{aligned}$$

The transition from (4.22) to (4.23) is simply a consequence of the fact that we are using a linear kernel.

HSIC is hence additive for our particular choice of kernel. This, however, causes problems, as can be easily seen from a simple example. Assume that S and S' are frequent subgraphs in D such that $S \sqsubseteq S'$ and S and S' occur in exactly the same instances of D. Hence $a_S = a_{S'}$ and $b_S = b_{S'}$, and thus $\mathrm{HSIC}(S) = \mathrm{HSIC}(S')$. However, HSIC of S and S' will be $\mathrm{HSIC}(S \vee S') = \mathrm{HSIC}(S) + \mathrm{HSIC}(S') = 2\mathrm{HSIC}(S)$. HSIC would deem S and S' together twice as informative as each of them individually — although they occur in exactly the same graph instances, and their union is not more helpful for discriminating classes than each of them alone.

We avoid these problems by defining an HSIC-based correlation score (HSICCS, or HSIX in short) for a union of two subgraph features S and S'. The key idea is that their common HSIX value should be larger, if they occur in different instances of D. HSIX uses a combination of HSIC values to assess the informativeness of a subgraph, while not being an instance of HSIC itself.

Let S and S' be two frequent subgraphs of graphs from D, *i.e.,* two features in our feature selection process. Our HSIC-based correlation score (HSIX) of the union of two frequent subgraphs S and S' is then defined as

$$HSIX(S \vee S') = \text{HSIC}(S) + \text{HSIC}(S') - \text{HSIC}(S \wedge S'),$$

where $\text{HSIC}(S)$ is defined as

$$\text{HSIC}(S) = (a_S - b_S)^2,$$

where a_S is the frequency of S in A, and b_S is the frequency of S in B.

Analogously, $\text{HSIC}(S')$ is defined as

$$\text{HSIC}(S') = (a_{S'} - b_{S'})^2,$$

where $a_{S'}$ is the frequency of S' in A, and $b_{S'}$ is the frequency of S' in B.

$\text{HSIC}(S \wedge S')$ is defined as

$$\text{HSIC}(S \wedge S') = (a_{S \wedge S'} - b_{S \wedge S'})^2,$$

where $a_{S \wedge S'}$ is the frequency of S and S' occurring simultaneously in the same graph in A, and $b_{S \wedge S'}$ is the frequency of S and S' occurring simultaneously in the same graph in B. Note that $\text{HSIC}(S \wedge S')$ could be written in terms of kernels on indicator vectors as

$$\sum_{i=1}^{m} \sum_{j=1}^{m} (\mathbf{k_S}(G_i, G_j)\mathbf{k_{S'}}(G_i, G_j))[\mathbf{HLH}]_{ij} \tag{4.25}$$

Using the HSIX formula for the union of two subgraphs, we can discover pairs of subgraphs that jointly lead to a high HSIX value. Note that obviously $HSIX(S) = HSIC(S)$ if we are looking at a single subgraph feature S.

For selecting more than 2 subgraphs, we can apply the above scheme iteratively. Assume that $\mathcal{T}$ is the set of subgraph features that have been selected so far, and $HSIX(\mathcal{T})$ the associated HSIX value. Then the HSIX value of the union of $\mathcal{T}$ and another subgraph S is defined as

$$HSIX(\mathcal{T} \vee S) = HSIX(\mathcal{T}) + \text{HSIC}(S) - \text{HSIC}(\mathcal{T} \wedge S) \tag{4.26}$$

with

$$\text{HSIC}(\mathcal{T} \wedge S) = (a_{\mathcal{T} \wedge S} - b_{\mathcal{T} \wedge S})^2$$

where $a_{\mathcal{T} \wedge S}$ (and $b_{\mathcal{T} \wedge S}$) is the frequency of S and at least one of the elements from $\mathcal{T}$ appearing in the same graph from A (or B, resp.).

Forward Selection Algorithm via HSIX

Now we have all ingredients for formulating a forward selection algorithm on frequent subgraphs using HSIC (see Algorithm 4). First of all, we initialize the solution set $\mathcal{T}$ as an empty set. In the next step, we compute HSIC for all subgraphs $\mathcal{S}$ selected by gSpan, and pick the one with maximum HSIC value (=HSIX value) as our top selected feature S. Afterwards, we repeat the following steps iteratively, as long as $HSIX(\mathcal{T}) < HSIX(\mathcal{T} \vee S)$. We add S to the set of selected features $\mathcal{T}$, and remove S from the set of frequent subgraphs $\mathcal{S}$. Then we search the next subgraph S that maximizes $HSIX(\mathcal{T} \vee S)$. This procedure is iterated until $HSIX(\mathcal{T} \vee S) = HSIX(\mathcal{T})$. This means that adding the last subgraph S does not increase HSIX, *i.e.,* our selected set $\mathcal{T}$ does not get more informative by adding S. Here the algorithm stops and $\mathcal{T}$ is the solution, the set of selected features.

Algorithm 4 Forward selection of frequent subgraphs using HSIX.

Input: frequent subgraphs $\mathcal{S}$ selected by gSpan

1: $\mathcal{T} := \emptyset$
2: Find frequent subgraph $S \in \mathcal{S}$ maximizing $HSIX(S)$
3: **while** $HSIX(\mathcal{T}) < HSIX(\mathcal{T} \vee S)$ **do**
4: $\quad \mathcal{T} := \mathcal{T} \cup \{S\}$;
5: $\quad \mathcal{S} := \mathcal{S} \setminus \{S\}$;
6: $\quad$ Find $S \in \mathcal{S}$ maximizing $HSIX(\mathcal{T} \vee S)$
7: **end while**

Output: selected subgraph features $\mathcal{T}$

Runtime Complexity

In worst case, our forward selection algorithm requires a runtime of $O(|\mathcal{S}||\mathcal{T}|)$ where $\mathcal{S}$ is the set of subgraph features discovered by gSpan, and $|\mathcal{T}|$ is the number of features selected by our algorithm until $HSIX(\mathcal{T} \vee \{S\}) = HSIX(\mathcal{T})$ for any of the remaining subgraphs $S \notin \mathcal{T}$.

Unbalanced Case

So far, we assumed that both classes A and B contained the same number of instances, $|A| = |B|$. If we drop this condition, HSIX changes as follows. To account for the differences in size between $|A|$ and $|B|$, we have to modify the kernel matrix on the labels $\mathbf{L}$.

We set $y_i = \frac{1}{|A|}$ if graph $G_i \in A$ and $y_i = -\frac{1}{|B|}$ if graph $G_i \in B$. We then apply a linear kernel to these labels to obtain $\mathbf{L}$.

Straightforward matrix multiplication then tells us that for a subgraph S, HSIC(S) changes into

$$\text{HSIC}(S) = \frac{a_S^2}{|A|^2} + \frac{b_S^2}{|B|^2} - 2\frac{a_S b_S}{|A||B|} = (\frac{a_S}{|A|} - \frac{b_S}{|B|})^2 \tag{4.27}$$

Obviously, the difference between the balanced and the unbalanced case is minor. Instead of dealing with absolute frequencies of S, we are dealing with relative frequencies of S in A and B.

4.2.4 Experiments

In this section, we conduct experiments to examine the efficiency and effectiveness of HSIX in frequent subgraphs. Through our experiments, we illustrate that HSIX (1) is efficient enough in comparison with feature generation, *i.e.,* frequent graph mining in our problem setting; and (2) has higher classification accuracy than other fast feature selection methods.

Datasets

To evaluate our algorithm, we employed experiments using two series of real-world data:

1. AIDS antiviral screen data: it contains the activity test information of 43,905 chemical compounds. Each chemical compound is labeled active (CA), moderately active (CM), or inactive (CI) to HIV virus. Among these 43,905 compounds, 423 of them belong to CA, 1081 are of CM, and the rest is in Class CI. This dataset is available publicly on the website of the Developmental Therapeutics Program (http://dtp.nci.nih.gov/docs/aids/aids_screen.html). In this experiment, we use CA vs. CI data.

2. Anti-cancer screen datasets: we collected 10 datasets from the PubChem website. They are selected from the bioassay records for cancer cell lines. Each of the anti-cancer screens forms a classification problem, where the class labels on these datasets are either active or inactive as a screen for anti-cancer activity. The active class is extremely rare compared with the inactive class. For a detailed description, please refer to [Wale and Karypis, 2006] and the website, http://pubchem. ncbi.nlm.nih.gov. Each dataset can be retrieved by submitting queries in the above website.

The AIDS antiviral screen dataset and some of the anti-cancer screen datasets are very skewed. In order to have a fair comparison, we make each dataset balanced by removing excessive instances from the larger class. We use 5-fold cross-validation. Each dataset is partitioned into five parts evenly. Each time, one part is used for testing and the other four are combined for frequent subgraph mining, feature selection and model learning. In our current implementation, we use LIBSVM [Chang and Lin, 2001] to train the SVM classifier based on the selected features.

Experimental Setting

We compare HSIX with two existing methods that are appropriate for efficient feature selection in thousands of frequent graph features: one is Pearson's correlation (PC) [Ein-Dor et al., 2006], the other is the sequential cover method (SC) proposed by Deshpande et al. [Deshpande et al., 2005]. We present both comparison methods in details here.

Pearson's correlation (PC) is commonly used in microarray data analysis [Ein-Dor et al., 2006], where discriminative genes for phenotype prediction need to be selected from

thousands of uninformative ones. Formally, for one dimensional feature, it is defined as,

$$r_{xy} = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{s_x s_y}, \tag{4.28}$$

where x_i is the feature value for sample i, and $\bar{x}$ and s_x computes sample mean and standard deviation respectively. y_i is the class label, and $\bar{y}$ and s_y are defined similarly for the labels. Pearson's correlation seeks linear relationship between two random variables. For Gaussian random variables, the coefficient is zero if the two random variables are independent. In our feature selection setting, we use the square, r_{xy}^2, of the Pearson's correlation to measure the predictive power of the subgraphs for the labels. This method is fast: it examines a single subgraph at a time and does not take into account the interaction between subgraphs. In practice, it usually works well and serves as a baseline method for comparison.

Algorithm 5 outlines the sequential cover method (SC) [Cheng et al., 2007]. Frequent graphs are first ranked according to their relevance measure such as information gain, fisher score, or confidence. In this experiment, we use confidence as the relevance measure. If a top-ranked frequent subgraph covers some of uncovered training instances, it will be inserted into $\mathcal{T}$ and removed from the feature set $\mathcal{S}$. The algorithm terminates if either all instances are covered or $\mathcal{S}$ becomes empty. SC can be executed multiple times to make several covers on the instances.

Algorithm 5 Comparison method: Sequential Cover (SC).

Input: A set of frequent subgraphs $\mathcal{S}$, a training dataset D

1: Sort subgraphs in $\mathcal{S}$ in decreasing order of relevance measure;
2: Start with the first subgraph S in $\mathcal{S}$;
3: **while** (true)
4: Find the next subgraph S;
5: If S covers at least one graph in D
6: $\mathcal{T} = \mathcal{T} \cup \{S\}$;
7: $\mathcal{S} = \mathcal{S} - \{S\}$;
8: If a graph G in D is covered
9: $D = D - \{G\}$;
10: If $D = \emptyset$ or $\mathcal{S} = \emptyset$
11: break;
12: **return** $\mathcal{T}$

Output: A selected set of subgraphs, $\mathcal{T}$

We found SC, PC, and HSIX may select different numbers of features. For example, PC requires the user to specify the number of features to select. HSIX often selects a 5-10 times smaller number of features than SC. In order to make the comparison fair, we take

the number of features that HSIX selects in one round as a cut-off and let PC and SC generate the same number of features.

Results

We first check the runtime performance of these three algorithms. The time cost T_{total} of training a classifier based on frequent subgraphs has three components: T_{mining}, $T_{selection}$, and $T_{learning}$. T_{mining} is the computation time for frequent subgraph mining; $T_{selection}$ is the feature selection time; and $T_{learning}$ is the classifier training time.

$$T_{total} = T_{mining} + T_{selection} + T_{learning}.$$

As a rule of thumb, for feature selection, as long as $T_{selection}$ does not dominate T_{total}, the selection algorithm is efficient. Figures 4.3 and 4.4 show the runtime comparison between the three algorithms: SC, PC, and HSIX by varying the minimum support threshold. We also plot the mining time of gSpan for comparison.

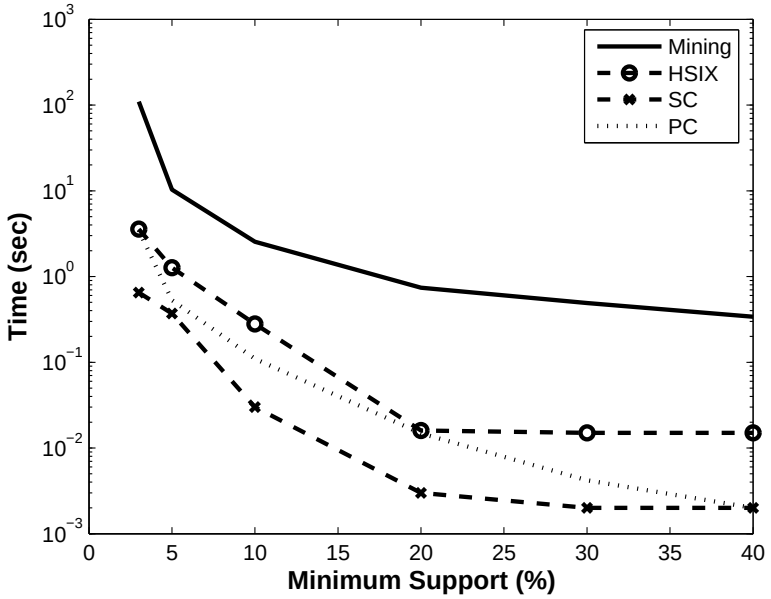

Figure 4.3: Runtime on AIDS data for gSpan (Mining), HSIX, Sequential Cover (SC), and Pearson's correlation (PC).

From Figures 4.3 and 4.4, we can see that, SC is the most efficient since it basically performs a sequential scan of the features. HSIX is slightly slower than PC. This is because HSIX not only considers the HSIC score, but also the correlation between features; while the other two methods consider each feature individually. However, HSIX is still much faster than gSpan, indicating that is can be used as an efficient component of a subgraph mining framework. For the AIDS dataset, at 3% minimum support, gSpan generates 191,328 frequent subgraphs, among which 23 are selected by HSIX. For the NCI83 dataset, there

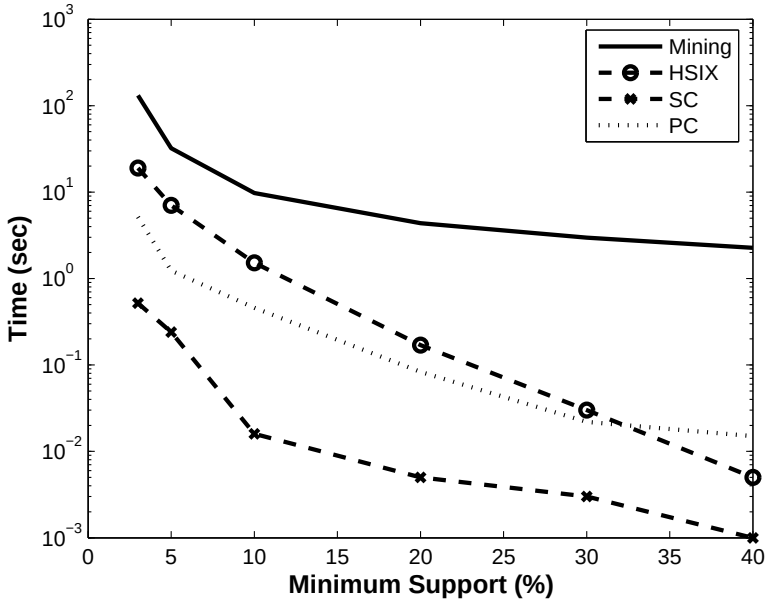

Figure 4.4: Runtime on NCI83 for gSpan (Mining), HSIX, Sequential Cover (SC), and Pearson's correlation (PC).

are 50, 102 frequent subgraphs, among which 67 are selected. This demonstrates that HSIX can really single out a compact feature set for classification.

In the next experiment, we test the classification accuracy of SC, PC, and HSIX on the real datasets we discussed above. Since each dataset is balanced, we define accuracy as # of true positives + # of true negatives divided by # of instances. Table 4.3 shows the number of selected features and the classification accuracy achieved by the three methods. As observed in Table 4.3, HSIX achieved the best classification accuracy and PC comes next, followed by SC. This result demonstrates that HSIX is effective at selecting a compact set of high quality features for classification.

4.2.5 Summary

In this chapter, we have defined a novel class of feature selection algorithms for supervised learning. They are based on maximizing the dependence between the features and the class labels of data objects.

In this section, we have extended our method to feature selection among frequent subgraphs, where the huge number of features makes feature selection particularly challenging. Our HSIC-based method extracts frequent subgraphs from the complete set of frequent subgraphs $\mathcal{S}$ determined by gSpan. Unlike its predecessors which use ad-hoc strategies for feature selection, our novel approach defines a whole class of principled and theoretically justified feature selection strategies in frequent subgraphs. One instance of this class provides us with a frequency-based criterion for subgraph selection that can be evaluated highly efficiently, is intuitive, and selects a compact set of features among the thousands

dataset	# of features	SC	PC	HSIX
NCI1	41	66.49	69.91	**72.52**
NCI109	53	65.27	70.51	**73.22**
NCI123	75	63.88	66.37	**69.15**
NCI145	60	66.44	70.70	**74.59**
NCI33	18	65.72	69.19	**71.52**
NCI330	20	71.23	68.64	**73.22**
NCI41	47	64.55	64.69	**69.72**
NCI47	42	66.72	68.17	**72.85**
NCI81	28	64.77	67.60	**72.75**
NCI83	57	64.08	67.03	**68.49**
AIDS	23	76.03	73.44	**80.13**

Table 4.3: Feature Selection among frequent subgraphs: Classification Accuracy.

and millions of frequent subgraphs gSpan detects. In our experimental evaluation, the features selected by our method lead to higher classification accuracies than those select by competing approaches.

Chapter 5

Summary and Outlook: Applications in Bioinformatics

In this chapter, we want to summarize our findings, and show that our efficient graph kernels and novel kernel methods have several immediate applications in bioinformatics. In addition, we will give an overview of the topics in bioinformatics we have already explored, and of the problems we want to study in future research.

5.1 Summary

In this thesis, we have tackled the problem of graph comparison via graph kernels. This task of measuring the similarity of two graphs is the fundamental algorithmic problem in graph mining. As graph mining is gaining more and more attention due to the availability of graph data in bioinformatics, social network analysis, and the Internet, graph comparison is now more important than ever.

Although graph comparison has been a long standing research topic in computer science, a general efficient solution to this problem has not been achieved. All principled approaches to graph comparison, such as isomorphism and edit distances based techniques, suffer from worst-case exponential runtime, as their search spaces are growing exponentially with the size of the graphs. Heuristic alternatives, such as some topological descriptors, might produce viable results on certain datasets in some applications, but do not grant a general solution. In addition, some of these approaches are hard to parameterize, and produce good results only after finding the right parameter setting.

In this thesis, we have taken a new road to the graph comparison problem. We have focused on graph comparison via graph kernels. Graph kernels have two great advantages over their competitors. First, as all kernel functions, they can compare graphs in a space of graph features, without even explicitly computing this feature space. This applies to all features of a graph: its topology, its edges, its nodes, its labels, and all other features that can be derived from a graph. Second, graph kernels can be combined with any kernel method, a huge family of machine learning algorithms for data mining and pattern recognition. This modularity makes them particularly attractive for graph mining.

Despite these advantages, graph kernels suffer from several weaknesses. A good graph kernel should provide an expressive measure of similarity on graphs, it should be efficient to compute, positive definite and not restricted to certain classes of graphs. However, none

of the state-of-the-art graph kernels meets all these requirements. This becomes most apparent in the classic graph kernels that count common walks in two graphs [Gärtner et al., 2003, Kashima et al., 2003]. Their runtime of $O(n^6)$ (n is the size of the larger of the two input graphs), while polynomial, is too slow for real-world applications. Furthermore, they suffer from a phenomenon called *tottering*. As walks allow for repetitions of nodes and edges, a walk common to two graphs may repeatedly visit the same set of nodes and edges, thereby creating an artificially high similarity score. Even worse, a second problem occurs that we referred to as *halting*. As walks can be of infinite length, random walk kernels employ a decaying factor to downweight longer walks. As we explain in this thesis, this decaying factor has to be set to values so small that often all walks of length 2 and longer hardly contribute anything to the similarity score. The random walk graph kernel then degenerates to a naive similarity measure that compares all pairs of edges in two graphs.

In Chapter 2, we overcome these problems of state-of-the-art graph kernels. In Section 2.1, we manage to speed up the random walk kernel to $O(n^3)$ and by a factor of more than 1,000 in CPU runtime, by extending concepts from linear algebra to Reproducing Kernel Hilbert Spaces. In Section 2.2, we define a graph kernel that compares shortest path distances in two graphs. It avoids suffering and halting, is computable in $O(n^4)$, and shows excellent performance in our experimental evaluation, both in terms of runtime and classification accuracy. To be able to cope with large graphs with hundreds and thousands of nodes, we present a graph kernel counting small common subgraphs in two graphs in Section 2.3; we refer to these small common subgraphs as *graphlets* [Przulj, 2007]. Transferring results from [Weissman et al., 2003] to graphs, we propose a sampling scheme for estimating graphlet distributions in graphs with a given level of confidence and precision. This sampling scheme allows us to compute graph kernels on graphs that were too large for graph kernels so far. In addition to convincing experimental results, this graphlet estimation kernel is both efficient to compute and is not afflicted by tottering and halting.

While our novel graph kernels are fast and expressive, and open the door to data mining and machine learning on large graphs, it sometimes seems difficult to interpret graph kernel values. Ideally, one would like to employ a statistical test to measure the significance of graph similarity. Unfortunately, no such test is described in the literature. In Chapter 3, we propose such a statistical test for graph similarity. Towards this end, we first define the first kernel-based two-sample test, based on a test statistic called Maximum Mean Discrepancy (MMD). We then explain how these two-sample tests in conjunction with graph kernels can measure similarity of two sets of graphs. Finally, we show that MMD can be employed for measuring similarity between a pair of graphs, and for defining a statistical test of graph similarity.

When we measure the similarity between two graphs in graph mining, the underlying question is usually whether these two graphs belong to the same group or class of graphs. Once we have established that they are indeed members of the same class, the natural question to ask next is: Which of their features determine their class membership? This problem is known as supervised *feature selection*. On graphs, it is equivalent to finding the subgraphs of a set of graphs which correlate with the class membership of these graphs. Only very few approaches to this problem exist, and they are all completely ad-hoc.

In Chapter 4, we develop a feature selection algorithm for graphs. We start by defining a kernel-based family of forward and backward feature selection algorithms. These employ the Hilbert-Schmidt Independence Criterion to select features that maximize dependence between the features and the class labels of data objects. We then extend this concept to feature selection on graphs, and apply it to the set of frequent subgraphs detected by gSpan, the state-of-the-art tool for frequent subgraph mining. While gSpan produces thousands and millions of frequent subgraphs in our experiments, our approach is able to identify a few dozens of informative features that outperform those selected by other competitors in our experimental evaluation on classification benchmarks.

Both our graph kernels and the novel kernel methods we have proposed have several important applications in bioinformatics. We will summarize our previous work in this area in the following sections, and give an outlook to future plans.

5.2 Graph Kernels in Bioinformatics

Graph kernels can be employed to measure similarity between graph-structured data, which are common in molecular biology. Above all, molecular structures and biological networks in bioinformatics can be represented as graphs.

5.2.1 Protein Function Prediction

In [Borgwardt et al., 2005], prior to this thesis, we have presented a graph kernel for protein function prediction on distantly related proteins. This protein graph kernel measures similarity of tertiary structures of proteins, enriched by additional information. This additional information comprised sequence and physicochemical properties of these proteins. Similarity between these graph models is measured in terms of a random walk kernel that compares both edge and node labels. We employed Support Vector Machines in combination with this graph kernel to predict whether proteins are enzymes or non-enzymes [Dobson and Doig, 2003b].

In the outlook of [Borgwardt et al., 2005], we stated that we had to look at the tertiary structure of proteins, because more detailed models using amino acids or even atoms would not be feasible for state-of-the-art kernels. In the light of our novel graphlet kernels from Section 2.3 that can deal with graphs with thousands of nodes, this statement is not true any more, and we plan to examine high resolution models of proteins using these scalable kernels. In our first experiment on such high-level resolution graph models of proteins in Section 2.3.5, we already achieved highly promising results. The ultimate challenge would be to define graph kernels that compare protein structures at the amino acid or even atomic level and outperform state-of-the-art methods for structure comparison. We are positive that these future studies will also benefit from the fact that our novel graph kernels do not suffer from tottering and from overweighting of single edges any more.

5.2.2 Biological Network Comparison

Our efficient graph kernels allow us to measure similarity between large graphs, such as protein-protein-interaction (PPI) networks [Borgwardt et al., 2007c] or metabolic networks [Oh et al., 2006].

Currently, only very few interaction networks for very few species are available [Xenarios et al., 2002]. This lack of data still limits the applicability of graph kernels in biological network comparison. As network data will become more abundant over coming years, graph kernels will then have the chance to reveal their full potential in large-scale biological network comparison.

To demonstrate this potential, we have created co-integrated gene expression and protein interaction networks in [Borgwardt et al., 2007c]. The gene expression data were obtained from two cancer studies by [Bullinger et al., 2004] and by [van't Veer et al., 2002]. Each comprised two groups of patients, one with positive disease outcome, one with negative outcome. In addition, we obtained a recent PPI network for *Homo sapiens* [Rual et al., 2005]. We integrated the expression data per patient and the PPI network into a co-integrated graph model: Each gene and its corresponding protein represent one node in that graph. Nodes are linked by an edge if

1. the corresponding genes are both up- or down-regulated with respect to a reference measurement, and
2. the corresponding genes are known to interact according to [Rual et al., 2005].

We employ an enhanced random walk kernel on these co-integrated gene expression/PPI networks in combination with SVM classifiers to predict disease outcome. The enhancement consists in performing a random both on the product graph and its complement. This way, missing edges are also taken into account. This enhanced graph kernel performs better than random on the outcome prediction task, while the classic random walk cannot reach results better than random. Furthermore, the problems inherent in the random walk kernel that we analyzed in Section 1 and solved in Section 2 contributed to this failure. These problems can be healed most easily by employing our novel graph kernels from Section 2 in future studies. Most of all, better results are hindered by the simplicity of the graph model employed, and the lack of reliability in both the gene expression and the protein interaction data. If more PPI data and more reliable PPI data is generated over coming years, these latter problems will also be solved.

5.2.3 Subgraph Sampling on Biological Networks

Besides applications of graph kernels for graph comparison, the sampling scheme we have developed in Section 2.3 as part of our graphlet kernel may have immediate implications for data mining and bioinformatics. In bioinformatics, there is huge interest in detecting network motifs, *i.e.,* small building blocks of networks that are frequent across species or within a species [Kashtan et al., 2004, Wernicke, 2005, Lee et al., 2006](see Section 3.2.4). To the best of our knowledge, none of these studies has provided a formula for determining the sample size that is required to approximate the distribution of these subgraphs with a given level of confidence and precision. In data mining, there are several studies on finding frequent subgraphs in a large graph [Kuramochi and Karypis, 2004b, Kuramochi and Karypis, 2004a], but none of these has made use of sampling techniques so far, let alone established a formula for sample complexity. Hence our graphlet sampling scheme opens

the door to a novel approach to motif discovery in bioinformatics and frequent subgraph mining in data mining.

5.3 Applications of Maximum Mean Discrepancy

Due to the modularity of kernel methods, our two novel kernel methods can — of course — not only be applied to graphs, but also to vectorial, string and other types of data. In fact, even on non-graph data, these kernel methods provide important contributions to bioinformatics. Maximum Mean Discrepancy (MMD) lends itself to several problems in data integration in bioinformatics, in which one has to determine whether two samples of data originate from the same source.

5.3.1 Data Integration in Bioinformatics

In [Borgwardt et al., 2006], we have explored this topic of data integration in bioinformatics on microarray data and protein structures. MMD is successful in telling apart microarray data from different microarray platforms, and detecting that expression levels were measured on the same platform. It is also successfully applied to confirm the existence of subtypes of cancer, as it is able to distinguish samples from different subtypes, but not within the same subtype. As shown in Section 3.2.1, it can also be used for automatic schema matching by comparing protein structures from different databases. Apart from distinguishing samples, MMD can be used as a pre-test for classification: if according to MMD, two classes originate from the same underlying distribution, binary classification might fail on this dataset.

For extending MMD to database applications in future, it has to be sped up. While MMD's quadratic runtime makes it the fastest two-sample test in the literature, database-scale applications would benefit from an even lower runtime, ideally linear runtime. This will be one focus of our future research.

5.3.2 Sample Bias Correction

Apart from two-sample problems, the idea to represent a distribution by its expectation in feature space can be exploited for designing new algorithms for many open problems.

MMD's underlying idea of representing samples by their means in feature space is a promising concept in its own right. It can be applied to a variety of other tasks in machine learning and data mining. In [Huang et al., 2007], we propose a solution to the sample bias correction problem. This problem describes the fact that often, training set and test set are drawn from different distributions. This is a major problem in classification, as classification methods from machine learning and data mining generally assume both sets to originate from the same distribution. To heal this problem, we propose to match the means of training and test set in feature space. In several experiments, this approach of *Kernel Mean Matching* is shown to outperform other approaches to sample bias correction.

In bioinformatics, sample bias often occurs in microarray analysis, where measurements from different labs tend to differ significantly due to the usage of different protocols, platforms and environmental conditions. Preliminary results on sample bias correction on microarray data in [Huang et al., 2007] were already very promising. Consequently sample

bias correction on microarrays and a comparison to the state-of-the-art techniques for this problem will be another future project of ours.

5.4 Applications of the Hilbert-Schmidt Independence Criterion

The feature selection approach that we propose in Section 4.1 is built on a powerful concept: Maximizing dependence between features and class labels. In fact, this principle allows us to define a unifying framework that subsumes many known feature selection algorithms. Furthermore, it can be transfered to other tasks in data mining and applications in bioinformatics.

5.4.1 Gene Selection via the BAHSIC Family of Algorithms

In [Song et al., 2007a], we show that the BAHSIC family of feature selection algorithms subsumes a whole battery of feature selectors known from the bioinformatics literature: Pearson's correlation coefficient [van't Veer et al., 2002, Ein-Dor et al., 2006], t-test [Tusher et al., 2001], signal-to-noise ratio [Golub et al., 1999], Centroid [Bedo et al., 2006, Hastie et al., 2001], Shrunken Centroid [Tibshirani et al., 2002, Tibshirani et al., 2003] and ridge regression [Li and Yang, 2005]. Due to the vast amount of different methods that have been defined, such a unifying framework can help to reveal their theoretical connection. Ultimately, by understanding the theoretical links between different feature selectors, we hope to understand why different gene selectors prefer different genes, and to be able to choose the best feature selector for a particular task based on theoretical considerations.

5.4.2 Dependence Maximization View of Clustering

The concept of maximizing dependence between features and class labels of data objects can be extended to other tasks in data mining. In clustering, class labels are assigned to data objects - such that dependence between their features and their labels is maximized! This is a novel view of clustering that we have recently begun to explore [Song et al., 2007b]. The fact that we maximize dependence in terms of a kernel matrix on the features and a kernel matrix on the labels creates a rich framework for expressing intra-dependencies between features and labels. In this fashion, we can design novel principled clustering algorithms. Clustering of microarray data is just one of the many potential applications of this technique in bioinformatics.

To conclude, based on our findings, we believe that graph kernel functions and kernel methods on graphs will be a key technique for exploiting the universality of graph models, and will significantly contribute to the advance of research in several areas of science, and in bioinformatics in particular.

Appendix A

Mathematical Background

A.1 Primer on Functional Analysis

Kernel methods borrow many concepts from Functional Analysis, as they compare objects in Hilbert spaces. In this section, we will define what a Hilbert Space is, starting from metric spaces and vector spaces, introducing norms, inner products, Banach spaces and their properties along the way [Schölkopf and Smola, 2002, Garrett, 2004].

A metric space is a set imbued with a distance metric:

Definition 47 (Metric Space) *A metric space M, d is a set M with a metric $d : M \times M \to \mathbb{R}$ such that for $x, x', x'' \in M$ the following conditions hold:*

$$d(x, x') \geq 0 \tag{A.1}$$
$$d(x, x') = 0 \Leftrightarrow x = x' \tag{A.2}$$
$$d(x, x') = d(x', x) \tag{A.3}$$
$$d(x, x'') \leq d(x, x') + d(x', x'') \tag{A.4}$$

A **Cauchy sequence** in a metric space M is a sequence $x_1, x_2, \ldots$ with the property that for every $\epsilon > 0$ there is an $N \in \mathbb{N}$ sufficiently large such that for $i, j \geq N$ we have $d(x_i; x_j) < \epsilon$. A point $x \in M$ is a limit of that Cauchy sequence if for every $\epsilon > 0$ there is an $N \in \mathbb{N}$ sufficiently large such that for $i \geq N$ we have $d(x_i, x) < \epsilon$. A subset M' of a metric space M is **dense** in M if every point in M is a limit of a Cauchy sequence in M'. A metric space M is **complete** if every Cauchy sequence has a limit in M. A metric space M is **bounded** if there exists some number r, such that $d(x, x') < r$ for all x and x' in M. A metric space M is **compact** if every sequence in M has a subsequence converging to a point in M. If a metric space has a countable dense subset, then it is called **separable**. Note that every compact metric space is separable.

Definition 48 (Vector Space) *A set X is called a vector space (or linear space) over $\mathbb{R}$ if addition and scalar multiplication are defined, and satisfy (for all $x,\ x',\ x'' \in X$, and*

$c, c' \in \mathbb{R}$)

$$x + (x' + x'') = (x + x') + x'', \tag{A.5}$$
$$x + x' = x' + x \in X, \tag{A.6}$$
$$0 \in X, x + 0 = x, \tag{A.7}$$
$$cx \in X, \tag{A.8}$$
$$1x = x, \tag{A.9}$$
$$c(c'x) = (cc')x, \tag{A.10}$$
$$c(x + x') = cx + cx', \tag{A.11}$$
$$(c + c')x = cx + c'x. \tag{A.12}$$

We restrict ourselves to vector spaces over $\mathbb{R}$, as these are of interest to us (definitions on $\mathbb{C}$ are analogous).

Definition 49 (Normed Space) *A* **normed space** *is a vector space X with a non-negative real-valued norm $\|\cdot\| : X \to \mathbb{R}_0^+$ with the following properties for $x, x', x'' \in X$ and $c \in \mathbb{R}$:*

$$\|x\| \geq 0 \tag{A.13}$$
$$\|x\| = 0 \Leftrightarrow x = 0. \tag{A.14}$$
$$\|cx\| = |c|\|x\|, \tag{A.15}$$
$$\|x + x'\| \leq \|x\| + \|x'\|. \tag{A.16}$$

When X has a norm $\|\cdot\|$, there is a metric naturally associated to it: $d(x, x') = \|x - x'\|$. A normed space X which is complete with the associated metric is said to be a **Banach space**.

To obtain a Hilbert space, we have to equip the vector space with an inner product.

Definition 50 (Inner Product) *Let X be a vector space. A real-valued function $\langle\cdot,\cdot\rangle : X \times X \to \mathbb{R}$ of two variables on X is an inner product if*

$$\langle x, x' \rangle = \langle x', x \rangle \tag{A.17}$$
$$\langle x + x'', x' \rangle = \langle x, x' \rangle + \langle x'', x' \rangle \tag{A.18}$$
$$\langle x, x' + x'' \rangle = \langle x, x' \rangle + \langle x, x'' \rangle \tag{A.19}$$
$$\langle x, x \rangle \geq 0 \qquad \textit{(and equality only for } x = 0\textit{)} \tag{A.20}$$
$$\langle cx, x' \rangle = c\langle x, x' \rangle \tag{A.21}$$
$$\langle x, cx' \rangle = c\langle x, x' \rangle \tag{A.22}$$

where $x, x', x'' \in X$ and $c \in \mathbb{R}$.

An inner product defines a corresponding norm on X via

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

which in turn defines a metric $d(x, x') = \|x - x'\|$.

Definition 51 (Hilbert Space) *A vector space X equipped with an inner product $\langle \cdot, \cdot \rangle$ is a pre-Hilbert space. If a pre-Hilbert space is complete with respect to the metric arising from its inner product (and norm), then it is called a Hilbert space.*

Note that every Hilbert space is a Banach space, but not vice versa.

In addition to these definitions, we will operate on the dual space of Hilbert spaces and Banach spaces in Section 3, which is defined as follows.

Definition 52 (Dual Space) *A linear functional on a vector space X with norm $\| \cdot \|_X$ is a mapping $f : X \to \mathbb{R}$ satisfying*

$$f(x + x') = f(x) + f(x'), \tag{A.23}$$

$$f(cx) = cf(x). \tag{A.24}$$

where $x, x' \in X$ and $c \in \mathbb{R}$. The dual space X^* is the set of all linear functionals on X. The (dual) norm $\| \cdot \|_{X^*}$ of a linear functional f on X is defined as

$$\|f\|_{X^*} = \sup\{f(x) : \|x\|_X \leq 1\} \tag{A.25}$$

A.2 Primer on Probability Theory and Statistics

In the following, we summarize basic terminology and concepts from probability theory and statistics [Casella and Berger, 2002, Dürr and Mayer, 2002]. In this thesis, we are dealing both with concepts from univariate and multivariate statistics. **Univariate statistics** describes a collection of procedures which involve observation and analysis of one statistical variable at a time, while **multivariate statistics** describes the statistical analysis of more than one statistical variable at a time.

σ-Algebra and Measures

To later define what a probability distribution and its expectation is, we first need the concept of a σ-algebra and a measure.

Definition 53 (σ-Algebra) *A collection of subsets of a set Ω is called a σ-algebra (or Borel field), denoted by Σ, if it satisfies the following three properties:*

- *The empty set is an element of Σ.*
- *If $A \in \Sigma$, then $A^c \in \Sigma$ (Σ is closed under complementation).*
- *If $A_1, A_2 \ldots \in \Sigma$, then $\cup_{i=1}^{\infty} A_i \in \Sigma$ (Σ is closed under countable unions).*

Definition 54 (Measure) *A measure ρ is a function defined on a σ-Algebra Σ over a set Ω and taking values in the extended interval $[0, \infty]$ such that the following properties are satisfied:*

- $\rho(\varnothing) = 0$

- $\rho\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \rho(A_i)$, *if* $A_1, A_2, A_3, ...$ *is a countable sequence of pairwise disjoint sets in* Σ

The triple (Ω, Σ, ρ) *is then called a measure space, and the members of* Σ *are called measurable sets.*

Note as an aside that one says that a property holds **almost everywhere** if the set of elements for which the property does not hold is a null set, i.e. is a set with measure zero.

Random Variables and Probabilities

We will now state the definitions necessary to define random variables and probability distributions.

Definition 55 (Sample Space) *The set* Ω *of all possible outcomes of a particular experiment is called the sample space of the experiment.*

Definition 56 (Event) *An event is any collection of possible outcomes of an experiment, that is, any subset of* Ω *(including* Ω *itself).*

Definition 57 (Random Variable) *A random variable* X *is a function* $X : \Omega \to S$ *from a sample space* Ω *into a state space* S*. If* $S = \mathbb{R}$*, then* X *is a real-valued random variable.*

Note that we concentrate on real-valued random variables in the following.

A **probability measure** P is a measure with total measure one (i.e., $P(\Omega) = 1$). If Σ is the Borel σ-algebra on a topological space, then a measure $\rho : \Sigma \to \mathbb{R}$ is said to be a *Borel probability measure* (for more details, see [Dudley, 1989]). **Probability distributions** are probability measures defined over the state space S of a random variable instead of the sample space Ω.

Definition 58 (Probability Space) *A* **probability space** *is a measure space* $(\Omega, \mathcal{E}, P)$*, where*

- Ω *is the sample space,*
- $\mathcal{E}$ *is a* σ*-algebra of subsets of* Ω *whose elements are called events,*
- P *is a probability measure mapping the elements of* $\mathcal{E}$ *to real numbers in the interval* $[0, 1]$*.*

Definition 59 (Statistical Independence) *Two events,* A_1 *and* A_2 *are statistically independent if*

$$P(A_1 \cap A_2) = P(A_1)P(A_2) \tag{A.26}$$

Similarly, two random variables, X and Y, are said to be independent if any event defined in terms of X is independent of any event defined in terms of Y. A sequence of random variables is **independent and identically distributed (i.i.d.)** if each has the same probability distribution as the others and all are mutually independent.

Definition 60 (Cumulative Distribution Function) *The (cumulative) distribution function or cdf of a random variable X, denoted by $F_X(x)$, is defined by*

$$F_X(x) = P(X \leq x), \tag{A.27}$$

for all x.

Definition 61 (Continuous and Discrete Random Variables) *A random variable X is said to be continuous if it has a cumulative distribution function which is continuous. A random variable X is said to be discrete if it has a cumulative distribution function which is a step function.*

Definition 62 (α-Quantile) *The α-quantile of the distribution of a random variable X is defined as the value(s) x such that:*

$$P(X \leq x) = \alpha \tag{A.28}$$

Definition 63 (Probability Density Function) *The probability density function $f(x)$ describes the distribution of a continuous random variable X and has the following properties:*

- $f(x) \geq 0$
- $\int_{-\infty}^{\infty} f(x)dx = 1$
- $P(a \leq X \leq b) = \int_a^b f(x)dx$ *for* $b \geq a$

Definition 64 (Probability Mass Function) *Suppose that X is a discrete random variable with values $\{x_1, x_2, x_3, \ldots\}$. Then the probability mass function $f(x)$ describes the distribution of X and is defined by*

$$f(x_i) = P(X = x_i) \tag{A.29}$$

Expectation and Central Moments

After clarifying essential prerequisites, we will now define the expectation of a random variable and its central moments.

Definition 65 (Expectation) *The expectation (expected value, mean) of a discrete random variable X with values $\{x_1, x_2, x_3, \ldots\}$ and the probability mass function $f(x)$ is*

$$\mathbf{E}[X] = \sum_i x_i f(x_i) \tag{A.30}$$

provided that the sum exists. The expectation of a continuous random variable X with probability density function $f(x)$ is

$$\mathbf{E}[X] = \int_{-\infty}^{\infty} x f(x)dx \tag{A.31}$$

provided that the integral exists.

Definition 66 (Central Moments and Variance) *The n-th central moment μ_n of a random variable X is the quantity*

$$\mathbf{E}[(X - \mathbf{E}[X])^n] \tag{A.32}$$

The second central moment is the variance.

The **standard deviation** σ is defined as the square root of the variance.

Definition 67 (Skewness and Kurtosis) *Let μ_n denote the n-th central moment of a random variable X. Two quantities of interest, in addition to the mean and variance are*

$$\alpha_3 = \frac{\mu_3}{(\mu_2)^{3/2}} \tag{A.33}$$

and

$$\alpha_4 = \frac{\mu_4}{(\mu_2)^2} \tag{A.34}$$

The value α_3 is called the skewness and α_4 is called the kurtosis of X.

The following theorem will be helpful in a proof in Appendix B.

Theorem 68 (Jensen's Inequality) *Let X be some random variable, and let $g(X)$ be a convex function. Then the expected value of $g(X)$ is at least the value of g at the mean of X:*

$$\mathrm{E}[g(X)] \geq g(\mathrm{E}[X]). \tag{A.35}$$

Estimator and Bias

Throughout this thesis, we define so-called *estimators* to estimate properties of underlying probability distributions. An **estimator** is a rule that tells how to calculate an estimate based on the measurements contained in a sample. For example, the sample mean average is an estimator for the population mean. An estimator may be biased or unbiased, as defined in the following.

Definition 69 (Bias) *The bias of an estimator W of a parameter θ is the difference between the expected value of W and θ; that is, $Bias_\theta W = \mathbf{E}_\theta W - \theta$. An estimator whose bias is identically (in θ) equal to 0 is called unbiased and satisfies $E_\theta W = \theta$ for all θ; otherwise it is called a biased estimator.*

Convergence in Distribution

In Section 3 we will repeatedly make use of two concepts, convergence in distribution and asymptotic normality, which we define here.

Definition 70 (Convergence in Distribution) *A sequence of random variables $X_1, X_2, \ldots$ converges to the random variable X in distribution, denoted $X_1, X_2, \ldots \xrightarrow{D} X$, if their respective cumulative distribution functions $F_1, F_2, \ldots$ converge to the cumulative distribution function F of X, wherever F is continuous.*

Definition 71 (Asymptotic Normality) *A sequence of random variables X_m is said to be asymptotically normal with mean $\mu[X_m]$ and standard deviation σ_m if $\sigma_m > 0$ for m sufficiently large and*

$$(X_m - \mu[X_m])/\sigma_m \xrightarrow{D} Z, \text{ where } Z \sim \mathcal{N}(0,1), \tag{A.36}$$

where $\mathcal{N}(0,1)$ is a normal distribution with zero mean and unit variance.

U-Statistics

Both novel kernel methods we define in this thesis employ U-statistics. Here we summarize their main characteristics (following [Ferguson, 2003]).

Definition 72 (U-Statistics) *For a real-valued measurable function, $h(x_1, ..., x_n)$ and for a sample, $X_1, ..., X_m$, of size $m \geq n$ from a distribution P, a U-statistic with kernel h is defined as*

$$U_m = U_m(h) = (m)_n^{-1} \sum_{\mathbf{i}_n^m} h(X_{i_1}, \ldots, X_{i_n})$$

where the summation is over the set $\mathbf{i}_n^m$, which denotes the set of all n-tuples drawn without replacement from $\{1, \ldots, m\}$, and $(m)_n$ is a Pochhammer coefficient, i.e., $(m)_n = \frac{m!}{(m-n)!}$.

When using U-statistics for testing hypotheses (see Section 3.1.2), it occasionally happens that at the null hypothesis, the asymptotic distribution of the U-statistics has variance zero. This is a degenerate case. The general definition of degeneracy for a U-statistic of order m and variances $\sigma_1^2 \leq \sigma_2^2 \leq \ldots \leq \sigma_m^2$ is as follows.

Definition 73 (Degeneracy of U-Statistics) *A U-statistic has a degeneracy of order k if $\sigma_1^2 = \cdots = \sigma_k^2 = 0$ and $\sigma_{k+1}^2 > 0$.*

Appendix B

Proofs on Maximum Mean Discrepancy

In this section, we provide proofs for three theorems from Section 3.1.

Proof of Theorem 30

Theorem 30 Denote by $\mathcal{B}$ a Banach space which is dense in $C(\mathcal{X})$ and let $\mathcal{F}$ be a unit ball in a $\mathcal{B}$. Then $\mathrm{MMD}\left[\mathcal{F}, p, q\right] = 0$ if and only if $p = q$.

Proof [Theorem 30]
It is clear that $\mathrm{MMD}(\mathcal{F}, p, q)$ is zero if $p = q$. We prove the converse by showing that $\mathrm{MMD}\left[C(\mathcal{X}), p, q\right] = D$ for some $D > 0$ implies $\mathrm{MMD}(\mathcal{F}, p, q) > 0$: this is equivalent to $\mathrm{MMD}(\mathcal{F}, p, q) = 0$ implying $\mathrm{MMD}(C(\mathcal{X}), p, q) = 0$ (where this last result implies $p = q$ by Lemma 28, noting that compactness of the metric space $\mathcal{X}$ implies its separability). Let $\mathcal{B}$ be a Banach space dense in $C(\mathcal{X})$ in the L_∞ norm. If $\mathrm{MMD}\left[C(\mathcal{X}), p, q\right] = D$, then there exists some $\tilde{f} \in C(\mathcal{X})$ for which $\mathbf{E}_p\left[\tilde{f}\right] - \mathbf{E}_q\left[\tilde{f}\right] \geq D/2$. Exploiting the properties of $\mathcal{B}$ we know that for all $\epsilon \in (0, D/8)$, we can find some $f^* \in \mathcal{B}$ satisfying $\left\|f^* - \tilde{f}\right\|_\infty < \epsilon$. Thus, we obtain $\left|\mathbf{E}_p\left[f^*\right] - \mathbf{E}_p\left[\tilde{f}\right]\right| < \epsilon$ and consequently

$$\left|\mathbf{E}_p\left[f^*\right] - \mathbf{E}_q\left[f^*\right]\right| > \left|\mathbf{E}_p\left[\tilde{f}\right] - \mathbf{E}_q\left[\tilde{f}\right]\right| - 2\epsilon > \tfrac{D}{2} - 2\tfrac{D}{8} = \tfrac{D}{4} > 0.$$

Finally, using $\left\|f^*\right\|_{\mathcal{B}} < \infty$, we have

$$\left[\mathbf{E}_p\left[f^*\right] - \mathbf{E}_q\left[f^*\right]\right] / \left\|f^*\right\|_{\mathcal{B}} \geq D/(4\left\|f^*\right\|_{\mathcal{B}}) > 0,$$

and hence $\mathrm{MMD}(\mathcal{F}, p, q) > 0$. ■

Proof of Theorem 37

Theorem 37 Let p, q, X, Y be defined as in Problem 1, and assume $|k(x, y)| \leq K$. Then

$$\Pr\left\{\left|\mathrm{MMD}(\mathcal{F}, X, Y) - \mathrm{MMD}(\mathcal{F}, p, q)\right| > 2\left(\left(K/m_1\right)^{\frac{1}{2}} + \left(K/m_2\right)^{\frac{1}{2}}\right) + \epsilon\right\} \leq 2\exp\left(\tfrac{-\epsilon^2 m_1 m_2}{2K(m_1+m_2)}\right).$$

To prove this theorem, we need the following theorem, due to [McDiarmid, 1969].

Theorem 74 (McDiarmid's Inequality) *Let $f : \mathcal{X}^m \to \mathbb{R}$ be a function such that for all $i \in \{1, \ldots, m\}$, there exist $c_i < \infty$ for which*

$$\sup_{X=(x_1,\ldots,x_m)\in\mathcal{X}^m, \tilde{x}\in\mathcal{X}} |f(x_1, \ldots, x_m) - f(x_1, \ldots x_{i-1}, \tilde{x}, x_{i+1}, \ldots, x_m)| \leq c_i.$$

Then for all probability measures p and every $\epsilon > 0$,

$$p_{\mathcal{X}^m}\left(f(X) - \mathbf{E}_{\mathcal{X}^m}(f(X)) > \epsilon\right) < \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^m c_i^2}\right).$$

We also define the Rademacher average of the function class $\mathcal{F}$ with respect to the m-sample X.

Definition 75 (Rademacher Average of $\mathcal{F}$ on X) *Let $\mathcal{F}$ be a universal RKHS on the compact domain $\mathcal{X}$, with kernel bounded by $|k(x, y)| \leq K$. Let X be an i.i.d. sample of size m drawn according to p, and let σ_i be i.i.d. and take values in $\{-1, 1\}$ with equal probability. We define the Rademacher average*

$$R_m(\mathcal{F}, X) := \mathbf{E}_\sigma \sup_{f\in\mathcal{F}} \left|\frac{1}{m}\sum_{i=1}^m \sigma_i f(x_i)\right| \leq (K/m)^{1/2},$$

where the upper bound follows [Bartlett and Mendelson, 2002, Lemma 22].

We want to show that the absolute difference between $\mathrm{MMD}(\mathcal{F}, p, q)$ and $\mathrm{MMD}(\mathcal{F}, X, Y)$ is close to its expected value, independent of the distributions p and q. To this end, we prove three intermediate results, which we then combine. The first result we need is an upper bound on the absolute difference between $\mathrm{MMD}(\mathcal{F}, p, q)$ and $\mathrm{MMD}(\mathcal{F}, X, Y)$. Given that $\mathcal{F}$ is closed under negation, we have

$$\begin{aligned}
& |\mathrm{MMD}(\mathcal{F}, p, q) - \mathrm{MMD}(\mathcal{F}, X, Y)| \\
= & \left|\sup_{f\in\mathcal{F}} (\mathbf{E}_p(f) - \mathbf{E}_q(f)) - \sup_{f\in\mathcal{F}} \left(\frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i) - \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j)\right)\right| \\
\leq & \underbrace{\sup_{f\in\mathcal{F}} \left|\mathbf{E}_p(f) - \mathbf{E}_q(f) - \frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i) + \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j)\right|}_{\Delta(p,q,X,Y)} \qquad \text{(B.1)}
\end{aligned}$$

Second, we provide an upper bound on the difference between $\Delta(p, q, X, Y)$ and its expectation. Changing either of x_i or y_i in $\Delta(p, q, X, Y)$ results in a change of at most $2K^{1/2}/m$ or $2K^{1/2}/n$, respectively. We can then apply McDiarmid's theorem, given a denominator in the exponent of

$$m_1\left(2K^{1/2}/m_1\right)^2 + m_2\left(2K^{1/2}/m_2\right)^2 = 4K\left(\frac{1}{m_1} + \frac{1}{m_2}\right) = 4K\frac{m_1 + m_2}{m_1 m_2},$$

to obtain

$$\Pr\left(\Delta(p,q,X,Y) - \mathbf{E}_{X,Y}\left[\Delta(p,q,X,Y)\right] > \epsilon\right) \leq \exp\left(-\frac{\epsilon^2 m_1 m_2}{2K(m_1+m_2)}\right). \tag{B.2}$$

For our final result, we exploit symmetrization, following e.g. [van der Vaart and Wellner, 1996][p. 108], to upper bound the expectation of $\Delta(p,q,X,Y)$. Denoting by X' an i.i.d sample of size m_1 drawn independently of X (and likewise for Y'), we have

$$\begin{aligned}
& \mathbf{E}_{X,Y}\left[\Delta(p,q,X,Y)\right] \\
= \;& \mathbf{E}_{X,Y} \sup_{f\in\mathcal{F}} \left| \mathbf{E}_p(f) - \frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i) - \mathbf{E}_q(f) + \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j) \right| \\
= \;& \mathbf{E}_{X,Y} \sup_{f\in\mathcal{F}} \left| \mathbf{E}_{X'}\left(\frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i')\right) - \frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i) - \mathbf{E}_{Y'}\left(\frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j')\right) + \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j) \right| \\
\underset{(a)}{\leq} \;& \mathbf{E}_{X,Y,X',Y'} \sup_{f\in\mathcal{F}} \left| \frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i') - \frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i) - \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j') + \frac{1}{m_2}\sum_{j=1}^{m_2} f(y_j) \right| \\
= \;& \mathbf{E}_{X,Y,X',Y',\sigma,\sigma'} \sup_{f\in\mathcal{F}} \left| \frac{1}{m_1}\sum_{i=1}^{m_1} \sigma_i\left(f(x_i') - f(x_i)\right) + \frac{1}{m_2}\sum_{j=1}^{m_2} \sigma_j'\left(f(y_j') - f(y_j)\right) \right| \\
\underset{(b)}{\leq} \;& \mathbf{E}_{X,X'\sigma} \sup_{f\in\mathcal{F}} \left| \frac{1}{m_1}\sum_{i=1}^{m_1} \sigma_i\left(f(x_i') - f(x_i)\right) \right| + \mathbf{E}_{Y,Y'\sigma} \sup_{f\in\mathcal{F}} \left| \frac{1}{m_2}\sum_{j=1}^{m_2} \sigma_j\left(f(y_j') - f(y_j)\right) \right| \\
\underset{(c)}{\leq} \;& 2\left[R_{m_1}(\mathcal{F},p) + R_{m_2}(\mathcal{F},q)\right]. \\
\underset{(d)}{\leq} \;& 4\left(K/m_1\right)^{1/2}, \qquad \text{(B.3)}
\end{aligned}$$

where (a) uses Jensen's inequality, (b) uses the triangle inequality, (c) substitutes Definition 75 (the Rademacher average), and (d) bounds the Rademacher averages, also via Definition 75.

Having established our preliminary results, we proceed to the proof of Theorem 37.

Proof [Theorem 37] Combining equations (B.2) and (B.3), gives

$$\Pr\left\{\Delta(p,q,X,Y) - 4\left(K/m\right)^{1/2} > \epsilon\right\} \leq \exp\left(-\frac{\epsilon^2 m_1 m_2}{2K(m_1+m_2)}\right).$$

Substituting equation (B.1) yields the result. ■

Proof of Theorem 38

Theorem 38 Under the conditions of Theorem 37 where additionally $p = q$ and $m = m_1 = m_2$,

$$\mathrm{MMD}(\mathcal{F}, X, Y) > \underbrace{m^{-\frac{1}{2}}\sqrt{2\mathbf{E}_p\left[k(x,x) - k(x,x')\right]}}_{B_1(\mathcal{F},p)} + \epsilon > \underbrace{2(K/m)^{1/2}}_{B_2(\mathcal{F},p)} + \epsilon,$$

both with probability less than $\exp\left(-\frac{\epsilon^2 m}{4K}\right)$.

Proof In the following we derive the Theorem 38 result, namely the large deviation bound on the MMD when $p = q$ and $m = m_1 = m_2$. Note that we consider only positive deviations of $\mathrm{MMD}(\mathcal{F}, X, Y)$ from $\mathrm{MMD}(\mathcal{F}, p, q)$, since negative deviations are irrelevant to our hypothesis test. The proof follows the same three steps as in the previous proof. The first step in (B.1) becomes

$$\begin{aligned}\mathrm{MMD}(\mathcal{F}, X, Y) - \mathrm{MMD}(\mathcal{F}, p, q) &= \mathrm{MMD}(\mathcal{F}, X, X') - 0 \\ &= \sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^{m}\left(f(x_i) - f(x_i')\right)\right). \qquad \text{(B.4)}\end{aligned}$$

The McDiarmid bound on the difference between (B.4) and its expectation is now a function of $2m$ observations in (B.4), and has a denominator in the exponent of $2m\left(2K^{1/2}/m\right)^2 = 8K/m$. We use a different strategy in obtaining an upper bound on the expected (B.4), however: this is now

$$\begin{aligned}&\mathbf{E}_{X,X'}\left[\sup_{f\in\mathcal{F}}\frac{1}{m}\sum_{i=1}^{m}\left(f(x_i) - f(x_i')\right)\right] \\ =\ &\frac{1}{m}\mathbf{E}_{X,X'}\left\|\sum_{i=1}^{m}\left(\phi(x_i) - \phi(x_i')\right)\right\| \\ =\ &\frac{1}{m}\mathbf{E}_{X,X'}\left[\sum_{i=1}^{m}\sum_{j=1}^{m}\left(k(x_i,x_j) + k(x_i',x_j') - k(x_i,x_j') - k(x_i',x_j)\right)\right]^{\frac{1}{2}} \\ \leq\ &\frac{1}{m}\left[2m\mathbf{E}_x k(x,x) + 2m(m-1)\mathbf{E}_{x,x'}k(x,x') - 2m^2\mathbf{E}_{x,x'}k(x,x')\right]^{\frac{1}{2}} \\ =\ &\left[\frac{2}{m}\mathbf{E}_{x,x'}\left(k(x,x) - k(x,x')\right)\right]^{\frac{1}{2}} \qquad \text{(B.5)} \\ \leq\ &(2K/m)^{1/2}. \qquad \text{(B.6)}\end{aligned}$$

We remark that both (B.5) and (B.6) are bounds the amount by which our biased estimate of the population MMD exceeds zero under $\mathcal{H}_0$. Combining the three results, we find that

under $\mathcal{H}_0$,

$$\Pr\left\{\mathrm{MMD}(\mathcal{F},X,X') - \left[\frac{2}{m}\mathbf{E}_{x,x'\sim p}\left(k(x,x)-k(x,x')\right)\right]^{\frac{1}{2}} > \epsilon\right\} \;<\; \exp\left(\frac{-\epsilon^2 m}{4K}\right) \quad \text{and}$$

$$\Pr\left\{\mathrm{MMD}(\mathcal{F},X,X') - (2K/m)^{1/2} > \epsilon\right\} \;<\; \exp\left(\frac{-\epsilon^2 m}{4K}\right).$$

■

List of Figures

List of Tables

Bibliography

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 487–499.

[Anderson et al., 1994] Anderson, N., Hall, P., and Titterington, D. (1994). Two-sample test statistics for measuring discrepancies between two multivariate probability density functions using kernel-based density estimates. *Journal of Multivariate Analysis*, 50:41–54.

[Arcones and Giné, 1992] Arcones, M. and Giné, E. (1992). On the bootstrap of u and v statistics. *The Annals of Statistics*, 20(2):655–674.

[Baker, 1973] Baker, C. (1973). Joint measures and cross-covariance operators. *Transactions of the American Mathematical Society*, 186:273–289.

[Bartlett and Mendelson, 2002] Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482.

[Bedo et al., 2006] Bedo, J., Sanderson, C., and Kowalczyk, A. (2006). An efficient alternative to SVM based recursive feature elimination with applications in natural language processing and bioinformatics. In *Artificial Intelligence*. to appear.

[Bennett and Mangasarian, 1993] Bennett, K. P. and Mangasarian, O. L. (1993). Multicategory separation via linear programming. *Optimization Methods and Software*, 3:27–39.

[Berman et al., 2000] Berman, H., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I., and Bourne, P. (2000). The protein data bank. *Nucleic Acids Research*, 28:235–242.

[Biau and Gyorfi, 2005] Biau, G. and Gyorfi, L. (2005). On the asymptotic properties of a nonparametric l_1-test statistic of homogeneity. *IEEE Transactions on Information Theory*, 51(11):3965–3973.

[Bickel, 1969] Bickel, P. (1969). A distribution free version of the Smirnov two sample test in the p-variate case. *The Annals of Mathematical Statistics*, 40(1):1–23.

[Blake and Merz, 1998] Blake, C. L. and Merz, C. J. (1998). UCI repository of machine learning databases.

[Borgelt and Berthold, 2002] Borgelt, C. and Berthold, M. (2002). Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 211–218.

[Borgwardt et al., 2007a] Borgwardt, K., Petri, T., Vishwanathan, S., and Kriegel, H.-P. (2007a). An efficient sampling scheme for comparison of large graphs. under preparation.

[Borgwardt et al., 2007b] Borgwardt, K., Yan, X., Cheng, H., Song, L., Gretton, A., Smola, A., Kriegel, H.-P., Han, J., and Yu, P. S. (2007b). Efficient feature selection in frequent subgraphs. under preparation.

[Borgwardt et al., 2006] Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schölkopf, B., and Smola, A. J. (2006). Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics (ISMB)*, 22(14):e49–e57.

[Borgwardt and Kriegel, 2005] Borgwardt, K. M. and Kriegel, H.-P. (2005). Shortest-path kernels on graphs. In *Proc. Intl. Conf. Data Mining*, pages 74–81.

[Borgwardt et al., 2007c] Borgwardt, K. M., Kriegel, H.-P., Vishwanathan, S. V. N., and Schraudolph, N. (2007c). Graph kernels for disease outcome prediction from protein-protein interaction networks. In Altman, R. B., Dunker, A. K., Hunter, L., Murray, T., and Klein, T. E., editors, *Proceedings of the Pacific Symposium of Biocomputing 2007*, Maui Hawaii. World Scientific.

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

[Bradley and Mangasarian, 1998] Bradley, P. S. and Mangasarian, O. L. (1998). Feature selection via concave minimization and support vector machines. In Shavlik, J., editor, *Proc. Intl. Conf. Machine Learning*, pages 82–90, San Francisco, California. Morgan Kaufmann Publishers. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.Z.

[Bron and Kerbosch, 1973] Bron, C. and Kerbosch, J. (1973). Algorithm 457 - finding all cliques of an undirected graph. *Comm. ACM*, 16:575–577.

[Bullinger et al., 2004] Bullinger, L., Dohner, K., Bair, E., Frohling, S., Schlenk, R. F., Tibshirani, R., Dohner, H., and Pollack, J. R. (2004). Use of gene-expression profiling to identify prognostic subclasses in adult acute myeloid leukemia. *New England Journal of Medicine*, 350(16):1605–1616.

[Bunke, 1999] Bunke, H. (1999). Error correcting graph matching: On the influence of the underlying cost function. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(9):917–922.

[Bunke, 2000] Bunke, H. (2000). Recent developments in graph matching. In *ICPR*, pages 2117–2124.

[Bunke, 2003] Bunke, H. (2003). Graph-based tools for data mining and machine learning. In *MLDM*, pages 7–19.

[Bunke and Allermann, 1983] Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253.

[Bunke et al., 2000] Bunke, H., Jiang, X., and Kandel, A. (2000). On the minimum common supergraph of two graphs. *Computing*, 65(1):13–25.

[Bunke and Shearer, 1998] Bunke, H. and Shearer, K. (1998). A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3-4):255–259.

[Burges, 1998] Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Min. and Knowl. Discov.*, 2(2):121–167.

[Caelli and Caetano, 2005] Caelli, T. and Caetano, T. S. (2005). Graphical models for graph matching: Approximate models and optimal algorithms. *Pattern Recognition Letters*, 26(3):339–346.

[Caruana and Joachims, 2004] Caruana, R. and Joachims, T. (2004). Kdd cup. http://kodiak.cs.cornell.edu/kddcup/index.html.

[Casella and Berger, 2002] Casella, G. and Berger, R. (2002). *Statistical Inference*. Duxbury, Pacific Grove, CA, 2nd edition.

[Chang and Lin, 2001] Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

[Chazelle, 2000] Chazelle, B. (2000). A minimum spanning tree algorithm with inverse-ackermann type complexity. *Journal of the ACM*, 47.

[Cheng et al., 2007] Cheng, H., Yan, X., Han, J., and Hsu, C. (2007). Discriminative frequent pattern analysis for effective classification. In *Proc. of ICDE*, Istanbul, Turkey.

[Chung-Graham, 1997] Chung-Graham, F. (1997). *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series in Mathematics. AMS.

[Conte et al., 2004] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20(3):273–297.

[Cristianini et al., 2003] Cristianini, N., Kandola, J., Elisseeff, A., and Shawe-Taylor, J. (2003). On optimizing kernel alignment. Technical report, UC Davis Department of Statistics.

[Davidson et al., 2002] Davidson, E. H., Rast, J. P., Oliveri, P., Ransick, A., Calestani, C., Yuh, C. H., Minokawa, T., Amore, G., Hinman, V., Arenas-Mena, C., Otim, O., Brown, C. T., Livi, C. B., Lee, P. Y., Revilla, R., Rust, A. G., Pan, Z., Schilstra, M. J., Clarke, P. J., Arnone, M. I., Rowen, L., Cameron, R. A., McClay, D. R., Hood, L., and Bolouri, H. (2002). A genomic regulatory network for development. *Science*, 295(5560):1669–1678.

[Debnath et al., 1991] Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. (1991). Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, 34:786–797.

[Deshpande et al., 2005] Deshpande, M., Kuramochi, M., Wale, N., and Karypis, G. (2005). Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050.

[Deutsch et al., 1999] Deutsch, A., Fernandez, M. F., Florescu, D., Levy, A. Y., and Suciu, D. (1999). A query language for XML. *Computer Networks*, 31(11-16):1155–1169.

[Diestel, 2006] Diestel, R. (2006). *Graph Theory*. Springer, 3rd edition.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. *Numerische Mathematics*, 1:269–271.

[Dipert, 1997] Dipert, R. R. (1997). The mathematical structure of the world: The world as graph. *The Journal of Philosophy*, 94(7):329–358.

[Dobson and Doig, 2003a] Dobson, P. D. and Doig, A. J. (2003a). Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol*, 330(4):771–783.

[Dobson and Doig, 2003b] Dobson, P. D. and Doig, A. J. (2003b). Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol*, 330(4):771–783.

[Duda et al., 2001] Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York. Second edition.

[Dudley, 1989] Dudley, R. M. (1989). *Real analysis and probability*. Mathematics Series. Wadsworth and Brooks/Cole, Pacific Grove, CA.

[Dudley, 2002] Dudley, R. M. (2002). *Real analysis and probability*. Cambridge University Press, Cambridge, UK.

[Dürr and Mayer, 2002] Dürr, W. and Mayer, H. (2002). *Wahrscheinlichkeitsrechnung und schließende Statistik*. Hanser Fachbuch Verlag.

[Ein-Dor et al., 2006] Ein-Dor, L., Zuk, O., and Domany, E. (2006). Thousands of samples are needed to generate a robust gene list for predicting outcome in cancer. *Proc. Natl. Acad. Sci. USA*, 103(15):5923–5928.

[Ferguson, 2003] Ferguson, T. S. (2003). U-statistics. Notes for Statistics.

[Fernández and Valiente, 2001] Fernández, M.-L. and Valiente, G. (2001). A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6/7):753–758.

[Floyd, 1962] Floyd, R. (1962). Algorithm 97, shortest path. *Comm. ACM*, 5:345.

[Fortet and Mourier, 1953] Fortet, R. and Mourier, E. (1953). Convergence de la réparation empirique vers la réparation théorique. *Ann. Scient. École Norm. Sup.*, 70:266–285.

[Fredman and Tarjan, 1987] Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *JACM*, 34(3):596–615.

[Friedman and Rafsky, 1979] Friedman, J. and Rafsky, L. (1979). Multivariate generalizations of the Wald-Wolfowitz and Smirnov two-sample tests. *The Annals of Statistics*, 7(4):697–717.

[Fröhlich et al., 2005] Fröhlich, H., Wegner, J., Sieker, F., and Zell, A. (2005). Optimal assignment kernels for attributed molecular graphs. In *Proc. of ICML*, pages 225–232, Bonn, Germany.

[Fukumizu et al., 2004] Fukumizu, K., Bach, F. R., and Jordan, M. I. (2004). Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *Journal of Machine Learning Research*, 5:73–99.

[Gardiner et al., 1992] Gardiner, J. D., Laub, A. L., Amato, J. J., and Moler, C. B. (1992). Solution of the Sylvester matrix equation $AXB^\top + CXD^\top = E$. *ACM Transactions on Mathematical Software*, 18(2):223–231.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in Mathematical Sciences. W. H. Freeman.

[Garrett, 2004] Garrett, P. (2004). Lecture notes on functional analysis. http://www.math.umn.edu/∼garrett/m/fun/.

[Gärtner, 2003] Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations*, 5(1):49–58.

[Gärtner et al., 2003] Gärtner, T., Flach, P., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In Schölkopf, B. and Warmuth, M. K., editors, *Proc. Annual Conf. Computational Learning Theory*, pages 129–143. Springer.

[Gasteiger and Engel, 2003] Gasteiger, J. and Engel, T., editors (2003). *Chemoinformatics. A Textbook*. Wiley-VCH.

[Giot et al., 2003] Giot, L., Bader, J. S., Brouwer, C., Chaudhuri, A., Kuang, B., Li, Y., Hao, Y. L., Ooi, C. E., Godwin, B., Vitols, E., Vijayadamodar, G., Pochart, P., Machineni, H., Welsh, M., Kong, Y., Zerhusen, B., Malcolm, R., Varrone, Z., Collis, A., Minto, M., Burgess, S., McDaniel, L., Stimpson, E., Spriggs, F., Williams, J., Neurath, K., Ioime, N., Agee, M., Voss, E., Furtak, K., Renzulli, R., Aanensen, N., Carrolla, S., Bickelhaupt, E., Lazovatsky, Y., DaSilva, A., Zhong, J., Stanyon, C. A., r. Finley RL, J., White, K. P., Braverman, M., Jarvie, T., Gold, S., Leach, M., Knight, J., Shimkets, R. A., McKenna, M. P., Chant, J., and Rothberg, J. M. (2003). A protein interaction map of drosophila melanogaster. *Science*, 302(5651):1727–1736.

[Golub and Van Loan, 1996] Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. John Hopkins University Press, Baltimore, MD, 3rd edition.

[Golub et al., 1999] Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. (1999). Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537.

[Gretton et al., 2007a] Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., and Smola, A. (2007a). A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA. MIT Press.

[Gretton et al., 2007b] Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., and Smola, A. (2007b). A kernel method for the two-sample-problem. Technical report, MPI Technical Report 157.

[Gretton et al., 2005] Gretton, A., Bousquet, O., Smola, A., and Schölkopf, B. (2005). Measuring statistical dependence with Hilbert-Schmidt norms. In *Proc. Intl. Conf. on Algorithmic Learning Theory*, pages 63–78.

[Grimmet and Stirzaker, 2001] Grimmet, G. R. and Stirzaker, D. R. (2001). *Probability and Random Processes*. Oxford University Press, Oxford, third edition.

[Gurevich, 2001] Gurevich, Y. (2001). *From invariants to canonization*, pages 327–331. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[Guyon and Elisseeff, 2003] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

[Guyon et al., 2002] Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422.

[Hall and Tajvidi, 2002] Hall, P. and Tajvidi, N. (2002). Permutation tests for equality of distributions in high-dimensional settings. *Biometrika*, 89(2):359–374.

[Harary, 1969] Harary, F. (1969). *Graph Theory*. Addison-Wesley, Reading, MA.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer, New York.

[Haussler, 1999] Haussler, D. (1999). Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, UC Santa Cruz.

[Hemminger, 1969] Hemminger, R. L. (1969). On reconstructing a graph. *Proceedings of the American Mathematical Society*, 20(1):185–187.

[Henze and Penrose, 1999] Henze, N. and Penrose, M. (1999). On the multivariate runs test. *The Annals of Statistics*, 27(1):290–298.

[Horvath et al., 2004] Horvath, T., Gärtner, T., and Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 158–167.

[Hotelling, 1951] Hotelling, H. (1951). A generalized t test and measure of multivariate dispersion. *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 23–41.

[Huan et al., 2003] Huan, J., Wang, W., and Prins, J. (2003). Efficient mining of frequent subgraph in the presence of isomorphism. In *Proc. 2003 Int. Conf. Data Mining (ICDM'03)*, pages 549–552.

[Huang et al., 2007] Huang, J., Smola, A., Gretton, A., Borgwardt, K., and Schölkopf, B. (2007). Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA. MIT Press.

[Huson and Bryant, 2006] Huson, D. H. and Bryant, D. (2006). Application of phylogenetic networks in evolutionary studies. *Mol Biol Evol*, 23(2):254–267.

[Imrich and Klavzar, 2000] Imrich, W. and Klavzar, S. (2000). *Product Graphs: Structure and Recognition*. Wiley Interscience Series in Discrete Mathematics). Wiley VCH.

[Inokuchi et al., 2000] Inokuchi, A., Washio, T., and Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. 2000 European Symp. Principle of Data Mining and Knowledge Discovery (PKDD'00)*, pages 13–23.

[Johnson et al., 1994] Johnson, N. L., Kotz, S., and Balakrishnan, N. (1994). *Continuous Univariate Distributions. Volume 1 (Second Edition)*. John Wiley and Sons.

[Jungnickel, 1994] Jungnickel, D. (1994). *Graphen, Netzwerke und Algorithmen*. BI-Wiss.-Verlag, Mannheim, Germany.

[Justice and Hero, 2006] Justice, D. and Hero, A. (2006). A binary linear programming formulation of the graph edit distance. *IEEE Trans. on Pattern Analysis ans Machine Intelligence*, 28(8):1200–1214.

[Kanehisa et al., 2004] Kanehisa, M., Goto, S., Kawashima, S., Okuno, Y., and Hattori, M. (2004). The kegg resource for deciphering the genome. *Nucleic Acids Res*, 32(Database issue):D277–D280.

[Kashima et al., 2003] Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *Proc. Intl. Conf. Machine Learning*, pages 321–328, San Francisco, CA. Morgan Kaufmann.

[Kashtan et al., 2004] Kashtan, N., Itzkovitz, S., Milo, R., and Alon, U. (2004). Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758.

[Kelly, 1957] Kelly, P. (1957). A congruence theorem for trees. *Pacific J. Math.*, 7(961-968):MR 19:442.

[Kira and Rendell, 1992] Kira, K. and Rendell, L. (1992). A practical approach to feature selection. In *Proc. 9th Intl. Workshop on Machine Learning*, pages 249–256.

[Koch, 2001] Koch, I. (2001). Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci.*, 250(1–2):1–30.

[Koebler and Verbitsky, 2006] Koebler, J. and Verbitsky, O. (2006). From invariants to canonization in parallel.

[Koller and Sahami, 1996] Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *Proc. Intl. Conf. Machine Learning*, pages 284–292. Morgan Kaufmann.

[Kramer et al., 2001] Kramer, S., Raedt, L., and Helma, C. (2001). Molecular feature mining in hiv data. In *Proc. of KDD*, pages 136–143, San Francisco, CA.

[Kuhn, 1955] Kuhn, H. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97.

[Kuramochi and Karypis, 2001] Kuramochi, M. and Karypis, G. (2001). Frequent subgraph discovery. In *Proc. 2001 Int. Conf. Data Mining (ICDM'01)*, pages 313–320.

[Kuramochi and Karypis, 2004a] Kuramochi, M. and Karypis, G. (2004a). Finding frequent patterns in a large sparse graph. In *SDM*.

[Kuramochi and Karypis, 2004b] Kuramochi, M. and Karypis, G. (2004b). Grew-a scalable frequent subgraph discovery algorithm. In *ICDM*, pages 439–442.

[Lawler, 1972] Lawler, E. (1972). A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18:401–405.

[Lee et al., 2006] Lee, W. P., Jeng, B. C., Pai, T. W., Tsai, C. P., Yu, C. Y., and Tzou, W. S. (2006). Differential evolutionary conservation of motif modes in the yeast protein interaction network. *BMC Genomics*, 7:89.

[Li and Yang, 2005] Li, F. and Yang, Y. (2005). Analysis of recursive gene selection approaches from microarray data. *Bioinformatics*, 21(19):3741–3747.

[Li et al., 2004] Li, S., Armstrong, C. M., Bertin, N., Ge, H., Milstein, S., Boxem, M., Vidalain, P. O., Han, J. D., Chesneau, A., Hao, T., Goldberg, D. S., Li, N., Martinez, M., Rual, J. F., Lamesch, P., Xu, L., Tewari, M., Wong, S. L., Zhang, L. V., Berriz, G. F., Jacotot, L., Vaglio, P., Reboul, J., Hirozane-Kishikawa, T., Li, Q., Gabel, H. W., Elewa, A., Baumgartner, B., Rose, D. J., Yu, H., Bosak, S., Sequerra, R., Fraser, A., Mango, S. E., Saxton, W. M., Strome, S., Heuvel, S. V. D., Piano, F., Vandenhaute, J., Sardet, C., Gerstein, M., Doucette-Stamm, L., Gunsalus, K. C., Harper, J. W., Cusick, M. E., Roth, F. P., Hill, D. E., and Vidal, M. (2004). A map of the interactome network of the metazoan c. elegans. *Science*, 303(5657):540–543.

[Liang et al., 2006] Liang, Z., Xu, M., Teng, M., and Niu, L. (2006). Netalign: a web-based tool for comparison of protein interaction networks. *Bioinformatics*, 22(17):2175–2177.

[Lodhi et al., 2002] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

[Mahé et al., 2004] Mahé, P., Ueda, N., Akutsu, T., Perret, J.-L., and Vert, J.-P. (2004). Extensions of marginalized graph kernels. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 552–559.

[Manvel and Stockmeyer, 1971] Manvel, B. and Stockmeyer, P. (1971). On reconstruction of matrices. *Mathematics Magazine*, 44(4):218–221.

[McDiarmid, 1969] McDiarmid, C. (1969). On the method of bounded differences. *Surveys in Combinatorics*, pages 148–188. Cambridge University Press.

[McKay, 1997] McKay, B. (1997). Small graphs are reconstructible. *Australas. J. Combin.*, 15:123–126.

[McKay, 1984] McKay, B. D. (1984). nauty user's guide. Technical report, Dept. Computer Science, Austral. Nat. Univ.

[Menchetti et al., 2005] Menchetti, S., Costa, F., and Frasconi, P. (2005). Weighted decomposition kernels. In *ICML*, pages 585–592.

[Nemenman et al., 2002] Nemenman, I., Shafee, F., and Bialek, W. (2002). Entropy and inference, revisited. In *Neural Information Processing Systems*, volume 14, Cambridge, MA. MIT Press.

[Neuhaus, 2006] Neuhaus, M. (2006). *Bridging the gap between graph edit distances and kernel machines*. PhD thesis, Universität Bern.

[Neuhaus and Bunke, 2005] Neuhaus, M. and Bunke, H. (2005). Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):503–514.

[Neuhaus and Bunke, 2007] Neuhaus, M. and Bunke, H. (2007). Automatic learning of cost functions for graph edit distance. *Inf. Sci.*, 177(1):239–247.

[Neumann et al., 2005] Neumann, J., Schnörr, C., and Steidl, G. (2005). Combined SVM-based feature selection and classification. *Machine Learning*, 61:129–150.

[Nijssen and Kok, 2004] Nijssen, S. and Kok, J. (2004). A quickstart in frequent structure mining can make a difference. In *Proc. 2004 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'04)*, pages 647–652.

[Nocedal and Wright, 1999] Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer.

[Oh et al., 2006] Oh, S. J., Joung, J. G., Chang, J. H., and Zhang, B. T. (2006). Construction of phylogenetic trees by kernel-based comparative analysis of metabolic networks. *BMC Bioinformatics*, 7:284.

[Page et al., 1998] Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, Stanford University, Stanford, CA, USA.

[Przulj, 2007] Przulj, N. (2007). Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183.

[Przulj et al., 2006] Przulj, N., Corneil, D. G., and Jurisica, I. (2006). Efficient estimation of graphlet frequency distributions in protein-protein interaction networks. *Bioinformatics*, 22(8):974–980.

[Ralaivola et al., 2005] Ralaivola, L., Swamidass, S. J., Saigo, H., and Baldi, P. (2005). Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110.

[Ramon and Gärtner, 2003] Ramon, J. and Gärtner, T. (2003). Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD'03).

[Riesen et al., 2006] Riesen, K., Neuhaus, M., and Bunke, H. (2006). Bipartite graph matching for computing the edit distance of graphs. Accepted for the 6th Int. Workshop on Graph-Based Representations in Pattern Recognition.

[Rosenbaum, 2005] Rosenbaum, P. (2005). An exact distribution-free test comparing two multivariate distributions based on adjacency. *Journal of the Royal Statistical Society B*, 67(4):515–530.

[Rual et al., 2005] Rual, J. F., Venkatesan, K., Hao, T., Hirozane-Kishikawa, T., Dricot, A., Li, N., et al. (2005). Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437(7062):1173–1178.

[Schölkopf, 1997] Schölkopf, B. (1997). *Support Vector Learning*. R. Oldenbourg Verlag, Munich. Download: http://www.kernel-machines.org.

[Schölkopf and Smola, 2002] Schölkopf, B. and Smola, A. (2002). *Learning with Kernels*. MIT Press, Cambridge, MA.

[Schölkopf et al., 2000] Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, 12:1207–1245.

[Schölkopf et al., 2004] Schölkopf, B., Tsuda, K., and Vert, J.-P. (2004). *Kernel Methods in Computational Biology*. MIT Press, Cambridge, Massachusetts.

[Schomburg et al., 2004a] Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. (2004a). Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32D:431–433.

[Schomburg et al., 2004b] Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. (2004b). Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Res*, 32 Database issue:D431–D433.

[Serfling, 1980] Serfling, R. (1980). *Approximation Theorems of Mathematical Statistics*. Wiley, New York.

[Shen-Orr et al., 2002] Shen-Orr, S. S., Milo, R., Mangan, S., and Alon, U. (2002). Network motifs in the transcriptional regulation network of escherichia coli. *Nat Genet*, 31(1):64–68.

[Song et al., 2007a] Song, L., Bedo, J., Borgwardt, K., Gretton, A., and Smola, A. (2007a). Gene selection via the BAHSIC family of algorithms. In *Intelligent Systems in Molecular Biology*.

[Song et al., 2007b] Song, L., Gretton, A., Smola, A., and Borgwardt, K. (2007b). A dependence maximization view of clustering. In *International Conference on Machine Learning*.

[Song et al., 2007c] Song, L., Smola, A., Gretton, A., Borgwardt, K., and Bedo, J. (2007c). Supervised feature selection via dependence estimation. In *International Conference on Machine Learning*.

[Steinwart, 2002] Steinwart, I. (2002). On the influence of the kernel on the consistency of support vector machines. *J. Mach. Learn. Res.*, 2:67–93.

[Tibshirani et al., 2002] Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. In *National Academy of Sciences*, volume 99, pages 6567–6572.

[Tibshirani et al., 2003] Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2003). Class prediction by nearest shrunken centroids, with applicaitons to DNA microarrays. *Stat Sci*, 18:104–117.

[Todeschini and Consonni, 2000] Todeschini, R. and Consonni, V. (2000). *Handbook of molecular descriptors*. Wiley-VCH.

[Toivonen et al., 2003] Toivonen, H., Srinivasan, A., King, R. D., Kramer, S., and Helma, C. (2003). Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1183–1193.

[Tsochantaridis et al., 2005] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484.

[Tusher et al., 2001] Tusher, V. G., Tibshirani, R., and Chu, G. (2001). Significance analysis of microarrays applied to the ionizing radiation response. *Proc. Natl. Acad. Sci. USA*, 98(9):5116–5121.

[van der Vaart and Wellner, 1996] van der Vaart, A. W. and Wellner, J. A. (1996). *Weak Convergence and Empirical Processes*. Springer.

[Van Loan, 2000] Van Loan, C. F. (2000). The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1–2):85–100.

[Vanetik et al., 2002] Vanetik, N., Gudes, E., and Shimony, S. E. (2002). Computing frequent graph patterns from semistructured data. In *Proc. 2002 Int. Conf. on Data Mining (ICDM'02)*, pages 458–465.

[van't Veer et al., 2002] van't Veer, L. J., Dai, H., van de Vijver, M. J., He, Y. D., Hart, A. A. M., et al. (2002). Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536.

[Vapnik and Lerner, 1963] Vapnik, V. and Lerner, A. (1963). Pattern recognition using generalized portrait method. *Autom. Remote Control*, 24:774–780.

[Vishwanathan et al., 2007a] Vishwanathan, S. V. N., Borgwardt, K., and Schraudolph, N. N. (2007a). Fast computation of graph kernels. In Schölkopf, B., Platt, J., and Hofmann, T., editors, *Advances in Neural Information Processing Systems 19*, Cambridge MA. MIT Press.

[Vishwanathan et al., 2007b] Vishwanathan, S. V. N., Borgwardt, K., Schraudolph, N. N., and Kondor, I. R. (2007b). On graph kernels. *J. Mach. Learn. Res.* under preparation.

[Vishwanathan and Smola, 2004] Vishwanathan, S. V. N. and Smola, A. J. (2004). Fast kernels for string and tree matching. In Schölkopf, B., Tsuda, K., and Vert, J. P., editors, *Kernel Methods in Computational Biology*, pages 113–130, Cambridge, MA. MIT Press.

[von Mering et al., 2002] von Mering, C., Krause, R., Snel, B., Cornell, M., Oliver, S. G., Fields, S., and Bork, P. (2002). Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417(6887):399–403.

[Wale and Karypis, 2006] Wale, N. and Karypis, G. (2006). Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proc. of ICDM*, pages 678–689, Hong Kong.

[Warshall, 1962] Warshall, S. (1962). A theorem on boolean matrices. *J. ACM*, 9:11–12.

[Wasserman and Faust, 1995] Wasserman, S. and Faust, K. (1995). *Social Network Analysis. Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press.

[Weis and Naumann, 2005] Weis, M. and Naumann, F. (2005). Dogmatix tracks down duplicates in XML. In *SIGMOD Conference*, pages 431–442.

[Weissman et al., 2003] Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S., and Weinberger, M. J. (2003). Inequalities for the l_1 deviation of the empirical distribution. Technical Report HPL-2003-97(R.1), HP Labs, HP Laboratories, Palo Alto.

[Wernicke, 2005] Wernicke, S. (2005). A faster algorithm for detecting network motifs. In Casadio, R. and Myers, G., editors, *WABI*, volume 3692 of *Lecture Notes in Computer Science*, pages 165–177. Springer.

[Weston et al., 2003] Weston, J., Elisseeff, A., Schölkopf, B., and Tipping, M. (2003). Use of zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461.

[Weston et al., 2000] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. (2000). Feature selection for SVMs. In *Advances in Neural Information Processing Systems 13*, pages 668–674.

[Whisstock and Lesk, 2003] Whisstock, J. C. and Lesk, A. M. (2003). Prediction of protein function from protein sequence and structure. *Q Rev Biophys*, 36(3):307–340.

[Wiener, 1947] Wiener, H. (1947). Structural determination of paraffin boiling points. *J. Am. Chem. Soc.*, 69(1):17–20.

[World Wide Web Consortium (W3C), 2005] World Wide Web Consortium (W3C) (2005). The XML data model. http://www.w3.org/XML/Datamodel.html.

[Wuchty et al., 2003] Wuchty, S., Oltvai, Z. N., and Barabasi, A. L. (2003). Evolutionary conservation of motif constituents in the yeast protein interaction network. *Nat Genet*, 35(2):176–179.

[Xenarios et al., 2002] Xenarios, I., Salwinski, L., Duan, X., Higney, P., Kim, S., and Eisenberg, D. (2002). Dip, the database of interacting proteins: a research tool for studying cellualr networks of protein interactions. *NAR*, 30:303–305.

[Yan and Han, 2002] Yan, X. and Han, J. (2002). gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724.

[Yen, 1971] Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Sciences*, 17:712–716.

[Zanzoni et al., 2002] Zanzoni, A., Montecchi-Palazzi, L., Quondam, M., Ausiello, G., Helmer-Citterich, M., and Cesareni, G. (2002). Mint: a molecular interaction database. *FEBS Lett*, 513(1):135–140.

Karsten M. Borgwardt

Chair Prof. Kriegel
Ludwig-Maximilians-Universität München
Oettingenstr. 67
80538 München
Germany

office: ++49 89 2180 9329
fax: ++49 89 2180 9192

kb@dbs.ifi.lmu.de
http://www.dbs.ifi.lmu.de/~borgward/

Education

Current status

Since Jan. 2005	**PhD student** in Computer Science Ludwig-Maximilians-Universität, Munich, Germany Advisor: Prof. Hans-Peter Kriegel

Degrees

Dec. 2004	**Diplom** (German M.Sc.) in Computer Science Ludwig-Maximilians-Universität, Munich, Germany
Sep. 2003	**M.Sc.** in Biology University of Oxford, United Kingdom

Studies abroad

Sep. to Oct. 2006 and July to Dec. 2004	**Visiting Academic** at Statistical Machine Learning Group National ICT Australia (NICTA), Canberra, Australia Advisor: Dr Alex Smola and Dr SVN Vishwanathan
Sep. 2002 to Sep. 2003	**Master Student** at University of Oxford M.Sc. in Biology Advisor: Dr Myles Axton and Dr Irmtraud Meyer

Awards and Honors

2007	German National Merit Scholarship
2006	Listed in Premier Edition of *Marquis Who's Who of Emerging Leaders*
2002	German National Merit Scholarship
1999	Stiftung Maximilianeum
1999	Bavarian Scholarship for the Gifted
1999	Finished *Gymnasium* (German high school) in 8 instead of 9 years

Research

Research Focus and Interests

General:	Intersection between machine learning, data mining and bioinformatics
	Learning on structured data and mining of structured data
Specific:	Graph mining
	Graph kernels
	Kernels for bioinformatics

Employment

2005–	Research and teaching assistant, Chair for Database Systems Ludwig-Maximilians-Universität, Munich, Germany

Teaching

Lecturer (developed and taught)

Summer 2006	Kernel Methods in Bioinformatics

Teaching Assistant

Summer 2007	Knowledge Discovery in Databases II
Winter 2007	Database Principles I
Winter 2006	Knowledge Discovery in Databases
Summer 2005	Efficient Algorithms
Winter 2005	Database Principles I

Student Tutor

Summer 2004	Database Principles II
Winter 2004	Database Principles I
Winter 2002	Introduction to Computer Science I

Professional Activities

Peer Review

- Program committee: ICML 2007, PKDD/ECML 2007, ICDM MGCS 2007.
- Reviewer for journals: Bioinformatics, Journal of Machine Learning Research, ACM TKDD, ACM TCBB, Journal of Proteome Research, Journal of Lipid Research
- Reviewer for workshops: NIPS Computational Biology Workshop 2005
- External reviewer: VLDB 2007, KDD 2007, SIGMOD 2007, VLDB 2006, SIGMOD 2006, ICDE 2006

Publications

Also available at
http://www.dbs.ifi.lmu.de/~borgward/ and at
http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/b/Borgwardt:Karsten_M=.html

Journal Articles

[1] L. Song, J. Bedo, K. M. Borgwardt, A. Gretton, and A.J. Smola. Gene selection via the BAHSIC family of algorithms. In *Intelligent Systems in Molecular Biology*, 2007.

[2] K. M. Borgwardt, A. Gretton, M. J. Rasch, H.-P. Kriegel, B. Schölkopf, and A. J. Smola. Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics (ISMB)*, 22(14):e49–e57, 2006.

[3] 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.

[4] S. V. N. Vishwanathan, K. M. Borgwardt, O. Guttman, and A. J. Smola. Kernel extrapolation. *Neurocomputing*, 69(7-9):721–729, 2006.

Peer-Reviewed Conferences

[1] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola. A kernel approach to comparing distributions. In *AAAI*, 2007. (**Highlights Track**).

[2] L. Song, A. Gretton, A. Smola, and K. Borgwardt. A dependence maximization view of clustering. In *ICML*, 2007.

[3] L. Song, A. Smola, A. Gretton, K. Borgwardt, and J. Bedo. Supervised feature selection via dependence estimation. In *ICML*, 2007.

[4] K. M. Borgwardt, H.-P. Kriegel, S. V. N. Vishwanathan, and N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Tiffany Murray, and Teri E Klein, editors, *Proceedings of the Pacific Symposium of Biocomputing 2007*, Maui Hawaii, January 2007. World Scientific.

[5] S. V. N. Vishwanathan, K. Borgwardt, and N. N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge MA, 2007. MIT Press.

[6] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Schölkopf. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[7] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola. A kernel method for the two-sample-problem. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

[8] K. M. Borgwardt, H.-P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *ICDM*, pages 818–822, 2006.

[9] K. M. Borgwardt, S. V. N. Vishwanathan, and H.-P. Kriegel. Class prediction from time series gene expression profiles using dynamical systems kernels. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Tiffany Murray, and Teri E Klein, editors, *Proceedings of the Pacific Symposium of Biocomputing 2006*, pages 547–558, Maui Hawaii, January 2006. World Scientific.

[10] K. M. Borgwardt, O. Guttman, S. V. N. Vishwanathan, and A. J. Smola. Joint regularization. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2005)*, Brugge, Belgium, 2005.

[11] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proc. Intl. Conf. Data Mining*, pages 74–81, 2005.