Learning with Graphs: Kernel and Neural Approaches

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Christopher Morris

Dortmund
2019

Tag der mündlichen Prüfung:
20.12.2019

Dekan:
Prof. Dr. Ing. Gernot A. Fink

Gutachter:
Prof. Dr. Petra Mutzel, TU Dortmund, Fakultät für Informatik
Prof. Dr. Kristian Kersting, TU Darmstadt, Fakultät für Informatik

Abstract

Linked data arise in many real-world settings—from chemical molecules, and protein-protein interaction networks, to social networks. Hence, in recent years, machine learning methods for graphs have become an important area of research. The present work deals with (supervised) graph classification, i.e., given a set of (class) labeled graphs we want to train a model such that it can predict the labels of unlabeled graphs. Thereto, we want to map a graph to a (usually) high-dimensional vector such that standard machine learning techniques can be applied. In the first part of this thesis, we present kernel methods for graphs. Specifically, we present a framework for scalable kernels that can handle continuous vertex and edge labels, which often arise with real-world graphs, e.g., chemical and physical properties of atoms. Moreover, we present a graph kernel which can take global or higher-order graph properties into account, usually not captured by other graph kernels. To this end, we propose a local version of the k-dimensional Weisfeiler-Leman algorithm, which is a well-known heuristic for the graph isomorphism problem. We show that a variant of our local algorithm has at least the same power as the original algorithm, while at the same time taking the sparsity of the underlying graph into account and preventing overfitting. To make this kernel scalable, we present an approximation algorithm with theoretical guarantees. Subsequently, we introduce a theoretical framework for the analysis of graph kernels showing that most kernels are not capable of distinguishing simple graph-theoretic properties and propose a theoretically well-founded graph kernel, which can distinguish these properties. The second part of this thesis deals with neural approaches to graph classification and their connection to kernel methods. We show that the expressivity of so-called graph neural networks can be upper-bounded by the 1-dimensional Weisfeiler-Leman graph isomorphism heuristic. We then leverage this insight to propose a generalization which is provable more powerful than graph neural networks regarding distinguishing non-isomorphic graphs.

Acknowledgement

I want to thank my advisor Petra Mutzel for giving me the opportunity to work in the Collaborative Research Center 876 and giving me all the freedom I needed to pursue my research ideas. I am thankful to Kristian Kersting for agreeing to act as my second reader as well as taking the time to discuss many ideas with me during the first part of my Ph.D. studies. I want to thank Nils M. Kriege for the interesting discussions about graph classification as well as helping me write my first paper. Furthermore, I want to thank Jure Leskovec for letting me stay at his group at Stanford University. Moreover, I want to thank my other co-authors Fritz Bökler, Martin Grohe, Matthias Ehrgott, Matthias Fey, William L. Hamilton, Fredrik D. Johansson, Jan E. Lenssen, Marion Neumann, Gaurav Rattan, Xiang Ren, Anja Rey, Martin Ritzert, Christian Sohler, Zhitao (Rex) Ying, and Jiaxuan You for sharing their knowledge. I want to thank Gundel Jankord for helping me with the paperwork. Finally, I want to thank the German Research Foundation (DFG) for funding my studies through the *Collaborative Research Center SFB 876 – Providing Information by Resource-Constrained Data Analysis*, and making various conference trips around the world and a research stay possible.

Contents

Chapter 1.

Introduction

Linked data arise in various domains such as chem- and bioinformatics, social network analysis, and pattern recognition. Graphs can naturally represent such data. Therefore, machine learning on graphs has become an active research area of increasing importance. This thesis deals with developing methods for (supervised) graph classification.

In the first part of this work, we focus on kernel methods for graph data, so-called *graph kernels*. Intuitively, graph kernels measure the similarity of a pair of graphs. More concretely, a graph kernel is a positive semi-definite function that maps a pair of graphs to a real number. Hence, it can be used together with established learning algorithms such as support vector machines. Since most real-world graphs have continuous features attached to vertices and edges, e.g., real-valued vectors modeling chemical or physical properties of molecules in chemistry, developing graph kernels for such data is crucial. In the past, graph kernels that were able to deal with such input did not scale to large datasets as they computed the kernel function for each pair of graphs, leading to a quadratic overhead in running time. Hence, in Section 3.2 of Chapter 3, we present a method to scale graph kernels to large datasets with continuous vertex and edge information. We circumvent the problem of quadratic overhead by mapping each graph to a finite-dimensional real vector. We show that the inner product between such vectors approximates well-known graph kernel functions that can handle continuous inputs. The basis of this work is the development of randomized hash functions, where the collision probability is equal to the value of a kernel function for comparing real-valued vectors. We evaluate our proposed method on numerous graph classification benchmark datasets showing state-of-the-art performance while being much faster.

The next section deals with extensions of kernels based on the 1-dimensional Weisfeiler-Leman graph isomorphism heuristic (1-WL). We derive a kernel based on the k-dimensional Weisfeiler-Leman algorithm [36, pp. 84 sqq.] which

is more powerful than the 1-WL in terms of distinguishing non-isomorphic graphs. To make the algorithm suitable for machine learning, we derive a local variant that prevents overfitting and takes the sparsity of the underlying graph into account. Moreover, we are able to show that a variant of the local variant has at least the same power as the original algorithm. In order to scale the kernel to large datasets, we show how to approximate it. For bounded-degree graphs, we show that it can be approximated in constant time. Finally, we empirically evaluate our proposed method showing that it often outperforms state-of-the-art kernel methods on a wide range of graph classification benchmark datasets.

The last section dealing with kernels investigates them from a theoretical perspective. In the past two decades, a large number of graph kernels have been proposed. Hence, it becomes increasingly difficult to perform a fair experimental comparison of kernels, and to assess their advantages and disadvantages for specific datasets. Indeed, current experimental comparisons cannot give a complete picture, and are of limited help to a practitioner who has to choose a kernel for a particular application. Moreover, graph kernels are developed with the (possibly conflicting) goals of being efficiently computable, and capturing the structural information of the input graphs adequately. There is no theoretical justification on why specific kernels perform better than others, but merely experimental evaluations. We address this by introducing a theoretical framework for the analysis of the expressivity of graph kernels motivated by concepts from property testing [32, 31]. We say that a graph kernel distinguishes a property if it guarantees a constant angle (independent of the graph size) between the feature vectors of any two graphs, one of which has the property, and the other is far away from doing so. We study well-known graph kernels, and their ability to distinguish fundamental properties such as connectivity showing that current graph kernels are not able to distinguish these basic properties. Subsequently, we propose a more powerful graph kernel.

The subsequent chapter deals with neural approaches for graph classification. In the past years, deep learning architectures revolutionized the fields of computer vision and natural language processing. Hence, the question arises if such methods can be applied to graph data. Graph neural networks (GNNs) have emerged as a machine learning framework addressing the above challenge. Up to now, the evaluation and analysis of GNNs have been mostly empirical, showing promising results compared to kernel approaches, see, e.g., [125]. However, it remains unclear how GNNs are encoding graph structure information into their vector representations, and whether there are theoretical advantages of GNNs compared to kernel-based approaches. We

offer a theoretical exploration of the relationship between GNNs and kernels that are based on the 1-WL. We show that GNNs cannot be more powerful than the 1-WL regarding distinguishing non-isomorphic (sub-)graphs, e.g., the properties of subgraphs around each vertex. This result holds for a broad class of GNN architectures and all possible choices of parameters for them. On the positive side, we show that given the right parameter initialization GNNs have the same expressiveness as the 1-WL, completing the equivalence.

Going further, we leverage these theoretical relationships to propose a generalization of GNNs, called k-GNNs, which are neural architectures inspired by the k-dimensional Weisfeiler-Leman algorithm. The critical insight in these higher-dimensional variants is that they perform message passing directly between subgraph structures, rather than individual vertices. This higher-order form of message passing can capture coarse-grained structures that are not identifiable at the vertex-level. Our experimental results demonstrate that these k-GNNs can consistently outperform traditional GNNs on a variety of graph classification and regression tasks.

1.1. Relevant publications

The following publications are relevant for the present work. Publications where the author is the first author are colored in maroon. The first two (three) authors of publication 3 (publication 5) share first authorship.

1. C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. "Faster Kernel for Graphs with Continuous Attributes via Hashing." In: *IEEE International Conference on Data Mining*. IEEE, 2016, pp. 1095–1100

2. C. Morris, K. Kersting, and P. Mutzel. "Glocalized Weisfeiler-Lehman Kernels: Global-Local Feature Maps of Graphs." In: *IEEE International Conference on Data Mining*. IEEE, 2017, pp. 327–336

3. N. M. Kriege and C. Morris. "Recent Advances in Kernel-Based Graph Classification." In: *The European Conference on Machine Learning & Principles and Practice of Knowledge Discovery In Databases*. Springer, 2017, pp. 388–392

4. N. M. Kriege, C. Morris, A. Rey, and C. Sohler. "A Property Testing Framework for the Theoretical Expressivity of Graph Kernels." In: *International Joint Conference on Artificial Intelligence*. IJCAI, 2018, pp. 2348–2354

5. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, Jan Eric Lenssen, G. Rattan, and M. Grohe. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks." In: *AAAI Conference on Artificial Intelligence*. AAAI. 2019, pp. 4602–4609

6. C. Morris and P. Mutzel. "Towards a practical k-dimensional Weisfeiler-Leman algorithm." In: *CoRR* abs/1904.01543 (2019)

Proofs to which I have not contributed are marked by the literature reference corresponding to the publication. A full list of publications can be found in Appendix A.

1.2. Structure

In the first chapter, we fix notation, and introduce various mathematical concepts that are used throughout this thesis. Moreover, we give an introduction to supervised machine learning, kernel methods, neural networks, and the Weisfeiler-Leman algorithm. The subsequent chapter deals with the work on graph kernels, namely the framework for scalable graphs for graphs with continuous information [82], the kernel based on the k-dimensional Weisfeiler-Leman algorithm [80, 81], and the theoretical framework for studying the expressiveness of well-known kernels [65]. In the fourth chapter, we study GNNs, their relation to kernel methods, and their k-dimensional generalization [83]. The last chapter acts as a conclusion and provides directions for future work.

Chapter 2.

Preliminaries

The following chapter introduces notation, defines various mathematical concepts, and gives an introduction to supervised machine learning, learning with kernels, and neural networks. Moreover, the last section describes the Weisfeiler-Leman graph isomorphism heuristic which is the basis for most algorithms described in this thesis.

2.1. Notation

We (usually) typeset sets in upper-case letters (S), matrices in upper-case bold letters ($\mathbf{M}$), vectors in lower-case bold or Greek letters ($\mathbf{v}$, ϕ), and scalars with Latin or Greek letters (a, α).

Let $\mathbf{v}$ be a vector in $\mathbb{R}^n$ then $\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} |v_i|^2}$, and $\|\mathbf{v}\|_p = \sqrt[p]{\sum_{i=1}^{n} |v_i|^p}$ for $p > 0$. Moreover, let $[m] = \{1, \ldots, m\} \subset \mathbb{N}$ for $m > 1$, $[m]_0 = \{0, \ldots, m\} \subset \mathbb{N}_0$, and let $\{\!\{\ldots\}\!\}$ denote a multiset. Finally, let $\mathbf{v}$ and $\mathbf{w}$ be vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively, then $[\mathbf{v}, \mathbf{w}]$ in $\mathbb{R}^{n+m}$ denotes the column-wise vector concatenation of $\mathbf{v}$ and $\mathbf{w}$.

2.2. Graph theory

A *graph* G is a pair (V, E) with a *finite* set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of vertices and the set of edges of G by $V(G)$ and $E(G)$, respectively. For ease of notation we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). In the case of *directed graphs* $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$. A *(vertex) labeled graph* is a graph G endowed with a *label function* $l\colon V(G) \to \Sigma$, where Σ is some finite alphabet. *Edge labeled graphs* are defined analogously. We say that $l(v)$ is a *label* of v for v in $V(G)$. If we replace Σ with $\mathbb{R}^n$, we say $l(v)$ is a *continuous (vertex) label*, and G is a *continuously labeled graph*. A *vertex coloring* is a function $V(G) \to \mathbb{S}$

with arbitrary codomain $\mathbb{S}$. We say that a vertex coloring c *refines* a vertex coloring d, written $c \sqsubseteq d$, if $c(v) = c(w)$ implies $d(v) = d(w)$ for every v and w in $V(G)$. If both directions hold, we write $c \equiv d$. A *color class* $Q \subseteq V(G)$ of a vertex coloring c is a maximal set of vertices with $c(v) = c(w)$ for every v and w in Q. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced* by S with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$.

We denote the set of all graphs by $\mathcal{G}$ and the set of all graphs on n vertices by $\mathcal{G}_n$. Moreover, $N(v)$ or $\delta(v)$ denotes the *neighborhood* of v in $V(G)$, i.e., $N(v) = \delta(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. Moreover, its complement $\overline{\delta}(v) = \{u \in V(G) \mid (v, u) \notin E(G)\}$. A *tree* is a connected graph without cycles. A *directed tree* is a directed acyclic graph whose underlying undirected graph is a tree. Let p be a vertex in a directed tree then we call the neighbors of s *children* with parent s. If s is not a child of another vertex then it is the *root* of the tree. A graph is of *d-bounded degree* if its maximum degree is at most d, where d is always independent of the number of vertices, i.e., d is in $\mathcal{O}(1)$. We say that two graphs G and H are *isomorphic* if there exists an edge preserving bijection $\varphi\colon V(G) \to V(H)$, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. In the case of labeled graphs, we require that $l(v) = l((\varphi(v))$ and $l((u, v))) = l((\varphi(u), \varphi(v)))$ holds. If G and H are isomorphic, we write $G \simeq H$ and call φ an *isomorphism* between G and H. In the case that G and H are directed, isomorphic trees rooted at v in $V(G)$ and w in $V(H)$, respectively, we write $G \simeq_{v \to w} H$. Moreover, we call the equivalence classes induced by $\simeq$ *isomorphism types*, and denote the isomorphism type of G by τ_G. Let S be a set then the set of *k-sets* $[S]^k = \{U \subseteq S \mid |U| = k\}$ for $k \geq 2$, which is the set of all subsets of S with cardinality k.

2.3. Supervised machine learning

In the following, we give an introduction to supervised machine learning. We loosely follow the exposition given in [79]. Let $\mathcal{X}$ be the set of all possible *examples*, e.g., a set of graphs, and let $\mathcal{Y}$ be the set of all possible *target values*, e.g., $\{0, 1\}$ in the case of *binary classification*, or the reals in the case of *regression*. We assume that the elements of $\mathcal{X}$ are independently and identically distributed to some fixed but unknown distribution $\mathcal{D}$. Moreover, we assume that there exists a *target concept* $c\colon \mathcal{X} \to \mathcal{Y}$ which maps each example to its target value. Given a sample $S = (s_1, \ldots, s_m)$ sampled independently and identically from $\mathcal{D}$, as well as the corresponding target values $(y_1, \ldots, y_m) = (c(s_1), \ldots, c(s_m))$, the aim of supervised machine learning is

to select a *hypothesis* $h\colon \mathcal{X} \to \mathcal{Y}$ from the set of possible hypothesis H that minimizes the *generalization error*

$$R(h) = \Pr_{x\sim\mathcal{D}}(L(h(x), c(x))) = \mathbb{E}_{x\sim\mathcal{D}}[L(h(x), c(x))],$$

where $L\colon \mathcal{X}\times\mathcal{Y} \to \mathbb{R}_{\geq 0}$ is a *loss function*, which measures how well the chosen hypothesis fits the target concept. One possible choice of loss function is the *binary loss*, which is equal to the indicator function of the event $h(x) \neq c(x)$. Note that a learning algorithm cannot compute the generalization error since it depends on the unknown distribution $\mathcal{D}$ as well the target concept c. Hence, we approximate it by the *empirical error*

$$\widehat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{h(x)\neq c(x)}.$$

The basic insight here, which follows by the linearity of expectation, is that

$$\mathbb{E}[\widehat{R}(h)] = R(h).$$

Clearly, the aim of supervised machine learning is to minimize the generalization error using the empirical error as a proxy. Hence, *empirical risk minimization* aims to find the hypothesis $\widehat{h}$ in H such that

$$\widehat{h} = \arg\min_{h\in H} \widehat{R}(h).$$

To avoid that the learning algorithm finds a hypothesis that is tailored to closely to the given sample, so-called *overfitting*, we employ *structural risk minimization* that adds a *regularization penalty* $\Omega\colon H \to \mathbb{R}_{\geq 0}$ to the empirical error, resulting in the following optimization problem

$$\arg\min_{h\in H} \widehat{R}(h) + \underbrace{\lambda\Omega(h)}_{\text{Regularizer}},$$

for $\lambda \geq 0$.

2.4. Learning with kernels

Machine learning methods for data that is linearly separable is well understood. However, most data occurring in the real-world often does not have this property. The idea of kernel methods is to merge these two viewpoints by using a *kernel* that maps each pair of examples to a real number that

corresponds to an inner product between two vectors in a (usually high-dimensional) Hilbert space. The hope is that in this space the data can be separated linearly. Hence, if we can define a kernel for the given data, we do not have to construct the vectors explicitly. In the following, we formally introduce kernels.

Definition 2.4.1. Let $\mathcal{X}$ be a set, then $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *kernel* over $\mathcal{X}$.

The idea here is to choose the kernel k in such a way so that there exists some *feature map* $\phi\colon \mathcal{X} \to \mathbb{H}$, where $\mathbb{H}$ is some Hilbert space, such that

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

for x and y in $\mathcal{X}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of $\mathbb{H}$. The space $\mathbb{H}$ is often called a *feature space*. Let x in $\mathcal{X}$ then $\phi(x)$ in $\mathbb{H}$ is called *feature vector*. An important subset of kernels are *(symmetric) positive semi-definite kernels*.

Definition 2.4.2. Let $\mathbf{K}$ be matrix in $\mathbb{R}^{m \times m}$, then $\mathbf{K}$ is positive semi-definite if

1. the eigenvalues of $\mathbf{K}$ are non-negative, and
2. for all vectors $\mathbf{c} = (c_1, \ldots, c_m)^T$ in $\mathbb{R}^{m \times 1}$,

$$\mathbf{c}^T \mathbf{K} \mathbf{c} = \sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0.$$

Moreover, we need the definition of a *gram matrix* corresponding to a kernel.

Definition 2.4.3. Let $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel, and let $x_1, \ldots, x_m \subseteq \mathcal{X}$. Then the *gram matrix* $\mathbf{K} = [k(x_i, x_j)]_{ij}$.

We can now define a *positive semi-definite kernel*.

Definition 2.4.4. Let $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel, then k is positive semi-definite if for all $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ the corresponding gram matrix is symmetric and positive semi-definite.

For example the *Dirac kernel*

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

for x and y in $\mathcal{X}$ is positive semi-definite. In the following, we use the term kernel and positive semi-definite kernel interchangeable. The following result is the most important for kernels. It states that for each kernel there exists a unique Hilbert space and a mapping such that the inner product between two points under the mapping is equal to the kernel value of the two points.

Theorem 2.4.5. Let $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a (positive semi-definite) kernel, then there exists a Hilbert space $\mathbb{H}$ and a mapping $\phi\colon \mathcal{X} \to \mathbb{H}$ such that

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

for all x and y in $\mathcal{X}$. Moreover $\mathbb{H}$ has the *reproducing property*:

$$h(x) = \langle h, k(x, \cdot) \rangle.$$

The space $\mathbb{H}$ is called a *reproducing kernel Hilbert space*.

By using the above definition, it becomes apparent that we do not need to construct the feature space explicitly. This insight is often referred to as the *kernel trick* and plays an important role when using kernels together with *support vector machines* (SVMs), which we describe in the next section.

Remark 2.4.6. In certain cases, e.g., when the number of components of the (explicit) feature vectors is not too high (e.g., finite), explicit feature maps may provide computational benefits. In the following, we use the term *explicit kernel* for cases where we compute the feature vector for each data point directly. Consequently, we use the term *implicit kernel* when we employ the kernel trick.

2.4.1. Support vector machines

Let $\mathcal{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ be a subset of $\mathbb{R}^n$ for $n > 0$, and let $\mathcal{Y} = \{-1, 1\}$ be the set of target values. SVMs choose a hypothesis from the set of *linear classifiers*, or hyperplanes,

$$H = \{\mathbf{x} \mapsto \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b) \mid \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}\}.$$

Thus, SVMs try to find a hypothesis from H such that points labeled -1 fall on one side of the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ and the others, labeled 1, on the other side. Clearly, we can only find such hypothesis if our data is linearly separable. We first assume that this is the case. The key idea behind SVMs is not to choose any such separating hyperplane but to choose the hyperplane

with the maximum *margin*, which we call *maximum margin hyperplane*. The margin is defined by

$$\min_{\mathbf{x}\in\mathcal{X}} \frac{\mathbf{w}\cdot\mathbf{x}+b}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}.^{1}$$

For example, by following the derivation in [79, pp. 65 sq.], one can show that this can be achieved by solving the following quadratic optimization problem with affine constraints:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$
$$\text{such that } y_i(\mathbf{w}\cdot\mathbf{x_i}+b)\geq 1,$$

for all i in $[m]$. In practice this problem can be solved efficiently, notably by block coordinate descent algorithms [15]. Moreover, there exists a dual problem of the above, which has the following form [79, pp. 67 sq.]:

$$\max_{\alpha} \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \underbrace{\langle\phi(\mathbf{x_i}),\phi(\mathbf{x_j})\rangle}_{k(\mathbf{x_i},\mathbf{x_j})}$$
$$\text{such that } \sum_{i=1}^{m}\alpha_i y_i = 0 \text{ and } \alpha_i \geq 0,$$

for i in $[m]$. The critical insight here is that we can replace the inner product in the nested sum of the objective function with the (precomputed) kernel function.

Non-separable case

In real-world settings, the data is often not linearly separable, hence the constraints of the optimization problems, discussed above, cannot hold for all points simultaneously. To solve this, we introduce the slack variable ξ_i for i in $[m]$ such that

$$y_i(\mathbf{w}\cdot\mathbf{x_i}+b)\geq 1-\xi_i.$$

Intuitively, the slack variable ξ_i measure how much the variable $\mathbf{x_i}$ violates the constraint $y_i(\mathbf{w}\cdot\mathbf{w}_i+b)\geq 1$. We now add the penalty term

$$\sum_{i=1}^{m}\xi_i^p,$$

[1] W.l.o.g., we consider the *canonical hyperplane*, i.e., $\min_{\mathbf{x}\in\mathcal{X}}|\mathbf{w}\cdot\mathbf{x}+b|=1$.

for $p > 0$, to the above objective of the primal problem, resulting in

$$\begin{aligned} &\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i^p, \\ \text{such that } &y_i(\mathbf{w}\cdot\mathbf{x_i} + b) \geq 1 - \xi_i, \\ &\xi_i \geq 0, \end{aligned}$$

for i in $[m]$. The parameter $C \geq 0$ is a trade-off parameter that leverages between minimizing $\|\mathbf{w}\|^2$, and thus maximizing the margin, and the slack penalty. The dual problem can be derived as

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \underbrace{\langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j})\rangle}_{k(\mathbf{x_i},\mathbf{x_j})} \\ \text{such that } &\sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C, \end{aligned}$$

for i in $[m]$.

2.5. Neural networks

In the following, we focus on a special case of neural networks, namely the *multi-layer perceptron* (MLP). Let $\mathcal{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ be a subset of $\mathbb{R}^n$ for $n > 0$, and let $\mathbf{W}$ be a matrix in $\mathbb{R}^{n\times m}$, then one *layer* of an MLP is a function $f_{\mathbf{W}} \colon \mathbb{R}^n \to \mathbb{R}^m$ where

$$f_{\mathbf{W}}(\mathbf{x}) = \sigma(\mathbf{W}^T\mathbf{x}),$$

for $\mathbf{x}$ in $\mathcal{X}$. Here $\sigma \colon \mathbb{R} \to \mathbb{R}$ is a non-linearity, such as a sigmoid function or rectifier function, i.e.,

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}},$$

or

$$\sigma(\mathbf{x}) = \max(0, \mathbf{x}),$$

respectively, applied component-wise. In order to make the set of hypotheses more expressive, one composes several layers, resulting in the function

$$\mathbf{x} \mapsto \mathbf{W_l}^T\sigma(\mathbf{W_{l-1}}^T\sigma(\cdots \mathbf{W_2}^T\sigma(\mathbf{W_1}^T\mathbf{x}))).$$

The parameters $\mathbf{W_1}, \ldots, \mathbf{W_l}$ are then jointly optimized using a loss function, in most cases the *cross-entropy loss*

$$-\frac{1}{m}\sum_{i=1}^{m}\Big(c(\mathbf{x})\log(\sigma_{\mathrm{S}}(f_{\mathbf{W}}(\mathbf{x}))) + (1-c(\mathbf{x}))\log(1-\sigma_{\mathrm{S}}(f_{\mathbf{W}}(\mathbf{x})))\Big)$$

for binary classification, where σ_{S} [33, p. 179] denotes the softmax function, or the *mean squared error loss*

$$\frac{1}{m}\sum_{i=1}^{m}(c(\mathbf{x}) - f_{\mathbf{W}}(\mathbf{x}))^2$$

in the case of regression is employed.

The above function is non-convex, however in practice first-order methods, e.g., variants of the gradient descent algorithm [33, pp. 147 sqq.], are used to optimize the above function.

2.6. The Weisfeiler-Leman algorithm

Vertex refinement algorithms are a class of heuristics for the graph isomorphism problem. Given two graphs G and H, the idea is to discard bijections between the set of vertices between G and H that do not induce an isomorphism between G and H. Hence, if the algorithm discards all possible bijections we can be sure that the two graphs are not isomorphic. A well-known instance of this class is the *k-dimensional Weisfeiler-Leman algorithm* (k-WL), which iteratively partitions the set of k-tuples defined over the set of vertices of a graph by considering neighboring k-tuples.

2.6.1. The 1-dimensional Weisfeiler-Leman algorithm

We first describe the 1-dimensional variant of the algorithm. Let G be a graph, and let l be a (vertex) label function $V(G) \to \Sigma$, e.g., $l(v) = |N(v)|$ for v in $V(G)$. In each iteration $i \geq 0$, the 1-WL computes a coloring $C^1_i \colon V(G) \to \mathbb{S}$. In iteration 0, we set $C^1_0 = l$. Now in iteration $i > 0$, we set

$$C^1_i(v) = (C^1_{i-1}(v), \{\!\{C^1_{i-1}(u) \mid u \in N(v)\}\!\}), \tag{2.1}$$

for v in $V(G)$. In practice one maps the above pair to an unique value in $\mathbb{S}$, which has not been used in previous iterations.

For two graphs G and H, we run the algorithm in "parallel" on both graphs. Then the 1-WL *distinguishes* between them if they have an unequal number

of vertices labeled s in $\mathbb{S}$. Moreover, if $C^1_{i-1} = C^1_i$, the algorithm terminates. After at most $|V(G)| + |V(H)| + 1$ iterations the algorithm terminates. It is easy to see that the algorithm is not able to distinguish all non-isomorphic graphs, e.g., see [5]. On the other hand, this simple algorithm is already quite powerful, since it can distinguish almost all graphs with high probability (depending on the number of vertices) [6].

2.6.2. The k-dimensional Weisfeiler-Leman algorithm

In the following, we introduce two extensions of the 1-WL which are more powerful. First, we define the k-WL due to László Babai, see, e.g., [13], which is based upon algorithms proposed by Weisfeiler and Leman, see, e.g., [116, 115]. Moreover, we define the *δ-k-dimensional Weisfeiler-Leman algorithm* (δ-k-WL), which is a variant of the *k-dimensional combinatorial vertex coloring* algorithm due to Malkin [75]. We first formally define the k-WL.

Let G be a graph, and let $k \geq 2$. Moreover, let $\mathbf{v}$ be a tuple in $V(G)^k$, then $G[\mathbf{v}]$ is the subgraph induced by the components of $\mathbf{v}$, where the vertices are labeled with integers from $\{1, \ldots, k\}$ corresponding to indices of $\mathbf{v}$. In each iteration $i \geq 0$, the algorithm computes a *coloring* $C^k_i \colon V(G)^k \to \mathbb{S}$, where $\mathbb{S}$ is some arbitrary codomain. In the first iteration ($i = 0$), two tuples $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ get the same color if the map $v_i \mapsto w_i$ induces an isomorphism between $G[\mathbf{v}]$ and $G[\mathbf{w}]$. Now, for $i \geq 0$, C^k_{i+1} is defined by

$$C^k_{i+1}(\mathbf{v}) = (C^k_i(\mathbf{v}), M_i(\mathbf{v})), \tag{2.2}$$

where the multiset

$$\begin{aligned} M_i(\mathbf{v}) = \Big(&\{\!\{C^k_i(\phi_1(\mathbf{v}, w)) \mid w \in V(G)\}\!\}, \ldots, \\ &\{\!\{C^k_i(\phi_k(\mathbf{v}, w)) \mid w \in V(G)\}\!\}\Big), \end{aligned} \tag{2.3}$$

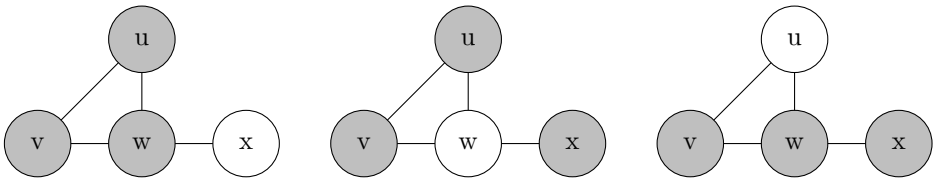
and

$$\phi_j(\mathbf{v}, w) = (v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k).$$

That is, $\phi_j(\mathbf{v}, w)$ replaces the j-th component of the tuple $\mathbf{v}$ with the vertex w. We run the algorithm until convergence, i.e.,

$$C^k_i(\mathbf{v}) = C^k_i(\mathbf{w}) \iff C^k_{i+1}(\mathbf{v}) = C^k_{i+1}(\mathbf{w}),$$

for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ holds, and call the partition of $V(G)^k$ induced by C^k_i the *stable partition*. For such i, we define $C^k_\infty(\mathbf{v}) = C^k_i(\mathbf{v})$ for $\mathbf{v}$ in $V(G)^k$.



(a) Underlying graph G, with tuple (u, v, w) (b) Local 3-neighbor (u, v, x) (c) Global 1-neighbor (x, v, w)

Figure 2.1.: Illustration of the local and global neighborhood of the 3-tuple (u, v, w).

For two graphs G and H, we run the algorithm in "parallel" on both graphs. Then the k-WL *distinguishes* between them if

$$|V(G)^k \cap (C^k_\infty)^{-1}(c)| \neq |V(H)^k \cap (C^k_\infty)^{-1}(c)|,$$

for some color c in the codomain of C^k_∞. Hence, if the k-WL distinguishes two graphs, the graphs are not isomorphic.

For $k = 1$, the Weisfeiler-Leman algorithm is based on the usual neighborhood of a vertex. That is, in the first iteration, we color the vertices uniformly. For $i \geq 0$, C^1_{i+1} is defined by

$$C^1_{i+1}(v) = (C^1_i(v), \{\!\{C^1_i(w) \mid w \in \delta(v)\}\!\}).$$

Hence, two vertices with the same color in iteration i get a different color in the next iteration if the number of neighbors colored with a certain color is different. Observe that it is straightforward to extend the 1-WL to labeled, directed graphs.

The δ-k-WL follows the same ratio but uses

$$C^{k,\delta,\overline{\delta}}_{i+1}(\mathbf{v}) = (C^{k,\delta,\overline{\delta}}_i(\mathbf{v}), M^{\delta,\overline{\delta}}_i(\mathbf{v})),$$

where,

$$\begin{aligned} M^{\delta,\overline{\delta}}_i(\mathbf{v}) = \Big(&\{\!\{(C^{k,\delta,\overline{\delta}}_i(\phi_1(\mathbf{v}, w)), 1_\delta((v_1, w))) \mid w \in V(G)\}\!\}, \ldots, \\ &\{\!\{(C^{k,\delta,\overline{\delta}}_i(\phi_k(\mathbf{v}, w)), 1_\delta((v_k, w))) \mid w \in V(G)\}\!\}\Big), \end{aligned} \tag{2.4}$$

instead of Equation (2.2) and Equation (2.3), respectively, where

$$1_\delta((u, w)) = \begin{cases} \mathrm{L} & \text{if } w \in \delta(u) \\ \mathrm{G} & \text{otherwise,} \end{cases}$$

for u and w in $V(G)$. For $i = 0$, we set $C_0^{k,\delta,\overline{\delta}} = C_0^k$. We say that $\phi_j(\mathbf{v}, w)$ is a *local j-neighbor* of $\mathbf{v}$ if w is in $\delta(v_j)$, and otherwise it is a *global j-neighbor*, which is indicated by L and G in Equation (2.4), respectively. See Figure 2.1 for an example. Hence, the difference between the two above algorithms is that the k-WL does not distinguish between local and global neighbors of a k-tuple. Observe that for $k = 1$, the above algorithms and the 1-WL Weisfeiler-Leman algorithm have the same power.

The following result relates the two algorithms from above. Since for a graph $G = (V, E)$, $M_i^{\delta,\overline{\delta}}(\mathbf{v}) = M_i^{\delta,\overline{\delta}}(\mathbf{w})$ implies $M_i(\mathbf{v}) = M_i(\mathbf{w})$ for all $\mathbf{v}$ and $\mathbf{w}$ in $V(G)^k$ and $i \geq 0$, the following holds.

Proposition 2.6.1. For all graphs and $k \geq 1$, the following holds:

$$\delta\text{-}k\text{-WL} \sqsubseteq k\text{-WL}.$$

Chapter 3.

Kernel methods for graphs

In the following, we present our work on graph kernels. A *graph kernel* is a kernel function defined on the set of graphs. In the first section of this chapter, we will review the related work. Subsequently, we present our work on scalable graph kernels for graphs with continuous labels, and kernels based on the k-dimensional Weisfeiler-Leman graph algorithm. Finally, we present our theoretical framework for investigating the expressiveness of graph kernels.

3.1. Related work

In the following, we give an overview of the graph kernel literature. We focus on the most influential work and work that has been published at major conferences. We start with kernels that are based on *neighborhood aggregation* techniques. The following subsections deal with *assignment-* and *matching-based* kernels, and kernels based on the extraction of *substructures*, respectively. The subsequent subsections deal with kernels based on *walks* and *paths*, kernels for graphs with continuous (vertex) labels, work that does not fit into the previous subsections, and *theoretical* work. The historical development of graph kernels and related methods is illustrated in Figure 3.1.

3.1.1. Neighborhood aggregation approaches

One of the dominating paradigms in the design of graph kernels is the representation and comparison of *local* structure. Two vertices are considered similar if they have identical labels—even more so if their neighborhoods are labeled similarly. Expanding on this notion, two graphs are considered similar if they are composed of vertices with similar neighborhoods, i.e., that they have similar local structure. The different ways by which local structure

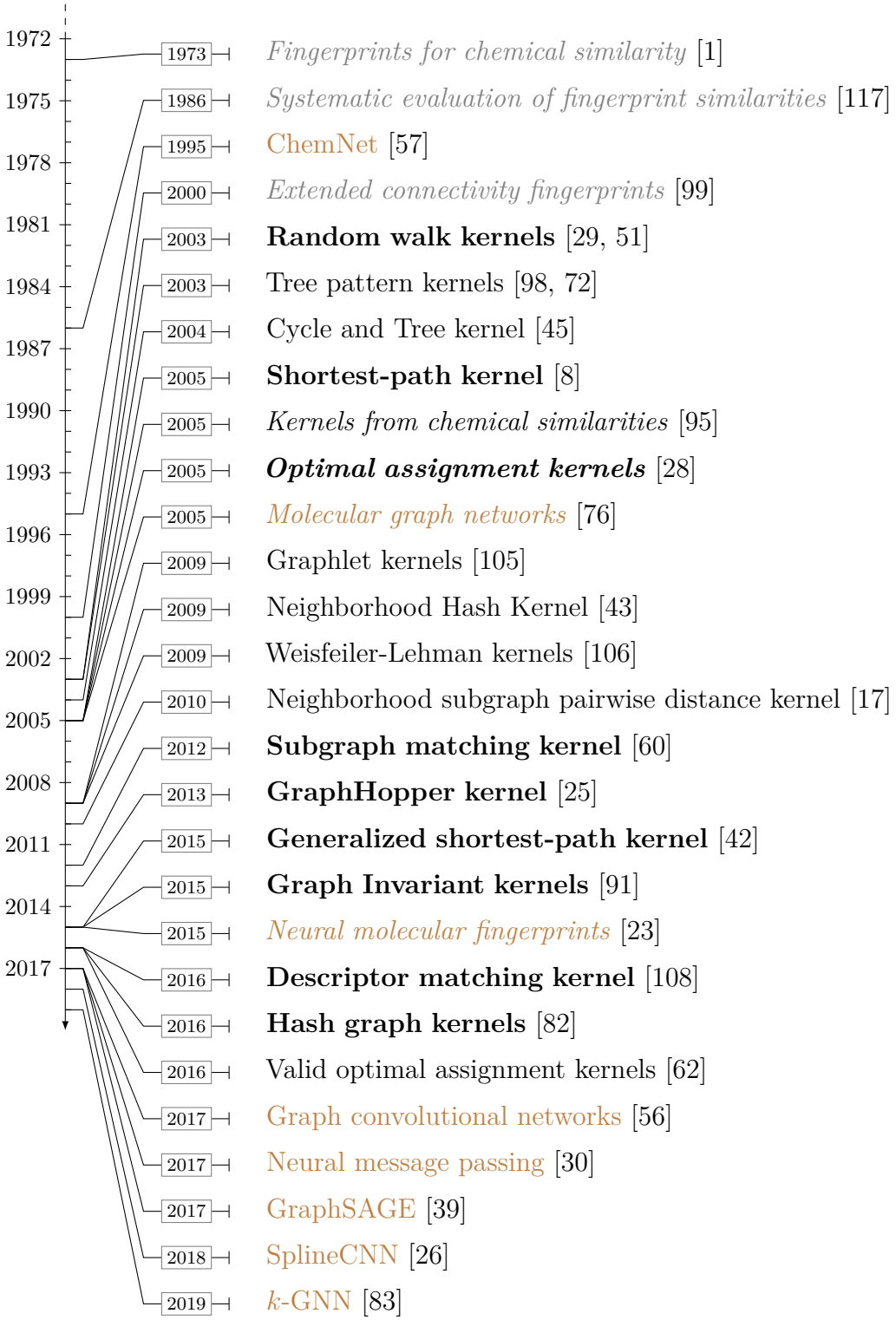


Figure 3.1.: Selected techniques for graph classification with a focus on kernels. Techniques based on fingerprints are marked in gray and methods using neural networks in brown. Methods proposed for cheminformatics are shown in *italics*, kernels for graphs with continuous labels in **bold**.

is defined, represented, and compared form the basis for several influential graph kernels. We describe a first example next.

Shervashidze et al. [106] introduced a class of kernels based on the 1-WL, see Section 2.6. The idea of the *Weisfeiler-Lehman subtree graph kernel* is to compute the above algorithm for $h \geq 0$ iterations resulting in a coloring function $C_i^1 \colon V(G) \to \mathbb{S}_i$ for each iteration $i \leq h$. After each iteration i, we compute a feature vector $\phi_i(G)$ for each graph G. Each component $\phi_i(G)_s$ counts the number of occurrences of vertices labeled with s in $\mathbb{S}_i$. The overall feature vector $\phi_{\mathrm{WL}}(G)$ is defined as the concatenation of the feature vectors of all h iterations, i.e.,

$$\phi_{\mathrm{WL}}(G) = \Big[\phi_0(G), \ldots, \phi_h(G)\Big].$$

The Weisfeiler-Lehman subtree kernel for h iterations then is computed as $k_{\mathrm{WL}}(G, H) = \langle \phi_{\mathrm{WL}}(G), \phi_{\mathrm{WL}}(H) \rangle$. The running time for a single feature vector computation is in $\mathcal{O}(hm)$ and $\mathcal{O}(Nhm + N^2hn)$ for the computation of the gram matrix for a set of N graphs [106], where n and m denote the maximum number of vertices and edges over all N graphs, respectively.

Moreover, based on the above Shervashidze et al. introduced two other graph kernels, the *Weisfeiler-Lehman edge* and the *Weisfeiler-Lehman shortest-path kernel*. Instead of counting the labels of vertices after each iteration the Weisfeiler-Lehman edge kernel counts the colors of edges corresponding to the colors of the incident vertices. The Weisfeiler-Lehman shortest-path kernel is a variant of the shortest-path kernel, see Section 3.1.4, where the vertex labels are replaced with the labels of the 1-WL after each iteration. In [43] a graph kernel similar to the 1-WL was introduced which replaces the neighborhood aggregation function by a function based on binary arithmetic. Similarly, in [84] another neighborhood aggregation function is defined which uses a randomized approach using p-stable locality-sensitive hashing [20], which is also able to handle real-valued vertex and edge labels.

3.1.2. Assignment- and matching-based approaches

A common approach to comparing two composite or structured objects is to identify the best possible matching of the components making up the two objects. For example, when comparing two chemical molecules it is instructive to map each atom in one graph to the atom in the other graph that is most similar in terms of, for example, neighborhood structure and attached chemical and physical measurements. This idea has been used also in graph kernels, an early example of which was proposed by Fröhlich

et al. [28] in the *optimal assignment* (OA) kernel. In the OA kernel, each vertex is endowed with a representation (e.g., a label) that is compared using a base kernel. Then, a similarity value for a pair of graphs is computed based on a mapping between their vertices such that the total similarity between the matched vertices with respect to a base kernel is maximized.

More formally, let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be subsets of objects from $\mathcal{X}$ and $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a base kernel. The *optimal assignment kernel* is

$$K_A(X, Y) = \max_{\pi \in S_n} \sum_{i=1}^{n} k(x_i, y_{\pi(i)}), \tag{3.1}$$

where π is a permutation of $\{1, \ldots, n\}$. To apply the assignment kernel to sets of different cardinality, we may fill the smaller set by objects z with $k(z, x) = 0$ for all x in $\mathcal{X}$. Unfortunately, Equation (3.1) is not a positive semi-definite kernel in general [111, 112]. This fact complicates the use of assignment similarities in kernel methods, although generalizations of SVMs for arbitrary similarity measures have been developed, see, e.g., [70] and references therein.

Different approaches using indefinite assignment similarities for kernel-based graph classification have been proposed. Woźnica, Kalousis, and Hilario [118] derived graph kernels from set distances and employed a matching-based distance, which was shown to be a metric [97]. To obtain a valid kernel, the authors use a set of *prototypes* and represent each graph by a feature vector, where each component is the distance to a prototype. Finally, a standard kernel is applied to compare the graph feature vectors. Johansson and Dubhashi [48] proposed to map the vertices of a graph into the d-dimensional real vector space to compute a matching between the vertices of two graphs with respect to the Euclidean distance. Several methods for the mapping are proposed, e.g., spectral approaches using the eigenvectors of the adjacency matrix, or an embedding associated with the Lovász number [71]. To obtain a valid kernel, graphs are again represented by feature vectors, where each component is the optimal assignment similarity to a prototype, here referred to as *landmark*.

Instead of generating feature vectors from prototypes, Kriege, Giscard, and Wilson [62] showed that Equation (3.1) is a valid kernel for a restricted class of base kernels k. These so-called *strong base kernels* give rise to hierarchies from which the optimal assignment kernels are computed in linear time by histogram intersection. For graph classification a base kernel based on the 1-WL was proposed. The derived *Weisfeiler-Lehman optimal assignment kernel* often provides better classification accuracies on real-world benchmark

datasets than the Weisfeiler-Lehman subtree kernel.

Pachauri, Kondor, and Singh [92] studied a generalization of the assignment problem to more than two sets, which was used to define *transitive assignment kernels* for graphs [102]. The method is based on finding a single assignment between the vertices of all graphs of the dataset instead of finding an optimal assignment for each pair of graphs. While this leads to positive-semidefinite kernels, non-optimal assignments between individual pairs of graphs are possible. Nikolentzos, Meladianos, and Vazirgiannis [88] proposed a matching-based approach based on the Earth Mover's Distance, which results in an indefinite kernel function. In order to deal with this they employ a variation of the SVM algorithm, specialized for learning with indefinite kernels. Additionally, they propose an alternative solution based on the *pyramid match kernel*, a generic kernel for comparing sets of features [35]. The pyramid match kernel avoids the indefiniteness of other assignment kernels by comparing features through a multi-resolution histogram (with bins determined globally, rather than for each pair of graphs).

3.1.3. Substructure-based approaches

In many applications, a strong baseline for representations of composite objects such as documents, images or graphs is one that ignores the structure altogether and represents objects as *bags of components*. A well-known example is the so-called bag-of-words representation of text—statistics of word occurrences without context—which remains a staple in natural language processing. For additional specificity, it is common to compare statistics also of bigrams (sequences of two words), trigrams, etc. A similar idea may be used to compare graphs by ignoring global structure and viewing graphs as bags of vertices or edges. The *vertex label kernel* does precisely this by comparing graphs only at the level of similarity between all pairs of vertex labels from two different graphs, i.e.,

$$k_{\mathrm{VL}}(G, H) = \sum_{u \in V(G)} \sum_{v \in V(G)} k(l(u), l(v)).$$

With the base kernel k the equality indicator function, k_{VL} is a linear kernel on the (unnormalized) distributions of vertex labels in G and H. Similar in spirit, the *edge label kernel* is defined as the sum of base kernel evaluations on all pairs of edge labels (or triplets of the edge label and incident vertex labels). A downside of vertex and edge label kernels is that they ignore the interplay between structure, and labels and are almost completely uninformative for unlabeled graphs. Instead of viewing graphs as bags of vertices or edges,

we may view them as bags of *subgraph patterns*. To this end, Shervashidze et al. [105] introduced a kernel based on counting occurrences of subgraph patterns of a fixed size—so called *graphlets*.

The Graphlet kernel counts the isomorphism types of all induced (possibly disconnected) subgraphs on $k > 0$ vertices of a graph G. Let $\phi(G)_{\tau_i}$ for $1 \leq i \leq N$ denote the number of instances of isomorphism type τ_i where N denotes the number of different types of subgraphs induced by k vertices. The kernel computes a feature vector $\phi_{\mathrm{GR}}(G)$ for G,

$$\phi_{\mathrm{GR}}(G) = [\phi(G)_{\tau_1}, \ldots, \phi(G)_N].$$

The graphlet kernel is finally defined as $k_{\mathrm{GR}}(G, H) = \langle \phi_{\mathrm{GR}}(G), \phi_{\mathrm{GR}}(H) \rangle$ for two graphs G and H.

The time required to compute the graphlet kernel scales exponentially with the size of the considered graphlets. To remedy this, Shervashidze et al. proposed two algorithms for speeding up the computation time of the feature vector for k in $\{3, 4\}$. In particular, it is common to restrict the kernel to connected graphlets. Additionally, the statistics used by the graphlet kernel may be estimated approximately by subgraph sampling, see, e.g., [49, 2, 16, 9]. Similarly, Horváth, Gärtner, and Wrobel [45] proposed a kernel based on cycles and tree patterns.

Costa and De Grave [17] introduced the *neighborhood subgraph pairwise distance kernel* which associates a string with every vertex representing its neighborhood up to a certain depth. To avoid solving the graph canonization problem, they proposed using a graph invariant that may map non-isomorphic neighborhood subgraphs to the same string. Then, pairs of these neighborhood graphs together with the shortest-path distance between their central vertices are counted as features.

An alternative to subgraph patterns are tree patterns which may contain repeated vertices and were initially proposed for use in graph comparison by Ramon and Gärtner [98], and later refined by Mahé and Vert [72]. Tree pattern kernels are similar to the Weisfeiler-Lehman subtree kernel, but do not consider all neighbors in each step, but also all possible subsets [106], and hence do not scale to larger datasets. Da San Martino, Navarin, and Sperduti [19] proposed decomposing a graph into trees and applying a kernel defined on trees. In Da San Martino, Navarin, and Sperduti [18], a fast hashing-based computation scheme for the aforementioned graph kernel is proposed.

3.1.4. Walk- and path-based approaches

A downside of the subgraph pattern kernels described in the previous section is that they require the specification of a set of patterns, or subgraph size, in advance. To ensure efficient computation, this often restricts the patterns to a fairly small scale, emphasizing local structure. A popular alternative is to compare the sequences of vertex or edge labels that are encountered through traversals through graphs. In this section, we describe two families of traversal algorithms which yield different labels sequences and thus different kernels—shortest paths and random walks.

A first graph kernel that fits into this framework is the *shortest-path kernel* [8]. The idea here is to compare the shortest-path distributions of two graphs. Formally, let G and H be graphs with label function $l\colon V(G) \cup V(H) \to \Sigma$ and let $d\colon V(G) \cup V(H) \times V(G) \cup V(H) \to \mathbb{N}$ denote the shortest-path distance function. Then the kernel is defined as

$$k_{\mathrm{SP}}(G, H) = \sum_{\substack{(u,v)\in V(G)^2 \\ u\neq v}} \sum_{\substack{(w,z)\in V(H)^2 \\ w\neq z}} k((u, v), (w, z)), \tag{3.2}$$

where

$$k((u, v), (w, z)) = k_{\mathrm{L}}(l(u), l(w)) \cdot k_{\mathrm{L}}(l(v), l(z)) \cdot k_{\mathrm{D}}(d(u, v), d(w, z)).$$

Here k_{L} is a kernel for comparing (continuous) vertex labels and k_{D} is a kernel to compare shortest-path distances, such that $k_{\mathrm{D}}(d(u, v), k(w, z)) = 0$ if $d(u, v) = \infty$ or $d(w, z) = \infty$. The running time for evaluating the kernel function for a pair of graphs is in $\mathcal{O}(n^4)$, using the Floyd-Warshall algorithm for solving the all-pair shortest-path problem. Observe that this kernel can naturally handle continuous labels. In [42] the shortest-path is generalized by considering all shortest paths between two vertices.

Gärtner, Flach, and Wrobel [29] and Kashima, Tsuda, and Inokuchi [51] derived graph kernels based on random walks, which count the common walks of two graphs and support user-specified kernels for (continuous) vertex and edge labels. Subsequently, Mahé et al. [73] extended this formulation of random walk kernels with a focus on application in cheminformatics [74]. An unfavorable characteristic of random walks is that they may visit the same vertex several times, resulting in walks that do not provide additional information about the structure of the graph. This was partially fixed by extending random walk kernels to second-order Markov processes. Gärtner, Flach, and Wrobel [29] explicitly define the feature space of their random walk kernel as label sequences derived from walks, but propose a different method

of computation based on the *direct product graph* of two labeled input graphs. Vishwanathan et al. [112] propose a generalizing framework for random walk based graph kernels and argue that the approach by Kashima, Tsuda, and Inokuchi and Gärtner, Flach, and Wrobel can be considered special cases of this kernel. Recently, the phenomenon of *halting* in random walk kernels has been studied by Sugiyama and Borgwardt [109], which refers to the fact that walk-based graph kernels may down-weight longer walks so much that their value is dominated by walks of length 1. Moreover, Zhang et al. [129] derived graph kernels based on return probabilities of random walks.

3.1.5. Convolution graph kernels for graphs with continuous labels

Most real-world graphs have continuous vertex and edge labels, mostly real-valued vectors, associated with their vertices and edges, respectively. For example, atoms of chemical molecules have physical and chemical properties; individuals in social networks have demographic information; and words in documents carry semantic meaning. Kernels based on pattern counting or neighborhood aggregation are of a discrete nature, i.e., two vertices are regarded as similar if and only if they exactly match, structure-wise as well as (continuous) label-wise. However, in most applications, it is desirable to compare real-valued labels with more nuanced similarity measures such as the Gaussian RBF kernel.

Kernels suitable for graphs with continuous labels typically use user-defined kernels for the comparison of vertex and edge labels. The graph kernel is then obtained by combining these kernels according to closure properties. Recently proposed kernels for graphs with continuous labels like *GraphHopper* [25] and *GraphInvariant* [91] use separate kernels for the graph structure and annotations. They can be expressed as

$$k_{\mathrm{WV}}(G,H) = \sum_{v \in V(G)} \sum_{w \in V(H)} k_W(v,w) \cdot k_V(v,w), \tag{3.3}$$

where k_V is a user-specified kernel comparing continuous vertex labels and k_W is a kernel that determines a weight for a vertex pair based on the individual graph structures. Kernels belonging to this family are easily identifiable as instances of R-convolution kernels [41].

For graphs with continuous labels one could set k_V to the *Gaussian RBF kernel*

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

for $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$, where σ is a free parameter. The selection of the kernel k_W is essential to take the graph structure into account and allows to obtain different instances of the above kernel. A reasonable implementation of k_W motivated along the lines of the *GraphInvariant* kernel [91] is

$$k_W(v, w) = \sum_{i=0}^{h} k_\delta(C_i^1(v), C_i^1(w)).$$

Intuitively, this kernel reflects to what extent the two vertices have a structurally similar neighborhood.

Another graph kernel, which fits into this framework, is the *GraphHopper* kernel [25] with

$$k_W(v, w) = \langle \mathbf{M}(v), \mathbf{M}(w) \rangle_F.$$

Here $\mathbf{M}(v)$ and $\mathbf{M}(w)$ are $\delta \times \delta$ matrices, where the entry $\mathbf{M}(v)_{ij}$ for v in $V(G)$ counts the number of times the vertex v appears as the i-th vertex on a shortest path of discrete length j in G, δ denotes the maximum diameter over all graphs, and $\langle \cdot, \cdot \rangle_F$ is the Frobenius inner product.

In [60], Kriege and Mutzel proposed the *subgraph matching kernel* which is computed by considering all bijections between all subgraphs on $k > 0$ vertices, and allows to compare vertex labels using a custom kernel. Moreover, in [108] the *Descriptor Matching kernel* is defined, which captures the graph structure by a propagation mechanism between neighbors, and uses a variant of the VG kernel [34] to compare labels between vertices. The kernel can be computed in linear-time in the number of edges.

3.1.6. Other approaches

Kondor, Shervashidze, and Borgwardt [59] derived a graph kernel using graph invariants based on group representation theory. In [58] a graph kernel is proposed which can capture the graph structure at multiple scales, i.e., neighborhoods around vertices of increasing depth using ideas from spectral graph theory. Moreover, the authors provide a low-rank approximation algorithm to scale the kernel computation to large graphs. Johansson et al. [50] defined a graph kernel based on the Lovász number [71].

Yanardag and Vishwanathan [121] used recent neural techniques from neural language modeling, such as *skip-gram* [77]. The authors build on known state-of-the-art kernels but allow to respect relationships between their features. This is demonstrated by hand-designed matrices encoding the similarities between features for selected graph kernels such as the graphlet and Weisfeiler-Lehman subtree kernel. Similar ideas were used in [122] where

smoothing methods for multinomial distributions were applied to the graph domain.

In [68] a kernel for dynamic graphs is proposed, where vertices and edges are added and deleted, respectively. The kernel is based on eigendecompositions. Kriege et al. [61] investigated under which conditions it is possible to compute the corresponding feature map of a graph kernel. They provide theoretical as well as empirical results for walk-based kernels. Li et al. [67] proposed a streaming version of the Weisfeiler-Leman algorithm using hashing techniques. Aiolli et al. [3] applied multiple kernel learning to the graph kernel domain.

Nikolentzos et al. [89] proposed to first build the k-core decomposition of graphs to obtain a hierarchy of nested subgraphs, which are then individually compared by a graph similarity measure. The approach has been combined with several graph kernels such as the Weisfeiler-Lehman subtree kernel. and was shown to improve the accuracy on some datasets.

3.1.7. Theoretical work

While a large literature has studied the empirical performance of various graph kernels, there exist comparatively few works that deal with graph kernels exclusively from a theoretical point of view. Most works that provide theoretical insights for graph kernels attempt to formalize their *expressivity*.

The *expressivity* of a graph kernel refers broadly to the kernel's ability to distinguish certain patterns and properties of graphs. In an early attempt to formalize this notion, Gärtner, Flach, and Wrobel introduced the concept of a *complete graph kernel*—kernels for which the corresponding feature map is an injection. If a kernel is not complete, there are non-isomorphic graphs G and H with $\phi(G) = \phi(H)$ that cannot be distinguished by the kernel. In this case, there is no way any classifier based on this kernel can separate these two graphs. However, computing a complete graph kernel is GI-hard, i.e., at least as hard as deciding whether two graphs are isomorphic [29]. Therefore, none of the graph kernels used in practice are complete. Note, however, that a kernel may be injective with respect to a finite or restricted family of graphs.

Johansson and Dubhashi [48] proved that optimal assignment kernels based on Laplacian embeddings of graphs can distinguish graphs with different densities as well as random graphs with and without planted cliques. In Johansson et al. [50], the authors studied global properties of graphs such as girth, density, and clique number and proposed kernels based on vertex embeddings associated with the Lovász-ϑ and SVM-ϑ numbers which have been shown to capture these properties.

The expressivity of graph kernels has been studied also from statistical

perspectives. In particular, Oneto et al. [90] used well-known results from statistical learning theory to give results which bound measures of expressivity in terms of Rademacher complexity and stability theory. Johansson et al. studied the statistical tradeoff between expressivity and differential privacy [24]. Finally, the framework in Section 3.4 proposes an alternative way to measure the expressiveness of graph kernels.

3.2. Fast kernels for graphs with continuous labels

The various graph kernels proposed in recent years can be divided into approaches that either compute feature maps (i) explicitly, or (ii) implicitly, see [61] and Section 3.1. Explicit computation schemes have been shown to be scalable and allow the use of fast linear kernel methods, e.g., linear support vector machines [47], while implicit computation schemes are typically slow due to the quadratic running time overhead for comparing each pair of graphs. Alternatively, we may divide graph kernels according to their ability to handle annotations of vertices and edges. The proposed graph kernels are either (i) restricted to discrete labels, or (ii) compare annotations like continuous labels by user-specified kernels. Kernels of the first category typically implicitly compare annotations of vertices and edges by the trivial Dirac kernel, which requires values to match exactly and is not adequate for continuous labels. Remarkably, the two classifications of graph kernels mentioned above largely coincide: Graph kernels supporting complex annotations use implicit computation schemes and do not scale well. Whereas graphs with discrete labels can be compared efficiently by graph kernels based on explicit feature maps. Table 3.1 gives an overview of the related kernels and the model of computation. In the following, we introduce *hash graph kernels* which use hash functions to approximate implicit kernels for continuous labels by finite dimensional feature vectors. Hence, they circumvent the problem of quadratic overhead.

3.2.1. Hash graph kernels

The main idea of hash graph kernels is to map continuous labels to discrete labels using a *family of hash functions*, and then apply a kernel for graphs with discrete labels. Let $\mathcal{H} = \{h \colon \mathbb{R}^d \to \mathbb{N}\}$ be a family of hash functions, and G a graph with continuous vertex labels $a \colon V(G) \to \mathbb{R}^d$. We can transform (G, a) to a graph with discrete labels by mapping each continuous label $a(v)$ to $h(a(v))$ with some function h in $\mathcal{H}$. For short, we write $\boldsymbol{h}(G)$ for the labeled graph obtained by this procedure. The function h is drawn uniformly at random from the family of hash functions $\mathcal{H}$. This procedure is repeated multiple times to lower the variance. Thus, we obtain a sequence of discretely labeled graphs $(\boldsymbol{h}_i(G))_{i=1}^{I}$, where I is the number of iterations. Hash graph kernels compare these sequences of labeled graphs by an arbitrary graph kernel for labeled graphs, which we refer to as *discrete base graph kernel*, e.g.,

Table 3.1.: Summary on selected graph kernels regarding computation by explicit (EX) and implicit (IM) feature map and support for continuously labeled graphs. ⋆— not considered in publication, but method can be extended; †— continuous vertex labels only).

Graph Kernel	Model of Computation	Labels	Continuous Labels
Random Walk ([29, 51])	IM	✓	✓
Tree Pattern ([98, 72])	IM	✓	✓⋆
Shortest-Path ([8])	IM	✓†	✓
Subgraph Matching ([60])	IM	✓	✓
GraphHopper ([25])	IM	✓†	✓
Graph Invariant kernels ([91])	IM	✓	✓
Descriptor Matching Kernel ([108])	IM	✓	✓
Weisfeiler-Lehman Subtree ([106])	EX	✓	✗
Graphlet ([105])	EX	✓⋆	✗
NSPDK ([17])	EX	✓	✗
Propagation ([84])	EX	✓	✓
Hash Graph Kernel with Shortest-path kernel	EX	✓	✓
Hash Graph Kernel with Weisfeiler-Lehman Subtree kernel	EX	✓	✓

the Weisfeiler-Lehman subtree or the shortest-path kernel.

Definition 3.2.1. Let $\mathcal{H}$ be a family of hash functions and k_{b} a discrete base graph kernel, then the *hash graph kernel* for two continuously labeled graphs G and H is defined as

$$k_{\mathrm{HGK}}(G, H) = \frac{1}{I} \sum_{i=1}^{I} k_{\mathrm{b}}(\boldsymbol{h}_i(G), \boldsymbol{h}_i(H)),$$

where $\boldsymbol{h}_i$ is obtained by choosing hash functions (uniformly at randomly) from $\mathcal{H}$.

We will discuss hash functions, possible ways to choose them from $\mathcal{H}$, and how they relate to the kernel value in Section 3.2.3. We proceed with the algorithmic aspects of hash graph kernels. It is desirable for efficiency to compute explicit feature vectors for graph kernels. We can obtain feature vectors for hash graph kernels under the assumption that the discrete base graph kernel can be computed by explicit kernels. This can be achieved by concatenating the feature vectors for each iteration and normalizing by $\sqrt{1/I}$ according to the pseudocode in Algorithm 1.

Note that lines 4 to 5 in Algorithm 1 can be easily executed in parallel. Moreover, when using, e.g., the Weisfeiler-Lehman subtree kernel as a discrete

Algorithm 1 Explicit feature maps for hash graph kernels

1: **Input:** A graph with continuous vertex labels (G, a), a graph feature map $\phi_{\mathrm{L}} \colon \mathcal{G} \to \mathbb{H}$ of the discrete base graph kernel k_{b} for labeled graphs, and number of iterations I in $\mathbb{N}_{>0}$
2: **Output:** A feature vector $\phi(G)$ for (G, a)

3: **parallel for** i in $\{1, \ldots, I\}$ **do**
4: $\quad (G, l) \leftarrow \boldsymbol{h}_i(G)$ ▷ Hash continuous labels to labels
5: $\quad \phi(G) \leftarrow [\phi(G), \phi_{\mathrm{L}}((G, l))]$ ▷ Concatenate vectors
6: **end**

7: $\phi(G) \leftarrow \sqrt{1/I} \cdot \phi(G)$ ▷ Normalize

8: **Return** $\phi(G)$

base kernel, line 5 can be executed in one step over the whole set of graphs, cf. [106, 61].

Difference to the propagation kernel

The *propagation kernel* [84] also employs hash functions to discretize continuous information. However, propagation kernels discretize continuous probability distributions. Moreover, they do not provide theoretical bounds, see Section 3.2.4, and the experimental results, see Section 3.2.5, indicate that hash graph kernels outperform propagation kernels.

3.2.2. Analysis

Since hash graph kernels are a normalized sum over discrete base graph kernels applied to a sequence of transformed input graphs, it is clear that we again obtain a valid kernel. For the explicit computation of feature maps by Algorithm 1 we get the following bound on the running time.

Proposition 3.2.2. Let $k \colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ be a graph kernel for labeled graphs and let $\phi_{\mathrm{L}} \colon \mathcal{G} \to \mathbb{H}$ be the corresponding feature map acting as a discrete base graph kernel for Algorithm 1, then the running time for the computation of a feature vector of G in $\mathcal{G}$ is in

$$\mathcal{O}(I \cdot (\mathsf{T}_{\mathrm{H}}(G) + \mathsf{T}_{\phi}(G))),$$

where $\mathsf{T}_{\mathrm{H}}(G)$ denotes an upper bound for the evaluation of the hash functions for G, and $\mathsf{T}_{\phi}(G)$ denotes an upper bound for the running time to compute the graph feature map $\phi_{\mathrm{L}}(G)$.

Proof. Directly follows from Algorithm 1. □

Notice that when we fix the number of iterations and assume $\mathsf{T}_{\mathrm{H}}(G) \leq \mathsf{T}_{\phi}(G)$, the hash graph kernel can be computed in the same asymptotic running time as the discrete base graph kernel.

3.2.3. Hash functions

In this section, we discuss possible realizations of the hashing functions used to obtain hash graph kernels according to Definition 3.2.1. Our goal is to establish the necessary background to study hash graph kernels obtained for concrete discrete base graph kernels, and how they implicitly compare continuous labels in Section 3.2.4. The key idea is to choose a family of hash functions, and draw hash functions h_1 and h_2 in each iteration such that

$$\Pr[h_1(\mathbf{x}) = h_2(\mathbf{y})]$$

is an adequate measure of similarity between $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^d$. For the case that $h_1 = h_2$ drawn at random, such families of hash functions have been proposed, e.g., by [94] and [4]. However, the results on these approaches do not lift to kernels composed of products of kernels, see the subsection below. Hence, they do not directly transfer to hash graph kernels, where complex discrete base graph kernels are employed. To overcome this issue, we introduce the following concept.

Definition 3.2.3. Let $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel, and let $\mathcal{H} = \{h\colon \mathcal{X} \to S\}$ for some set S be a family of hash functions. Then $\mathcal{H}$ is an *independent k-hash family* if

$$\Pr[h_1(x) = h_2(y)] = k(x, y),$$

where h_1 and h_2 are chosen independently and uniformly at random from $\mathcal{H}$.

In the following, we consider a particularly simple example of a hash family and illustrate the kernels obtained from either applying the same hash function to both objects under comparison or drawing them independently. Then we review locality sensitive hashing techniques for hash graph kernels, which were used in the empirical evaluation.

Randomized binning

We discuss the technique proposed by [94] to approximate the *triangular* or *hat kernel* as a normalized sum of Dirac kernels applied after randomized binning. The hat kernel on $\mathbb{R}$ is defined as

$$k_{\Delta}(x, y) = \max\left\{0, 1 - \frac{|x - y|}{\gamma}\right\},$$

where $\gamma > 0$ is a parameter describing the maximal discrepancy of x and y, for which k_{Δ} takes a non-zero value, cf. Figure 3.2. Consider the family of hash functions

$$d^{u}_{\text{bin}}(x) = \left\lfloor \frac{x - u}{\gamma} \right\rfloor,$$

where u is drawn uniformly at random from $[0, \gamma]$. Let h be a hash function chosen at random from this family, then

$$\Pr\left[h(x) = h(y)\right] = k_{\Delta}(x, y)$$

and

$$\frac{1}{I}\sum_{i=1}^{I} k_{\delta}(h_i(x), h_i(y))$$

converges to $k_{\Delta}(x, y)$ with increasing parameter I [94]. However, when this approach is used to obtain hash graph kernels, the above result does not imply that continuous labels are compared by an (approximated) hat kernel. Consider two pairs of vertices with continuous labels a, b and c, d in $\mathbb{R}$, respectively. The discrete base graph kernel might require that the hash values of a and c as well as b and d both exactly match in order to contribute to the kernel value. However, in general

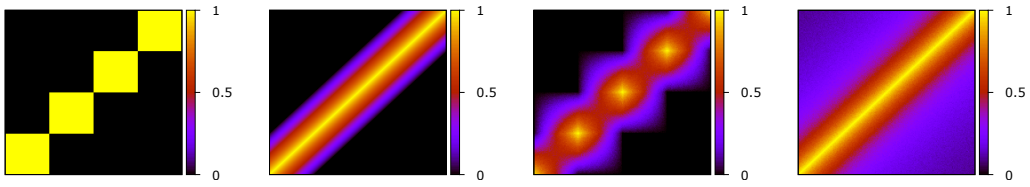
$$k_{\Delta}(a, c) \cdot k_{\Delta}(b, d) \neq \Pr\left[h(a) = h(c) \wedge h(b) = h(d)\right],$$

and as a consequence the approach does not generalize to kernels composed by multiplication like most graph kernels. Indeed, the hash family defined above is not an independent k_{Δ}-hash family according to Definition 3.2.3.

However, it is an independent k_{ξ}-hash family for some kernel

$$k_{\xi}(x, y) = \Pr[h_1(x) = h_2(y)],$$

where the hash functions h_1 and h_2 both are selected uniformly at random. This kernel is illustrated in Figure 3.2. Although it does not guarantee $k_{\xi}(x, x) = 1$ for all x in $\mathbb{R}$ like the hat kernel, see Figure 3.2, it is continuous on the bin boundaries compared to the simple binning kernel.



(a) $k_\delta(\lfloor x \rfloor, \lfloor y \rfloor)$. (b) k_Δ with $\gamma = 1$. (c) k_ξ with $\gamma = 1$. (d) 2-stable LSH.

Figure 3.2.: Illustration of different kernels applied to x and y in $\{0, \ldots, 4\}$. The kernel in (a) is based on straight-forward binning, while the kernels (b) to (d) are obtained by iterated randomized binning.

Locality sensitive hashing

In this section, we review *locality sensitive hashing* (LSH) [46]. Its general idea is to hash points to the same bin with high probability if they are close together with regard to some distance function. Let S be a set and $D\colon S \times S \to \mathbb{R}_{\geq 0}$ some distance function defined on S, then an *LSH family* is defined as follows.

Definition 3.2.4 ([20])**.** A family of hash functions $\mathcal{H} = \{h\colon \mathcal{X} \to S\}$ for some set S is (r_1, r_2, p_1, p_2)-sensitive for D if for all v and w in $\mathcal{X}$

1. if v in $B(w, r_1)$, then $\Pr_{\mathcal{H}}[h(v) = h(w)] \geq p_1$, and
2. if v not in $B(w, r_2)$, then $\Pr_{\mathcal{H}}[h(v) = h(w)] \leq p_2$

holds, where $B(w, r) = \{s \in S \mid D(w, s) \leq r\}$.

Finally, we require *p-stable* LSH [20]. We first need to define *s-stable distributions*.

Definition 3.2.5 ([130])**.** Let $\mathbf{v}$ be a vector in $\mathbb{R}^n$, $\mathcal{D}$ be a distribution on $\mathbb{R}$, and p in $(0, 2]$. Moreover, let $X_1, \ldots, X_n$, and Z be random variables drawn i.i.d. from $\mathcal{D}$, then $\mathcal{D}$ is p-stable if $\mathbf{v}^T(X_1, \ldots, X_n) \sim \|\mathbf{v}\|_p Z$.

For example the Cauchy distribution and the standard normal distribution are 1-stable and 2-stable [130], respectively. Let $\mathbf{v_1}$ and $\mathbf{v_2}$ in $\mathbb{R}^n$ be two vectors we want to map to an integer, and let $\mathbf{a}$ be a vector in $\mathbb{R}^n$ such that each component of $\mathbf{a}$ is i.i.d. drawn from a p-stable distribution $\mathcal{D}$. Because of p-stability the distance $(\mathbf{a}^T\mathbf{v_1} - \mathbf{a}^T\mathbf{v_2})$ is distributed as $\|\mathbf{v_1} - \mathbf{v_2}\|_p Z$, where Z is a random variable with distribution $\mathcal{D}$. The dot product $\mathbf{a}^T\mathbf{v_i}$ maps $\mathbf{v_i}$

to the real line. By adding an offset b, chosen uniformly at random from $[0, r]$, and partitioning the real line into equidistant intervals of length r, we get the following binning function:

$$d_{\text{bin}}^{\mathbf{a},b}(\mathbf{v}) = \left\lfloor \frac{\mathbf{a}^T\mathbf{v} + b}{r} \right\rfloor \text{ for } \mathbf{v} \text{ in } \mathbb{R}^n.$$

Datar et al.[20] showed the following result.

Proposition 3.2.6 ([20])**.** Let $\mathbf{v_1}$ and $\mathbf{v_2}$ be in $\mathbb{R}^n$, then

$$p(c) = \Pr\left[d_{\text{bin}}^{\mathbf{a},b}(\mathbf{v_1}) = d_{\text{bin}}^{\mathbf{a},b}(\mathbf{v_2})\right] = \int_0^r \frac{1}{c} f_p\left(\frac{t}{c}\right)\left(1 - \frac{1}{r}\right) dt,$$

where $c = \|\mathbf{v_2} - \mathbf{v_2}\|_p$ and f_p denotes the probability density function of the absolute value of the p-stable distribution, e.g., the folded normal distribution for $p = 2$. Moreover, $d_{\text{bin}}^{\mathbf{a},b}$ is (r_1, r_2, p_1, p_2)-sensitive for $p_1 = p(1)$ and $p_2 = p(c)$ for $r_2/r_1 = c$.

The binning function used for LSH is similar to the one discussed in the previous subsection. However, LSH allows to map multi-dimensional real values to discrete labels and, thus, extends the applicability of hash graph kernels to graphs annotated with multidimensional continuous labels. See Figure 3.2 for an illustration of the computed kernel.

3.2.4. Hash graph kernel instances

In the following, we prove that hash graph kernels approximate implicit variants of the shortest-path, and the Weisfeiler-Lehman subtree kernel for graphs with continuous vertex labels.

Shortest-path kernel

We first describe a hash graph kernel instance for the implicit shortest-path graph kernel which can handle continuous labels, cf. Equation (3.2). If we assume a labeled graph (G, l) and set k_{L} and k_{D} to the Dirac kernel, we can compute an feature map $\phi_{\text{SP}} \colon \mathcal{G} \to \mathbb{R}^d$ for some $d > 0$ for $k_{\text{SP}}^{k_{\text{L}},k_{\text{D}}}(G, H)$. Each component of $\phi_{\text{SP}}(G)$ counts the number of occurrences of a triple of the form $(l(u), l(v), d_{uv})$ for (u, v) in $V(G)^2$, $u \neq v$, and $d_{uv} < \infty$. It is easy to see that

$$\phi_{\text{SP}}(G)^T \phi_{\text{SP}}(H) = k_{\text{SP}}^{k_\delta,k_\delta}(G, H). \tag{3.4}$$

The following theorem shows that we can approximate $k_{\text{SP}}^{k_{\text{L}},k_\delta}$ arbitrarily close by hash graph kernels by using the explicit shortest-path kernel as a discrete base kernel and an independent k_{L}-hash family.

Theorem 3.2.7. Let $k_{\mathrm{L}}\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a kernel, and let $\mathcal{H}$ be an independent k_{L}-hash family. Assume that in each iteration of Algorithm 1 each continuous vertex label is mapped to a label using a hash function chosen independently and uniformly at random from $\mathcal{H}$. Then Algorithm 1, with the explicit shortest-path graph kernel acting as the discrete base kernel, approximates $k_{\mathrm{SP}}^{k_{\mathrm{L}},k_\delta}$ such that

$$\Pr\Big[\Big|\phi(G)^T\phi(H) - k_{\mathrm{SP}}^{k_{\mathrm{L}},k_\delta}(G,H)\Big| \geq \lambda\Big] \leq 2\exp\Big(-2\lambda^2 I\Big).$$

Moreover, by setting

$$I = \frac{1}{2\epsilon^2}\log(|\mathcal{G}|^2 \cdot c),$$

for some constant $c > 0$ it holds with any constant probability that

$$\sup_{G,H\in\mathcal{G}} \Big|\phi(G)^T\phi(H) - k_{\mathrm{SP}}^{k_{\mathrm{L}},k_\delta}(G,H)\Big| \leq \epsilon,$$

for $\epsilon > 0$.

Proof. By assumption, we have $\Pr[h_1(\mathbf{a}) = h_2(\mathbf{b})] = k_{\mathrm{L}}(\mathbf{a},\mathbf{b})$ for h_1 and h_2 chosen independently and uniformly at random from $\mathcal{H}$. Since we are using a Dirac kernel to compare discrete labels, we get

$$\mathbb{E}[k_\delta(h_1(\mathbf{a}), h_2(\mathbf{b}))] = k_{\mathrm{L}}(\mathbf{a},\mathbf{b}).$$

Since $\mathcal{H}$ is an independent k_{L}-hash family, $\Pr[h_1(\mathbf{a}) = h_2(\mathbf{b}) \wedge h_3(\mathbf{c}) = h_4(\mathbf{d})] = k_{\mathrm{L}}(\mathbf{a},\mathbf{b}) \cdot k_{\mathrm{L}}(\mathbf{c},\mathbf{d})$ for h_1, h_2, h_3, and h_4 chosen independently and uniformly at random from $\mathcal{H}$. Hence, by the above,

$$\mathbb{E}[k_\delta(h_1(\mathbf{a}), h_2(\mathbf{b})) \cdot k_\delta(h_3(\mathbf{c}), h_4(\mathbf{d}))] = k_{\mathrm{L}}(\mathbf{a},\mathbf{b}) \cdot k_{\mathrm{L}}(\mathbf{c},\mathbf{d}).$$

By Equation (3.4), and using the linearity of expectation,

$$\mathbb{E}[\phi(G)^T\phi(H)] = k_{\mathrm{SP}}(G,H).$$

Now assume that $k^{k_{\mathrm{L}},k_\delta}$ is normalized to $[0,1]$[1], then the first claim follows from the Hoeffding bound [44]. We now show the second claim. From the first claim, we get

$$\begin{aligned}
&\Pr\Big[\Big|\phi(G)^T\phi(H) - k_{\mathrm{SP}}^{k_{\mathrm{L}},k_\delta}(G,H)\Big| > \epsilon\Big] \\
&\leq 2\exp\Big(-\log(|\mathcal{G}|^2 \cdot c)\Big) \\
&= \frac{1}{c/2 \cdot |\mathcal{G}|^2}\,.
\end{aligned}$$

The claim then follows from the Union bound. □

[1]It is straightforward to extend the argument to $[a,b]$.

Notice that we can also approximate $k_{\mathrm{SP}}^{k_{\mathrm{L}},k_{\mathrm{D}}}$ by employing a k_{D}-independent hash family.

Weisfeiler-Lehman subtree kernel

By the same arguments, we can derive a similar result for the Weisfeiler-Lehman subtree kernel. To this end, the following Proposition derives an implicit version of the Weisfeiler-Lehman subtree kernel.

Proposition 3.2.8. Let

$$k_{\text{Imp-WL}}^{h}(G,H) = \sum_{i=0}^{h} \sum_{v\in V(G)} \sum_{w\in V(H)} k_i(v,w).$$

The function k_i is recursively defined as

$$k_i(v,w) = \begin{cases} k_{i-1}(v,w)\cdot f(v,w) & i>0 \text{ and } M_i(v,w)\neq\emptyset, \\ k_\delta(l(v),l(w)) & i=0, \\ 0 & \text{otherwise.} \end{cases}$$

The function f is defined as

$$f(v,w) = |M_i(v,w)|^{-1} \sum_{R\in M_i(v,w)} \prod_{(w,w')\in R} k_{i-1}(w,w'),$$

where $M_i(v,w)$ is the set of bijections $b\colon V(G)\to V(H)$ between $N(v)$ and $N(w)$ such that $k_{i-1}(v',w')>0$ for $b(v')=w'$. Then $k_{\text{Imp-WL}}^{h}$ is equivalent to the Weisfeiler-Lehman subtree kernel.

Proof. Follows from [106, Theorem 8]. □

We show that Algorithm 1, with the (explicit) Weisfeiler-Lehman subtree kernel acting as the discrete base graph kernel, probabilistically approximates the graph kernel $k_{\text{Imp-WL}}^{h,k_{\mathrm{A}}}$, where $k_{\text{Imp-WL}}^{h,k_{\mathrm{A}}}$ is defined by substituting δ in the definition of $k_{\text{Imp-WL}}^{h}(G,H)$ by a kernel $k_{\mathrm{A}}\colon\mathbb{R}^n\times\mathbb{R}^n\to\mathbb{R}$.

Corollary 3.2.9. Let $k_{\mathrm{A}}\colon\mathbb{R}^n\times\mathbb{R}^n\to\mathbb{R}$ be a kernel and let $\mathcal{H}$ be an independent k_{A}-hash family. Assume that in each iteration of Algorithm 1 each continuous vertex label is mapped to a label using a hash function chosen independently and uniformly at random from $\mathcal{H}$. Then Algorithm 1, with the Weisfeiler-Lehman subtree kernel with h iterations acting as the discrete base kernel, approximates $k_{\text{Imp-WL}}^{h,k_{\mathrm{A}}}$ such that

$$\Pr\Big[\Big|\phi(G)^T\phi(H) - k_{\text{Imp-WL}}^{h,k_{\mathrm{A}}}(G,H)\Big| \geq \lambda\Big] \leq 2\exp\Big(-2\lambda^2 I\Big).$$

Table 3.2.: Dataset statistics and properties.

Dataset	Properties					
	Number of graphs	Number of classes	∅ Number of vertices	∅ Number of edges	Vertex labels	Cont. label dim.
ENZYMES	600	6	32.6	62.1	✓	18
FRANKENSTEIN	4337	2	16.9	17.9	✗	780
PROTEINS	1113	2	39.1	72.8	✓	1
SYNTHETICNEW	300	2	100.0	196.3	✗	1
SYNTHIE	400	4	95	172.9	✗	15

Moreover, by setting

$$I = \frac{1}{2\epsilon^2} \log(|\mathcal{G}|^2 \cdot c),$$

for some constant $c > 0$ it holds with any constant probability that

$$\sup_{G,H \in \mathcal{G}} \left| \phi(G)^T \phi(H) - k^{h,k_\mathrm{A}}_{\text{Imp-WL}}(G,H) \right| \leq \epsilon,$$

for $\epsilon > 0$.

Proof. Since $k^h_{\text{Imp-WL}}$ is written as a sum of products, and sums of Dirac kernels, we can again use the property of k-independent hash functions and argue analogously to the proof of Theorem 3.2.7. □

Other discrete base kernels

The above technique can also be adapted to other graph kernels for discrete labels. For kernels that use subgraphs up to a fixed size as features, this appears to be highly promising: The graphlet kernel [105] has originally been proposed for unlabeled graphs, but the same idea can be applied to graphs with discrete labels [113]. However, such approaches are infeasible for graphs with continuous labels, which give rise to subgraphs that cannot be mapped to a discrete feature in a straight-forward manner. To overcome this, subgraph matching kernels have been proposed [60], which consider structure-preserving bijections between subgraphs instead. Subgraph matching kernels have been shown to be able to handle continuous labels adequately, but are orders of magnitude slower [60, 84]. Hence, a hash graph kernel with a graphlet kernel as discrete base kernel could combine the advantages of both. Moreover, it is straightforward to prove a result analogous to Theorem 3.2.7.

3.2.5. Experimental evaluation

Our intention here is to investigate the benefits of hash graph kernels compared to the state-of-the-art. More precisely, we address the following questions:

Q1 How do hash graph kernels compare to state-of-the-art graph kernels for continuously labeled graphs in terms of classification accuracy, and computation time?

Q2 How does the choice of the discrete base kernel influence the classification accuracy?

Q3 How does the number of iterations influence the classification accuracy of hash graph kernels in practice?

Datasets

We used the following datasets to evaluate and compare hash graph kernels.

Enzymes and Proteins contain graphs representing proteins according to the graph model of [8]. Each vertex is annotated with a discrete label and a continuous label containing physical or chemical measurements of dimension 18 and 1, respectively. The datasets are subdivided into six and two classes, respectively. Note that this is the same dataset used in [25], which does not contain all the annotations described and used in [8].

Frankenstein is a synthetic dataset from [91]. It is a modified version of the Bursi dataset, cf. [52], which contains 4337 graphs modeling chemical compounds. In Frankenstein the original discrete vertex labels are replaced by 780-dimensional Mnist vectors modeling pixel intensities. The dataset is subdivided into two classes.

SyntheticNew is a synthetic dataset taken from [25, Erratum]. Each of the 300 graphs are derived from a random graph with 100 vertices, 196 edges, and normally distributed one dimensional continuous vertex labels, by rewiring edges, permuting continuous vertex labels, and adding noise to continuous vertex labels. The dataset is subdivided into two classes.

Synthie is a synthetic datasets consisting of 400 graphs. The dataset is subdivided into four classes. Each vertex has a real-valued vector of dimension 15 and no discrete labels. We used the following procedure to generate the dataset: First, we generated two Erdős-Rényi graphs

using the $G(n, p)$ model with $p = 0.2$ and $n = 10$. For each graph, we generated a seed set $\mathcal{S}_i$ for i in $\{1, 2\}$ of 200 graphs by randomly adding or deleting 25% of the edges. From these seed sets, we generated two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ of 200 graphs each by randomly sampling 10 graphs from $\mathcal{S}_1 \cup \mathcal{S}_2$ and randomly connecting these graphs. For $\mathcal{C}_1$ we choose a seed graph with probability 0.8 from $\mathcal{S}_1$ and with probability 0.2 from $\mathcal{S}_2$. The class $\mathcal{C}_2$ was generated the same way but with interchanged probabilities. Finally, we generated a set of real-valued vectors of dimension 15 subdivided into two classes $\mathcal{A}$ and $\mathcal{B}$ using the `make_classification` method from *Scikit-learn* [93], which implements the method described in [38]. We then subdivided $\mathcal{C}_i$ into two classes $\mathcal{C}_i^A$ and $\mathcal{C}_i^B$ by drawing a random continuous label from $\mathcal{A}$ or $\mathcal{B}$ for each vertex. For class $\mathcal{C}_i^A$, we drew a continuous label from $\mathcal{A}$ if the vertex belonged to a seed graph of seed set $\mathcal{S}_1$, and from $\mathcal{B}$ otherwise. Class $\mathcal{C}_i^B$ was created the same way but with interchanged seed sets.

See Table 3.2 for an overview of dataset statistics and properties.[2]

Graph kernels

We implemented hash graph kernels with the explicit shortest-path graph kernel (HGK-SP), and the Weisfeiler-Lehman subtree kernel (HGK-WL) acting as discrete base kernels in Python.[3] For the Weisfeiler-Lehman subtree kernel (WL) we used the fast linear algebra-based algorithm described in [54]. We compared our kernels to the GraphHopper kernel (GH) [25], an instance of the graph invariant kernels (GI) [91], and the propagation kernel from [84] which supports continuous labels (P2K). Additionally, we compared our kernel to the Weisfeiler-Lehman subtree graph kernel, and the explicit shortest-path graph kernel (SP), which only take discrete label information into account, to exemplify the usefulness of using continuous labels. Since the FRANKENSTEIN, SYNTHETICNEW, and SYNTHIE dataset do not have discrete labels, we used vertex degrees as labels instead.

For the graph invariant kernel, we used the original Python implementation provided by the authors of [91]. The variants of the hash graph kernel are computed on a single core only. For the GraphHopper and P2K kernel we used the original Matlab implementation provided by the authors of [25] and [84], respectively.

[2] All dataset can downloaded from `http://graphkernels.cs.tu-dortmund.de`.

[3] The source code can be obtained from `https://github.com/chrsmrrs/hashgraphkernel`.

Table 3.3.: Classification accuracies in percent and standard deviations (Number of iterations for HGK-WL and HGK-SP: 20 (100 for SYNTHIE), *— Kernel uses discrete labels only), OOM— Out of Memory.

Kernel	Dataset				
	ENZYMES	FRANKENSTEIN	PROTEINS	SYNTHETICNEW	SYNTHIE
WL*	54.0 ±1.3	73.5 ±0.3	75.0 ±0.6	**98.6** ±0.3	53.6 ±0.8
SP*	42.9 ±1.0	69.5 ±0.4	75.7 ±0.3	83.3 ±1.4	53.8 ±0.6
HGK-SP	**71.3** ±0.9	70.1 ±0.3	**77.5** ±0.4	96.5 ±0.6	94.3 ±0.5
HGK-WL	67.6 ±1.0	73.6 ±0.4	76.7 ±0.4	**98.8** ±0.3	**96.8** ±0.5
GH	68.8 ±1.0	68.5 ±0.3	72.3 ±0.3	85.1 ±1.0	73.2 ±0.8
GI	**71.7** ±0.8	**76.3** ±0.3	76.9 ±0.5	83.1 ±1.1	95.8 ±0.5
P2K	69.2 ±0.7	OOM	73.5 ±0.5	91.7 ±0.9	50.2 ±1.9

Experimental protocol

For each kernel, we computed the normalized gram matrix. For explicit kernels, we computed the gram matrix via the linear kernel. Note that the computation times of the hash graph kernels could be further reduced by employing linear kernel methods.

We computed the classification accuracies using the *C-SVM* implementation of *LIBSVM* [14], using 10-fold cross validation. The C-parameter was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ by 10-fold cross validation on the training folds. We repeated each 10-fold cross validation ten times with different random folds and report average accuracies and standard deviations. Since the hash graph kernels and P2K are randomized algorithms we computed each gram matrix ten times and report average classification accuracies and computation times. We report computation times for WL, HGK-WL, and P2K with the number of iterations set to 5.

We fixed the number of iterations of the hash graph kernels for all but the SYNTHIE dataset to 20, since this was sufficient to obtain state-of-the-art classification accuracies. For the SYNTHIE dataset, we set the number of iterations to 100, which indicates that this dataset is harder to classify. The number of iterations for the WL, HGK-WL, and P2K was trained on the training folds only using 10-fold cross validation ($\{0, \ldots, 4\}$). For the hash graph kernels, we centered the continuous labels dimension-wise to the mean, scaled to unit variance, and used 2-stable LSH as hash functions to hash continuous labels to discrete labels. For the ENZYMES, the PROTEINS, SYNTHETICNEW, and SYNTHIE dataset we set the interval length to 1. Due

Table 3.4.: Computations times in seconds (Number of iterations for HGK-WL and HGK-SP: 20 (100 for SYNTHIE), number of iterations of WL, HGK-WL, and PK: 5, *— Kernel uses discrete labels only, †— Matlab code, ‡— Code is executed in parallel using eight processes), OOM— Out of Memory.

Kernel	Dataset				
	ENZYMES	FRANKENSTEIN	PROTEINS	SYNTHETICNEW	SYNTHIE
WL*	1.3	25.1	4.1	0.7	1.0
SP*	1.5	22.9	5.9	3.3	3.7
HGK-SP	43.3	**197.8**	107.1	80.6	714.4
HGK-WL	**32.1**	497.7	**82.4**	22.5	168.0
GH†	365.8	16 329.0	3 396.2	275.3	348.2
GI‡	1 748.8	26 717.3	7 905.2	3 814.5	5 523.0
P2K†	43.8	OOM	208.1	**15.1**	**45.3**

to the high dimensional sparse continuous labels of the FRANKENSTEIN dataset we set the interval length to 100. For the HGK-WL we propagated (discrete) labels and hashed continuous labels separately. To speed up the computation, we used the same LSH hash function for all continuous labels in one iteration.

For the graph invariant kernel, we used the LWL_V variant, which has been reported to perform overall well on all datasets [91]. The implementation is using parallelization to speed up the computation and we set the number of parallel processes to eight. For GH and GI we used the Gaussian RBF kernel to compare continuous vertex labels. For all datasets except FRANKENSTEIN, we set the parameter γ of the RBF kernel to $\sqrt{D/2}$ where D is the dimension of the continuous vertex labels, cf. [25, 91]. For FRANKENSTEIN, we set $\gamma = 0.0073$ [91, 104]. Moreover, to study the influence of the number of iterations of the hash graph kernels, we computed classification accuracies and computation times of hash kernels with 1 to 50 iterations on the ENZYMES dataset. Computation times were averaged over ten independent runs. All experiments were conducted on a workstation with an Intel Core i7-3770@3.40GHz, 16 GB of RAM, and Ubuntu 14.04 LTS with Python 2.7.6 and Matlab R2015b.

Results and discussion

In the following, we answer questions Q1–Q3.

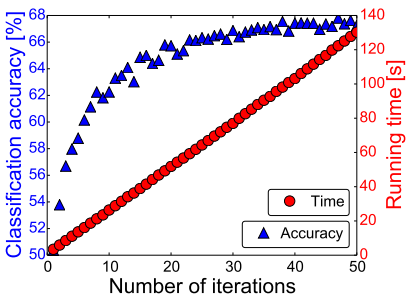
A1 The classification accuracies and the computation times are depicted in Table 3.3 and Table 3.4, respectively. In terms of classification accuracies HGK-WL achieves state-of-the-art results on the SyntheticNew and the Synthie dataset. Notice that the WL kernel, without using continuous label information, achieves (almost) the same classification accuracy on SyntheticNew. This indicates that on this dataset the continuous labels are only of marginal relevance for classification. A different result is observed for the other datasets. On the Synthie dataset HGK-WL achieves the overall best accuracy and is more than 20% better than GH and 40% better than P2K. The kernel almost achieves state-of-the art classification accuracy on the Frankenstein dataset. Notice that the γ parameter of the RBF kernel used in GI and GH was finely tuned. We believe by further optimizing the parameters of 2-stable LSH, we could also achieve state-of-the-art classification accuracy on this dataset.

HGK-SP achieves state-of-the-art classification accuracy on the Enzymes and Proteins dataset and compares favorably on the SyntheticNew and the Synthie dataset. On the Frankenstein dataset, we observe better classification accuracy than GH.

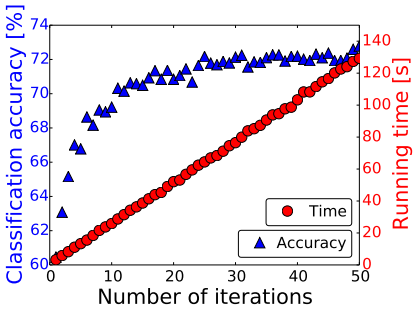
In terms of computation times, both instances of the hash graph kernel framework perform very well. On all datasets, HGK-WL obtains computation times that are several orders of magnitude faster than GH and GI. A similar result can be observed for the HGK-SP (except for Synthie). Naturally, WL and SP have the lowest computation times. However, on three out of five datasets the classification accuracies are severely lower than HGK-SP and HGK-WL. This indicates that taking continuous information into account is indeed useful.

A2 As Table 3.3 shows, the choice of the discrete base kernel has major influence on the classification accuracy for some datasets. On the Enzymes and the Proteins datasets HGK-SP performs favorably, while HGK-WL achieves higher classification accuracies on Frankenstein, SyntheticNew, and Synthie.

A3 Figure 3.3 illustrates the influence of the number iterations on HGK-SP and HGK-WL on the Enzymes dataset. Both plots show that a higher number of iterations leads to better classification accuracies while the computation time grows linearly. In the case of the HGK-SP, the classification accuracy on the Enzymes dataset improves by more than 12% when using 20 instead of 1 iteration. The improvement



(a) Weisfeiler-Lehman subtree.



(b) Shortest-path.

Figure 3.3.: Influence of the number of iterations on the classification accuracy for HGK-SP and HGK-WL on the ENZYMES dataset.

on the ENZYMES dataset is even more substantial for HGK-SP: the classification accuracy improves by more than 16%. At about 30 and 40 iterations for the HGK-SP and HGK-WL, respectively, the algorithms reach a point of saturation. Hence, the empirical results align with your theoretical findings.

3.2.6. Conclusion

We have introduced the hash graph kernel framework which allows applying the various existing scalable and well-engineered kernels for graphs with discrete labels to graphs with continuous vertex labels. The derived kernels outperform other kernels tailored to graphs with continuous labels in terms of computation time without sacrificing classification accuracy. To validate our empirical findings, we showed that the hash graph kernel framework approximates implicit variants of the shortest-path and the Weisfeiler-Lehman subtree kernel with an arbitrarily small error. In the course of our theoretical analysis, we have introduced the concept of an independent k-hash family, which allows approximating the discrete base graph kernel where k is used to compare continuous labels. Studying such families of hash functions in more detail could eventually improve the efficiency and generality of our approach further. We believe that hash graph kernels will also be effective with other discrete base graph kernels like the graphlet kernel.

3.3. Expressive graph kernels based on the Weisfeiler-Leman algorithm

Most state-of-the-art graph kernels only take *local* graph properties into account, i.e., they are computed based on properties of the neighborhood of vertices or other small substructures, e.g., see [105, 17, 106, 91] and Section 3.1. A prominent instance is the 1-WL and the corresponding kernel, see Section 3.1.1. Although it is already quite powerful [6], and has been applied in other areas [37, 54, 69, 123, 127], it is easy to see that the algorithm is not able to distinguish all non-isomorphic graphs, e.g., see [13]. For example, it is not able to distinguish cycles of different lengths, which is an important feature in social network analysis and cheminformatics. Intuitively, the 1-WL takes only local graph properties into account when performed for a fixed number of iterations. The k-WL does take more global properties into account but does not consider local properties. To address this, we propose a graph kernel based on the δ-k-WL, see Section 2.6. To take local properties into account and prevent overfitting, we propose a *local* variant of the above algorithm. Moreover, we show that a variant of the local algorithm converges to the (global) δ-k-WL, which by Proposition 2.6.1 has at least the power as the k-WL. Hence, our new algorithm also takes global properties into account. Since the running times of all three algorithms are in $\Omega(n^k)$ for fixed k, where n denotes the number of vertices of the graph, our local kernel does not scale to large graph databases. Hence, we propose an algorithm to approximate it. For bounded-degree graphs, we show that the running time of the approximation algorithm is *constant*, i.e., it does not depend on the number of vertices or edges of a graph.

3.3.1. The local k-dimensional Weisfeiler-Leman algorithm

In this section, we define the new *local δ-k-dimensional Weisfeiler-Leman algorithm* (δ-k-LWL), which is a variant of the δ-k-WL considering only local neighbors. That is, instead of Equation (2.4), it uses

$$M_i^{\delta}(\mathbf{v}) = \Big(\{\!\{C_i^{k,\delta}(\phi_1(\mathbf{v}, w)) \mid w \in \delta(v_1)\}\!\}, \ldots, \{\!\{C_i^{k,\delta}(\phi_k(\mathbf{v}, w)) \mid w \in \delta(v_k)\}\!\}\Big). \tag{3.5}$$

Hence, the labeling function is defined by

$$C_{i+1}^{k,\delta}(\mathbf{v}) = (C_i^{k,\delta}(\mathbf{v}), M_i^{\delta}(\mathbf{v})). \tag{3.6}$$

Therefore, the algorithm only considers the local j-neighbors of the vertex $\mathbf{v}$ in each iteration. In following, we derive a slight variation of the above algorithm, and show that this variant, named δ-k-LWL$^+$, has at least the power as the δ-k-WL. That is, instead of Equation (3.6), it uses

$$C^{k,\delta,+}_{i+1}(\mathbf{v}) = (C^{k,\delta,+}_i(\mathbf{v}), M^{\delta,+}_i(\mathbf{v})). \tag{3.7}$$

Here

$$\begin{aligned} M^{\delta,+}_i(\mathbf{v}) = \Big(&\{\!\{(C^{k,\delta}_i(\phi_1(\mathbf{v},w)), \#^1_i(\phi_1(\mathbf{v},w))) \mid w \in \delta(v_1)\}\!\}, \ldots, \\ &\{\!\{(C^{k,\delta}_i(\phi_k(\mathbf{v},w)), \#^k_i(\phi_k(\mathbf{v},w))) \mid w \in \delta(v_k)\}\!\}\Big), \end{aligned} \tag{3.8}$$

where

$$\begin{aligned} \#^j_i(\mathbf{v}) = \Big|\{\mathbf{w} \mid \mathbf{w} \in V(G)^k, C^{k,\delta,+}_i(\mathbf{w}) = C^{k,\delta,+}_i(\mathbf{v}), \\ \text{and } \exists e \in V(G) \text{ such that } \phi_j(\mathbf{v},e) = \mathbf{w}\}\Big| \end{aligned} \tag{3.9}$$

for j in $[k]$. That is, $\#^j_i(\mathbf{v})$ counts how often the color of the tuple $\mathbf{v}$ from iteration i occurs as a j-neighbor of $\mathbf{v}$. Since a tuple $\mathbf{w}$ is a j-neighbor of $\mathbf{v}$, anytime $\mathbf{v}$ is a j-neighbor of tuple $\mathbf{t}$ the tuple $\mathbf{w}$ is also one. Note that after the stable partition has been reached $\#^j_i(\mathbf{v})$ will not change anymore. In the next subsection, we prove the following theorem.

Theorem 3.3.1. For all connected graphs, the following holds:

$$\delta\text{-}k\text{-LWL}^+ \sqsubseteq \delta\text{-}k\text{-WL}.$$

Moreover, using Proposition 2.6.1, it immediately follows that the δ-k-LWL$^+$ has at least the same power as the k-WL.

Corollary 3.3.2. For all connected graphs the following holds:

$$\delta\text{-}k\text{-LWL}^+ \sqsubseteq k\text{-WL}.$$

3.3.2. Proof of Theorem 3.3.1

The idea of the proof is to show that both algorithms, the local and the global one, can be "simulated" on directed, labeled trees by the 1-WL by recursively unrolling the local or global neighborhood of each k-tuple. We then show that two such local trees are isomorphic *if and only if* the corresponding global trees are isomorphic. Since the 1-WL computes the isomorphism type

for trees, the result follows. Observe that the local trees might need to be unrolled further than the global trees to catch all "combinatorial patterns".

To formalize the above idea, we need to introduce some terminology. We introduce the *k-tuple graph* and the *unrolling* of the neighborhood around a vertex. Together, these two definitions enable us to reduce the equivalence of both algorithms to a tree isomorphism problem. The k-tuple graph essentially contains the set of all k-tuples as vertices. Two such vertices are joined by an edge if the associated k-tuples are neighbors. The formal definition of a k-tuple graph is as follows.

Definition 3.3.3. Let G be a graph, and let $\mathbf{s}$ and $\mathbf{t}$ be tuples in $V(G)^k$, then the directed, labeled *k-tuple graph* $T^k(G) = (V_T, E_T, l_T)$, where $V_T = \{v_t \mid \mathbf{t} \in V(G)^k\}$, and

$$(v_s, v_t) \in E_T \iff \mathbf{t} = \phi_j(\mathbf{s}, w), \tag{3.10}$$

for j in $[k]$ and some w in $V(G)$. Let $l_T((v_s, v_t)) = (j, \mathrm{L})$ if $\mathbf{t}$ is a local j-neighbor of $\mathbf{s}$ and $l_T((v_s, v_t)) = (j, \mathrm{G})$, otherwise, and let $l_T(v_s) = \tau_{G[\mathbf{s}]}$. Analogously, we define the *local k-tuple graph* $T^k_L(G)$ that uses

$$(v_s, v_t) \in E_T \iff \mathbf{t} = \phi_j(\mathbf{s}, w) \text{ for } w \in \delta(v_j),$$

instead of Equation (3.10).

The following lemma states that the δ-k-WL can be simulated on the k-tuple graph using a variant of the 1-WL.

Lemma 3.3.4. Let G be a graph and let $\mathbf{s}$ and $\mathbf{t}$ be k-tuples in $V(G)^k$, then there exists a variant of the 1-WL with coloring $C_i^{1,*} \colon V(G)^k \to \mathbb{S}$ such that

$$C_i^{k,\delta,\overline{\delta}}(\mathbf{s}) = C_i^{k,\delta,\overline{\delta}}(\mathbf{t}) \iff C_i^{1,*}(v_s) = C_i^{1,*}(v_t),$$

for all $i \geq 0$. The same result holds for $C_i^{k,\delta}$ and $C_i^{k,\delta,+}$ (with regard to the local k-tuple graph).

Proof. We show the results by induction on the number of iterations. For $i = 0$, the result follows from the definition of the label function l_T. Now assume the result holds for some $i > 0$. Let $\mathbf{t}$ be a tuple in $V(G)^k$ and v_t the corresponding vertex in the k-tuple graph. We show how to construct $M_i^{\delta,\overline{\delta}}(\mathbf{t})$ from the labels of the former. The other direction follows by the same means. By assumption, we get

$$\begin{aligned} M_i^{\delta,\overline{\delta}}(\mathbf{t}) = \Big(&\{\!\{(C_i^{1,*}(v_{\phi_1(\mathbf{t},w)}), 1_\delta((v_t, v_{\phi_1(\mathbf{t},w)}))) \mid w \in V(G)\}\!\}, \dots, \\ &\{\!\{(C_i^{1,*}(v_{\phi_k(\mathbf{t},w)}), 1_\delta((v_t, v_{\phi_k(\mathbf{t},w)}))) \mid w \in V(G)\}\!\}\Big), \end{aligned}$$

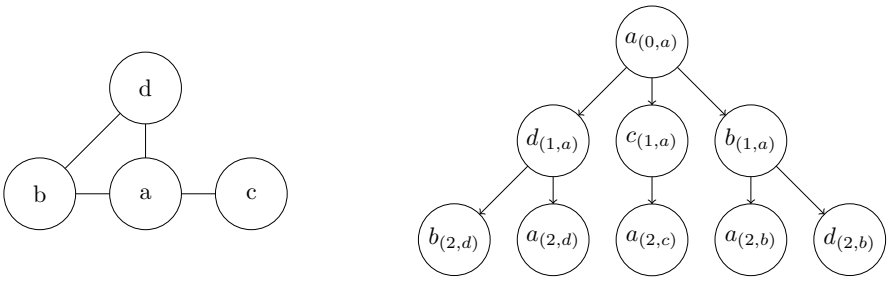


Figure 3.4.: Illustration of the unrolling operation around the vertex a for $i = 2$.

Hence, can define the coloring $C^{k,\delta,\overline{\delta}}_{i+1}(\mathbf{t})$. The coloring of the needed variant of the 1-WL is defined as

$$C^{1,*}_{i+1}(v_t) = (C^{1,*}_i(v_t), M^1_i(v_t)), \tag{3.11}$$

where the vector $M^1_i(v_t)$ is equal to $M^{\delta,\overline{\delta}}_i(\mathbf{t})$ $\square$.

The *unrolling* of a neighborhood around a vertex of a given graph to a tree is defined as follows, see Figure 3.4 for an illustration.

Definition 3.3.5. Let $G = (V, E, l)$ be a labeled (directed) graph and let v be in V. Then $U^i_{G,v} = (W_i, F_i, l_i)$ for $i \geq 0$ denotes the *unrolled tree* G around v at depth i, where

$$W_i = \begin{cases} \{v_{(0,v)}\} & \text{if } i = 0 \\ W_{i-1} \cup \{u_{(i,w_{(i-1,p)})} \mid u \in \delta(w) \text{ for } w_{(i-1,p)} \in W_{i-1}\} & \text{otherwise,} \end{cases}$$

and

$$F_i = \begin{cases} \emptyset & \text{if } i = 0 \\ F_{i-1} \cup \{(w_{(i-1,p)}, u_{(i,w)}) \mid u \in \delta(w) \text{ for } w_{(i-1,p)} \in W_{i-1}\} & \text{otherwise.} \end{cases}$$

The label function is defined as $l_i(u_{(j,p)}) = l(u)$ for u in V, and $l_i(u_{(j,w)}) = l((w, u))$. For notational convenience, we usually omit the subscript i.

In the following, we use the unrolled tree for the above defined (local) k-tuple graph. For $k \geq 2$, we denote the *directed*, unrolled tree of the k-tuple graph of G around the vertex v_s at depth i for the tuple s in $V(G)^k$ by $U^i_{T^k(G),v_s}$. For notational convenience, we write $\mathbf{U}^i_{T,v}$, the analogous local tree is denoted by $\mathbf{U}^i_{T,v,\mathrm{L}}$. We write

$$\mathbf{U}^i_{T,v} \simeq_{v \to w} \mathbf{U}^i_{T,w} \tag{3.12}$$

if there exists an isomorphism φ between the two unrolled trees that also respects the label functions l_i. When considering a local tree of height i, the vertices at depth $d \leq i$ of the unrolled tree are additionally labeled with $\#_d^j$ for j in $[k]$, and we write $\mathbf{U}^i_{T,v,L} \simeq_{v \to w} \mathbf{U}^i_{T,w,L}$ if the isomorphism φ between the two unrolled trees also respects the label functions l_i and $\#_d^j$ for $d \leq i$. Moreover, we need the following two results. The first one states that the 1-WL can distinguish any two directed, labeled non-isomorphic trees.

Theorem 3.3.6 ([11, 110])**.** The 1-WL distinguishes any two directed, labeled *non-isomorphic* trees.

Using the first result, the second one states that the (local) δ-k-WL can be simulated by the 1-WL on the unrolled tree of the k-tuple graph, and hence can be reduced to a tree isomorphism problem.

Lemma 3.3.7. Let G be a *connected* graph then there exists an $h \geq 0$ such that the coloring of the stable partition of the δ-k-WL colors $\mathbf{s}$ and $\mathbf{t}$ in $V(G)^k$ the same if and only if the corresponding unrolled k-tuple trees of height h are isomorphic, i.e.,

$$C^{k,\delta,\overline{\delta}}_{\infty}(\mathbf{s}) = C^{k,\delta,\overline{\delta}}_{\infty}(\mathbf{t}) \iff \mathbf{U}^h_{T,v_s} \simeq_{v_s \to v_t} \mathbf{U}^h_{T,v_t},$$

for all i in $\mathbb{N}_0$. The same holds for the δ-k-LWL$^+$, i.e., there exsits a $h \geq 0$ such that

$$C^{k,\delta,+}_{\infty}(\mathbf{s}) = C^{k,\delta,+}_{\infty}(\mathbf{t}) \iff \mathbf{U}^h_{T,v_s,\mathrm{L}} \simeq_{v_s \to v_t} \mathbf{U}^h_{T,v_t,\mathrm{L}},$$

for all i in $\mathbb{N}_0$.

Proof. First, by Lemma 3.3.4, we can simulate the (local) δ-k-WL$^+$ for the graph G in the k-tuple graph $T^k(G)$ by the 1-WL. Secondly, consider a vertex v_s in the k-tuple graph $T^k(G)$ and a corresponding vertex in the unrolled tree around v_s. Observe that the neighborhoods for both vertices are identical. By definition, this holds for all vertices (excluding the leaves) in the unrolled tree. Hence, by Lemma 3.3.4, we can simulate the (local) δ-k-WL$^+$ for each tuple $\mathbf{t}$ by running the 1-WL in the unrolled tree around v_t in the k-tuple graph. Since the 1-WL solves the isomorphism problem for trees, cf. Theorem 3.3.6, the result follows. □

Given the h from the previous result, we write $\mathbf{U}^{\infty}_{T,v}$ instead of $\mathbf{U}^h_{T,v}$, for local trees we write $\mathbf{U}^{\infty}_{T,v,L}$. That is, the trees $\mathbf{U}^{\infty}_{T,v}$ and $\mathbf{U}^{\infty}_{T,v,L}$ contain all information to compute the stable color of the tuples corresponding to the roots of the trees.

We can now prove the essential Lemma for the proof of Theorem 3.3.1. It states that the trees of the unrolled neighborhoods of two vertices in the global k-tuples graphs are isomorphic if the same holds for the corresponding local trees (although the local trees might need to be unrolled further).

Lemma 3.3.8. Let G be a *connected* graph. Let G be a graph, and let v and w be roots of unrolled trees with regard to the k-tuple graph $T^k(G)$. Then there exists an h such that

$$\mathbf{U}^h_{T,v,\mathrm{L}} \simeq_{v \to w} \mathbf{U}^h_{T,w,\mathrm{L}} \Longrightarrow \mathbf{U}^\infty_{T,v} \simeq_{v \to w} \mathbf{U}^\infty_{T,w}.$$

Proof. Assume that there is a bijection φ_{L} between the two trees $\mathbf{U}^h_{T,v,\mathrm{L}}$ and $\mathbf{U}^h_{T,w,\mathrm{L}}$ that is a (labeled) tree isomorphism that respects the labeling functions l_h and $\#^j_h$ for j in $[k]$. The exact choice of h will be determined later. We now extend this isomorphism to the mapping φ_{G} between the global trees and argue that it is again a tree isomorphism, namely an isomorphism between $\mathbf{U}^\infty_{T,v}$ and $\mathbf{U}^\infty_{T,w}$ that respects the labeling functions l_∞. We proceed in an iterative fashion, i.e., we extend φ_{L} iteratively for each depth of the two trees.

Let s be a vertex from $\mathbf{U}^h_{T,v,\mathrm{L}}$ and let t in $N(s)$ be a *global* j-neighbor of s for which we like to define the mapping φ_{G}. Let τ denote the isomorphism type of $\mathbf{U}^{h-1}_{T,t,\mathrm{L}}$. We search the local tree, starting at the vertex s, for all trees of height $(h-1)$ of type τ reachable on a path, where each edge has a label of the form (j, L). Since the graph is connected, there must exist such a path. Observe that the number of such trees is finite when we only consider trees where the root nodes represent different k-tuples, and there must exist at least one, e.g., the tree of type τ whose root represents the same tuple as t.

Moreover, observe that these trees are also j-neighbors of s. Since, by assumption, the local trees are isomorphic, we can discover identical trees (up to isomorphism) in the other unrolled (local) tree (rooted at $\varphi(s)$). Indeed, the number of trees of type τ are the same in both local trees. Moreover, since we respect the labeling $\#^j_h$ the number of trees of type τ whose roots represents different tuples is also the same. Specifically, the number of local neighbors of s and $\varphi(s)$, respectively, that induce trees of type τ are also the same. Consequently, the same holds for global neighbors. Hence, we can extend the (local) isomorphism to an isomorphism between the global neighbors.

By applying the above procedure in a top-down fashion, we eventually get the desired isomorphism for the global trees. We now set l to some large finite value such that all trees of type τ are found. □

Practicality

As Theorem 3.3.1 shows, the δ-k-WL and the δ-k-LWL$^+$ have the same power in terms of distinguishing non-isomorphic graphs. Although for complete graphs the local algorithm will have the same running time, for sparse graphs the running time for each iteration can be upper-bounded by $|V(G)^k| \cdot kd$, where d denotes the maximum or average degree of the graph. Hence, the local algorithm takes the sparsity of the underlying graph into account, resulting in (iteration-wise) improved computation times compared to the non-local δ-k-WL and the k-WL.

3.3.3. A kernel based on the δ-k-LWL

The idea for a kernel based on the δ-k-LWL (δ-k-LWL$^+$) is the same as for the 1-WL. We compute the δ-k-LWL for $h \geq 0$ iterations, and after each iteration i compute a feature vector $\phi^i_{k\text{-LWL}}(G)$ in $\mathbb{R}^{|\mathbb{S}_i|}$ for each graph G, where $\mathbb{S}_i \subseteq \mathbb{S}$ denotes the image of $C^{k,\delta}_i$. Each component $\phi^i_{k\text{-LWL}}(G)_s$ counts the number of occurrences of k-tuples labeled with s in $\mathbb{S}_i$. The overall feature vector $\phi_{k\text{-LWL}}(G)$ is defined as the concatenation of the feature vectors of all h iterations, i.e.,

$$\left[\phi^0_{k\text{-LWL}}(G), \ldots, \phi^h_{k\text{-LWL}}(G)\right].$$

Let G and H be two graphs then the kernel $k_{k\text{-LWL}} = \langle \phi_{k\text{-LWL}}(G), \phi_{k\text{-LWL}}(H)\rangle$ and $k^h_{k\text{-LWL}} = \langle \phi^h_{k\text{-LWL}}(G), \phi^h_{k\text{-LWL}}(H)\rangle$. Moreover, we will need the *normalized feature vector*

$$\widehat{\phi}^h_{k\text{-LWL}}(G) = \phi^h_{k\text{-LWL}}(G)/\|\phi^h_{k\text{-LWL}}(G)\|_1. \tag{3.13}$$

Observe that the components of the normalized feature vector are in $[0, 1]$. We denote the corresponding kernel by $\widehat{k}^h_{k\text{-LWL}}$.

3.3.4. An approximation algorithm for the δ-k-LWL for bounded-degree graphs

Since the worst-case running time of the computation of $\phi^h_{k\text{-LWL}}(G)$ can be lower bounded by $|V(G)^k|$ for fixed k, computing the corresponding kernel is not feasible for large graphs. Thereto, we describe an approximation algorithm for the δ-k-LWL (δ-k-LWL$^+$) kernel. For bounded-degree graphs we show that it can be computed in *constant time*. One key ingredient for this algorithm is the following definition.

Definition 3.3.9. Let G be a graph, and let $\mathbf{t}$ be a k-tuple from $V(G)^k$, then the *c-neighborhood* of $\mathbf{t}$ for $c \geq 0$,

$$\mathbf{A}(\mathbf{t}, c) = \{\mathbf{s} \in V(G)^k \mid d(v_t, v_s) \leq c\},$$

where $d\colon V_T \times V_T \to \mathbb{N}$ denotes the shortest-path distance in the local k-tuple graph $T_{\mathrm{L}}^k(G)$ of G.

Algorithm 2 Approximation algorithm for the δ-k-LWL for bounded-degree graphs

---

Require: A (d-bounded degree) graph G, number of iterations $h \geq 0$, $k \geq 2$, failure parameter δ in $(0, 1)$, and an additive error term ε in $(0, 1]$.
Ensure: A feature map $\widetilde{\phi}^h_{k\text{-LWL}}(G)$ according to Inequality (3.15).

Let $\widetilde{\phi}^h_{k\text{-LWL}}(G)$ be a feature vector
Draw a multiset S of k-tuples independently and uniformly from $V(G)^k$ with $|S|$ according to Equation (3.14)

parallel for $\mathbf{s} \in S$ **do**
    Compute the h-neighborhood $\mathbf{A}(\mathbf{s}, h)$ around s
    Compute $\sigma = l^h_{\mathrm{L},\mathfrak{N}}(\mathbf{s})$ on $G[\mathfrak{N}(\mathbf{s}, h)]$
    $\widetilde{\phi}^h_{k\text{-LWL}}(G)_\sigma = \widetilde{\phi}^h_{k\text{-LWL}}(G)_\sigma + 1/|S|$
end

return $\widetilde{\phi}^h_{k\text{-LWL}}(G)$

The idea of the algorithm is the following: Let G be a graph and let $\widetilde{\phi}^h_{k\text{-LWL}}(G)$ be a zero vector with the same number of components as $\widehat{\phi}^h_{k\text{-LWL}}(G)$. First, we sample a multiset S of k-tuples, where each element is sampled independently and uniformly at random from $V(G)^k$. The exact cardinality of S will be determined later. Secondly, for each such k-tuple $\mathbf{s}$ in S, we compute the h-neighborhood $\mathbf{A}(\mathbf{s}, h)$ and compute the δ-k-LWL for h iterations on the subgraph induced by all vertices in

$$\mathfrak{N}(\mathbf{s}, h) = \{t_1, \ldots, t_k \mid (t_1, \ldots, t_k) \in \mathbf{A}(\mathbf{s}, h)\},$$

resulting in a label σ for the k-tuple $\mathbf{s}$. The following result shows that the label of a k-tuple after h iterations can be computed *locally* by only considering its h-neighborhood.

Lemma 3.3.10. Let G be a graph, and let $\mathbf{s}$ be a k-tuple in $V(G)^k$. Moreover, let $l^h_{\mathrm{L},\mathfrak{N}}(\mathbf{s})$ be the label of $\mathbf{s}$ after the h-th iteration of the δ-k-LWL on $G[\mathfrak{N}(\mathbf{s}, h)]$, then

$$l^h_{\mathrm{L},\mathfrak{N}}(\mathbf{s}) = C^{k,\delta}_h(\mathbf{s}).$$

Proof (Sketch). Induction on the number of iterations. □

Now the algorithm proceeds by adding $1/|S|$ to $\widetilde{\phi}^h_{k\text{-LWL}}(G)_\sigma$. See Algorithm 2 for pseudo code. Note that lines 4 to 6 can be computed in parallel for all samples. Moreover, note that the algorithm can be easily adapted so that it approximates the feature vector over all h iterations. Let G be a d-bounded degree graph, and let $\Gamma(d, h)$ be an upper bound on the maximum number of different labels of the δ-k-LWL after h iterations on G. We get the following result.

Theorem 3.3.11. Let G be a d-bounded degree graph, and let

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d, h) \cdot 1/\delta)}{2(\varepsilon/\Gamma(d,h))^2} \right\rceil. \tag{3.14}$$

Then Algorithm 2 approximates the normalized feature vector $\widehat{\phi}^h_{k\text{-LWL}}(G)$ of the δ-k-LWL such that with probability $(1 - \delta)$ for δ in $(0, 1)$,

$$\left\| \widehat{\phi}^h_{k\text{-LWL}}(G) - \widetilde{\phi}^h_{k\text{-LWL}}(G) \right\|_1 \leq \varepsilon \tag{3.15}$$

for any ε in $(0, 1]$. Moreover, the running time of the algorithm is only dependent of d, k, and h, i.e., it does not depend on $|V(G)|$.

Proof. First, observe that $\Gamma(d, h)$ is only dependent on the number of iterations h, k, and the maximum degree d. Let $X_{i,\sigma}$ denote the random variable that is 1 if we sample a k-tuple $\mathbf{s}$ such that $l^h_{\mathrm{L},\mathfrak{N}}(\mathbf{s}) = \sigma$ in iteration i of Algorithm 2, otherwise 0. Now observe that

$$\mathbb{E}(X_{i,\sigma}) = \widehat{\phi}^h_{k\text{-LWL}}(G)_\sigma.$$

Moreover, let

$$\bar{X}_\sigma = 1/|S| \cdot \sum_{i=1}^{S} X_{i,\sigma} = \widetilde{\phi}^h_{k\text{-LWL}}(G)_\sigma,$$

then, by the linearity of expectation,

$$\mathbb{E}\big(\bar{X}_\sigma\big) = \widehat{\phi}^h_{k\text{-LWL}}(G)_\sigma.$$

Hence, by the Hoeffding bound [44], we get

$$\Pr\Big(\Big|\bar{X}_\sigma - \widehat{\phi}^h_{k\text{-LWL}}(G)_\sigma\Big| \geq \lambda\Big) \leq 2e^{-2|S|\cdot\lambda^2}.$$

By setting the sample size

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta)}{2\lambda^2} \right\rceil, \tag{3.16}$$

it follows that

$$\Pr\Big(\Big|\bar{X}_\sigma - \widehat{\phi}^h_{k\text{-LWL}}(G)_\sigma\Big| \geq \lambda\Big) \leq \frac{\delta}{\Gamma(d,h)}.$$

The result then follows by setting $\lambda = \varepsilon/\Gamma(d,h)$, and the Union bound. Finally the bound on the running time follows from the observation that Equation (3.14), and the running time of lines 4 to 6 in Algorithm 2 are independent of the size of G, i.e., the number of vertices and edges. The correctness follows from Lemma 3.3.10. □

Now let $\widetilde{k}^h_{k\text{-LWL}}(G,H)$ denote the corresponding kernel, i.e.,

$$\widetilde{k}^h_{k\text{-LWL}}(G,H) = \langle \widetilde{\phi}^h_{k\text{-LWL}}(G), \widetilde{\phi}^h_{k\text{-LWL}}(H) \rangle$$

for two graphs G and H. The following proposition shows that the above kernel approximates the normalized δ-k-LWL kernel arbitrarily close.

Proposition 3.3.12. Let $\mathcal{G}$ be (non-empty, finite) set of d-bounded degree graphs, let $\widehat{k}^h_{k\text{-LWL}}$ be the normalized δ-k-LWL kernel, and let

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta \cdot |\mathcal{G}|)}{2(\lambda/\Gamma(d,h))^2} \right\rceil.$$

Then with probability $(1-\delta)$ for δ in $(0,1)$, Algorithm 2 approximates $\widehat{k}^h_{k\text{-LWL}}$ such that

$$\sup_{G,H \in \mathcal{G}} \Big|\widehat{k}^h_{k\text{-LWL}}(G,H) - \widetilde{k}^h_{k\text{-LWL}}(G,H)\Big| \leq 3\lambda$$

for any λ in (0,1]. The running time for computing the gram matrix for $\mathcal{G}$ does only depend on the cardinality of $\mathcal{G}$, the number of iterations h, k, and the maximum degree d.

Proof. First observe that by setting the sample size $|S|$ to

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta \cdot |\mathcal{G}|)}{2\varepsilon^2} \right\rceil,$$

we get that with probability $(1-\delta)$ for all G in $\mathcal{G}$

$$\left|\widehat{\phi}^h_{k\text{-LWL}}(G)_i - \widetilde{\phi}^h_{k\text{-LWL}}(G)_i\right| \leq \varepsilon$$

for any $1 \leq i \leq \Gamma(d,h)$ holds. Let G and H in $\mathbb{G}$, then

$$\begin{aligned}
&\widetilde{k}^h_{k\text{-LWL}}(G,H) = \left\langle \widetilde{\phi}^h_{k\text{-LWL}}(G), \widetilde{\phi}^h_{k\text{-LWL}}(H) \right\rangle \\
&= \sum_{i=1}^{\Gamma(d,h)} \widetilde{\phi}^h_{k\text{-LWL}}(G)_i \cdot \widetilde{\phi}^h_{k\text{-LWL}}(H)_i \\
&\leq \sum_{i=1}^{\Gamma(d,h)} \left(\widehat{\phi}^h_{k\text{-LWL}}(G)_i + \varepsilon\right) \cdot \left(\widehat{\phi}^h_{k\text{-LWL}}(H)_i + \varepsilon\right) \\
&\leq \sum_{i=1}^{\Gamma(d,h)} \left(\widehat{\phi}(G)_i \cdot \widehat{\phi}(H)_i\right) + \varepsilon \cdot \sum_{i=1}^{\Gamma(d,h)} \left(\widehat{\phi}(G)_i + \widehat{\phi}(H)_i\right) \\
&+ \sum_{i=1}^{\Gamma(d,h)} \varepsilon^2 \leq \widehat{k}^h_{k\text{-LWL}}(G,H) + 2\Gamma(d,h) \cdot \varepsilon + \Gamma(d,h) \cdot \varepsilon\,.
\end{aligned}$$

The last inequality follows from the fact that the components of $\widehat{\phi}(\cdot)$ are in $[0,1]$. The result then follows by setting $\varepsilon = \lambda/\Gamma(d,h)$. □

Note that the above technique also leads to an approximation result for the (normalized) Weisfeiler-Lehman subtree kernel.

Corollary 3.3.13. Let $\mathcal{G}$ be (non-empty, finite) set of d-bounded degree graphs, let $\widehat{k}^h_{\text{WL}}$ be the normalized Weisfeiler-Lehman subtree kernel, and let

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta \cdot |\mathcal{G}|)}{2(\lambda/\Gamma(d,h))^2} \right\rceil.$$

Then with probability $(1-\delta)$ for δ in $(0,1)$, Algorithm 2 approximates $\widehat{k}^h_{\text{WL}}$ such that

$$\sup_{G,H \in \mathcal{G}} \left|\widehat{k}^h_{\text{WL}}(G,H) - \widetilde{k}^h_{\text{WL}}(G,H)\right| \leq 3\lambda$$

for any λ in $(0,1]$. The running time for computing the gram matrix for $\mathcal{G}$ does only depend on the size of $\mathcal{G}$, the number of iterations h, and the maximum degree d.

Observe that instead of considering bounded-degree graphs, the above results, i.e., Proposition 3.3.12 and Corollary 3.3.13, also hold for general graphs with a constant number of colors, i.e., the number of colors is independent of the size of the graphs. Moreover, observe that Proposition 3.3.12 and Corollary 3.3.13 can be easily adapted such that we get approximation results for the kernels over all h iterations.

3.3.5. Experimental evaluation

Our intention here is to investigate the benefits of the δ-k-LWL kernel compared to the δ-k-WL and the k-WL kernel. More precisely, we address the following questions:

Q1 How much does the local algorithm speed up the computation times compared to the non-local algorithms?

Q2 Does the local algorithm lead to improved classification accuracies on real-world benchmark datasets?

Q3 Does the local algorithm prevent overfitting to the training set?

Q4 Does the sampling algorithm speed up the computation times for larger datasets while still achieving meaningful accuracy scores?

3.3.6. Datasets and graph kernels

We used the following well-known datasets: ENZYMES, IMDB-BINARY, IMDB-MULTI, NCI1, NCI109, PTC_FM, PROTEINS, and REDDIT-BINARY to evaluate our kernels. To answer Q4, we used the larger datasets REDDIT-BINARY, and REDDIT-MULTI-5K. See Table 3.5 for statistics and properties.[4] See Appendix B for detailed descriptions of the datasets.

We implemented the δ-k-LWL, the δ-k-WL, and the k-WL kernel for k in $\{2, 3\}$. We compared our kernels to the Weisfeiler-Lehman subtree kernel [106], the graphlet kernel [105], and the shortest-path kernel [8]. All kernels were (re-)implemented in C++11.[5]

[4] All datasets can obtained from `http://graphkernels.cs.tu-dortmund.de`.
[5] The source code can be obtained from `https://github.com/chrsmrrs/localwl`.

Table 3.5.: Dataset statistics and properties.

Dataset	Properties				
	Number of graphs	Number of classes	∅ Number of vertices	∅ Number of edges	Vertex labels
ENZYMES	600	6	32.6	62.1	✓
IMDB-BINARY	1 000	2	19.8	96.5	✗
IMDB-MULTI	1 500	3	13.0	65.9	✗
NCI1	4 110	2	29.9	32.3	✓
NCI109	4 127	2	29.7	32.1	✓
PTC_FM	349	2	14.1	14.5	✓
PROTEINS	1 113	2	39.1	72.8	✓
REDDIT-BINARY	2 000	2	429.6	497.8	✗
REDDIT-BINARY-5K	4 999	5	508.5	594.9	✗

Table 3.6.: Overall computation times for the whole datasets in seconds (Number of iterations for 1-WL, 2-WL, 3-WL, δ-2-WL, δ-3-WL, δ-2-LWL, and δ-3-LWL: 3, OOT)— Computation did not finish within one day (24h), OOM— Out of memory.

	Graph Kernel	Dataset							
		ENZYMES	IMDB-BINARY	IMDB-MULTI	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
Baseline	GRAPHLET	<1	<1	<1	<1	<1	<1	<1	2
	SHORTEST-PATH	<1	<1	<1	2	2	<1	<1	1 125
	1-WL	<1	<1	<1	2	2	<1	<1	2
Global	2-WL	296	86	42	1 402	1 410	11	14 237	OOM
	3-WL	74 430	17 552	5 184	OOT	OOT	1117	OOM	OOM
	δ-2-WL	296	88	44	1 465	1 472	11	14 390	OOM
	δ-3-WL	70 736	17 809	3 433	OOT	OOT	1114	OOM	OOM
Local	δ-2-LWL	30	24	19	99	100	< 1	241	57 110
	δ-3-LWL	4 464	3 433	2 070	17 832	17 898	96	OOM	OOM

3.3.7. Experimental protocol

For each kernel, we computed the normalized gram matrix. We computed the classification accuracies using the C-SVM implementation of LIBSVM [14], using 10-fold cross validation. The C-parameter was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ by 10-fold cross validation on the training folds.

We repeated each 10-fold cross validation ten times with different random folds, and report average accuracies and standard deviations. We report running times for the 1-WL, the δ-k-LWL, the δ-k-LWL, and the k-WL with three refinement steps. For the graphlet kernel, we counted (labeled) connected subgraphs of size three. For measuring the classification accuracy, the number of iterations of the 1-WL, δ-k-LWL, the δ-k-LWL, and the k-WL were selected from $\{0, \ldots, 5\}$ using 10-fold cross validation on the training

Table 3.7.: Classification accuracies in percent and standard deviations, OOT— Computation did not finish within one day, OOM— Out of memory.

	Graph Kernel	Dataset							
		ENZYMES	IMDB-BINARY	IMDB-MULTI	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
Baseline	GRAPHLET	41.0 ±1.2	59.4 ±0.4	40.8 ±0.4	72.1 ±0.3	72.3 ±0.2	58.3 ±1.6	72.9 ±0.3	60.1 ±0.2
	SHORTEST-PATH	42.3 ±1.3	59.2 ±0.3	39.6 ±0.4	74.5 ±0.3	73.4 ±0.1	62.1 ±0.9	**76.4** ±0.4	84.7±0.2
	1-WL	53.4 ±1.4	72.4 ±0.5	**50.6** ±0.6	83.1 ±0.2	**85.2** ±0.2	**62.9** ±1.6	73.7 ±0.5	75.3 ±0.3
Global	2-WL	39.4 ±0.8	69.6 ±0.6	48.0 ±0.3	67.3 ±0.3	68.0 ±0.3	57.1 ±1.3	75.2 ±0.4	OOM
	3-WL	46.1 ±1.0	69.7 ±0.8	47.8 ±0.3	OOT	OOT	58.0 ±0.7	OOM	OOM
	δ-2-WL	39.3 ±0.9	69.8 ±0.7	48.0 ±0.5	67.5 ±0.3	68.1 ±0.3	57.1 ±1.4	75.2 ±0.4	OOM
	δ-3-WL	46.1 ±1.0	69.7 ±0.8	47.8 ±0.3	OOT	OOT	58.0 ±0.7	OOM	OOM
Local	δ-2-LWL	57.5 ±0.9	**73.7** ±0.6	**50.8** ±0.4	**85.3** ±0.2	84.6 ±0.3	57.3 ±0.3	74.6 ±0.6	**89.4** ±0.3
	δ-3-LWL	**60.9** ±0.9	**73.6** ±0.5	49.7 ±0.4	84.0 ±0.3	83.0 ±0.3	58.0 ±0.5	OOM	OOM

Table 3.8.: Training versus test accuracy.

	Set	Dataset							
		ENZYMES	IMDB-BINARY	IMDB-MULTI	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
δ-2-WL	Train	100.0	88.7	63.5	99.8	99.9	97.1	98.7	–
	Test	39.3	69.8	48.0	67.5	68.1	57.1	75.2	–
δ-2-LWL	Train	99.6	88.8	63.6	98.2	98.2	97.0	98.2	98.3
	Test	57.5	73.7	50.8	85.3	84.6	57.3	74.6	89.4

folds only.[6] To answer Question 3 we used a single 10-fold cross validation with the hyperparameters found in the former experiment and report average training and test accuracies. To answer Question 4, we set the sample to 50 and 100. We repeated each experiment ten times, and report average accuracies and computation times.

All experiments were conducted on a workstation with an Intel Xeon E5-2690v4 processor with 2.60GHz and 384GB of RAM running Ubuntu 16.04.6 LTS using a single core. Moreover, we used the GNU C++ Compiler 5.5.0 with the flag `-O2`.

3.3.8. Results and discussion

In the following we answer questions **Q1** to **Q4**. See also Tables 3.6 to 3.10.

A1 See also Table 3.6. The local algorithm severely speeds up the computation time compared to the k-WL and the δ-k-WL for $k = 2$ and 3. For example, on the ENZYMES dataset the 2-LWL is almost 10 times faster than 2-WL, the same holds for the 3-LWL. The improvement

[6] As already shown in [106], choosing the number of iterations too large will lead to overfitting.

Table 3.9.: Classification accuracies in percent and standard deviations of the approximation algorithms, OOT— Computation did not finish within one day, OOM— Out of memory.

	Graph Kernel	Dataset		
		PROTEINS	REDDIT-BINARY	REDDIT-MULTI-5K
Base	GRAPHLET	72.9 ±0.3	60.1 ±0.2	23.9 ±0.5
Base	1-WL	73.7±0.5	75.3 ±0.3	48.9 ±0.3
Local	δ-3-LWL	OOM	OOM	OOM
Local	δ-3-LWL-SAMPLE ($S = 50$)	75.1 ±0.4	84.7 ±0.3	48.9 ±0.2
Local	δ-3-LWL-SAMPLE ($S = 100$)	75.1 ±0.3	85.6 ±0.2	51.2 ±0.2

Table 3.10.: Computation times in seconds of the approximation algorithms (Number of iterations: 3), OOM— Out of memory.

	Graph Kernel	Dataset		
		PROTEINS	REDDIT-BINARY	REDDIT-MULTI-5K
Base	GRAPHLET	<1	2	5
Base	1-WL	<1	2	5
Local	δ-3-LWL	OOM	OOM	OOM
Local	δ-3-LWL-SAMPLE ($S = 50$)	241	4700	9 634
Local	δ-3-LWL-SAMPLE ($S = 100$)	518	8 948	21 530

of the computation times can be observed across all datasets. For some datasets, the $\{2, 3\}$-WL and δ-$\{2, 3\}$-WL did not finish within the given time limit or went out of memory. For example, on four out of eight datasets the δ-3-WL is out of time or out of memory, whereas for the corresponding local algorithm this happens only two out of eight times. This indicates that the local algorithm is more suitable for practical applications. Of course, this is not surprising as the local algorithm considers only a subset of the neighbors compared to the global algorithm.

A2 See also Table 3.7. The local algorithm for $k = 2$ and 3 severely improves the classification accuracy compared to the k-WL and the δ-k-WL. For example, on the ENZYMES dataset the δ-2-LWL achieves an improvement of almost 20%, and the δ-3-LWL achieves the best accuracies over all employed kernels, improving over the 3-WL and the δ-3-WL by almost 15%. This observation holds over all datasets (excluding PTC_FM). However, is has to be noted that increasing k does not always result in increased accuracies. For example, on all

datasets (excluding ENZYMES and PTC_FM), the performance of the δ-2-LWL is better or on par with the δ-3-LWL. On NCI109 the 1-WL achieves the overall best accuracy. This indicates that with increasing k the local algorithm is more prone to overfitting.

A3 As Table 3.8 shows the δ-2-WL reaches slightly higher training accuracies over all datasets compared to the δ-2-LWL, while the testing accuracies are much lower, excluding PROTEINS. This indicates that the δ-2-WL overfits on the training set. The higher test accuracies of the local algorithm are likely due to the smaller neighborhood which promotes that the number of colors grow slower compared to the global algorithm. Hence, the smaller neighborhood of the local algorithms acts as a graph-based regularization.

A4 As can be seen in Tables 3.9 and 3.10, the sampling algorithm severely speeds up the computation time and reduces memory consumption, while still achieving higher accuracies than the baselines. This aligns with the theoretical results.

3.3.9. Conclusion

We proposed a local variant of the (global) δ-k-WL, which takes the sparsity of the underlying graph into account. Moreover, we showed that a variant of the local algorithm has the same power as the δ-k-WL. We demonstrated that the local algorithm can be computed approximately in constant time for bounded-degree graphs. Finally, we employed the local variant as a kernel, showed that it prevents overfitting, and leads to improved classification accuracies compared to the corresponding global algorithms and baseline approaches. Not surprisingly, we found out that increasing k may not lead to improvements over all datasets.

We believe that the introduction of our local algorithm provides directions for future work. For example, it would be interesting to investigate if considering only k-tuples that induce connected subgraphs result in a less expressive algorithm (for small k). Furthermore, it would be interesting to better understand why Weisfeiler-Leman type algorithms work well for supervised graph classification, e.g., characterizing for which real-world graphs they compute meaningful features.

3.4. A theoretical framework for the expressiveness of graph kernels

In the past two decades a large number of graph kernels have been proposed, see, e.g., [112, 84, 63] and references therein, and Section 3.1. As already seen in Section 3.1, most graph kernels decompose graphs and add up the pairwise similarities between their substructures following the seminal concept of convolution kernels [41]. Considering the large number of available graph kernels and the wealth of available benchmark datasets [53], it becomes increasingly difficult to perform a fair experimental comparison of kernels and to assess their advantages and disadvantages for specific datasets. Indeed, current experimental comparisons cannot give a complete picture, and are of limited help to a practitioner who has to choose a kernel for a particular application.

Graph kernels are developed with the (possibly conflicting) goals of being efficiently computable and capturing the structural information of the input graphs adequately. Newly proposed graph kernels are often justified by their ability to take structural graph properties into account that were ignored by previous kernels. Yet, to the best of our knowledge, this argument has not been formalized. Moreover, there is no theoretical justification for why certain kernels perform better than others, but merely experimental evaluations. We address this by introducing a theoretical framework for the analysis of the expressivity of graph kernels motivated by concepts from property testing, see, e.g., [31]. We consider normalized kernels, which measure similarity in terms of angles in a feature space. We say that a graph kernel *identifies* a property if no two graphs are mapped to the same normalized feature vector unless they both have or both do not have the property. A positive angle between two such feature vectors can be helpful to classify the property. As the graph size increases, on the one hand, this angle can become very small (dependent on the graph size), which is hindering when applying this knowledge to a learning setting. On the other hand, we observe that a constant angle between any two feature vectors of two graphs with complementing properties can only rarely be the case, since only a marginal change in a graph's features can change its property. If a graph can be edited slightly to obtain a property, it can, however, be viewed as close enough to the property to be ignored. Thus, in the sense of property testing, it is desirable to differentiate between the graph set far away from a property and the property itself, which motivates the following concept. We say that a graph kernel *distinguishes* a property if it guarantees a constant angle (independent of the graph size) between the

feature vectors of any two graphs, one of which has the property and the other is far away from doing so. We study well-known graph kernels and their ability to identify and distinguish fundamental properties, e.g., connectivity or connectivity. We propose a new graph kernel based on local k-discs which can, in contrast to previous kernels, distinguish global properties such as planarity in bounded-degree graphs. Moreover, the k-disc kernel is efficiently computable and has a feature space of constant dimension. For a constant dimensional feature space, we obtain learning guarantees for kernels that distinguish the class label property.

3.4.1. Definitions from property testing

Here, we assume the *bounded-degree graph model*. Let G and H be two d-bounded degree graphs in $\mathcal{G}_n$. The *edit distance* $\Delta(G, H)$ between G and H is the minimum number of edge modifications, i.e., adding or deleting edges, that have to be performed on G to obtain an isomorphic copy of H. A *graph property* is a set $\mathcal{P}$ of graphs that is closed under isomorphism. We denote the set of graphs in $\mathcal{P}$ on n vertices by $\mathcal{P}_n$. Let $\mathcal{P}_n$ be a non-empty graph property. A d-bounded degree graph G with n vertices is *ε-far* from $\mathcal{P}_n$ in the bounded degree model if for all d-bounded degree graphs H in $\mathcal{P}_n$ we have

$$\Delta(G, H) > \varepsilon dn$$

for $\varepsilon > 0$. Here we study the following graph properties. A graph $G = (V, E)$ is called *connected* if for every two vertices u and v in $V(G)$ there exists a path from u to v. A graph G is *planar* if there exists an embedding of G into the plane such that no edges cross, it is *bipartite* if $V(G)$ can be partitioned into two sets V_1 and $V_2 \subset V(G)$ such that for each edge (u, v), u in V_1 and v in V_2 or vice versa. A graph is *triangle-free* if it does not contain a cycle with three vertices.

3.4.2. Distinguishable graph properties

Let $\hat{k}$ be the cosine normalized version of a kernel k and denote its normalized feature map by $\hat{\phi}$, i.e.,

$$\begin{aligned}\hat{k}(x, y) &= \left\langle \hat{\phi}(x), \hat{\phi}(y) \right\rangle = \left\langle \frac{\phi(x)}{\|\phi(x)\|_2}, \frac{\phi(y)}{\|\phi(y)\|_2} \right\rangle \\ &= \frac{k(x, y)}{\sqrt{k(x, x) \cdot k(y, y)}} \in [-1, 1].\end{aligned}$$

The normalized kernel $\hat{k}(x, y)$ is equal to the cosine of the angle between $\phi(x)$ and $\phi(y)$ in the feature space. We say that a graph kernel *identifies* a property if no two graphs are mapped to the same normalized feature vector unless they both have or both do not have the property.

Definition 3.4.1. Let $\mathcal{P}$ be a graph property. If a graph kernel $k\colon \mathcal{G}\times\mathcal{G} \to \mathbb{R}$ for each n in $\mathbb{N}$, and every G in $\mathcal{P}_n$ and H not in $\mathcal{P}_n$, satisfies

$$\frac{k(G,H)}{\sqrt{k(G,G)\cdot k(H,H)}} < 1,$$

we say that $\mathcal{P}$ can be *identified* by k.

For a graph kernel to be able to distinguish a graph property, and to use this knowledge in a learning context, a desirable goal is to have a constant angle independent of n. However, in the following instance, a constant difference cannot be achieved.

Proposition 3.4.2. For the shortest-path kernel k, it holds that for each constant c, $0 < c < 1$, there exists some n in $\mathbb{N}$ and two graphs G and H in $\mathcal{G}_n$ with G connected, and H not connected such that

$$\frac{k(G,H)}{\sqrt{k(G,G)\cdot k(H,H)}} > 1 - c. \tag{3.17}$$

Proof. Let, for each n in $\mathbb{N}$, G be a path with n vertices, and let H consist of a path with $n-1$ vertices and one isolated vertex. Note that H is not connected, whereas adding one edge to H is enough to transform it into a connected graph which is isomorphic to G. The feature vectors for G and H counting the number of vertex pairs with distances 1 to $n-1$ are $\phi = (n-1, n-2, \ldots, 1)$ in $\mathbb{R}^{n-1}$ and $\psi = (n-2, n-3, \ldots, 1, 0)$ in $\mathbb{R}^{n-1}$, respectively. Additionally, in H there are $n-1$ vertex pairs that are not connected. It can be computed that $\langle\hat{\phi}, \hat{\psi}\rangle^2 = 1 - \frac{3}{4n^2-8n+3}$. Assume there is a constant c, $0 < c < 1$, such that, for each n in $\mathbb{N}$, it holds that $\langle\hat{\phi}, \hat{\psi}\rangle \leq 1 - c$, then there would be a constant $c' = (1-c)^2$, $0 < c' < 1$ such that $c' \geq 1 - \frac{3}{4n^2-8n+3}$ which does not hold for a large choice of n. Thus, for each constant c there exists an n in $\mathbb{N}$ such that, for the graphs G and H as chosen above, $k(G,H) > 1 - c$ holds. □

Note that both graphs in the proof of Proposition 3.4.2 have a maximum degree of 2. Therefore, the statements hold if any degree bound $d \geq 2$ is

required. To be able to achieve an angle independent of the graph size, we suggest employing the notion of a graph being ε-far from a property as used in property testing. We aim to obtain a constant[7] angle between the feature vectors of two graphs whenever one graph has a certain property and the other is ε-far from having that property. In this context, we define *distinguishability* of a graph property by a graph kernel as follows.

Definition 3.4.3. In the *bounded-degree graph model*, a graph property $\mathcal{P}$ is called *distinguishable* by a graph kernel $k\colon \mathcal{G} \times \mathcal{G} \to \mathbb{R}$, if for every $\varepsilon > 0$, and d in $\mathbb{N}$, there exists some $\delta = \delta(\varepsilon, d) > 0$ such that for every n in $\mathbb{N}$, every G in $\mathcal{P}_n$, and every graph H that is ε-far from $\mathcal{P}_n$, we have

$$\frac{k(G,H)}{\sqrt{k(G,G)\cdot k(H,H)}} \leq 1 - \delta. \tag{3.18}$$

Note that this notion does not guarantee an accurate learning algorithm in general. Consider the isomorphism kernel that is 1 for two isomorphic graphs and 0 otherwise. It distinguishes every property, but as a classifier will not generalize to unseen data. Nevertheless, we obtain some learning guarantees, see Section 3.4.5.

3.4.3. Properties distinguishable by popular graph kernels

In this section, we study the identifiability and distinguishability of the random walk, the Weisfeiler–Lehman subtree, the shortest-path, and the graphlet kernel. Table 3.11 sums up these results in comparison to the k-disc kernel studied in Section 3.4.4. Both, the feature maps of a random walk kernel and the Weisfeiler–Lehman subtree kernel cannot identify a regular graph. In particular, for the random walk kernel, the number of walks of length ℓ starting in a vertex of a regular graph with degree d is d^ℓ. Hence, for two regular graphs the kernel function is independent from the adjacency matrix of the product graph.

For the Weisfeiler–Lehman subtree kernel, two regular graphs with the same degree obtain the same feature vector due to [5]. Therefore, as soon as for some graph property $\mathcal{P}$ and n in $\mathbb{N}$ there exists one regular graph in $\mathcal{P}_n$ and another regular graph in $\mathcal{G}_n \setminus \mathcal{P}_n$, both kernels cannot identify the graph property.

[7] By constant we refer to a value independent of the input size n, which, however, can depend on ε or d.

Property	Graph Kernel				
	WL	RW	SP	Graphlet	k-Disc
Connectivity	✗	✗	✓	✗	✓
Planarity	✗	✗	✗	✗	✓
Bipartitness	✗	✗	✗	✗	
Triangle-freeness	✗	✗	✗	•	✓

Table 3.11.: Distinguishability of graph properties for the Weisfeiler-Lehman subtree kernel (WL), random walk (RW), graphlet, shortest-path (SP), and k-disc, Key: ✓ distinguishable, • identifiable (but not distinguishable), and ✗ not identifiable.

Graph G Graph $H = K_{3,3}$

Figure 3.5.: Counterexample for the proof of Theorem 3.4.4, Corollary 3.4.5, and Theorem 3.4.7.

Theorem 3.4.4. The random walk kernel cannot identify connectivity, planarity, bipartiteness, or triangle freeness.

Proof. A cycle with six vertices and a graph consisting of two disconnected triangles with three vertices, both regular graphs, are a counterexample to the identifiability of connectivity. Furthermore, consider the graphs G and H as illustrated in Figure 3.5. Since G is planar, but not bipartite, and contains triangles, whereas H is not planar, but bipartite, and triangle-free, the result follows. □

By the same arguments we obtain the following.

Corollary 3.4.5. The Weisfeiler–Lehman subtree kernel cannot identify connectivity, planarity, bipartiteness, or triangle freeness.

Next, we attend to a positive result regarding connectivity and the shortest-path kernel. We will make use of the following technical lemma throughout proofs in this section.

Lemma 3.4.6 ([65])**.** Let n and r in $\mathbb{N}$, $\mathbf{x}$ in $\mathbb{R}^r_{\geq 0}$, and $\varepsilon > 0$. For a non-empty subset of indices $S \subseteq \{1, \ldots, r\}$ such that $\sum_{i \in S} |x_i| = \eta > 0$ the following

holds for every $\mathbf{y}$ in $\mathbb{R}^r_{\geq 0}$ with $y_i = 0$ for each i in S:

$$\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} \leq \sqrt{1 - \frac{\eta^2}{|S| \cdot \|\mathbf{x}\|_2^2}}.$$

Proof. Without loss of generality, let $S = \{1, \ldots, s\}$. For each $\mathbf{y}$ in $\mathbb{R}^r_{\geq 0}$ with $y_i = 0$, $1 \leq i \leq s$, it holds that

$$\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} = \left\langle \left(\frac{x_{s+1}}{\|\mathbf{x}\|_2}, \ldots, \frac{x_r}{\|\mathbf{x}\|_2} \right), \left(\frac{y_{s+1}}{\|\mathbf{y}\|_2}, \ldots, \frac{y_r}{\|\mathbf{y}\|_2} \right) \right\rangle,$$

which, by the Cauchy-Schwarz inequality, is at most

$$\frac{\sqrt{\sum_{i=s+1}^r x_i^2}}{\|\mathbf{x}\|_2} \cdot \frac{\|(y_{s+1}, \ldots, y_r)\|_2}{\|\mathbf{x}\|_2} = \frac{\sqrt{\|\mathbf{x}\|_2^2 - \sum_{i=1}^s x_i^2}}{\|\mathbf{x}\|_2} = \sqrt{1 - \frac{\sum_{i=1}^s x_i^2}{\|\mathbf{y}\|_2^2}}.$$

Moreover, since $\sum_{i=1}^s x_i^2 \geq \frac{1}{s} \cdot \left(\sum_{i=1}^s x_i\right)^2 = \frac{1}{s}\eta^2$, this is at most

$$\sqrt{1 - \frac{\eta^2}{s \|\mathbf{x}\|_2^2}}.$$

□

We get the following result.

Theorem 3.4.7 ([65])**.** The shortest-path kernel

1. cannot identify planarity, bipartiteness, or triangle freeness, and
2. can distinguish connectivity.

Proof.

1. While in general two regular graphs may have different feature vectors, the graphs in Figure 3.5 also serves as a counterexample here. In both cases the shortest path feature vector are equal, as there are nine shortest paths of length 1 and six of length 2, each.

2. Let n and d in $\mathbb{N}$, $\varepsilon > 0$, and H in $\mathcal{G}_n$ be ε-far from being connected. For the shortest-path feature vector $\psi = (\psi_1, \ldots, \psi_{n-1})$ and each connected graph G in $\mathcal{G}_n$ with shortest-path feature vector $\phi = (\phi_1, \ldots, \phi_{n-1})$, it holds that

$$\langle (\psi_1, \ldots, \psi_{n-1}), (\phi_1, \ldots, \phi_{n-1}) \rangle = \langle (\psi_1, \ldots, \psi_{n-1}, \eta), (\phi_1, \ldots, \phi_{n-1}, 0) \rangle,$$

where η denotes the number of disconnected vertex pairs in H. By Lemma 3.4.6 it holds that

$$\langle \hat{\phi}, \hat{\psi} \rangle \leq \sqrt{1 - \frac{\eta^2}{\|\psi\|_2^2}}. \tag{3.19}$$

Assume that $n > 4/\varepsilon d$. Otherwise with $\eta \geq 1$ and $\|\psi\|_2 \leq n^2 \leq (4/\varepsilon d)^2$, it holds that Equation (3.19) is at most 1 minus a constant. Now, it is known that there are more than $\varepsilon dn/2$ connected components of which at least $\varepsilon dn/4$ have a size smaller than $4/\varepsilon d$ [32]. At least one vertex in such a small component is disconnected from each vertex outside the component, that is, η is at least $1/2 \cdot \varepsilon dn/4 \cdot (n - 4/\varepsilon d) = \varepsilon n^2 d/8 - n/2$. Moreover, with $\langle \psi, \psi \rangle \leq \left(n(n-1)/2 - \eta\right)^2 + \eta^2$ we obtain the following:

$$\begin{aligned}
\frac{\|\psi\|_2^2}{\eta^2} &\leq \frac{n^2(n-1)^2}{4\left(\frac{\varepsilon n^2 d}{8} - \frac{n}{2}\right)^2} - \frac{4n(n-1)}{\frac{\varepsilon n^2 d}{2} - 2n} + 2 \\
&= \frac{(n-1)^2}{\left(\frac{\varepsilon n d}{2} - 2\right)\left(\frac{\varepsilon n d}{8} - \frac{1}{2}\right)} - \frac{4(n-1)}{\frac{\varepsilon n d}{2} - 2} + 2 \\
&\leq \frac{16(n-1)^2}{(\varepsilon n d - 4)^2} - \frac{8(n-1)}{\varepsilon n d - 4} + 2 \\
&= \frac{8}{\varepsilon d}\left(1 + \underbrace{\frac{4 - \varepsilon d}{\varepsilon dn - 4}}_{\in]0,1[}\right)\underbrace{\left(\frac{2}{\varepsilon d}\left(1 + \underbrace{\frac{4 - \varepsilon d}{\varepsilon dn - 4}}_{\in]0,1[}\right) - 1\right)}_{>0} + 2 \\
&\leq \underbrace{\frac{16}{\varepsilon d}\left(\frac{4}{\varepsilon d} - 1\right)}_{\zeta} + 2.
\end{aligned}$$

Note that ζ is a positive number independent of n. All in all,

$$1 - \frac{\eta^2}{\|\psi\|_2^2} \leq \frac{1}{2 + \zeta},$$

for some $\zeta > 0$ independent of n, which implies that Equation (3.19) is smaller than 1 by a constant strictly between 0 and 1. □

Finally, we consider the graphlet kernel. Although the it appears to be rather expressive, from the considered properties it can only identify triangle-freeness.

Theorem 3.4.8. The graphlet kernel can identify triangle-freeness for $k \geq 3$, but unless the graphlet size k depends on the graph size, it cannot identify connectivity, bipartiteness, or planarity. Moreover, the graphlet kernel cannot distinguish any graph property.

Proof. Triangle-freeness is identifiable, because, for each graphlet size $k \geq 3$, and for each graph that contains a triangle, there exists at least one k-graphlet that contains a triangle. The corresponding entry in the feature vector of a triangle-free graph is 0.

In the following, let k be a graphlet size. For connectivity and bipartiteness, consider the following two graphs as a counterexample. The first graph is an even cycle of size $4k + 2$, which is connected and bipartite; the second graphs consists of two odd cycles of size $2k + 1$ each, which is not connected and not bipartite. Note that the number of graphlets of each type (paths or a collection of paths of length up to k) is the same in both graphs.

For planarity, consider the two graphs G and H in Figure 3.6 with $n = 20(k+1)$ vertices each, where dotted lines denote paths of length $n/20 = k+1$.

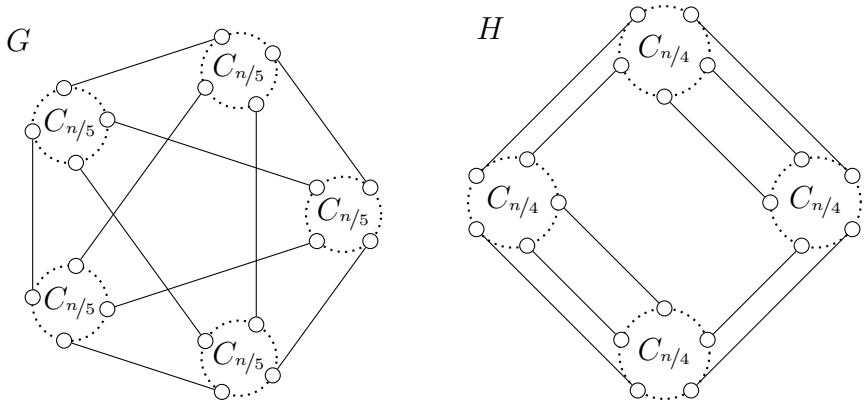


Figure 3.6.: Counterexample for the proof of Theorem 3.4.8.

Note that H is planar, and G is not. Furthermore, observe that all graphlets of size k, that occur in either of the two graphs, cannot contain any cycle. There are in fact only a few different graphlet types to be found, namely subgraphs whose connected components contain at most one solid edge, each of which occur in both graphs evenly distributed.

For distinguishability of any graph property, there is one general obstacle, namely the fact that graphlets do not have to be connected. For each graph with n vertices and bounded degree d, there are at least

$$1 - \frac{d(k-1)k}{2(n-1)}$$

graphlets induced by k vertices with k independent vertices. Thus, for each constant $\delta > 0$, and each G and H in $\mathcal{G}_n$ it holds that

$$\frac{k^k_{\mathrm{GR}}(G,H)}{\sqrt{k^k_{\mathrm{GR}}(G,G) \cdot k^k_{\mathrm{GR}}(H,H)}} > 1 - \delta,$$

which implies that there cannot be a constant angle. □

3.4.4. Graph kernels that distinguish graph properties

While we have observed that established graph kernels often cannot distinguish basic properties, we aim to find a graph kernel that can distinguish fundamental properties and is efficiently computable. Following the notation of [85], we define a histogram $\mathrm{hist}_G(k)$ of the numbers of different (i.e., non-isomorphic) k-discs around vertices in G. Here, the k-disk around a vertex v in $V(G)$ is the induced subgraph of all vertices reachable from v on a path of length smaller or equal than k, cf. Definition 3.3.9 (for $c = k$). Moreover, we define the frequency vector

$$\mathrm{freq}_G(k) = \mathrm{hist}_G(k)/n,$$

for G in $\mathcal{G}_n$. Consider the following graph kernel.

Definition 3.4.9. Given two graphs G and H, the *k-disc graph kernel* is defined by

$$k_{\mathrm{KD}}(G,H) = \langle \mathrm{freq}_G(k), \mathrm{freq}_H(k) \rangle.$$

A significant difference between the k-disc kernel and the graphlet kernel is that a k-disc is a connected subgraph of a graph, while a graphlet may be disconnected. For d-bounded-degree graphs, the k-disc kernel can be computed in time linear in the graph size.

Theorem 3.4.12 comprises the main results of this section about graph properties distinguishable by the k-disc kernel. From property testing studies, see, e.g., [85], we often obtain information about the 1-norm of the distance between the frequency vectors of a graph ε-far from a property and all graphs satisfying the property. To translate these facts to a positive angle between the frequency vectors, we need the following two lemmas. Firstly, it can be seen that for two normalized real vectors with at least one index at which the entries differ by at least a constant positive value, their (standard) inner product is strictly less than 1.

Lemma 3.4.10 ([65])**.** Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors in $\mathbb{R}^n_{\geq 0}$ for some n in $\mathbb{N}$ with $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1$. Let $\zeta > 0$ be an arbitrarily small real value. If there exists some i and $1 \leq i \leq n$ such that $|x_i - y_i| \geq \zeta$, then

$$\langle \mathbf{x}, \mathbf{y} \rangle \leq 1 - {}^{\zeta^2}/_2.$$

Proof. It holds that,

$$\begin{aligned}\langle \mathbf{x}, \mathbf{y} \rangle &= \frac{1}{2}\left(\|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - \|\mathbf{x} - \mathbf{y}\|_2\right) \\ &= 1 - \frac{1}{2}\Big(\underbrace{(x_i - y_i)^2}_{\geq \zeta^2} + \sum_{j=1, j\neq i}^{n} \underbrace{(x_j - y_j)^2}_{\geq 0}\Big) \leq 1 - \frac{\zeta^2}{2}.\end{aligned}$$

Thus, the angle between $\mathbf{x}$ and $\mathbf{y}$ is positive and independent of n. □

Secondly, we need to take care of the fact that the studied frequency vectors are normalized with respect to their 1-norm. However, since the number of different k-discs is independent of the number of vertices in the bounded-degree model, we can show the following lemma for two frequency vectors with a positive distance with respect to their 1-norm.

Lemma 3.4.11 ([65])**.** Let ϕ and ψ be two vectors in $\mathbb{R}^n_{\geq 0}$ with $\|\phi\|_1 = \|\psi\|_1 = 1$ and $\|\phi - \psi\|_1 \geq \eta$. Then, there exists an index i, for $1 \leq i \leq n$, and a real value $\zeta > 0$ such that

$$\left|\frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2}\right| \geq \zeta.$$

Proof. Let without loss of generality $\|\phi\|_2 \geq \|\psi\|_2$. Moreover, let s denote the number of positive entries in both, ϕ and ψ. Observe that $\|\phi - \psi\|_1 \geq \eta$ implies the existence of a j: $\phi_j - \psi_j \geq {}^{\eta}/_s$.

Case 1: If $\|\phi\|_2 - \|\psi\|_2 \leq \frac{\eta}{2s^{3/2}}$, then

$$\frac{\phi_j}{\|\phi\|_2} - \frac{\psi_j}{\|\psi\|_2} \geq \underbrace{\|\psi\|_2}_{\geq \frac{1}{\sqrt{s}}} \underbrace{(\phi_j - \psi_j - j)}_{\geq \frac{\eta}{s}} - \underbrace{\psi_j}_{\leq 1-\eta} \underbrace{(\|\phi\|_2 - \|\psi\|_2)}_{\leq \frac{\eta}{2s^{3/2}}} \geq \frac{\eta}{2s^{3/2}}.$$

Case 2: $\|\phi\|_2 - \|\psi\|_2 > \frac{\eta}{2s^{3/2}}$. Let R denote the subset of indices i such that ${}^{\phi_i}/_{\|\phi\|_2} > {}^{\psi_i}/_{\|\psi\|_2}$.

Note that $|R| < s$, otherwise

$$\sum_{i=1}^{n} \left(\frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2}\right) > 0,$$

which implies that

$$\frac{1}{\|\phi\|_2} - \frac{1}{\|\psi\|_2} > 0,$$

a contradiction to the assumption that $\|\phi\|_2 \geq \|\psi\|_2$. For $0 \leq |R| < s$ it holds that

$$\begin{aligned}\left\|\hat{\phi} - \hat{\psi}\right\|_1 &= \sum_{i=1}^{n} \left| \frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2} \right| \\ &= \sum_{i \in R} \left(\frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2} \right) \\ &\quad + \sum_{i \notin R} \left(\frac{\psi_i}{\|\psi\|_2} - \frac{\phi_i}{\|\phi\|_2} \right).\end{aligned}$$

Since $\|\psi\|_1 = \|\phi\|_1 = 1$, this is equal to

$$\begin{aligned}&\sum_{i \in R} \left(\frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2} \right) + \frac{1 - \sum_{i \in R} \psi_i}{\|\psi\|_2} - \frac{1 - \sum_{i \in R} \phi_i}{\|\phi\|_2} \\ &= \frac{1}{\|\psi\|_2} - \frac{1}{\|\phi\|_2} + 2 \sum_{i \in R} \underbrace{\left(\frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2} \right)}_{\geq 0} > \frac{\eta}{2s^{3/2}}.\end{aligned}$$

Thus, there exists an index i, $1 \leq i \leq n$ such that

$$\left| \frac{\phi_i}{\|\phi\|_2} - \frac{\psi_i}{\|\psi\|_2} \right| > \frac{\eta}{2s^{5/2}}.$$

With $0 < \zeta \leq \eta / 2s^{5/2}$ in both cases, this completes the proof. □

Finally, we can proof the main theorem of this section.

Theorem 3.4.12 ([65])**.** For the k-disc graph kernel, it holds that

1. connectivity is distinguishable for $k \geq 4/\varepsilon d$,
2. triangle-freeness is distinguishable for $k \geq 1$, and
3. for each $\varepsilon > 0$, d in $\mathbb{N}$ there exists some k in $\mathbb{N}_{>0}$ such that distinguishability is satisfied for planarity.

Proof.

1. Let H in $\mathcal{G}_n$ be a graph with bounded degree d that is ε-far from being connected for some $\varepsilon > 0$. By [32] we know that the number of connected components of a size smaller than $4/\varepsilon d$ is at least $\varepsilon dn/4$. For each vertex in such a small component, the full component is found as a k-disc of size $4/\varepsilon d$ in H. For each connected graph G, the frequency of such a small component is 0, since each k-disc covers at least $4/\varepsilon d$ vertices. Therefore,

$$\|\text{freq}_G(k) - \text{freq}_H(k)\|_1 \geq \varepsilon d/4.$$

The conditions of Lemma 3.4.6 are satisfied for $\mathbf{x} = \text{freq}_H(k)$ and $\mathbf{y} = \text{freq}_G(k)$ with S indicating the occurrence of small components. Again, observe that $|S|$ is independent of n. By $\|\mathbf{x}\|_2^2 \leq 1$ and $\eta \geq \frac{\varepsilon d}{4}$ we obtain

$$\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2} \leq \sqrt{1 - \frac{\varepsilon d}{4|S|}},$$

which is a positive constant strictly less than 1.

2. For triangle-freeness, again, we can use similar arguments to [32]. If a graph is ε-far from being triangle-free, there are εdn superfluous edges in H in contrast to any triangle-free graph G. Note that only edges that are part of a triangle are to be removed. Each such edge is shared by two vertices, and there can be at most d edges involved per vertex. That means, that at least $2\varepsilon n$ vertices are incident to a superfluous edge. These vertices hence have k-discs, for each $k \geq 1$, that contain triangles, whereas in G there are no such k-discs. Therefore,

$$\|\text{freq}_G(k) - \text{freq}_H(k)\|_1 \geq 2\varepsilon.$$

Since the frequency vectors are always normalized with respect to their 1-norm, the conditions of Lemma 3.4.11 hold. Thus, there exists an index i, for $1 \leq i \leq n$, and a constant $\zeta > 0$ such that

$$\left| \frac{\text{freq}_G(k)_i}{\|\text{freq}_G(k)\|_2} - \frac{\text{freq}_G(k)_i}{\|\text{freq}_H(k)\|_2} \right| \geq \zeta.$$

Then, by Lemma 3.4.10,

$$\left\langle \frac{\text{freq}_G(k)_i}{\|\text{freq}_G(k)\|_2}, \frac{\text{freq}_G(k)_i}{\|\text{freq}_H(k)\|_2} \right\rangle$$

is smaller than 1 by a constant.

3. Benjamini, Schramm, and Shapira [7] show that for each $\varepsilon > 0$ and degree bound d, there exists a positive integer k independent of n such that for any two graphs G and H in $\mathcal{G}_n$ with bounded degree d, G planar, H ε-far from being planar, it holds that

$$\|\mathrm{freq}_G(k) - \mathrm{freq}_H(k)\| \geq 1/k.$$

Therefore, via Lemmas 3.4.11 and 3.4.10, we obtain the claimed result. □

3.4.5. A learning algorithm

In this section, we study a kernel nearest neighbor classifier for graphs and show that its prediction error can be bounded under the assumption that the employed kernel can distinguish the class label property, and that all considered graphs either satisfy the property or are ε-far from it. We assume the following supervised binary classification problem: Let $\mathcal{Y} = \{0, 1\}$ be the set of possible class labels, which represent if a graph has a property or not. We aim to learn a concept $c\colon \mathcal{G}_n \to \mathcal{Y}$ such that the binary loss is minimized. Thereto we receive a training set $\{g_1, \ldots, g_m\} \subset \mathcal{G}_n$ and a test graph from $\mathcal{G}_n$ sampled independently and identically from some unknown distribution, as well as the set of class labels $\{c(g_1), \ldots, c(g_n)\}$ for the training set with regard to the concept. We assume in the following that for the considered graph property, $\mathcal{G}_n$ only contains graphs that either have the property or are ε-far from it.

Kernel nearest neighbor classification

Based on a training set T of data points in $\mathbb{R}^D$ with known class labels, the k-nearest neighbor classifier (k-NN) assigns a test data point to the class most common among its k nearest neighbors in T. Here, the nearest neighbors are commonly determined based on the Euclidean distance between data points. Kernel nearest neighbor classifiers have been realized by substituting this distance by a kernel metric in a Hilbert space, see, e.g., [126]. For a kernel k with feature map ϕ, we consider the 1-NN algorithm using the kernel metric

$$d_k(\mathbf{x}, \mathbf{y}) = \|\phi(\mathbf{x}) - \phi(\mathbf{y})\|_2 = \sqrt{k(\mathbf{x}, \mathbf{x}) + k(\mathbf{y}, \mathbf{y}) - 2k(\mathbf{x}, \mathbf{y})}. \tag{3.20}$$

Learning with distinguishing kernels

We again consider the cosine normalized version $\hat{k}$ of a kernel k and its normalized feature map $\hat{\phi}$. For dimension D of the feature space, the normalized

feature map $\hat{\phi}$ assigns graphs to points on the unit sphere S^{D-1}. Let us assume that $\hat{k}$ distinguishes the class label property. Then, there is a δ, such that for all graphs G that have the property and H that are ε-far from it, we have $\hat{k}(G,H) \leq 1-\delta$ and, consequently,

$$d_{\hat{k}}(G,H) \geq \sqrt{1+1-2(1-\delta)} = \sqrt{2\delta}.$$

We denote this guaranteed minimum distance by

$$\Delta = \sqrt{2\delta}.$$

Consider the spherical cap C within the open ball centered at $\hat{\phi}(G)$ with radius Δ. According to the assumption, every graph H with $\hat{\phi}(H)$ lying on C, must have the same class label.

Proposition 3.4.13. Let G be a graph of the training set. Then every graph H with $d_{\hat{k}}(G,H) < \Delta$ is correctly classified by 1-NN where the base set contains all graphs that either have the property or are ε-far from it.

Proof. Assume H is not correctly classified and $d_{\hat{k}}(G,H) < \Delta$. Due to distinguishability, H must belong to the same class as G. Since H is not correctly classified by 1-NN, there must be a nearest neighbor $N \neq G$ of H with a different class label. Since N is a nearest neighbor of G, we have $d_{\hat{k}}(H,N) \leq d_{\hat{k}}(G,H) < \Delta$, contradicting distinguishability. □

We say that an algorithm (ε,λ)-*learns* a property if a test graph G drawn from the underlying distribution is correctly classified with probability $(1-\lambda)$ and the base set consists of all graphs that either have the property or are ε-far from it.

Theorem 3.4.14. Let k be a kernel that distinguishes the class label property according to Definition 3.4.3 with some fixed δ and a feature space of dimension D. Let $\Delta = \sqrt{2\delta}$. Let all graphs either satisfy the property or be ε-far from it. Assume that the *training set* has cardinality

$$m \geq {}^{(1+6/\Delta)^D}/_{\lambda} \ln\left({}^{(1+6/\Delta)^D}/_{\lambda}\right)$$

then the 1-NN algorithm (ε,λ)-learns the property.

Proof. We cover the unit sphere with balls of radius $\Delta/3$. It is well-known that such a cover of size $B = (1+{}^{6}/_{\Delta})^D$ exists. We observe that if a training example falls into a ball of the cover then by Proposition 3.4.13 any other example inside this ball is correctly classified. We observe that with probability $1-\lambda$ every ball with probability mass at least ${}^{\lambda}/_{B}$ contains a training example. The overall probability of the remaining balls is at most λ. Therefore, with probability $1-\lambda$ the algorithm (ε,λ)-learns the property. □

3.4.6. Conclusion

We have introduced a framework for analyzing graph kernels with respect to their ability to distinguish graphs that satisfy a property from those that are far away from doing so. We showed that some popular graph kernels fail to even identify fundamental graph properties. Subsequently, we introduced the k-disc kernel which is able to distinguish many basic graph properties. The framework opens up interesting directions for future work, e.g., analyzing more graph properties. For instance, a graph is *cycle-free* if it does not contain any cycle, that is, each connected component is a tree. The Weisfeiler–Lehman subtree kernel can identify cycle-freeness, because graphs from this class have unique feature vectors up to isomorphism, see [5]. Thus, their feature vectors are distinct and the angle between them is positive. Note that δ still depends on n. Similarly, the shortest-path kernel can identify cycle-freeness via the number of edges. We assume that it can also distinguish cycle-freeness, since a graph ε-far from cycle-freeness contains at least εnd edges more than each cycle-free graph. The k-disc kernel can distinguish cycle-freeness for some k in $\mathbb{N}$ via arguments shown in [7]. Finally, the development of more realistic learning setting should be considered, e.g., by changing Definition 3.4.3 by emposing a smoothness assumption on δ.

Chapter 4.

Neural methods for graphs

In this chapter, we present our work on neural methods for graphs. In recent years, neural network-based models had a surge of interest in the machine learning community. Hence, the question arises if such methods can be applied to graph data. Graph neural networks (GNNs) have emerged as a machine learning framework addressing the above challenge. In the following, we investigate their power. More specifically, we show that they cannot be more powerful than the 1-WL in terms of distinguishing non-isomorphic graphs. Going further, we leverage these theoretical relationships to propose a generalization of GNNs, called k-GNNs, which are neural architectures based on the k-WL, see Section 2.6, and are strictly more powerful than GNNs. The key insight in these higher-dimensional variants is that they perform message passing directly between subgraph structures, rather than individual vertices. This higher-order form of message passing can capture structural information that is not visible at the vertex-level.

Graph kernels based on the k-WL have been proposed in the past, see Section 3.3. However, a key advantage of implementing higher-order message passing in GNNs—which we demonstrate here—is that we can design hierarchical variants of k-GNNs, which combine graph representations learned at different granularities in an end-to-end trainable framework. Concretely, in the presented hierarchical approach, the initial messages in a k-GNN are based on the output of lower-dimensional k'-GNNs (with $k' < k$), which allows the model to effectively capture graph structures of varying granularity. Since many real-world graphs inherit a hierarchical structure—e.g., in a social network we must model both the ego-networks around individual vertices, as well as the coarse-grained relationships between entire communities, see, e.g., [86], we believe that this layer offers empirical utility.

4.1. Related work

We will first give a general description of GNNs and then discuss related work. Standard GNNs can be viewed as a neural version of the 1-WL algorithm, where colors are replaced by continuous feature vectors and neural networks are used to aggregate over vertex neighborhoods [39, 56]. In effect, the GNN framework can be viewed as implementing a continuous form of graph-based "message passing", where local neighborhood information is aggregated and passed on to the neighbors [30]. By deploying a trainable neural network to aggregate information in local vertex neighborhoods, GNNs can be trained in an end-to-end fashion together with the parameters of the classification or regression algorithm, possibly allowing for greater adaptability and better generalization.

Let (G, l) be a labeled graph with an initial vertex coloring $f^{(0)} \colon V(G) \to \mathbb{R}^{1\times d}$ that is *consistent* with l. This means that each vertex v is annotated with a feature $\mathbf{f^{(0)}}(\mathbf{v})$ in $\mathbb{R}^{1\times d}$ such that $\mathbf{f^{(0)}}(\mathbf{u}) = \mathbf{f^{(0)}}(\mathbf{v})$ if and only if $l(u) = l(v)$. Alternatively, $\mathbf{f^{(0)}}(\mathbf{v})$ can be an arbitrary real-valued feature vector associated with v. Examples include continuous atomic properties in cheminformatic applications where vertices correspond to atoms, or vector representations of text in social network applications. A GNN model consists of a stack of neural network layers, where each layer aggregates local neighborhood information, i.e., features of neighbors, and then passes this aggregated information on to the next layer.

A basic GNN model can be implemented as follows [40]. In each layer $t > 0$, we compute a new feature

$$\mathbf{f^{(t)}}(\mathbf{v}) = \sigma\Big(\mathbf{f^{(t-1)}}(\mathbf{v}) \cdot \mathbf{W_1^{(t)}} + \sum_{w\in N(v)} \mathbf{f^{(t-1)}}(\mathbf{w}) \cdot \mathbf{W_2^{(t)}}\Big) \tag{4.1}$$

in $\mathbb{R}^{1\times e}$ for v in $V(G)$, where $\mathbf{W_1^{(t)}}$ and $\mathbf{W_2^{(t)}}$ are parameter matrices from $\mathbb{R}^{d\times e}$, and σ denotes a component-wise non-linear function, e.g., a sigmoid or a rectifier function.[1] Following [30], one may also replace the sum defined over the neighborhood in the above equation by a permutation-invariant, differentiable function, and one may substitute the outer sum, e.g., by a column-wise vector concatenation or LSTM-style update step. Thus, in full generality a new feature $\mathbf{f^{(t)}}(\mathbf{v})$ is computed as

$$f_{\text{merge}}^{\mathbf{W_2}}\Big(\mathbf{f^{(t-1)}}(\mathbf{v}), f_{\text{aggr}}^{\mathbf{W_1}}\big(\{\!\{\mathbf{f^{(t-1)}}(\mathbf{w}) \mid w \in N(v)\}\!\}\big)\Big), \tag{4.2}$$

[1] For clarity of presentation, we omit biases.

where $f_{\text{aggr}}^{\mathbf{W_1}}$ aggregates over the set of neighborhood features and $f_{\text{merge}}^{\mathbf{W_2}}$ merges the vertex's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\text{aggr}}^{\mathbf{W_1}}$ and $f_{\text{merge}}^{\mathbf{W_2}}$ may be arbitrary differentiable, permutation-invariant functions (e.g., neural networks), and by analogy to Equation 4.1, we denote their parameters as $\mathbf{W_1}$ and $\mathbf{W_2}$, respectively. In the rest of this chapter, we refer to neural architectures implementing Equation (4.2) as 1-*dimensional GNN architectures* (1-GNNs).

A vector representation $\mathbf{f}_{\text{GNN}}$ over the whole graph can be computed by summing over the vector representations computed for all vertices, i.e.,

$$\mathbf{f}_{\text{GNN}}(G) = \sum_{v \in V(G)} \mathbf{f}^{(\mathbf{T})}(\mathbf{v}), \tag{4.3}$$

where $T > 0$ denotes the last layer. More refined approaches use differentiable pooling operators based on, e.g., sorting [128], and soft assignments [125]. To adapt the parameters $\mathbf{W_1}$ and $\mathbf{W_2}$ of Equations (4.1) and (4.2) to a given data distribution, they are optimized in an end-to-end fashion (usually via some variant of stochastic gradient descent) together with the parameters of a neural network used for classification or regression.

Most of the neural approaches fit into the graph neural network framework proposed by [30]. Notable instances of this model include *Neural Fingerprints* [23], *Gated Graph Neural Networks* [69], *GraphSAGE* [39], *SplineCNN* [26], and the spectral approaches proposed in [10, 22, 56]—all of which descend from early work in [57, 107, 76, 101]. Recent extensions and improvements to the GNN framework include approaches to incorporate different local structures around subgraphs [120] and novel techniques for pooling vertex representations in order to perform graph classification [128, 125]. GNNs have achieved state-of-the-art performance on several graph classification benchmarks in recent years, see, e.g., [125]—as well as applications such as protein-protein interaction prediction [27], recommender systems [124], and the analysis of quantum interactions in molecules [103]. A survey of recent advancements in GNN techniques can be found in [40].

4.2. Relationship between the 1-WL and 1-GNNs

In the following, we explore the relationship between the 1-WL and 1-GNNs. Let (G, l) be a labeled graph, and let $\mathcal{W}^{(t)} = \left(\mathbf{W_1^{(t')}}, \mathbf{W_2^{(t')}}\right)_{t' \leq t}$ denote the GNN parameters given by Equation (4.1) or Equation (4.2) up to iteration

t. We encode the initial labels $l(v)$ by vectors $\mathbf{f^{(0)}}(\mathbf{v})$ in $\mathbb{R}^{1\times d}$, e.g., using a 1-hot encoding.

Our first theoretical result shows that 1-GNN architectures do not have more power in terms of distinguishing between non-isomorphic (sub-)graphs than the 1-WL algorithm. More formally, let $f_{\text{aggr}}^{\mathbf{W_1}}$ and $f_{\text{merge}}^{\mathbf{W_2}}$ be any two functions chosen in Equation (4.2). For every encoding of the labels $l(v)$ as vectors $\mathbf{f^{(0)}}(\mathbf{v})$, and for every choice of $\mathcal{W}^{(t)}$, we have that the coloring C_t^1 of 1-WL always refines the coloring $\mathbf{f^{(t)}}$ induced by a 1-GNN parameterized by $\mathcal{W}^{(t)}$.

Theorem 4.2.1. Let (G, l) be a labeled graph. Then for all $t \geq 0$, and for all choices of initial colorings $\mathbf{f^{(0)}}$ consistent with l, and weights $\mathcal{W}^{(t)}$,

$$C_t^1 \sqsubseteq \mathbf{f^{(t)}}.$$

Proof. We show for an arbitrary iteration t and vertices u and v in $V(G)$, that

$$C_{(t+1)}^1(u) = C_{(t+1)}^1(v)$$

implies

$$\mathbf{f^{(t+1)}}(\mathbf{u}) = \mathbf{f^{(t+1)}}(\mathbf{v}).$$

In iteration 0, we have

$$C_0^1(u) = C_0^1(v) \iff \mathbf{f^{(0)}}(\mathbf{u}) = \mathbf{f^{(0)}}(\mathbf{v})$$

as the initial vertex coloring $\mathbf{f^{(0)}}$ is chosen consistent with l.

Let u and v in $V(G)$ and t in $\mathbb{N}$ such that $C_{t+1}^1(u) = C_{t+1}^1(v)$. Assume for the induction that

$$C_t^1(u) = C_t^1(v) \implies \mathbf{f^{(t)}}(\mathbf{u}) = \mathbf{f^{(t)}}(\mathbf{v})$$

holds. As $C_{(t+1)}^1(u) = C_{(t+1)}^1(v)$, we know from the refinement step of the 1-WL that the colors of the previous iterations, $C_t^1(u)$ and $C_t^1(v)$ of u and v, respectively, as well as the multisets $\{\!\{C_t^1(w) \mid w \in N(u)\}\!\}$ and $\{\!\{C_t^1(w) \mid w \in N(v)\}\!\}$ of colors of the neighbors of u and v are identical.

Let $M_u = \{\!\{\mathbf{f^{(t)}}(\mathbf{w}) \mid w \in N(u)\}\!\}$ and $M_v = \{\!\{\mathbf{f^{(t)}}(\mathbf{v}) \mid w \in N(v)\}\!\}$ be the multisets of feature vectors of the neighbors of u and v, respectively. By the induction hypothesis, we know that $M_u = M_v$ and $\mathbf{f^{(t)}}(\mathbf{u}) = \mathbf{f^{(t)}}(\mathbf{v})$ such that independent of the choice of f_{merge} and f_{aggr}, we get

$$\mathbf{f^{(t+1)}}(\mathbf{u}) = \mathbf{f^{(t+1)}}(\mathbf{v}).$$

This holds as the input to both functions f_{merge} and f_{aggr} is identical. This proves

$$C^1_{(t+1)}(u) = C^1_{(t+1)}(v) \implies \mathbf{f^{(t+1)}}(\mathbf{u}) = \mathbf{f^{(t+1)}}(\mathbf{v}),$$

and thereby the theorem. □

Our second result states that there exists a sequence of parameter matrices $\mathcal{W}^{(t)}$ such that 1-GNNs have exactly the same power in terms of distinguishing non-isomorphic (sub-)graphs as the 1-WL algorithm. This even holds for the simple architecture of Equation (4.1), provided we choose the encoding of the initial labeling l in such a way that different labels are encoded by linearly independent vectors.

Theorem 4.2.2 ([83])**.** Let (G, l) be a labeled graph. Then for all $t \geq 0$ there exists a sequence of weights $\mathcal{W}^{(t)}$, and a 1-GNN architecture such that

$$C^1_t \equiv \mathbf{f^{(t)}}.$$

Hence, in the light of the above results, 1-GNNs may be viewed as an extension of the 1-WL which in principle have the same power but are more flexible in their ability to adapt to the learning task at hand and can handle continuous vertex features.

4.3. Proof of Theorem 4.2.2

For the proof, we first consider graphs where all vertices have the same initial color (or label) and then extend it to colored graphs. To do that we use a slightly adapted but equivalent version of the 1-WL. Note that the extension to colored graphs is mostly technical, while the important idea is already contained in the first case.

4.3.1. Uncolored graphs

Let Γ_G be the refinement operator for the 1-WL, mapping the old coloring $C^1_{(t-1)}$ to the updated one C^1_t:

$$C^1_t(v) = \left(\Gamma_G\left(C^1_{(t-1)}\right)\right)(v) = \left(C^1_{(t-1)}(v), \{\!\{C^1_{(t-1)}(u) \mid u \in N(v)\}\!\}\right).$$

We first show that for uncolored graphs, this is equivalent to the update rule $\tilde{\Gamma}_G$:

$$\tilde{C}^1_t(v) = \left(\tilde{\Gamma}_G\left(\tilde{C}^1_{(t-1)}\right)\right)(v) = \left(\{\!\{\tilde{C}^1_{(t-1)}(u) \mid u \in N(v)\}\!\}\right).$$

We denote $\mathbf{J}$ as the all-1 matrix where the size will always be clear from the context.

Lemma 4.3.1 ([83])**.** Let G be a graph, v and w in $V(G)$, and t in $\mathbb{N}$ such that $\tilde{C}^1_t(u) \neq \tilde{C}^1_t(v)$. Then $\tilde{C}^1_{t'}(u) \neq \tilde{C}^1_{t'}(v)$ for all $t' \geq t$.

Proof. Let t in $\mathbb{N}$ be minimal such that there are v and w with

$$\tilde{C}^1_t(u) \neq \tilde{C}^1_t(v), \tag{4.4}$$

and

$$\tilde{C}^1_{(t+1)}(u) = \tilde{C}^1_{(t+1)}(v). \tag{4.5}$$

Then $t \geq 1$, because $\tilde{C}^1_0 = \mathbf{J}$ as there are no initial colors. Let $P_1, \ldots, P_p$ be the color classes of $\tilde{C}^1_{(t-1)}$. That is, for all x and y in $V(G)$, we have

$$\tilde{C}^1_{(t-1)}(x) = \tilde{C}^1_{(t-1)}(y)$$

if and only if there is an i in $[p]$ such that x and y in P_i. Similarly, let $Q_1, \ldots, Q_q$ be the color classes of $\tilde{C}^1_t$. Observe that the partition $\{Q_1, \ldots, Q_q\}$ of $V(G)$ refines the partition $\{P_1, \ldots, P_p\}$.

Indeed, if there were $i \neq i'$ in $[p]$, and k in $[q]$ such that $P_i \cap Q_k \neq \emptyset$ and $P_{i'} \cap Q_k \neq \emptyset$, then all x in $P_i \cap Q_k$, y in $P_{i'} \cap Q_k$ would satisfy

$$\tilde{C}^1_{(t-1)}(x) \neq \tilde{C}^1_{(t-1)}(y) \text{ and } \tilde{C}^1_t(x) = \tilde{C}^1_t(y),$$

contradicting the minimality of q.

Choose v and w in $V(G)$ satisfying Equation (4.4) and Equation (4.5). By Equation (4.4), there is an i in $[p]$ such that $|N(v) \cap P_i| \neq |N(w) \cap P_i|$. Let $j_1, \ldots, j_\ell$ in $[q]$ such that $P_i = Q_{j_1} \cup \ldots \cup Q_{j_\ell}$. By Equation (4.5), for all k in $[\ell]$ we have

$$|N(v) \cap Q_{j_k}| = |N(w) \cap Q_{j_k}|.$$

As the Q_j are disjoint, this implies

$$|N(v) \cap P_i| = |N(w) \cap P_i|,$$

which is a contradiction. □

Hence, the two update rules are equivalent.

Corollary 4.3.2. For all G and all t in $\mathbb{N}$, we have $\tilde{C}^1_t \equiv C^1_t$.

Thus, we can use the update rule $\tilde{\Gamma}$ for the proof on unlabeled graphs. For the proof, it will be convenient to assume that $V(G) = [n]$ (although we still work with the notation $V(G)$), i.e., $n = |V|$. A vertex coloring $\mathbf{f^{(t)}}$ defines a matrix $\mathbf{F^{(t)}}$ in $\mathbb{R}^{n\times d}$ where the i-th row of $\mathbf{F^{(t)}}$ is defined by $\mathbf{f^{(t)}(i)}$ in $\mathbb{R}^{1\times d}$. Here we interpret i as a vertex from $V(G)$. As colorings and matrices can be interpreted as one another, given a matrix $\mathbf{F}$ in $\mathbb{R}^{V(G)\times d}$ we write $\mathit{\Gamma}_G(\mathbf{F})$ (or $\tilde{\mathit{\Gamma}}_G(\mathbf{F})$) for a Weisfeiler-Leman iteration on the coloring $c_{\mathbf{F}}$ induced by the matrix $\mathbf{F}$. For the GNN computation, we provide a matrix-based notation. Using the adjacency matrix $\mathbf{A}$ in $\mathbb{R}^{n\times n}$ of G and a coloring $\mathbf{F}$ in $\mathbb{R}^{n\times d}$, we can write the update rule of the GNN layer as

$$\mathbf{F^{t+1}} = \mathit{\Lambda}_{\mathbf{A},\mathbf{W},b}(\mathbf{F^{(t)}}) = \sigma(\mathbf{A}\mathbf{F^{(t)}}\mathbf{W^{(t)}} + b\mathbf{J}),$$

where $\mathit{\Lambda}_{\mathbf{A},\mathbf{W},b}$ is the refinement operator of GNNs corresponding to a single iteration of the 1-WL. For simplicity of the proof, we choose

$$\sigma(x) = \operatorname{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise,} \end{cases}$$

applied component-wise, and the bias as

$$b = -1.$$

Note that in [83], we provide a way to simulate the sign function using rectifier functions to indicate that choosing the sign function is not a hard restriction.

Lemma 4.3.3 ([83])**.** Let $\mathbf{B}$ in $\mathbb{Z}^{s\times t}$ be a matrix such that $0 \leq B_{ij} \leq n-1$ for all i and j, and the rows of $\mathbf{B}$ are pairwise distinct. Then there is a matrix $\mathbf{X}$ in $\mathbb{R}^{t\times s}$ such that the matrix $\operatorname{sign}\big(\mathbf{BX} - \mathbf{J}\big)$ in $\{-1,1\}^{s\times s}$ is non-singular.

Proof. Let $\mathbf{z} = (1, n, n^2, \ldots, n^{t-1})^T$ in $\mathbb{R}^t$ where n is the upper bound on the matrix entries of $\mathbf{B}$ and $\mathbf{b} = \mathbf{Bz}$ in $\mathbb{R}^s$. Then the entries of $\mathbf{b}$ are non-negative and pairwise distinct. Without loss of generality, we assume that $\mathbf{b} = (b_1, \ldots, b_s)^T$ such that $b_1 > b_2 > \cdots > b_s \geq 0$. We choose numbers $x_1, \ldots, x_s$ in $\mathbb{R}$ such that

$$\begin{cases} b_i \cdot x_j < 1 & \text{if } i \geq j, \\ b_i \cdot x_j > 1 & \text{if } i < j \end{cases} \tag{4.6}$$

for all i and j in $[s]$ as the b_i are ordered. Let $\mathbf{x} = (x_1, \ldots, x_s)$ in $\mathbb{R}^{1\times s}$ and $\mathbf{C} = \mathbf{b} \cdot \mathbf{x}$ in $\mathbb{R}^{s\times s}$ and $\hat{\mathbf{C}} = \operatorname{sign}(\mathbf{C} - \mathbf{J})$. Then $\mathbf{C}$ has entries $C_{ij} = b_i \cdot x_j$,

and thus, by Equation (4.6),

$$\hat{\mathbf{C}} = \begin{pmatrix} -1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ -1 & -1 & 1 & & \cdots & & 1 \\ & & & & & & \\ \vdots & & & \ddots & \ddots & & \vdots \\ & & & & & & \\ -1 & & & \cdots & & -1 & 1 \\ -1 & & & \cdots & & -1 & -1 \end{pmatrix} \tag{4.7}$$

Thus, $\hat{\mathbf{C}}$ is non-singular. We simply let $\mathbf{X} = \mathbf{z} \cdot \mathbf{x}$. Then $\mathbf{BX} = \mathbf{C}$. □

Let us call a matrix *row-independent modulo equality* if the set of all rows appearing in the matrix is linearly independent or equal. Note that the all-1 matrix $\mathbf{J}$ is row-independent modulo equality.

Lemma 4.3.4 ([83])**.** Let d in $\mathbb{N}$, and let $\mathbf{F}$ in $\mathbb{R}^{n\times d}$ be row independent modulo equality. Then there is a $\mathbf{W}$ in $\mathbb{R}^{d\times n}$ such that the matrix $\mathbf{\Lambda}_{\mathbf{A},\mathbf{W},-1}(\mathbf{F})$ is row independent modulo equality and

$$\mathbf{\Lambda}_{\mathbf{A},\mathbf{W},-1}(\mathbf{F}) \equiv \tilde{\Gamma}_G(\mathbf{F}).$$

Proof. Let $Q_1, \ldots, Q_r$ be the color classes of $\mathbf{F}$. That is, for all v and v' in $V(G)$ it holds that

$$\mathbf{F}_{\mathbf{v}} = \mathbf{F}_{\mathbf{v}'} \iff \exists j \in [r] \text{ such that } v \text{ and } v' \in Q_j.$$

Let $\tilde{\mathbf{F}}$ in $\mathbb{R}^{r\times d}$ be the matrix with rows $\tilde{\mathbf{F}}_j = \mathbf{F}_v$ for all j in $[r]$, and v in Q_j. Then the rows of $\tilde{\mathbf{F}}$ are linearly independent, and thus there is a matrix $\mathbf{M}$ in $\mathbb{R}^{d\times r}$ such that $\tilde{\mathbf{F}}\mathbf{M}$ is the $(r \times r)$ identity matrix. It follows that $\mathbf{FM}$ in $\mathbb{R}^{n\times r}$ is the matrix with entries

$$(\mathbf{FM})_{vj} = \begin{cases} 1 & \text{if } v \text{ in } Q_j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

Let $\mathbf{D}$ in $\mathbb{Z}^{n\times r}$ be the matrix with entries $D_{vj} = |N(v) \cap Q_j|$. Note that

$$\mathbf{AFM} = \mathbf{D}, \tag{4.9}$$

because for all v in V and j in $[t]$ we have

$$(\mathbf{AFM})_{vj} = \sum_{v'\in V(G)} \mathbf{A}_{vv'}(\mathbf{FM})_{v'j} = \sum_{v'\in Q_j} \mathbf{A}_{vv'} = \mathbf{D}_{vj},$$

where the second equality follows from Equation (4.8). By definition of $\mathit{\Gamma}_G$ as the 1-WL operator on uncolored graphs, we have

$$\mathit{\Gamma}_G(\mathbf{F}) \equiv \mathbf{D} \tag{4.10}$$

if we view $\mathbf{D}$ as a coloring of V.

Let $P_1, \ldots, P_s$ be the color classes of D, and let $\tilde{\mathbf{D}}$ in $\mathbb{Z}^{s\times r}$ be the matrix with rows $\tilde{\mathbf{D}}_i = \mathbf{D}_v$ for all i in $[s]$ and v in P_i. Then $0 \leq \tilde{D}_{ij} \leq n-1$ for all i and j, and the rows of $\tilde{\mathbf{D}}$ are pairwise distinct. By Lemma 4.3.3, there is a matrix $\mathbf{X}$ in $\mathbb{R}^{r\times s}$ such that the matrix

$$\operatorname{sign}(\tilde{\mathbf{D}}\mathbf{X} - \mathbf{J}) \text{ in } \mathbb{R}^{s\times s}$$

is non-singular. This implies that the matrix

$$\operatorname{sign}(\mathbf{AFMX} - \mathbf{J}) = \operatorname{sign}(\mathbf{DX} - \mathbf{J})$$

is row-independent modulo equality. Moreover,

$$\operatorname{sign}(\mathbf{AFMX} - \mathbf{J}) \equiv \mathbf{D} \equiv \mathit{\Gamma}_G(\mathbf{F})$$

by Equation (4.10). We let $\mathbf{W}$ in $\mathbb{R}^{p\times n}$ be the matrix obtained from $\mathbf{MX}$ in $\mathbb{R}^{p\times s}$ by adding $(n-s)$ all-0 columns. Then,

$$\mathit{\Lambda}_{\mathbf{A},\mathbf{W},-1}(\mathbf{F}) = \operatorname{sign}(\mathbf{AFW} - \mathbf{J})$$

is row-independent modulo equality and

$$\mathit{\Lambda}_{\mathbf{A},\mathbf{W},-1}(\mathbf{F}) \equiv \operatorname{sign}(\mathbf{AFMX} - \mathbf{J}) \equiv \tilde{\mathit{\Gamma}}_G(\mathbf{F}). \qquad \square$$

Hence, we get the following result.

Corollary 4.3.5 ([83])**.** There is a sequence $\mathcal{W} = (\mathbf{W}^{(\mathbf{t})})_{t\in\mathbb{N}}$ with $\mathbf{W}^{(\mathbf{t})}$ in $\mathbb{R}^{n\times n}$ such that for all t in $\mathbb{N}$,

$$\tilde{C}^1_t \equiv \mathit{\Lambda}^{(t)}_{\mathbf{A},\mathbf{W},-1},$$

where $\tilde{C}^1_t$ is given by the t-fold application of $\tilde{\mathit{\Gamma}}_G$ on the initial uniform coloring $\mathbf{J}$.

Remark 4.3.6. The construction in Lemma 4.3.4 always outputs a matrix with as many columns as there are color classes in the resulting coloring. Thus, we can choose d to be n and pad the matrix using additional 0-columns.

4.3.2. Colored graphs

We now extend the computation to colored or labeled graphs. To do that, we again use an equivalent but slightly different variant of the Weisfeiler-Leman update rule leading to the coloring $C^{1,0}_t$ instead of the usual C^1_t. We again show that both update rules are equivalent. We define Γ_G to be the refinement operator for the 1-WL, mapping a coloring $C^{1,0}_{(t-1)}$ to the updated one, $C^{1,0}_t$, as follows:

$$C^{1,0}_t(v) = \left(\Gamma_G\left(C^{1,0}_{(t-1)}\right)\right)(v) = \left(C^{1,0}_0(v), \{\!\{C^{1,0}_{(t-1)}(u) \mid u \in N(v)\}\!\}\right). \quad (4.11)$$

We use the initial coloring $C^{1,0}_0$ to make sure that any two vertices which have been assigned a different color in iteration t, get different colors in iteration $t' > t$. This is formalized by the following lemma.

Lemma 4.3.7. Let (G, l) be a colored graph, v and w in $V(G)$, and t in $\mathbb{N}$ such that $C^{1,0}_t(v) \neq C^{1,0}_t(w)$. Then $C^{1,0}_{t'}(v) \neq C^{1,0}_{t'}(w)$ for all $t' \geq t$.

Proof. Let t in $\mathbb{N}$ be minimal such that there are v and w with

$$C^{1,0}_t(v) \neq C^{1,0}_t(w) \quad (4.12)$$

and

$$C^{1,0}_{(t+1)}(v) = C^{1,0}_{(t+1)}(w). \quad (4.13)$$

Then $t \geq 1$, because by Equation (4.11),

$$C^{1,0}_1(v) = C^{1,0}_1(w)$$

implies

$$C^{1,0}_0(v) = C^{1,0}_0(w).$$

Let $P_1, \ldots, P_p$ be the color classes of $C^{1,0}_{(t-1)}$, and let $Q_1, \ldots, Q_q$ be the color classes of $C^{1,0}_t$. Observe that the partition $\{Q_1, \ldots, Q_q\}$ of $V(G)$ refines the partition $\{P_1, \ldots, P_p\}$. The argument is the same as in the proof of Lemma 4.3.1.

Choose v and w in $V(G)$ satisfying Equation (4.12) and Equation (4.13). By Equation (4.12), either

$$C^{1,0}_t(v) \neq C^{1,0}_t(w)$$

or there is an i in $[p]$ such that

$$|N_G(v) \cap P_i| \neq |N_G(w) \cap P_i|.$$

By Equation (4.11), $C^{1,0}_{(t+1)}(v) \neq C^{1,0}_{(t+1)}(w)$ contradicts Equation (4.13). Thus, $|N(v) \cap P_i| \neq |N_G(w) \cap P_i|$ for some i in $[p]$. Let $j_1, \ldots, j_\ell$ in $[q]$ such that $P_i = Q_{j_1} \cup \ldots \cup Q_{j_\ell}$. By Equation (4.12), for all k in $[\ell]$ we have

$$|N(v) \cap Q_{j_k}| = |N(w) \cap Q_{j_k}|.$$

As the Q_j are disjoint, this implies

$$|N_G(v) \cap P_i| = |N_G(w) \cap P_i|,$$

which is a contradiction. □

Hence, we get the following result.

Corollary 4.3.8. For all graphs G and initial vertex colorings l of G, we have $C^{1,0}_t \equiv C^1_t$ for all t in $\mathbb{N}$.

We consider the slightly modified update rule for 1-GNNs which takes the initial colors l into account. Let the matrix $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l},\mathbf{0}}$ in $\mathbb{R}^{n \times d}$ be an encoding, e.g., an one-hot encoding, of l such that $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l},\mathbf{0}}$ is linearly independent modulo equality. Then

$$\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l},\mathbf{0}} = \mathit{\Lambda}_{\mathbf{A},0,\mathbf{W},b}(\mathbf{F}^{(\mathbf{t})}_{\mathbf{l},\mathbf{0}}) = \Big[\mathbf{F}^{(\mathbf{0})}_{\mathbf{l},\mathbf{0}},\ \ \mathit{\Lambda}_{\mathbf{A},\mathbf{W},b}\Big(\mathit{\Lambda}_{\mathbf{A},0,\mathbf{W},b}(\mathbf{F}^{(\mathbf{t})}_{\mathbf{l},\mathbf{0}})\Big)\Big],$$

where square brackets denote column-wise matrix concatenation.

We show that this version of GNNs, which can be implemented by the GNN of Equation (4.2), is equivalent to the 1-WL on colored graphs, proving the theorem for colored graphs.

Proof of Theorem 4.2.2. We prove the Theorem by induction over the number of iterations t. The initial colorings are chosen consistent with l. Hence,

$$C^{1,0}_0 \equiv \mathbf{F}^{(\mathbf{0})}_{\mathbf{l},\mathbf{0}}.$$

For the induction step we assume

$$C^{1,0}_t \equiv \mathbf{F}^{(\mathbf{t})}_{\mathbf{l},\mathbf{0}}$$

for iteration t. We know by Lemma 4.3.4 that the inner part of the update rule $\mathit{\Lambda}_{\mathbf{A},\mathbf{W},b}\Big(\mathit{\Lambda}_{\mathbf{A},0,\mathbf{W},\mathbf{b}}(\mathbf{F}^{(\mathbf{t})}_{\mathbf{l},\mathbf{0}})\Big)$ results in color classes which are identical to the ones that $\tilde{\Gamma}_G$ would produce. This implies that restricted to each color class Q from $Q^t_1, \ldots, Q^t_q$ of iteration t, the new color classes $Q^{(t+1)}_1 \cap Q, \ldots, Q^{(t+1)}_{q'} \cap Q$ restricted to Q match the coloring $C^{1,0}_{(t+1)}|_Q$, that is, $C^{1,0}_{(t+1)}$ restricted to Q.

This holds as within one color class, the common color of the vertices contains no further information as shown in Lemma 4.3.1. Observe that colors are represented by linearly independent row vectors. This especially holds for $\mathbf{F}^{(\mathbf{t+1})} = \mathit{\Lambda}_{\mathbf{A},\mathbf{W},b}\Big(\mathit{\Lambda}_{\mathbf{A},0,\mathbf{W},b}(\mathbf{F}^{(\mathbf{t})}_{\mathbf{l,0}})\Big)$. In order to show that $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}$ represents the coloring $C^{1,0}_{(t+1)}$, we have to prove two properties.

1. Given $Q^{(0)}_i \neq Q^{(0)}_j$ and u in $Q^{(0)}_i, v$ in $Q^{(0)}_j$, we have $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}(\mathbf{u}) \neq \mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}(\mathbf{v})$.

2. $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}$ is linearly independent modulo equality.

In the first item, we interpret $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}$ as a coloring function. Both properties follow directly from the definition of $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}$ (the concatenation of the matrices $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l,0}}$ and $\mathbf{F}^{(\mathbf{t+1})}$). The first property holds as the row vectors of $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l,0}}$ are clearly different, as otherwise $Q_i = Q_j$. The second property follows from the fact that linear independence cannot be lost by extending a matrix. Thus, within each old color class $Q^{(t)}_i$, all vectors are linearly independent modulo equality as $\mathbf{F}^{(\mathbf{t+1})}$ (without subscript) is linearly independent modulo equality. This then extends to all combinations of old color classes and colors from $\mathbf{F}^{(\mathbf{t+1})}$ as $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l,0}}$ is also linearly independent modulo equality. By induction, this shows that $c^{(t)}_{l,0} \equiv \mathbf{F}^{(\mathbf{t})}_{\mathbf{l,0}}$ for all t. Note that the width of the matrices $\mathbf{F}^{(\mathbf{t+1})}_{\mathbf{l,0}}$ can be bounded by $2n$. The width of $\mathbf{F}^{(\mathbf{0})}_{\mathbf{l,0}}$ can trivially be bounded by n (in the worst case every vertex has a different initial color). The width of $\mathbf{F}^{(\mathbf{t+1})}$ can also be bounded by n. Using Corollary 4.3.8 for the step from $C^{1,0}_{(t+1)}$ to $C^{1}_{(t+1)}$ finishes the proof. □

4.3.3. Shortcomings of both approaches

The power of the 1-WL has been completely characterized, see, e.g., [5, 55]. Hence, by using Theorems 4.2.1 and 4.2.2, this characterization is also applicable to 1-GNNs. Consequently, 1-GNNs have the same shortcomings as the 1-WL. For example, both methods will give the same color to every vertex in a graph consisting of a triangle and a 4-cycle, although vertices from the triangle and the vertices from the 4-cycle are clearly different. Moreover, they are not capable of capturing simple graph theoretic properties, e.g., triangle counts, which are an important measure in social network analysis [78, 86].

4.4. The k-dimensional graph neural network architecture

In the following, we propose a generalization of 1-GNNs, so-called k-GNNs, which are inspired on the k-WL. Due to scalability and limited GPU memory, we consider a set-based version of the k-WL. For a given k, we consider all k-element subsets $[V(G)]^k$ over $V(G)$. Let $s = \{s_1, \ldots, s_k\}$ be a *k-set* in $[V(G)]^k$, then we define the *neighborhood* of s as

$$N(s) = \{t \in [V(G)]^k \mid |s \cap t| = k - 1\}.$$

The *local neighborhood* $N_L(s)$, similarly to Definition in Section 3.3.1, consists of all t in $N(s)$ such that (v, w) in $E(G)$ for the unique v in $s \setminus t$ and the unique w in $t \setminus s$. The *global neighborhood* $N_G(s)$ then is defined as $N(s) \setminus N_{\mathrm{L}}(s)$.

The set-based k-WL works analogously to the 1-WL, i.e., it computes a coloring $c^{(t)}_{\mathrm{s},k,l} \colon [V(G)]^k \to \mathbb{S}$ as in Equation (2.1) based on the above neighborhood. That is, two k-sets that have the same color in iteration t get the same color in iteration $(t+1)$ if their local and global neighborhood is colored the same. Initially, $c^{(0)}_{\mathrm{s},k,l}$ colors each element s in $[V(G)]^k$ with the isomorphism type of $G[s]$.

Let (G, l) be a labeled graph. In each k-GNN layer $t \geq 0$, we compute a feature vector $\mathbf{f}^{(\mathbf{t})}_{\mathbf{k}}(\mathbf{s})$ for each k-set s in $[V(G)]^k$. For $t = 0$, we set $\mathbf{f}^{(\mathbf{0})}_{\mathbf{k}}(\mathbf{s})$ to $\mathbf{f}^{\mathrm{iso}}(\mathbf{s})$, a one-hot encoding of the isomorphism type of $G[s]$ labeled by l. In each layer $t > 0$, we compute new features by

$$\mathbf{f}^{(\mathbf{t})}_{\mathbf{k}}(\mathbf{s}) = \sigma\Big(\mathbf{f}^{(\mathbf{t-1})}_{\mathbf{k}}(\mathbf{s}) \cdot \mathbf{W}^{(\mathbf{t})}_{\mathbf{1}} + \sum_{u \in N_L(s) \cup N_G(s)} \mathbf{f}^{(\mathbf{t-1})}_{\mathbf{k}}(\mathbf{u}) \cdot \mathbf{W}^{(\mathbf{t})}_{\mathbf{2}}\Big).$$

Moreover, one could split the sum into two sums ranging over $N_L(s)$ and $N_G(s)$, respectively, using distinct parameter matrices to enable the model to learn the importance of local and global neighborhoods. To scale k-GNNs to larger datasets and to prevent overfitting, we propose *local* k-GNNs, where we omit the global neighborhood of s, i.e.,

$$\mathbf{f}^{(\mathbf{t})}_{\mathbf{k},\mathrm{L}}(\mathbf{s}) = \sigma\Big(\mathbf{f}^{(\mathbf{t-1})}_{\mathbf{k},\mathrm{L}}(\mathbf{s}) \cdot \mathbf{W}^{(\mathbf{t})}_{\mathbf{1}} + \sum_{u \in N_L(s)} \mathbf{f}^{(\mathbf{t-1})}_{\mathbf{k},\mathrm{L}}(\mathbf{u}) \cdot \mathbf{W}^{(\mathbf{t})}_{\mathbf{2}}\Big).$$

The running time for the evaluation of the above depends on $|V|$, k, and the sparsity of the graph (each iteration can be bounded by the number of subsets of size k times the maximum degree). Note that we can scale our method to larger datasets by using the sampling strategies introduced in Section 3.3.4. We can now lift the results of the previous section to the k-dimensional case.

Proposition 4.4.1. Let (G, l) be a labeled graph and let $k \geq 2$. Then for all $t \geq 0$, for all choices of initial colorings $\mathbf{f}_{\mathbf{k}}^{(\mathbf{0})}$ consistent with l, and for all weights $\mathcal{W}^{(t)}$,

$$c_{\mathrm{s},k,l}^{(t)} \sqsubseteq \mathbf{f}_{\mathbf{k}}^{(\mathbf{t})}.$$

Proof. The proof follows the arguments in the proof of Theorem 4.2.1. We therefore only provide a brief proof by induction on the iteration t. For the base case, i.e., iteration $t = 0$, the statement holds because the initial coloring $\mathbf{f}_{\mathbf{k}}^{(\mathbf{0})}$ is chosen to be consistent with the isomorphism types.

For the inductive step, assume that the statement holds until iteration $(t-1)$. Consider two k-sets u and v which (i) are not distinguished in the first $(t-1)$ iterations and (ii) are not distinguished in the t-th iteration. Therefore, u and v must have an equal number of neighbors from every color class. This implies that the k-GNN update rule yields the same output for u and v. Hence, if two such k-sets are distinguished by the k-GNN in the t-th iteration, they must be distinguished by the set-based k-WL as well. This finishes the induction and proves the proposition. □

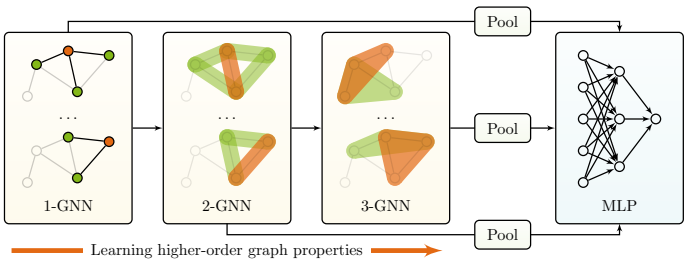
Again the second result states that there exists a suitable initialization of the parameter matrices $\mathcal{W}^{(t)}$ such that k-GNNs have exactly the same power in terms of distinguishing non-isomorphic (sub-)graphs as the set-based k-WL.

Proposition 4.4.2. Let (G, l) be a labeled graph and let $k \geq 2$. Then for all $t \geq 0$ there exists a sequence of weights $\mathcal{W}^{(t)}$, and a k-GNN architecture such that

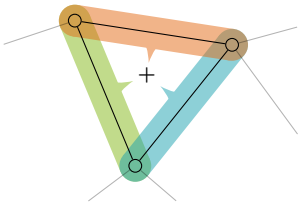
$$c_{\mathrm{s},k,l}^{(t)} \equiv \mathbf{f}_{\mathbf{k}}^{(\mathbf{t})}.$$

Proof. We simulate the set-based k-WL on an n-vertex graph G via a 1-WL on a graph $G^{\otimes k}$ on $O(n^k)$ vertices, defined as follows. The vertex set of $G^{\otimes k}$ is the set $[V(G)]^k$ of all k-element subsets of $V(G)$. The edge set of $G^{\otimes k}$ is defined as follows: two sets s and t are connected by an edge in $G^{\otimes k}$ if and only if $|s \cap t| = k - 1$. Observe that the neighborhood of a vertex s in this graph is exactly the set $N(s)$ defined earlier. The initial labeling of the vertices of the graph $G^{\otimes k}$ is determined as follows: For s in $V(G^{\otimes k})$ the initial label of s is its isomorphism type.

For the above construction, it immediately follows that performing the 1-WL on the graph $G^{\otimes k}$ yields the same coloring, as the one obtained by performing k-WL for the graph G. It remains to define the sequence $(\boldsymbol{W}^t)_{t>0}$ such that k-GNN simulates the set-based k-WL on G. Applying Theorem 4.2.2 to the graph $G^{\otimes k}$ results in a sequence $\widetilde{\boldsymbol{W}}^t$ such that the 1-GNN can simulate



(a) Hierarchical 1-2-3-GNN network architecture.



(b) Pooling from 2- to 3-GNN.

Figure 4.1.: Illustration of the proposed hierarchical variant of the k-GNN layer. For each subgraph S on k vertices a feature is learned, which is initialized with the learned features of all $(k-1)$-element subgraphs of S. Hence, a hierarchical representation of the input graph is learned.

the 1-WL on $G^{\otimes k}$ using $\widetilde{\boldsymbol{W}}^t$. Hence, this sequence can be directly used in the k-GNN to simulate k-WL on G. □

4.4.1. Hierarchical variant

One key benefit of the end-to-end trainable k-GNN framework—compared to the combinatorial (set-based) k-WL algorithm—is that we can hierarchically combine representations learned at different granularities. Concretely, rather than simply using one-hot indicator vectors as initial feature inputs in a k-GNN, we propose a *hierarchical* variant of k-GNNs that uses the features learned by a $(k-1)$-dimensional GNN, in addition to the (labeled) isomorphism type, as the initial features, i.e.,

$$\mathbf{f}_{\mathbf{k}}^{(\mathbf{0})}(\mathbf{s}) = \sigma\left(\left[\mathbf{f}^{\text{iso}}(\mathbf{s}), \sum_{u \subset s} \mathbf{f}_{\mathbf{k-1}}^{(\mathbf{T_{k-1}})}(\mathbf{u})\right] \cdot \mathbf{W_{k-1}}\right),$$

for some $T_{k-1} > 0$, where $|u| = k-1$, $\mathbf{W_{k-1}}$ is a matrix of appropriate size, and square brackets denote column-wise matrix concatenation.

Hence, the features are recursively learned from dimensions 1 to k in an end-to-end fashion. See Figure 4.1 for an illustration. This hierarchical model also satisfies Propositions 4.4.1 and 4.4.2, so its representational capacity is theoretically equivalent to a standard k-GNN (in terms of its relationship to the set-based k-WL). Nonetheless, hierarchy is a natural inductive bias for graph modeling, since many real-world graphs incorporate hierarchical structure, so we expect this hierarchical formulation to offer empirical utility.

4.5. Experimental study

In the following, we want to investigate potential benefits of GNNs over graph kernels as well as the benefits of our proposed k-GNN architectures over 1-GNN architectures. More precisely, we address the following questions:

Q1 How do the (hierarchical) k-GNNs perform in comparison to state-of-the-art graph kernels?

Q2 How do the (hierarchical) k-GNNs perform in comparison to the 1-GNN in graph classification and regression tasks?

Q3 How much (if any) improvement is provided by optimizing the parameters of the GNN aggregation function, compared to just using random GNN parameters while optimizing the parameters of the downstream classification/regression algorithm?

4.5.1. Datasets

To compare our k-GNN architectures to kernel approaches we used well-established benchmark datasets from the graph kernel literature [53]. The vertices of each graph in these datasets is annotated with (discrete) labels or no labels. See Table 4.1 for statistics and properties, and see Appendix B for detailed descriptions.

Table 4.1.: Dataset statistics and properties.

Dataset	Properties				
	Number of graphs	Number of classes	∅ Number of vertices	∅ Number of edges	Vertex labels
PROTEINS	1113	2	39.1	72.8	✓
IMDB-BINARY	1000	2	19.8	96.5	✗
IMDB-MULTI	1500	3	13.0	65.9	✗
PTC_FM	349	2	14.1	14.5	✓
NCI1	4110	2	29.9	32.3	✓
MUTAG	188	2	17.9	19.8	✓
PTC_MR	344	2	14.3	14.7	✓

To demonstrate that our architectures scale to larger datasets and offer benefits on real-world applications, we conducted experiments on the QM9 dataset [96, 100, 119], which consists of 133 385 small molecules. The aim here is to perform regression on twelve targets representing energetic, electronic,

Table 4.2.: Classification accuracies in percent on various graph benchmark datasets.

	Method	Dataset						
		PROTEINS	IMDB-BIN.	IMDB-MUL.	PTC_FM	NCI1	MUTAG	PTC_MR
Kernel	GRAPHLET	72.9	59.4	40.8	58.3	72.1	87.7	54.7
	SHORTEST-PATH	**76.4**	59.2	40.5	62.1	74.5	81.7	58.9
	1-WL	73.8	72.5	**51.5**	62.9	83.1	78.3	61.3
	2-WL	75.2	72.6	50.6	**64.7**	77.0	77.0	61.9
	3-WL	74.7	73.5	49.7	61.5	83.1	83.2	62.5
	WL-OA	75.3	73.1	50.4	62.7	**86.1**	84.5	**63.6**
GNN	DCNN	61.3	49.1	33.5	—	62.6	67.0	56.6
	PATCHYSAN	75.9	71.0	45.2	—	78.6	**92.6**	60.0
	DGCNN	75.5	70.0	47.8	—	74.4	85.8	58.6
	1-GNN NO TUNING	70.7	69.4	47.3	59.0	58.6	82.7	51.2
	1-GNN	72.2	71.2	47.7	59.3	74.3	82.2	59.0
	1-2-3-GNN NO TUNING	75.9	70.3	48.8	60.0	67.4	84.4	59.3
	1-2-3-GNN	75.5	**74.2**	49.5	62.8	76.2	86.1	60.9

geometric, and thermodynamic properties, which were computed using density functional theory.[2]

4.5.2. Baselines

We used the following kernel and GNN methods as baselines for our experiments.

Kernel baselines We used the Graphlet kernel [105], the shortest-path kernel [8], the Weisfeiler-Lehman subtree kernel (WL) [106], the Weisfeiler-Lehman Optimal Assignment kernel (WL-OA) [62], and the global-local k-WL [80] with k in $\{2, 3\}$ as kernel baselines. For each kernel, we computed the normalized Gram matrix. We used the C-SVM implementation of LIBSVM [14] to compute the classification accuracies using 10-fold cross validation. The parameter C was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$ by 10-fold cross validation on the training folds.

Neural baselines To compare GNNs to kernels, we used the basic 1-GNN layer of Equation (4.1), DCNN [114], PatchySan [87], and DGCNN [128]. For the QM9 dataset we used a 1-GNN layer similar to [30], where we replaced the inner sum of Equation (4.1) with a 2-layer MLP in

[2]The dataset was obtained from `http://moleculenet.ai/datasets-1`.

Table 4.3.: Mean absolute errors on the QM9 dataset. The far-right column shows the improvement of the best k-GNN model in comparison to the 1-GNN baseline.

Target	Method						
	DTNN [119]	MPNN [119]	1-GNN	1-2-GNN	1-3-GNN	1-2-3-GNN	Gain
μ	**0.244**	0.358	0.493	0.493	**0.473**	0.476	4.0%
α	0.95	0.89	0.78	**0.27**	0.46	**0.27**	65.3%
$\varepsilon_{\text{HOMO}}$	0.00388	0.00541	**0.00321**	0.00331	0.00328	0.00337	–
$\varepsilon_{\text{LUMO}}$	0.00512	0.00623	0.00355	**0.00350**	0.00354	0.00351	1.4%
$\Delta\varepsilon$	0.0112	0.0066	0.0049	0.0047	**0.0046**	0.0048	6.1%
$\langle R^2 \rangle$	**17.0**	28.5	34.1	21.5	25.8	22.9	37.0%
ZPVE	0.00172	0.00216	0.00124	**0.00018**	0.00064	0.00019	85.5%
U_0	2.43	2.05	2.32	**0.0357**	0.6855	0.0427	98.5%
U	2.43	2.00	2.08	**0.107**	0.686	0.111	94.9%
H	2.43	2.02	2.23	0.070	0.794	**0.0419**	98.1%
G	2.43	2.02	1.94	0.140	0.587	**0.0469**	97.6%
C_{v}	0.27	0.42	0.27	0.0989	0.158	**0.0944**	65.0%

order to incorporate edge features (bond type and distance information). Moreover, we compared against the numbers provided in [119].

4.5.3. Model configuration

We always used three layers for 1-GNNs, and two layers for (local) 2-GNNs and (local) 3-GNNs, all with a hidden-dimension size of 64. For the hierarchical variant we used architectures that use features computed by 1-GNNs as initial features for the 2-GNNs (1-2-GNNs) and 3-GNNs (1-3-GNNs), respectively. Moreover, using the combination of the former, we componentwise concatenated the computed features of the 1-2-GNNs and the 1-3-GNNs (1-2-3-GNNs). For the final classification and regression steps, we used a three layer MLP, with binary cross entropy and mean squared error for the optimization, respectively. For classification, we used a dropout layer with $p = 0.5$ after the first layer of the MLP. We applied global average pooling to generate a vector representation of the graph from the computed vertex features for each k. The resulting vectors are concatenated column-wise before feeding them into the MLP. Moreover, we used the Adam optimizer with an initial learning rate of 10^{-2} and applied an adaptive learning rate

decay based on validation results to a minimum of 10^{-5}. We trained the classification networks for 100 epochs and the regression networks for 200 epochs.

4.5.4. Experimental protocol

For the smaller datasets, which we used for comparison against the kernel methods, we performed a 10-fold cross validation where we randomly sampled 10% of each training fold to act as a validation set. For the QM9 dataset, we followed the dataset splits described in [119]. We randomly sampled 10% of the examples for validation, another 10% for testing, and used the remaining for training. We used the same initial vertex features as described in [30]. Moreover, to illustrate the benefits of our hierarchical k-GNN architecture, we did not use a complete graph, where edges are annotated with pairwise distances, as input. Instead, we only used pairwise Euclidean distances for connected vertices, computed from the provided vertex coordinates.

All experiments were conducted on a workstation with an Intel i7-6850K with 4.00GHz and 128GB of RAM running Ubuntu 18.04.2 LTS using an Nvdia Gefore GTX 1080 Ti with 11GB of memory.[3]

4.5.5. Results and discussion

In the following, we answer questions **Q1** to **Q3**. Table 4.2 shows the results for comparison with the kernel methods on the graph classification benchmark datasets. Here, the hierarchical k-GNN is on par with the kernels despite the small dataset sizes (answering question **Q1**). We also find that the 1-2-3-GNN significantly outperforms the 1-GNN on all seven datasets (answering **Q2**), with the 1-GNN being the overall weakest method across all tasks. We can further see that optimizing the parameters of the aggregation function only leads to slight performance gains on four out of seven datasets (for the 1-2-3-GNN), and that no optimization even achieves better results on the PROTEINS benchmark dataset (answering **Q3**). A similar result is obtained for the 1-GNN. We contribute this effect to the one-hot encoded vertex labels, which allow the GNN to gather enough information out of from the neighborhood of a vertex, even when this aggregation is not learned.

Table 4.3 shows the results for the QM9 dataset. On eleven out of twelve targets, all of our hierarchical variants beat the 1-GNN baseline, providing

[3]The code was built upon the work of [26], and is provided at `https://github.com/chrsmrrs/k-gnn`.

further evidence for **Q2**. For example, on the target H we achieve a large improvement of 98.1% in MAE compared to the baseline. Moreover, on ten out of twelve datasets, the hierarchical k-GNNs beat the baselines from [119]. However, the additional structural information extracted by the k-GNN layers does not serve all tasks equally well, leading to huge differences in gains across the targets. For example, for the $\varepsilon_{\text{LUMO}}$ target, the best k-GNN layer only achieves a gain of 1.4% over the 1-GNN baseline.

It should be noted that our k-GNN models have more parameters than the 1-GNN model, since we stack two additional GNN layers for each k. However, extending the 1-GNN model by additional layers to match the number of parameters of the k-GNN did not lead to better results in any experiment.

4.6. Conclusion

We presented a theoretical investigation of GNNs, showing that a wide class of GNN architectures cannot be stronger than the 1-WL. On the positive side, we showed that, in principle, GNNs possess the same power in terms of distinguishing between non-isomorphic (sub-)graphs, while having the added benefit of adapting to the given data distribution. Based on this insight, we proposed k-GNNs which are a generalization of GNNs based on the k-WL. This new model is strictly stronger than GNNs in terms of distinguishing non-isomorphic (sub-)graphs and is capable of distinguishing more graph properties. Moreover, we devised a hierarchical variant of k-GNNs, which can exploit the hierarchical organization of most real-world graphs. Our experimental study shows that k-GNNs consistently outperform 1-GNNs and beat state-of-the-art neural architectures on large-scale molecule learning tasks. Future work includes designing task-specific k-GNNs, e.g., devising k-GNN layers that exploit expert-knowledge in bio- and cheminformatic settings. For example, neural architectures for molecules need to incorporate graph structure at different levels of granularity (atom level as well as on the level of functional groups). Hence, designing k-GNN architectures that take into account domain specific substructures would be an interesting problem for future work.

Chapter 5.

Conclusion

In this work, we proposed several methods for (supervised) graph classification. The first part of this thesis dealt with kernel-based graph classification. We showed how to deal with graphs with continuous labels attached to vertices and edges in a scalable manner, which is an important requirement for various applications, e.g., in chemoinformatic settings atoms are annotated with physical and chemical measurements. Prior proposed graph kernels that could handle this kind of input did not scale to large-scale datasets because they compared each pair of graphs, resulting in a quadratic overhead. By introducing special hash functions we showed how to (provable) approximate implicit kernels for graphs with continuous labels by explicit, finite-dimensional feature vectors. Our experimental study showed that our new approach achieves state-of-the-art results while being orders of magnitude faster than its competitors, validating our theoretical results. Our techniques can also be incorporated into well-known similarity measures for molecules such as extended connectivity fingerprints [99] and enable them to make use of continuous measurements.

As most graph kernels rely on comparing graphs by their local structure, we investigated how to incorporate more global or higher-order structures into graph kernels. To this end, we derived graph kernels based on variants of the k-WL, which is a provable more powerful extension of the 1-WL. The 1-WL has shown good performance in machine learning settings. However, it misses important structures, e.g., distinguishing cycles of different lengths, which is an important measure in social network and chemoinformatic settings. Since the ordinary k-WL does not take the local structure into account and is prone to overfitting, we derived a local variant of the k-WL. This variant takes the sparsity of the underlying graph into account, and hence each iteration of the algorithm can be computed much faster. Moreover, we showed that a variant of the local algorithm has at least the same power as the k-WL. To scale up the kernel to large-scale datasets we derived a stochastic variant of our

algorithm, and showed that it can approximate the exact kernel in constant time for bounded degree graphs. Our experimental study showed that our proposed kernels often perform better than the 1-WL and other baselines.

To conclude the first part, we developed a theoretical framework to better understand the expressivity of well-known graph kernels. We showed that they are not able to distinguish simple graph properties, and proposed a more powerful kernel based on frequency counts of isomorphism types of k-discs. Finally, we studied a kernel nearest neighbor classifier for graphs and showed that its prediction error can be bounded under the assumption that the employed kernel can distinguish the class label property.

The second part of this work dealt with neural approaches for graph classification. Specifically, we investigated GNNs and showed that the 1-WL is an upper-bound for the former in terms of its ability to distinguish non-isomorphic (sub-)graphs. That is, we showed that any GNN architecture with any possible parameter setting, cannot be more powerful than the 1-WL. Hence, we showed the limits of GNNs. Moreover, we showed that there exists a GNN architecture that has the same power as the 1-WL. Based on these insights, we proposed k-GNNs which are strictly more powerful than GNNs. Since many real-world graphs exhibit a hierarchical structure, we introduced a (hierarchical) variant of k-GNNs, which can capture these hierarchies. Our experimental study showed that this variant (often) beats baseline approaches on large-scale molecule learning tasks. We believe that our new architecture is a stepping stone for deriving new more powerful neural architectures for the graph domain.

5.1. Directions for future work and open problems

Since most work in supervised graph classification is conducted from an empirical point of view, we believe that the theoretical principled design of new methods is a promising direction, e.g., extending the methods developed in Section 3.4 to handle more realistic learning settings.

For GNNs, we believe that more work should be conducted in the area of interpretability, i.e., making sense of the learned weights, and injecting expert knowledge, e.g., chemical or physical knowledge. Moreover, we believe that there should be a better understanding of what GNNs learn in the presence of continuous vertex and edge labels.

For example, we propose the following open problems:

P1 Derive a better understanding of what GNNs, optimized with variants of the stochastic gradient descent algorithm, learn.

P2 Derive a better understanding of why or when Weisfeiler-Leman type kernels (and hence GNNs) work on real-world datasets. Subsequently, derive a theoretical understanding.

P3 Develop a better understanding of the trade-offs between sparsity, the number of iterations, and k for the k-WL.

P4 Design a k-GNN layer that takes physical and chemical expert knowledge into account.

P5 Develop a theory of generalization for graph-structured data.

Appendix A.

Full list of publications

1. C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. "Faster Kernel for Graphs with Continuous Attributes via Hashing." In: *IEEE International Conference on Data Mining*. IEEE, 2016, pp. 1095–1100

2. F. Bökler, M. Ehrgott, C. Morris, and P. Mutzel. "Output-sensitive complexity of multiobjective combinatorial optimization." In: *Journal of Multi-Criteria Decision Analysis* 24.1-2 (2017). Wiley, pp. 25–36

3. C. Morris, K. Kersting, and P. Mutzel. "Glocalized Weisfeiler-Lehman Kernels: Global-Local Feature Maps of Graphs." In: *IEEE International Conference on Data Mining*. IEEE, 2017, pp. 327–336

4. N. M. Kriege and C. Morris. "Recent Advances in Kernel-Based Graph Classification." In: *The European Conference on Machine Learning & Principles and Practice of Knowledge Discovery In Databases*. Springer, 2017, pp. 388–392

5. N. M. Kriege, M. Neumann, C. Morris, K. Kersting, and P. Mutzel. "A Unifying View of Explicit and Implicit Feature Maps for Structured Data: Systematic Studies of Graph Kernels." In: *CoRR* abs/1703.00676 (2017). Accepted for publication in *Data Mining and Knowledge Discovery*

6. N. M. Kriege, C. Morris, A. Rey, and C. Sohler. "A Property Testing Framework for the Theoretical Expressivity of Graph Kernels." In: *International Joint Conference on Artificial Intelligence*. IJCAI, 2018, pp. 2348–2354

7. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, Jan Eric Lenssen, G. Rattan, and M. Grohe. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks." In: *AAAI Conference on Artificial Intelligence*. AAAI. 2019, pp. 4602–4609

8. R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. "Hierarchical Graph Representation Learning with Differentiable Pooling." In: *Advances in Neural Information Processing Systems*. NIPS, 2018, pp. 4800–4810

9. N. M. Kriege, F. D. Johansson, and C. Morris. "A Survey on Graph Kernels." In: *CoRR* abs/1903.11835 (2019). Accepted for publication in *Applied Network Science*

10. C. Morris and P. Mutzel. "Towards a practical k-dimensional Weisfeiler-Leman algorithm." In: *CoRR* abs/1904.01543 (2019)

Appendix B.

Datasets

Table B.1.: Dataset statistics and properties.

Dataset	Properties				
	Number of graphs	Number of classes	∅ Number of vertices	∅ Number of edges	Vertex labels
Enzymes	600	6	32.6	62.1	✓
IMDB-Binary	1 000	2	19.8	96.5	✗
IMDB-Multi	1 500	3	13.0	65.9	✗
Mutag	188	2	17.9	19.8	✓
NCI1	4 110	2	29.9	32.3	✓
NCI109	4 127	2	29.7	32.1	✓
PTC_FM	349	2	14.1	14.5	✓
PTC_MR	344	2	14.3	14.7	✓
Proteins	1 113	2	39.1	72.8	✓
Reddit-Binary	2 000	2	429.6	497.8	✗
Reddit-Binary-5k	4 999	5	508.5	594.9	✗

In the following, we describe the datasets used in the experimental evaluations of Section 3.3 and Chapter 4.

Enzymes and Proteins contain graphs representing proteins according to the graph model of [8]. Each vertex is annotated with a discrete label and a continuous label containing physical or chemical measurements of dimension 18 and 1, respectively. Note that we ignored the continuous labels for the experiments in Section 3.3 and Chapter 4. The datasets are subdivided into six and two classes, respectively. Note that this is the same dataset used in [25], which does not contain all the annotations described and used in [8].

IMDB-Binary and IMDB-Multi are movie collaboration datasets first used in [122] based on data from IMDB[1]. Each vertex represents an actor or an actress, and there exists an edge between two vertices if the

[1] `https://www.imdb.com/`

corresponding actor or actress appears in the same movie. Each graph represents an ego network of an actor or actress. The vertices are unlabeled and each dataset is divided into two (three) classes corresponding to movies genres.

Mutag is a dataset consisting of mutagenetic aromatic and heteroaromatic nitro compounds [21, 60] with seven discrete vertex labels.

NCI1 and NCI109 are (balanced) subsets of datasets made available by the National Cancer Institute[2] [113, 106], consisting of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively. The vertices are annotated with discrete labels.

PTC_FM and PTC_MR are datasets from the Predicte Toxicology Challenge (PTC)[3] containing chemical compounds labeled according to carcinogenicity on female mice (FM) or male rats (MR). The vertices are annotated with discrete labels. Both datasets are divided into two classes.

Reddit-Binary and Reddit-Binary-5k are social network datasets based on data from the content-aggregation website Reddit[4] [122]. Each vertex represents a user and two vertices are connected by an edge if one user responded to the other user's comment. The vertices are unlabeled and the datasets are divided into two (five) classes representing different communities.

[2] `https://www.cancer.gov/`
[3] `https://www.predictive-toxicology.org/ptc/`
[4] `https://www.reddit.com/`

Bibliography

[1] G. W. Adamson and J. A. Bush. "A method for the automatic classification of chemical structures." In: *Information Storage and Retrieval* 9.10 (1973). Elsevier, pp. 561 –568.

[2] N. K. Ahmed, T. Willke, and R. A. Rossi. "Estimation of Local Subgraph Counts." In: *IEEE International Conference on Big Data*. IEEE, 2016, pp. 1–10.

[3] F. Aiolli, M. Donini, N. Navarin, and A. Sperduti. "Multiple Graph-Kernel Learning." In: *IEEE Symposium Series on Computational Intelligence*. IEEE, 2015, pp. 1607–1614.

[4] A. Andoni. "Nearest Neighbor Search: The Old, the New, and the Impossible." Ph.D. thesis. MIT, 2009.

[5] V. Arvind, J. Köbler, G. Rattan, and O. Verbitsky. "On the Power of Color Refinement." In: *International Symposium on Fundamentals of Computation Theory*. Vol. 9210. Lecture Notes in Computer Science. Springer, 2015, pp. 339–350.

[6] L. Babai and L. Kucera. "Canonical Labelling of Graphs in Linear Average Time." In: *Symposium on Foundations of Computer Science*. IEEE, 1979, pp. 39–46.

[7] I. Benjamini, O. Schramm, and A. Shapira. "Every Minor-Closed Property of Sparse Graphs is Testable." In: *Advances in Mathematics* 223.6 (2010). Elsevier, pp. 2200–2218.

[8] K. M. Borgwardt and H.-P. Kriegel. "Shortest-path kernels on graphs." In: *IEEE International Conference on Data Mining*. IEEE, 2005, pp. 74–81.

[9] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. "Counting Graphlets: Space vs Time." In: *ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 557–566.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. "Spectral Networks and Deep Locally Connected Networks on Graphs." In: *International Conference on Learning Representation*. 2014.

[11] R. G. Busacker and T. L. Saaty. *Finite graphs and networks: an introduction with applications*. McGraw-Hill, 1965.

[12] F. Bökler, M. Ehrgott, C. Morris, and P. Mutzel. "Output-sensitive complexity of multiobjective combinatorial optimization." In: *Journal of Multi-Criteria Decision Analysis* 24.1-2 (2017). Wiley, pp. 25–36.

[13] J. Cai, M. Fürer, and N. Immerman. "An optimal lower bound on the number of variables for graph identifications." In: *Combinatorica* 12.4 (1992). Springer, pp. 389–410.

[14] C.-C. Chang and C.-J. Lin. "LIBSVM: A library for support vector machines." In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). ACM, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 27:1–27:27.

[15] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. "Coordinate Descent Method for Large-scale L2-loss Linear Support Vector Machines." In: *Journal of Machine Learning Research* 9 (2008). JMLR, pp. 1369–1398.

[16] X. Chen, Y. Li, P. Wang, and J. C. S. Lui. "A General Framework for Estimating Graphlet Statistics via Random Walk." In: *VLDB Endowment* 10.3 (Nov. 2016). VLDB, pp. 253–264.

[17] F. Costa and K. De Grave. "Fast Neighborhood Subgraph Pairwise Distance Kernel." In: *International Conference on Machine Learning*. Omnipress, 2010, pp. 255–262.

[18] G. Da San Martino, N. Navarin, and A. Sperduti. "A memory efficient graph kernel." In: *International Joint Conference on Neural Networks*. IEEE, 2012, pp. 1–7.

[19] G. Da San Martino, N. Navarin, and A. Sperduti. "A Tree-Based Kernel for Graphs." In: *SIAM International Conference on Data Mining*. SIAM, 2012, pp. 975–986.

[20] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions." In: *ACM Symposium on Computational Geometry*. ACM, 2004, pp. 253–262.

[21] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity." In: *Journal of Medicinal Chemistry* 34.2 (1991). ACS, pp. 786–797.

[22] M. Defferrard, Bresson X., and P. Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." In: *Advances in Neural Information Processing Systems*. NIPS, 2016, pp. 3844–3852.

[23] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. "Convolutional Networks on Graphs for Learning Molecular Fingerprints." In: *Advances in Neural Information Processing Systems*. NIPS, 2015, pp. 2224–2232.

[24] C. Dwork and A. Roth. "The algorithmic foundations of differential privacy." In: *Foundations and Trends in Theoretical Computer Science* 9.3–4 (2014). Now, pp. 211–407.

[25] A. Feragen, N. Kasenburg, J. Petersen, M. D. Bruijne, and Borgwardt K. M. "Scalable kernels for graphs with continuous attributes." In: *Advances in Neural Information Processing Systems*. Erratum available at `http://image.diku.dk/aasa/papers/graphkernels_nips_erratum.pdf`. NIPS, 2013, pp. 216–224.

[26] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller. "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels." In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 869–877.

[27] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. "Protein Interface Prediction using Graph Convolutional Networks." In: *Advances in Neural Information Processing Systems*. NIPS, 2017, pp. 6533–6542.

[28] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. "Optimal assignment kernels for attributed molecular graphs." In: *International Conference on Machine Learning*. ACM, 2005, pp. 225–232.

[29] T. Gärtner, P. Flach, and S. Wrobel. "On Graph Kernels: Hardness Results and Efficient Alternatives." In: *Learning Theory and Kernel Machines*. Vol. 2777. Lecture Notes in Computer Science. Springer, 2003, pp. 129–143.

[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural Message Passing for Quantum Chemistry." In: *International Conference on Machine Learning*. PMLR, 2017.

[31] O. Goldreich. "Introduction to testing graph properties." In: *Property Testing*. Vol. 6390. Lecture Notes in Computer Science. Springer, 2010.

[32] O. Goldreich and D. Ron. "Property Testing in Bounded Degree Graphs." In: *Algorithmica* 32 (2002). Springer, pp. 302–343.

[33] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[34] K. Grauman and T. Darrell. "Approximate Correspondences in High Dimensions." In: *Advances in Neural Information Processing Systems*. NIPS, 2007, pp. 505–512.

[35] K. Grauman and T. Darrell. "The pyramid match kernel: Efficient learning with sets of features." In: *Journal of Machine Learning Research* 8.Apr (2007). JMLR, pp. 725–760.

[36] M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.

[37] M. Grohe, K. Kersting, M. Mladenov, and E. Selman. "Dimension Reduction via Colour Refinement." In: *European Symposium on Algorithms*. Vol. 8737. Lecture Notes in Computer Science. Springer, 2014, pp. 505–516.

[38] I. Guyon. *Design of experiments for the NIPS 2003 variable selection benchmark*. 2003. URL: `http://clopinet.com/isabelle/Projects/NIPS2003/Slides/NIPS2003-Datasets.pdf` (visited on 09/29/2016).

[39] W. L. Hamilton, R. Ying, and J. Leskovec. "Inductive Representation Learning on Large Graphs." In: *Advances in Neural Information Processing Systems*. NIPS, 2017, pp. 1025–1035.

[40] W. L. Hamilton, R. Ying, and J. Leskovec. "Representation Learning on Graphs: Methods and Applications." In: *IEEE Data Engineering Bulletin* 40.3 (2017). IEEE, pp. 52–74.

[41] D. Haussler. *Convolution Kernels on Discrete Structures*. Tech. rep. UCS-CRL-99-10. University of California at Santa Cruz, 1999.

[42] L. Hermansson, F. D. Johansson, and O. Watanabe. “Generalized Shortest Path Kernel on Graphs.” In: *International Conference on Discovery Science*. Vol. 9356. Lecture Notes in Computer Science. Springer, 2015, pp. 78–85.

[43] S. Hido and H. Kashima. “A Linear-Time Graph Kernel.” In: *IEEE International Conference on Data Mining*. IEEE, 2009, pp. 179–188.

[44] W. Hoeffding. “Probability Inequalities for Sums of Bounded Random Variables.” In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30.

[45] T. Horváth, T. Gärtner, and S. Wrobel. “Cyclic pattern kernels for predictive graph mining.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2004, pp. 158–167.

[46] P. Indyk and R. Motwani. “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality.” In: *ACM Symposium on Theory of Computing*. ACM, 1998, pp. 604–613.

[47] T. Joachims. “Training Linear SVMs in Linear Time.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2006, pp. 217–226.

[48] F. D. Johansson and D. Dubhashi. “Learning with Similarity Functions on Graphs Using Matchings of Geometric Embeddings.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 467–476.

[49] F. D. Johansson, O. Frost, C. Retzner, and D. Dubhashi. “Classifying large graphs with differential privacy.” In: *Modeling Decisions for Artificial Intelligence*. Vol. 9321. Lecture Notes in Computer Science. Springer, 2015, pp. 3–17.

[50] F. D. Johansson, V. Jethava, D. P. Dubhashi, and C. Bhattacharyya. “Global graph kernels using geometric embeddings.” In: *International Conference on Machine Learning*. PMLR, 2014, pp. 694–702.

[51] H. Kashima, K. Tsuda, and A. Inokuchi. “Marginalized Kernels Between Labeled Graphs.” In: *International Conference on Machine Learning*. AAAI, 2003, pp. 321–328.

[52] J. Kazius, R. McGuire, and R. Bursi. "Derivation and validation of toxicophores for mutagenicity prediction." In: *Journal Medicinal Chemistry* 48.13 (2005). ACS, pp. 312–320.

[53] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. *Benchmark Data Sets for Graph Kernels*. 2016. URL: `http://graphkernels.cs.tu-dortmund.de`.

[54] K. Kersting, M. Mladenov, R. Garnett, and M. Grohe. "Power iterated color refinement." In: *AAAI Conference on Artificial Intelligence*. AAAI, 2014, pp. 1904–1910.

[55] S. Kiefer, P. Schweitzer, and E. Selman. "Graphs Identified by Logics with Counting." In: *International Symposium on Mathematical Foundations of Computer Science*. 2015, pp. 319–330.

[56] T. N. Kipf and M. Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *International Conference on Learning Representation*. 2017.

[57] D. B. Kireev. "ChemNet: A Novel Neural Network Based Method for Graph/Property Mapping." In: *Journal of Chemical Information and Computer Sciences* 35.2 (1995). ACS, pp. 175–180.

[58] R. Kondor and H. Pan. "The Multiscale Laplacian Graph Kernel." In: *Advances in Neural Information Processing Systems*. NIPS, 2016, pp. 2982–2990.

[59] R. Kondor, N. Shervashidze, and K. M. Borgwardt. "The graphlet spectrum." In: *International Conference on Machine Learning*. ACM, 2009, pp. 529–536.

[60] N. Kriege and P. Mutzel. "Subgraph Matching Kernels for Attributed Graphs." In: *International Conference on Machine Learning*. Omnipress, 2012.

[61] N. Kriege, M. Neumann, K. Kersting, and M. Mutzel. "Explicit versus Implicit Graph Feature Maps: A Computational Phase Transition for Walk Kernels." In: *IEEE International Conference on Data Mining*. IEEE, 2014, pp. 881–886.

[62] N. M. Kriege, P.-L. Giscard, and R. C. Wilson. "On Valid Optimal Assignment Kernels and Applications to Graph Classification."

In: *Advances in Neural Information Processing Systems*. NIPS, 2016, pp. 1615–1623.

[63] N. M. Kriege, F. D. Johansson, and C. Morris. "A Survey on Graph Kernels." In: *CoRR* abs/1903.11835 (2019). Accepted for publication in *Applied Network Science*.

[64] N. M. Kriege and C. Morris. "Recent Advances in Kernel-Based Graph Classification." In: *The European Conference on Machine Learning & Principles and Practice of Knowledge Discovery In Databases*. Springer, 2017, pp. 388–392.

[65] N. M. Kriege, C. Morris, A. Rey, and C. Sohler. "A Property Testing Framework for the Theoretical Expressivity of Graph Kernels." In: *International Joint Conference on Artificial Intelligence*. IJCAI, 2018, pp. 2348–2354.

[66] N. M. Kriege, M. Neumann, C. Morris, K. Kersting, and P. Mutzel. "A Unifying View of Explicit and Implicit Feature Maps for Structured Data: Systematic Studies of Graph Kernels." In: *CoRR* abs/1703.00676 (2017). Accepted for publication in *Data Mining and Knowledge Discovery*.

[67] B. Li, X. Zhu, L. Chi, and C. Zhang. "Nested Subtree Hash Kernels for Large-Scale Graph Classification over Streams." In: *IEEE International Conference on Data Mining*. IEEE, 2012, pp. 399–408.

[68] L. Li, H. Tong, Y. Xiao, and W. Fan. "*Cheetah*: Fast Graph Kernel Tracking on Dynamic Graphs." In: *SIAM International Conference on Data Mining*. SIAM, 2015, pp. 280–288.

[69] W. Li, H. Saidi, H. Sanchez, M. Schäf, and P. Schweitzer. "Detecting Similar Programs via the Weisfeiler-Leman Graph Kernel." In: *International Conference on Software Reuse*. Vol. 9679. Lecture Notes in Computer Science. Springer, 2016, pp. 315–330.

[70] G. Loosli, S. Canu, and C. S. Ong. "Learning SVM in Kreĭn Spaces." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.6 (2016), pp. 1204–1216.

[71] L. Lovász. "On the Shannon Capacity of a Graph." In: *IEEE Transactions on Information Theory* 25.1 (2006). IEEE, pp. 1–7.

[72] P. Mahé and J.-P. Vert. "Graph kernels based on tree patterns for molecules." In: *Machine Learning* 75.1 (2009). Springer, pp. 3–35.

[73] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. "Extensions of marginalized graph kernels." In: *International Conference on Machine Learning*. ACM, 2004, pp. 552–559.

[74] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert. "Graph Kernels for Molecular Structure-Activity Relationship Analysis with Support Vector Machines." In: *Journal of Chemical Information and Modeling* 45.4 (2005). ACS, pp. 939–951.

[75] P. N. Malkin. "Sherali–Adams relaxations of graph isomorphism polytopes." In: *Discrete Optimization* 12 (2014). Elsevier, pp. 73 –97.

[76] C. Merkwirth and T. Lengauer. "Automatic Generation of Complementary Descriptors with Molecular Graph Networks." In: *Journal of Chemical Information and Modeling* 45.5 (2005). ACS, pp. 1159–1168.

[77] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient Estimation of Word Representations in Vector Space." In: *CoRR* abs/1301.3781 (2013).

[78] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. "Network motifs: simple building blocks of complex networks." In: *Science* 298.5594 (2002). AAAS, pp. 824–827.

[79] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

[80] C. Morris, K. Kersting, and P. Mutzel. "Glocalized Weisfeiler-Lehman Kernels: Global-Local Feature Maps of Graphs." In: *IEEE International Conference on Data Mining*. IEEE, 2017, pp. 327–336.

[81] C. Morris and P. Mutzel. "Towards a practical k-dimensional Weisfeiler-Leman algorithm." In: *CoRR* abs/1904.01543 (2019).

[82] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel. "Faster Kernel for Graphs with Continuous Attributes via Hashing." In: *IEEE International Conference on Data Mining*. IEEE, 2016, pp. 1095–1100.

[83] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, Jan Eric Lenssen, G. Rattan, and M. Grohe. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks." In: *AAAI Conference on Artificial Intelligence*. AAAI. 2019, pp. 4602–4609.

[84] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. “Propagation kernels: Efficient graph kernels from propagated information.” In: *Machine Learning* 102.2 (2016). Springer, pp. 209–245.

[85] I. Newman and C. Sohler. “Every Property of Hyperfinite Graphs Is Testable.” In: *SIAM Journal on Computing* 42.3 (2013). SIAM, pp. 1095–1112.

[86] M. E. J. Newman. “The structure and function of complex networks.” In: *SIAM review* 45.2 (2003), pp. 167–256.

[87] M. Niepert, M. Ahmed, and K. Kutzkov. “Learning Convolutional Neural Networks for Graphs.” In: *International Conference on Machine Learning*. PMLR, 2016, pp. 2014–2023.

[88] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis. “Matching Node Embeddings for Graph Similarity.” In: *AAAI Conference on Artificial Intelligence*. AAAI, 2017, pp. 2429–2435.

[89] G. Nikolentzos, P. Meladianos, S. Limnios, and M. Vazirgiannis. “A Degeneracy Framework for Graph Similarity.” In: *International Joint Conference on Artificial Intelligence*. IJCAI, 2018, pp. 2595–2601.

[90] L. Oneto, N. Navarin, M. Donini, A. Sperduti, F. Aiolli, and D. Anguita. “Measuring the expressivity of graph kernels through Statistical Learning Theory.” In: *Neurocomputing* 268.Supplement C (2017). Elsevier, pp. 4–16.

[91] F. Orsini, P. Frasconi, and L. De Raedt. “Graph Invariant Kernels.” In: *International Joint Conference on Artificial Intelligence*. IJCAI, 2015, pp. 3756–3762.

[92] D. Pachauri, R. Kondor, and V. Singh. “Solving the multi-way matching problem by permutation synchronization.” In: *Advances in Neural Information Processing Systems*. NIPS, 2013, pp. 1860–1868.

[93] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12 (2011). JMLR, pp. 2825–2830.

[94] A. Rahimi and B. Recht. "Random features for large-scale kernel machines." In: *Advances in Neural Information Processing Systems*. NIPS, 2008, pp. 1177–1184.

[95] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi. "Graph kernels for chemical informatics." In: *Neural Networks* 18.8 (2005). Elsevier, pp. 1093 –1110.

[96] R. Ramakrishnan, O. Dral P., M. Rupp, and O. Anatole von Lilienfeld. "Quantum chemistry structures and properties of 134 kilo molecules." In: *Scientific Data* 1 (2014). Nature.

[97] J. Ramon and M. Bruynooghe. "A polynomial time computable metric between point sets." In: *Acta Informatica* 37.10 (2001). Springer, pp. 765–780.

[98] J. Ramon and T. Gärtner. "Expressivity versus Efficiency of Graph Kernels." In: *International Workshop on Mining Graphs, Trees and Sequences*. 2003, pp. 65–74.

[99] D. Rogers and M. Hahn. "Extended-connectivity fingerprints." In: *Journal of Chemical Information and Modeling* 50.5 (2010). ACS, pp. 742–754.

[100] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond. "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17." In: *Journal of Chemical Information and Modeling* 52 11 (2012). ACS, pp. 2864–75.

[101] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The Graph Neural Network Model." In: *IEEE Transactions on Neural Networks* 20.1 (2009). IEEE, pp. 61–80.

[102] M. Schiavinato, A. Gasparetto, and A. Torsello. "Transitive Assignment Kernels for Structural Classification." In: *Similarity-Based Pattern Recognition: Third International Workshop*. Vol. 9370. Lecture Notes in Computer Science. Springer, 2015, pp. 146–159.

[103] K. Schütt, P. J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K. R. Müller. "SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions." In: *Advances in Neural Information Processing Systems*. NIPS, 2017, pp. 992–1002.

[104] R. K. Sevakula and N. K. Verma. “Support Vector Machine for Large Databases as Classifier.” In: *SEMCCO*. Vol. 7677. Lecture Notes in Computer Science. Springer, 2012.

[105] N. Shervashidze, S. V. N. Vishwanathan, T. H. Petri, K. Mehlhorn, and K. M. Borgwardt. “Efficient Graphlet Kernels for Large Graph Comparison.” In: *International Conference on Artificial Intelligence and Statistics*. PMLR, 2009, pp. 488–495.

[106] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. “Weisfeiler-Lehman Graph Kernels.” In: *Journal of Machine Learning Research* 12 (2011). JMLR, pp. 2539–2561.

[107] A. Sperduti and A. Starita. “Supervised neural networks for the classification of structures.” In: *IEEE Transactions on Neural Networks* 8.2 (1997). IEEE, pp. 714–35.

[108] Y. Su, F. Han, R. E. Harang, and X. Yan. “A fast Kernel for Attributed Graphs.” In: *SIAM International Conference on Data Mining*. SIAM, 2016, pp. 486–494.

[109] M. Sugiyama and K. M. Borgwardt. “Halting in Random Walk Kernels.” In: *Advances in Neural Information Processing Systems*. NIPS, 2015, pp. 1639–1647.

[110] G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.

[111] J.-P. Vert. “The optimal assignment kernel is not positive definite.” In: *CoRR* abs/0801.4061 (2008).

[112] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. “Graph Kernels.” In: *Journal of Machine Learning Research* 11 (2010). JMLR, pp. 1201–1242.

[113] N. Wale, I. A. Watson, and G. Karypis. “Comparison of descriptor spaces for chemical compound retrieval and classification.” In: *Knowledge and Information Systems* 14.3 (2008). Springer, pp. 347–375.

[114] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. “Dynamic Graph CNN for Learning on Point Clouds.” In: *CoRR* abs/1801.07829 (2018).

[115] B. Weisfeiler. *On Construction and Identification of Graphs*. Lecture Notes in Mathematics, Vol. 558. Springer, 1976.

[116] B. Weisfeiler and A. Leman. "The reduction of a graph to canonical form and the algebra which appears therein." In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968). English translation by G. Ryabov is available at `https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf`, pp. 12–16.

[117] P. Willett and V. Winterman. "A Comparison of Some Measures for the Determination of Inter-Molecular Structural Similarity Measures of Inter-Molecular Structural Similarity." In: *Quantitative Structure-Activity Relationships* 5.1 (1986). Elsevier, pp. 18–25.

[118] A. Woźnica, A. Kalousis, and M. Hilario. "Adaptive Matching Based Kernels for Labelled Graphs." In: *Advances in Knowledge Discovery and Data Mining*. Vol. 6119. Lecture Notes in Computer Science. Springer, 2010, pp. 374–385.

[119] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. "MoleculeNet: a benchmark for molecular machine learning." In: *Chemical Science* 9 (2 2018). RSC, pp. 513–530.

[120] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. "Representation Learning on Graphs with Jumping Knowledge Networks." In: *International Conference on Machine Learning*. PMLR, 2018, pp. 5453–5462.

[121] P. Yanardag and S. V. N. Vishwanathan. "A Structural Smoothing Framework For Robust Graph Comparison." In: *Advances in Neural Information Processing Systems*. NIPS, 2015, pp. 2125–2133.

[122] P. Yanardag and S. V. N. Vishwanathan. "Deep Graph Kernels." In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[123] Y. Yao and L. B. Holder. "Scalable classification for large dynamic networks." In: *IEEE International Conference on Big Data*. IEEE, 2015, pp. 609–618.

[124] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems." In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018.

[125] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. "Hierarchical Graph Representation Learning with Differentiable Pooling." In: *Advances in Neural Information Processing Systems*. NIPS, 2018, pp. 4800–4810.

[126] K. Yu, L. Ji, and X. Zhang. "Kernel Nearest-Neighbor Algorithm." In: *Neural Processing Letters* 15.2 (2002). Springer, pp. 147–156.

[127] M. Zhang and Y. Chen. "Weisfeiler-Lehman Neural Machine for Link Prediction." In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 575–583.

[128] M. Zhang, Z. Cui, M. Neumann, and C. Yixin. "An End-to-End Deep Learning Architecture for Graph Classification." In: *AAAI Conference on Artificial Intelligence*. AAAI, 2018, pp. 4428–4435.

[129] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai. "RetGK: Graph Kernels based on Return Probabilities of Random Walks." In: *Advances in Neural Information Processing Systems*. NIPS, 2018, pp. 3964–3974.

[130] V. M. Zolotarev. *One-dimensional stable distributions*. Translations of mathematical monographs. Providence, RI: AMS, 1986.