Learning graph similarity measures using the Weisfeiler-Lehman label hierarchy

Fabrice Beaumont

University of Bonn, Germany

Abstract. In this expose I am outlining the research questions and research strategy of my master thesis. In my thesis, I am going to investigate the effect of learning such similarities between Weisfeiler Lehman labels and using these in the definition of a graph similarity measure, structured similarly to the Wasserstein Weisfeiler Lehman graph kernel proposed in 2019 by Togninalli et al. The goal is to discuss, whether learning such similarities can improve the flexibility of the resulting graph similarity measure. The machine learning will be restricted in the sense that the ground distance is set as a a tree metric on the Weisfeiler Lehman label hierarchy tree. The learning procedure is to iteratively update the edge weights in this tree and thus to change the resulting distance between the Weisfeiler-Lehman labels used.

Keywords: Weisfeiler Lehman · Tree Metric · Graph Classification · Metric Learning · Wasserstein distance. . . .

1 Introduction

Digitization is rapidly increasing the amount of digitally stored data. Such data is often represented by graphs, which may capture potentially complex structures. Common examples of such data storing graphs are social and sensor networks, program flows and structures in chemo- and bio-informatics (e.g. molecules, proteins, genes and chemical compounds). Based on functional properties of the graphs one may define classifications on them. To extend the classification of known graph instances to unknown instances is one of many different challenges when working with such graph data. Such classification can indicate, for example, whether a particular molecule can trigger mutations or not. In this framework it is usually assumed, that graphs with similar structure have similar properties and therefore should be classified similarly. Therefore, it is important to quantify the similarity between graphs [22]. Due to the lack of efficient similarity measures between graphs, the analysis, classification and prediction of graph data stays challenging.

The most sensitive similarity measure is the binary decision whether two given graphs are isomorphic (topologically identical) or not. There are several different approaches to defining a more sophisticated similarity measure. Some examples are measures based on the graph edit distance [2, 4], optimal assignment kernels [4], the skew spectrum [12] or the graphlet spectrum [13, 21]. In the context of similarity measures, graph kernels have the desirable property of enabling the application of a larger toolbox of kernelized learning methods on graph-structured data.

By restricting the kernels to substructures of graphs, they can be computable in polynomial time [7]. This restriction however may at the same time limit their ability to capture complex characteristics of the graphs. That is why most proposed variants do not generalize well to graphs with high-dimensional continuous vertex attributes [22]. In 2019, Togninalli et al. established their Wasserstein Weisfeiler-Lehman (WL) Graph Kernels as the new state of the art. These kernels are capable of processing both categorical labels and continuous attributes by utilizing the the Hamming distance and Euclidean distance as ground metric for the Wasserstein distance [23]. However, these rigid, rather arbitrary definition does not permit to adjust the distances between the labels (or attributes) used in the graph. This may be disadvantageous, since the labels constructed by the WL labeling scheme represent entire substructures, also known as unfolding trees [20]. In its naive version, two labels are distinguished as either equal or different. But since the substructures, that they represent, can differ largely or only little, a more nuanced differentiation may be appropriate. To do so, I introduce adjustable distances between the labels. These can also be useful to introduce-application specific knowledge about the similarity of the original labels (or attributes) or even whole substructures. Since such application-based differences may not be known or only partially known, one may try to use machine learning to find such database-specific distances. If this is possible, the learned distances could potentially reveal unknown application-based knowledge on the substructures of the used graphs.

In my masters thesis I am going to investigate, how well a measure for similarity can be learned, based on a similarity definition using the WL-labeling hierarchy. The structure of the similarity measure is a Wasserstein distance between the WL-label distribution (WL-features) of two given graphs. The ground distance of the Wasserstein distance is defined as a tree metric on the WL-labels. Thus it highly depends on the edge weights in the hierarchy tree. The research question is, if it is possible to learn how to adjust these edge weights, in order to yield a better similarity measure, compared to a static ground distance definition. The usage of the Wasserstein distance is motivated by Togninalli et al., who successfully used it to construct a similarity measure for graphs with both categorical vertex labels and continuous vertex attributes [23]. As they mention, optimal transport had been used successfully to tackle the graph alignment problem before [25].

2 Related work

XY proposed spectral distances as graph similarity measures, which focus on the Laplacian matrix eigenvectors [9, 6].

Maretic et al. proposed a framework for graph similarity measures based on the Wasserstein distance for comparing vector representations of entire graphs [16].

Many graph kernels can be categorized as $\mathcal{R}$-convolution kernels [23]. That is, they define similarity in terms of aggregation of local similarity. The local similarity is computed between mutual substructures such as walks [5, 11], paths [1], limited-size subgraphs like graphlets [8, 21], subtree patterns [19, 15, 22] or combinations of them. One important example for such a combination is the General WL kernel proposed by Nino Shervashidze et. al [22]. This kernel is based on the WL labels and can be combined with any other graph kernel (base kernel), since it simply sums up the base kernels results for every WL label iteration. The authors themselves present a subtree, edge and shortest path variation in their paper “Weisfeiler-Lehman Graph Kernels” [22].

Later work has moved away from solely counting equivalent substructures [20]. - Compute optimal assignment between vertices [14] - Compute optimal transport as a form of ’soft-matching’ between vertices [23] - Neighborhoods as Rooted DAG

3 Framework and definitions

Before describing the details of the proposed method, let me introduce some basic definitions. An (undirected) graph is a tuple $G = (V, E)$ where V is a finite set of vertices and $E \subseteq 2^V$ is a finite set of undirected edges on them. G is called labeled, if there exists a vertex labeling function $\ell : V \to \Sigma_V$ for a finite label alphabet Σ. The labels can be included in the definition of the graph by writing $G = (V, E, \ell)$. G is called weighted if there exists an edge-weight function $w : E \to \mathbb{R}$. Two graphs G_1 and G_2 are isomorphic ($G_1 \equiv G_2$), if there exists a bijective function between their vertices, that preserves all edges and labels. For a graph $G = (V, E)$ and a vertex $v \in V$ we call the set $\mathcal{N}^1(v) := \{u \in V|\ (u, v) \in E\}$ the first neighborhood of v in G. The degree $\delta(v)$ of a vertex $v \in V$ is given as the size of its first neighborhood, that is $\delta(v) := |\mathcal{N}(v)|$. Similarly we define the d-th neighborhood (sometimes referred to as d-hop neighborhood) of v in G (for $1 < d \in \mathbb{N}$) as the set of all vertices that are not v itself, that can be connected with v by a path of length d:

$$\mathcal{N}^d(v) := \left\{u \in V|\ \exists w \in \mathcal{N}^{d-1}(v) \text{ s.t. } (u, w) \in E\right\} \backslash \{v\}$$

Call $\mathcal{N}_+^d(v) := \mathcal{N}^d(v) \cup \{v\}$ the extended d-th neighborhood of v in G.

3.1 Weisfeiler-Lehman labeling scheme

The here used Weisfeiler-Lehman labeling scheme is vertex labeling scheme at the center of the Weisfeiler-Lehman test of isomorphism. More specifically I will make use of its 1-dimensional variant (also known as 1-WL vertex embedding, WL color refinement, naive vertex refinement or 2-WL as higher-dimensional WL labeling scheme) [22, 25]. The scheme iteratively propagates neighborhood information by compressing this information in labels and iteratively assigning these to the vertices [24]. We will call such assigned labels WL-labels. The general WL color refinement is described more formally in algorithm 1.

Algorithm 1 WL color refinement

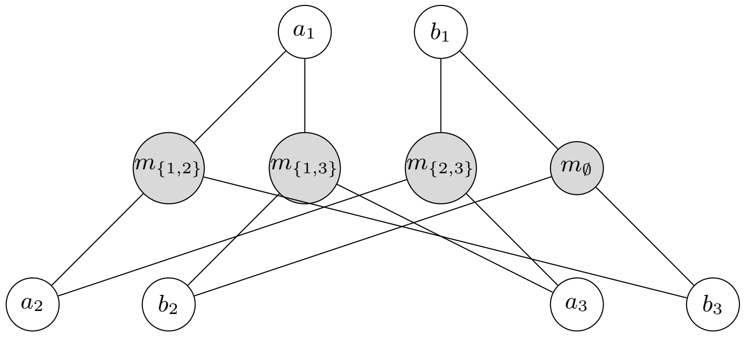
Input: a graph $G = (V, E, \ell)$,
a perfect hash function hash.
Output: a vector representation of G of size $|V|$.

```
1: $c(v)^0 \leftarrow \text{hash}(\ell)$ for all $v \in V$
2: while $\left(c^k(v)\right)_{v \in V} \neq \left(c^{k-1}(v)\right)_{v \in V}$ do
3:     $c^k(v) \leftarrow \text{hash}\left(c^{\ell-1}(v), \{\!\{c^{k-1}(w) |\ w \in \mathcal{N}_G(v)\}\!\}\right)$ for all $v \in V$
4: return $\{\!\{c^k(v) |\ v \in V\}\!\}$
```

Note that there are several possible implementations of the hash function and the concatenation step in line 3.

In algorithm 1 the color refinement is repeated until the representations do not change, that is until a stable configuration is found. This can be used as an imperfect graph isomorphism test. However, for our purposes, an intermediate compression level of the graph structure may be sufficient. Thus we add as input the desired number of labeling iterations k and execute the commands in the while-loop $k - 1$ times. The new set of WL-labels in the i-th iteration (V, c^i) are called the i-th WL-feature of the graph.

Note that after the first iteration, the WL labels encode the information of the labels of the first neighborhood (those reachable with paths of length one). In the d-th iteration, the WL labels encode the information of the labels of the d-th neighborhood, since they are constructed on previously defined WL labels, which in turn encode information about neighbors that are further away.



$X_3 = (V_3, E_3)$ with $V_3 = A_3 \cup B_3 \cup M_3$ and $A_3 = \{a_1, a_2, a_3\}$, $B_3 = \{b_1, b_2, b_3\}$, $M_3 = \{m_{1,2}, m_{1,3}, m_{2,3}, m_\emptyset\}$. E_3 contains edges $\{a_i, m_S\}$ and $\{b_j, m_S\}$ such that $i \in S$ and $j \neq S$ [3].

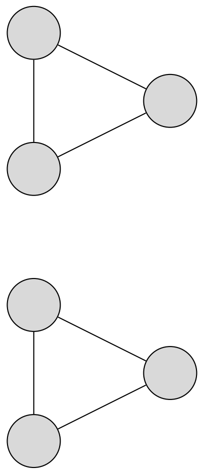


Fig. 1. G_1 - Two disconnected circles, each of length three.

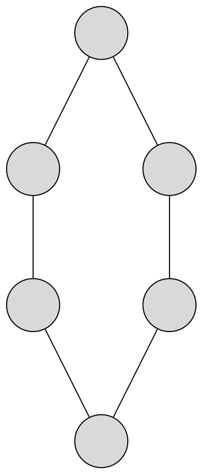


Fig. 2. G_2 - A circle of length six.

Consider two graphs G_1 and G_2 (as sketched in figures 1 and 2) on six vertices. G_1 consists of two disconnected circles each of length three. G_2 is a circle of length six. Given no initial labels, the WL labeling scheme considers all vertices equally labeled. Since every vertex in both graphs has exactly two neighbors (with the same label), every vertex will be assigned the same WL-label in the next iteration. The obtained labeling is equivalent to original case, and no number of WL-iterations will generate a different outcome. In this case, the WL-labeling scheme would wrongfully declare the two graphs to be isomorphic.

Remarks on the implementation When applied to labeled graphs (with a finite set of labels), the labeling scheme can be implemented using integers as WL-labels and with simple perfect hash functions. The method can be implemented as an efficient matrix vector multiplication between the adjacency matrix and the WL-label vector using logarithms of prime numbers as WL-labels.

Implementation as matrix vector multiplication The naive implementation of the WL-labeling scheme requires to aggregate the WL-labels in the neighborhood of each vertex, combine this aggregation with the WL-label of the vertex itself and hash it to obtain a new WL-label (with a perfect hash function). To obtain a maximally powerful embedding, the aggregation and hashing must be injective [25]. This aggregation is often realized by sorting the WL-labels of the neighborhood first. By design, the labeling scheme only requires information about the neighborhoods (adjacency matrix) and the WL-labels of the last iteration in order to compute the next WL-labels (containing information about a neighborhood or larger degree). This can intuitively be understood as a recursive matrix-vector multiplication. Given the fixed adjacency matrix

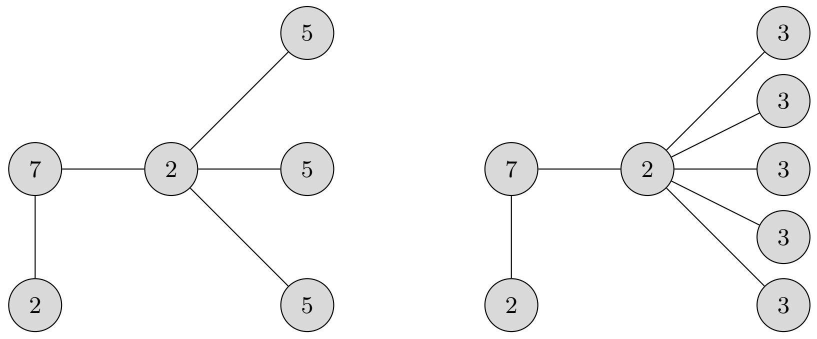
$A \in \{0,1\}^{|V|}$, the goal is to define WL-labels (WL-features) $w_i \in \mathbb{R}^{|V|}$ such that the multiplication Aw_i leads to the next WL-labels w_{i+1}. This way, the new WL-labels are computed purely as an addition of the old WL-labels. Using any numbers is not injective, since addiction is not an injective function. One may think of prime numbers for the WL-labels, since these are at least injective with respect to multiplication. To get from addiction to multiplication one can use the (injective) logarithm of prime numbers. Thus the proposed method goes as follows: Map the original n vertex labels (or attributes) to the logarithms of the first n prime numbers. With original labels given by $\ell_0 : V \to [n]$ and prime numbers $p_i \in \mathbb{P}$ this can be written as:

$$\forall v : \ell_0(v) \mapsto \log(p_{\ell_0(v)}) =: \ell_1(v)$$

Now, the 1-th WL-labels can be computed as

$$\ell_1'(\overleftarrow{v}) := \pi \ell_0(\overleftarrow{v}) + A\,\ell_0(\overleftarrow{v}) = (A + \pi \mathbb{1})\ell_0(\overleftarrow{v})$$

The factor π can be replaced by any transcendent number. It ensures that in the computation of the new label for a vertex v the old label is differentiated from the labels of the neighborhood. Consider for example a vertex with WL-label 2 and two neighboring vertices with labels 3. If the old labels are not differentiated from the labels of the neighborhood, the vertex with label 2 in this example would obtain the same label as a vertex with label 3 and two neighboring vertices with labels 2 and 3. To ensure injectivity of the multiplication, ℓ_1' needs to return prime numbers. With each iteration the used label values are getting bigger. This their logarithms may have less and less difference, which could lead to practical problems due to the limited machine precision. Thus repeat the mapping step between the matrix-vector multiplications to use the logarithm of a prime number for each new WL-label.



The adjacency matrices below list the vertices in the order from bottom left to top right (with labels 2, 7, 2 and then the vertices with label 5 or 3).

$$\begin{bmatrix} 1 & 1 & . & . & . & . \\ 1 & 1 & 1 & . & . & . \\ . & 1 & 1 & 1 & 1 & 1 \\ . & . & 1 & 1 & . & . \\ . & . & 1 & . & 1 & . \\ . & . & 1 & . & . & 1 \end{bmatrix} \log \begin{bmatrix} 2 \\ 7 \\ 2 \\ 5 \\ 5 \\ 5 \end{bmatrix} = \begin{bmatrix} \log(2)+\log(7) \\ \log(7)+\log(2)+\log(2) \\ \log(2)+\log(7)+\log(5)+\log(5)+\log(5) \\ \log(5)+\log(2) \\ \log(5)+\log(2) \\ \log(5)+\log(2) \end{bmatrix}$$

$$= \begin{bmatrix} \log(2*7) \\ \log(7*2^2) \\ \log(2*7*5^3) \\ \log(5*2) \\ \log(5*2) \\ \log(5*2) \end{bmatrix} \quad \begin{matrix} 2|7 \\ 7|2,2 \\ 2|7,5,5,5 \\ 5|2 \\ 5|2 \\ 5|2 \end{matrix}$$

$$\begin{bmatrix} 1 & 1 & . & . & . & . & . & . \\ 1 & 1 & 1 & . & . & . & . & . \\ . & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ . & . & 1 & 1 & . & . & . & . \\ . & . & 1 & . & 1 & . & . & . \\ . & . & 1 & . & . & 1 & . & . \\ . & . & 1 & . & . & . & 1 & . \\ . & . & 1 & . & . & . & . & 1 \end{bmatrix} \log \begin{bmatrix} 2 \\ 7 \\ 2 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \log(2*7) \\ \log(7*2^2) \\ \log(2*7*3^5) \\ \log(3*2) \\ \log(3*2) \\ \log(3*2) \\ \log(3*2) \\ \log(3*2) \end{bmatrix} \quad \begin{matrix} 2|7 \\ 7|2,2 \\ 2|7,3,3,3,3,3 \\ 3|2 \\ 3|2 \\ 3|2 \\ 3|2 \\ 3|2 \end{matrix}$$

Algorithm 2 Prime WL-labeling scheme

Input: a graph $G = (V, E)$,
WL labeling depth k.
Output: WL-labels on G for k iterations.

```
1: $\ell_0(\overleftarrow{v}) = \text{enumerate}(\ell(\overleftarrow{v}))$          ▷ Enumerate the original labels
2: $\forall v \in V:\ \ell_0(v) = \log(p_{\ell_0(v)})$          ▷ Map the enumerated labels to primes
3: for $i = 0$ to $k$ do
4:     $\ell' = (A + \pi\mathbb{1})\ell_0(\overleftarrow{v})$          ▷ Aggregate own and neighborhood label
5:     $\forall v \in V:\ \ell_{i+1}(v) = \log(p_{\ell'})$          ▷ Map to primes
6: return $\{\ell_i |\ i \in [k]\}$
```

Notes:

- For m graphs of at most n vertices, at most the first $m * n$ prime numbers are required.
- Note that it is not necessary to use the first primes or to maintain any ordering when mapping to the primes. To use the first (small) primes has only the benefit to keep the numbers low.

3.2 Weisfeiler-Lehman labeling tree (WLLT)

Given k iterations of WL-label refinements on a database $\mathcal{D}$ of graphs, one can construct a WL-labeling tree (WLLT) of the dependencies or inheritance of the WL labels for this database. Labeled and un-labeled graphs can be treated in the same way, by considering vertex labels as the zeroth WL-labels. With this in mind, use an artificial label to initialize the root of the WLLT. The i-th layer of WLLT vertices consists of the arising WL-labels in the i-th iteration of the label refinement. That is, there exists an edge (l, l') in the tree between vertices l and l' if there exists a vertex $v \in V(G)$ in some graph $G \in \mathcal{D}$ and $k \in \mathbb{N}$ with $l = c^k(v)$ and $l' = c^{k+1}(v)$. The complete construction of the WLLT is sketched in algorithm 3.

Algorithm 3 WLLT construction

```
Input:  a graph dataset $\mathcal{G}$,
        WL labeling depth $k$.
Output: WLLT $T$ of height $k$ (unweighted).

1: Initialize the WLLT as a single root $T = (r, \emptyset)$
2: for $G$ in $\mathcal{G}$ do
3:     for $i = 0$ to $k$ do
4:         Generate the $i$-th WL-feature $F$ of $G$
5:         for $l \in F$ do
6:             if $l \notin T$ then
7:                 $V(T) = V(T) \cup \{l\}$
8:                 $E(T) = E(T) \cup (p, l)$ where $c^{i-1}(v) = p$ and $c^i(v) = l$
9: return $T$
```

Remarks on the implementation Note that it is efficient to construct the WLLT while defining the WL-labels. For every iteration of WL labeling, a new layer is added to the WLLT.

Adding edge weights There are several possibilities to compare two given WL-labels, that arise from the same parent WL-label and to define a distance between them. In order to extend such a comparison to any two WL-labels (of the same WL labeling iteration) one could use a French Railway Metric (FRM) to define edge weights in the WLLT. These can be used as a tree metric (shortest-path distance), by adding the weights along the (unique) path between two vertices. While this seems like a reasonable choice, incorporating any knowledge of the application is difficult. This would require to know, how different WL-labels relate to one-another. Which in turn requires knowledge on the relation of both, single vertex attributes and the attributes of substructures of the graph (which are encoded by WL-labels of later iterations).

Thus instead, one may use machine learning to adjust the edge weights in a desirable way. It is one of the research questions of my thesis, whether this can be done. If so, the learned weights may reveal, possibly unknown, application-based knowledge.

From any given edge weights the tree metric arises naturally. The distance between two vertices in the tree is defined as the sum of all edge weights in the path that connects the two vertices.

3.3 Optimal transport - Wasserstein distance

Optimal transport (OT) was introduced by Monge [17] and reformulated by Kantorovich [10]. It has been applied in image processing, data analysis and machine learning [18].

In the proposed method, the Wasserstein distance will be used as graph similarity measure on vector representations of the WL-features of graphs. A suitable definition goes as follows. For two vectors $x \in \mathbb{R}^n$ and $x' \in \mathbb{R}^{n'}$ the (discrete) L^1-Wasserstein distance (also known as earth mover distance) is defined as:

$$\mathcal{W}_M(x, x') := \min_{T \in \mathcal{T}(x,x')} \langle T, C \rangle \tag{1}$$

$C \in \mathbb{R}^{n \times n'}$ is a cost or distance matrix containing $d(x, x')$ for some ground distance d. $T \in \Gamma$ is often called transport matrix or joint probability. It is $\mathcal{T} \subseteq \mathbb{R}^{n \times n'}$. For $T \in \mathcal{T}$ it is $T\mathbb{1}_{n'} = X$ and $T\mathbb{1}_n = X'$. That is, T contains fractions that indicate how to map (transport) the values from x to x' with minimal total cost (transport effort), with respect to the costs in C. $\langle \cdot, \cdot \rangle$ is the Frobenius inner product. $\mathcal{W}_C(x, x')$ itself is called optimal transport matrix. Note that if d is a metric, so is the $\mathcal{W}_C$.

In the context of the proposed research, C will contain the distances between different WL-labels, given by the tree-metric d.

Ground metric The actual values of the Wasserstein distance depends on the definition of the ground metric d which is used to define the cost matrix M. Togninalli et al. used the normalized Hamming distance for categorical vertex features and the Euclidean distance for continuous vertex features [23]. A key difference between their approach and the one I am going to investigate in this thesis, is to iteratively adapt the used ground distance. More precisely, an initial ground distance will be defined using an initial tree metric on the WL-labeling hierarchy tree. Next, graphs with known classification will be used to train the model in order to adapt the tree metric as needed. This process will be denoted as edge weight learning. It is expected, that this will help to determine the impact of the ground metric on the approach proposed by Togninalli et al. and if it varies depending on the dataset.

3.4 Supervised learning

The evaluation of the proposed method is tied to the supervision of the learning process. The supervision is given by a feedback loop, which evaluates the accuracy of the current similarity measure, defined by the current set of edge weights in the WLLT. The accuracy of the similarity measure can be evaluated in several different ways, depending on the given dataset. Overall the similarity measure will be used to classify the given graphs. Thus its performance is measured proportional to the performance of the classification. The goal of the learning is to increase the performance of the similarity measure. From this, several degrees of success can be derived. For example:

1. Is it possible to reliably improve the performance of the similarity measure?
2. Is it possible to learn a similarity measure, which performs better than the one used in the Wasserstein Weisfeiler-Lehman Graph Kernels? That is, it allows for a more accurate classification. Or a faster classification which is at least as accurate as the one given by the WWL kernels.
3. Is it possible to learn a similarity measure, which performs better than some other kernels?

This kind of evaluation will be used to guide the research since it aligns with the research questions. The evaluation of the similarity measure itself will at least be done analogously to the Wasserstein Weisfeiler-Lehman Graph Kernel by Togninalli et al. That is to use the similarity measure to construct a Laplacian kernel and evaluate its performance using an support vector machine [23]. Besides this, other evaluations of the similarity measures may be considered.

One approach of evaluating the similarity measure is to compare its clustering capabilities to some kind of classification clas(G) for any given graph G. The learning of the ground distance can be derived as sketched in algorithm 4. However, the precise update mechanism may change and several approaches may be investigated. For example by comparing the quality of the clusters derived from the learned distance to other clustering methods.

Algorithm 4 Ground distance learning

```
Input:  a graph dataset $\mathcal{G}$,
        learning rate $\lambda$,
        number of learning epochs $k$.
Output: Distance matrix $D$ on $\mathcal{G}$.

1: for $i = 0$ to $k$ do
2:     for $G_1$ and $G_2$ in $\mathcal{G}$ do
3:         $C = \mathrm{FRM}_T(G_1, G_2)$                          ▷ Ground distance computation
4:         $i, j = \mathrm{argmax}\left(\min_{T \in \mathcal{T}} \langle T, C \rangle\right)$     ▷ Wasserstein distance computation
5:         $\Delta w(P_{i,j}) = \lambda \left| c(G_1) - c(G_2) \right|$        ▷ Evaluation
6:         $w(P_{i,j}) = w(P_{i,j}) + \Delta w(P_{i,j})$              ▷ Distance update
7: return $D$ with $D_{i,j} = \min_{P \in \Gamma}\left(P,\ \mathrm{FRM}_T(G_i, G_j)\right)$   ▷ Distance matrix on $\mathcal{G}$
```

Since a successful (with respect to the research goal) implementation of the ground distance learning is yet unknown, the pseudo-code given in algorithm 4 shall give a first implementation goal. The final procedure may be different in several aspects.

There are three lines of code, which may nor may not be changed from the following initial interpretation.

First, in line 4 it is denoted that the weight learning will target only the two weights which are linked to the highest cost in the optimal transport solution. It may be suitable to adjust all weights according to the costs of their mapping.

Second, line 6 aims to increase the distance between two graphs if their (binary) classes are distinct (line 5). However, it may also be suitable to decrease their distance, if their classes are equal. And, a more complex update procedure may be needed, if more that two classes are present. Furthermore there are several ways to in- or decrease their distance. Before investigating methods like for example gradient descent, a fixed distance increment shall be used.

Third, line 6 denotes the edge weight update such that the tree distance (path weight) between the two given labels is changed. In general, this cannot be done, such that distances between other labels are not changed. But it can be done by changing all edge weights for all edges on the path (influencing a lot of other distances). Or by changing only the first and last edge weights (influencing only the distances concerning these two labels). The latter mechanism shall be investigated (first).

4 Research goal

Variations of the described method shall be implemented with the goal to answer the following research questions:

1. Is it possible to reliably increase the predictive power (accuracy) of the similarity measure?
2. Is it possible to learn a similarity measure, which performs better than the one used in the Wasserstein Weisfeiler-Lehman Graph Kernels?

If these questions can be answered positively, several further questions arise naturally and shall be investigated as well:

1. Is it possible to learn a similarity measure, which performs better than some other kernels? Which?
2. Are there example of graph dataset, where the method does not yield in a significant improvement? Can this be explained with application-based knowledge?
3. How adaptable is the usage of the tree-metric as ground distance? Is it possible to learn other distance functions?
4. How does the learning approach perform compared to
   – constant edge weights,
   – different static edge weight initialization (using the Hamming distance, Jaccard distance or others),
   – k-means classification.

4.1 Roadmap

Since the goal is to define and implement a learning method, a training set is needed. By definition, it is possible to construct the training set once and use it for many different configurations of the learning method. It can be expected, that the performance of the similarity measure changes over time with the edge weight updates. This learning or training process shall be monitored in order to elaborate on the success and efficiency of the proposed method. At last, a few implementation choices are not defined jet and it may be beneficial to change them. To what extent the method can be changed successfully shall be investigated as well. These three sub tasks may change, as long as answers to the overall research questions are generated.

Training set Since the learning of the similarity measure will be limited to changing edge weights in the WLLT, the training set consists of weighted WLLTs and the WL-features of the corresponding dataset. Both can be computed independently and in advance of the actual training. The construction of training sets includes constructing (possibly several) WLLTs of classified graph datasets and WL-features. For computational efficiency, they shall be stored in a file format, including the map of WL-features to the original labels and the WL-features of all graphs.

TODO: A few words on the datasets. Detailed structural properties of all dataset in Appendix ...

TU Dortmund Bio, Chemic, Networks
Subgraphs of online networks:

- IMDB-BINARY: Collaboration networks between actors and actresses, each annotated against movie genres.
- REDDIT-BINARY: User interactions in discussion forums, annotated against the type of forum.

EGONETS-x is a collection of ego network graphs which are extracted from four different social networks. An egonet is a network of instances (people), that are affected by one single instance (user) of a social network.With growing index x, the ego networks get larger and more dense. In this context, ego networks are subgraphs induced by the neighbors of a vertex. It is ensured, that simple count of vertices and edges is not sufficient for prediction tasks. The learning task is to assign an egonet to the network it was extracted from.

OGB

–
–

Training The training consists of a supervised feedback loop. In each epoch, the distance between pairs of graphs (of a subset) from the graph dataset is computed. If the graphs have a different classification (supervised knowledge), the distance according to the Wasserstein distance shall be increased. This is done by identifying the most significant WL-label (highest WL-label distance) with respect to the tree-metric in optimal assignment given by the Wasserstein distance. The edge weights along the corresponding path in the WLLT are increased by a finite amount λ. If the graphs have the same classification, the weights are decreased.

Although not specified at this point, it may be desirable to use other kinds of feedback as well. This also will depend on the given datasets and knowledge on them.

Furthermore, the specifics of the target weights of the update and the definition of the update (λ) itself may vary.

Variations Different kinds of feedback loops may require different implementation choices with respect to the definition of the tree-metric, Wasserstein distance, or the update choices. While different choices may be investigated in order to answer the research question properly, a comparison between these choices is desired as well. If possible, such variations shall be set in relation to the different datasets and even the different resulting metrics on the WL-labels.

References

[1] Karsten Borgwardt and Hans-Peter Kriegel. “Shortest-Path Kernels on Graphs”. In: *Fifth IEEE International Conference on Data Mining (ICDM’05)*. IEEE, 2005. DOI: 10.1109/icdm.2005.132.

[2] H. Bunke and G. Allermann. “Inexact graph matching for structural pattern recognition”. In: *Pattern Recognition Letters* 1.4 (Mar. 1983), pp. 245–253. DOI: 10.1016/0167-8655(83)90033-8.

[3] J.-Y. Cai, M. Furer, and N. Immerman. “An optimal lower bound on the number of variables for graph identification”. In: *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1992. DOI: 10.1109/sfcs.1989.63543.

[4] Holger Fröhlich et al. “Optimal assignment kernels for attributed molecular graphs”. In: *Proceedings of the 22nd international conference on Machine learning - ICML ’05*. ACM Press, 2005. DOI: 10.1145/1102351.1102380.

[5] Thomas Gärtner, Peter Flach, and Stefan Wrobel. “On Graph Kernels: Hardness Results and Efficient Alternatives”. In: *Learning Theory and Kernel Machines*. Springer Berlin Heidelberg, 2003, pp. 129–143. DOI: 10.1007/978-3-540-45167-9_11.

[6] Ralucca Gera et al. “Identifying network structure similarity using spectral graph theory”. In: *Applied network science* 3.1 (2018), pp. 1–15.

[7] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. “Kernel methods in machine learning”. In: *The Annals of Statistics* 36.3 (June 2008). DOI: 10.1214/009053607000000677.

[8] Tamás Horváth, Thomas G?rtner, and Stefan Wrobel. “Cyclic pattern kernels for predictive graph mining”. In: *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’04*. ACM Press, 2004. DOI: 10.1145/1014052.1014072.

[9] Irena Jovanović and Zoran Stanić. “Spectral distances of graphs”. In: *Linear Algebra and its Applications* 436.5 (Mar. 2012), pp. 1425–1435. DOI: 10.1016/j.laa.2011.08.019.

[10] Leonid Vitaliyevich Kantorovich. “On the transfer of masses”. In: *Doklady Akademii nauk USSR* (1942), pp. 227–229.

[11] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. “Marginalized Kernels Between Labeled Graphs”. In: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. Ed. by Tom Fawcett and Nina Mishra. AAAI Press, 2003, pp. 321–328.

[12] Risi Kondor and Karsten M. Borgwardt. “The skew spectrum of graphs”. In: *Proceedings of the 25th international conference on Machine learning - ICML ’08*. ACM Press, 2008. DOI: 10.1145/1390156.1390219.

[13] Risi Kondor, Nino Shervashidze, and Karsten M. Borgwardt. “The graphlet spectrum”. In: *Proceedings of the 26th Annual International Conference on Machine Learning - ICML ’09*. ACM Press, 2009. DOI: 10.1145/1553374.1553443.

[14] Nils M. Kriege et al. “On Valid Optimal Assignment Kernels and Applications to Graph Classification”. In: 2016.

[15] Pierre Mahé and Jean-Philippe Vert. “Graph kernels based on tree patterns for molecules”. In: *Machine Learning* 75.1 (Oct. 2008), pp. 3–35. DOI: 10.1007/s10994-008-5086-2.

[16] Hermina Petric Maretic et al. “GOT: An Optimal Transport framework for Graph comparison”. In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[17] Gaspard Monge. “Mémoire sur la théorie des déblais et des remblais”. In: *Histoire de l’Académie Royale des Sciences de Paris* (1781).

[18] Gabriel Peyré and Marco Cuturi. “Computational Optimal Transport: With Applications to Data Science”. In: *Foundations and Trends® in Machine Learning* 11.5-6 (2019), pp. 355–607. DOI: 10.1561/2200000073.

[19] Jan Ramon and Thomas Gartner. “Expressivity versus Efficienvy of Graph Kernels”. In: *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*. 2003.

[20] Till Hendrik Schulz et al. “A Generalized Weisfeiler-Lehman Graph Kernel”. In: 2021.

[21] Nino Shervashidze and Karsten M. Borgwardt. “Fast subtree kernels on graphs”. In: *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*. Ed. by Yoshua Bengio et al. Curran Associates, Inc., 2009, pp. 1660–1668.

[22] Nino Shervashidze et al. “Weisfeiler-Lehman Graph and Kernels”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 2539–2561.

[23] Matteo Togninalli et al. “Wasserstein Weisfeiler-Lehman Graph Kernels”. In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[24] Weisfeiler and Leman. “THE REDUCTION OF A GRAPH TO CANONICAL FORM AND THEALGEBRA WHICH APPEARS THEREIN”. In: *Journal of Applied Mathematics and Physics*. 1968.

[25] Keyulu Xu et al. “HOW POWERFULARE GRAPH NEURAL NETWORKS?” In: *International Conference on Learning Representations*. 2019.