Machine Learning (2022) 111:2601–2629
https://doi.org/10.1007/s10994-022-06131-w

A generalized Weisfeiler-Lehman graph kernel

Till Hendrik Schulz[1] · **Tamás Horváth**[1,2,3] · **Pascal Welke**[1] · **Stefan Wrobel**[1,2,3]

Received: 31 January 2021 / Revised: 3 December 2021 / Accepted: 6 February 2022 /
Published online: 27 April 2022

Abstract

After more than one decade, Weisfeiler-Lehman graph kernels are still among the most prevalent graph kernels due to their remarkable predictive performance and time complexity. They are based on a fast iterative partitioning of vertices, originally designed for deciding graph isomorphism with one-sided error. The Weisfeiler-Lehman graph kernels retain this idea and compare such labels with respect to equality. This binary valued comparison is, however, arguably too rigid for defining suitable graph kernels for certain graph classes. To overcome this limitation, we propose a generalization of Weisfeiler-Lehman graph kernels which takes into account a more natural and finer grade of similarity between Weisfeiler-Lehman labels than equality. We show that the proposed similarity can be calculated efficiently by means of the Wasserstein distance between certain vectors representing Weisfeiler-Lehman labels. This and other facts give rise to the natural choice of partitioning the vertices with the Wasserstein k-means algorithm. We empirically demonstrate on the Weisfeiler-Lehman subtree kernel, which is one of the most prominent Weisfeiler-Lehman graph kernels, that our generalization significantly outperforms this and other state-of-the-art graph kernels in terms of predictive performance on datasets which contain structurally more complex graphs beyond the typically considered molecular graphs.

Keywords Graph kernel · Weisfeiler-Lehman · Tree edit distance · Wasserstein distance

Editor: Sergio Escalera.

✉ Till Hendrik Schulz
schulzth@cs.uni-bonn.de

Tamás Horváth
horvath@cs.uni-bonn.de

Pascal Welke
welke@cs.uni-bonn.de

Stefan Wrobel
wrobel@cs.uni-bonn.de

[1] Department of Computer Science, University of Bonn, Bonn, Germany

[2] Fraunhofer IAIS, Sankt Augustin, Germany

[3] Center for Machine Learning, Sankt Augustin, Germany

1 Introduction

Since Haussler's pioneer work (Haussler, 1999) on convolution kernels over discrete structures, *graph kernels* have become one of the most common tools for learning with graphs. One prominent family of graph kernels is the *Weisfeiler-Lehman kernel* framework (Shervashidze et al., 2011), which relies on the Weisfeiler-Lehman label propagation algorithm (Weisfeiler & Lehman, 1968), originally designed for deciding isomorphism between graphs. To this day, graph kernels based on the Weisfeiler-Lehman label propagation algorithm rank among the very best state-of-the-art graph kernels on a majority of benchmark datasets, as has been experimentally shown in a recent survey (Kriege et al., 2020). Motivated by their outstanding predictive performance, in this work we focus on graph kernels based on the Weisfeiler-Lehman label propagation algorithm.

The main idea behind the algorithm in Weisfeiler and Lehman (1968) is that it iteratively relabels vertices by propagating neighborhood information. Each such label implicitly corresponds to a rooted tree, called *unfolding tree* (see Fig. 1b). This iterative vertex relabeling procedure can in fact be combined with any classical graph kernel (e.g. Gärtner et al., 2003; Borgwardt and Kriegel, 2005; Shervashidze et al., 2009, 2011). For simplicity, we limit the discussion to the most established member of the family, the *Weisfeiler-Lehman subtree kernel* (Shervashidze et al., 2011). However, we note that our approach can be applied to *all* graph kernels relying on the Weisfeiler-Lehman label propagation algorithm.

This and other Weisfeiler-Lehman graph kernels are conceptually limited to comparing Weisfeiler-Lehman vertex labels, or equivalently, unfolding trees, w.r.t. *equality*. While this comparison is extremely well-suited for deciding graph isomorphism, which was the original problem considered by Weisfeiler and Lehman, it is arguably too restrictive for defining *similarities*, in particular, graph kernels. As an example, consider the unfolding trees depicted in Fig. 1b. While T_1 visibly resembles T_2 much more than T_3, Weisfeiler-Lehman graph kernels simply treat them all as unequal and are thus unable to *quantify* the apparent difference among the pairwise similarities between these three unfolding trees.

Motivated by these considerations, we *generalize* Weisfeiler-Lehman graph kernels by *relaxing* the above strict comparison of unfolding trees. In particular, instead of

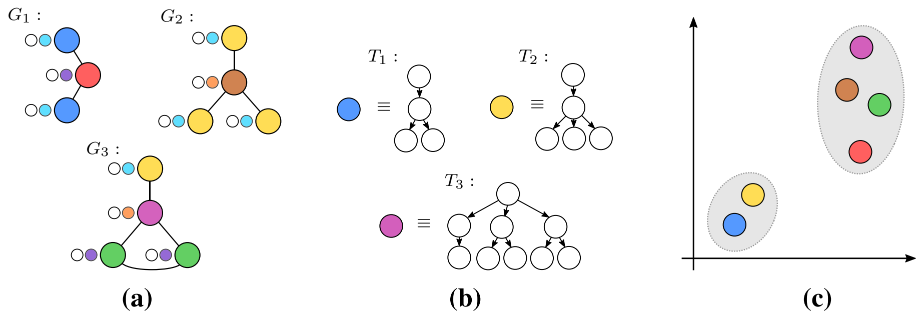


Fig. 1 **a** depicts (initially unlabeled) graphs where vertices are labeled with the first two Weisfeiler-Lehman labels (colored). **b** shows the rooted unfolding trees corresponding to the blue, yellow and pink WL-labels, each representing a neighborhood. T_1 (blue) differs from T_2 (yellow) by only a single vertex while it differs from T_3 (pink) by significantly more. The tree edit distance between T_1 and T_2 is therefore much smaller than that between T_1 and T_3. **c** conceptually visualizes the latent space representing the pairwise tree edit distances between unfolding trees. Clusterings in this space identify groups of pairwise similar unfolding trees

distinguishing between Weisfeiler-Lehman labels (or equivalently, unfolding trees) by the binary valued equality relation, we propose a *natural* similarity measure to compare them on a much *finer* grade. This distance between Weisfeiler-Lehman labels is a *modified* tree edit distance between their respective unfolding trees, which provides a *semantically* adequate comparison for this kind of trees. We show that in contrast to more general tree edit distances, this distance can in fact be *efficiently* calculated.

The key concept of our generalization is to identify groups of *similar* Weisfeiler-Lehman labels by *clustering* (visualized in Fig. 1c). The elements within a cluster are then treated as *equal* labels. That is, we generalize the ordinary Weisfeiler-Lehman graph kernels by regarding two unfolding trees equivalent if they belong to the same cluster, i.e., have a *small* distance to each other. In this way, the ordinary Weisfeiler-Lehman graph kernel becomes the *special* case in which labels are considered equivalent if and only if they have distance zero. For partitioning the Weisfeiler-Lehman labels, we use *Wasserstein k-means* clustering (Irpino et al. 2014). This choice is motivated by our result that our adaptation of tree edit distance between unfolding trees can in fact be reformulated in terms of the Wasserstein distance.

We have empirically evaluated the predictive performance of our generalization of the Weisfeiler-Lehman subtree kernel on various real-world and synthetic datasets. The experimental results clearly show that while our more general approach does not result in an improvement on small molecular graphs, which are *sparse* and *structurally simple*, it considerably outperforms state-of-the-art graph kernels, and most importantly the *ordinary* Weisfeiler-Lehman subtree kernel, on datasets containing *dense* and *structurally diverse* graphs.

The main contributions of this paper can be summarized as follows:

- We generalize the Weisfeiler-Lehman graph kernels by considering a finer similarity measure between Weisfeiler-Lehman labels than the binary valued comparison used in the original Weisfeiler-Lehman graph kernels.
- To do this, we introduce a specifically adapted tree edit distance for unfolding trees which provides a natural distance definition between the corresponding Weisfeiler-Lehman labels and propose a polynomial-time algorithm for computing this type of tree edit distance.
- We show that the concept of Wasserstein k-means clustering (Irpino et al. 2014) can be used for partitioning Weisfeiler-Lehman unfolding trees (or equivalently Weisfeiler-Lehman labels) w.r.t. the above natural tree edit distance.
- We empirically evaluate our generalized kernel on various benchmark datasets and show that it significantly outperforms state-of-the-art graph kernels, including the ordinary Weisfeiler-Lehman subtree kernel on graph datasets *beyond* the typically considered molecular graphs.

The rest of the paper is organized as follows. We collect the necessary notions in Sect. 2, define our adapted notion of the tree edit distance and discuss its algorithmic aspects in Sect. 3. We present our generalization of Weisfeiler-Lehman graph kernels in Sect. 4 and cover related work in Sect. 5. Finally, we report the empirical results in Sect. 6 and conclude in Sect. 7.

2 Preliminaries

Graphs. An *(undirected) graph* $G = (V, E, \ell)$ consists of a finite set V of vertices, a set $E \subseteq \{X \subseteq V : |X| = 2\}$ of edges, and a label function $\ell : V \to \Sigma$ for some finite alphabet Σ. When G is clear from the context, we use $n := |V|$ and $m := |E|$. For $v \in V$, $\mathcal{N}(v)$ is the set of *neighbors* of node v. Two graphs G, G' are *isomorphic*, denoted $G \equiv G'$, if there exists a bijective function between the vertices of G and those of G' preserving all edges and labels in both directions. A *(rooted) tree* is a connected graph $T = (V, E)$ that has $n - 1$ edges and a *root* $r(T) \in V$. For any $v \in V \setminus \{r(T)\}$, $par(v)$ is the *parent* of v, i.e., the unique neighbor of v on the path to $r(T)$; accordingly, the children of v are all vertices that have v as parent. The *subtree rooted at* v, denoted $T[v]$, is the subgraph of T with $r(T[v]) = v$ induced by the set of descendants of v. $F(v)$ then denotes the set of subtrees rooted at the children of v.

Tree edit distance. Let $\perp \notin \Sigma$ be a special *blank* symbol. For $\Sigma^\perp = \Sigma \cup \{\perp\}$ we define a *cost function* $\gamma : \Sigma^\perp \times \Sigma^\perp \to \mathbb{R}$ and require γ to be a metric. An *edit script* or *edit sequence* from a tree T into a tree T' is a sequence of edit operations turning T into T'. An edit operation can (i) *relabel* a single node v, (ii) *delete* v and connect all its children to the parent of v, or (iii) *insert* a single node w between v and a subset of v's children. The cost of such edit operations is defined by γ; relabeling v from a to b costs $\gamma(a, b)$ and adding or deleting v costs $\gamma(\ell(v), \perp)$. An edit script between T and T' of minimum cost is called *optimal* and its cost is called *tree edit distance*. It is a metric if γ is a metric.

Wasserstein distance. The Wasserstein distance is a distance function between probability distributions on some given metric space. Intuitively, it can be viewed as the minimum cost necessary to transform one pile of earth into another. It is also known as the *earth movers* distance or *optimal transportation* distance. More precisely, given two vectors $x \in \mathbb{R}^n$ and $x' \in \mathbb{R}^{n'}$ with $\|x\|_1 = \|x'\|_1$ and a cost matrix $C^{n \times n'}$ containing pairwise distances between entries of x and x', the *Wasserstein distance* is defined by

$$\mathcal{W}^C(x, x') = \min_{T \in \mathcal{T}(x, x')} \langle T, C \rangle$$

with $\mathcal{T}(x, x') \subseteq \mathbb{R}^{n \times n'}$ and $T\mathbf{1}_{n'} = x$, $\mathbf{1}_n^\top T = x'$ for all $T \in \mathcal{T}(x, x')$, where $\langle ., . \rangle$ is the Frobenius inner product. A $T \in \mathcal{T}(x, x')$ is called *transport matrix* and a minimizer of the above is called *optimal* transport matrix. If the cost matrix is defined by a metric, then the Wasserstein distance is a metric. For a set of vectors $x_1, ..., x_k \in \mathbb{R}^n$ and a cost matrix $C^{n \times n}$, we define the *barycenter* as

$$\arg\min_{c \in \mathbb{R}^n} \sum_{i \in [k]} \mathcal{W}^C(x_i, c).$$

3 The Weisfeiler-Lehman tree edit distance

In this section, we briefly recap the Weisfeiler-Lehman label propagation algorithm (Weisfeiler & Lehman, 1968) and define a distance on Weisfeiler-Lehman labels. We give an algorithm computing this distance and prove that it can be calculated in polynomial time.

3.1 The Weisfeiler-Lehman method

The *Weisfeiler-Lehman* (WL) method (Weisfeiler & Lehman, 1968) was originally designed to decide isomorphism between graphs with one-sided error. Its key idea is to iteratively refine a partitioning of the vertex set by compressing the labels of each node and its neighbors into a new label. This is done by concatenating a node's label and its ordered (multi-)set of neighbor labels and subsequently hashing it to a new label by a perfect hash function. Thus, with each iteration, labels incorporate increasingly large substructures. The injectivity of the hash function ensures that different sorted lists of labels cannot be mapped to the same (new) label.

More precisely, let $G = (V, E, \ell_0)$ be a graph with initial vertex label function $\ell_0 : V \to \Sigma_0$, where Σ_0 is the alphabet of the original vertex labels. In case of unlabeled graphs, we assume all vertices to have the same mutual label. Assuming that there is a total order on alphabet Σ_i for all $i \geq 0$, the Weisfeiler-Lehman algorithm recursively computes the new label of v in iteration $i + 1$ by

$$\ell_{i+1}(v) = f_{\#}(\ell_i(v), [\ell_i(u) : u \in \mathcal{N}(v)]) \in \Sigma_{i+1}$$

for all vertices v, where the list of labels in the second argument of $f_{\#}$ is sorted by the total order on Σ_i and $f_{\#} : \Sigma_i \times \Sigma_i^* \to \Sigma_{i+1}$ is a perfect (i.e., injective) hash function. Two graphs G, G' are not isomorphic if the corresponding multisets $\{\!\{\ell_i(v) : v \in V(G)\}\!\}$ and $\{\!\{\ell_i(v') : v' \in V(G')\}\!\}$ are different for some $i \in \mathbb{N}$; otherwise they may or may not be isomorphic. However, $G \equiv G'$ holds with high probability when the two multisets are equal (Babai & Kucera, 1979).

Shervashidze et al. (2011) employed the Weisfeiler-Lehman method to define a family of parameterized kernels measuring the similarity between graphs based on their relabeled versions. For a graph $G = (V, E, \ell_0)$ they consider the sequence of WL-graphs $G_0, G_1, ..., G_h$ with $G_i = (V, E, \ell_i)$, where h is the number of performed WL iterations. The Weisfeiler-Lehman kernel of depth h for two graphs G, G', given some base graph kernel k, is then defined as

$$k^h_{WL}(G, G') = \sum_{i=0,\dots h} k(G_i, G'_i).$$

In other words, the kernel k is applied to G, G' for all labeling functions ℓ_i $(0 \leq i \leq h)$ and the $h + 1$ values obtained are subsequently summed up. We note that each component $k(G_i, G'_i)$ in $k^h_{WL}(G, G')$ can be assigned a non-negative real weight α_i. This allows e.g. to emphasize larger substructures (i.e., labels in higher iterations contribute more to the overall similarity). While the base kernel k can be an arbitrary positive semi-definite kernel on graphs, for simplicity we focus on the *subtree kernel* (Shervashidze et al., 2011) which employs the base kernel

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

where δ is the Kronecker delta. Thus, k^h_{WL} simply counts the pairs of matching labels of all WL-iterations. With complexity $O(hm)$, where m is the number of edges, the WL subtree kernel is highly efficient and has proven to provide state-of-the-art results on a broad range of datasets (Kriege et al., 2020).

Another view of the Weisfeiler-Lehman label propagation is that for each iteration i, it implicitly constructs tree patterns of depth i which are being compressed into new labels.

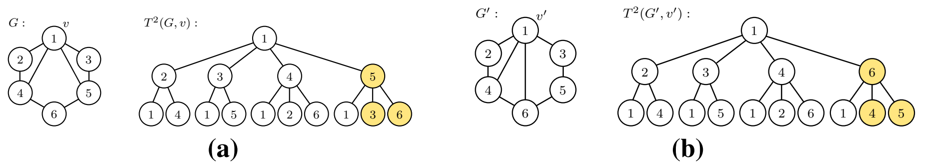


Fig. 2 Unfolding trees $T^2(G,v)$ and $T^2(G',v')$. As v and v' have structurally similar roles in G, resp. G', their unfolding trees differ only slightly (labeled yellow). The vertex corresponding to v, resp. v', appears again several times at depth 2 of $T^2(G,v)$, resp. $T^2(G',v')$

Each such tree, denoted $T^i(G,v)$, is called the depth-i *unfolding tree* (or simply, i-unfolding tree) of G at v. It corresponds to all possible walks of length i starting at node v (Dell et al., 2018). Figure 2 visualizes this concept and illustrates that there is a function from the vertices in the unfolding tree of G at v into the corresponding vertices of graph G. Thus, a node of G can appear several times in $T^i(G,v)$ for $i > 1$. It is easy to see that there is a bijection between labels in Σ_i and the set of (pairwise non-isomorphic) i-unfolding trees.

3.2 The structure and depth preserving tree edit distance

While the *strict* comparison of labels, or equivalently, that of unfolding trees is advantageous for the original intention of the Weisfeiler-Lehman method, it is a severe drawback of all Weisfeiler-Lehman graph kernels, including the Weisfeiler-Lehman subtree kernel. The reason is that comparing unfolding trees with each other by *equality* (i.e., tree isomorphism), or equivalently, taking merely into account whether the labels of vertices and those of their neighborhoods differ or not, is arguably too restrictive for kernel design, as in case of kernels, we are interested in defining similarities. Our typical observation is that the i-unfolding trees (i.e., labels at iteration i) of most vertices will be unique for very small values of i. In other words, the limitation of the Weisfeiler-Lehman graph kernels is that two structurally completely different unfolding trees are treated identically to two unfolding trees which differ by only very little.

To overcome this drawback, we propose a finer label comparison by defining a new *similarity measure* between unfolding trees that employs a *specialized* form of the well-known *tree edit distance*. On an abstract level, the tree edit distance measures the minimum amount of edit operations necessary to turn one tree into another. Calculating this distance is NP-hard in general (see, e.g., Bille, 2005). However, for our purpose it suffices to consider a restricted type of tree edit distance which preserves essential properties of unfolding trees. Below we show that, in contrast to the general case, this variant can be calculated efficiently.

The construction procedure of unfolding trees as demonstrated above shows that they reflect the neighborhoods of a specific vertex. Therefore, we require the edit scripts between unfolding trees to preserve the neighborhood relationships of vertex pairs as well as the depth of vertices. This leads to the following definition of constrained tree edit scripts:

Definition 1 A *structure and depth preserving mapping* (SDM) between two rooted trees T and T' is a triple (M,T,T') with $M \subseteq V(T) \times V(T')$ satisfying

1. $\forall (v_1, v_1'), (v_2, v_2') \in M : v_1 = v_2 \iff v_1' = v_2'$, (definite)
2. $(r(T), r(T')) \in M$, (root preserving)
3. $\forall (v, v') \in M : (par(v), par(v')) \in M$. (structure preserving)

The set of all structure and depth preserving mappings between T and T' is denoted by $\textsc{Sdm}(T, T')$.

SDMs represent sequences of edit operations subject to the above constraints that transform trees into trees. More precisely, for an SDM (M, T, T') let $T = T_0, T_1, \ldots, T_k$ be a sequence of trees such that T_{i+1} is obtained from T_i by applying one of the following atomic transformations:

relabel: If $(v, v') \in M$, then replace the label of v in T_i by that of v'.
delete: If v is a leaf in T_i and it does not occur in a pair of M, then remove v from T_i.
insert: If v' is a vertex in T' which does not occur in a pair of M and for which the corresponding parent u already exists in T_i, then add a child to u with the label of v'.

The proof of the following claim is straightforward.

Proposition 1 *Let (M, T, T') be an* SDM *and $T_0 = T, T_1, \ldots, T_k$ be a sequence of trees obtained by the above atomic transformations such that every $v \in T$ and $v' \in T'$ has been considered in exactly one transformation. Then $T_k = T'$.*

Note that SDMs uphold some essential properties of unfolding trees. In particular, they ensure that siblings are preserved (i.e., for any SDM (M, T, T'), v_1' and v_2' are siblings in T' whenever $(v_1, v_1'), (v_2, v_2') \in M$ and v_1, v_2 are siblings in T) and that vertices can only be mapped onto vertices of the same depth. Recall, that our goal is to measure similarities between neighborhoods of vertices. It is thus essential that roots are being preserved; this is guaranteed by the second constraint in Def. 1. Furthermore, Def. 1 implies that M maps a connected subtree of T onto a connected subtree of T'. That is, the first (resp. second) components of the pairs in M form a connected subtree of T (resp. T').

Figure 3 demonstrates the motivation of SDMs. The mapping displayed in Fig. 3a is a structure and depth preserving mapping from T into T' which visibly preserves the depth as well as the pairwise sibling relationships for all mapped vertices. In contrast, while the edit script in Fig. 3b is valid for more general definitions of edit operation sequences, the transformation constructs a tree which heavily distorts neighborhood relationships and arbitrarily inserts nodes such that the set of vertices in T' touched by a line preserve only very little of the topology of those in T. In particular, leafs that have distance 4 from each other in T are mapped onto vertices in T' which are now direct siblings. Furthermore, the mapping does not maintain root nodes, as a root is mapped to a non-root node.

Using these notions, we are ready to define the distance between unfolding trees.

Definition 2 Let T, T' be unfolding trees over a vertex label alphabet Σ and $\gamma : \Sigma^{\perp} \times \Sigma^{\perp} \to \mathbb{R}$ a cost function (i.e., metric), where $\perp$ is the blank symbol. Then the *cost* for an SDM (M, T, T') is

$$\gamma(M) = \sum_{(v,v') \in M} \gamma(\ell(v), \ell(v')) + \sum_{v \in N} \gamma(\ell(v), \perp) + \sum_{v' \in N'} \gamma(\perp, \ell(v'))$$

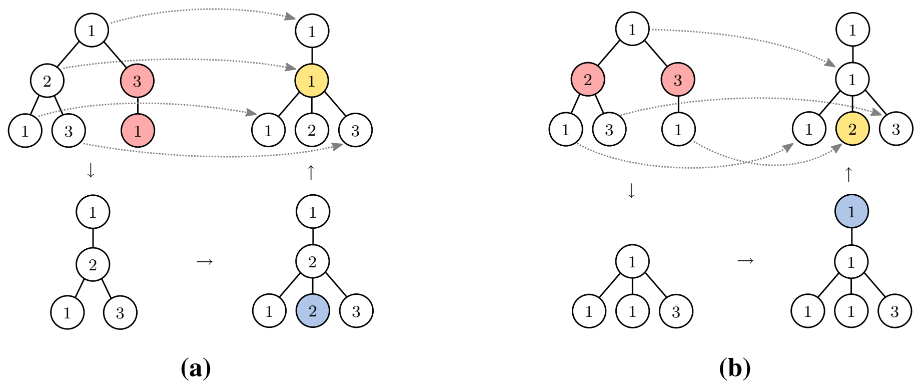


Fig. 3 Two mappings from one unfolding tree into another. **a** depicts a mapping which is structure and depth preserving whereas the one in **b** is not. Dashed lines correspond to pairs contained in the respective mappings M, red vertices are being deleted, blue vertices inserted and yellow vertices relabeled

where N (resp. N') are the vertices of T (resp. T') that do not occur in any pair of M. The *structure and depth preserving tree edit distance* from T into T', denoted $\textsc{SdTed}(T, T')$, is then defined by

$$\textsc{SdTed}(T, T') = \min\{\gamma(M) : (M, T, T') \in \textsc{SdM}(T, T')\}$$

Thus, the cost of an SdM (M, T, T') is defined by the sum of the individual costs of relabeling, insertion, and deletion operations over all vertices of T and T', where the cost of the insertion (resp. deletion) of a vertex v is given by $\gamma(\ell(v), \perp)$ (resp. $\gamma(\perp, \ell(v))$). The structure and depth preserving tree edit distance between trees T and T' is then simply the minimal cost over all possible mappings.

Algorithm 1 Compute SdTed

```
input: Trees $T, T'$, cost function $\gamma : \Sigma^\perp \times \Sigma^\perp \to \mathbb{R}$
output: Structure and depth preserving tree edit distance between $T$ and $T'$

$\textsc{SdTed}(T, T')$:
1: $F := F(r(T)),\ F' := F(r(T'))$
2: Pad $F$ and $F'$ with empty trees $T_\perp$ such that $|F| = |F'| = deg(r(T)) + deg(r(T'))$
3: for all $T_i \in F,\ T'_j \in F'$ do
```

$$\delta_{ij} = \begin{cases} \textsc{SdTed}(T_i, T'_j) & \text{if } T_i \in F(r(T)) \text{ and } T'_j \in F(r(T')) \\ \sum\limits_{v \in V(T_i)} \gamma(\ell(v), \perp) & \text{if } T_i \in F(r(T)) \text{ and } T'_j \notin F(r(T')) \\ \sum\limits_{v' \in V(T'_j)} \gamma(\ell(v'), \perp) & \text{if } T_i \notin F(r(T)) \text{ and } T'_j \in F(r(T')) \\ 0 & \text{o/w}. \end{cases}$$

```
4: Let $S \subseteq F \times F'$ be a minimum cost perfect bipartite matching w.r.t. distances $\delta$
5: return $\gamma(\ell(r(T)), \ell(r(T'))) + \sum_{(T_i, T'_j) \in S} \delta_{ij}$
```

3.3 The unfolding tree edit distance algorithm

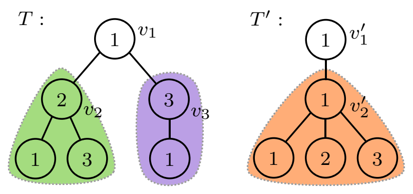
We now show that for any pair of unfolding trees T, T', $\textsc{SdTed}(T, T')$ can efficiently be calculated in a recursive manner (see Alg. 1). It follows from the properties of SDMs that subtrees of T are mapped onto subtrees of T'. Thus, finding an optimal SDM (i.e. an SDM of minimal cost) from T into T' is equivalent to finding the set of optimal SDMs turning the trees below the root of T (i.e. $F(r(T))$) into the trees below the root of T' (i.e., $F(r(T'))$). In order to find this set of optimal SDMs, we need the pairwise distances $\textsc{SdTed}(T_i, T'_j)$ between all $T_i \in F(r(T))$ and $T'_i \in F(r(T'))$ as well as the costs of deleting, resp. inserting, trees T_i, resp. T'_j. The computation of these costs is done in line 3 of Alg. 1. The first case recursively calculates the $\textsc{SdTed}(T_i, T'_j)$ for all pairs of trees in $F(r(T))$ and $F(r(T'))$. The second case considers the instance where the root of some tree T_i is not part of a mapping, which implies that all vertices in T_i are deleted. A similar argument follows for the insertion of trees T'_j (third case of line 3). The task of finding an optimal SDM can in fact be reduced to the *minimum cost perfect bipartite matching* problem as follows: Let the sets of trees below the roots of T and T' be $F = \{T_1, \dots, T_k\}$ and $F' = \{T'_1, \dots, T'_{k'}\}$, respectively. We first expand the set of trees F by k', resp. F' by k, auxiliary empty graphs $T_\perp$ (line 2) such that both sets have equal cardinality. The distance (δ in Alg. 1) between a tree and an empty graph is defined as the cost of deleting, resp. inserting that tree. Furthermore, two empty graphs clearly have distance 0. One can check that the optimal set of SDMs directly corresponds to a perfect bipartite matching of minimum cost between trees in the expanded sets F and F' (line 4) with distances as defined above. Finally, the SDTED between trees T and T' is the cumulative cost of the distance between their roots and the minimal cost perfect bipartite matching between the trees below them (line 5). The above considerations imply the following result:

Theorem 1 *Given unfolding trees* T, T' *with labels from* Σ *and a cost function* $\gamma : \Sigma^\perp \times \Sigma^\perp \to \mathbb{R}$ *over* $\Sigma^\perp$, *Alg. 1 returns* $\textsc{SdTed}(T, T')$.

As an example, consider the SDTED between graphs T and T' of Fig. 4a. We assume that each insertion, deletion, and relabeling operation has cost 1. Following Definition 1, the root of T is mapped onto the root of T'. As both vertices have the same label, the respective cost is zero (i.e. $\gamma(\ell(v_1), \ell(v'_1)) = 0$). Due to the structure preserving property of SDMs, calculating the edit costs for the remaining vertices beneath the roots comes down to matching (resp. inserting and deleting) the highlighted subtrees. It can easily be checked that matching $T[v_2]$ with $T[v'_2]$ (which has cost 2) and thus deleting $T[v_3]$ (which has cost 2) has minimal cost over all possible matchings. The individual edit operations corresponding to this case are depicted in Fig. 3a.

By the construction of unfolding trees, vertices closer to v in G begin to appear at smaller depths in $T^i(G, v)$. In fact, the number of occurrences in $T^i(G, v)$ of a node $u \in V(G)$ grows exponentially with i once it has appeared for the first time. This indirectly assigns higher weights to vertices closer to v in the calculation of the structure and depth preserving tree edit distance.

Notice that Algorithm 1 describes a naive implementation which in general requires an exponential number of recursion calls. However, it is easy to see that the number of i-unfolding trees in T and T' is bounded by their sizes $n = V(T)$ and $n' = V(T')$. Once $\textsc{SdTed}(T_i, T_j)$ between two i-unfolding trees T_i, T_j has been calculated, it can be stored in a lookup table. Thus, for each level i, we need to invoke Algorithm 1 a maximum of nn'



(a)

M_r :

	①	②	③	⊥
①	0	1	1	1
②	1	0	1	1
③	1	1	0	1
⊥	1	1	1	0

M_c :

	🟩	🟪	🟧	⊥
🟩	0	2	2	3
🟪	2	0	3	2
🟧	2	3	0	4
⊥	3	2	4	0

(b)

Fig. 4 **b** Computing SDTED(T, T') for trees T and T' in **a** requires the costs for mapping the root nodes as well as for matching the highlighted subtrees onto another. The corresponding costs M_r, resp. M_c, are provided in **b**. Following the order on node labels, resp. child trees, as in M_r, resp. M_c, the unfolding tree vectors of T and T' have the form $\mathbb{V}_r(T) = [1, 0, 0, 0]$, $\mathbb{V}_c(T) = [1, 1, 0, 1]$ and $\mathbb{V}_r(T') = [1, 0, 0, 0]$, $\mathbb{V}_c(T') = [0, 0, 1, 2]$. One can check that $\mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) = 0$ and $\mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')) = 4$, resulting in SDTED$(T, T') = \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')) = 4$

times. With a lookup table for distances between unfolding tree pairs we thus require at most $nn'h$ invocations of a minimum cost perfect bipartite matching algorithm, each of complexity $\tilde{O}(d^3)$, where h is the depth and d the maximum degree of T, T'.

4 The generalized Weisfeiler-Lehman subtree kernel

Using the definitions and results of Sect. 3, we now introduce our novel *generalized Weisfeiler-Lehman subtree kernel* and show that it is in fact a generalization of the original Weisfeiler-Lehman subtree kernel (Shervashidze et al. 2011). Its key idea is to *relax* the rigid comparison of unfolding trees by equality (i.e., isomorphism) used in the original Weisfeiler-Lehman graph kernel by considering the structure and depth preserving distances between unfolding trees. Using SDTED, we identify groups of similar trees by means of *hard* clustering. This ensures that similar unfolding trees will belong to the same clusters, while dissimilar to different ones. Two unfolding trees are then regarded equivalent by the relaxed Weisfeiler-Lehman subtree kernel iff they belong to the same cluster.

More precisely, for a set $\mathcal{G}$ of graphs, let Θ_i be a set of hard clustering functions (i.e., partitionings) of the set of depth-i unfolding trees $\mathcal{T}^{(i)}$ appearing in the graphs in $\mathcal{G}$. We regard each element of Θ_i as a function $\rho : \mathcal{T}^{(i)} \to [k]$, where k is the number of clusters defined by ρ. Then, for any graphs $G, G' \in \mathcal{G}$ and depth parameter h, the *relaxed Weisfeiler-Lehman subtree kernel* is defined by

$$k^h_{\text{R-WL}}(G, G') = \sum_{i=0,..,h} \sum_{\rho \in \Theta_i} \sum_{v \in V} \sum_{v' \in V'} \delta(\rho(T^i(G, v)), \rho(T^i(G', v'))),$$

where δ is the Kronecker delta. The proof to the following is straightforward and can be found in Appendix 1.

Theorem 2 *The generalized Weisfeiler-Lehman subtree kernel* $k^h_{\text{R-WL}}(G, G')$ *is positive semi-definite.*

Notice that $k^h_{\text{R-WL}}$ is equivalent to the original Weisfeiler-Lehman subtree kernel k^h_{WL} for the case that $\Theta_i = \{\rho_i\}$ with ρ_i defined as follows: For all $T, T' \in \mathcal{T}^{(i)}$, $\rho_i(T) = \rho_i(T')$ iff T and T' are isomorphic (or equivalently, $\textsc{SdTed}(T, T') = 0$). Thus, our definition generalizes the ordinary Weisfeiler-Lehman subtree kernel in two ways: First, while the ordinary Weisfeiler-Lehman subtree kernel regards two unfolding trees T, T' to be equivalent iff $\textsc{SdTed}(T, T') = 0$, our definition allows $\textsc{SdTed}(T, T') \geq 0$ as well. Second, our definition enables more than one partitioning (or hard clustering) function, in contrast to k^h_{WL}.

We employ the concept of *Wasserstein k-means clustering* (Irpino et al. 2014) as a method to partition the set of unfolding trees. This choice is motivated by several arguments. As mentioned above, the purpose of clustering is to group similar unfolding trees w.r.t. SdTed. We therefore require the clusters to be *convex* such that unfolding trees of a cluster ideally have pairwise small distance. Another requirement is to be able to control the number of clusters which also influences the complexity of the approximation variant of the generalized Weisfeiler-Lehman kernel discussed in Sect. 4.1. We show that the SdTed can in fact be calculated using the discrete Wasserstein distance. Thus, we use the *same* distance in the cost matrix as in the clustering process. Finally, the Wasserstein distance has recently been the focus of comprehensive research leading to *fast* approximation methods for distance and center computations (Cuturi & Doucet, 2014; Cuturi, 2013).

Below we address the most important ingredients of Wasserstein k-means needed for our purpose. In particular, we first discuss how unfolding trees can be represented by real-valued vectors. Subsequently, we state that the Wasserstein distance between such vectors corresponds to the SdTed of the respective unfolding trees. This representation, furthermore, allows for the calculation of center points using *Wasserstein barycenters*. To keep the presentation concise we solely outline these concepts in this article. We provide a more detailed description and a complexity analysis in Appendices 2 and 3.

Unfolding Tree Vectors In order to effectively apply Wasserstein k-means, we represent i-unfolding trees by (sparse) real-valued vectors. Recall that the structure and depth preserving tree edit distance is calculated as the *sum* of

(A) the distance between the root nodes and
(B) the minimum cost of the perfect bipartite matching between child trees below these roots

as described in Alg. 1. We accordingly represent an i-unfolding tree T as a pair of vectors $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$, where

(A) $\mathbb{V}_r(T)$ encodes the root node's label $\ell(r(T))$ and
(B) $\mathbb{V}_c(T)$ encodes the set of $(i-1)$-unfolding child trees $F(r(T))$ below the root $r(T)$.

We define $\mathbb{V}_r(T)$ as a *one-hot* vector with entry 1 at index corresponding to its root node label $\ell(r(T))$ and 0 everywhere else. $\mathbb{V}_c(T)$ corresponds to a histogram counting isomorphic child trees below the root. Analogously to Alg. 1, the vector $\mathbb{V}_c(T)$ furthermore contains an entry for empty child trees ($\perp$) to account for insertion and deletion. We give an example of these vector representations in the description of Fig. 4.

4.1 Wasserstein Distance on Unfolding Tree Vectors

Using the vector representations of unfolding trees, we are able to reformulate the computation of the structure and depth preserving distance in terms of the Wasserstein distance. We show that the structure and depth preserving distance between trees T, T' can in fact be calculated as the sum of (A) the Wasserstein distance between the root node vectors $\mathbb{V}_r(T), \mathbb{V}_r(T')$ and (B) the Wasserstein distance between the child tree histogram vectors $\mathbb{V}_c(T), \mathbb{V}_c(T')$. More precisely, assume that the pairwise distances between child trees (as well as the empty tree) have already been calculated and are stored in a matrix M_c. Furthermore, let M_r be the distance matrix between the original node labels. It can easily be shown that for two depth-i unfolding trees T and T', the distance between their root nodes is equal to $\mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T'))$. Furthermore, the calculation of the minimum cost perfect bipartite matching between the sets of child trees below these roots (cf. Alg. 1) can be reduced to computing the Wasserstein distance between $\mathbb{V}_c(T)$ and $\mathbb{V}_c(T')$, i.e., $\mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. This can be shown using the following lemma which follows from the integral flow theorem.

Lemma 1 *For $x, x' \in \mathbb{N}^d$ there exists a transportation matrix $T \in \mathcal{T}(x, x')$ with $T \in \mathbb{N}^{d \times d}$ such that $\langle T, C \rangle = \mathcal{W}^C(x, x')$ for any cost matrix $C \in \mathbb{R}^{d \times d}$.*

This lemma implies that the minimum cost perfect bipartite matching between the sets of child trees in T and T' is equivalent to $\mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. Putting all together we have:

$$\textsc{SdTed}(T, T') = \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')).$$

An example demonstrating this equivalence is given in Fig. 4.

4.2 Unfolding Tree Barycenters

The above reformulation allows to calculate *barycenters* of sets of unfolding trees to perform Wasserstein k-means (Irpino et al., 2014). A barycenter of a set S of unfolding trees is a point which minimizes the sum of distances to unfolding tree vectors corresponding to S. Similarly to unfolding tree vectors, this barycenter is a pair of real-valued vectors (μ_r, μ_c), where μ_r is the center of the $\mathbb{V}_r$s and μ_c of the $\mathbb{V}_c$s. Formally, the barycenter of S is a pair (μ_r, μ_c) defined by:

$$\underset{\mu_r, \mu_c}{\arg\min} \sum_{T \in S} \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mu_c) \tag{1}$$

Note that the set of considered vectors $\mathbb{V}_r$, resp. $\mathbb{V}_c$, need to have equal mass, i.e. $l1$-norm, in order to compute barycenters. This is accomplished by adjusting the entry corresponding to the empty tree ($\perp$) by an adequate number. While a barycenter, in general, does not correspond to an existing unfolding tree, the Wasserstein distance between an unfolding tree vector $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$ and a center vector $\mu = (\mu_r, \mu_c)$ can be computed nonetheless as follows:

$$\mathcal{W}^{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mu_c) \tag{2}$$

4.3 The Wasserstein k-Means Algorithm for Unfolding Trees

Using the above concepts, the Wasserstein k-means clustering algorithm can be stated in form of Lloyd's algorithm (Lloyd, 1982).

i. In the initialization step, a subset of k unfolding trees is selected as initial centers.
ii. Each unfolding tree is then assigned to its nearest center point (using Eq. (2)).
iii. Finally, the centers of the newly defined clusters are recalculated (using Eq. (1)).

Steps (ii) and (iii) are repeated until clusters do not change anymore, i.e., the algorithm converges, or a predefined number of iterations has been reached.

4.4 A faster kernel variant

For many graph datasets the number of unfolding trees,[1] grows rapidly with increasing iterations (although it is bounded by the total number of vertices in the database). Dealing with large amounts of unfolding trees in the Wasserstein k-means step becomes, however, computationally expensive. We thus propose a practical variant of our kernel that addresses this issue by approximating distances between unfolding trees using their cluster centers. Figure 5 shows the high level idea of this approach.

Consider the calculation of pairwise distances between unfolding trees as in Sect. 3.3. That is, the distances of 0-unfolding trees are defined by the metric γ and the SDTEDs for all pairs of $(i+1)$-unfolding trees are computed using distances of i-unfolding trees. To reduce the number of distinct i-unfolding trees $\mathcal{T}^{(i)}$ (or equivalently labels Σ_i), we perform a clustering $C_1, \ldots, C_k$ of $\mathcal{T}^{(i)}$ with centers $\mu_1, \ldots, \mu_k$ as in Sect. 4. We then effectively replace each i-unfolding tree with its cluster center μ_j and compute the distance between i-unfolding trees $T \in C_j$, $T' \in C_{j'}$ by the distance between their cluster centers. More precisely, the structure and depth preserving tree edit distance between T and T' is approximated by

$$\mathcal{W}^{M_r}(\mu_r^j, \mu_r^{j'}) + \mathcal{W}^{M_c}(\mu_c^j, \mu_c^{j'})$$

where T and T' have been assigned to clusters with centers $\mu_j = (\mu_r^j, \mu_c^j)$, and $\mu_{j'} = (\mu_r^{j'}, \mu_c^{j'})$, respectively. Subsequently, these distances are used in iteration $i+1$, greatly reducing the

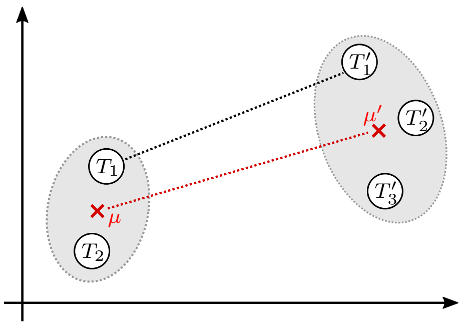


Fig. 5 Visualization of a clustering over a set of unfolding trees and the respective cluster centers. Instead of computing the SDTED for all pairs of unfolding trees, the kernel variant $k^h_{\text{R-WL}*}$ approximates their distances by the distance between their center points. E.g. $\text{SDTED}(T_1, T_1') = \mathcal{W}^{M_r}(\mathbb{V}_r(T_1), \mathbb{V}_r(T_1')) + \mathcal{W}^{M_c}(\mathbb{V}_c(T_1), \mathbb{V}_c(T_1'))$ is approximated by $\mathcal{W}^{M_r}(\mu_r, \mu_r') + \mathcal{W}^{M_c}(\mu_c, \mu_c')$

[1] More precisely, the number of pairwise non-isomorphic unfolding trees, i.e., Weisfeiler-Lehman labels.

number of distance calculations. That is, in contrast to the computation of $k^h_{\text{R-WL}}(G, G')$, our kernel variant $k^h_{\text{R-WL}*}(G, G')$ considers only k labels instead of $|\mathcal{T}^{(i)}|$ labels in iteration i.

The concept of clustering unfolding trees (or equivalently WL labels) after *each* WL iteration and then continuing the process with representatives of clusters, can be considered as slowing down the WL relabeling process. When compared to the ordinary Weisfeiler-lehman relabeling scheme, this generally means that vertices are split into different label classes at a later iteration.

5 Related work

While conventional graph kernels define similarity in terms of mutual substructures such as walks (Gärtner et al., 2003), paths (Borgwardt & Kriegel, 2005), small subgraphs (Shervashidze et al., 2009), or subtrees (Shervashidze et al., 2011), recent work has moved away from solely counting equivalent substructures.

For example, Kriege et al. (2016) introduce a kernel which computes an optimal assignment between vertices. Similarly, Togninalli et al. (2019) relax this idea and employ the concept of optimal transportation as a form of "soft-matching" on vertices. Both methods perform a vertex matching based on node similarities derived from the Weisfeiler-Lehman hierarchy tree. More precisely, the similarity between two vertices v, v' is measured in terms of the maximum depth $i \in \mathbb{N}$ such that v and v' have the same i-unfolding trees. Recall that if the i-unfolding trees of v and v' differ, so must their j-unfolding trees for all $j > i$. Consequently, the number of possible similarity values between v and v' is bounded by the considered unfolding-tree depth. In extreme scenarios where neighborhoods are highly diverse such that no two vertices have identical 1-unfolding trees, all vertices are regarded equally (dis-)similar. A visualization for the limitation of the methods in Kriege et al. (2016) and Togninalli et al. (2019) is given in Fig. 6. While $T^1(G, v_1)$ can be considered more similar

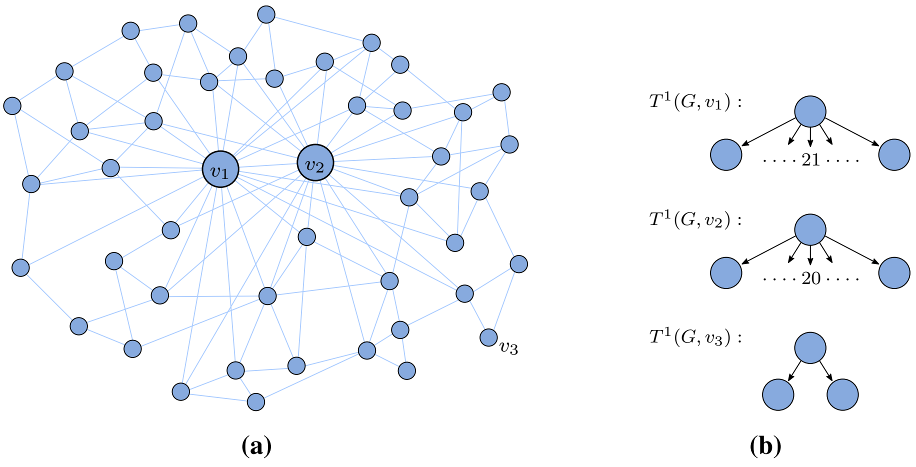


Fig. 6 The neighborhoods of vertices in **a** are highly diverse. While the neighborhoods of v_1 and v_2 (as shown by their unfolding trees in **b**) are arguably more similar than those of e.g. v_1 and v_3, methods relying on the rigid comparison of Weisfeiler-Lehman labels are unable to quantify the apparent similarity differences

to $T^1(G, v_2)$ than to $T^1(G, v_3)$, the methods are unable to quantify the apparent difference among the pairwise similarities. In contrast, the approach introduced in our article provides a similarity with a much finer granularity which is capable of making such a distinction.

The concept of comparing vertices by a finer similarity measure than the equivalence of their neighborhoods was also considered by Da San Martino et al. (2012). In that work, Da San Martino et al. represent neighborhoods by rooted directed acyclic trees (DAG) and define a kernel over these DAGs which reflects their similarity. The difference to our approach is twofold. Firstly, compared to Weisfeiler-Lehman labels, the DAGs describe structurally different representations of neighborhoods. Secondly, while Da San Martino et al. (2012) compute similarities by applying tree kernels (e.g. Smola & Vishwanathan, 2003) on sets of trees extracted from the DAGs, we employ the concept of a specific tree edit distance as similarity measure.

6 Empirical evaluation

We now evaluate the predictive performance of our approach on a set of established as well as novel real-world and synthetic datasets. Our results show that our approach increasingly outperforms all considered competitor kernels with growing density of dataset graphs in a majority of cases. Furthermore, experiments on synthetic graphs indicate that our approach is more robust to (structural) noise than competing kernels. We provide our implementation, as well as novel datasets on the accompanying web page.[2]

We note that in the following, we limit the evaluation to the approximation kernel R-WL* as discussed in Sect. 4.1. This choice was made due to the fact that while the R-WL kernel is well applicable to sparse graphs such as molecules, an explicit consideration of all unfolding trees may become computationally too expensive on more complex graphs.

6.1 Experimental setup

We compare our approach to a selection of graph kernels and provide a baseline method to put the performances into perspective. We consider the Weisfeiler Lehman subtree (WL) kernel (Shervashidze et al., 2011), the recently published Wasserstein Weisfeiler-Lehman graph (WWL) kernel (Togninalli et al., 2019), and the Persistent Weisfeiler-Lehman (PWL) graph method (Rieck et al., 2019) as examples of approaches that are based on comparing Weisfeiler-Lehman labels (i.e., unfolding trees) by equality and select depth parameter from $h \in [5]$. We furthermore consider the graphlet sampling (GS) kernel (Shervashidze et al., 2009) (with parameters $\epsilon = 0.1$, $\delta = 0.1$ and $k \in \{3, 4, 5\}$) and the shortest-path (SP) kernel (Borgwardt & Kriegel, 2005) as examples of other kernel classes. The ODD-STh kernel (Da San Martino et al., 2012) (with parameter $h \in [4]$) compares vertex neighborhoods as directed acyclic graphs and is in that sense most similar to our approach. We use the implementation of Siglidis et al. (2018) or that of the respective authors. As a baseline method (VE-Hist), we employ a simple histogram kernel over the set of edge and node labels. In case of our relaxed Weisfeiler-Lehman kernel R-WL* (as described in Sect. 4.1), we choose the number of clusters $k = \sqrt{|\Sigma_i|}$, use depth parameter h up to 4 and unit costs for all relabeling, deletion and insertion operations. We perform

[2] https://github.com/mlai-bonn/GenWL.

Table 1 Structural information of graph benchmark datasets

Dataset	$\|D\|$	$\|C\|$	$\varnothing\|V\|$	$\varnothing\|E\|$	$\frac{\varnothing\|E\|}{\varnothing\|V\|}$	Δ	Number of node labels			
							Σ_0	Σ_1	Σ_2	Σ_3
IMDB	1000	2	19.8	96.5	4.9	135	1	65	2931	3595
REDDIT	2000	2	429.6	497.8	1.2	3062	1	565	$72k$	$244k$
EGONETS-1	200	4	139.0	594.6	4.3	140	1	112	$21k$	$25k$
EGONETS-2	200	4	178.6	1445.0	8.1	180	1	140	$33k$	$35k$
EGONETS-3	200	4	220.0	2613.5	11.9	203	1	169	$42k$	$43k$
EGONETS-4	200	4	259.8	4135.9	15.9	237	1	209	$51k$	$51k$

$|D|$, $|C|$, and Δ, denote the number of graphs, number of classes, and maximum degree in a dataset. Σ_0 is the number of distinct vertex labels. $\Sigma_1, \Sigma_2, \Sigma_3$ are the amounts of distinct Weisfeiler-Lehman labels for depth $h = 1, 2, 3$, respectively

a total of 3 clusterings. This particular choice for k selects the number of clusters relative to the amount of Weisfeiler-Lehman labels in each iteration and significantly reduces the computational complexity of the clustering. We perform a total of 3 clusterings (i.e. $|\Theta_i| = 3$) to make up for the randomness caused by the k-means initialization step.

We measure the prediction performance in terms of accuracy obtained by support vector machines (SVM) using a 10-fold cross-validation. If not explicitly chosen otherwise by the authors of the individual implementations, the parameter C is selected from the value set $2^i : i \in \{-12, -8, -5, -3, -1, 1, 3, 5, 8, 12\}$. In each fold, a grid search is used to identify the optimal kernel parameters. We report the mean and standard deviation over 5 such cross-validation repetitions.

6.1.1 Datasets

Real-world Datasets We conduct experiments on several social network datasets. The benchmark datasets IMDB-BINARY and REDDIT-BINARY (IMDB, resp. REDDIT for short, provided by Morris et al. (2020)) contain subgraphs of online networks. IMDB consists of collaboration networks between actors, each annotated against movie genres. Graphs in REDDIT represent user interactions in discussion forums with graphs being annotated by the type of forum.

Furthermore, we provide a set of novel real-world benchmark datasets extracted from the social networks Buzznet, Digg, Flickr (provided by Zafarani & Liu, 2009) and LiveJournal (provided by Leskovec & Krevl, 2014). Each of the EGONETS datasets consists of 50 random ego network graphs from each of the 4 social networks where graphs are annotated against the social network they were extracted from. Here, ego networks are subgraphs induced by a vertex's neighbors. Graphs within each dataset were randomly chosen from the set of all ego networks but underlie size- and density-specific constraints to ensure that a simple count of nodes and edges is not sufficient for prediction tasks. The EGONETS-x datasets contain increasingly larger and more dense ego networks with growing index x. The learning task is to assign each ego network to the network they were extracted from. Details about the structural properties on these datasets can be found in Table 1.

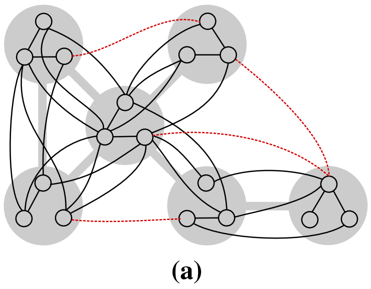
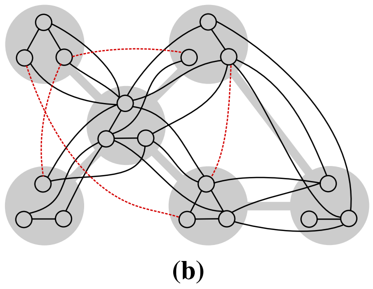


Fig. 7 Graphs generated by slightly different underlying structures (depicted in grey). Each block in the underlying structure contains 3 vertices. Two vertices are connected by an edge with some probability p if they belong to the same block or their blocks are connected in the underlying structure (depicted by solid black lines). Furthermore, both graphs additionally contain $m_x = 4$ noise edges (depicted by dashed red lines)



Synthetic Datasets To systematically evaluate the predictive performance of our kernel with varying structural complexity, we consider graphs generated by the stochastic block model (Wang & Wong, 1987). The specific generation of graphs is described by the following process: Let T be some random tree of a predefined size. Create two (non-isomorphic) graphs G_1, G_2 by adding a new edge to T.[3] We then generate a set of graphs for each $G \in G_1, G_2$ by repeating the following process: Let $\widehat{G}$ be the empty graph.

(i) For each $v \in G$, add a set of c vertices $v_1, ..., v_c$ to $\widehat{G}$.
(ii) For each $v \in G$, connect pairs $\{v_i, v_j\} \in V(\widehat{G})$ by an edge with probability p.
(iii) For all pairs $\{v_i, u_j\} \in V(\widehat{G})$ with $\{v, u\} \in E(G)$, connect them by an edge with probability p, as well.
iv) Connect a number of m_x prior unconnected vertex pairs $\{v_i, u_j\}$ in $\widehat{G}$ as noise edges.

The resulting classification task is to assign the generated graph $\widehat{G}$ to the underlying graph structure $G \in G_1, G_2$. Figure 7 depicts an example of two such generated graphs. All datasets considered in the following evaluations were created starting with a random tree of size 16 which was extended by a single random edge resulting in graphs G_1, G_2. For each classification task we generated 200 random graphs for each $G \in G_1, G_2$. The number of vertices c contained in a block was set to 8 in all experiments. The remaining parameters were selected as stated in Fig. 9. For each set of parameter choices, we generated 5 datasets and provide the mean accuracy.

6.2 Real-world benchmarks

Figure 8 lists the classification accuracies for datasets containing graphs extracted from online networks. While there are no large discrepancies between our method and the best performing comparison kernels on datasets IMDB, REDDIT and EGONETS-1 (which all have an average node-to-edge ratio up to roughly 1 : 4), the R-WL* kernel considerably

[3] To ensure that the classification task is non-trivial, we require that G_1 and G_2 have the same multiset of vertex degrees.

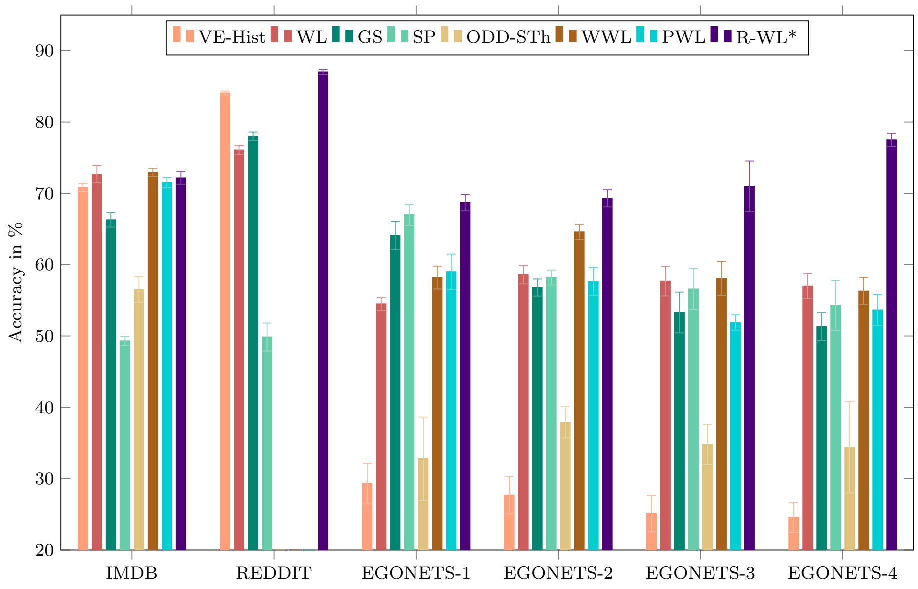


Fig. 8 Classification accuracies and std. deviations for large network benchmark datasets in %. ODD-STh, PWL and WWL did not finish within 24 hours on REDDIT

Table 2 Runtimes in seconds

Dataset	WL	GS	SP	ODD-STh	WWL	PWL	R-WL*
IMDB-B.	26	2438	30	252	4081	174	44
REDDIT-B.	397	4803	46778	–	–	–	1462
EGONETS-1	21	251	191	888	258	115	393
EGONETS-2	26	267	393	2154	349	186	1275
EGONETS-3	34	287	747	4275	492	275	2739
EGONETS-4	44	301	1152	7638	650	400	4173

We measured the total time it took to perform a 10-fold cross-validation (as described in Sect. 6.1). "–" indicates that the computation took longer than 24 hours or halted with an error

outperforms all others on the three remaining EGONETS datasets which contain significantly higher density graphs. The performance gap between the R-WL* kernel and the best performing competitor becomes increasingly larger with a growing density in the dataset graphs, leading to an above 20% accuracy difference. To formally evaluate statistical significance, we perform two-sample t-tests (with a significance threshold of 0.05) corrected by the Bonferroni method to make up for the number of tests within each dataset. The results show that our R-WL* variant significantly outperforms its competitors on the datasets REDDIT, EGONETS-2, EGONETS-3 and EGONETS-4. Table 2 shows the running times to obtain the accuracy values.

It is noteworthy that in case of the EGONETS datasets, already for depth $h = 2$ nearly all unfolding trees (i.e. depth-2 unfolding trees) appear only once in the respective dataset. Thus, the original WL kernel, WWL, and PWL kernels are not able to profit from any structural information exceeding node degrees, as graphs share almost no i-unfolding trees for $i \geq 2$. Interestingly, these kernels still achieve similar results to SP and GS on EGONETS-2/3/4, while VE-Hist is no better than random chance. In contrast, our approach clearly improves upon this limitation.

We furthermore evaluate our approach on traditional molecular datasets. The results suggest that our relaxation of the Weisfeiler-Lehman kernel matches but does not further improve the predictive performance over other classifiers on such datasets which contain mainly sparse and noise-free graphs. These evaluations can be found in Appendix 4.

Given these two high level experimental results, we conjecture that identifying *similar* unfolding trees instead of *identical* unfolding trees becomes the more advantageous, the more complex and diverse the graph database becomes. In other words, relaxing a strict comparison by equality of Weisfeiler-Lehman labels by clustering together similar WL labels seems to be beneficial when the growth rate of the label sets is high.

6.3 Noise and structural deviation

To validate our claim above, we investigate the effect of noise and structural deviation on the predictive performance of each kernel. To this end, we vary the values of the parameters m_x and p of the synthetic datasets (see Sect. 6.1.1). The parameter p governs the probabilities of edges within and between vertex blocks while m_x indicates the number of randomly added noise edges. Hence, they directly influence the noise and structural deviation of graphs within a class. Due to the construction of the datasets, VE-Hist cannot beat the accuracy of a random classifier (i.e., 50%) for any choice of parameters and is thus excluded from this analysis.

Figure 9a investigates the methods' robustness to noise. Higher values of m_x increase the deviation of graphs within the same class. For $m_x = 0$ (with $p = 1.0$ fixed) all graphs in the database have 128 nodes and 2048 edges, and graphs of the same underlying structure are pairwise isomorphic. Thus, all kernels except GS achieve 100%. While our method achieves 100% accuracy for all choices of m_x, the remaining kernels gradually and significantly decrease in predictive performance with increasing values for m_x. It is noteworthy

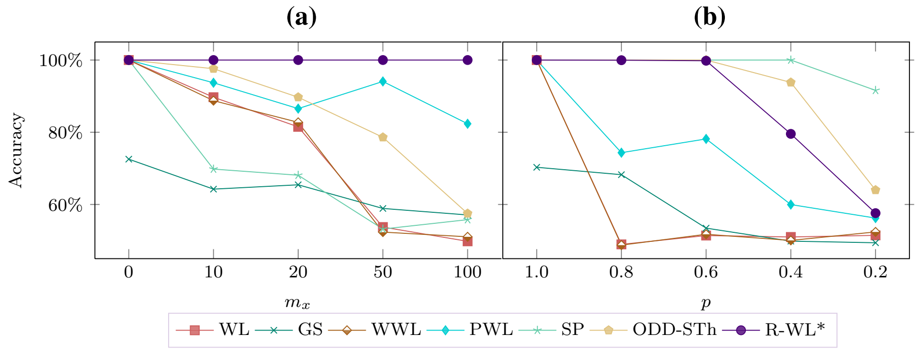


Fig. 9 Classification accuracies for synthetic dataset evaluations. **a** analyzes the influence of different amounts of noise edges m_x (with $p = 1.0$ fixed) and **b** shows results obtained for different values of edge probabilities parameter p (with $m_x = 0$ fixed). Block size c has been set to 8 for all cases. A statistical significance evaluation using paired t-tests (with significance threshold 0.05) corrected by the Bonferroni method shows that our method significantly outperforms all others on the experiments performed in **a** while it is beaten only by two competitors in **b**

that for the case of 100 noise edges, no competitor kernel but PWL performed significantly better than random.

Figure 9b analyzes the methods' ability to identify the underlying structure using different values of the edge probability parameter p (where parameter $m_x = 0$ is fixed). The lower the value p, the more the dataset graphs within a class deviate from each other, and the less of the underlying structure is being reflected. While in the trivial case $p = 1.0$ (where graphs belonging to the same class are pairwise isomorphic) all methods except GS achieve 100% accuracy, we observe a rapid performance decline for all methods but R-WL*, SP and ODD-STh for values $p < 1.0$. In particular, WL and WWL do not perform better than random other than for the trivial case. PWL, which is based on Weisfeiler-Lehman labels as well, only performs slightly better. While ODD-STh underperformed in all previous experiments, it seems that its approach to neighborhood similarity is well suited for this kind of structural deviation.

In summary, it is apparent that kernels based on the rigid equality of Weisfeiler-Lehman labels are less suited when structural noise distorts the graphs or when the graphs in a class structurally deviate significantly. Our method mitigates this drawback: Its ability to identify *similar* vertex neighborhoods leads to major increases in predictive performance on datasets containing *noisy* and *structurally diverse* graphs.

7 Concluding remarks

We experimentally demonstrated a drawback of the original Weisfeiler-Lehman graph kernels (Shervashidze et al., 2011) which is caused by their rigid comparison of Weisfeiler-Lehman labels w.r.t. equality. To overcome this limitation, we introduced a generalization of Weisfeiler-Lehman graph kernels which allows for a finer similarity measure between Weisfeiler-Lehman labels. The experimental results reported in this paper clearly show that the proposed generalization outperforms other state-of-the-art methods on graphs with structural complexity and edge density beyond the typically considered molecular graphs of small pharmacological compounds. We stress that although for simplicity we presented our approach only for the Weisfeiler-Lehman subtree kernel, our generalization is naturally applicable to all Weisfeiler-Lehman graph kernels.

In recent years, graph neural networks (GNNs) relying on the principle of message passing have become increasingly prevalent. Much like the Weisfeiler-Lehman label propagation scheme (Weisfeiler & Lehman, 1968), GNNs compute node feature representations by aggregating neighborhood information which are subsequently used for learning tasks, such as node or graph classification. GNNs generally aim at learning node feature representations which reflect pairwise node *similarities* in a high-dimensional metric space. While our approach does not explicitly learn node representations, it shares with GNNs the ingredient of fine grained *(dis-)similarity* measures comparing nodes and their neighborhoods.

The Weisfeiler-Lehman method was originally designed to decide graph isomorphism by comparing the WL-label feature sets of graphs. In this context, clustering similar WL-labels can be perceived as effectively lowering the dimensionality of the feature domain. That is, in our approach we deliberately decrease the representational power of the respective embeddings and thus relax the isomorphism test to obtain a more general similarity measure. In fact, the notion of isomorphism relaxation can also be found in graph neural networks for which recent studies provide some theoretical results concerning their expressiveness which relates to the learnable graph similarity measures. In particular, Xu et al.

(2019) and Morris et al. (2019) investigate the relationship of GNNs to the Weisfeiler-Lehman method, showing that their GNNs can be at most as discriminative as the Weisfeiler-Lehman method in terms of distinguishing non-isomorphic graphs. Xu et al. (2019) introduce the so-called Graph Isomorphism Networks (GIN) for which they prove that for certain properties such GINs map any two graphs to the same embedding if and only if the Weisfeiler-Lehman test considers the graphs to be isomorphic. Errica et al. (2020) further increases the expressivity of GINs by incorporating edge attributes as well as node or edge representations across network layers leading to strictly more expressive GNNs. Chen et al. (2019) establish a formal framework which classifies the representational power of GNNs, where its most powerful members correspond to collections of permutation-invariant functions which are able to distinguish non-isomorphic graphs. Furthermore, the concept of relaxing isomorphism to more general similarities can also be be found in Al-Rfou et al. (2019). In that paper, the authors introduce a neural network encoder architecture which is used to compute a divergence score between graphs. Conceptually, the more similar two graphs are, the smaller their divergence becomes. Another possible connection of our kernel to GNNs is the vision of *hybrid-approaches* which point towards promising methods that combine aspects of both, kernels and GNNs.

Our results raise several other interesting questions for further research. One open question is whether the tree edit distance can directly be used as a ground distance in vertex matching kernels - similar to approaches such as in Kriege et al. (2016) or Togninalli et al. (2019) - and, thus, eliminating the need for a hard partitioning of unfolding trees. Unfortunately, straightforward approaches such as replacing the ground distance in the Wasserstein Weisfeiler-Lehman kernel (Togninalli et al., 2019) by the tree edit distance does not yield a positive semi-definite kernel in general. However, there has been comprehensive research addressing the problem of dealing with indefinite kernels. For instance, indefinite kernel matrices may be converted to positive definite ones using spectrum transformation approaches, which, e.g., aim at flipping all negative eigenvalues to zero (Wu et al., 2005) or alter the matrix's diagonal by adding a positive term (Roth et al., 2003). Furthermore, we note that our node distance function SDTED is not restricted to be applied in the context of graph kernels. In fact, using SDTED as Wasserstein ground distance in order to compute distances between graphs, directly allows the application of other (dis-)similarity-based classifiers such as k-nearest-neighbor approaches.

Another particularly important research direction is to study other meaningful similarities between labels that allow for a *faster* calculation of minimum cost perfect bipartite matchings (or Wasserstein distances). As the cost function γ on the original vertex labels can be defined by an arbitrary metric, the application of our approach to attributed graphs is another natural research question.

Appendix 1 Positive semi-definiteness

Claim 1 *The relaxed Weisfeiler-Lehman kernel* $k^h_{\text{R-WL}}(G, G')$ defined as

$$\sum_{i=0,..,h} \sum_{\rho \in \Theta_i} \sum_{v \in V} \sum_{v' \in V'} \delta(\rho(T^i(G, v)), \rho(T^i(G', v')))$$

is positive semi-definite.

Proof We first show that the term

$$\sum_{v \in V} \sum_{v' \in V'} \delta(\rho(T^i(G,v)), \rho(T^i(G',v')))$$

can be expressed as an inner product. Let $\rho : \mathcal{T}^{(i)} \to [k]$ be some partitioning on the i-unfolding trees $\mathcal{T}^{(i)}$ which includes all i-unfolding trees appearing in G and G'. For some i-unfolding tree T, $\psi_{\rho(T)}$ denotes the vector with elements indexed by the k partitions of ρ. At index corresponding to the partition that T belongs to (i.e. index $\rho(T)$) set entry to 1, and to 0 everywhere else. Let $\Psi(G)$ be the sum over such vectors of i-unfolding trees of all vertices in graph G, i.e., $\Psi(G) = \sum_{v \in V(G)} \psi_{\rho(T^i(G,v))}$. Then, one can check the following equivalence:

$$\langle \Psi(G), \Psi(G') \rangle = \sum_{v \in V} \sum_{v' \in V'} \delta(\rho(T^i(G,v)), \rho(T^i(G',v')))$$

As the sum of kernels is also a kernel, it follows that the Relaxed Weisfeiler-Lehman kernel $k^h_{\text{R-WL}}(G,G')$ is positive semi-definite. □

Appendix 2 Clustering of unfolding trees (details)

In this section we provide a detailed and more formal description of the Wasserstein k-means algorithm applied to unfolding trees.

Unfolding tree vectors

We first show how to represent unfolding trees by real-valued vectors. Recall that the structure and depth preserving tree edit distance is calculated as the sum of

(A) the distance between the roots and
(B) the minimum cost of the perfect bipartite matching between child trees below these roots.

We therefore represent a depth-h unfolding tree T as a pair $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$, where the vectors $\mathbb{V}_r(T)$ and $\mathbb{V}_c(T)$ represent the root node's label $\ell(r(T))$ and the set of $(h-1)$ -unfolding child trees $F(r(T))$, respectively.

More precisely, let $\Sigma^\perp = (l_1, \dots, l_p, \perp)$ be the ordered set of all original vertex labels appearing in the graph dataset $\mathcal{G}$ and blank symbol $\perp$. Then the root node label of an unfolding tree T is represented by the vector $\mathbb{V}_r(T) = (x_1, \dots, x_p, x_\perp)$, where

$$x_i = \begin{cases} 1 & \text{if } i \in [p] \text{ and } \ell(r(T)) = l_i \\ 0 & \text{o/w} . \end{cases}$$

Furthermore, let $\mathcal{T}^{(h-1)} = (T_1^{(h-1)}, \dots, T_q^{(h-1)})$ be the ordered set of all pairwise non-isomorphic $(h-1)$-unfolding trees in $\mathcal{G}$. Then, the child trees $F(r(T))$ below the root of T are represented by the vector $\mathbb{V}_c(T) = (x_1, ..., x_q, x_{q+1})$ with

$$x_i = \begin{cases} |\{T' \in F(r(T)) : T' \equiv T_i^{(h-1)}\}| & \text{if } i \in [q] \\ 2d - deg(r(T)) & \text{o/w} \end{cases}$$

where d is the maximum vertex degree in $\mathcal{G}$.

The Wasserstein distance over unfolding tree vectors

We now show that the terms (A) and (B) above correspond to Wasserstein distances between root and between child vectors, respectively. In order to properly formulate the Wasserstein distances, we require cost matrices containing pairwise distances between labels of root nodes (i.e., elements of $\Sigma^{\perp}$) and pairwise distances between child trees (i.e., $(h-1)$-unfolding trees).

More precisely, for the alphabet Σ with $|\Sigma| = p$, let $M_r \in \mathbb{R}^{(p+1)\times(p+1)}$ be the distance matrix between labels and blank symbol $\perp = l_{p+1}$ according to $\gamma : \Sigma^{\perp} \times \Sigma^{\perp} \to \mathbb{R}$, i.e.:

$$M_r = (m_{ij})_{i,j\in[p+1]} \text{ with } m_{ij} = \gamma(l_i, l_j).$$

The pairwise distance matrix over the set $\mathcal{T}^{(h-1)}$ of $(h-1)$-unfolding trees together with the empty graph $T_{\perp} = T^{(h-1)}_{q+1}$ is given by

$$M_c = (m_{ij})_{i,j\in[q+1]} \text{ with } m_{ij} = \textsc{SdTed}(T^{(h-1)}_i, T^{(h-1)}_j),$$

where $q = |\mathcal{T}^{(h-1)}|$.

Then, for two depth-h unfolding trees T and T', the Wasserstein distance between their roots is equal to $\mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T'))$. Furthermore, the calculation of the minimum cost perfect bipartite matching between the sets of child trees below these roots can be reduced to computing the Wasserstein distance between vectors $\mathbb{V}_c(T)$ and $\mathbb{V}_c(T')$ using the following lemma which follows from the integral flow theorem.

Lemma 1 *For $x, x' \in \mathbb{N}^d$ there exists a transportation matrix $T \in \mathcal{T}(x, x')$ with $T \in \mathbb{N}^{d\times d}$ such that $\langle T, C\rangle = \mathcal{W}^C(x, x')$ for any cost matrix $C \in \mathbb{R}^{d\times d}$.*

This lemma implies that (B) above is equivalent to $\mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. Putting all together we have:

$$\begin{aligned}&\textsc{SdTed}(T, T')\\ &\quad = \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mathbb{V}_r(T')) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T')).\end{aligned}$$

Unfolding tree barycenters

We now show how to represent and calculate the barycenters of sets of unfolding trees for Wasserstein k-means. A barycenter of a set of unfolding tree vectors is a point which minimizes the sum of distances to all vectors in the set. Similarly to unfolding tree vectors defined in Sect. 4, the barycenter is a pair $(\mathbb{R}^{p+1}, \mathbb{R}^{q+1})$. While a barycenter, in general, does not correspond to an existing unfolding tree, the Wasserstein distance between unfolding tree and center vectors can be computed nonetheless.

Formally, for a set of depth-h unfolding trees $S \subseteq \mathcal{T}^{(h)}$, consider the set

$$\mathbb{V}^{(h)} = \{(V_r, V_c) \in \mathbb{R}^{p+1} \times \mathbb{R}^{q+1} : ||V_r||_1 = 1 \text{ and } ||V_c||_1 = 2d\}.$$

Note that $\mathbb{V}(T) \in \mathbb{V}^{(h)}$ for all $T \in \mathcal{T}^{(h)}$. For any two $X, Y \in \mathbb{V}^{(h)}$, we define their Wasserstein distance by

$$\mathcal{W}_{M_c}^{M_r}(X, Y) = \mathcal{W}^{M_r}(X_r, Y_r) + \mathcal{W}^{M_c}(X_c, Y_c).$$

Using these notions, for the barycenter $\mu \in \mathbb{V}^{(h)}$ minimizing the sum of Wasserstein distances to depth-h unfolding trees in S we have

$$\begin{aligned}&\min_{\mu} \sum_{T \in S} \mathcal{W}_{M_c}^{M_r}(\mathbb{V}(T), \mu)\\ &= \min_{\mu_r} \sum_{T \in S} \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mu_r) + \min_{\mu_c} \sum_{T \in S} \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mu_c)\end{aligned}$$

Thus, the barycenter μ is the pair (μ_r, μ_c).

The Wasserstein *k*-means algorithm for unfolding trees

Using the above definitions, we are ready to formulate the Wasserstein k-means clustering algorithm for unfolding trees in the form of Lloyd's algorithm (Lloyd, 1982):

1. **Initialization**: Choose a k-subset $\{\mu_1, ..., \mu_k\} \subseteq \{\mathbb{V}(T) : T \in \mathcal{T}^{(h)}\}$
2. **Assignment**: $C_i = \{T \in \mathcal{T}^{(h)} : \mathcal{W}_{M_c}^{M_r}(\mathbb{V}(T), \mu_i) \leq \mathcal{W}_{M_c}^{M_r}(\mathbb{V}(T), \mu_j), \forall j \in [k]\}$
3. **Update**: $\mu_i = \arg\min_{\mu \in \mathbb{V}^{(h)}} \sum_{T \in C_i} \mathcal{W}_{M_c}^{M_r}(\mathbb{V}(T), \mu)$

Steps 2 and 3 are repeated until clusters do not change anymore, i.e., the algorithm converges, or a predefined number of iterations has been reached.

Figure 10 visualizes the concept of clustering unfolding trees using Wasserstein k-means.

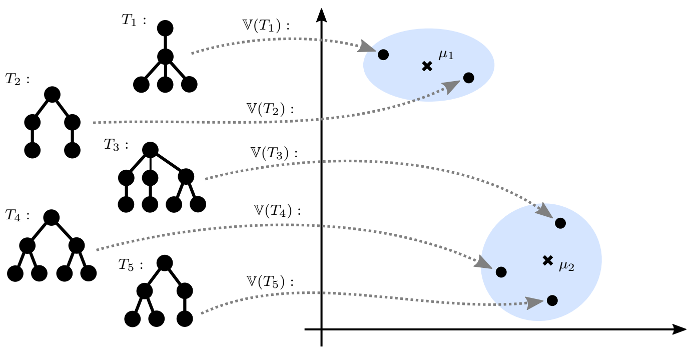


Fig. 10 Conceptual visualization of the mapping of 2-unfolding trees into $\mathbb{V}^{(2)}$ space. Center points μ_1, μ_2 minimize the sum of distances (defined by $\mathcal{W}_{M_c}^{M_r}$) to all points in the clusters (depicted blue). Thus, the corresponding unfolding trees within a cluster have pairwise small SDTED

Table 3 Structural information of molecular graph benchmark datasets $|D|$, $|C|$, and Δ, denote the number of graphs, number of classes, and maximum degree in a dataset. Σ_0 is the number of distinct vertex labels. Σ_1, Σ_2, Σ_3 are the amounts of distinct Weisfeiler-Lehman labels for depth $h = 1, 2, 3$, respectively

Dataset	$\|D\|$	$\|C\|$	$\varnothing\|V\|$	$\varnothing\|E\|$	$\frac{\varnothing\|E\|}{\varnothing\|V\|}$	Δ	Number of node labels			
							Σ_0	Σ_1	Σ_2	Σ_3
BZR	405	2	35.8	38.4	1.1	4	10	89	677	2058
COX2	467	2	41.2	43.4	1.1	4	8	71	537	1610
DHFR	756	2	42.4	44.5	1.0	4	9	71	630	2478
MUTAG	188	2	17.9	19.8	1.1	4	7	33	174	572
NCI1	4110	2	29.9	32.3	1.1	4	37	292	4058	23k

Appendix 3 Complexity analysis

The computation of clusters over a set of i-unfolding trees $\mathcal{T}^{(i)}$ requires the respective distance matrix M_c. As $|\mathcal{T}^{(i)}|$ is bounded by the number of vertices N in graph dataset $\mathcal{G}$ for all values i, M_c can be computed by $O(N^2)$ invocations of the Wasserstein distance function. Employing a linear programming solution yields a complexity of $\tilde{O}(N^3)$ for each such invocation, which can however be improved using approximation methods (Cuturi, 2013). We note that generally $|\mathcal{T}^{(i)}| \ll N$ for relatively small i, which immensely lowers the complexity for the calculation of M_c as well as Wasserstein distance computations in practice.

Let us consider a single iteration of the k-means Wasserstein algorithm. The assignment step of the above algorithm requires kN invocations of the Wasserstein distance function. To compute the set of cluster centers, we consider the Iterative Bregman Projections algorithm which provides an ϵ-approximation yielding an overall complexity of $\tilde{O}(\frac{k(2d)^2N^3}{2\epsilon})$ (Kroshnin et al., 2019) in the update step.

Appendix 4 Experimental evaluation (cont.)

In this section we provide additional experimental evaluations.

Datasets. We include several molecular benchmark datasets (Morris et al., 2020) which contain small graphs of fairly simple structure, i.e., they have roughly as many nodes as edges and a small maximum degree. The datasets are annotated for binary target properties. Detailed structural information on these graphs can be found in Table 3.

Evaluation. Figure 11 shows classification accuracies for real-world molecular datasets. On all datasets, our approach is in close range to the best performing kernels. Only, for NCI1 there is a noticeable performance gap of about 4%. To statistically evaluate the results, we performed two-sample t-tests (with a significance threshold of 0.05) corrected by the Bonferroni method to make up for the number of tests within each dataset. The results show that for none of the datasets there is a method that significantly outperforms all others.

The overall results suggest that a relaxation of the Weisfeiler-Lehman kernel is not advantageous when applied to these simple molecular graphs. This may be explained by

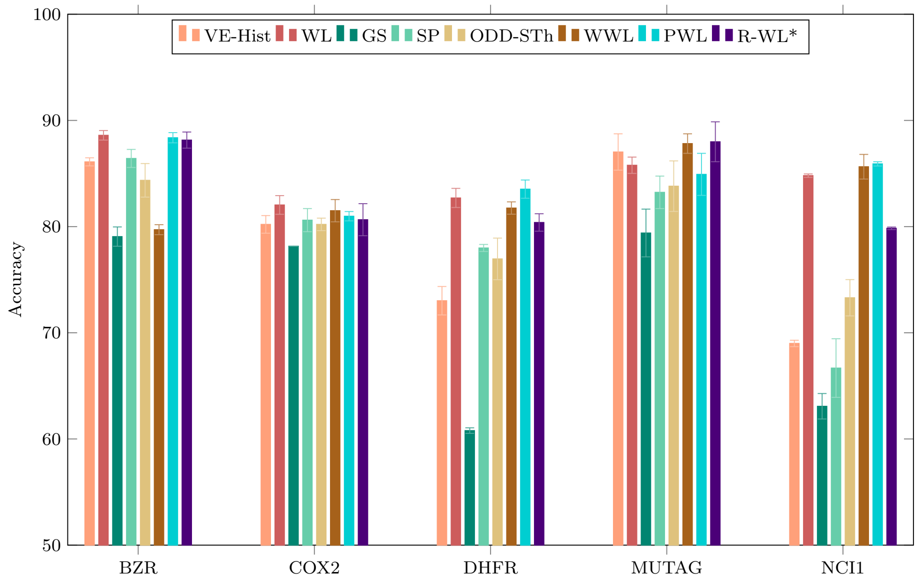


Fig. 11 Classification accuracies and std. deviations for molecular datasets in %

the assumption that structurally similar unfolding trees (corresponding to e.g. functional groups) can have completely opposing chemical properties. Clustering might thus even be disadvantageous for this kind of data. It is noteworthy that except for the datasets DHFR and NCI1, the simple histogram baseline kernel (VE-Hist) is on par with the best performing kernels.

Investigation of the depth parameter *h*

Figure 12 investigates the influence of the depth parameter h on the predictive performance of our kernel variant R-WL* for datasets IMDB-B, REDDIT-B and the EGONETS-1 to EGONETS-4. The results show a significant increase from $h = 1$ to $h = 2$, while in the majority of considered datasets, the predictive performance does not notably change any further for values $h > 2$.

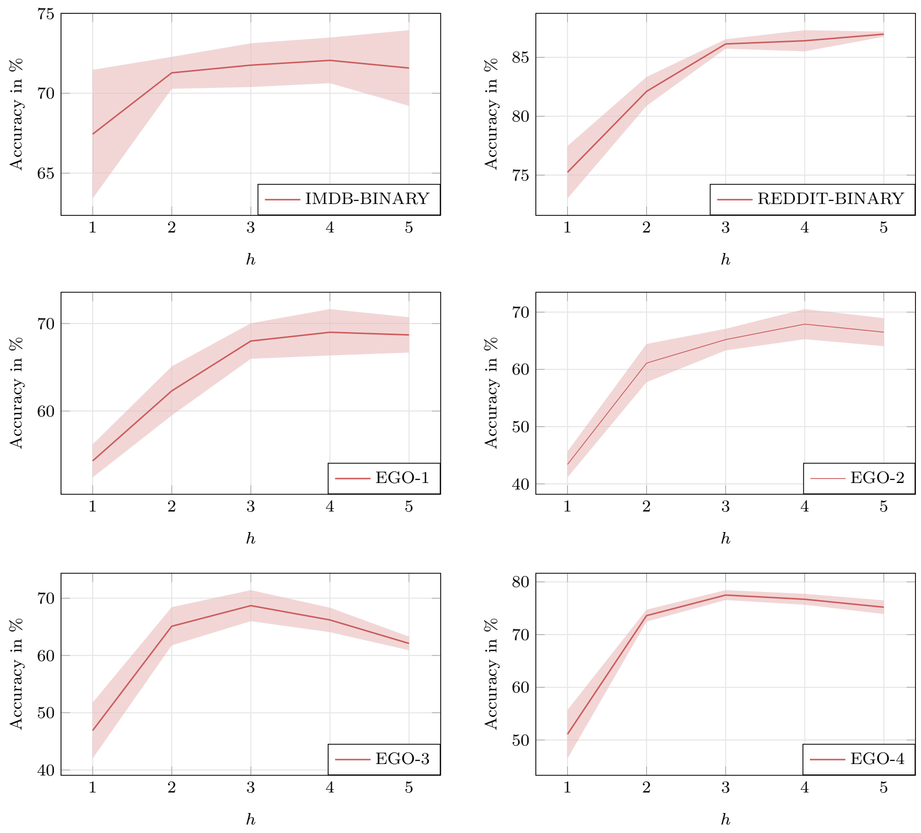


Fig. 12 Predictive performance of the R-WL* kernel for different choices of parameter *h*

Acknowledgements This material was produced within the Competence Center for Machine Learning Rhine-Ruhr (https://www.ml2r.de) which is funded by the Federal Ministry of Education and Research of Germany (Grant No. 01|S18038C). The authors gratefully acknowledge this support.

Author contributions All authors contributed to the submitted manuscript. Material preparation, data collection and analysis were performed by Till Schulz, Pascal Welke and Tamás Horváth. The first draft of the manuscript was written by Till Schulz and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This material was produced within the Competence Center for Machine Learning Rhine-Ruhr (https://www.ml2r.de) which is funded by the Federal Ministry of Education and Research of Germany (Grant No. 01|S18038C). The authors gratefully acknowledge this support.

Availability of data and material All material on the performed experiments including datasets are available online. The link https://github.com/mlai-bonn/GenWL can be found in the article.

Code availability The code used in experiments is available online. A references https://github.com/mlai-bonn/GenWL can be found in the article.

Declarations

Conflict of interest This study was funded by Competence Center for Machine Learning Rhine-Ruhr (see above). The authors have no other financial or proprietary interests in any material discussed in this article.

Ethical approval The authors assure that all potential conflicts of interest have been disclosed above. All research performed in this article was done in accordance with the ethical standards of the Springer ethics code of conduct.

Consent to participate All listed authors have agreed to submitting the attached article to the Machine Learning Journal.

Consent for publication Not applicable. (The authors assure that no figures, tables and text passages have been obtained from third party authors.)

References

Al-Rfou, R., Perozzi, B., & Zelle, D. (2019). DDGK: learning graph representations for deep divergence graph kernels. In: *The World Wide Web Conference 2019, ACM*, pp. 37–48.

Babai, L., & Kucera, L. (1979). Graph canonization in linear average time. In: *FOCS*, pp. 39–46.

Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science, 337*(1–3), 217–239.

Borgwardt, K. M., & Kriegel, H. P. (2005). Shortest-path kernels on graphs. In: *ICDM '05*, pp 74 — 81.

Chen, Z., Villar, S., Chen, L., & Bruna, J. (2019). On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in Neural Information Processing Systems, 2019,* 15868–15876.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In: *Advances in Neural Information Processing Systems*, pp. 2292–2300.

Cuturi, M., & Doucet, A. (2014). Fast computation of wasserstein barycenters. In: *International Conference on Machine Learning*, pp. 685–693.

Da San Martino, G., Navarin, N., & Sperduti, A. (2012). A tree-based kernel for graphs. In: *SIAM International Conference on Data Mining*, pp. 975–986.

Dell, H., Grohe, M., & Rattan, G. (2018). Lovász meets Weisfeiler and Leman. In: *International Colloquium on Automata, Languages and Programming*, pp. 40:1–40:14.

Errica, F., Bacciu, D., & Micheli, A. (2020). Theoretically expressive and edge-aware graph learning. *ESANN, 2020,* 175–180.

Gärtner, T., Flach, P., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In: *COLT/Kernel*, pp. 129–143.

Haussler, D. (1999). *Convolution kernels on discrete structures*. Technical report, Department of Computer Science, University of California at Santa Cruz.

Irpino, A., Verde, R., & de Carvalho, F. A. T. (2014). Dynamic clustering of histogram data based on adaptive squared Wasserstein distances. *Expert Systems with Applications, 41*(7), 3351–3366.

Kriege, N. M., Giscard, P., & Wilson, R. C. (2016). On valid optimal assignment kernels and applications to graph classification. In: *Advances in Neural Information Processing Systems*, pp. 1615–1623.

Kriege, N. M., Johansson, F. D., & Morris, C. (2020). A survey on graph kernels. *Applied Network Science, 5*(1), 6.

Kroshnin, A., Tupitsa, N., Dvinskikh, D., Dvurechensky, P., Gasnikov, A., & Uribe, C. (2019). On the complexity of approximating Wasserstein barycenters. *Proceedings of the International Conference on Machine Learning, 97,* 3530–3540.

Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory, 28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., & Grohe, M. (2019). Weisfeiler and leman go neural: Higher-order graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence, 2019,* 4602–4609.

Morris, C., Kriege, N. M., Bause, F., Kersting, K., Mutzel, P., & Neumann, M. (2020). Tudataset: A collection of benchmark datasets for learning with graphs. In: *GRL+@ICML*, arXiv:2007.08663.

Rieck, B., Bock, C., & Borgwardt, K. (2019). A persistent Weisfeiler-Lehman procedure for graph classification. In: *International Conference on Machine Learning*, pp. 5448–5458.

Roth, V., Laub, J., Kawanabe, M., & Buhmann, J. (2003). Optimal cluster preserving embedding of nonmetric proximity data. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(12), 1540–1551. https://doi.org/10.1109/TPAMI.2003.1251147.

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., & Borgwardt, K. M. (2009). Efficient graphlet kernels for large graph comparison. In: *Artificial Intelligence and Statistics*, pp. 488–495.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research, 12,* 2539–2561.

Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., & Vazirgiannis, M. (2018). GraKel: A graph kernel library in Python. arXiv preprint arXiv:1806.02193. https://github.com/ysig/GraKeL.

Smola, A. J., & Vishwanathan, S. (2003). Fast kernels for string and tree matching. In: *Kernel Methods in Computational Biology*, pp. 585–592.

Togninalli, M., Ghisu, E., Llinares-López, F., Rieck, B., & Borgwardt, K. (2019). Wasserstein Weisfeiler-Lehman graph kernels. In: *Advances in Neural Information Processing Systems*, pp. 6439–6449.

Wang, Y. J., & Wong, G. Y. (1987). Stochastic blockmodels for directed graphs. *Journal of the American Statistical Association, 82*(397), 8–19.

Weisfeiler, B., & Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia, 2*(9).

Wu, G., Chang, E. Y., & Zhang, Z. (2005). An analysis of transformation on non-positive semidefinite similarity matrix for kernel machines. In: *Proceedings of the 22nd International Conference on Machine Learning*.

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In: *ICLR 2019*.

Zafarani, R., & Liu, H. (2009).Social computing data repository at ASU. http://socialcomputing.asu.edu, accessed: 2019-06-25, offline at the time of submission.