

## Weisfeiler-Lehman graph kernels

**Nino Shervashidze**

NINO.SHERVASHIDZE@TUEBINGEN.MPG.DE

*Machine Learning & Computational Biology Research Group  
Max Planck Institutes Tübingen  
Spemannstr. 38  
72076 Tübingen  
Germany*

**Pascal Schweitzer**

PASCAL@MPI-INF.MPG.DE

*Max Planck Institute for Informatics  
Campus E1 4  
66123 Saarbrücken  
Germany*

**Erik Jan van Leeuwen**

E.J.VAN.LEEUWEN@II.UIB.NO

*Department of Informatics  
University of Bergen  
Postboks 7803  
N-5020 Bergen  
Norway*

**Kurt Mehlhorn**

MEHLHORN@MPI-INF.MPG.DE

*Max Planck Institute for Informatics  
Campus E1 4  
66123 Saarbrücken  
Germany*

**Karsten M. Borgwardt**

KARSTEN.BORGWARDT@TUEBINGEN.MPG.DE

*Machine Learning & Computational Biology Research Group  
Max Planck Institutes Tübingen  
Spemannstr. 38  
72076 Tübingen  
Germany*

**Editor:**

### Abstract

In this article, we address the problem of defining scalable kernels on large graphs with discrete node labels. Key to our approach is the Weisfeiler-Lehman test of isomorphism, which allows us to compute a sequence of graphs which capture the topological and label information of the original graph in a runtime which is linear in the number of edges. We can apply existing graph kernels on this graph sequence and make them take into account the structural information which they ignored before. We can also define new, efficient graph kernels: In particular, a subtree kernel whose runtime is linear in the number of edges in the input graphs and in the maximum height of the subtrees considered.

Our experiments show that kernels based on the Weisfeiler-Lehman sequence of graphs allow us to improve over state-of-the-art graph kernels both in terms of runtime and accuracy.

**Keywords:** Graph kernels, graph classification, similarity measures for graphs, Weisfeiler-Lehman algorithm

## 1. Introduction

Graph-structured data is becoming more and more abundant: examples are social networks, protein or gene regulation networks, chemical pathways and protein structures, or the growing body of research in program flow analysis. To analyze and understand this data, one needs data analysis and machine learning methods that can handle large-scale graph data sets. For instance, a typical problem of learning on graphs arises in chemoinformatics: In this problem one is given a large set of chemical compounds, represented as node- and edge-labeled graphs, that have a certain function (e.g., mutagenicity or toxicity) and another set of molecules that do not have this function. The task then is to accurately predict whether a new, previously unseen molecule will exhibit this function or not. A common assumption made in this problem is that molecules with similar structure have similar functional properties. The problem of measuring the similarity of graphs is therefore at the core of learning on graphs.

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

Subgraph isomorphism checking is the analogue of graph isomorphism checking in a setting in which the two graphs have different sizes. Unlike the graph isomorphism problem, the problem of subgraph isomorphism has been proven to be NP-complete (Garey and Johnson, 1979, Section 3.2.1). A slightly less restrictive measure of similarity can be defined based on the size of the largest common subgraph in two graphs, but unfortunately the problem of finding the largest common subgraph of two graphs is NP-complete as well (Garey and Johnson, 1979, Section 3.3).

Besides being computationally expensive or even intractable, similarity measures based on graph isomorphism and its variants are too restrictive in the sense that graphs have to be exactly identical or contain large identical subgraphs in order to be deemed similar by these measures. More flexible similarity measures, based on inexact matching of graphs, have been proposed in the literature. Graph comparison methods based on graph edit distances (Bunke and Allermann, 1983; Neuhaus and Bunke, 2005) are expressive similarity measures respecting the topology, as well as node and edge labels of graphs, but they are hard to parameterize and involve solving NP-complete problems as intermediate steps. Another type of graph similarity measures, optimal assignment kernels (Fröhlich et al.,

2005), arise from finding the best match between substructures of graphs. However, these kernels are not positive semidefinite in general (Vert, 2008).

Graph kernels have recently evolved into a rapidly developing branch of learning on structured data. They respect and exploit graph topology, but restrict themselves to comparing substructures of graphs that are computable in polynomial time. Graph kernels bridge the gap between graph-structured data and a large spectrum of machine learning algorithms called kernel methods (Schölkopf and Smola, 2002), that include algorithms such as support vector machines, kernel regression, or kernel PCA (see Hofmann et al. (2008) for a recent review of kernel algorithms).

Informally, a kernel is a function of two objects that quantifies their similarity. Mathematically, it corresponds to an inner product in a reproducing kernel Hilbert space (Schölkopf and Smola, 2002). Graph kernels are instances of the family of so-called R-convolution kernels by Haussler (1999). R-convolution is a generic way of defining kernels on discrete compound objects by comparing all pairs of decompositions thereof. Therefore, a new type of decomposition of a graph results in a new graph kernel.

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

## Review of graph kernels

Before we review graph kernels from the literature, we clarify our terminology. We define a graph  $G$  as a triplet  $(V, E, \ell)$ , where  $V$  is the set of vertices,  $E$  is the set of undirected edges, and  $\ell : V \rightarrow \Sigma$  is a function that assigns labels from an alphabet  $\Sigma$  to nodes in the graph<sup>1</sup>. The neighbourhood  $\mathcal{N}(v)$  of a node  $v$  is the set of nodes to which  $v$  is connected by an edge, that is  $\mathcal{N}(v) = \{v' | (v, v') \in E\}$ . For simplicity, we assume that every graph has  $n$  nodes,  $m$  edges, and a maximum degree of  $d$ . The size of  $G$  is defined as the cardinality of  $V$ .

A walk is a sequence of nodes in a graph, in which consecutive nodes are connected by an edge. A path is a walk that consists of distinct nodes only. A (*rooted*) *subtree* is a subgraph of a graph, which has no cycles, but a designated root node. A subtree of  $G$  can thus be seen as a connected subset of distinct nodes of  $G$  with an underlying tree structure. The height of a subtree is the maximum distance between the root and any other node in the subtree. Just as the notion of walk is extending the notion of path by allowing nodes to be equal, the notion of subtrees can be extended to *subtree patterns* (also called ‘tree-walks’ (Bach, 2008)), which can have nodes that are equal (see Figure 1). These repetitions of the same node are then treated as distinct nodes, such that the pattern is still a cycle-free tree. Note that all subtree kernels compare subtree *patterns* in two graphs, not (strict) subtrees.

Several different graph kernels have been defined in machine learning which can be categorized into three classes: graph kernels based on walks (Kashima et al., 2003; Gärtner

---

1. An extension of this definition and of our results to graphs with discrete edge labels is straightforward, but omitted for clarity of presentation.

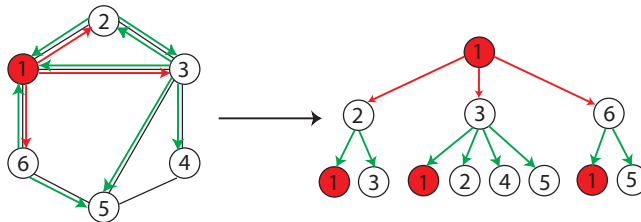


Figure 1: A subtree pattern of height 2 rooted at the node 1. Note the repetitions of nodes in the unfolded subtree pattern on the right.

et al., 2003) and paths (Borgwardt and Kriegel, 2005), graph kernels based on limited-size subgraphs (Horvath et al., 2004; Shervashidze et al., 2009), and graph kernels based on subtree patterns (Ramon and Gärtner, 2003; Mahé and Vert, 2009).

The first class, graph kernels on walks and paths, compute the number of matchings of pairs of random walks (resp. paths) in two graphs. The standard formulation of the random walk kernel, based on the direct product graph of two graphs, is computable in  $O(n^6)$  for a pair of graphs (Gärtner et al., 2003). However, the same problem can be stated in terms of Kronecker products that can be exploited to bring down the runtime complexity to  $O(n^3)$  (Vishwanathan et al., 2010). For a computer vision application, Harchaoui and Bach (2007) have proposed a dynamic programming-based approach to speed up the computation of the random walk kernel, but at the cost of considering walks of fixed size. Suard et al. (2005) and Vert et al. (2009) present other applications of random walk kernels in computer vision. The shortest path kernel by Borgwardt and Kriegel (2005) counts pairs of shortest paths having the same source and sink labels and the same length in two graphs. The runtime of this kernel scales as  $O(n^4)$ .

The second class, graph kernels based on limited-size subgraphs, so-called *graphlets*, represent graphs as counts of all types of subgraphs of size  $k \in \{3, 4, 5\}$ . There exist efficient computation schemes for these kernels based on sampling or exploitation of the low maximum degree of graphs (Shervashidze et al., 2009), but these apply to unlabeled graphs only.

The first kernel from the third class, subtree kernels, was defined by Ramon and Gärtner (2003). Intuitively, to compare graphs  $G$  and  $G'$ , this kernel iteratively compares all matchings between neighbours of two nodes  $v$  from  $G$  and  $v'$  from  $G'$ . In other words, for all pairs of nodes  $v$  from  $G$  and  $v'$  from  $G'$ , it counts all pairs of matching substructures in subtree patterns rooted at  $v$  and  $v'$ . The runtime complexity of the subtree kernel for a dataset of  $N$  graphs is  $O(N^2 n^2 h 4^d)$ . For a detailed description of this kernel, see Section 3.2.2.

The subtree kernels in (Mahé and Vert, 2009) and (Bach, 2008) refine the Ramon-Gärtner kernel for applications in chemoinformatics and hand-written digit recognition. Both Mahé and Vert (2009) and Bach (2008) propose to consider  $\alpha$ -ary subtrees with at most  $\alpha$  children per node. This restricts the set of matchings to matchings of up to  $\alpha$

nodes, but the runtime complexity is still exponential in this parameter  $\alpha$ , which both papers describe as feasible on small graphs (with approximately twenty nodes on average) with many distinct node labels.

It is a general limitation of all the aforementioned graph kernels that they scale poorly to large, labeled graphs with more than 100 nodes: In the worst case, none of them scale better than  $O(n^3)$ . The efficient comparison of large, labeled graphs remained an unsolved challenge for almost a decade. We present a general definition of graph kernels that encompasses many previously known graph kernels, and instances of which are efficient to compute for both unlabeled and discretely labeled graphs with thousands of nodes next. Moreover, in terms of prediction accuracy in graph classification tasks its instances are competitive with or outperform other state-of-the-art graph kernels.

The remainder of this article is structured as follows. In Section 2, we describe the Weisfeiler-Lehman isomorphism test that our main contribution is based on. In Section 3, we describe what we call the Weisfeiler-Lehman graphs and our proposed general graph kernels based on them, followed by some examples. In Section 4, we compare these kernels to each other, as well as to a set of five other state-of-the-art graph kernels. We report results on kernel computation runtime and classification accuracy on graph benchmark datasets.

## 2. The Weisfeiler-Lehman test of isomorphism

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

The key idea of the algorithm is to augment the node labels by the sorted set of node labels of neighbouring nodes, and compress these augmented labels into new, short labels. These steps are then repeated until the node label sets of  $G$  and  $G'$  differ, or the number of iterations reaches  $n$ . See Figure 2, a-d, for an illustration of these steps (note however, that the two graphs in the figure would directly be identified as non-isomorphic by the Weisfeiler-Lehman test, as their label sets are already different in the beginning).

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

The Weisfeiler-Lehman algorithm terminates after step 4 of iteration  $i$  if  $\{l_i(v)|v \in V\} \neq \{l_i(v')|v' \in V'\}$ , that is, if the sets of newly created labels are not identical in  $G$  and  $G'$ . The

---

2. For unlabeled graphs, node labels  $M_0(v) := l_0(v)$  can be initialized with letters corresponding one to one to node degrees  $|\mathcal{N}(v)|$ .

---

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

---

- 1: Multiset-label determination
    - For  $i = 0$ , set  $M_i(v) := l_0(v) = \ell(v)$ .<sup>2</sup>
    - For  $i > 0$ , assign a multiset-label  $M_i(v)$  to each node  $v$  in  $G$  and  $G'$  which consists of the multiset  $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$ .
  - 2: Sorting each multiset
    - Sort elements in  $M_i(v)$  in ascending order and concatenate them into a string  $s_i(v)$ .
    - Add  $l_{i-1}(v)$  as a prefix to  $s_i(v)$  and call the resulting string  $s_i(v)$ .
  - 3: Label compression
    - Sort all of the strings  $s_i(v)$  for all  $v$  from  $G$  and  $G'$  in ascending order.
    - Map each string  $s_i(v)$  to a new compressed label, using a function  $f : \Sigma^* \rightarrow \Sigma$  such that  $f(s_i(v)) = f(s_i(w))$  if and only if  $s_i(v) = s_i(w)$ .
  - 4: Relabeling
    - Set  $l_i(v) := f(s_i(v))$  for all nodes in  $G$  and  $G'$ .
- 

graphs are then not isomorphic. If the sets are identical after  $n$  iterations, it means that either  $G$  and  $G'$  are isomorphic, or the algorithm has not been able to determine that they are not isomorphic (see Cai et al., 1992, for examples of graphs that cannot be distinguished by this algorithm or its higher-dimensional variants). As a side note, we mention that the 1-dimensional Weisfeiler-Lehman algorithm has been shown to be a valid isomorphism test for almost all graphs (Babai and Kucera, 1979).

Note that in Algorithm 1 we used the same node labeling functions  $\ell, l_0, \dots, l_h$  for both  $G$  and  $G'$  in order not to overload the notation. We will continue using this notation throughout the paper and assume without loss of generality that the domain of these functions  $\ell, l_0, \dots, l_h$  is the set of all nodes in our dataset of graphs, which corresponds to  $V \cup V'$  in the case of Algorithm 1.

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

**Link with subtree patterns** Note that the compressed labels  $l_i(v)$  correspond to subtree patterns of height  $i$  rooted at  $v$  (see Figure 1 for an illustration of subtree patterns).

### 3. The general Weisfeiler-Lehman kernels

In this section, we first define the Weisfeiler-Lehman graph sequence and the general graph kernels based on them. We then present two instances of this kernel, the Weisfeiler-Lehman subtree kernel (Section 3.2) and the Weisfeiler-Lehman shortest path kernel (Section 3.3).

#### 3.1 The Weisfeiler-Lehman kernel framework

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

**Definition 1** Define the Weisfeiler-Lehman graph at height  $i$  of the graph  $G = (V, E, \ell) = (V, E, l_0)$  as the graph  $G_i = (V, E, l_i)$ . We call the sequence of Weisfeiler-Lehman graphs

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

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

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

**Definition 2** Let  $k$  be any kernel for graphs, that we will call the base kernel. Then the Weisfeiler-Lehman kernel with  $h$  iterations with the base kernel  $k$  is defined as

$$k_{WL}^{(h)}(G, G') = k(G_0, G'_0) + k(G_1, G'_1) + \dots + k(G_h, G'_h), \quad (1)$$

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

**Theorem 3** Let the base kernel  $k$  be any positive semidefinite kernel on graphs. Then the corresponding Weisfeiler-Lehman kernel  $k_{WL}^{(h)}$  is positive semidefinite.

**Proof** Let  $\phi$  be the feature mapping corresponding to the kernel  $k$ :

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

We have

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

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

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

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

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

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

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

$$k_{WL}^{(h)}(G, G') = \alpha_0 k(G_0, G'_0) + \alpha_1 k(G_1, G'_1) + \dots + \alpha_h k(G_h, G'_h).$$

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

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

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

### 3.2 The Weisfeiler-Lehman subtree kernel

In this section we present the Weisfeiler-Lehman subtree kernel (Shervashidze and Borgwardt, 2009), which is a natural instance of Definition 2.

**Definition 4** *Let  $G$  and  $G'$  be graphs. Define  $\Sigma_i \subseteq \Sigma$  as the set of letters that occur as node labels at least once in  $G$  or  $G'$  at the end of the  $i$ -th iteration of the Weisfeiler-Lehman algorithm. Let  $\Sigma_0$  be the set of original node labels of  $G$  and  $G'$ . Assume all  $\Sigma_i$  are pairwise disjoint. Without loss of generality, assume that every  $\Sigma_i = \{\sigma_{i1}, \dots, \sigma_{i|\Sigma_i|}\}$  is ordered.*

Define a map  $c_i : \{G, G'\} \times \Sigma_i \rightarrow \mathbb{N}$  such that  $c_i(G, \sigma_{ij})$  is the number of occurrences of the letter  $\sigma_{ij}$  in the graph  $G$ .

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

$$k_{WLSubtree}^{(h)}(G, G') = \langle \phi_{WLSubtree}^{(h)}(G), \phi_{WLSubtree}^{(h)}(G') \rangle, \quad (2)$$

where

$$\phi_{WLSubtree}^{(h)}(G) = (c_0(G, \sigma_{01}), \dots, c_0(G, \sigma_{0|\Sigma_0|}), \dots, c_h(G, \sigma_{h1}), \dots, c_h(G, \sigma_{h|\Sigma_h|})),$$

and

$$\phi_{WLSubtree}^{(h)}(G') = (c_0(G', \sigma_{01}), \dots, c_0(G', \sigma_{0|\Sigma_0|}), \dots, c_h(G', \sigma_{h1}), \dots, c_h(G', \sigma_{h|\Sigma_h|})).$$

That is, the Weisfeiler-Lehman subtree kernel counts common *original and compressed labels* in two graphs. See Figure 2 for an illustration.

**Theorem 5** *The Weisfeiler-Lehman subtree kernel on a pair of graphs  $G$  and  $G'$  can be computed in time  $O(hm)$ .*

**Proof** This follows directly from the definition of the Weisfeiler-Lehman subtree kernel and the runtime complexity of the Weisfeiler-Lehman test, as described in Section 2. ■

The following theorem shows that (2) is indeed a special case of the general Weisfeiler-Lehman kernel (1).

**Theorem 6** *Let the base kernel  $k$  be a function counting pairs of matching node labels in two graphs:*

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

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

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

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

Adding up these sums for all  $i \in \{0, 1, \dots, h\}$  gives us  $k_{WL}^{(h)}(G, G') = k_{WLSubtree}^{(h)}(G, G')$ . ■

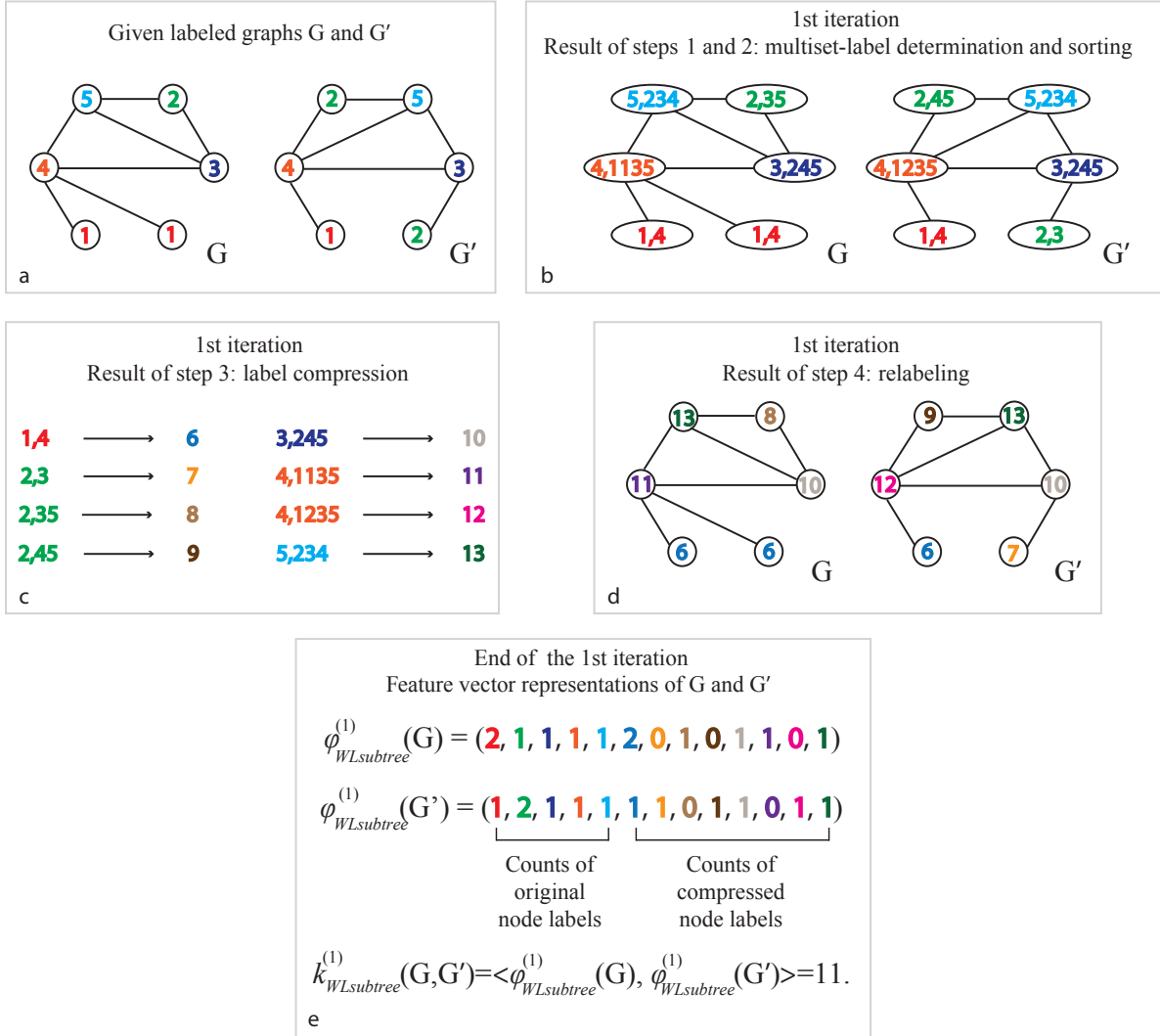


Figure 2: Illustration of the computation of the Weisfeiler-Lehman subtree kernel with  $h = 1$  for two graphs. Here  $\{1, 2, \dots, 13\} \in \Sigma$  are considered as letters. Note that compressed labels denote subtree patterns: For instance, if a node has label 8, this means that there is a subtree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

### 3.2.1 PERFORMING THE WEISFEILER-LEHMAN SUBTREE KERNEL ON MANY GRAPHS

To compute the Weisfeiler-Lehman subtree kernel on  $N$  graphs, we propose Algorithm 2, which improves over the naive,  $N^2$ -fold application of the kernel from Definition 4. We now process all  $N$  graphs simultaneously and conduct the steps given in Algorithm 2 on each graph  $G$  in each of  $h$  iterations.

---

**Algorithm 2** One iteration of the Weisfeiler-Lehman subtree kernel computation on  $N$  graphs

---

- 1: Multiset-label determination
    - Assign a multiset-label  $M_i(v)$  to each node  $v$  in  $G$  which consists of the multiset  $\{l_{i-1}(u) \mid u \in \mathcal{N}(v)\}$ .
  - 2: Sorting each multiset
    - Sort elements in  $M_i(v)$  in ascending order and concatenate them into a string  $s_i(v)$ .
    - Add  $l_{i-1}(v)$  as a prefix to  $s_i(v)$ .
  - 3: Label compression
    - Map each string  $s_i(v)$  to a compressed label using a hash function  $f : \Sigma^* \rightarrow \Sigma$  such that  $f(s_i(v)) = f(s_i(w))$  if and only if  $s_i(v) = s_i(w)$ .
  - 4: Relabeling
    - Set  $l_i(v) := f(s_i(v))$  for all nodes in  $G$ .
- 

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

One way of implementing  $f$  is to sort all neighbourhood strings using Radix Sort, as done in step 4 in Algorithm 1. The resulting complexity of this step would be linear in the sum of the size of the current alphabet and the total length of strings, that is  $O(Nn + Nm) = O(Nm)$ . An alternative implementation of  $f$  would be by means of a perfect hash function.

**Theorem 7** *For  $N$  graphs, the Weisfeiler-Lehman subtree kernel with  $h$  iterations on all pairs of these graphs can be computed in  $O(Nhm + N^2hn)$ .*

**Proof** Naive application of the kernel from Definition 4 for computing an  $N \times N$  kernel matrix would require a runtime of  $O(N^2hm)$ . One can improve upon this runtime complexity by computing  $\phi_{WLsubtree}^{(h)}$  explicitly for each graph and only then taking pairwise inner products.

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

The effort of computing  $\phi_{WLsubtree}^{(h)}$  on all  $N$  graphs in  $h$  iterations is then  $O(Nhm)$ , assuming that  $m > n$ . To get all pairwise kernel values, we have to multiply all feature vectors, which requires a runtime of  $O(N^2hn)$ , as each graph  $G$  has at most  $hn$  non-zero entries in  $\phi_{WLsubtree}^{(h)}(G)$ . In Section 4.1, we empirically show that the first term  $Nhm$  dominates the overall runtime in practice. ■

While our Weisfeiler-Lehman subtree kernel matches neighbourhoods of nodes in a graph exactly, one could also think of other strategies of comparing node neighbourhoods, and still retain the favourable runtime of our graph kernel. In research that was published in parallel to ours, Hido and Kashima (2009) present such an alternative kernel based on node neighbourhoods which uses hash functions and logical operations on bit-representations of node labels and which also scales linearly in the number of edges.

### 3.2.2 THE RAMON-GÄRTNER SUBTREE KERNEL

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

$$k_{RG}^{(h)}(G, G') = \sum_{v \in V} \sum_{v' \in V'} k_{RG,h}(v, v'), \quad (3)$$

where

$$k_{RG,h}(v, v') = \begin{cases} \delta(\ell(v), \ell(v')), & \text{if } h = 0 \\ \lambda_v \lambda_{v'} \delta(\ell(v), \ell(v')) \sum_{R \in \mathcal{M}(v, v')} \prod_{(w, w') \in R} k_{RG, h-1}(w, w'), & \text{if } h > 0, \end{cases} \quad (4)$$

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

$$\mathcal{M}(v, v') = \{R \subseteq \mathcal{N}(v) \times \mathcal{N}(v') \mid (\forall (u, u') \in R : u = w \Leftrightarrow u' = w') \wedge (\forall (u, u') \in R : \ell(u) = \ell(u'))\}. \quad (5)$$

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

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

### 3.2.3 LINK TO THE WEISFEILER-LEHMAN SUBTREE KERNEL

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

**Theorem 8** *The kernel  $k_{rec}^{(h)}$  defined as*

$$k_{rec}^{(h)}(G, G') = \sum_{i=0}^h \sum_{v \in V} \sum_{v' \in V'} k_{rec,i}(v, v'), \quad (6)$$

where

$$k_{rec,i}(v, v') = \begin{cases} \delta(\ell(v), \ell(v')), & \text{if } i = 0 \\ \delta(\ell(v), \ell(v')) k_{rec,i-1}(v, v') \max_{R \in \mathcal{M}(v, v')} \prod_{(w, w') \in R} k_{rec,i-1}(w, w'), & \text{if } i > 0 \text{ and } \\ & \mathcal{M} \neq \emptyset \\ 0, & \text{if } i > 0 \text{ and } \\ & \mathcal{M} = \emptyset, \end{cases} \quad (7)$$

$\delta$  is the indicator function again, and

$$\mathcal{M}(v, v') = \{R \subseteq \mathcal{N}(v) \times \mathcal{N}(v') \mid |R| = |\mathcal{N}(v)| = |\mathcal{N}(v')| \\ \wedge (\forall (u, u'), (w, w') \in R : u = w \Leftrightarrow u' = w') \wedge (\forall (u, u') \in R : \ell(u) = \ell(u'))\}, \quad (8)$$

is equivalent to the Weisfeiler-Lehman subtree kernel  $k_{WLSubtree}^{(h)}$ .

In other words,  $\mathcal{M}(v, v')$  is the set of exact matchings of the neighbourhoods of  $v$  and  $v'$ . It is nonempty only in the case where the neighbourhoods of  $v$  and  $v'$  have exactly the same size and the multisets of labels of their neighbours  $\{\ell(u) \mid u \in \mathcal{N}(v)\}$  and  $\{\ell(u') \mid u' \in \mathcal{N}(v')\}$  are identical.

**Proof** We prove this theorem by induction over  $h$ . Induction initialisation  $h = 0$ :

$$k_{WLSubtree}^{(0)} = \langle \phi_{WLSubtree}^{(0)}(G), \phi_{WLSubtree}^{(0)}(G) \rangle = \sum_{j=1}^{|\Sigma_0|} c_0(G, \sigma_{0j}) c_0(G', \sigma_{0j}) = \quad (9)$$

$$= \sum_{v \in V} \sum_{v' \in V'} \delta(\ell(v), \ell(v')) = k_{rec}^{(0)}, \quad (10)$$

where  $\Sigma_0$  is the initial alphabet of node labels and  $c_0(G, \sigma_{0j})$  is the number of occurrences of the letter  $\sigma_{0j}$  as a node label in  $G$ . The equality follows from the definitions of  $k_{rec}^{(h)}$  and  $k_{WLSubtree}^{(h)}$ .

Induction step  $h \rightarrow h + 1$ : Assume that  $k_{WLSubtree}^{(h)} = k_{rec}^{(h)}$ . Then

$$k_{rec}^{(h+1)} = \sum_{v \in V} \sum_{v' \in V'} k_{rec,h+1}(v, v') + \sum_{i=0}^h \sum_{v \in V} \sum_{v' \in V'} k_{rec,i}(v, v') = \quad (11)$$

$$= \sum_{j=1}^{|\Sigma_{h+1}|} c_{h+1}(G, \sigma_{h+1,j}) c_{h+1}(G', \sigma_{h+1,j}) + k_{WLSubtree}^{(h)} = k_{WLSubtree}^{(h+1)}, \quad (12)$$

where the equality of (11) and (12) follows from the fact that  $k_{h+1,rec}(v, v') = 1$  if and only if the labels and neighbourhoods of  $v$  and  $v'$  are identical, that is, if  $f(s_{h+1}(v)) = f(s_{h+1}(v'))$ .  $\blacksquare$

Theorem 8 highlights the following differences between the Weisfeiler-Lehman and the Ramon-Gärtner subtree kernels: In Equation (6), Weisfeiler-Lehman considers all subtrees

up to height  $h$ , whereas the Ramon-Gärtner kernel looks at subtrees of exactly height  $h$ . In Equations (7) and (8), the Weisfeiler-Lehman subtree kernel checks whether the neighbourhoods of  $v$  and  $v'$  match exactly, while the Ramon-Gärtner kernel considers all pairs of matching subsets of the neighbourhoods of  $v$  and  $v'$  in Equation (5). In our experiments, we examine the empirical differences between these two kernels in terms of runtime and prediction accuracy on classification benchmark datasets (see Section 4.2).

### 3.3 The Weisfeiler-Lehman shortest path kernel

Another example of the general Weisfeiler-Lehman subtree kernels that we consider is the Weisfeiler-Lehman shortest path kernel. Here we take the base kernel to be a node-labeled shortest path kernel (Borgwardt and Kriegel, 2005) to derive a new instance of the general Weisfeiler-Lehman kernel.

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

According to (1), we have

$$k_{WL\text{shortest path}}^{(h)} = k_{SP}(G_0, G'_0) + k_{SP}(G_1, G'_1) + \dots + k_{SP}(G_h, G'_h). \quad (13)$$

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

Let us first consider the Dirac ( $\delta$ ) kernel on the shortest path lengths, which means that the similarity of two paths in two graphs equals 1 if they have exactly the same length and identically labeled start and end nodes and 0 otherwise. Then, in iteration  $i$  of the Weisfeiler-Lehman relabeling, we can bound the number of features, triplets  $(a, b, p)$  where  $a, b \in |\Sigma_i|$  are ordered start and end node labels and  $p \in \mathbb{N}_0$  the shortest path length, by  $\frac{|\Sigma_i|(|\Sigma_i|+1)}{2}P$ . It is easy to notice by looking at the Algorithm 1 that for each  $i \in \{0, \dots, h-1\}$ ,  $|\Sigma_i| \leq |\Sigma_{i+1}|$ . Therefore, if we compute the shortest path kernel by first explicitly computing  $\phi_{SP}(G)$  for each  $G$  in the dataset, the computation will get increasingly expensive in each iteration  $i$  of the Weisfeiler-Lehman relabeling.

However, in a more general case where we do not assume that edges are unweighted and use any kernel  $k_l$  (not necessarily the Dirac kernel) on shortest path lengths, or the alphabet size gets prohibitively large, computing the feature map explicitly might not be possible or reasonable any more. In this case, the runtime of computing  $k_{SP}(G_i, G'_i)$  will not depend on  $i$  any more. It will scale as  $O(n^4)$  for each pair of graphs as we have to compare all pairs of the  $O(n^2)$  shortest path lengths, and  $O(N^2n^4)$  for the whole dataset.

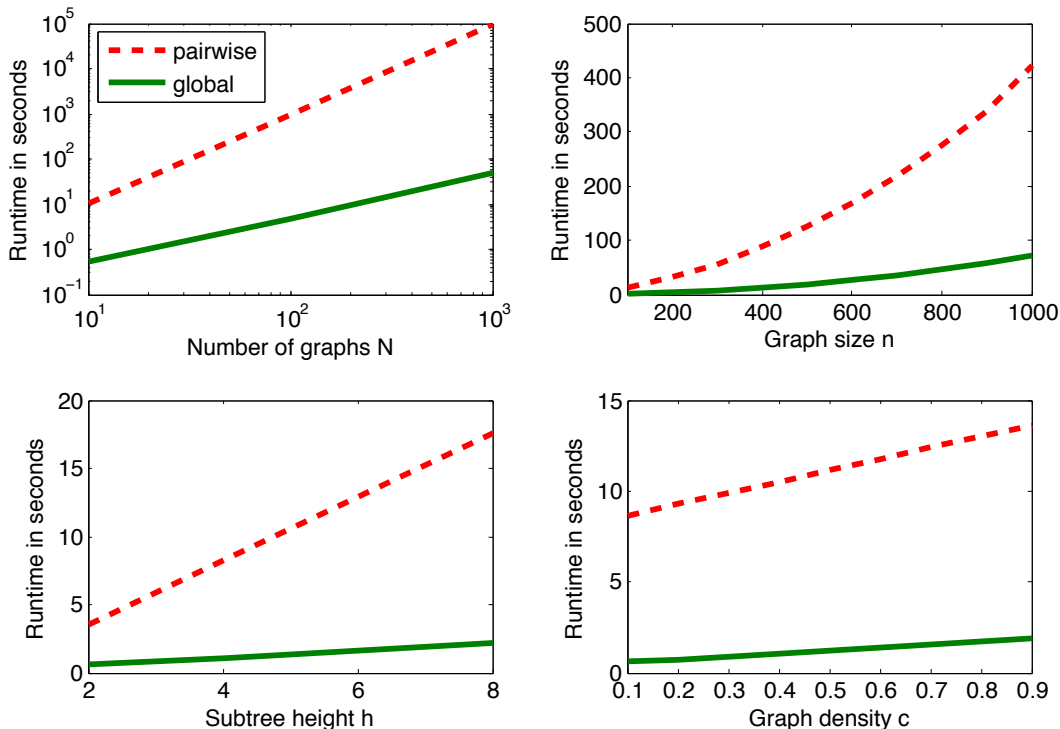


Figure 3: Runtime in seconds for kernel matrix computation on synthetic graphs using the pairwise (red, dashed) and the global (green, solid) computation schemes for the Weisfeiler-Lehman subtree kernel (Default values: dataset size  $N = 10$ , graph size  $n = 100$ , subtree height  $h = 4$ , graph density  $c = 0.4$ ).

### 3.4 Other Weisfeiler-Lehman kernels

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

## 4. Experiments

In this section, we first empirically study the runtime behaviour of the Weisfeiler-Lehman subtree kernel on synthetic graphs (Section 4.1). Next, we compare the Weisfeiler-Lehman subtree kernel and the Weisfeiler-Lehman shortest path kernel to state-of-the-art graph ker-

nels in terms of kernel computation runtime and classification accuracy on graph benchmark datasets (Section 4.2).

#### 4.1 Runtime behaviour of Weisfeiler-Lehman subtree kernel

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

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

For each individual experiment, we generated  $N$  graphs with  $n$  nodes, and inserted edges randomly until the number of edges reached  $\lfloor cn(n-1)/2 \rfloor$ . We then computed the pairwise and the global WL kernel on these synthetic graphs. We report CPU runtimes in seconds in Figure 3, as measured in Matlab R2008a on an Apple MacPro with 3.0GHz Intel 8-Core with 16GB RAM.

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

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

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

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

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

## 4.2 Graph classification

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

**Datasets** We employed the following datasets in our experiments: MUTAG, NCI1, NCI109, ENZYMES and D&D. MUTAG (Debnath et al., 1991) is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds labeled according to whether or not they have a mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium*. We also conducted experiments on two balanced subsets of NCI1 and NCI109, which classify compounds based on whether or not they are active in an anti-cancer screen (Wale and Karypis (2006) and <http://pubchem.ncbi.nlm.nih.gov>). ENZYMES is a dataset of protein tertiary structures obtained from (Borgwardt et al., 2005) consisting of 600 enzymes from the BRENDA enzyme database (Schomburg et al., 2004). In this case the task is to correctly assign each enzyme to one of the 6 EC top level classes. D&D is a dataset of 1178 protein structures (Dobson and Doig, 2003). Each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Ångstroms apart. Note that nodes are labeled in all datasets. The prediction task is to classify the protein structures into enzymes and non-enzymes.

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

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

From the family of kernels based on limited-size subgraphs, we chose an extension of the graphlet kernel by Shervashidze et al. (2009) that counts common induced labeled connected subgraphs of size 3.

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

Note that whenever possible, we used fast computation schemes based on explicitly computing the feature map (similar to that in Algorithm 2) before taking the inner product, in order to speed up kernel computation. In particular, we used this technique for computing shortest path and graphlet kernels. For connected 3-node graphlet kernels it is rather intuitive to imagine the explicit feature map: First, we have only 4 types of different graphlets with 3 nodes. Second, for each type of graphlet we can determine the number of possible labelings of the three nodes as a function of the size of the node label alphabet. In the case of shortest paths, the explicit feature map may or may not exist. In our case, as edges were not weighted, we used the number of edges in a path as a measure of its length. In addition, we used the Dirac kernel on shortest path distances. This allowed us

to explicitly compute the feature map corresponding to the shortest path kernel for each graph.

We performed 10-fold cross-validation of C-Support Vector Machine Classification using LIBSVM (Chang and Lin, 2001), using 9 folds for training and 1 for testing. All parameters of the SVM were optimised on the training dataset only. To exclude random effects of fold assignments, we repeated the whole experiment 10 times. We report average prediction accuracies and standard deviations in Tables 1 and 2.

We chose  $h$  for our Weisfeiler-Lehman subtree kernel by cross-validation on the training dataset for  $h \in \{1, \dots, 10\}$ , which means that we computed 10 different WL kernel matrices in each experiment. We reported the total runtime of these computations (*not* the average per kernel matrix). It is worth mentioning that rather small values of  $h$ , such as 1, 2, or 3, systematically gave the best results for all datasets used.

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

**Results** In terms of runtime, the Weisfeiler-Lehman subtree kernel could easily scale up even to graphs with thousands of nodes. On D&D, subtree-patterns of height up to 10 were computed in 11 minutes, while no other comparison method could handle this dataset in less than half an hour. The shortest path kernel and the WL shortest path kernel were competitive to the WL subtree kernel on smaller graphs (MUTAG, NCI1, NCI109, ENZYMES), but on D&D their runtime degenerated to more than 23 hours for the shortest path kernel and to more than a year for the WL shortest path kernel. The Ramon and Gärtner kernel was computable on MUTAG in approximately 40 minutes, but it finished computation in more than a month on ENZYMES and the computation took even longer time on larger datasets. The random walk kernel was competitive on MUTAG and ENZYMES in terms of runtime, but took more than a week on each of the NCI datasets more than a month on D&D. The graphlet kernel was faster than our WL kernel on MUTAG and the NCI datasets, and about a factor of 3 slower on D&D. However, this efficiency came at a price, as the kernel based on size-3 graphlets turned out to lead to poor accuracy levels on four datasets.

On NCI1, NCI109 and ENZYMES, the Weisfeiler-Lehman shortest path kernel reached the highest accuracy. On all three datasets, the Weisfeiler-Lehman subtree kernel yielded the second best result. While on NCI1 and NCI109 this second best result is close to the best result, on ENZYMES the WL shortest path kernel dramatically improved over the WL subtree kernel. On D&D the shortest path and graphlet kernels yielded similarly good results, while on NCI1 and NCI109 the Weisfeiler-Lehman subtree kernel improved by more than 8% the best accuracy attained by other methods. On MUTAG, the WL kernels reached the third and the fourth best accuracy levels among all methods considered. The labeled size-3 graphlet kernel achieved low accuracy levels, except on D&D. The random walk and

Method/Dataset	MUTAG	NCI1	NCI109	ENZYMES	D & D
WL subtree	82.05 ( $\pm 0.36$ )	82.19 ( $\pm 0.18$ )	82.46 ( $\pm 0.24$ )	46.42 ( $\pm 1.35$ )	79.78 ( $\pm 0.36$ )
WL shortest path	83.78 ( $\pm 1.46$ )	84.55 ( $\pm 0.36$ )	83.53 ( $\pm 0.30$ )	59.05 ( $\pm 1.05$ )	79.43 ( $\pm 0.55$ )
Ramon & Gärtner	85.72 ( $\pm 0.49$ )	61.86 ( $\pm 0.27$ )	61.67 ( $\pm 0.21$ )	13.35 ( $\pm 0.87$ )	57.27 ( $\pm 0.07$ )
$p$ -random walk	79.19 ( $\pm 1.09$ )	58.66 ( $\pm 0.28$ )	58.36 ( $\pm 0.94$ )	27.67 ( $\pm 0.95$ )	66.64 ( $\pm 0.83$ )
Random walk	80.72 ( $\pm 0.38$ )	64.34 ( $\pm 0.27$ )	63.51 ( $\pm 0.18$ )	21.68 ( $\pm 0.94$ )	71.70 ( $\pm 0.47$ )
Graphlet count	75.61 ( $\pm 0.49$ )	66.00 ( $\pm 0.07$ )	66.59 ( $\pm 0.08$ )	32.7 ( $\pm 1.20$ )	78.59 ( $\pm 0.12$ )
Shortest path	87.28 ( $\pm 0.55$ )	73.47 ( $\pm 0.11$ )	73.07 ( $\pm 0.11$ )	41.68 ( $\pm 1.79$ )	78.45 ( $\pm 0.26$ )

Table 1: Prediction accuracy ( $\pm$  standard deviation) on graph classification benchmark datasets

Dataset	MUTAG	NCI1	NCI109	ENZYMES	D & D
Maximum # nodes	28	111	111	126	5748
Average # nodes	17.93	29.87	29.68	32.63	284.32
# labels	7	37	38	3	82
Number of graphs	188	4110	4127	600	1178
WL subtree	6"	7'20"	7'21"	20"	11'
WL shortest path	1.5"	2'20"	2'23"	1'3"	484 days
Ramon & Gärtner	40'6"	81 days	80.5 days	38 days	103 days
$p$ -random walk	4'42"	4.5 days	5 days	10'	4 days
Random walk	12"	8.5 days	9 days	12'19"	48 days
Graphlet count	3"	1'27"	1'27"	25"	30'21"
Shortest path	2"	4'38"	4'39"	5"	23h 17'2"

Table 2: CPU runtime for kernel computation on graph classification benchmark datasets

the  $p$ -random walk kernels, as well as the Ramon-Gärtner kernel, were less competitive to kernels that performed the best on datasets other than MUTAG.

To summarize, the WL subtree kernel turned out to be competitive in terms of runtime on all smaller datasets, fastest on the large protein dataset, and its accuracy levels were competitive on all datasets. The WL shortest path kernel achieved the highest accuracy level on three out of five datasets, and was competitive on the remaining datasets.

## 5. Conclusions

We have defined a general framework for constructing graph kernels on graphs with unlabeled or discretely labeled nodes. Instances of our framework include a fast subtree kernel that combines scalability with the ability to deal with node labels. Our kernels are competitive in terms of accuracy with state-of-the-art kernels on several classification benchmark datasets, even reaching the highest accuracy level on four out of five datasets. Moreover, in terms of runtime on large graphs, instances of our kernel outperform other kernels, even the efficient computation schemes for random walk kernels (Vishwanathan et al., 2010) and graphlet kernels (Shervashidze et al., 2009) that were recently developed.

Our new kernels open the door to applications of graph kernels on large graphs in bioinformatics, for instance, protein function prediction via detailed graph models of protein structure on the amino acid level, or on gene networks for phenotype prediction. An exciting algorithmic question for further studies will be to consider kernels on graphs with continuous or high-dimensional node labels and their efficient computation.

## Acknowledgements

The authors would like to thank Ulrike von Luxburg for useful comments. N.S. was funded by the DFG project “Kernels for Large, Labeled Graphs (LaLa)”.

## References

- L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *Proceedings Symposium on Foundations of Comp. Sci.*, pages 39–46, 1979.
- F. R. Bach. Graph kernels between point clouds. In *ICML*, pages 25–32, 2008.
- K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the International Conference on Data Mining*, pages 74–81, 2005.
- K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(Suppl 1):i47–i56, Jun 2005.
- H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253, 1983.
- J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.
- C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- 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. *J Med Chem*, 34: 786–797, 1991.
- P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol*, 330(4):771–783, Jul 2003.
- H. Fröhlich, J. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *Proc. of ICML*, pages 225–232, Bonn, Germany, 2005.
- M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

- T. Gärtner, P.A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. Warmuth, editors, *Sixteenth Annual Conference on Computational Learning Theory and Seventh Kernel Workshop, COLT*. Springer, 2003.
- Z. Harchaoui and F. Bach. Image classification with segmentation graph kernels. In *CVPR*, 2007.
- D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99 - 10, Computer Science Department, UC Santa Cruz, 1999.
- S. Hido and H. Kashima. A linear-time graph kernel. In *ICDM*, pages 179–188, 2009.
- T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 2008.
- T. Horvath, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2004.
- H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20<sup>th</sup> International Conference on Machine Learning*, Washington, DC, United States, 2003.
- P. Mahé and J.-P. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, 2009.
- K. Mehlhorn. *Data Structures and Efficient Algorithms*. Springer, 1984.
- M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):503–514, 2005.
- J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD’03), 2003.
- B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002.
- I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic Acids Research*, 32D:431–433, 2004.
- N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1660–1668, 2009.
- N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, 2009.
- F. Suard, V. Guigue, A. Rakotomamonjy, and A. Benschrair. Pedestrian detection using stereo-vision and graph kernels. In *IEEE Symposium on Intelligent Vehicles*, 2005.

- J.-P. Vert. The optimal assignment kernel is not positive definite. *CoRR*, abs/0801.4061, 2008.
- J.-P. Vert, T. Matsui, S. Satoh, and Y. Uchiyama. High-level feature extraction using svm with walk-based graph kernel. In *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '09*, pages 1121–1124, Washington, DC, USA, 2009. IEEE Computer Society.
- S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.
- N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. In *Proc. of ICDM*, pages 678–689, Hong Kong, 2006.
- B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, Ser. 2*, 9, 1968.