Learning Graph Similarities Using The Weisfeiler-Leman Label Hierarchy.

Fabrice Beaumont

Born on the 17th of September 1996 in Bonn

25th November 2022

Master's Thesis Computer Science

Advisor: Dr. Pascal Welke

Second Advisor: Prof. Dr. Christian Bauckhage

Department of Information Systems and Artificial Intelligence

Faculty of Mathematics and Natural Sciences of the

Rhenish Friedrich Wilhelm University of Bonn

Contents

1 Introduction

Digitization is steadily increasing the amount of digitally stored structured data. Common examples of such are images [1]–[4], natural language texts [5], semi-structured data such as HTML and XML [6], 3d shapes [7], social and sensor networks, program flows, and structures in chemo- and bioinformatics (e.g. molecules, proteins, genes and chemical compounds) [8]–[10]. Based on functional properties of the represented structures, one may define classifications on the representing graphs. For example, such classification can indicate the mutagenic effect of chemical compounds on a bacterium [11].

To extend the classification of known graph instances to unknown instances is one of many different challenges when working with such graph data. In this framework it is usually assumed that graphs with similar structure represent structures with similar properties. Therefore one assumes that such graphs with are classified similarly too. Thus, it is important to quantify the structural similarity between graphs [10]. 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. This solves the so called graph isomorphism problem, which is in NP [12]. That is, no polynomial algorithm for the graph isomorphism is known and it has been conjectured that no such algorithm can exist [13]. Furthermore, the task of testing, weather for two given graphs one is isomorphic to a subgraph of the other one, is NP-complete [12]. The seemingly reasonable approach to restrict the isomorphism test to a unique subgraph, like the largest common subgraph, is of limited use, since finding such largest common subgraphs is a NP-complete problem as well [12].

There are several different approaches to defining a more sophisticated similarity measure. To give a few examples, researchers have based similarity measures on the graph edit distance [9], [14], optimal assignment kernels [9], the skew spectrum [15] and the graphlet spectrum [16], [17]. A common approach to define graph similarity measures, is to define a kernel function on graph representations. Such a graph kernel have the desirable property of enabling the application of kernelized learning methods on graph-structured data. Most prominently a reduced computational cost when invoking the kernel trick. Furthermore, by restricting the kernels to substructures of graphs, they sometimes can be computed in polynomial time [18], [19]. This restriction however may at the same time limit their ability to capture complex characteristics of the graphs. For example the WL Subtree Kernel uses special vertex labels, so-called Weisfeiler-Leman labels, which recursively summarize information on the neighborhoods in the graph. Given WL-labels for a fixed number of iterations (recursions),

the WL Subtree Kernel counts the WL-labels both graphs have in common [10] (proposed in 2011). This counting procedure is a simplified aggregation of information, and discards both the information on what WL-labels the two graphs have in common, and all information on WL-labels, which the two graphs do not have in common. The information loss due to simplified aggregations like these may be the reason, why most proposed variants do not generalize well to graphs with high-dimensional continuous vertex attributes [10]. Generally speaking, the trade-off between the expressiveness of graph representations and the computational effort of their comparison, guides the research in this area. In 2016, Kriege, Giscard, Computer, et al. proposed their WL-Optimal Assignment Kernel (WL-OA), which opted for defining a bijection between vertices such that as many iterations of equal WL-labels as possible were preserved. Three years later, in 2019, Togninalli, Ghisu, Llinares-López, et al. established their Wasserstein WL Graph Kernels as the new state of the art. Unlike before, two graphs are not compared by counting common WL-labels between them. Instead, the graphs are represented as vectors of WL labels, which are compared by using a Wasserstein Distance. The Wasserstein Distance in turn is defined based on a ground distance between the WL labels themselves. For categorical original vertex labels, the authors propose to use the normalized Hamming distance for this ground distance. Similarly to the WL-OA, using the Wasserstein Distance can be seen as an assignment between the graphs as well.

However, these rigid ground metrics (e.g. the normalized Hamming distance) do not permit to adjust the distances between the WL-labels (or attributes) used in the graphs. Such adjustment may be advantageous, since the WL-labels represent entire substructures (also known as unfolding trees [22]), whose importance may vary, depending on the classification task. We argue that it is reasonable to assume that the same set of graphs (e.g., a set of molecules), can be classified differently according to different structural properties. But using the same ground distance, yields the same distance measurements for the graphs, independent of their classification. That is, the proposed Wasserstein WL Graph Kernel does allow to compare substructures between graphs, but does not distinguish between their importance with respect to the classification task at hand.

In this thesis, I present research on an approach to iteratively change and adapt the used ground distance. This allows to introduce application specific knowledge about the similarity of the original vertex labels (or attributes) or even whole substructures (unfolding trees). Since this knowledge may not be known or only partially known, I will use machine learning to learn these similarities. If such a learning procedure succeeds, the learned similarities on the WL-labels may reveal application based knowledge, linked to the given classifications.

Outline of the Thesis The remaining document is structured as follows. This chapter continues with summarizing the idea of the researched method, and the research question. Next, related work is presented, the research plan sketched, and the results are briefly summarized. The next chapter contains theoretical background on the implemented method (section 2) and includes the most necessary definitions and theorems to understand the proposed method and its evaluation. The chapter after this, presents the implemented method in detail (section 3) and its evaluation (subsection 3.5). Finally, the experiments on the implemented method are presented and discussed (section 4). At last, a conclusion is given, to review and answer the research questions, and sketch an outlook for future work (section **??**).

1.1 Problem Statement and Method

The goal of this thesis is to define a similarity measure between graphs. The approach is based on the Wasserstein Weisfeiler-Leman (WWL) Graph Kernel proposed by Togninalli, Ghisu, Llinares-López, et al. [21]. The key difference is, to not use a fixed ground distance in the definition of the Wasserstein Distance used in the WWL Kernel, but to use machine learning to learn a more favorable ground distance. To do so, the Wasserstein Distance is formulated as Tree-sliced Wasserstein Distance, and its ground distance as tree-metric. The tree metric in turn is based on edge weights, assigned to the hierarchy tree induced by WL-labels. The update rule in the learning process aims to decrease the distance between two graph samples, if they belong to the same class, and increase it otherwise. The distance deand increase is realized by changing the edge weights for the tree-metric. The precise definition of the learning process shall be developed during the research. In this thesis, we will refer to the described method as **learned Wasserstein Weisfeiler Leman labeling tree** (**l-WWLLT**) method and to the implied similarity measure and graph kernel as **l-WWLLT distance** and **l-WWLLT Kernel** accordingly.

1.2 Research Question

As mentioned, the goal is to implement a learning procedure, which improves the implied graph distances. This improvement is evaluated in two ways. First, by using clustering metrics to evaluate the relation between the given classification and the l-WWLLT distances. And secondly, by evaluating the accuracy of a Support Vector Machine (SVM) with respect to the l-WWLLT Kernel. An improvement is indicated by improving the clustering metrics with respect to their definition of better clustering or by improving the accuracy of the SVM.

The research question of this thesis is to decide, if the l-WWLLT method can improve the resulting graph similarity measure over its definition at the initialization. Furthermore, the flexibility of the graph similarity measure shall be investigated by evaluating several different implementation and parameter choices.

1.3 Related Work

This section contains other definitions of graph similarity measures, which relate to the proposed l-WWLLT method. The section is structured mostly in chronological order. Early studies use almost exclusively vector-based descriptions of the graphs [23]. Later, the similarities are defined on simply counting equivalent substructures. Over time, research shifted to more complex comparisons such as optimal assignment functions.

Early Graph Kernels In 1999, Haussler introduced a more sophisticated way of designing kernels on structured data by proposing so-called $\mathcal{R}$-convolution kernels, where $\mathcal{R}$ indicates a relation between graphs and its substructures [18], [21], [24]. Convolution kernels generalize the class of radial basis and simple exponential kernels, and they are well suited to represent joint probability distributions on pairs of substructures [18]. With respect to graph kernels, this approach translates to decomposing a graph into substructures and defining the kernel value as a combination (for example sums or averages) of similarities defined on these substructure. Many graph kernels can be categorized as such convolution kernels and use substructures like for example walks [24], [25], paths [23], graphlets [17], [26], subtree patterns [27], [28] or combinations of them [10]. Convolution kernels are characterized by defining the similarity as a summation of local substructures similarity.

In 2003, Thomas Gärtner, Peter Flach and Stefan Wrobel showed, that computing any kernel, that is capable of fully recognizing the structure of graphs (i.e. solving the graph isomorphism problem) is NP-hard [25]. Such kernels are also referred to as complete graph kernels [27]. There are however polynomial time algorithms to decide the isomorphism problem for several restricted graph classes, for example planar graphs [29]. Thus for an efficient (incomplete) graph kernel, based on the substructures of graphs, it can be expected that some amount of structural information may be discarded.

The first attempts in the quest of finding an efficient and yet expressive graph kernel, are based on strings and trees, transducers, dynamical systems [24], [25], [30], (random) walks (2002 and 2003) [24], [31][1], frequent

[1]Walk kernels usually count the number of matching walks of fixed length. These can be efficiently computed as the direct product of adjacency matrices.

subgraphs (FSG, 2004) [32], and cyclic patterns (2003 and 2004) [26], [27]. Still, some of these substructures are computationally expensive or even NP-hard to compute. For example, the computation of general cycles is NP-hard, and a kernel based on cyclic patterns may only be efficiently applicable to graphs with a fixed number of cycles of limited length [23]. On the other hand, if the substructure is to simple, their expressiveness may be to low. For example, with an increasing walk length in Random Walk Kernels one may find that classification accuracy decreases [23]. The kernel proposed in 2003 by Ramon and Gartner is based on subtree patterns and trades an expensive computation against improvements over walk based kernels [27]. There also exist variations, adapted to interpolate between different subtree patterns [28]. In 2005, Borgwardt and Kriegel proposed a shortest-path kernel which improved over all walk kernels [23]. In addition to that, the kernel has the desired qualities, that is computable in polynomial time (using Dijkstra's or Floyd-Warshall's algorithm), it is positive definite, and it is applicable to a wide range of graphs.

So far, the mentioned graph kernels scale at least qubically in the number over graph vertices [33]. To overcome this drawback and introduce a graph kernel, which is more efficiently applicable to bigger graphs, Shervashidze, Vishwanathan, Petri, et al. proposed in 2009 a Graphlet Kernel, which counts pre-defined subgraphs of limited size. The classification accuracies of their kernel is comparable to the previous state-of-the art kernels like the shortest path kernel. Significant is the reduced runtime on slightly larger datasets [33].

Weisfeiler-Leman Graph Kernels However, the success of all these approaches (in terms of SVM classification accuracy) was overshadowed by the Weisfeiler-Leman Graph Kernel proposed by Shervashidze, Schweitzer, Leeuwen, et al. in 2011 [10]. This kernel is based on the 1-dimensional WL-labeling scheme and it can be combined with any other graph kernels (base kernels), since it simply sums up their results over the graphs, labeled WL-labels for every WL iteration. By definition of the WL-labeling scheme, the kernel can be interpreted as a specific kind of subtree pattern kernel. The authors themselves present a subtree, edge and shortest path variation of the WL Kernel in their paper "Weisfeiler-Leman Graph Kernels" [10]. For example, the Weisfeiler-Leman Subtree Kernel counts the WL-labels two graphs have in common in the first d Wl labeling iterations Considering the normalized histogram of all WL-labels up to iteration d for a graph allows to compute this WL Subtree Kernel by taking the dot product of such feature vectors. The proposed WL Subtree, WL Edge and WL Shortest Path Kernels improved in terms of classification accuracy on the datasets *NCI1*, *NCI109*, and *ENZYMES*[2] over the mentioned Walk Kernel [27], the

[2]These datasets are from the TUDatasets [34], [35]

Random Walk and p-Random Walk Kernel [24], [36], the Graphlet Count Kernel [33], and over the Shortest Path Kernel [23]. For the dataset *DD* (or *D&D*)[2] their accuracies were comparable to the Graphlet Count Kernel but still improved over all other kernels. In terms of runtime, the experiments with the three WL Kernels were competitive to the other kernels on the datasets of smaller graphs. But especially the WL Subtree Kernel greatly outperforms the other kernels on datasets of graphs with thousands of vertices. On the dataset DD^2, subtree-patterns of height up to ten vertices were computed in eleven minutes, while all other kernels required at least half an hour and in extreme cases over one year of runtime [10]. These results motivated the usage of datasets with larger graphs and continuous or high-dimensional vertex attributes.

Optimal-Assignment Graph Kernels As mentioned, most of the graph kernels up to this point in time were convolution kernels. While convolution kernels add up pairwise similarities of substructures, assignment kernels on the other hand define bijection between such substructures (and thereby solving assignment problem). This can provide a more valid notion of similarity but also often yields indefinite functions, which complicates their use in kernel methods. The approach has been successfully applied to for example general graph matching [37], [38] and kernel-based classification [7], [9], [39]. In 2016, Nils M. Kriege, Pierre-Louis Giscard, and Richard C. Wilson improved over the WL Kernel by defining an assignment kernel, which they named Weisfeiler-Leman Optimal Assignment Kernel (WL-OA) [20]. This kernel defines a bijection between as many equal WL-labels for all vertices between two graphs as possible [20].Given the associated hierarchy on which the WL-labels are based, this kernel can be computed in linear time. They also presented optimal assignment kernels based on vertices or edge matchings only.

Wasserstein Weisfeiler Leman Graph Kernels As mentioned in the introduction, the Wasserstein WL (WWL) Graph Kernels were established as the new state of the art in 2019 [21]. These kernels are computed using the Laplacian kernel on a graph similarity measure which is based on the WL-labeling scheme and the Wasserstein Distance. More precisely, the graph similarity is computed using a discrete $\mathcal{L}_1$-Wasserstein distance (Graph Wasserstein Distance) on a graph embedding scheme which concatenates the WL-labels for all vertices in a graph for a fixed number of iterations. As ground distance for the Wasserstein Distance the Hamming distance is used (in the case of categorical vertex labels). The WWL Kernel performs comparable to the WL-OA Kernel on categorical vertex labels. However it generalizes well to attributed vertex labels and is thus better applicable to more complex datasets. One may consider the WL-OA (op-

timal assignment) as a "hard-matching" and the WWL (optimal transport) as a "soft-matching" between vertices.

A baseline for Graph Kernels At least until 2020, many researchers[3] used graph datasets, which were summarized in the graph dataset collection **TUDatasets** [34], [35]. However, Schulz and Welke showed in 2019, that many of these datasets may be insufficient in their role as benchmark datasets when it comes to the task of measuring the ability to capture and compare graph structures [44]. As a comparison, they defined a simple baseline graph kernel, which ignores graph structures completely. Their so-called No-graph (NoG) Kernel simply accumulates all vertex and edge features and ignores the edge definitions themselves, thereby ignoring the graph structure. They compared the performance of the NoG Kernel against the Probabilistic Frequent Subtree Kernel [45], the Frequent Subgraph Kernel [32], the Cyclic Pattern Kernel [26], the Graphlet Sampling Kernel [33], the Shortest Path Kernel [23], the Random Walk Kernel [25], and against the WL Kernels [10]. Their NoG Kernel outperformed almost all kernels on almost all previously used datasets, such as *BZR*, *ENZYMES*, *MSRC_21*, and *REDDIT-BINARY*[2]. Only the WL Kernel, performed better than the NoG in more cases than not. But it did so still on only few datasets like *NCI1*, *NCI109*, and *SYNTHETICnew*[2]. Since the goal of graph kernels is to capture the graph structures and not just the itemsets of their vertex and edge attributes, one can consider the No-graph Kernel as baseline kernel.

Mentioning of other strategies It should be mentioned that there exist other attempts to improve the tools for graph learning, besides graph kernels. For example Graph Neural Networks (GNNs) [43], [46], [47]. The WL-labeling scheme can be seen as a GNN, which iteratively propagates vertex states until an equilibrium is found. But there exists more research on different kinds of GNNs, like for example Graph Attention Networks by Veličković, Cucurull, Casanova, et al. These generalize recursive neural networks to deal with general classes of graphs such as cyclic directed and undirected graphs [48]. Another related approach was presented in 2016 by Cheng, Dong, and Lapata on Long Short-Term Memory-Networks for Machine Reading [49].

Since the approach researched in this paper is focused on the WL-labeling scheme, we will not go into further detail of these other approaches. After describing the proposed l-WWLLT method in section 3 in more detail, we will reflect on similarities to the just presented related work in section 3.5.5.

[3] For example see [10], [20], [22], [40]–[43]

1.4 Research Plan

The research plan can be divided into the following five steps. First, the proposed method and its evaluation are implemented. This includes redefining and specifying the specific methods and evaluations if deemed necessary. Second, experiments are conducted to search for appropriate parameters and successful application.

Due to the amount of (hyper)parameters and limited time, an extensive grid search could not be performed. Third, research questions are answered and attempts are made to improve the proposed method or parameter settings. Fourth, meaningful results are reported and the method is compared with the state of the art. Fifth, provide an outlook on how the method may be further improved.

1.5 Results

The experiments (section 4.3) show that the proposed l-WWLLT method can indeed improve the original definition of the similarity measure. In other words, there are configurations of the learning method where the edge weights are adjusted favorably in terms of both clustering metrics and SVM accuracy. The main research question can be answered positively. However, the experiments also show that this is highly dependent on the parameter settings and may perform poorly for most configurations. It is also observed that the learning method may improve individual success metrics (e.g., SVM accuracy) but not the others (e.g., Cluster scores) . The failed configurations show how changing edge weights based on two sample graphs, affects the distance between many graphs in a potentially degenerative way. Thus, local improvement can lead to global degradation of the resulting similarity measure.

Nevertheless for four our of seven dataset, a l-WWLLT Kernel was computed, which improves over all other considered kernels with state of the art performance. On three datasets a comparable performance was measured and on only one dataset all computed l-WWLLT Kernels perform worse than the state of the art. Performance wise, the l-WWLLT can be related to the No-Graph Kernel [44].

Further research is needed to extend the experiments to more bigger datasets. And to argue, whether predicable performances of the resulting l-WWLLT Kernel can be learned with more suitable parameters or method variations.

2 Theoretical background

This chapter serves the presentation of the most relevant definitions and methods for understanding the l-WWLLT method and its performance. Each section will be summarized by a reference that indicates how the definitions given are used in the method.

First, definitions from Graphtheory are presented. These are necessary to understand both the used data and the structures to define the graph similarity measure on them. Next, different metrics and the basics of the used graph similarity measure are presented. Thirdly, the already mentioned Weisfeiler-Leman labeling scheme and a hierarchy on the resulting Weisfeiler-Leman labels is defined. Both are central in the computation of the graph representations used in the similarity measure. Next, as indicated in the related work paragraph (in section 1.3), graph kernels prove useful to apply and compare the similarity measure. Thus kernels in general, graph kernels, and important example are introduced. At last, some terminology to evaluate clustering algorithms and Support Vector Machines is given. These will be used in the evaluation of the l-WWLLT method.

2.1 Graph Theory

The goal of the l-WWLLT method is to define a similarity measure on graphs. In order to be able to do so we will first define similarity measure on substructures of such graphs. This similarity measure is based on a special graph called tree, which is the backbone of the method. Therefore the relevant definitions from the field of Graphtheory are stated in this section.

We define a **graph** as a tuple $G = (V, E)$, where V denotes a finite set of vertices and $E \subseteq 2^V$ a finite set of edges on V. We also write $V(G)$ to refer to the vertices of a graph G. In this thesis we consider edges to be undirected. That is an **edge** is a set of two distinct vertices:

$$E = \{\{v, w\} | v \neq w \ \wedge \ v, w \in V\}$$

Such graphs are sometimes called undirected simple graphs. Since the following definitions relate to graph, keep the notation $G = (V, E)$ fix in this chapter. Analogously we will refer to two arbitrary vertices $v, w \in V$. If for subsets $V' \subseteq V$ and $E' \subseteq 2^{V'}$ with $E' \subset E$, the tuple $T' = (V', E')$ is also a graph, we call it **subgraph** of G. We say G is **labeled**, if its vertices have categorical labels. These are denoted by a function $L : V \to \Sigma$ for a finite alphabet Σ [21]. In the context of this thesis, we will have natural numbers as labels Σ. Such that for an $s \in \mathbb{N}$ it is $\Sigma = [-1, s] \subseteq \mathbb{N}$. We may write $G = (V, E)$ as $G = (V, E, L)$ to indicate the usage of a specified labeling function L on G. We say a graph is **weighted**, if there is a weight function

$w : E \to \mathbb{R}$ on the edges of the graph. If the set of edges is clear from the context and can be viewed as a vector, we may use a vectorized notation of the edge weights too. Thus for m edges in vector form $e \in (V \times V)^m$ we can denote the edge weights as vector $w \in \mathbb{R}^m$ such that

$$\forall i \in [0, m-1] : \ w(e_i) = w_i$$

When G is clear from the context, denote the number of vertices as $n := |V|$ and the number of edges as $m := |E|$. Two graphs G and G' are **isomorphic** ($G \simeq G'$), if there exists a bijective function between their vertices, that preserves all edges, and vertex labels. That is $G \simeq G'$ if and only if there exists a function $f : V(G) \to V(G')$ such that

$$\forall v, w \in V(G) : \ \{v, w\} \in E(G) \iff \{f(v), f(w)\} \in E(G')$$

In this case, the graphs differ only in the enumeration of their vertex sets (and thus edges), but not in their structure [25]. Recall that **bijective** functions (also called invertible functions) are injective and surjective. **Injective** functions map distinct elements of its domain, to distinct elements in its range ($f(x) = f(y) \implies x = y$). And **surjective** functions are characterized by the property that the function's co-domain is the image of at least one element of its domain.

For $1 \leq k \in \mathbb{N}$ we define a **walk** x in G from vertex $v_1 \in V$ to vertex $v_{k+1} \in V$ as a sequence of edges and vertices $x = \{v_1, e_1, v_2, \ldots v_k, e_k, v_{k+1}\}$ where $e_i = \{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, k$. The **length** of this walk is denoted $|x| = k$. Thus edges can be seen as walks of length one. The set of all walks from v_1 to v_{k+1} is denoted as $\mathcal{W}(v_1, v_{k+1})$. G is called **connected**, if for every two vertices v and w there exists a walk $x \in \mathcal{W}(v_1, v_{k+1})$ in G. If v is connected to w by an edge ($e = \{v, w\}$), w is called **neighbor** of v. The **neighborhood** $\mathcal{N}(v)$ of a vertex v as the set of all its neighbors [21]:

$$\mathcal{N}(v) = \{u \in V | \ \{u, v\} \in E\}$$

More generally we define the **k-th-order neighborhood** $\mathcal{N}_k(v)$ of v as the set of vertices (k-th-order neighbors) n such that there exists a walk $x \in \mathcal{W}(v, n)$ of length of at most k:

$$\mathcal{N}_k(v) = \left\{u \in V \middle| \ \exists x \in \mathcal{W}(v, u) \text{ s.t. } |x| \leq k\right\}$$

Notice that $\mathcal{N}(v) = \mathcal{N}_1(v)$. And if a vertex has at least one neighbor, the vertex is contained in all of its k-th order neighborhoods ($\mathcal{N}(v) \neq \emptyset \implies v \in \mathcal{N}_k(v)$ for $k > 1$). The **degree** $\delta(v)$ of a vertex v is given as the size of its first neighborhood, that is $\delta(v) := |\mathcal{N}(v)|$. It is $\delta : V \to \mathbb{R}$. If clear from context, the function $\delta : X \times X \to [0, 1]$ on the other hand denotes be the **Kronecker delta** (meaning $\delta(x, y) = 1$ if $x = y$ and 0 otherwise).

We define a **path** P as a walk $p \in \mathcal{W}(v_1, v_{k+1})$ such that no vertex (and thus no edge) in the walk equals another ($v_i \neq v_j$ for all $1 \leq i < j \leq k+1$). Let G be weighted, with a weight function $w : E \to \mathbb{R}$. The set of all paths from v_1 to v_{k+1} is denoted as $\mathcal{P}(v_1, v_{k+1})$. We define the **distance** $d_G(v, w)$ between v and w as the minimal sum of all weights of all edges for every path between them:

$$d_G(v, w) = \min_{P \in \mathcal{P}(v,w)} \sum_{e \in P \cap E} w(e)$$

We define a **circle** in G as $x = \{v_0, \{v_0, v_1\}, P, \{v_{k+1}, v_0\}, v_0\}$ such that $P \in \mathcal{W}(v_1, v_{k+1})$ is a walk and $\{v_0, v_1\}, \{v_{k+1}, v_0\} \in E$. We define a **tree** as a connected graph $T = (V, E)$, which does not contain any circles [50]. Notice that this implies that all paths in T are unique. That is there cannot be two different path between two vertices. The notion of subgraphs is extended to subtrees. A tree $T = (V, E)$ is called **rooted** if there exists a distinguished vertex $r \in V$ called **root**. In every tree we call a vertex with only one neighbor a **leaf** of the tree. The **depth** of a vertex $v \in V$ in a rooted tree with root r is defined as length of the path between v and r ($\mathcal{P}(v, r)$). Denote the depth with the variable D (as differentiation to a metric d). We say the root has depth zero. For two neighboring vertices v, w and the edge that connects them $e = \{v, w\}$, we call v the **parent** of w, if v has smaller depth than w. w is then called **child** of v. We write this as $v_c = w$ and $w_p = v$. Notice that in this definition w cannot be the root r, since no vertex has smaller depth than r. To express that the root has no parent we set $r_p := \emptyset$. We extend the definition to the connecting edge e and say v is the parent vertex of e and w is the child vertex of e. Analogously we write $e_p = w$ and $e_c = w$. All vertices on the path from v to the root r are called **ancestors** of v. We call a tree **leveled**, if all paths between all its leaves and its root have the same length.

If not explicitly stated otherwise, the notion of graphs or graph datasets in this thesis will refer to undirected, labeled graphs. If not explicitly stated otherwise, the notion of trees in this thesis will refer to undirected, connected, weighted, labeled, rooted and leveled trees. That is because we will use graphs as data structures and trees as meta data structures on a set of such graphs. The details of this relation are explained in section 2.3.2.

2.2 Metrics

Let M be a set and $m : M \times M \to \mathbb{R}$ on it, such that for all points $x, y, z \in M$ it is:

- $d(x, x) = 0$ (definiteness)
- $x \neq y \implies d(x, y) > 0$ (positive)

- $d(x,y) = d(y,x)$ (symmetry)
- $d(x,z) \leq d(x,y) + d(y,z)$ (triangle inequality)

We call the tuple (M,d) a **metric space**.

Let $T = (V,E)$ be a tree with non-negative edge weights and distance function d_G. Consider the domain Ω for a metric as a subset of the vertices of T ($\Omega \subseteq V$). A metric $d : \Omega \times \Omega \to \mathbb{R}$ is called a **tree metric** on Ω if $d = d_G$. That for two vertices $v, w \in \Omega$, d is computed by summing up the weights on the walk $\mathcal{W}(v,w)$ in T [51].

In this work, we will use tree metrics with $\Omega = V$. Also note, that we will only consider rooted trees and compute distances between vertices that have equal depth. This greatly reduces the actually used domain for our tree metrics.

2.2.1 Wasserstein Distance

One may use metrics as similarity measures between lower dimensional objects. However, it is not trivial to define a low-dimensional representation of graphs in such a way that a metric can be meaningfully applied. In the method presented in this thesis, we will use simple metrics as so called ground distances for arguably more complex similarity measures. The latter are defined in a research field called Optimal Transport theory (OT) [52]. It defines many powerful tools to compare probability distributions and has been used successfully to tackle the graph alignment problem before [47], [51]. OT was introduced by Monge [53] and reformulated by Kantorovich [54]. It has been applied in image processing, data analysis and machine learning [55]. Optimal transport problems ask for the most inexpensive way (with respect to a ground distance), to transport all mass from one probability distribution μ to another one ν [56]. The naive computation of an optimal transport plan between involves solving a network flow problem whose computation scales typically cubically in the size of the measures [57]. Togninalli, Ghisu, Llinares-López, et al. successfully used the Wasserstein distance instead, to construct a similarity measure for graphs with both categorical vertex labels and continuous vertex attributes [21]. Particularly the tree sliced variant offers a closed-form formulation computation [51]. It requires a hierarchy tree for the ground metric, which arises naturally from the definitions of the WL-labels. In this chapter we state definitions of Wasserstein distances (also referred to as earth mover distances). A specific configuration of a Wasserstein distance is used in this work as graph similarity measure.

Let μ and ν be two probability distributions on a metric space (M,d). The (continuous) **L^p-Wasserstein distance** for $p \in [1,\infty)$ is defined as

$$\mathbb{W}_p(\mu,\nu) := \Big(\inf_{\gamma \in \mathcal{T}(\mu,\nu)} \int_{M \times M} d(x,y)^p d\gamma(x,y) \Big)^{\frac{1}{p}}$$

where $\mathcal{T}(\mu, \nu)$ is the set of all transportation plans $\gamma \in \mathcal{T}(\mu, \nu)$ over $M \times M$. In this sense, we refer to d as the **ground distance** of the Wasserstein distance.

We can already specify the metric space in our application as $(\mathbb{R}^n, d)$. Now, a discrete closed form definition of the Wasserstein distance goes as follows. For two vectors $\mu \in \mathbb{R}^{n_1}$ and $\nu \in \mathbb{R}^{n_2}$ with $n_1, n_2 \leq n$, the (discrete) L^1-**Wasserstein distance** is defined as:

$$\mathcal{W}_{\mathcal{T}}(\mu, \nu) := \min_{T \in \mathcal{T}(\mu,\nu)} \langle T, C \rangle \tag{1}$$

Here, C is a cost or distance matrix:

$$\big(d(\mu_j, \nu_i)\big)_{\substack{i \in \{1,\dots,n_1\} \\ j \in \{1,\dots,n_2\}}} \in \mathbb{R}^{n_1 \times n_2}$$

$T \in \mathcal{T}$ is often called transport matrix or joint probability. It is $\mathcal{T} \subseteq \mathbb{R}^{n_1 \times n_2}$ and for $T \in \mathcal{T}$ it is $T\mathbb{1}_{n_2} = \mu$ and $T\mathbb{1}_{n_1} = \nu$. That is, T contains fractions that indicate how to map (transport) the values from μ to ν with minimal total cost (transport effort), with respect to the costs in C. $\langle \cdot, \cdot \rangle$ is the Frobenius inner product. Note that since d is a metric, so is the Wasserstein distance $\mathcal{W}_{\mathcal{T}}$ (Theorem 6.18 in [56]).

Tree Wasserstein Distance Later in this thesis, we will formulate graph representations as vectors of occurring WL-labels for different WL-labeling iterations. Thus as indicated above, we would like to compute the distance between such distributions of WL-labels. To do so, we can make use the hierarchy of the WL-labels and the thus derived WLLT, to compute the Wasserstein distance more easily. Instead of computing C as the distances between different WL-labels, given by the tree-metric d_T of the WLLT, we consider yet another definition of a Wasserstein distance. The negative definite tree-sliced Wasserstein distance (Tree Wasserstein Distance) proposed by Le, Yamada, Fukumizu, et al. in 2019 is computed by averaging between probability distributions using any given tree metric [51]. Le, Yamada, Fukumizu, et al. also showed, how to derive a positive definite kernel from this distance.

Let $T = (V, E)$ be a rooted, weighted tree with non-negative edge weights $w \in \mathbb{R}_+$, a root $r \in V$ and a tree metric d_T. We define $\Gamma(x)$ as the set of vertices in the rooted subtree of T that contains x as root and all other vertices have higher depth than x in T. That is to say all vertices, such that the unique path $\mathcal{P}(r, z)$ from them to the root r contains the vertex x:

$$\Gamma(x) := \{z \in V(T) |\ x \in \mathcal{P}(r, z)\}$$

Using this, we can define the closed form, negative definite **Tree Wasserstein Distance** on T in the following way [51]. For two (Borel) probability

distributions μ, ν on V, the Tree Wasserstein Distance can be computed as

$$\mathbb{W}_{d_T}(\mu, \nu) := \sum_{e \in E} w(e) \left| \mu\big(\Gamma(e_c)\big) - \nu\big(\Gamma(e_c)\big) \right| \tag{2}$$

If the tree metric d_T is given by a weight vector $w \in \mathbb{R}(|E(T)|)$, we will use the notation $\mathbb{W}_{d_T} = \mathbb{W}_w$ too. Notice, that for an edge e it is $d_T(e_p, e_c) = w(e)$. This Tree Wasserstein distance $\mathbb{W}_{d_T}$ is negative definite [51].

If not specified otherwise, we refer to this definition when using the term Wasserstein distance. Again, we will use this distance, to compute a similarity between graphs. Note, that the learning component of the presented method aims at iteratively altering the ground distance, and thus the evaluation of the Wasserstein distance. The definitions needed to understand the graph representations are given in section 2.3.1.

2.3 Weisfeiler-Leman Labels

WL-labels are used in the l-WWLLT method to summarize graph structures, which in turn are used to represent graphs and complete graph databases. [4] Thus we will first define them and the process of their computation, and then how to organize them in a meaningful manner, which allows to better understand their relation and how they are used as representations.

2.3.1 Weisfeiler-Leman Labeling Scheme

The **Weisfeiler-Leman** (WL) **labeling scheme** is vertex labeling scheme at the center of the Weisfeiler-Leman test of graph isomorphism proposed by Boris Weisfeiler and Andrei Leman [58]. In this thesis we will consider the 1-dimensional version of the WL-labeling scheme. It is also denoted as WL-labeling scheme, 1-WL vertex embedding, WL color refinement, naive vertex refinement. In terms of the higher-dimensional i-WL-labeling scheme it is equivalent to 2-WL-labeling scheme and thus the 2-variable language with counting [10], [47], [59].

The scheme iteratively propagates neighborhood information by compressing it in vertex labels [58]. These vertex labels will be called WL-labels, where the zeroth WL-labels can also denote original vertex labels.

The algorithm proceeds in iterations, which are denoted by index k. In each iteration, the current vertex labels are used to define new vertex labels, which then replace them in the next iteration. One iteration comprises the following steps.

[4] Often the notation "Weisfeiler-Lehman" can be found in the literature. Since this differs from the name stated in the original paper, we will use "Weisfeiler-Leman" instead [58].

Algorithm 1 WL-labeling scheme

Input: a graph $G = (V, E, L)$ and
an injective function hash.
Output: a vector representation of G of size $|V|$.

1: $\ell_0(v) \leftarrow \text{hash}\big(L(v)\big)$ for all $v \in V$
2: $\big(\ell_1(v)\big)_{v\in V} = \emptyset$
3: $i = 1$
4: **while** $\big(\ell_i(v)\big)_{v\in V} \neq \big(\ell_{i-1}(v)\big)_{v\in V}$ **do**
5: $\quad \ell_i(v) \leftarrow \text{hash}\big(\ell_{i-1}(v), \{\!\{\ell_{i-1}(n) |\ n \in \mathcal{N}_G(v)\}\!\}\big)$ for all $v \in V$
6: $\quad i = i + 1$
7: **end while**
8: **return** $\big(\ell_{i-1}(v)\big)_{v\in V}$

The procedure was generalized by [47] with $\ell_0(v) = L(v)$ for labeled graphs and $\ell_0(v) = |\mathcal{N}(v)|$ for unlabeled graphs [17]. However using uniform labels $\ell_0(v) = 0$ for unlabeled graphs results in an equivalent labeling, just delayed by one iteration.

Calling the injective function a hash function (line 5) is a reference to the originally proposed method. Typically, one refers to hash functions as almost injective functions, which map from a high dimensional domain to a low dimensional range. However, in algorithm 1 we benefit from a so called perfect hash function, which is an injective function. Since we can keep the dimension of the domain low, this has no drawbacks. Using such an injective function allows to compute a maximally powerful embedding [47].

The aggregation step in line 5 is often realized by sorting the WL-labels of the neighborhood first. For the perfect hash function we consider the set of integers as its range. More precisely the hash function shall map given inputs to the next integer, which is not the image of any input yet.

Let Σ_0 denote the set of original vertex labels (for any database). Define $\Sigma_i \subseteq \Sigma$ as the set of WL-labels that occur in the WL-labeling of the graphs in iteration i. Note that all Σ_i are pairwise disjoint.

WL-labels and Unfolding Trees Notice how WL-labels (of deeper iterations) relate iteratively to the n-th order neighborhoods of the vertices. Thus comparing two WL-labels implicitly relates to comparing complete neighborhoods.

Furthermore, WL-labels can be visualized as so called unfolding trees [22]. Consider a labeled graph $G = (V, E, \ell)$ and a vertex $v \in V$. The 1**-unfolding tree** $U_1(v)$ of v is defined as the rooted tree with root $\ell(v)$ and leaves $\ell(n)$ for all neighbors $n \in \mathcal{N}(v)$. Recursively for $i > 1$ the i**-unfolding tree** $U_i(v)$

of v is defined as the rooted tree $U_{i-1}(v)$, where every leaf n was replaced by $U_1(n)$. If D WL-labeling iterations were performed on G, we say $U_D(v)$ is the **unfolding tree** of v. Notice that $U_1(v)$ contains all vertex labels and implied neighborhoods needed for the definition of the first non-trivial WL label $\ell_1(v)$ of v. Intuitively, an unfolding tree is a leveled tree, which shows the neighborhood labels used in the aggregation before applying the hash function in line 5 or algorithm 1. The recursive definition implies that there exists a bijection between WL-labels and the set of pairwise non-isomorphic i-unfolding trees [22].

Limitations of the WL Labeling Scheme In this sense, vertex representation of WL-labels can reduce the graph isomorphism to multiset equality. But the WL test of graph isomorphism is a linear-time algorithm that works for almost all graphs [59], [60]. That is because there are not-isomorphic graphs with an equal embedding, constructed by the WL-labeling scheme. This is important, since we derive graph representations from these labels. Thus notice, that the derived function of computing graph representations is not injective. That is two different graphs can have the same graph representation.

As an example consider two graphs G_1 and G_2 on six vertices (as sketched in figures 1 and 2). Let G_1 consist of two disconnected circles, each of length three. Let G_2 be a circle of length six. Given no or uniform initial labels, the WL-labeling scheme considers all vertices equally labeled. Lets call this label 0. 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 (based on the hash of "$0, \{\{0, 0\}\}$"). Again all vertices have the same label, and thus no further iteration of the WL-labeling scheme will generate a different labeling. In this case, the graph representations are equal and the WL test of graph isomorphism wrongfully declares the two graphs to be isomorphic. However, there are not many of such not-distinguished graphs and this effect will be negligible in the evaluation of the method [59].

Runtime of the WL-labeling Scheme In this paragraph, we reflect on the runtime of the WL-labeling procedure for D iterations. Consider a graph $G = (V, E)$ with $n = |V(G)|$ vertices and $m = |E(G)|$ edges. We consider the hashing operation to have constant runtime ($\mathcal{O}(1)$) since it can be easily implemented using an integer counter and a hash-dictionary with constant lookup time. The key insight is now, that sorting the WL-labels of the neighborhood can be done in linear time. To do so, we make use of the fact, that for each call of the sorting function, the neighborhood labels can have at most n different values. Thus using counting sort on this range allows for a sorting runtime of $\mathcal{O}(m)$. Thus for all D iterations the runtime

is $\mathcal{O}(Dm)$).

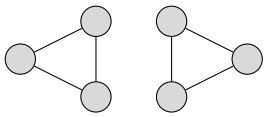


Figure 1: G_1 - Two disconnected circles, each of length three.

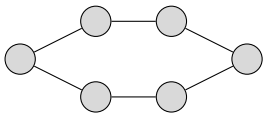


Figure 2: G_2 - A circle of length six.

2.3.2 Weisfeiler-Leman Labeling Tree

The method is based on the hierarchy of the WL-labels which are constructed by the 1-dimensional WL-labeling scheme (see algorithm 1). This hierarchy can be represented by a hierarchy tree, which we will call **Weisfeiler-Leman labeling tree** (**WLLT**). The WLLT is used in the l-WWLLT method to organize and distinguishing between representations for graphs in a given dataset.

In order to construct a WLLT, we require $D > 0$ iterations of WL-labelings on a non-empty database of graphs $\mathcal{G}$. Labeled and un-labeled graphs can be treated in the same way, by considering vertex labels as the zeroth WL-labels. A WLLT is a rooted tree T, where the root is an artificial label. The i-th layer of T consists of all WL-labels arising in iteration i of the labeling in any graph in $\mathcal{G}$. That is, there exists an edge (l, l') in T between vertices l and l' if there exists a vertex $v \in V(G)$ in some graph $G \in \mathcal{G}$ and $i \in \mathbb{N}$ with $l = \ell_i(v)$ and $l' = \ell_{i+1}(v)$. The complete construction of the WLLT is sketched in algorithm 2.

Algorithm 2 WLLT construction

Input: a graph dataset $\mathcal{G}$,
WL-labelings $\ell_0, \dots \ell_D$ up to depth D.
Output: WLLT T of height k (unweighted).

Initialize the WLLT as a single root $T = (r, \emptyset)$
for G in $\mathcal{G}$ **do**
  for $i = 0$ to D **do**
    for $l \in \{\ell_i(v) |\ v \in V(G)\}$ **do**
      if $l \notin T$ **then**
        $V(T) = V(T) \cup \{l\}$
        $E(T) = E(T) \cup (p, l)$ where $\ell_{i-1}(v) = p$ and $\ell_i(v) = l$
      end if
    end for
  end for
end for
return T

The i-th WL label of vertex v is given by $\ell_i(u)$. Recall, that we use Σ_i to denote the set of WL-labels of iteration i. This set relates to the i-th layer T.

Notice that if two vertices have different WL-labels in one refinement step, this implies that they have different WL-labels in all later refinement steps [20], [58]. For a proof by contradiction assume that two WL-labels for two vertices v and w in iteration j are equal ($\ell_j(v) = \ell_j(w)$), but different in any previous iteration $i < j$ ($\ell_i(v) \neq \ell_i(w)$). Without loss of generality, say i is maximal and j minimal in this property and thus $i = j - 1$. But now it is:

$$\begin{aligned}
\ell_j(v) &= \text{hash}\left(\ell_{j-1}(v),\ \{\!\{\ell_{j-1}(n) |\ n \in \mathcal{N}_G(v)\ \}\!\}\right) \\
&\neq \text{hash}\left(\ell_{j-1}(w), \{\!\{\ell_{j-1}(n) |\ n \in \mathcal{N}_G(w)\}\!\}\right) \qquad \text{since } \ell_{j-1}(v) \neq \ell_{j-1}(w) \\
&= \ell_j(w)
\end{aligned}$$

This implies that $\ell_j(v) \neq \ell_j(v)$, which contradicts the assumptions.

$$\forall u, v \in V\ \forall i \in \mathbb{N}: \quad \ell_k(u) \neq \ell_k(v) \implies \ell_{k+1}(u) \neq \ell_{k+1}(v)$$

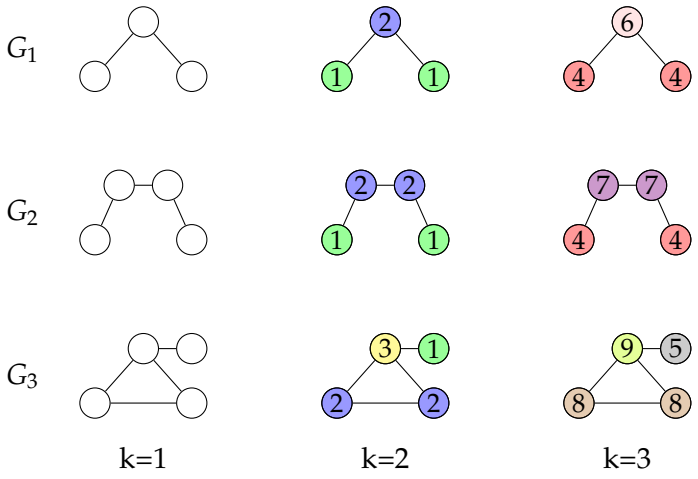


Figure 3: Examplary WL-labeling on three graphs.

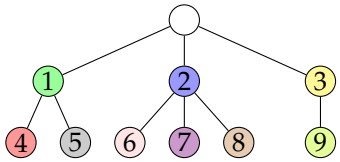


Figure 4: WLLT to the graphs in figure 3.

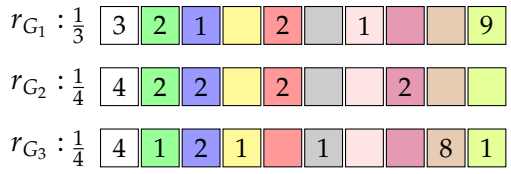


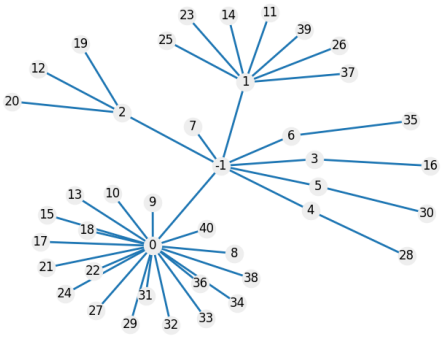
Figure 5: Graph representations of the graphs in figure 3 with respect to the WLLT in figure 4.

In this thesis, we restrict ourselves to the case that the given graph dataset contains labeled graphs. The set of these initial labels can be interpreted as zeroth WL-labels and the set of the vertices of the first layer in a corresponding WLLT (Σ_0). Notice that these have depth 1 in the WLLT and further layers of depth D correspond to the D-1-th WL-labeling.

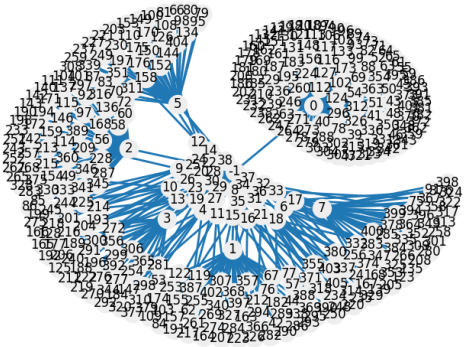
Figures 6b and 6a display two examples of WLLTs. Both examples display the tree for depth $D = 2$, and thus with two layers. Notice, how the much bigger and diverse dataset AIDS generates a much bigger tree than the dataset MUTAG with the same number of WL-labeling iterations.

As mentioned before, the l-WWLLT method aims to define meaningful distances between the WL-labels in the WLLT by adjusting weights on its

edges. An example of a weighted WLLT is shown in figure 7. The edge colors and edge lengths correspond to the edge weights.



(a) Example WLLT: MUTAG, $d = 2$.



(b) Example WLLT: AIDS, $d = 2$.

Figure 6: Unweighted WLLT examples

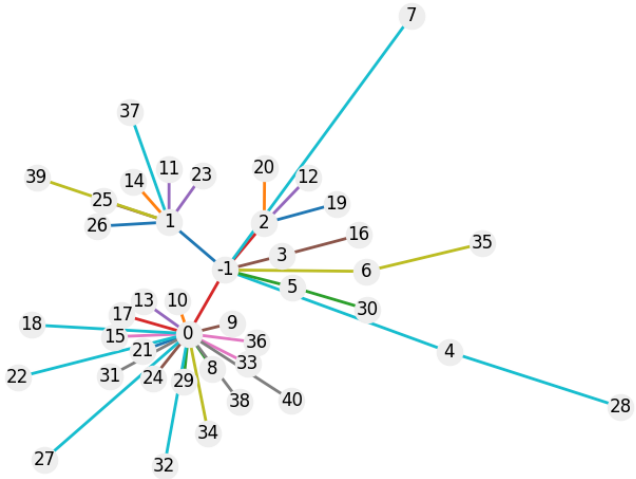


Figure 7: Weighted WLLT example: AIDS, $d = 2$.

Runtime of the WLLT Construction As shown in section 2.3.1, the computation of d iterations of WL-labels for a graph with m edges can be done in $\mathcal{O}(dm)$. Using non-negative integers as WL-labels allows for an easy construction of the WL label hierarchy (parent list) in linear time. The definition of edge weights requires to initialize p values, if the WLLT holds p many WL-labels and the (possibly artificial) root. Depending on the desired initialization, the edge weights can be defined during the construction. For a more sophisticated initialization note, we may limit p by the number of WL iterations, and the total number of vertices in the whole graph dataset. A stricter limitation of p will not be useful, if the initialization method is

not specified. Thus we conclude that for a graph dataset $\mathcal{G}$ of N graphs with at most m edges, the WLLT can be constructed with uniform weights in $\mathcal{O}(Ndm)$.

2.4 Kernels

A function $f : \mathbb{R} \to \mathbb{C}$ is called **positive-definite** if for any real numbers $x_1, \ldots, x_n$ the matrix $A = (a_{i,j})_{i,j=1}^n$ with $a_{i,j} = f(x_i - x_j)$ is positive **semi-definite**, that is $x^\intercal Ax \geq 0$ for all $x \in \mathbb{R}^n$. In case of a symmetric matrix, this implies that it is positive definite if all its eigenvalues are non-negative. If it is $x^\intercal Ax = 0$ only for $0 = x \in \mathbb{R}^n$, the matrix is called **strictly positive definite** [19]. A function f is (**strictly**) **negative-definite** if $-f$ is (strictly) positive-definite [61].

Kernels are similarity functions that can be interpreted as a dot product in a high-dimensional space. Using the kernel trick, they are often used in learning algorithms that rely on dot products, such as classifications tasks using support vector machines (SVMs) [21], [62]. Other applications are regression [63], clustering [64] and principal component analysis [65].

A **kernel** on a set X is a function $k : X \times X \to \mathbb{R}$ such that there is a (reproducing) real Hilbert space $\mathcal{H}$ and a mapping $\phi : X \to \mathcal{H}$ (feature map) such that $k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$ for all $x, y \in X$ [18], [19]. Here, $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product of $\mathcal{H}$. The definition of definiteness extends from matrices to kernels, in terms of the Gram matrix or kernel matrix $K := \big(k(x, y)\big)_{x,y}$ [19]. Equivalently, a function $k : X \times X \to \mathbb{R}$ is a kernel if and only if for every subset $\{x_1, \ldots, x_n\} \subseteq X$ the $n \times n$ matrix $[m]_{i,j} = k(x_i, x_j)$ is positive semi-definite [20], [21]. Note that it is equivalent to say, a function $k : X \times X \to \mathbb{R}$ is a kernel if and only if for every subset $\{x_1, \ldots, x_n\} \subseteq X$ the $n \times n$ matrix $K_{i,j} = k(x_i, x_j)$ is positive semi-definite [20]. $K_{i,j}$ is called **Gram matrix** of k with respect to $x_1, \ldots, x_n$. Or in other words a symmetric function $k : X \times X \to \mathbb{R}$ is called a positive definite kernel, if

$$\forall c_i \in \mathbb{R}\ \forall n \in \mathbb{N}\ \forall x_i \in X : \qquad \sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \tag{3}$$

k is called **conditional positive definite**, if it satisfies equation 3 for all $c_i \in \mathbb{R}$ with $\sum_{i=1}^n c_i = 0$ [21]. A method referred to as kernelized, if the kernel can be computed without explicitly representing the feature vectors $\phi(x)$, but instead relying only on indirect computations. This is most useful for high dimensional feature spaces, since the explicit computation for associated feature maps usually involves high costs [25].

The Dirac Kernel k_δ is defined by $k_\delta(x, y) = 1$, if $x = y$ and 0 otherwise.

A **graph kernel** is a symmetric, positive semidefinite function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ on the set of graphs $\mathcal{G}$, such that there exists a map $\phi : \mathcal{G} \to \mathcal{H}$ into

a Hilbert space $\mathcal{H}$ such that

$$\forall G_i, G_j \in \mathcal{G} : \; k(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle_{\mathcal{H}}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Notice that graph kernels can be defined on both the vertices or the edges of a graph [19].

Note that graph kernels in general do not need to be positive (semi-)definite. But if they are not, one cannot make use of the kernel trick by representing the computation as an inner product in a any Hilbert space. However for example the Optimal Assignment Kernel proved useful (see [9]) despite being not positive definite in general [66].

2.4.1 Laplacian Kernel

The **Laplacian Kernel** has favorable conditions for positive definiteness in the case of non-Euclidean distances [67], [68], and when using noiseless data [69]. Since many graph similarity measure, as well as the l-WWLLT distance, may not be an Euclidean distance, this kernel is useful to derive a kernel from such a proposed graph similarity measure. It is defined as:

$$k(x, y) = \exp\big(-\lambda d(x, y)\big) \tag{4}$$

Schoenberg showed that the Laplacian Kernel is positive definite for all $0 < \lambda \in \mathbb{R}$, if and only if the ground distance d is (conditional) negative definite. For a proof of this claim see Theorem C.3.2 in [70], [71].

2.4.2 Tree-sliced Wasserstein Kernel

Recall equation 2 for the definition of the negative definite Tree Wasserstein Distance. Le, Yamada, Fukumizu, et al. proposed a the **Tree-sliced Wasserstein Distance**

$$\mathrm{TSW}(\mu, \nu) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{W}_{d_{T_i}}(\mu, \nu) \tag{5}$$

Using it in the Laplacian kernel (equation 4) results in the **Tree-sliced-Wasserstein** (**TSW**) **Kernel** k_{TSW} [51]:

$$k_{\mathrm{TSW}}(\mu, \nu) := \exp\big(-\lambda \mathrm{TSW}(\mu, \nu)\big)$$

Since we define the l-WWLLT as an instance of the Tree-sliced Wasserstein Kernel, it is important to notice that this kernel is positive definite. As mentioned in section 2.4.1, the Laplacian kernel is positive definite for all $\lambda > 0$, if and only if the ground distance d is (conditional) negative definite. This in turn highly depends on the used ground metric. For example, Wasserstein distances with discrete metrics as ground distances are conditional negative definite [72]. Le, Yamada, Fukumizu, et al. showed that

the Tree Wasserstein distance is negative definite (Proposition 2 in [51] following [67] on pages 66-67), and thus also conditionally negative definite. Since the Tree-sliced Wasserstein Distance is thus a sum of negative definite functions, it is negative definite itself and the TSW Kernel is positive definite as desired.

2.4.3 Weisfeiler-Leman Graph Kernel

For d iterations of the WL-labeling scheme, lets call a graph G with the WL-labeling of the i-th iteration the **WL graph at depth** i:

$$G_i = (V, E, l_i)$$

Furthermore lets define the **WL-sequence up to height** i of set graph G as the sequence:

$$(G_0, G_1, \ldots, G_i) \;=\; ((V, E, \ell_0), (V, E, \ell_1), \ldots, (V, E, \ell_i))$$

As said before, we will consider the original labels of a graph $G = (V, E, L)$ as labels of the zeroth WL iteration ($L = \ell_0$).

Let k be any positive semi-definite kernel on two graphs G and H. Then the **Weisfeiler-Leman (WL) Graph Kernel** [10] with D iterations with k as base kernel is defined as

$$k_{\mathrm{WL}}^{(D)}(G, H) \;= \sum_{i=0}^{D} k(G_i, H_i)$$

A more general definition can be derived, when weighting the WL iterations with $0 \leq \alpha_i \in \mathbb{R}$:

$$k_{\mathrm{WL}}^{(D)}(G, H) \;= \sum_{i=0}^{D} \alpha_i k(G_i, H_i)$$

This weight function allows for example to emphasize larger substructures, which are represented by WL-labels from deeper WL iterations [22].

2.4.4 Weisfeiler-Leman Optimal Assignment Graph Kernel

The **Weisfeiler-Leman Optimal Assignment (WL-OA) Graph Kernel** proposed by Kriege, Giscard, Computer, et al. is similarly based on the hierarchy implied by the WL-labels [20]. To understand the difference to the approach in this thesis, we briefly summarize the definition of the WL-OA. Consider two graphs G and H and their vertex sets $V(G)$, $V(H) \subseteq \mathcal{V}$. Let $\mathcal{B}\big(V(G), V(H)\big)$ denote the set of all bijections between $V(G)$, $V(H)$. Given d iterations of the WL-labeling scheme for two vertices $v, w \in \mathcal{V}$, define the

base kernel of the WL-OA as the sum of equal WL-labels over all these iterations:

$$k_d(v,w) := \sum_{i=0}^{d} k_\delta\big(\ell^i(v), \ell^i(w)\big)$$

Now the WL-OA Kernel is defined as

$$K(G,H) := K_{\mathcal{B}}^k\big(V(G),V(H)\big) = \max_{B\in\mathcal{B}\left(V(G),V(H)\right)} \sum_{(v,w)\in B} k_d(v,w)$$

If the vertex sets have different carnality one may introduce artificial vertices z to the smaller set with $k^d(z,x) = 0$ for all $x \in \mathcal{V}$.

Notice that $k^d(v,w)$ corresponds to the number of matching WL-labels in the refinement sequence. As mentioned before, if two vertices have different WL-labels at any depth, all their WL-labels in deeper depths are different too. Thus the WL-OA Kernel bases its similarity measure on the length of the path in the WLLT to the lowest common ancestor of the WL-labelings of two vertices.

2.4.5 Wasserstein Weisfeiler-Leman Graph Kernel

Similarly to the WL Graph Kernel (section 2.4.3) and the WL-OA Kernel (section 2.4.4), the **Wasserstein Weisfeiler-Leman** (**WWL**) **Graph Kernel** is constructed using WL-labels of a graphs vertices. Different is however, that all WL-labels are used. Let there be D iterations of the WL-labeling scheme and consider a graph $G = (\{v_1,\dots,v_n\}, E)$ with n vertices. Again let $\Sigma = \mathbb{N}$ be the set of all possible WL-labels. For a vertex v in a graph G, define the vertex embedding $f_{\mathrm{WL}}^D : V \to \Sigma^{D+1}$ as the (row) vector of all WL-labels, that are assigned to v:

$$f_{\mathrm{WL}}^D(v) = \big(\ell_0(v),\dots,\ell_D(v)\big)$$

Now define a graph representation f_{WL}^D (graph embedding scheme [21]) as

$$F_{\mathrm{WL}}^D(G) = \begin{pmatrix} f_{\mathrm{WL}}^D(v_1) \\ \dots \\ f_{\mathrm{WL}}^D(v_n) \end{pmatrix} = \big(\ell_i(v_j)\big)_{\substack{j\in\{0,\dots,D\}\\ i\in\{1,\dots,n\}}} \in \Sigma^{n\times(D+1)}$$

Note that Togninalli, Ghisu, Llinares-López, et al. described this graph representation as the concatenation of WL features, which are the *column* vectors of all WL-labels for all vertices, in one fixed WL-labeling iteration. Next, we define the normalized Hamming distance $d_{\mathrm{Ham}} : \Sigma^{(D+1)} \to \mathbb{R}$ as the normalized count of equal entries in two vectors:

$$d_{\mathrm{Ham}}(f_0,f_1) = \frac{1}{D+1}\sum_{i=0}^{D} \delta(f_0,f_1)$$

For $\lambda \in \mathbb{R}^+$ and using d_{Ham} as ground metric for the L^1-Wasserstein distance (see equation 1), we can define the WWL Kernel as:

$$K_{\text{WWL}}(G, H) = \exp(-\lambda \mathcal{W}(F^D_{\text{WL}}(G), F^D_{\text{WL}}(H))) \tag{6}$$

Notice, that we will define the l-WWLLT distance, very similarly. As ground distance, we use a tree metric, which depends on edge weights in the WLLT, which will be changed by a learning process. Due to the nature of the used ground distance, we also switch the computation of the Wasserstein distance to the tree-sliced variation (section 2.2.1).

2.5 Cluster Learning

There are many different metrics to evaluate a given clustering. Different scores measure the tightness or denseness, separation, and intra and inter dispersion of clusters. In general, we prefer clusterings with high separation and inter dispersion and with tight and dense clusters or low intra dispersion. We now state definitions of three cluster scores, which were used in the evaluation process.

The **Silhouette coefficient** (SS) S_{SC} is based on the comparison of the clusters tightness and separation [73]. Let a be the mean intra-cluster distance and b be the distance to the nearest cluster for each sample. Then S_{SC} is defined as

$$S_{\text{SC}} := \frac{b - a}{\max(a, b)} \quad \in [-1, 1]$$

That is the mean of the distances between a sample and the nearest cluster that it is not a part of. Thus positive values close to 1 indicate that the data is clustered in *well separated* clusters, and the samples are less similar to the other samples in the assigned cluster, than the to the ones in the closest other cluster. Negative values indicate that a sample has been assigned to the wrong cluster, i.a. another cluster is denoted as more similar. Values near zero indicate *overlapping clusters* - indicating that samples are cannot reliably be separated from other clusters [73]. The used implementation was provided by **Scikit-Learn.org**. [5]

The **Davies-Bouldin score** (SDB) S_{DB} gives an average similarity of each cluster with its most similar cluster [74]. Let $M_{i,j}$ be the distance between characteristic vectors of cluster i and j, and S_i the dispersion of cluster i. The function

$$R_{i,j} := \frac{S_i + S_j}{M_{i,j}}$$

[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score, https://github.com/scikit-learn/scikit-learn/blob/f3f51f9b6/sklearn/metrics/cluster/_unsupervised.py#L39

is a non-negative, symmetric similarity function. Furthermore, the similarity expressed by $R_{i,j}$ decreases, if the distance $M_{i,j}$ between the clusters increases, while the dispersion of the individual clusters remains constant. Now the Davies-Bouldin score S_{DB} is defined as the average maximum dissimilarity $R_{i,j}$ between all n clusters:

$$S_{\mathrm{DB}} := \frac{1}{n} \sum_{i=0}^{n-1} \max_{i \neq j} R_{i,j}$$

Thus the score is based on the ratio of within-cluster distances to between-cluster distances and clusters which are *farther apart* and *less dispersed* will result in a better score. Lower (positive) values indicate a better clustering. But the score is only zero if the dispersion of both clusters i and j vanish (that is each cluster consists of indistinguishable samples). Note that the dataset must be partitioned into at least two clusters with different cluster centers for the score to have meaning. The used implementation was provided by **Scikit-Learn.org**. [6]

The **Calinski-Harabasz score** (SCH) S_{CH} (also known as variance ration criterion) is the ratio of the mean between-cluster (intra) dispersion and the within-cluster (inter) dispersion [75]. Let n be the size of the dataset and $k \geq 1$ the number of clusters. For cluster i let C_i be the set of data points in it, c_i its the center and n_i its size. Let c_D be the center of the whole dataset. Then define the *between group dispersion matrix* as

$$B_k := \sum_{i=0}^{k-1} n_i (c_i - c_E)(c_i - c_E)^{\mathsf{T}}$$

and the *within-cluster dispersion matrix* as

$$W_k := \sum_{i=0}^{k-1} \sum_{x \in C_i} (x - c_i)(x - c_i)^{\mathsf{T}}$$

Now the Calinski-Harabasz score is defined as the

$$S_{\mathrm{CH}} = \frac{\mathrm{tr}\left(B_k\right)}{\mathrm{tr}\left(W_k\right)} \frac{n-k}{k-1}$$

The score is higher when clusters are *dense* and *well separated*. The used implementation was provided by **Scikit-Learn.org**. [7]

[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html, https://github.com/scikit-learn/scikit-learn/blob/f3f51f9b6/sklearn/metrics/cluster/_unsupervised.py#L307

[7] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html, https://github.com/scikit-learn/scikit-learn/blob/f3f51f9b6/sklearn/metrics/cluster/_unsupervised.py#L253

3 Method

In this section, the researched method itself is explained. The explanation follows roughly the implementation, but a more in-depth explanation of the code can be found in subsection **??**. Before describing important steps in the method in detail, we give an overview over the method. The first step is to take a classified graph dataset (training set), and subjected all graphs to the WL-labeling scheme up to a specified fixed number of iterations (algorithm 1). During this, the corresponding WLLT is constructed (algorithm 2). After the complete construction of the WLLT, it is equipped with initial edge weights (section 3.2). This allows to define vector representations of the graphs (section 3.1). Now machine learning is applied, to iteratively update these edge weights (section 3.3). The update decision is based on decreasing the distance between graph samples from the same class, and increasing the distance between sample from different classes. It is crucial to notice, that changing the distance between two samples influences edge weights in the WLLT, which may be used to compute the distance between many other samples. The performance of the learner is instead measured on the resulting distance matrix for all graphs. Some of the used performance evaluation is presented at the end of this section, too.

3.1 Graph Representation

One of the easiest data-structures to represent complex structure data are fixed-size vectorial representations. Similarly to the approach used in the No-graph kernel proposed by Schulz, Horváth, Welke, et al., we used a vertex label histogram as graph representations. However unlike the histogram used for the definition of the No-graph kernel, the vertex labels we use, compress information on the structure of the (labeled) graph. We define the **graph representation** (graph **feature vectors**) r_G for a graph G in the following way. Let T be the WLLT with p vertices ($|V(T)| = p$) and depth d (i.e., d iterations of WL-labelings were performed on the graph dataset). Note, that this includes an artificial root and $p - 1$ assigned WL-labels, where the WL-labels of the zeroth iteration are the original vertex labels in the graph dataset. Further note, that all labels in the WLLT are distinct. For a graph G let S_i denote the set of vertices, that were assigned the label i in any of all d iterations of the labeling scheme:

$$S_i := \{v \in V(G) | \exists j : \ \ell^j(v) = i\}$$

Now define the graph representation r_G of G as the normalized distribution of WL-labels among the vertices of G:

$$r_G := \frac{1}{|V(G)|}\big(|S_0|, |S_1| \dots |S_p|\big) \ \in \mathbb{R}^p \tag{7}$$

For high d and a diverse dataset, it is expected that these representations are sparse vectors.

3.2 WLLT Edge Weight Initialization

Since we measure the performance of the learner by comparison to the initial state (see paragraph "Research question" in section 1.2), the initialization of the edge weights in the WLLT effect the computations and evaluations significantly. The edge weights are used additatively in the computation of the distance between two vertices (WL-labels) in the WLLT (recall the definition of **distance** in section 2.1). Furthermore, we require non-negative edge weights for a resulting positive definite Tree-sliced Wasserstein Kernel (section 2.2.1). Thus it is reasonable to initialize the edge weights with positive values. In all edge weights in the zeroth layer of the WLLT, that is the weights of all edges incident to the root, completely define the distances between the zeroth WL-labels. If initial vertex labels are known and used as zeroth WL-labels, one may use application based knowledge to initialize these edge weights. Otherwise, and for all other edge weights, the uniform initialization is reasonable. This way, the tree-structure itself ensures that WL-labels of different subtrees are further apart, which reflects the idea of the WL-labeling scheme. For most experiments and during the development of the method, a uniform initialization to value 1.0 (with no prior application based knowledge) was used.

Other possible initialization, like based on a gradient through the depth of the WLLT (fixed or based on the layer size) may be applicable too. One may assume that the learning process could be able to reach a desirable configuration, largely independently of the initialization. The experiments however were not extensive enough do not support this assumption empirically.

3.3 Edge Weight Learner

For reference, the processes described in this subsection are implemented in the script `x3_wllt_edge_weight_learner.py`.

In order for the edge weight learning to begin, a complete WLLT (up to some depth and with edge weights), the feature vectors (section 3.1) of all graphs, and the graph classifications must be given. The outer loop in the computations performed by the edge weight learner, loops over all learning epochs and returns newly defined edge weights each time. Each learning epoch consists of an inner loop over the graph samples in a chosen batch of graph samples for this learning epoch. Since the inner loop loops over all pairs of graphs (computing their distance), the batch size enters quadratically into the calculation of the runtime of the learning process. Before

starting the first epoch, the batches for every epoch are constructed. To simplify later computations, each such batch contains approximately the same number of graphs from every given class. If the sizes of different graph classes vary greatly, this is not ideal, with respect to an equalized weighting of the graph samples. To diminish such issues, the batch sizes were chosen accordingly for the respective datasets. For each learning epoch, all different graph combinations of the graphs in the batch are used to compute a weight update Δw for the WLLT edge weights w:

$$w' = w + \eta * \Delta w$$

Here, $\eta \in (0, 1]$ is the learning rate. Let $r_{G_0}, r_{G_1} \in \mathbb{R}^p$ be the graph representations for two graphs and $w \in \mathbb{R}^p$ the vector of current edge weights in the WLLT. Recall equation 2 with distributions $\mu = r_{G_0}$ and $\nu = r_{G_1}$. Denote the weighted difference vector between the graphs as $\Delta r_{G_0,G_1} \in \mathbb{R}_+$:

$$\mathbb{W}_{d_T}(r_{G_0}, r_{G_1}) = \sum_{e \in E} \underbrace{w(e) \left| r_{G_0}\big(\Gamma(e_c)\big) - r_{G_1}\big(\Gamma(e_c)\big) \right|}_{=:\Delta r_{G_0,G_1}(e)} = \sum \Delta r_{G_0,G_1}$$

Now if the graphs are in the same class ($c(G_0) = c(G_1)$), their distance $\mathbb{W}_{d_T}(r_{G_0}, r_{G_1})$ shall be decreased, and increased otherwise. To effect as little other distances as possible, we update only the edge weights $w(e)$ where $\Delta r_{G_0,G_1}(e) \neq 0$. These are exactly all edge weights, which contribute in the computation of the Wasserstein distance between the considered graphs. Let $\delta^>$ be an indicator function such that $\delta^>(x) = \{i | x_i > 0\}$. If not stated otherwise, in the following we simplify the notation by writing $w_{\delta^>(\Delta r_{G_0,G_1})}$ as w. We use a **pull factor** $f_{\text{pull}} \in (0, 1]$ and a **push factor** respectively $f_{\text{push}} \in (0, 1]$ and set:

$$\Delta w = \begin{cases} f_{\text{pull}} w & \text{if } c(G_0) = c(G_1) \\ f_{\text{push}} w & \text{otherwise} \end{cases} \tag{8}$$

Notice that by definition of the push and pull factors (**pp-factors**), and the learning rate, the edge weight update is the addition or subtraction of a fraction of the already used weights. Thus the weights remain non-negative during all learning epochs. As mentioned, this is desired in order to use the resulting tree metric in the definition of a graph kernel.

If the edge weights were updated after each sample, the order of sample from the same class and from different classes would matter and the batch size has effectively size one. To prevent this, all weight updates during one learning epoch are stored but not applied. Applying all updates afterwards may lead to an exponential scaling effect, which we naturally would like to avoid. Thus, after each epoch the mean of all computed (non-negative) updates is applied.

Using the Wasserstein distance here to align the structural graph representations may be beneficial in capturing more complex characteristics of the graph, compared to taking the mean or other additive simplifications as were used for the definition of many graph convolution kernels (see related work in section 1.3) [21].

Absolute pp-Factors The described edge weight update has the benefit, that the edge weights remain non-negative all the time. However the definition of the weight delta as a percentage of existing weight may seem unusual. To compare the effects of a more traditional, additive update, such an update was implemented too. Using this setting in the learner, will make use of the following definition:

$$\Delta w = \begin{cases} f_{\text{pull}} & \text{if } c(G_0) = c(G_1) \\ f_{\text{push}} & \text{otherwise} \end{cases} \tag{9}$$

Since we do not need to fear exponentially de- or increasing weights in this setting, the accumulated weight updates are applied after each epoch without taking their mean.

Class-Imbalance Factor Since the update mechanism increases the edge weights for graphs from different classes and decreases the edge weights graphs from the same class, depending on the number of classes and their sizes, the relation between added and removed weight may vary greatly. To ensure better comparability between datasets, and for example different fractions of dataset, we reduce this effect by introducing a factor which will be referred to as **class imbalance factor**.

Let m be the batch size, and let there be n (selected) graphs for every of c classes in each batch. The number of possibilities to draw two graphs from the same class p_S amounts to the number of possibilities to draw the first graph from one fixed class ($n/2$), times the number of possibilities to draw a second, different graph from the same class ($(n-1)/2$). One may summarize these possibilities to $\binom{n}{2}$ as well. This has to be calculated for every of the c classes. Thus it is:

$$p_S = \binom{n}{2} c = \frac{n(n-1)}{2} c$$

The number of possibilities to draw two graphs from different classes p_D amounts to the number of possibilities to draw two graphs, each from two different fixed classes (nn), times the number of possibilities to draw such two different classes ($\binom{c}{2}$). Thus it is:

$$p_D = nn \binom{c}{2} = \frac{cnn(c-1)}{2}$$

Since these factors will be used only for scaling the applied weights, we ignore the common factor $nc/2$. Now one can scale the weight update of same class samples with $1/n-1$ and different class samples with $1/n(c-1)$. This however greatly decreases the weights, and in the implementation these factors were scaled, such that only different class samples are scaled by $1/((n-1)n(c-1))$. Thus the mentioned push factor in equations 8 and 9 is scaled accordingly:

$$f_{\text{push}} = \frac{f_{\text{push}}}{(n-1)n(c-1)}$$

Weights Scaling Using the absolute pp-factors requires to prevent negative edge weights, which where set to zero. On top of that the first experimentation showed, that the multiplicative weight update (according equation 8) has the potential to define exponentially increasing edge weights and thereby skewing the overall used edge weight vector. To prevent this effect, a parameter setting allows to set an upper bound for the weights too. The standard setting is $[0, 2]$, which is as powerful as scaling to the uniform interval 1.0, but more suitable to the described intuitive uniform edge weight initialization to the value 1.0 as the mean of the tolerated interval (section 3.2).

Heaviest Earth Threshold The method described so far updates all edge weights in the subtree of the WLLT that includes the root and all leaves, associated with positive entries in $\Delta r_{G_0,G_1}$ for two graphs G_0 and G_1. And the strength of the updates may depend on the pp-factors (the classes of the graphs), the number of graph classes and their sizes and may be limited to an interval.

Now consider the following example. Let G_0 and G_1 bet two almost identical graphs of the same class, but G_1 contains one vertex more, with an WL-label, which does not occur in G_0. Since the structures of the graphs are almost identical, the method should focus on decreasing the importance of the distinct label (to decrease the distance between the two graphs). However, since the graphs do not have the same number of vertices, their graph representations are likely to be different at almost any non-zero position (recall the normalization of the graph representation). Thus the so far described update method may decrease the weights to almost all associated WL-labels and impact the distances to other graphs greatly.

To limit the update effect in similar cases, and to reduce the overall introduced or subtracted weight, we define a parameter we call **Heaviest Earth Threshold** h. Using the definitions from above, $w_{\delta(\Delta r_{G_0,G_1})}$ is the vector used for the computation of the Wasserstein Distance. In tribute to the name Earth Mover Distance, this vector can be seen as the costs or weight of the earth which is needed to align the given distributions. Thus we can use

h to focus the weight update on the WL-labels, which have the most impact to the computation of the Wasserstein distance. Let H be the h-th highest value in w. Using an indicator function $\delta^{\geq}$ such that $\delta^{\geq}(x) = \{i | x_i \geq 0\}$, we only the updated weights with the indices

$$\delta^{+}(\Delta r_{G_0,G_1} - H)$$

These are all edge weights, which contribute at least the the h-th heaviest values to the distance computation. Notice, that the usage of the Heaviest Earth Threshold is effectively canceled, if it is set to the dimension of the weight vectors ($h = \dim(w) = p$).

In the implementation and when reporting the experiments in section 4.3 we refer to Heaviest Earth Threshold as t_{he}. This notation gives the percentage of non-zero edge weights which shall be updated. h is set accordingly.

Runtime As already discussed, the computation of d iterations of WL-labels for a graph with m edges can be done in $\mathcal{O}(dm)$ (section 2.3.1). Notice that constructing a graph representation has runtime $\mathcal{O}(dm)$ too (independently of the existence of the WLLT). and the construction of the WLLT for N graphs with at most m edges in $\mathcal{O}(Ndm)$ (section 2.3.2). Let there be p WL-labels in the WLLT. Thus the graph representations have dimension p. The computation of the distance between two graphs with known graph representation can be done in $\mathcal{O}(p)$ since it requires a subtraction, taking the absolute value and addition of two p dimensional real graph representations. Notice that this is a significant improvement over one of the native computation of the Wasserstein Distance (see equation 1), which has cubic runtime complexity ($\mathcal{O}(n^3 \log(n))$, where n gives the maximum number of vertices) [21].[8] For two graphs with unknown representation, which both have at most m edges, it requires a runtime in $\mathcal{O}(mh + p)$.

The learning procedure runs linearly in the number of requested learning epochs $e \in \mathbb{N}$. However the runtime of one learning epoch grows quadratically in the batch size b since all different pairs of graphs are used. In side the batch loop, the candidate selection step requires to sort a p dimensional real vector, which can be done in $\mathcal{O}(p \log(p))$. The other operations inside the batch loop have runtime linear in p. The other operations outside of the batch loop have runtime linear in p as well (e.g. accumulating and averaging the weight update). Thus the learning procedure for e epochs, b graphs per batch and an already constructed WLLT with p vertices and already constructed graph representation vectors has runtime complexity

$$\mathcal{O}\big(e(b^2 p \log(p) + p)\big)$$

[8] Although the naive computation can be improved to a near-linear time computation as well, when utilizing some tricks like the Sinkhorn regularisation [21].

These theoretical considerations should be supported by runtime measurements on implementations. Since the the focus of this thesis lies on the functionality and applicability of the proposed approach, and since the theoretical runtime is comparative to other kernels, runtime measurements experiments were omitted.

3.4 l-WWLLT Kernel

In order to apply a multitude of different tools from Machine Learning and to compare the performance of the implemented learner, we would like to translate the computed distance measure into a graph kernel. Given the WLLT T, and an non-negative edge weight vector $w \in \mathbb{R}^+$ (learned or defined) on it, we can compute a distance matrix D for any set of graphs $\mathcal{G}$ by using the graph representations defined in equation 7 and the introduced Tree-slice Wasserstein distance from equation 5:

$$D = \left\{ \mathbb{W}_w(r_{G_0}, r_{G_1}) \right\}_{G_0, G_1 \in \mathcal{G}}$$

We can make use of the Tree-sliced Wasserstein kernel (section 2.4.2) and set

$$k_{\text{lWWKKT}}(G_0, G_1) := \exp\big(-\lambda \mathbb{W}_w(r_{G_0}, r_{G_1}) \big)$$

Since this Wasserstein distance is positive-definite (following the argumentation with non-negative edge weights in [51]), the kernel is positive definite as well (following Theorem C.3.2 in [71] and the argumentation in [70]).

For the kernel coefficient λ a few parameters were used in the experiments. By far the best configuration was setting it as the inverse of the number of graphs (dimension of the computed distance matrix)[9].

3.5 Evaluation

The evaluation of the l-WWLLT method in general is by definition not part of the research itself and relies mostly on established and implemented functions. However, at least one of the used evaluation methods is tailored closely to the error function of the implemented method and thus useful for understanding and analyzing it (section 3.5.1). The experiments were partially guided by comparing different evaluation methods. Thus in order to better understand the experiments and the used evaluation, we state the most important evaluation methods in this section.

[9] In accordance with *sklearn*'s gamma setting called 'auto' for their implementation of the Support Vector Classification SVC (`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`) and their default setting for the Laplacian kernel (`https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.laplacian_kernel.html`).

3.5.1 Sample Movement Error

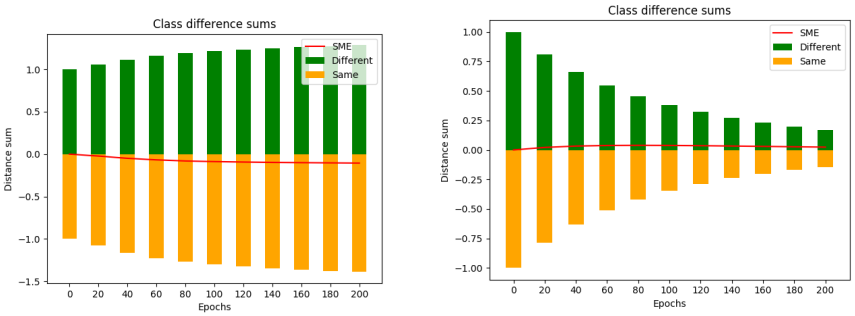
The proposed update rule (section 3.3) is defined with the intention in mind, to pull samples from the same graph class closer together (that is reduce their distance), and push samples from different classes further apart. Thus, the desired error function to be minimized is decreased by decreasing the sum of all distances between graphs from the same class ($\Sigma_{\text{same}}(w)$), and increase the distance between graphs from different classes ($\Sigma_{\text{diff}}(w)$), with respect to some edge weights w in the WLLT. This error function will be referred to as (global) **cluster movement error** (**SME**) E_{CM} and defined as:

$$E_{\text{CM}}(w) = \Sigma_{\text{same}}(w) - \Sigma_{\text{diff}}(w)$$

Note that the distances between different cluster samples may be allowed to increase limitless, thus negative error values may arise naturally. And depending on the size and number of classes, the absolute values of the error function may vary greatly from dataset to dataset. To reduce this effect, and since only the derivative, e.g. the change of the error function is important, we normalize the distance sums such that both sums for the zeroth iteration are 1.0 and consequently the error E_{CM} starts at zero. For epoch $e \in \mathbb{N}$ we define the sample movement error to be

$$E_{\text{CM}}(w_e) := \frac{\Sigma_{\text{same}}(w_e)}{\Sigma_{\text{same}}(w_0)} - \frac{\Sigma_{\text{diff}}(w_e)}{\Sigma_{\text{diff}}(w_0)}$$

Consider the example of the cluster movement error in figure 8. The x-axis lists learning epochs during which the used edge weights were changed. The z-axis denotes the normalized values for the sum of same- and different class samples and the cluster movement error itself. The yellow bars indicate the negative normalized sum of all distances between graphs from the same class, the green bars respectively the positive sums for those from different classes. And the red line plot indicates the cluster movement error. Thus an optimal plot has a decreasing red line, increasing green and decreasing yellow bars. However it is possible to have a decreasing red line, if the green bars decrease as well, but slower than the yellow ones.



(a) SME decreasing (Dataset *AIDS*) (b) SME increasing (Dataset *MSRC_9*)

Figure 8: SME error examples

The two examples visualize a decreasing and thereby improving SME (8a) and an increasing and thereby degenerating SME (8b). Notice how the SME is decreasing, but the sum of the same cluster distances is increasing. Nevertheless, the decreasing SME states, that the different cluster distances are increasing faster. Similarly the SME is increasing, but the sum of the different cluster distances is decreasing, all be it slower than the decreasing of the same cluster distances.

Global, batch-wise and local interpretation It is crucial to note, that the update rule (section 3.3) adjust the edge weights in the WLLT based on a *local* evaluation of the edge weights at the time of the update. That is, the edge weight changes are defined using two graph samples each. Let us call the SME computed right after the definition of the edge weight adjustment for an individual pair of graphs in the batch the local SME. This local SME decreases compared to right before the definition of the edge weight adjustment every time, since the update rule is designed to do so. However, as described in section 3.2, the edge weight changes originating from comparing one pair of graphs may be changed (averaged) and are only applied after each epoch. Computing the SME at this point we call computing the batch-wise SME. It is no longer guaranteed that the SME will decrease for all (hyper-)parameter settings. That is because changing one edge weight in the WLLT may influences the distance between several graphs, both from the same and from different classes. The same holds true for the global SME, which we defined first. The first and most direct indicator for success of the l-WWLLT method is, to reduce this global SME.

As mentioned before, we make use of additional evaluation methods. For example the classification accuracy using the l-WWLLT-kernel (section 3.4) in a support vector machine (section 3.5.2), or by measuring the quality of the given clusters of graph classes with respect to the similarity measure (section 3.5.3). Notice how these three evaluation methods measure differ-

ent aspects of the defined similarity measure. After presenting these two other evaluation methods, the reasoning behind this evaluation strategy are summarized in section 3.5.4.

3.5.2 Support Vector Machine

One of the easiest way to compare the performance of this learner to other methods with the similar goal to define a graph similarity measure, is to use a C-Support Vector machine (SVM) [76], [77]. SVMs try to define a model, which predicts target values (i.e. class labels) of features of given test data. In essence, they map feature vectors with known target values (training data) into a higher (maybe infinite) dimensional space and find a linear separating hyperplane (or support vectors) with the maximal margin in this higher dimensional space. This hyperplane is then used for further classification decision on the test data [76]. Many researchers measure their implementations in the mean classification accuracy (and standard deviations) of a computed kernel in a 10-fold cross validation SVM (for example [10], [20], [42], [44]) (repeated 10 times with random folds). That is, to partition the graph dataset in a training and evaluation set, and evaluate the performance of the computed support vectors on the training set. For better comparability, we use this method as well.

To use a suitable SVM, the computed distance matrix is transformed into a kernel matrix using the definition of the Tree-sliced Wasserstein kernel (section 3.4). This is the main motivation to keep the edge weights positive, because this implies a positive definite kernel [51]. It is recommended to scale the kernel linearly to the interval $[0, 1]$. This scaling may reduce dominating effects of attributes with greater numeric range over those with smaller ones and it may avoid numerical difficulties during the calculation [77]. However using only non-negative distances, the kernel definition using the exponential function, and having distances of value zero on the diagonal of the distance matrix already lead to properly scaled values.

3.5.3 Cluster Evaluations

In order to evaluate the already present clustering, when using the similarity measure induced by some (learned) edge weights w, we use the scores presented in section 2.5. Given the distance matrix of all graph classes according to the defined l-WWLLT-distance, no further parameters are required. To summarize, we use the Silhouette coefficient S_{SC} to evaluate the tightness and separation of the cluster. An increasing Silhouette coefficient (possibly approximating 1.0) is desirable in this application. We use the Davies-Bouldin score S_{DB} to evaluate the similarity between the clusters. A decreasing Davies-Bouldin score is desirable in this application. We

use the Calinski-Harabasz score S_{CH} to evaluate the cluster dispersion. An increasing Calinski-Harabasz score is desirable in this application.

Besides these scores, we track a multitude of other values with respect to the clustering. For example the maximum, minimum and mean distance between all samples the same and different classes, for all classes. We may refer to the distance between samples of the same class as intra-distance and to to the distance between samples of the different class as inter-distance. Notice that not all of these measurements have a fixed desired trajectory. Key indicators for an improved clustering are increasing minimum inter-distance, decreasing maximum intra-distance.

As a visualization of the clustering a T-distributed Stochastic Neighbor Embedding (t-SNE) [78] was used[10]. These embeddings are able to visualize high-dimensional data by considering data point similarities as (low-dimensional) joint probabilities. They then try to minimize the Kullback-Leibler divergence between these. Note that the visualizations are not deterministic and can be visualized differently in multiple computations.

3.5.4 Method Evaluation Strategy

Measurements like the mentioned statistics of all computed distances or visualizations of the edge weights in the WLLT, the distance matrix, and the WLLT itself were used to guide the implementation and the understanding of the method. The experiments with the l-WWLLT method, and thus its success were guided by considering the SME, the accuracy of the implied kernel, and the implied scores of the graph clustering. It is not mandatory for these three evaluations to align in their assessment. However we assume, that if a distance matrix, reflects a clearer separation between the samples of different classes, also relates to a better accuracy of the SVM, since the optimal separating hyperplane (or support vectors) will be less faulty. This may be true for the other way as well. Although a clear cluster separation does not improve all cluster scores equally.

It is important to keep in mind, that in the training process and our experiments, the clustering is given by the dataset and not computed by the learner itself. We only measure that the graph structures, represented in the graph representations relates to the given graph classifications. This also means, that if the classifications do not relate to the WL-labels (unfolding trees), there may be little chance, that the computed weights in the WLLT can reflect the classifications at all.

[10] More specifically the default implementation from the package `sklearn.manifold.TSNE` on `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`.

3.5.5 Comparison to Other Methods

After this explanation of the l-WWLLT method, we come back to the related work mentioned in section 1.3 and compare this approach to similar ones.

The researched method is similar to the WL-OA Kernel (see section 2.4.4) but differs in an important aspect. While the WL-OA only considers the equal subset of WL-labelings between two graphs, our approach considers the complete comparison of all labels, and especially the differences. Our approach emphasizes the differences, since similarities between equal feature vectors of two graphs are reduced when computing their L_1-norm. The between different feature vectors remain when computing the L_1-norm. In other words, the WL-OA focuses on the paths of the WLLT from the root to the lowest common ancestors (WL-labels) which both graphs have in common [20]. Our method focuses on the part of the WLLT from the lowest common ancestor to the leaves, but considers both paths as well. This may seem more intuitive for the task of defining a dis-similarity measure, since all dis-similarities with respect to the WL-label representation are considered. Note, that unlike Kriege, Giscard, Computer, et al. when proposing the WL-OA, we formulate the l-WWLLT with respect to categorical vertex labels only.

As mentioned above, the Wasserstein distance depends greatly on its ground metric d. In different formulas for different Wasserstein distances given in section 2.2.1, the ground metric is used either directly, to define the cost matrix or to define the used edge weights. Using a tree metric as ground distance, allows us to use the efficient close formula definition presented in equation 2 in section 2.2.1. Togninalli, Ghisu, Llinares-López, et al. used the Wasserstein distance on a histogram like representation of graphs too [21]. But as ground metric they used the normalized Hamming distance (for categorical vertex features) or the Euclidean distance (for continuous vertex features). Therefore a key difference between their approach and the one presented in this thesis, is to define and alter the ground distance as a tree metric instead. If the learning process can improve the overall performance of the similarity measure based on the tree metric, one may relate this performance to the performance when using the normalized Hamming distance or the Euclidean distance again.

The graph representations used in the NoG Kernel by Schulz and Welke only consider quantities of vertex and edge labels. Using a uniform edge weight initialization for the WLLT, this quantitative approach is included in the representation of the graph representation in the l-WWLLT (for the zeroth layer of the WLLT). One may argue, that the second layer in the WLLT holds similar information as the edge label count in the NoG. However we did not consider original edge labels. By including more layers, than just the zeroth, the l-WWLLT method includes structural information (on the neighborhoods and thus edges). Which differs from the NoG Kernel since

it ignores structural information [44]. Note however, that depending on the dataset, the original vertex (and edge) labels themselves can contain structural information as well. For example in graph modeling molecular structures often include information on the used atoms in the vertex labels. Often, the definition of an atom relates to possible structural properties as well.

4 Experiments

In this section the conducted experiments are motivated and presented. Therefore, we first present the graph datasets used for the l-WWLLT distance learning (section 4.1). Then the general setup of the implemented method is explained (section 4.2). We summarize the most important parameters, and how different settings were combined. Note that some parameters significantly change the behavior of the implemented method and are primarily used for the research on what specific approach may produce the desired results. Further note, that the number of parameters was to great, to allow for an extensive grid search. Instead, a few guiding experiments are presented (section 4.3). More evaluations can be found in the appendix (appendix A.2).

192 different experiments on 17 different datasets were conducted with the final implemented method. Since at first suitable parameter configurations were unknown many of these experiments are spot checks for some parameter configurations but no extensive gird search was performed for all parameter combinations. No reliable trend in performance could be discovered for all datasets. That is why in this section we consider sets of experiments and discuss their implications for the research on the l-WWLLT method. In section 5 the best results over all computations are briefly summarized and presented.

4.1 Datasets

For the experiments, three major sources for datasets of classified graphs were used. The TUDataset [35] and versions of its graph datasets [79][11] and the OGB dataset [80]. The versions of the TUDataset are based on the research by Ivanov, Sviridov, and Burnaev and can be seen as a pre-processing of the TUDatasets, done before the pre-processing in the implementation for the experiments. Since they have the same names as the datasets in the TUDataset, we refer to them by adding "_c" as a suffix to the original name (see for example table 4). Sine the results between these different versions of the datasets did not reveal insight on the l-WWLLT with respect to the research question, we will not go further into their construction here.

All loaded datasets were pre-processed, to ensure equalized treatment between different sources. The pre-processing includes the omitting of not needed graph information, deletion of graphs without vertices (where the WL-labeling is not applicable) and adding uniform vertex labels, if none are present. The resulting standardized graphs consist only of a vertex set with vertex labels, and an edge set.

[11] `https://github.com/nd7141/graph_datasets`

4.1.1 TU Dataset

The increased need of benchmark graph classification datasets lead to the construction of a graph dataset collection at TU Dortmund [34], [35]. We will refer to these datasets as TUDatasets. Notice that the datasets in the collection contain only 200 to $5,000$ graphs, which may be to small for meaningful tests of graph kernels on classification tasks [22] (see table 1 for reference.[12]). Since they were used by many other researchers, and allow for a faster test driven development of the l-WWLLT method, we use them for the evaluation of the method as well (for example [10], [20], [22], [40]–[43]).

Dataset	Source	#Graphs	#Classes	Avg. $\lvert V\rvert$	Avg. $\lvert E\rvert$
Small molecules					
AIDS	[81], [82]	2000	2	15.69	16.20
MUTAG	[11], [83]	188	2	17.93	19.79
NCI1	[10], [40]	4110	2	29.87	32.30
NCI109	[10], [40]	4127	2	29.68	32.13
PTC_MR	[83], [84]	344	2	14.29	14.69
Tox21_AhR*	[85]	8169	2	18.09	18.50
Bioinformatics					
DD	[10]	1178	2	284.32	715.66
ENZYMES	[8]	600	6	32.63	62.14
PROTEINS	[8]	1113	2	39.06	72.82
Computer vision					
MSRC_9	[86]	221	8	40.58	97.94
MSRC_21	[86]	563	20	77.52	198.32
MSRC_21C	[86]	209	20	40.28	96.60

Table 1: Statistics on some TUDatasets

The datasets are based on different graphs, representing different data structures. For example, the *MUTAG* dataset contains 188 graphs, depicting chemical compounds. In this representation, explicit hydrogen atoms have been removed. The (initial) vertex labels stand for example for Carbon (0), Nitrogen (1), Oxygen (2), Fluorine (3), Iodine (4), Chlorine (5) and Bromine (6). The edges labels indicate the bond type of either aromatic (0), single (1), double (2) or triple (3). The classifications in the dataset are made according to their mutagenic effect on a bacterium [11].

Notice how most, but not all these datasets have two class labels and recall, that datasets with high amounts of different class labels require bigger batch sizes in order to represent all classes equally. More information

[12]*The dataset *Tox21_AhR* is also called *Tox21_AhR_training*.

on the individual datasets can be found online.[13]

4.1.2 OGB Dataset

As mentioned above, many graph datasets from the TUDataset can be classified by using the No-graph kernel, which disregards graph structures [22]. Thus in order to evaluate graph kernels based on graph structures, there is a high demand for graph datasets, where the graph classification is highly dependent on the graph structure. In 2020 Hu, Fey, Zitnik, et al. presented the Open Graph Benchmark (OGB) [80]. This collection of more realistic benchmark graph datasets includes large-scale datasets from different domains. The collection includes graph datasets for vertex, link, and graph property predictions. The datasets are classified in datasets with small (100 thousand to 1 million vertices), medium (>1 million vertices) and large (on the order of 100 Mio. vertices or 1 billion edges) graphs [80].

The datasets *ogbg-molhiv* and *ogbg-molpcba* datasets were initially considered for experimentation. But their size of 41,127 and 437,929 graphs was to big for computations and storage on the used machines. Notice that for the evaluation, the distance matrix (and kernel matrix) is computed for every learned and stored edge weight vector. It is possible to continue the evaluation on a fraction of the matrices, or fractions of the datasets. Instead, the research was reduced to the smaller datasets, in order to execute and react to more experiments. The OGB collection provides ten smaller datasets from MoleculeNet [87]. However only the datasets *ogbg-molbace* and *ogbg-molbbbp* match the requirements of samples for binary (not multi-task) classifications.

Each graph represents a molecule, where vertices are atoms, and edges are chemical bonds. Input node features are 9-dimensional, containing atomic number and chirality, as well as other additional atom features such as formal charge and whether the atom is in the ring or not. The full description of the features is provided in code...

The leaderboard[14] of classification performance on these datasets is organized by ROC-AUC (Receiver Operating Characteristics-Area Under The Curve) scores. This quality measurement and evaluation process was not introduced to the implemented method for this thesis and the evaluation on these two datasets from the OGB collection was not be compared to the state of the art. Because the datasets were loaded and pre-processed differently than how the OGB framework proposes it, and in order to avoid introducing yet another evaluation strategy. Instead, these datasets are used only to track the performance against the l-WWLLT metric at its initialization, as described in the research question in section 1.2.

[13] TUDataset list `https://chrsmrrs.github.io/datasets/docs/datasets/`

[14] `https://ogb.stanford.edu/docs/leader_graphprop/#ogbg-molhiv`

4.2 Set Up

For all experiments a few parameter settings were fixed. One may change them for later research. These parameters are the update frequency, -score and -intensity. For all experiments, the **update frequency** is set to "epoch", which means that the edge weights were updated after computing the mean weight change for all edges and all graphs in the batch of one epoch. For all experiments, except the single layer training (see subsection **??**), the **update scope** is set to "All layers equal", which means that all layers are treated equally. One may use this parameter to change the update intensity for individual layers individually.

The implemented Edge Weight Learner has a few more parameters, whose values were changed during the experimentation phase. In the following passage I explain these parameters.

The **WLLT depth** refers to the deepest layer of the WLLT, whose edge weights will be changed. If it is set to 2 only the edge weights of the layers 0, 1 and 2 are updated (and used to define the update). The **learning rate** refers to the same parameter which was explained in section 3.3. The **batch size** is the percentage of graphs of the whole dataset that are used for the computation of the edge weight updates in one epoch. Notice that the actual set of graphs for each epoch is determined before all epochs, such that each class is represented almost equally. The **cluster Pull, Push factors** and **Heaviest Earth Threshold** refers to the same parameter which was explained in section 3.3. The flag **weights in** $[0, 2]$ indicates, if all edge weights were limited to the interval $[0, 2]$. This parameter was introduced to avoid exponentially increasing weights, since the standard weight update is relative and experiments show, that the same are updated predominantly in the same direction. That is predominantly increased, or predominantly decreased. Thus by nature of the relative updates, some weights may increase exponentially.

All experiments were implemented in Python 3.7.7. The reported runtimes were measured using an Intel Core i5-8300 CPU at 2.30GHz with 16GB of RAM using a single processor only.

4.3 Experiments

As described in section 3, several different parameters were used to configure the implemented l-WWLLT method. We use the following notations for these parameters:

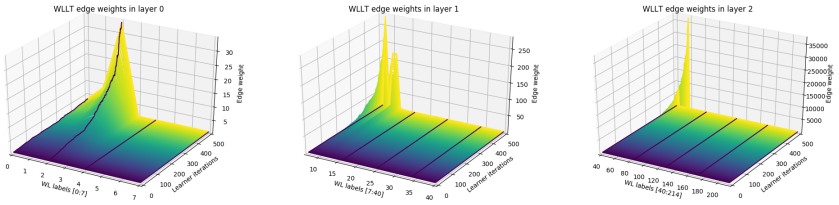
- WLLT Depth D
- Learning rate lr
- Batch size bs
- Push factor f_{push}
- Pull factor f_{pull}
- Heaviest earth threshold t_{he}

We indicate an imposed limitation of the edge weights by writing $w \in [0, 2]$, $w \in \mathbb{R}$ respectively.

First experiments were conduced to guide implementation choices and shape the evaluation process. This includes not the evaluation with respect the proposed SVM and cluster scores, but with respect to tracking and visualizing the computed edge weights and the resulting clustering. Most of these evaluations will be used in the next sections to explain the experiments and their interpretation.

4.3.1 WLLT weight

The initial configuration of the implemented learner did not include a weight limit. However the drawbacks of the multiplicative updates, in the context of the WLLT and the graph representations can be seen in the plots in figure 9. Each plot shows the edge weight distribution over all its WL-labels for all 500 learning epochs. The x-axes denote the WL-label (child to the considered edge), the y-axes the learning epoch and the z-axes the edge weights. The plots visualize how the (non-zero) development of most edge weights is overshadowed by an exponential growth of a few weights.



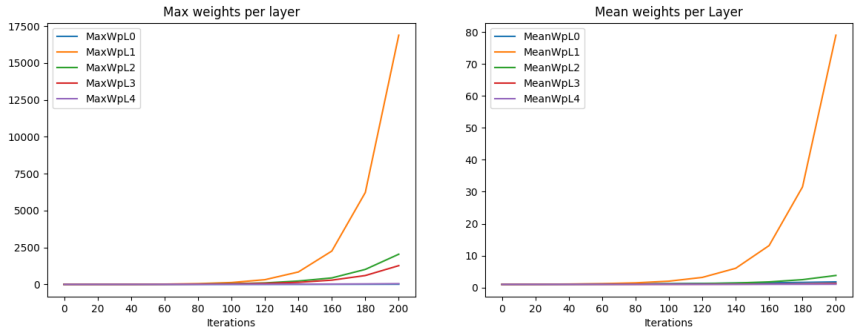
(a) Layer 0 (b) Layer 1 (c) Layer 2

Figure 9: Edge weights per WLLT layer - *MUTAG*

A weight limit (to the interval $[0, 2]$) was introduced in order to reduce this overshadowing effect and keep the edge weight values reasonable. The effect, that only a few edge weights are updated most frequently and strongly remains, and is discussed in the following paragraph.

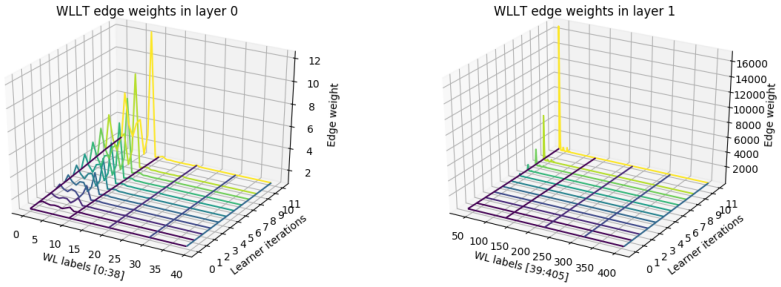
Redemption of unlimited weights The idea was, to limit the edge weights to the interval $[0, 2]$. As we can see in the descriptions of the other experiments, it was difficult to find a configuration for the implementation, such that the edge weights do not clip to the imposed ceiling and also do not vanish. Both results were observed. Since no reliable improving configuration was found, the edge weight adjustment may have been to coarse. This assumption can be supported by a rather positive result, when lifting the edge weight constraints on a learning setting with different parameters.

The plots in figure 10 show how the maximum edge weight and the mean edge weight grow exponentially in the number of learning epochs. Similarly, the plots in figure 11 highlight the domination of individual edge weights in the layers 0 and 1. These wire-frame plots are a great tool to compare the development of all edge weights, per layer. Noticeable in all experiments is, that especially in the higher lower layers, the edge weight per layer is concentrated on a small portion of WL-labels. These are the most common WL-labels and dominate the graph representations especially in the lower layers. With each layer, the differentiation of the vertices (their WL-labels and thus their k-th neighborhoods) increases and the frequencies of all WL-labels decreases. Accordingly, the strength of the update rule on these WL-labels decreases. The domination of the weight of the "lowest edge" in figure **??** however is unusually high.



(a) MaxWpl (b) MeanWpl

Figure 10: Maximum and mean weights per layer - *AIDS*

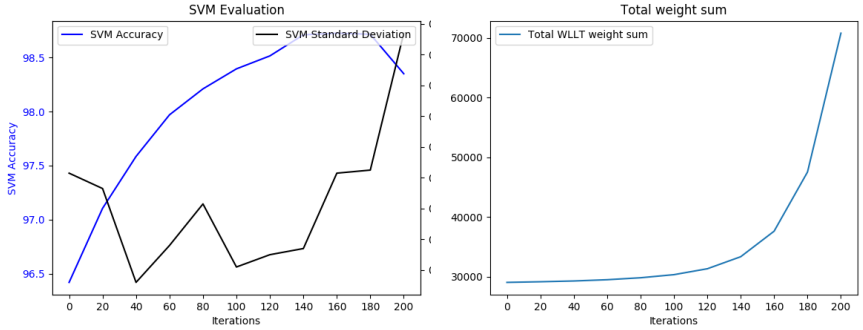


(a) Layer 0 (b) Layer 1

Figure 11: Edge weights per layer, per iteration - *AIDS*

Despite this exponential edge weight increase, which was deemed un-

desirable in the early experiments, the evaluation shows how the SVM accuracy improves in this setting. The exponentially increasing total weight in the WLLT, shown in figure 12b and the almost monotonically improving SVM accuracy in figure 12a indicate, that more research on different settings need to be considered. The so far used update increments may not balance the pushing and pulling effect of the clusters as desired, but even exponentially increasing edge weights can lead to a graph similarity measure, which allows a good classification with a SVM.



(a) SVM (b) Total weight sum

Figure 12: SVM and total weight sum - *AIDS*

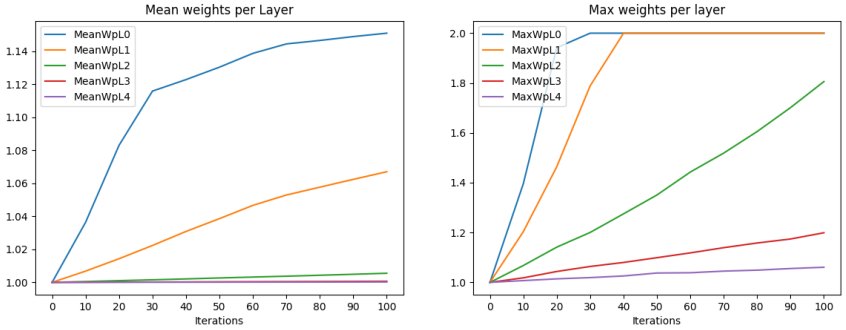
4.3.2 Artificial dataset

To test the correctness of the implemented method an artificial dataset was created. This dataset shall represent a perfect separation of the classes with respect to the l-WWLLT with uniformly initialized edge weights. Therefore the 2000 graphs of the dataset **AIDS** were subdivided in class A, consisting of the first 1000 graphs and class B, consisting of the second 1000 graphs. On top of that, all occurring vertex labels for the graphs in class B were changed to new vertex labels, which did not occur in class A. This implies, that the WLLT T can be separated in vertices, representing WL-labels which occur in exactly one class as well. Even further, the set of original labels (in the first layer) can be separated into vertices belonging to graphs of exactly one class, and all WL-labels in its subtree belong to the same class. We refer to this artificial dataset as *AIDS_perfect*.

Now notice, that the update rule still may increase and decrease all edges. However we may expected (depending on the dataset) that the edges in the first layer are increased over time, or at least less decreased, than all other edges.

The plots in figure 13 refer to an execution with the settings $D = 5$, $lr = 1.0$, $bs = 5\%$ (100 graphs), $f_{\text{pull}} = f_{\text{push}} = 0.1$, $t_{\text{he}} = 0.6$ and $w \in [0,2]$. Figure 13a shows the mean weights per WLLT layer (MeanWpL). It is no-

ticeable, how the mean weight of the first layer increases more, than the one of the other layers. However this should be considered with the fact, that the other layers are much larger, and the graph representations contain lower values (normalized frequencies) for these WL-labels in general. Therefore it is more difficult to changer their mean significantly during the learning procedure. Figure 13b does not debunk the argumentation as well. It shows the maximum weights per layer (MaxWpL) and that these increase more rapidly in the lower layers.

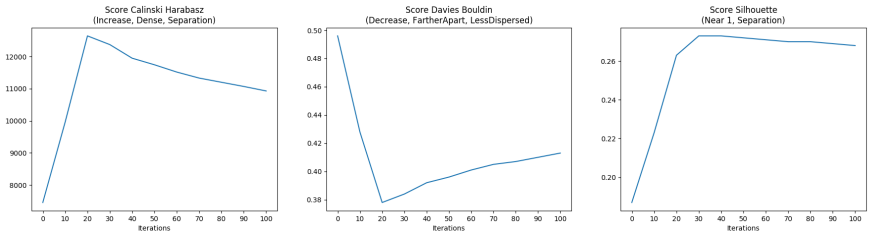


(a) MeanWpL (b) MaxWpL

Figure 13: Mean and Maximum weight per layer - *AIDS_perfect*

Notice how the limitation of the edge weights ($w \in [0,2]$) is visible in both graphs at around epoch 30. Figure 13b visualized, how the maximum edge weights in the zeroth and the first layer are no longer increased and stay at the maximum of 2.0. In figure 13a the growth of the mean weights per layer is decreased afterwards. Recall, that the update rule allows to update edge weights with the same value in the same way. Thus these observations cannot be related to single edge weights in general.

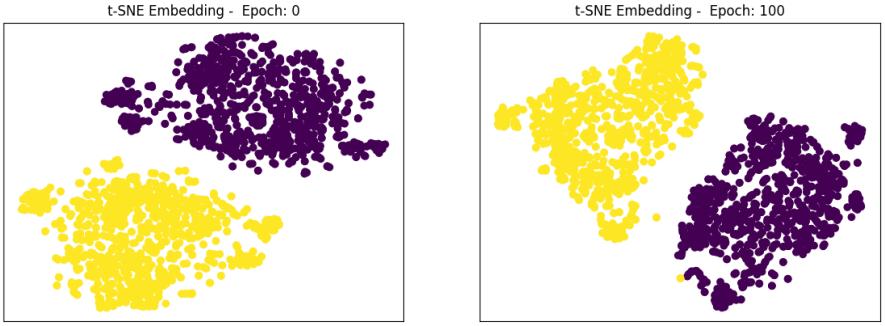
The plots in figure 14 show the evaluations on the Silhouette coefficient (SS), Davies-Bouldin score (DBS), and Calinski-Harabasz score (CHS). They reflect on the reached maximal edge weights as well. Up to epoch 20 or 30 they indicate an improvement. After this, the cluster dispersion (caused by overall increasing weights) may be the reason to degenerating scores.



(a) CHS (b) DBS (c) SS

Figure 14: Cluster scores - *AIDS_perfect*

A T-distributed Stochastic Neighbor Embedding (t-SNE) [78] visualize the separation of the clusters in figure 15 Overall, the clusters stay rather separated. An in- or decrease of the distance between them or the cluster denseness can not be noticed. However keep in mind, that the plots show a two-dimensional embedding of high dimensional (sparse) feature vectors. In this case, there are 34,296 WL-labels in the WLLT (with five layers $D = 5$). Thus the feature vectors are 34,296-dimensional.



(a) Epoch 0 (b) Epoch 100

Figure 15: T-SNE embeddings - *AIDS_perfect*

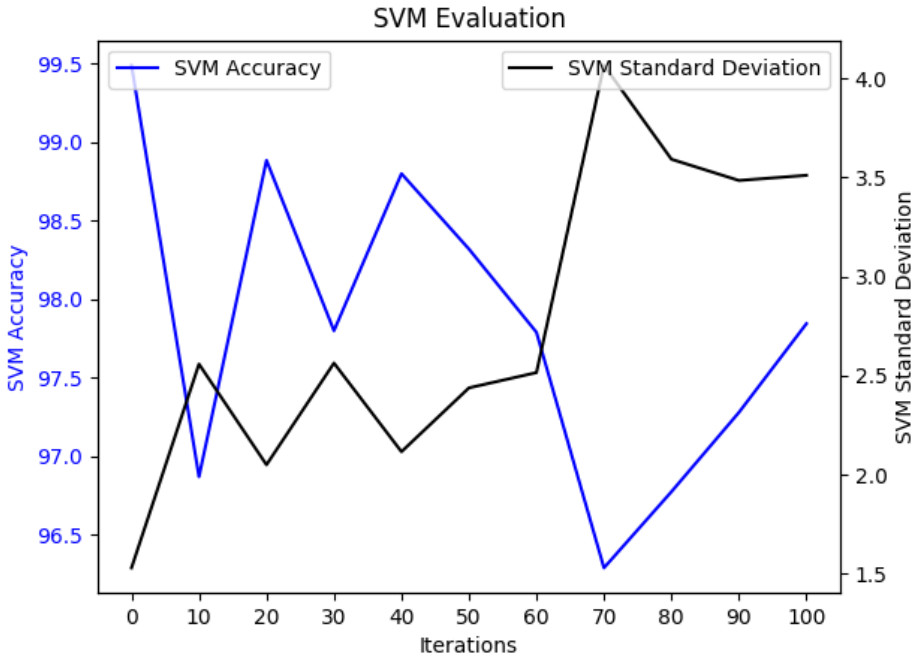


Figure 16: SVM - *AIDS_perfect*

On the other hand figure 16 illustrates that the SVM accuracy does not directly relate to these statistics. Overall, the SVM accuracy decreases and the best accuracy is reported for the initial state. This is a first indication that the cluster scores and the SVM accuracy do not need improve or degenerate simultaneously.

To summarize, this experiment supports the claim that the WLLT and the graph representations can be used in general to reflect a separation of graph clusters.

4.3.3 Batch Size, WLLT Depth and Number of Epochs

Batch Size The number of epochs is closely related to the batch size in the sense that both parameters relate positively to the possibilities of introduced or removed weight in the WLLT. However the batch size is a goes quadratically into the runtime and thus should be kept low if possible. Furthermore, the evaluation is after each batch and since the weight updates are applied after each batch, a smaller batch size allows for a finer evaluation. That is why in general the batch size was kept at 50 to 200 graphs (1%-30%) for the datasets, depending on the size of the dataset and sizes of the graph classes.

WLLT Depth The depth of the WLLT has a significant impact on the runtime as well (and on the required storage). Although as discussed, the size of the WLLT has linear effect on the runtime, the WLLT depth may have exponential effects. That is because the amount of WL-labels increases drastically, depending on the dataset (recall figures 6b and 6a). Table 5 in the appendix (A.1) lists the WLLT size up to layer four for a selection of datasets. Figure 17 visualizes this table.

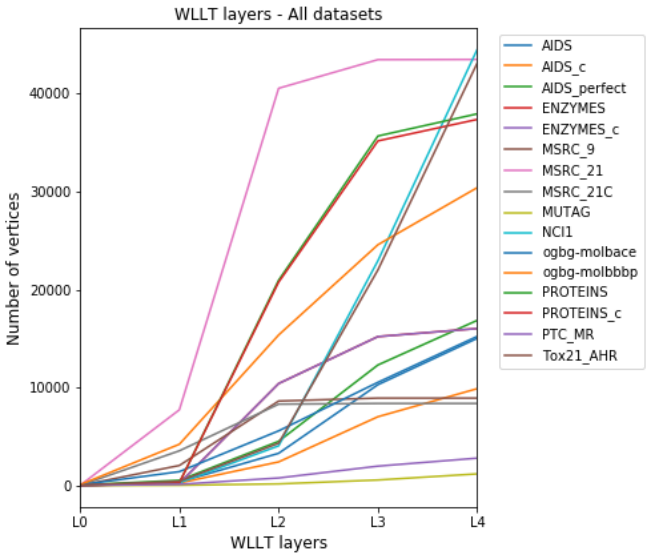


Figure 17: Size of the WLLTs for various datasets.

Consider a graph representation of a graph G, and a WL label l which is a leaf in the WLLT and does not occur in the labelings for G. Since l is present in the WLLT it is present in the dataset and adding a next WLLT layer will add at least one child vertex to l. Depending on the depth and the dataset, l may get many more children and all these do not occur in the labeling for G. Thus notice, how higher layers imply a more sparse graph representation. After conducting some initial experiments with smaller datasets for WLLT depths of up to 10, the WLLT depth was kept at 3 or 4. Higher layers do not seem to improve the resulting similarity measure.

Number of Epochs To determine how many epochs a learning algorithm should compute is usually related to the task of detecting over-fitting. For efficiency reasons, the evaluation and the learning procedure were separated, which is why an automatic detection of over-fitting was not implemented. Instead, the experiments were focused on collecting data for several parameter settings, and possibly detect over-fitting in hindsight. For a more granular evaluation the implementation saves intermediate results (a

weight vector for the WLLT) every 10-th epoch.

The highest number of epochs was 1000. For the dataset *AIDS* with 100 graphs per batch this took approximately 7.5 seconds per epoch (75 milliseconds per batch) and a total runtime of approximately two hours. The overall performance of the learned similarity measure did not show a clear relation (improvement or degeneration) with the number of epochs. Since all evaluations show noticeable development in the first 200 epochs, this was used as the standard number of epochs for further experiments. See figure 18 for example. The visualized Calinski-Harabasz cluster score only changes significantly for the dataset *PROTEINS_c*. All other trajectories of the plots were maintained for most of the experiments for further learning epochs. No clear relation to other performance criteria (e.g. SVM accuracy or SME) of the l-WWLLT was detected.

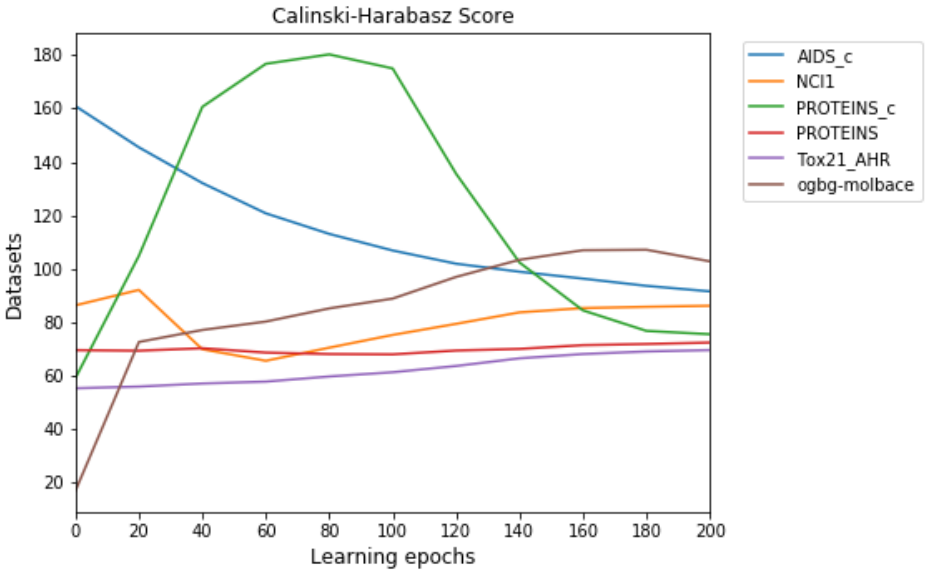
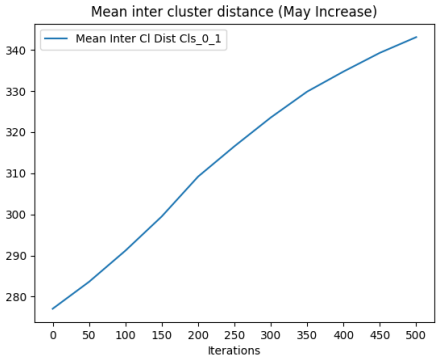
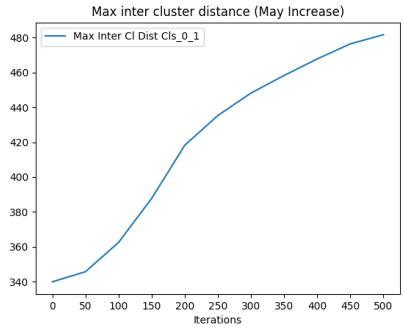


Figure 18: Calinski-Score for several different datasets.

The following plots for the dataset *ogbg-molbace* are exemplary visualizations for these observations. They correspond to a run with learning epochs $e = 500$, WLLT depth $D = 5$, batch size of 100 graphs, learning rate $lr = 1.0$ and push and pull factors of 0.2 each with limited edge weight in the interval $[0, 2]$ (figures 19, 20, 22, 25, 24, 23, and 21).



(a) Mean cluster distance



(b) Maximum cluster distance

Figure 19: Mean and Maximum cluster distances - *ogbg-molbace*

The dataset *ogbg-molbace* contains graphs, which are grouped into two clusters (0 and 1). The two plots in figure 19 show the mean and maximum distance between these classes (inter-distance). As desired, these distances increase over time. Round epoch 200, a slightly but rather insignificant reduced growth of the maximum distance can be noticed.

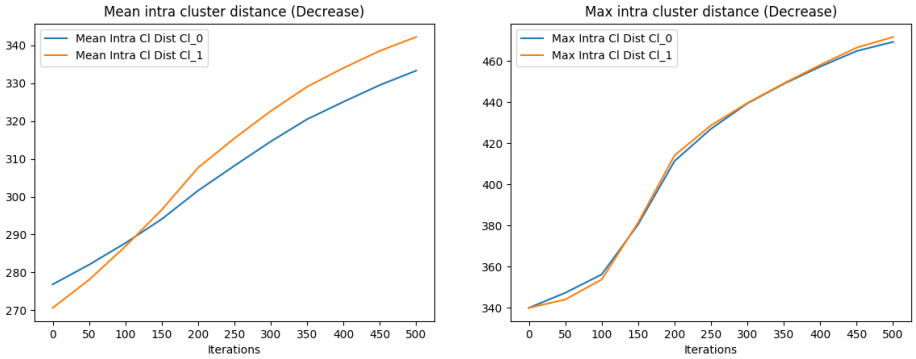


Figure 20: Intra-cluster statistics - *ogbg-molbace*

The two plots in figure 20 show the mean and maximum distance between samples of the same class each (intra-distance). Again around epoch 200, the reduced growth of the maximum distance can be noticed. One may as well describe it as a significant increase in the epochs 100 to 200. In contrast to the distances between different clusters however, the desired behavior would be decreasing intra-distances instead. Also notice that the order of magnitudes are equal to the inter-distances.

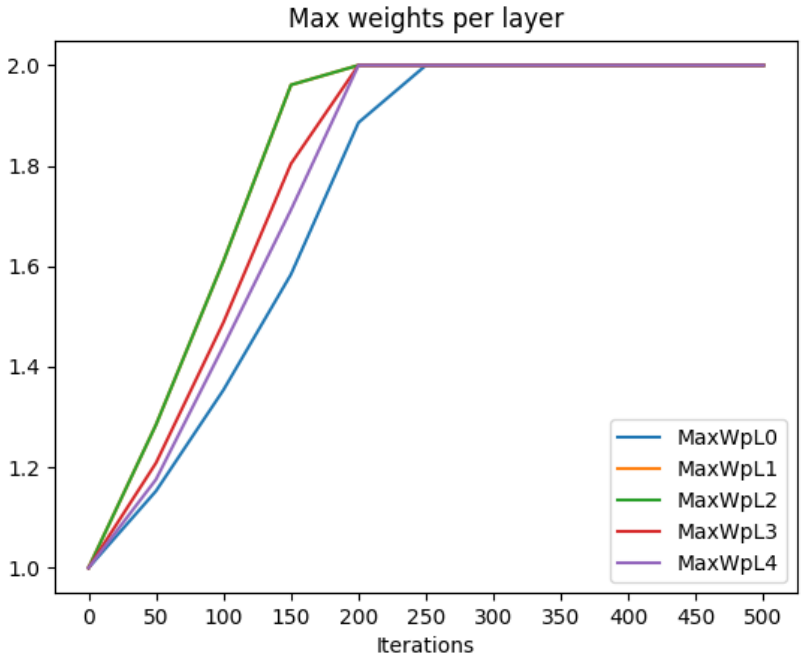


Figure 21: Maximum weights per WLLT layer - *ogbg-molbace*

The plot in figure 21 gives a good explanation for the observed statistics up to epoch 200. It shows the maximum edge weights, to each of the five layers in the WLLT. Around epoch 200 the maximum edge weight for every epoch has reached the allowed limit, and stays there. Recall, that the update rule may treat old weights of the same value equally. One may assume, that after epoch 200 more and more edge weights are set to the maximum of 2.0 and no longer decreased.

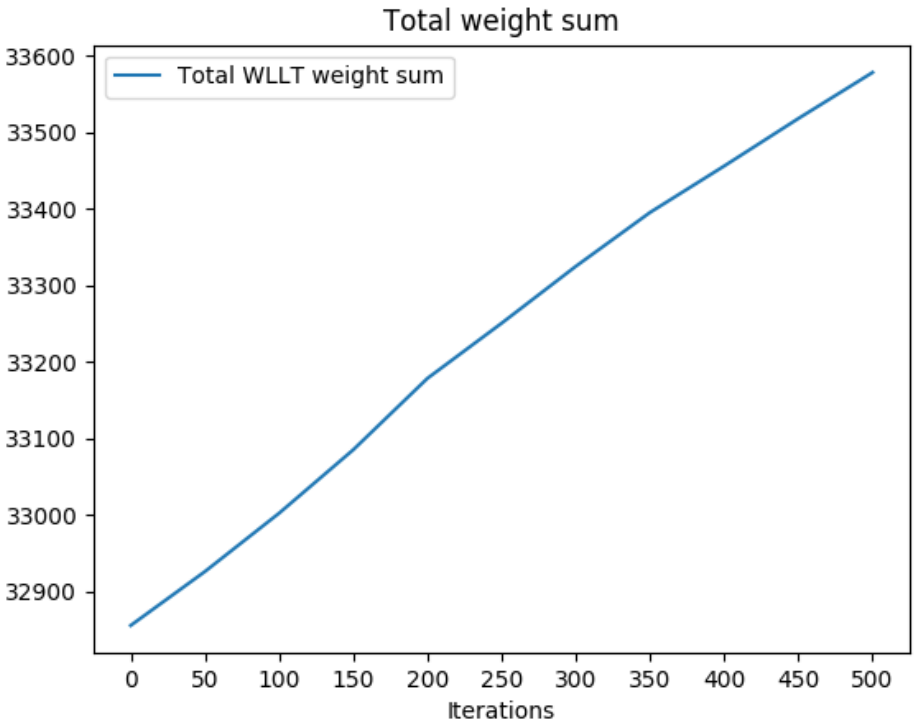


Figure 22: Total WLLT weight - *ogbg-molbace*

These observations are supported by figure 22, which shows that the overall weight in the WLLT grows monotonically.

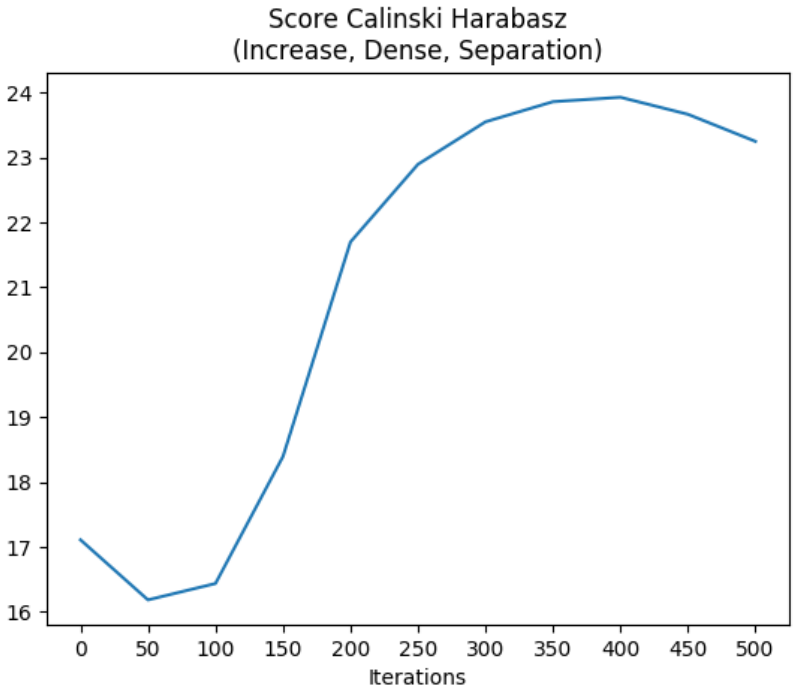


Figure 23: Calinski Harabasz score - *ogbg-molbace*

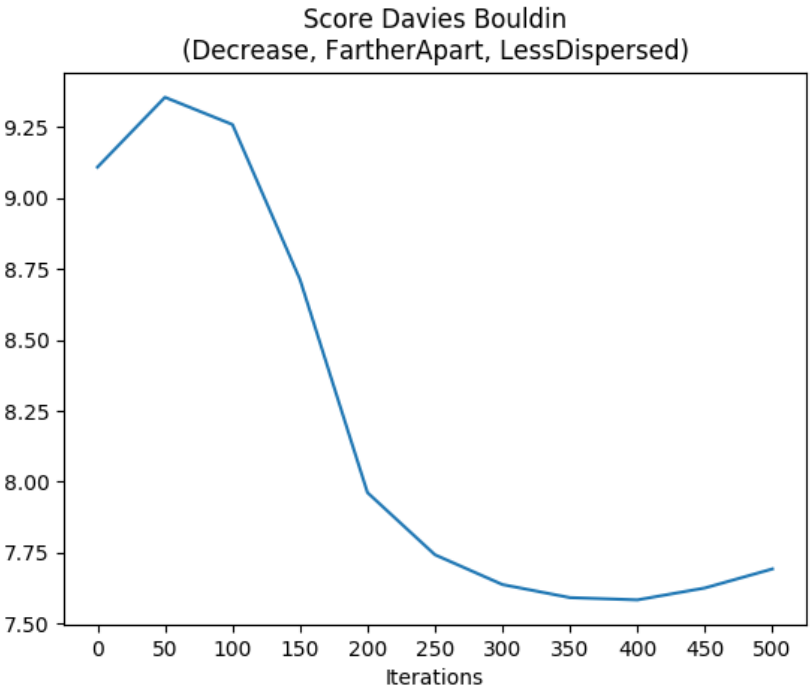


Figure 24: Davies Bouldin score - *ogbg-molbace*

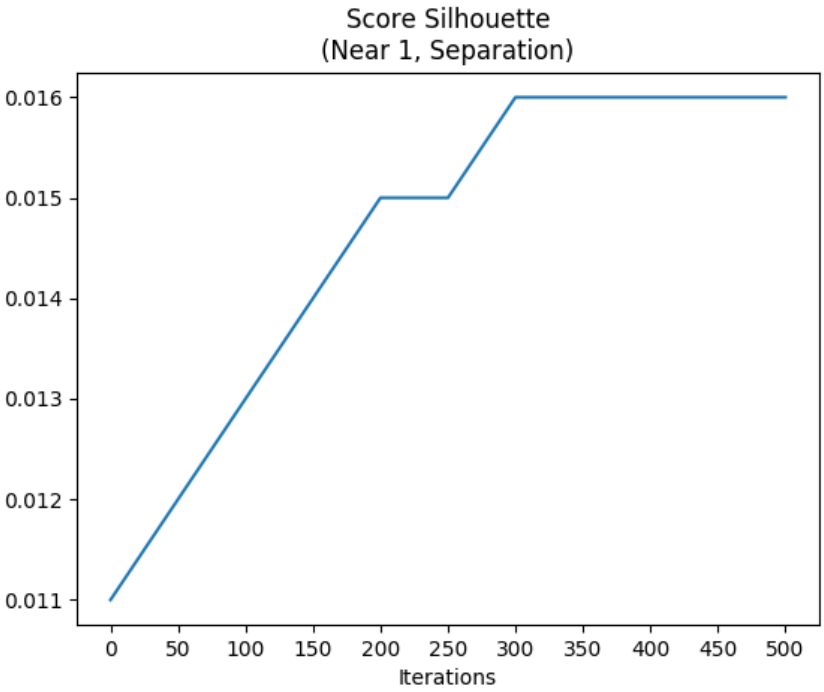


Figure 25: Silhouette score - *ogbg-molbace*

The cluster scores (see figures 23, 24 and 25) indicate, that this edge weight maximum reduced the quality of the clustering. After epoch 200 all three scores indicate a reduced improvement. The Calinski-Harabasz score corresponds to the denseness of the clusters and their separation. The improvement can be traced to the improved separation. The analogue situation can seen in the plot of the Davies-Bouldin score. The Silhouette Score stagnates after epoch 300 (at a low level). Since it corresponds to the separation, and not directly to the denseness of the clusters, one can deduce, that the degeneration of the other two scores can be explained by a degeneration of the cluster denseness. Combined with the insight from figure 21 we can deduce, that the chosen setting leads to a reliable increase of the inter-distances, but not of the intra-distances and thus decreases the denseness of the clusters.

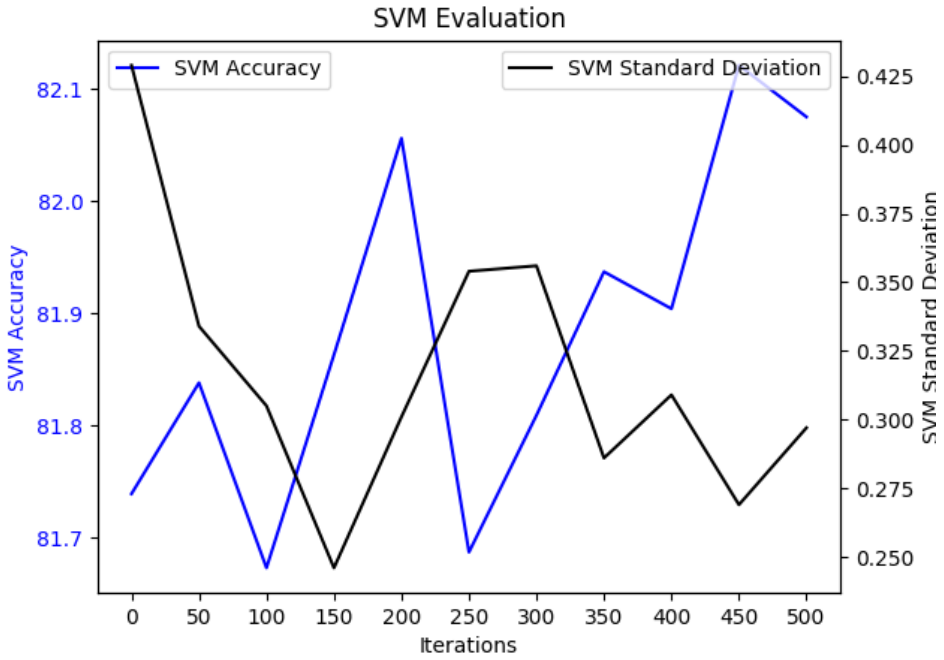


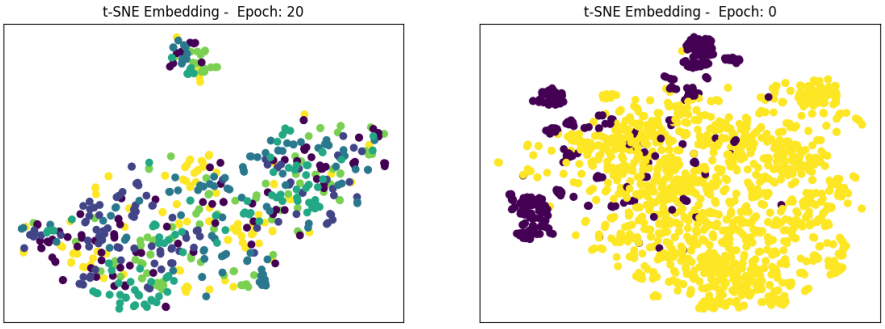
Figure 26: SVM accuracy and standard deviation

Interestingly, the SVM accuracies (shown in figure 26) do not obviously relate to the other plots. The SVM accuracy fluctuates at almost 82%.

4.3.4 Relation between Pushing and Pulling

The success of the learning process seems to behave quite sensitively to the push and pull factors. This includes their absolute strength and their relation between each other. Intuitively, if the pull factor is to big, the clusters collapse into each other. If the push factor is to big, the dispersion of the clusters increases which may lead to overlapping clusters.

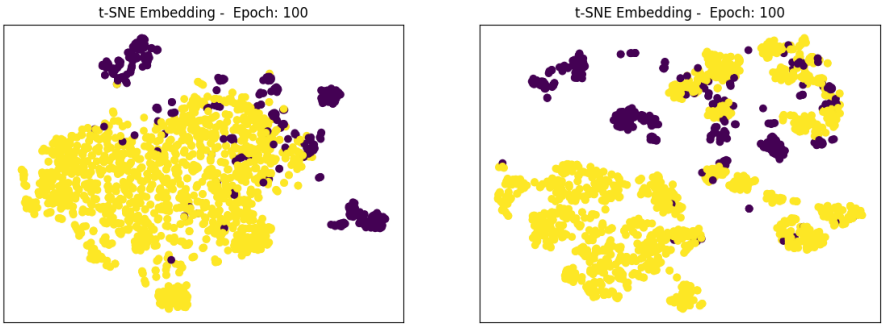
Researching the effect of the push and pull factors and initializing them meaningfully becomes even more challenging, when different datasets are involved. Figure 27a and 27b show a t-SNE for the datasets *ENZYMES* and *AIDS* respectively. Notice how the different clusters overlap to different extends.



(a) *ENZYMES* (b) *AIDS*

Figure 27: t-SNE of the uniformly initialized l-WWLLT

The following four t-SNE show the implied l-WWLLT distance after 100 learning epochs. In all configurations, five WLLT layers were used. If not stated otherwise, the plots refer to the multiplicative update rule and $lr = 1.0$.

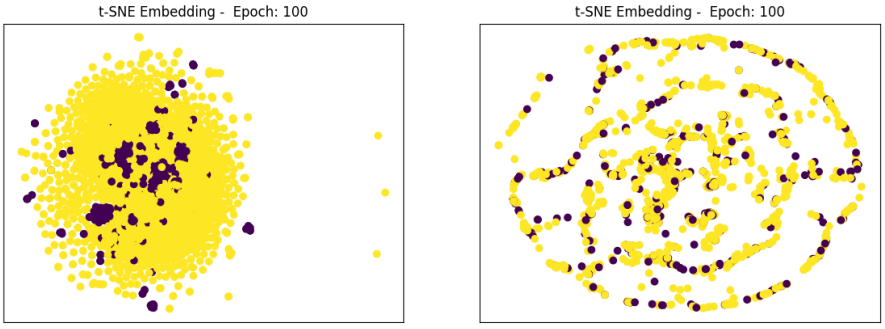


(a) pp-Factors: 0.3 (b) pp-Factors: 0.5

Figure 28: t-SNE of the l-WWLLT distances after 100 epochs. Equally strong push and pull factors. - *AIDS*

Figure 28a shows a configuration with $f_{\text{push}} = f_{\text{pull}} = 0.3$ ($lr = 0.9$). Almost no change to the initial state (figure 27b) can be seen. One could argue that the clusters appear denser, but certainly not better separated.

Figure 28b shows a configuration with $f_{\text{push}} = f_{\text{pull}} = 0.5$. Again, the pp-factors are equally strong. But their intensity (including the learning rate) is higher. Denser sub-clusters form in both clusters but the cluster separation remains weak.

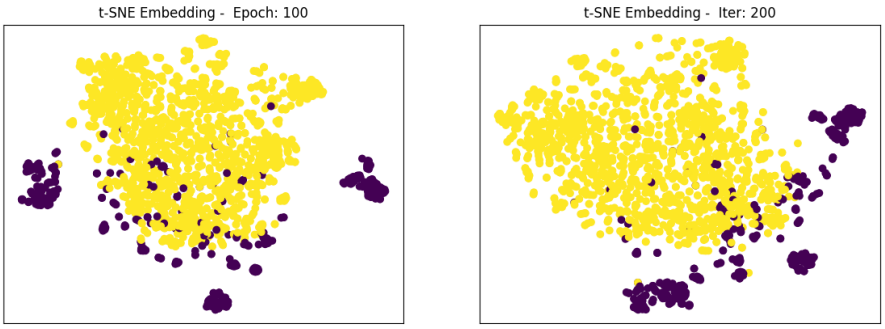


(a) Pull: 0.1, push: 0.05 (b) Pull: 0.1, push: 0.05 (absolute)

Figure 29: t-SNE of the l-WWLLT distances after 100 epochs. Stronger pull. - *AIDS*

Figure 29a shows a configuration with $f_{\text{push}} = 0.05$ and $f_{\text{pull}} = 0.1$. This is a good example for a typical case of heavily overlapping clusters because the pull factor is stronger that the push factor.

Figure 29b shows a configuration with $f_{\text{push}} = 0.05$ and $f_{\text{pull}} = 0.1$. Here, the pp-factors are used in the additive update rule. Interestingly the pull factor seems to pull again samples from both the same and different clusters together (compare to figure 29a). However a significant separation is visible place at well. But such that the different clusters are separated.



(a) Pull: 0.1, push: 0.2 (b) Pull: 0.1, push: 0.3

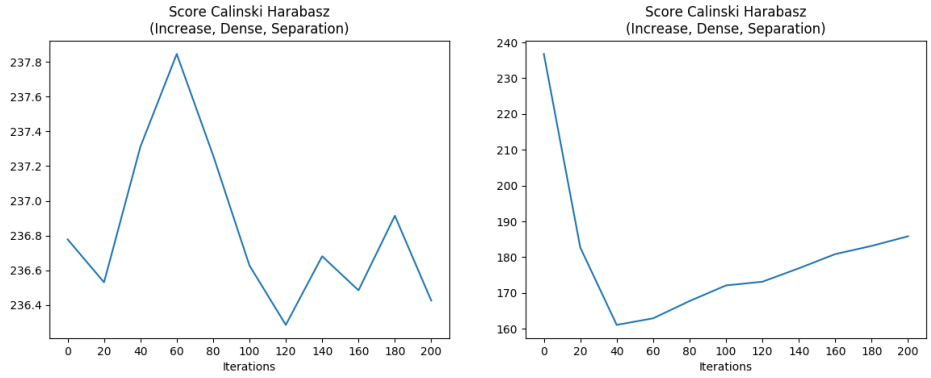
Figure 30: t-SNE of the l-WWLLT distances after 100 epochs. Stronger push. - *AIDS*

Experiments, where the push factor is bigger than the pull factor tended to look like the original configuration, similar to figure 28a again. Two examples of a common configuration from within a grid-search on the pp-

Factors are visualized in figure 30.

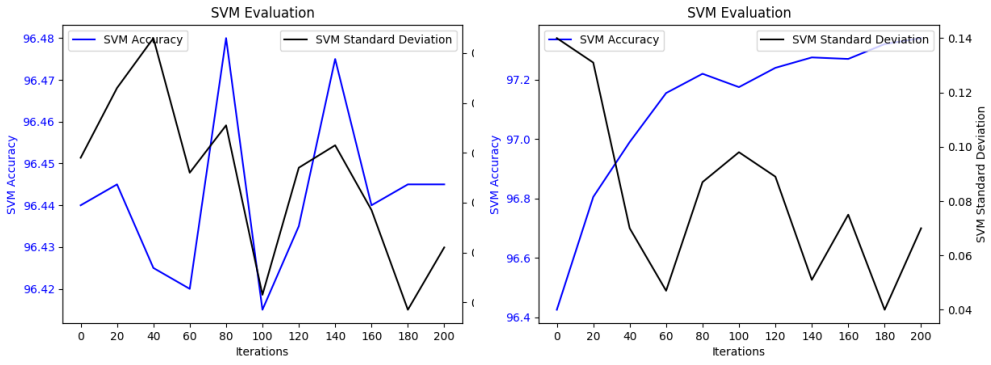
4.3.5 Arbitrary Classifications

In this experiment the classification for the graphs in the given dataset were changed. The performance of the leaner is far less predictable. Since the original classifications are related to the structures of the graph, but the changed ones are most likely not related to the structures of the graph, this indicates that the learned edge weights do indeed capture more information about the graph structures.



(a) Randomized classes. (b) Not randomized classes.

Figure 31: CHS for the arbitrary classifications experiment. - *AIDS*



(a) Randomized classes. (b) Not randomized classes.

Figure 32: SVM for the arbitrary classifications experiment. - *AIDS*

4.3.6 Other Experiments

A few other experiments were made, but yield a similarly vague results when it comes to predicable behavior in the learning process. They are briefly mentioned in this section.

Single Layer Training The implementation allows to update edge weights of different layers differently. So far, the other experiments do not motivate to use this. For experimental variety however, a couple of experiments were conduced, such that only the edge weights of a single layer in the WLLT were updated. Spot checks for different datasets and different WL layers did not show promising results. While it was still possible to improve the cluster scores, the SVM performances were far behind the ones when updating all layers equally.

One may deduce from this, that the similarity of the graphs should be expressed as a matter of substructures of different sizes. Since the WL-labels in layer i only correspond to the i-1-th neighborhoods. Adjusting the weights in different layers may allow to distinguish more accurately between variations in substructures of different sizes, which are relevant for the classification.

5 Results

The experiments and the results presented in section 4 are centered around the research question, of realizing the l-WWLLT method in a way, that leads to a (predictable) improvement of the resulting similarity measure. As stated in the introduction, the main goal of the thesis is to elaborate, if the proposed method can improve over the initial state. We reflect on the research question in section 5.2. However the evaluation of the proposed method may be considered incomplete, if the results are not also compared to the performance of other kernels. A short comparison of the best results is presented in section 5.1. Finally we reflect on the presented results and thus the l-WWLLT method in section 5.3.

5.1 Comparison to other Kernels

As stated before, performances of the datasets *ogbg-molbace* and *ogbg-molbbbp* of the OGB collection are generally measured differently, which is why we exclude them in this comparison. The same goes for the cleaned versions of the TUDatasets. The best reported SVM accuracies on these dataset are however included in table 4. Tables 2 and 3 belong together and list reported SVM accuracies for a variety of kernels and datasets.

The kernels in tables 2 are the l-WWLLT Kernel as proposed in this thesis, the *No-graph Kernel* (**NoG**) [44], the *Probabilistic Frequent Subtree Kernel* (**PFS**) [45], the *Boosted Probabilistic Frequent Subtree Kernel* (**BPFS**) [45], the *Frequent Subgraph Kernel based on FSG* (**FSG**) [32], the *Graphlet Sampling Kernel* (**GS**) [33], and the *Random Walk Kernel* (**RW**) [25]. The results for kernels other than the l-WWLLT are reported by Schulz and Welke [44]. The entry '$\approx$' indicates a SVM accuracy comparable to the one reported for the NoG Kernel [44].

The kernels in tables 3 are again the l-WWLLT Kernel for comparison, the *Weisfeiler-Leman Optimal Assignment Kernel* (WL-OA) [20]. the *Weisfeiler-Leman Kernel* (WL) [10], and the *Wasserstein Weisfeiler-Leman Kernel* (WWL) [21]. The results for kernels other than the l-WWLLT are reported by Kriege, Giscard, Computer, et al. [20]. They also reported results on the *Vertex Kernel* (V), which is based on the dot products on vertex label histograms, the *Edge Kernel* (E), which is based on the dot products on edge label histograms (edge labels as the set of labels of its endpoints), the *Vertex Optimal Assignment Kernel* (V-OA) [20], and the *Edge Optimal Assignment Kernel* (E-OA) [20]. These four kernels (V, E, V-OA, E-OA) all performed worse than the reported results of the WL-OA on these datasets [83], which is why we omit them here. The SVM accuracy of 93.36% for the WL Kernel [10] on the dataset *Tox21_AHR* was reported by Schulz and Welke [44].

The results for the **Std WL** are based on own experiments with the *WL Kernel* (WL) [10] and an implementation provided by the GraKel frame-

work [42]. The results are visualized in the plot 34 in the appendix. [15]. The attached column D indicates for which WL iteration $i \in [1, 10]$ the highest accuracy was reached.

Dataset	l-WWLLT	NoG	PSF	BPSF	FSG	GS	RW
AIDS	98.73	**99.65**	98.25	98.45	97.85	≈	≈
ENZYMES	59.04	43.33	28.33	32.00		30.50	17.33
MSRC_9	**90.83**	88.35	≈	≈		25.09	11.73
MSRC_21	**87.58**	86.46	51.84	52.22	46.79	14.60	5.05
MUTAG	**87.33**	87.31	≈	≈	≈	≈	≈
NCI1	79.44	69.93	≈	74.33	76.28	62.68	
PROTEINS	73.57	74.58	≈	≈		≈	
PTC_MR	**64.92**	57.60	≈	≈	≈	≈	≈
Tox21_AHR	88.37	**90.89**	88.47	88.36	88.37	88.38	

Table 2: SVM accuracies for various kernels (1/2)

Ds	L-WWLLT	WL-OA	WL	WWL	Std WL	D
A	98.73				98.54	1
E	59.04	**59.9**±1.1	53.7±1.4	73.25±0.87	53.95	3
M_9	**90.83**					
M_21	**87.58**					
MU	**87.33**	84.5±1.7	86.0±1.7		83.85	10
N	79.44	86.1±0.2	**85.8**±0.2		85.09	8
P	73.57	76.4±0.4	75.6±0.4	77.91±0.80	**75.63**	10
P_MR	**64.92**	63.6±1.5	61.3±1.4			
T	88.37		93.36			

Table 3: SVM accuracies for various kernels (2/2)

We can report a better SVM accuracies for the datasets *MSRC_21*, *MSRC_9*, *MUTAG*, and *PTC_MR* compared to all other enlisted kernels. On the datasets *AIDS* and *Tox21_AHR* the SVM accuracies are comparable to the best accuracies among all other kernels. On the dataset *NCI1* we report a significantly worse accuracy compared to the best listed accuracies of the WL-OA Kernel and the WL Kernel.

The best improvements over other reported accuracies are achieved on the datasets *ENZYMES* and *PTC*.

Table 4 contains the configurations for the best SVM accuracies over all experiments. The columns denote the configuration in the same notation used in the section 4.3. bs_p and bs_a denote the batch size as percentage and

[15] Using a vertex histogram as base kernel. See `https://ysig.github.io/GraKeL/0.1a8/generated/grakel.GraphKernel.html#grakel.GraphKernel`.

Dataset	D	lr	bs_p	bs_a	f_{pull}	f_{push}	t_{he}	SVM	E
AIDS ($w \notin [0,2]$)	4	1	5%	100	0.50	0.50	0.9	**98.73**	160
AIDS_c	4	1	5%	56	0.10	0.50	1.0	**97.75**	200
AIDS_perfect	4	1	5%	100	0.10	0.10	0.6	**99.49**	0
ENZYMES	4	1	5%	30	0.10	0.50	1.0	**59.04**	140
ENZYMES_c	4	1	5%	29	0.10	0.50	1.0	**59.06**	180
MSRC_9	4	1	30%	66	0.10	0.40	1.0	**90.83**	40
MSRC_21	4	1	30%	169	0.20	0.20	1.0	**87.58**	160
MUTAG	1	1	5%	9	0.01	0.01	1.0	**87.33**	160
NCI1	4	1	5%	206	0.10	0.50	1.0	**79.44**	200
ogbg-molbace	2	1	7%	100	0.20	0.20	1.0	**83.56**	160
ogbg-molbbbp	4	1	5%	100	0.20	0.20	1.0	**84.30**	200
PROTEINS	4	1	5%	56	1.00	1.00	0.6	**73.57**	80
PROTEINS_c	4	1	5%	49	1.00	1.00	0.6	**72.87**	160
PTC_MR	4	1	30%	103	0.40	0.40	0.6	**64.92**	20
Tox21_AHR	4	1	1%	82	0.10	0.10	0.6	**88.37**	0

Table 4: Best SVM accuracies for the l-WWLLT Kernel

absolute number respectively. Note that unlike the others, the first row (for the *AIDS* dataset) reports on an execution where the edge weights where not limited to the interval $[0, 2]$. The last column E indicates, in which learning epoch this best SVM accuracy was reached. Depending on the datasets, the reported accuracies are the best in 200, 500, or 1000 learning epochs. Since the set of used parameters varies between these evaluations, and since only every 10-th epoch for every experiment was evaluated, we omit a detailed comparison of the SVM accuracies with respect to the number of learning epochs. As stated in section 4.3, no relation between these values as found.

It is important to notice, that this last column reveals, that the best SVM accuracies were achieved after only a few learning epochs. While most of the experiments aimed at improving the cluster statistics and the SVM accuracies steadily and reliable over many epochs, very good accuracies were reached early in the learning process.

It should be emphasized that no grid search was performed for all parameter settings and no clear reliable trend in performance could be discovered for all datasets. These results indicate that the method has potential and the impact of parameter settings should be further investigated.

5.2 Answering the Research Questions

The research question of this thesis (see section 1.2) is, whether the l-WWLLT method can improve the resulting graph similarity measure over its defi-

nition at the initialization. As presented in section this question can be answered positively.

The evaluations of the proposed graph similarity measure are not unanimous. Experiments and configurations for both SVM improvements and degeneration of the clusterings, and the other way around are presented. Furthermore, a variety of different implementation choices (via parameter settings) are presented. No clear winning or losing strategy could be identified.

5.3 Reflection

The experiments shown in section 4 may indicate, that a lot of different parameter settings and variations for the l-WWLLT method can be constructed. One may notice, that the initialized edge weights in the WLLT already imply a good similarity measure. Recall for example the SVM accuracy for the *AIDS* dataset, arising from the uniformly initialized WLLT edge weights of 98.56% for $D = 3$ depicted in figure 12a and of 96.42% for $D = 5$ depicted in figure 12 (or see results in the table in section A.3 of the appendix). This is comparable to accuracies of more sophisticated kernels like PSF, FSG, GS, RW and WL [44]. Although note, that the graph structure ignoring NoG Kernel already outperforms these accuracies with a reported accuracy of 99.65% [44].

The update of the edge weights was often concentrated on a few edges only, and lead to a rather monotone improvement or degeneration of some evaluation methods. Thus it would be useful to justify the decision of predominantly updating the "heaviest earth". For example by randomizing the candidate selection in the update process. Similarly, a randomized initialization of the edge weights in the WLLT could be used to test the effectiveness of the chosen update method further.

Although in the experiments, some configurations could be identified, which improved the resulting SVM accuracy, no systematic strategy to find such a configuration was found. We conclude that the implemented update method does not necessarily improve neither the (global) SME, nor the presented cluster scores, nor the SVM accuracy of the resulting kernel. That more research on the existing implementation and variations of it should yield a strategy when applying the l-WWLLT method.

6 Outlook

As argued in section 5.3, the l-WWLLT shows potential. Research on more datasets, parameter settings and implementation choices may prove useful. Using the experience from developing the l-WWLLT method, and the conducted experiments, we can propose new research plans in hind sight. This section contains such ideas, on how to continue the research with the l-WWLLT method.

Controlled distance improvements In order to better understand the conditions under which the local SME reduction translates to a global SME reduction, one may construct artificial datasets similarly to the proposed *AIDS_perfect* (section 4.3.2). Possibly with an increasing intensity of "imperfections" in the classification and the structure of the WLLT. These datasets should also be used to find general conditions under which the implied similarity measure is improved or at least maintained over a high number of learning epochs. In theory no degeneration should occur which is not corrected in the next epochs.

Other parameters As mentioned, a non-uniform treatment of the WL-layers could be implemented. This may include updates in only a selected number of WL-layers, or different strength of the weight updates for the layers. One may formulate this as a **layer gradient**. It could be implemented for example linearly, exponentially and in relation to the size of the layers.

Another interesting approach is to change parameter settings during the learning epochs. For example with respect to the push- and pull-factors. This resembles the approach of Simulated Annealing, which has been useful in many applications.

The proposed l-WWLLT method only works with categorical vertex labels. To be applicable to more datasets and to allow comparison with other kernels, it might be helpful to develop a method to extend the definition of the l-WWLLT method to multi-task vertex classifications, edge labels, or **vertex and edge attributes**.

Edge weight initialization Many machine learning method depend on the initialized state. For example learning methods based on a gradient descent approach. One may argue, that the l-WWLLT method follows this approach, by iteratively improving the edge weights of the WLLT, based on their last definition, in the direction of the biggest improvement (with respect to the local SME).

Thus different edge weight initialization should be considered. For example randomized, based on the layer sizes or more sophisticated ones

based on the WLLT structure. For example the amount of children per vertex.

Another set of experiments arises, if application based knowledge on the original vertex labels, or even entire substructures can be used. One can try to derive edge weights from this knowledge and compare the resulting similarity measure with uninformed initializations.

Update rule The implemented update rule is based on de- and increasing the distance between samples of two graph. The method itself however was based on the idea of decreasing the distance between all samples of the same class and increasing the distance between samples of different classes. One may think of different update rules, which relate to this idea in another way. For example by identifying specific sets of edge weights, which are provably more important for either graph in the same or in different classes. If such edges can be identified, the update rule may target them more specifically, which could circumvent edge weight updates which improve the local SME but strongly degenerate the global SME.

Graph representations The graph representations can amount to sparse, high-dimensional real valued vectors. By introducing unlabeled, disconnected dummy vertices, these representations could be replaced by natural valued vectors. One may implement dimensionality reduction methods (e.g. TruncatedSVD for sparse data). This is at least beneficial for low dimensional visualization methods when evaluating the performance of the implemented method (e.g. t-SNE).

Datasets The motivation of the l-WWLLT method arises from the ability of the WL-labels to capture different graph substructures. Therefore, even if the method works predictably and as desired, the minimum requirement for a successful implementation could be, to outperform the NoG Kernel. As explained before, Schulz and Welke argued that the TUDatasets may not be optimal for graph kernel evaluations [44]. More expensive experiments (with respect to runtime and required storage) should be made on bigger datasets. Preferably datasets, whose graph classification highly depends on the graph structures and or vertex labels.

Performance comparability As stated before, considering only the zeroth WLLT layer in the l-WWLLT method (with uniformly initialized edge weights) strongly relates to the NoG Kernel. The NoG Kernel also considers edge weights for the graph representations. It can be insightful, to compare using one or two WLLT layers in the l-WWLLT method, and only vertex or vertex and edge labels in the NoG method. Similarly, other con-

figurations of the l-WWLLT method can be used to closely resemble other kernels. For example also the WWL Kernel.

Runtime complexity Extensive runtime measurements were omitted in this thesis. The presented estimation of the runtime complexity should be verified empirically. Preferably with a range of configurations that provide more reliable performance.

We conclude that research on the l-WWLLT should be continued.

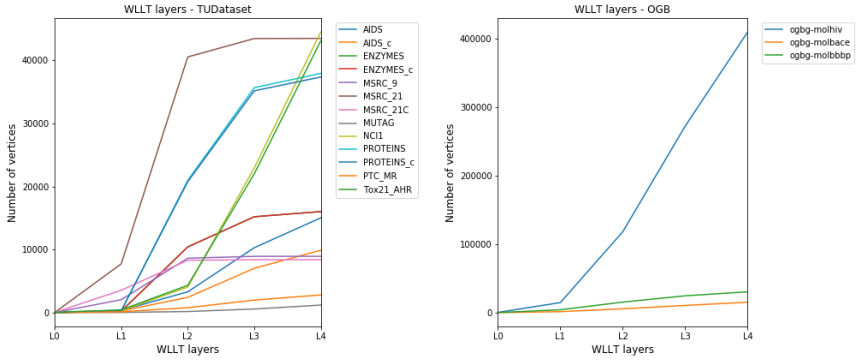
A APPENDIX

A.1 WLLT Statistics

	Layer 0	Layer 1	Layer 2	Layer 3	Layer 4	Sum
AIDS	38	367	3297	10298	15030	29030
AIDS_c	31	263	2419	7028	9893	19634
AIDS_perfect	61	538	4522	12310	16865	34296
ENZYMES	4	231	10416	15208	16029	41888
ENZYMES_c	4	229	10414	15206	16027	41880
MSRC_9	11	2053	8632	8929	8929	28554
MSRC_21	23	7736	40547	43470	43489	135265
MSRC_21C	22	3551	8312	8382	8382	28649
MUTAG	7	33	174	572	1197	1983
NCI1	38	292	4058	22948	44508	71844
ogbg-molbace	74	1425	5588	10540	15228	32855
ogbg-molbbbp	95	4221	15373	24581	30407	74677
PROTEINS	4	297	20962	35676	37940	94879
PROTEINS_c	4	296	20764	35176	37369	93609
PTC_MR	19	130	780	1987	2803	5719
Tox21_AHR	50	432	4325	22012	43112	69931

Table 5: WLLT sizes for different datasets. See section 4.1 for more information on the datasets.

See figures 33a, 33b, and 17 for visualizations of the table 5.



(a) TUDatasets. (b) OGB datasets.

Figure 33: Figure 17 divided by dataset collection.

A.2 Weisfeiler Lehman Kernel on TUDatasets

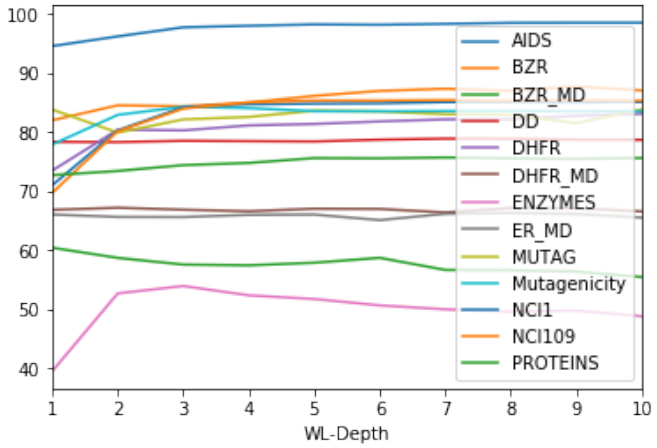


Figure 34: TUDataset - SVM accuracies.

A.3 Overview Graph Kernel Methods

The following table presents more SVM accuracies of the l-WWLLT Kernel for different configurations and datasets. For all these configurations, the push- and pull-factors are applied in a multiplicative way and the edge weights are limited to the interval $[0, 2]$.

Dataset	D	lr	bs_p	f_{pull}	f_{push}	t_{he}	**SVM**	E
AIDS	9	1.00	0.05	0.50	0.50	1.00	80.00	0
AIDS	4	0.20	0.05	0.30	0.20	0.30	96.57	0
AIDS	4	0.20	0.10	0.30	0.20	0.30	96.48	0
AIDS	4	0.50	0.05	0.30	0.20	0.30	96.43	0
AIDS	2	0.50	0.05	0.30	0.20	0.30	98.57	1
AIDS	4	0.50	0.05	0.30	0.20	0.30	96.46	0
AIDS	4	0.80	0.05	0.50	0.50	1.00	97.28	9
AIDS	4	0.80	0.05	0.50	0.50	1.00	97.24	9
AIDS	4	0.80	0.05	0.50	0.50	1.00	97.49	9
AIDS	4	0.80	0.05	0.20	0.20	0.60	97.29	10
AIDS	4	0.80	0.05	0.40	0.10	1.00	96.50	0
AIDS	4	1.00	0.10	0.30	0.05	1.00	96.51	0
AIDS	4	1.00	0.05	0.40	0.40	1.00	97.32	9
AIDS	4	0.80	0.05	0.50	0.50	1.00	97.27	10
AIDS	4	0.80	0.05	0.50	0.50	1.00	97.45	8
AIDS	4	0.80	0.05	0.10	0.10	0.60	97.24	10
AIDS	4	0.80	0.05	0.40	0.40	0.60	97.38	9

AIDS	4	0.80	0.05	0.10	0.20	1.00	97.67	6
AIDS	4	1.00	0.10	0.20	0.20	1.00	96.48	4
AIDS	4	1.00	0.10	0.20	0.20	1.00	97.34	10
AIDS	4	0.10	0.05	0.30	0.30	0.90	96.89	7
AIDS	4	0.20	0.05	0.30	0.30	0.90	97.18	10
AIDS	4	0.30	0.05	0.30	0.30	0.90	97.22	8
AIDS	4	0.40	0.05	0.30	0.30	0.90	97.28	7
AIDS	4	0.50	0.05	0.30	0.30	0.90	97.28	10
AIDS	4	0.60	0.05	0.30	0.30	0.90	97.28	8
AIDS	4	0.70	0.05	0.30	0.30	0.90	97.34	9
AIDS	4	1.00	0.05	0.40	0.40	1.00	80.00	0
AIDS	4	1.00	0.05	0.40	0.40	1.00	80.00	0
AIDS	4	1.00	0.05	1.00	0.80	1.00	80.00	0
AIDS	4	1.00	0.05	0.30	0.10	0.60	80.00	0
AIDS	4	1.00	0.05	0.50	0.50	0.60	80.00	0
AIDS	4	1.00	0.05	0.10	0.30	0.60	80.00	0
AIDS	4	1.00	0.05	0.30	0.10	0.60	80.00	0
AIDS	4	1.00	0.05	0.50	0.50	1.00	96.48	0
AIDS	4	1.00	0.05	0.50	0.50	1.00	96.46	0
AIDS	4	1.00	0.05	0.50	0.50	1.00	96.38	0
AIDS	4	0.80	0.05	0.30	0.30	0.90	97.37	10
AIDS	4	1.00	0.05	0.10	0.05	0.90	96.39	0
AIDS	4	0.80	0.05	0.30	0.30	0.90	97.33	10
AIDS	4	0.90	0.05	0.30	0.30	0.90	97.40	9
AIDS	4	1.00	0.05	0.30	0.30	0.90	97.45	10
AIDS	9	1.00	0.05	0.50	0.50	1.00	80.00	0
AIDS	9	1.00	0.05	0.50	0.50	1.00	80.00	0
AIDS	9	1.00	0.05	0.50	0.50	1.00	80.00	0
AIDS	4	1.00	0.05	1.00	0.10	1.00	80.00	0
AIDS	4	1.00	0.05	0.50	0.10	1.00	80.00	0
AIDS	4	1.00	0.05	0.10	1.00	1.00	80.00	0
AIDS	4	1.00	0.05	0.10	0.10	1.00	80.00	0
AIDS	4	1.00	0.05	0.50	0.50	1.00	80.00	0
AIDS	4	1.00	0.05	1.00	1.00	1.00	80.00	0
AIDS_c	4	1.00	0.05	0.10	0.10	0.60	97.10	0
AIDS_c	4	1.00	0.05	0.50	0.50	0.60	97.51	0
AIDS_c	4	1.00	0.05	1.00	1.00	0.60	97.54	0
AIDS_c	4	1.00	0.05	1.00	0.10	1.00	96.65	0
AIDS_c	4	1.00	0.05	0.10	0.50	1.00	97.75	10
ENZYMES	4	1.00	0.05	0.10	0.10	0.60	54.38	4
ENZYMES	4	1.00	0.05	0.50	0.50	0.60	54.67	10

ENZYMES	4	1.00	0.05	1.00	1.00	0.60	54.40	9
ENZYMES	4	1.00	0.05	1.00	0.10	1.00	53.75	0
ENZYMES	4	1.00	0.05	0.10	0.50	1.00	59.43	7
ENZYMES_c	4	1.00	0.05	0.10	0.10	0.60	54.15	7
ENZYMES_c	4	1.00	0.05	0.50	0.50	0.60	54.24	4
ENZYMES_c	4	1.00	0.05	1.00	1.00	0.60	54.10	1
ENZYMES_c	4	1.00	0.05	1.00	0.10	1.00	52.69	0
ENZYMES_c	4	1.00	0.05	0.10	0.50	1.00	59.06	9
MSRC_9	4	1.00	0.3	0.20	0.20	1.00	91.01	10
MSRC_9	4	1.00	0.3	0.20	0.20	1.00	90.83	3
MSRC_9	4	1.00	0.3	0.10	0.10	0.60	90.87	6
MSRC_9	4	1.00	0.3	0.40	0.40	0.60	90.87	5
MSRC_9	4	1.00	0.3	0.40	0.10	1.00	90.63	1
MSRC_9	4	1.00	0.3	0.10	0.40	1.00	90.81	2
MSRC_9	4	1.00	0.3	0.10	0.40	1.00	91.05	2
MSRC_9	4	1.00	0.3	0.10	0.40	1.00	91.01	2
MSRC_9	4	1.00	0.3	0.30	0.05	1.00	90.60	0
MSRC_9	4	1.00	0.3	0.20	0.20	1.00	90.95	9
MSRC_9	4	1.00	0.3	0.20	0.20	1.00	91.01	3
MSRC_9	2	1.00	0.3	0.20	0.20	1.00	90.83	0
MSRC_21	4	1.00	0.3	0.20	0.20	1.00	85.36	10
MSRC_21	4	1.00	0.3	0.20	0.20	1.00	87.58	8
MUTAG	1	1.00	0.05	0.01	0.01	1.00	87.13	8
MUTAG	1	1.00	0.05	0.80	0.80	1.00	86.66	4
MUTAG	1	1.00	0.05	0.01	0.01	1.00	87.33	8
MUTAG	1	1.00	0.05	0.01	0.01	1.00	87.25	2
MUTAG	1	1.00	0.5	1.00	1.00	0.90	86.90	8
MUTAG	4	1.00	0.2	0.80	0.80	0.90	66.51	10
MUTAG	4	1.00	0.2	0.80	0.80	0.90	66.51	0
MUTAG	4	1.00	0.2	0.30	0.10	0.60	66.53	0
NCI1	4	1.00	0.05	0.10	0.10	0.60	76.65	10
NCI1	4	1.00	0.05	0.30	0.30	1.00	76.83	10
NCI1	4	1.00	0.05	0.10	0.50	1.00	79.44	10
NCI1	4	1.00	0.05	0.50	0.50	0.60	77.26	10
NCI1	4	1.00	0.05	1.00	1.00	0.60	77.63	10
NCI1	4	1.00	0.05	1.00	0.10	1.00	76.57	0
NCI1	4	1.00	0.05	0.10	0.10	1.00	76.66	8
NCI1	4	1.00	0.05	0.30	0.30	1.00	76.88	9
ogbg-molbace	4	1.00	100	0.20	0.20	1.00	82.12	9
ogbg-molbace	4	1.00	100	0.10	0.10	0.60	81.88	9
ogbg-molbace	4	1.00	100	0.40	0.40	0.60	82.04	9

ogbg-molbace	4	1.00	100	0.40	0.10	1.00	81.46	0
ogbg-molbace	4	1.00	100	0.10	0.40	1.00	83.07	5
ogbg-molbace	4	1.00	100	0.30	0.05	1.00	81.57	0
ogbg-molbace	4	1.00	100	0.20	0.20	1.00	81.81	0
ogbg-molbace	4	1.00	100	0.20	0.20	1.00	82.18	10
ogbg-molbace	2	1.00	100	0.20	0.20	1.00	83.56	8
ogbg-molbbbp	4	1.00	100	0.20	0.20	1.00	84.30	10
PROTEINS	4	1.00	0.05	0.10	0.10	0.60	73.22	10
PROTEINS	4	1.00	0.05	0.50	0.50	0.60	73.50	6
PROTEINS	4	1.00	0.05	1.00	1.00	0.60	73.57	4
PROTEINS	4	1.00	0.05	1.00	0.10	1.00	72.49	0
PROTEINS	4	1.00	0.05	0.10	0.50	1.00	73.81	10
PROTEINS_c	4	1.00	0.05	0.10	0.10	0.60	72.61	7
PROTEINS_c	4	1.00	0.05	0.50	0.50	0.60	72.81	6
PROTEINS_c	4	1.00	0.05	1.00	1.00	0.60	72.87	8
PROTEINS_c	4	1.00	0.05	1.00	0.10	1.00	72.52	0
PROTEINS_c	4	1.00	0.05	0.10	0.50	1.00	72.56	10
PTC_MR	4	1.00	0.3	0.20	0.20	1.00	64.59	4
PTC_MR	4	1.00	0.3	0.20	0.20	1.00	64.41	10
PTC_MR	4	1.00	0.3	0.20	0.20	1.00	64.72	10
PTC_MR	4	1.00	0.3	0.10	0.10	0.60	63.87	6
PTC_MR	4	1.00	0.3	0.40	0.40	0.60	64.92	1
Tox21_AHR	4	1.00	0.01	0.10	0.10	0.60	88.37	0

References

[1] A. Barla, F. Odone, and A. Verri. "Histogram intersection kernel for image classification". In: *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*. IEEE, 2003. DOI: 10.1109/icip.2003.1247294.

[2] S. Boughorbel, J.-P. Tarel, and N. Boujemaa. "Generalized histogram intersection kernel for image recognition". In: *IEEE International Conference on Image Processing 2005*. IEEE, 2005. DOI: 10.1109/icip.2005.1530353.

[3] Kristen Grauman and Trevor Darrell. "The pyramid match kernel: Efficient learning with sets of features." In: *Journal of Machine Learning Research* 8.4 (2007).

[4] Zaid Harchaoui and Francis Bach. "Image Classification with Segmentation Graph Kernels". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2007. DOI: 10.1109/cvpr.2007.383049.

[5] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[6] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[7] Lu Bai, Luca Rossi, Zhihong Zhang, et al. "An Aligned Subtree Kernel for Weighted Graphs". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 30–39.

[8] K. M. Borgwardt, C. S. Ong, S. Schonauer, et al. "Protein function prediction via graph kernels". In: *Bioinformatics* 21.Suppl 1 (June 2005), pp. i47–i56. DOI: 10.1093/bioinformatics/bti1007.

[9] Holger Fröhlich, Jörg K. Wegner, Florian Sieker, 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.

[10] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, et al. "Weisfeiler-Lehman Graph and Kernels". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2539–2561.

[11] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, et al. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity". In: *Journal of Medicinal Chemistry* 34.2 (Feb. 1991), pp. 786–797. DOI: 10.1021/jm00106a046.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., Jan. 1979. 340 pp. ISBN: 0716710455.

[13] Ronald C. Read and Derek G. Corneil. "The graph isomorphism disease". In: *Journal of Graph Theory* 1.4 (1977), pp. 339–363. DOI: 10.1002/jgt.3190010410.

[14] 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.

[15] 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.

[16] 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.

[17] 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, Dale Schuurmans, John D. Lafferty, et al. Curran Associates, Inc., 2009, pp. 1660–1668.

[18] David Haussler. "Convolution Kernels on Discrete Structures". In: 1999.

[19] 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.

[20] Nils M. Kriege, Pierre-Louis Giscard, Department of Computer, et al. "On Valid Optimal Assignment Kernels and Applications to Graph Classification". In: 2016.

[21] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, et al. "Wasserstein Weisfeiler-Lehman Graph Kernels". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[22] Till Hendrik Schulz, Tamás Horváth, Pascal Welke, et al. "A Generalized Weisfeiler-Lehman Graph Kernel". In: 2021.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] Johannes Kobler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Springer Science & Business Media, 2012.

[30] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". In: *Edsger Wybe Dijkstra*. ACM, July 2022, pp. 287–290. DOI: 10.1145/3544585.3544600.

[31] T. Gartner. "Exponential and geometric kernels for graphs". In: *NIPS^* 02 Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*. 2002.

[32] M. Kuramochi and G. Karypis. "An efficient algorithm for discovering frequent subgraphs". In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (Sept. 2004), pp. 1038–1051. DOI: 10.1109/tkde.2004.33.

[33] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, et al. "Efficient graphlet kernels for large graph comparison". In: 2009.

[34] Kristian Kersting, Nils M. Kriege, Christopher Morris, et al. "Benchmark data sets for graph kernels". In: (2016).

[35] Christopher Morris, Nils M. Kriege, Franka Bause, et al. "TUDataset: A collection of benchmark datasets for learning with graphs". In: 2020.

[36] S. Vichy N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, et al. "Graph kernels". In: *Journal of Machine Learning Research* 11 (2010), pp. 1201–1242.

[37] M. Gori, M. Maggini, and L. Sarti. "Exact and approximate graph matching using random walks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.7 (July 2005), pp. 1100–1111. DOI: 10.1109/tpami.2005.138.

[38] Kaspar Riesen and Horst Bunke. "Approximate graph edit distance computation by means of bipartite graph matching". In: *Image and Vision Computing* 27.7 (June 2009), pp. 950–959. DOI: 10.1016/j.imavis.2008.04.004.

[39] Michele Schiavinato, Andrea Gasparetto, and Andrea Torsello. "Transitive Assignment Kernels for Structural Classification". In: *Similarity-Based Pattern Recognition*. Springer International Publishing, 2015, pp. 146–159. DOI: 10.1007/978-3-319-24261-3_12.

[40] Nikil Wale, Ian A. Watson, and George Karypis. "Comparison of descriptor spaces for chemical compound retrieval and classification". In: *Knowledge and Information Systems* 14.3 (Aug. 23, 2007), pp. 347–375. DOI: 10.1007/s10115-007-0103-5.

[41] Pinar Yanardag and S. V. N. Vishwanathan. "Deep Graph Kernels". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2015. DOI: 10.1145/2783258.2783417.

[42] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, et al. "GraKeL: A Graph Kernel Library in Python". In: ed. by Antti Honkela. 2020.

[43] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, et al. "Benchmarking Graph Neural Networks". In: arXiv, 2020. DOI: 10.48550/ARXIV.2003.00982.

[44] Till Schulz and Pascal Welke. "On the Necessity of Graph Kernel Baselines". In: 2019.

[45] Pascal Welke, Tamás Horváth, and Stefan Wrobel. "Probabilistic frequent subtrees for efficient graph classification and retrieval". In: *Machine Learning* 107.11 (Nov. 2017), pp. 1847–1873. DOI: 10.1007/s10994-017-5688-7.

[46] F. Scarselli, M. Gori, Ah Chung Tsoi, et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. DOI: 10.1109/tnn.2008.2005605.

[47] Keyulu Xu, Weihua Hu, Jure Leskovec, et al. "HOW POWERFULARE GRAPH NEURAL NETWORKS?" In: *International Conference on Learning Representations*. 2019.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, et al. "Graph Attention Networks". In: arXiv, 2017. DOI: 10.48550/ARXIV.1710.10903.

[49] Jianpeng Cheng, Li Dong, and Mirella Lapata. *Long Short-Term Memory-Networks for Machine Reading*. 2016. DOI: 10.48550/ARXIV.1601.06733.

[50] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer Berlin Heidelberg, 2018. DOI: 10.1007/978-3-662-56039-6.

[51] Tam Le, Makoto Yamada, Kenji Fukumizu, et al. "Tree-Sliced Variants of Wasserstein Distances". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[52] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, et al. "GOT: An Optimal Transport framework for Graph comparison". In: *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*. 2019.

[53] Gaspard Monge. "Mémoire sur la théorie des déblais et des remblais". In: *Histoire de l'Académie Royale des Sciences de Paris* (1781).

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

[55] 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.

[56] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[57] Rainer E. Burkard and Eranda Çela. "Linear Assignment Problems and Extensions". In: *Handbook of Combinatorial Optimization*. Springer US, 1999, pp. 75–149. DOI: 10.1007/978-1-4757-3023-4_2.

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

[59] 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.

[60] Laszlo Babai and Ludik Kucera. "Canonical labelling of graphs in linear average time". In: *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE, Oct. 1979. DOI: 10.1109/sfcs.1979.8.

[61] Karsten Michael Borgwardt. *Graph Kernels*. 2007.

[62] Bernhard Schökopf, Alexander J. Smola, Francis Bach, et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[63] Harris Drucker, Christopher J. Burges, Linda Kaufman, et al. "Support vector regression machines". In: *Advances in neural information processing systems* 9 (1996).

[64] Vladimir Vapnik, Steven Golowich, and Alex Smola. "Support vector method for function approximation, regression estimation and signal processing". In: *Advances in neural information processing systems* 9 (1996).

[65] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Kernel principal component analysis". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1997, pp. 583–588. DOI: 10.1007/bfb0020217.

[66] Jean-Philippe Vert. "The optimal assignment kernel is not positive definite". In: Oct. 27, 2018.

[67] Christian Berg, Jens Peter Reus Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups*. Springer New York, 1984. DOI: 10.1007/978-1-4612-1128-0.

[68] Aasa Feragen, Hauberg Søren, Compute Dtu, et al. "Geodesic Exponential Kernels: When Curvature and Linearity Conflict". In: 2015.

[69] Matthias Rupp. "Machine learning for quantum mechanics in a nutshell". In: *International Journal of Quantum Chemistry* 115.16 (July 2015), pp. 1058–1073. DOI: 10.1002/qua.24954.

[70] I. J. Schoenberg. "Metric Spaces and Completely Monotone Functions". In: *The Annals of Mathematics* 39.4 (Oct. 1938), p. 811. DOI: 10.2307/1968466.

[71] Béla Bollobás, William Fulton, Anatole Katok, et al. *New Mathematical Monographs*. Cambridge University Press, 2017.

[72] Andrew Gardner, Christian A. Duncan, Jinko Kanno, et al. "On the Definiteness of Earth Mover's Distance and Its Relation to Set Intersection". In: *IEEE Transactions on Cybernetics* 48.11 (Nov. 2018), pp. 3184–3196. DOI: 10.1109/tcyb.2017.2761798.

[73] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7.

[74] David L. Davies and Donald W. Bouldin. "A Cluster Separation Measure". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (Apr. 1979), pp. 224–227. DOI: 10.1109/tpami.1979.4766909.

[75] T. Calinski and J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics - Theory and Methods* 3.1 (1974), pp. 1–27. DOI: 10.1080/03610927408827101.

[76] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*. ACM Press, 1992. DOI: 10.1145/130385.130401.

[77] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. "A Practical Guide to Support Vector Classification". In: 2003.

[78] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[79] Sergei Ivanov, Sergei Sviridov, and Evgeny Burnaev. "Understanding Isomorphism Bias in Graph Data Sets". In: (2019). DOI: 10.48550/ARXIV.1910.12091.

[80] Weihua Hu, Matthias Fey, Marinka Zitnik, et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: arXiv, 2020. DOI: 10.48550/ARXIV.2005.00687.

[81] Kaspar Riesen and Horst Bunke. "IAM Graph Database Repository for Graph Based Pattern Recognition and Machine Learning". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 287–297. DOI: 10.1007/978-3-540-89689-0_33.

[82] National Cancer Institute. *AIDS Antiviral Screen Data*. 2022. URL: https://jonas-moennig.de/how-to-cite-a-website-with-bibtex/ (visited on 11/22/2022).

[83] Nils Kriege and Petra Mutzel. "Subgraph Matching Kernels for Attributed Graphs". In: 2012.

[84] C. Helma, R. D. King, S. Kramer, et al. "The Predictive Toxicology Challenge 2000-2001". In: *Bioinformatics* 17.1 (Jan. 2001), pp. 107–108. DOI: 10.1093/bioinformatics/17.1.107.

[85] National Cancer for Advancing Translational Sciences. *Tox21 Data Challenge 2014*. 2022. URL: https://tripod.nih.gov/tox21/challenge/data.jsp (visited on 11/22/2022).

[86] Marion Neumann, Roman Garnett, Christian Bauckhage, et al. "Propagation kernels: efficient graph kernels from propagated information". In: *Machine Learning* 102.2 (July 2015), pp. 209–245. DOI: 10.1007/s10994-015-5517-9.

[87] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, et al. "MoleculeNet: a benchmark for molecular machine learning". In: *Chemical Science* 9.2 (2018), pp. 513–530. DOI: 10.1039/c7sc02664a.

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information, including graphs and datasets, have been specifically acknowledged. This thesis has not yet been submitted in the same or similar form to any examination authority.

Ort, Datum Unterschrift