GloCal: Global-local Graph Kernels

Christopher Morris, Kristian Kersting, Petra Mutzel
TU Dortmund University
{christopher.morris, kristian.kersting, petra.mutzel}@tu-dortmund.de

***Abstract*—Most state-of-the-art graph kernels only take local graph properties into account, i.e., the kernel is computed with regard to properties of the neighborhood of vertices or other small substructures only. On the other hand, kernels that do take global graph properties into account may not scale well to large graph databases. Here we propose to start exploring the space between local and global graph kernels, striking the balance between both worlds. Specifically, we introduce a novel graph kernel based on the k-dimensional Weisfeiler-Lehman algorithm, and show that it takes local as well as global properties into account. Unfortunately, the k-dimensional Weisfeiler-Lehman algorithm scales exponentially in k. Consequently, we devise a stochastic version of the kernel with provable approximation guarantees using conditional Rademacher averages. On bounded-degree graphs, it can even be computed in constant time. We support our theoretical results with experiments on several graph classification benchmarks, showing that our kernels often outperform the state-of-the-art in terms of classification accuracies.**

I. Introduction

In several domains like chemo- and bioinformatics, or social network analysis large amounts of structured data, i.e., *graphs*, are prevalent. Surprisingly, most state-of-the-art *graph kernels* only take *local* graph properties into account, i.e., they compute the kernel based on properties of the neighborhood of vertices or other small substructures, e.g., see [1]–[4]. Moreover, kernels that do take global graph properties into account may not scale to large graph databases, e.g., see [5].

Recently, a graph kernel based on the 1-*dimensional Weisfeiler-Lehman* algorithm [6] has been proposed [3]. The 1-dimensional Weisfeiler-Lehman or *color refinement* algorithm is a well known heuristic for deciding whether two graphs are isomorphic: Given an initial *coloring* or *labeling* of the vertices, e.g., their degree, of both graphs, in each iteration two vertices with same label are assigned different labels if the number of neighbors labeled with a certain label is not equal. If the number of vertices labeled with a certain label is different after some iteration, the algorithm terminates and we conclude that the two graphs are not isomorphic. It is easy to see that the algorithm is not able to distinguish all non-isomorphic graphs, e.g., see [7]. On the other hand, this simple algorithm is already quite powerful since it can distinguish almost all graphs with high probability [6], [8], and has been applied in other areas [9]–[11]. Moreover, it can be generalized to k-tuples leading to a more powerful graph isomorphism heuristic [12], which has been investigated in deepth by the theoretical computer science community, e.g., see [12]–[14]. Now the so called *Weisfeiler-Lehman subtree kernel* is defined by computing the above heuristic for a predefined number of steps and the kernel finally counts the number of common labels arising in all refinement steps.

Clearly, the Weisfeiler-Lehman subtree kernel takes only local graph properties into account when performed for a fixed number of iterations. On the other hand, the k-dimensional variant does take global properties into account but does not consider local properties.

A. Contributions

Our contributions can be summed up as follows.

a) A Kernel Based on the k-dimensional Weisfeiler-Lehman Algorithm: We propose a graph kernel based on the k-dimensional Weisfeiler-Lehman algorithm. In order to take local properties into account we propose a *local* variant of the above algorithm. Moreover, we show that our local variant converges to the global kernel, which implies that it also takes global properties into account.

b) An Approximation Algorithm for the Local k-dimensional Weisfeiler-Lehman Kernel: Since the k-dimensional Weisfeiler-Lehman algorithm has a worst case running time of $\mathcal{O}(k^2 n^{k+1} \log n)$ [15], where n denotes the number of nodes of the graph, our local kernel does not scale to large graph databases. Hence, we propose algorithms 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. Moreover, for general graphs, we propose an *adaptive* sampling algorithm which uses results from statistical learning theory, in particular, *conditional Rademacher averages*. Additionally, we present a procedure based on sparse linear algebra to compute our local kernel, which speeds up the running time of the exact algorithms.

c) Experimental Evaluation: Finally, we implemented our algorithms and evaluated them on graph databases stemming from chemoinformatics and social network analysis. The result state that using global *and* local properties indeed improves performance over purely local or purely global approaches. Moreover, our kernels perform well over all data sets which is not the case for the other (local or global) kernels.

B. Related Work

In recent years, various graph kernels have been proposed, e.g., see [16], [17]. Gärtner, Flach, and Wrobel and Kashima, Tsuda, and Inokuchi simultaneously proposed graph kernels based on random walks, which count the number of walks two graphs have in common. Since then, random walk kernels have been studied intensively [16], [20]–[23]. Kernels based

on tree patterns were initially proposed by Ramon and Gärtner and later refined by Mahé and Vert. Kernels based on shortest paths were first proposed by Borgwardt and Kriegel, and are computed by performing one-step walks on transformed input graphs, where edges are annotated with shortest-path distances. A drawback of the approaches mentioned above is their *high computational cost*. They all employ the kernel trick, leading to a quadratic overhead in the size of the data set, and the running time for a single kernel function evaluation can be only bounded by a polynomial of relatively high degree, e.g., the random-walk and the shortest-path graph kernel have a running time in $\mathcal{O}(n^{2\omega})$ and $\mathcal{O}(n^4)$, respectively, where n denotes the maximum number of vertices of two graphs, and ω denotes the exponent for the running time of matrix multiplication. Moreover, recently graph kernels using matchings [27] and geometric embeddings [28] have been proposed.

A different line in the development of graph kernels focused particularly on scalable graph kernels. These kernels are typically computed efficiently by explicit feature maps, which allow to bypass the computation of a gram matrix, and allow applying scalable linear classification algorithms [29]–[31]. Prominent examples are kernels based on subgraphs up to a fixed size, so called *graphlets* [1], [32], or specific subgraphs like cycles and trees [33]. Other approaches of this category encode the neighborhood of every node by different techniques, e.g., see [17], [34], [35], and most notably the Weisfeiler-Lehman subtree kernel [3]. Subgraph and Weisfeiler-Lehman kernels have been successfully employed within frameworks for smoothed and deep graph kernels [36], [37]. Moreover, procedures to speed-up computation time of the Weisfeiler-Lehman subtree kernel in practice have been proposed in [10].

In the past few works considered graph kernels that use global graph properties. In [5] a kernel based on the Lovàsz number is proposed. Moreover, Kondor and Pan proposed a graph kernel based on the graph Laplacian.

Moreover, few works considered sampling as way to speed up graph kernel computation. In [1] a sampling algorithm for the graphlet kernel is introduced, which relies on approximating the relative counts of graphlets of a certain size. The authors provide a bound on the number of samples needed to approximate this quantity for a specific graphlet. However, they do not show an approximation result for the kernel. One drawback here is that the algorithm has to solve the graph isomorphism problem as a subproblem. More refined approaches can be found, e.g., in [39]–[41].

In [5] sampling techniques for approximating the kernel based on the Lovàsz number were used. However, the running time of their algorithm is at least quadratic in the number of vertices for a single evaluation of the kernel function.

C. Outline

In section 2, we fix some notation and describe the 1-dimensional Weisfeiler-Lehman algorithm and the corresponding kernel. Moreover, we describe its k-dimensional sibling. In the subsequent section we propose our local kernel based on the k-dimensional Weisfeiler-Lehman algorithm and show that it converges to the global kernel. In Section III-B, we propose a procedure to speed our kernels based on sparse matrix computation. Subsequently, we present our approximation algorithm for bounded-degree graphs and the approximation algorithm for general graphs based on conditional Rademacher averages. In Section IV, we report on the results of our experimental evaluation.

II. Preliminaries

An *(undirected) graph* G is a pair (V, E) with a *finite* set of *nodes* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of nodes 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). A *labeled graph* is a graph G endowed with a *label function* $l\colon V(G) \to \Sigma$, where Σ is some finite alphabet. We say that $l(v)$ is a *label* of v for v in $V(G)$.

Moreover, $N(v)$ denotes the *neighborhood* of v in $V(G)$, i.e., $N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. 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 $\varphi : V(G) \to V(H)$, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced* by S with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$.

Let χ be a non-empty set and let $k\colon \chi \times \chi \to \mathbb{R}$ be a function. Then k is a *kernel* on χ if there is a real Hilbert space $\mathcal{H}_k$ and a mapping $\phi\colon \chi \to \mathcal{H}_k$ such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$ for x and y in χ, where $\langle \cdot, \cdot \rangle$ denotes the inner product of $\mathcal{H}_k$. We call ϕ a *feature map*, and $\mathcal{H}_k$ a *feature space*. Moreover, $\mathbf{K}$ in $\mathbb{R}^{n \times n}$ denotes the *gram matrix* for a kernel k for some finite subset S of χ of cardinality n, i.e., $\mathbf{K}_{ij} = k(x_i, x_j)$ for x_i and x_j in S. Let $\mathbb{G}$ be a non-empty set of graphs, then a kernel $k\colon \mathbb{G} \times \mathbb{G} \to \mathbb{R}$ is called *graph kernel*. Moreover, let $[1\colon n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n > 1$, 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 with cardinality k, and let $\{\!\{\ldots\}\!\}$ denote a multiset.

A. Weisfeiler-Lehman Subtree Kernel

We now describe the 1-dimensional Weisfeiler-Lehman algorithm (1-WL) and the corresponding kernel. Let G and H be graphs, and let l be a label function $V(G) \cup V(H) \to \Sigma$, e.g., $l(v) = |N(v)|$ for v in $V(G) \cup V(H)$. In each iteration $i \geq 0$, the 1-WL algorithm computes a new label function $l^i\colon V(G) \cup V(H) \to \Sigma$. In iteration 0 we set $l^0 = l$. Now in iteration $i > 0$, we set

$$l^i(v) = \mathsf{relabel}((l^{i-1}(v), \mathsf{sort}(\{\!\{l^{i-1}(u) \mid u \in N(v)\}\!\}))),$$

for v in $V(G) \cup V(H)$, where $\mathsf{sort}(S)$ returns a (ascendantly) sorted tuple of the multiset S and the bijection $\mathsf{relabel}(p)$ maps the pair p to an unique value in Σ, which has not been used in previous iterations. Now if G and H have an unequal number of vertices labeled σ in Σ, we conclude that the graphs are not isomorphic. Moreover, if $l^{i-1} = l^i$

the algorithm terminates. After at most $|V(G)| + |V(H)|$ iterations the algorithm terminates. It is easy to see that the algorithm is not able to distinguish all non-isomorphic graphs, e.g., see [7]. On the other hand, this simple algorithm is already quite powerful, since it can distinguish almost all graphs with high probability [6], [8]. The running time of the algorithm is in $\mathcal{O}(m \log n)$ where $n = \max\{|V(G)|, |V(H)|\}$ and $m = \max\{|E(G)|, |E(H)|\}$, e.g., see [42].

The idea of the Weisfeiler-Lehman subtree graph kernel is to compute the above algorithm for $h \geq 0$ iterations, and after each iteration i compute a feature map $\phi^i(G)$ in $\mathbb{R}^{|\Sigma_i|}$ for each graph G, where $\Sigma_i \subseteq \Sigma$ denotes the codomain of l^i. Each component $\phi^i(G)_{\sigma_j}$ counts the number of occurrences of vertices labeled with σ_j in Σ_i. The overall feature map $\phi(G)$ is defined as the concatenation of the feature maps of all h iterations, i.e.,

$$\left(\phi^0(G)_{\sigma_1^1}, \ldots, \phi^0(G)_{\sigma_{|\Sigma_0|}^1}, \ldots, \phi^h(G)_{\sigma_1^h}, \ldots \phi^h(G)_{\sigma_{|\Sigma_h|}^h}\right).$$

Then the Weisfeiler-Lehman subtree kernel for h iterations is $k_{\text{WL}}^h(G, H) = \langle \phi(G), \phi(H) \rangle$. The running time for a single feature map 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 [3], where n and m denote the maximum number of vertices and edges over all N graphs, respectively.

B. The k-dimensional Weisfeiler-Lehman Algorithm

Based on the 1-WL, we consider the following variant[1] of the k-dimensional Weisfeiler-Lehman algorithm (k-WL) for $k \geq 2$. Let G and H be graphs. Instead of iteratively labeling vertices, the k-WL computes a labeling function defined on the set of k-sets $V(G)_k \cup V(H)_k$. In order to describe the algorithm, we define the neighborhood $N(t)$ of a k-set t in $V(G)_k$ (analogously for $V(H)_k$):

$$N(t) = \{\{t_1, \ldots, t_{j-1}, r, t_{j+1}, \ldots, t_k\} \mid r \in V(G) \setminus t\}. \quad (1)$$

That is, the neighborhood $N(t)$ of t is obtained by replacing a vertex from t by a vertex from $V(G) \setminus t$, see Figure 1b for a graphical illustration. In iteration 0, the algorithm labels each k-set with its isomorphism type, i.e., two k-sets s and t in $V(G)_k$ get the same label if the corresponding induced subgraphs are isomorphic. Now in iteration $i > 0$ we set

$$l^i(t) = \mathsf{relabel}((l^{i-1}(t), \mathsf{sort}(\{\!\{l^{i-1}(s) \mid s \in N(t)\}\!\}))), \quad (2)$$

where $\mathsf{sort}(S)$ returns a (ascendantly) sorted tuple of the multiset S of labels. The algorithm then proceeds analogously to the 1-WL.

III. A Local Kernel Based on the k-dimensional Weisfeiler-Lehman Algorithm

The k-WL, see the Introduction and Section II-B, is inherently *global*, i.e., it labels a k-set t by considering k-sets whose vertices are not connected to the vertices of t. Moreover, it does not take the sparsity of the underlying graph into account.

[1]In theoretical computer science it is usually defined on k-tuples instead of k-sets. Due to scalability we consider k-sets instead.

In order to capture the local properties of a graph, we propose a *local* variant. The idea of our *local* k-WL (k-LWL) is the following: The algorithm again labels all k-sets. But in order to extract local features we define the *local* neighborhood of a k-set t in $V(G)_k$,

$$N^{\text{L}}(t) = \{\{t_1, \ldots, t_{j-1}, r, t_{j+1}, \ldots, t_k\} \mid r \in V(G) \setminus t, \text{ and } \exists\, l \in [1, k] \setminus \{j\} \colon (t_l, r) \in E(G)\},$$

i.e., we consider a k-set s a local neighbor of a k-set t if s is in $N(t)$, and there is at least one edge between vertices of s and t, see Figure 1a for a graphical illustration. Now the local algorithm works the same way as the k-WL but in each iteration $i > 0$ considers the local neighbors of a k-set and computes a labeling function $l_{\text{L}}^i(t) \colon V(G)_k \to \Sigma$.

The following theorem shows that the above algorithm is able to capture the same global properties of a graph as the ordinary k-WL, i.e., after a finite number of iterations two k-sets have the same label with regard to the k-LWL if and only if they have the same label with regard to the k-WL.

Theorem 1. *Let G be a connected graph, and let s and t be k-sets from $V(G)_k$. Then for every iteration $h^* \geq 0$ of the k-WL there exists $h \geq h^*$ such that*

$$l_L^h(s) = l_L^h(t) \iff l^{h^*}(s) = l^{h^*}(t).$$

In order to prove the above theorem, we need to define the *k-set graph* and the *c-neighborhood* of a k-set.

Definition 1. *Let G be a connected graph, and let s and t be k-sets from $V(G)_k$, then the k-*set graph *$T(G) = (V_T, E_T)$, where $V_T = \{v_r \mid r \in V(G)_k\}$, and*

$$(v_s, v_t) \in E_T \iff s \in N^L(t).$$

Definition 2. *Let G be a connected graph, and let t be a k-set from $V(G)_k$, then the c-*neighborhood *of t for $c \geq 0$,*

$$\mathbf{N}(t, c) = \{s \in V(G)_k \mid d_T(v_t, v_s) \leq c\},$$

where $d_T \colon T(G) \times T(G) \to \mathbb{N}$ denotes the shortest-path distance in the k-set graph $T(G)$ of G.

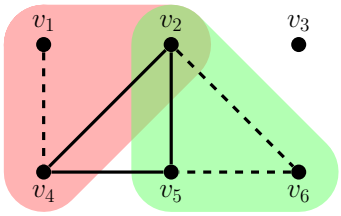
Moreover, we need the following result which states that two k-sets s and t get a different label if and only if there exists a different number of k-sets in the c-neighborhood of s and t labeled σ in Σ at some iteration of the k-LWL.

Lemma 1. *Let G be a connected graph, and let s and t be k-sets from $V(G)_k$, then $l_L^h(s) \neq l_L^h(t)$ for some iteration $h \geq 0$ if and only if there exists some iteration $0 \leq i \leq h$, and $c \geq 0$ such that there exists σ in Σ_i such that*

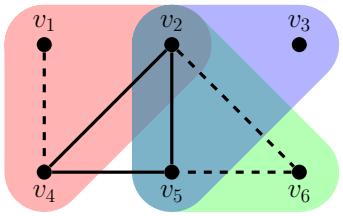
$$\left|\left\{r \in \mathbf{N}(s, c) \mid l_L^i(r) = \sigma\right\}\right| \neq \left|\left\{r \in \mathbf{N}(t, c) \mid l_L^i(r) = \sigma\right\}\right|.$$

Proof. Assume that $l_{\text{L}}^h(s) \neq l_{\text{L}}^h(t)$. After some iteration $j \leq h$, the labels of s and t first differ. Hence, we set $i = j$ and $c = 0$.

Now assume that the above inequality holds. Due to the different cardinalities, and the fact that relabel from Equation (2) is bijective, there is an iteration $i \leq i^* \leq h$, such that, without loss of generality, there will be a k-set r^* in $\mathbf{N}(s, c)$ that is



(a) Illustration of the local neighborhood of the k-set $\{v_2, v_5, v_6\}$, for better clarity we did not depict the neighbors $\{v_1, v_4, v_5\}$ and $\{v_4, v_5, v_6\}$.



(b) Illustration of the global neighborhood of the k-set $\{v_2, v_5, v_6\}$, for better clarity we did not depict the neighbors $\{v_1, v_2, v_5\}$, $\{v_1, v_3, v_5\}$, $\{v_2, v_3, v_4\}$, $\{v_2, v_3, v_5\}$, $\{v_2, v_3, v_6\}$, $\{v_3, v_4, v_5\}$ and $\{v_4, v_5, v_6\}$.

Fig. 1: Illustration the of the local and the global neighborhood the k-set v_2, v_3, v_6

labeled $\sigma^* = l_{\mathrm{L}}^{i^*}(r^*)$, and there is no k-set in $\mathbf{N}(t, c)$ that is labeled σ^* after iteration i^*. Now by applying Proposition 1 iteratively, the result follows. □

Proposition 1. *Let G be a connected graph, and let s and t be k-sets from $V(G)_k$ with $l_L^{i-1}(s) = l_L^{i-1}(t)$, then $l_L^i(s) \neq l_L^i(t)$ for some iteration $i \geq 1$ if and only if there exists σ in Σ_{i-1} such that*

$$\left|\left\{r \in \mathbf{N}(s,1) \mid l_L^{i-1}(r) = \sigma\right\}\right| \neq \left|\left\{r \in \mathbf{N}(t,1) \mid l_L^{i-1}(r) = \sigma\right\}\right|.$$

Proof. Follows directly from Equation (2), and the fact that relabel is bijective. □

We can now prove Theorem 1.

Proof of Theorem 1. Assume that $l^{h^*}(s) = l^{h^*}(t)$ holds. Since $N_j^{\mathrm{L}}(t) \subseteq N_j(t)$ for j in $[1, k]$ for t in $V(G)^k$ holds, it follows that $l_{\mathrm{L}}^{h^*}(s) = l_{\mathrm{L}}^{h^*}(t)$ by definition of the algorithm.

Now assume that $l^{h^*}(s) \neq l^{h^*}(t)$ and $l_{\mathrm{L}}^h(s) = l_{\mathrm{L}}^h(t)$ for all $h \geq h^*$ holds. Since $l^{h^*}(s) \neq l^{h^*}(t)$, observe that after some iteration $i < h^*$, without loss of generality, there will be a k-set s^* in $\mathbf{N}(s, c)$ for $c > 0$ such that $l^i(s^*) = \sigma^*$, and there will be no such k-set in $\mathbf{N}(t, c)$. Moreover, observe that the same is true for some iteration $j \geq i$ with regard to the k-LWL. To see this, assume that the label σ^* is caused by the different definitions of neighborhood of the k-LWL and the k-GWL. Since G is connected, the labels will eventually be propagated to s^*. Hence, by applying Lemma 1 with an appropiate choice of c, we arrive at a contradiction. □

A. A Kernel Based on the k-LWL

The idea for a kernel based on the k-LWL is the same as for the 1-WL. We compute the k-LWL for $h \geq 0$ iterations, and after each iteration i compute a feature map $\phi_{k\text{-LWL}}^i(G)$ in $\mathbb{R}^{|\Sigma_i|}$ for each graph G, where $\Sigma_i \subseteq \Sigma$ denotes the codomain of l_{L}^i. Each component $\phi_{k\text{-LWL}}^i(G)_{\sigma_j}$ counts the number of occurrences of k-sets labeled with σ_j in Σ_i. The overall feature map $\phi_{k\text{-LWL}}(G)$ is defined as the concatenation of the feature maps of all h iterations, i.e.,

$$\left(\phi^0(G)_{\sigma_1^1}, \ldots, \phi^0(G)_{\sigma_{|\Sigma_0|}^1}, \ldots, \phi^h(G)_{\sigma_1^h}, \ldots \phi^h(G)_{\sigma_{|\Sigma_h|}^h}\right).$$

Let G and H be two graphs then the kernel $k_{k\text{-LWL}}^h = \langle \phi(G)_{k\text{-LWL}}, \phi(H)_{k\text{-LWL}} \rangle$.

Moreover, one can show that the local kernel converges to global kernel after a finite number of iterations.

Proposition 2. *Let G and H be connected graphs, and let s and t be k-sets from $V(G)_k$ and $V(H)_k$, respectively. Moreover, let hen for every iteration $h^* \geq 0$ of the k-WL there exists $h \geq h^*$ such that*

$$l_L^h(s) = l_L^h(t) \iff l^{h^*}(s) = l^{h^*}(t).$$

Proof. rr □

Moreover, we will need the *normalized feature vector*

$$\widehat{\phi}_{k\text{-LWL}}(G) = \phi_{k\text{-LWL}}(G) / \|\phi_{k\text{-LWL}}(G)\|_1. \tag{3}$$

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

B. Sparse Linear Algebra Implementation

Kersting, Mladenov, Garnett, *et al.* proposed an implementation of the 1-WL based on sparse linear algebra: Let $G = (\{v_1, \ldots, v_n\}, E)$ be a graph with adjacency matrix A, let $c^0 = (l^0(v_1), \ldots, l^0(v_n))$, and let $\pi \colon \mathbb{R}_{\geq 0} \to \mathbb{N}$ be a function that maps a positive real to a unique prime number, then

$$c^h = \pi(c^{h-1}) + A \log(\pi(c^{h-1})), \text{ and } l^h(v_i) = c_i^h$$

for $h > 0$, where we applied $\log$ and π componentwise. This kann be trivially extended to the k-GWL and the k-LWL by considering the adjacency matrix of the k-tuple graph. Observe that for the k-LWL that a sparse graph will lead to a sparse adjacency matrix of the k-tuple graph, i.e., the sparsity only depends on the sparsity of the underlying graph and k. This is not the case for the k-GWL.

C. A Sampling Algorithm for Bounded-degree Graphs

Since the worst-case running time of the computation of $\phi_{k\text{-LWL}}(G)$ is in $\mathcal{O}(h \cdot n^k)$ for fixed k, computing the corresponding kernel is not feasible for large graphs. Thereto, we describe approximation algorithms for the k-LWL kernel. First, we introduce an algorithm restricted to bounded-degree graphs, and show that it can be computed in *constant time*. Subsequently, in Section III-D, we describe an algorithm for general graphs which employs an adaptive sampling strategy.

Algorithm 1 Approximation Algorithm for the 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 $\widetilde{\phi}_{k\text{-LWL}}(G)$ according to Inequality (5).

1: Let $\widetilde{\phi}_{k\text{-LWL}}(G)$ be a feature vector
2: Draw a multiset S of k-sets independently and uniformly from $V(G)_k$ with $|S|$ according to Equation (4).
3: **parallel for** $s \in S$ **do**
4: Compute the h-neighborhood $\mathbf{N}(s,h)$ around s
5: Compute $\sigma = l^h_{\mathrm{L},\mathfrak{N}}(s)$ on $G[\mathfrak{N}(s,h)]$
6: $\widetilde{\phi}_{k\text{-LWL}}(G)_\sigma = \widetilde{\phi}(G)_\sigma + {}^1\!/_{|S|}$
7: **end**
8: **return** $\widetilde{\phi}_{k\text{-LWL}}(G)$

The idea of the first algorithm is the following: Let G be a graph and let $\widetilde{\phi}_{k\text{-LWL}}(G)$ be a zero vector with the same number of components as $\widehat{\phi}_{k\text{-LWL}}(G)$. First, we sample a multiset S of k-sets from $V(G)_k$. The exact cardinality of S will be determined later. Secondly, for each such k-set s in S, we compute the h-neighborhood $\mathbf{N}(s,h)$ and compute the k-LWL on the subgraph induced by all vertices in $\mathfrak{N}(s,h) = \{u, v, \ldots, w \mid \{u, v, \ldots, w\} \in \mathbf{N}(s,h)\}$ resulting in a label σ for the k-set s. The following result shows that the label of a k-set after h iterations can be computed *locally* by only considering its h-neighborhood.

Lemma 2. *Let G be a graph, and let s be a k-set in $V(G)_k$. Moreover, let $l^h_{L,\mathfrak{N}}(s)$ be the label of s after the h-th iteration of the k-LWL on $G[\mathfrak{N}(s,h)]$, then $l^h_{L,\mathfrak{N}}(s) = l^h_L(s)$.*

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

Now the algorithm proceeds by adding ${}^1\!/_{|S|}$ to $\widetilde{\phi}_{k\text{-LWL}}(G)_\sigma$. An easy implication of the above lemma is that for k-sets that are in $\mathbf{N}(t,i)$ for $i \leq h$ it holds that $l^j_{\mathrm{L},\mathfrak{N}}(t) = l^j_{\mathrm{L}}(t)$ for $j \leq h$ such that $i + j \leq h$. See Algorithm 1 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 computes the feature vector of Equation (3), i.e., the feature vector over all h iterations. Let $\Gamma(d,h)$ be an upper bound on the maximum number of different labels of the k-LWL induced by d-bounded-degree h-neighborhoods.[2] We get the following result for bounded-degree graphs.

Theorem 2. *Let G be a d-bounded degree graph, and let*

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta)}{2({}^{\varepsilon}\!/_{\Gamma(d,h)})^2} \right\rceil . \tag{4}$$

[2]Observe that $\Gamma(d,h)$ can be easily upperbounded by the number of isomorphism types of d-bounded-degree h-neighborhoods.

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

$$\left\| \widehat{\phi}_{k\text{-LWL}}(G) - \widetilde{\phi}_{k\text{-LWL}}(G) \right\|_1 \leq \varepsilon \tag{5}$$

for any ε in $(0,1]$. Morever, 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 for bounded-degree graphs $\Gamma(d,h)$ 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-set s such that $l^h_{\mathrm{L},\mathfrak{N}}(s) = \sigma$ in iteration i of Algorithm 1, otherwise 0. Now observe that

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

Moreover, let $\bar{X}_\sigma = {}^1\!/_{|S|} \cdot \sum_{i=1}^{S} X_{i,\sigma} = \widetilde{\phi}_{k\text{-LWL}}(G)_\sigma$, then, by the linearity of expectation,

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

Hence, by the Hoeffding bound [43], we get

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

By setting the sample size

$$|S| \geq \left\lceil \frac{\log(2 \cdot \Gamma(d,h) \cdot 1/\delta)}{2\lambda^2} \right\rceil , \tag{6}$$

it follows that

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

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 (4), and the running time of lines 4 to 6 in Algorithm 1 are independent of the size of G, i.e., the number of vertices and edges. The correctness follows from Lemma 2. □

Now let $\widetilde{k}^h_{k\text{-LWL}}$ denote the corresponding kernel, i.e., $\widetilde{k}^h_{k\text{-LWL}} = \langle \widetilde{\phi}_{k\text{-LWL}}(G), \widetilde{\phi}_{k\text{-LWL}}(H) \rangle$ for two graphs G and H. The following proposition shows that the above kernel approximates the normalized k-LWL kernel arbitrarily close.

Proposition 3. *Let $\mathbb{G}$ be (non-empty, finite) set of d-bounded degree graphs, let $\widehat{k}^h_{k\text{-LWL}}$ be the normalized k-LWL kernel, and let*

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

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

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

for any λ in (0,1]. The running time for computing $\widetilde{\mathbf{K}}^h_{k\text{-LWL}}$ for $\mathbb{G}$ does only depend on the cardinality of $\mathbb{G}$, the number of iterations, k, and the maximum degree d.

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

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

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

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

for any $1 \leq i \leq \Gamma(d,h)$ holds. Now

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

Where 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 1. *Let $\mathbb{G}$ be (non-empty, finite) set of d-bounded degree graphs, let $\widehat{k}^h_{WL}$ be the normalized Weisfeiler-Lehman subtree kernel, and let*

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

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

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

for any λ in $(0,1]$. The running time for computing $\widetilde{\mathbf{K}}^h_{WL}$ for $\mathbb{G}$ does only depend on the size of $\mathbb{G}$, the number of iterations, and the maximum degree d.

Observe that instead of considering bounded-degree graphs Proposition 3 and Corollary 1 also hold for general graphs with a constant label alphabet, i.e., the cardinality of the label alphabet is independent of the size of the graphs.

D. Adaptive Sampling Algorithm for General Graphs

In order to calculate the sample size of Algorithm 1, we assumed to know the maximum degree or the number of different labels in advance. Hence, in order to circumvent this problem, we propose a variant of Algorithm 1 that relies on an adaptive sampling scheme based on *conditional Rademacher averages*.

Let $\mathcal{D}$ be a finite set, and let $\mathcal{F} = \{f \mid \mathcal{D} \to [0,1]\}$ be a family of functions. Now let $S = \{s_1, \ldots, s_m\}$ be sample of elements sampled independently and uniformly from $\mathcal{D}$. Next we define the *true average* and the *sample average*, respectively,

$$L_{\mathcal{D}}(f) = \mathbb{E}_{s \sim \mathcal{U}(\mathcal{D})}[f(s)], \quad \text{and} \quad L_S(f) = \frac{1}{m} \sum_{i=1}^{m} f(s_i) \,.$$

Given S we are interested in bounding

$$\sup_{f \in \mathcal{F}} |L_{\mathcal{D}}(f) - L_S(f)|, \tag{7}$$

i.e., determining the maximum deviation of the sample average to the true average using only the sample S. For i in $[1\,{:}\,m]$ let σ_i be a *Rademacher variable*, i.e., it is i.i.d. to $\mathbb{P}[\sigma_i = 1] = \mathbb{P}[\sigma_i = -1] = 1/2$. Now the conditional Rademacher average [44] is defined as

$$\mathcal{R}_{\mathcal{F}}(S) = \frac{1}{m} \mathbb{E}_{\sigma \sim \{\pm 1\}^m} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^{m} \sigma_i f(s_i) \right]. \tag{8}$$

We can now state a classical result from learning theory which employs Equation (8) to bound Equation (7) depending on the sample S.

Theorem 3 ([44])**.** *Let δ be in $(0,1)$, then with probability of at least $(1-\delta)$,*

$$\sup_{f \in \mathcal{F}} |L_{\mathcal{D}}(f) - L_S(f)| \leq 2 \cdot \mathcal{R}_{\mathcal{F}}(S) + 3\sqrt{\frac{\log(2/\delta)}{2m}} \,. \tag{9}$$

This bound can be further improved [45]. In order to bound the conditional Rademacher average we can use *Massart's Lemma*. Let $\mathbf{v}_{f,S} = (f(s_1), \ldots, f(s_m))$ for f in $\mathcal{F}$, and let the set $\mathcal{V}_S = \{v_S \mid f \in \mathcal{F}\}$. We get the following result.

Lemma 3 ([44])**.**

$$\mathcal{R}_{\mathcal{F}}(S) \leq \max_{f \in \mathcal{F}} \|v_{f,S}\| \frac{\sqrt{2 \ln |\mathcal{V}_S|}}{m} \,.$$

Moreover, based on the above lemma Riondato and Upfal proposed a convex optimization problem to achieve tighter (empirical) bounds on the conditional Rademacher average.

Now in order to use the above theorem to approximate the feature vector of the k-LWL, we define the following family of functions

$$\mathcal{F}_{G,k\text{-LWL}} = \{\mathbf{1}_{\sigma,h}(t) \mid t \in V(G)_k, \sigma \in \Sigma, \text{ and } h \geq 0\}$$

for a graph G, which equals 1 for a k-set t in $V(G)_k$ if it has label σ after h iterations of the k-LWL, and otherwise 0, i.e.,

$$\mathbf{1}_{\sigma,h}(t) = \begin{cases} 1 & \text{if } l^h_{\mathrm{L}}(t) = \sigma, \\ 0 & \text{otherwise} \,. \end{cases}$$

Observe that

$$L_{\mathcal{D}}(\mathbf{1}_{\sigma,h}) = \widehat{\phi}(G)^h_{\sigma} \,.$$

Moreover, observe that $\|v_{S,f}\| \leq \sqrt{m}$ for $f = \mathbf{1}_{\sigma,h}(t)$ for f in $\mathcal{F}_{G,k\text{-LWL}}$, and that $\mathcal{V}_S$ can be computed from S. The idea of the sampling algorithm is the following: In each iteration

$i \geq 0$ we determine a sample size $|S_i|$. Draw $|S_i|$ samples independently and uniformly at random from $V(G)_k$, and proceed as in Algorithm 1, i.e., we compute the h-neighborhood of each k-sets t in S_i, and compute its label $l^h_{\mathrm{L},\mathfrak{N}}(t)$ after h iterations on the induced subgraph of the set $\mathfrak{N}(t,h)$. The algorithms terminates if Inequality (9) together with Lemma 3 holds, otherwise we proceed with iteration $i+1$, and sample a multiset S_{i+1} such that $|S_i| \leq |S_{i+1}|$. See Algorithm 2 for pseudo code.

Algorithm 2 Adaptive Approximation Algorithm for the k-LWL

Require: A 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 $\widetilde{\phi}_{k\text{-LWL}}(G)$ according to Theorem 4.

```
Let φ̃_{k-LWL}(G) be a feature vector
i ← 0, s ← 0
while Inequality (9) not satisfied do
    Compute S_i
    s ← s + |S_i|
    parallel for t ∈ S_i do
        Compute the h-neighborhood N(t,h) around t
        Compute l^h_{L,𝔑}(t) = σ on G[𝔑(t,h)]
        φ̃_{k-LWL}(G)_σ = φ̃_{k-LWL}(G)_σ + 1
    end
    i ← i + 1
end while
φ̃_{k-LWL}(G) = 1/s · φ̃_{k-LWL}(G)
return φ̃_{k-LWL}(G)
```

We get the following result.

Theorem 4. *Let G be a graph, then Algorithm 2 approximates the normalized feature vector $\widehat{\phi}_{k\text{-LWL}}(G)$ of the k-LWL for h iterations such that with probability $(1-\delta)$ for δ in $(0,1)$,*

$$\sup_{\sigma \in \Sigma} \left| \widehat{\phi}_{k\text{-LWL}}(G)_\sigma - \widetilde{\phi}_{k\text{-LWL}}(G)_\sigma \right| \leq \varepsilon, \tag{10}$$

for any ε in $(0,1]$.

Proof. The finiteness follows from the fact that we increase the sample size in every iteration. Hence, eventually Inequality (9) will hold. The approximation guarantee then follows from Algorithm 2, Inequality (9), and the correctness of Algorithm 1. □

The above algorithm and the above result can be adapted straightforwardly to a set of graphs instead of a single graph.

IV. Experimental Evaluation

Our intention here is to investigate the benefits of the k-LWL and the adaptive sampling algorithm compared to the state-of-the-art. More precisely, we address the following questions:

- **Q1** How does the k-LWL compare to state-of-the-art graph kernels in terms of classification accuracy?
- **Q2** How does the k-LWL compare to the (global) k-WL in terms of classification accuracy and running time?
- **Q3** Does our adaptive sampling algorithm to approximate the k-LWL speed up the computation time of the kernel computation?
- **Q4** Does the adaptive sampling algorithm lead to worse classification accuracies compared to the exact algorithm?
- **Q5** Does the linear algebra based implementation of the k-LWL lead to a speed up in running time? Does parallelization speed up the computation time?

A. Data Sets and Graph Kernels

We used the following data sets to evaluate our kernels, see Table I for properties and statistics.

Enzymes and Proteins contain graphs representing proteins according to the graph model of [47]. Each node is annotated with a discrete label. The data sets are subdivided into six and two classes, respectively. Note that this is the same data set as used in [48], which does not contain all the annotations described and used in [47].

IMDB-BINARY is a movie collaboration data set first used in [37] based on data from IMDB[3]. Each node represents an actor or an actress, and there exists an edge if the corresponding actor or actress appears in the same movie. The nodes are unlabeled. Each graph represents an ego network of an actor or actress. The data set is divided into two classes corresponding to action or romantic movies.

Mutag is a data set consisting of mutagenetic aromatic and heteroaromatic nitro compounds [49], [50] with seven discrete node labels.

NCI1 and NCI109 Is a (balanced) subset of a data set made available by the National Cancer Institute[4] [3], [32] consisting of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively. The nodes are annotated with discrete labels.

PTC_FM is data set from the Predicte Toxicology Challenge (PTC)[5] containing chemical compounds labeled according to carcinogenicity on rodents female mice (FM). The nodes are annotated with discrete labels. It is divided into two classes.

Reddit-Binary is a social network data set based on data from the content-aggregration website Reddit[6] [37]. Each node represents a user and two nodes are connected by an node if one user responded to the other users comment. The nodes are unlabeled and the data set is divided into two classes representing question-answer-based or discussion-based communities.

We implemented the $\{2,3\}$-LWL and the adaptive sampling algorithm for the 3-LWL (3-LWL-Approx). Moreover, for the $\{2,3\}$-LWL we implemented the linear algebra based algorithm ($\{2,3\}$-LWL-LA) described in Section III-B. Moreover, we

[3]https://www.imdb.com/
[4]https://www.cancer.gov/
[5]http://www.predictive-toxicology.org/ptc/
[6]http://www.reddit.com/

TABLE I: Data set statistics and properties.

Data Set	Properties				
	Number of Graphs	Number of Classes	∅ Number of Nodes	∅ Number of Edges	Node Labels
ENZYMES	600	6	32.6	62.1	✓
IMDB-BINARY	1000	2	19.8	96.5	✗
MUTAG	188	6	17.9	19.8	✓
NCI1	4110	2	29.9	32.3	✓
NCI109	4127	2	29.7	32.1	✓
PTC_FM	349	2	14.1	14.5	✓
PROTEINS	1113	2	39.1	72.8	✓
REDDIT-BINARY	2000	2	429.6	497.8	✗

implemented a parallel version of the 3-LWL, where we parallelized the label computation in each iteration (3-LWL-PAR), and implemented a parallel version of the 3-LWL-APPROX (3-LWL-APPROX-PAR), where we computed the colors of the sampled k-sets in parallel.

We compare our kernels to the Weisfeiler-Lehman subtree kernel [3], the graphlet kernel [1], and the shortest-path kernel [26]. Moreover, we implemented a kernel (k-GWL) based on the (global) k-WL that uses the definition of neighborhood of Equation (1) for k in $\{2, 3\}$. Here we also implemented the linear algebra based algorithm ($\{2, 3\}$-GWL-LA). All kernels were implemented in C++11.[7]

B. Experimental Protocol

For each kernel, we computed the normalized gram matrix. We computed the classification accuracies using the C-SVM implementation of LIBSVM [51], 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. For the 3-LWL-APPROX we set the ε of Inequality (10) to 0.05, (0.1) and the initial sample size was set to 100. We doubled the sampled size when Inequality (9) was not fulfilled.

We repeated each 10-fold cross validation ten times with different random folds, and report average accuracies and standard deviations. Since the adaptive approximation algorithm is a randomized algorithm, we computed each gram matrix ten times and report average classification accuracies, standard deviations, and running times. We report running times for the 1-WL, $\{2, 3\}$-GWL, the $\{2, 3\}$-LWL, and the 3-LWL-APPROX, and the corresponding linear based and and parallel variants with 5 refinement steps. For the graphlet kernel we counted (labeled) connected subgraphs of size three. The number of iterations for the 1-WL, $\{2, 3\}$-GWL, $\{2, 3\}$-LWL, and 3-LWL-APPROX were selected from $\{0, \ldots, 5\}$ using 10-fold cross validation on the training folds only.

Moreover, for the 3-$\{$GWL, LWL, LWL-APPROX$\}$ we selected by 10-fold cross validation on the training folds only if the initial labels were set to the isomorphism type or the number of edges of the 3-sets and the node labels. All experiments were conducted on a workstation with an Intel Core i7-3770 with 3.40GHz and 16GB of RAM running Ubuntu 16.04.5 LTS. The parallel algorithms were executed on four cores. Moreover, we used GNU C++ Compiler 5.4.1 with the flag –O2.

[7]The source code can be obtained from https://github.com/chrsmrrs/globalwl.

C. Results and Discussion

In the following we answer questions **Q1** to **Q4**. See also Tables II and III.

A1 On four out of eight data sets the 3-LWL or the 3-LWL-APPROX performs better than the state-of-the-art, e.g., on the challenging ENZYMES data set the 3-LWL performs more than 8% better than the 1-WL. Moreover, on the other data sets (excluding NCI109) the 3-LWL is at most 3% worse than the best performing kernel. Observe that the 3-LWL performs well over all data sets. This is not the case for the other kernels. In order to further exemplify this fact, we computed average rankings (scores 1 to 9) with regard to classification accurary. We grouped the kernels into three groups (local kernel, global kernels, local and global kernels). For each group we computed an average ranking by choosing the classification accuracy of the best performing kernel in the group, see Table II (last column). The 3-LWL, which consider local as well as global properties, perform best on average.

A2 On all data sets besides PTC_FM the 3-LWL achieves better classification accuracies than the 3-GWL. This is further illustrated by the much lower average ranking of the k-LWL compared to the k-GWL kernels. On the ENZYMES data set the 3-LWL is more than 12% better. This shows the benefit of using local as well global features. On all data sets the 3-LWL is faster, e.g., on the ENZYMES data set, the 3-LWL is more than seven times faster. On the PROTEINS and the REDDIT-BINARY data set the 3-GWL did not finish within one day. On these data sets the 3-LWL-APPROX was able to finish within one day. The same is true for the 2-LWL compared to the 2-GWL.

A3 The approximation algorithm speeds up the computation time of the kernel on the larger data sets (PROTEINS and REDDIT-BINARY). For example on the PROTEINS data set it (with inital sample size 100 and $\varepsilon = 0.1$) is over 42 times faster than the exact algorithm.

A4 On all data sets excluding ENZYMES the 3-LWL-APPROX is at most 2% (with inital sample size 100 and $\varepsilon = 0.05$) worse than the exact algorithm in terms of classification accuracies. On some data sets it even achieves better accuracies. On ENZYMES the 3-LWL-APPROX is about

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

	Graph Kernel	Data Set								
		ENZYMES	IMDB-BINARY	MUTAG	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY	∅ Rank
Local	GRAPHLET	41.0 ±1.2	59.4 ±0.4	**87.7** ±1.4	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	81.7 ±1.3	74.5 ±0.3	73.4 ±0.1	62.1 ±0.9	**76.4** ±0.4	**84.7** ±0.2	1.8
	1-WL	53.4 ±1.4	72.4 ±0.5	78.3 ±1.9	**83.1** ±0.2	**85.2** ±0.2	62.9 ±1.6	73.7 ±0.5	75.3 ±0.3	
Glob.	2-GWL	49.7 ±1.6	71.5 ±0.8	83.6 ±1.6	71.3 ±0.3	70.8 ±0.4	**64.7** ±0.2	75.2 ±0.4	67.0 ±0.1	4.0
	3-GWL	49.6 ± 1.3	73.4 ±0.8	86.0 ±1.3	73.6 ±0.2	72.4 ±0.2	63.7 ±0.7	OOT	OOM	
Loc.+Glob.	2-LWL	51.6 ±0.8	72.1 ±0.4	85.2 ±1.6	77.0 ±0.2	76.9 ±0.2	**64.7** ±0.2	75.0 ±0.3	75.0 ±0.4	
	3-LWL	**61.8** ±1.1	**75.0** ±0.7	83.8 ±1.2	80.8 ±0.4	80.2 ±0.3	60.7 ±1.3	74.8 ±0.5	OOM	**1.4**
	3-LWL-APPROX†	54.3 ±0.9	72.8 ±0.7	**87.7** ±1.4	79.4 ±0.2	78.0 ±0.3	62.9 ±0.9	75.2 ±0.7	82.8 ±0.6	
	3-LWL-APPROX‡	52.7 ±1.1	72.7 ±1.0	86.3 ±1.5	78.4 ±0.4	77.3 ±0.4	62.7 ±0.9	75.4 ±0.6	82.7 ±0.7	

TABLE III: Average running times in seconds (Number of iterations for 1-WL, 3-LWL, 3-LWL-APPROX, and 3-GWL: 5, OOT— Computation did not finish within one day, OOM— Out of memory.

Graph Kernel	Data Set							
	ENZYMES	IMDB-BINARY	MUTAG	NCI1	NCI109	PTC_FM	PROTEINS	REDDIT-BINARY
GRAPHLET	<1	<1	<1	<1	<1	<1	<1	3.3
SHORTEST-PATH	<1	<1	<1	2.3	2.1	<1	<1	1 115.6
1-WL	<1	<1	<1	1.6	1.6	<1	<1	3.1
2-LWL	< 1	1.1	< 1	4.5	4.5	< 1	5.2	3 726.4
2-LWL-LA	< 1	< 1	< 1	3.8	3.8	< 1	2.9	1 234.1
2-GWL	6.5	3.2	2.0	37.0	37.0	< 1	368.0	OOT
2-GWL-LA	1.7	< 1	< 1	11.5	13.3	< 1	62.8	OOM
3-LWL	44.6	59.1	1.0	156.5	155.4	2.1	4 098.7	OOM
3-LWL-PAR	24.2	17.4	< 1	96.3	97.5	1.7	1 490.8	OOM
3-LWL-LA	13.1	13.6	< 1	58.9	58.9	< 1	1 005.6	OOM
3-LWL-APPROX†	216.3	1 183.8	14.7	688.6	712.6	40.7	383.4	2 440.9
3-LWL-APPROX-PAR†	63.5	270.1	4.7	241.0	247.3	8.3	112.5	932.2
3-LWL-APPROX‡	54.0	280.8	7.1	186.2	185.9	9.9	96.0	553.4
3-LWL-APPROX-PAR‡	20.6	71.3	1.5	91.7	92.4	2.9	37.6	331.0
3-GWL	335.6	119.2	2.7	1 253.2	1 238.9	8.7	OOT	OOT
3-GWL-LA	56.0	23.7	< 1	243.4	239.3	< 1	OOM	OOM

6% worse than the exact algorithm. But it still achieves better classification accuracies than the other kernels.

A5 Both, the linear algebra based algorithm and the parallel variants, greatly improve the running time. For example on the PROTEINS data set the 3-LWL-LA (3-LWL-APPROX-PAR with inital sample size 100 and $\varepsilon = 0.05$) is more than four (almost three) times faster than the 3-LWL (3-LWL-APPROX).

V. CONCLUSION AND FUTURE WORK

We proposed a graph kernel based on a local variant of the (global) k-dimensional Weisfeiler algorithm, which explores the space between local and global graph properties. We demonstrated that it can be computed approximately in constant time for bounded-degree graphs. For general graphs we proposed an adaptive sampling algorithm. Our experimental study showed that our kernel advances the state-of-the-art. In particular, the predictive accuracy of our kernel is favorable on all data sets which is not the case for the other kernels. We can draw the following conclusion:

> *The local* k-LWL *is able to extract* local as well as global *graph properties, and behaves favorable for all data sets. Moreover, it can be approximated efficiently. For bounded-degree graphs the approximation can be computed in* constant *time.*

Our work provides several interesting directions for future work. First, we believe that the space between local and global graph properties provide a growth path for designing novel graph kernels. Secondly, it should be combined with the recent progress in deep learning approaches for graph classification [52]–[54] as our sampling is a form of dropout. Moreover, the running time should be reduced further, e.g., by random walks [10] and by further engineering.

ACKNOWLEDGEMENT

This work has been supported by the German Science Foundation (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Data Analysis", project A6 "Resource-efficient Graph Mining".

REFERENCES

[1] N. Shervashidze, S. V. N. Vishwanathan, T. H. Petri, K. Mehlhorn, and K. M. Borgwardt, "Efficient graphlet kernels for large graph comparison," in *12th International Conference on Artificial Intelligence and Statistics*, 2009, pp. 488–495.

[2] F. Costa and K. De Grave, "Fast neighborhood subgraph pairwise distance kernel," in *26th International Conference on Machine Learning*, 2010, pp. 255–262.

[3] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-Lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, pp. 2539–2561, 2011.

[4] F. Orsini, P. Frasconi, and L. De Raedt, "Graph invariant kernels," in *24th International Joint Conference on Artificial Intelligence*, 2015, pp. 3756–3762.

[5] F. D. Johansson, V. Jethava, D. P. Dubhashi, and C. Bhattacharyya, "Global graph kernels using geometric embeddings," in *31st International Conference on Machine Learning*, 2014, pp. 694–702.

[6] L. Babai and L. Kucera, "Canonical labelling of graphs in linear average time," in *20th Annual Symposium on Foundations of Computer Science*, 1979, pp. 39–46.

[7] V. Arvind, J. Köbler, G. Rattan, and O. Verbitsky, "On the power of color refinement," in *20th International Symposium on Fundamentals of Computation Theory*, 2015, pp. 339–350.

[8] L. Babai, P. Erdős, and S. M. Selkow, "Random graph isomorphism," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635, 1980.

[9] M. Grohe, K. Kersting, M. Mladenov, and E. Selman, "Dimension reduction via colour refinement," in *22th European Symposium on Algorithms*, 2014, pp. 505–516.

[10] K. Kersting, M. Mladenov, R. Garnett, and M. Grohe, "Power iterated color refinement," in *28th AAAI Conference on Artificial Intelligence*, 2014, pp. 1904–1910.

[11] W. Li, H. Saidi, H. Sanchez, M. Schäf, and P. Schweitzer, "Detecting similar programs via the weisfeiler-leman graph kernel," in *15th International Conference on Software Reuse: Bridging with Social-Awarenes*, 2016, pp. 315–330.

[12] J. Cai, M. Fürer, and N. Immerman, "An optimal lower bound on the number of variables for graph identifications," *Combinatorica*, vol. 12, no. 4, pp. 389–410, 1992.

[13] S. Kiefer and P. Schweitzer, "Upper bounds on the quantifier depth for graph differentiation in first order logic," in *31st ACM/IEEE Symposium on Logic in Computer Science*, 2016, pp. 287–296.

[14] L. Babai, "Graph isomorphism in quasipolynomial time," in *48th ACM SIGACT Symposium on Theory of Computing*, 2016, pp. 684–697.

[15] N. Immerman and E. Lander, "Describing graphs: A first-order approach to graph canonization," in *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*. 1990, pp. 59–81.

[16] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, pp. 1201–1242, 2010.

[17] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting, "Propagation kernels: Efficient graph kernels from propagated information," *Machine Learning*, vol. 102, no. 2, pp. 209–245, 2016.

[18] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*, 2003, pp. 129–143.

[19] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *20th International Conference on Machine Learning*, 2003, pp. 321–328.

[20] U. Kang, H. Tong, and J. Sun, "Fast random walk graph kernel," in *SIAM International Conference on Data Mining*, 2012, pp. 828–838.

[21] N. Kriege, M. Neumann, K. Kersting, and M. Mutzel, "Explicit versus implicit graph feature maps: A computational phase transition for walk kernels," in *14th IEEE International Conference on Data Mining*, 2014, pp. 881–886.

[22] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, "Extensions of marginalized graph kernels," in *21st International Conference on Machine Learning*, 2004, pp. 552–559.

[23] M. Sugiyama and K. M. Borgwardt, "Halting in random walk kernels," in *Advances in Neural Information Processing Systems*, 2015, pp. 1639–1647.

[24] J. Ramon and T. Gärtner, "Expressivity versus efficiency of graph kernels," in *1st International Workshop on Mining Graphs, Trees and Sequences*, 2003.

[25] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine Learning*, vol. 75, no. 1, pp. 3–35, 2009.

[26] K. M. Borgwardt and H.-P. Kriegel, "Shortest-path kernels on graphs," in *5th IEEE International Conference on Data Mining*, 2005, pp. 74–81.

[27] N. M. Kriege, G. P.-L., and R. C. Wilson, "On valid optimal assignment kernels and applications to graph classification," in *Advances in Neural Information Processing Systems*, 2016, pp. 1615–1623.

[28] F. D. Johansson and D. Dubhashi, "Learning with similarity functions on graphs using matchings of geometric embeddings," in *21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 467–476.

[29] W.-L. Chiang, M.-C. Lee, and C.-J. Lin, "Parallel dual coordinate descent method for large-scale linear classification in multi-core environments," in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1485–1494.

[30] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[31] T. Joachims, "Training linear SVMs in linear time," in *12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 217–226.

[32] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[33] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 158–167.

[34] L. Bai, L. Rossi, Z. Zhang, and E. R. Hancock, "An aligned subtree kernel for weighted graphs," in *32nd International Conference on Machine Learning*, 2015, pp. 30–39.

[35] S. Hido and H. Kashima, "A linear-time graph kernel," in *Ninth IEEE International Conference on Data Mining*, 2009, pp. 179–188.

[36] P. Yanardag and S. V. N. Vishwanathan, "A structural smoothing framework for robust graph comparison," in *Advances in Neural Information Processing Systems*, 2015, pp. 2125–2133.

[37] ——, "Deep graph kernels," in *21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1365–1374.

[38] R. Kondor and H. Pan, "The multiscale laplacian graph kernel," in *Advances in Neural Information Processing Systems*, 2016, pp. 2982–2990.

[39] N. K. Ahmed, T. Willke, and R. A. Rossi, "Estimation of local subgraph counts," in *IEEE International Conference on Big Data*, 2016, pp. 1–10.

[40] X. Chen, Y. Li, P. Wang, and J. C. S. Lui, "A general framework for estimating graphlet statistics via random walk," *VLDB Endowment*, vol. 10, no. 3, pp. 253–264, Nov. 2016.

[41] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi, "Counting graphlets: Space vs time," in *10th ACM International Conference on Web Search and Data Mining*, 2017, pp. 557–566.

[42] C. Berkholz, P. S. Bonsma, and M. Grohe, "Tight lower and upper bounds for the complexity of canonical colour refinement," in *21st European Symposium on Algorithms*, 2013, pp. 145–156.

[43] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[44] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Press, 2012.

[45] L. Oneto, A. Ghio, D. Anguita, and S. Ridella, "An improved analysis of the rademacher data-dependent bound using its self bounding property," *Neural Networks*, vol. 44, pp. 107–111, 2013.

[46] M. Riondato and E. Upfal, "ABRA: Approximating betweenness centrality in static and dynamic graphs with rademacher averages," in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1145–1154.

[47] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels.," *Bioinformatics*, vol. 21, no. Supplement 1, pp. i47–i56, 2005.

[48] A. Feragen, N. Kasenburg, J. Petersen, M. D. Bruijne, and B. 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, 2013, pp. 216–224.

[49] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[50] N. Kriege and P. Mutzel, "Subgraph matching kernels for attributed graphs," in *29th International Conference on Machine Learning*, 2012.

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

[52] 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*, 2015, pp. 2224–2232.

[53] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *33rd International Conference on Machine Learning*, 2016, pp. 2014–2023.

[54] T. Lei, W. Jin, R. Barzilay, and J. Tommi, "Deriving neural architectures from sequence and graph kernels," *CoRR, accepted at ICML 2017*, vol. abs/1703.00676, 2017.