A Persistent Weisfeiler–Lehman Procedure for Graph Classification

Bastian Rieck [*] [1] **Christian Bock** [*] [1] **Karsten Borgwardt** [1]

Abstract

The Weisfeiler–Lehman graph kernel exhibits competitive performance in many graph classification tasks. However, its subtree features are not able to capture connected components and cycles, topological features known for characterising graphs. To extract such features, we leverage propagated node label information and transform unweighted graphs into metric ones. This permits us to augment the subtree features with topological information obtained using *persistent homology*, a concept from topological data analysis. Our method, which we formalise as a generalisation of Weisfeiler–Lehman subtree features, exhibits favourable classification accuracy and its improvements in predictive performance are mainly driven by including cycle information.

1. Introduction

Graph-structured data sets are ubiquitous in a variety of different application domains, each of them posing a separate challenge while also requiring different tasks to be solved. A common task involves *graph classification*, for which a variety of methods exists. These methods comprise convolutional neural networks (Duvenaud et al. 2015), recurrent neural networks (Lei et al. 2017), or Hilbert space methods (Vishwanathan et al. 2010), the latter also being referred to as *graph kernels*. While several approaches for defining graph kernels exist, the most common one uses the $\mathcal{R}$-convolution framework (Haussler 1999), which makes it possible to define the similarity between two graphs as a function of the similarity of their substructures.

Substructures that have been used for graph classification range from graphlets (Shervashidze et al. 2009), i.e. small non-isomorphic graphs of fixed size, over shortest paths (Borgwardt & Kriegel 2005), to random walks (Gärtner et al. 2003, Kashima et al. 2003, Sugiyama & Borgwardt 2015). One of the most powerful substructures is the set of *subtree patterns* (Ramon & Gärtner 2003), i.e. patterns based on rooted subgraphs of a graph. Their computational complexity made their applicability somewhat limited, until the Weisfeiler–Lehman (WL) graph kernel framework (Shervashidze & Borgwardt 2009, Shervashidze et al. 2011) was developed. Properly trained, it still constitutes the state-of-the-art method for many graph classification tasks. The framework is based on the idea of iteratively propagating (node) label information through a graph, leading to a feature vector representation that can be used to assess the dissimilarity of two graphs.

One of the disadvantages of this framework is that its relabelling step, i.e. the step in which subtree patterns are being *compressed*, is somewhat “brittle”: labels are only compared with a Dirac kernel, making their dissimilarity a coarse function. Moreover, the subtree feature vector only contains counts of compressed labels and can neither account for their relevance with respect to the topology of the graph nor capture connected components and cycles, both of which are important and interpretable features for characterising graphs (Rieck et al. 2018, Sizemore et al. 2017). We thus propose an enhancement of the original WL stabilisation procedure that uses recent advances in topological data analysis (Munch 2017) to alleviate these issues. Our contributions are as follows:

- We measure the relevance of topological features (connected components and cycles) in graphs and use them to define a novel set of WL subtree features, which we show to be a generalised version of the original ones.
- We develop a topology-based kernel that uses an iterative variant of the WL stabilisation procedure to classify non-attributed graphs.
- We demonstrate that our proposed features perform favourably on a range of graph classification benchmark data sets. In particular, we empirically show that the inclusion of *cycle* information yields classification accuracy improvements over state-of-the-art methods.

[*]Equal contribution [1]Department of Biosystems Science and Engineering, ETH Zurich, 4058 Basel, Switzerland. Correspondence to: Bastian Rieck <bastian.rieck@bsse.ethz.ch>, Karsten Borgwardt <karsten.borgwardt@bsse.ethz.ch>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

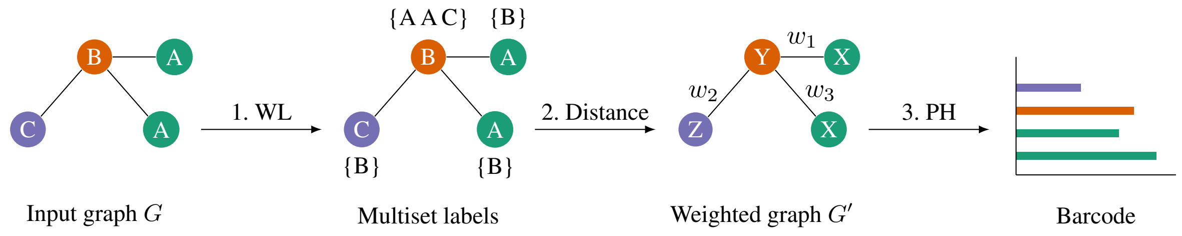

Figure 1. Schematic illustration of our proposed P-WL feature generation process for a labelled graph G. Step 1: In an initial neighbourhood propagation step, Weisfeiler–Lehman (WL) label multisets are generated for each node (depicted by vertex annotations in curly braces). Step 2: Next, we calculate a metric between these multisets according to Definition 2 in order to obtain edge weights. Together with the compressed multisets (X, Y, Z), we thus construct a weighted relabelled graph G'. Step 3: Finally, we calculate persistent homology (PH) features as described in Section 2.3.2 and attribute them to each node in G'. The set of graph features is iteratively extended by h repetitions of Steps 1–3 with G' and extending its feature vector.

2. Method

Our method builds on the seminal Weisfeiler–Lehman stabilisation algorithm (Douglas 2011, Weisfeiler & Lehman 1968) and on recent advances in topological data analysis, namely *persistent homology* (Edelsbrunner & Harer 2008). In the following, we first give a brief overview of the WL algorithm. We then show how to leverage multi-scale information to obtain a set of features based on topological invariants (connected components and cycles) and elucidate its connection to the WL subtree features (Shervashidze & Borgwardt 2009, Shervashidze et al. 2011).

2.1. Weisfeiler–Lehman Stabilisation

The 1-dimensional WL stabilisation algorithm was initially developed as a test for graph isomorphism (Weisfeiler & Lehman 1968). It is also known as *colour refinement* or *canonical labelling*. In the following, we assume that we are given a graph $G = (V, E)$ with a label function $\mathrm{l}\colon V \to \Sigma$ that assigns each vertex $v \in V$ a label $\mathrm{l}(v)$. The labels constitute the initial colours $C^{(0)}$ of the algorithm so that $C^{(0)}(v) = \mathrm{l}(v)$ for every vertex $v \in V$. For each vertex v and each iteration h, the algorithm creates a new set of colours from the colour $C^{(h-1)}(v)$ and the colours $C^{(h-1)}(w)$ of every vertex w that is adjacent to v. This multiset of colours is then mapped to a new colour. The mapping is unique in the sense that the same multiset always maps to the same colour. Afterwards, the next iteration begins, and the process stops as soon as the list of colours stabilises, meaning that for some h, no new colours are being created.

This procedure can be used to detect whether two graphs G and G' are isomorphic to each other, i.e. whether there exists a bijection between their vertex sets V and V': for two graphs with at most n vertices each, the algorithm may stop as soon as $n+1$ distinct colours have been encountered. In this case, the two graphs are guaranteed *not* to be isomorphic to each other. If, however, the two colour sequences of the graphs are equal, there is a high probability that the two graphs are isomorphic (Babai & Kucera 1979).

On an abstract level, the WL iteration can be seen as a process that, given two labelled graphs G and G', generates a sequence of labelled graphs $(G_0, G'_0), \ldots, (G_h, G'_h)$. To simplify the notation, we will also write $\mathrm{l}_u^{(h)}$ to denote the label of a vertex u in iteration h of the procedure, and thus drop the notion of vertex colours. Shervashidze et al. (2011) showed that any valid (graph) kernel can be used to obtain a valid graph kernel for the whole sequence of graphs. Among others, they present the *Weisfeiler–Lehman subtree kernel*, which represents each iteration of the algorithm by a set of feature vectors that contain the *counts* of each individual label. Each iteration h of the stabilisation procedure thus results in a feature vector

$$\phi_{\mathrm{WL}}^{(h)} := \left[\mathrm{c}^{(h)}(l_0), \mathrm{c}^{(h)}(l_1), \ldots\right], \tag{1}$$

where l_i denotes the ith compressed label that appears in iteration h, and $\mathrm{c}^{(h)}$ is a counting function. Feature vectors of all iterations are concatenated to form one large feature vector $\phi_{\mathrm{WL}}(G)$ for every graph G. The similarity of two graphs is then assessed by calculating a linear kernel between their feature vectors. This feature vector assignment only requires a perfect hashing function; by definition of the algorithm, no more than $2n$ unique labels are required per iteration. The algorithm is highly efficient, having a runtime complexity of $\mathcal{O}(hm)$, where h denotes the number of iterations and m denotes the number of edges. One disadvantage is that the perfect hashing scheme is somewhat brittle—one change of a label in a neighbourhood will result in two vertices *never* getting assigned the same compressed label again. The assignment of counts to a certain label is thus somewhat arbitrary. We aim to address this issue by providing an algorithm that also takes into account how much any label contributes to the topology of the graph at multiple scales. This requires using methods from the field of computational topology.

2.2. Computational Topology

Computational topology refers to a family of methods that captures *topological features* such as connected components. Originally developed for the analysis of high-dimensional manifold data sets (Edelsbrunner & Harer 2010), it has recently seen increased use in the analysis of graphs or networks (Carstens 2016, Horak et al. 2009, Rieck et al. 2018, Rieck et al. 2019). We briefly review the most important topics to formalise the problem and develop some intuition.

2.2.1. Topological Features

Given a topological space $\mathbb{M}$, such as a manifold, k-dimensional topological features are represented as elements of the kth homology group $\mathrm{H}_k(\mathbb{M})$. In lower dimensions, i.e. for $k \in \{0, 1, 2\}$, these features are also known as *connected components*, *cycles*, and *voids*, respectively. Even though there are several algorithms for the extraction of higher-dimensional topological features from graphs (Jonsson 2008), we restrict[1] ourselves to $k \in \{0, 1\}$ in this paper, which corresponds to connected components and cycles. Given a graph G, the Euler characteristic formula (Gross & Tucker 1987, p. 135) relates these two concepts. It states that the number of cycles in G is

$$\beta_1 = m - n + \beta_0, \tag{2}$$

where m denotes the number of edges, n denotes the number of vertices, and β_0 denotes the number of connected components. Using a disjoint set data structure (Cormen et al. 2009, Chapter 21), the number of connected components of a graph can be determined in almost linear time of its size (Hopcroft & Tarjan 1973).

The number of connected components β_0 and the number of cycles β_1 constitute a simple graph invariant: if $(\beta_0, \beta_1) \neq (\beta_0', \beta_1')$ for two graphs G, G', the graphs cannot be isomorphic (Hatcher 2002, pp. 103–133). Equality of the tuples is a necessary—but not sufficient—condition for the existence of an isomorphism.

2.2.2. Persistent Homology

For many graphs, in particular labelled ones, information of the form (β_0, β_1) is too coarse to be useful. Thus, *persistent homology* (Edelsbrunner & Harer 2008, Edelsbrunner & Morozov 2014) was developed to describe topological spaces in terms of multi-scale topological features. For clarity of exposition, we only describe this process in terms of graphs[2]. Given a graph $G = (V, E)$, persistent homology requires a *filtration*, i.e. a sequence of nested subgraphs such that

$$\emptyset \subseteq G_0 \subseteq G_1 \subseteq \dots \subseteq G_{k-1} \subseteq G_k = G \tag{3}$$

and each G_i is of the form $G_i = (V, E_i \subseteq E)$. Hence, only the *edges* change between different steps of the iteration. There are alternative formulations of filtrations that permit the vertex set to vary (Rieck et al. 2018), but we do not consider them in this paper because of their higher complexity. Since the vertex set remains fixed and only edges are added in each step, the number of connected components can only *decrease* (by introducing an edge that merges two connected components, thus "destroying" one of them) or remain the same (by introducing an edge that connects two vertices that are already in the same connected component) as we go from G_i to G_{i+1}. Analogously, the number of cycles can only *increase* (by introducing an edge that closes or "creates" a cycle).

Persistent homology thus tracks changes in connected components and cycles over the complete filtration: for each component that is destroyed in a graph G_i of the filtration, a tuple of the form $(0, i)$ is stored by the algorithm, while every cycle that is created in, say, a graph G_j is assigned the tuple (j, ∞). These tuples are also known as *index persistence tuples* (Zomorodian & Carlsson 2005). Given an associated sequence of real-valued weights

$$a_0 \leq a_1 \leq \dots \leq a_{k-1} \leq a_k \tag{4}$$

with the same cardinality as the filtration, each tuple (i, j) corresponds to a pair (a_i, a_j) that can be seen as a point in $\mathbb{R}^2_\infty := \mathbb{R} \times \{\mathbb{R} \cup \{\infty\}\}$. If both a_i and a_j are finite, the quantity $\mathrm{pers}(a_i, a_j) := |a_j - a_i|$ is referred to as the *persistence* of a pair (Edelsbrunner et al. 2002); large values imply that a feature persists over multiple scales because it appears in many subgraphs of the filtration and might thus have a higher importance (Carlsson 2009, Edelsbrunner et al. 2002). If $a_j = \infty$, we set $\mathrm{pers}(a_i, a_j) := |a_i|$. These pairs are often represented as horizontal line segments in $\mathbb{R}^2$, where the vertical coordinate y may be assigned arbitrarily, so that each pair becomes a line from (a_i, y) to (a_j, y) while segments of infinite length are cropped. This representation is known as a *persistence barcode* (Ghrist 2008); together with a related representation, it can be shown to be a stable descriptor of topological activity (Cohen-Steiner et al. 2007).

Similar to the Euler characteristic formula from Eq. 2, persistence tuples induce a relation on the graph: for tuples that correspond to the merge of two connected components, one vertex is *paired* with one edge, while for tuples that give rise to a cycle, exactly one edge of the cycle is *marked*. An edge cannot be paired with a vertex and marked at the same time (Edelsbrunner et al. 2002), so a graph with n

[1]It is possible to generalise our methods to include higher-dimensional features, even though their calculation has a higher computational complexity.

[2]It works equally well for higher-dimensional manifolds, but this is beyond the scope of this paper.

vertices and m edges always gives rise to exactly $n + m$ persistence tuples. The uniqueness of the pairing relation permits us to write $\operatorname{pers}(v)$ and $\operatorname{pers}(u, v)$ to denote the persistence of a vertex v (which we consider to create a certain connected component) or an edge (u, v) (which we consider to create a cycle), respectively. Hence, we do not have to calculate the persistence of edges that are paired with a vertex because they are already being accounted for.

2.3. A Persistent Weisfeiler–Lehman Procedure

In the following, we describe our methods. We first define a novel procedure for leveraging the WL multisets to obtain weighted graphs, after which we describe how to obtain *persistent* subtree features. We also explain how to rephrase the original WL subtree features as a special case of our method. Furthermore, we demonstrate how to include cycle features, which (a) have been largely ignored in the literature so far due to their computational complexity (Horváth et al. 2004), and (b) turn out to be beneficial for the classification performance.

2.3.1. A Filtration for Node-Labelled Graphs

Typically, filtrations in computational topology require the use of a *metric*[3], i.e. a distance function, defined on substructures of the objects in an input data set (Ghrist 2008). The resulting topological features then represent the data set at different scales; features that only exist at small scales are considered to be noise. However, there are currently no algorithms for obtaining a filtration that handles graphs with node labels. We thus propose a new method that employs the WL stabilisation procedure (Section 2.1). More precisely, in iteration h of the procedure, we define a distance between the label multisets $C^{(h)}(u)$ and $C^{(h)}(v)$ of two vertices u and v. We then use this distance to obtain *edge weights*, which in turn can be used to obtain a filtration. First, we define a distance between multisets.

Definition 1 (Multiset distance)**.** *Let $A = \{l_1^{a_1}, l_2^{a_2}, \ldots\}$ and $B = \{l_1^{b_1}, l_2^{b_2}, \ldots\}$ be two multisets that are defined over the same label alphabet $\{l_1, l_2, \ldots\}$, where a_i and b_i refer to the number of occurrences of label l_i in each multiset, respectively. We transform each set into a vector of its counts, leading to $\mathbf{x} := [a_1, a_2, \ldots]$ and $\mathbf{y} := [b_1, b_2, \ldots]$. Given $p \in \mathbb{R}_{>0}$, we calculate the multiset distance between $\mathbf{x}$ and $\mathbf{y}$ as*

$$\mathrm{d}_{\mathrm{M}}(\mathbf{x}, \mathbf{y}) := \left(\sum_i |a_i - b_i|^p\right)^{\frac{1}{p}}, \tag{5}$$

i.e. the pth Minkowski distance. The parameter p has the role of a smoothing parameter.

[3]In the supplementary materials, we discuss alternative dissimilarity measures, such as the Kullback–Leibler divergence.

For a vertex u and its neighbour v in a graph, we extend the previous distance to a distance between the compressed label of the previous WL iteration $(h - 1)$ and the multiset label of the current iteration (h).

Definition 2 (Label distance)**.** *Given two adjacent vertices u and v in a graph, let $\mathrm{l}_u^{(h)}$ and $\mathrm{l}_v^{(h)}$ refer to their multiset label in iteration h of the stabilisation procedure. We define their distance as*

$$\mathrm{d}_{\mathrm{L}}(u, v) := \left[\mathrm{l}_u^{(h-1)} \neq \mathrm{l}_v^{(h-1)}\right] + \mathrm{d}_{\mathrm{M}}\left(\mathrm{l}_u^{(h)}, \mathrm{l}_v^{(h)}\right) + \tau, \tag{6}$$

where $[\cdot]$ is an Iverson bracket, i.e. a function that is 1 whenever the condition in the bracket is satisfied, and 0 otherwise, and $\tau \in \mathbb{R}_{>0}$ is an offset. We use $\tau := 1$ in this paper.

We can show that this function[4] constitutes a metric on labelled graphs, thus yielding a suitable filtration.

Theorem 1. $\mathrm{d}_{\mathrm{L}}(\cdot, \cdot)$ *is a metric on a node-labelled graph provided that the graph does not have self-loops, i.e. edges whose source and target node are the same.*

Proof. We first note that the function in the Iverson bracket is a metric, namely the *discrete metric* on a topological space (Ó Searcóid 2007, p. 4). Moreover, the multiset distance is a metric by its definition because the Minkowski distance is a metric for $p > 0$. Last, τ can be seen as a *uniform metric* on the graph edges, assigning each edge (u, v) the same value. This only violates the conditions of a metric if self-loops exist, because a metric d has to satisfy $\mathrm{d}(x, y) = 0$ if and only if $x = y$. This case never occurs unless the graph has a self-loop. We may thus treat each term as a metric on the graph vertices. Since the sum of two metrics is a metric, the claim follows. □

Using the preceding theorem, we thus obtain a weighted graph at each stabilisation step h, from which we derive a filtration by sorting all edge weights in ascending order, i.e. $w_0 \leq w_1 \leq w_2 \leq \ldots$, and define the set of edges E_i for G_i in Eq. 3 as

$$E_i := \{(u, v) \mid (u, v) \in E, \mathrm{w}(u, v) \leq w_i\}, \tag{7}$$

where $\mathrm{w}(u, v)$ refers to the weight of an edge (u, v). Since weights encode distances, large edge weights are assigned whenever the two vertices of an edge are dissimilar in terms of their neighbouring labels.

It is interesting to observe the effect of h on the distances. Intuitively, larger values of h have a "smoothing" effect on the distance between two nodes. For the initial case $h = 0$,

[4]We also present a recursively-defined variant of this function in the supplementary materials.

Algorithm 1 Persistent Subtree Feature Generation

```
Input: Graph $G = (V, E)$, number of iterations $H$
 1: for $h \in \{0, \dots, H-1\}$ do
 2:     $\phi^{(h)}_{\text{P-WL}} := \emptyset$, $\phi^{(h)}_{\text{P-WL-C}} := \emptyset$
 3:     // Get WL multiset labels (Shervashidze et al. 2011)
 4:     $L^{(h)} := \{\mathrm{l}^{(h)}_v \mid v \in V\} \leftarrow \texttt{WL\_LABELS}(G, h)$
 5:     // Use Eq. 6 to assign each edge $(u, v)$ its weight
 6:     for all $(u, v) \in E$ do
 7:         $\mathrm{w}(u, v) \leftarrow \mathrm{d_L}(u, v)$
 8:     end for
 9:     // Compress and re-assign vertex labels
10:     for all $v \in V$ do
11:         $\mathrm{l}^{(h)}_v \leftarrow \texttt{COMPRESS}\left(\mathrm{l}^{(h-1)}_v, \mathrm{l}^{(h)}_v\right)$
12:     end for
13:     // Calculate persistent subtree features
14:     for all $v \in V$ do
15:         $\phi^{(h)}_{\text{P-WL}}\left[\mathrm{l}^{(h)}_v\right] \leftarrow \phi^{(h)}_{\text{P-WL}}\left[\mathrm{l}^{(h)}_v\right] + \mathrm{pers}(v)^p$
16:     end for
17:     // Calculate persistent cycle features
18:     for all $(u, v) \in E$ do
19:         $\phi^{(h)}_{\text{P-WL-C}}\left[\mathrm{l}^{(h)}_v\right] \leftarrow \phi^{(h)}_{\text{P-WL-C}}\left[\mathrm{l}^{(h)}_v\right] + \mathrm{pers}(u, v)^p$
20:     end for
21: end for
22: return $\left[\phi^{(0)}_{\text{P-WL}}, \phi^{(0)}_{\text{P-WL-C}}, \dots \phi^{(H-1)}_{\text{P-WL}}, \phi^{(H-1)}_{\text{P-WL-C}}\right]$
```

Eq. 6 reduces to

$$\mathrm{d_L}(u, v) = \begin{cases} \tau & \text{if } \mathrm{l}(u) = \mathrm{l}(v) \\ \tau + 1 & \text{otherwise} \end{cases} \tag{8}$$

because there are no previous labels to fall back to. Since subsequent iterations include more information about neighbours, this rigid initial distance will become progressively smoother.

2.3.2. P-WL FOR NODE-LABELLED GRAPHS

Having introduced the required terminology and developed a suitable filtration, we are now ready to describe a persistent form of the WL subtree feature generation method for node-labelled graphs. We refer to this method as P-WL; Algorithm 1 shows the full feature calculation procedure. In the following, we will provide details for the individual steps. A high-level description of our algorithm in iteration h consists of the following operations:

1. Perform the WL stabilisation procedure and assign every vertex v in the graph a multiset label $\mathrm{l}^{(h)}_u$.
2. Use the metric from Definition 6 to assign a weight to each edge of the graph. Create a filtration by sorting all edges according to their edge weight.
3. Use the filtration to calculate persistent homology of the weighted graph.

For a graph G with n vertices and m cycles, this procedure results in n persistence tuples of the form $\{(0, w_{i_1}), (0, w_{i_2}), \dots, (0, w_{i_n})\}$ and m tuples of the form $\{(w_{j_1}, \infty), (w_{j_2}, \infty), \dots, (w_{j_m}, \infty)\}$, where each w represents a certain edge weight of the graph. Letting $\{l_0, l_1, \dots\}$ refer to the sequence of compressed vertex labels at iteration h, we define the *persistent subtree feature vector* as

$$\phi^{(h)}_{\text{P-WL}} := \left[\mathfrak{p}^{(h)}(l_0), \mathfrak{p}^{(h)}(l_1), \dots\right], \tag{9}$$

where $\mathfrak{p}^{(h)}(\cdot)$ sums persistence values of vertices with a given label, i.e.

$$\mathfrak{p}^{(h)}(l_i) := \sum_{\mathrm{l}(v) = l_i} \mathrm{pers}(v)^p, \tag{10}$$

and $p > 0$ is a smoothing factor, which we choose to be the same as the one used for the distance calculations in Eq. 5. Subsequently, we will use $\phi^{(h)}_{\text{P-WL}}[\mathrm{l}(v)]$ to denote the respective component of a label $\mathrm{l}(v)$ in a feature vector.

This equation can be seen as a more complex variant of Eq. 1. It does not merely count how often a certain label occurs, but each occurrence is weighted according to its persistence, i.e. according to the relevance of the corresponding topological feature in the graph. Feature vectors are then concatenated for different values of h just like for the original WL subtree features. In addition to these features, our filtration also results in information about the cycles, without any computational overhead. More precisely, while listing cycles in graphs is known to be computationally demanding (Horváth et al. 2004), the pairing relation as introduced at the end of Section 2.2.2 makes it possible to count cycles in a graph. This leads us to define the *persistent cycle feature vector* as

$$\phi^{(h)}_{\text{P-WL-C}} := \left[\mathfrak{z}^{(h)}(l_0), \mathfrak{z}^{(h)}(l_1), \dots\right], \tag{11}$$

where $\mathfrak{z}^{(h)}(\cdot)$ sums persistence values of *cycle edges* whose source or target node matches a given label, i.e.

$$\mathfrak{z}^{(h)}(l_i) := \sum_{l_i \in \mathrm{l}(u,v)} \mathrm{pers}(u, v)^p, \tag{12}$$

where $p > 0$ is the same smoothing factor as in Eq. 10, $(u, v) \in E$, and $\mathrm{l}(u, v) := \{\mathrm{l}(u), \mathrm{l}(v)\}$. This counts all cycles *twice*, which does not impact classification performance. Subsequently, we refer to the method that integrates cycle features as P-WL-C. As we will demonstrate in the experimental section, the integration of cycle features can be beneficial for several data sets, in particular those that exhibit pronounced cycles (such as the `PTC-MR` data set of molecular graphs).

Properties of $\phi^{(h)}_{\text{P-WL}}$ and $\phi^{(h)}_{\text{P-WL-C}}$ Since we use the distance from Definition 2, we know that $\operatorname{pers}(v) \geq \tau$ for every vertex v: in the worst case, all labels coincide, such that all parts of the sum become zero except for τ. As $\tau > 0$, the ith entry in $\phi^{(h)}_{\text{P-WL}}$ is zero if and only if the ith label does *not* occur in the respective graph. This demonstrates the necessity of using τ; a classifier trained on these features should be able to distinguish between features that are not important for the topological structure (small persistence values) and those that are non-existent (zero). The same reasoning applies to the persistent cycle features.

Computational complexity Calculating persistent features from a filtration requires an additional sorting of all edges, which has a computational complexity of $\mathcal{O}(m \log m)$ for a graph with m edges. The calculation of persistence tuples has a complexity of $\mathcal{O}(m \cdot \alpha(m))$, where $\alpha(\cdot)$ is the extremely slow-growing inverse of the Ackermann function; in practice, it is linear in the number of edges (Chen & Kerber 2013). This brings up the total runtime complexity to $\mathcal{O}(hm \log m)$, in comparison to the complexity $\mathcal{O}(hm)$ of the original WL subtree feature calculation (Shervashidze et al. 2011).

2.3.3. $\phi^{(h)}_{\text{WL}}$ as a Special Case of $\phi^{(h)}_{\text{P-WL}}$

It is possible to rephrase the generation of the original WL subtree feature vectors, i.e. $\phi^{(h)}_{\text{WL}}$, in terms of persistent feature vectors, provided the graph does not exhibit self-loops. In this case, let $\mathrm{d}(u, v) := 1$ for all edges (u, v) in the graph. Since no self-loops exist, this is a discrete metric. Now $w_{i_1} = w_{i_2} = \cdots = 1$ for all persistence tuples, so Eq. 9 simplifies to a *count* function again. As a consequence, the feature vector is equal to the one calculated in Eq. 1, making $\phi^{(h)}_{\text{WL}}$ a special case of $\phi^{(h)}_{\text{P-WL}}$.

This permits us to define a variant of P-WL for uniform edge weights. In the preceding paragraph, we already established the equivalence of the vertex features to the original subtree features. Our filtration additionally leads to cycle features, which degenerate to cycle counts in the uniform case. We use these counts to obtain a feature vector analogously to Eq. 11. The resulting method, which we refer to as P-WL-UC (uniform metric with cycles), enriches the WL subtree features with cycle counts.

2.3.4. P-WL for Non-Attributed Graphs

Application domains such as social network analysis tend to make use of *non-attributed* graphs, i.e. graphs without any attributes (node labels, edge labels, etc.). While it is possible to apply the WL subtree feature generation approach here through the assignment of uniform labels in the first step of the algorithm, we propose a new method that is inspired by the stabilisation procedure while also being able to leverage more topological information. Our method is motivated by the insight that non-attributed graphs result in a set of compressed WL labels that correspond to the degree of a vertex. The distance between each of these labels is uniform by construction; hence the algorithm treats all different degrees the same. By contrast, we can incorporate degree information in a continuous manner if we restrict ourselves to topological descriptors. For a non-attributed graph $G = (V, E)$, we perform the following steps:

1. Assign each vertex $v \in V$ its degree $\deg(v)$ as an initial attribute value $f(v)$.
2. Replace $f(v)$ by $(\deg(v)+1)^{-1}(f(v)+\sum_{u\sim v} f(u))$, where $u \sim v$ denotes adjacent vertices u and v.
3. Assign each edge $(u, v) \in E$ a weight of $\max(f(u), f(v))$ and obtain a filtration by sorting all edges according to their edge weight.
4. Calculate persistent homology of the filtration.

Step 2 can be repeated multiple times. This has the effect of smoothing degree information within a graph; in the limit, all vertices will have a similar $f(v)$ value. The topological features calculated by this procedure are of the same form as in Section 2.3.2. Letting $\mathcal{D}$ refer to the persistence tuples of G that correspond to the vertices, i.e. the tuples of the form $\{(0, w_{i_1}), (0, w_{i_2}), \ldots, (0, w_{i_n})\}$, we can calculate a topological kernel $\mathrm{k_{PSS}}$ between the tuples $\mathcal{D}$ and $\mathcal{D}'$ of two graphs G and G' as

$$\mathrm{k_{PSS}}^{(h)}(\mathcal{D}, \mathcal{D}') := c^{-1} \sum_{x\in\mathcal{D},\ y\in\mathcal{D}'} \mathrm{e}^{\frac{-\|x-y\|^2}{8\sigma}} - \mathrm{e}^{\frac{-\|x-\overline{y}\|^2}{8\sigma}}, \quad (13)$$

where $c := 8\pi\sigma$ for a smoothing parameter $\sigma \in \mathbb{R}$, and $\bar{y}$ denotes a set of tuples that has been "mirrored" such that $(0, w_{i_1})$ becomes $(w_{i_1}, 0)$, for example. This expression, the persistence scale-space-kernel (Reininghaus et al. 2015), is a valid kernel between sets of persistence tuples. We define the *degree-based persistent WL kernel* as

$$\mathrm{k_{P\text{-}WL}}(\mathcal{D}, \mathcal{D}') := \sum_{h} \mathrm{k_{PSS}}^{(h)}(\mathcal{D}_G, \mathcal{D}_{G'}), \quad (14)$$

which is a valid kernel because the sum of kernels is also a kernel (Hofmann et al. 2008). In practice, $\mathrm{k_{PSS}}$ is general and can be applied to all valid filtrations; we restrict its usage, and hence our $\mathrm{k_{P\text{-}WL}}$ kernel, to the (smoothed) degree information in this paper. In the experimental section, we will use P-WL-D to denote this kernel.

3. Experiments

In the following, we describe the practical performance of our methods on numerous graph classification benchmark data sets. Since all of our methods can be seen as generalised variants of WL subtree features (Shervashidze & Borgwardt 2009, Shervashidze et al. 2011), we primarily compare against them under the same training conditions. Moreover, we show histogram kernels to establish baselines for each of the data sets. Methods marked with an asterisk indicate that the results have initially been reported in another paper.

Setup We follow the standard setup for graph classification and perform a 10-fold cross-validation that we repeat 10 times, reporting the average and standard deviation for all runs. For hyperparameter tuning, we use an inner 5-fold cross-validation on each of the training splits to perform a grid search. Since we want to show that our proposed enrichment of the original subtree features is beneficial, we use a random forest classifier (Breiman 2001) to compare P-WL, P-WL-C, and P-WL-UC against WL. For P-WL-D and for the histogram kernels, we use a support vector machine (Cortes & Vapnik 1995). As for the hyperparameters, we choose $p \in \{1, 2\}$ for P-WL and P-WL-C, and $h \in \{0, \ldots, 10\}$ for methods based on WL features, whereas we choose $C \in \{0.1, 1, 10\}$ for training an SVM on the P-WL-D kernel values. Moreover, since we did not observe an effect in changing σ for P-WL-D, we leave $\sigma = 1.0$ fixed.

Data sets We use common graph benchmark data sets in our experiments, comprising graphs from chemoinformatics problems (Debnath et al. 1991), toxicology prediction (Helma et al. 2001), protein function/structure prediction (Borgwardt et al. 2005, Dobson & Doig 2003), carcinogenicity prediction (Wale et al. 2008), and social network analysis (Leskovec et al. 2005, Yanardag & Vishwanathan 2015).

Implementation We implemented our methods in `Python`. Please refer to our repository[5] for the code and additional experiments.

3.1. Results for Node-Labelled Graphs

Table 1 shows the results of our methods for the classification of node-labelled graphs. Two histogram kernel methods (one using vertices, the other one using edges) provide a simple baseline. Moreover, we show the accuracies that we obtain for the Weisfeiler–Lehman (WL) subtree features, plus the accuracies for the deep variant of these features (DEEP-WL). This framework (Yanardag & Vishwanathan 2015) involves a weight scheme for the subtree features, but unlike our novel topology-based assignment described in Section 2.3.2, DEEP-WL aims to learn those weights based on the data set. As the most recent state-of-the-art (SOTA) comparison partner, we use RETGK, a graph kernel based on the return probabilities of random walks (Zhang et al. 2018).

[5] `https://github.com/BorgwardtLab/P-WL`

Overall, we observe that our novel persistent features are favourable: their mean accuracy is either higher than that of the original subtree features, or very close to them; we use a bold font to indicate the highest mean accuracy, marking multiple cells whenever one of our methods is able to outperform the respective SOTA method. Interestingly, P-WL, which only uses persistent variants of the subtree features, does not outperform any method on these data sets, except for `PTC-MM`. On `NCI1` and `NCI109`, P-WL performance gets close to the best result, with a slightly smaller standard deviation. For `D&D`, P-WL exhibits the best mean accuracy of our methods, but it is still unable to match RETGK. In general, we observe that the addition of cycle features following Eq. 11 is beneficial for most data sets, and even the uniform-weighted variant of our algorithm that includes cycles, i.e. P-WL-UC, is seen to perform well.

For the toxicology challenge data sets (Helma et al. 2001) `PTC-MR`, `PTC-FR`, `PTC-MM`, and `PTC-FM`, our features are favourable in comparison to the original subtree features and the deep graph kernel features. Each of these data sets is known to be hard to classify, but our methods improve on the accuracies reported for RETGK on three out of the four data sets. Again, we observe that the inclusion of cycles positively impacts predictive performance. P-WL exhibits a higher mean accuracy than WL on `PTC-MM`, while staying very close to its performance on the other data sets. Interestingly, while P-WL-C performs best overall, the unweighted cycle-based variant P-WL-UC also has a higher mean accuracy than the SOTA method on three of these data sets, which demonstrates the relevance of cycles.

3.2. Investigating the Influence of h

By changing the parameter h, the WL sequence of graphs starts to include more global information about the neighbourhoods, while at the same time increasing the runtime. It is thus beneficial to keep h as small as possible while maintaining high predictive accuracy. To analyse the effect of h on accuracy, we exemplarily use the `PTC-MR` data set and train the same classifier with both the WL subtree features and our novel persistent features (using $p = 1$). Since the purpose of this experiment is to compare the behaviour of both types of features at the same subtree depth h, we do not perform an additional grid search, leading to slightly different accuracies than the ones reported in Table 1.

Figure 2 depicts the results. We observe that persistent features (orange, violet) tend to reach higher mean accuracies at smaller values of h. In particular P-WL-C, with its usage of cycle features, reaches its maximum accuracy for smaller values of h than WL. In our other experiments, we also observed that whenever one of our methods outperforms WL in terms of mean accuracy, it requires on average a lower value of h to do so. As h influences the size of

	D & D	MUTAG	NCI1	NCI109	PROTEINS	PTC-MR	PTC-FR	PTC-MM	PTC-FM
V-Hist	78.32 ± 0.35	85.96 ± 0.27	64.40 ± 0.07	63.25 ± 0.12	72.33 ± 0.32	58.31 ± 0.27	**68.13 ± 0.23**	66.96 ± 0.51	57.91 ± 0.83
E-Hist	72.90 ± 0.48	85.69 ± 0.46	63.66 ± 0.11	63.27 ± 0.07	72.14 ± 0.39	55.82 ± 0.00	65.53 ± 0.00	61.61 ± 0.00	59.03 ± 0.00
RetGK*	**81.60 ± 0.30**	90.30 ± 1.10	84.50 ± 0.20		75.80 ± 0.60	62.15 ± 1.60	67.80 ± 1.10	67.90 ± 1.40	63.90 ± 1.30
WL	79.45 ± 0.38	87.26 ± 1.42	85.58 ± 0.15	84.85 ± 0.19	**76.11 ± 0.64**	63.12 ± 1.44	67.64 ± 0.74	67.28 ± 0.97	64.80 ± 0.85
Deep-WL*		82.94 ± 2.68	80.31 ± 0.46	80.32 ± 0.33	75.68 ± 0.54	60.08 ± 2.55			
P-WL	79.34 ± 0.46	86.10 ± 1.37	85.34 ± 0.14	84.78 ± 0.15	75.31 ± 0.73	63.07 ± 1.68	67.30 ± 1.50	**68.40 ± 1.17**	64.47 ± 1.84
P-WL-C	78.66 ± 0.32	**90.51 ± 1.34**	85.46 ± 0.16	**84.96 ± 0.34**	75.27 ± 0.38	**64.02 ± 0.82**	67.15 ± 1.09	**68.57 ± 1.76**	**65.78 ± 1.22**
P-WL-UC	78.50 ± 0.41	85.17 ± 0.29	**85.62 ± 0.27**	**85.11 ± 0.30**	75.86 ± 0.78	**63.46 ± 1.58**	67.02 ± 1.29	**68.01 ± 1.04**	**65.44 ± 1.18**

Table 1. Classification accuracies for node-labelled graphs. Empty cells are used to indicate data sets that do not report accuracies on a certain data set. We used the highest available accuracy values from the respective publications for all cited algorithms (marked with $*$).

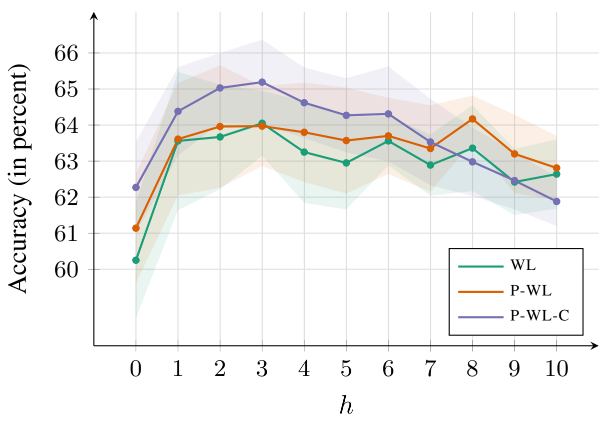

Figure 2. Performance of the WL subtree features and the persistent features for the PTC-MR data set. WL requires higher values of h to reach a suitable predictive performance.

the subtree features, obtaining higher accuracies for lower values of h is a desirable property.

3.3. Results for Non-Attributed Graphs

For non-attributed graphs, a common strategy is to use uniform labels before applying a subtree-based method. This is equivalent to using vertex degrees as categorical node labels. To determine the information carried by these labels, we remove all node labels from the PROTEINS data and apply the degree propagation procedure from Section 2.3.4. For a maximum of $h = 5$ accumulation operations, P-WL-D achieves an accuracy of 73.31 ± 0.53. While this is not on a par with WL, it demonstrates the potential of using only degree information: filtrations can be made sparse (Sheehy 2013), so P-WL-D could scale better to larger data sets than subtree features, thus providing a trade-off between runtime and predictive performance.

We also applied P-WL-D to IMDB-BINARY, a non-attributed graph data set. Using node degrees without any propagation, i.e. $h = 0$, we obtain a mean accuracy of 72.88 ± 0.55, which is higher than the accuracy reported by RetGK (72.3 ± 0.6). We can obtain an accuracy of 73.02 ± 0.95 using P-WL-UC (with uniform node labels as described above). However, on average, P-WL-UC uses subtree features with $h = 2$. This shows the favourable performance of topological features in a continuous setting. These results are to be taken with a grain of salt, though, because a recent preprint (Cai & Wang 2018) reports that all non-attributed graph data sets that are currently available to the community can equally well be classified by simple low-dimensional local degree features.

4. Discussion

We augmented Weisfeiler–Lehman (WL) subtree features with topological information to improve graph classification performance. To this end, we developed a distance between multiset labels and defined a subgraph-based filtration that generalises WL subtree features. Our method manages to efficiently include low-dimensional topological features—connected components and cycles—for classifying node-labelled and non-attributed graphs. Experiments on several data sets show a generally favourable performance of our persistent feature vectors. The integration of cycle-based features is thus shown to be capable of improving classification performance.

Future work will involve analysing different metrics for multiset labels; we also conjecture that “robust” filtrations (Anai et al. 2018, Chazal et al. 2018) will be beneficial. The use of other kernels for comparing subtree features might also lead to performance increases. Optimal assignment kernels (Kriege et al. 2016) are particularly interesting because of their favourable performance and conceptual similarity to distance measures for persistence tuples (Cohen-Steiner et al. 2007). Similarly, for P-WL-D, different kernels between persistence representations can be investigated (Carrière et al. 2017, Kusano et al. 2018). The extension of this method to higher-dimensional continuous node features is also desirable. While the theory behind multidimensional persistence (Carlsson & Zomorodian 2009) is considerably more involved than for the one-dimensional case, a recent preprint (Corbet et al. 2018) introduces a new kernel method for these representations.

Acknowledgements

We thank the anonymous reviewers for their detailed comments, which helped us improve the paper. We also thank Max Horn and Michael Moor for valuable discussions. This study was funded in part by the Alfried Krupp Prize for Young University Teachers of the Alfried Krupp von Bohlen und Halbach-Stiftung (K.B.).

References

Anai, H., Chazal, F., Glisse, M., Ike, Y., Inakoshi, H., Tinarrage, R., and Umeda, Y. DTM-based filtrations. art. arXiv:1811.04757, 2018.

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

Borgwardt, K. and Kriegel, H.-P. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*, pp. 74–81, Washington, DC, USA, 2005. IEEE Computer Society.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001.

Cai, C. and Wang, Y. A simple yet effective baseline for non-attribute graph classification. art. arXiv:1811.03508, 2018.

Carlsson, G. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

Carlsson, G. and Zomorodian, A. J. The theory of multidimensional persistence. *Discrete & Computational Geometry*, 42(1):71–93, 2009.

Carrière, M., Cuturi, M., and Oudot, S. Sliced Wasserstein kernel for persistence diagrams. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 664–673. PMLR, 2017.

Carstens, C. J. *Topology of Complex Networks: Models and Analysis*. PhD thesis, RMIT University, 2016.

Chazal, F., Fasy, B., Lecci, F., Michel, B., Rinaldo, A., and Wasserman, L. Robust topological inference: Distance to a measure and kernel distance. *Journal of Machine Learning Research*, 18(159):1–40, 2018.

Chen, C. and Kerber, M. An output-sensitive algorithm for persistent homology. *Computational Geometry*, 46(4): 435–447, 2013.

Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

Corbet, R., Fugacci, U., Kerber, M., Landi, C., and Wang, B. A kernel for multi-parameter persistent homology. art. arXiv:1809.10231, 2018.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 3rd edition, 2009.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Debnath, A. K., Lopez de Compadre, R. L., Debnath, G., Shusterman, A. J., and Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

Dobson, P. D. and Doig, A. J. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

Douglas, B. L. The Weisfeiler–Lehman method and graph isomorphism testing. art. arXiv:1101.5211, 2011.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2224–2232. Curran Associates, Inc., 2015.

Edelsbrunner, H. and Harer, J. Persistent homology—a survey. In Goodman, J. E., Pach, J., and Pollack, R. (eds.), *Surveys on discrete and computational geometry: Twenty years later*, number 453 in Contemporary Mathematics, pp. 257–282. American Mathematical Society, Providence, RI, USA, 2008.

Edelsbrunner, H. and Harer, J. *Computational topology: An introduction*. American Mathematical Society, Providence, RI, USA, 2010.

Edelsbrunner, H. and Morozov, D. Persistent homology: Theory and practice. In Latała, R., Ruciński, A., Strzelecki, P., Świątkowski, J., Wrzosek, D., and Zakrzewski, P. (eds.), *European Congress of Mathematics*. European Mathematical Society Publishing House, Zürich, Switzerland, 2014.

Edelsbrunner, H., Letscher, D., and Zomorodian, A. J. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

Gärtner, T., Flach, P., and Wrobel, S. On graph kernels: Hardness results and efficient alternatives. In Schölkopf, B. and Warmuth, M. K. (eds.), *Learning Theory and Kernel Machines*, pp. 129–143, Heidelberg, Germany, 2003. Springer.

Ghrist, R. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

Gross, J. L. and Tucker, T. W. *Topological graph theory*. Wiley, New York, NY, USA, 1987.

Hatcher, A. *Algebraic topology*. Cambridge University Press, Cambridge, England, 2002.

Haussler, D. Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz, 1999.

Helma, C., King, R. D., Kramer, S., and Srinivasan, A. The Predictive Toxicology Challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

Hofmann, T., Schölkopf, B., and Smola, A. J. Kernel methods in machine learning. *The Annals of Statistics*, 36(3): 1171–1220, 2008.

Hopcroft, J. and Tarjan, R. Algorithm 447: Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

Horak, D., Maletić, S., and Rajković, M. Persistent homology of complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03034:1–P03034:24, 2009.

Horváth, T., Gärtner, T., and Wrobel, S. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 158–167, New York, NY, USA, 2004. ACM.

Jonsson, J. *Simplicial complexes of graphs*, volume 1928 of *Lecture Notes in Mathematics*. Springer, Heidelberg, Germany, 2008.

Kashima, H., Tsuda, K., and Inokuchi, A. Marginalized kernels between labeled graphs. In Fawcett, T. and Mishra, N. (eds.), *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pp. 321–328. AAAI Press, 2003.

Kriege, N. M., Giscard, P.-L., and Wilson, R. On valid optimal assignment kernels and applications to graph classification. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1623–1631. Curran Associates, Inc., 2016.

Kusano, G., Fukumizu, K., and Hiraoka, Y. Kernel method for persistence diagrams via kernel embedding and weight factor. *Journal of Machine Learning Research*, 18(189):1–41, 2018.

Lei, T., Jin, W., Barzilay, R., and Jaakkola, T. Deriving neural architectures from sequence and graph kernels. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 2024–2033. PMLR, 2017.

Leskovec, J., Kleinberg, J., and Faloutsos, C. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pp. 177–187, New York, NY, USA, 2005. ACM.

Munch, E. A user's guide to topological data analysis. *Journal of Learning Analytics*, 4(2):47–61, 2017.

Ó Searcóid, M. *Metric spaces*. Springer Undergraduate Mathematics Series. Springer, London, England, 2007.

Ramon, J. and Gärtner, T. Expressivity versus efficiency of graph kernels. In *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74, 2003.

Reininghaus, J., Huber, S., Bauer, U., and Kwitt, R. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4741–4748, 2015.

Rieck, B., Fugacci, U., Lukasczyk, J., and Leitte, H. Clique community persistence: A topological visual analysis approach for complex networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):822–831, 2018.

Rieck, B., Togninalli, M., Bock, C., Moor, M., Horn, M., Gumbsch, T., and Borgwardt, K. Neural persistence: A complexity measure for deep neural networks using algebraic topology. In *International Conference on Learning Representations (ICLR)*, 2019.

Sheehy, D. R. Linear-size approximations to the Vietoris–Rips filtration. *Discrete & Computational Geometry*, 49 (4):778–796, 2013.

Shervashidze, N. and Borgwardt, K. Fast subtree kernels on graphs. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 22*, pp. 1660–1668. Curran Associates, Inc., 2009.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In van Dyk, D. and Welling, M. (eds.), *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pp. 488–495. PMLR, 2009.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler–Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011.

Sizemore, A., Giusti, C., and Bassett, D. S. Classification of weighted networks through mesoscale homological features. *Journal of Complex Networks*, 5(2):245–273, 2017.

Sugiyama, M. and Borgwardt, K. Halting in random walk kernels. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1639–1647. Curran Associates, Inc., 2015.

Vishwanathan, S., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

Wale, N., Watson, I. A., and Karypis, G. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14 (3):347–375, 2008.

Weisfeiler, B. and Lehman, A. A. The reduction of a graph to canonical form and the algebra which appears therein. *Nauchno–Technicheskaja Informatsia*, 9:12–16, 1968.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1365–1374, New York, NY, USA, 2015. ACM.

Zhang, Z., Wang, M., Xiang, Y., Huang, Y., and Nehorai, A. RetGK: Graph kernels based on return probabilities of random walks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 3968–3978. Curran Associates, Inc., 2018.

Zomorodian, A. J. and Carlsson, G. Computing persistent homology. *Discrete & Computational Geometry*, 33(2): 249–274, 2005.