Idea:
Instead of the difficult WL isomophism test,
use the graph edit distance
to get a more relaxed similarity measure.

A Generalized Weisfeiler-Lehman Graph Kernel

Till Hendrik Schulz * Tamás Horváth *† Pascal Welke * Stefan Wrobel *†

Abstract

The Weisfeiler-Lehman graph kernels are among the most prevalent graph kernels due to their remarkable time complexity and predictive performance. Their key concept is based on an implicit comparison of neighborhood representing trees with respect to equality (i.e., isomorphism). This binary valued comparison is, however, arguably too rigid for defining suitable similarity measures over graphs. To overcome this limitation, we propose a generalization of Weisfeiler-Lehman graph kernels which takes into account the similarity between trees rather than equality. We achieve this using a specifically fitted variation of the well-known tree edit distance which can efficiently be calculated. We empirically show that our approach significantly outperforms state-of-the-art methods in terms of predictive performance on datasets containing structurally more complex graphs beyond the typically considered molecular graphs.

1 Introduction

Since Haussler's pioneer work [5] on convolution kernels over discrete structures, *graph kernels* have become one of the most common tools for learning with graphs. They gained major popularity by enabling the application of kernel methods. For example, excellent predictive performance can be obtained by combining graph kernels with support vector machines. One prominent family of graph kernels is the *Weisfeiler-Lehman kernel* framework [11]. Kernels in this family are based on the idea of the *Weisfeiler-Lehman isomorphism test* [16], which iteratively relabels vertices by propagating neighborhood information. Each such label implicitly corresponds to a rooted tree, called *unfolding tree* (see Fig. 1b). For space limitations, we limit the scope of this work to the most established member, the *Weisfeiler-Lehman subtree kernel*. However, we note that the generality of our approach allows its application to *all* Weisfeiler-Lehman graph kernels.

Despite their distinguished speed, Weisfeiler-Lehman graph kernels are conceptually limited to comparing labels, or equivalently, unfolding trees w.r.t. *equality*. While this comparison is extremely well-suited for graph isomorphism tests, 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 it resembles T_3, the Weisfeiler-Lehman kernel [11] simply treats them all as unequal and is thus unable to *quantify* the apparent difference among the pairwise similarities between the unfolding trees.

Motivated by these considerations, we *relax* the above strictness by proposing a method which compares Weisfeiler-Lehman labels, or equivalently unfolding trees, with regard to a much *finer* similarity measure than the binary valued one. More precisely, we employ a similarity between Weisfeiler-Lehman labels based on the concept of *tree edit distances* between their respective unfolding trees. These kind of distances provide a natural comparison for trees. On an abstract level, they are defined by the minimum cumulative cost of *edit operations* needed to transform one tree into another. Since in this work we deal with unfolding trees, we define a variant of the tree edit distance specific to this special type of rooted 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 *relaxed Weisfeiler-Lehman subtree kernel* is to identify groups of *similar* Weisfeiler-Lehman labels by *clustering* (visualized in Fig. 1c). The elements within a cluster are then treated as *identical* labels. That is, we generalize the ordinary Weisfeiler-Lehman kernel [11] by regarding two unfolding trees equivalent if they belong to the same cluster, i.e., if they have a *small* distance to each other. In this way, the ordinary Weisfeiler-Lehman kernel is the *special* case where labels are considered equivalent only if they have distance zero. For partitioning the Weisfeiler-Lehman labels, we use *Wasserstein k-means* clustering [6]. This choice is motivated by our result that the 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 relaxed Weisfeiler-Lehman kernel on a set of real-world datasets. Our experimental results clearly show that our approach considerably outperforms state-of-the-art kernels (including the ordinary Weisfeiler-Lehman subtree kernel) on datasets containing dense and structurally diverse graphs.

Related Work While conventional graph kernels define similarity in terms of mutual substructures such as walks [4], paths [2], small subgraphs [12] or subtrees

*Dept. of Computer Science, University of Bonn, Germany
Email: {schulzth, horvath, welke, wrobel}@cs.uni-bonn.de
†Fraunhofer IAIS, Sankt Augustin, Germany

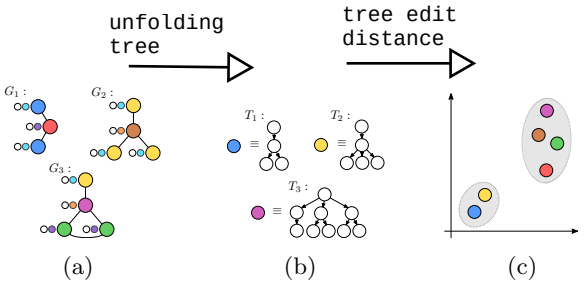


Figure 1: (a) depicts (initially unlabeled) graphs where vertices are labeled with the first two Weisfeiler-Lehman labels (colored). (b) shows the rooted unfolding trees corresponding to the blue, yellow and pink WL-labels, each representing a neighborhood. T_1 (blue) differs from T_2 (yellow) by only 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.

[11], recent work has moved away from solely counting equivalent substructures. For example, [8] introduces a kernel which computes an optimal assignment between vertices. Similarly, in [15] the authors employ the concept of optimal transportation as a form of 'soft-matching' on vertices. Both methods measure similarity between vertices based on variants of Weisfeiler-Lehman induced similarity. However, their vertex matching abilities are ultimately still limited by the rigid Weisfeiler-Lehman label refinement method.

The concept of comparing vertices by a finer similarity measure than the equivalence of their neighborhoods can also be found in [9]. In that work, 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 the authors in [9] compute similarities by applying tree kernels (e.g. [14]) on sets of trees extracted from the DAGs, we employ the concept of tree edit distances as similarity measure.

The rest of the paper is organized as follows. We collect the necessary notions in Sect. 2, discuss the tree edit distance in Sect. 3, and present our graph kernel in Sect. 4. We report the empirical results in Sect. 5 and conclude in Sect. 6.

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 in* v, denoted $T[v]$, is the subgraph of T that is rooted at v and induced by all 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 edits 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 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 $T\mathbf{1}_{n'} = x$, $\mathbf{1}_n^T T = x'$ 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 $x_1, ..., x_n \in \mathbb{R}^n$ and a cost matrix $C^{n \times n}$, we define the *barycenter* as $\arg\min_c \sum_{i \in [n]} \mathcal{W}^C(x_i, c)$.

3 The Weisfeiler-Lehman Tree Edit Distance

In this section, we briefly recap the Weisfeiler-Lehman vertex relabeling method [16] and define a distance function on Weisfeiler-Lehman labels. We give an algorithm computing this distance and prove that it can be efficiently calculated.

3.1 The Weisfeiler-Lehman method The *Weisfeiler-Lehman* (WL) method [16] was originally designed to decide isomorphism between graphs

```
Is it true that
 1. \mathbb{V}_c(T)[0] = 1, due to the subgraph 1-2-3,
 2. \mathbb{V}_c(T)[1] = 1, due to the subgraph 3-1,
 3. \mathbb{V}_c(T)[2] = 0, due to NO subgraph 1-(1,2,3),
 4. \mathbb{V}_c(T)[3] = 1, due to the empty subgraph (padding to have the array length at r(T)+r(T')+1=2+1+1,
    to encode all possible  subgraphs from the other graph,
 5. \mathbb{V}_c(T')[2] = 1, due to the subgraph 1-(1,2,3) and
 6. \mathbb{V}_c(T')[3] = 2, due to two empty subgraphs, corresponding to the two subgraphs present in T
     but not in T'
?
```

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.

Shervashidze et al. [11] 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

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

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 space limitations we focus on the *subtree kernel* [11] which employs the base kernel

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

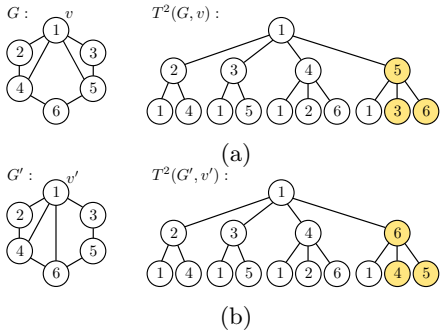


Figure 2: Unfolding trees $T^2(G, v)$ and $T^2(G', v')$. As v and v' have structurally similar roles in G, resp. G', their unfolding trees differ only slightly (labeled yellow). The vertex corresponding to v, resp. v', appears again several times at depth 2 of $T^2(G, v)$, resp. $T^2(G', v')$.

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.

Another view of the Weisfeiler-Lehman procedure is that for each iteration i, it implicitly constructs tree patterns of depth i which are being compressed into labels. Each such tree, denoted $T^i(G, v)$, is called the depth-i *unfolding tree* (or simply, i-unfolding tree) of G at v. 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 too restrictive, 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 differ-

ent unfolding trees are treated identically to two unfolding trees which differ by only very little.

Solution: To overcome this drawback, we propose a finer 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., [1]). However, for our purpose it suffices to consider a constrained 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 (cf. [1]):

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. $\forall (v_1, v_1'), (v_2, v_2') \in M : v_1 = v_2 \Leftrightarrow v_1' = v_2'$, *(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 $\mathrm{SDM}(T, T')$.

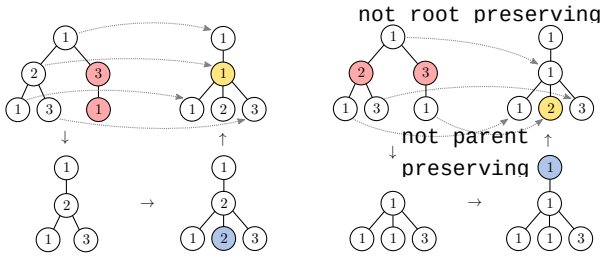
SDMs represent sequences of edit operations subject to the above constraints that transform trees into trees. More precisely, for an SDM (M, T, T') let $T = T_0, T_1, \ldots, T_k$ be a sequence of trees such that T_{i+1} is obtained from T_i by applying one of the following atomic transformations:

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.



(a) (b)

Figure 3: Two mappings from one unfolding tree into another. 3a depicts a mapping which is structure and depth preserving whereas 3b 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.

PROPOSITION 3.1. *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 (a) is a structure and depth preserving mapping from T into T' which visibly preserves the depth as well as pairwise sibling relationships of all mapped vertices. In contrast, while the edit script in (b) 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 define the distance between two unfolding trees.

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'

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**

$$\delta_{ij} = \begin{cases} \text{SDTED}(T_i, T'_j) & \text{if } T_i \not\equiv T_{\perp} \text{ and } T'_j \not\equiv T_{\perp} \\ \sum_{v \in V(T_i)} \gamma(\ell(v), \perp) & \text{if } T_i \not\equiv T_{\perp} \text{ and } T'_j \equiv T_{\perp} \\ \sum_{v' \in V(T'_j)} \gamma(\ell(v'), \perp) & \text{if } T_i \equiv T_{\perp} \text{ and } T'_j \not\equiv T_{\perp} \\ 0 & \text{o/w .} \end{cases}$$

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

DEFINITION 2. *Let T, T' be unfolding trees over the 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 $\gamma(M)$ for an SDM (M, T, T') is*

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

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 SDTED(T, T'), is then defined by

$$\text{SDTED}(T, T') = \min\{\gamma(M) : (M, T, T') \in \text{SDM}(T, T')\}$$

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.

3.3 The Unfolding Tree Edit Distance Algorithm We now show that for any pair of unfolding trees T, T', SDTED(T, T') can efficiently be calculated in a recursive manner. 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 SDTED(T_i, T'_j) 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 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 $F = \{T_1, \ldots, T_k\}$ and $F' = \{T'_1, \ldots, T'_{k'}\}$, 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 (c.f. 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). We have the following result:

Proof

Correctness of Alg.1

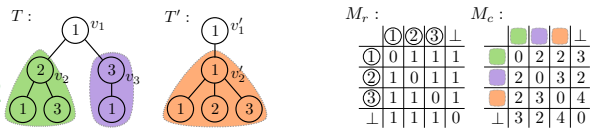
THEOREM 3.1. *Given unfolding trees T, T' with labels from Σ and a cost function $\gamma : \Sigma^{\perp} \times \Sigma^{\perp} \to \mathbb{R}$ over Σ and $\perp$, Alg. 1 returns* SDTED(T, T').

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 Def. 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.

Example Fig. 4

By the construction of unfolding trees, vertices closer to v in G begin to appear at smaller depths in

cost = 4
-del 1
-del 3
-relabel 2
-ins 2



(a) (b)

Figure 4: (b) provides the SDTED between pairs of 0-unfolding trees (M_r) and 1-unfolding trees (M_c) which are necessary to compute $\text{SDTED}(T, T')$. 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 $\text{SDTED}(T, T') = 4$.

$T^i(G, v)$. 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 $\text{SDTED}(T_i, T_j)$ 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' 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 $\tilde{O}((2d)^3)$, where h is the depth and d the maximum degree of T, T'.

4 The Relaxed Weisfeiler-Lehman Subtree Kernel

Using the definitions and results of Sect. 3, we now introduce our novel *relaxed Weisfeiler-Lehman subtree kernel* and show that it is in fact a *generalization* of the original Weisfeiler-Lehman subtree kernel [11]. Its key idea is to *relax* the rigid comparison of unfolding trees by equality (i.e., isomorphism) used in the Weisfeiler-Lehman 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. Clearly, $k^h_{\text{R-WL}}(G, G')$ is positive semi-definite and hence a kernel as the right hand side can be rewritten as the inner product of graph feature vectors consisting of cluster membership counts (proof in Appenix A). 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 $\text{SDTED}(T, T') = 0$). 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 $\text{SDTED}(T, T') = 0$, our definition allows $\text{SDTED}(T, T') \geq 0$ 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* [6] 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 relaxed 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 [3].

Below we address the most important ingredients of Wasserstein k-means 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 description, furthermore, allows for the calculation of center points using *Wasserstein barycenters*. For space limitations, we solely outline these concepts in this

article. A more detailed description as well as a complexity analysis can be found in the appendix.

Unfolding Tree Vectors In order to effectively apply Wasserstein k-means, the unfolding trees need to be represented 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 a perfect bipartite matching between child trees below these roots (c.f. Alg. 1). We therefore represent an i-unfolding tree T as a pair $\mathbb{V}(T) = (\mathbb{V}_r(T), \mathbb{V}_c(T))$, where the vector $\mathbb{V}_r(T)$ represents the root node's label $\ell(r(T))$ and $\mathbb{V}_c(T)$ represents the set of $(i-1)$-unfolding child trees $F(r(T))$. $\mathbb{V}_r(T)$ is realized by a vector with entry 1 at index corresponding to its root node label $\ell(r(T))$ and 0 everywhere else, and $\mathbb{V}_c(T)$ is made up of counts of 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. An example of these vector representations is contained in the description of Fig. 4.

The Wasserstein Distance over 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. 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 original node labels. We can show that for two depth-i unfolding trees T and T', the 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 (cf. Alg. 1) can be reduced to computing the Wasserstein distance between $\mathbb{V}_c(T)$ and $\mathbb{V}_c(T')$, i.e., $\mathcal{W}^{M_c}(\mathbb{V}_c(T), \mathbb{V}_c(T'))$. 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 of this equivalence is given in Fig. 4.

Unfolding Tree Barycenters The above reformulation allows us to calculate *barycenters* of sets of unfolding trees for Wasserstein k-means. 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. More formally, the barycenter of S is a pair (μ_r, μ_c) calculated as follows:

$$\arg\min_{\mu_r, \mu_c} \sum_{T \in S} \mathcal{W}^{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mu_c) \tag{4.1}$$

Note that 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:

$$\mathcal{W}^{M_r}(\mathbb{V}_r(T), \mu_r) + \mathcal{W}^{M_c}(\mathbb{V}_c(T), \mu_c) \tag{4.2}$$

The Wasserstein k-Means Algorithm for Unfolding Trees Using the above concepts, the application of the Wasserstein k-means clustering algorithm for unfolding trees is straightforward. (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 equation 4.2). (iii) Finally, the centers of the newly defined clusters are recalculated (using equation 4.1). 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.

4.1 A Faster Kernel Variant

For many graph datasets the number of Weisfeiler-Lehman labels, or equivalently the number of (pairwise non-isomorphic) unfolding trees, 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 is computationally expensive. We thus propose a variant of our kernel which approximates distances between unfolding trees using their cluster centers.

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 $C_1, ..., C_k$ of $\mathcal{T}^{(i)}$ with centers $\mu_1, ..., \mu_k$ 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. Subsequently, these distances are used in iteration $i+1$. Hence, 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.

5 Empirical Evaluation

Below, we evaluate the predictive performance of our approach on a set of established as well as novel real-world datasets. Our results show that our approach increasingly outperforms all considered competitor kernels with growing density of dataset graphs.

We note that in this short version, we limit the eval-

uation 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 (see Appendix D), an explicit consideration of all unfolding trees may become computationally too expensive on more complex graphs.

5.1 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 [11] (with parameter $h \in [5]$), the graphlet sampling (GS) kernel [12] (with parameters $\epsilon = 0.1$, $\delta = 0.1$ and $k \in \{3, 4, 5\}$), the shortest-path (SP) kernel [2], and the ODD-STh kernel [9] (with parameter $h \in [4]$) using the implementation of [13]. Furthermore, we include the recently published Wasserstein Weisfeiler-Lehman graph (WWL) kernel [15] and the Persistent Weisfeiler-Lehman (PWL) graph kernel [10]. In both cases, we select the depth parameter $h = 5$. 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*, we choose the number of clusters $k = \sqrt{|\Sigma_i|}$ and perform a total of 3 clusterings (i.e. $|\Theta_i| = 3$), using depth parameter h up to 4 and cost 1 for all relabeling, deletion and insertion operations. This particular choice for k is made in order to select the number of clusters relative to the amount of Weisfeiler-Lehman labels in each iteration as well as to significantly limit the computational complexity of the clustering. The prediction performances are measured in terms of accuracy obtained by support vector machines (SVM) using a 10-fold cross-validation. 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. Furthermore, runtimes can be found in Appendix D.

5.2 Datasets We conduct experiments on the benchmark datasets IMDB-BINARY and REDDIT-BINARY containing subgraphs of online networks [7]. IMDB-BINARY consists of collaboration networks between actors/actresses each annotated against movie genres, whereas graphs in REDDIT-BINARY 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 of varying size and density. The datasets *EGONETS-x* contain ego network graphs extracted from four different social networks. They contain increasingly larger and more dense ego networks with growing index x. Here, ego networks are subgraphs induced by a vertex's neighbors. Graphs within each dataset were randomly chosen from the set of all egonets but underlie size- and density-specific constraints to ensure that a simple count of nodes and edges is not sufficient for prediction tasks. The learning task is to assign an egonet to the network it was extracted from. We provide detailed structural properties of all datasets in Appendix D.

5.3 Results & Discussion Table 1 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-BINARY, REDDIT-BINARY and EGONETS-1 (which all have an average node-to-edge ratio up to roughly $1 : 4$), the R-WL* kernel considerably outperforms all others on the three remaining EGONETS datasets which contain significantly higher density graphs. The performance gap between the RWL* kernel and the best performing competitor becomes increasingly larger with a growing density in the dataset graphs, leading to an above 20% accuracy difference. 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 is not able to profit from any structural information exceeding node degrees as graphs share almost no i-unfolding trees for $i \geq 2$. In contrast, our approach clearly improves upon this limitation by identifying similar unfolding trees.

We, furthermore, evaluated our approach on traditional molecular datasets as well as synthetic benchmark datasets. We observed that while our approach does not prove to be advantageous on datasets containing mainly sparse and noise-free graphs such as molecular data, it soon outperforms all other considered kernels on datasets containing fuzzy and structurally diverse graphs. (For a detailed description see Appendix D.)

In summary, it is apparent that the original Weisfeiler-Lehman kernel is only suitable when there are only few different unfolding trees in the graphs of the dataset such that these graphs share sufficiently many labels in order to compute meaningful similarities. Our method makes up for this drawback. Its ability to identify similar vertex neighborhoods leads to major increases in predictive performance on datasets containing *noisy* and *structurally diverse* graphs.

6 Concluding Remarks

We introduced a generalization of the Weisfeiler-Lehman graph kernel which allows for finer similarity measures between Weisfeiler-Lehman labels. Our evaluation showed that this generalization improves upon a key weakness of the original Weisfeiler-Lehman graph

	IMDB-B.	REDDIT-B.	EGONETS-1	EGONETS-2	EGONETS-3	EGONETS-4
VE-Hist	70.82 ± 0.53	84.07 ± 0.28	29.30 ± 2.84	27.70 ± 2.64	25.10 ± 2.56	24.60 ± 2.10
WL	72.68 ± 1.20	76.08 ± 0.66	54.50 ± 0.94	58.60 ± 1.29	57.70 ± 2.08	57.00 ± 1.77
GS	66.28 ± 1.00	78.02 ± 0.58	64.10 ± 1.98	56.80 ± 1.20	53.30 ± 2.86	51.30 ± 1.96
SP	49.32 ± 0.62	49.86 ± 1.96	67.00 ± 1.46	58.20 ± 1.04	56.60 ± 2.90	54.30 ± 3.49
ODD-STh	56.52 ± 1.86	–	32.80 ± 5.84	37.90 ± 2.19	34.80 ± 2.80	34.40 ± 6.40
WWL	$\mathbf{72.94 \pm 0.59}$	–	58.20 ± 1.60	64.60 ± 1.08	58.10 ± 2.38	56.30 ± 1.92
PWL	72.58 ± 0.80	78.82 ± 0.31	56.95 ± 2.15	56.85 ± 2.48	50.35 ± 2.60	48.70 ± 1.55
R-WL*	72.16 ± 0.87	$\mathbf{87.02 \pm 0.38}$	$\mathbf{68.70 \pm 1.15}$	$\mathbf{69.30 \pm 1.20}$	$\mathbf{71.00 \pm 3.54}$	$\mathbf{77.50 \pm 0.94}$

Table 1: Classification accuracies and std. deviations for large network benchmark datasets in %. Cells marked "–" indicate computations which did not finish within 24 hours.

kernel and outperforms state-of-the-art methods. We stress that while we presented our relaxed Weisfeiler-Lehman graph kernel only for the *subtree* kernel, the generality of our approach allows its application to *all* Weisfeiler-Lehman graph kernels. Furthermore, our results motivate several research questions. For one, alternative approaches to define meaningful similarities between labels might reduce expensive minimum cost perfect bipartite matching (or equivalently, Wasserstein) computations. On another note, as the distance function γ on initial graph labels can be defined by an arbitrary metric, the extension to attributed graphs is straightforward and promising.

References

[1] P. Bille. A survey on tree edit distance and related problems. *Theor Comp Sci*, 337(1-3):217–239, 2005.

[2] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. ICDM '05, pages 74 —- 81, 2005.

[3] M. Cuturi and A. Doucet. Fast computation of wasserstein barycenters. In *ICML*, pages 685–693, 2014.

[4] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT/Kernel*, pages 129–143, 2003.

[5] D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Univerisity of California - Santa Cruz, July 1999.

[6] A. Irpino, R. Verde, and F. de A. T. de Carvalho. Dynamic clustering of histogram data based on adaptive squared Wasserstein distances. *Expert Syst. Appl.*, 41(7):3351–3366, 2014.

[7] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann. Benchmark data sets for graph kernels, 2016.

[8] N. M. Kriege, P. Giscard, and R. C. Wilson. On valid optimal assignment kernels and applications to graph classification. In *NIPS*, pages 1615–1623, 2016.

[9] G. D. S. Martino, N. Navarin, and A. Sperduti. A tree-based kernel for graphs. In *SIAM SDM*, pages 975–986, 2012.

[10] B. Rieck, C. Bock, and K. Borgwardt. A persistent Weisfeiler-Lehman procedure for graph classification. In *ICML*, pages 5448–5458, 2019.

[11] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-Lehman graph kernels. *JMLR*, 12:2539–2561, 2011.

[12] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, pages 488–495, 2009.

[13] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis. GraKel: A graph kernel library in Python. *arXiv preprint arXiv:1806.02193*, 2018.

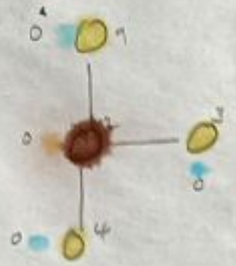
[14] A. J. Smola and S. Vishwanathan. Fast kernels for string and tree matching. In *NIPS*, pages 585–592. 2003.

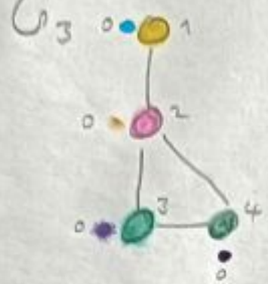
[15] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein Weisfeiler-Lehman graph kernels. In *NeurIPS*, pages 6439–6449. 2019.

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

GRAPH

G_1 

G_2 

G_3 

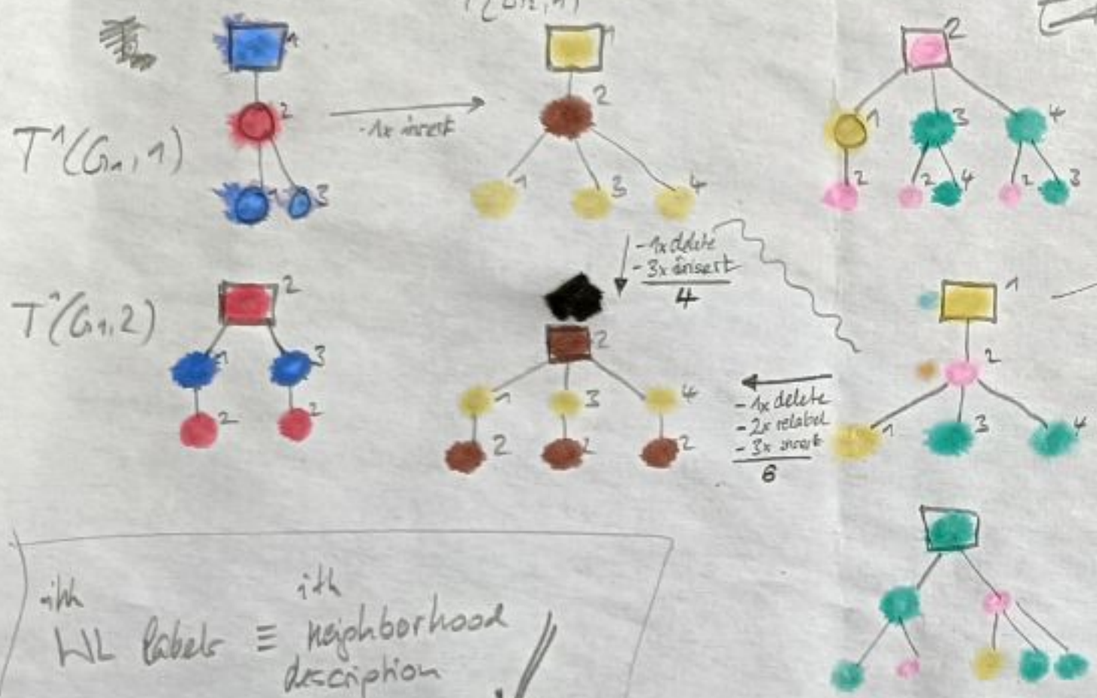
Example

Mapping:

Unfolding Tree

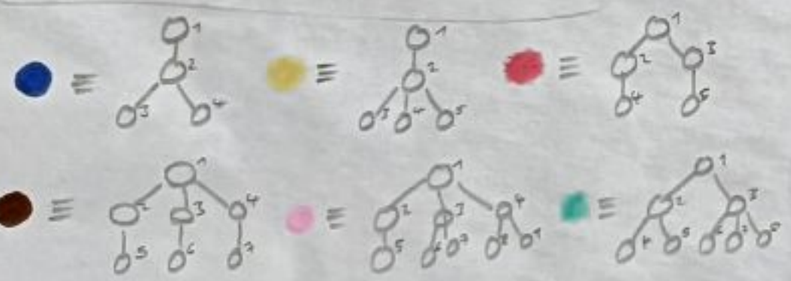
Neighborhood "size"	label
1 (○) 2 (∞) 3 (∞∞)	
neighborhood labels	?
	?

UNFOLDING TREES of the WL-Labels of depth 2 after 2 WL iterations



with WL-label ≡ ith neighborhood description

≡ rooted tree with height (i+1)



Tree edit

- relabel
- delete
- insert

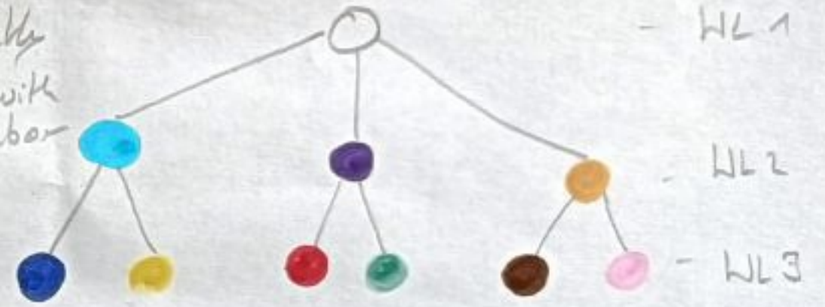
assume relabelling not of WL-label but vertex enumeration in the unfolding tree

?

[P. J. Schulz] tree pattern, that is compressed into a label (yellow)!

"yellow" has first one neighbour with exactly three neighbours

"brown" has exactly three neighbours with exactly one neighbor each



assume:

		- del 3 - 2x insert	- relabel - 4x insert	- del - 4x insert	- del - 6x insert
→	1	3	5	5	7
		- 2x del - 2x insert	- del - 4x insert	- del (no effect, 2nd) - 3x del	- del - 1x relabel - 3x insert
=	=	2	5	6	8
			4	4	6
				3x del 2x insert	2x del 3x insert
				5	5
				-	3
					-