LEARNING WITH GRAPHS USING KERNELS FROM PROPAGATED INFORMATION

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

vorgelegt von
MARION NEUMANN
aus
Augsburg

Bonn, 2014

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

1. Gutachter: Prof. Dr. Stefan Wrobel
2. Gutachter: Prof. Dr. Kristian Kersting
Tag der Promotion: 21.01.2015
Erscheinungsjahr: 2015

Marion Neumann

Fraunhofer Institut für Intelligente Analyse-
und Informationssysteme IAIS

und

Bonn-Aachen International Center for
Information Technology B-IT

und

Rheinische Friedrich-Wilhelms-Universität Bonn,
Institut für Informatik III

ACKNOWLEDGEMENTS

First, I would like to thank Prof. Dr. Stefan Wrobel, Prof. Dr. Kristian Kersting, and Prof. Dr. Christian Bauckhage for giving me the opportunity to work on my thesis in cooperation with the Computer Science Department at the University of Bonn and the Knowledge Discovery Department at Fraunhofer IAIS.

This work would not have been possible without the support and advice of two people: Roman Garnett and Kristian Kersting. Throughout the work on my thesis, many constructive discussions with them and helpful comments from them paved the way to new ideas, interpretations, solutions, and experiments.[1] This also includes proofreading the drafts of this thesis. I am especially grateful to Roman for *everything* and to Kristian for his ceaseless enthusiasm, all his suspenseful ideas, and the meetings at Starbucks.

I also wish to thank the members of the STREAM and CAML groups at Fraunhofer IAIS for all the lively discussions, interesting spotlights, reading group seminars, and breakfasts: Ahmed Jawad, Fabian Hadiji, Babak Ahmadi, Martin Mladenov, Mirwaes Whabzada, Shan Huang, Martin Schiegg, Zhao Xu, Anja Pilz, Daniel Paurat, Katrin Ullrich, Michael Kamp, Olana Missura, Pascal Welke, Mario Boley, Roman Garnett, Tamás Horváth, and Thomas Gärtner.

Especially, I would like to thank Dr. Zhao Xu for her scientific advice, Dr. Babak Ahmadi for answering all my coding and Linux questions and for being a best friend, Daniel Paurat for being the best office-mate (ever), and Anja Pilz for her honest friendship. Further, I want to thank Shan Huang and Daniel Marthaler for their cooperation and the fantastic development of our `pyGPs` implementation, Plinio Moreno, Laura Antanas, and Luc De Raedt for our joint project work on robotic grasping, Christian Bauckhage, Lisa Hallau, and Benjamin Klatt for our joint project work on plant disease classification, and Roman Garnett and Jeff Schneider for giving me the opportunity to visit CMU.

This work vastly benefited from collaborations with my co-authors: Kristian Kersting, Roman Garnett, Novi Patricia, Christian Bauckhage, Plinio Moreno, Laura Antanas, Babak Ahmadi, Lisa Hallau, Benjamin Klatt, Nils Kriege, Petra Mutzel, Rui Pimentel de Figueiredo, José Santos–Victor, and Luc De Raedt.

Lastly, I would like to thank my family at home, my friends from DORO 48, and all the climbing friends in and around Bonn. They supported me and helped

[1] [gulp]

me to stay clear-headed *and* to rock the 8- indoors, the 6b outdoors, and the 6a multi-pitch routes in beautiful Switzerland.

This work was supported by the Fraunhofer ATTRACT fellowship STREAM, by the European Commission under contract number FP7-248258-First-MM, and by the Federal Ministry of Food, Agriculture, and Consumer Protection (BMELV) based on a decision of the German Federal Office for Agriculture and Food (BLE) under the innovation support program with grant number "2815411310."

ABSTRACT

Traditional machine learning approaches are designed to learn from independent vector-valued data points. The assumption that instances are independent, however, is not always true. On the contrary, there are numerous domains where data points are cross-linked, for example social networks, where persons are linked by friendship relations. These relations among data points make traditional machine learning difficult and often insufficient. Furthermore, data points themselves can have complex structure, for example molecules or proteins constructed from various bindings of different atoms. Networked and structured data are naturally represented by graphs, and for learning we aim to exploit their structure to improve upon non-graph-based methods.

However, graphs encountered in real-world applications often come with rich additional information. This naturally implies many challenges for representation and learning: node information is likely to be incomplete leading to partially labeled graphs, information can be aggregated from multiple sources and can therefore be uncertain, or additional information on nodes and edges can be derived from complex sensor measurements, thus being naturally continuous. Although learning with graphs is an active research area, *learning with structured data*, substantially modeling structural similarities of graphs, mostly assumes fully labeled graphs of reasonable sizes with discrete and certain node and edge information, and *learning with networked data*, naturally dealing with missing information and huge graphs, mostly assumes homophily and forgets about structural similarity. To close these gaps, we present a novel paradigm for learning with graphs, that exploits the intermediate results of iterative information propagation schemes on graphs. Originally developed for within-network relational and semi-supervised learning, these propagation schemes have two desirable properties: they capture structural information and they can naturally adapt to the aforementioned issues of real-world graph data.

Additionally, information propagation can be efficiently realized by random walks leading to fast, flexible, and scalable feature and kernel computations. Further, by considering intermediate random walk distributions, we can model structural similarity for learning with structured and networked data. We develop several approaches based on this paradigm. In particular, we introduce *propagation kernels* for learning on the graph level and *coinciding walk kernels* and *Markov logic sets* for learning on the node level. Finally, we present two application domains where kernels from propagated information successfully tackle real-world problems.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

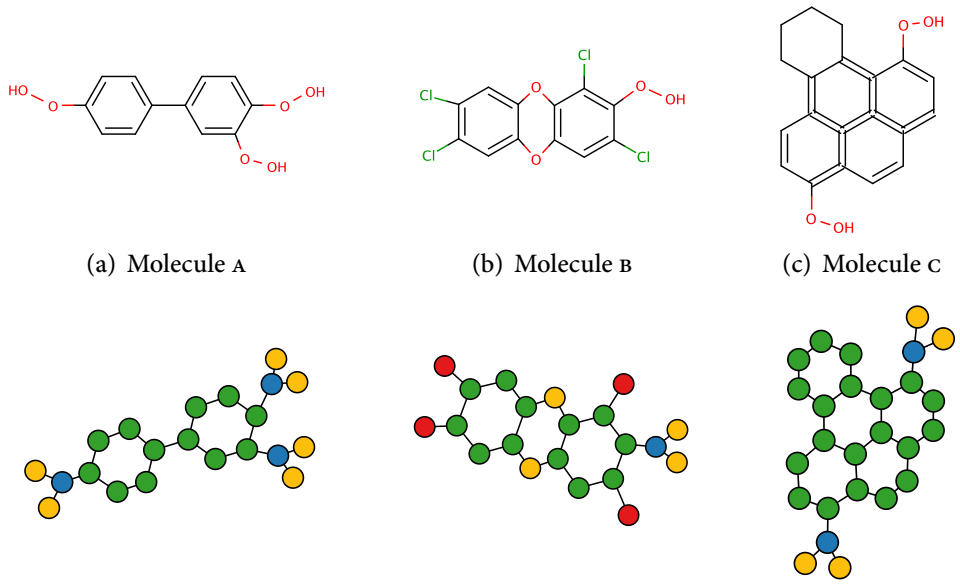
CHAPTER 1

INTRODUCTION

1.1 Networks and Structured Data: New Challenges for Learning

Whereas several decades ago networked and structured data almost always represented "real" networks such as traffic or electricity networks, "plain" chemical structures like molecular connections of atoms, or "simple" social relationships such as family trees, nowadays networked and structured data is not only universally present, it also comes with diverse and complex additional information. We use the world wide web, a network of approximately 15 billion webpages[1] with vast amounts of information such as text, images, or videos, on a daily basis. We live in a society of interconnected individuals where social and professional networks are common means of communication and organization. Every day, users upload terabytes of data such as images, videos, or status messages to their social network accounts, and thousands of new friendship and like relations are established every second of the day. Similarly, immense scientific progress leads to ever more refined models and theories of interactions in biological processes and structures. For example, biological interactions giving rise to protein–protein interaction networks can vary widely in their nature and are spatially and temporally heterogeneous. Autonomous mobile manipulation robots collect massive amounts of continuous sensor data to build networked models of the environment they act in. Furthermore, phenomena such as the increasing use of sensors to track our behaviour and the inexorable progress in the development of better communication, sensing, storage, and computing devices indicates that this trend of collecting and analyzing networks with heterogeneous information will most likely continue in the future. As the networks are getting bigger and bigger and structured data is getting more and more complex and diverse, it is likely to encounter missing or uncertain information. Further, with the development of better sensing techniques, networks can

[1] http://www.worldwidewebsize.com/



(a) Molecule A (b) Molecule B (c) Molecule C

(d) Graph G_A (e) Graph G_B (f) Graph G_C

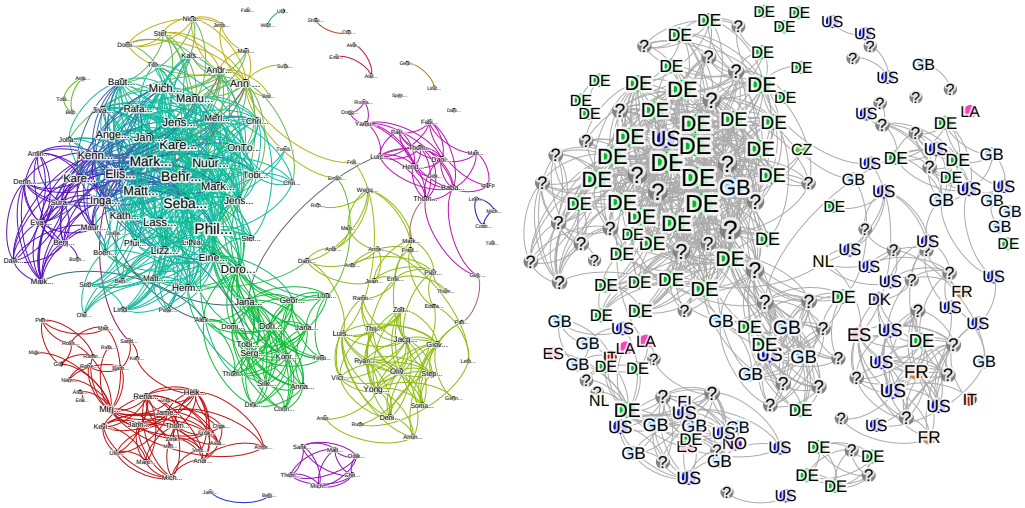
Figure 1.1: **Chemical Compounds (Structured Data).** Chemical structures (a–c) and corresponding graph representations (d–f) for three molecules: A, B, and C. Node colors represent node types, which here are different elements such as carbon (green), oxygen (yellow), hydrogen (blue), and chlorine (red). Edge types, lengths, and other node information is not depicted for the sake of simplicity.

be more easily enhanced with continuous measurements in addition to discrete labels. Thus, representing, analyzing, and learning from this kind of data becomes more important, more demanding, and also more exciting (LOW, *et al.*, 2014; PARK and FRISTON, 2013; CHAKRABARTI and FALOUTSOS, 2012; STROGATZ, 2001).

Structured and networked data are commonly represented by *graphs* and we can distinguish domains where a single graph represents relations among data instances (*networked data*) from domains of multiple graphs, where whole data entities are themselves structured objects (*structured data*). The following two examples illustrate the difference between networked and structured data and exemplify occurrences of additional and missing information.

EXAMPLE 1.1 (CHEMICAL COMPOUNDS – STRUCTURED DATA)

Consider a domain of chemical compounds where each molecule itself is represented by a set of atoms and a set of bonds connecting pairs of atoms. Three example chemical compounds are shown in the top row of Figure 1.1. Each atom is of a certain element (hydrogen, oxygen, etc.) and



(a) Nodes indicated by names (b) Missing information

Figure 1.2: **Social Network (Networked Data).** Graph representing a social network of persons and their friendship relations. In Panel (a) each (abbreviated) name represents a person and the text size is proportional to the number of connected friends. The edge colors represent groups of friends from different social contexts; for instance, purple friendship relations are "friends from work," red "friends from study and travel abroad," turquoise "friends from sports," and so on. Panel (b) shows the country of residence of each person encoded by color and a two-letter label; *?* indicates missing information.

the bonds have a certain type (single, double, etc.). As each object of interest – each molecule – can be represented by one graph, we have a domain of structured data. The bottom row of Figure 1.1 illustrates example graph representations of the three chemical compounds. Moreover, we have access to additional information for atoms and bonds such as the length of the secondary structure elements (the nodes) or measurements for various properties, such as hydrophobicity, van der Waals volume, polarity, 3D coordinates of the structures, or distances between them.

Example 1.2 (Social Network – Networked Data)

A classical example for networked data are social-network graphs such as shown in Figure 1.2(a). Here individual persons are the data points of interest and a graph is created from their relations among one another. Our respective domain is then one (possibly disconnected) graph, where two nodes representing two persons are linked if they are in a relation, for

instance a friendship relation. Now, we can imagine that persons provide additional information other than their names, such as their country of residence. However, as this information is revealed voluntarily it is likely that it is not available for every person. Thus, when analyzing the social network with respect to where the persons live, we will have to deal with missing information as illustrated in Figure 1.2(b).

Graphs are an expressive form of representation for networked and structured data. To make use of the encoded information for reasoning, it is of great interest to compare and analyze graphs efficiently and automatically. In the case of chemical compounds introduced in Example 1.1, we might like to predict a certain property of a protein from the arrangement of its atoms and bindings. Social networks as exemplified in Example 1.2 constitute a deep source of information for sociologists studying role models and social behaviour, as well as for companies marketing their products on social-network platforms. One approach towards achieving these goals is to apply machine learning to graph data. *Learning with graphs* – intersecting machine learning (BISHOP, 2006; MITCHELL, 1997), graph theory (BOLLOBÁS, 1998), and graph mining (COOK and HOLDER, 2006; HAN and KAMBER, 2006; HADZIC, *et al.*, 2010), as well as statistical relational learning (GETOOR and TASKAR, 2007) – has become an active and growing research area in computer science (FRASCONI, *et al.*, 2007; BREFELD, *et al.*, 2010; VISHWANATHAN, *et al.*, 2011). This thesis will make contributions in this interdisciplinary field of statistical machine learning with graph data.

The essential step in learning with graphs is to compare graph structure, either entire graphs in the case of structured data or neighborhood structures of nodes in the case of networked data. The comparison of complex data instances can elegantly be achieved by *kernels*. Intuitively, kernels measure the similarity among all pairs of data points; mathematically, kernels correspond to an inner product in a reproducing kernel Hilbert space. Kernel-based learning and kernels for graph data will be introduced in depth in Section 2.4 in the next chapter. So, to achieve learning with graphs we can essentially compute kernels for structured and networked data. More precisely, the considered kernels are *graph kernels* and *kernels on graphs*.

However, graphs encountered in real-world applications come with many challenges. *Structured data* is often enriched with additional information attached to the graphs' nodes and edges. This naturally implies several challenges for representation, kernel construction, and learning such as:

- missing information occurring from partially observable data,
- uncertain information arising from aggregating information from multiple sources, and

- continuous information derived from complex and possibly noisy sensor measurements.

Learning with *networked data* is not only commonly applied to large-scale graphs, where building a kernel matrix between all pairs of data points is costly, but also to domains where structural similarity is present. Example application domains are relational knowledge bases like yago[2] and DBpedia.[3] This leads to new challenges for kernel construction and learning that require going beyond the exploitation of homophily, which is the assumption that nearby nodes have the same properties.

So, to achieve successful learning with graphs one has to account for these properties not only while constructing the graphs, but also when measuring and comparing their structure. Despite the growing interest in research on learning with graphs, there are still many open issues resulting from the characteristics of real-world graph data. General limitations of existing approaches to learning with graphs are the ability to deal with uncertain and incomplete information, space and time efficiency, and the insufficient use of available information. Existing kernels for learning with graphs, for example, are not fast, flexible, and powerful at the same time. Learning with structured data, and especially graph kernel methods, substantially model the structural similarities of graphs, however they mostly assume fully labeled graphs of reasonable sizes with discrete and certain node and edge information. Graphs with uncertain and incomplete information are not considered and only recently scalable kernels for graphs with continuous node attributes have been introduced. Learning with networked data is naturally dealing with missing information. An example task is to predict the missing labels in a partially labeled graph. Also efficient algorithms for learning on large graphs have been developed. However, these methods mostly exploit homophily and fail for graphs with high structure similarity and sparsely observed labels[4] as they use the available structural information insufficiently. In general, there is only limited knowledge transfer among the subareas: learning with structured data (*learning on the graph level*), learning with networked data (*learning on the node level*), and learning with relational data. Particular approaches from the literature and their limitations will be discussed in the chapters introducing the problems and respective solutions studied in this thesis.

Despite these limitations, there are well established machine learning methods such as kernel machines for vector-valued data, and efficient and fast algorithms to propagate information in networks that can naturally incorporate missing and uncertain information. Example algorithms based on the idea of propagating

[2] `www.mpi-inf.mpg.de/yago-naga/yago/`

[3] `http://dbpedia.org/`

[4] We will use the term *sparsely labeled* (graph) instead of *partially labeled* (graph) to indicate that only very few labels are observed.

information on graphs are PageRank and label propagation (BRIN and PAGE, 1998; ZHU, 2005). Both tackle tasks of great practical and theoretical importance, and thoroughly developed and optimized implementations have been proposed in the recent years. Hence, the key insight leading to the main contribution of this thesis is that we can use these iterative information propagation schemes on graphs – and especially their intermediate results – to efficiently represent graph structure for labeled, partially labeled, and attributed graphs. The created features or kernels from graph data will then serve as input to traditional machine learning algorithms. The following section lists the contributions of this thesis specifically and relates them to the author's previously published work.

1.2 Contributions

This thesis makes several contributions to learning with graphs.

First, it introduces a graph kernel that efficiently captures the structure of data instances represented as graphs for learning on the graph level (**Part I; Contributions 1 and 2**).

Second, it claims and shows that efficient and meaningful structure representation through iterative information propagation schemes on graphs improves node classification in sparsely labeled graphs and in settings where only few example seeds for retrieval are provided. (**Part II; Contributions 1, 3, and 4**).

And finally, it showcases that kernels from propagated information can be successfully applied to real-world learning problems that are novel to the graph kernel community (**Part III; Contributions 1 and 5**).

The following list summarizes the contributions of this thesis.

1. The thesis presents a novel paradigm that exploits probability distributions of node information evolving from iterative information propagation schemes on graphs to efficiently capture graph structure and naturally deal with missing information for learning tasks in domains of both networked and structured data.

2. We introduce and study *propagation kernels* (PKs) for learning on the graph level.

    - PKs are a general family of graph kernels based on random walks computable for graphs with discrete node labels and continuous node attributes.
    - They provide the first graph kernel framework naturally designed for partially labeled graphs.
    - PKs are more scalable and faster than state-of-the-art kernels for structured data.

- They are the first graph kernels feasible for large-scale image data.

3. We present the *coinciding walk kernel* (CWK) for learning on the node level.
   - The CWK is the first label-dependent kernel on graphs leveraging information propagation, in particular parallel partially absorbing random walks.
   - We claim and show that CWKs exploit structure similarity in addition to homophily – even for sparsely labeled graphs – in a meaningful way.
   - CWKs are more scalable and can be computed more efficiently than state-of-the-art kernels on graphs.
4. We introduce *Markov logic sets* (MLS), a PageRank-based approach to set completion for relational data leveraging structural graph features.
   - The MLS approach shows that graph-based learning algorithms, in particular those leveraging information propagation schemes, are useful for learning in relational domains.
   - It is the first approach to set completion for relational data.
   - MLS represents the first use of lifted PageRank and label propagation in information retrieval.
5. We show that kernels from propagated information can be applied in real-world applications.
   - We present the first application of graph kernels to 3D object classification for robotic grasping
   - We showcase the first application of kernels derived from graph data to image-based plant disease detection and classification.

The contributions of this thesis are based on work previously published in the following peer-reviewed journals, conferences, and workshops:

Contribution 2:
Neumann, M., Garnett, R., Bauckhage, C. and Kersting, K. (2014*a*). Propagation Kernels: Efficient Graph Kernels from Propagated Information. *Machine Learning*. Under review.

Neumann, M., Patricia, N., Garnett, R. and Kersting, K. (2012*b*). Efficient Graph Kernels by Randomization. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases* (ECML/PKDD–2012), pp. 378–393.

Neumann, M., Garnett, R., Moreno, P., Patricia, N. and Kersting, K. (2012*a*). Propagation Kernels for Partially Labeled Graphs. In: *10th Workshop on Mining and Learning with Graphs* (MLG–2012). Edinburgh, UK.

Contribution 3:
NEUMANN, M., GARNETT, R. and KERSTING, K. (2013*b*). Coinciding Walk Kernels: Parallel Absorbing Random Walks for Learning with Graphs and Few Labels. In: *Asian Conference on Machine Learning* (ACML–2013), pp. 357–372.

NEUMANN, M., GARNETT, R. and KERSTING, K. (2013*a*). Coinciding Walk Kernels. In: *11th Workshop on Mining and Learning with Graphs* (MLG–2013). Chicago, Illinois, USA.

Contribution 4:
NEUMANN, M., KERSTING, K. and AHMADI, B. (2011). Markov Logic Sets: Towards Lifted Information Retrieval using PageRank and Label Propagation. In: *Proceedings of the 25th AAAI Conference on Artificial Intelligence* (AAAI–2011), pp. 447–452.

Contribution 5:
NEUMANN, M., MORENO, P., ANTANAS, L., GARNETT, R. and KERSTING, K. (2013*c*). Graph Kernels for Object Category Prediction in Task-Dependent Robot Grasping. In: *11th Workshop on Mining and Learning with Graphs* (MLG–2013). Chicago, Illinois, USA.

NEUMANN, M., HALLAU, L., KLATT, B., KERSTING, K. and BAUCKHAGE, C. (2014*b*). Erosion Band Features for Cell Phone Image Based Plant Disease Classification. In: *Proceedings of the 22nd International Conference on Pattern Recognition* (ICPR–2014), pp. 3315–3320.

1.3 Structure of the Thesis

The thesis is organized in three parts. Part I and II cover the main algorithmic contributions to learning with graphs, the *propagation kernel*, the *coinciding walk kernel*, and *Markov logic sets*. Part III presents applications of the approaches developed in the previous parts to graphs arising in real-world learning problems. Before the main contributions of this thesis are derived, Chapter 2 provides a gentle introduction to learning with graphs. Therefore, a casual title for this background chapter could be "things to know before we start." First, we establish the basis for understanding this thesis, namely we introduce networked, structured and relational data, exemplify learning tasks, and define the concepts of graphs and graph structure. Then, we introduce statistical machine learning as well as the considered learning tasks involving graph data and give a brief outline of related research fields. One of the main ideas in this thesis is to propagate information on graphs in order to learn something about their structure. Therefore, the third section in this background chapter will cast information propagation on graphs in the light of random walks, a fundamental concept intersecting statistics and graph theory. Moreover, we will derive kernel-based learning within the previously developed framework of statistical machine learning and give an overview on kernels for networked data, referred to as *kernels on graphs* and kernels for

structured data called *graph kernels*. To understand the contributions of this thesis in the broader context of relational learning, Chapter 2 is concluded by changing the perspective of data representation from graphs to relational data.

Part I and II derive the main contributions of this work and hence successively answer the following questions: *what?*, *why?*, *how?*, and *does it work?* That means, after stating the problem definition (*what?*) and showing limitations of existing related work (*why?*), the respective approaches are technically developed (*how?*). Then they will be empirically evaluated on comprehensive sets of benchmark problems (*does it work?*). Part III will then investigate the question: *does it really work?* which could be also rephrased to: *can the developed techniques help to solve relevant problems?*

More specifically, Part I introduces propagation kernels (PKs). After stating the problem of graph classification and discussing previous approaches to it, we define and evaluate propagation kernels for general graphs in Chapter 4. Then, we present a technique to efficiently compute PKs for large grid graphs and use them for image based texture classification (Chapter 5). A transition section summarizes Part I and leads over to Part II. This part is concerned with node classification in sparsely labeled graphs. After defining the problem and reviewing related work, we introduce coinciding walk kernels (CWKs) in Chapter 7. Therein, partially absorbing random walks are discussed and the kernel computation is derived. CWKs are evaluated on benchmark graphs and a knowledge completion scenario using graphs with high structure similarity. Another transition section summarizes the first two chapters of Part II and leads over to a more specific learning on the node level task, namely relational set completion. Chapter 8 discusses Markov logic sets, our approach to set completion leveraging relational data and graphs in order to derive structural features for query sets consisting of very few items only. Part III then again consists of two chapters presenting novel applications for learning with graphs: 3D object categorization (Chapter 9) and image-based plant disease classification (Chapter 10). Both chapters introduce the respective problems and discuss the performance of the derived kernels on comprehensive datasets. 3D object category prediction will be tackled with propagation kernels. Plant disease classification from image data allows the use of both coinciding walk kernels and propagation kernels. CWKs can be used for the detection of leaf spots caused by plant diseases and the PK version for grid graphs can be applied to disease classification. We compare the graph kernel based classifier to a texture feature based approach, which we specifically designed for the problem of leaf spot classification. This approach is also introduced briefly.

The conclusion summarizes all contributions of this thesis, discusses its achievements, and points out open problems and future research directions for learning with graphs and relational data.

CHAPTER 2

BACKGROUND: LEARNING WITH GRAPHS

2.1 Graphs: Networked and Structured Data

In the following, we introduce more examples of networked and structured data, outline tasks for learning with graphs, and give the basic definitions of graphs and graph structure. Further, we will briefly discuss graph construction and general graph analysis.

2.1.1 Networks, Relations, and Structured Data

Networked data is usually modeled by a single graph representing relations among data instances. *Structured data* are domains of multiple graphs, where whole data entities are themselves structured objects. Networked data is closely related to *relational data*; however, relational data is usually represented by tuples of constants or variables grouped into relations, so-called predicates. Relational data can be modeled by graphs, and this graph view can then be regarded as networked data. However, relational data is generally more expressive in the sense that the constants, which correspond to the nodes in the graph view, are instantiations of general variables. That is not necessarily the case for networked data where this information is often lost once the graphs are created. Relational data will be introduced in Section 2.5.

Note that the general term *network* can actually refer to all three types of data. For instance traffic networks constitute networked data as defined before; however, protein–protein interaction networks can form a domain of structured data. Relational data is sometimes referred to as logic networks. Therefore, we will not give a strict definition for the term network and when used its meaning will be clear from the context.

In Examples 1.1 and 1.2, we have seen two example domains of structured and networked data, namely chemical compounds and social networks. Further real-world domains of structured data are superpixel graphs representing semantic image scene compositions (HARCHAOUI and BACH, 2007), networks extracted from MRI images modeling structural and functional connections in the brain (PARK and FRISTON, 2013), text documents reflecting complex content dependencies (JIANG, *et al.*, 2010), graphs representing mathematical formulas and proofs (TSIVTSIVADZE, *et al.*, 2011), or manifold data modeling objects and scenes observed by autonomous mobile robots (KOPPULA, *et al.*, 2011; DUCHENNE, *et al.*, 2011). Other examples of networked data, also called single-graph domains, are graphs derived from the world wide web or Wikipedia linking web or wiki pages (BRIN and PAGE, 1998), product–user graphs indicating user and item similarities as well as purchase relations, that is, which user bought which items (HUANG, *et al.*, 2007), graphs representing massive online game universes where solar systems are connected by "jump gates" (GARNETT, *et al.*, 2014), traffic and public transportation networks (NEUMANN, *et al.*, 2009) and graphs modeling sediment movements in geomorphology (HECKMANN and SCHWANGHART, 2013).

As indicated by these examples, graph data arises in various contexts and models very diverse phenomena. Nevertheless, in all these scenarios we almost always want to make use of the information encoded in the graphs for reasoning. Classical example tasks in *learning with graphs* are unsupervised tasks such as clustering, that is, for example the grouping of nodes in a graph, supervised tasks such as classification and regression, where we want to predict a discrete class label or a continuous value for unseen target nodes or graphs, and information retrieval tasks such as ranking, where the goal is, for example to rank the nodes in a graph according to their importance in the network. A comprehensive introduction to machine learning and learning tasks for graph data will be provided in Section 2.2. The following examples, extending Examples 1.1 and 1.2, illustrate the classification of molecules as an example graph-classification task and community detection in social networks as an example node-clustering task.

EXAMPLE 2.1 (CHEMICAL COMPOUNDS – GRAPH CLASSIFICATION)

In the context of drug design, new chemical compounds are developed and analyzed. So, let us consider a large database of chemical compounds such as introduced in Example 1.1. Further, assume that these molecules have observable properties that can be confirmed by some expensive chemical studies. Such properties are for example against which biological target they are active and whether they are non-mutagenic to prevent the causing of cancer. For instance, molecule A in Figure 1.1(a) is an example of a mutagenic compound and molecule B (Figure 1.1(b)) is an example of

a non-mutagenic compound. Now, we design new chemical compounds including for example molecule C depicted in Figure 1.1(c). Thus, molecule C is one of our potential candidates for being used in a new drug against a certain disease. We know its structure, that is, the elements it consists of and how they are connected. Hence, we know its graph representation illustrated in Figure 1.1(f). However, we do not know any properties of our newly designed candidate drug. To avoid unnecessary, expensive laboratory experiments, we want to compare it to the data in our database. Assuming that similarly structured molecules have similar properties, we can then get an idea about whether our candidate drug is mutagenic or not. Applying machine learning to the graph database of molecules with known properties and the graph of our new test molecule, we are able to infer whether or not it is mutagenic. Leveraging graph classification, a classical task in the area of learning with graphs, we predict that molecule C will most likely be mutagenic. Hence, we do not want to prioritize considering it as our new drug. As we could make this decision without conducting expensive chemical tests, we saved time and money in our drug-development process.

Example 2.2 (Social Network – Community Detection)

In social network analysis one of the most frequently analyzed properties is the network's community structure. That is, we want to know how many communities there are, how big they are, how they are connected among each other, and how big the overlap of the different communities is. To answer these questions for our social network shown in Figure 1.2, we need to find its communities. A community in a social network is defined to be a group of persons with a high number of friendship relations among each other and a low number of connections to persons in other groups. Hence, community detection in social networks is actually the same as clustering the nodes in the social network graph such that nodes in the same cluster are densely connected and nodes in different clusters have low connectivity. Thus, computing the node clustering of our graph leads to the communities in the corresponding social network. For the network in Figure 1.2, where the communities are highlighted by different colors, we find that there are four major, mainly disconnected communities and one huge cluster consisting of four sub-communities with high overlap. Further, there are several disconnected small groups of one to three persons. All this information could be revealed without knowing any of the persons in the network. We merely used the graph structure to infer the clusters. If we

> *now assume that we have further knowledge about some of the persons, such as where they live, where they work, where they went to school, or what kind of sports they play, then it would even be possible to deduce that the "purple group" are friends knowing each other from work, the "red group" are friends knowing each other form studying abroad, and the huge connected cluster of the "lilac, turquoise, yellow, and green groups" are friends knowing each other from climbing in Bonn, where sub-communities are different training groups preferring different climbing gyms or knowing each other from their studies or jobs.*

Both examples show that we compare graph structure, that is, either entire graphs in the case of structured data or neighborhood structures of nodes in the case of networked data. To be able to develop information propagation methods on graphs (Section 2.3) and techniques to compare graphs or nodes in a graph for learning (Section 2.4), we next provide the main definitions of graph data.

2.1.2 *Definitions*

In this section, we introduce commonly used definitions and terms for graphs, graph types, and node and edge information. A graph is defined as follows.

DEFINITION 2.1 (GRAPH, ADJACENCY MATRIX) *A graph* $G = (V, E)$ *is a set of vertices* $V = \{v_1, v_2, \ldots, v_i, v_j, \ldots, v_n\}$*, also called nodes, and a set of undirected binary edges*[1] $E = \{\{v_i, v_j\} \mid v_i, v_j \in V\}$*. Let the number of nodes be* $|V| = n$ *and the number of edges* $|E| = m$*; then a graph can be represented by its adjacency matrix* $A \in \mathbb{N}^{n \times n}$*, where* $a_{ij} = 1$ *iff* $\{v_i, v_j\} \in E$ *and* $a_{ij} = 0$ *iff* $\{v_i, v_j\} \notin E$*.*

Thus, a graph is fully specified by its adjacency matrix, and a non-zero element a_{ij} of the adjacency matrix indicates an edge between nodes v_i and v_j. Edges connecting a node with itself are called *self-edges* and appear on the diagonal of the adjacency matrix; that is, if v_i has a self edge, then $a_{ii} = 1$. For a graph without self edges the adjacency matrix A has $2m$ non-zero entries. Now we have established the basic representation of a graph. However, there is more to graphs than vertices and edges. In the introduction we already saw that nodes can have types, which is already a simple extension to Definition 2.1. A graph can have various kinds of discrete or continuous information attached to its nodes and edges. When representing and analyzing node information, we distinguish *labels* and *attributes*.

[1] In the context of this thesis we consider edges to be binary; that is, each edge connects two nodes. Graphs with edges connecting more than two nodes, so-called *hypergraphs*, commonly occur in relational domains. For this thesis it is sufficient to approximate such data by graphs with binary edges where one hyperedge connecting k nodes is divided into $k(k-1)/2$ edges connecting every pair of the incident k nodes.

DEFINITION 2.2 (LABELS) *Label information $\ell_i \in \mathbb{N}$ of a node $v_i \in V$ can be specified by a label function $\ell: V \to \mathcal{L}$, where $\mathcal{L} = [k] = \{1, 2, \ldots, k\}$ is a fixed set of k possible labels.*

We distinguish between binary and multi-class labels, where binary means that we have only two classes $\mathcal{L} = \{+, -\}$, and multi-class means that there are $k > 2$ classes. The terms "labels" and "classes" are used interchangeably. We treat the binary case as a special case, because if in the presence of binary labels a node is not of class + (does not have label +) it automatically means that it has to be of class − (has to have label −). In this thesis, we assume each node has exactly one label. Multi-label problems, where a single node can have multiple labels, are not considered (BOUTELL, *et al.*, 2004; READ, *et al.*, 2011).

DEFINITION 2.3 (ATTRIBUTES) *Attribute information $\mathbf{x}_i$ of a node $v_i \in V$ can be specified by a D-dimensional vector of continuous values $\mathbf{x}_i \in \mathbb{R}^D$.*

In this thesis, we consider labels to be categorical and typically one-dimensional. Attributes will typically be multi-dimensional with numerical values. To ease notation, we can subsume the given information on the nodes into the label function $\ell: V \to \mathcal{L}$, where $\mathcal{L} = ([k], \mathbb{R}^D)$. So, a graph with additional information on its nodes is specified by $G = (V, E, \ell)$. In classical machine learning and statistics, the distinction between labels and attributes arises from the fact that some values are given and some are inferred. Labels are usually *dependent variables* that need to be estimated based on the measured attributes, termed *independent variables*. This distinction will not always be important when considering graph data in this work.

So, graph types with respect to the given information on their nodes are:

- *unlabeled graphs*, where no label information is given,
- *labeled graphs*, also *fully labeled graphs*, where each node is endowed with a known label,
- *partially labeled graphs*, where only a subset of nodes has observed labels, and
- *attributed graphs*, where continuous information on the nodes is available.

EXAMPLE 2.3 (SOCIAL NETWORK – LABELS AND ATTRIBUTES)

In our social network introduced in Example 1.2 the nodes labels could be the gender of a person, that is $\mathcal{L} = \{male, female\} = \{1, 2\}$ and the attributes of a person could be their age and height. That is, node v_i representing Anna would have label $\ell_i = 2$ and attributes $\mathbf{x}_i = [27, 1.64]^\top$

where age is given in years and the height is measured in meters. Our social network graph is now an attributed and labeled graph. In the case of missing information, if for instance a person does not reveal her gender, we deal with a partially labeled graph. Similarly, attribute information can also be partially observed.

The types of edge information we will consider are continuous edge weights, discrete edge types, and binary edge directions. Edge weights occur when we want to model the distance or similarity between the nodes. This is naturally the case when the nodes represent points in a metric space, for instance in traffic networks or 3D point clouds. Edge weights will be stored in the *weight matrix*.

DEFINITION 2.4 (WEIGHT MATRIX) *In the presence of edge weights a graph can be represented by its weighted adjacency matrix, or short, weight matrix,* $W \in \mathbb{R}^{n \times n}$. *We define edge weights to be positive; i.e.,* $w_{ij} \geq 0$. *The weight* w_{ij} *of an edge* $\{v_i, v_j\}$ *is proportional (inversely proportional) to the similarity (distance) of its incident nodes.*

That is, the larger the weights, the stronger the connection between two nodes. The smaller the weights, the weaker the connection between the nodes. And in turn a weight of zero indicates that the nodes are disconnected. In the example of a traffic network, large edge weights therefore correspond to short actual distances between two intersections. Intuitively this means that large weights model an easy transition between the two intersections, and small weights reflect low connectivity, and therefore a more difficult transition between nodes.

Edges in a graph can be directed or undirected. Directed edges can be seen as a sort of directed flow such as one way streets in a traffic network. Another example of a directed graph is a webgraph constructed from hyperlinks having an origin (the page where the hyperlink is placed on) and a destination (the page where the hyperlink is linking to).

LEMMA 2.1 *If the edges of a graph are undirected, its adjacency matrix* A *is symmetric. The same holds for the weight matrix* W *of weighted graphs.*

Note that every undirected graph can be viewed as a directed graph where each undirected edge (v_i, v_j) is replaced by two directed edges (v_i, v_j) and (v_j, v_i).

Edge types were briefly mentioned in Example 1.1, where atoms in chemical compounds are connected by single or double bonds. They occur mostly in relational data, where multiple types of relations can model for example similarity relations or purchase relations in a product–user graph. Similarity relations link two items or two users if they are similar, and purchase relations connect a user to a product if the user bought that particular item. Many graph representations – especially

when used in learning with graphs – ignore the edge types once the graph is constructed for the sake of simplicity.

Grouping graphs by the information on their edges gives the following list of graph types:

- *weighted graphs* modeling the similarity/distance between nodes,
- *multi-relational graphs* with typed edges,
- *directed graphs* with edges having an origin and a destination, and
- *undirected graphs* where edges do not have a direction.

When analyzing graphs it is common to measure structure indicators such as the minimum or maximum *node degree*, the number or length of (shortest) *paths* or the number or length of *cycles* in a graph.

Definition 2.5 (Node Degree) *The degree of a node is given by the number of neighbors of that node,* $\deg(v_i) = |v_j \in \mathcal{N}_i|$*, where* $\mathcal{N}_i$ *is the set of nodes connected to* v_i*. The node degree can be computed as the row sum of the adjacency matrix* $\deg(v_i) = \sum_{j=1}^{n} a_{ij}$.

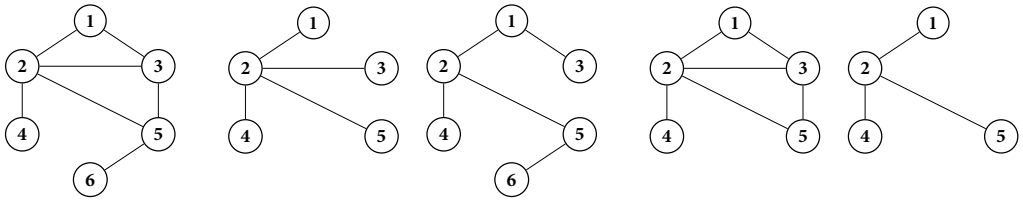
Definition 2.6 (Path, Cycle, Tree) *A path of length* k *is a sequence of vertices* $\{v_1, v_2, v_3, \ldots, v_k, v_{k+1}\}$ *connected by* k *edges* $\{(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_{k+1})\}$ *such that all vertices are distinct. A cycle is a closed path with* $v_1 = v_{k+1}$*. A tree is a graph without cycles.*

Definition 2.7 (Connected Graph) *A graph is connected if there exists a path from any node to any other node in the graph.*

Grouping graphs by their graph structure leads to the following list of graph types:

- *line graphs* or *chains* are connected graphs with a maximum node degree of 2,
- *trees* are connected graphs without cycles,
- *grids* are graphs derived from the graph Cartesian product $G_m \times G_n$ of line graphs consisting of m and n nodes,
- *bounded degree graphs* are graphs where the degree of each node is bounded by a fixed number b,
- *fully connected graphs* are (typically weighted) graphs where each node is connected to every other node in the graph ($\deg(v_i) = n - 1, \forall v_i \in V$), and
- *general graphs* are graphs without any assumption on their structure.

Most real-world graphs are general graphs and when comparing graphs it is common to consider their parts, so-called *subgraphs*.



(a) Graph G (b) Subtrees of G (c) Induced subgraphs of G

Figure 2.1: **Graph, Subtrees, and Induced Subgraphs.** Panel (a) illustrates a graph with six nodes. Panel (b) shows two subtrees of graph G, and panel (c) shows two induced subgraphs of G.

DEFINITION 2.8 (SUBGRAPH, SUBTREE) *$G' = (V', E')$ is a subgraph of graph $G = (V, E)$ if $E' \subset E$ and the incident vertices $V' \subset V$. G' is called a subtree if it does not contain cycles.*

Often we consider *induced subgraphs*, where $V' \subset V$ and E' includes all edges in G that join two nodes in V'. Figure 2.1 illustrates a graph with example subtrees and induced subgraphs. The most straightforward way of comparing subgraphs is to assess whether they are actually "the same", that is, whether they are *isomorphic*. *Graph isomorphism* is defined as follows.

DEFINITION 2.9 (GRAPH ISOMORPHISM) *An isomorphism between two graphs $G^{(1)}$ and $G^{(2)}$ is a bijective mapping $f : V^{(1)} \to V^{(2)}$ between the nodes of the graphs such that any two nodes u, v in the first graph $G^{(1)}$ are connected by an edge if and only if their respective maps $f(u), f(v)$ are connected in the second graph $G^{(2)}$.*

When considering graph isomorphism, graph structure is seen as the arrangement of nodes in the graphs. Such a graph comparison, however, completely ignores additional information on the nodes such as labels and attributes. In this thesis, we will be concerned with kernels for structured and networked data measuring the structural similarity among graphs or among the nodes of a graph by considering the arrangement of node information. Hence, in the context of this thesis graph structure is defined as the interplay of node connections (e.g., links and edges) with the actual information on the nodes, which is, in the simplest case, encoded by numerical labels. Thus, in this case we define graph structure to mean *label structure*. Intuitively label structure takes the node information into account and two graphs with the same edge structure are actually very different if the nodes in the respective graphs have different labels. Figure 2.2 illustrates two graphs with the same edge structure but different label structure. In the context of this thesis, we are interested in measuring and comparing the structure of two graphs

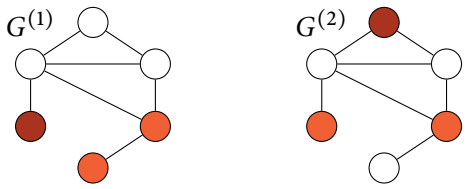


Figure 2.2: **Label Structure.** The two graphs $G^{(1)}$ and $G^{(2)}$ have the same node and edge structure as well as the same number of node labels, however, their label structure is different. Node labels are encoded by color.

or subgraphs by means of their label arrangement. Hence, graph structure in our work implies label structure and in turn, the interplay of edges with the actual information on the nodes, e.g., their labels.

DEFINITION 2.10 (LABEL STRUCTURE) *Label structure is defined as the arrangement of labels in a graph.*

Note that unlabeled graphs are implicitly endowed with labels, namely the node degrees. So, for graphs without label information, we can measure label structure by setting the label function to be $\ell(v_i) = \deg(v_i)$. In this thesis, we will derive methods to represent graph structure of (partially) labeled graphs by their label structure.

2.1.3 Graph Construction and Analysis

So far, we have defined graphs and introduced several types of graphs. Further, we introduced graph structure as a key concept to study graph similarity and in turn to achieve learning with graphs. Graphs arise in several contexts: graphs model the dependencies among data points (networked data), graph model the structure of objects (structured data), graphs model the dependencies of variables in statistical models, so-called *graphical models*, graphs can be generated from random graph models such as Erdős–Rényi random graphs or Barabási–Albert random graphs, and graphs can be created from a distance or similarity measure among data points. In such metric-induced graphs, the nodes are points living in a metric space and the edges can be derived from the distance or similarity of the points' attributes by considering nearest-neighbor graphs or fully connected graphs with appropriately weighted edges. These graph representations are often used in semi-supervised learning (CHAPELLE, *et al.*, 2006).

As graph domains can be of such different nature, graph construction is inherently application dependent (LIU and KIRCHHOFF, 2012; MAIER, *et al.*, 2008; ZHU, 2005; LIU, 2006) and an in-depth discussion of graph construction goes beyond the

scope of this background chapter. In general, graph construction can be divided into two steps: establishing the graph topology and quantifying the graph. The first step determines the nodes and edges in the graph, and the second step assigns labels and attributes to the nodes and weights, types, and directions to the edges. Once a graph is constructed, we can assume that all relevant structural information has been embedded. To make use of the graph data for learning, however, the encoded information can – in contrast to vector-valued data – not be used directly. Thus, further analysis on the graph(s) has to be performed in order to identify properties or derive approximations of simpler structure. Graph analysis can be tackled by approaches from different fields: graph mining (COOK and HOLDER, 2006; HADZIC, *et al.*, 2010; HAN and KAMBER, 2006), spectral graph theory (VON LUXBURG, 2007; CHUNG, 1997), random field theory (Markov random fields, also called Markov networks, cf. Chapter 4 in (KOLLER and FRIEDMAN, 2009) and Chapter 8 in (BISHOP, 2006), and Gaussian random fields (ZHU, 2005)), and matrix approximation and factorization (SUN, *et al.*, 2008; MENON and ELKAN, 2011; YANG, *et al.*, 2012). The contributions in this thesis can be linked to both graph mining and random field theory. In general, the considered technique for graph analysis has to fit the graph domain it is applied to and in turn needs to match the learning task to be solved. In general, we distinguish *single-graph domains*, where we have one connected or disconnected graph and reasoning is preformed on the node level, from *multiple-graph domains*, where each graph represents one data point and learning and inference is done on the graph level. The following section first introduces machine learning in general and then common learning tasks occurring on both levels.

2.2 Machine Learning with Graphs

One of the main goals of this thesis is to apply statistical machine learning to graph data. In Examples 2.1 and 2.2 the tasks of graph classification and node clustering were illustrated. In this section, we introduce the main concepts and terminology of statistical machine learning and define learning tasks involving graph data. Statistical machine learning can be divided into two areas: *supervised learning* and *unsupervised learning*. Supervised learning tasks are *classification* and *regression*, and examples for unsupervised learning are *clustering* and *ranking*. In fact, we will see that all these classical machine learning tasks can be applied to single- and multiple- graph domains. Notation and content in this section largely follow (HASTIE, *et al.*, 2009) with parts adapted from (BISHOP, 2006; RASMUSSEN and WILLIAMS, 2005).

2.2.1 *Supervised Learning*

Supervised learning is the problem of learning a mapping from input to output values from empirical data. Input data instances with observed output values have to be generalized to predict the responses for data instances with unknown target

values. The input–output pairs with observed targets are called *training data* and new inputs for which we want to make predictions are called *test data*. The general assumption is that there is an unknown function $f_{true}\colon \mathcal{X} \to \mathcal{Y}$ mapping from the input space $\mathcal{X}$ to the target space $\mathcal{Y}$. In this work, we will consider one-dimensional target spaces. The learning task is to find a good approximation of this function in order to predict the target values of unknown test cases. If the target space $\mathcal{Y}$ is continuous, that is $\mathcal{Y} = \mathbb{R}$, the learning task is called *regression*. If it is discrete, the task is called *classification*. For classification we assume that $\mathcal{Y}$ is a finite set of k classes. If $k = 2$ the problem is termed *binary classification*. The classes are also called *class labels* or just *labels*; cf. Definition 2.2, where we defined labels for the nodes of a graph. In classification in general, the observed data is often called labeled data, whereas the unobserved data points are considered unlabeled. In classical machine learning, $\mathcal{X}$ is a space of D-dimensional vectors, $\mathcal{X} = \mathbb{R}^D$, called *feature space* and each coordinate x_i of a data point $\mathbf{x} \in \mathcal{X}$ is a *feature*. Features can be real, ordinal, or discrete. Sometimes features are also referred to as *attributes*; cf. Definition 2.3, where we defined attributes for the nodes of a graph.

Viewing supervised learning from a statistical perspective, we assume that our data comes from a statistical model. Thus, there is a statistical dependency between the input vector $\mathbf{x} \in \mathcal{X}$ and the output $y \in \mathcal{Y}$, and we want to uncover this relationship. This is usually modeled by an *unknown* function f and an additive noise component ε such that

$$Y = f_{true}(X) + \varepsilon, \tag{2.1}$$

where Y and X are random variables. ε models any kind of departure from the true function such as measurement errors, and we assume that ε has an independent, identically distributed Gaussian distribution with zero mean and variance σ_n^2. As Equation (2.1) is a statistical model, $f_{true}(X)$ can be seen as the expected value of the conditional probability distribution $\Pr(Y \mid X)$, hence $f_{true}(\mathbf{x}) = \mathrm{E}(Y \mid X = \mathbf{x})$.

In this thesis, we focus on *discriminative models* for learning, as we are interested in modeling $\Pr(Y \mid X)$ in order to make predictions about y from $\mathbf{x}$. Solving discriminative learning tasks does not require the joint distribution $\Pr(X, Y)$. Note that if the task is to generate $(\mathbf{x}, y)$ samples then this joint distribution needs to be modeled; such approaches are called *generative models*. The most prominent generative model is the *hidden Markov model*, see for example Chapter 13 in (BISHOP, 2006). It can be used to recover data sequences not being directly observable. Applications tasks are for instance speech recognition, machine translation, or part-of-speech tagging in natural language processing.

The general task of discriminative machine learning is now to move from the finite set of observed data $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$ to a function f that predicts the output for unseen input values $\mathbf{x} \in \mathcal{X}$. This is usually done by a computer program analyzing the difference of the function values on the training data to the observed

data, $y_i - f(\mathbf{x}_i)$, and modifying the function f accordingly. This procedure is called *learning*. There is no chance to uniquely determine the *true* function f_{true} since infinitely many solutions will match the measured input–output pairs. In other words, the learning problem is ill-posed. Thus, an approximation of f_{true} has to be found, for example by restricting the number of considered functions. In general there are two approaches, *parametric models* limit the class of functions to be parameterized by a fixed-dimensional vector. *Non-parametric models*, on the other hand, allow the dimension of the parameterization to grow with the observed data. Some non-parametric models regularize the functions considered according to their complexity or their contribution towards learning the model. Before we discuss parametric and non-parametric learning in detail, we briefly introduce the inductive and transductive learning paradigms.

INDUCTIVE VS. TRANSDUCTIVE LEARNING

Approaches where a mapping function is learned from the training data that is applicable to *any* test data is called *inductive learning*. Reasoning can also be performed directly from the training data to a *specific* set of test points without learning a general mapping function. This approach is called *transductive learning* (VAPNIK, 1998). Assuming a graph database with a set of fixed graphs, the approaches to learning with graphs on the graph level introduced in this work are considered inductive. Learning on the node level, however, is transductive in the sense that the whole graph including the unlabeled nodes is used for inference or in order to establish a new feature representation. In the latter approach, once the new feature representation is established, an inductive learner is used for making predictions. Therefore, we focus on inductive learning algorithms in the overview chapter.

PARAMETRIC MODELS

A common parametric approach is to assume a linear approximation for f:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_D x_D = w_0 + \mathbf{x}^\top \mathbf{w},$$

where D is the dimension of the inputs and w_0 and $\mathbf{w}$ are parameters. In the following, we omit the special treatment of the intercept w_0 by either shifting the data to have zero mean prior to learning or by including an all ones variable into the training data so that the first dimension of $\mathbf{x}$, $x_1 = 1$ for all data points. The standard linear model, however, can easily be too restrictive and we have to use a more general basis function model, which takes f to be a linear combination of the so-called basis functions

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}, \tag{2.2}$$

where $\boldsymbol{\phi}$ maps the D-dimensional input vector $\mathbf{x}$ into an N-dimensional feature space. The parameter vector $\mathbf{w}$ is of length N now, and once the basis functions $\phi_i(\mathbf{x})$ are determined the model is linear in the new variables. Example basis functions are polynomials, e.g., $\phi_i(\mathbf{x}) = x_j^2$ or $\phi_i(\mathbf{x}) = x_j x_k$, or other nonlinear transformations of single or multiple input dimensions such as log or square root transformations, $\phi_i(\mathbf{x}) = \log(x_j)$ or $\phi_i(\mathbf{x}) = \sqrt{x_j}$. It is also possible to define basis functions that only consider specific input regions, inducing piecewise constant models. The linear basis expansion still leads to a parametric model as long as the number of bases and in turn the number of parameters is independent of the size of the training data. For parametric models, the training data is only used to learn the parameter vector $\mathbf{w}$ which is commonly done by minimizing a *loss function* L:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} L(y_i, \boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w}) \;=\; \min_{f} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)), \tag{2.3}$$

where f is defined by Equation (2.2) and a set or class of possible basis functions. An example loss function, usually used for regression data, is the squared loss, $L(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$. Applying this loss and solving Equation (2.3) leads to the least-squares regression model,[2] a very popular and simple parametric model. Parametric models discard the training data after learning and thus are usually fast in making predictions at the cost of posing stronger model assumptions.

NON-PARAMETRIC MODELS

Non-parametric models, on the contrary, make less assumptions about the data distribution; however, the parameter vector is of the same size as the training data. Non-parametric models are therefore also called *memory-based* models as all the training data is kept and used for making predictions. In a nearest-neighbor model, for example, all training points are considered to find the nearest ones to a given test point, which are then used to derive its predictions. Such models have some obvious drawbacks, for example, they fail in high-dimensional input spaces as the nearest neighbors might actually not be close to the target point. Second, they do not consider a structural form of the underlying predictive function, which might be known. So, to overcome these problems and to be able to learn a model, we again try to learn a function, but we have to control the complexity of the considered functions. Instead of only allowing specific (sets of) basis functions ϕ_i, however, we directly impose restrictions on the functions f. In the following, we will introduce *regularization methods*, where model restriction is achieved by adding a penalty term to the loss function:

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda J(f). \tag{2.4}$$

[2]The least squares estimate of $\mathbf{w}$ is given by $\hat{\mathbf{w}} = (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{y}$, where $\Phi_{ij} = \phi_j(x_i)$.

$J(f)$ is a functional defined on a function space $\mathcal{F}$ such that it results in lower values for smooth functions f and larger values if f varies too rapidly over small regions of the input space. λ is a weight parameter controlling the importance of the regularizer. The representer theorem, which will be stated in Theorem 2.2 and discussed in more detail in Section 2.4, ensures that, although the criterion in Equation (2.4) is defined over an infinite-dimensional space of functions, its solution is finite-dimensional. The non-linear least-squares regression model can be extended to the regularized least-squares model, also known as smoothing splines. If the regularized least-squares model is used for a classification task, it is called logistic regression model. We will extend non-parametric learning to kernel-based learning in Section 2.4. Now, we will briefly introduce another basic and important learning paradigm, namely *Bayesian learning*.

BAYESIAN LEARNING

Bayesian machine learning approaches apply the concept of Bayesian statistics to perform inference. A prior belief about the model is updated as additional evidence, usually in the form of training data, is available. Bayesian inference algorithms follow three steps:

- Specify a *prior* over the parameters, expressing your beliefs about the unknown parameters before having seen any observations.
- Calculate the *likelihood* of the data which in words is the probability density of the observations regarded as a function of the model parameters.
- Determine the *posterior* over the parameters given the data using Bayes' Rule: $\Pr(A \mid B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$.

Assuming that the training data is independent, the likelihood can be written as

$$\Pr(\mathbf{y} \mid X, \mathbf{w}) = \prod_{i=1}^{n} \Pr(y_i \mid \mathbf{x}_i, \mathbf{w}),$$

where X is the $D \times n$ matrix of training inputs sometimes also referred to as *design matrix* and $\mathbf{y}$ is the vector of training outputs. Now, we can calculate the posterior distribution as

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad \Pr(\mathbf{w} \mid \mathbf{y}, X) = \frac{\Pr(\mathbf{y} \mid X, \mathbf{w})\Pr(\mathbf{w})}{\Pr(\mathbf{y} \mid X)},$$

where the marginal likelihood $\Pr(\mathbf{y} \mid X) = \int \Pr(\mathbf{y} \mid X, \mathbf{w})\Pr(\mathbf{w})\mathrm{d}\mathbf{w}$ acts as a normalizing constant to ensure that $\Pr(\mathbf{w} \mid \mathbf{y}, X)$ has an integral of 1. This posterior distribution combines information about the parameters deduced from the

training data (i.e., the likelihood) and the background knowledge (i.e., the prior). Analogue to the above, one could also use the same technique to infer which model is the best given the data. This second level of inference is for example described in (MACKAY, 1992). To make predictions for an unseen test case $\mathbf{x}_*$, we have to consider the predictive distribution for $f(\mathbf{x}_*) = f_*$. That is to say, we integrate the predictions of the model with respect to the posterior distribution of the parameters

$$\Pr(f_* \mid \mathbf{x}_*, X, \mathbf{y}) = \int \Pr(f_* \mid \mathbf{x}_*, \mathbf{w}) \Pr(\mathbf{w} \mid X, \mathbf{y}) \mathrm{d}\mathbf{w}.$$

The above specified predictive distribution supplies the complete Bayesian inference and its mean can be used as prediction, thus the approximation for f at $\mathbf{x}_*$.

2.2.2 *Unsupervised and Semi-supervised Learning*

In *unsupervised learning*, there are no output observations and the task is to organize or group the data in a meaningful way. Unsupervised learning tasks include *clustering* or *ranking*. The former means that data instances are grouped into so-called clusters such that there is high within-cluster similarity, and such that the clusters themselves are well separable. In ranking, data instances are ranked according to some score measured for all data points. In many applications we are interested in a *personalized ranking*, that is, a ranking according to the preferences of a user or some given query data. Now, the ranking score is influenced by the similarity of the data points to the query. A famous example problem of this kind is ranking webpages according to a user provided search string. If the task is to retrieve similar data points to a given query example while ignoring the ordering in the output cluster, then we speak of *clustering on demand*. This is often achieved by thresholding a personalized ranking result. Another term describing the problem of finding similar items to a possibly small set of query items is *set completion*. We will discuss a solution to the problem of set completion using structural graph features in relational data in Chapter 8.

Approaches in *semi-supervised learning* use both labeled and unlabeled data for supervised tasks like classification. A detailed introduction of semi-supervised learning concepts and various state-of-the-art methods can be found in (CHAPELLE, *et al.*, 2006). Besides approaches directly implementing the low-density separation assumption (CHAPELLE, *et al.*, 2008; SINDHWANI, *et al.*, 2005; JOACHIMS, 1999), graph-based methods are actively developed (ZHU, 2005; ZHOU, *et al.*, 2003; ZHU, *et al.*, 2004). That is, supervised learning is applied to single-graph domains in such a way that a metric-induced graph is created from in depended data points and their attributes. Then, in addition to the labeled data points, the graph structure is leveraged for prediction. Hence, semi-supervised learning is very similar to transductive learning and, in turn, graph-based semi-supervised learning is closely related to the approaches developed in Chapter 7 of this thesis.

2.2.3 *Learning Tasks for Graph Data*

In the beginning of this thesis, we introduced networked data, where instances are linked by relations (single-graph domains), and structured data, where each entity is represented by a graph (multiple-graph domains). Now, we take a different view and distinguish two learning levels: learning on the *node level* and on the *graph level*. These two levels of learning tasks can naturally be brought in line with the different graph domains. Learning on the node level is performed for networked data, whereas learning on the graph level is applied to structured data.

NODE LEVEL

Learning in single graph domains is also referred to as within-network classification or within-network relational learning (XIANG and NEVILLE, 2008; SUTTON and MCCALLUM, 2006). Learning tasks on the node level essentially applies supervised and unsupervised learning to the nodes of a graph and prominent examples are:

- node classification,
- node regression,
- node clustering[3] (community detection),
- node clustering on demand (set completion),
- node ranking, and
- edge prediction (also called link prediction).

The main assumption when tackling learning tasks on the node level is the *homophily* assumption. Homophily is a concept from sociology and is defined as follows (EDITORS OF THE AMERICAN HERITAGE DICTIONARIES, 2000).

DEFINITION 2.11 (HOMOPHILY) *Homophily is a theory in sociology that people tend to form connections with others who are similar to them in characteristics such as socioeconmic status, values, beliefs, or attitudes.*

Although originally developed in social sciences, the homophily assumption naturally carries over to not only social networks (MCPHERSON, *et al.*, 2001), but also to many other domains of networked data. This is obvious for graph representations formed by obeying some similarity measure among the data points. However, it also holds for webgraphs, as links are placed according to some content-based similarity, or traffic networks, where streets connect nearby, that is, spatially close, places.

Thus, node-level learning and inference can often be approached by exploiting – or more precisely reversing – the homophily definition, leading to the homophily

[3] In the literature node clustering is often spuriously termed "graph clustering" as the actual task of clustering graphs is rather rare.

assumption, also *termed autocorrelation* assumption in the literature on graph-based machine learning (NEVILLE and JENSEN, 2005).

HYPOTHESIS 2.1 (HOMOPHILY/AUTOCORRELATION ASSUMPTION) *Nodes that are close to one another in the graph are likely to*

- *be similar,*
- *have the same properties,*
- *have the same label,*
- *be in the same cluster, or*
- *form an edge.*

So, learning on the node level will be heavily based on measuring closeness in the graph. One popular and successful technique to assess the proximity of nodes in a graph comes from the intersection of graph theory and probability theory, namely *random walks*. Before introducing random walks in the next section, we will now change our view on learning tasks for graph data from the node to the graph level.

GRAPH LEVEL

Learning in multiple-graph domains is performed on the graph level and is also called across-network learning, multiple-network learning, or learning with structured data. Learning tasks on the graph level are:

- graph classification,
- graph regression,
- graph clustering (not to be confused with node clustering), and
- graph ranking.

Rather than measuring closeness in the graph, we now measure the similarity of entire graphs. We can achieve this, for example, by comparing various characteristics of the graphs in the dataset. This approach is closely related to *graph mining*, a data mining approach to find interesting or frequent patterns in graphs (HAN and KAMBER, 2006). The simplest way of comparing patterns across graphs is to count their occurrences per graph. This approach establishes an explicit feature transformation turning the graph data into vector-valued data points (of counts). Now the new feature representation computed for each graph can be used in combination with any classical machine learning technique. It is also possible to design kernels among graphs and perform kernel-based learning. This approach will be discussed in Section 2.4.

This thesis will approach tasks on both levels. Doing so will reveal that it is possible to take surprisingly similar approaches to tackle problems on both levels.

In Part I, we will be concerned with learning on the graph level. Especially, we will tackle graph classification (and ranking) by developing a novel kernel framework between graphs (Chapters 4 and 5). In Part II, we will then transfer the developed ideas to learning on the node level. In particular, we will investigate node classification in sparsely labeled graphs (Chapter 7) and study a similar task, namely set completion, in the more general context of relational data (Chapter 8). To understand the contributions of this thesis in a broader context, this section about machine learning with graphs is concluded by changing the perspective from learning with graphs to to a more general view on related research fields.

2.2.4 The More General Perspective: Related Research Fields

Learning with graphs largely depends on graph analysis, which we will tackle in this work by means of graph mining and random walks. Taking a more general view, learning with graphs in general is also related to other research directions, such as graph matching, graph partitioning, learning and inference in graphical models, statistical relational learning and collective inference, and information retrieval. In the following, we will briefly discuss these related research fields; an in-depth introduction, however, goes beyond the scope of this thesis.

Graph matching is closely related to learning tasks on the graph level such as graph classification. In its most general definition, it is the problem of finding a matching between the nodes and edges of two graphs. The simplest form of graph matching when considering unlabeled graphs is constructing an *isomorphism* between two graphs, cf. Definition 2.9. Although the assessment of graph isomorphism is a problem of great interest in graph theory, it is not very useful for learning as the isomorphism test would only provide us with the information whether or not two graphs are isomorphic. For machine learning in general and in this thesis in particular, however, we are more interested in how similar two graphs are. Thus, the notion of similarity is relaxed beyond strict isomorphism. Inexact or approximate graph matching for (labeled) graphs is more useful in this context (GORI, *et al.*, 2005; WASHIO and MOTODA, 2003; DEPIERO, *et al.*, 1996; UMEYAMA, 1988). An overview of various graph matching problems and a survey of existing solution approaches is given in (GALLAGHER, 2006).

Learning on the node level, especially the task of node clustering, is highly related to *graph partitioning*. In graph partitioning the problem is to group together nodes in a graph such that certain properties hold. A classical problem is, for example, given a source and sink node, to find a partitioning separating them such that a minimum number of edges is "cut". By solving an energy minimization problem, graph cuts provide a popular approach to achieve a good solution (BOYKOV, *et al.*, 2001). Practical applications can be found in many computer vision tasks such as the stereo correspondence problem or image segmentation. Another interesting closely related problem is local graph partitioning, where the goal is to efficiently

find a group of nodes near a starting vertex in a possibly very large graph (ANDERSEN, *et al.*, 2006). Graph partitioning can also be achieved by spectral methods. Spectral graph partitioning uses the eigendecomposition of the graph Laplacian and especially the eigenvector corresponding to the second smallest eigenvalue, the so-called Fiedler vector, to bisect the nodes of a graph (BOLDI, *et al.*, 2006; JOACHIMS, 2003). The graph Laplacian will be introduced in Section 2.4.2 in the context of kernels on graphs; an introduction to spectral graph theory is provided by CHUNG (1997).

Graphs also model the inference structure of statistical models, that is the nodes constitute variables and the edges represent (conditional) dependencies of the variables. Examples of such probabilistic graphical models, commonly modeled as factor graphs, are Bayesian networks, Markov networks (also Markov random fields), and dependency networks. *Learning and inference in graphical models* is a growing research field with numerous modeling approaches, solution algorithms, and applications; a comprehensive overview can be found in (KOLLER and FRIEDMAN, 2009).

In *statistical relational learning* and *collective inference*, variables and their dependencies are again modeled by graphs. Instead of grounded models, where each variable reflects one entity of interest, relational learning assumes that data instances are only particular instances of general variables. Thus, probabilistic graphical models are extended to relational variants, such as relational Bayesian networks, relational Markov networks, relational dependency networks, and Markov logic networks. Section 2.5 will introduce relational data in more detail. Introductions to various statistical relational learning approaches are provided in (GETOOR and TASKAR, 2007). One of the main differences to graph-based machine learning and collective inference is that in the latter latent variables are used to model various dependencies and in turn simultaneous statistical judgments about the same variables for a set of related data instances is achieved (JENSEN, *et al.*, 2004; NEVILLE and JENSEN, 2005). Whereas collective inference can result in more accurate predictions than conditional inference for each instance independently, it comes at the cost of more complicated data representation and learning procedures.

Information retrieval is the problem of finding suitable information in text documents, web pages, or image or video databases, given a search query. Information retrieval is closely related to learning with graphs, as graphs can be used to model the relatedness of documents and the link structure of webpages (LIU, 2006), or the content of a single document (JIANG, *et al.*, 2010). Nowadays web search is one of the most prominent information-retrieval applications, intersecting text mining and network analysis. An introduction to this problem and general approaches to information retrieval can be found in (MANNING, *et al.*, 2008).

2.3 Information Propagation on Graphs

Learning with graphs is an active and well established area in machine learning and it turns out that many of the node-level tasks listed in the previous section can be tackled by means of the same technique, namely *random walks*. Random walks formalize the proximity of nodes in a graph probabilistically and thus are suitable to imply the homophily assumption stated in Hypothesis 2.1. Learning algorithms based on random walks have proven powerful for node classification (WU, *et al.*, 2012; ZHU, *et al.*, 2003; SZUMMER and JAAKKOLA, 2001), node clustering (FORTUNATO, 2010; PONS and LATAPY, 2006; TISHBY and SLONIM, 2000), link prediction (LÜ and ZHOU, 2011; BACKSTROM and LESKOVEC, 2011), and node ranking (HOTHO, *et al.*, 2006; DE KNIJF, *et al.*, 2011). Probably the most famous random walk-based algorithm is Google's PageRank (BRIN and PAGE, 1998) developed to rank webpages seen as nodes of the webgraph to present relevant search results. In this work, we will use random walks to model information propagation schemes on graphs in order to derive features representing both global as well as local graph structure for learning on the graph and node levels. In the following, we will define Markov random walks on graphs and introduce several algorithms implementing iterative information propagation schemes.

2.3.1 *Random Walks*

Before developing the technical details, let us first provide an intuition. Consider a particle traveling from node to node via the edges of a graph such that the decision of where to go next only depends on its current location. That is, the next node to visit is selected proportional to the number of edges and edge weights connecting the current node to its neighbors. This is a Markov random walk on a graph. We can modify the walk behaviour by introducing *restarts*. When considering restarts, in each step of the walk, the particle will with a certain probability be relocated to a restart node independent of the edge structure. The restart node is selected uniformly at random or again with a given probability from a predefined set of restart nodes. After the so-called teleportation, the random walk with restarts continues according to its definition. This section is mainly based on (LOVÁSZ, 1996; WU, *et al.*, 2012) and Chapter 9 in (BOLLOBÁS, 1998).

GENERAL DEFINITION

A random walk on a graph G is a Markov process $X = \{X_t \mid t \geq 0\}$ with a given initial state $X_0 = v_i$. We will also write $X_t^{(i)}$ to indicate the walk began at v_i. Each X_t is a random variable and the probability that the walk jumps from v_i to v_j, i.e., the transition probability $\tau_{ij} = \Pr(X_{t+1} = v_j \mid X_t = v_i)$, only depends on the direct neighborhood of the current state $X_t = v_i$ and not on past states. In other words, the walk is memoryless. The one-step transition probabilities τ_{ij} for all pairs of nodes in V can be represented by the row-normalized weight or adjacency matrix

called *transition matrix*,

$$T = D^{-1}W, \text{ where } D = \operatorname{diag}\left(\sum_{j=1}^{n} w_{ij}\right). \tag{2.5}$$

DEFINITION 2.12 (RANDOM WALK) *Let $G = (V, E)$ be a graph and $X = \{X_t \mid t > 0\}$ a Markov process on the nodes V with transition probabilities given by the transition matrix T defined in Equation (2.5). X is called a (Markov) random walk on G if $P(X_{t+1} = v_j \mid X_t = v_i) = \tau_{ij}$.*

Since each row in T sums to one, the transition matrix is *row stochastic*. Further, the t-step transition probability between two nodes v_i and v_j can be computed using the matrix power

$$\Pr(X_t = v_j \mid X_0 = v_i) = \Pr(X_t^{(i)} = v_j) = \left(T^t\right)_{ij}, \tag{2.6}$$

where $T^0 = I$. If graph G consists of several components and the nodes v_i and v_j are disconnected, then $\Pr(X_t^{(i)} = v_j) = 0, \ \forall t$.

Now, assume that G is connected and non-bipartite, that is, V cannot be divided into two disjoint sets V_1 and V_2, where every edge in E exclusively connects nodes in V_1 to nodes in V_2. Further, let the starting node be chosen according to some initial probability distribution $\boldsymbol{\pi}_0$, for instance uniformly at random; i.e., $\boldsymbol{\pi}_{0,i} = \Pr(X_0 = v_i) = 1/n$. Then the probability distribution of state X_t, which is given by $\boldsymbol{\pi}_t = [\pi_{t,1}, \pi_{t,2}, \ldots, \pi_{t,n}]^\top \in \mathbb{R}^n$ with $\sum_i \pi_{t,i} = 1$, is defined according to the following recursion:

$$\begin{aligned} \boldsymbol{\pi}_t &= T^\top \boldsymbol{\pi}_{t-1}, \\ \boldsymbol{\pi}_t &= \left(T^\top\right)^t \boldsymbol{\pi}_0, \end{aligned} \tag{2.7}$$

where $\boldsymbol{\pi}_t = \Pr(X_t)$. For $t \to \infty$, $\boldsymbol{\pi}_t$ converges to a stationary distribution $\boldsymbol{\pi}$ with $\pi_i \propto \frac{\deg(v_i)}{2m}$. The discrete time Markov process, also called Markov chain, is *ergodic*.

We can also compute the probability distributions of state X_t given all possible starting states X_0 by matrix multiplication. Let $P_t \in \mathbb{R}^{n \times n}$ where $(P_t)_{ij} = \Pr(X_t = v_j \mid X_0 = v_i) = \Pr(X_t^{(i)} = v_j)$ and $P_0 = I$, then

$$P_{t+1} = TP_t. \tag{2.8}$$

Note that $(P_t)_{i,:} = \Pr(X_t^{(i)})$; that is, each row in P_t is the probability distribution of X_t for a walk starting in vertex v_i. Equation (2.8) is equivalent to Equation (2.6). Now, $\boldsymbol{\pi}_t$ in Equation (2.7) can be obtained from P_t by averaging its rows weighted by the initial distribution $\boldsymbol{\pi}_0$; that is, $\boldsymbol{\pi}_t = P_t^\top \boldsymbol{\pi}_0$.

RANDOM WALKS WITH RESTARTS

To model *random walks with restarts*, the most simple approach is to introduce a new transition matrix modeling the desired properties of the walk. First, let $Q \subset V$ be the set of restart nodes and let $\mathbf{q} \in \mathbb{R}^n$ indicate the members of Q, where $q_i > 0$ for $v_i \in Q$ and $\sum_i q_i = 1$. That is, $\mathbf{q}$ is a probability distribution among restart nodes. Further, let the transition matrix T be defined as in Equation (2.5). For a given restart probability $\beta \in [0,1]$, Equation (2.7) turns into

$$\boldsymbol{\pi}_t = \beta\, T^\top \boldsymbol{\pi}_{t-1} + (1-\beta)\, \mathbf{q}.$$

This is analogous to adapting the transition matrix to

$$\bar{T} = \beta\, T + (1-\beta)\, E, \tag{2.9}$$

where $E = \mathbf{e}\mathbf{q}^\top$ with $\mathbf{e} \in \mathbb{R}^n$ being the all ones vector; i.e., $e_i = 1, \forall i$.

DEFINITION 2.13 (RANDOM WALK WITH RESTART) *Let $G = (V, E)$ be a graph and $X = \{X_t : t > 0\}$ a Markov process on the vertices V. Let $\bar{T}$ be a transition matrix as defined in Equation (2.9) with $\beta \in [0,1]$. Then X is called a random walk with restart if $P(X_{t+1} = v_j \mid X_t = v_i) = \bar{\tau}_{ij}$.*

2.3.2 *Algorithms based on the Random Walk Model*

In the following, we review several algorithms based on the concept of Markov random walks (RWS). So far, we have only talked about random walks on the nodes, ignoring their label information. To employ random walks for node classification on partially labeled graphs or to measure label-structure similarity, the walk has to be lifted from being on the nodes of a graph, cf. Equation (2.8), to being on its labels.

LABEL DIFFUSION

Consider a (partially) labeled graph without attributes, $G = (V, E, \ell)$, where $V = V_L \cup V_U$ is the union of labeled and unlabeled nodes, respectively, $\ell\colon V \to [k]$ is a label function with known values for the nodes in V_L, and k is the number of available labels. Note that for unlabeled graphs we can set the label function to be $\ell(v_i) = \sum_j A_{ij} = \deg(v_i)$ and hence for fully labeled and unlabeled graphs we have $V_U = \varnothing$. Let the matrix $P_0 \in \mathbb{R}^{n \times k}$ give the prior label distributions of all nodes in V. If node $v_i \in V_L$ is observed with label $\ell(v_i)$, then the ith row in P_0 can be conveniently set to a Kronecker delta distribution concentrating at $\ell(v_i)$; i.e., $(P_0)_i = \delta_{\ell(v_i)}$. Thus, the simplest label-based RW on a graph is a *diffusion process*:

$$P_{t+1} \leftarrow T P_t, \tag{2.10}$$

where $(P_t)_i$ gives the distribution over $\ell(X_t^{(i)})$ at iteration t.

This diffusion process, also called isotropic diffusion in image processing and computer vision, is for example used for image denoising (JÄHNE, 2005). In machine learning it has been introduced for semi-supervised classification (SZUMMER and JAAKKOLA, 2001).

LABEL PROPAGATION

Another algorithm developed for graph-based semi-supervised learning is label propagation (ZHU, *et al.*, 2003). To infer the labels of the unlabeled data points, label propagation performs a random walk starting in each unlabeled node that stops when hitting a labeled point. For a binary classification task, for example, we are interested in the probability of hitting a positive labeled point before a negative labeled point. Assuming that labels vary smoothly on the graph, cf. Hypothesis 2.1, we will then assign the label with the highest probability.

The label probability matrix $P_0 \in \mathbb{R}^{n \times k}$ is initialized as follows. If node $v_i \in V_L$ is observed with label $\ell(v_i)$, then the ith row in P_0 is the Kronecker delta distribution concentrating at $\ell(v_i)$; i.e., $(P_0)_{i,:} = \delta_{\ell(v_i)}$. We initialize the label distributions for the unlabeled nodes V_U with some prior, for example a uniform distribution.[4] The ith row of P_0 now gives the initial probability distribution for the first label encountered, $\ell(X_0^{(i)})$, for a random walk of length 0 starting at v_i. To model the fact that we are only interested in the first label encountered, we will have to push back the labels of the nodes in V_L after each propagation step. So, let $P_0 = \left[P_{0,[labeled]}, P_{0,[unlabeled]}\right]^\top$ be the initial label distributions of all nodes in the graph, where the distributions in $P_{0,[labeled]}$ are those of the observed nodes and $P_{0,[unlabeled]}$ are the distributions of the unobserved nodes. Now, label propagation is defined by

$$\begin{aligned} P_{t,[labeled]} &\leftarrow P_{0,[labeled]}, \\ P_{t+1} &\leftarrow T\,P_t. \end{aligned} \tag{2.11}$$

ZHU, *et al.* (2003) have shown that the solution to this process corresponds to the minimum energy in a Gaussian random field. The classification algorithm for Gaussian fields can be viewed as a form of nearest neighbor approach, where the nearest labeled examples are computed in terms of a random walk on the graph.

PAGERANK

PageRank was originally introduced to rank webpages according to the structure of the webgraph (BRIN and PAGE, 1998). Webpages with a larger number of links pointing to them get ranked higher. Apart from its intended use, PageRank is a powerful method for general information retrieval tasks and due to its popularity

[4] This prior could also be the output of an external classifier built on additionally available node information such as attributes.

highly efficient algorithms for its computation have been engineered (LANGVILLE and MEYER, 2006). Intuitively PageRank tries to model a random surfer following the edges in a graph (the hyperlinks in the web). Given the graph's transition matrix T and $E = 1/n\,\mathbf{e}\mathbf{e}^\top$

$$\boldsymbol{\pi}_t^\top = \boldsymbol{\pi}_{t-1}^\top \tilde{G}, \tag{2.12}$$

where $\tilde{G} = \beta T + (1-\beta)E$ is called *Google matrix* and $\beta \in [0,1]$ controls the probability that the random surfer will not follow the edge structure but jump to a random node (webpage). The PageRank vector is the solution of the converged power method applied to $\tilde{G}$. A comprehensive derivation and various computation techniques can be found in (LANGVILLE and MEYER, 2006).

One extension to this basic PageRank model is the introduction of a probability vector $\mathbf{v}$, also called *personalization vector* controlling the random restarts. Given $\mathbf{v}$, the teleportation matrix in Equation (2.12) is now $E = \mathbf{e}\mathbf{v}^\top$. This model, called *personalized PageRank* (HAVELIWALA, *et al.*, 2003), is in fact equivalent to the random walk with restarts introduced in Definition 2.13. Personalized PageRank or random walks with restarts are common approaches to rank items in information retrieval (TONG, *et al.*, 2006; LIN and COHEN, 2010*b*). Replacing the PageRank vector by a label probability vector leads to an approach for semi-supervised learning called learning with local and global consistency (ZHOU, *et al.*, 2003). This approach is sometimes also called *label spreading* and is again closely related to label propagation (BENGIO, *et al.*, 2006).

Another algorithm, based on the idea of propagating information in directed networks representing the web, is the hyperlink-induced topic search (HITS) algorithm, also known as *hubs and authorities* (KLEINBERG, 1999).

2.3.3 Steady-State Distributions vs. Early Stopping

Assuming connected and non-bipartite graphs, all random walks introduced previously converge to a unique steady-state distribution $\boldsymbol{\pi}_\infty$ (LOVÁSZ, 1996; BRIN and PAGE, 1998; WU, *et al.*, 2012). Most existing approaches based on random walks on graphs only analyse the walks' steady-state distributions and use them to perform clustering, classification, or ranking on the node level (KONDOR and LAFFERTY, 2002; ZHU, *et al.*, 2003; DE KNIJF, *et al.*, 2011; WU, *et al.*, 2012). Instead of analyzing the limiting distribution of $\Pr(X_t)$, i.e., $\boldsymbol{\pi}_\infty$, it is also common to analyze the rows of P_t. $\Pr(X_t^{(i)})$, however, also converges to a stationary distribution. In fact, for random walks on the nodes without absorbing states as defined in Equation (2.8), all rows of P_∞, $(P_\infty)_{i,:}$, are equal. That means that no matter where the walk started, the steady-state distribution will be the same. Hence, when using such non-absorbing random walks for learning it is crucial to apply some kind of early termination in order to learn meaningful clusters or class labels (TISHBY and SLONIM, 2000). This idea, also termed *early stopping*, was

successfully introduced in power iteration clustering (LIN and COHEN, 2010*a*) and node label prediction on graphs (SZUMMER and JAAKKOLA, 2001). Power iteration is a general iterative algorithm to compute the eigenvector of a matrix. The power iteration using the transition matrix is $\mathbf{x}_{t+1} = T\mathbf{x}_t$. This is closely related to the random walk, Equation (2.7); however, the transition matrix is used directly instead of its transpose and $\mathbf{x}_t$ is not a probability. Thus, there is no probabilistic interpretation. As T is row-stochastic, $\mathbf{x}_t$ converges to a constant vector, however, monitoring the evolution of $\mathbf{x}_t$ reveals that its values converge first locally according to the structure of the underlying graph and then globally to one constant value (LIN and COHEN, 2010*a*).

So, the general insight here is that the same holds for random walks and therefore the intermediate distributions obtained by the random walks during the convergence process are extremely interesting. In this thesis, we will adopt this idea and use the evolution of label distributions encountered during absorbing and partially absorbing random walks up the a given length to represent graph structure. That is, both techniques – partial label-absorbing random walks and early stopping – are combined into a measure for local graph structure.

2.4 Kernels and Graphs

In this section, we introduce kernel-based learning as an important solution to the regularization problem stated in Equation (2.4). Further, we show how graph data can be integrated into classical machine learning methods by designing kernels among nodes or entire graphs. Kernels among nodes are referred to as *kernels on graphs* and are computed on networked data. *Graph kernels*, on the other hand, are kernels for structured data that are defined among entire graphs.[5]

2.4.1 *Kernel-based Learning*

Inference for supervised learning is based on the assumption that close points in the input space $\mathcal{X}$ should have similar output values y. This means that training points near the test points influence the predictions highly and this influence should decrease with increasing distance. One option to capture this intuition is to model the covariances of the outputs by a function of the inputs:

$$\operatorname{cov}(y, y') = k(\mathbf{x}, \mathbf{x}') + \sigma_n^2 \delta_{x,x'},$$

where $k(\mathbf{x}, \mathbf{x}')$ is a *kernel*, that is, a function measuring the similarity of any two vectors $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. However, not every function $f(\mathbf{x}, \mathbf{x}') \to \mathbb{R}$ leads to a valid covariance function. Considering its purpose, a covariance function, and in turn a kernel, has to be symmetric and *positive semidefinite*. From the view of probability theory, this can be understood when viewing the covariance function

[5] Unfortunately, the terms are not strictly distinguished in the literature.

as a function among two random variables Y and Y' and the kernel as a function among random variables X and X'. Hence, the covariance function is an extension of the covariance matrix of a random vector, which is positive semidefinite. The covariance matrix is itself a generalization of the scalar-valued variance of a random variable, which is always positive. Definitions can be found in any textbook about statistics, e.g., Chapter 3 in (KRENGEL, 2005; WASSERMAN, 2010).

Formally, a matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite iff $\forall \mathbf{x} \in \mathbb{R}^n$ it holds that $\mathbf{x}^\top M \mathbf{x} \geq 0$, a kernel is defined as follows.

DEFINITION 2.14 (POSITIVE-SEMIDEFINITE KERNEL) *$k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if*

- *k is symmetric, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$, and*
- *is positive semidefinite, i.e., $\forall$ finite subsets $\{\mathbf{x_1}, \ldots, \mathbf{x_i}, \mathbf{x_j}, \ldots, \mathbf{x_n}\} \subseteq \mathcal{X}$ the matrix K with $K_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$ is positive semidefinite.*

A positive-semidefinite kernel is also termed *Mercer kernel* and K is also called the *kernel matrix* or *Gram matrix*.

Taking a different view, sometimes called the dual representation, the kernel arises naturally from minimizing the regularized loss function defined previously in Equation (2.4):

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda J(f), \tag{2.13}$$

where we recall that L is a loss function, J is a penalty term, and $\mathcal{F}$ is a space of functions. Equipped with a kernel, there is a natural space of associated functions $\mathcal{F}$ for performing this optimization over. To see this we need the following definition.

DEFINITION 2.15 (REPRODUCING KERNEL HILBERT SPACE (RKHS)) *A Hilbert space $\mathcal{H}$ is a space of functions f defined on $\mathcal{X}$ with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. An RKHS $\mathcal{H}_k$ is a Hilbert space with reproducing property, that is, there is a function $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $f(\mathbf{x}) = k(\mathbf{x}', \mathbf{x}) \in \mathcal{H}_k$ and $\langle f(\cdot), k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}_k} = f(\mathbf{x})$.*

Note that $k(\mathbf{x}, \mathbf{x}') = \langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle_{\mathcal{H}_k}$ and hence it follows from the definition of an inner product that k is symmetric and positive definite. Further, the Moore–Aronszajn theorem, stated and proven in (ARONSZAJN, 1950), guarantees the uniqueness of the kernel.

THEOREM 2.1 (MOORE–ARONSZAJN THEOREM) *Every positive-definite function k on $\mathcal{X} \times \mathcal{X}$ is uniquely determined by an RKHS and vice versa.*

It turns out that if we restrict the space of functions $\mathcal{F}$ to be a reproducing kernel Hilbert space, that is $\mathcal{F} = \mathcal{H}_k$ as defined in Definition 2.15, then the solution to Equation (2.13) can be stated in terms of a kernel. This result is stated in the representer theorem.

THEOREM 2.2 (REPRESENTER THEOREM) *The minimizer of a regularized empirical loss function defined over a reproducing kernel Hilbert space (RKHS) can be represented as a finite linear combination of kernel products evaluated on the training data.*

A generalized version is stated and proven in (SCHÖLKOPF, *et al.*, 2001*a*). So, restricting $\mathcal{F}$ to be an RKHS, Equation (2.13) turns into

$$\min_{f \in \mathcal{H}_k} \sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i)) + \lambda \|f\|^2_{\mathcal{H}_k}, \tag{2.14}$$

where the penalty term for the space $\mathcal{H}_k$ is defined to be the squared norm $J(f) = \|f\|^2_{\mathcal{H}_k}$, where $\|f\|_{\mathcal{H}_k} = \sqrt{\langle f, f \rangle_{\mathcal{H}_k}}$.

It can be shown that the solution to Equation (2.14) is a finite linear combination of kernel values (WAHBA, 1990). More precisely it is the weighted sum of the kernel evaluated at the finite set of training inputs $\mathbf{x}_i$,

$$f^*(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i).$$

Recall that we denote the training data by $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$. As the penalty term can also be stated in terms of the kernel

$$J(f) = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j),$$

Equation (2.14) reduces to a finite-dimensional optimization problem

$$\min_{\boldsymbol{\alpha}} L(\mathbf{y}, K\boldsymbol{\alpha}) + \lambda \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}, \tag{2.15}$$

where K is the positive-semidefinite $n \times n$ kernel matrix on the training data with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Therefore, given that L is convex, the regularization problem is convex and can be solved by simple numerical optimization algorithms. The kernel solution to the regularized least-squares model was called the *regularization network* by POGGIO and GIROSI (1990); it is an example of a so-called kernel method. The use of different loss functions results in different kernel methods. Replacing the squared loss by the hinge loss $L(y, f(\mathbf{x})) = \max\{0, 1 - y f(\mathbf{x})\}$ leads to *support vector machines*, first introduced in (BOSER, *et al.*, 1992). The support vector machine classifier is a sparse maximum margin classifier that essentially selects the training points most informative for the distinction of classes. The decision boundary and support vectors can be learned by solving a convex optimization problem resulting from Equation (2.15). A comprehensive introduction to support vector machines can be found in (CRISTIANINI and SHAWE-TAYLOR, 2000).

Other kernel methods are the *Nadaraya–Watson model* (NADARAYA, 1964; WATSON, 1964) also known as kernel regression, kernel density classification, or short kernel classification, *Gaussian processes* (RASMUSSEN and WILLIAMS, 2005) implementing the Bayesian learning paradigm introduced in Section 2.2.1, and the lesser used *relevance vector machines* (TIPPING, 2001).

So far, we have learned that kernels are defined by the inner product of an RKHS. However, we can also view kernels as the inner product of a fixed non-linear feature space mapping $\phi(\mathbf{x})$:

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

In the case of finite-dimensional feature mappings the inner product is actually the dot product, which can be computed explicitly as

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}').$$

This view allows us to extend any learning method using the scalar product of the input vectors to use kernels. Intuitively we substitute $\mathbf{x}^\top \mathbf{x}'$ by $\langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ and consider the linear model in the high-dimensional (possibly infinite-dimensional) feature space. As the feature transformation does not need to be computed explicitly, this kernel substitution is also known as the *kernel trick*. So, to construct kernels we either choose the feature mapping and then use it to define the kernel or we construct the kernel directly. When taking the latter approach, one has to ensure that the constructed function is a valid kernel, in other words that it corresponds to an inner product in some possibly infinite-dimensional feature space. For simple kernels and low-dimensional input spaces, this verification can be done explicitly, as illustrated in the following example.

EXAMPLE 2.4 (SIMPLE KERNEL AND CORRESPONDING SCALAR PRODUCT)

Let $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$. $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$ *is a valid kernel. We can see this easily be expanding out the terms so that*

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^\top \mathbf{x}')^2 = (x_1 x_1' + x_2 x_2')^2 \\ &= x_1^2 x_1'^2 + 2 x_1 x'_1 x_2 x'_2 + x_2^2 x_2'^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^\top (x_1'^2, \sqrt{2} x_1' x'_2, x_2'^2) \\ &= \phi(\mathbf{x})^\top \phi(\mathbf{x}'), \end{aligned}$$

where $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^\top$.

In general, however, such an explicit construction is not possible. Another way to prove that a kernel is valid is via Definition 2.14. That is, we have to show that the

Gram matrix K is positive semidefinite. This can for example be done by showing that the determinants of all upper-left $k \times k$ sub-matrices of K are non-negative or that the eigenvalues of K are non-negative.

Examples of simple kernels are the *linear kernel*:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}',$$

and the *Dirac kernel*:

$$k(\mathbf{x}, \mathbf{x}') = \delta(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x} = \mathbf{x}' \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

A commonly used parametric kernel is the *Gaussian kernel*, also called the radial basis function kernel or squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right), \tag{2.17}$$

where ℓ is the length scale parameter. More expressive kernels can be constructed by combining simple kernels. Sums, products, and positive scalar powers of kernels are also valid kernels. Further, a kernel multiplied by a positive scalar or by a polynomial with non-negative coefficients is again a kernel.

Another powerful idea, when considering complex discrete objects, is to construct kernels from the objects' composite structures or parts. There are several lines of research establishing such composite kernels: kernels on (multi-) sets (SHI, *et al.*, 2009; GÄRTNER, *et al.*, 2002), marginalized kernels (TSUDA, *et al.*, 2002; KASHIMA, *et al.*, 2003), and convolution kernels (HAUSSLER, 1999; SHIN and KUBOYAMA, 2008). All these approaches allow us to define kernels between discrete structures such as strings, paths, trees, and graphs, which do not have to be of the same size. As we will study the construction of kernels for graphs later in the thesis, we will introduce the *convolution kernel* defined by HAUSSLER (1999) in more detail here.

THEOREM 2.3 (CONVOLUTION KERNEL) *Let $x \in \mathcal{X}$ be an input space with decompositions $\vec{x}$ consisting of parts $x_1, \ldots, x_P$, where $x_p \in \mathcal{X}_p$ for all $1 \leq p \leq P$ are countable sets.*[6] *Given a finite "part" relation $R \subseteq \mathcal{X}_1 \times \ldots \times \mathcal{X}_P \times \mathcal{X}$ with $R(\vec{x}, x) = 1$ iff $x_1, \ldots, x_P$ are the parts of x and positive-semidefinite kernels $k_p \colon \mathcal{X}_p \times \mathcal{X}_p$ on the parts, then $K \colon \mathcal{X} \times \mathcal{X}$ given by*

$$K(x, x') = \sum_{\vec{x} \in R^{-1}(x)} \sum_{\vec{x}' \in R^{-1}(x')} \prod_{p=1}^{P} k_p(x_p, x'_p) = \sum_{\vec{x} \in R^{-1}(x)} \sum_{\vec{x}' \in R^{-1}(x')} \kappa(\vec{x}, \vec{x}') \tag{2.18}$$

is positive semidefinite.

[6] In a more general version $\mathcal{X}_p$ only need to be separable metric spaces. All countable metric spaces are separable metric spaces.

Note that the relation R is *finite* if $R^{-1}(x) = \{\vec{x} \mid R(\vec{x}, x) = 1\}$ is finite for all $x \in \mathcal{X}$. See (HAUSSLER, 1999) for the proof of Theorem 2.3. The convolution kernel was further generalized to the so-called *mapping kernel* (SHIN and KUBOYAMA, 2008), which is defined for arbitrary transitive mapping systems $\{M_{x,x'} \subseteq \hat{\mathcal{X}}_x \times \hat{\mathcal{X}}_{x'} \mid (x, x') \in \mathcal{X} \times \mathcal{X}\}$, where $\hat{\mathcal{X}} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_p$. The mapping system for the convolution kernel $\{M_{x,x'} = \hat{\mathcal{X}}_x \times \hat{\mathcal{X}}_{x'}\}$ is indeed transitive. SHIN and KUBOYAMA (2008) also introduced distribution kernels which evaluate distributional features rather than maximum or minimum values of similarity or dissimilarity measures. The basis for these results were the convolution kernel and the *probability kernel* introduced in (HAUSSLER, 1999).

DEFINITION 2.16 (PROBABILITY KERNEL) *Let K be a valid kernel. If $K(x, x')$ is positive, $K(x, x') \geq 0 \; \forall x, x'$, and $\sum_{x,x'} K(x, x') = 1$, then K is a probability distribution on $\mathcal{X} \times \mathcal{X}$ called a probability kernel.*

Now we can construct kernels for probabilistic models with hidden variables that model the parts of the visible variables corresponding to structured data. Models generating data from hidden variables are called generative models. Probabilistic generative models, which can naturally deal with missing data and objects of variable size, can be applied in a discriminative setting once the kernel is constructed ((HAUSSLER, 1999; WATKINS, 1999) and Chapter 6 in (BISHOP, 2006)). Another approach to define kernels using generative models are *Fisher kernels* (JAAKKOLA and HAUSSLER, 1998).

As this thesis focuses on learning with graphs, we will introduce kernels for graph data in the following two subsections. This will be the basis for the derivation of kernels from RW-based node label probabilities, which is one of the main contributions of this thesis.

2.4.2 *Kernels on Graphs – Kernels between Nodes*

In this section, we introduce kernels among the nodes of a graph, which are used in kernel-based approaches to learning on the node level. We will see that kernels on graphs are closely related to random walks on graphs as introduced in Section 2.3. The derivation largely follows results presented in (SMOLA and KONDOR, 2003). Note that in this section, unlike in the previous sections, n denotes the number of nodes in a graph and thus covers all data points and not just the training examples.

DEFINITION 2.17 (KERNEL ON A GRAPH) *Given a graph $G = (V, E, \ell)$ a kernel on a graph $k(v_i, v_j)$ is defined between the vertices $v_i, v_j \in V$ of a graph and its kernel matrix is given by $K \in \mathbb{R}^{n \times n}$, where $K_{ij} = k(v_i, v_j)$.*

Once determined, kernels on graphs are used in supervised learning to regularize the model f to be smooth on the given input graph. Thus the kernel should be

deduced from the graph structure. Structural properties of a graph are nicely encoded in the *graph Laplacian*, which is therefore often used to design kernels on graphs.

DEFINITION 2.18 (GRAPH LAPLACIAN) *Given a weighted graph $G = (V, E)$ with weight matrix W, the Laplacian of G is defined as $L = D - W$, where D is the $n \times n$ diagonal matrix with entries $d_{ii} = \sum_{j=1}^{n} w_{ij}$. The normalized Laplacian is defined by $\tilde{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$.*

In the following we use spectral graph theory to analyze the spectral decomposition of the Laplacian and in turn to construct different kernels on graphs implementing smoothness constraints via the regularization framework. The spectral decomposition of the Laplacian is given by

$$L = \sum_{i=1}^{n} \lambda_i \mathbf{x}_i \mathbf{x}_i^\top,$$

where $\lambda_1 \leq \ldots \leq \lambda_n$ denote the eigenvalues of L and $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are the corresponding eigenvectors. An analysis of the eigensystem $(\mathbf{x}_i, \lambda_i)$ of L yields that the eigenvectors with small eigenvalues are smooth and therefore indicate large clusters within the data, and those with large eigenvalues are rugged and hence can be interpreted as noise. For an extensive investigation of the geometric and algebraic aspects of graph eigenvalues and spectral graph theory in general see (CHUNG, 1997). For any function $f: V \to \mathbb{R}$ on the finite graph G, it holds that

$$f^\top L f = \frac{1}{2} \sum_{i,j} \left(f_i - f_j \right)^2 \geq 0, \tag{2.19}$$

which can easily be seen by plugging in the definition of the Laplacian. Equation (2.19) is a measure of smoothness of f. Smaller values indicate smoother functions f; that is, f varies slowly over the graph. Thus to achieve a desired level of smoothness, we can vary the properties of the Laplacian, which means a different weighting of the eigenvectors of L by applying a (parametric) spectral transformation

$$\tilde{r}(L) = \sum_{i=1}^{n} r(\lambda_i) \mathbf{x}_i \mathbf{x}_i^\top,$$

where $r: \mathbb{R}_+ \to \mathbb{R}_+$ is a non-negative monotonically increasing function of λ_i. This regularization function can now be used to define different kernels on the given graph.

To get a kernel matrix K that respects the smoothness properties of the data encoded in the graph, we can create an RKHS according to Definition 2.15 with an

inner product based on the regularization of the graph Laplacian. Thus we define the Hilbert space $\mathcal{H}$ on $\mathbb{R}^n$ equipped with the inner product

$$\langle f, g \rangle_{\mathcal{H}} = \langle f, \tilde{r}(L) g \rangle = f^\top \tilde{r}(L) g.$$

$\mathcal{H}$ is a proper Hilbert space, since the graph Laplacian is a positive-semidefinite matrix, which can be derived from Equation (2.19). Now it suffices to show that $\mathcal{H}$ satisfies Definition 2.15 with the kernel matrix defined as

$$K = \tilde{r}(L)^{-1} = \sum_{i=1}^{n} r^{-1}(\lambda_i) \mathbf{x}_i \mathbf{x}_i^\top.$$

The reproducing property is met iff $f(i) = f^\top \tilde{r}(L) K_{i,:} \; \forall i$. This again is the case iff $f^\top = f^\top \tilde{r}(L) K$, which holds iff $K = \tilde{r}(L)^{-1}$. Theorem 2.1 guarantees the uniqueness of the kernel k, where $k(u, v) = K_{ij}$. Some particular regularization functions, for instance

$$\begin{aligned} r(\lambda) &= 1 + \sigma^2 \lambda \\ r(\lambda) &= \exp(\sigma^2 / 2\lambda) \\ r(\lambda) &= (\alpha - \lambda)^{-p} \text{ with } \alpha \geq \max_i(\lambda_i), \end{aligned}$$

lead to commonly used kernels like the regularized Laplacian kernel, the diffusion kernel, or the p-step random walk kernel. Note that all derivations also hold for the normalized Laplacian $\tilde{L}$.

In the following, we summarize the definitions of kernel matrices for commonly used kernels on graphs, which are also used as baselines in this thesis and point out their random walk interpretations. Most of the kernels include a parameter which we will denote by α. The *diffusion kernel* originally introduced by KONDOR and LAFFERTY (2002) is defined as

$$K_{\text{DIFF}} = \exp\left(\alpha \tilde{L}\right), \tag{2.20}$$

where $\exp(\cdot)$ is the matrix exponential. The diffusion kernel is closely related to random walks as introduced in Section 2.3 of this background chapter. In fact, it is the limiting distribution of $\Pr(X_{\hat{t}} = v_j \mid X_0 = v_i)$ of a lazy random walk, where $\hat{t} = 1/\Delta t$ with $\Delta t \to 0$. A *lazy random walk* is a Markov random walk, cf. Definition 2.12, with constant transition probabilities to all its neighbors, $\Pr(X_{t+1} = v_j \mid X_t = v_i) = \alpha$ if $i \neq j$, and with probability $1 - \alpha \deg(v_i)$ it will stay in place, where $\alpha \leq 1/\max_i \deg(v_i)$. The ijth entry of K_{DIFF} can be regarded as the sum of the probabilities that a lazy walk takes each path from v_i to v_j.

The *regularized Laplacian kernel* is defined as

$$\begin{aligned} K_{\text{REGLAP}} &= \left(I - \alpha\, \tilde{L}\right)^{-1} \\ &= \left(I - \alpha\, D^{-1/2} L D^{-1/2}\right)^{-1} \\ &= \left(I - \alpha\, (I - D^{-1/2} W D^{-1/2})\right)^{-1} \\ &= \left((1+\alpha)\, I - \alpha\, D^{-1/2} W D^{-1/2}\right)^{-1}, \end{aligned} \tag{2.21}$$

where $\tilde{L}$ is the normalized Laplacian. The *Moore–Penrose pseudoinverse of the Laplacian*

$$K_{\text{INVLAP}} = L^{+} \tag{2.22}$$

is a valid kernel closely related to average commute times in graphs (FOUSS, *et al.*, 2012). The average commute time $c(v_i, v_j)$ between two nodes v_i and v_j is defined as the average number of steps a random walk starting in v_i will take until it reaches v_j for the first time and then gets back to v_i. It holds that $c(v_i, v_j) = \text{vol}(G)((L^{+})_{ii} + (L^{+})_{jj} - 2(L^{+})_{ij})$, where $\text{vol}(G)$ is the volume of graph G given by $\text{vol}(G) = \sum_i D_{ii}$. K_{INVLAP} is therefore also called the commute-time kernel. The pseudoinverse of the Laplacian is also related to the regularized Laplacian kernel defined on the unnormalized Laplacian. Both kernels have the same set of eigenvectors and their non-zero eigenvalues are reciprocal.

Another kernel closely related to the regularized Laplacian is the *von Neumann diffusion kernel* (ZHOU, *et al.*, 2003; FOUSS, *et al.*, 2012), given by

$$K_{\text{VND}} = \left(I - \alpha\, D^{-1/2} W D^{-1/2}\right)^{-1}. \tag{2.23}$$

The entries of the p-step random walk kernel

$$K_{\text{RW}} = (\alpha\, I - \tilde{L})^{p} = ((\alpha - 1) I + D^{-1/2} W D^{-1/2})^{p} \tag{2.24}$$

are proportional to the probability of a p-step random walk with restarts, cf. Definition 2.13, where the walk restarts at the starting node. Note that the following relation among the parameters holds: $\beta = 1/\alpha$. In general for all kernels involving the Laplacian, we can technically either use its normalized or unnormalized version. Some of the interpretations in terms of probabilities will however only work for the above choices.

Closely related to kernels on graphs are *data-dependent kernels*, which also use all labeled and unlabeled data points (not necessarily being nodes in a graph) during their computation. Data-dependent kernels are widely used in semi-supervised learning (ZHOU, *et al.*, 2003; SINDHWANI, *et al.*, 2005; LEVER, *et al.*, 2012), where one strategy is to construct a graph modeling the data geometry and then compute a kernel on this graph. Other approaches like semi-supervised support vector

machines (see (CHAPELLE, *et al.*, 2008) for an extensive comparison and review) try to enforce smoothness of predictions along a manifold defined by the data in feature space, typically by modifying the optimization objective. In Part II of this thesis, we will investigate kernel construction leveraging label-structure information (cf. Definition 2.10) in plain graph data where no features on the nodes are given. This means we will construct a kernel which – in contrast to the kernels used in standard semi-supervised learning – is a *label-dependent kernel* rather than a data-dependent kernel.

As an extension of the parametric spectral transformation developed in (SMOLA and KONDOR, 2003) and sketched above, there is the possibility to adapt the transformation automatically from the data by performing a search over a fixed set of non-parametric transformations using a convex maximization of the kernel alignment to the labeled data. This concept leads to label-dependent kernels also referred to as kernel–target alignment (CRISTIANINI, *et al.*, 2001; ZHU, *et al.*, 2004; MIN, *et al.*, 2007) and Chapter 15 of (CHAPELLE, *et al.*, 2006). These methods essentially use label information to improve previously computed kernels.

2.4.3 Graph Kernels – Kernels between Graphs

To apply kernel methods to learning on the graph level, a positive-semidefinite kernel between graphs has to be constructed. This overview on existing graph kernels mainly follows (VISHWANATHAN, *et al.*, 2010; KRIEGE, *et al.*, 2014). In contrast to the previous section, where the input domain were the nodes of a graph, the domain of interest is now a set of graphs. In this section, n denotes the number of graphs, $n_i = |V^{(i)}|$ denotes the number of nodes of graph $G^{(i)}$, and $N = \sum_i n_i$ is the total number of nodes of all considered graphs.

DEFINITION 2.19 (GRAPH KERNEL) *Given a graph database of n graphs $\mathbf{G} = \{G^{(i)} = (V^{(i)}, E^{(i)}, \ell)\}_{i=1,\dots,n}$ a graph kernel $K(G^{(i)}, G^{(j)})$ is defined between graphs $G^{(i)}$, $G^{(j)} \in \mathbf{G}$, and its kernel matrix is given by a positive-semidefinite matrix $K \in \mathbb{R}^{n \times n}$, where $K_{ij} = K(G^{(i)}, G^{(j)})$.*

A common strategy to design graph kernels is to decompose the graph into smaller substructures and then define kernels on these possibly overlapping parts. The graph kernel is then again a combination, for example a sum, of the part kernels

$$K(G^{(i)}, G^{(j)}) = \sum_{P^{(i)} \in G^{(i)}} \sum_{P^{(j)} \in G^{(j)}} \delta(P^{(i)}, P^{(j)})\, \kappa(P^{(i)}, P^{(j)}), \tag{2.25}$$

where $P^{(i)}$ are the parts of $G^{(i)}$ and $\delta(P^{(i)}, P^{(j)})$ is 1 if the parts match structurally, and 0 otherwise. As graphs are discrete objects consisting of finitely many nodes and edges, graph kernels can be viewed as convolution kernels, cf. Theorem 2.3. Given a relation R with $R^{-1}(G) = \{\vec{x} : R(\vec{x}, G) = 1\}$, the convolution kernel

compares all substructures defined by the decomposition $\vec{x}$ of a graph in a substructure (or part) P and its complement $\bar{P}$. The part kernel can then be defined as $\kappa(\vec{x}_i, \vec{x}_j) = k_P(P^{(i)}, P^{(j)})\, k_{\bar{P}}(\bar{P}^{(i)}, \bar{P}^{(j)})$, where $k_{\bar{P}}(\bar{P}^{(i)}, \bar{P}^{(j)}) := 1$, cf. (BORGWARDT and KRIEGEL, 2005). To ensure positive semidefiniteness, the kernel $k_P(P^{(i)}, P^{(j)})$ needs to be positive semidefinite.

Other approaches to guarantee positive semidefiniteness of graph kernels are to implement a weight function $w(P^{(i)}, P^{(j)})$ into the part kernel $\kappa(P^{(i)}, P^{(j)})$ ensuring convergence (GÄRTNER, *et al.*, 2003), or to view $\kappa(P^{(i)}, P^{(j)})$ itself as a convolution kernel according to the assumptions of Theorem 2.3 (KRIEGE and MUTZEL, 2012). In this case, the part kernel could boil down to comparing nodes and edges for matching substructures, e.g.,

$$\kappa(P^{(i)}, P^{(j)}) = \sum_{e_{u,u'} \in P^{(i)}} \sum_{e_{v,v'} \in P^{(j)}} k_E(e_{u,u'}, e_{v,v'})\; k(u, v)\; k(u', v'). \tag{2.26}$$

where $k_E(\cdot, \cdot)$ is an edge kernel comparing edge types (in the simplest case Dirac kernel on the edge labels), and $k(\cdot, \cdot)$ is a node kernel comparing node labels and/or attributes. Equation (2.26) is a convolution kernel, as the input graphs and in turn their parts have a finite number of nodes and edges and the part relation (being the decomposition of graph substructures into nodes and edges) is finite and the parts themselves are countable sets. A closely related graph kernel measuring the intersection of parts quantitatively is the *intersection kernel* (GÄRTNER, 2005).

The various graph kernels proposed in the literature mainly differ in the way the parts are constructed and in the similarity measure used to compare them. The most natural way of decomposing a graph is to consider its subgraphs. However, we cannot simply compare all subgraphs of two graphs $G^{(i)}$ and $G^{(j)}$ to get their kernel value as this was shown to be at least as hard as solving the graph isomorphism problem (GÄRTNER, *et al.*, 2003). So, we have to limit the subgraphs, or more general the substructures, to be compared by limiting the kind and/or size of substructures considered. The substructures, however, need to be big enough to actually represent the graph structure. Therefore, comparing two graphs by only comparing all pairs of nodes,

$$K(G^{(i)}, G^{(j)}) = \sum_{u \in G^{(i)}} \sum_{v \in G^{(j)}} k(u, v), \tag{2.27}$$

is too restrictive. Commonly used substructures are *paths*, cf. Definition 2.6, as for instance shortest-paths or paths generated by random walks, and limited size subgraphs of a particular structure, such as *trees*, *cycles*, or induced subgraphs so-called *graphlets*, cf. Definitions 2.6 and 2.8. Further, existing graph kernels also differ in which information they use to compare the parts, such as labels and attributes on the nodes and edges.

In general there are two strategies to actually compute the kernel values between graphs. One way is to compute a feature map $\phi(G^{(i)})$ for each graph directly and then the kernel value is computed from the inner product between the feature maps $K(G^{(i)}, G^{(j)}) = \langle \phi(G^{(i)}), \phi(G^{(j)}) \rangle$. The explicitly computed features are usually *count features* indicating how often certain substructures occur. This strategy is also called *explicit* kernel computation as the feature map is explicitly created. One drawback of this approach is that we cannot use an arbitrary kernel to compare node and edge information. A sufficient condition for graph kernels to be convolution kernels is that the part kernel in Equation (2.18), $\kappa(\vec{x}, \vec{x}')$, be a product of Dirac kernels, cf. Equation (2.16), or more general be either 0 or 1 (KRIEGE, *et al.*, 2014). The other strategy is to compute the kernel between two graphs implicitly. This is usually done by considering their product graph. Doing so allows one to employ general kernels among node attributes and edges as it is not necessary to construct a feature map explicitly. However, this approach comes with the cost of creating and analyzing the product graph $G_\times = G^{(i)} \times G^{(j)}$ of every pair of graphs in the database.

DEFINITION 2.20 (PRODUCT GRAPH) *Given two graphs $G^{(i)} = (V^{(i)}, E^{(i)})$ and $G^{(j)} = (V^{(j)}, E^{(j)})$, their product graph is denoted by $G_\times = (V_\times, E_\times)$ and has a vertex set*

$$V_\times = \{(u, v) \mid u \in V^{(i)}, v \in V^{(j)}\}$$

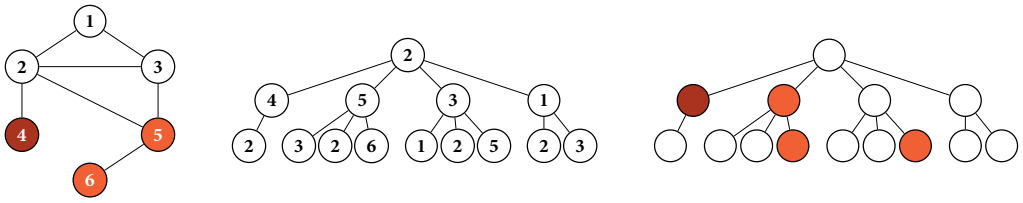
and an edge set

$$E_\times = \{((u, v), (w, z)) \mid (u, w) \in E^{(i)}, (v, z) \in E^{(j)}\}.$$

The product graph has a node for each pair of nodes in the original graphs, and an edge in $G_\times$ corresponds to one edge in each of the original graphs. Two nodes in the product graph are neighbors only if the corresponding nodes in the original graphs were both neighbors as well. Product graphs have some nice properties making them suitable for the computation of graph kernels. First, the adjacency matrix of a product graph is the Kronecker product of the adjacency matrices of the original graphs, $A_\times = A^{(i)} \otimes A^{(j)}$; the same holds for the weight matrix $W_\times$ and the transition matrix $T_\times$. $W_\times$ can, however, in general be used to capture any kind of edge similarity. Further, performing random walks as defined in the previous section on the product graph is equivalent to performing simultaneous random walks on the original graphs (IMRICH and KLAVŽAR, 2000). That is, $\Pr(X_t = (u, v) \mid X_0 = (w, z)) = (T_\times^t)_{ab}$ with $a = (u-1)n_j + v$ and $b = (w-1)n_j + z$ corresponds to the probability that random walks of length t on $G^{(i)}$ and $G^{(j)}$, where the walk on $G^{(i)}$ starts in w and ends in u and the walk on $G^{(j)}$ starts in z and ends in v, are equal.

The *random walk kernel* by VISHWANATHAN, *et al.* (2010) is then defined as

$$K(G^{(i)}, G^{(j)}) = \sum_{t=0}^{\infty} \mu_t T_\times^t \boldsymbol{\pi}_\times, \tag{2.28}$$



(a) Graph (b) Node-based Subtree pattern (c) Label-based subtree pattern

Figure 2.3: **Subtree Patterns.** Panel (a) shows an example graph where node labels are encoded by color. Panel (b) illustrates a (node-based) subtree pattern of depth 2 rooted at node 2. Panel (c) shows the label-based subtree pattern of depth 2 rooted at node 2.

where μ_t is a weight guaranteeing the convergence of the infinite sum and $\boldsymbol{\pi}_\times = \boldsymbol{\pi}_i \otimes \boldsymbol{\pi}_j$ is the initial probability distribution among nodes in $G^{(i)}$ and $G^{(j)}$, cf. $\boldsymbol{\pi}_0$ in Equation (2.7). Several variations of the random walk kernel have been introduced in the literature. Instead of adding the probabilities of two walks being the same, GÄRTNER, *et al.* (2003) counted the number of matching walks. Other variants take node labels into account and compare the label sequences instead of node sequences (KASHIMA, *et al.*, 2003; MAHE, *et al.*, 2004; BORGWARDT, *et al.*, 2005; HARCHAOUI and BACH, 2007). Also computed via the product graph, the *common subgraph matching kernel* compares subgraphs of small sizes instead of random walks (KRIEGE and MUTZEL, 2012).

Avoiding the construction of potentially huge product graphs, explicit feature maps for graph kernels can be computed in a more memory-efficient manner and often also faster. The features are either some kind of similarity measure among substructures or counts or indicators of occurrences of substructures of particular sizes. The *graphlet kernel*, for example, counts induced subgraphs of small sizes:

$$K(G^{(i)}, G^{(j)}) = \mathbf{f}^{(i)\top}\mathbf{f}^{(j)}, \tag{2.29}$$

where $\mathbf{f}^{(i)}$ are the count features (SHERVASHIDZE, *et al.*, 2009). The *cyclic pattern kernel* measures the occurrence of cyclic and tree patterns and maps the graphs to pattern indicator features which are independent of the pattern frequency (HORVÁTH, *et al.*, 2004). The *Weisfeiler–Lehman subtree kernel* counts label-based subtree patterns (SHERVASHIDZE, *et al.*, 2011)

$$K_{h_{\text{MAX}}}(G^{(i)}, G^{(j)}) = \sum_{h=0}^{h_{\text{MAX}}} K(G_h^{(i)}, G_h^{(j)}), \tag{2.30}$$

where $K(G_h^{(i)}, G_h^{(j)}) = \langle \mathbf{f}_h^{(i)}, \mathbf{f}_h^{(j)} \rangle$ and $\mathbf{f}_h^{(i)}$ is a count feature vector counting subtree patterns of depth h. Figure 2.3 illustrates node- and label-based subtree patterns.

A subtree pattern is a tree rooted at a particular node where each level contains the neighbors of its parent node. Therefore, the same nodes can appear repeatedly. Other graph kernels based on subtree patterns have been proposed in the literature (RAMON and GÄRTNER, 2003; BACH, 2008).

Although they have the advantage of allowing for feature analysis and sparse feature representations, explicit feature maps can only be created efficiently when counting discrete features such as labeled substructures or discretized node attributes. If we want to compare continuous node features pairwise by applying for instance a continuous valued node kernel, then an explicit feature computation is difficult or even impossible (KRIEGE, *et al.*, 2014).

The *shortest-path kernel* is a flexible kernel comparing all shortest paths in two graphs according to their lengths and the information on the respective nodes and edges

$$K(G^{(i)}, G^{(j)}) = \sum_{u,v \in G^{(i)}} \sum_{w,z \in G^{(j)}} k(u,w)\ \delta(d_{G^{(i)}}(u,v), (d_{G^{(j)}}(w,z))\ k(v,z), \tag{2.31}$$

where $k(\cdot,\cdot)$ is a node kernel comparing labels and attributes of the respective starting and end nodes of the paths and $\delta(\cdot,\cdot)$ is the Dirac kernel (Equation (2.16)) among the lengths of the paths (BORGWARDT and KRIEGEL, 2005). Edge weights and labels, if present, can also be captured by replacing $\delta(\cdot,\cdot)$ by a general kernel between edges. The shortest-path kernel can be computed implicitly or explicitly depending on the applied node and edge kernels. Another recently introduced graph kernel based on paths is the *GraphHopper kernel*, which compares the nodes encountered while hopping along shortest paths (FERAGEN, *et al.*, 2013).

The large number of recently introduced graph kernels, which differ vastly in their notation and computation, indicates that the comparison of graphs in order to perform machine learning on structured data is both considerably difficult and extremely important. Surprisingly, none of the above introduced kernels is flexible enough to consider arbitrary node and edge information while still being fast and memory efficient. In the first part of this thesis, we will define a novel flexible and fast graph kernel framework overcoming these limitations of existing kernels.

2.5 The Relational Perspective

Learning with graphs deals with reasoning in domains of interrelated and/or complex objects. So far, we have seen that graphs are an elegant way to represent networked and structured data being backed up by strong theoretical results from (spectral) graph theory and efficient computational techniques employing matrix manipulations. Closely related to machine learning with graphs is logical and relational learning. To employ relational learning we have to change our perspective on the data representation from sets of vertices and edges to logical

combinations of predicates applied to constants and variables. Being grounded in logic, relational data representations are the basis of artificial intelligence (AI) (RUSSELL and NORVIG, 2003). Whereas in the past research in AI focused on logic programming and satisfiability, machine learning has built its strength in statistical learning in the presence of noisy observations. Both fields derived powerful representations meeting the requirements of their problems. However, as often in science, both communities can learn a lot by extending their perspective to the other closely related field. One example of a fruitful combination is the emerging research area of statistical relational learning (SRL) joining logical reasoning and uncertainty (GETOOR and TASKAR, 2007). In this thesis, we will also exploit the benefits of combining both worlds by retrieving structural patterns in relational data representations to enhance graph-based learning. Therefore, we need a basic understanding of the representation of relational data. This background section introduces relational data; for an introduction to relational learning, we refer to (FLACH, 1994), (GETOOR and TASKAR, 2007), and (DE RAEDT, 2008). Notation and illustrations in this section largely follow (FLACH, 1994) and (DE RAEDT, 2008).

Like in statistical learning, the syntax of relational logic comprises *variables* and particular data instances. The data instances, being thought of concrete individuals, are called *constants*. Variables are donated by uppercase letters (`X`) and constants by lowercase words (`observation1`). Both, variables and constants are *terms*. Properties of individuals and relations among terms are encoded by *predicates* and the number of terms a predicate is defined on is called *arity*. The building blocks for knowledge representation are *atoms*, which are predicates together with particular arguments (constants or variables). In the context of the social network illustrated in Example 1.2 and Figure 1.2, constants are for example `phil`, `doro`, `dirk`, and `unibonn` and example predicates are `friends` and `works_at`. Now, atoms are

$$\texttt{friends}(\texttt{phil}, \texttt{doro}), \tag{2.32}$$

$$\texttt{works\_at}(\texttt{dirk}, \texttt{unibonn}), \tag{2.33}$$

or

$$\texttt{friends}(\texttt{phil}, \texttt{X}). \tag{2.34}$$

The atoms in Equations (2.32) and (2.33) are *ground atoms*, as the terms involved consist of constants only. The predicate in Equation (2.34) contains a variable (`X`) and could be read as "all friends of Philipp". The power of variables comes from the fact that we can represent (and reason about) sets of objects with certain properties, e.g., all persons with the property of being friends with Philipp. In a logical program or relational dataset the set of all considered constants is called

the (Herbrand[7]) *universe*[8], the set of all possible ground atoms is the (Herbrand) *domain*, and the set of all *true* ground atoms is called (Herbrand) *interpretation*. Often, when specifying relational data, we will list the true ground atoms only assuming that everything not listed is false; this assumption is called the *closed-world assumption*. If a database of atoms does not involve variables, then we speak of *propositional data*. In Part II of this thesis we will tackle set completion in relational data, where the given data is propositional; however, the algorithm developed leverages *variabilization*. Variabilization is the technique of generalizing ground atoms by inserting variables. For example by substituting the constant `doro` in Equation (2.32) by a variable leading to Equation (2.34).

Reasoning in relational data is performed by logic programs assessing the satisfiability of interpretations or deducing concepts. One popular technique here is inductive logic programming (ILP) WROBEL (2001); DŽEROSKI and LAVRAČ (2001). Rules for reasoning are, for example, organized in *clauses*, which are if/then statements. As opposed to *clausal logic*, rules can also be represented in (first-order) *predicate logic* involving formulas with logical connectives (negation, implication, conjunction, disjunction, equivalence) and quantifiers (for all, there exists). Both formalisms are semantically equivalent. As we will not study approaches to (statistical) relational learning but leverage logical features in graph-based machine learning, we will not cover reasoning in relational data in depth here, but introduce a possible graph representation for relational data.

For our work we will represent relational data by a hypergraph defined as follows. In general, a hypergraph is a straightforward generalization of a graph, in which an edge can link any number of nodes, rather than just two. More formally, a hypergraph is a pair (V, E) where V is a set of nodes, and E is a multiset of labeled non-empty subsets of V, called hyperedges. Now, the constants in a given universe can be seen as nodes labeled by the predicates of the unary ground atom(s) including the respective constant and ground atoms of larger arity form hyperedges. Each hyperedge is labeled with the predicate symbol of the corresponding ground atom. Nodes (constants) are linked by a hyperedge if and only if they appear as arguments in the same predicate. If the relational database only consists of binary predicates, then the data can be represented by a "normal" graph, as defined in Definition 2.1. To visualize such a graph let us consider the following example. Figure 2.4 shows the corresponding (hyper)graph.

Example 2.5 (Houses in Pompeii (Relational Data))

Our domain of discourse is rooms and houses discovered when excavating

[7] Jacques Herbrand (1908–1931) was a French mathematician who studied and contributed to mathematical logic.

[8] A universe can be infinite dimensional.

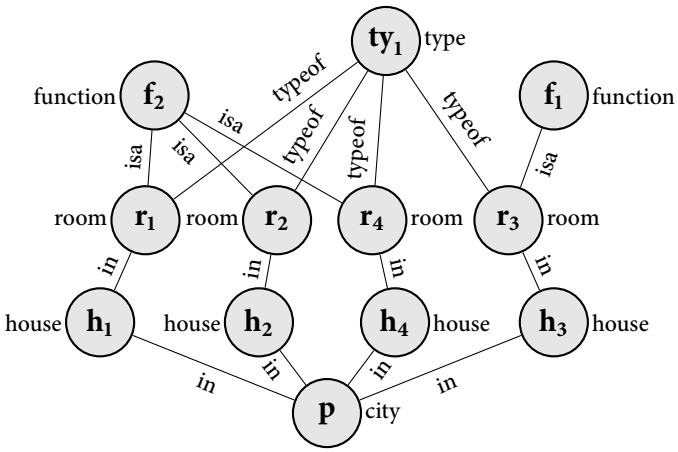


Figure 2.4: **(Hyper)graph of Houses in Pompeii.** (Hyper)graph representing a toy version of the Pompeii dataset introduced in Example 2.5. Nodes are labeled by the predicates of unary ground atoms where node identifiers are the constants. Edges are formed by binary ground atoms having the adjacent nodes as arguments and the respective edge label is the predicate.

Pompeii, a Roman city in southern Italy that was covered in volcanic ash during the eruption of Mt. Vesuvius in AD 79*. This toy example is motivated by a real-world database*[9] *analyzed in (*ALLISON*, 2001). The constants are houses such as* $\mathtt{h}_1$*,* $\mathtt{h}_2$*, etc., cities such as* pompeii*, rooms such as* $\mathtt{r}_1$*,* $\mathtt{r}_2$*, etc., functions* $\mathtt{f}_1$*, and* $\mathtt{f}_2$ *and types* $\mathtt{ty}_1$*. Predicates represent relations among houses and rooms (e.g.,* in*) or attributes of rooms, e.g.,* isa *or* typeog*. All true ground atoms representing our knowledge about the excavation are now:*

$$\{\mathtt{city}(\mathtt{p}), \mathtt{house}(\mathtt{h}_1), \mathtt{house}(\mathtt{h}_2), \mathtt{house}(\mathtt{h}_3), \mathtt{house}(\mathtt{h}_4),$$
$$\mathtt{room}(\mathtt{r}_1), \mathtt{room}(\mathtt{r}_2), \mathtt{room}(\mathtt{r}_3), \mathtt{room}(\mathtt{r}_4), \mathtt{function}(\mathtt{f}_1),$$
$$\mathtt{function}(\mathtt{f}_2), \mathtt{type}(\mathtt{ty}_1), \mathtt{in}(\mathtt{h}_1, \mathtt{p}), \mathtt{in}(\mathtt{h}_2, \mathtt{p}), \mathtt{in}(\mathtt{h}_3, \mathtt{p}),$$
$$\mathtt{in}(\mathtt{h}_4, \mathtt{p}), \mathtt{in}(\mathtt{r}_1, \mathtt{h}_1), \mathtt{in}(\mathtt{r}_2, \mathtt{h}_2), \mathtt{in}(\mathtt{r}_3, \mathtt{h}_3), \mathtt{in}(\mathtt{r}_4, \mathtt{h}_4),$$
$$\mathtt{isa}(\mathtt{r}_1, \mathtt{t}), \mathtt{isa}(\mathtt{r}_2, \mathtt{t}), \mathtt{isa}(\mathtt{r}_4, \mathtt{t}), \mathtt{isa}(\mathtt{r}_3, \mathtt{f}_1), \mathtt{typeof}(\mathtt{r}_1, \mathtt{ty}_1),$$
$$\mathtt{typeof}(\mathtt{r}_2, \mathtt{ty}_1), \mathtt{typeof}(\mathtt{r}_3, \mathtt{ty}_1), \mathtt{typeof}(\mathtt{r}_4, \mathtt{ty}_1)\}.$$

Note that in this work we will consider graphs with pairwise edges to be able to use the information propagation techniques introduced in Section 2.3. For

[9] `http://www.stoa.org/projects/ph/home`

hypergraphs with non-pairwise edges only recently approaches to learning on the node level started to be investigated (ZHOU, *et al.*, 2006).

Part I

Propagation Kernels

Learning from structured data is an active area of research. As domains of interest become increasingly diverse and complex, it is important to design flexible and powerful methods for analysis and learning. By structured data we refer to situations where objects of interest are structured and hence can naturally be represented using graphs. Examples 1.1 and 2.1 introduced graphs representing the bindings of different atoms of chemical compounds, and the drug design problem as a real-world example of immense practical relevance. Other real-world examples of structured data are scene graphs representing semantic information in images, text documents reflecting complex content dependencies, and manifold data modeling objects and scenes in robotics. The goal of learning with graphs is to exploit the valuable and rich information inherent in graphs representing structured data. The main challenge thereby is to efficiently exploit the graph structure – while allowing for missing, uncertain, and continuous information – for machine learning tasks such as classification or retrieval. A popular approach to learning from structured data is to design graph kernels measuring the similarity between graphs. For classification or regression problems, the graph kernel can then be plugged into a kernel machine, such as a support vector machine or a Gaussian process, for efficient learning and prediction.

In this part, we introduce propagation kernels, a general graph kernel framework for efficiently measuring the similarity of structured data. Propagation kernels are based on the idea of monitoring how information spreads through a set of given graphs. They leverage early-stage distributions from propagation schemes based on random walks, introduced in Section 2.3, to capture structural information encoded in node labels, attributes, and edges. Doing so has two benefits. First, off-the-shelf propagation schemes can be used to naturally realize kernels for graphs with various kinds of additional information attached to their nodes and edges such as labeled, partially labeled, unlabeled, directed, undirected and attributed graphs. Second, since propagation kernels are not tied to the substructures being compared to asses graph similarity, but instead to the propagation scheme, they can be considerably faster than state-of-the-art approaches without sacrificing performance.

We will also show that if the graphs at hand have a regular structure, for instance when modeling image or video data, one can exploit this regularity to scale the kernel computation to large databases of graphs with thousands of nodes. As an extension to the computation for general graphs, we will develop an algorithm for efficient propagation kernel computation for grid graphs whose nodes can be embedded in a square lattice.

We support our contributions by exhaustive experiments on a number of real-world graphs, including benchmark graph databases from bioinformatics and image data for semantic scene prediction, as well as texture classification.

CHAPTER 3

PROBLEM, PREVIOUS WORK, AND PROPOSED SOLUTION

In this chapter, we will formally define the problem of graph classification and investigate existing solutions thereof. We mainly focus on kernel-based approaches for which we review previous work and show some of their weaknesses and limitations. Based on these we will then introduce propagation kernels and discuss how they can overcome the limitations of existing approaches. Then, we briefly discuss related work on within-network relational learning and kernels on graphs, and kernels between probability distributions and sets.

3.1 Kernel-based Graph Classification

Graph classification is the problem of predicting the class label for a data point which is itself structured and therefore can be represented as graph. We will define the problem in terms of statistical machine learning on structured data.

Problem 3.1 (Graph Classification) *Given a graph database* $\mathbf{G} = \{G^{(i)}\}_{i=1,\dots,n}$ *with training data* $\mathcal{D} = \{(G^{(i)}, y_i) | i = 1, \dots, m\}$*, where* $y_i \in [k]$ *with* k *being the number of possible graph labels, find the labels* $y_j \in [k]$ *for all unlabeled graphs* $G^{(j)}$ *not in the training data.*

Binary graph classification is illustrated in Figure 3.1. To solve this task with classical supervised learning as introduced in Section 2.2.1 the graph representation of the structured data entities has to be converted to vector-valued input data. Once we have this feature representation we can leverage any learning algorithm from the large pool of statistical machine learning methods. However, the graphs representing the data points in structured domains are encoding complex dependencies enriched with versatile information on the nodes and edges. So, to capture this complexity in the data explicit feature maps are usually of high, possibly infinite, dimension. One class of approaches that are able to elegantly deal with high dimensional feature representations are kernel methods as introduced in Section 2.4. There are two ways of designing kernels between graphs. By computing a kernel

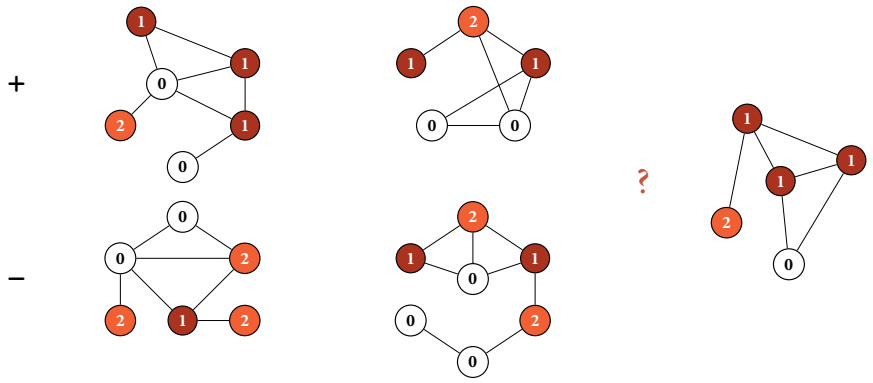


Figure 3.1: **Graph Classification.** Binary graph classification is the task of inferring the class labels of structured data instances with unobserved labels (graph on the right) from a training set of graphs with observed class labels (graphs on the left). The graph labels in this illustration are $\mathcal{L} = \{+, -\}$.

between explicitly computed graph features we get an *explicit graph kernel*. Graph kernels can also be computed *implicitly* not taking the detour via an explicit feature map. Advantages and disadvantages of both approaches were discussed in Section 2.4.3.

Several graph kernels have been proposed in the literature, but they often make strong assumptions as to the nature and availability of information related to the graphs at hand. The most simple of these proposals assume that graphs are unlabeled and have no structure beyond that encoded by their edges. However, graphs encountered in real-world applications often come with rich additional information attached to their nodes and edges. This naturally implies many challenges for representation and learning such as:

- missing information occurring from partially observable data,
- uncertain information arising from aggregating information from multiple sources, and
- continuous information derived from complex and possibly noisy sensor measurements.

Images, for instance, often have metadata and semantic annotations which are likely to be only partially available due to the high cost of collecting training data; Figure 3.2 illustrates a partially labeled graph representing an image scene. Point clouds captured by laser range sensors consist of continuous 3D coordinates and curvature information; in addition, part detectors can provide possibly noisy semantic annotations, cf. Figure 3.3. Entities in text documents can be backed by entire Wikipedia articles providing huge amounts of structured information,

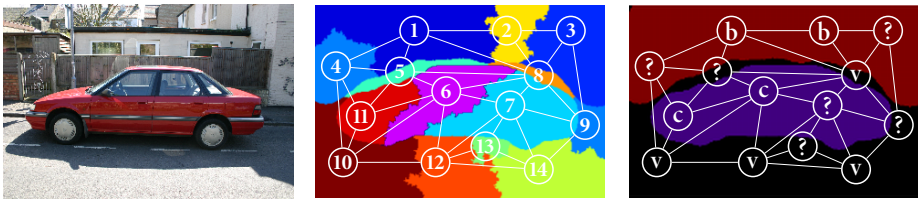


Figure 3.2: **Image Scene Graph.** Scene graph representation of and image with partially annotated regions. The image an ground truth labeling is taken from the "MSRC semantic scenes" dataset which is publicly available at `http://research.microsoft.com/en-us/projects/objectclassrecognition/`.

(a) Household objects (b) Office environment

Figure 3.3: **Point Clouds.** Semantically annotated point clouds of household objects (a) and part of an entire office environment (b). The figures in panel (b) are taken from the "Scene Understanding for Personal Robots" dataset created by the Robot Learning Lab at Cornell University (`http://pr.cs.cornell.edu/sceneunderstanding/`).

themselves forming another network. Further, information on the nodes can be incomplete and uncertain as it often comes from complex sensor measurements or manual labeling processes.

Surprisingly, existing work on graph kernels does not broadly account for these challenges. In recent years several graph kernels have been developed within the graph mining community. Categorizing graph kernels with respect to how the graph structure is captured, we can distinguish four classes: graph kernels based on walks, e.g. Equation (2.28) (GÄRTNER, *et al.*, 2003; KASHIMA, *et al.*, 2003; VISHWANATHAN, *et al.*, 2010; HARCHAOUI and BACH, 2007; BAI, *et al.*, 2014), on paths, e.g. Equation (2.31) (BORGWARDT and KRIEGEL, 2005; FERAGEN, *et al.*, 2013), graph kernels based on limited-size subgraphs, e.g. Equation (2.29) (HORVÁTH, *et al.*, 2004; SHERVASHIDZE, *et al.*, 2009; KRIEGE and MUTZEL, 2012), and graph kernels based on subtree-patterns, e.g. Equation (2.30) (RAMON and GÄRTNER, 2003; BACH, 2008; MAHÉ and VERT, 2009; SHERVASHIDZE, *et al.*, 2011). Further,

Table 3.1: **Graph Kernels and their Intended Use.** Intended use of state-of-the-art graph kernels: graphlet count kernel (GC), Weisfeiler–Lehman subtree kernel (WL), cyclic pattern kernel (CP), random walk kernel (RWK), shortest-path kernel (SP), graphHopper kernel (GH), common subgraph matching kernel (CSM), and propagation kernel (PK). *Fast scaling to huge graphs or grid graphs.

kernel	information type						
	NODE LABELS	PARTIAL LABELS	EDGE WEIGHTS	EDGE LABELS	NODE ATTRIBUTES	HUGE GRAPHS*	HUGE GRIDS*
GC	(yes)	–	–	–	–	yes	–
WL	yes	–	–	–	–	yes	–
CP	yes	–	–	yes	–	yes	–
RWK	yes	–	yes	yes	yes	–	–
SP	yes	–	yes	yes	yes	–	–
GH	yes	–	yes	–	yes	–	–
CSM	yes	–	yes	yes	yes	–	–
PK	yes	yes	yes	–	yes	yes	yes

graph kernels can be grouped with respect to their intended use of information. We separate graph kernels designed for unlabeled graphs, labeled graphs, and attributed graphs. An overview of existing graph kernels and their intended use is shown in Table 3.1. This summary clearly reveals that most existing graph kernels suffer from one of the following drawbacks: they can only handle graphs with complete label or attribute information in a principled manner and they are either efficient, but limited to specific graph types, or they are flexible, but their computation is memory and/or time consuming. Whereas there are efficient graph kernels specifically designed for unlabeled graphs (SHERVASHIDZE, *et al.*, 2009), fully labeled graphs (SHERVASHIDZE, *et al.*, 2011), attributed graphs (FERAGEN, *et al.*, 2013), or planar labeled graphs (HARCHAOUI and BACH, 2007), there are more flexible but slow and memory intense graph kernels such as the shortest paths, subgraph matching, and random walk kernels (GÄRTNER, *et al.*, 2003; BORGWARDT and KRIEGEL, 2005; VISHWANATHAN, *et al.*, 2010; KRIEGE and MUTZEL, 2012). Moreover, most of the existing graph kernels cannot take uncertain node information, or weighted and directed edges into account in a principled manner.

The Weisfeiler–Lehman (WL) subtree kernel, one instance of the recently introduced family of WL-kernels (SHERVASHIDZE, *et al.*, 2011), computes for example count features for each graph based on the signatures arising from iterative multiset label determination and compression steps. In every kernel iteration, these features are then the inputs to a base kernel and the WL-kernel is the sum of those base kernels over the iterations, cf. Equation (2.30). This approach is currently one of the most efficient and accurate graph kernels for fully labeled graphs. The

challenge of comparing large, partially labeled graphs, however, remains to a large extent unsolved. One proposal for extending a graph kernel to the partially labeled case is to mark unlabeled nodes with a unique symbol, as suggested in (SHERVASHIDZE, *et al.*, 2011). However, collapsing all unlabeled nodes into a single label neglects any notion of uncertainty in the labels. Another option is to propagate labels across the graph and then run a graph kernel on the imputed labels. Unfortunately, this also ignores the uncertainty induced by the inference procedure, as hard labels have to be assigned after convergence. Moreover, a two-stage approach may run many unnecessary label propagation iterations.

Further, many datasets, as for instance images and videos, can be naturally modeled by graphs with regular neighborhood structure, also called *grid graphs*. None of the existing graph kernels scales to huge grid graphs. If, however, graph kernels would be capable to handle such structures we could connect graph mining and machine learning on structured data to the entire research field of pattern recognition and computer vision. Thus, both fields could benefit from new theoretical insights, applications, and solution approaches.

3.2 Our Approach: Propagation Kernels

To overcome these problems, we introduce *propagation kernels*. Their design is motivated by the observation that iterative information propagation schemes originally developed for within-network relational and semi-supervised learning have two desirable properties: they capture structural information and they can often adapt to the aforementioned issues of real-world data. A key observation motivating propagation kernels is that intermediate label distributions will, before convergence, carry information about the structure of the graph. Propagation kernels interleave label inference and kernel computation steps, avoiding the requirement of running inference to termination. In particular, propagation schemes such as diffusion or label propagation, introduced in Section 2.3.2, can be computed efficiently and they can be initialized with uncertain and partial information.

3.2.1 Introduction

A high-level overview of the propagation kernel algorithm is as follows. We begin by initializing label and/or attribute distributions for every node in the graphs at hand. We then iteratively propagate this information along edges using an appropriate propagation scheme. By maintaining entire distributions of labels and attributes, we can accommodate uncertain information in a natural way. After each iteration, we compare the similarity of the induced node distributions between each pair of graphs. Structural similarities between graphs will tend to induce similar local distributions during the propagation process, and our kernel will be based on approximate counts of the number of induced similar distributions throughout the information propagation.

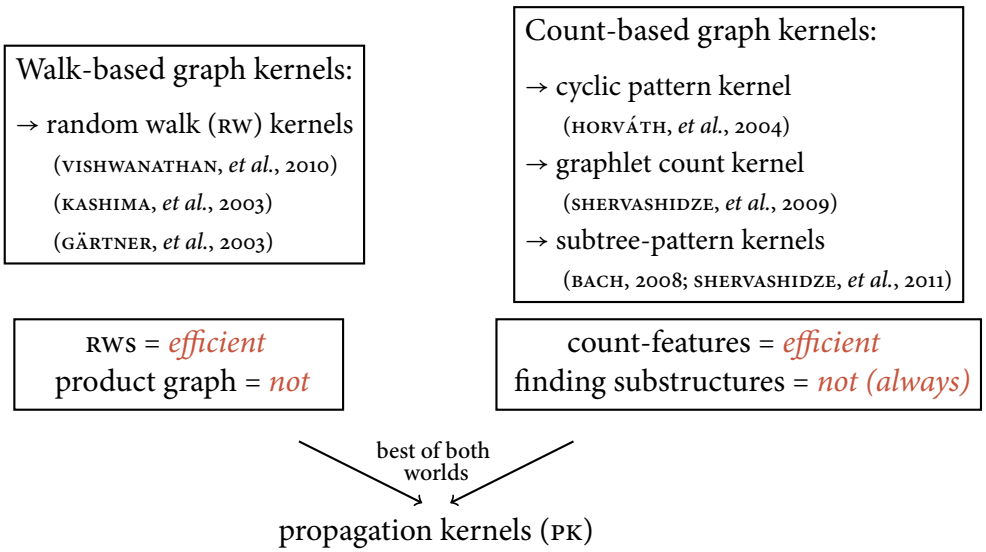


Figure 3.4: **Propagation Kernels: Relation to other Graph Kernels.** This figure illustrates the relation of propagation kernels to other graph kernels. PKs interleave random walk-based kernels and count feature-based kernels by taking the efficient computation steps of both graph kernel computation strategies.

To achieve competitive running times and to avoid having to compare the distributions of all pairs of nodes between two given graphs, we will exploit *locality sensitive hashing* (LSH) to bin the label and attribute distributions into efficiently computable graph feature vectors in time linear in the total number of nodes. These new graph features will then be fed into a base kernel, a common scheme for constructing graph kernels. Whereas LSH is usually used to preserve the ℓ^1 or ℓ^2 distance, we are able to show that the hash values can preserve both the total variation and the Hellinger probability metrics. Exploiting explicit feature computation and efficient information propagation, propagation kernels allow for using graph kernels to tackle novel applications beyond the classical benchmark problems on datasets of chemical compounds and small- to medium-sized image or point-cloud graphs.

As illustrated in Figure 3.4, propagation kernels interleave random walk-based and count feature-based graph kernels by taking the computationally efficient parts of both computation strategies. Propagation kernels explicitly compute count features, which allows them to be computed faster and more memory efficient than graph kernels relying on the product graph. By counting similar distributions resulting from random walks lifted to the labels or attributes of the graphs' nodes they avoid searching for substructures in the graphs and they can naturally adapt to missing and uncertain information.

Many state-of-the-art graph kernels are tailored to compare chemical compounds (HORVÁTH, *et al.*, 2004; BORGWARDT, *et al.*, 2005; WALE, *et al.*, 2008; MAHÉ and VERT, 2009). And a lot of generally defined graph kernels are exclusively evaluated on molecule benchmark datasets from bioinformatics (KRIEGE and MUTZEL, 2012; SHERVASHIDZE, *et al.*, 2011; VISHWANATHAN, *et al.*, 2006; KASHIMA, *et al.*, 2003). Other graph kernels are tailored to very specific settings as structured mathematical domains such as formulas and terms (TSIVTSIVADZE, *et al.*, 2011). The datasets in these domains, however, mainly consist of "nice" graphs, that are fully labeled, have discrete and certain node and edge information, and reasonable graph sizes. Thus, it is little surprising that most existing graph kernels do not broadly account for challenges arising in general domains of structured data, such as uncertain and partial label information, and densely connected and huge individual graphs. So, how can we construct an expressive kernel which is both flexible, and memory and time efficient? The main insight towards answering this question is that:

A suitable propagation scheme is the key to design
fast and powerful propagation kernels.

The main goal in choosing a suitable propagation scheme is to allow for both runtime and memory efficient kernel computation as well as meaningful structure representation. For example when dealing with domains of graphs of regular structure, such as grid graphs representing images or videos, we can employ efficient propagation schemes based on discrete convolution to achieve fast and powerful graph kernels allowing us to perform graph-based image analysis not only on the scene level (HARCHAOUI and BACH, 2007) but also on the pixel level opening up novel application domain for graph kernels. One interesting application here is texture classification.

3.2.2 Related Work

Beside graph kernels, propagation kernels are related to two lines of research: within-network relational learning and kernels on graphs, cf. Section 2.4.2, and kernels between probability distributions and sets.

Measuring the structural similarity of local node neighborhoods has recently become popular for inference in networked data (KONDOR and LAFFERTY, 2002; DESROSIERS and KARYPIS, 2009) where this idea has been used for designing kernels on graphs (kernels between the nodes of a graph) and for within-network relational learning approaches. DESROSIERS and KARYPIS (2009) use a similarity measure based on parallel random walks with constant termination probability in a relaxation-labeling algorithm. Another approach exploiting random walks and the structure of subnetworks for node label prediction is heterogeneous label propagation (HWANG and KUANG, 2010). Random walks with restart are used as proximity weights for so-called "ghost edges" in (GALLAGHER, *et al.*, 2008), but then the features considered by a later bag of logistic regression classifiers are only based on a one-step neighborhood. The connection between these approaches and

propagation kernels, which is based on the use of random walks to measure structure similarity, constitutes an important contact point of graph-based machine learning for inference about node- and graph-level properties.

Propagation kernels mark another contact point, namely between graph kernels and kernels between probability distributions (JAAKKOLA and HAUSSLER, 1998; LAFFERTY and LEBANON, 2002; MORENO, *et al.*, 2003; JEBARA, *et al.*, 2004) and between sets (KONDOR and JEBARA, 2003; SHI, *et al.*, 2009). However, whereas the former essentially build kernels based on the outcome of probabilistic inference after convergence, propagation kernels intuitively count common sub-distributions induced after each iteration of running inference in two graphs. Kernels between sets and more specifically between structured sets, also called *hash kernels* (SHI, *et al.*, 2009), have been successfully applied to strings, data streams, and unlabeled graphs. While propagation kernels hash probability distributions and derive count features from them, hash kernels directly approximate the kernel values $k(x, x')$, where x and x' are (structured) sets. Propagation kernels iteratively approximate node kernels $k(v_i, v_j)$ comparing a node v_i in graph $G^{(i)}$ with a node v_j in graph $G^{(j)}$. Counts summarizing these approximations are then fed into a base kernel that is computed exactly.

In the following chapter, we introduce the family of propagation kernels and discuss specific examples of the two main components of propagation kernels: node kernels for comparing propagated information (Section 4.2) and propagation schemes for various graph types, such as labeled, partially labeled, unlabeled, directed, and attributed graphs (Section 4.3). In Chapter 5, we will show how to efficiently compute propagation kernels for grid graphs by means of convolutions. In both chapters, we will demonstrate the feasibility and power of propagation kernels on various real-world graph databases by providing experimental results on several challenging classification problems including commonly used bioinformatics benchmark problems, semantic scene prediction, and image-based texture classification.

Even though the problem definition and evaluation framework focus on classification all the techniques developed in the following apply to graph regression and ranking without exceptions.

CHAPTER 4

PROPAGATION KERNELS FOR GENERAL GRAPHS

In the following, we introduce *propagation kernels* (PKs). The main insight here is, that the intermediate node label distributions, e.g., the iterative distribution updates of a random-walk-based information propagation scheme, capture label *and* structure information of a graph. Further, the framework naturally extends to missing, uncertain, and continuous information on the nodes.

4.1 Propagation Kernel Framework

In this section, we provide the general definitions of the propagation kernel family and analyze their computational complexity.

4.1.1 General Definition

Here we will define a kernel $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ among graph instances $G^{(i)} \in \mathcal{X}$ according to Definition 2.19. Hence, the input space $\mathcal{X}$ comprises graphs $G^{(i)} = (V^{(i)}, E^{(i)}, \ell)$, where $V^{(i)}$ is the set of nodes and $E^{(i)}$ is the set of edges in graph $G^{(i)}$. Edge weights are represented by weighted adjacency matrices $A^{(i)} \in \mathbb{R}^{n_i \times n_i}$ and the label function ℓ endows nodes with label and attribute information[1] as defined in Section 2.1.2.

A simple way to compare two graphs $G^{(i)}$ and $G^{(j)}$ is to compare all pairs of nodes in the two graphs:

$$K(G^{(i)}, G^{(j)}) = \sum_{v \in G^{(i)}} \sum_{u \in G^{(j)}} k(u, v),$$

where $k(u, v)$ is an arbitrary node kernel determined by node labels and, if present, node attributes. This simple graph kernel, however, does not account for graph

[1] Note that not both label and attribute information have to present and both could also be partially observed.

structure given by the arrangement of node labels and attributes in the graphs. Hence, we consider a *sequence* of graphs $G_t^{(i)}$ with evolving node information based on information propagation, as introduced for node labels in Section 2.3. We define the kernel contribution of iteration t by

$$K(G_t^{(i)}, G_t^{(j)}) = \sum_{v \in G_t^{(i)}} \sum_{u \in G_t^{(j)}} k(u, v). \tag{4.1}$$

An important feature of propagation kernels is that the node kernel $k(u, v)$ is defined in terms of the nodes' corresponding probability distributions $p_{t,u}$ and $p_{t,v}$, which we update and maintain throughout the process of information propagation. For propagation kernels between labeled and attributed graphs we define

$$k(u, v) = k_l(u, v) \cdot k_a(u, v), \tag{4.2}$$

where $k_l(u, v)$ is a kernel corresponding to label information and $k_a(u, v)$ is a kernel corresponding to attribute information. If no attributes are present, then $k(u, v) = k_l(u, v)$. The t_{MAX}-iteration propagation kernel is now given by

$$K_{t_{\text{MAX}}}(G^{(i)}, G^{(j)}) = \sum_{t=1}^{t_{\text{MAX}}} K(G_t^{(i)}, G_t^{(j)}). \tag{4.3}$$

LEMMA 4.1 (PROPAGATION KERNELS ARE POSITIVE SEMIDEFINITE) *Given that $k_l(u, v)$ and $k_a(u, v)$ are positive-semidefinite node kernels, then the propagation kernel $K_{t_{MAX}}$ is a positive-semidefinite kernel.*

PROOF As $k_l(u, v)$ and $k_a(u, v)$ are assumed to be valid node kernels, $k(u, v)$ is a valid node kernel as the product of positive-semidefinite kernels is again positive semidefinite. As for a given graph $G^{(i)}$ the number of nodes is finite, we can apply Theorem 2.3. $K(G_t^{(i)}, G_t^{(j)})$ is a convolution kernel (HAUSSLER, 1999). As sums of positive-semidefinite matrices are again positive semidefinite, the propagation kernel as defined in Equation (4.3) is positive semidefinite. □

Let $|V^{(i)}| = n_i$ and $|V^{(j)}| = n_j$. Assuming that all node information is given, the complexity of computing each contribution between two graphs, Equation (4.1), is $\mathcal{O}(n_i\, n_j)$. Even for medium-sized graphs this can be prohibitively expensive considering that the computation has to be performed for *every* pair of graphs in a possibly large graph database. However, if we have a node kernel of the form

$$k(u, v) = \begin{cases} 1 & \text{if } \textit{condition} \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

where *condition* is an equality condition on the information of nodes u and v, we can compute K efficiently by *binning* the node information, *counting* the respective

Algorithm 1 General Propagation Kernel Computation

given: graph database $\{G^{(i)}\}_i$, # iterations $t_{\max}$, propagation scheme(s), base kernel $\langle \cdot, \cdot \rangle$
initialization: $K \leftarrow 0$, *initialize distributions* $P_0^{(i)}$
for $t \leftarrow 0 \dots t_{\max}$ **do**
  for all graphs $G^{(i)}$ **do**
    for all nodes $u \in G^{(i)}$ **do**
      quantize $p_{t,u}$, where $p_{t,u}$ is u-th row in $P_t^{(i)}$ ▷ bin node inform.
    end for
    compute $\Phi_{i\cdot} = \phi(G_t^{(i)})$ ▷ count bin strengths
  end for
  $K \leftarrow K + \langle \Phi, \Phi \rangle$ ▷ compute and add kernel contribution
  for all graphs $G^{(i)}$ **do**
    $P_{t+1}^{(i)} \leftarrow P_t^{(i)}$ ▷ propagate node information
  end for
end for

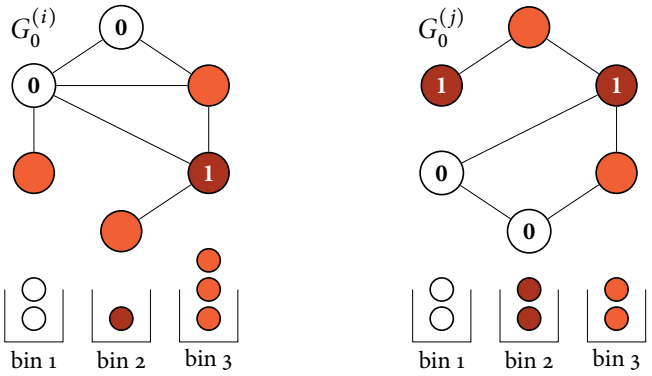
bin strengths for all graphs, and *computing a base kernel* among these counts. That is, we compute count features $\phi(G_t^{(i)})$ for each graph and plug them into a base kernel $\langle \cdot, \cdot \rangle$:

$$K(G_t^{(i)}, G_t^{(j)}) = \langle \phi(G_t^{(i)}), \phi(G_t^{(j)}) \rangle. \tag{4.5}$$

In the simple case of a linear base kernel, the last step is just an outer product of count vectors $\Phi_t \Phi_t^\top$, where the ith row of Φ_t, $(\Phi_t)_{i,:} = \phi(G_t^{(i)})$. Now, for two graphs, binning and counting can be done in $\mathcal{O}(n_i + n_j)$ and the computation of the linear base kernel value is $\mathcal{O}(|\text{bins}|)$. This is a commonly exploited insight for efficient graph-kernel computation and it has already been used for labeled graphs in previous work (SHERVASHIDZE, *et al.*, 2011).

Figure 4.1 illustrates the propagation kernel computation for $t = 0$ and $t = 1$ for two example graphs and Algorithm 1 summarizes the kernel computation for a graph database $\mathbf{G} = \{G^{(i)}\}_i = (V, E, \ell)$ with a total number of N nodes. From this general algorithm and Equations (4.1) and (4.3), we see that the two main components to design a propagation kernel are

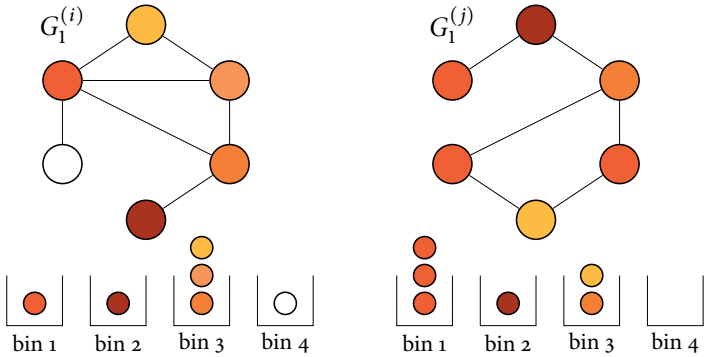
- the **node kernel** $k(u, v)$ comparing propagated information, and
- the **propagation scheme** $P_{t+1}^{(i)} \leftarrow P_t^{(i)}$ propagating the information within the graphs.



$\phi(G_0^{(i)}) = [2, 1, 3]^\top$ $\phi(G_0^{(j)}) = [2, 2, 2]^\top$

$$\langle \phi(G_0^{(i)}), \phi(G_0^{(j)}) \rangle = 12$$

(a) Initial label distributions and kernel contribution ($t = 0$)



$\phi(G_1^{(i)}) = [1, 1, 3, 1]^\top$ $\phi(G_1^{(j)}) = [3, 1, 2, 0]^\top$

$$\langle \phi(G_1^{(i)}), \phi(G_1^{(j)}) \rangle = 10$$

(b) Updated label distributions and kernel contribution ($t = 1$)

Figure 4.1: **Propagation Kernel Computation.** Distributions, bins, count features, and kernel contributions for two graphs $G^{(i)}$ and $G^{(j)}$ with binary node labels and one iteration of label propagation, cf. Equation (7.2), as the propagation scheme. Node-label distributions are decoded by color: white means $p_{0,u} = [1, 0]$, dark red stands for $p_{0,u} = [0, 1]$, and the initial distributions for unlabeled nodes (light red) are $p_{0,u} = [1/2, 1/2]$.

The propagation scheme depends on the input graphs and we will give specific suggestions for different graph types in Section 4.3. Before defining the node kernels depending on the available node information in Section 4.2, we briefly discuss the general runtime complexity of propagation kernels.

4.1.2 *Complexity Analysis*

The total runtime complexity of propagation kernels for a set of n graphs with a total number of N nodes and M edges is $\mathcal{O}\big((t_{\max}-1)M + t_{\max}\, n^2\, n^\star\big)$, where $n^\star := \max_i(n_i)$. For a pair of graphs the runtime complexity of computing the count features, that is, binning the node information and counting the bin strengths is $\mathcal{O}(n_i + n_j)$. Computing and adding the kernel contribution is $\mathcal{O}(|\text{bins}|)$, where $|\text{bins}|$ is bounded by $n_i + n_j$. So, one iteration of the kernel computation for all graphs is $\mathcal{O}(n^2\, n^\star)$. Note that in practice $|\text{bins}| \ll 2n^\star$ as we aim to bin together similar nodes to derive a meaningful feature representation.

Feature computation basically depends on propagating node information along the edges of all graphs. This operation depends on the number of edges and the information propagated, so it is $\mathcal{O}((k+D)M) = \mathcal{O}(M)$, where k is the number of node labels and D is the attribute dimensionality. This operation has to be performed $t_{\max} - 1$ times. Note that the number of edges is usually much lower than N^2.

4.2 PK–Component 1: Node Kernel

In this section, we define node kernels comparing propagated information appropriate for the use in propagation kernels. Moreover, we introduce *locality sensitive hashing,* which is used to discretize the distributions arsing from RW-based information propagation and the continuous attribute vectors directly.

4.2.1 *Definitions*

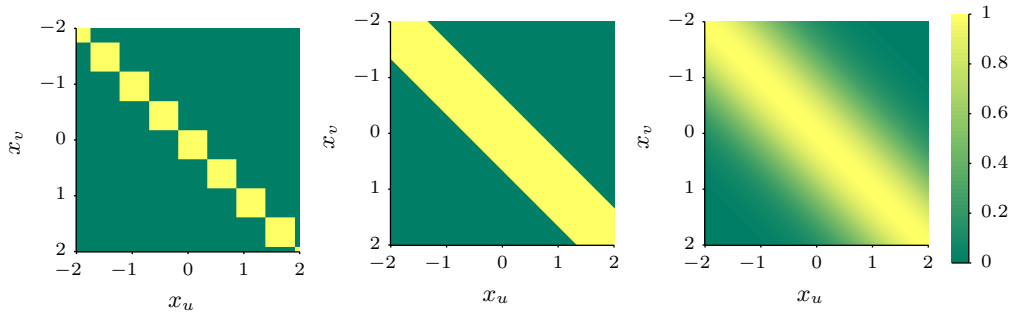
Above, we saw that one way to allow for efficient computation of propagation kernels is to restrict the range of the node kernels to $\{0,1\}$. Let us now define the two components of the node kernel (Equation (4.2)) in this form. The *label kernel* can be represented as

$$k_l(u,v) = \begin{cases} 1 & \text{if } h_l(p_{t,u}) = h_l(p_{t,v}) \\ 0 & \text{otherwise,} \end{cases} \tag{4.6}$$

where $p_{t,u}$ is the node-label distribution of node u at iteration t and $h_l(\cdot)$ is a quantization function (GERSHO and GRAY, 1991), more precisely a locality sensitive hash (LSH) function (DATAR, *et al.*, 2004), which will be introduced in more detail in the next section. Note that $p_{t,u}$ denotes a row in the label distribution matrix $P_t^{(i)}$, namely the row corresponding to node u of graph $G^{(i)}$.

Propagation kernels can be computed for various kinds of attribute kernels as long as they have the form of Equation (4.4). The most rudimentary *attribute kernel* is

$$k_a(u,v) = \begin{cases} 1 & \text{if } h_a(x_u) = h_a(x_v) \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$



(a) $k_a(u, v)$ (b) thresholded Gaussian (c) Gaussian kernel

Figure 4.2: **Attribute Kernels.** Three different attribute functions among nodes with continuous attributes $x_u, x_v \in [-2, 2]$. Panel (a) illustrates an attribute kernel suitable for the efficient computation in propagation kernels, panel (b) shows a thresholded Gaussian, and panel (c) a Gaussian kernel. Note that each of the illustrated functions has a parameter regulating its "widths", which is either its support or variance.

where x_u is the one-dimensional continuous attribute of node u and $h_a(\cdot)$ is again an LSH function. Figure 4.2 contrasts this simple attribute kernel for a one-dimensional attribute to a thresholded Gaussian function and the Gaussian kernel (Equation (2.17)) commonly used to compare node attributes in graph kernels. To deal with higher-dimensional attributes, we can choose the attribute kernel to be the product of kernels on each attribute dimension:

$$k_a(u, v) = \prod_{d=1}^{D} k_{a_d}(u, v), \text{ where}$$
$$k_{a_d}(u, v) = \begin{cases} 1 & \text{if } h_{a_d}(x_u^{(d)}) = h_{a_d}(x_v^{(d)}) \\ 0 & \text{otherwise,} \end{cases} \tag{4.8}$$

where $x_u^{(d)}$ is the respective dimension of the attribute $\mathbf{x}_u$ of node u. Note that each dimension now has its own LSH function $h_{a_d}(\cdot)$. However, analogous to the label kernel, we can also define an attribute kernel based on propagated attribute distributions

$$k_a(u, v) = \begin{cases} 1 & \text{if } h_a(q_{t,u}) = h_a(q_{t,v}) \\ 0 & \text{otherwise,} \end{cases} \tag{4.9}$$

where $q_{t,u}$ is the attribute distribution of node u at iteration t. Next we explain the locality sensitive hashing approach used to discretize distributions and continuous attributes. In Section 4.3.3, we will then derive an efficient way to propagate and hash continuous attribute distributions.

4.2.2 *Locality Sensitive Hashing*

We now describe our quantization approach for implementing propagation kernels for graphs with node label distributions and continuous attributes. We take our inspiration from locality sensitive hashing (DATAR, *et al.*, 2004), which seeks for quantization functions on metric spaces where points "close enough" to each other in that space are "probably" assigned to the same bin. In the case of distributions, we will consider each node label vector as being an element of the space of discrete probability distributions on k items equipped with an appropriate probability metric. If we want to hash attributes directly we simply consider metrics for continuous values.

DEFINITION 4.1 (LOCALITY SENSITIVE HASH (LSH)) *Let $\mathcal{X}$ be a metric space with metric $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and let $\mathcal{Y} = \{1, 2, \ldots, k'\}$. Let $\theta > 0$ be a threshold, $c > 1$ be an approximation factor, and $p_1, p_2 \in (0,1)$ be the given success probabilities. A set of functions $\mathcal{H}$ from $\mathcal{X}$ to $\mathcal{Y}$ is called a* $(\theta, c\theta, p_1, p_2)$-locality sensitive hash *if for any function $h \in \mathcal{H}$ chosen uniformly at random, and for any two points $x, x' \in \mathcal{X}$, it holds that*

- *if $d(x, x') < \theta$, then $\Pr(h(x) = h(x')) > p_1$, and*
- *if $d(x, x') > c\theta$, then $\Pr(h(x) = h(x')) < p_2$.*

It is known that we can construct LSH families for ℓ^p spaces with $p \in (0, 2]$ (DATAR, *et al.*, 2004). Let V be a real-valued random variable. V is called p-stable if for any $\{x_1, x_2, \ldots, x_d\}$, $x_i \in \mathbb{R}$ and independently sampled $v_1, v_2, \ldots, v_d$, we have $\sum x_i v_i \sim \|\mathbf{x}\|_p V$. Explicit p-stable distributions are known for some p; for example, the standard Cauchy distribution is 1-stable, and the standard normal distribution is 2-stable. Given the ability to sample from a p-stable distribution V, we may define a LSH $\mathcal{H}$ on $\mathbb{R}^d$ with the ℓ^p metric (DATAR, *et al.*, 2004). An element h of $\mathcal{H}$ is specified by three parameters: a width $w \in \mathbb{R}^+$, a d-dimensional vector $\mathbf{v}$ whose entries are independent samples of V, and $b \in [0, w]$ drawn from $\mathcal{U}[0, w]$, and defined as

$$h(\mathbf{x}; w, \mathbf{v}, b) = \left\lfloor \frac{\mathbf{v}^\top \mathbf{x} + b}{w} \right\rfloor. \tag{4.10}$$

We may now consider $h(\cdot)$ to be a function mapping our distributions or attribute values to integer-valued bins, where similar distributions end up in the same bin. Hence, we get node kernels as defined in Equations (4.6) and (4.9) in the case of distributions, as well as simple attribute kernels as defined in Equations (4.7) and (4.8). To decrease the probability of collision it is common to choose more than one random vector $\mathbf{v}$. For propagation kernels, however, we only use one hyperplane, as we effectively have $t_{\max}$ hyperplanes for the whole kernel computation and the probability of a hash conflict is reduced over the iterations.

The intuition behind the expression in Equation (4.10) is that p-stability implies that two vectors that are close under the ℓ^p norm will be close after taking the dot

Algorithm 2 CALCULATE-LSH

given: matrix $X \in \mathbb{R}^{N \times D}$, bin width w, metric M
if M = H **then**
  $X \leftarrow \sqrt{X}$ ▷ square root transformation
end if
if M = H or M = L2 **then** ▷ generate random projection vector
  $v \leftarrow$ RAND-NORM(D) ▷ sample from $\mathcal{N}(0,1)$
else if M = TV or M = L1 **then**
  $v \leftarrow$ RAND-NORM(D)/RAND-NORM(D) ▷ sample from $\mathcal{C}auchy(0,1)$
end if
$b = w *$ RAND-UNIF() ▷ random offset $b \sim \mathcal{U}[0, w]$
$h(X) = \text{floor}((X * v + b)/w)$ ▷ compute hashes

product with $\mathbf{v}$; specifically, $(\mathbf{v}^\top \mathbf{x} - \mathbf{v}^\top \mathbf{x}')$ is distributed as $\|\mathbf{x} - \mathbf{x}'\|_p V$. So, in the case where we want to construct a hashing for D-dimensional continuous node attributes to preserve ℓ^1 (L1) or ℓ^2 (L2) distance

$$d_{\text{L1}}(x_u, x_v) = \sum_{d=1}^{D} |x_u^{(d)} - x_v^{(d)}|, \quad d_{\text{L2}}(x_u, x_v) = \left(\sum_{d=1}^{D} \left(x_u^{(d)} - x_v^{(d)} \right)^2 \right)^{1/2},$$

we directly apply Equation (4.10). In the case of distributions, we are concerned with the space of discrete probability distributions on k elements, endowed with a probability metric d. Here we specifically consider the *total variation* (TV) and *Hellinger* (H) distances:

$$d_{\text{TV}}(p_u, p_v) = 1/2 \sum_{i=1}^{k} |p_u^{(i)} - p_v^{(i)}|, \quad d_{\text{H}}(p_u, p_v) = \left(1/2 \sum_{i=1}^{k} \left(\sqrt{p_u^{(i)}} - \sqrt{p_v^{(i)}} \right)^2 \right)^{1/2}.$$

The total variation distance is simply half the ℓ^1 metric, and the Hellinger distance is a scaled version of the ℓ^2 metric after applying the map $p \mapsto \sqrt{p}$. We may therefore create a locality-sensitive hash family for d_{TV} by direct application of Equation (4.10), and create a locality-sensitive hash family for d_{H} by applying Equation (4.10) after applying the square root map to our label distributions. The LSH computation for a matrix $X \in \mathbb{R}^{N \times D}$, where the uth row of X is $\mathbf{x}_u$, is summarized in Algorithm 2.

4.3 PK–Component 2: Propagation Scheme

As pointed out in Section 2.1.1 and the previous chapter, the input graphs for graph kernels may vary considerably. One key insight to the design of efficient and powerful propagation kernels is to choose an appropriate propagation scheme for the graph dataset at hand. By utilizing random walks (RWS) we are able to use efficient

Algorithm 3 Propagation Kernel for Fully Labeled Graphs

given: graph database $\mathbf{G} = (V, E, \ell)$, # iterations $t_{\max}$, transition matrix T, bin width w, metric M, base kernel $\langle \cdot, \cdot \rangle$
initialization: $K \leftarrow 0$, $P_0 \leftarrow \delta_{\ell(V)}$
for $t \leftarrow 0 \dots t_{\max}$ **do**
  CALCULATE-LSH(P_t, w, M) ▷ bin node information
  for all graphs $G^{(i)}$ **do**
    compute $\Phi_{i\cdot} = \phi(G_t^{(i)})$ ▷ count bin strengths
  end for
  $K \leftarrow K + \langle \Phi, \Phi \rangle$ ▷ compute and add kernel contribution
  $P_{t+1} \leftarrow TP_t$ ▷ label diffusion
end for

off-the-shelf algorithms, such as label diffusion or label propagation (SZUMMER and JAAKKOLA, 2001; ZHU, *et al.*, 2003; WU, *et al.*, 2012), to implement information propagation within the input graphs. In this section, we explicitly define propagation kernels appropriate for fully labeled, unlabeled, partially labeled, directed, and attributed graphs. In each particular algorithm, the specific parts changing compared to the general propagation kernel computation (Algorithm 1) will be marked in color.

4.3.1 Labeled and Unlabeled Graphs

For *fully labeled graphs* we suggest the use of the label diffusion process from Equation (2.10) as the propagation scheme. Given a database of fully labeled graphs $\mathbf{G} = \{G^{(i)}\}_{i=1,\dots,n}$ with a total number of $N = \sum_i n_i$ nodes, label diffusion on all graphs can be efficiently implemented by multiplying a sparse block-diagonal transition matrix $T \in \mathbb{R}^{N \times N}$, where the blocks are the transition matrices $T^{(i)}$ of the respective graphs, with the label distribution matrix $P_t = \left[P_t^{(1)}, \dots, P_t^{(n)}\right]^\top \in \mathbb{R}^{N \times k}$. This can be done efficiently due to the sparsity of T. The propagation kernel computation for labeled graphs is summarized in Algorithm 3. The specific parts compared to the general propagation kernel computation (Algorithm 1) for fully labeled graphs are marked in green (input) and blue (computation). For *unlabeled graphs* we suggest to set the label function to be the node degree $\ell(u) = \deg(u)$ and then apply the same PK computation as for fully labeled graphs.

4.3.2 Partially Labeled and Directed Graphs

For *partially labeled graphs,* where some of the node labels are unknown, we suggest *label propagation* as an appropriate propagation scheme. Label propagation differs from label diffusion in the fact that before each iteration of the information propagation, the labels of the originally observed nodes are pushed back (ZHU, *et al.*, 2003). Label propagation as defined in Equation (2.11) can be implemented

for all graphs in the graph database in the same way as described for the label diffusion process previously. That is,

$$P_{t,[labeled]} \leftarrow P_{0,[labeled]}$$
$$P_{t+1} \leftarrow T\, P_t$$

with $P_t \in \mathbb{R}^{N\times k}$, $T \in \mathbb{R}^{N\times N}$, and $P_0 = \left[P_{0,[labeled]}, P_{0,[unlabeled]}\right]^\top$, where the distributions in $P_{0,[labeled]}$ represent observed labels and $P_{0,[unlabeled]}$ are initialized uniformly. Other similar update schemes, such as "label spreading" (ZHOU, *et al.*, 2003), could be used in a propagation kernel as well. Thus, the propagation kernel computation for partially labeled graphs is essentially the same as Algorithm 3, where the initialization for the unlabeled nodes has to be adapted, and the (partial) label push back has to be added before the node information is propagated. The relevant parts are the ones marked in blue. Note that for graphs with large fractions of labeled nodes it might be preferable to use label diffusion even though they are partially labeled, as the push back prevents the kernel to capture any graph structure beyond the one-step neighborhoods.

To implement propagation kernels between *directed graphs*, we can proceed as above after simply deriving transition matrices computed from the potentially non-symmetric adjacency matrices. That is, for the propagation kernel computation only the input changes (marked in green in Algorithm 3). The same idea allows weighted edges to be accommodated; again, only the transition matrix has to be adapted. Obviously, we can also combine partially labeled graphs with directed or weighted edges by changing both the blue and green marked parts accordingly.

4.3.3 Graphs with Continuous Node Attributes

Nowadays, learning tasks often involve graphs whose nodes are attributed with continuous information. Chemical compounds can be annotated with the length of the secondary structure elements (represented as nodes) or measurements for various properties, such as hydrophobicity or polarity. 3D point clouds can be enriched with curvature information, and images are inherently composed of 3-channel color information. All this information can be modeled by continuous node attributes. In Equation (4.7) we introduced a simple way to deal with attributes. The resulting propagation kernel essentially counts similar label arrangements only if the corresponding node attributes are similar as well. Note that for higher-dimensional attributes it can be advantageous to compute separate LSHs per dimension, leading to the node kernel introduced in Equation (4.8). This has the advantage that if we standardize the attributes, we can use the same bin-width parameter w_a for all dimensions. In all our experiments we take this approach and normalize each attribute to have unit standard deviation; w_a is set to 1. The disadvantage of this method, however, is that the arrangement of attributes in the graphs is ignored.

Algorithm 4 Propagation Kernel (P2K) for Attributed Graphs

given: graph database $\mathbf{G} = (V, E, \ell)$, # iterations t_{MAX}, transition matrix T, bin widths w_l, w_a, metrics M_l, M_a, base kernel $\langle \cdot, \cdot \rangle$
initialization: $K \leftarrow 0$, $P_0 \leftarrow \delta_{\ell(V)}$
$\boldsymbol{\mu} = X$, $\Sigma = cov(X)$, $W_0 = I$ ▷ GM initialization
$y \leftarrow \text{RAND}(num\_samples)$ ▷ sample points for GM evaluations
for $t \leftarrow 0 \ldots t_{\text{MAX}}$ **do**
  $h_l \leftarrow \text{CALCULATE-LSH}(P_t, w_l, \text{M}_l)$ ▷ bin label distributions
  $Q_t \leftarrow \text{EVALUATE-PDFS}(\boldsymbol{\mu}, \Sigma, W_t, y)$ ▷ evaluate GMs at y
  $h_a \leftarrow \text{CALCULATE-LSH}(Q_t, w_a, \text{M}_a)$ ▷ bin attribute distributions
  $h \leftarrow h_l \wedge h_a$ ▷ combine label and attribute bins
  for all graphs $G^{(i)}$ **do**
    compute $\Phi_{i\cdot} = \phi(G_t^{(i)})$ ▷ count bin strengths
  end for
  $K \leftarrow K + \langle \Phi, \Phi \rangle$ ▷ compute and add kernel contribution
  $P_{t+1} \leftarrow TP_t$ ▷ propagate label information
  $W_{t+1} \leftarrow TW_t$ ▷ propagate attribute information
end for

In the following, we derive P2K, a variant of propagation kernels for *attributed graphs* based on the idea of propagating both attributes and labels. That is, we model graph similarity by comparing the arrangement of labels *and* the arrangement of attributes in the graph. The attribute kernel for P2K is defined as in Equation (4.9); now the question is how to efficiently propagate the continuous attributes and how to efficiently model and hash the distributions of (multivariate) continuous variables. Let $X \in \mathbb{R}^{N \times D}$ be the design matrix, where the uth row in X is the attribute vector of node u, $\mathbf{x}_u$. We will associate with each node of each graph a probability distribution defined on the attribute space, q_u, and will update these as attribute information is propagated across graph edges as before. One challenge in doing so is ensuring that these distributions can be represented with a finite description. The discrete label distributions from before were naturally finite dimensional and could be compactly represented and updated via the P_t matrices. We seek a similar representation for attributes. Our proposal is to define the node-attribute distributions to be mixtures of D-dimensional multivariate Gaussians, one centered on each attribute vector in X:

$$q_u = \sum_v W_{uv} \mathcal{N}(\mathbf{x}_v, \Sigma),$$

where the sum ranges over all nodes v, $W_{u,:}$ is a vector of mixture weights, and Σ is a shared $D \times D$ covariance matrix for each component of the mixture. In particular, here we set Σ to be the sample covariance matrix calculated from the N

vectors in X. Now the $N \times N$ row-normalized matrix W can be used to compactly represent the entire set of Gaussian mixture (GM) distributions of the attributes. As before, we will use the graph structure to iteratively spread attribute information, updating W, and in turn deriving a sequence of attribute distributions for each node to use as inputs to node attribute kernels in a propagation kernel scheme.

We begin by defining the initial weight matrix W_0 to be the identity matrix; this is equivalent to beginning with each node attribute distribution being a single Gaussian centered on the corresponding attribute vector:

$$W_0 = I$$
$$q_{0,u} = \mathcal{N}(\mathbf{x}_u, \Sigma).$$

Now, in each propagation step the attribute distributions are updated by the distribution of their neighboring nodes $Q_{t+1} \leftarrow Q_t$. We accomplish this by propagating the mixture weights W across the edges of the graph according to a row-normalized transition matrix T, derived as before:

$$W_{t+1} \leftarrow T W_t = T^t$$
$$q_{t+1,u} = \sum_v (W_t)_{uv} \mathcal{N}(\mathbf{x}_v, \Sigma). \tag{4.11}$$

We have described how attribute distributions are associated with each node and how they are updated via propagating their weights across the edges of the graph. However, the weight vectors contained in W are not themselves directly suitable for comparing in an attribute kernel $k_a(\cdot, \cdot)$, because any information about the similarity of the mean vectors is ignored. For example, imagine that two nodes u and v had exactly the same attribute vector, $\mathbf{x}_u = \mathbf{x}_v$. Then mass on the u component of the Gaussian mixture is exchangeable with mass on the v component, but typical vectorial kernels cannot capture this. For this reason, we use our Gaussian mixtures to associate a vector more appropriate for kernel comparison with each node. Namely, we select a fixed set of sample points in the attribute space (in our case, chosen uniformly from the node attribute vectors in X), evaluate the probability density functions (PDFs) associated with each node at these points, and use this vector to summarize the nodes. This handles the exchangeability issue from above and also allows a more compact representation for hash inputs; in our experiments, we used 100 sample points and achieved good performance. As before, these vectors can then be hashed jointly or individually for each sample point. Note that the bin width w_a has to be adapted accordingly. In our experiments, we will use the latter option and set $w_a = 1$ for all datasets. The computational details of P2K are given in Algorithm 4, where the additional parts compared to Algorithm 1 are marked in blue (computation) and green (input). An extension to Algorithm 4 would be to refit the GMs after a couple of propagation iterations. We did not consider refitting in our experiments as the number of kernel iterations t_{MAX} was set to 10 or 15 for all datasets.

4.4 Empirical Evaluation

Our intent here is to empirically investigate the power of propagation kernels (PKs) for the task of graph classification stated in Problem 3.1. Specifically, we ask:

(Q4.1) How sensitive are propagation kernels with respect to their parameters and how should propagation kernels be used for graph classification?

(Q4.2) How sensitive are propagation kernels to missing and noisy information?

(Q4.3) Are propagation kernels more flexible than state-of-the-art graph kernels?

(Q4.4) Can propagation kernels be computed faster than state-of-the-art graph kernels while achieving comparable classification performance?

Towards answering these questions, we consider several evaluation scenarios on classical benchmark graph datasets of chemical compounds, as well as on partially labeled semantic image scenes. We compare propagation kernels to several state-of-the-art graph kernels including the Weisfeiler–Lehman subtree kernel (WL) (SHERVASHIDZE, *et al.*, 2011), shortest path kernel (SP) (BORGWARDT and KRIEGEL, 2005), graph hopper kernel (GH) (FERAGEN, *et al.*, 2013), and common subgraph matching kernel (CSM) (KRIEGE and MUTZEL, 2012).

4.4.1 Datasets

The datasets used for evaluating propagation kernels come from a variety of different domains and thus have diverse properties. We distinguish graph databases of labeled and attributed graphs. where attributed graphs usually also have label information on the nodes. Table 4.1 summarizes the properties of all datasets used in the following experiments.

Labeled Graphs. For labeled graphs, we consider the following benchmark datasets from bioinformatics: MUTAG, NCI1, NCI109, and D&D. MUTAG contains 188 sets of mutagenic aromatic and heteroaromatic nitro compounds, and the label refers to their mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium* (DEBNATH, *et al.*, 1991). NCI1 and NCI109 are anti-cancer screens, in particular for cell lung cancer and ovarian cancer cell lines, respectively (WALE and KARYPIS, 2006). D&D consists of 1178 protein structures (DOBSON and DOIG, 2003), where the nodes in each graph represent amino acids and two nodes are connected by an edge if they are less than six ångströms apart. The graph classes are *enzymes* and *non-enzymes*.

Partially Labeled Graphs. The two real-world image datasets MSRC 9-class and MSRC 21-class[2] are state-of-the-art datasets in semantic image processing originally

[2] `http://research.microsoft.com/en-us/projects/objectclassrecognition/`

Table 4.1: **Dataset statistics and properties.**

	properties						
dataset	# graphs	median # nodes	max # nodes	total # nodes	# node labels	# graph labels	attr. dim.
MUTAG	188	17.5	28	3 371	7	2	–
NCI1	4 110	27	111	122 747	37	2	–
NCI109	4 127	26	111	122 494	38	2	–
D&D	1 178	241	5 748	334 925	82	2	–
MSRC9	221	40	55	8 968	10	8	–
MSRC21	563	76	141	43 644	24	20	–
SYNTHETIC	300	100	100	30 000	–	2	1
ENZYMES	600	32	126	19 580	3	6	18
PROTEINS	1 113	26	620	43 471	3	2	1
PRO-FULL	1 113	26	620	43 471	3	2	29
BZR	405	35	57	14 479	10	2	3
COX2	467	41	56	19 252	8	2	3
DHFR	756	42	71	32 075	9	2	3

introduced in (WINN, *et al.*, 2005). Each image is represented by a conditional Markov random field graph, as illustrated in Figure 3.2. The nodes of each graph are derived by oversegmenting the images using the quick shift algorithm,[3] resulting in one graph among the superpixels of each image. Nodes are connected if the superpixels are adjacent, and each node can further be annotated with a semantic label. Imagining an image retrieval system, where users provide images with semantic information, it is realistic to assume that this information is only available for parts of the images as it is relatively easier for a human annotator to label a small number of image regions rather than the full image. As the images in the MSRC datasets are fully annotated, we can derive semantic (ground-truth) node labels by taking the mode ground-truth label of all pixels in the corresponding superpixel. Semantic labels are for example *building, grass, tree, cow, sky, sheep, boat, face, car, bicycle*, and a label *void* to handle objects that do not fall into one of these classes. We removed images consisting of solely one semantic label leading to a classification task among 8 classes for MSRC9 and 20 classes for MSRC21.

Attributed Graphs. To evaluate the ability of PKS to incorporate continuous node attributes, we consider the attributed graphs used in (FERAGEN, *et al.*, 2013; KRIEGE and MUTZEL, 2012). Apart from one synthetic dataset (SYNTHETIC), the graphs are all chemical compounds (ENZYMES, PROTEINS, PRO-FULL, BZR, COX2,

[3] `http://www.vlfeat.org/overview/quickshift.html`

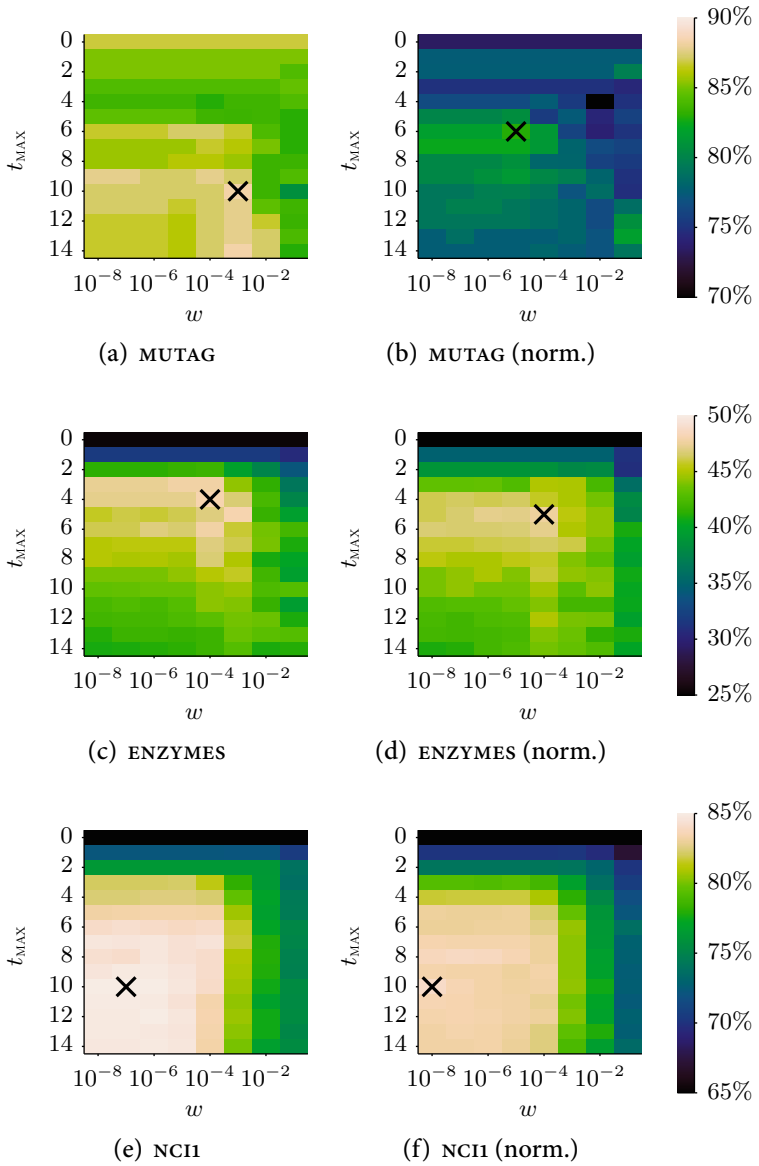
and DHFR). SYNTHETIC comprises 300 graphs with 100 nodes endowed with a one-dimensional normally distributed attribute and 196 edges each. Each graph class, A and B, has 150 examples, where 10 resp. 5 node attributes in A resp. B were flipped randomly. Further, noise drawn from $\mathcal{N}(0, 0.45^2)$ was added to the attributes in B. PROTEINS is a dataset of chemical compounds with two classes (*enzyme* and *non-enzyme*) introduced in (DOBSON and DOIG, 2003). ENZYMES is a dataset of protein tertiary structures belonging to 600 enzymes from the BRENDA database (SCHOMBURG, *et al.*, 2004). The graph classes are their EC (enzyme commission) numbers which are based on the chemical reactions they catalyze. In both datasets, nodes are secondary structure elements (SSE), which are connected whenever they are neighbors either in the amino acid sequence or in 3D space. Node attributes contain physical and chemical measurements including length of the SSE in ångström, its hydrophobicity, its van der Waals volume, its polarity, and its polarizability. For BZR, COX2, and DHFR – originally used in (MAHÉ and VERT, 2009) – we use the 3D coordinates as attributes.

Before we evaluate classification performance and runtimes of the proposed propagation kernels, we analyse their parameter sensitivity.

4.4.2 Parameter Analysis

To analyse parameter sensitivity with respect to the kernel parameters w (LSH bin width) and $t_{\max}$ (number of kernel iterations), we computed average accuracies over 10 randomly generated test sets for all combinations of w and $t_{\max}$, where $w \in \{10^{-8}, 10^{-7}, \ldots, 10^{-1}\}$ and $t_{\max} \in \{0, 1, \ldots, 14\}$ on MUTAG, ENZYMES, and NCI1. The propagation kernel computation is as described in Algorithm 3, that is, we used the label information on the nodes and the label diffusion process as propagation scheme. To assess classification performance, we first learn the SVM cost parameter ($c \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$) on the full dataset for each parameter combination and then performed a 10-fold cross validation (CV). Further, we repeated each of these experiments with the normalized kernel, where normalization means dividing each kernel value by the square root of the product of the respective diagonal entries. Note that for normalized kernels we test for larger SVM cost values ($c \in \{10^{-3}, 10^{-1}, \ldots, 10^{3}\}$). Figure 4.3 shows heatmaps of the results.

In general, we observe that the PK performance is relatively smooth, especially if $w < 10^{-3}$ and $t_{\max} > 4$. Specifically, the number of iterations leading to the best results are in the range from $\{4, 5, \ldots, 10\}$ meaning that we do not have to use a larger number of iterations in the PK computations helping to keep a low computation time. This is especially important for parameter learning. Comparing the heatmaps of the normalized PK to the unnormalized one leads to the conclusion that normalizing the kernel matrix can actually hurt performance. For MUTAG, Figure 4.3(a) and 4.3(b), the performance drops from 88.2% to 82.9%, indicating that for this dataset the size of the graphs, or more specifically the amount of labels



(a) MUTAG (b) MUTAG (norm.)

(c) ENZYMES (d) ENZYMES (norm.)

(e) NCI1 (f) NCI1 (norm.)

Figure 4.3: **Parameter Sensitivity of Propagation Kernels.** The plots show heatmaps of average accuracies (10-fold CV) of PK (labels only) w.r.t. the bin widths parameter w and the number of kernel iterations t_{MAX} for three datasets MUTAG, ENZYMES, and NCI1. In panels (a, c, e) we show the kernel matrix directly; in panels (b, d, f) we show the normalized kernel matrix. The SVM cost parameter is learned for each combination of w and t_{MAX} on the full dataset. × marks the highest accuracy.

from the different kind of node classes are a strong class indicator for the graph label. Nevertheless, incorporating the graph structure, i.e., comparing $t_{\text{MAX}} = 0$ to $t_{\text{MAX}} = 10$, can still improve classification performance by 1.5%. Thus, for the datasets of chemical compounds we will use unnormalized kernels unless stated otherwise.

Recall that our propagation kernel computation is a randomized algorithm, as there is randomization inherent in the choice of hyperplanes used during the LSH computation. We ran a simple experiment to test the sensitivity of the resulting graph kernels with respect to the hyperplane used. We computed the PK with $t_{\text{MAX}} = 10$ between all graphs in the datasets MUTAG, ENZYMES, MSRC9, and MSRC21 100 times, differing only in the random selection of the LSH hyperplanes. To make comparisons easier, we normalized each of these kernel matrices. We then measured the standard deviation of each kernel entry across the repetitions to gain insight into the stability of the PK to changes in the LSH hyperplanes. The median standard deviations were: MUTAG: 5.5×10^{-5}, ENZYMES: 1.1×10^{-3}, MSRC9: 2.2×10^{-4}, and MSRC21: 1.1×10^{-4}. The maximum standard deviations over all pairs of graphs were: MUTAG: 6.7×10^{-3}, ENZYMES: 1.4×10^{-2}, MSRC9: 1.4×10^{-2}, and MSRC21: 1.1×10^{-2}. Clearly the PK values are not overly sensitive to random variation due to differing random LSH hyperplanes.

In summary, we can answer **(Q4.1)** by concluding that PKs are not overly sensitive to the random selection of the hyperplane as well as to the choice of parameters. We propose to learn $t_{\text{MAX}} \in \{0, 1, \ldots, 10\}$ and fix $w \leq 10^{-3}$. Further, we recommend to decide on using the normalized version of PKs only when graph size invariance is deemed important for the classification task. In Part III, we will apply propagation kernels to object category prediction based on point cloud graphs and for this task we will see that normalizing the kernel matrix is essential.

4.4.3 Sensitivity to Missing and Noisy Information

This section analyzes the performance of propagation kernels in the presence of missing and noisy information. Classification performance is evaluated by running C-SVM classifications using `libSVM`.[4] For the sensitivity analysis, the cost parameter c was set to its default value of 1 for all datasets. The number of kernel iterations t_{MAX} was learned on the training splits. Reported accuracies are an average of 10 reruns of a stratified 10-fold cross-validation.

To asses how sensitive propagation kernels are to missing information,we compare the predictive performance of propagation kernels when only some graphs, as for instance graphs at prediction time, have missing labels. Therefore, we divided the graphs of the following datasets, MUTAG, ENZYMES, MSRC9, and MSRC21, into two groups. For one group (fully labeled) we consider all nodes to be labeled, and for

[4] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

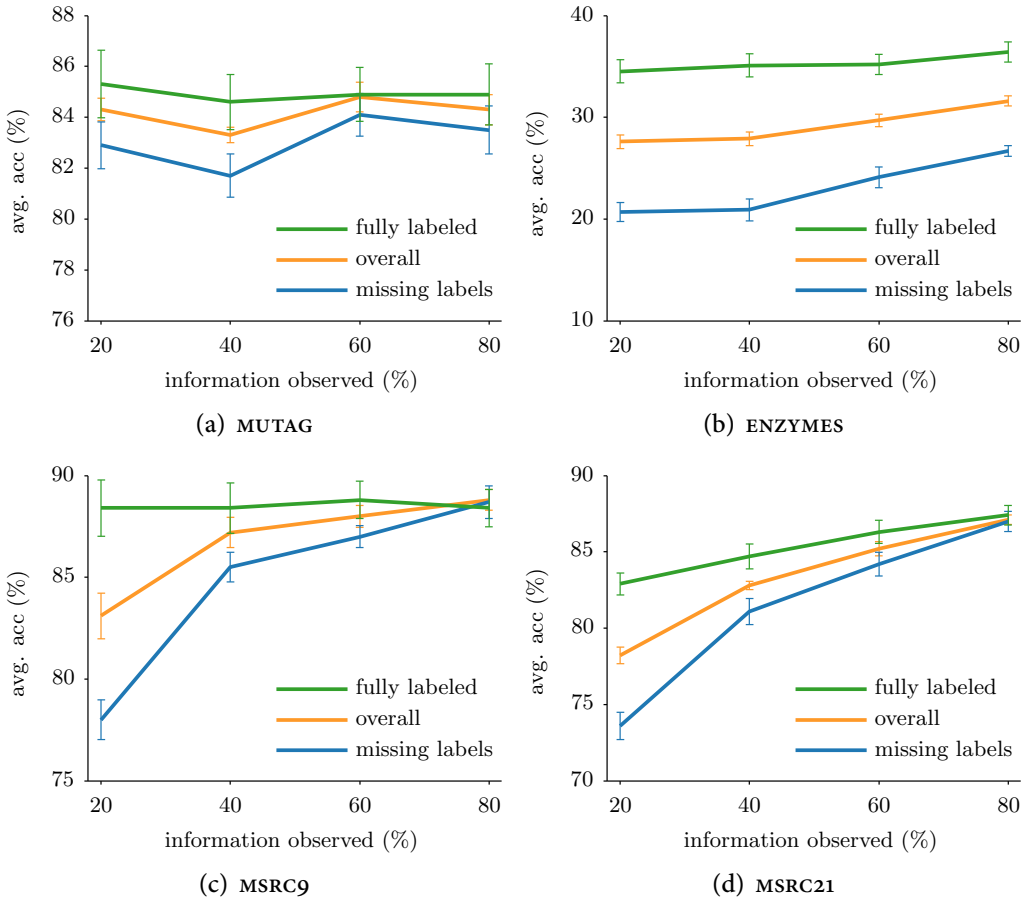


Figure 4.4: **Sensitivity to Missing Node Labels.** We plot observed labels (%) vs. avg. accuracy (%) ± standard error for MUTAG, ENZYMES, MSRC9, and MSRC21. We divided the graphs randomly in two equally sized groups (fully labeled and missing labels). The reported accuracies are averaged over 10 reruns.

the other group (missing labels) we remove x% of the labels at random, where $x \in \{80, 60, \ldots, 20\}$. Figure 4.4 shows average accuracies over 10 reruns for each dataset. Whereas for MUTAG we do not observe a significant difference of the two groups, for ENZYMES the graphs with missing labels could only be predicted with lower accuracy, even when only 20% of the labels were missing. For both MSRC datasets, we observe that we can still predict the graphs with full label information quite accurately; however, the classification accuracy for the graphs with missing information decreases significantly with the amount of missing labels.

The next experiment analyzes the performance of propagation kernels when label information is encoded as attributes in a one-hot encoding. We also examine how

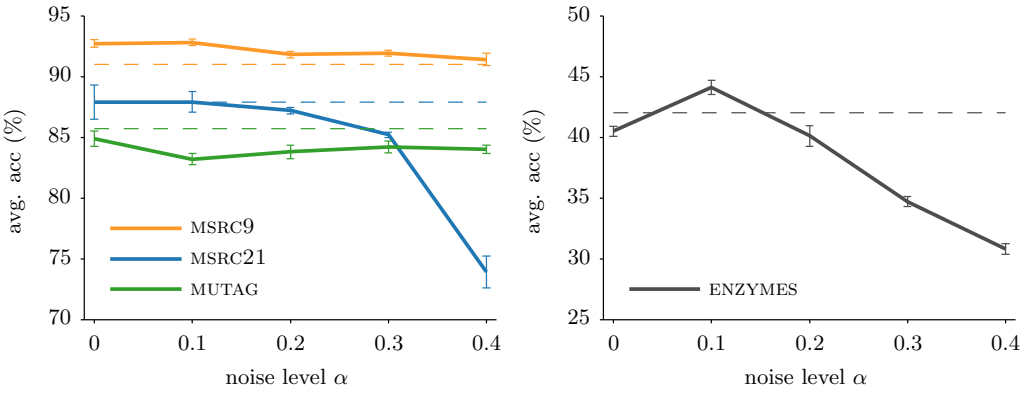


Figure 4.5: **Sensitivity to Noisy Node Labels.** Noise level α is plotted vs. avg. accuracy ± standard error of 10 reruns on MSRC9, MSRC21, MUTAG, and ENZYMES when encoding the labels as k-dimensional attributes using a one-hot representation and attribute propagation as in Algorithm 4. The dashed lines indicate the performance when using the usual label encoding without noise and Algorithm 3.

sensitive they are in the presence of label noise. We corrupted the label encoding by an increasing amount of noise. A noisy label distribution vector $\mathbf{n}_u$ was generated by sampling $n_{u,i} \sim \mathcal{U}(0,1)$ and normalizing so that $\sum n_{u,i} = 1$. Given a noise level α, we used the following values encoded as attributes

$$\mathbf{x}_u \leftarrow (1-\alpha)\,\mathbf{x}_u + \alpha\,\mathbf{n}_u.$$

Figure 4.5 shows average accuracies over 10 reruns for MSRC9, MSRC21, MUTAG, and ENZYMES. First, we see that using the attribute encoding of the label information in a P2K variant only propagating attributes achieves similar performances to propagating the labels directly in PK (dashed lines). This confirms that the Gaussian mixture approximation of the attribute distributions is a reasonable choice. Moreover, we can observe that the performance on MSRC9 and MUTAG is stable across the tested noise levels. For MSRC21 the performance drops for noise levels larger than 0.3. Whereas the same happens for ENZYMES, adding a small amount of noise ($\alpha = 0.1$) actually increases performance. This could be due to a regularization effect caused by the noise.

Finally, we performed an experiment to test the sensitivity of PKS with respect to noise in edge weights. For this experiment, we used the datasets BZR, COX2, and DHFR, and defined edge weights between connected nodes according to the distance between the corresponding structure elements in 3D space. Namely, the edge weight (before row normalization) was taken to be the inverse Euclidean distance between the incident nodes. Given a noise-level σ, we corrupted each

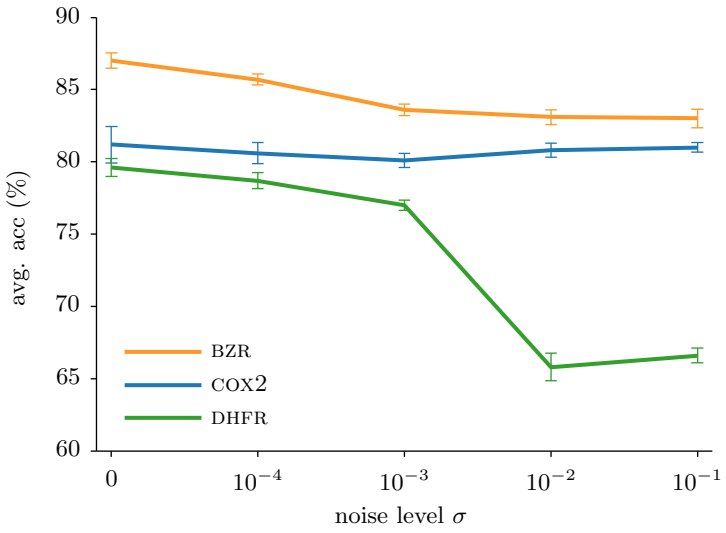


Figure 4.6: **Noisy Edge Weights.** Noise level σ is plotted vs. avg. accuracy ± standard deviation of 10 reruns on BZR, COX2, and DHFR.

edge weight by multiplying by random log-normally distributed noise:

$$w_{ij} \leftarrow \exp(\log(w_{ij}) + \varepsilon),$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Figure 4.6 shows the average test accuracy across 10 repetitions of 10-fold cross-validation for this experiment. The BZR and COX2 datasets tolerated a large amount of edge-weight noise without a large effect on predictive performance, whereas DHFR was somewhat more sensitive to larger noise levels.

Summing up these experimental results we answer **(Q4.2)** by concluding that propagation kernels behave well in the presence of missing and noisy information.

4.4.4 *Experimental Protocol*

We compare classification accuracy and runtime of propagation kernels (PK) with the following state-of-the-art graph kernels: Weisfeiler–Lehman subtree kernel (WL) (SHERVASHIDZE, *et al.*, 2011), shortest path kernel (SP) (BORGWARDT and KRIEGEL, 2005), graph hopper kernel (GH) (FERAGEN, *et al.*, 2013), and common subgraph matching kernel (CSM) (KRIEGE and MUTZEL, 2012). Table 3.1 lists all graph kernels and their intended use. In SP, GH, and CSM we used a Dirac kernel, cf. Equation (2.16), to compare node labels and a Gaussian kernel $k_a(u, v) = \exp(-\gamma\|x_u - x_v\|^2)$ with $\gamma = 1/D$ for attribute information if feasible. CSM for the bigger datasets (ENZYMES, PROTEINS, SYNTHETIC) was computed using a Gaussian truncated for inputs with $\|x_u - x_v\| > 1$. We made this decision to encourage sparsity

in the generated (node) kernel matrices, reducing the size of the induced product graphs and speeding up computation. Note that this is technically not a valid kernel between nodes; nonetheless, the resulting graph kernels were always positive definite. Further, we consider several baselines not taking the graph structure into account. LABELS, corresponding to a PK with $t_{\text{MAX}} = 0$, only compares the label proportions in the graphs, A takes the mean of a Gaussian node kernel among all pairs of nodes in the respective graphs, and A LSH again corresponds to a PK with $t_{\text{MAX}} = 0$ using the attribute information only. We consider graph classification for fully labeled, partially labeled, and attributed graphs.

We implemented propagation kernels in Matlab[5] and classification performance on all datasets is evaluated by running C-SVM classification using libSVM,[6] where the cost parameter is learned via cross-validation on the training set. For PK and WL the number of kernel iterations ($t_{\text{MAX}}, h_{\text{MAX}} \in \{0, 1, \ldots, 10\}$) and for CSM the maximum size of subgraphs ($k \in \{3, 5, 7\}$) is learned on the training splits. Reported accuracies are an average of 10 re-runs of a stratified 10-fold cross-validation. For all runtime experiments all kernels are computed for the largest tested value of t_{MAX}, h_{MAX}, and k. We used a linear base kernel for all kernels involving count features, and attributes if present were standardized. For all PKS the LSH bin width parameters were set to $w_l = 10^{-5}$ resp. $w_a = 1$ and as LSH metrics we chose M_l = TV and M_a = L1 in all experiments. For all WL computations we used the fast implementation[7] introduced in (KERSTING, *et al.*, 2014).

4.4.5 Graph Classification on Benchmark Data

Fully Labeled Graphs. The experimental results for labeled graphs are shown in Table 4.2. On MUTAG the baseline using label information only (LABELS) already gives the best performance indicating that for this dataset the actual graph structure is not adding any predictive information. Note that LABELS corresponds to a PK with $t_{\text{MAX}} = 0$. On NCI1 and NCI109 WL performs best; however, propagation kernels come in second while being computed over one minute faster. Although SP can be computed quickly, it performs significantly worse than PK and WL. The same holds for GH, where for this kernel the computation is also significantly slower. In general, the results on labeled graphs show that propagation kernels can be computed faster than state-of-the-art graph kernels while achieving comparable classification performance, thus question **(Q4.4)** can be answered affirmatively.

Partially Labeled Graphs. To assess the predictive performance of propagation kernels on partially labeled graphs, we ran the following experiments 10 times. We

[5] All Matlab implementations of PK and P2K are publicly available at `https://github.com/marionmari/propagation_kernels`. A Python implementation of PK is included in the pyGPs-toolbox publicly available at `https://github.com/marionmari/pyGPs`.

[6] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[7] `https://github.com/rmgarnett/fast_wl`

Table 4.2: **Results on Labeled Graphs.** Average accuracies ± standard error of 10-fold cross validation (10 runs). Average runtimes in sec (x''), min (x'), or hours (xh) are given in parentheses. The kernel parameters t_{MAX} for PK and h_{MAX} for WL were learned on the training splits ($t_{\text{MAX}}, h_{\text{MAX}} \in \{0, 1, \ldots, 10\}$). LABELS corresponds to PK with $t_{\text{MAX}} = 0$. OUT OF TIME indicates that the kernel computation did not finish within 24h. Bold indicates that the method performs significantly better than the second best method under a paired t-test ($p < 0.05$).

	dataset			
method	MUTAG	NCI1	NCI109	DD
PK	84.5 ± 0.6 (0.2″)	84.5 ± 0.1 (4.5′)	83.5 ± 0.1 (4.4′)	78.8 ± 0.2 (3.6′)
WL	84.0 ± 0.4 (0.2″)	**85.9** ± 0.1 (5.6′)	**85.9** ± 0.1 (7.4′)	79.0 ± 0.2 (6.7′)
SP	85.8 ± 0.2 (0.2″)	74.4 ± 0.1 (21.3″)	73.7 ± 0.0 (19.3″)	OUT OF TIME
GH	85.4 ± 0.5 (1.0′)	73.2 ± 0.1 (13.0h)	72.6 ± 0.1 (22.1h)	68.9 ± 0.2 (69.1h)
LABELS	85.8 ± 0.2 (0.2″)	64.6 ± 0.0 (4.5′)	63.6 ± 0.0 (4.4′)	78.4 ± 0.1 (3.6′)

Table 4.3: **Results on Partially Labeled Graphs.** Average accuracies (and standard errors) on 10 different sets of partially labeled images for PK and WL with unlabeled nodes treated as additional label (WL) and with hard labels derived from converged label propagation (LP + WL). Bold indicates that the method performs significantly better than the second best method under a paired t-test ($p < 0.05$).

		labels missing			
dataset	method	20%	40%	60%	80%
MSRC9	PK	90.0 ± 0.4	88.7 ± 0.3	86.6 ± 0.4	80.4 ± 0.6
	LP + WL	90.0 ± 0.2	87.9 ± 0.6	83.2 ± 0.6	77.9 ± 1.0
	WL	89.2 ± 0.5	88.1 ± 0.5	85.7 ± 0.6	78.5 ± 0.9
MSRC21	PK	**86.9** ± 0.3	**84.7** ± 0.3	**79.5** ± 0.3	**69.3** ± 0.3
	LP + WL	85.8 ± 0.2	81.5 ± 0.3	74.5 ± 0.3	64.0 ± 0.4
	WL	85.4 ± 0.4	81.9 ± 0.4	76.0 ± 0.3	63.7 ± 0.4

randomly removed 20–80% of the node labels in all graphs in MSRC9 and MSRC21 and computed cross-validation accuracies and standard errors. Because the WL-subtree kernel was not designed for partially labeled graphs, we compare PK to two variants: one where we treat unlabeled nodes as an additional label "u" (WL) and another where we use hard labels derived from running label propagation (LP) until convergence (LP + WL). The results are shown in Table 4.3. For larger fractions of missing labels, PK obviously outperforms the baseline methods, and surprisingly running label propagation until convergence and then computing WL gives slightly

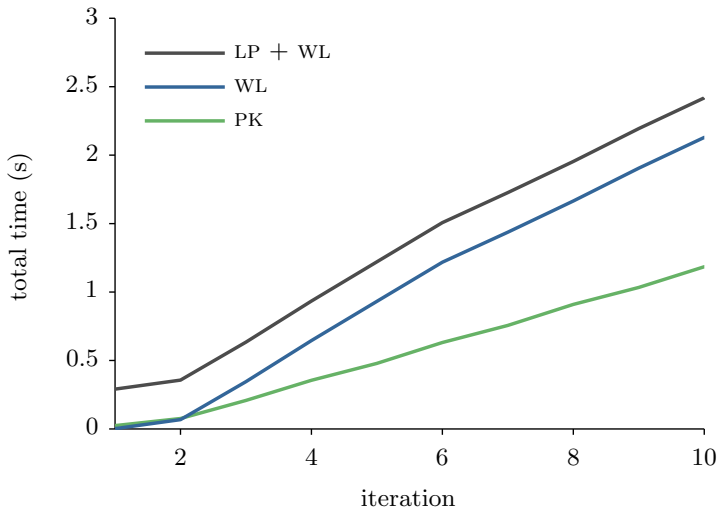


Figure 4.7: **Runtime for Partially Labeled MSRC21.** Average time in seconds over 10 different instances of the MSRC21 dataset with 50% labeled nodes for kernel iterations t_{MAX} from 0 to 10. We compare PK to WL with unlabeled nodes treated as additional label and with hard labels derived from converged label propagation distributions (LP + WL). The WL implementation suggested in (SHERVASHIDZE, *et al.*, 2011) required 36s for $t_{\text{MAX}} = 10$ and is not included in the figure.

worse results than WL. However, label propagation might be beneficial for larger amounts of missing labels. The runtimes of the compared methods on MSRC21 are shown in Figure 4.7. WL computed as suggested in (SHERVASHIDZE, *et al.*, 2011) is over 36 times slower than PK. These results again confirm that propagation kernels have attractive scalability properties for large datasets. The LP + WL approach wastes computation time while running LP to convergence before it can even begin calculating the kernel. The intermediate label distributions obtained during the convergence process are already extremely powerful for classification. These results clearly show that propagation kernels can successfully deal with partially labeled graphs and thus questions **(Q4.3)** and **(Q4.4)** can be answered affirmatively.

Attributed Graphs. For PK not propagating the attributes we standardized the attributes and computed one hash per attribute dimension. The experimental results for various datasets with attributed graphs are illustrated in Figure 4.8. The plots show runtime versus average accuracy, where the error bars reflect standard deviation of the accuracies. As we are interested in good predictive performance while achieving fast kernel computation, methods in the upper left corners provide the best performance with respect to both quality and speed. For PK, SP, GH, and CSM we compare three variants. One where we use the labels only,

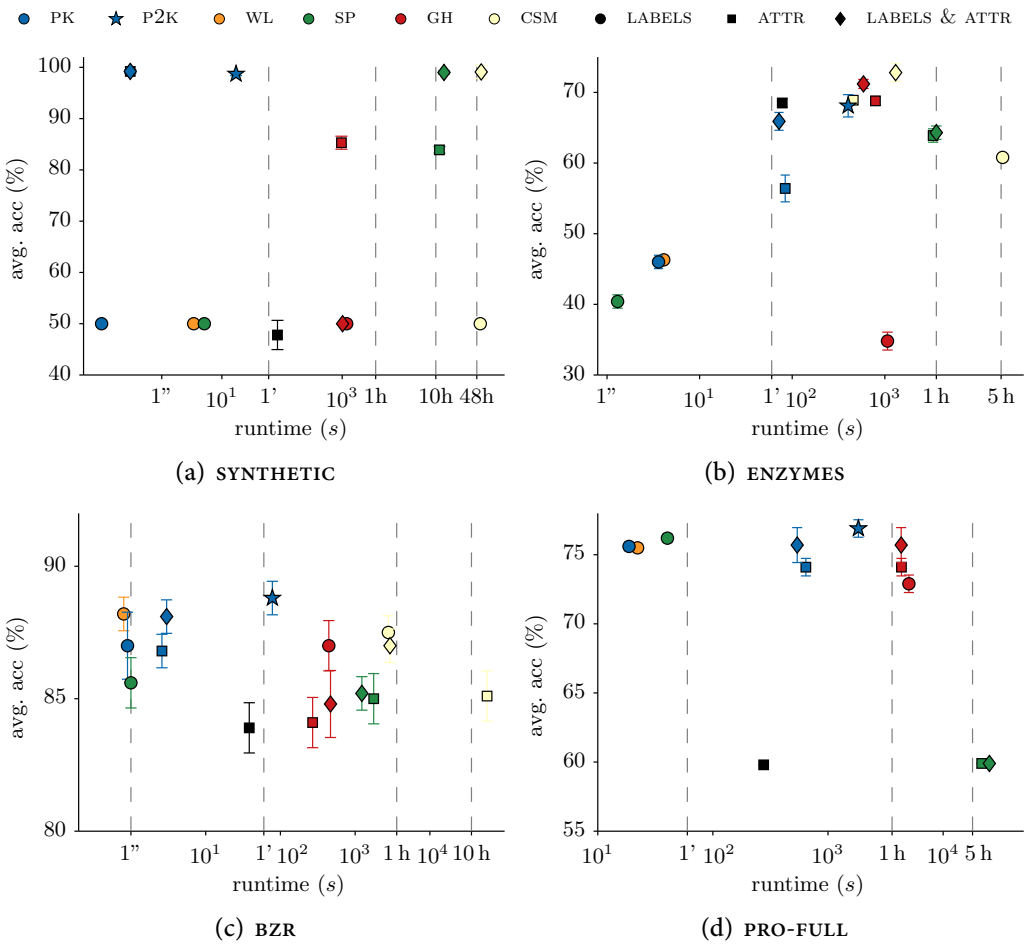


Figure 4.8: **Log Runtime vs. Accuracy on Attributed Graphs.** The plots show log(runtimes) in seconds plotted against average accuracy (± standard deviation). Methods are encoded by color and the used information (labels, attributes or both) is encoded by shape. Note that P2K also uses both, labels and attributes, however in contrast to PK both are propagated. For PRO-FULL the CSM kernel was OUT OF MEMORY meaning that 32 GB were exceeded. On SYNTHETIC CSM using the attribute information only could not be computed within 72h.

one where we use the attribute information only and one where both labels and attributes are used. WL is computed with label information only. For SYNTHETIC, cf. Figure 4.8(a), we used the node degree as label information. Further, we compare the performance of P2K, which propagates labels and attributes as described in Section 4.3.3. Detailed results on SYNTHETIC and all bioinformatics datasets are provided in Table 4.4 (runtimes) and Table 4.5 (average accuracies). From Figure 4.8 we clearly see that propagation kernels tend to appear in the upper left

Table 4.4: **Runtimes on Attributed Graphs.** All runtimes are times for kernel computation given in sec (x''), min (x'), or hours (xh). For all PKs $t_{\text{MAX}} = 10$, for WL $h_{\text{MAX}} = 10$ and for CSM $k = 7$. All computations are performed on machines with Intel cores i7 with 3.4 GHz. Note that CSM is implemented in Java, so comparing computation times is only possible to a limited extent. OUT OF TIME means that the computation did not finish within 72h.

		dataset						
method		SYNTHETIC	ENZYMES	PROTEINS	PRO_FULL	BZR	COX2	DHFR
LABELS	PK	**0.1″**	**3.6″**	**17.8″**	**18.7″**	0.9″	1.1″	3.4″
	WL	3.4″	4.1″	25.9″	22.2″	**0.8″**	1.0″	4.2″
	SP	5.1″	1.3″	38.7″	40.3″	1.0″	1.2″	**2.3″**
	GH	19.8′	17.8′	2.2h	1.4h	7.4′	10.8′	29.5′
	CSM	54.8h	5.2h	–	–	46.4′	1.6h	3.7h
ATTR	A	1.4′	1.3′	6.4′	4.6′	38.3″	1.1′	3.2′
	PK	**0.3″**	**1.4′**	**23.6″**	**10.7′**	**2.6″**	**2.8″**	**14.8″**
	SP	11.5h	55.4′	5.9h	6.0h	29.5′	25.9′	1.3h
	GH	16.2′	13.2′	1.9h	72.4′	4.5′	6.6′	15.8′
	CSM	OUT OF TIME	7.6′	–	–	16.2h	31.5h	97.5h
LABELS ATTR	PK	**0.3″**	**1.2′**	**20.0″**	**9.0′**	**3.0″**	**3.5″**	**15.1″**
	P2K	17.2″	6.7′	31.4′	30.6′	1.3′	1.8′	5.9′
	SP	13.7h	1.0h	6.8h	7.0h	20.5′	32.9′	1.6h
	GH	16.9′	9.8′	1.2h	1.2h	7.8′	11.7′	31.2′
	CSM	56.8h	21.8′	–	–	48.8′	1.6h	3.7h

corner, that is, they are achieving good predictive performance while being fast, leading to a positive answer of question **(Q4.4)**. Note that the runtimes are on a log scale. We can also see that P2K propagating both labels and attributes (blue star) usually outperforms the simple PK implementation not considering attribute arrangements (blue diamond). However, this comes at the cost of being slower. So, we can use the flexibility of propagation kernels to trade predictive quality against speed or vice versa according to the requirements of the application at hand. This supports a positive answer to question **(Q4.3)**.

Table 4.5: **Accuracies on Attributed Graphs.** Average accuracies ± standard error of 10-fold cross validation (10 runs). The kernel parameters t_{MAX} for all PKs and h_{MAX} for WL were learned on the training splits ($t_{\text{MAX}}, h_{\text{MAX}} \in \{0, 1, \ldots, 10\}$). Whenever the normalized version of a kernel performed better than the unnormalized version we report these results and mark the method with $*$. CSM is implemented in Java and computations were performed on a machine with 32 GB of memory. OUT OF MEMORY indicates a Java *outOfMemeoryError*. Bold indicates that the method performs significantly better than the second best method under a paired t-test ($p < 0.05$). The SVM cost parameter is learned on the training splits. We choose $c \in \{10^{-7}, 10^{-5}, \ldots, 10^{5}, 10^{7}\}$ for normalized kernels and $c \in \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}\}$ for unnormalized kernels.

		dataset						
method		SYNTHETIC	ENZYMES	PROTEINS	PRO-FULL	BZR	COX2	DHFR
LABELS	PK	50.0 ± 0.0*	46.0 ± 0.3	75.6 ± 0.1*	75.6 ± 0.1*	87.0 ± 0.4	81.0 ± 0.2	83.5 ± 0.2*
	WL	50.0 ± 0.0*	46.3 ± 0.2*	75.5 ± 0.1*	75.5 ± 0.1*	88.2 ± 0.2	**83.2** ± 0.2*	84.1 ± 0.2
	SP	50.0 ± 0.0*	40.4 ± 0.3*	76.2 ± 0.1*	76.2 ± 0.1*	85.6 ± 0.3	81.0 ± 0.4*	82.0 ± 0.3*
	GH	50.0 ± 0.0*	34.8 ± 0.4*	72.9 ± 0.2*	72.9 ± 0.2*	87.0 ± 0.3	81.4 ± 0.3*	79.5 ± 0.4
	CSM	50.0 ± 0.0*	60.8 ± 0.2	OUT OF MEMORY	OUT OF MEMORY	87.5 ± 0.2*	80.7 ± 0.3	82.6 ± 0.2*
ATTR	PK	99.2 ± 0.1*	56.4 ± 0.6	72.7 ± 0.2	74.1 ± 0.2*	86.8 ± 0.2*	78.2 ± 0.4	84.3 ± 0.1
	SP	83.9 ± 0.2*	63.9 ± 0.3*	74.3 ± 0.1*	59.9 ± 0.0*	85.0 ± 0.3*	78.2 ± 0.0	78.9 ± 0.3*
	GH	85.3 ± 0.4	68.8 ± 0.2*	72.6 ± 0.1*	61.2 ± 0.0*	84.1 ± 0.3*	79.5 ± 0.2*	79.0 ± 0.2
	CSM	–	68.9 ± 0.2*	OUT OF MEMORY	OUT OF MEMORY	85.1 ± 0.3*	77.6 ± 0.3	79.5 ± 0.2
LABELS ATTR	PK	99.2 ± 0.1*	65.9 ± 0.4*	76.3 ± 0.2*	75.7 ± 0.4*	88.1 ± 0.2*	79.4 ± 0.6	84.1 ± 0.3*
	P2K	98.7 ± 0.1	68.1 ± 0.5	75.9 ± 0.2*	**76.9** ± 0.2*	88.8 ± 0.2	80.9 ± 0.4	83.5 ± 0.3*
	SP	99.0 ± 0.1	64.3 ± 0.3*	73.2 ± 0.2*	59.9 ± 0.0*	85.2 ± 0.2*	78.5 ± 0.1	79.7 ± 0.2*
	GH	50.0 ± 0.0*	71.2 ± 0.2*	73.0 ± 0.1*	60.9 ± 0.0*	84.8 ± 0.4	79.5 ± 0.2*	80.0 ± 0.2
	CSM	99.0 ± 0.1	**72.8** ± 0.4*	OUT OF MEMORY	OUT OF MEMORY	87.0 ± 0.2*	79.2 ± 0.4*	80.1 ± 0.3*
ATTR	A	47.8 ± 0.9*	68.5 ± 0.2*	72.8 ± 0.2*	59.8 ± 0.0*	83.9 ± 0.3*	78.2 ± 0.0	74.8 ± 0.2*

CHAPTER 5

PROPAGATION KERNELS FOR GRID GRAPHS

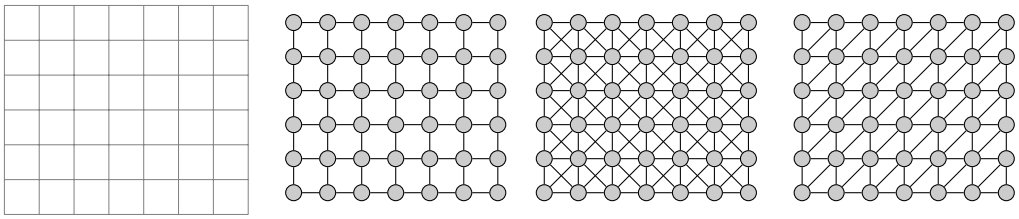
The goal of this chapter is to compute propagation kernels for pixel grid graphs. A graph kernel among grid graphs can be defined such that two grids should have a high kernel value if they have similarly arranged node information. This can be naturally captured by propagation kernels as they can monitor information spread on the grids. Naïvely, one could think that we can simply apply Algorithm 3 to achieve this goal. However, given that the space complexity of this algorithm scales with the number of edges and even medium sized images such as texture patches will easily contain thousands of nodes, this is not feasible. For example considering 100×100-pixel image patches with an 8-neighborhood graph structure, the space complexity required would be 2.4 million units[1] (floating point numbers) *per graph*. Fortunately, we can exploit the flexibility of propagation kernels by exchanging the propagation scheme. Rather than label diffusion as used earlier, we employ discrete convolution; this idea was introduced for efficient clustering on discrete lattices in (BAUCKHAGE and KERSTING, 2013). In fact, isotropic diffusion for denoising or sharpening is a highly developed technique in image processing (JÄHNE, 2005). In each iteration, the diffused image is derived as the convolution of the previous image and an isotropic (linear and space-invariant) filter. In the following, we derive a space- and time-efficient way of computing propagation kernels for grid graphs by means of convolutions.

5.1 Grid Graphs and Discrete Convolution

Given that the neighborhood of a node is the subgraph induced by all its adjacent vertices, we define a *grid graph* as follows.

DEFINITION 5.1 (GRID GRAPH) *A d-dimensional grid graph is a lattice graph whose node embedding in $\mathbb{R}^d$ forms a regular square tilling and the neighborhoods $\mathcal{N}$ of each non-border node are the same, i.e., they are isomorphic.*

[1] Using a coordinate list sparse representation, the memory usage per pixel grid graph for Algorithm 3 is $\mathcal{O}(3m_1m_2p)$, where $m_1 \times m_2$ are the grid dimensions and p is the size of the pixel neighborhood.



(a) Square tilling (b) 4-neighborhood (c) 8-neighborhood (d) non-symmetric neighborhood

Figure 5.1: **Grid Graph.** Regular square tilling (a) and three example neighborhoods (b-d) for a 2-dimensional grid graph derived from line graphs L_7 and L_6.

Figure 5.1 illustrates a regular square tilling and several isomorphic neighborhoods of a 2-dimensional grid graph. If we ignore boundary nodes a grid graph is a regular graph, i.e., each non-border node has the same degree. Note that the size of the border depends on the radius of the neighborhood. In order to be able to neglect the special treatment for border nodes it is common to view the actual grid graph as a finite section of an actually infinite graph.

A grid graph whose node embedding in $\mathbb{R}^d$ forms a regular square tilling can be derived from the graph Cartesian product of line graphs. So, a 2-dimensional grid is defined as

$$G^{(i)} = L_{m_{i,1}} \times L_{m_{i,2}},$$

where $L_{m_{i,1}}$ is a line graph with $m_{i,1}$ nodes. $G^{(i)}$ consists of $n_i = m_{i,1}\, m_{i,2}$ nodes, where non-border nodes have the same number of neighbors. Note that the grid graph $G^{(i)}$ only specifies the node layout in the graph but not the edge structure. The edges are given by the neighborhood $\mathcal{N}$ which can be defined by any arbitrary matrix B encoding the weighted adjacency of its center node. The nodes being for instance image pixels can carry discrete or continuous vector-valued information. So, in the most general setting the database of gird graphs is given by $\mathbf{G} = \{G^{(i)}\}_{i=1,\ldots,n}$ with $G^{(i)} = (V^{(i)}, \mathcal{N}, \ell)$, where $\ell : V^{(i)} \to \mathcal{L}$ with $\mathcal{L} = ([k], \mathbb{R}^D)$.

Commonly used neighborhoods $\mathcal{N}$ are the 4-neighborhood and the 8-neighborhood illustrated in Figure 5.1(b) and (c); the resulting graphs are called *simple grid graph* for the 4-neighborhood and *king's graph* for the 8-neighborhood as it's edges show the valid moves of the king on a chessboard. In the terminology of cellular automata the 4-neighborhood is called the *Von Neumann neighborhood* and the 8-neighborhood is termed *Moore neighborhood*.

The general *convolution* operation on two functions f and g is defined as

$$f(x) * g(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(\tau)\, g(x - \tau) d\tau.$$

That is, the convolution operation produces a modified, also called *filtered*, version of the original function f. The function g is called a filter. For 2-dimensional grid graphs interpreted as discrete functions of two variables x and y, e.g., the vertex location in the grid, we consider the *discrete spatial convolution* defined as

$$f(x,y) * g(x,y) = (f * g)(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)\, g(x-i, y-j),$$

where the computation is in fact done for finite intervals. As convolution is a well studied operation in low-level signal processing and discrete convolution is a standard operation in digital image processing we can resort to highly developed algorithms for its computation; see for example Chapter 2 in (JÄHNE, 2005). Convolutions can be computed efficiently via the fast Fourier transformation in $\mathcal{O}(n_i \log n_i)$ per graph.

5.2 Efficient Propagation Kernel Computation

Let $\mathbf{G} = \{G^{(i)}\}_i$ be a database of grid graphs. To simplify notation, however without loss of generality, we assume 2-dimensional grids $G^{(i)} = L_{m_{i,1}} \times L_{m_{i,2}}$ and discrete label information on the nodes. Unlike in the case of general graphs, each graph now has a natural 2-dimensional structure, so we will update our notation to reflect this structure. Instead of representing the label probability distributions of each node as rows in a 2-dimensional matrix, we now represent them in the third dimension of a 3-dimensional tensor $P_t^{(i)} \in \mathbb{R}^{m_{i,1} \times m_{i,2} \times k}$. Modifying the structure makes both the exposition more clear and also enables efficient computation. Now, we can simply consider discrete convolution on k matrices of label probabilities $P^{(i,j)}$ per grid graph $G^{(i)}$, where $P^{(i,j)} \in \mathbb{R}^{m_{i,1} \times m_{i,2}}$ contains the probabilities of being label j and $j \in \{1, \ldots, k\}$. For observed labels $P_0^{(i)}$ is again initialized with a Kronecker delta distribution across the third dimension and, in each propagation step, we perform a discrete convolution of each matrix $P^{(i,j)}$ per graph. Thus, we can realize various propagation schemes efficiently by applying appropriate filters, which are represented by matrices B in our discrete case. We use circular symmetric neighbor sets $\mathcal{N}_{r,p}$ as introduced in (OJALA, *et al.*, 2002), where each pixel has p neighbors which are equally spaced pixels on a circle of radius r. We use the following approximated filter matrices in our experiments:

$$\mathcal{N}_{1,4} = \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix}, \; \mathcal{N}_{1,8} = \begin{bmatrix} 0.06 & 0.18 & 0.06 \\ 0.18 & 0.05 & 0.18 \\ 0.06 & 0.18 & 0.06 \end{bmatrix}, \text{ and}$$

$$\mathcal{N}_{2,16} = \begin{bmatrix} 0.01 & 0.06 & 0.09 & 0.06 & 0.01 \\ 0.06 & 0.04 & 0 & 0.04 & 0.06 \\ 0.09 & 0 & 0 & 0 & 0.09 \\ 0.06 & 0.04 & 0 & 0.04 & 0.06 \\ 0.01 & 0.06 & 0.09 & 0.06 & 0.01 \end{bmatrix}. \tag{5.1}$$

Algorithm 5 Propagation Kernel for Grid Graphs

given: graph database $\mathbf{G} = (V, E, \ell)$, # iterations $t_{\max}$, filter matrix B, bin width w, metric M, base kernel $\langle \cdot, \cdot \rangle$
initialization: $K \leftarrow 0$, $P_0^{(i)} \leftarrow \delta_{\ell(V_i)} \; \forall i$
for $t \leftarrow 0 \dots t_{\max}$ **do**
  CALCULATE-LSH($\{P_t^{(i)}\}_i, w$, M) ▷ bin node information
  for all graphs $G^{(i)}$ **do**
    compute $\Phi_{i\cdot} = \phi(G_t^{(i)})$ ▷ count bin strengths
  end for
  $K \leftarrow K + \langle \Phi, \Phi \rangle$ ▷ compute and add kernel contribution
  for all graphs $G^{(i)}$ and labels j **do**
    $P_{t+1}^{(i,j)} \leftarrow P_t^{(i,j)} * B$ ▷ discrete convolution
  end for
  end for

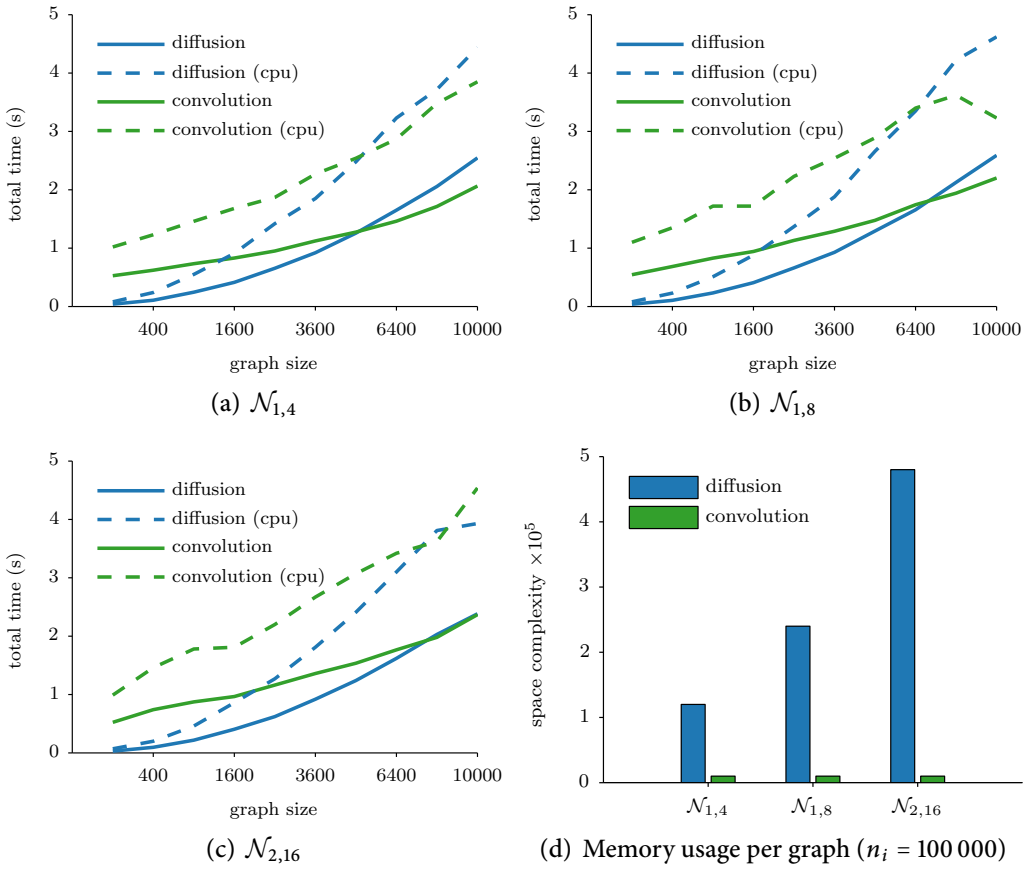
The propagation kernel computation for grid graphs is summarized in Algorithm 5, where the specific parts compared to the general propagation kernel computation (Algorithm 1) are highlighted in green (input) and blue (computation). Using fast Fourier transformation, the time complexity of Algorithm 5 is $\mathcal{O}((t_{\max} - 1) N \log N + t_{\max} n^2 n^\star)$, where N is the total number of nodes, n is the number of graphs, and $n^\star$ is the number of nodes of the largest graph in $\mathbf{G}$. Note that for the purpose of efficient computation, CALCULATE-LSH has to be adapted to take the label distributions $\{P_t^{(i)}\}_i$ as a set of 3-dimensional tensors. By virtue of the invariance of the convolutions used, propagation kernels are translation invariant, and when using the circular symmetric neighbor sets they are also 90-degree rotation invariant. These properties make them attractive for image-based texture classification. The use of other filters implementing for instance anisotropic diffusion depending on the local node information is a straightforward extension.

5.3 Empirical Evaluation: Image-based Texture Classification

Now, we want to empirically investigate whether propagation kernels can be computed on huge grid graphs which clearly makes – if answered affirmatively – propagation kernels more flexible and powerful than existing graph kernels. Specifically, we pose two questions:

(Q5.1) Are propagation kernels feasible for huge grid graphs and thus more flexible than state-of-the-art graph kernels?

(Q5.2) Can propagation kernels achieve comparable classification performance on texture classification compared to computer vision approaches?



(a) $\mathcal{N}_{1,4}$ (b) $\mathcal{N}_{1,8}$

(c) $\mathcal{N}_{2,16}$ (d) Memory usage per graph (n_i = 100 000)

Figure 5.2: **Time and Space Complexity Analysis.** Comparison of two different implementations of information propagation (diffusion and convolution) in the propagation kernel computation for a synthetic dataset of 100 grid graphs of varying sizes. Panels (a-b) compare runtime complexities (cputime and actual elapsed time) of the kernel computation for different neighborhoods $\mathcal{N}_{r,p}$ and panel (d) illustrates the memory usage for one grid graph with 100 000 nodes.

5.3.1 *Complexity Analysis*

To analyse the space and time complexity of propagation kernels empirically we created a dataset of 100 2-dimensional grid graphs of varying sizes with randomly assigned binary node labels. We compare the computation of propagation kernels as stated in Algorithm 3 (diffusion) to the one derived for grid graphs, cf. Algorithm 5 (convolution); both implemented in Matlab.[2] With respect to computation times we compare cputime and actual elapsed time of both algorithms performed

[2]All implementations including the PK grid version using convolutions for information propagation are publicly available at `https://github.com/marionmari/propagation_kernels`.

on a machine with an 1.7 GHz Intel Core i5 processor. Cputime is usually higher as the actual elapsed runtime as it adds up the runtimes of parallel computations achieved by hyperthreading. Figure 5.2(a-c) shows both runtimes for diffusion and convolution for different neighborhood structures for the 100 graphs of sizes $n_i \in \{10^2, 20^2, \cdots, 100^2\}$. First, we observe that for larger graphs convolution, i.e., Algorithm 5, performs faster than the standard propagation kernel implementation using label diffusion; for 100 000 nodes per graphs and $\mathcal{N}_{1,4}$, for example, the elapsed time for diffusion is 2.5 seconds whereas convolution only takes 2.0 seconds. So, for larger grid graphs ($n_i \geq 5\,000$) it is beneficial to use convolutions both measuring cputime and actual elapsed time. Thinking of pixel images, graphs with 100×100 pixels are actually still rather small, so considering such kind of data will most likely result in huge grid graphs, i.e. $n_i \gg 5\,000$. Further, we see form all plots that both algorithms can exploit parallel computation allowed by hyperthreading as the actual runtimes are roughly half as long as cputime.

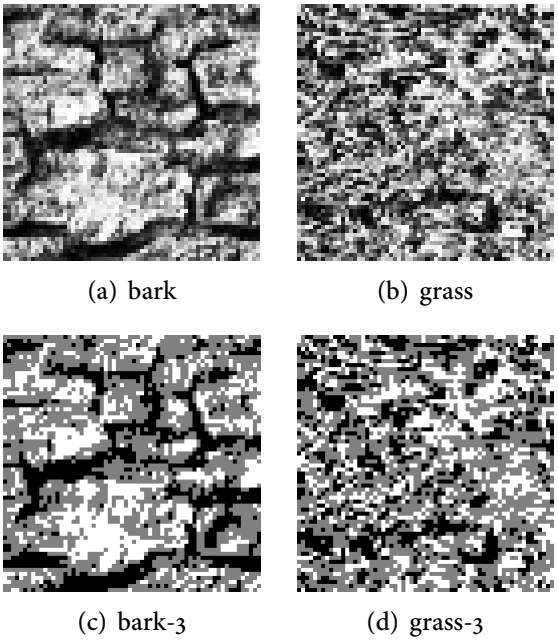
The original computation scheme (diffusion) relies on the transition probabilities for all nodes in all graphs which is stored in a sparse block diagonal transition matrix $T \in \mathbb{R}^{N \times N}$. The graphs for Algorithm 5 (convolution) can be stored as grids, and hence, one matrix $I_i \in \mathbb{R}^{m_{i,1} \times m_{i,2}}$ per graph and the neighborhood structure $\mathcal{N}_{r,p}$ are hold in memory. Using a coordinate list sparse representation memory usage per graph for Algorithm 3 is $\mathcal{O}(3m_{i,1}m_{i,2}p)$, whereas for Algorithm 5 it is $\mathcal{O}(m_{i,1}m_{i,2}+p)$. The memory usage for one graph with 100 000 nodes is illustrated in Figure 5.2(d). From this analysis we can conclude that propagation kernels, especially the implementation realizing information propagation via discrete convolution, are feasible for huge grid graphs both with respect to time and space complexity. Thus, question **(Q5.1)** can be answered positively.

5.3.2 *Texture Classification*

We now consider the BRODATZ dataset, a classical benchmark dataset of pixel intensity images for texture classification.

Dataset. BRODATZ,[3] introduced in (VALKEALAHTI and OJA, 1998), covers 32 textures from the Brodatz album with 64 2-dimensional images per class comprising the following subsets of images: 16 "original" images (O), 16 rotated versions (R), 16 scaled versions (S), and 16 rotated and scaled versions (RS) of the "original" images. Each image is represented by a weighted grid graph using circular symmetric neighborhoods as defined in Equation (5.1). The node labels are computed from the intensity values by quantization. Table 5.1 summarizes the data statistics for two datasets: one using only the original images and their rotated versions (BRODATZ-O-R) and the full dataset (BRODATZ-O-R-S-RS). Figure 5.3 shows two example images with their corresponding quantized versions using three labels.

[3] `http://www.ee.oulu.fi/research/imag/texture/image_data/Brodatz32.html`



(a) bark (b) grass

(c) bark-3 (d) grass-3

Figure 5.3: **Brodatz Dataset.** Example images from BRODATZ (a,b) and the corresponding quantized versions with 3 colors (c,d) of the two texture classes bark and grass.

Table 5.1: **Dataset statistics and properties.** Note that for both datasets median and maximum number of nodes are the same.

	properties				
dataset	# graphs	median # nodes	total # nodes	# node labels	# graph labels
BRODATZ-O-R	1 024	4 096	4 194 304	3	32
BRODATZ-O-R-S-RS	2 048	4 096	8 388 608	3	32

For parameter learning we used a random subset of 20% of the original images and their rotated versions.

Experimental Protocol. We implement the following experimental protocol. The training data comprises a subset of randomly selected 20% of the BRODATZ-O-R data to account for the large number of classes (32 textures). On this training subset we performed a grid search over the PK parameter t_{MAX}, the quantization values *col*, and the neighborhoods B. We used the following parameter ranges: $t_{\text{MAX}} \in \{0, 3, 5, 8, 10, 15, 20\}$, $col \in \{3, 5, 8, 10, 15\}$, and $B \in \{\mathcal{N}_{1,4}, \mathcal{N}_{1,8}, \mathcal{N}_{2,16}\}$, cf.

Table 5.2: **Results on Grid Graphs.** Average accuracies ± standard errors of 10-fold CV (10 runs). The PK parameter t_{MAX} as well as color quantization and pixel neighborhood was learned on a training subset of the full dataset. Average cputimes in sec (x'') or min (x') given in parentheses refer to the learned parameter settings. For GLCM-QUANT and LABELS the same color quantization as for PK was applied. LABELS corresponds to PK with $t_{MAX} = 0$.

	dataset	
method	BRODATZ-O-R	BRODATZ-O-R-S-RS
PK	**89.6** ± 0.0 (3.5′)	**85.7** ± 0.0 (7.1′)
LABELS	5.0 ± 0.0 (1.1′)	4.9 ± 0.0 (2.2′)
GLCM-GRAY	87.2 ± 0.0 (29.5″)	79.4 ± 0.0 (44.8″)
GLCM-QUANT	78.6 ± 0.0 (24.9″)	68.6 ± 0.0 (44.8″)

Equation (5.1). The best performance on the training data was achieved with 3 colors and an 8-neighborhood. For testing we use test suites similar to the ones provided with the dataset.[4] All train/test splits are created such that whenever an original image (O) occurs in one split, their modified versions (R,S,RS) are also included in the same split. We compare PK to the simple baseline LABELS using label counts only and to a commonly-used second-order statistical feature based on the gray-level co-occurrence matrix comparing intensities (GLCM-GRAY) resp. quantized labels (GLCM-QUANT) of neighboring pixels originally introduced in (HARALICK, *et al.*, 1973).

Results. The experimental results are shown in Table 5.2. While not outperforming sophisticated, highly tuned, and thus complex state-of-the-art computer vision approaches to texture classification (TOU, *et al.*, 2009; OJALA, *et al.*, 2002), we find that it is feasible to compute PKS on the huge image datasets achieving respectable performance basically *out-of-the-box*. This is – compared to the immense tuning of features and methods commonly done in computer vision – a great success. PK achieves an average accuracy of almost 90% on this 32-class classification problem. Comparing this result with the baseline LABELS using no graph structure, we observe that texture classification heavily relies on the arrangement of intensity values and that propagation kernels are indeed capable to capture this by measuring label structure, cf. Definition 2.10, In conclusion, we can answer question **(Q5.2)** affirmatively based on the observation that propagation kernels are an extremely promising approach intersecting machine learning, graph mining, and computer vision.

[4]The test suites provided with the data are incorrect. We use a corrected version in all our experiments.

TRANSITION: FROM THE GRAPH TO THE NODE LEVEL

To summarize, random walk-based models provide a principled way of spreading information and even handling missing and uncertain information within graphs. Known labels are, for example, propagated through the graph in order to label all unlabeled nodes. In this first part of the thesis, we showed how to use random walks to discover structural similarities for the construction of a graph kernel, namely the propagation kernel.

As our experimental results demonstrate, propagation kernels are competitive in terms of accuracy with state-of-the-art kernels on several classification benchmark datasets of labeled and attributed graphs. In terms of runtime, propagation kernels outperform all recently developed efficient and scalable graph kernels. Moreover, being tied to the propagation scheme, propagation kernels can be easily adapted to novel applications, such as the classification of partially labeled graphs and huge grid graphs, which have not been tractable for graph kernels before. Summarizing all experimental results performed on general and grid graphs, the capabilities claimed in Table 3.1 are supported. Propagation kernels have proven extremely flexible and efficient, and thus overcome the limitations of existing graph kernels, which are either flexible but lack efficient computations, or they are fast but specialized to limited application settings.

While we have used classification to guide the development of propagation kernels, the results are directly applicable to regression, clustering, and ranking, among other tasks. Employing message-based probabilistic inference schemes such as (loopy) belief propagation (YEDIDIA, *et al.*, 2003) directly paves the way to dealing with graphical models.

The goal of propagation kernels is to uncover label-structure similarity, cf. Definition 2.10, in order to compare structured data. Intuitively, propagation kernels count common sub-distributions induced in each iteration of running inference in two graphs leading to the insight that graph kernels are much closer to graph-based learning on the node level than assumed before. This naturally leads to the question whether label-structure similarity can also be leveraged for the comparison of nodes in a graph and thus can be used for learning tasks on the node level such as node label prediction. Except for rare recently proposed methods, approaches to node classification, clustering, and ranking heavily rely on the assumption that nodes that are close to one another in the graph are likely to have

the same properties, and thus should have the same label, be in the same cluster, or should have similar ranking values. This homophily assumption was stated in Hypothesis 2.1. Following the key insight of propagation kernels, that two graphs are similar if the probability distributions on their node information is evolving similarly, we assume that two nodes in a graph should also have similar labels if their label distributions behave similarly. Going beyond the classical homophily assumption, this insight leads to the claim that local structure similarity can also be an indicator for the class label of a vertex.

In the second part of this thesis, we will therefore investigate the use of information propagation schemes to measure local structure similarity of graph vertices in networked data. Refining and transferring the insights and techniques developed for learning on the graph level, we will design a kernel among the nodes of a graph. The *coinciding walk kernel* will be defined in terms of the probability that the labels encountered during parallel random walks coincide. Being a kernel on a graph, the coinciding walk kernel will then be used for learning on the node level.

Part II

Coinciding Walk Kernels

In this part, we are concerned with learning on the node level in networked data. The entities of interest are represented as nodes in a graph. Example domains with problems of great practical importance can easily be found in web sciences, for instance the ranking of webpages in the web graph, the retrieval of similar items for recommendation systems, the placement of advertisements on social network platforms, or the completion of knowledge in the wikipedia graph created from interlinked wiki pages. We will assume that for the problem at hand the graph representation of the data is given as defined in Section 2.1.2. That is, we have a set of nodes and edges represented as an adjacency matrix and, if available, some additional information on the nodes and edges is provided in form of a label vector, and attribute and weight matrices. Learning on the node level applies techniques from statistical machine learning in order to cluster the nodes, retrieve interesting nodes, or predict unknown values on the nodes in a graph. In the light of the applications mentioned above, this is an important and challenging task.

The challenges, however, do not only arise from the data representation as sets of nodes and edges and the interpretation of the link structure or the meaning of the versatile relations encoded. Networked data is usually huge, and the available information for training is often very sparse and can be uncertain. In the following, we will study the problem of predicting unknown properties of the nodes in a graph in the presence of very few nodes with observed target values. The main contribution is the design of coinciding walk kernels, which leverage – next to the homophily assumption – local structure similarity to tackle learning on the node level with sparse training data. Inspired by the success of random walk-based schemes for the construction of propagation kernels in the first part of this thesis, coinciding walk kernels are defined in terms of the probability that the labels encountered during parallel random walks coincide. In addition to its intuitive probabilistic interpretation, coinciding walk kernels outperform existing kernel- and walk-based methods on the task of node label prediction in sparsely labeled graphs with high label-structure similarity. We also show that their computation is faster than many state-of-the-art kernels on graphs.

Another problem closely related to classification in sparsely labeled graphs is the retrieval of similar items to a very small query set consisting of positive examples only. In order to be able to tackle this task, often called set completion, we will broaden our view on data representation from plain graphs to the relational perspective. As the items can be modeled by nodes in a graph, we can employ random walk-based propagation techniques to find items that are similar to the query items in the sense of either being close in the graph, or having a similar local structure. However, as the task is considerably more difficult, we will have to go beyond the analysis of pure graph representations as used so far in this thesis. The relational perspective on the data allows us to investigate this challenging retrieval task by enriching graph-based approaches with relational techniques such as variabilization of predicates and relational pathfinding.

CHAPTER 6

PROBLEM, PREVIOUS WORK, AND PROPOSED SOLUTION

In the following, we will define the problem of *node classification* in sparsely labeled graphs and review state-of-the-art approaches to tackle it. We will mainly focus on two lines of previous approaches, methods based on random walks (RWs) and kernels on graphs. After identifying the weaknesses of existing work when dealing with sparsely labeled graphs, we will introduce coinciding walk kernels – our approach to overcome them.

6.1 Node Classification in Sparsely Labeled Graphs

Given a partially labeled graph with a set of labeled nodes, node classification is the task of predicting the labels of the remaining unobserved nodes. Node classification in a partially labeled graph is illustrated in Figure 6.1. In contrast to partially labeled graphs, sparsely labeled graphs have only a small fraction of labeled vertices. In the following, we will define the problem of node classification in sparsely labeled graphs in terms of statistical machine learning with networked data.

Problem 6.1 (Node Classification in Sparsely Labeled Graphs) *Given a graph $G = (V, E, \ell)$ with $|V| = n$, $V = V_L \cup V_U$ where $V_L = \{v_i\}_{i=1,\dots,m}$ is the set of nodes with known labels $\ell_i \in [k]$ and $m \ll n$, find the labels ℓ_j of the unlabeled nodes $v_j \in V_U$.*

Node classification in a partially labeled graphs is a well-studied problem. Two of the most popular approaches to solve this task are random walk-based algorithms as introduced in Section 2.3 and kernel methods, cf. Section 2.4. In contrast to the previous part, the kernels are now defined on the nodes of a graph and therefore also termed *kernels on graphs*. In recent years various kernels among the nodes of a graph have been introduced. Commonly used kernels on graphs are the diffusion kernel, Equation (2.20) (Kondor and Lafferty, 2002), the p-step

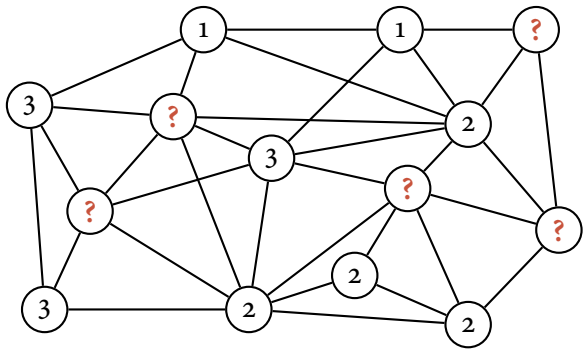


Figure 6.1: **Node Classification.** Illustration of the node classification task in a partially labeled graph with label set $\mathcal{L} = \{1, 2, 3\}$. Given the nodes with observed labels the task is to infer the labels of the nodes marked with ? using the graph structure and, if present, additional information on the nodes and edges.

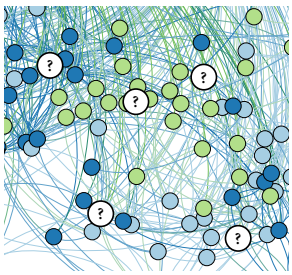
random walk kernel, Equation (2.24) (SMOLA and KONDOR, 2003), the Moore–Penrose pseudoinverse of the Laplacian, Equation (2.22) (FOUSS, *et al.*, 2012), and the regularized Laplacian kernel, Equation (2.21) (SMOLA and KONDOR, 2003). In the related field of semi-supervised learning, so-called data-dependent kernels are often constructed from the Laplacian of a graph modeling the data geometry of vector-valued independent data (ZHOU, *et al.*, 2003; SINDHWANI, *et al.*, 2005; MELACCI and BELKIN, 2011; LEVER, *et al.*, 2012). All of these kernels have random walk interpretations; however, none of them considers known labels during their computation, which is – as we will see in the next chapter – crucial when dealing with sparsely labeled graphs. Whereas kernel methods such as support vector machines enable us to construct flexible decision boundaries often resulting in classification performance superior to that of the model based approaches, RW-based algorithms use the known labels directly by propagating them on the graph in order to infer the labels of unobserved nodes. Successful random walk-based algorithms are label propagation (ZHU, *et al.*, 2003), partially labeled classification with Markov random walks (SZUMMER and JAAKKOLA, 2001), learning with partially absorbing random walks (WU, *et al.*, 2012), and learning with local and global consistency (ZHOU, *et al.*, 2003). Most RW-based approaches, absorbing or not, only analyse the walks' steady-state distributions (KONDOR and LAFFERTY, 2002; ZHU, *et al.*, 2003; LIN and COHEN, 2010*b*; WU, *et al.*, 2012). Random walks using the graph's row-normalized adjacency matrix as the transition matrix converge to a constant steady-state distribution as already discussed in Section 2.3. For these walks, the idea of *early stopping* was successfully introduced in power iteration methods for clustering (LIN and COHEN, 2010*a*) and node label prediction (SZUMMER and JAAKKOLA, 2001). The insight here is that the intermediate distributions obtained

by the random walks during the convergence process are extremely interesting; most existing approaches, however, only consider the probability distributions at (the possibly early) termination, ignoring their evolution.

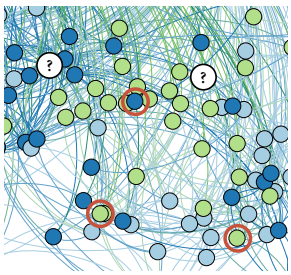
Most walk- and kernel-based methods directly or indirectly imply the homophily or autocorrelation assumption already stated in Hypothesis 2.1. This assumption, however, claiming that nodes that are close to one another in the graph are likely to have the same label, comes with two major drawbacks. The first problem is that for some datasets the assumption is simply not valid. So, in this case by completely ignoring structure similarity and other class indicators we will not be able to solve the classification task satisfactorily (Limitation 1). Second, homophily is extremely difficult to exploit when having access to only few training data, as within a small neighborhood of each unlabeled node we will not have a sufficient amount of label evidence to make confident predictions (Limitation 2). Thus, when dealing with either the first problem or with sparsely labeled graphs, we will have to go beyond labeling or retrieving nodes according to the homophily assumption. Figure 6.2 illustrates both Limitation 1 and Limitation 2. That is, to overcome both problems we have to make use of additional information encoded in the data. This can be achieved, for example, by exploiting additional information in the form of node attributes. Unfortunately, attribute information is not always available. Thus, inspired by the success of predictive graph mining, we propose to exploit structural similarities. In particular, we design a kernel among the nodes of a graph to analyze the local graph structure, which is represented by the nodes, their labels, and the edge structure in a small neighborhood around each node of interest. To do so, we will leverage information propagation techniques based on random walks. Exploiting structure similarity will turn out to be the key insight to derive a good solution to Problem 6.1.

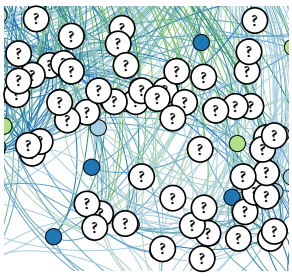
An existing approach to learning with structure similarity for node classification was introduced by DESROSIERS and KARYPIS (2009). A similarity measure based on parallel RWs with constant termination probability is used in a relaxation labeling algorithm; we will denote this method by RL. Intuitively, parallel RWs are label sequences generated from multiple random walks starting from different nodes. However, instead of comparing actual walks, we compare the probabilities of their occurrences. Parallel RWs will be defined in the next chapter. The similarity measure used in RL corresponds to the probability that parallel random walks with constant termination probability are of exactly the same length and generate exactly the same sequence of labels at all times. Even though RL is rather general in the sense of being able to use labeled edges, it has two major drawbacks. First, it has four parameters: two regulating the influence of label uncertainty and of the similarity measure in the relaxation labeling iterations, the constant termination probability γ, and the maximum walk length t_{MAX}. Second, the minimal space complexity for node label prediction scales with the number of unlabeled nodes ($|V_U| \times n$), which is unfavorable for node classification in large sparsely labeled

(a) Populated places graph



(b) Limitation 1





(c) Limitation 2

Figure 6.2: **Limitations of Hypothesis 2.1.** This figures illustrates the limitations of the homophily assumption for node label prediction on a subgraph of the POPULATED-PLACES graph extracted from DBpedia consisting of the 500 nearest nodes to the node "Atlanta." The graph layout algorithm used (OpenOrd) was force-directed; nearby nodes have a high connectivity. Class labels are encoded by color (*country* (green), *administrative region* (light green), *city* (blue), *town* (light blue), *village* (pink)). The edge colors are created by perceptual blending of the colors of the incident nodes. Panel (a) shows the whole subgraph and the detail used to illustrate the limitation. Panel (b) illustrates Limitation 1: for the three nodes marked red the homophily assumption is not valid. Panel (c) illustrated Limitation 2: in sparsely labeled graphs homophily is difficult to exploit.

graphs. Moreover, their similarity measure $R^{(t_{\text{MAX}})}$ is used in an iterative relaxation labeling approach in which $R^{(t_{\text{MAX}})}$ is changing over the iterations, and despite being a valid kernel[1] it is not clear how to use it directly in a kernel method.

[1]Note that DESROSIERS and KARYPIS (2009) did not show that their similarity matrix R is a kernel, but by taking a slightly different view we can write $R^{(t_{\text{MAX}})} = \gamma^2 \sum_{t=0}^{t_{\text{MAX}}} (1-\gamma)^{2t} \prod_{\tilde{t}=0}^{t} P_{\tilde{t}} P_{\tilde{t}}^{\top}$,

Another approach to node classification, exploiting the structure of subnetworks to derive local and global features for the discovery of disease genes, is heterogeneous label propagation (HWANG and KUANG, 2010). The biggest limitation of this work is that it needs explicitly known subnetworks. Label-independent features such as node degrees or number of incident links, or measures like betweenness centrality or clustering coefficients, are used for node classification in (GALLAGHER and ELIASSI-RAD, 2008). However, these measures are incorporated in a rather *ad hoc* fashion and no kernel is defined. Random walks with restart are used as proximity weights for ghost edges in (GALLAGHER, *et al.*, 2008), but then the features considered by a later bag of logistic regression classifiers are only based on a one-step neighborhood. Other previous approaches to classification, in particular in sparsely labeled graphs, introduce latent graphs by adding additional edges (SHI, *et al.*, 2011), run multiple random walks with restarts (LIN and COHEN, 2010*b*), and suggest schemes for active learning (JI and HAN, 2012). Methods using label information to improve kernels, known as label-dependent kernels or kernel–target alignment, have proven successful in semi-supervised and supervised learning (CRISTIANINI, *et al.*, 2001; ZHU, *et al.*, 2004; MIN, *et al.*, 2007). A similar approach, performing label-dependent feature extraction, was successfully applied to node classification in social networks (KAJDANOWICZ, *et al.*, 2010). All these approaches either do not define a kernel – which is one of our goals here – or they do not use the label information directly in the kernel construction. This can, however, be elegantly done by monitoring label-absorbing random walks, as we will show in the following.

6.2 Our Approach: Coinciding Walk Kernels

To overcome the problems of merely exploiting the homophily assumption, changing the graph structure, or adapting the kernel to the observed class labels after its computation, we propose an alternative approach: *coinciding walk kernels* (CWKS). Coinciding walk kernels move beyond the straightforward homophily assumption by making an extended assumption. Not only are nearby nodes likely to have the same label, but also nodes with similar local structure. The goal of CWKS is to exploit the label-structure similarity assumption, where label structure is the arrangement and connectivity of labels on nearby nodes, as stated in Definition 2.10.

HYPOTHESIS 6.1 (LABEL-STRUCTURE SIMILARITY ASSUMPTION) *Nodes with similarly arranged labels in their local neighborhoods are likely to*

- *be similar,*

where $\prod$ is the Hadamard product , i.e., the element-wise product of matrices. This leads to a straightforward proof of positive definiteness for graphs with unlabeled edges; see also Appendix A.

- *have the same properties,*
- *have the same label,*
- *be in the same cluster, or*
- *form an edge.*

In particular, coinciding walk kernels use short random walks to quantify how similarly the labels surrounding each node are arranged in the graph. The design of CWKs is clearly inspired by the construction of propagation kernels; however, they define a kernel among the nodes of a graph.

None of the existing kernels on graphs, cf. Section 2.4.2, considers known labels during their computation, and as a result, they cannot take advantage of Hypothesis 6.1. So, in contrast to these approaches, we view known node labels as providing valuable information that should be considered in the construction of a kernel used for node label prediction. In coinciding walk kernels *partially absorbing random walks* (PARWs) with the absorbing states being the observed nodes give the known labels influence over the walk process (ZHU, *et al.*, 2003; WU, *et al.*, 2012). Intuitively, a PARW stops progressing with some probability once it hits a labeled node. PARWs will be introduced formally in the next chapter. We consider the distribution over sequences of labels encountered during a PARW from a node as encoding its label structure. To address Hypothesis 6.1, we then define the CWK between two nodes to be the probability that parallel PARWs leaving from those nodes coincide, that is, hit the same label at the same time. By lifting the random walk from being on the nodes of a graph to being on its labels, two nodes can be similar even if they are very distant from each other in the graph, or even on disconnected graphs. On the other hand, two PARWs could encounter similar label sequences simply by virtue of having left from nearby nodes in the graph, so the CWK is also compatible with Hypothesis 2.1.

In addition, coinciding walk kernels implement the early stopping criteria, cf. Section 2.3, by using the entire evolution of labels encountered during partially absorbing random walks up the a given length as representing local structure, rather than using only the limiting distribution. CWKs therefore substantially leverage inference by aggregating label predictions based on different walk lengths. The distribution of labels encountered during PARWs clearly depends on the locations and labels of previously observed nodes, connecting the CWK to data-dependent kernels (ZHOU, *et al.*, 2003; SINDHWANI, *et al.*, 2005; MELACCI and BELKIN, 2011; LEVER, *et al.*, 2012). Our approach, however, investigates kernel construction leveraging label-structure information in plain graph data where no features on the nodes are given. This means that the kernel developed is – in contrast to the kernels used in standard semi-supervised learning – a *label-dependent kernel* rather than a data-dependent kernel. Instead of modifying an existing kernel to improve

alignment on the labeled data as done in kernel–target alignment approaches, CWKs use the label information directly in the kernel construction by exploiting label absorbing random walks.

The main contribution to solve Problem 6.1 is the introduction of the coinciding walk kernel, the first label-dependent kernel on graphs leveraging label information directly in the kernel construction, which provides a learning method for node classification that intertwines inference and kernels on graphs. Moreover, CWKs efficiently combine the benefits of kernel methods and RW-based inference approaches in networked data. We will demonstrate empirically that this can considerably improve node label prediction, especially in sparsely labeled graphs, overcoming the limitations of merely exploiting the homophily assumption.

CHAPTER 7

COINCIDING WALK KERNELS

In this chapter, we derive and evaluate coinciding walk kernels for node classification on sparsely labeled graphs. First, we will define the main ingredient of coinciding walk kernels, namely partially absorbing random walks. After introducing the coinciding walk kernel and its probabilistic interpretation, we show its positive definiteness, illustrate its capability to capture label-structure similarity, and evaluate its parameter sensitivity, runtime, and predictive performance empirically.

7.1 Partially Absorbing and Parallel Random Walks

As the main ingredient of coinciding walk kernels, the label-structure similarity of nodes in a graph, is modeled by the probability that parallel partially absorbing random walks coincide, we will now review partially absorbing and parallel Markov random walks on graphs. Further, we will explain how we examine label-structure similarity via parallel partially label-absorbing random walks. General Markov random walks on a graph are defined in Section 2.3. Recall that random walks can be intuitively illustrated by a particle traveling from node to node via the edges of the graph. In every node, the decision where to go next only depends on the current location and thus, the node to visit next is selected proportional to the number of edges and edge weights connecting the current node to its neighbors. This walk behaviour can be modified by introducing *absorbing states*. When reaching an absorbing state, our particle is not able to continue its walk, but is instead caught in a loop staying at that node. The absorbing states could for example be chosen as the subset of the nodes having observed labels, a natural choice in our paradigm. Further, we can define partially absorbing random walks, where absorbing nodes only activate with a given probability.

7.1.1 Absorbing Random Walks

Consider a graph $G = (V, E)$ with $|V| = n$ vertices and a set of edges E specified by a weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$. Let now $S \subseteq V$ be a subset of nodes

in G. Given T as defined in Equation (2.5), we define an *absorbing random walk* to have the modified transition probabilities $\hat{T}$, defined as

$$\hat{\tau}_{ij} = \begin{cases} 0 & \text{if } i \in S \text{ and } i \neq j \\ 1 & \text{if } i \in S \text{ and } i = j \\ \tau_{ij} & \text{otherwise,} \end{cases} \tag{7.1}$$

where τ_{ij} refers to the entries of T. Nodes in S are "absorbing" in that a walk never leaves a node in S after it is encountered. A node $s \in S$ is also called a *sink*.

In analogy with the label diffusion and label propagation algorithms introduced in Section 2.3.2, we lift the random walk from being on the nodes of a graph, cf. Equation (2.8), to being on its labels. Consider a partially labeled graph $G = (V, E, \ell)$ where $V = V_L \cup V_U$ is the union of labeled and unlabeled nodes, $\ell \colon V \to [k]$ is a label function with known values for the nodes in V_L, and k is the number of available labels. We will describe how we can monitor the distribution of labels encountered during absorbing random walks on G. Let the matrix $P_0 \in \mathbb{R}^{n \times k}$ give the prior label distributions of all nodes in V. If node $v_i \in V_L$ is observed with label $\ell(v_i)$, then the ith row in P_0 is the Kronecker delta distribution concentrating at $\ell(v_i)$; i.e., $(P_0)_{i,:} = \delta_{\ell(v_i)}$. We initialize the label distributions for the unlabeled nodes V_U with some prior, for example a uniform distribution.[1] The ith row of P_0 now gives the initial probability distribution for the first label encountered, $\ell(X_0^{(i)})$, for an absorbing random walk of length 0 starting at v_i. Now, it is easy to see by induction that by iterating the map

$$P_t \leftarrow \hat{T} P_{t-1}, \tag{7.2}$$

$(P_t)_{i,:}$ gives the distribution over $\ell(X_t^{(i)})$. If we now set the absorbing states to be the labeled nodes, $S = V_L$, then we have a *label-absorbing random walk*. This label-absorbing random walk is equivalent to the label propagation algorithm (ZHU, *et al.*, 2003) introduced in Section 2.3.2 as modifying the transition matrix, Equation (7.1), and performing iterative distribution updates with it, Equation (7.2), is the same as pushing back the observed labels after each iteration, Equation (2.11). Hence, label propagation can be understood as simulating label-absorbing random walks and predicting for an unlabeled node the label most likely encountered first on these walks.

DEFINITION 7.1 (ABSORBING RANDOM WALK) *Let $G = (V_L \cup V_U, E)$ be a partially labeled graph and $X = \{X_t \mid t > 0\}$ a Markov process on the nodes $V = V_L \cup V_U$. Let $\hat{T}$ be a transition matrix as defined in Equation (7.1) with $S = V_L$. Then X is called an absorbing random walk if $P(X_{t+1} = v_j \mid X_t = v_i) = \hat{\tau}_{ij}$.*

We will use the term absorbing random walk to actually mean label-absorbing random walk for the rest of this thesis.

[1] This prior could also be the output of an external classifier built on available node attributes.

7.1.2 Partially Absorbing Random Walks

Random walks with fully absorbing states at the labeled nodes as defined above are somewhat restrictive, as only the first label encountered will have an impact on the evolution of a particular random walk. A straightforward extension is to soften the definition of absorbing states. This can be naturally achieved by employing *partially absorbing random walks* (PARWS) (WU, *et al.*, 2012). In the setting of label-absorbing random walks, the simplest way to define PARWS is to extend our graph G by adding a special node for each label in $[k]$ and adding edges from each labeled node $v_i \in V_L$ to its respective label node. We then make these auxiliary nodes absorbing states and vary the transition probabilities from these to the labeled nodes. The transition probabilities in this graph $\tilde{G} = (V \cup [k], \tilde{E})$ are given by $\tilde{T}$ having the following block structure:

$$\tilde{T} = \begin{bmatrix} T_{U,U} & T_{U,L} & 0 \\ (1-\alpha)\, T_{L,U} & (1-\alpha)\, T_{L,L} & \alpha\, \delta_L \\ 0 & 0 & I \end{bmatrix}, \tag{7.3}$$

where $\alpha \in [0,1)$ is the absorbing probability. Partially absorbing random walks generalize both, fully absorbing random walks and non-absorbing random walks defined above. By setting $\alpha = 1$ we can exactly model the fully absorbing random walks in Definition 7.1. On the other hand, by setting $\alpha = 0$ we get the simple non-absorbing random walk from Definition 2.12, which when lifted to the labels leads the label diffusion process stated in Equation (2.10). Note that again we will refer to the partially label-absorbing random walk by the simple term partially absorbing random walk.

DEFINITION 7.2 (PARTIALLY ABSORBING RANDOM WALK) *Let $G = (V_L \cup V_U, E)$ be a partially labeled graph and $X = \{X_t \mid t > 0\}$ a Markov process on the nodes $V = V_L \cup V_U$. Let $\tilde{T}$ be a transition matrix of the extended graph $\tilde{G} = (V \cup L^{(L)}, \tilde{E})$ as defined in Equation (7.3) with $\alpha \in [0,1]$. Then X is called a partially absorbing random walk if $P(X_{t+1} = v_j \mid X_t = v_i) = \tilde{\tau}_{ij}$.*

7.1.3 Parallel Random Walks

The final ingredient we need are parallel random walks, as they allow one to refer to the sequences of states of two or more random walks of the same length. Co-occurring random walks can be used to describe the similarity of either entire graphs or nodes in a graph based on the structure of the local neighbourhood of the nodes. These similarities will be the basis of the coinciding walk kernel defined in the next section. Let us now give a formal definition of parallel random walks. A parallel random walk of length t_{MAX} among a set of nodes S is given by the sequences $\{X_t^{(i)}\}_{0 \le t \le t_{\text{MAX}}}$ of t_{MAX} states visited by the random walks starting at the respective nodes $v_i \in S$.

DEFINITION 7.3 (PARALLEL RANDOM WALK) *Let $G = (V, E)$ be a graph and $X = \{X_t \mid t > 0\}$ a Markov process on the nodes V. A parallel random walk of length t_{MAX} among a set of nodes S is given by the sequences $\{X_t^{(i)}\}_{0 \leq t \leq t_{MAX}}$ of t_{MAX} states visited by the random walks starting at the respective nodes $v_i \in S$.*

7.2 KERNEL COMPUTATION

Now we will introduce and define the coinciding walk kernel, which is the main contribution of our work. The intuition underlying CWKS is simple: PARWS on partially labeled graphs encode both label and structure similarity. Thus, CWKS can exploit Hypotheses 2.1 and 6.1 for learning tasks on graphs. Before we show that K_{CW} is a valid kernel, we discuss its probabilistic interpretation as well as some interesting properties.

7.2.1 *Definition and Random Walk Interpretation*

The coinciding random walk kernel on a graph $G = (V, E)$ is defined as

$$K_{\text{CW}} = \frac{1}{t_{\text{MAX}} + 1} \sum_{t=0}^{t_{\text{MAX}}} P_t P_t^\top, \tag{7.4}$$

where the matrices of label probabilities $P_t \in \mathbb{R}^{n \times k}$ are obtained by replacing $\hat{T}$ by $\tilde{T}$ in Equation (7.2) and considering the respective entries in the extended label probability matrix $\tilde{P}_t$,

$$\tilde{P}_{t+1} \leftarrow \tilde{T}\tilde{P}_t$$
$$P_t = (\tilde{P}_t)_{i|v_i \in V},$$

and $P_t P_t^\top$ is an outer product of the discrete label probability distributions. Note that $\tilde{P}_t \in \mathbb{R}^{n+k \times k}$ is simply the probability matrix P_t extended by a $k \times k$ identity matrix. K_{CW} has two kernel parameters: the absorbing probability α, and the maximum walk length t_{MAX}, where α controls trade-off between the homophily and label-structure similarity assumptions.

$(P_t)_i (P_t)_j^\top$ can be interpreted as the probability that parallel PARWS leaving from v_i and v_j are on nodes with the same label at time t, that is, that $\ell(X_t^{(i)}) = \ell(X_t^{(j)})$. Hence, the coinciding walk kernel has the following intuitive interpretation: its value for two nodes v_i and v_j is the probability that parallel PARWS of length t_{MAX} starting from v_i and v_j encounter the same label at any given time $0 \leq t \leq t_{\text{MAX}}$.

LEMMA 7.1 (COINCIDING WALK KERNELS ARE POSITIVE SEMIDEFINITE) *K_{CW} as defined in Equation (7.4) is positive semidefinite (i.e., is a valid Mercer kernel).*

PROOF It is obvious that K_{CW} is a positive-semidefinite kernel as it is the scaled sum of polynomial kernels $k(x, y) = (x^\top y + c)^d$, with $c = 0$ and $d = 1$; i.e., $K_{\text{CW}}(v_i, v_j) \propto \sum_{t=0}^{t_{\text{MAX}}} (P_t)_i (P_t)_j^\top$. □

Algorithm 6 Coinciding Walk Kernel Computation

given: max walk length t_{MAX}, absorbing rate α, inital label distributions $P_0 \in \mathbb{R}^{n\times k}$, weighted adjacency matrix $W \in \mathbb{R}^{n\times n}$
initialization: $K \leftarrow P_0 P_0^\top$, $T = D^{-1}W$, where $D = \text{diag}(\sum_j w_{ij})$
$\tilde{T} \leftarrow \text{CONSTRUCT-TRANS}(T, \alpha)$ ▷ cf. Eq. (7.3)
for $t \leftarrow 1 \ldots t_{\text{MAX}}$ **do**
  $\tilde{P}_t \leftarrow \tilde{T}\tilde{P}_{t-1}$ ▷ one step transition
  $P_t \leftarrow (\tilde{P}_t)_i$, $i \in \{1, .., n\}$
  $K \leftarrow K + P_t P_t^\top$ ▷ add kernel contribution
end for
$K_{\text{CW}} \leftarrow \frac{1}{t_{\text{MAX}}+1} K$ ▷ normalize kernel

The computation of the coinciding walk kernel on a graph G is summarized in Algorithm 6.

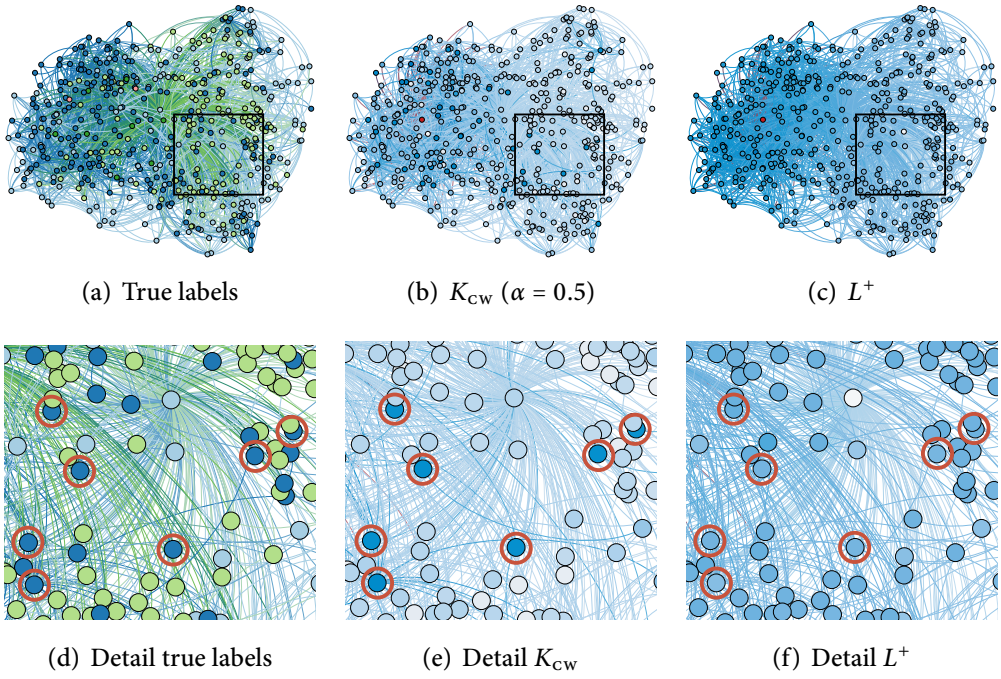
7.2.2 Complexity Analysis

The runtime complexities of the required naïve calculations are $\mathcal{O}(k\, t_{\text{MAX}}\, |E|\, n)$ for the one step transition and $\mathcal{O}(k\, t_{\text{MAX}}\, n^2)$ for the kernel contribution, where $|E|$ is the number of edges. It is worth mentioning that for most learning tasks it is sufficient to compute the train–train and train–test fractions of the kernel matrix. This can be accomplished efficiently by precomputing the $\{P_t\}$ and summing only the required outer products with a complexity of $\mathcal{O}(k\, t_{\text{MAX}}\, |V_L| n)$. Algorithm 6 has an overall computational complexity of $\mathcal{O}(k\, t_{\text{MAX}}\, |E|\, n)$; however, the kernel computation for sparse graphs (small $|E|$) with few labeled nodes ($|V_L| \ll n$) is efficient.

7.2.3 Illustration

Figure 7.1 illustrates the coinciding walk kernel on a subgraph of a labeled graph built from concepts in the DBpedia ontology marked as "populated places."[2] Each concept is a node in our graph and is backed by a Wikipedia page. We added an undirected edge between two places if one of their corresponding Wikipedia pages links to the other. In the DBpedia ontology populated places are divided into five classes *country*, *administrative regions*, *city*, *town*, and *village*. This example was chosen because the resulting graph does not necessarily exhibit homophily; for example, villages (approximately half the dataset) are much more likely to link to countries than to other villages. For our illustration, we built a graph with $|V| = 500$ nodes by taking a breadth-first search from "Atlanta." We then calculate the pseudoinverse of the Laplacian kernel (L^+) as well as the coinciding walk kernel

[2] An implementation of the used DBpedia (`www.dbpedia.org`) graph extractor is available at `https://github.com/rmgarnett/dbpedia_graph_extractor`.

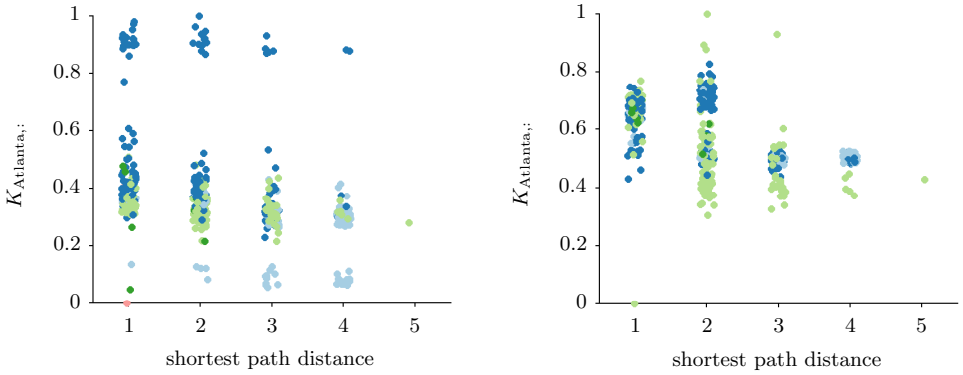


(a) True labels (b) K_{cw} ($\alpha = 0.5$) (c) L^+

(d) Detail true labels (e) Detail K_{cw} (f) Detail L^+

Figure 7.1: **Subgraph of the POPULATED-PLACES Dataset.** Panels (a-c) show a subgraph of the POPULATED-PLACES graph extracted from DBpedia consisting of the 500 nearest nodes to the node "Atlanta." The graph layout algorithm used (OpenOrd) was force-directed; nearby nodes have a high connectivity. The edge colors are created by perceptual blending of the colors of the incident nodes. Panel (a) shows the class labels (*country* (green), *administrative region* (light green), *city* (blue), *town* (light blue), *village* (pink)). Panel (b) and (c) illustrate the values of the coinciding walk kernel and L^+ of the kernel row for Atlanta, colored red. Dark blue means high similarity, i.e., high kernel value, and white represents low similarity. Panels (d-f) show a detail of the graphs in (a-c), where some *city* nodes are highlighted by red circles.

(with $\alpha = 0.5$ and $t_{\mathrm{MAX}} = 10$), using a random selection of 20% of the nodes for V_L. Atlanta was not among the labeled nodes. The graph layout reflects connectivity so that nearby nodes in the plot have higher connectivity. The rows of K_{cw} corresponding to $K_{\mathrm{Atlanta},:}$ are illustrated in Figure 7.1 (b) and (e). As Atlanta is a city we expect other nodes with the ground truth label *city* to have high kernel values. One can clearly see that CWK is able to capture structure similarity as several distant nodes with ground truth *city* nodes have high values and nearby nodes having ground truth labels other than *city* including nodes in the direct neighbourhood of Atlanta show low values. The rows of L^+ are shown in Figure 7.1 (c) and (f). Here, we see that L^+ (on average) decreases smoothly with increasing distance



(a) K_{CW} ($\alpha = 0.5$) (b) L^+

Figure 7.2: **Correlation of Kernel Values with Distance and Class Labels.** Scatter plots of the ground truth class labels (encoded by color) of the nodes in the subgraph of the POPULATED-PLACES graph. Shortest path distance is plotted against the normalized values of $K_{\text{Atlanta},:}$ for K_{CW} (a) and L^+ (b), respectively.

from Atlanta reflecting the homophily assumption, Hypothesis 2.1; whereas the value of K_{CW} also shows some highly correlated far-away nodes, cf. Figure 7.1 (e), as well as less correlated nearby nodes. This behavior can also be observed by comparing the shortest path distance to the respective kernel values as illustrated in Figure 7.2. Here, the magnitude of K_{CW} is highly correlated with the correct label (*city*) – the highest kernel values are exclusively achieved by other cities throughout the network, exactly the behavior desired for predicting Atlanta's label. It is also interesting to note that the lowest kernel values are exclusively among nodes in the "town" class (with one exception where the label is "administrative region"), perhaps due to strikingly different label structure in their neighborhoods. For L^+ the correlation between kernel values and class labels is not clearly visible.

7.2.4 *Limit Case: Label Propagation Kernel*

In the following, we will give further intuition of CWKS by considering $t_{\text{MAX}} \to \infty$ and $\alpha = 1$. In this case, we are effectively taking infinitely long label-absorbing RWS. Because every such walk is eventually constant, the sequence of label probabilities $\{P_t\}$ will eventually converge; call this converged matrix P_∞. Notice that P_∞ is exactly the matrix calculated when performing label propagation, as given in Equation (7.2). We define the following *label propagation kernel:*

$$K_{\text{LP}} = P_\infty P_\infty^\top. \tag{7.5}$$

Notice that for two observed nodes $i, j \in V_L$, we have $K_{\text{LP}}(i, j) = 1$ if $\ell(i) = \ell(j)$ and $K_{\text{LP}}(i, j) = 0$ otherwise. For a labeled node i and an unlabeled node j, we have

$K_{\text{LP}}(i,j) = (P_\infty)_{\ell(i),j}$; that is, the K_{LP} matrix is simply populated by the converged label propagation probabilities. Now, we have the following corollary of Lemma 7.1.

COROLLARY 7.1 *K_{LP} as defined in Equation (7.5) is positive semidefinite (i.e. is a valid Mercer kernel).*

Further, for $\alpha = 1$ K_{CW} converges to K_{LP} in the limit as $t_{\text{MAX}} \to \infty$,

$$K_{\text{LP}} = \lim_{t_{\text{MAX}} \to \infty} K_{\text{CW}}.$$

This can be easily seen as the sum in Equation (7.4) is eventually dominated by outer products of the converged P_t terms.

7.3 Empirical Evaluation

Our intention here is to investigate the power of coinciding walk kernels for the task of node label prediction in sparsely labeled graphs.[3] We compare their performance to existing methods from the kernel and collective inference community. The main questions to answer are:

(Q7.1) Are CWKs able to capture structure similarity and therewith improve over state-of-the-art graph-based learning methods on datasets suggesting Hypothesis 6.1?

(Q7.2) Can CWKs perform competitive with state-of-the-art methods on datasets where mostly homophily (Hypothesis 2.1) holds?

(Q7.3) Can CWKs be computed efficiently with respect to runtime and space complexity?

Further, we analyse parameter sensitivity and computational properties of CWKs.

7.3.1 Experimental Protocol

We compare the classification accuracy in several real-world graphs of the following methods:

- CWK: coinciding walk kernels,
- LP: label propagation (ZHU, *et al.*, 2003),
- RL: relaxation labeling using structure similarity (DESROSIERS and KARYPIS, 2009),
- LGC: local and global consistency (ZHOU, *et al.*, 2003),

[3] Our CWK implementation is available at `https://github.com/rmgarnett/coinciding_walk_kernel`.

- VND: von Neumann diffusion kernel (ZHOU, *et al.*, 2003; FOUSS, *et al.*, 2012),
- DIFF: diffusion kernel (KONDOR and LAFFERTY, 2002), and
- L+: pseudoinverse of the (normalized) Laplacian (FOUSS, *et al.*, 2012).

LP as defined in Equation (2.11) is the obvious baseline approach. RL was briefly introduced in Section 6.1. It is currently the most accurate method in the area of collective classification, cf. results in (DESROSIERS and KARYPIS, 2009). LGC is a diffusion scheme for semi-supervised learning suggested in (ZHOU, *et al.*, 2003). During the development of their method, ZHOU, *et al.* (2003) also suggested the kernel $K_{\text{VND}} = (I - \alpha D^{-1/2} A D^{-1/2})^{-1}$ which is the von Neumann diffusion kernel on the normalized adjacency matrix (VND) (FOUSS, *et al.*, 2012), cf. Equation (2.23) with the adjacency matrix instead of the weight matrix. Note that VND is closely related to the regularized Laplacian kernel, Equation (2.21) (SMOLA and KONDOR, 2003). Hence, we choose VND, DIFF and L+ to represent existing successful kernels on graphs. All kernel-based predictions (CWK, VND, DIFF, and L+) are achieved via support vector machine (SVM) classification.

The following graph datasets are used for evaluation:

- POPULATED-PLACES[4] (link graph extracted from DBpedia, described in Section 7.2.3),
- WEBKB[5] (cocitation graph of webpages from computer science departments of four universities),
- DBLP[6] (connected coauthor graph extracted from the DBLP database),
- CORA[7] (citation network of scientific papers), and
- CITESEER[7] (citation network of scientific papers).

To measure to which extend homophily, cf. Definition 2.11, is present in the datasets, we compute a statistic, P_{switch}, as the probability that a mixed random walk switches labels on adjacent nodes. That is, if P_∞ is the stationary distribution of the random walk[8] and $P_{\text{switch}|i}$ is the vector of conditional probabilities of switching labels from a given node, then $P_{\text{switch}} = P_\infty^\top P_{\text{switch}|i}$. Low values of this measure signal the presence of homophily (favoring Hypothesis 2.1), whereas high values indicate a lack of label smoothness (rejecting Hypothesis 2.1). For datasets with low P_{switch}, exploiting label-structure similarity (Hypothesis 6.1) may be more beneficial. P_{switch} and other properties of all datasets are summarized in Table 7.1. For the POPULATED-PLACES dataset, we created graphs of varying sizes by performing

[4] `http://www-kd.iai.uni-bonn.de/pubattachments/727/populated_places.tar.xz`

[5] `http://www.netkit-srl.sourceforge.net/data.html`

[6] `http://www.cs.illinois.edu/homes/mingji1/DBLP_four_area.zip`

[7] `http://www.cs.umd.edu/projects/linqs/projects/lbc/index.html`

[8] P_∞ can be calculated as the normalized eigenvector of $T^\top$ with maximal eigenvalue.

Table 7.1: **Dataset Statistics and Properties.** PP-xK is short for POPULATED-PLACES-xK. # graphs indicates the number of connected components and P_{freq} the proportion of the most frequent class. P_{switch} reflects the probability of adjacent nodes switching their labels.

	properties					
dataset	# nodes	# edges	# labels	# graphs	P_{freq}	P_{switch}
PP-1K	1 000	5 253	5	1	43%	69%
PP-3K	3 000	16 546	5	1	50%	66%
PP-5K	5 000	26 648	5	1	53%	70%
WEBKB	1 462	61 766	6	4	28%	33%
DBLP	1 711	2 898	4	1	36%	21%
CORA	2 708	5 278	7	78	30%	18%
CITESEER	3 264	4 536	6	390	21%	26%
PP-100K	100 000	374 480	(used for runtime analysis cf. Fig. 7.6)			

a breadth-first search from the first node in the graph (Alabama). Note that the POPULATED-PLACES datasets have a rather high probability that adjacent nodes are of different labels, i.e. P_{switch} is high. This can also be seen in Figure 7.1(a). Hence, for these datasets we expect structure similarity to be important for node label prediction. For WEBKB we combined the cocitation networks of all universities (Cornell, Texas, Washington, and Wisconsin) into one disconnected graph.

We focus on sparsely labeled graphs and use 20 randomly generated test splits for 1% up to 15% labeled nodes. The test sets are the same for each method and all reported classification accuracies are an average over the results on the 20 test sets. The performance of all kernel-based classifiers is evaluated by running C-SVM classifications using `libSVM`.[9] Parameter learning is done by the following protocol. For each method we train all parameters (including the SVM cost parameter) jointly via grid search on 10 randomly generated training splits having 5% and 10% labeled nodes. Again, the training sets are the same for each method. For prediction we use the first set of parameters (trained for 5% labeled data) for training percentages from 1% to 7% and the second set of parameters for all scenarios with more than 7% labeled data. The following parameter values were tested:

- CWK: $t_{\text{MAX}} \in \{0, 1, \ldots, 10, 20, \ldots, 200\}$,
  $\alpha \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 0.98, 0.99, 1\}$,

[9] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

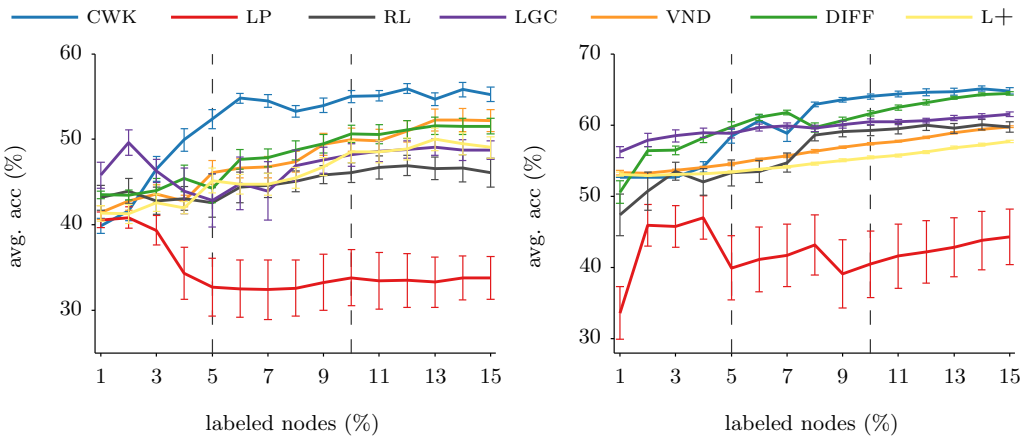
Table 7.2: **Results on POPULATED-PLACES.** Average accuracies (%) on 20 test sets of the POPULATED-PLACES datasets for 5% and 10% labeled nodes. PP-xk is short for POPULATED-PLACES-xk. • indicates statistically significant best performance among the kernel methods and bold indicates statistically significant best performance among all methods both under a paired t-test ($p < 0.05$).

dataset		method						
		CWK	DIFF	L+	VND	RL	LGC	LP
PP-1k	5%	**52.4** •	44.3	45.2	46.1	42.6	42.8	32.7
PP-3k	5%	61.0	60.2	51.5	52.6	57.7	60.9	44.7
PP-5k	5%	58.4	59.7	53.4	54.5	53.3	58.8	39.9
PP-1k	10%	**55.0** •	50.6	48.5	49.9	46.1	48.2	33.8
PP-3k	10%	63.3	63.2	53.9	55.8	59.4	62.0	50.2
PP-5k	10%	**64.0** •	61.6	55.5	57.4	59.2	60.4	40.4

- RL: $N \in \{1, 2, \ldots, 5\}$, $\gamma \in \{0.1, 0.3, 0.5, 0.7\}$, $\alpha_{\text{RL}} \in \{0.25, 0.5, \ldots, 1.5\}$, $\beta_{\text{RL}} \in \{0.5, 1.0, \ldots, 3.0\}$,
- LGC and VND: $\alpha \in \{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 0.8, 0.9, 0.95, 0.98, 0.99\}$,
- DIFF: temperature $\beta \in 2^{\{-7,\ldots,7\}}$, and
- all kernel methods: SVM cost $C \in 2^{\{-7,\ldots,7\}}$.

7.3.2 *Predictive Performance*

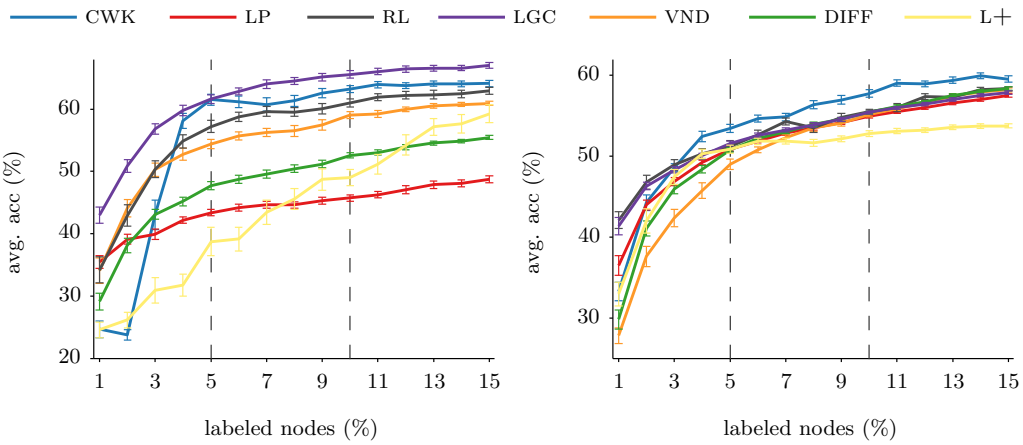
Exploiting Structure Similarity. Here, we analyse the performance of all methods on the POPULATED-PLACES datasets where – indicated by a high switching probability P_{switch}, cf. Table 7.1 – one can expect structure similarity to be useful for classification. The predictive performances for three variations with 1 000, 3 000, and 5 000 nodes (PP-1k, PP-3k, PP-5k) for 5% and 10% labeled nodes are summarized in Table 7.2. CWK performed significantly better (under a paired t-test with $p < 0.05$) than the comparing methods on three out of six experiments. Only in one of six cases (5% on PP-5k) did DIFF and LGC perform slightly better than CWK; however, the difference was not significant. Hence, exploiting label-structure similarity via partially label absorbing random walks clearly improves over existing graph-based learning on non-homophilic datasets. Figure 7.3 shows results for 1% to 15% labeled nodes for PP-1k and PP-5k. In general, we observe that CWK achieves the best results followed by all other kernels on graphs (VND, DIFF, and L+), LGC and RL. Thus, question **(Q7.1)** can be answered affirmatively. LP fails to accurately predict the labels in the populated places graphs as it relies purely on the homophily assumption and therefore cannot leverage the structure similarity inherent to these networks. Surprisingly, RL does not achieve convincing results either.



(a) PP-1k (b) PP-5k

Figure 7.3: **Results on POPULATED-PLACES.** Average accuracies and standard errors (%) on PP-1k and PP-5k for all compared methods. The dashed lines indicate 5% and 10% training data, corresponding to the results in Table 7.2.

Common Benchmark Graphs. The predictive performances for four common benchmark graphs (DBLP, WEBKB, CORA, and CITESEER) with 5% and 10% labeled nodes are summarized in Table 7.3. These datasets mostly obey the autocorrelation assumption (Hypothesis 1) which can be seen from the rather low label switching probabilities reported in Table 7.1. In the scenario with 5% labeled nodes, CWK performed significantly better (under a paired t-test with $p < 0.05$) than the comparing kernel methods (DIFF, L+, and VND) on all four datasets. Further, CWK performs significantly best for CITESEER, whereas LGC outperforms all other methods for DBLP and CORA. When considering 10% labeled nodes, CWK performed significantly best among the kernels on graphs in three out of four cases. Comparing all methods, CWK and LCG both win significantly on one dataset. Overall, CWK outperforms all kernels on graphs on a representative sample of common benchmark graphs. These results indicate that incorporating label information into the kernel construction, i.e., using a label-dependent kernel such as CWK for node classification, improves performance over kernels on graphs using graph structure only. Further, CWK perform competitively compared to a range of successful prediction methods from the areas of semi-supervised learning and collective inference. Figure 7.4 shows average accuracies and standard errors of all compared methods for 1% up to 15% labeled nodes for WEBKB and CITESEER. On WEBKB, CWK is clearly the best performing kernel on graphs and in comparison to all methods it is the second best classifier. On CITESEER, CWK performs significantly better than all baselines for label fractions larger then 3%. As P_{switch} (26%) is fairly low on this dataset, this success might be explained by the huge number



(a) WEBKB (b) CITESEER

Figure 7.4: **Results on Benchmark Graphs.** Average accuracies and standard errors (%) on on WEBKB and CITESEER for all compared methods. The dashed lines indicate 5% and 10% training data, corresponding to the results in Table 7.3.

Table 7.3: **Results on Benchmark Graphs.** Average accuracies (%) on 20 test sets of the datasets DBLP, WEBKB, CORA, and CITESEER for 5% and 10% labeled nodes. • indicates statistically significant best performance among the kernel methods and bold indicates statistically significant best performance among all methods both under a paired t-test ($p < 0.05$).

dataset		method						
		CWK	DIFF	L+	VND	RL	LGC	LP
DBLP	5%	62.8 •	55.6	60.2	56.7	61.7	**64.0**	61.0
WEBKB	5%	61.5 •	47.7	38.8	54.4	57.2	61.6	43.4
CORA	5%	72.1 •	70.6	57.2	59.9	67.9	**73.8**	73.2
CITESEER	5%	**53.5** •	50.8	50.9	49.0	51.2	51.6	50.9
DBLP	10%	69.3 •	65.5	67.9	66.2	67.9	69.4	69.1
WEBKB	10%	63.2 •	52.6	49.0	59.0	61.0	**65.6**	45.7
CORA	10%	76.0	77.2 •	67.4	70.9	73.5	78.2	78.2
CITESEER	10%	**57.8** •	55.4	52.8	55.2	55.5	55.4	54.9

of connected components (390) – CWK is able to capture similarities between two nodes in disconnected components, whereas other kernels on graphs always give a value of zero in these cases. To summarize the results on graph benchmark datasets with low switching probabilities indicating the presence of homophily, CWK does

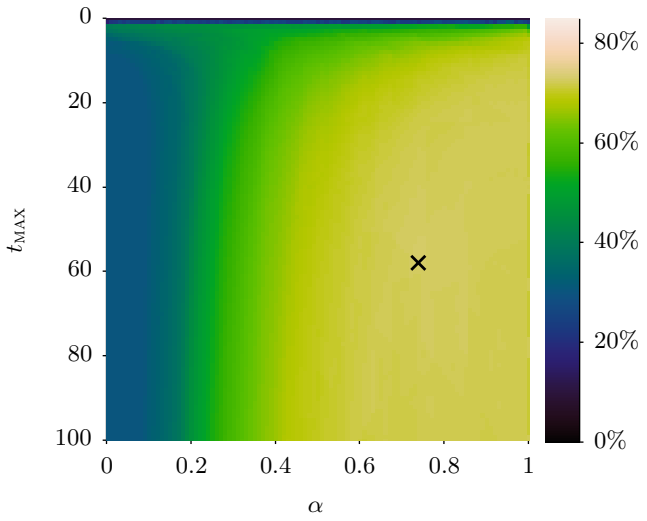


Figure 7.5: **Parameter Sensitivity of CWK.** Heatmap of average accuracies of CWK w.r.t. t_{MAX} and α on the CORA dataset. The accuracies are averaged over 10 randomly generated test sets with 5% labeled nodes. × marks the highest accuracy for $t_{MAX} = 59$ and $\alpha = 0.75$.

outperform all compared kernels on graphs. And also in comparison to all other methods coinciding walk kernels achieve good results. Only LGC performs slightly better in some cases. Nevertheless, we can conclude that CWKs are also able to exploit homophily and thus we can give a positive answer to question **(Q7.2)**.

7.3.3 *Parameter Analysis and Runtimes*

To analyse the sensitivity of CWK's predictive power with respect to changes in the kernel parameters, we computed the average accuracies over 10 randomly generated test sets for all combinations of α and t_{MAX}, where $\alpha \in \{0.0, 0.01, \ldots, 1.0\}$ and $t_{MAX} \in \{0, 1, \ldots, 100\}$ on the CORA dataset with 5% labeled data. A heatmap of the results is shown in Figure 7.5. Whereas the highest accuracy (72.1%) is achieved for an absorbing probability of $\alpha = 0.75$ and a maximum walk length of $t_{MAX} = 59$, we see that for all $\alpha > 0.4$ and $t_{MAX} > 2$, the accuracy is higher than 65%. This shows that CWK is not eminently sensitive to its parameters. The slight slope to the isoperformance curves suggest that walks of a given length and absorbing probability behave somewhat like slightly longer walks with a slightly smaller absorbing probability, which agrees with intuition.

In the following we analyse the scalability of the CWK computation for sparsely labeled networks. As investigating fast and scalable kernel methods goes beyond the scope of our work, we focus our analysis on the scalability of the kernel computation. We compare the runtimes for calculating all tested kernels on the POPULATED-PLACES dataset with up to 100 000 nodes. Once the kernel matrix is

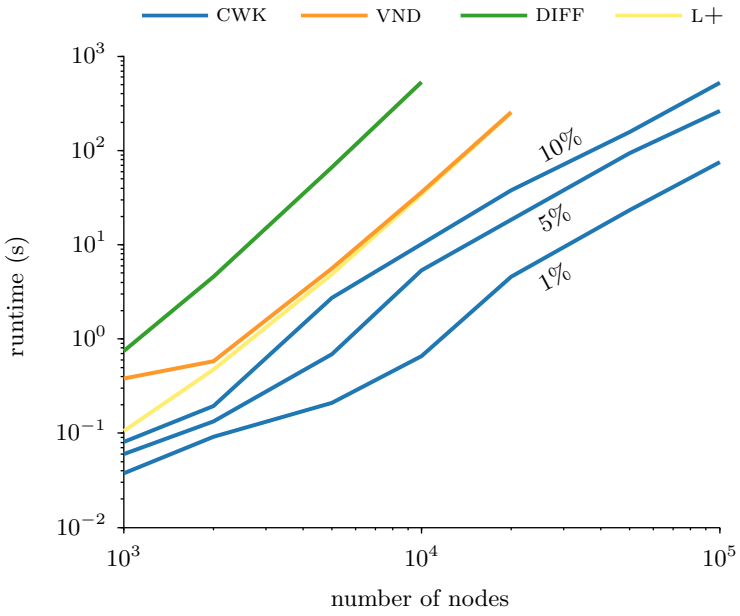


Figure 7.6: **Runtimes on PP-xk.** We show the loglog plot of number of nodes versus runtimes for the kernel computations of all kernels on graphs (CWK with 1%, 5% and 10% labeled nodes, VND, DIFF, and L+) on PP-xk where $x \in \{10^3, 2 \times 10^3, 5 \times 10^3, \ldots, 10^5\}$.

computed, all kernel-based methods scale comparably. Note that the iterative LP method not having to compute a kernel is usually faster then kernel-based classification; however, prediction results are also significantly worse for most datasets. Figure 7.6 shows the runtimes on the PP-xk dataset for $x \in \{1, 2, 5, 10, 20, 50, 100\}$. For CWK, whose runtime depends on the training fraction, we show curves for 1%, 5%, and 10% labeled nodes. We note that the CWK took about the same amount of time with $n = 100\,000$ as DIFF did for $n = 10\,000$ and L+ and VND did for $n = 20\,000$. The smaller slope for the CWK also shows a more slowly growing runtime in general. Finally, we make two remarks regarding the RL method, whose runtime is on a similar order as CWK's. First, RL must recalculate the kernel matrix multiple times, whereas we only compute it once. Second, the time and storage requirements of RL grow with the test size rather than the training size. For example, on the PP-100k dataset with 5% training data, CWK requires approximately 3.7 GB of storage, whereas RL requires about 71 GB.

The computational complexities of the compared methods are

- CWK: $\mathcal{O}(k\, t_{\text{MAX}}\, |E|\, n)$,
- LP: $\mathcal{O}(k\, N_{\text{iter}}\, |E|)$,
- RL: $\mathcal{O}(k\, t_{\text{MAX}}\, N_{\text{iter}} |V_U| n)$,

- DIFF: $\mathcal{O}(n^3)$, and
- L+: $\mathcal{O}(n^3)$,

where k is the number of labels, $|E|$ is the number of edges, t_{MAX} is the maximum walk length, N_{iter} is the number of iterations with $t_{\text{MAX}} \leq N_{\text{iter}}$, and n is the total number of nodes in the graph. Note that both LP and CWK computations are very efficient for sparse graphs (small $|E|$) with few labeled nodes ($|V_L| \ll n$). Further, if CWK is used for node classification in a kernel machine, then we do not need to compute the full kernel matrix but it suffices to compute train–train and train–test fractions. If we use a sparse kernel machine, such as support vector machines, then it even suffices to compute (support vectors)–test fractions for predictions. So, compared to other methods the coinciding walk kernel computation for node label prediction scales favorably especially if the number of labeled nodes is small. Coinciding walk kernels have nice memory and computation complexities and can be computed faster than state-of-the-art kernels on graphs. These results support a positive answer of question (**Q7.3**).

TRANSITION: FROM NODE CLASSIFICATION TO SET COMPLETION

To summarize, probabilistic models based on random walks lifted to the node labels propagate information through the graph in order to label all unlabeled nodes. By leveraging graph structure, usually encoding some kind of similarity of adjacent vertices, they exploit the homophily assumption. However, as we have seen in the previous two chapters, approaches that assume only homophily have two problems. First, they rely on having sufficient label information within a small neighborhood of each unlabeled node. Due to the high costs of acquiring training data and the enormous size of present-day networks, however, it is common to have only very few labeled nodes. This results in having too few observations near many unlabeled nodes to effectively apply Hypothesis 2.1. Thus, when data are sparsely labeled, classification gets increasingly more difficult and we therefore have to do more than simply exploiting closeby labels to accurately classify unlabeled nodes. The second drawback of approaches entirely based on the homophily assumption is that they cannot make use of structure similarity and thus the similarity of distant or even disconnected nodes in a network based on their local graph structure is not exploitable for node label prediction. To overcome these limitations, we introduced a new kernel on graphs, the coinciding walk kernel, bringing together graph-based label inference and kernel methods to leverage benefits from both fields for learning tasks in sparsely labeled networks. The kernel values of CWKS are given by the probability that the labels encountered during parallel absorbing random walks on partially labeled graphs coincide. That is, two nodes have a high kernel value if the labels surrounding each node are arranged similarly. Our extensive experiments demonstrated that CWK, which takes both Hypotheses 2.1 and 6.1 into account, scales and is robust across all tested datasets for node classification in sparsely labeled graphs.

In addition to sparse training data, the objective – that is, the question of what it is we want to find in the data – might be only indirectly given. On a social network platform, a user might have liked only two or three brands and might not have clicked on any advertisements at all; nevertheless, it is of great interest for the provider of the platform that the *right* pages or products are recommended to the user or that the *right* ads are placed on the user's site. *Right* in this context could mean being likely to be clicked on. Hence, the task is to find suitable information in the data and/or to deduce a concept describing the query from only a small set of examples. Whereas we are still able to model items and the similarities among

them by nodes and edges of a graph, the learning setting is even more restrictive than having access to sparse label information. The first problem is that we have even less training data and the second is that the labels are positives only. So, it is less surprising that traditional graph-based supervised learning methods are powerless.

Structure similarity, however could be useful to find the hidden query concept exemplified by the query items or expand the query set. Unfortunately, there is no obvious way to use coinciding walk kernels to perform concept learning or solve the set completion task as even methods such as one-class support vector machines need a sufficient amount of positively labeled training nodes. So, instead of monitoring label distributions evolving from label-absorbing random walks one idea is to run random walks with restarts from the query nodes. Unfortunately, the ranking produced by random walks with restarts cannot capture structure similarity; all nodes close to the query nodes will be ranked high and distant nodes will be ranked low. To solve the set completion task, however, we want nodes to be ranked high if they are connected to the query nodes in the same way. One way of achieving this is to analyze the paths starting from the query nodes. To do so, we will change the way of modeling the data from plain graph representations to relational ones in order to leverage relational pathfinding enhancing the random walk-based methods. Our strategy will be to compare paths leaving from the query nodes and evolving on a graph representing relational data in order to find interesting nodes uncovering more details about the latent query. The idea behind this approach is to bring together graph-based learning, relational data representation, and structure similarity to tackle the challenging set completion task.

CHAPTER 8

BEYOND GRAPHS: RELATIONAL SET COMPLETION

In this chapter, we investigate the problem of dealing with extremely sparse training data comprising only a few positively labeled nodes in a graph, and the task of finding the most-similar nodes to these so-called query points. We call this task *set completion*, as we are concerned with the problem of retrieving items meeting the condition or concept implicitly posed by a set of query items. We will consider domains of relational data, and our goal in this chapter is to transfer the graph-based techniques applied in this thesis so far to this more-general data representation and demanding learning setting. In doing so, we will tackle set completion of the nodes in a graph by leveraging the power of information propagation and relational data representation.

8.1 Problem, Proposed Solution, and Related Work

If the node classification task as stated in Problem 6.1 is even more specialized in the sense that we only have positively labeled data points in the training set, then we speak of *unary* or *one-class classification* (SCHÖLKOPF, *et al.*, 2001*b*; KEMMLER, *et al.*, 2013; MANEVITZ and YOUSEF, 2001). The same problem is also referred to as *learning from positives only* (MUGGLETON, 1996; ELKAN and NOTO, 2008). From this perspective, node classification in sparsely labeled graphs can be reformulated as given a small set of positively labeled data, find the labeling of the unobserved data points without knowing any negative training examples. Another view on the same problem is information retrieval. Here, the problem is formulated as finding the most-similar items to some user-provided query items, where the importance lies on the quality of the set of retrieved items neglecting its completeness. We will view this task from the relational perspective and define it as set completion in relational data. Relational data and all relevant terminology was introduced in Section 2.5.

PROBLEM 8.1 (SET COMPLETION IN RELATIONAL DATA) *Given relational data* $\mathcal{D} = \{(C, P)\}$ *as a set of atoms on the universe of constants* C *and predicates* P*, and a query set* $\mathcal{Q} = \{q_i\}_{i=1,\ldots,m}$*, whose items are constants, predicates, or atoms, from* $\mathcal{D}$*, i.e.,* $q_i \in \mathcal{D}$*, and* $m \ll |\mathcal{D}|$*, set completion is the problem of retrieving a set* $\tilde{\mathcal{Q}} \supset \mathcal{Q}$ *where the items in* $\tilde{\mathcal{Q}}$ *are related to the query items.*

Set completion is also referred to as *clustering on demand*, which can be described from a clustering point of view as follows. Clustering on demand is the problem of grouping the data in such a way that data points similar to those given in the query form a cluster. Loosely speaking we cluster the data in such a way that the demands or restrictions posed by the user providing the query are met. Originally the problem has been addressed by "Google™ Sets," however, since September 2011 this service is not online anymore. "Google™ Sets" treated the clustering on demand problem as a large-scale clustering problem that involved millions of data instances extracted from web data. GHAHRAMANI and HELLER (2005) have proposed a Bayesian approach, called Bayesian sets (BSETS), for solving it. For a brief introduction of Bayesian learning in general, see Section 2.2.1. As in other retrieval problems, BSETS computes a score measuring for each item how relevant it is to the query. Specifically, using a probabilistic model of the unknown concept, it computes the relevance score by comparing the posterior probability of an item, given the query, to the prior probability of that item. A very similar approach to BSETS is Bayesian surprise, which is measured based on the Kulback–Leibler divergence to quantify the differences between prior and posterior distributions (ITTI and BALDI, 2005). Bayesian surprise, however, was used in signal processing to detect surprising events in videos, acoustic signals, or images (SCHAUERTE and STIEFELHAGEN, 2013; KITTLER, *et al.*, 2014). A similar measure applied to nodes in networked data is differential PageRank (DIFFPR) (HOTHO, *et al.*, 2006). DIFFPR produces a ranking among the nodes of a graph based on the difference of general importance and query relatedness. In fact, we will utilize DIFFPR in our proposed solution, and we will show in the next section why DIFFPR alone is not enough to solve Problem 8.1 satisfactory.

All mentioned approaches, including "Google™ Sets" and BSETS, have focused on propositional universes assuming that the items are representable as attribute–value vectors. Nowadays, the role of structure and relations in the data has becomes more and more important (GETOOR and TASKAR, 2007). Information about one item can help us to reach conclusions about other items. Such domains are hard to represent meaningfully using a set of propositional features. Thus, it is natural to ask the question: *how do we solve the clustering on demand problem for relational domains?* An investigation of this question leads to our main contribution: *Markov logic sets* (MLS).

Our objective is to retrieve complex objects and relations among them – that is, (ground) atoms from a logical concept – given a query consisting of a few atoms

from that concept. To leverage techniques from graph-based machine learning we formulate this task as a within-network relational learning problem using few labels only. Our proposed algorithm ranks atoms using a score based on random walks with restart (RWR), cf. Definition 2.13, namely the probability that a random surfer hits an atom starting from the query atoms. Specifically, we compute an initial ranking using personalized PageRank, a particular model of random walks with restart, cf. Section 2.3.2. Then, we find paths of atoms that are connected via their arguments, and variablize the ground atoms in each path, in order to create features for the query. These features are used to re-personalize the original RWR and to finally compute the set completion, based on label propagation (ZHU, *et al.*, 2003) also introduced in Section 2.3.2. MLS is related to relational pathfinding, which has been proven successful for learning the structure of Markov logic networks (MLNs) (RICHARDSON and DOMINGOS, 2006). For instance, MIHALKOVA and MOONEY (2007) as well as KOK and DOMINGOS (2009) proposed to use it to find a path of ground atoms in the training data. They also variablize ground atoms in the path but then they construct a MLN whose nodes are the paths viewed as Boolean variables (conjunctions of atoms). In contrast, MLS uses the features to compute a PageRank that generalizes well across similar items. Moreover, we exploit that random walk-based techniques can naturally exploit computational symmetries and show that lifted inference for label propagation is indeed possible.

Similar to the approach presented in (SILVA, *et al.*, 2010), MLS considers the clustering on demand problem as within-network relational learning problem: the input is a single data graph in which some of the instances are labeled and the task is to infer the labels of the remaining instances, see e.g. (NEVILLE and JENSEN, 2005) and references in there. Existing within-network relational learning methods indeed exploit the statistical dependencies among instances in order to improve classification performance. However, they assume that positive and negative labels are given for some nodes, and inference is generally still at the level of propositional logic. The latter also holds for the relational variants of BSETS (SILVA, *et al.*, 2010). For these approaches it is difficult – if not impossible – to employ existing inference techniques exploiting computational symmetries, termed *lifted inference*. Making use of computational symmetries of information propagation schemes is one of our goals here. Specifically, we will study lifted versions of PageRank and label propagation. Moreover existing clustering on demand and within-network relational learning approaches assume a single, finite feature space for all items; MLS does not since it employs RWR and learns query specific features automatically. LAO and COHEN (2010) present an approach to relational retrieval based on path-constrained RWR; MLS, however, constructs positive and negative labels from logical features and finally applies label propagation to derive a ranking based on the probabilities of being in the positive class. Recently, LIN and COHEN (2010*b*) have shown how to employ RWR for within-network relational learning using few positive and negative labels only; MLS assumes only positive labels. MUGGLETON (1996)

has proposed inductive logic programming (ILP) techniques for single-predicate learning from positive examples only; MLS solves a multi-predicate learning problem in a transductive setting. Probabilistic explanation based learning (KIMMIG, *et al.*, 2007) produces generalized explanations from examples. Similar to this approach but leveraging conceptual graphs similar to the (hyper)graphs used in MLS is graph-based relational concept learning (GONZALEZ, *et al.*, 2002). However, both approaches learn concepts to explain a query; MLS provides set completions, even if no concept/explanation is present. In the following, we will first introduce and illustrate the Markov logic sets algorithm, and then evaluate it empirically on a real-world relational dataset by finding completions of sets of objects describing the Roman city of Pompeii.

8.2 Markov Logic Sets

Now we can state the MLS algorithm and show how its main parts, namely PageRank and label propagation, can be lifted by exploiting computational symmetries.

8.2.1 Introduction

Imagine you catch parts of a conversation of two archaeologists: "... the room is a taberna ... City of Pompeii ..." What is the conversation about? Which concept is described by `function(taberna)` and `city(pompeii)`? "Taberna" is the Latin word for "shop", and Pompeii is the Roman city in southern Italy that was covered in volcanic ash during the eruption of Mt. Vesuvius in AD 79. So, the two archaeologists might have talked about the daily life of Romans in Pompeii and where they went shopping.

This example encapsulates the very practical and interesting machine learning and information retrieval problem, namely to automatically create sets or clusters of items from a few examples only. More formally, consider a universe of items $\mathcal{I}$. For instance, $\mathcal{I}$ may consist of publications, web pages, images, movies, places and artifacts found in Pompeii, or any other type of objects we may wish to form queries on. The user provides a query in the form of a very small subset of items $\mathcal{Q} \subset \mathcal{I}$ constituting examples of some concept in the data, say $\{$`function(taberna)`, `city(pompeii)`$\}$, which we will abbreviate in the following by $\{$`function(t)`, `city(p)`$\}$. The algorithm then has to provide a completion to the set $\mathcal{Q}$, that is, some set $\tilde{\mathcal{Q}} \subset \mathcal{I}$ that presumably includes all the elements in $\mathcal{Q}$ and other elements in $\mathcal{I}$, which are also elements of this concept, say $\{$`house`$(\mathtt{h}_1)$, `house`$(\mathtt{h}_2)$, `house`$(\mathtt{h}_4)$, $\ldots\}$ and other objects that are related to shops in Pompeii.

Defining a relevance score among nodes is one of the fundamental building blocks in graph-based machine learning. One very successful technique is based on random walks with restart (RWR). Concretely, Markov logic sets employ the seminal PageRank (PR) algorithm (BRIN and PAGE, 1998) as introduced in Section 2.3.2, which is the (limit) stationary probability that a random walker is at some node,

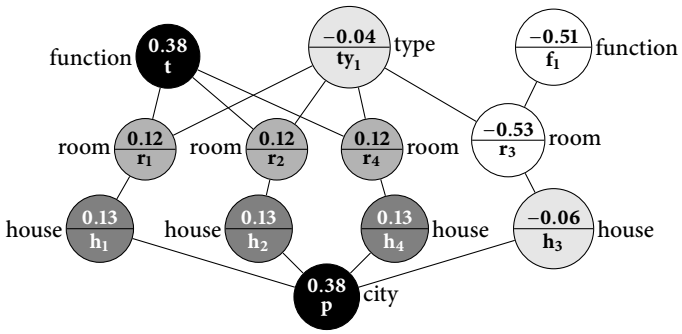


Figure 8.1: **Example MLS Ranking.** Ranking (rounded to the second digit) of Markov logic sets (MLS) for the query {`function(t)`, `city(p)`} on the toy version of the Pompeii dataset (Example 2.5). All houses with a shop are ranked high, whereas $\mathtt{h}_3$, which has no shop, is ranked low. The sign of the score actually suggests to exclude $\mathtt{h}_3$ from the final set completion.

but personalized on the query. Following the idea underlying BSETS, MLS actually computes the relevance score by comparing the *posterior PageRank* of an item given the query, i.e., the personalized PageRank, to the *prior PageRank* of that item, i.e., the uniform PageRank. This differential PageRank (DIFFPR) has been proven to be successful for ranking folksonomies (HOTHO, *et al.*, 2006) but has two major drawbacks for our setting:

1. it does not make use of the rich structure relational domains provide as it treats items which are ground atoms monolithically, and
2. it ranks items high that are generally surprising but unrelated to the query.

To generalize across items, we find paths of ground atoms that are connected via their arguments and variablize the ground atoms in each path, in order to create logical features, and use the features to compute a personalization vector for (DIFF)PR. Further, to overcome the second limitation of DIFFPR, we only estimate negative and positive labels for items ranked lowest and highest, respectively, by DIFFPR if they conform the features. Then, the labels are spread across the graph using label propagation. Hence, the name Markov logic sets originated from the use of *Markov* random walks as well as *logical* features to compute *set* completions.

Recall the conversation of the two archaeologists introduced in the beginning of this section. In the small toy universe considered, cf. Example 2.5, MLS discovers that they talked about *shopping places in Pompeii* and it perfectly retrieves houses and rooms used as shops as shown in Figure 8.1. A comparison of MLS to the

Table 8.1: **Comparison of MLS and DIFFPR.** Rankings found by Markov logic sets (MLS) and differential PageRank (DIFFPR) for the query $\{\texttt{function(t)}, \texttt{city(p)}\}$ on the toy version of the Pompeii dataset described in Example 2.5.

MLS		DIFFPR	
rank	item	rank	item
0.38	$\texttt{function(t)}$	1.00	$\texttt{function(t)}$
0.38	$\texttt{city(p)}$	0.64	$\texttt{house}(\texttt{h}_4)$
0.13	$\texttt{house}(\texttt{h}_4)$	0.64	$\texttt{house}(\texttt{h}_1)$
0.13	$\texttt{house}(\texttt{h}_2)$	0.64	$\texttt{house}(\texttt{h}_2)$
0.13	$\texttt{house}(\texttt{h}_1)$	0.63	$\texttt{city(p)}$
0.12	$\texttt{room}(\texttt{r}_4)$	0.63	$\texttt{function}(\texttt{f}_1)$
0.12	$\texttt{room}(\texttt{r}_1)$	0.52	$\texttt{house}(\texttt{h}_3)$
0.12	$\texttt{room}(\texttt{r}_2)$	0.44	$\texttt{room}(\texttt{r}_2)$
−0.04	$\texttt{type}(\texttt{ty}_1)$	0.44	$\texttt{room}(\texttt{r}_4)$
−0.06	$\texttt{house}(\texttt{h}_3)$	0.44	$\texttt{room}(\texttt{r}_1)$
−0.51	$\texttt{function}(\texttt{f}_1)$	0.04	$\texttt{type}(\texttt{ty}_1)$
−0.53	$\texttt{room}(\texttt{r}_3)$	0.00	$\texttt{room}(\texttt{r}_3)$

differential PageRank is given in Table 8.1. For MLS, all houses containing a shop are ranked high, whereas $\texttt{h}_3$, which has no shop, is ranked low. Actually the sign suggests to exclude $\texttt{h}_3$ from the final set completion. DIFFPR ranks $\texttt{h}_3$ and the type of its room $\texttt{ty}_1$ essentially on par with the shops, and the rooms that were used as shops too low. Next we will formally state the MLS algorithm and provide another illustrative example.

8.2.2 *MLS Algorithm*

Algorithm 7 summarizes MLS, and Figure 8.2 provides an example run with the query $\{\texttt{house}(\texttt{h}_1), \texttt{house}(\texttt{h}_2)\}$ for the toy universe of objects in the city of Pompeii described previously.

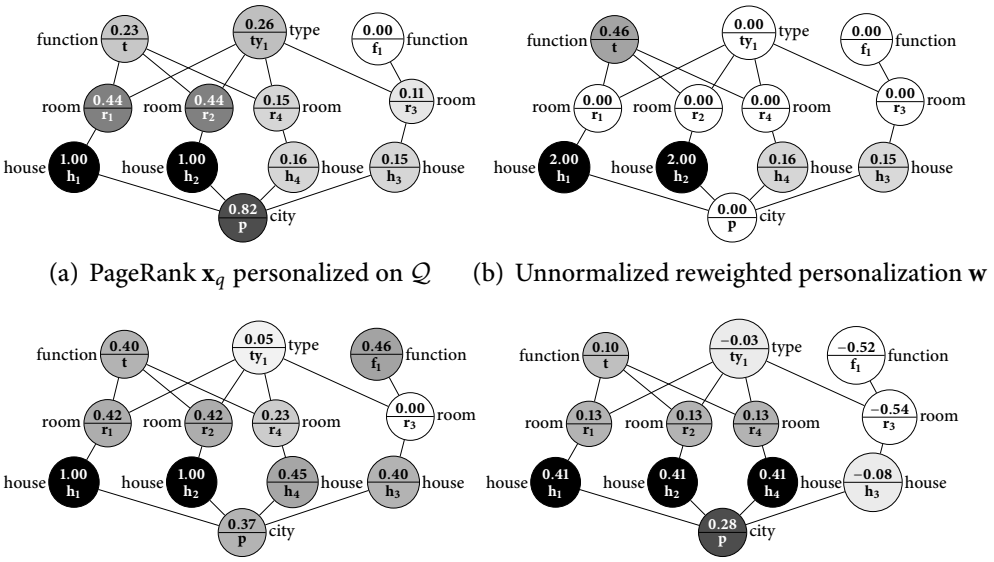
MLS begins with computing the PPR on the query Q. The result, see Figure 8.2(a), illustrates that this PPR is not enough to solve the set completion problem satisfactory. House $\texttt{h}_4$ should be in the set completion as it has a shop. Including also $\texttt{r}_4$, the room of $\texttt{h}_4$, also includes $\texttt{h}_3$, the only house without a shop. Excluding $\texttt{r}_4$, however, is suboptimal as it is the only room and the shop of $\texttt{h}_4$.

To improve upon this, MLS generalizes the personalization values across the network. This is done by the following procedure. First, MLS finds paths $\mathcal{P}$ of length starting from 0 up to k ($k = 2$ in our example) in the hypergraph starting in

Algorithm 7 Markov Logic Sets

given: Query set $\mathcal{Q}$, Graph $G = (V, E)$, max path length k
initialization: $V_+ = \mathcal{Q}$, $V_- = \varnothing$, $\mathcal{L} = \varnothing$ ▷ label and feature sets
$\mathbf{x}_q = \text{PPR}(G, \mathcal{Q})$ ▷ personalized PageRank on $\mathcal{Q}$
$\mathcal{P} \leftarrow$ COMPUTE-PATHS$(G, k, \mathcal{Q})$ ▷ paths starting in query atoms
$\mathcal{F} \leftarrow$ COMPUTE-PATHFEATURES$(\mathcal{P})$ ▷ variablizing some of the arguments
$\mathcal{L} \leftarrow$ COMPUTE-LOGICALFEATURES$(\mathcal{F})$ ▷ find discriminative features
for all $v \in V$ **do** ▷ reweight personalization
  $\mathbf{w}(v) = 0$ ▷ weight of item v
  for $\ell \in \mathcal{L}$ **do**
    if $\ell(end) = v$ **then**
      $\mathbf{w}(v) = \mathbf{w}(v) + \mathbf{x}_q(\ell(end))$
    end if
  end for
end for
$\mathbf{x}_p = \text{PPR}(G, \mathbf{w})$ ▷ reweighted personalized PageRank
$\mathbf{x}_u = \text{PR}(G)$ ▷ uniform PageRank
$\mathbf{x}_d = \mathbf{x}_p - \mathbf{x}_u$ ▷ differential PageRank
for all $v \in V$ **do** ▷ construct label set
  for $i \in \{p, d\}$ **do**
    if $|\mathbf{x}_i(v) - \min \mathbf{x}_i| < \varepsilon$ & no ground feature fired for v **then**
      $V_- \leftarrow v$
    else if $|\mathbf{x}_i(v) - \max\{\mathbf{x}_i(u) | u \notin \mathcal{Q}\}| < \varepsilon$ & a feature fired for v **then**
      $V_+ \leftarrow v$
    end if
  end for
end for
$\mathbf{r} = \text{LABEL-PROPAGATION}(G, V_+, V_-)$ ▷ label propagation
return: ranking of positive label probabilites of $\mathbf{r}$

the query nodes (COMPUTE-PATHS$(G, k, \mathcal{Q})$), where a path is defined as a set of hypernodes and hyperedges such that for any two hyperedges e_0 and e_n in the set, there exists an ordering of (a subset of) hyperedges in the set $e_0, e_1, \ldots, e_n$ such that e_i and e_{i+1} share at least one node. The path length is the number of hyperedges it contains. Hence, length-0 paths do only contain one hypernode and no hyperedges. In our toy example a path of length 2 is for example p = "$\texttt{house}(\texttt{h}_1)$ – $\texttt{in}(\texttt{r}_1, \texttt{h}_1)$ – $\texttt{room}(\texttt{r}_1)$ – $\texttt{typeof}(\texttt{r}_1, \texttt{t})$ – $\texttt{function}(\texttt{t})$". The paths of hyperedges are then generalized into pathfeatures $\mathcal{F}$ (COMPUTE-PATHFEATURES$(\mathcal{P})$). Pathfeatures are conjunctions of relational atoms with variablized arguments. Therefore, we first variablize all arguments except the atoms in the start and end nodes, e.g.



(a) PageRank $\mathbf{x}_q$ personalized on $\mathcal{Q}$ (b) Unnormalized reweighted personalization $\mathbf{w}$

(c) Differential PageRank $\mathbf{x}_d = \mathbf{x}_p - \mathbf{x}_u$ (d) Final label propagation ranking $\mathbf{r}$

Figure 8.2: **Illustration of Markov Logic Sets.** The major steps of running Markov logic sets for the query $\{\texttt{house}(\texttt{h}_1), \texttt{house}(\texttt{h}_2)\}$ on the toy version of the Pompeii dataset, which is described in the main text. The final ranking is shown in (d). Using 0 as threshold would result in $\{\texttt{h}_1, \texttt{h}_2, \texttt{h}_4, \texttt{p}, \texttt{r}_1, \texttt{r}_2, \texttt{r}_4, \texttt{t}\}$ as set completion.

"$\texttt{house}(\texttt{h}_1)$ – $\texttt{in}(\texttt{Y},\texttt{X})$ – $\texttt{room}(\texttt{Y})$ – $\texttt{typeof}(\texttt{Y},\texttt{t})$ – $\texttt{function}(\texttt{t})$". To create candidate logical features per path the constants in the start and end nodes are then variablized in all possible combinations, for the example path above, one of the four possible pathfeatures is: f = "$\texttt{house}(\texttt{X})$ – $\texttt{in}(\texttt{Y},\texttt{X})$ – $\texttt{room}(\texttt{Y})$ – $\texttt{typeof}(\texttt{Y},\texttt{t})$ – $\texttt{function}(\texttt{t})$", where the constant in the start node $\texttt{h}_1$ is variablized. However, MLS selects only discriminative features, called logical features $\mathcal{L}$, (COMPUTELOGICALFEATURES($\mathcal{F}$)). To assess whether a feature is discriminative, we run a PR personalized on end atoms and check whether

1. the variance of the PPR scores of the query items covered by the feature is small, and
2. their mean is significantly different from the mean of PPR scores of items over the same predicate that are not in the query.

Only if (1) and (2) succeed using a χ^2-test, do we include the feature; this applies for example to f.

Afterwards, MLS recomputes the personalization of each "interesting" node in G using the PPR on $\mathcal{Q}$, $\mathbf{x}_q$, of the nodes generating the respective logical pathfeatures. Interesting are all nodes that are end nodes $\ell(end)$ in the logical pathfeatures. And the respective personalization weight per feature is then $\mathbf{x}_q(\ell(end))$. In our running example the set of end nodes is $\ell(end)$ = {`house(X)`, `function(Y)`, `function(t)`}. Intuitively, nodes that are structurally related to all query items in the same way get assigned a personalization weight. For our running example, this yields the unnormalized personalization vector shown in Figure 8.2(b). As one can see, the weight of item `t` (taberna) is increased from 0 (it is not a query member, consequently its initial personalization was 0) to 0.46. One is tempted to just recompute the PPR using this new personalization, $\mathbf{x}_p$, and indeed items with resulting minimal rank are definitely unrelated to the query – therefore we give them a negative label (cf. construct label set below) but only if no feature fired for the item – and `t` (taberna) gets ranked higher now. However, $\mathtt{h}_3$ and $\mathtt{h}_4$ still get similar, low relevance scores: the concept "houses with shops" has not been identified yet. Why is this the case?

Ranking items simply by the (reweighted) personalized PR is not sensible since some items may be more probable than others, regardless of the query. MLS removes this effect by computing the differential PR (DIFFPR), $\mathbf{x}_d$, w.r.t. the uniform PR, $\mathbf{x}_u$. The DIFFPR scores for our running example are shown in Figure 8.2(c). Now, it is sensible to assign positive (negative) labels to the items with maximal (minimal) $\mathbf{x}_p$ or $\mathbf{x}_d$, if features have (not) fired. For our example, the set of positively labeled nodes is $V_+ = \{\mathtt{h}_1, \mathtt{h}_2, \mathtt{h}_3\}$ and the set of negatively labeled nodes is $V_+ = \{\mathtt{f}_1, \mathtt{r}_3\}$. Finally, MLS runs label propagation, cf. Equation (2.11), using V_+ and V_- to achieve specialization, that is, to create the final ranking (LABELPROPAGATION(G, V_+, V_-)). For our toy example this final ranking is shown in Figure 8.2(d). As one can see, $\mathtt{h}_4$ and $\mathtt{h}_3$ can now clearly be separated.

8.2.3 Exploiting Symmetries: Lifted Information Retrieval

However, we make another important and novel contribution. We note that the core computations in both PageRank and label propagation, namely solving systems of linear equations, can be done distributively and efficiently using Gaussian belief propagation (GABP) (SHENTAL, *et al.*, 2008). Gaussian belief propagation extends the message passing algorithm (loopy) belief propagation (BP) (YEDIDIA, *et al.*, 2003) from binary to Gaussian random variables. There are, however, many improvements that can be made to (GA)BP, especially when applied to inference in graphical models containing structural symmetries. Such symmetries are commonly found in relational domains (AHMADI, 2014). To exploit these symmetries, lifted (loopy) belief propagation (LBP) approaches have been recently developed (SINGLA and DOMINGOS, 2008; KERSTING, *et al.*, 2009) to automatically group nodes and potentials together into supernodes and superpotentials if they

send and receive identical messages. LBP then runs a modified BP on this compressed network, also referred to as *lifted network*. LBP has been proven to yield significant efficiency gains compared to belief propagation on important AI tasks such as link prediction and entity resolution. Here, we demonstrate its potential for another important AI and novel lifted inference task: information retrieval using PageRank and label propagation. All core computations in both PageRank and label propagation can be solved by combining the GABP approach to solving linear systems of SHENTAL, *et al.* (2008) together with the color passing approach LBP of KERSTING, *et al.* (2009), yielding lifted GABP (LGABP). This approach was introduced in (AHMADI, *et al.*, 2011). Here, we present its first application to the information retrieval task of relational set completion.

8.3 Empirical Evaluation

Our intention here is to investigate the following questions:

(Q8.1) Is MLS effective in ranking atoms with respect to a given query?

(Q8.2) How does MLS perform compared to BSETS and the intermediate rankings $\mathbf{x}_p$ and $\mathbf{x}_d$ of MLS?

(Q8.3) Are there symmetries in the label propagation matrix that can be used by LGABP?

To this aim, we implemented MLS and BSETS in Python using LGABP for the PageRank and label propagation computations. In all experiments, we set the restart probability $\beta = 0.5$ for the RWR computations. For (L)GABP, we used the "flooding" message protocol where messages are passed in parallel each step. Messages were not damped, and the convergence threshold was 10^{-8}. We did not compare to the relational BSETS variant of SILVA, *et al.* (2010) as they assume a single joint feature space. To accommodate for this, we ran BSETS (with its parameter $c = 2$) using our feature sets $\mathcal{F}$ resp. $\mathcal{L}$. More precisely, $\text{BSETS}_{\mathcal{F}}$ uses all possible relational pathfeatures, whereas $\text{BSETS}_{\mathcal{L}}$ includes only the discriminative features found by MLS. For evaluation purposes, we ran all approaches on two datasets: a relational datasets in the spirit of the smoker–friends Markov logic network and the Pompeii dataset introduced in (ALLISON, 2001) and mentioned already earlier. Allison has used data about artifacts in their original context to challenge many common assumptions about the function of particular types of objects, the use of particular spaces, and the consistency of patterns of use across different houses. The dataset contains more than 6 000 artifact records for finds in 30 architecturally similar "atrium-style" houses in Pompeii. Artifacts are annotated with one of 240 typological categories (coin, amphora, etc.) and in which of the rooms they were found. In total, there are 863 rooms in the 30 houses. For each room, we have its type (main garden, front hall, shop, etc.) and function (culina, taberna, etc.). We have focused on the rooms and houses providing us with a dataset consisting of 959

node potentials and 6063 edge potentials (viewed as a graphical model as used by (L)GABP). Table 8.2 shows the summarized results.

The first query consisted of 4 of the in total 8 houses with shops. As one can see the rankings/completions of $\mathbf{x}_p$, $\mathbf{x}_d$, and BSETS$_{\mathcal{F}}$ do not cover the set of houses with shops and the query relevant information. BSETS$_{\mathcal{L}}$ and MLS, however, are able to retrieve the houses with shops, the respective rooms, and type or function information. In other words, they have discovered the concept "houses with shops". Other houses are also ranked high, since one of the discriminative features is `house(X)`. By just removing one house from the query, as done in $\mathcal{Q}_2$, the concept "houses with shops" is not clearly identified by any method. MLS, though, provides query-relevant information such as common discriminative types and functions of the query houses (`function(t)`, among others). Apart from completing the set of all houses, it also retrieves `city(p)`. The intermediate MLS rankings fail to return reasonable information as for example other houses. The third query, $\mathcal{Q}_3$, consisted of 5 of the in total 20 rooms in $\texttt{house}(\texttt{h}_1)$. We show the non-compressed top-30 ranked items for this query in Table 8.3. The rankings illustrate that MLS is able to capture all relevant information, by ranking $\texttt{house}(\texttt{h}_1)$ the highest followed by all its rooms, the types and functions of the query atoms and `city(p)`. $\mathbf{x}_d$ also retrieves all rooms of $\texttt{house}(\texttt{h}_1)$; however, it then fails to rank more relevant room functions and types fitting the query rooms. BSETS as well as $\mathbf{x}_p$ also provide only parts of the information, but no complete summary. The results for query $\mathcal{Q}_4$ on the smokers–friends network are shown in Table 8.4. For this dataset the results are qualitatively similar. In particular, all elements of clique a are ranked highest only by MLS and BSETS$_{\mathcal{L}}$.

All together, this clearly shows that **(Q8.1)** MLS finds reasonable set completions and that **(Q8.2)** its performance is better than its intermediate rankings and as good as or even better than BSETS using relational features. The difference in performances of the two BSETS variants demonstrate that discriminative relational pathfeatures indeed carry useful information. The compression ratios of the label propagation matrices – ratio of (# nodes + # edges) lifted vs. ground – were 0.55 for $\mathcal{Q}_1$ – $\mathcal{Q}_3$ and 0.73 for $\mathcal{Q}_4$. This means that for example for query $\mathcal{Q}_1$ instead of 959 variables and 6 063 factors the lifted network had 567 variables and 3 305 factors. These compression rates show that lifted inference for label propagation is possible and sensible. **(Q8.3)** can be answered affirmatively. Note that GABP and LGABP produce the same results.

8.4 Summary and Discussion

In conclusion, *Markov logic sets* is the first, generally applicable, retrieval framework for relational data that can naturally be lifted by exploiting computational

Table 8.2: **Results on Pompeii.** Top-40 ranked unary ground atoms produced by the intermediate rankings of MLS $\mathbf{x}_p$ (PPR) and $\mathbf{x}_d$ (DIFFPR), BSETS$_{\mathcal{F}}$, BSETS$_{\mathcal{L}}$ and MLS for three different queries $\mathcal{Q}_1$–$\mathcal{Q}_3$ on the Pompeii dataset. The Pompeii dataset consists of rooms in 30 Pompeian households. The rankings are shown using a compressed, almost natural language like representation. E.g. "4 query houses, 1 function, city(p), ..." denotes a ranking in which the four houses in the query are ranked first, followed by a ground atom describing a function of a room, followed by the ground atom city(p), and so on.

$\mathcal{Q}_1 = \{\mathbf{house}(\mathbf{h_2}), \mathbf{house}(\mathbf{h_5}), \mathbf{house}(\mathbf{h_9}), \mathbf{house}(\mathbf{h_{12}})\}$	
$\mathbf{x}_p$ (PPR)	4 query houses, 1 function, city(p), 3 types, 2 functions, 13 rooms of $\mathtt{h}_2$, 2 types, 1 function, 13 rooms of $\mathtt{h}_5$
$\mathbf{x}_d$ (DIIFPR)	4 query houses, 13 rooms of $\mathtt{h}_2$, 20 rooms of $\mathtt{h}_5$, 3 rooms of $\mathtt{h}_{12}$, 4 rooms of $\mathtt{h}_9$
BSETS$_{\mathcal{F}}$	all 30 houses, 1 rooms of $\mathtt{h}_2$, 4 rooms of $\mathtt{h}_5$, 2 rooms of $\mathtt{h}_{12}$, 3 types
BSETS$_{\mathcal{L}}$	8 houses with shops, 11 rooms (shops), function(t), typeof($\mathtt{ty}_{20}$), 19 other houses
MLS	typeof($\mathtt{ty}_{20}$), function(t), 8 houses with shops, 22 other houses, city(p), 7 rooms (shops)
$\mathcal{Q}_2 = \{\mathbf{house}(\mathbf{h_2}), \mathbf{house}(\mathbf{h_5}), \mathbf{house}(\mathbf{h_9})\}$	
$\mathbf{x}_p$ (PPR)	3 query houses, 1 function, city(p), 3 types, 13 rooms of $\mathtt{h}_2$, 1 function, 1 type, 17 rooms of $\mathtt{h}_5$
$\mathbf{x}_d$ (DIFFPR)	3 query houses, 13 rooms of $\mathtt{h}_2$, 20 rooms of $\mathtt{h}_5$, 4 rooms of $\mathtt{h}_9$
BSETS$_{\mathcal{F}}$	3 query houses, all remaining 27 houses, 8 rooms of $\mathtt{h}_2$ resp. $\mathtt{h}_5$, 2 types
BSETS$_{\mathcal{L}}$	3 query houses, all remaining 27 houses, 10 rooms
MLS	3 types and 3 functions of rooms appearing in the query houses, all 30 houses, city(p), 3 rooms
$\mathcal{Q}_3 = \{\mathbf{room}(\mathbf{r_{1e}}), \mathbf{room}(\mathbf{r_{1r}}), \mathbf{room}(\mathbf{r_{1h}}), \mathbf{room}(\mathbf{r_{1f}}), \mathbf{room}(\mathbf{r_{1o}})\}$	
$\mathbf{x}_p$ (PPR)	house($\mathtt{h}_1$), 5 query rooms, 4 functions, 5 types, city(p), 15 rooms of $\mathtt{h}_1$, 6 houses, 2 types, 1 function
$\mathbf{x}_d$ (DIFFPR)	house($\mathtt{h}_1$), 5 query rooms, 15 rooms of $\mathtt{h}_1$, 1 type, 15 functions, 5 other rooms
BSETS$_{\mathcal{F}}$	5 query rooms, 15 rooms of $\mathtt{h}_1$, 20 other rooms
BSETS$_{\mathcal{L}}$	20 rooms of $\mathtt{h}_1$, city(p), house($\mathtt{h}_1$), 18 other rooms
MLS	house($\mathtt{h}_1$), 5 query rooms, 15 rooms of $\mathtt{h}_1$, 3 types resp. functions of rooms in $\mathcal{Q}_3$, city(p), 12 other rooms

Table 8.3: **Full Results on Pompeii.** Top-30 ranked unary ground atoms on Pompeii of query $\mathcal{Q}_3$.

$\mathcal{Q}_3 = \{\mathbf{room}(\mathbf{r_{1e}}), \mathbf{room}(\mathbf{r_{1r}}), \mathbf{room}(\mathbf{r_{1h}}), \mathbf{room}(\mathbf{r_{1f}}), \mathbf{room}(\mathbf{r_{1o}})\}$									
$\mathbf{x}_p$ (PPR)		$\mathbf{x}_d$ (DIFFPR)		MLS		BSETS_$\mathcal{F}$		BSETS_$\mathcal{L}$	
1.0	house(h_1)	1.0	house(h_1)	0.7168	house(h_1)	9.33	room(r_{1e})	6.03	room(r_{1a})
0.7219	room(r_{1e})	0.8831	room(r_{1e})	0.5486	room(r_{1e})	9.21	room(r_{1f})	6.03	room(r_{1g})
0.7214	room(r_{1f})	0.8828	room(r_{1f})	0.5484	room(r_{1f})	7.6	room(r_{1g})	6.03	room(r_{1e})
0.7163	room(r_{1r})	0.8798	room(r_{1r})	0.5466	room(r_{1r})	7.6	room(r_{1r})	6.03	room(r_{1s1})
0.716	room(r_{1o})	0.8797	room(r_{1o})	0.5346	room(r_{1o})	7.55	room(r_{1o})	6.03	room(r_{1i})
0.7152	room(r_{1h})	0.8792	room(r_{1h})	0.5343	room(r_{1h})	7.55	room(r_{1q})	6.03	room(r_{1r})
0.3461	fctn(cubiculum)	0.4392	room(r_{1g})	0.0555	room(r_{1ST})	7.0	room(r_{1i})	6.03	room(r_{1o})
0.185	type(ty_4)	0.4391	room(r_{1q})	0.0554	room(r_{1m})	7.0	room(r_{1d})	6.03	room(r_{1p})
0.1729	fctn(fauces)	0.4388	room(r_{1l})	0.0546	room(r_{1s1})	7.0	room(r_{1h})	6.03	room(r_{1m})
0.1586	type(ty_{12})	0.4388	room(r_{1i})	0.0545	room(r_{1s})	7.0	room(r_{1l})	6.03	room(r_{1c})
0.158	fctn(tablinum)	0.4388	room(r_{1d})	0.0466	room(r_{1g})	5.32	room(r_{1a})	6.03	room(r_{1f})
0.158	type(ty_7)	0.4388	room(r_{1m})	0.0459	room(r_{1a})	4.76	room(r_{1s})	6.03	room(r_{1ST})
0.1579	type(ty_5)	0.4386	room(r_{1s1})	0.0453	room(r_{1p})	4.76	room(r_{1ST})	6.03	room(r_{1UF})
0.1579	fctn(ala)	0.4382	room(r_{1ST})	0.0453	room(r_{1b})	4.74	room(r_{1m})	6.03	room(r_{1d})
0.1577	type(ty_8)	0.4382	room(r_{1s})	0.0453	room(r_{1c})	4.71	room(r_{1s1})	6.03	room(r_{1h})
0.0367	city(pompeii)	0.4382	room(r_{1a})	0.0453	room(r_{1n})	3.47	room(r_{1b})	6.03	room(r_{1n})
0.0342	room(r_{1g})	0.4374	room(r_{1c})	0.0453	room(r_{1UF})	3.45	room(r_{1n})	6.03	room(r_{1s})
0.034	room(r_{1q})	0.4373	room(r_{1p})	0.0346	room(r_{1q})	3.44	room(r_{1UF})	6.03	room(r_{1q})
0.0333	room(r_{1i})	0.4373	room(r_{1n})	0.0343	room(r_{1l})	3.44	room(r_{1p})	6.03	room(r_{1l})
0.0333	room(r_{1l})	0.4373	room(r_{1UF})	0.0343	room(r_{1i})	3.43	room(r_{1c})	6.03	room(r_{1b})
0.0333	room(r_{1d})	0.4372	room(r_{1b})	0.0343	room(r_{1d})	3.36	room(r_{22F})	3.42	city(pompeii)
0.0327	room(r_{1a})	0.4278	type(ty_5)	0.0331	fctn(ala)	3.36	room(r_{6e})	3.42	house(h_1)
0.0316	room(r_{1s})	0.4278	fctn(ala)	0.0331	type(ty_5)	3.36	room(r_{1102})	1.33	room(r_{14UF})
0.0315	room(r_{1p})	0.4223	fctn(peristylum_ambulatio)	0.0321	type(ty_7)	3.36	room(r_{8r})	1.33	room(r_{15A})
0.0314	room(r_{1b})	0.4223	fctn(oecus_triclinium)	0.0321	fctn(tablinum)	3.36	room(r_{1206})	1.33	room(r_{15V})
0.0314	room(r_{1ST})	0.4223	fctn(cubiculum_ala)	0.0234	type(ty_8)	3.36	room(r_{14m})	1.33	room(r_{150})
0.0314	room(r_{1c})	0.4223	fctn(peristylum_viridarium)	0.0203	fctn(fauces)	3.36	room(r_{26q})	1.33	room(r_{15R})
0.0314	room(r_{1n})	0.4223	fctn(fauces_androne s)	0.0107	city(pompeii)	3.36	room(r_{26h})	1.33	room(r_{15P})
0.0314	room(r_{1UF})	0.4223	fctn(oecus_exedra)	0.0036	type(ty_{17})	3.35	room(r_{23I})	1.33	room($r_{15E'}$)
0.0313	room(r_{1m})	0.4223	fctn(atrium_vestibulum)	0.0033	room(r_{15D})	3.35	room(r_{3006})	1.33	room($r_{15L'}$)

Table 8.4: **Results on Smokers-Friends.** Top-20 ranked unary ground atoms produced by the intermediate rankings of MLS $\mathbf{x}_p$ (PPR) and $\mathbf{x}_d$ (DIFFPR), BSETS$_{\mathcal{F}}$, BSETS$_{\mathcal{L}}$ and MLS for query $\mathcal{Q}_4$ consisting of 3 persons from the same clique (clique a) on the smokers dataset. The smokers dataset is composed of 20 persons of which 9 smoke and 5 have cancer organized in 4 friendship cliques {a, b, c, d} with few inter-clique links. The subscripts denote the ith person in a clique. Items of clique a are highlighted in gray.

$\mathcal{Q}_4 = \{\mathtt{p}(\mathtt{a}_1), \mathtt{p}(\mathtt{a}_3), \mathtt{p}(\mathtt{a}_4)\}$				
$\mathbf{x}_p$ (PPR)	$\mathbf{x}_d$ (DIFFPR)	BSETS$_{\mathcal{F}}$	BSETS$_{\mathcal{L}}$	MLS
p(a$_4$)	p(a$_3$)	p(a$_5$)	p(a$_1$)	p(a$_5$)
p(a$_1$)	p(a$_1$)	p(a$_3$)	p(a$_4$)	p(a$_3$)
p(a$_3$)	s(a$_2$)	s(a$_3$)	p(a$_2$)	p(a$_4$)
p(a$_2$)	s(a$_1$)	s(a$_4$)	p(a$_3$)	p(a$_1$)
p(a$_5$)	s(a$_4$)	s(a$_2$)	p(a$_5$)	s(a$_2$)
p(b$_5$)	c(a$_2$)	s(a$_1$)	s(a$_4$)	c(a$_2$)
p(d$_2$)	c(a$_4$)	s(b$_2$)	s(a$_1$)	p(a$_2$)
c(a$_2$)	p(a$_5$)	s(b$_4$)	s(a$_2$)	s(a$_3$)
s(a$_2$)	s(a$_3$)	s(b$_1$)	s(a$_3$)	s(a$_1$)
s(a$_3$)	p(a$_4$)	s(b$_5$)	c(a$_2$)	s(a$_4$)
p(d$_1$)	s(b$_5$)	s(d$_2$)	c(a$_4$)	c(a$_4$)
p(d$_3$)	c(b$_5$)	p(a$_1$)	c(b$_1$)	p(d$_2$)
p(b$_2$)	s(d$_2$)	p(a$_2$)	s(b$_1$)	s(d$_2$)
p(c$_4$)	s(b$_2$)	c(a$_2$)	s(b$_2$)	p(d$_4$)
p(d$_4$)	c(d$_1$)	c(a$_4$)	s(b$_4$)	p(d$_5$)
p(d$_5$)	c(b$_1$)	c(b$_1$)	s(b$_5$)	p(d$_1$)
p(b$_1$)	s(b$_1$)	c(b$_5$)	c(b$_5$)	c(d$_1$)
p(b$_4$)	s(b$_4$)	c(d$_1$)	s(d$_2$)	p(c$_4$)
p(b$_3$)	p(b$_3$)	p(a$_4$)	c(d$_1$)	p(d$_3$)
p(c$_2$)	p(d$_5$)	p(d$_5$)	p(b$_5$)	p(c$_3$)

symmetries.[1] It takes a query consisting of a small set of ground atoms and returns additional ground atoms that belong in this set. Specifically, it computes a relevance score for each ground atom by comparing the personalized PageRank for it given the query to the unpersonalized PageRank of that ground atom. Since

[1]We attempt to develop a pragmatic and general framework that is liftable; at the moment, we make no claim that lifted techniques necessarily run faster than a specialized, one-off solution.

similar items should intuitively get similar relevance scores, MLS uses relational pathfinding to find generalized personalization vectors. Based on the relevance, positive and negative labels are constructed that are turned in the final ranking using label propagation on the lifted network. The experimental results demonstrate that the networks used can be compressed and thus, lifted retrieval is not insurmountable: MLS performs effectively in retrieving relational set completions.

Part III

Beyond Molecules: Novel Applications for Learning with Graphs

Learning with graphs has numerous application areas. As we have seen in the introduction and previous parts, the variety reaches from the classification of chemical structures, over social network analysis and reasoning in web data, knowledge graphs, or citation networks, to predicting usage frequencies in traffic and communication networks, or massive online gaming universes. However, most of these domains are networked data – with one exception: databases of molecules. Molecules are atoms connected by bonds, which can be elegantly modeled as graphs. Other types of structured data used in machine learning are sets, strings, and trees. Among these structures, graphs have the most expressive form and thus are one of the most challenging and interesting to work with. Most algorithms for predictive graph mining are, however, designed for or exclusively evaluated on the classification of molecules or proteins. Only recently have other applications been studied, for instance the analysis of graphs generated from FMRI data (GKIRTZOU, *et al.*, 2013).

Problems in bioinformatics such as drug design are of high practical and scientific relevance. Combined with easy access to benchmark data, this is a strong reason for the popularity of evaluating methods such as graph kernels on molecule data. Another reason is that the molecule graphs are "nice" graphs in the sense that they are fully annotated without uncertainty in the atom and bond labels and there exist standard ways of how to represented chemical compounds as graphs. In addition, the available benchmark datasets either comprise many small graphs (such as NCI) or smaller sets of possibly larger graphs (such as D&D), cf. Table 4.1. In the following, we will move beyond these "nice" learning settings and introduce two exciting and important applications in the fields of robotic grasping and computer vision that have not been addressed by learning with graphs. Here, the graphs are typically only partially observed, they can be huge, and nodes and edges are naturally endowed with continuous information. We will show how the challenges arising in these domains can be tackled with the techniques developed throughout this thesis. Specifically, we will apply propagation kernels to 3D object category prediction and image-based plant disease classification. Further, we will demonstrate the use of coinciding walk kernels for image-based leaf spot detection.

In fact, kernels from propagated information are mainly based on easily interchangeable propagation schemes. Due to this flexibility and their efficiency, they can be applied to large graph databases with missing and uncertain node information, and thereby to novel domains beyond databases of molecules. First, propagation kernels are able to compare larger graphs with reasonable time expended, opening up the use of graph kernels for object category prediction of 3D point clouds in the context of robotic grasping. Point clouds of household objects, for example, consist of partially observed curvature and part information, and depending on their size and the perception distance they can easily consist of several thousands of nodes. Traditional graph kernels suffer from enormous computation times or memory problems even for medium-sized datasets, rendering their use

for this task problematic. These issues are aggravated even more when considering image data. So far, graph kernels have been used for image classification on the scene level where the nodes comprise segments of similar pixels and one image is then represented by less than 100 so-called superpixels (HARCHAOUI and BACH, 2007). Utilizing off-the-shelf techniques for efficient diffusion on grid graphs as introduced in this thesis allows the use of kernels from propagated information to analyze images on the pixel level and thus opens up a whole area of interesting problems in the intersection of graph-based machine learning and computer vision. As a first step, we apply coinciding walk kernels and propagation kernels in the context of image-based plant disease detection and classification, where we consider a dataset of thousands of graphs containing a total amount of several millions of nodes.

CHAPTER 9

3D-OBJECT CLASSIFICATION

Grasping is a critical and difficult problem in robotics. The problem of simply finding a stable grasp is difficult enough, but to perform a useful grasp, we must also consider other aspects of the grasping task: the object, its properties, and any task-related constraints. The choice of grasping region is highly dependent on the category of an object. Thus, we will study the automated prediction of object categories of a large variety of household objects such as cups, pots, pans, bottles, and various tools. Category prediction is achieved by employing propagation kernels between point cloud graphs measuring the similarity of objects based on their manifold information and semantic parts. Our work highlights the importance of moving towards the use of machine learning approaches for structured data in order to achieve the dream of autonomous and intelligent robot grasping: learning to map low-level visual features to good grasping points under consideration of object–task affordances and high-level world knowledge. We evaluate propagation kernels for object category prediction on a (synthetic) dataset of 41 objects with 11 categories and a (more realistic) dataset of 126 point clouds derived from laser range data with part labels estimated by a part detector. Finally, we point out the benefit of leveraging kernel-based object category distributions for task-dependent robot grasping.

9.1 Problem Introduction and Related Work

In the following, we present a novel application area for learning with graphs: vision-based robotic grasping. Objects can be grasped in different ways. For manipulation tasks in arbitrary and dynamic environments, it is essential to perform a *good* grasp. That is, the grasp depends on the specific manipulation scenario: the task, the object, its properties, as well as grasp constraints (e.g., the gripper configuration). Thus, autonomous and intelligent robot grasping heavily relies on reasoning about world knowledge, in the form of object ontologies and object–task

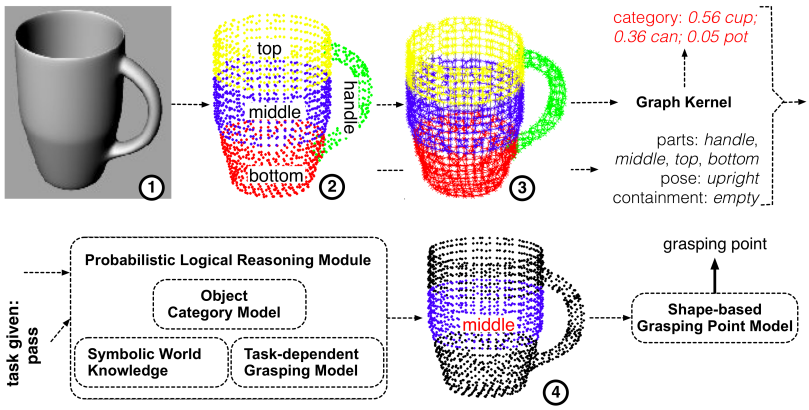
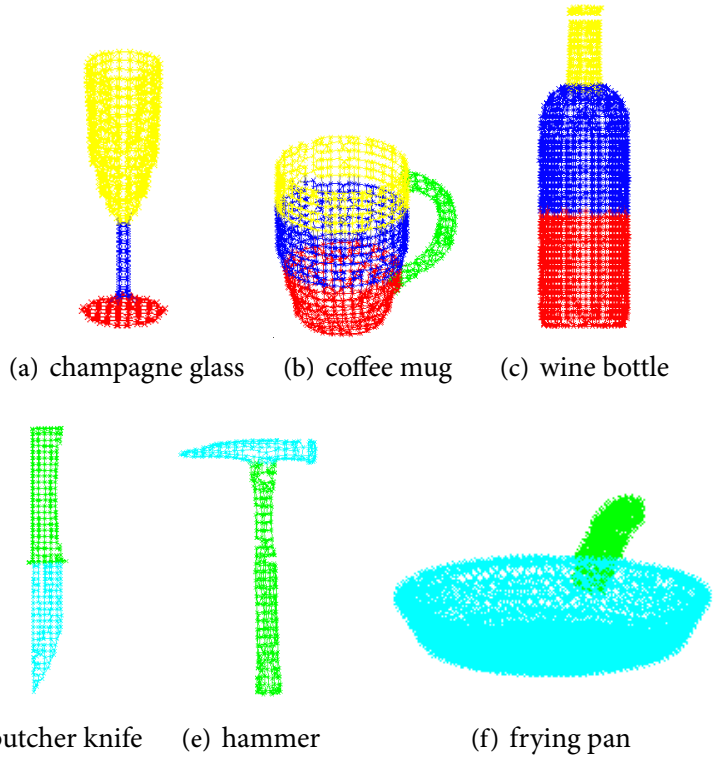


Figure 9.1: **Task-Dependent Grasping Pipeline.** Top row: ① object (cup); ② pose and symbolic parts (with labels *top* (yellow), *middle* (blue), *bottom* (red), and *handle* (green)); ③ k-nn graph[1] with part labels; graph kernel-based category distribution. Bottom row: probabilistic logical module; ④ predicted pre-grasp (*middle*) and grasping point module.

affordances, visual features, as well as object categorical and task-based information.

Consequently, it is difficult – if not impossible – to learn an unstructured model that directly maps visual perceptions to task-dependent grasps, as for instance introduced in (BOHG and KRAGIC, 2010; MONTESANO and LOPES, 2012; LENZ, *et al.*, 2013; SAXENA, *et al.*, 2008). This may be especially the case if the robot acts in highly dynamical real-world environments handling a huge range of objects having different categories, functionalities and task affordances, such as objects found in households, supermarkets, or industrial sites. Instead, we investigate an intermediary probabilistic logical module to semantically reason about the most likely object part to be grasped, given the scene description, object category, and task constraints. Then, a mapping is learned from part-related local visual features to good grasping points. We propose a probabilistic logical pipeline, illustrated in Figure 9.1, exploiting graph kernels, object and task ontologies, semantic reasoning, as well as perceptual low-level learning. By leveraging world knowledge and relations to encode compact grasping models, our pipeline can generalize over similar object and task categories, thus offering a natural way to encode the non-trivial realization of high-level knowledge. This allows us to experiment with a wide range of object categories and is therefore a critical aspect of autonomous agents acting deliberately in new environments. In this context, the exploitation of object ontologies, task affordances, and grasping constraints, and thus, successful

[1]The edges are colored according to the colors of the adjacent nodes.



(a) champagne glass (b) coffee mug (c) wine bottle

(d) butcher knife (e) hammer (f) frying pan

Figure 9.2: **Semantic k-nn Graphs of Household Objects.** Semantic k-nn graphs ($k = 4$) for six objects, (a) champagne glass, (b) coffee mug, (c) wine bottle (d) butcher knife, (e) hammer, and (f) frying pan with part labels *top* (yellow), *middle* (blue), *bottom* (red), *usable_area* (cyan), and *handle* (green). The edges are colored according to the colors of the adjacent nodes.

generalization, heavily depends on a good object category estimation. For example, a *cup* cannot be stored upside-down in a cupboard if it contains liquid, whereas for a full *can* this should be possible. As an example for task-dependent grasping, a *knife* should be grasped by its handle if its intended use is for cutting, whereas it has to be grasped at the blade if the task is passing it to a human. For the same task a *screwdriver*, however, could be grasped in any region as there is no danger for the interacting human to cut him/herself. So, our focus here will be on object category prediction, cf. the top row of Figure 9.1. The main contribution of this chapter is the use of propagation kernels for 3D point clouds to perform object categorization in the context of robot grasping. The full reasoning approach for task-dependent robot grasping is described in (ANTANAS, *et al.*, 2014).

Object categories are predicted by employing object similarity based on manifold and semantic part information via propagation kernels as introduced in the first

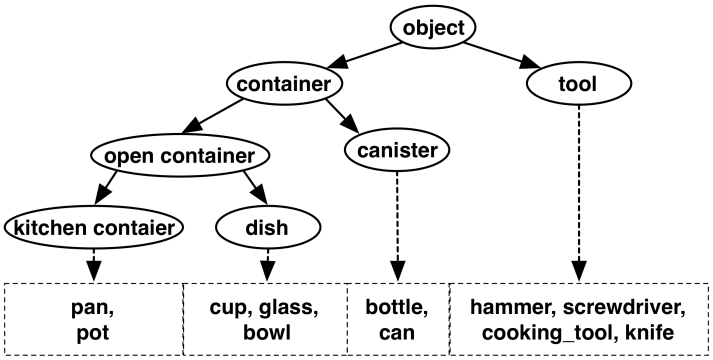


Figure 9.3: **Object Ontology for Task-dependent Grasping.** Ontology for the objects of different 11 categories considered for task-dependent grasping.

part of this thesis. Given a (partial) 3D point cloud, the surface normals, and part labels of each point, we construct a k-nearest neighbour (k-nn) graph and compute object similarities based on the kernel values among the respective labeled and attributed k-nn graphs. Object category prediction is achieved by applying a weighted vote on the categories of the most similar objects. Figure 9.2 shows k-nn graphs for six different objects. To enhance robustness, we compute a distribution over categories rather than hard predictions. Knowing the object category allows the robot to reason about object–task affordances and to generalize over object parts in similar task-dependent grasping situations. Generalization is achieved by utilizing an object ontology as illustrated in Figure 9.3.

Object categorization is an important problem in computer vision and robotics with numerous and versatile approaches. An extensive overview of related work is therefore beyond the scope of this section. In the following, we review approaches based on graph kernels as well as some recent ones considering 3D point cloud data. Considerably previous work addresses the problem of object categorization via graph kernels for 2D images (HARCHAOUI and BACH, 2007; MAHBOUBI, *et al.*, 2010; SEMENOVICH and SOWMYA, 2010; ANTANAS, *et al.*, 2012; DUCHENNE, *et al.*, 2011; ZHANG, *et al.*, 2013). Although the approaches are quite successful for medium sized graphs derived from 2D data, it is not clear how to generalize them to handle general 3D point clouds, where exploiting the manifold structure of the data is more challenging. Thus, we exploit the graph representation of 3D point clouds and the expressive features generated by the propagation kernel, which is able to deal with large graphs and missing information, commonly encountered in point clouds derived from laser range sensors.

Closely related is the work in (BACH, 2008), which introduces a subtree-pattern kernel for point clouds. This graph kernel is a special instance of the Weisfeiler–

Lehman subtree kernel, cf. Equation (2.30), (SHERVASHIDZE, *et al.*, 2011) which has proven to be successful on various bioinformatics benchmark datasets. Employing the Weisfeiler–Lehman test of isomorphism to compare subtree-patterns is highly efficient and hence, bounding the number of child nodes in the subtrees as suggested in (BACH, 2008) is not necessary for the sake of efficiency. Further, experimental evaluation in (BACH, 2008) is only carried out for character recognition based on small 2D point clouds and not for more challenging problems, such as general object categorization on 3D point cloud data. We will compare propagation kernels to the Weisfeiler–Lehman subtree kernel in our experimental evaluation.

Several other closely related lines of work tackle the problem of 3D object recognition (FARID and SAMMUT, 2013; MADRY, *et al.*, 2012; SAVARESE and LI, 2007). However, they do not exploit manifold information in a graph kernel. Thus, to the best of our knowledge we present the first approach utilizing manifold-based graph kernels for 3D point cloud categorization and real-world robotic grasping. In the following, we will introduce our approach to object category prediction employing propagation kernels.

9.2 Object Category Prediction with Propagation Kernels

Our main goal is to estimate the category of an object by retrieving the objects with the most similar global properties based on a graph kernel leveraging manifold information and semantic object parts. While beneficial for grasping point prediction (NEUMANN, *et al.*, 2012*a*), global object similarity also ensures a strong enough appearance-based predictor for object category. This prediction in the form of a distribution on object categories is then used as a prior for the probabilistic logical reasoning for task-dependent grasping. We will use propagation kernels as introduced in Chapter 4 of this thesis to model global object similarity for object category prediction.

9.2.1 Graph Representation of 3D Point Clouds

To get the distribution on object categories for a particular query object, we first represent its 3D point cloud as a graph. Then the graph kernel between this graph and graphs built from objects in an object database can be computed to retrieve the objects deemed most similar. We represent each object by a labeled and attributed graph where the labels are semantic parts and the graph structure is given by a k-nearest neighbour (k-nn) graph by connecting the $k = 4$ nearest points in the point clouds with respect to the Euclidean distance in 3D space. For each directed edge we add an edge pointing in the opposite direction to make the graph undirected. k-nn graphs of several objects are illustrated in Figure 9.2.

Curvature information can be captured in two different ways, either by node attributes or edge weights. In our experiments we will compare both representations

and discuss the trade-off between accuracy and speed of the kernel computation. When representing curvature by edge weights, we assign a weight reflecting the change in the object surface between the incident nodes encoded by the angle between the normals to the tangent planes at the respective points. The weight of edge (u, v) between two nodes is given by

$$w_{uv} = |\mathbf{n}_u \cdot \mathbf{n}_v|, \tag{9.1}$$

where $\mathbf{n}_u$ is the normal of point u and $\cdot$ is the dot product. When using attributes, we endow each node with a continuous curvature attribute approximated by its derivative, that is, by the tangent plane orientations of its adjacent nodes. The attribute of node u is given by

$$x_u = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} |\mathbf{n}_u \cdot \mathbf{n}_v|, \tag{9.2}$$

where $\mathbf{n}_u$ is the normal of point u and $\mathcal{N}(u)$ are the neighbors of node u. Some notes on geometry and the derivation of these expressions are provided in Appendix B.

The nodes have five semantic classes encoding object part information: *top*, *middle*, *bottom*, *handle* and *usable_area*. We stress that our graph representation of 3D point clouds as illustrated in Figure 9.2 captures both manifold information (geodesic distance) via their structure and semantic information (object parts) via their node labels.

9.2.2 Graph-kernel Based Object Classification

The similarity measure among objects is a kernel function over counts of similar node label distributions of the intermediate iterations of a information propagation scheme on the graphs. We use propagation kernels as defined in Equation (4.3) applied to fully labeled graphs, cf. Section 4.3.1. Note that we use label diffusion as propagation scheme even though there could be a small fraction of unlabeled nodes in the graphs due to missing sensor measurements. The node label distributions for unlabeled nodes will be initialized uniformly. However, no label push back, cf. Equation (2.11), is performed as missing labels are expected to occur in rare cases only; namely when the part detector is not able to uniquely map the points in the point clouds to a specific part.

Propagation kernels leverage the power of evolving continuous node label distributions as graph features and hence capture both manifold and semantic information. Given a new object with graph representation G_* that the robot aims to grasp, we first select the top n most similar graphs $\{G_1, \cdots, G_n\}$, where similarity is given by the respective row of the correlation matrix $\hat{K}$, where

$$\hat{K}_{ij} = \frac{K_{ij}}{\sqrt{K_{ii} K_{jj}}} \tag{9.3}$$

and K is the propagation kernel as computed by Algorithm 3. Note that by using $\hat{K}$ instead of K we achieve a normalization with respect to the number of points in the point clouds and hence, with respect to the scale of the objects. We set $n = 10$ in all our experiments. Second, we build a weighted average over the categories of the objects corresponding to $\{G_1, \cdots, G_n\}$, where the weight function is defined as

$$f(x) = \exp(-x) \tag{9.4}$$

with x being the rank after sorting the kernel row $\hat{K}_{*,:}$. This average is finally used as a prior distribution on the object category for a query object with graph representation G_* in the probabilistic logical module, where the final reasoning about the grasp to be performed is executed. For example, the prior distribution over object categories for the cup in Figure 9.1 using $t_{\text{MAX}} = 10$ is:

$$0.56 : \texttt{cup}; 0.36 : \texttt{can}; 0.05 : \texttt{pot}; 0.02 : \texttt{pan}; 0.002 : \texttt{bottle}.$$

Now, we will proceed by presenting experimental results on object category prediction.

9.3 Experimental Evaluation

Our intention here is to investigate the power of graph kernels, especially the propagation kernel, for object category prediction and thus its applicability in a probabilistic logical approach to task-dependent robot grasping. The three main questions we aim to answer in this section, are:

(Q9.1) How sensitive are propagation kernels with respect to their parameters, and with respect to missing label and attribute information for the task of object category prediction?

(Q9.2) Does incorporating manifold information, i.e. graph structure, improve upon label-based category prediction?

(Q9.3) Are propagation kernels competitive with state-of-the-art graph kernels the use in task-dependent robot grasping?

9.3.1 Datasets

The three datasets (DB, SEMI, and REAL) for quantitative evaluation are generated with the help of the ORCA simulator[2] and their statistics are summarized in Table 9.1.

For the first dataset (DB) data acquisition is fully simulated. That is, we have the complete point cloud of the input object and the semantic part labels are provided as ground truth. The point cloud is obtained from a previously defined

[2] `http://orca-robotics.sourceforge.net`

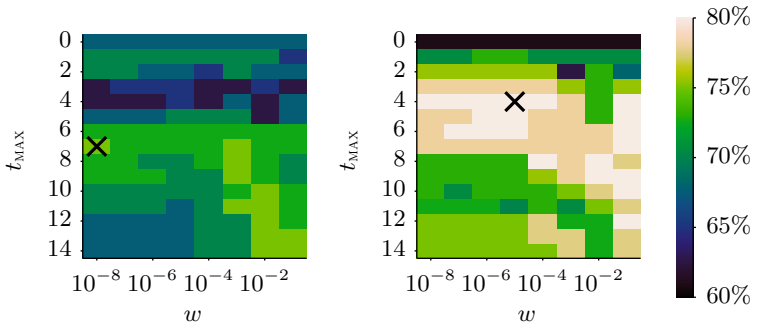
Table 9.1: **Dataset Statistics and Properties.** Note, that SEMI and REAL differ in the node labeling. In SEMI nodes are manually labeled, whereas in REAL semantic node labels are estimated by a part detector. DB is used in all experiments as database, so that graph kernels for the more realistic scenarios (SEMI and REAL) are computed between the 41 database graphs and the respective query object.

	properties						
dataset	# graphs	median # nodes	max # nodes	total # nodes	# node labels	# graph labels	attr. dim.
DB	41	964	5 037	1 377	5	11	1
SEMI/REAL	126	1 229	4 447	1 442	5	11	1

3D mesh of the object by up-sampling points using midpoint surface subdivision. Object parts are extracted manually from the point cloud. This "perfect scenario" is then used twofold. First, we use this dataset to analyze the parameter sensitivity of propagation kernels and to evaluate object category prediction in a synthetic scenario. The dataset denoted as DB contains 41 objects belonging to 11 categories modeled by the ontology illustrated in Figure 9.3. Second, we use these 41 objects $G_{\text{DB}} = \{G_1, \ldots, G_{41}\}$ as database to compute the propagation kernel $K_{t_{\text{MAX}}}(G_*, G_{\text{DB}}) = \sum_{t=0}^{t_{\text{MAX}}} K(G_*^{(t)}, G_{\text{DB}}^{(t)})$ to get the similarity among any query object G^* and the database objects. Hence, DB is used as object database in all experiments; it has a total of 56 468 nodes. Further, we plan to use it in execution time on a robot performing task-dependent grasps.

For the other two datasets (SEMI and REAL) the point clouds are simulated laser range data and the following steps of obtaining the scene description are executed in the ORCA simulator. Each point cloud of a query object is a partial view from one out of eight possible angles. The 3D data for each view is acquired from a simulated range camera (with the parameters of the Kinect sensor), placed on the robot platform. For each object we use between one and eight views depending on whether the object is symmetric or not. So, for symmetric objects as the champagne glass, cf. Figure 9.2, we only use one point cloud. We further consider two possible settings. In a first setting, we obtain the labels manually from the point cloud, that is we use ground truth label information. We denote this dataset as SEMI. In a second setting we also replace ground truth parts with more realistic part detector estimates.[3] We denote this dataset as REAL. The two more realistic datasets SEMI and REAL each contain 126 labeled point clouds which are instances of 26 objects of all categories except *cooking_tool*. The datasets DB and SEMI are available for download at `http://www.first-mm.eu/data.html`.

[3]Note, that the part detector estimates are corrected for various tool objects as the current detector only detects correct parts but not their correct location.



(a) DB (b) DB normalized

Figure 9.4: **Parameter Sensitivity of Propagation Kernels on DB.** The plots show heatmaps of average accuracies (leave-one-out CV) of PK (labels only) w.r.t. the bin widths parameter w and the number of kernel iterations t_{MAX} for the dataset DB. In panel (a) we show the kernel matrix directly; in panel (b) we show the normalized kernel matrix. × marks the highest accuracy.

9.3.2 *Parameter Analysis and Sensitivity to Missing Information*

To analyse parameter sensitivity for object category prediction with respect to the kernel parameters w (LSH bin width) and t_{MAX} (number of propagation kernel iterations), we computed average accuracies over 10 randomly generated test sets for all combinations of w and t_{MAX}, where $w \in \{10^{-8}, 10^{-7}, \ldots, 10^{-1}\}$ and $t_{MAX} \in \{0, 1, \ldots, 14\}$, on the DB dataset. From Figure 9.4 showing the resulting heatmaps of accuracies from a leave-one-out corss validation, we see that it is favorable to normalize the kernel matrix in order to make the predictions independent of the object scale. A cup scanned from a larger distance being represented by a smaller graph is still a cup and should be similar to a larger cup scanned from a closer view. So, for our experiments on object category prediction we will use normalized graph kernels. Further, we will set the bin width parameter of LSH to $w = 10^{-4}$.

To asses how sensitive propagation kernels are to missing information, we randomly selected x% of the nodes in all graphs of DB and removed their labels (LABELS) or attributes (ATTR), where $x \in \{90, 80, \ldots, 0\}$. To study the performance when both label and attribute information is missing, we selected (independently) x% of the nodes to remove their label information and x% of the nodes to remove their attribute information (LABELS & ATTR). Figure 9.5 shows the average accuracy of 10 reruns. While we see that the accuracy decreases with more missing information, the performance stays more stable in the case when attribute information is missing. This suggests that semantic part information is more important for the problem of object category prediction. Further, the standard error is increasing with more missing information, which also follows

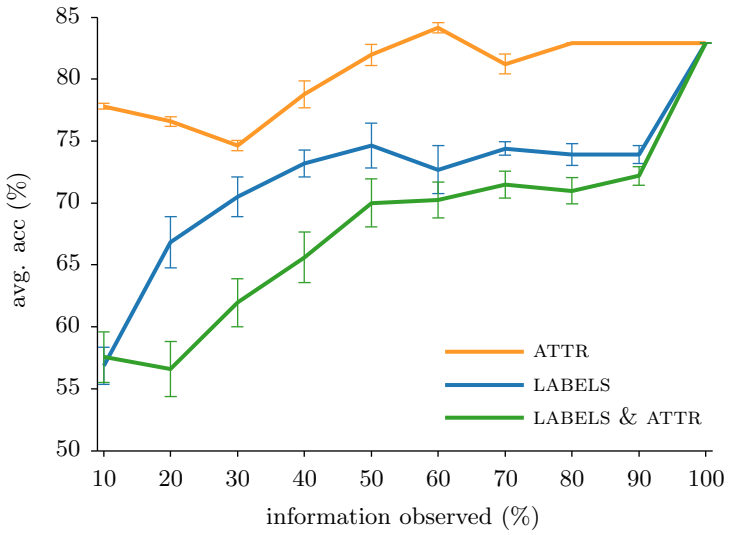


Figure 9.5: **Missing Information vs. Accuracy on DB.** We plot observed information (%) vs. avg. accuracy (%) ± standard error of P2K on DB. For LABELS only label information is missing, for ATTR only attribute information is missing and for LABELS & ATTR both is missing. The reported accuracies are averaged over 10 reruns.

the intuition that more unknown information results in a higher variance in the predictions.

In summary, we can answer question **(Q9.1)** by concluding that PKs are not overly sensitive to the choice of parameters and that they behave well in the presence of missing information.

9.3.3 *Experimental Protocol*

We compare propagation kernels (PK) to the Weisfeiler–Lehman subtree kernel (WL) and several baselines: a linear kernel (L) taking as input only the label counts but no graph structure, an attribute kernel (A) taking the mean of a Gaussian node kernel among the attributes of all pairs of nodes in the respective graphs, and A LSH corresponding to PK with $t_{\text{MAX}} = 0$ using the attribute information only. For all WL computations we use the fast (but exact) hashing-based implementation of the Weisfeiler–Lehman kernel as introduced in (KERSTING, *et al.*, 2014). The propagation kernel leveraging continuous node attributes (P2K) is described in Algorithm 4. To achieve classification we select the 10 most similar objects from the database DB based on the respective row of the normalized kernel matrix, Equation (9.3), and build a weighted average of their categories using Equation (9.4). Note that predicting the object category utilizing a kernel machine directly (we

Table 9.2: **Results on Attributed Point Cloud Graphs (DB).** Average accuracies (avg. acc in %) and average ranks (avg. rank) of the true category (the lower the better) of LOO CV on DB. The reported standard errors (stderr) refer to 10 kernel recomputations for all methods. Average (elapsed) runtimes for kernel computations (possibly utilizing hyperthreading) are given in seconds (x''). PK* uses the edge weights defined in Equation (9.1), whereas all other methods use unweighted graphs. The kernel parameters, t_{MAX} for all PKs and h_{MAX} for WL, were learned on the training splits ($t_{\text{MAX}}, h_{\text{MAX}} \in \{0, 1, \ldots, 15\}$). A LSH corresponds to PK with $t_{\text{MAX}} = 0$. Bold indicates best performance.

	LABELS			LABELS ATTR			ATTR	
	L	PK	WL	PK*	PK	P2K	A	A LSH
avg. acc	63.4	75.6	70.7	71.0	76.8	**82.9**	36.4	63.4
± stderr	0.0	0.6	0.0	0.2	1.3	0.0	0.0	0.0
avg. rank	1.88	1.54	1.83	1.70	1.60	**1.41**	4.34	1.88
runtime	0.0″	0.2″	0.4″	0.2″	0.3″	34.8″	40.0″	0.0″

tried support vector machine classification) did not give competitive results. This might be caused by the large number of classes in combination with few training examples. For missing part label information on the nodes we initialize the label distribution uniformly. For WL we use an additional label as suggested in (SHERVASHIDZE, *et al.*, 2011). For all experiments on DB we learn the PK and WL kernel parameter t_{MAX} and h_{MAX} on each training set by leave-one-out cross validation (LOO CV). The parameter ranges are set to $t_{\text{MAX}}, h_{\text{MAX}} \in \{0, 1, \ldots, 15\}$. For the experiments on SEMI and REAL we use the best parameter values based on one LOO CV on all objects in DB. For DB, performance is measured by LOO CV. On SEMI and REAL we report average results on all objects as DB is used as separate training data set.

9.3.4 *Quantitative Results on Object Category Prediction*

In our first set of experiments we consider attributed graphs where the curvature information is encoded as a one-dimensional continuous node attribute as given by Equation (9.2). We use the 41 objects in DB and follow the experimental protocol introduced above. The experimental results for the 3D object classification are summarized in Table 9.2. Propagation kernels can easily deal with the point cloud graphs. From the set of baseline graph kernels considered in the experimental evaluation in Chapter 4, only WL could be computed. Shortest path kernel (SP) (BORGWARDT and KRIEGEL, 2005), graph hopper kernel (GH) (FERAGEN, *et al.*, 2013), and common subgraph matching kernel (CSM) (KRIEGE and MUTZEL, 2012)

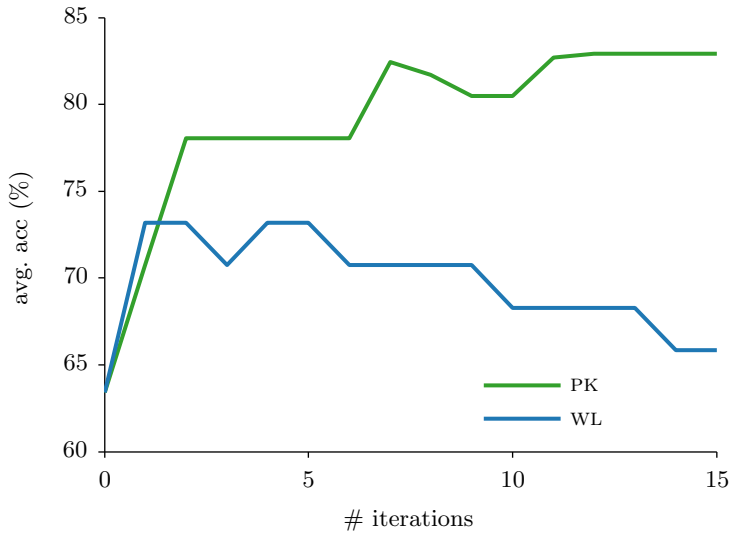


Figure 9.6: **Number of Iterations vs. Accuracy on DB.** Accuracy (%) is plotted against the number of kernel iterations of PK and WL. For PK accuracies are averaged over 10 kernel recomputations; the standard error was 0% except for $t_{\text{MAX}} = 7, 8, 11$ it was 0.3%, 0.7%, 0.2%. Both methods use unweighted graphs and label information only; no attributes are used.

were either OUT OF MEMORY[4] or the computation did not finish within 24h for all settings. On DB we clearly see that both graph kernels PK and WL improve upon using label or attribute information only; however, WL performs worse than PK. PK* using the curvature information encoded in the edge weights, cf. Equation (9.1), gives worse results than considering unweighted graphs (PK). Propagation kernels clearly benefit form their flexibility as classification accuracy can be improved from 75.6% to 82.9% when considering the object curvature attribute in P2K, cf. Equation (9.2). Further, propagation kernels significantly leverage manifold information as the performance of all PK methods is better than using label or attribute information only.

In the next experiment we study the behavior of PK and WL with respect to their kernel iterations. Average accuracies plotted against the number of kernel iterations are shown in Figure 9.6. The best results for fixed number of iterations of PK on unweighted graphs using the labels only reach 82.9% accuracy and are achieved for kernel iterations larger than 12. The result when using label information only is 63.4% accuracy. It corresponds to the results of PK and WL for $t_{\text{MAX}} = h_{\text{MAX}} = 0$. WL can only improve upon this baseline by 9.8% which is achieved after one iteration. PK improves the baseline by 19.5% resulting in an accuracy over 80%, which is

[4] 32 GB of memory were exceeded.

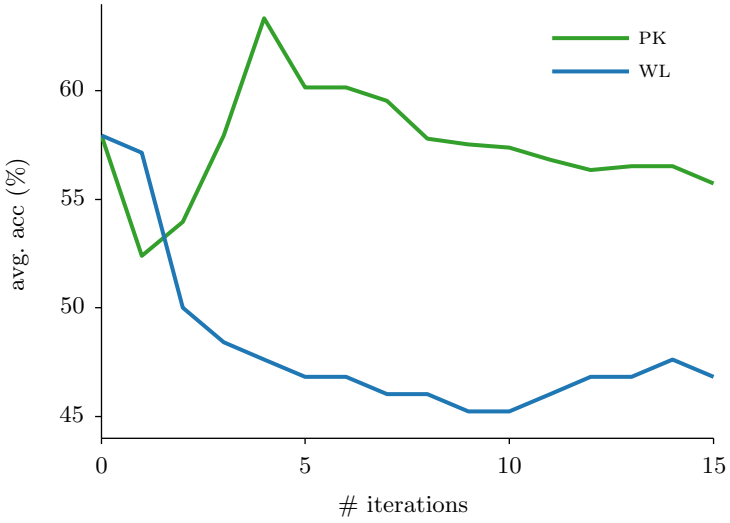


Figure 9.7: **Number of Iterations vs. Accuracy on REAL.** Accuracy (%) is plotted against the number of kernel iterations of PK and WL. For PK accuracies are averaged over 10 kernel recomputations; standard errors were below 0.3% for all values of t_{MAX}.

considered a good performance in object recognition tasks. These results are extremely promising given that we tackle a classification problem with 11 classes having only 40 training examples for each query object. Thus, questions **(Q9.2)** and **(Q9.3)** can already be answered affirmatively. Unfortunately, computing P2K on DB took over half a minute. This is not desirable in a scenario where a robot actually attempts to grasp an object. In the following set of experiments, simulating the more realistic scenarios in task-dependent grasping, we will therefore not consider curvature information.

Now, we will evaluate PK and WL on the more realistic scenarios SEMI and REAL. We choose the kernel iteration parameters for both methods according to the best results on DB leading to $t_{\text{MAX}} = 12$ for PK and $h_{\text{MAX}} = 1$ for WL. Predictive performance and runtimes are reported in Table 9.3. On SEMI only propagation kernels are able to improve the performance of the baseline method. In the third scenario (REAL) average accuracies are slightly worse for both graph kernels than just using the label information only. However, plotting average accuracies against the number of kernel iterations, as shown in Figure 9.7, reveals that PK actually can improve upon using labels only. Thus by changing the learning procedure to set t_{MAX} we can hope to improve object category prediction even in the challenging scenario REAL. Thus, question **(Q9.2)** can be answered affirmatively. For WL setting $h_{\text{MAX}} = 0$ gives the best performance leading to the conclusion that this kernel cannot leverage manifold information encoded in the graph structure of

Table 9.3: **Results on Object Category Prediction (SEMI and REAL).** Average accuracies (avg. acc in % ± stderr) and average rank (avg. rank) of the true category in the distributions (the lower the better) on all objects in the datasets SEMI and REAL for graph kernels PK and WL, and the baseline L. The reported standard errors (stderr) refer to 10 kernel recomputations for all methods. Bold indicates best performance for the respective measure.

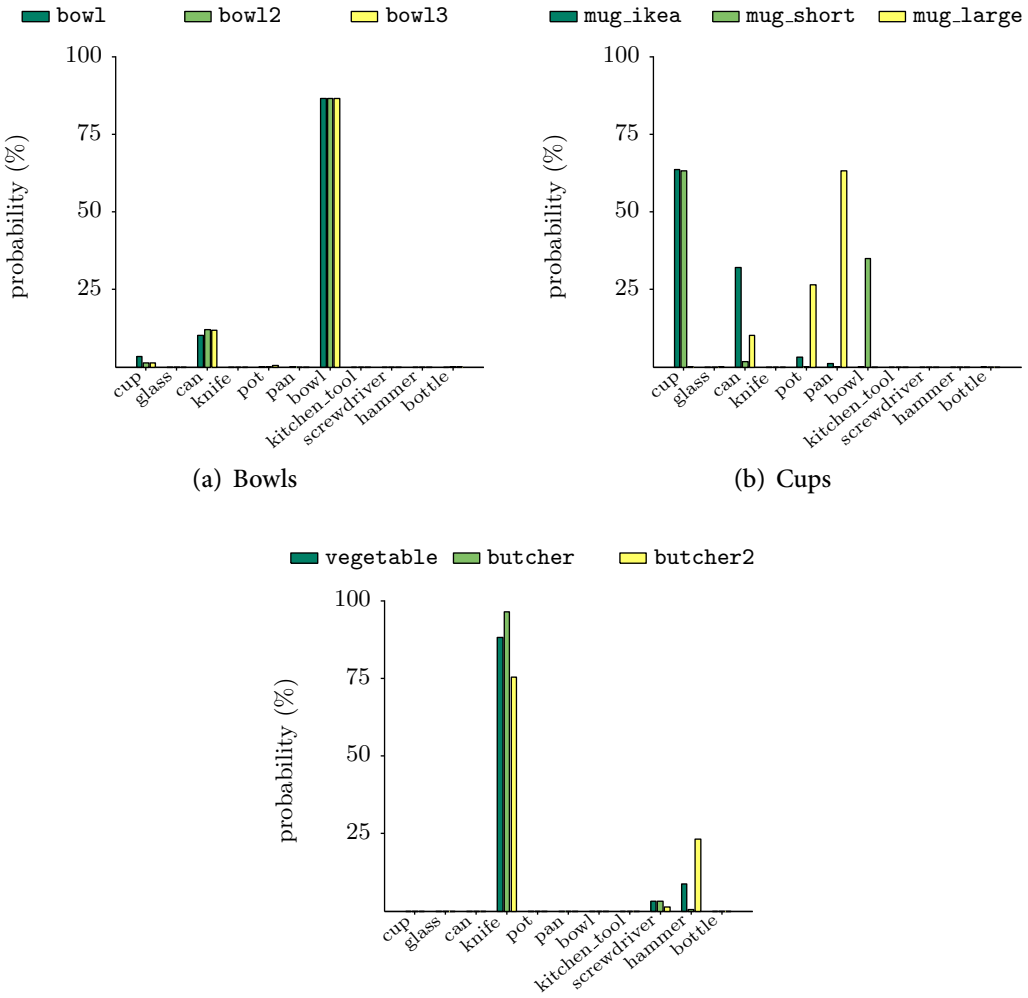
		L ($t_{MAX} = 0$)	WL ($h_{MAX} = 1$)	PK ($t_{MAX} = 12$)
SEMI	avg. acc	61.1 ±0.0	58.7 ±0.0	**64.4** ±0.2
	avg. rank	1.78	1.71	**1.56**
	runtime	0.0″	0.3″	0.1″
REAL	avg. acc	**57.9** ±0.0	57.1 ±0.0	56.4 ±0.2
	avg. rank	2.09	2.01	**1.83**
	runtime	0.0″	0.3″	0.1″

the 3D point clouds in REAL. These results lead to an affirmative answer of **(Q9.3)**; propagation kernels outperform the considered sate-of-the-art graph kernels on the task of object category prediction in the context of task-dependent robot grasping.

Besides analyzing the most probable category, our aim here is to also evaluate the category distributions and their use in the probabilistic logical approach to task-dependent grasping. First, we analyze example distributions for objects of three different categories *bowl*, *cup*, and *knife* in Figure 9.8. The category distributions for all bowls have its highest probability at the correct category (Figure 9.8(a)). Predicting the category of `mug_large`, however, fails (Figure 9.8(b)). This might be because the body of this particular cup is shaped more like one of the pans and some pots in the database. In general it is reasonable that the cups look more like pans and pots as they have a similar shape[5] and the same parts (*top*, *middle*, *bottom*, and *handle*). The distributions of the knives (Figure 9.8(c)) clearly show their similarity to other knives but also to other tool objects like hammers and screwdrivers. This is consistent with their semantic parts (*usable_area* and *handle*). The higher probability mass for the category *knife* is caused by their shape, i.e., the graph structure of the corresponding point clouds. This behaviour is exactly desired in the probabilistic logical grasping model as task affordances and object ontology can be used to achieve generalization.

In order to quantitatively compare the distributions we compute the average rank of the true category in the distributions obtained from all kernels L, WL, and PK.

[5]Note, that we use the correlation matrix (Equation (9.3)) which yields a normalization with respect to the size (number of nodes).



(a) Bowls (b) Cups

(c) Knives

Figure 9.8: **Estimated Category Distributions.** Distributions among categories derived from PK with $t_{MAX} = 12$ for three example objects of each of the following categories: *bowl* (a), *cup* (b), and *knife* (c).

The results are also listed in Table 9.3. Again, we see that leveraging the graph structure improves the average rank especially for the third scenario REAL. As we are also interested in achieving good performance as fast as possible we show the average rank of the true category plotted against the number of kernel iterations in Figure 9.9. On REAL 14 iterations of WL result in the lowest average rank for the Weisfeiler–Lehman subtree kernel. However, PK results in a not much worse rank after only 4 kernel iterations and hence in faster time. In robotics computation time is an important criteria. Obviously we do not want the robot pausing too

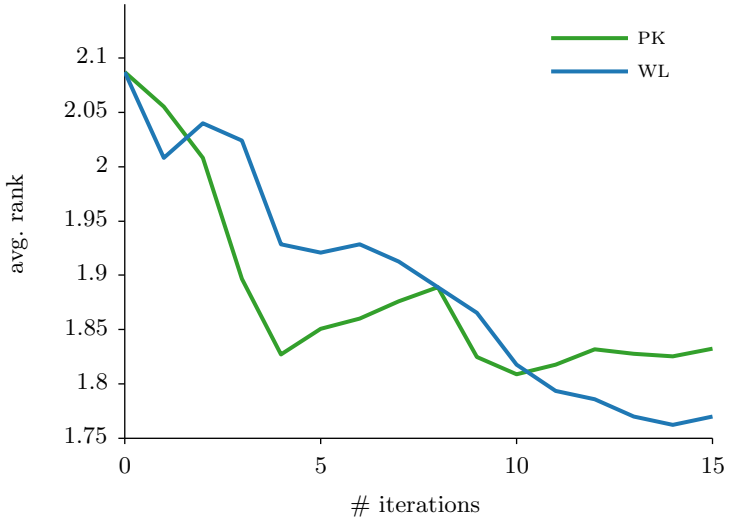


Figure 9.9: **Number of Iterations vs. Average Rank on REAL.** Average ranks of the true category in the distribution are plotted against the number of kernel iterations of PK and WL. For both axes lower values are better as we would like to achieve the best possible result, i.e., an average rank of 1, as fast as possible.

long before actually grasping the desired object appropriately. Hence, evaluating applicability in robotics also involves analyzing the trade-off between qualitative performance and computation time. Figure 9.9 clearly shows a superior behaviour for propagation kernels as a lower rank is reached in shorter time. These results answer question **(Q9.3)** affirmatively.

9.3.5 Benefits for Task-dependent Grasping

In this paragraph, we aim to investigate the importance of using the object category distributions derived from the propagation kernel in the probabilistic logical pipeline described in the problem introduction. We compare the performance of incorporating kernel-based distributions versus uniform distributions on object categories for two inference problems. In the first problem, we want to infer the most suitable grasping task to be performed on a given query object. The second problem is to predict the most suitable pre-grasp, i.e. graspable region for a given grasping task. This setting is depicted in the bottom row of Figure 9.1 for the given task *pass*. Fraction of successful predictions on DB for task selection is 50.98% when using the uniform prior and 72.55% when incorporating the manifold prior derived by propagation kernels. For the pre-grasp selection we achieve 85.85% accuracy with the uniform prior and 87.82% with the PK-based manifold prior. A more detailed evaluation of the whole probabilistic logical pipeline can be found in (ANTANAS, *et al.*, 2014). In summary, the results for both grasping tasks are

significantly better when using the graph kernel distribution as opposed to the uniform one.

9.4 Summary and Discussion

To summarize, we have presented a novel application of graph kernels, specifically, object category prediction as a prerequisite for task-dependent robotic grasping. Given an object to grasp, we convert its 3D point cloud into a graph and use a part detector to predict part labels for each node. Continuous curvature information is encoded as node attributes. This graph representation of an object is then compared to graphs of known objects in a database, and the most similar ones are used to predict the category of the unknown object. Whereas many state-of-the-art graph kernels cannot cope with the graphs derived from 3D point clouds, propagation kernels can be computed among the database of 41 objects covering more than 55 000 nodes in under 200 milliseconds. Further, propagation kernels outperform Weisfeiler–Lehman subtree kernels and we showed in a series of experiments that it is able to quickly and accurately predict an object's category. Further, we show that propagation kernels behave naturally in the presence of missing label and attribute information. This is especially important for object category prediction based on 3D point clouds as labels and attributes are derived from sensor measurements of a laser range scanner mounted on a mobile robotic platform. Finally, our graph-kernel-based object category predictor has measurable impact on the overall quality of the task-dependent robotic grasping pipeline it is part of. Our results highlight the importance of good category distributions for task-dependent robot grasping and encourage further research on graph kernels for object category prediction.

CHAPTER 10

IMAGE-BASED PLANT DISEASE CLASSIFICATION

Modern communication and sensor technology coupled with powerful pattern recognition algorithms for information extraction and classification allow the development and use of integrated systems to tackle environmental problems. This integration is particularly promising for applications in crop farming, where such systems can help to control growth and improve yields while harmful environmental impacts are minimized. Thus, the vision of sustainable agriculture for anybody, anytime, and anywhere in the world can be put into reach. In this chapter, we study the use of kernels from propagated information for the task of disease classification based on cell phone images of sugar beet leaves. In particular we analyze the use of coinciding walk kernels for leaf spot detection, a first crucial step in the pattern recognition pipeline. Further, we apply propagation kernels to texture-based disease classification. The classification of disease symptoms caused by various fungi or bacteria is evaluated on a comprehensive dataset of 2 957 regions extracted from 495 images of sugar beet leaves.

10.1 Problem Introduction and Related Work

In this section, we will motivate the importance of accurate plant disease classification and give a brief introduction to diseases of sugar beet plants. Moreover, we will introduce the core steps of the considered pattern recognition pipeline for plant disease classification.

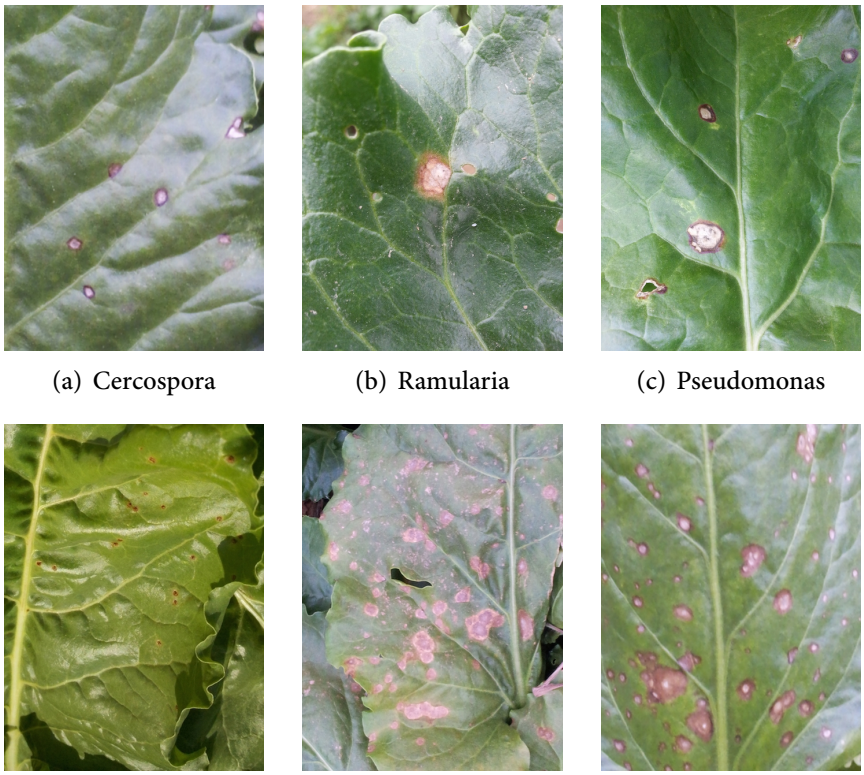
10.1.1 Sustainable Agriculture for Anybody, Anytime, and Anywhere

In the presence of increasing environmental challenges such as water scarcity, climate change, concerns to food safety and the reduction of adverse environmental impacts, sustainable agriculture is an extremely important resort to cope with a rapidly growing world population. The scope of farming that is sustainable outreaches short-term yield maximization and efficiency steered exploitation of

resources. Instead the focus lies on farming methods to secure long-term yields of products satisfying the food and fiber needs of animals and humans while even enhancing the quality of the environment. One important aspect to achieve this goal is to control the outbreak and spread of plant diseases causing significant reduction in the quantity and quality of farming products. Hereby, it is crucial to apply targeted rather than broad and overdosed preventive control to minimize soil and water damage. Targeted disease control, however, relies on two major requirements. First, plant diseases need to be identified reliably, even for early stage outbreaks. And second, information about diseases and possible treatments has to reach crop producers and farmers promptly, especially in remote areas that are difficult to access by experts. The approach presented in this chapter aims to achieve both requirements, accurate plant disease classification and real-time forecasting based on the particular state in the field.

In addition to providing farmers with treatment recommendations, accurate plant disease classification is also pivotal in order to monitor the outbreak and spread of diseases. This information can then again be used to predict yield losses and to forecast the temporal and spatial disease spread facilitating the coordination of countermeasures. While there exist many methods to classify plant diseases as for instance molecular techniques, non-invasive approaches such as visual assessment, are more favorable. Modern techniques of visual disease assessment are based on digital photography combined with image analysis (BOCK, *et al.*, 2010; CAMARGO and SMITH, 2009*a*) or hyperspectral imaging (RUMPF, *et al.*, 2010; LIU, *et al.*, 2010). Visual image analysis also contributes to the understanding of biological processes such as the molecular mechanisms of photosynthesis (RASCHER, *et al.*, 2007) or the growth rate and spread of symptoms on the disease carrier (DUNCAN and HOWARD, 2000; ROSSI, *et al.*, 2000).

Mobile devices such as smart phones have become widespread consumer products and the prevalence and ease of use of smart phone cameras provides new challenges and opportunities for image processing and analysis. Opportunities arise from the availability of digital cameras in environments where they were not naturally present only a few years ago. Challenges are due to the fact that implementations of image processing algorithms on a cell phone have to comply with particular characteristics such as constrained battery life, restricted computational power, or limited bandwidth. The work reported here results from a project on using cell phone images in an agricultural scenario. We are developing a system where farmers take pictures of plants they suspect to be infected by a disease such as shown in Figure 10.1. Information extracted therefrom are then send to a central server and analysis results are supposed to be reported back to the farmer while still in the field. In this setting, efficient and reliable image analysis is pivotal. Given the weak connection strengths out in the fields or the increased fees for high volume data transfer, it is hardly possible to transmit several pictures of sufficient resolution. If, on the other hand, the extraction of regions of interest or even



(a) Cercospora (b) Ramularia (c) Pseudomonas

(d) Rust (e) Phoma (f) Cercospora &Phoma

Figure 10.1: **Example Images of Sugar Beet Leaves.** Cell phone camera images of beet leaves showing leaf spots caused by (a) Cercospora beticola *(cerc)*, (b) Ramularia beticola *(ram)*, (c) Pseudomonas syringae *(pseu)*, (d) Uromyces betae *(rust)*, (e) Phoma betae *(pho)*, and (f) combined infestation of Cercospora and Phoma.

the feature computation were performed by an app running on the cell phone, transmission times and costs reduce considerable. In this case, however, elaborate image processing techniques being both robust and limited to the restricted computational resources need to be applied. State-of-the-art approaches address these issues by implementing cascades of efficient image preprocessing, region detection, feature extraction, and classification steps (NEUMANN, *et al.*, 2014*b*; AL-HIARY, *et al.*, 2011; CAMARGO and SMITH, 2009*a*,*b*). Usually these steps are heavily tailored to the recognition of pathogens that infect crop plants (AGRIOS, 2005).

10.1.2 Diseases of Sugar Beet Plants

Sugar Beet (*Beta vulgaris subsp. vulgaris*) is a widely cultivated commercial crop used to produce, for example, table sugar. Unfortunately, fungal and bacterial

attacks frequently reduce yields. An early recognition of disease onsets assisted by our pattern recognition approach can trigger targeted control reducing costs and environmental burden. In particular, we attempt to automatically recognize symptoms of five common kinds of infections of sugar beet plants:

Cercospora beticola (cf. Figure 10.1(a)) is a fungal plant pathogen. Infected beet plants show leaf spots that are round blemishes with a definite edge between infected and healthy leaf tissue; while the border of a spot is typically darker and of brownish-reddish hue, spot interiors appear bright with dark spores.

Ramularia beticola (cf. Figure 10.1(b)), also a fungal pathogen infecting beet plants, shows irregular leaf spots with light interior surrounded by a light brown to brown border. The interior shows white spores. Leaf spots at a pronounced stage are likely to chap or form coadunate regions.

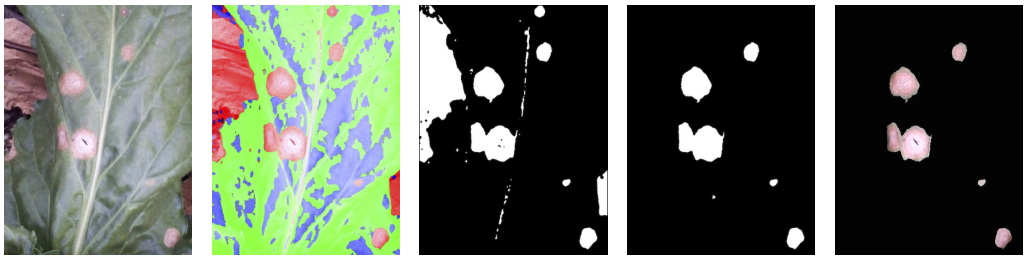
Pseudomonas syringae pv. aptata (cf. Figure 10.1(c)), the only considered bacterial pathogen commonly infecting beets, shows light to dark brown spots potentially with a light interior. The spots may have a brown to black edge.

Uromyces betae (cf. Figure 10.1(d)) is a fungal pathogen causing sugar beet rust. Typical symptoms are small reddish-brownish spots surrounded by yellow areas.

Phoma betae (cf. Figure 10.1(e)) is a soil-borne fungus and plant pathogen showing rather large leaf spots of concentric rings with irregular shape and less pronounced edges; borders of a spot are yellowish or light brown, spot interiors are characterized by darker brownish hues; when several spots are present, they can grow together and form larger blotches.

10.1.3 Pattern Recognition Pipeline for Plant Disease Classification

Following existing work, we take a statistical machine learning approach using field data reflecting the variety of the real environment to deal with changes in illumination, scale, and perspective (NEUMANN, *et al.*, 2014*b*; AL-HIARY, *et al.*, 2011; CAMARGO and SMITH, 2009*a*,*b*). The data considered for learning and evaluation in our approach are cell phone images from unconstrained settings – different camera types, different resolutions, no constraints on how to take the image. Pattern recognition pipelines for plant disease classification essentially consist of three steps: region detection, feature extraction, and class prediction. *Region detection* is performed on the whole input image to extract regions of interest being likely to show disease symptoms. In this step a common approach is to apply a color filter, cf. Figure 10.2 (NEUMANN, *et al.*, 2014*b*; CAMARGO and SMITH, 2009*b*). While being feasible on the cell phone the computation is rather ad-hoc and therefore very sensitive to the applied color thresholds. To minimize the data volume to be sent to the central sever for further processing the color filters are likely to detect only a small subset of the regions showing disease symptoms. Therefore, we study the application of coinciding walk kernels introduced in Chapter 7 of this thesis to



(a) input image (b) max RGB (c) filtered image (d) binary regs (e) extracted regs

Figure 10.2: **Color-Based Symptom Extraction.** Cell phone camera image of a beet leaf suffering from Phoma betae (a), and preprocessing and region detection steps: maximal RGB values (b), color-filtered binary image (c), binary region image (d), and extracted regions (c).

detect leaf spots based on a few positive training regions derived by a color filter. We will introduce this approach in more detail in Section 10.2.

Feature computation and classification is then performed on each extracted region independently. This approach enables us to identify leaves showing infection symptoms from multiple diseases as illustrated in Figure 10.1(f). *Feature computation* is the core step in the pattern recognition pipeline. A wide range of possible techniques can be considered to extract meaningful descriptors. On the one hand, the input regions contain complex information, such as intensity and color values, edge contours, or intensity changes, which can be revealed by applying various transformations to the input region images. On the other hand, a variety of statistics can be extracted from the original image and its transformations. Existing approaches to plant disease classification as described in (NEUMANN, *et al.*, 2014*b*; AL-HIARY, *et al.*, 2011; CAMARGO and SMITH, 2009*a*) rely on statistical texture descriptors (HARALICK, *et al.*, 1973) for efficient and accurate classification. However, considering all possible combinations of statistical measures and input images leads to a high-dimensional, complex feature space being both costly to compute and prone to overfitting. For example, if we consider the following commonly used image transformations: intensity image, red, green, and blue channel, local binary patterns (OJALA, *et al.*, 2002) of the intensity image, gradient magnitude image, gradient direction image, and the local binary pattern of the gradient magnitude image as input values, and three simple statistics: mean, variance, and entropy, as well as three co-occurrence-based features according to (HARALICK, *et al.*, 1973) (e.g. for four different pixel offsets) we get 56 single (possibly multi-dimensional) descriptors. This considerably small set of candidate features leads to 2^{56} possible descriptor sets being extremely costly to compute and to evaluate. Thus, deriving a meaningful feature set turns into a highly non-trivial feature selection process

(NEUMANN, *et al.*, 2014*b*; CAMARGO and SMITH, 2009*a*). To overcome this problem we propose the use of propagation kernels for texture-based plant disease classification. Surprisingly, applying the grid graph implementation of propagation kernels, introduced in Chapter 5, *straight out-of-the-box* results in highly promising classification accuracies while avoiding the complex feature selection process. We will evaluate this approach on a comprehensive dataset of 2 957 regions extracted from 495 images recorded with six different cell phone camera types in Section 10.3.

10.2 Leaf Spot Detection with Coinciding Walk Kernels

In this section, we study the use of coinciding walk kernels introduced in Chapter 7 for the task of image-based leaf spot detection.

10.2.1 Limitations of Color-Filter Approaches

Extraction of diseased regions in images of plant leaves is most commonly tackled by simple color filters. Color-based leaf spot detectors essentially compute a binary image B from the input image I such that all pixels with color values above a given threshold are foreground pixels and all other pixels are background pixels. We can also compare the values of the different color channels up to a threshold to derive B. The choice of color-filter and threshold heavily depends on the appearance of the disease symptoms. Figure 10.2(b) illustrates the maximal color values of an example image I (Figure 10.2(a)) showing a sugar beet leaf and Figure 10.2(c) shows the color-filtered binary image when selecting all pixels with maximal red value. Unfortunately, this result is not yet satisfactory, and thus, various additional filter steps have to be applied; e.g., median filtering, connected component analysis, hole filling, filtering of regions adjacent to the image borders, as well as filtering of regions with undesired shapes. Although these steps can lead to good results as shown in Figures 10.2(d) and 10.2(e), they need to be implemented carefully and more importantly their parameters need to be adjusted specifically for each plant or even each plant disease.

An additional problem of color-filter based approaches is that the decision whether a pixel is a foreground or background pixel is only based on the values of each pixel separately ignoring the color structure of their neighbors. Figure 10.3 illustrates the problem for two cases, where based on the single pixel values the target regions would be detected wrongly. In this thesis, we have introduced coinciding walk kernels to measure label structure of the nodes in a graph in their local neighborhoods. Viewing images as grid graphs and color information as node labels, we can apply coinciding walk kernels to measure the color arrangement of pixels in order to classify whether a pixel is showing a disease symptom or not.

10.2.2 Region Detection with Coinciding Walk Kernels: An Empirical Demonstration

To study the capability of coinciding walk kernels to detect disease regions given very few areas of an image showing both healthy and diseased leaf parts, we

(a) Case 1: healthy region, disease region, target region

(b) Case 2: healthy region, disease region, target region

Figure 10.3: **Limitations of Color Filter.** In both cases depicted in Panel (a) and (b) a threshold-based color filter would wrongly detect the target regions (depicted on the right) given the training data for healthy (left image) and disease regions (middle image).

derived the following graph representation of the two images depicted in Panel (a) in Figures 10.4 and 10.5. The graph structure is given by the pixel grid graph according to the derivation introduced in Section 5.1. We use a 4-neighborhood and further quantized the input image using 30 colors to get the node labels, cf. Panel (b) in Figures 10.4 and 10.5. However, these labels are not the target classes for prediction. The classification problem is to predict whether a pixel is a foreground ("diseased") or a background ("healthy") pixel. That is, the class labels used as training data are not the ones being propagated in the coinciding walk kernel computation. Also note that, as we have fully observed node information, we will set $\alpha = 0$ in Algorithm 6. No absorbing states are used in the random walks; the information propagation process is label diffusion, cf. Equation (2.10). The positively labeled regions were generated by a color-filter simply selecting the pixels with maximal red values (MAXRED) followed by several additional filters (connected component analysis, hole filling, as well as filtering of regions with undesired shapes); the negative training regions were selected by manually placing a bounding box around healthy leaf parts, cf. Panels (c,e) in Figures 10.4 and 10.5. Detection is achieved via binary SVM classification using the precomputed CWK kernel matrix. To account for the high class imbalance we used a class weighting

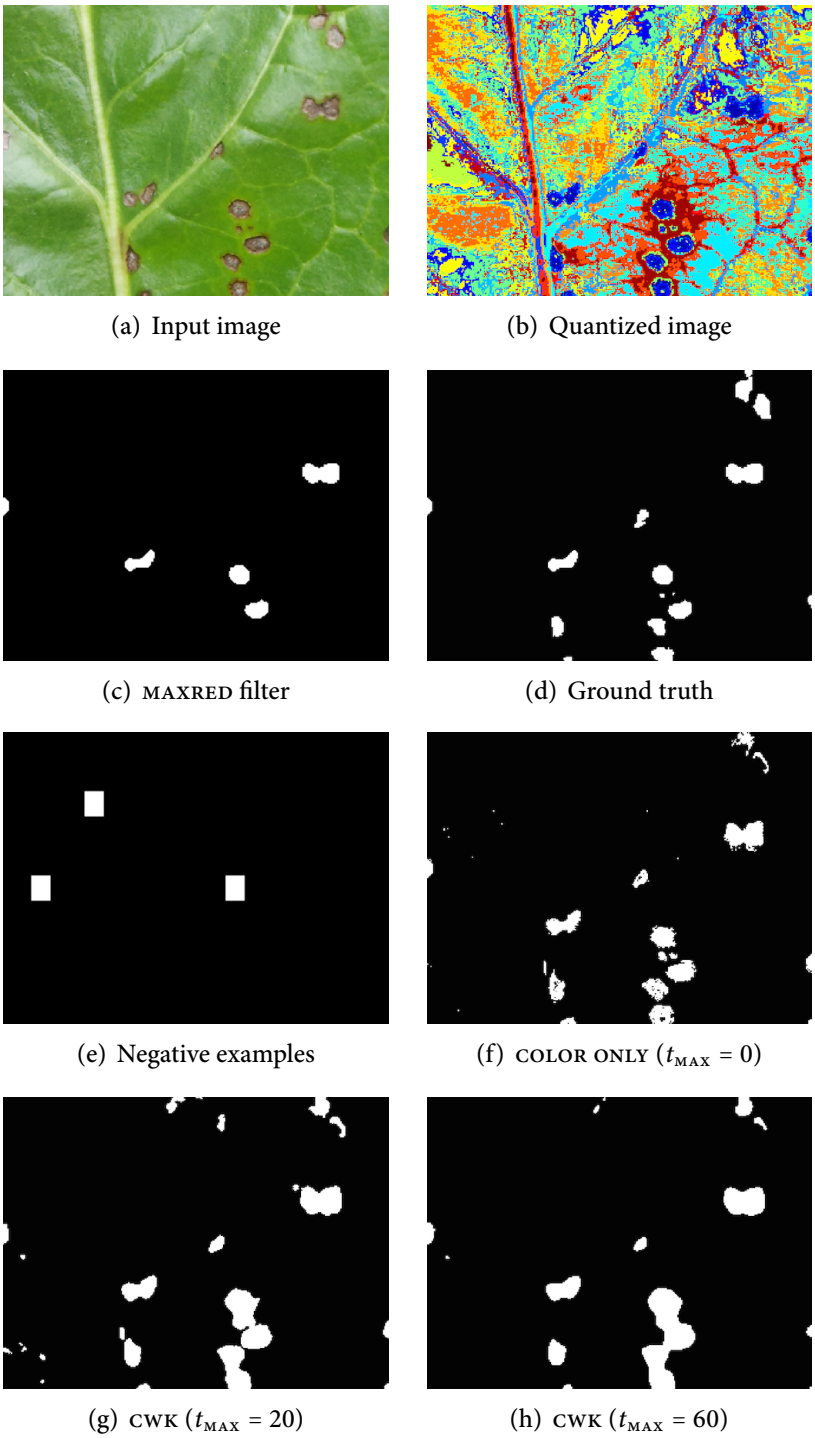
Table 10.1: **Results For Leaf Spot Detection.** Precision, recall and area under the curve (AUC) in % for leaf spot detection on Example 1 and Example 2 using color information only (COLOR ONLY, $t_{MAX} = 0$) and two CWKs with $t_{MAX} = 20$ and 60. Bold indicates best performance for the respective measure.

		COLOR ONLY $t_{MAX} = 0$	CWK $t_{MAX} = 20$	CWK $t_{MAX} = 60$
Example 1	precision	**39.6**	28.2	28.4
	recall	56.5	**72.2**	66.6
	AUC	86.2	**92.1**	91.7
Example 2	precision	36.0	**40.4**	39.2
	recall	41.7	82.8	**90.0**
	AUC	70.2	96.9	**98.8**

of 1 : 50 in `libSVM`;[1] the cost parameter was set to the default value of 1 for all experiments.

Figures 10.4 and 10.5 show the ground truth regions (Panel (d)) and the predictions of coinciding walk kernels (Panel (f-h)) for the two example images. The result in Panel (f) is a CWK with $t_{MAX} = 0$ corresponding to a classification using color values of each single pixel only (COLOR ONLY). Panels (g,h) show the results when incorporating color arrangements of neighboring pixels. First, we can see that the initially applied MAXRED filter fails in both examples to detect all disease regions. Coinciding walk kernels basically detect all infected regions for both example images. Note that small detection artifacts are not problematic as they can be handled by applying an additional size filter. Comparing these detection results to the ones achieved by an approach merely based on the color information of each single pixel, cf. Panel (f) in Figures 10.4 and 10.5, we observe that leveraging the arrangement of pixel values, i.e., using CWK with $t_{MAX} > 0$, leads to a clear improvement. In Example 2 COLOR ONLY fails to detect the lighter leaf spot in the top left corner. To quantitatively compare the different methods we compare precision, recall and area under the curve (AUC), when comparing the true labels to the decision values of the SVM. The results for both example images are summarized in Table 10.1. CWK with $t_{MAX} = 20$ and $t_{MAX} = 60$ clearly outperform COLOR ONLY by achieving recall rates of over 70% for Example 1 and 90% for Example 2. Further, AUC is clearly improved to above 90% for both images. The results in this demonstration show that coinciding walk kernels are a promising approach to leaf spot detection.

[1] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`



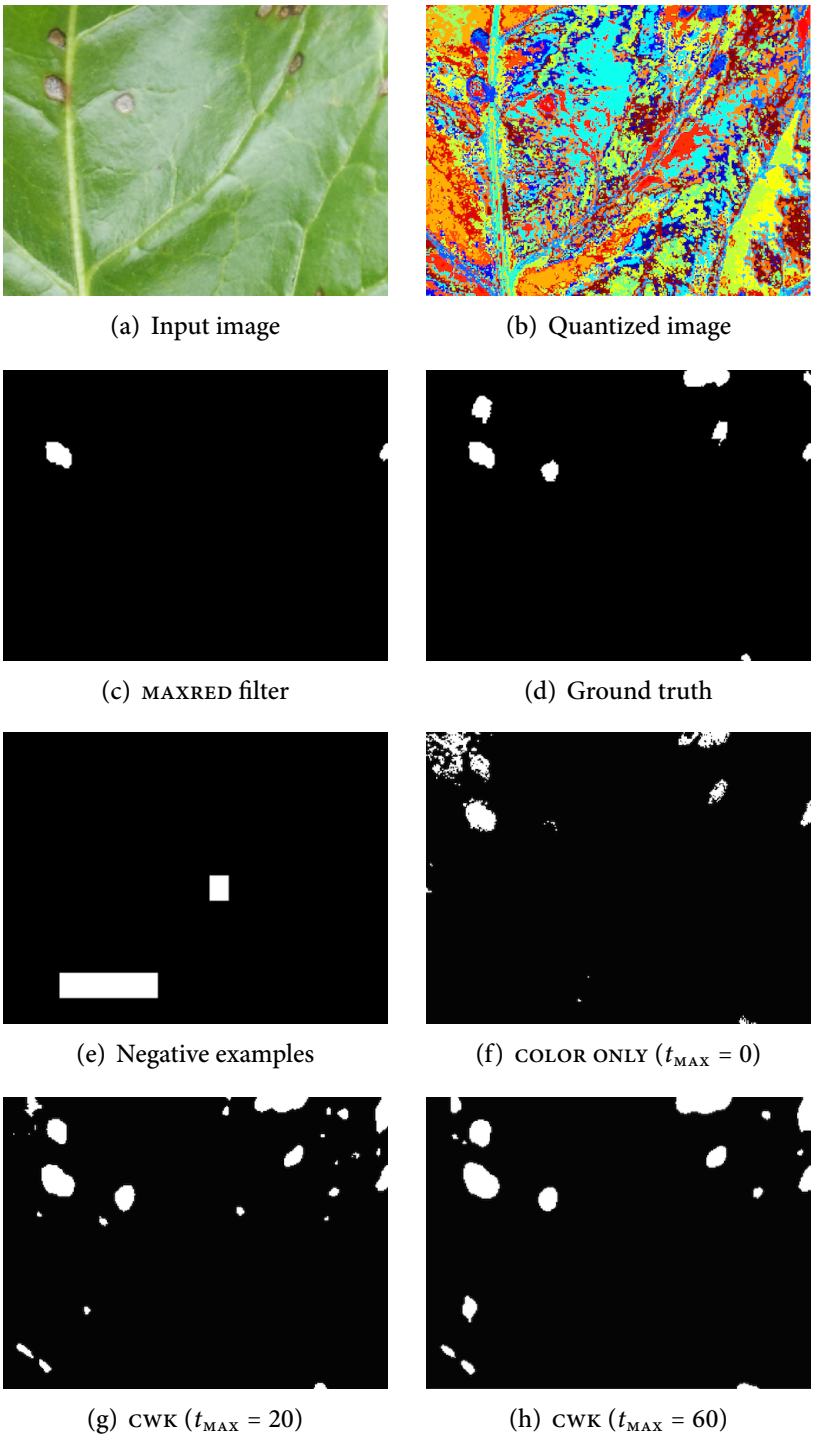
(a) Input image (b) Quantized image

(c) MAXRED filter (d) Ground truth

(e) Negative examples (f) COLOR ONLY (t_{MAX} = 0)

(g) CWK (t_{MAX} = 20) (h) CWK (t_{MAX} = 60)

Figure 10.4: **CWK for Leaf Spot Detection – Example 1.** Input image (a), quantized version (b), regions detected by MAXRED filter – also used as positive examples (c), ground truth (d), negative examples (d), and CWK results (f-h).



(a) Input image (b) Quantized image

(c) MAXRED filter (d) Ground truth

(e) Negative examples (f) COLOR ONLY (t_{MAX} = 0)

(g) CWK (t_{MAX} = 20) (h) CWK (t_{MAX} = 60)

Figure 10.5: **CWK for Leaf Spot Detection – Example 2.** Input image (a), quantized version (b), regions detected by MAXRED filter – also used as positive examples (c), ground truth (d), negative examples (d), and CWK results (f-h).

Table 10.2: **Dataset Statistics and Properties.** Number of graphs per class, total number of graphs, and median and total number of nodes for the two considered datasets TRAIN and PLANTS.

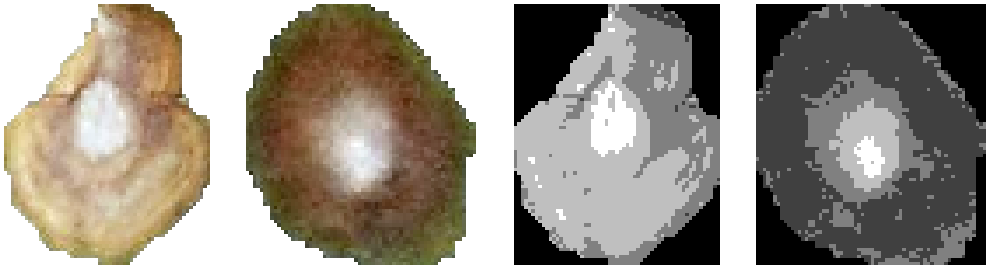
	class frequencies						statistics		
dataset	*cerc*	*ram*	*pseu*	*rust*	*pho*	*n-inf*	total # graphs	median # nodes	total # nodes
TRAIN	57	57	44	47	36	55	296	4 800	1 391 925
PLANTS	1 003	255	494	72	55	1 108	2 957	4 725	13 587 375

10.3 Plant Disease Classification with Propagation Kernels

In Chapter 5 we have seen that propagation kernels can be employed for pixel-image-based texture classification. The classification of disease regions extracted from images of plant leaves is another computer vision task relying on the comparison of color and intensity changes. Classical approaches capture these changes of pixel intensity values by computing high-dimensional statistical features based on the color level co-occurrence or based on gradient information (NEUMANN, *et al.*, 2014*b*). Here, we will use propagation kernels computed between pixel grid graphs for this task.

10.3.1 Datasets and Experimental Protocol

The images in PLANTS, introduced in (NEUMANN, *et al.*, 2014*b*), are regions showing disease symptoms extracted from a database of 495 RGB images of beet leaves taken with six different cell phone cameras. The dataset has six classes: five are leaf spots caused by the previously introduced pathogens Cercospora beticola (*cerc*), Ramularia beticola (*ram*), Pseudomonas syringae (*pseu*), Uromyces betae (*rust*), and Phoma betae (*pho*). To handle regions extracted by the region detector not belonging to one of these classes we consider a sixth class non-infected (*n-inf*). Example regions of this class could be healthy leaf parts, such as reflections or leaf veins, dirt on the leave, holes, or earth on the ground. These regions occur as a simple and efficient region detector was applied being feasible on the smart phone. We use 10% of the full data covering a balanced number of classes (296 regions) for parameter learning; this dataset is called TRAIN. The full dataset (PLANTS) will then be used for evaluation. Note that this dataset is highly imbalanced with two infrequent classes accounting for only 2% of the examples and two frequent classes covering 35% of the examples. The class frequencies of both datasets are listed in Table 10.2. For each region we form a graph according to the pixel grid as described in Section 5.1. To get discrete node labels we quantize the color values using a fixed number of equidistant levels. Figure 10.6(a) and (b) illustrates two regions and their quantized versions (c) and (d).



(a) phoma (b) cercospora (c) phoma-5 (d) cercospora-5

Figure 10.6: **Example Regions of PLANTS.** Example leaf spot images (a,b) and the corresponding quantized versions with 5 colors (c,d) of the two classes phoma and cercospora.

We use the following experimental protocol. Performance is measured by average accuracy and classification is achieved by running C-SVM using `libSVM`.[2] First, the PK parameters ($t_{\text{MAX}} \in \{0, 3, 5, 8, 10, 15, 20\}$ and bin width $w_l \in \{10^{-2}, 10^{-4}, 10^{-6}\}$), quantization value ($col \in \{3, 5, 8, 10, 15\}$), neighborhood ($B \in \{N_{1,4}, N_{1,8}, N_{2,16}\}$, cf. Equation (5.1)), and SVM-cost parameter $c \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ are learned on TRAIN via a 20-fold cross validation. For TRAIN the best performance was achieved with $t_{\text{MAX}} = 5$, $w_l = 10^{-4}$, $col = 5$, $N_{1,4}$, and $c = 10^{-4}$. Now, we evaluate PK on the full dataset PLANTS using these parameter values. The performance is averaged over 20 randomly generated but fixed train/test splits.

We compare PK to the simple baseline LABELS using label counts only and to a 36-dimensional second-order statistical feature[3] based on the gray-level co-occurrence matrix comparing intensities (GLCM-GRAY) resp. quantized labels (GLCM-QUANT) of neighboring pixels originally introduced in (HARALICK, *et al.*, 1973). Note that the computations were not feasible for other graph kernels such as the Weisfeiler–Lehman subtree kernel as the median number of nodes per graph on PLANTS is 4 725 and the total number of nodes exceeds 13.5 million.

10.3.2 Results

The experimental results on TRAIN and PLANTS are shown in Table 10.3. While not outperforming sophisticated approaches specifically tailored to the problem of leaf spot classification, we can show that it is feasible to compute PKs on the, for graph-kernel applications, huge image datasets. Further, on PLANTS PK achieves an average accuracy of 82.5% *out-of-the-box*. The best reported result on this

[2] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

[3] We computed the following statistics for four pixel offsets (pixel distance $d = 1$ and angles $\theta = \{0°, 45°, 90°, 135°\}$): angular second moment, correlation, entropy, inverse difference moment, sum entropy, difference entropy, sum variance, difference variance, and variance.

Table 10.3: **Results on TRAIN and PLANTS.** Average accuracies ± standard errors of 20-fold CV (10 reruns). Average runtimes in sec (x'') or min (x') given in parentheses refer to the kernel computation or feature computation for the learned parameter settings. For GLCM-QUANT and LABELS the same color quantization as for PK was applied. LABELS corresponds to PK with $t_{\text{MAX}} = 0$.

	dataset	
method	TRAIN	PLANTS
PK	**72.7** ± 0.4 (17.6″)	**82.5** ± 0.1 (3.0′)
LABELS	43.1 ± 0.0 (4.9″)	59.5 ± 0.0 (11.5″)
GLCM-GRAY	67.2 ± 0.0 (6.7″)	76.6 ± 0.0 (1.4′)
GLCM-QUANT	13.2 ± 0.0 (6.7″)	37.5 ± 0.0 (1.1′)

problem is 87.6% accuracy, which could only be achieved by a 960-dimensional feature ensemble of several first- and second-order statistics being computed locally – incorporating expert knowledge about the phenotype of the considered diseases – from the full color and gradient information (NEUMANN, *et al.*, 2014*b*). Comparing the performance of PK to only counting the quantized color labels, we see that for both datasets leveraging the arrangement of labels is clearly beneficial. Classical second-order statistics only analyzing the changes of the quantized pixel values of neighboring pixels (GLCM-QUANT) fail to predict the diseases accurately. GLCM-GRAY using the original intensity values performs also worse than PK. We can conclude that propagation kernels are a promising approach to plant disease classification.

10.4 Summary and Discussion

In this chapter, we presented a novel and important application domain for kernels from propagated information. Both, coinciding walk kernels and propagation kernels, are able to exploit the arrangement of color-based node information in pixel grid graphs for learning tasks on the node and on the graph level. Specifically, we showed that coinciding walk kernels seem promising for leaf spot detection given a small subset of training regions. The grid graph implementation of propagation kernels can be employed to plant disease classification out-of-the-box. Given these successful initial results and the importance of the problem this is a good starting point for a further investigation of propagation kernels in this domain. More elaborate input values such as local binary patterns or gradient information could be explored as node information. Further, propagation kernels can be computed without any modification for local image regions such as the recently introduced erosion bands (NEUMANN, *et al.*, 2014*b*). These adaptions will most likely boost the performance of PKs already significantly.

CONCLUSION

Learning with graphs, with its versatile approaches and challenging applications of growing importance, is an actively studied research area in computer science. As novel methods often either require in-depth technical understanding or are application driven, most approaches have so far been developed for either structured data, networked data, or relational data and models. However, after studying techniques and applications separately for several decades now, it is time to move towards a better understanding of what it means to learn with graph data by starting to incorporate insights from other learning domains.

Summary

We took a first step towards this goal by applying propagation techniques on graphs for learning tasks in structured, networked, *and* relational data. The main insight, leading to the results presented in this work, is that information propagation on graphs – so far only used for inference on the node level – captures graph structure, adapts to missing, uncertain, and continuous information, scales, and is efficiently computable. Thus, we can use it to design graph features and kernels for learning on the graph level, learning on the node level, and learning in relational domains. In this way, we were able to develop a graph kernel for learning with structured data that naturally copes with missing and uncertain information, a kernel between the nodes of a graph for learning with networked data that exploits structural similarities in addition to the homophily assumption, and a set-completion algorithm that leverages structural information in relational data.

Specifically, in Part I we introduced *propagation kernels* for learning on the graph level, and in Part II *coinciding walk kernels* for learning on the node level and *Markov logic sets* for graph-based information retrieval in relational data. Furthermore, by design kernels from propagated information have two valuable benefits: they are flexible and scale to large graphs. Thus, we were able to apply them to novel application domains in Part III. Specifically, we applied propagation kernels to object category prediction and image-based plant disease classification, and we studied the use of coinciding walk kernels for leaf spot detection in plant images.

Future Work

Kernels from propagated information are already a promising learning paradigm for graph data. However, there are several interesting technical aspects and extensions worth exploring. In the following, we discuss these in more detail.

Technical Future Work

In both of our kernel frameworks, propagation kernels and coinciding walk kernels, it would be beneficial to understand and automatically learn which information – graph structure or attribute/label information – is most important for prediction tasks. Moreover, learning the number of iterations used to propagate information for both kernels is time consuming. To guide parameter learning or even make it dispensable, the design of heuristics, along the lines of those introduced in previous work on random-walk based clustering (LIN and COHEN, 2010*a*), is an important open question. Further insights in understanding these parameters would also be beneficial for other iterative graph kernels (SHERVASHIDZE, *et al.*, 2011) and learning approaches using structural similarities (DESROSIERS and KARYPIS, 2009).

A similar concern holds for ranking-based set completion, where we have not yet addressed in detail how to set the size of the completion. Since we use label propagation, using zero as the threshold would be a natural choice, but other thresholds are possible. An interesting extension to Markov logic sets would be to apply inductive logic programming techniques (MUGGLETON, 1996) on the retrieved completion to extract a general description of the latent concept. Using dynamic PageRank techniques (ROSSI and GLEICH, 2012) would pave the way to online clustering on demand and is definitely a growth path for lifted inference. Since we were able to show that label propagation can be performed on the lifted network, which is itself is an advance, further experimentation and applications exploiting computational symmetries for graph based learning via lifted inference is another important direction.

When considering benchmark data for evaluation and real-world applications, it is essential to gain a good understanding of the problems, and the graph representations used have to be sensible and suitable for the learning method applied. For example, biological interactions giving rise to protein–protein interaction networks can vary widely in their nature and are spatially and temporally heterogeneous. As a result, abstract representations of such networks and their analysis need to account for the expressiveness of the interactions (HAKES, *et al.*, 2008). In general, graph construction is not a well-studied problem. Specific open questions are: how to construct a reliable graph from partially observed data, how to construct graphs based on dissimilarities, and how to make a robust graph when multiple evidence gives rise to possibly inconsistent graph structure (LIU, 2006).

Open Research Questions

Beside these more technical issues, open questions of broader scope are: further challenges of graph data beyond the ones discussed in this thesis, memory-efficient representation of graph structure and compatible time-efficient algorithms for learning with graphs, and a deeper exploration of the close relation of graph-based and semi-supervised learning, graph kernels, and relational learning.

Whereas in this work, we tackled the challenges of missing information on graphs and/or nodes, and uncertain and continuous node annotations for the construction of graph kernels, kernels for graph data do not consider missing edges or dynamic graphs with evolving node and edge structure (RAYMOND and KASHIMA, 2010; LIKHACHEV, *et al.*, 2008; GRAHAM, 2004). The central open question, not only for graph construction, but also for feature construction, kernel design, and learning is: how can we deal with dynamic networks/graphs? These are for example common in social networks, where new users sign up and new friendship relations are established or deleted, and how can we address uncertainty in the graph structure, occurring, for example, in web graphs created from noisy hyperlinks (LIU, 2006). By introducing graph-based relational set completion, we provided an initial approach to connecting learning with graphs and relational data. Further exploiting the rich structure in relational data, beyond binary relations encoded in predicates, types, variables, and formulas, would most-likely be beneficial for learning with structured and networked data (DICK and KERSTING, 2006).

As indicated by the real-world application scenarios in Part III of this thesis, an essential goal of future research on learning with graphs should be to improve the efficiency of feature extraction, kernel construction, and learning. Towards scaling machine learning with graphs to "big data", we believe that three techniques are worthwhile to be explored: *parallel computing*, the *exploitation of symmetries*, and *graph approximation*. In Chapter 5, we have shown how to leverage the regular structure in gird graphs to scale graph kernels to large graph databases with a total number of millions of nodes. In the same spirit, techniques to exploit symmetries (AHMADI, 2014) should be explored for both graph kernels and kernels on graphs. Architectures that allow for parallel algorithms to analyze graphs have recently been introduced (LOW, *et al.*, 2014; MALEWICZ, *et al.*, 2010). Whereas popular algorithms such as PageRank can be implemented easily, kernel construction and kernel-based learning for graph data has not been considered yet. Graph approximation (LESKOVEC and FALOUTSOS, 2007) and matrix factorization (YANG, *et al.*, 2012; MENON and ELKAN, 2011; SUN, *et al.*, 2008) are other promising techniques to cope with huge graphs. While matrix factorization has been applied to kernel matrices (ZHANG, *et al.*, 2006; FINE and SCHEINBERG, 2002) and matrix factorization itself has been kernelized (GÖNEN and KASKI, 2014), to the best of our knowledge neither graph kernels nor kernels on graphs from approximated graphs have been derived and studied.

In this thesis, we exploited that graph kernels and graph-based learning (learning on the node level) are closely related. We developed a graph kernel for predictive graph mining by leveraging the intermediate results of information propagation on the input graphs – a technique traditionally introduced for graph-based and semi-supervised learning – and we introduced a kernel between the nodes of a graph that exploits structural similarities. A natural extension to this work is the derivation of a unifying propagation-based framework for structure representation

independent of the learning task being on the graph or on the node level. Such an approach could open the door for joint inference for node-level *and* graph-level tasks. Further, we could study which information is important for which task and thus data acquisition and learning procedures could be designed accordingly.

As shown in this work the propagation kernel framework is a flexible and powerful approach to learning from structured data. An interesting avenue for future extensions are the development of kernels among graphical models. As propagation kernels between graphs leverage graph-based inference techniques, a propagation scheme to realize kernels between graphical models could be belief propagation, an iterative message passing algorithm to infer marginal distributions of the variables in graphical models (YEDIDIA, *et al.*, 2003). BACH and JORDAN (2002) use kernels among variables to learn the structure of graphical models. In a similar spirit, comparing graphical models with kernels might also be useful for structure learning. Recently, COHEN (2010) established a connection between acyclic graphical models and graph-based learning. In particular, it was shown that the PageRank solution of a suitable graph representation corresponds to the single node marginals in the graphical model. Moreover, approaches to interlink statistical relational learning and (kernel-based) regression were introduced (FRASCONI, *et al.*, 2014; SCHIEGG, *et al.*, 2012; CHOPRA, 2008). These connections indicate that it is possible to move from graph-based learning in relational data to (graph- and kernel-based) learning in and among relational models.

Lessons Learned

In most parts of this thesis, with exception of Chapter 8, we have focused on kernels for machine learning with graph data. Whereas a great amount of graph kernels has been proposed in the recent years, it is not clear whether this is the best way to approach learning with structured data. Only recently explicit and implicit computation schemes of graph kernels have been started to be analyzed theoretically and empirically (KRIEGE, *et al.*, 2014). Note that explicit computation can be actually seen as feature extraction from graph inputs, and the computation of explicit graph kernels is often more efficient and usually yields the same performance as implicit kernels. Also in other research fields, such as computer vision, high-dimensional feature maps are highly successful. So, why did we never ask whether we actually need to focus on kernel design or if it is more promising to concentrate on feature extraction for graph data? The focus of learning with graphs could benefit from an in-depth discussion of this question and maybe instead of overly focusing on the development of novel kernels, we should start integrating graph data and learning methods in general. So, the focus of future research should be bringing together learning methods and feature extraction, where feature extraction subsumes kernel construction.

Another conclusion from this thesis is that graph structure can be measured by running inference in graphs representing structured, networked, and relational

data. On the other hand, kernels for graph data are based on graph structure. So, a more general view of this work could lead to the development of kernels from inference problems in general. This would have the benefit that we do not have to care about the format of the input data, but the inference procedure(s) performed can themselves lead to kernels for improved learning and prediction.

APPENDIX

APPENDIX A

NOTES ON A SIMILARITY MEASURE BASED ON PARALLEL RWS

The similarity $R_{u,v}^{(t_{\text{MAX}})}$ in (DESROSIERS and KARYPIS, 2009) is defined to be the probability that parallel random walks leaving from u and v:

- have the same length,
- have length less than or equal to t_{MAX}, and
- encounter exactly the same label sequence before termination.

Define the random variable t_u to be the length of a walk leaving from node u, where "length" indicates the number of transitions made before termination.

The walks considered have a constant termination probability of γ. Therefore the probability of a walk having exactly length t is independent of u. The exact value is $\gamma(1-\gamma)^t$, because it must transition t times and terminate once:

$$\begin{aligned}\Pr(t_u = t \mid u, t, \gamma) &= \Pr(t_u = t \mid t, \gamma) \\ &= \gamma(1-\gamma)^t.\end{aligned}$$

The walks are independent, so the probability that both walks have exactly the length t is simply this quantity squared:

$$\begin{aligned}\Pr(t_u = t_v = t \mid u, v, t, \gamma) &= \Pr(t_u = t \mid t, \gamma)\Pr(t_v = t \mid t, \gamma) \\ &= \gamma^2(1-\gamma)^{2t}.\end{aligned}$$

Finally, to find the probability that both walks are the same length and have length less than or equal to t_{MAX}, we take the sum over all allowable walk lengths:

$$\begin{aligned}\Pr\big((t_u = t_v) \wedge (t_u \le t_{\text{MAX}}) \wedge (t_v \le t_{\text{MAX}}) \mid u, v, t_{\text{MAX}}, \gamma\big) &= \sum_{t=0}^{t_{\text{MAX}}} \Pr(t_u = t_v = t \mid t, \gamma) \\ &= \gamma^2 \sum_{t=0}^{t_{\text{MAX}}} (1-\gamma)^{2t}.\end{aligned}$$

We now have the probability of two of the three conditions above. As for CWKS, cf. Equation (7.4), we may compute the probability that the labels encountered on

the t-th step of parallel random walks leaving from u and v are equal from the P_t matrix, cf. label diffusion Equation (2.10). Define the random variable u_t to be the node visited on the t-th step by a random walk leaving from u (so u_0 is always u).

$$\Pr\big(\ell(u_t) = \ell(v_t) \mid u, v, t\big) = \big[P_t P_t^\top\big]_{u,v}.$$

If we are given the lengths of both walks, t, then we may find the probability that all labels encountered are equal by taking a product:

$$\Pr\left(\bigwedge_{t=0}^{t_{\text{MAX}}} \ell(u_t) = \ell(v_t) \mid u, v, t\right) = \left[\prod_{t=0}^{t}{}_{\text{MAX}} P_t P_t^\top\right]_{u,v},$$

where the $\prod$ indicates a Hadamard[1] (also Schur[2]) product of matrices. The Hadamard product is the element-wise product of matrices.

Finally, combining all three conditions, we may write the entire $R^{(t_{\text{MAX}})}$ matrix as

$$R^{(t_{\text{MAX}})} = \gamma^2 \sum_{t=0}^{t}{}_{\text{MAX}} (1-\gamma)^{2t} \prod_{\tilde{t}=0}^{t} P_{\tilde{t}} P_{\tilde{t}}^\top,$$

which is positive definite because the set of positive-definite matrices is closed under sums and Hadamard products. The latter is given by the Schur product theorem stating that the Hadamard product of two positive-definite matrices is again a positive-definite matrix (SCHUR, 1911).

[1] Jacques Hadamard (1865–1963) was a French mathematician contributiong to function theroy, differential geometry and partial differential equations.

[2] Issai Schur (1875–1941) was Jewish mathematician – born in Russia and working in Germany – contributing to group theory, algebra, and theoretical physics.

APPENDIX B

GEOMETRY: NOTES ON REPRESENTING CURVATURE

Intuitively curvature can be understood as how smooth the surface of a curve or a point cloud is in Euclidean space. The straight line or the (infinite) (hyper)plane, for example, has zero curvature. For our data representation we are interested in measuring the change in curvature at a certain location on the surface of a point cloud in 3D Euclidean space. This can be achieved by comparing the respective surface normals of two or several points.

So, given two points represented by 3-dimensional column vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$, we denote their normals by $\mathbf{n}_u$ and $\mathbf{n}_v$, where $\mathbf{n}_u, \mathbf{n}_v \in \mathbb{R}^3$. The surface normal of a point is the same as the normal to the tangent plane of the point. We consider unit length normals, that is $|\mathbf{n}_u| = \sqrt{\mathbf{n}_u \cdot \mathbf{n}_u} = 1$ and we denote the scalar or dot product beween two vectors by $\cdot$, where for $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ it holds that $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^\top \mathbf{b}$. Note that the direction of a normal is not unique as the vector pointing in the opposite direction of a normal is also a normal. Algorithms computing the normals of points in a 3D point cloud, try to adjust the directions to be smooth among neighboring points, however, we cannot guarantee that the normals point in the same direction.

In the graph representation of 3D point clouds we employ curvature information twofold. On the one hand, we can model the change of the curvature between two points by *edge weights*. On the other hand, we can also assign a curvature *attribute* to each node in the graph. That is, for each point we will approximate its curvature by its approximate derivative, which is given by the mean change in curvature from the respective point to its neighbors.

In the first case our goal is to have smaller weights for larger changes in curvature. So, a measure of the change in curvature is the angle between the (unit length) normals of two points,

$$\mathbf{n}_u \cdot \mathbf{n}_v = \cos\theta \quad \Leftrightarrow \quad \arccos \mathbf{n}_u \cdot \mathbf{n}_v = \theta. \tag{C.1}$$

As the cosine is a strictly monotonically decreasing function on the interval $[0, \pi]$, corresponding range to angles between 0° and 180°, we can directly use $\mathbf{n}_u \cdot \mathbf{n}_v \in [-1, 1]$ to represent the edge weight between two points $\mathbf{u}$ and $\mathbf{v}$. Cosine plotted on the interval $[0, \pi]$ is shown in Figure C.1. To guarntee that the weights are posititve we could now add 1 to the cosine of the angle between the normals of the nodes u and v corresponding to the points $\mathbf{u}$ and $\mathbf{v}$,

$$w_{u,v} = \mathbf{n}_u \cdot \mathbf{n}_v + 1. \tag{C.2}$$

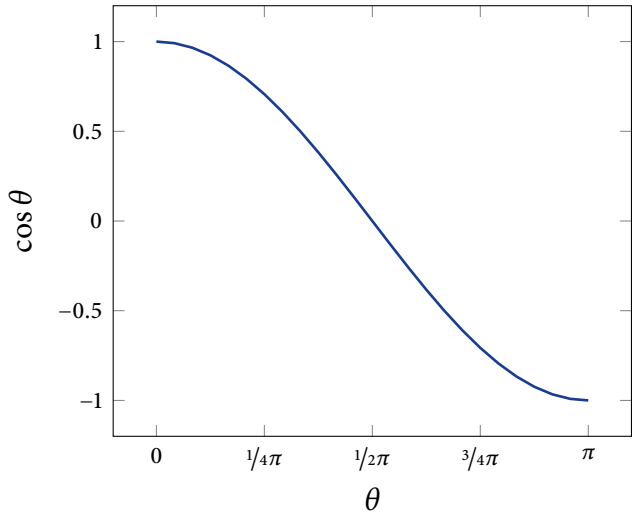


Figure C.1: **Cosine Function.** Strictly monotonically decreasing cosine function on the interval $[0, \pi]$ with values in $[-1, 1]$.

However, Equation (C.2) has one problem. As we cannot guarantee that the normals are pointing in the same direction we could measure "wrong" angles and therefore get wrong edge weights. This could then lead to edge weights indicating a drastic change in curvature in actually smooth regions. To avoid this problem, we assume that the angles between the normals of two adjacent points is not larger than 90°, that is, $\theta \in [0, 1/2\pi]$. With this assumption $\mathbf{n}_u \cdot \mathbf{n}_v$ now takes values in $[0, 1]$ if we compare normals pointing in the same direction and we get values in $[-1, 0]$ if we compare normals pointing in opposite directions. The magnitudes of the values are the same. So, to avoid comparing "wrong" normals we take the norm of the cosine of the angle as edge weight

$$w_{uv} = |\mathbf{n}_u \cdot \mathbf{n}_v|. \tag{C.3}$$

For node attributes we approaximate the change of curvature in one point by the average change of curvature between that point and its neighbors, where in our representation neighborhood is given by the k-nn graph. By taking the same assuption as for the edge weights that the normals of two incident point should point in the same direction, the attribute x_u for node u is given by

$$x_u = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} |\mathbf{n}_u \cdot \mathbf{n}_v|, \tag{C.4}$$

where $\mathcal{N}(u)$ are all nodes adjacent to u in the graph.

BIBLIOGRAPHY

Agrios, G. (2005). *Plant Pathology*. 5th edn. Elsevier Inc.

Ahmadi, B. (2014). *Graphical Models and Symmetries: Loopy Belief Propagation Approaches*. Ph.D. thesis, University of Bonn, Bonn, Germany. urn:nbn:de:hbz:5n-37209.

Ahmadi, B., Kersting, K. and Sanner, S. (2011). Multi-Evidence Lifted Message Passing, with Application to PageRank and the Kalman Filter. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (ijcai–2011)*, pp. 1152–1158.

Al-Hiary, H., Bani-Ahmad, S., Reyalat, M., Braik, M. and Alrahamneh, Z. (2011). Fast and Accurate Detection and Classification of Plant Diseases. *International Journal of Computer Applications*, vol. 17, no. 1, pp. 31–38.

Allison, P. (2001). Pompeian Households: An Analysis of the Material Culture. Tech. Rep. 42.DG70.P7 A645, Cotsen Institute of Archaeology, Los Angeles, CA, US.

Andersen, R., Chung, F.R.K. and Lang, K.J. (2006). Local Graph Partitioning using PageRank Vectors. In: *47th Annual ieee Symposium on Foundations of Computer Science (focs–2006)*, pp. 475–486.

Antanas, L., Frasconi, P., Costa, F., Tuytelaars, T. and De Raedt, L. (2012). A Relational Kernel-Based Framework for Hierarchical Image Understanding. In: *Joint iapr International Workshop on Structural, Syntactic, and Statistical Pattern Recognition (sspr/spr–2012)*, pp. 171–180.

Antanas, L., Moreno, P., Neumann, M., de Figueiredo, R.P., Kersting, K., Santos-Victor, J. and De Raedt, L. (2014). High-level Reasoning and Low-level Learning for Grasping: A Probabilistic Logic Pipeline. arXiv:1411.1108v1 [cs.RO].

Aronszajn, N. (1950). Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, vol. 68, no. 3, pp. 337–404.

Bach, F.R. (2008). Graph Kernels between Point Clouds. In: *Proceedings of the 25th International Conference on Machine Learning (icml–2008)*, pp. 25–32.

Bach, F.R. and Jordan, M.I. (2002). Learning Graphical Models with Mercer Kernels. In: *Advances in Neural Information Processing Systems 15 (nips–2002)*, pp. 1009–1016.

Backstrom, L. and Leskovec, J. (2011). Supervised Random Walks: Predicting and Recommending Links in Social Networks. In: *Proceedings of the 4th International Conference on Web Search and Web Data Mining (wsdm–2011)*, pp. 635–644.

Bai, L., Rossi, L., Torsello, A. and Hancock, E.R. (2014). A quantum Jensen–Shannon graph kernel for unattributed graphs. *Pattern Recognition*. To appear.

Bauckhage, C. and Kersting, K. (2013). Efficient Information Theoretic Clustering on Discrete Lattices. `arXiv:1310.7114 [cs.CV]`.

Bengio, Y., Delalleau, O. and Le Roux, N. (2006). Label Propagation and Quadratic Criterion. In: Chapelle, O., Schölkopf, B. and Zien, A. (eds.), *Semi-Supervised Learning*, pp. 193–216. MIT Press.

Bishop, C.M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer Science + Business Media, LLC.

Bock, C.H., Poole, G.H., Parker, P.E. and Gottwald, T.R. (2010). Plant Disease Severity Estimated Visually, by Digital Photography and Image Analysis, and by Hyperspectral Imaging. *Critical Reviews in Plant Sciences*, vol. 29, no. 2, pp. 59–107.

Bohg, J. and Kragic, D. (2010). Learning Grasping Points with Shape Context. *Robotics and Autonomous Systems*, vol. 58, no. 4, pp. 362–377.

Boldi, P., Lonati, V., Santini, M. and Vigna, S. (2006). Graph fibrations, graph isomorphism and PageRank. *rairo – Theoretical Informatics and Applications*, vol. 40, no. 2.

Bollobás, B. (1998). *Modern Graph Theory*. Springer Science + Business Media, Inc.

Borgwardt, K.M. and Kriegel, H.-P. (2005). Shortest-Path Kernels on Graphs. In: *5th ieee International Conference on Data Mining (icdm–2005)*, pp. 74–81.

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. In: *Proceedings of the 13th International Conference on Intelligent Systems for Molecular Biology (ismb–2005)*, pp. 47–56.

Boser, B.E., Guyon, I.M. and Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In: *Proceedings of the 5th Annual acm Workshop on Computational Learning Theory (colt–1992)*, pp. 144–152. acm Press.

Boutell, M.R., Luo, J., Shen, X. and Brown, C.M. (2004). Learning multi-label scene classification. *Pattern Recognition*, vol. 37, no. 9, pp. 1757–1771.

Boykov, Y., Veksler, O. and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239.

Brefeld, U., Getoor, L. and Macskassy, S.A. (eds.) (2010). *Eighth Workshop on Mining and Learning with Graphs (mlg–2010)*, sigkdd Explorations, vol. 12, no. 2.

Brin, S. and Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*, vol. 30, no. 1-7, pp. 107–117.

Camargo, A. and Smith, J. (2009*a*). Image pattern classification for the identification of disease causing agents in plants. *Computers and Electronics in Agriculture*, vol. 66, no. 2, pp. 121–125.

Camargo, A. and Smith, J. (2009*b*). An image-processing based algorithm to automatically identify plant disease visual symptoms. *Biosystems Engineering*, vol. 102, no. 1, pp. 9–21.

Chakrabarti, D. and Faloutsos, C. (2012). *Graph Mining: Laws, Tools, and Case Studies*. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers.

Chapelle, O., Schölkopf, B. and Zien, A. (eds.) (2006). *Semi-Supervised Learning*. MIT Press.

Chapelle, O., Sindhwani, V. and Keerthi, S.S. (2008). Optimization Techniques for Semi-Supervised Support Vector Machines. *Journal of Machine Learning Research*, vol. 9, pp. 203–233.

Chopra, S.P. (2008). *Factor Graphs for Relational Regression*. Ph.D. thesis, New York University, New York, NY, US. 978-1-109-90255-6.

Chung, F. (1997). *Spectral Graph Theory*. American Mathematical Society.

Cohen, W.W. (2010). Graph Walks and Graphical Models. Tech. Rep. CMU-ML-10-102, Carnegie Mellon University, Pittsburgh, US.

Cook, D.J. and Holder, L.B. (2006). *Mining Graph Data*. John Wiley & Sons, Inc.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge University Press.

Cristianini, N., Shawe-Taylor, J., Elisseeff, A. and Kandola, J.S. (2001). On Kernel–Target Alignment. In: *Advances in Neural Information Processing Systems 14 (nips–2001)*, pp. 367–373.

Datar, M., Immorlica, N., Indyk, P. and Mirrokni, V.S. (2004). Locality-sensitive Hashing Scheme Based on p-Stable Distributions. In: *Proceedings of the 20th Annual Symposium on Computational Geometry (scg–2004)*, pp. 253–262.

De Knijf, J., Liekens, A.M.L. and Goethals, B. (2011). "Tell Me More": Finding Related Items from User Provided Feedback. In: *14th International Conference on Discovery Science (ds–2011)*, pp. 76–90.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer-Verlag Berlin Heidelberg.

Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Schusterman, A.J. and Hansch, C. (1991). Structure–Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797.

Depiero, F., Trivedi, M. and Serbin, S. (1996). Graph matching using a direct classification of node attendance. *Pattern Recognition*, vol. 29, no. 6, pp. 1031–1048.

Desrosiers, C. and Karypis, G. (2009). Within-Network Classification Using Local Structure Similarity. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ecml/pkdd–2009)*, pp. 260–275.

Dick, U. and Kersting, K. (2006). Fisher Kernels for Relational Data. In: *Proceedings of the 17th European Conference on Machine Learning (ecml–2006)*, pp. 114–125.

Dobson, P.D. and Doig, A.J. (2003). Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology*, vol. 330, no. 4, pp. 771–783.

Duchenne, O., Joulin, A. and Ponce, J. (2011). A Graph-Matching Kernel for Object Categorization. In: *13th ieee International Conference on Computer Vision (iccv–2011)*, pp. 1792–1799.

Duncan, K.E. and Howard, R.J. (2000). Cytological analysis of wheat infection by the leaf blotch pathogen Mycosphaerella graminicola. *Mycological Research*, vol. 104, no. 9, pp. 1074–1082.

Džeroski, S. and Lavrač, N. (2001). An Introduction to Inductive Logic Programming. In: Džeroski, S. and Lavrač, N. (eds.), *Relational Data Mining*, pp. 48–72. Springer-Verlag Berlin Heidelberg.

Editors of the American Heritage Dictionaries (ed.) (2000 January). *The American Heritage Dictionary of the English Language*. Houghton Mifflin Company.

Elkan, C. and Noto, K. (2008). Learning Classifiers from Only Positive and Unlabeled Data. In: *Proceedings of the 14th acm sigkdd International Conference on Knowledge Discovery and Data Mining (kdd–2008)*, pp. 213–220.

Farid, R. and Sammut, C. (2013). Plane-based object categorization using relational learning. *Machine Learning*, vol. 94, no. 1, pp. 1–21.

Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M. and Borgwardt, K.M. (2013). Scalable kernels for graphs with continuous attributes. In: *Advances in Neural Information Processing Systems 26 (nips–2013)*, pp. 216–224.

Fine, S. and Scheinberg, K. (2002). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, vol. 2, pp. 243–264.

Flach, P.A. (1994). *Simply Logical – Intelligent Reasoning by Example*. Wiley Professional Computing. John Wiley & Sons, Inc.

Fortunato, S. (2010). Community detection in graphs. `arXiv:0906.0612v2 [physics.soc-ph]`.

Fouss, F., Françoisse, K., Yen, L., Pirotte, A. and Saerens, M. (2012). An Experimental Investigation of Kernels on Graphs for Collaborative Recommendation and Semisupervised Classification. *Neural Networks*, vol. 31, pp. 53–72.

Frasconi, P., Costa, F., De Raedt, L. and De Grave, K. (2014). kLog: A Language for Logical and Relational Learning with Kernels. *Artificial Intelligence*, vol. 217, pp. 117–143.

Frasconi, P., Kersting, K. and Tsuda, K. (eds.) (2007). *Fifth Workshop on Mining and Learning with Graphs (mlg–2007)*.

Gallagher, B. (2006). Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. Tech. Rep., aaai FS-06-02, pp. 45–53.

Gallagher, B. and Eliassi-Rad, T. (2008). Leveraging Label-Independent Features for Classification in Sparsely Labeled Networks: An Empirical Study. In: *Second International Workshop on Advances in Social Network Mining and Analysis (SNAKDD–2008), Revised Selected Papers*, pp. 1–19.

Gallagher, B., Tong, H., Eliassi-Rad, T. and Faloutsos, C. (2008). Using Ghost Edges for Classification in Sparsely Labeled Networks. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD–2008)*, pp. 256–264.

Garnett, R., Gärtner, T., Ellersiek, T., Gumondsson, E. and Óskarsson, P. (2014). Predicting Unexpected Influxes of Players in EVE Online. In: *IEEE Conference on Computational Intelligence and Games (CIG–2014)*.

Gärtner, T. (2005). Predictive Graph Mining with Kernel Methods. In: Bandyopadhyay, S., Maulik, U., Holder, L.B. and Cook, D.J. (eds.), *Advanced Methods for Knowledge Discovery from Complex Data*, Advanced Information and Knowledge Processing, pp. 95–121. Springer Science + Business Media.

Gärtner, T., Flach, P.A., Kowalczyk, A. and Smola, A.J. (2002). Multi-Instance Kernels. In: *Proceedings of the 19th International Conference on Machine Learning (ICML–2002)*, pp. 179–186.

Gärtner, T., Flach, P.A. and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In: *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop (COLT/KERNEL–2003)*, pp. 129–143.

Gersho, A. and Gray, R. (1991). *Vector Quantization and Signal Compression*. Kluwer Academic Publishers.

Getoor, L. and Taskar, B. (eds.) (2007). *Introduction to Statistical Relational Learning*. Adaptive Computation and Machine Learning. MIT Press.

Ghahramani, Z. and Heller, K.A. (2005). Bayesian Sets. In: *Advances in Neural Information Processing Systems 18 (NIPS–2005)*, pp. 1351–1358.

Gkirtzou, K., Honorio, J., Samaras, D., Goldstein, R. and Blaschko, M. (2013). fMRI Analysis with Sparse Weisfeiler–Lehman Graph Statistics. In: *4th International Workshop on Machine Learning in Medical Imaging (MLMI–2013)*, pp. 90–97.

Gönen, M. and Kaski, S. (2014). Kernelized Bayesian Matrix Factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 10, pp. 2047–2060.

Gonzalez, J.A., Holder, L.B. and Cook, D.J. (2002). Graph-Based Relational Concept Learning. In: *Proceedings of the 19th International Conference on Machine Learning (icml–2002)*, pp. 219–226.

Gori, M., Maggini, M. and Sarti, L. (2005). Exact and approximate graph matching using random walks. *ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 7, pp. 1100–1111.

Graham, F.C. (2004). Large Dynamic Graphs: What Can Researchers Learn from Them? *siam News*, vol. 37, no. 3.

Hadzic, F., Tan, H. and Dillon, T.S. (2010). *Mining of Data with Complex Structures*, chap. 11: Graph Mining, pp. 287–300. Studies in Computational Intelligence. Springer-Verlag Berlin Heidelberg.

Hakes, L., Pinney, J.W., Robertson, D.L. and Lovell, S.C. (2008). Protein–protein interaction networks and biology – what's the connection? *Nature Biotechnology*, vol. 26, no. 1, pp. 69–72.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*, chap. 9: Graph Mining, Social Network Analysis and Multirelational Data Mining, pp. 535–589. Morgan Kaufmann.

Haralick, R.M., Shanmugam, K. and Dinstein (1973). Textural Features for Image Classification. *ieee Transactions on Systems, Man and Cybernetics*, vol. smc–3, no. 6, pp. 610–621.

Harchaoui, Z. and Bach, F. (2007). Image Classification with Segmentation Graph Kernels. In: *ieee Conference on Computer Vision and Pattern Recognition (cvpr–2007)*.

Hastie, T., Tibshirani, R. and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd edn. Springer Science + Business Media, LLC.

Haussler, D. (1999). Convolution Kernels on Discrete Structures. Tech. Rep., University of California, Santa Cruz, US.

Haveliwala, T., Kamvar, S. and Jeh, G. (2003). An Analytical Comparison of Approaches to Personalizing PageRank. Tech. Rep. 2003-35, Stanford InfoLab, Stanford, US.

Heckmann, T. and Schwanghart, W. (2013). Geomorphic coupling and sediment connectivity in an alpine catchment – Exploring sediment cascades using graph theory. *Geomorphology*, vol. 182, no. 0, pp. 89–103.

Horváth, T., Gärtner, T. and Wrobel, S. (2004). Cyclic pattern kernels for predictive graph mining. In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD–2004)*, pp. 158–167.

Hotho, A., Jäschke, R., Schmitz, C. and Stumme, G. (2006). Information Retrieval in Folksonomies: Search and Ranking. In: *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference (ESWC–2006)*, pp. 411–426.

Huang, Z., Zeng, D.D. and Chen, H. (2007). Analyzing Consumer–Product Graphs: Empirical Findings and Applications in Recommender Systems. *Management Science*, vol. 53, no. 7, pp. 1146–1164.

Hwang, T. and Kuang, R. (2010). A Heterogeneous Label Propagation Algorithm for Disease Gene Discovery. In: *Proceedings of the 10th SIAM International Conference on Data Mining (SDM–2010)*, pp. 583–594.

Imrich, W. and Klavžar, S. (2000). *Product Graphs, Structure and Recognition*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc.

Itti, L. and Baldi, P. (2005). Bayesian Surprise Attracts Human Attention. In: *Advances in Neural Information Processing Systems 18 (NIPS–2005)*, pp. 265–272.

Jaakkola, T. and Haussler, D. (1998). Exploiting Generative Models in Discriminative Classifiers. In: *Advances in Neural Information Processing Systems 11 (NIPS–1998)*, pp. 487–493.

Jähne, B. (2005). *Digital Image Processing*. 6th edn. Springer-Verlag Berlin Heidelberg.

Jebara, T., Kondor, R. and Howard, A. (2004). Probability Product Kernels. *Journal of Machine Learning Research*, vol. 5, pp. 819–844.

Jensen, D., Neville, J. and Gallagher, B. (2004). Why Collective Inference Improves Relational Classification. In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD–2004)*, pp. 593–598.

Ji, M. and Han, J. (2012). A Variance Minimization Criterion to Active Learning on Graphs. In: *15th International Conference on Artificial Intelligence and Statistics (AISTATS–2012)*, JMLR Workshop and Conference Proceedings, pp. 556–564.

Jiang, C., Coenen, F., Sanderson, R. and Zito, M. (2010). Text Classification using Graph Mining-based Feature Extraction. *Knowledge-Based Systems*, vol. 23, no. 4, pp. 302–308.

Joachims, T. (1999). Transductive Inference for Text Classification using Support Vector Machines. In: *Proceedings of the 16th International Conference on Machine Learning (ICML–1999)*, pp. 200–209.

Joachims, T. (2003). Transductive Learning via Spectral Graph Partitioning. In: *Proceedings of the 20th International Conference on Machine Learning (ICML–2003)*, pp. 290–297.

Kajdanowicz, T., Kazienko, P. and Doskocz, P. (2010). Label-Dependent Feature Extraction in Social Networks for Node Classification. In: *Second International Conference on Social Informatics (SOCINFO–2010)*, pp. 89–102.

Kashima, H., Tsuda, K. and Inokuchi, A. (2003). Marginalized Kernels Between Labeled Graphs. In: *Proceedings of the 20th International Conference on Machine Learning (ICML–2003)*, pp. 321–328.

Kemmler, M., Rodner, E., Wacker, E.-S. and Denzler, J. (2013). One-class classification with Gaussian processes. *Pattern Recognition*, vol. 46, no. 12, pp. 3507–3518.

Kersting, K., Ahmadi, B. and Natarajan, S. (2009). Counting Belief Propagation. In: *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI–2009)*, pp. 277–284.

Kersting, K., Mladenov, M., Garnett, R. and Grohe, M. (2014). Power Iterated Color Refinement. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI–2014)*, pp. 1904–1910.

Kimmig, A., De Raedt, L. and Toivonen, H. (2007). Probabilistic Explanation Based Learning. In: *Proceedings of the 18th European Conference on Machine Learning (ECML–2007)*, pp. 176–187.

Kittler, J., Christmas, W.J., de Campos, T., Windridge, D., Yan, F., Illingworth, J. and Osman, M. (2014). Domain anomaly detection in machine perception: A system architecture and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 5, pp. 845–859.

Kleinberg, J.M. (1999). Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, vol. 46, no. 5, pp. 604–632.

Kok, S. and Domingos, P. (2009). Learning Markov Logic Network Structure via Hypergraph Lifting. In: *Proceedings of the 26th International Conference on Machine Learning (ICML–2009)*, pp. 505–512.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. MIT Press.

Kondor, R. and Jebara, T. (2003). A Kernel Between Sets of Vectors. In: *Proceedings of the 20th International Conference on Machine Learning (ICML–2003)*, pp. 361–368.

Kondor, R.I. and Lafferty, J.D. (2002). Diffusion Kernels on Graphs and Other Discrete Input Spaces. In: *Proceedings of the 19th International Conference on Machine Learning (ICML–2002)*, pp. 315–322.

Koppula, H.S., Anand, A., Joachims, T. and Saxena, A. (2011). Semantic Labeling of 3D Point Clouds for Indoor Scenes. In: *Advances in Neural Information Processing Systems 24 (NIPS–2011)*, pp. 244–252.

Krengel, U. (2005). *Einführung in die Wahrscheinlichkeitstheorie und Statistik*. Vieweg und Sohn Verlag.

Kriege, N. and Mutzel, P. (2012). Subgraph matching kernels for attributed graphs. In: *Proceedings of the 29th International Conference on Machine Learning (ICML–2012)*. `arXiv:1206.6483v1 [cs.LG]`.

Kriege, N., Neumann, M., Kersting, K. and Mutzel, P. (2014). Explicit versus Implicit Graph Feature Maps: A Computational Phase Transition for Walk Kernels. In: *14th IEEE International Conference on Data Mining (ICDM–2014)*. To appear.

Lafferty, J.D. and Lebanon, G. (2002). Information Diffusion Kernels. In: *Advances in Neural Information Processing Systems 15 (NIPS–2002)*, pp. 375–382.

Langville, A.N. and Meyer, C.D. (2006). *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press.

Lao, N. and Cohen, W.W. (2010). Relational Retrieval using a Combination of Path-Constrained Random Walks. *Machine Learning*, vol. 81, no. 1, pp. 53–67.

Lenz, I., Lee, H. and Saxena, A. (2013). Deep Learning for Detecting Robotic Grasps. In: *Proceedings of Robotics: Science and Systems IX (RSS–2013)*.

Leskovec, J. and Faloutsos, C. (2007). Scalable Modeling of Real Graphs using Kronecker Multiplication. In: *Proceedings of the 24th International Conference on Machine Learning (ICML–2007)*, pp. 497–504.

Lever, G., Diethe, T. and Shawe-Taylor, J. (2012). Data Dependent Kernels in Nearly-linear Time. In: *15th International Conference on Artificial Intelligence and Statistics (AISTATS–2012)*, JMLR Workshop and Conference Proceedings, pp. 685–693.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A. and Thrun, S. (2008). Anytime search in dynamic graphs. *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643.

Lin, F. and Cohen, W.W. (2010*a*). Power Iteration Clustering. In: *Proceedings of the 27th International Conference on Machine Learning (icml–2010)*, pp. 655–662.

Lin, F. and Cohen, W.W. (2010*b*). Semi-Supervised Classification of Network Data Using Very Few Labels. In: *International Conference on Advances in Social Networks Analysis and Mining (asonam–2010)*, pp. 192–199.

Liu, Y. (2006). Graph-based learning models for information retrieval: A survey. Tech. Rep., Michigan State University, East Lansing, US.

Liu, Y. and Kirchhoff, K. (2012). A Comparison of Graph Construction and Learning Algorithms for Graph-Based Phonetic Classification. Tech. Rep., University of Washington, Seattle, US.

Liu, Z.-Y., Wu, H.-F. and Huang, J.-F. (2010). Application of Neural Networks to Discriminate Fungal Infection Levels in Rice Panicles Using Hyperspectral Reflectance and Principal Components Analysis. *Computers and Electronics in Agriculture*, vol. 72, no. 2, pp. 99–106.

Lovász, L. (1996). Random Walks on Graphs: A Survey. In: Miklós, D., Sós, V.T. and Szőnyi, T. (eds.), *Combinatorics, Paul Erdős is Eighty*, vol. 2, pp. 353–398. János Bolyai Mathematical Society.

Low, Y., Gonzalez, J.E., Kyrola, A., Bickson, D., Guestrin, C. and Hellerstein, J.M. (2014). GraphLab: A New Framework For Parallel Machine Learning. `arXiv:1006.4990v1 [cs.LG]`.

Lü, L. and Zhou, T. (2011). Link Prediction in Complex Networks: A Survey. *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170.

MacKay, D.J.C. (1992). Bayesian Interpolation. *Neural Computation*, vol. 4, no. 3, pp. 415–447.

Madry, M., Ek, C.H., Detry, R., Hang, K. and Kragic, D. (2012). Improving Generalization for 3D Object Categorization with Global Structure Histograms. In: *2012 ieee/rsj International Conference on Intelligent Robots and Systems (iros–2012)*, pp. 1379–1386.

Mahboubi, A., Brun, L. and Dupé, F. (2010). Object Classification Based on Graph Kernels. In: *Proceedings of the 2010 International Conference on High Performance Computing and Simulation (hpcs–2010)*, pp. 385–389.

Mahe, P., Ueda, N., Akutsu, T., Perret, J.-L. and Vert, J.-P. (2004). Extensions of Marginalized Graph Kernels. In: *Proceedings of the 21st International Conference on Machine Learning (ICML–2004)*, pp. 552–559.

Mahé, P. and Vert, J.-P. (2009). Graph kernels based on tree patterns for molecules. *Machine Learning*, vol. 75, no. 1, pp. 3–35.

Maier, M., von Luxburg, U. and Hein, M. (2008). Influence of graph construction on graph-based clustering measures. In: *Advances in Neural Information Processing Systems 21 (NIPS–2008)*, pp. 1025–1032.

Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N. and Czajkowski, G. (2010). Pregel: A System for Large-scale Graph Processing. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (MOD–2010)*, pp. 135–146.

Manevitz, L.M. and Yousef, M. (2001). One-Class SVMs for Document Classification. *Journal of Machine Learning Research*, vol. 2, pp. 139–154.

Manning, C.D., Raghavan, P. and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

McPherson, M., Smith-Lovin, L. and Cook, J.M. (2001). Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444.

Melacci, S. and Belkin, M. (2011). Laplacian Support Vector Machines Trained in the Primal. *Journal of Machine Learning Research*, vol. 12, pp. 1149–1184.

Menon, A.K. and Elkan, C. (2011). Link Prediction via Matrix Factorization. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD–2011)*, pp. 437–452.

Mihalkova, L. and Mooney, R. (2007). Bottom-up learning of Markov logic network structure. In: *Proceedings of the 24th International Conference on Machine Learning (ICML–2007)*, pp. 625–632.

Min, M.R., Bonner, A.J. and Zhang, Z. (2007). Modifying Kernels Using Label Information Improves SVM Classification Performance. In: *6th International Conference on Machine Learning and Applications (ICMLA–2007)*, pp. 13–18.

Mitchell, T.M. (1997). *Machine Learning*. 1st edn. McGraw-Hill, Inc.

Montesano, L. and Lopes, M. (2012). Active Learning of Visual Descriptors for Grasping using Non-parametric Smoothed Beta Distributions. *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 452–462.

Moreno, P.J., Ho, P. and Vasconcelos, N. (2003). A Kullback–Leibler Divergence Based Kernel for SVM Classification in Multimedia Applications. In: *Advances in Neural Information Processing Systems 16 (NIPS–2003)*, pp. 1385–1392.

Muggleton, S. (1996). Learning from Positive Data. In: *Proceedings of the Inductive Logic Programming Workshop (ILP–1996)*, pp. 358–376.

Nadaraya, E.A. (1964). On Estimating Regression. *Theory of Probability and its Applications*, vol. 9, no. 1, pp. 141–142.

Neumann, M., Garnett, R., Bauckhage, C. and Kersting, K. (2014*a*). Propagation Kernels: Efficient Graph Kernels from Propagated Information. *Machine Learning*. Under review.

Neumann, M., Garnett, R. and Kersting, K. (2013*a*). Coinciding Walk Kernels. In: *11th Workshop on Mining and Learning with Graphs (MLG–2013)*.

Neumann, M., Garnett, R. and Kersting, K. (2013*b*). Coinciding Walk Kernels: Parallel Absorbing Random Walks for Learning with Graphs and Few Labels. In: *Asian Conference on Machine Learning (ACML–2013)*, pp. 357–372.

Neumann, M., Garnett, R., Moreno, P., Patricia, N. and Kersting, K. (2012*a*). Propagation Kernels for Partially Labeled Graphs. In: *10th Workshop on Mining and Learning with Graphs (MLG–2012)*.

Neumann, M., Hallau, L., Klatt, B., Kersting, K. and Bauckhage, C. (2014*b*). Erosion Band Features for Cell Phone Image Based Plant Disease Classification. In: *Proceedings of the 22nd International Conference on Pattern Recognition (ICPR–2014)*, pp. 3315–3320.

Neumann, M., Kersting, K. and Ahmadi, B. (2011). Markov Logic Sets: Towards Lifted Information Retrieval using PageRank and Label Propagation. In: *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI–2011)*, pp. 447–452.

Neumann, M., Kersting, K., Xu, Z. and Schulz, D. (2009). Stacked Gaussian Process Learning. In: *9th IEEE International Conference on Data Mining (ICDM–2009)*, pp. 387–396.

Neumann, M., Moreno, P., Antanas, L., Garnett, R. and Kersting, K. (2013*c*). Graph Kernels for Object Category Prediction in Task–Dependent Robot Grasping. In: *11th Workshop on Mining and Learning with Graphs (MLG–2013)*.

Neumann, M., Patricia, N., Garnett, R. and Kersting, K. (2012*b*). Efficient Graph Kernels by Randomization. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD–2012)*, pp. 378–393.

Neville, J. and Jensen, D. (2005). Leveraging Relational Autocorrelation with Latent Group Models. In: *5th IEEE International Conference on Data Mining (ICDM–2005)*, pp. 322–329.

Ojala, T., Pietikäinen, M. and Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987.

Park, H.-J. and Friston, K. (2013). Structural and Functional Brain Networks: From Connections to Cognition. *Science*, vol. 342, no. 6158, pp. 1238411+.

Poggio, T. and Girosi, F. (1990). Networks for Approximation and Learning. *Proceedings of the IEEE*, vol. 78, no. 9.

Pons, P. and Latapy, M. (2006). Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218.

Ramon, J. and Gärtner, T. (2003). Expressivity versus Efficiency of Graph Kernels. In: *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74.

Rascher, U., Nichol, C.J., Small, C. and Hendricks, L. (2007). Monitoring spatio-temporal dynamics of photosynthesis with a portable hyperspectral imaging system. *Photogrammetric Engineering & Remote Sensing*, vol. 73, no. 1, pp. 45–56.

Rasmussen, C. and Williams, C. (2005). *Gaussian Processes for Machine Learning*. Adaptive Computation And Machine Learning. MIT Press.

Raymond, R. and Kashima, H. (2010). Fast and Scalable Algorithms for Semi-supervised Link Prediction on Static and Dynamic Graphs. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD–2010)*, pp. 131–147.

Read, J., Pfahringer, B., Holmes, G. and Frank, E. (2011). Classifier Chains for Multi-label Classification. *Machine Learning*, vol. 85, no. 3, pp. 333–359.

Richardson, M. and Domingos, P. (2006). Markov Logic Networks. *Machine Learning*, vol. 62, pp. 107–136.

Rossi, R.A. and Gleich, D.F. (2012). Dynamic PageRank Using Evolving Teleportation. In: *9th International Workshop on Algorithms and Models for the Web Graph (WAW–2012)*, pp. 126–137.

Rossi, V., Battilani, P., Chiusa, G., Giosuè, S., Languasco, L. and Racca, P. (2000). Components of rate-reducing resistance to cercospora leaf spot in sugar beet: conidiation length, spore yield. *Journal of Plant Pathology*, pp. 125–131.

Rumpf, T., Mahlein, A.-K., Steiner, U., Oerke, E.-C., Dehne, H.-W. and Plümer, L. (2010). Early detection and classification of plant diseases with Support Vector Machines based on hyperspectral reflectance. *Computers and Electronics in Agriculture*, vol. 74, no. 1, pp. 91–99.

Russell, S.J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. 2nd edn. Pearson Education.

Savarese, S. and Li, F.-F. (2007). 3D Generic Object Categorization, Localization and Pose Estimation. In: *11th ieee International Conference on Computer Vision (iccv–2007)*.

Saxena, A., Wong, L.L.S. and Ng, A.Y. (2008). Learning Grasp Strategies with Partial Shape Information. In: *Proceedings of the 23rd aaai Conference on Artificial Intelligence (aaai–2008)*, pp. 1491–1494.

Schauerte, B. and Stiefelhagen, R. (2013). "Wow!" Bayesian surprise for salient acoustic event detection. In: *ieee International Conference on Acoustics, Speech and Signal Processing (icassp–2013)*, pp. 6402–6406.

Schiegg, M., Neumann, M. and Kersting, K. (2012). Markov Logic Mixtures of Gaussian Processes: Towards Machines Reading Regression Data. In: *15th International Conference on Artificial Intelligence and Statistics (aistats–2012)*, jmlr Workshop and Conference Proceedings, pp. 1002–1011.

Schölkopf, B., Herbrich, R. and Smola, A.J. (2001*a*). A Generalized Representer Theorem. In: *14th Annual Conference on Computational Learning Theory and 5th European Conference on Computational Learning Theory (colt/eurocolt–2001)*, pp. 416–426.

Schölkopf, B., Platt, J.C., Shawe-Taylor, J., Smola, A.J. and Williamson, R.C. (2001*b*). Estimating the Support of a High-Dimensional Distribution. *Neural Computation*, vol. 13, no. 7, pp. 1443–1471.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G. and Schomburg, D. (2004). brenda, the enzyme database: Updates and major new developments. *Nucleic Acids Research*, vol. 32, no. Database-Issue, pp. 431–433.

Schur, J. (1911). Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen. *Journal für die reine und angewandte Mathematik*, vol. 140, pp. 1–28.

Semenovich, D. and Sowmya, A. (2010). Geometry Aware Local Kernels for Object Recognition. In: *10th Asian Conference on Computer Vision (ACCV–2010)*, pp. 490–503.

Shental, O., Siegel, P.H., Wolf, J.K., Bickson, D. and Dolev, D. (2008). Gaussian Belief Propagation Solver for Systems of Linear Equations. In: *IEEE International Symposium on Information Theory (ISIT–2008)*, pp. 1863–1867.

Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K. and Borgwardt, K.M. (2011). Weisfeiler–Lehman Graph Kernels. *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561.

Shervashidze, N., Vishwanathan, S.V.N., Petri, T., Mehlhorn, K. and Borgwardt, K.M. (2009). Efficient graphlet kernels for large graph comparison. In: *12th International Conference on Artificial Intelligence and Statistics (AISTATS–2009)*, JMLR Workshop and Conference Proceedings, pp. 488–495.

Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A.J. and Vishwanathan, S.V.N. (2009). Hash Kernels for Structured Data. *Journal of Machine Learning Research*, vol. 10, pp. 2615–2637.

Shi, X., Li, Y. and Yu, P.S. (2011). Collective Prediction with Latent Graphs. In: *Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM–2011)*, pp. 1127–1136.

Shin, K. and Kuboyama, T. (2008). A Generalization of Haussler's Convolution Kernel – Mapping Kernel. In: *Proceedings of the 25th International Conference on Machine Learning (ICML–2008)*, pp. 944–951.

Silva, R., Heller, K., Ghahramani, Z. and Airoldi, E. (2010). Ranking Relations using Analogies in Biological and Information Networks. *Annals of Applied Statistics*, vol. 4, no. 2, pp. 615–644.

Sindhwani, V., Niyogi, P. and Belkin, M. (2005). Beyond the Point Cloud: from Transductive to Semi-supervised Learning. In: *Proceedings of the 22nd International Conference on Machine Learning (ICML–2005)*, pp. 824–831.

Singla, P. and Domingos, P. (2008). Lifted First-Order Belief Propagation. In: *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI–2008)*, pp. 1094–1099.

Smola, A. and Kondor, R.I. (2003). Kernels and Regularization on Graphs. In: *Proceedings of the 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop (COLT/KERNEL–2003)*, pp. 144–158.

Strogatz, S.H. (2001). Exploring complex networks. *Nature*, vol. 410, no. 6825, pp. 268–276.

Sun, J., Xie, Y., Zhang, H. and Faloutsos, C. (2008). Less is More: Sparse Graph Mining with Compact Matrix Decomposition. *Statistical Analysis and Data Mining*, vol. 1, no. 1, pp. 6–22.

Sutton, C. and McCallum, A. (2006). An Introduction to Conditional Random Fields for Relational Learning. In: *Introduction to Statistical Relational Learning*, Adaptive Computation and Machine Learning, pp. 93–128. MIT press.

Szummer, M. and Jaakkola, T. (2001). Partially Labeled Classification with Markov Random Walks. In: *Advances in Neural Information Processing Systems 14 (nips–2001)*, pp. 945–952.

Tipping, M.E. (2001). Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, vol. 1, pp. 211–244.

Tishby, N. and Slonim, N. (2000). Data Clustering by Markovian Relaxation and the Information Bottleneck Method. In: *Advances in Neural Information Processing Systems 13 (nips–2000)*, pp. 640–646.

Tong, H., Faloutsos, C. and Pan, J.-Y. (2006). Fast Random Walk with Restart and Its Applications. In: *6th ieee International Conference on Data Mining (icdm-2006)*, pp. 613–622.

Tou, J.Y., Tay, Y.H. and Lau, P.Y. (2009). Gabor Filters as Feature Images for Covariance Matrix on Texture Classification Problem. In: Köppen, M., Kasabov, N. and Coghill, G. (eds.), *Advances in Neuro-Information Processing*, vol. 5507, pp. 745–751. Springer-Verlag Berlin Heidelberg.

Tsivtsivadze, E., Urban, J., Geuvers, H. and Heskes, T. (2011). Semantic Graph Kernels for Automated Reasoning. In: *Proceedings of the 11th siam International Conference on Data Mining (sdm–2011)*, pp. 795–803.

Tsuda, K., Kin, T. and Asai, K. (2002). Marginalized Kernels for Biological Sequences. In: *Proceedings of the 10th International Conference on Intelligent Systems for Molecular Biology (ismb–2002)*, pp. 268–275.

Umeyama, S. (1988). An Eigendecomposition Approach to Weighted Graph Matching Problems. *ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 5, pp. 695–703.

Valkealahti, K. and Oja, E. (1998). Reduced multidimensional co-occurrence histograms in texture classification. *ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 90–94.

VAPNIK, V. (1998). *Statistical Learning Theory*, chap. 8: Estimating the Values of Function at Given Points, pp. 339–371. John Wiley & Sons, Inc.

VISHWANATHAN, S.V.N., BORGWARDT, K.M. and SCHRAUDOLPH, N.N. (2006). Fast Computation of Graph Kernels. In: *Advances in Neural Information Processing Systems 19 (NIPS–2006)*, pp. 1449–1456.

VISHWANATHAN, S.V.N., KASKI, S., NEVILLE, J. and WROBEL, S. (2011). Introduction to the special issue on mining and learning with graphs. *Machine Learning*, vol. 82, no. 2, pp. 91–93.

VISHWANATHAN, S.V.N., SCHRAUDOLPH, N.N., KONDOR, R.I. and BORGWARDT, K.M. (2010). Graph kernels. *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242.

VON LUXBURG, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, vol. 17, no. 4, pp. 395–416.

WAHBA, G. (1990). *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics.

WALE, N. and KARYPIS, G. (2006). Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. In: *6th IEEE International Conference on Data Mining (ICDM–2006)*, pp. 678–689.

WALE, N., WATSON, I.A. and KARYPIS, G. (2008). Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375.

WASHIO, T. and MOTODA, H. (2003). State of the Art of Graph-based Data Mining. *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, pp. 59–68.

WASSERMAN, L. (2010). *All of Statistics: A Concise Course in Statistical Inference*. Springer Science + Business Media New York.

WATKINS, C. (1999). Dynamic Alignment Kernels. In: *Advances in Large Margin Classifiers*, pp. 39–50.

WATSON, G.S. (1964). Smooth Regression Analysis. *Sankhyā: The Indian Journal of Statistics*, vol. 26, no. 4, pp. 359–372.

WINN, J.M., CRIMINISI, A. and MINKA, T.P. (2005). Object categorization by learned universal visual dictionary. In: *10th IEEE International Conference on Computer Vision (ICCV–2005)*, pp. 1800–1807.

Wrobel, S. (2001). Inductive Logic Programming for Knowledge Discovery in Databases. In: Džeroski, S. and Lavrač, N. (eds.), *Relational Data Mining*, pp. 74–101. Springer-Verlag Berlin Heidelberg.

Wu, X.-M., Li, Z., So, A.M.-C., Wright, J. and Chang, S.-F. (2012). Learning with Partially Absorbing Random Walks. In: *Advances in Neural Information Processing Systems 25 (nips–2012)*, pp. 3086–3094.

Xiang, R. and Neville, J. (2008). Pseudolikelihood EM for Within-Network Relational Learning. In: *8th ieee International Conference on Data Mining (icdm–2008)*, pp. 1103–1108.

Yang, Z., Hao, T., Dikmen, O., Chen, X. and Oja, E. (2012). Clustering by Nonnegative Matrix Factorization Using Graph Random Walk. In: *Advances in Neural Information Processing Systems 25 (nips–2012)*, pp. 1088–1096.

Yedidia, J.S., Freeman, W.T. and Weiss, Y. (2003). Understanding Belief Propagation and Its Generalizations. In: Lakemeyer, G. and Nebel, B. (eds.), *Exploring Artificial Intelligence in the New Millennium*, pp. 239–269. Morgan Kaufmann Publishers.

Zhang, D., Zhou, Z.-H. and Chen, S. (2006). Non-negative matrix factorization on kernels. In: *9th Pacific Rim International Conference on Artificial Intelligence (pricai–2006)*, pp. 404–412.

Zhang, L., Song, M., Liu, X., Bu, J. and Chen, C. (2013). Fast Multi-View Segment Graph Kernel for Object Classification. *Signal Processing*, vol. 93, no. 6, pp. 1597–1607.

Zhou, D., Bousquet, O., Lal, T.N., Weston, J. and Schölkopf, B. (2003). Learning with Local and Global Consistency. In: *Advances in Annual Conference on Neural Information Processing Systems 16 (nips–2003)*, pp. 321–328.

Zhou, D., Huang, J. and Schölkopf, B. (2006). Learning with Hypergraphs: Clustering, Classification, and Embedding. In: *Advances in Neural Information Processing Systems 19 (nips–2006)*, pp. 1601–1608.

Zhu, X. (2005). *Semi-supervised Learning with Graphs*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, US. `0-542-19059-1`, `AAI3179046`.

Zhu, X., Ghahramani, Z. and Lafferty, J.D. (2003). Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In: *Proceedings of the 20th International Conference on Machine Learning (icml–2003)*, pp. 912–919.

Zhu, X., Kandola, J.S., Ghahramani, Z. and Lafferty, J.D. (2004). Nonparametric Transforms of Graph Kernels for Semi-Supervised Learning. In: *Advances in Neural Information Processing Systems 17 (nips–2004)*, pp. 1641–1648.