WASSERSTEIN WEISFEILER-LEHMAN SUBTREE DISTANCE FOR GRAPH-STRUCTURED DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Defining a valid graph distance is a challenging task in graph machine learning because we need to consider the theoretical validity of the distance, its computational complexity, and effectiveness as a distance between graphs. Addressing the shortcomings of the popular Weisfeiler-Lehman (WL) test for the graph isomorphism problem, this paper proposes a novel distance between graph structures. More specifically, we first analyze the WL algorithm from a geometric point of view and argue that discriminating nodes based on only the consistency of categorical labels do not fully capture important structural information. Therefore, instead of using such categorical labels, we define a node distance between WL subtrees with tree edit distance and propose an efficient calculation algorithm. We then apply the proposed node distance to define a graph Wasserstein distance on tree edit embedding space exploiting Optimal Transport framework. To summarize, these two distances have been proposed at the node level and graph level, respectively. Numerical experimentations on graph classification tasks show that the proposed graph Wasserstein distance performs equally or better than conventional methods.

1 INTRODUCTION

Many real-world data can be represented as graph structures. Hence, generalizable graph machine learning models have a wide range of applications. To build such graph machine learning models, we need to accurately capture the feature information of graphs and the inherent structural information of graphs. Recently, Graph Neural Networks (GNNs) have achieved remarkable results in knowledge graphs (Wang et al., 2019; Hu et al., 2020) and recommender systems (Ying et al., 2018) with superior graph embedding capability. However, they cannot describe structural information due to their near-exclusive focus on feature marginals. This is also the case for other graph embedding methods using probabilistic search algorithms, such as DeepWalk (Perozzi et al., 2014) and Node2vec (Grover & Leskovec, 2016). Although they successfully extract network information such as social relationships, they only address the sequential connections of nodes visited in a probabilistic manner and are inadequate for capturing structural information. Furthermore, most of them cannot leverage node attributes and have drawbacks in embedding capability (Deng et al., 2020). On the other hand, some methods are specifically designed for measuring structural differences, such as graph edit distance (Gao et al., 2010), Gromov-Wasserstein distance (Memoli, 2007), and Fused Gromov-Wasserstein (FGW) distance (Vayer et al., 2020). However, because of the NP-hardness of the problems, their approximate solutions require at least $\mathcal{O}(n^3)$ for the input nodes (Riesen, 2015; Peyré et al., 2016), which limits their practical applications. Nevertheless, is should be emphasized the study of structural information and the distance between graph structures received less research attention in recent years. Also, the current general neural network model cannot yet handle this problem.

To this end, we focus on comparison of graph structures. Specifically, we dive into the Weisfeiler-Lehman (WL) kernel (Shervashidze et al., 2011) and Wasserstein WL (WWL) kernel (Togninalli et al., 2019). We then propose a more effective way to obtain structural information, and subsequently define a new distance between graph structures. Graph kernels (Vishwanathan et al., 2010) are kernel functions that compute inner products on graphs, and most of them are constructed based on the $\mathcal{R}$-convolutional theory (Haussler, 1999), which measures the graph similarities by comparing their subgraphs. In order to capture key subgraphs, various types of graph decomposition methods have been proposed in the literature of graph kernels. Among them, the WL kernel is known as one

of the most reliable and consistent methods to provide high accuracy on graph classification tasks. Our key insight is that, from a geometric perspective, the WL algorithm compresses the structural information of subgraphs with hash values, and this results in describes of the measurement power of structural similarities and differences. Hence, using the WL algorithm to construct a subgraph, *a.k.a. WL subtree*, we apply it for the peripheral structural information of each node. By defining the distance between WL subtrees, we overcome the issue that WL only distinguishes whether nodes are the same or not and cannot define a valid distance between nodes.

In order to define a valid distance between nodes, our framework first constructs a WL subtree, which is a rooted, unordered tree for each node. Then, we use tree edit distance to define the node distance. Considering that the edit distance between unordered trees is a MAX SNP-hard problem, we adopt an approximate solver (Garofalakis & Kumar, 2005) to solve it. As a condition for using this approximate solver, we must enumerate the patterns of all complete subtrees for given trees. For this purpose, we propose an efficient algorithm for enumerating all the patterns based on the Schwartz-Zippel lemma. Furthermore, we define the graph Wasserstein distance by considering the distance between each node of graphs on tree edit embedding space, *i.e.*, the tree edit distance between WL subtrees, as the ground distance in Optimal Transport (OT).

Our main contributions are summarized as follows:

1. We show that the hash update mechanism of the WL algorithm loses structural information of the graph. Instead, we propose to use the WL subtree as node feature.
2. We design an efficient algorithm to hash all complete subtrees of WL subtrees, allowing the approximate tree edit distance to be used to define the node distance.
3. We construct the first graph Wasserstein distance that reflects the tree edit distance to the graph level and focuses on the differences in structural information between graphs.

We designate our proposed distance measure as the *Wasserstein Weisfeiler-Lehman Subtree (WWLS) distance* and show its effectiveness in experiments on graph classification tasks. As numerical evaluations reveal, WWLS reaches a level of performance comparable to state-of-the-art graph kernels and distance measures on several real-world benchmark datasets.

2 Related Work

The Weisfeiler-Lehman (WL) test, originally an approximate solution to the graph isomorphism problem, has gained much attention in the graph machine learning community due to its linear computational complexity and the high accuracy of the WL kernel (Shervashidze et al., 2011), for graph classification tasks. Noteworthy, the WL algorithm is, in fact, a special case of message passing in Graph Neural Networks (GNNs) and has been proven as the upper limit of message passing GNNs in terms of theoretical expressive power (Xu et al., 2019). In addition, most realistic graph data has node and/or edge attributes so that the WL test can work on almost all graphs in practice (Balcilar et al., 2021). Togninalli et al. (2019) combined WL algorithm and Wasserstein distance (Villani, 2009) to define a graph Wasserstein distance. Furthermore, he constructed a kernel function named Wasserstein Weisfeiler-Lehman (WWL) kernel. WWL often provides better classification accuracies on real-world benchmark datasets compared to other graph kernels; however, it has a critical flaw when dealing with categorical labels: node features are sequences of node labels obtained by WL algorithm, and the Hamming distance between them defines the distance between nodes. According to the node label update mechanism of the WL algorithm, different inputs will yield a different new label. Thus, for two nodes with different initial labels, the distance between them is always zero. Moreover, considering that in practice, nodes are updated only a few times, the Hamming distance can only take on a few different values, which leads to the loss of structural information. Overcoming this deficiency is a priority in our research.

3 Preliminaries

We begin by introducing some important mathematical notations, as well as the notations for graphs and trees that we use in this paper. In addition, we provide the necessary background knowledge on the Weisfeiler-Lehman algorithm and the Wasserstein distance.

The bold typeface lower-case and upper-case letters such as $\mathbf{x}$ and $\mathbf{X}$ represent a vector and a matrix, respectively. $\mathbf{X}_{i,j}$ represents the element at (i,j) of $\mathbf{X}$. $\mathbb{R}_+^n$ is the nonnegative n-dimensional vector, and $\mathbb{R}_+^{m\times n}$ denotes the nonnegative $m\times n$ size matrix. $\triangle_n$ stands for the probability simplex with n bins. δ_x is the Dirac function at x. Also, $\mathbb{1}_n = (1,\ldots,1)^{\mathrm{T}} \in \mathbb{R}^n$. $\{\}$ indicates the set that does not allow duplication of elements. On the other hand, $\{\!\{\}\!\}$ indicates the multiset that allows elements to be repeated. $\mathcal{G}$ is graph-structured data with a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V}^2$, denoted as $\mathcal{G}(\mathcal{V},\mathcal{E})$. Assume that the edges of $\mathcal{G}$ are *undirected* and that the number of nodes in $\mathcal{G}$ is $n_{\mathcal{G}} = |\mathcal{V}|$. Furthermore, the neighborhood of a node v in $\mathcal{G}$ is written as $\mathcal{N}_{\mathcal{G}}(v) = \{u \in \mathcal{V} \mid (v,u) \in \mathcal{E}\}$. $\deg(v)$ is the degree of node v. Node v may also have a categorical label and/or continuous attribute vector, denoted by $\ell(v) \in \mathbb{R}$ and $\mathbf{x}_v \in \mathbb{R}^d$, respectively, where d is the dimension of the node feature. T is a tree, and we specifically refer to a rooted, unordered *WL subtree*. For a tree T whose root node is v, it is denoted by $T(v)$. A non-root node $v \in \mathcal{V}(T)$ has a *parent* denoted by parent(v), and v is also called a *child* of parent(v). A subtree T' of T is *complete* if, for node $v \in \mathcal{V}(T)$, parent(v) implies $v \in \mathcal{V}(T')$, and we write t for such a complete subtree. Also, for complete subtree t whose root node is v, it is denoted by $t(v)$. The sets of nodes, edges, and leaves of T are denoted by $\mathcal{V}(T)$, $\mathcal{E}(T)$, and leaf(T), respectively. The depth of node v in T is denoted by $\mathrm{dep}_T(v)$. Lastly, an isomorphism from T_1 to T_2 is denoted by $T_1 \approx T_2$. (resp. of a complete subtree t).

Weisfeiler-Lehman algorithm. The graph isomorphism problem is the computational problem of determining whether two finite graphs are isomorphic, it is known as an NP-intermediate problem (Babai, 2016; Takapoui & Boyd, 2016). The Weisfeiler-Lehman (WL) test is a linear-time approximate solution to the problem, and its validity has been verified on most graphs. It is accomplished by aggregating the labels of the nodes and their neighbors to create ordered strings and then hashing these strings to create new node labels. As the number of iterations increases, these labels will represent a larger neighborhood of each node, allowing for more extended substructures to be compared (Togninalli et al., 2019). The WL algorithm follows a recursive scheme that updates each node label multiple times. Consider a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ and let $\ell^{(k)}(v)$ be the node label for each node $v \in \mathcal{V}$ at the k-th iteration of the WL algorithm, especially, let $\ell^{(0)}(v)$ be initialized with a categorical label, then the update formula for each node is defined as

$$\ell^{(k+1)}(v) = \mathrm{HASH}\left(\ell^{(k)}(v), \{\!\{\ell^{(k)}(u) \mid u \in \mathcal{N}_{\mathcal{G}(v)}\}\!\}\right). \tag{1}$$

Definition 1 (Wasserstein distance)**.** *The Wasserstein distance stems from the Optimal Transport (OT) problem, which attempts to calculate the minimum transportation cost by finding an optimal transportation plan between two probability distributions. Since the original Monge's problem is difficult to solve due to its strict conditions, the existing OT usually refers to the Monge-Kantorovich problem. The discrete case is defined as below.*

Let $\triangle_m = \{\mathbf{a} \in \mathbb{R}_+^m \mid \sum_{i=1}^m \mathbf{a}_i = 1\}$ and $\triangle_n = \{\mathbf{b} \in \mathbb{R}_+^n \mid \sum_{j=1}^n \mathbf{b}_j = 1\}$ are two simplexes of the histogram with m and n on the same matrix space. Their probability measures are $\alpha = \sum_{i=1}^m \mathbf{a}_i \delta_{xi}$ and $\beta = \sum_{i=1}^n \mathbf{b}_j \delta_{y_j}$, respectively. $\mathbf{C} \in \mathbb{R}_+^{m\times n}$ is a ground cost matrix, especially where $\mathbf{C}_{i,j}$ represents the transportation cost between bin i and j (location x_i and y_j). When the ground cost $\mathbf{C}$ is the distance, the overall minimum transportation cost between α and β is defined as *p-Wasserstein distance*, which is given by

$$\mathcal{W}_p(\alpha,\beta;\mathbf{C}) = \left(\min_{\mathbf{P}\in\mathbf{U}(\mathbf{a},\mathbf{b})} \sum_{i=1}^m \sum_{j=1}^n (\mathbf{C}_{i,j})^p \mathbf{P}_{i,j}\right)^{\frac{1}{p}}, \tag{2}$$

where $p \in [1,\infty)$, $\mathbf{U}(\mathbf{a},\mathbf{b}) = \{\mathbf{P} \in \mathbb{R}^{m\times n} \mid \mathbf{P}\mathbb{1}_n = \mathbf{a} \text{ and } \mathbf{P}^{\mathrm{T}}\mathbb{1}_m = \mathbf{b}\}$, and $\mathbf{P}$ is a coupling matrix that describes the transportation plan. 1-Wasserstein distance is also called Earth Mover's Distance (EMD). In general, the computation of the Wasserstein distance has a complexity of $\mathcal{O}(n^3\log(n))$ when $n = m$ (Pele & Werman, 2009). Cuturi (2013) proposed Sinkhorn's algorithm, an approximate solver for Eq. (2), which reduces the complexity to almost $\mathcal{O}(n^2)$. Moreover, Sinkhorn's algorithm is GPU friendly, which can further speed up the computational process.

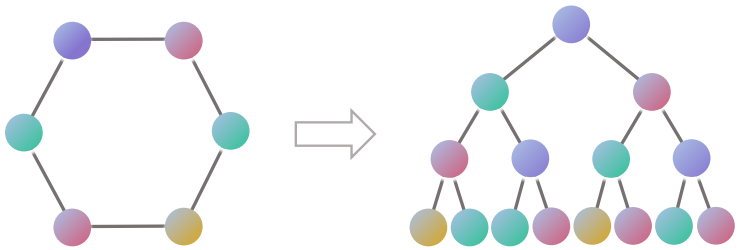

Figure 1: Weisfeiler-Lehman subtree with blue node as root, when the number of iterations is 3.

4 WASSERSTEIN WEISFEILER-LEHMAN APPROACH FOR STRUCTURED DATA

4.1 GEOMETRIC PROPERTIES OF WEISFEILER-LEHMAN ALGORITHM

The Weisfeiler-Lehman (WL) algorithm adopts a recursive node label update of Eq. (1). On the other hand, a geometric analysis constructs a rooted, unordered tree called the *WL subtree* at each node. Fig. 1 is an illustration of it. All WL subtrees are balanced trees whose height is always equal to the number of iterations. With the increase of the number of iterations, the tree's height gets higher, and the new label obtained each time can be regarded as the hash value corresponding to the newly constructed tree. More importantly, the WL subtree has the structural information around the corresponding node. By repeating the WL algorithm k times for node v, the WL subtree can obtain the structural information of the subgraph within k-hop from v. If k is sufficiently large, the WL subtree will contain the global information from the whole graph structure. However, many experiments have shown that GNNs and the WL-based kernels have the highest accuracy on graph classification tasks when k takes a small value such as one in the range $[2, 4]$. Even for other tasks, such as node classification and edge prediction, GNNs also take a small k. Hence, this empirically shows that the local structure is more important in many cases.

Next, we consider the WL algorithm in terms of graph isomorphism. An isomorphism from graph $\mathcal{G}_1(\mathcal{V}_1, \mathcal{E}_1)$ to graph $\mathcal{G}_2(\mathcal{V}_2, \mathcal{E}_2)$ refers to the existence of a one-to-one mapping from $\mathcal{V}_1$ to $\mathcal{V}_2$. It is difficult to determine whether the mapping exists. We assume that if we can somehow successfully represent the structural information around each node, we can determine the consistency between two nodes with high probability. According to our analysis in the previous paragraph, the WL algorithm constructs WL trees in a geometric sense, including the structural information within each node k-hop. Since we want to know *only the consistency of nodes*, each WL tree is hashed to a real value, efficiently determining whether the nodes are the same by comparing corresponding hash values. As a result, the WL test can determine whether two graphs are "isomorphic" or not by focusing on the types of node labels and their counts. While the WL test had great success with the graph isomorphism problem, capturing the structural information between graphs is another problem. In the WL algorithm, only the discrete measure can measure the difference between two nodes, and consequently, it is impossible to define a valid node distance.

The WL kernel uses inner product of the counts of all node labels to measure the graph similarity; in the case of WWL's categorical embedding, it uses the Hamming distance between two sequences of node labels to measure node dissimilarity. Since Eq. (1) is a hash function, if two labels are different at iteration k_0 (where $k_0 \geq 0$), their labels obtained by subsequent updates at iteration $k' > k_0$ are also different. In other words, neither WL nor WWL works well when comparing two graphs with significantly different structures because the obtained node labels will be almost different. In addition, for the WWL, the sequence length must equal to iteration number $k+1$; thus, the Hamming distance only takes on a few different values, which means that the ground distance between nodes on the embedding space is only defined by several values. The practical effect of OT depends on the ground distance, which in this case results in coarse pairwise matching. These are the shortcomings of the method designed along the lines of the WL test.

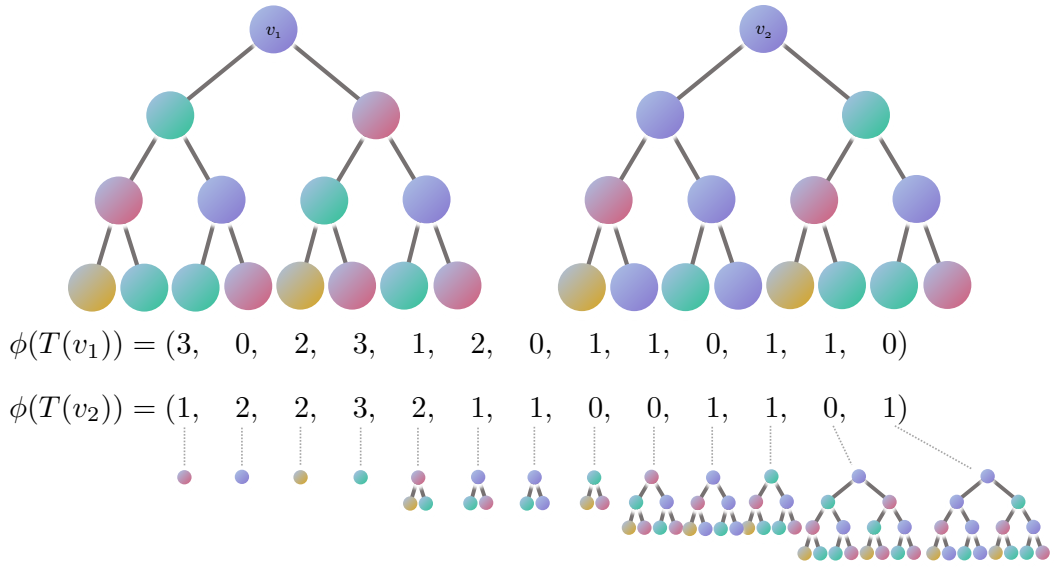

Figure 2: $T(v_1)$ and $T(v_2)$ are WL subtrees whose root nodes are v_1 and v_2, respectively. $\phi(T(v_1))$ and $\phi(T(v_2))$ are feature vectors for trees. The original figure was created by Fukagawa et al. (2009).

4.2 Tree edit distance between Weisfeiler-Lehman subtrees

Instead of measuring node dissimilarity with node labels obtained from the WL algorithm, we focus on the WL subtrees to measure the differences between tree structures. Specifically, we define the distance between WL subtrees of nodes v_1 and v_2 using the tree edit distance. The tree edit distance is MAX SNP-hard, which makes it hard to apply directly; thus, we use an approximate solver that embeds the ordered edit distance into the L_1-normed vector space (Garofalakis & Kumar, 2005; Fukagawa et al., 2009). To be precise, for all complete subtrees of $T(v_1)$ and $T(v_2)$, we use a function $d_\phi : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that constructs a feature vector using the number of complete subtrees corresponding to each of them in $T(v_1)$ or $T(v_2)$ and compute the L_1 distance between two features:

$$d_\phi(v_1, v_2) = ||\phi(T(v_1)) - \phi(T(v_2))||_1, \tag{3}$$

where $\phi(T) = \{\#(T, t)\}_{t \in \mathcal{T}} \in \mathbb{R}^{|\mathcal{T}|} = \Omega_\phi$. $\mathcal{T} = \{t(v_1), t(v_2) \dots\}$ is the set of all complete subtrees of $T(v_1)$ and $T(v_2)$, and $\#(T, t)$ denotes the number of $t(v)$'s isomorphisms with respect to t, specifically, $\#(T, t) = |\{v \in \mathcal{V}(T) \mid t(v) \approx t\}|$. Fig. 2 is a concrete image of $\phi(T)$. The true tree edit distance $d_{TED}(v_1, v_2)$ between T_1 and T_2 can be bounded using Eq. (3) as $d_\phi(v_1, v_2)/(2h+2) \leq d_{TED}(v_1, v_2) \leq d_\phi(v_1, v_2)$, where h is the maximum height of $T(v_1)$ and $T(v_2)$ (All nodes of the WL subtree have the same height). This illustrates that $d_\phi(\cdot, \cdot)$ will be closer to $d_{TED}(\cdot, \cdot)$ when h takes a smaller value. Considering that h and the number of iterations k are always the same, k will only take small values in practice, providing $d_{TED}(\cdot, \cdot)$ a tight lower bound.

In order to compute $\phi(T(v_1))$ or $\phi(T(v_2))$ in Eq. (3), we need to know all complete subtree patterns of $T(v_1)$ and $T(v_2)$. It is not practical to enumerate all complete subtrees directly, both in terms of computational complexity and memory usage; thus, we propose a scheme to label each complete subtree when searching the WL subtree T. Precisely, we use *Schwartz-Zippel lemma* (see Lemma 1 in Appendix) to design a good hash function that map the $t(v) \in \mathcal{V}(T)$ when we visit v twice in the Euler tour of T in Depth-First Search (DFS) order. First, assume that there is one random number for each depth except depth h in T: gives a different positive random number $r_{i \neq h} \in (0, M]$ for depth $i \in \{0, 1, \dots, h-1\}$, where M is the maximum value of the random number. In addition, we set $r_h = 0$ for depth h. Second, we perform DFS traversal for T. At the same time, if v is visited twice in the Euler tour of T, we compute the hash value of $t(v)$ using the hash function $\mathrm{HASH}_{\mathcal{SZ}} : \mathbb{R} \times \mathbb{R} \times \mathcal{A} \to \mathbb{R}$ based on the Schwartz-Zippel lemma, where the $\mathcal{A}$ is a multiset consisting of node labels. $\mathrm{HASH}_{\mathcal{SZ}}$ is formally defined by

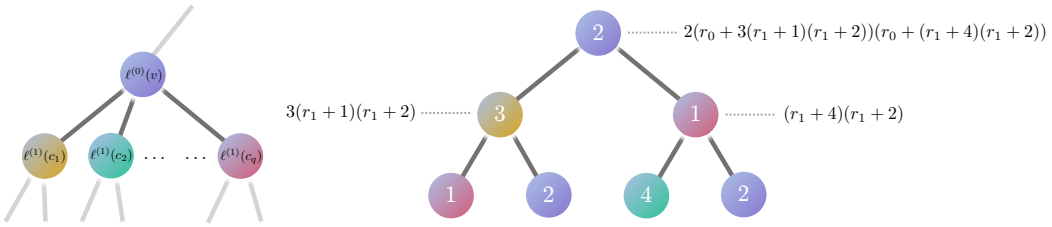

Figure 3: Left: When we visit v twice in the Euler tour of a WL subtree, we calculate $\ell^{(1)}(v)$ using Eq. (4). Right: Polynomials of a WL subtree.

$$\begin{aligned}\ell^{(1)}(v) &= \mathrm{HASH}_{\mathcal{SZ}}\left(\ell^{(0)}(v), r_{\mathrm{dep}_t(v)}, \{\!\{\ell^{(1)}(c_1), \ldots, \ell^{(1)}(c_q)\}\!\}\right) \\ &= \ell^{(0)}(v) \times \left(r_{\mathrm{dep}_t(v)} + \ell^{(1)}(c_1)\right) \times \cdots \times \left(r_{\mathrm{dep}_t(v)} + \ell^{(1)}(c_q)\right), \end{aligned} \tag{4}$$

where c_i is a child of v for $i \in \{1, 2, \ldots, q\}$, and q is the number of children of v. Since c_i is always visited twice before visiting v twice in the Euler tour, we can obtain $\ell^{(1)}(c_i)$ before computing $\ell^{(1)}(v)$ and consequently compute each hash value dynamically. The reason why we use the random number $r_{\mathrm{dep}_t(v)}$ in Eq. (4) is that we construct a polynomial on a finite filed M by representing the variables with random numbers. In addition, $\ell^{(1)}(c_i)$ is a polynomial that includes the variable $r_{\mathrm{dep}_t(c_i)}$; thus, $\ell^{(1)}(v)$ is a multi-variable polynomial. For the design of the algorithm, any polynomial corresponding to the $t(v)$ in which $v \notin \mathrm{leaf}(T)$ is a multi-variable polynomial of $\{r_{\mathrm{dep}_t(v')}\}_{v' \in \mathcal{V}(t(v))}$ of degree $|\mathrm{leaf}(t(v))|$. Fig. 3 is an illustration of Eq. (4). The next theorem states that Lemma 1 can give an upper bound on the collision probability of $\mathrm{HASH}_{\mathcal{SZ}}(\cdot,\cdot,\cdot)$.

Theorem 1. *Assume that $\ell^{(1)}(v_1)$ and $\ell^{(1)}(v_2)$ are hash values corresponding to two complete subtrees $t(v_1)$ and $t(v_2)$, respectively. Then, the upper bound of the collision probability between $\ell^{(1)}(v_1)$ and $\ell^{(1)}(v_2)$ is $\frac{|\mathrm{leaf}(t(v_1))|+|\mathrm{leaf}(t(v_2))|}{M}$.*

Proof. Assume that P_1 and P_2 are polynomials corresponding to two complete subtrees $t(v_1)$ and $t(v_2)$, respectively. Let $P = P_1 - P_2 = p(r_1, r_2, \ldots, r_q)$ be a polynomial of degree d over a field F, where $d \leq |\mathrm{leaf}(t(v_1))| + |\mathrm{leaf}(t(v_2))|$. Let S be a finite subset of F. Since we constraint $r_i \in (0, M]$ for $i \in \{1, 2, \ldots, q\}$, $|S| = M$. Let $r'_1, r'_2, \ldots, r'_q$ are random numbers chosen from S. According to the Lemma 1, $\Pr[p(r'_1, r'_2, \ldots, r'_q) = 0] \leq \frac{d}{M} \leq \frac{|\mathrm{leaf}(t(v_1))|+|\mathrm{leaf}(t(v_2))|}{M}$. □

By Theorem 1, we can reduce the collision probability to a sufficiently small value by taking a large M. We next provide a computational complexity analysis for computing $\phi(\cdot)$. Considering the worst case for graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, assume that we obtain a WL subtree T that is a complete K-ary tree with height h, where $K = \max\left(\deg(v_1), \deg(v_2), \ldots, \deg(v_{n_\mathcal{G}})\right)$. $|\mathcal{V}(T)| = \frac{1-K^{(h+1)}}{1-K}$. $|\mathcal{E}(T)| = |\mathcal{V}(T)| - 1$. Since the hash function is implemented in DFS, we can conclude that the computational complexity of enumerating all hash values is the same that of DFS, that is $\mathcal{O}(|\mathcal{V}(T)| + |\mathcal{E}(T)|) = \mathcal{O}(\frac{2K^{(h+1)}-K-1}{K-1})$. The upper bound of the total computational complexity is $\mathcal{O}(\frac{2K^{(h+1)}-K-1}{K-1} n_\mathcal{G})$. For many graph data, K is usually not very large and h actually takes only a small value; thus the computational complexity is not a big issue.

4.3 Wasserstein distance between graphs

This subsection presents the graph Wasserstein distance exploiting the tree edit distance as the cost matrix $\mathbf{C}$ in the Optimal Transport problem. This enables us to perform various graph analysis tasks such as graph classification and graph alignment.

The proposed graph Wasserstein distance measures the dissimilarity to match the nodes of one graph with the nodes of another, which is formulated as the graph node alignment problem. This is analogous to the discrete Monge's problem. Therefore, we attempt to transform the graph alignment

problem into an OT problem. Then we define the OT cost as the dissimilarity between two graphs. It is assumed that we have two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$. We embed two graphs into the same metric space (Ω_ϕ, d_ϕ) by computing ϕ for all nodes. All the nodes of the two graphs are located respectively at points in $x_1, x_2, \ldots, x_{n_{\mathcal{G}_1}}$ and $y_1, y_2, \ldots, y_{n_{\mathcal{G}_2}}$. We here consider each node of $\mathcal{G}_1$ as the starting point and each node of $\mathcal{G}_2$ as the endpoint. The two histograms of $\mathbf{a}$ and $\mathbf{b}$ are defined respectively in the probability simplex $\triangle_{n_{\mathcal{G}_1}}$ and $\triangle_{n_{\mathcal{G}_2}}$. Furthermore, as a discrete measure $\mu(\mathcal{G}_1)$ with weights $\mathbf{a}$ on the locations $\{x_1, x_2, \ldots, x_{n_{\mathcal{G}_1}}\}$, we write $\mu(\mathcal{G}_1) = \sum_{i=1}^{n_{\mathcal{G}_1}} \mathbf{a}_i \delta_{x_i}$. $\mu(\mathcal{G}_1)$ denotes the distribution of nodes on (Ω_ϕ, d_ϕ). Similarly, we define the measure $\mu(\mathcal{G}_2) = \sum_{j=1}^{n_{\mathcal{G}_2}} \mathbf{b}_j \delta_{y_j}$. We obtain $\mathbf{C} \in \mathbb{R}^{n_{\mathcal{G}_1} \times n_{\mathcal{G}_2}}$ by calculating pairwise distances between all samples. Finally, inputting d_ϕ into $\mathbf{C}_{i,j}$ in Eq. (2), one obtains $\mathbf{P}^*$ by calculating

$$\mathbf{P}^* = \underset{\mathbf{P} \in \mathbf{U}(\mathbf{a}, \mathbf{b})}{\operatorname{argmin}} \sum_{i=1}^{n_{\mathcal{G}_1}} \sum_{j=1}^{n_{\mathcal{G}_2}} d_\phi \left(T(v_i), T(v_j) \right) \mathbf{P}_{i,j}. \tag{5}$$

Consequently, letting $[d(v_i, v_j)]_{i,j}^{m,n}$ be a $\mathbf{C} \in \mathbb{R}^{m \times n}$ that is calculated using distance $d(\cdot, \cdot)$, the proposed graph Wasserstein distance is derived as $\mathcal{W}_1 \left(\mu(\mathcal{G}_1), \mu(\mathcal{G}_2); [d_\phi(v_i, v_j)]_{i,j}^{n_{\mathcal{G}_1}, n_{\mathcal{G}_2}} \right)$. It should be noted that an important feature of OT is that it provides a distance between probabilities when the cost matrix satisfies the axiom of distance. In fact, $d_\phi(\cdot, \cdot)$ is a metric measure; thus, we can conclude that the $\mathcal{W}_1 \left(\mu(\mathcal{G}_1), \mu(\mathcal{G}_2); [d_\phi(v_i, v_j)]_{i,j}^{n_{\mathcal{G}_1}, n_{\mathcal{G}_2}} \right)$ satisfies the axiom of distance. Also, the EMD with L_1 as the ground distance equals to EMD-L_1. The EMD-L_1 can be computed with a time complexity $\mathcal{O}(n^2)$ (Ling & Okada, 2007).

Algorithm 1 Algorithm for Wasserstein Weisfeiler-Lehman Subtree distance

Input: Two graphs $\mathcal{G}_1(\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2(\mathcal{V}_2, \mathcal{E}_2)$
Output: $\langle \mathbf{C}, \mathbf{P}^* \rangle$
1: $\mathbf{a} = 1/n_{\mathcal{G}_1}$; $\mathbf{b} = 1/n_{\mathcal{G}_2}$
2: $\phi(T(v_i)) \leftarrow v_i$ for all $v_i \in \mathcal{V}_1$
3: $\phi(T(v_j)) \leftarrow v_j$ for all $v_j \in \mathcal{V}_2$
4: $\mathbf{C} \leftarrow d_\phi(\phi(T(v_i)), \phi(T(v_j)))$ for all i, j combinations.
5: $\mathbf{P}^* \leftarrow$ Eq. (5)
6: $\langle \mathbf{C}, \mathbf{P}^* \rangle = \sum_{i=1}^{n_{\mathcal{G}_1}} \sum_{j=1}^{n_{\mathcal{G}_2}} \mathbf{C}_{i,j} \mathbf{P}^*_{i,j}$
7: **return** $\langle \mathbf{C}, \mathbf{P}^* \rangle$

5 EXPERIMENTS

This section presents graph classification experiments using both kernel and distance methods to evaluate our model. We use PyTorch Geometric (Fey & Lenssen, 2019) to implement our model.

5.1 EXPERIMENTAL SETUP

Datasets. We use the popular TUD benchmark datasets (Morris et al., 2020) for graph classification tasks. Among them, we use eight commonly used benchmarks datasets. There are six bioinformatics datasets (MUTAG, PTC-FM, PTC-MR, BZR, and PROTEINS) and two social network datasets (IMDB-B and IMDB-M), respectively. In the bioinformatic graph, the nodes are assigned categorical labels. However, they are not in the social network graph; thus, we create categorical labels using their node degrees. For more detailed information about the above datasets, including the number of graphs, nodes, and edges, see Appendix.

Models and experimental settings. We evaluate the performance of WWLS through graph classification experiments compared with several representative methods. As a classification method, we use a multi-class Support Vector Machine (SVM) (Chang & Lin, 2011) in the first graph classification experiment and use 1-Nearest-Neighbours (1-NN) in the second graph classification experiment. Especially for the first experiment, we construct an indefinite kernel matrix $\exp\left(-\gamma \mathcal{W}_1 \left(\mu(\mathcal{G}_1), \mu(\mathcal{G}_2); [d_\phi(v_i, v_j)]_{i,j}^{n_{\mathcal{G}_1}, n_{\mathcal{G}_2}} \right)\right)$ for WWLS, which is the same form as FGW's kernel. It is an exponential kernel and can be considered a noisy observation of the true positive

Table 1: Classification accuracies of experiment using graph kernels (1)

METHOD	MUTAG	PTC-FM	PTC-MR	BZR
SP	83.24±8.81	63.35±5.19	59.34±8.78	86.53±4.55
RW	74.68±9.08	65.04±5.24	56.36±5.01	78.77±1.01
FGW RAW SP	79.20±8.45	64.81±5.19	56.57±6.84	78.77±1.01
WL	79.32±7.91	62.78±7.67	62.24±8.10	**88.58±4.02**
WL-OA	82.72±7.09	64.93±6.31	63.46±8.63	87.98±3.68
WWL	85.91±7.40	**66.59±7.19**	**65.32±7.06**	88.38±3.84
GIN	**88.60±6.90**	**66.70±7.41**	64.77±7.68	87.76±4.74
WWLS	**88.78±6.92**	65.13±6.91	**67.44±8.50**	**88.45±4.09**

Table 2: Classification accuracies of experiment using graph kernels (2)

METHOD	PROTEINS	ENZYMES	IMDB-B	IMDB-M
SP	**76.02±3.81**	41.30±4.89	57.92±5.05	39.30±3.43
RW	66.60±2.86	28.65±5.23	70.88±4.35	47.21±3.91
FGW RAW SP	59.57±0.17	16.67±0.00	50.00±0.00	33.33±0.00
WL	74.25±3.76	53.17±5.94	72.77±4.31	50.44±3.73
WL-OA	73.84±3.61	**60.23±5.50**	72.87±3.93	50.33±3.83
WWL	74.13±3.48	57.40±6.63	72.75±4.02	50.64±4.24
GIN	73.72±4.27	47.00±5.43	**74.88±4.17**	**51.25±4.13**
WWLS	**74.56±3.75**	**59.40±6.28**	**74.59±3.97**	**51.29±4.14**

Table 3: Classification accuracies of experiment using distance methods (1)

METHOD	MUTAG	PTC-FM	PTC-MR	BZR
FGW RAW SP	75.52±8.69	57.64±6.85	56.63±7.85	78.63±4.90
WWL	86.87±6.94	60.44±7.60	63.01±7.60	85.27±5.23
WWLS	**88.03±6.53**	**61.21±7.37**	**65.13±7.97**	**87.19±4.53**

Table 4: Classification accuracies of experiment using distance methods (2)

METHOD	PROTEINS	ENZYMES	IMDB-B	IMDB-M
FGW RAW SP	63.32±3.78	29.00±5.06	**69.61±4.51**	41.69±4.32
WWL	65.27±4.27	61.98±5.93	67.99±4.00	41.49±4.11
WWLS	**66.50±4.13**	**64.05±5.68**	68.43±4.73	**41.73±4.76**

semidefinite kernel (Luss & d'Aspremont, 2009). The machine environment is macOS BigSur, Intel Core i5 CPU with 2.3GHz, and 8GB RAM. The parameters used in each model and their respective settings are presented in the following paragraphs.

First, we introduce the experiments with graph kernels. We compare WWLS as a kernel method with other models in two categories: graph kernels and GNNs. As for graph kernels, we choose a few representative methods: Shortest Path (SP) kernel (Borgwardt & Kriegel, 2005), Random Walk (RW) kernel Vishwanathan et al. (2010), Weisfeiler-Lehman (WL) kernel, Weisfeiler-Lehman Optimal Assignment (WL-OA) kernel (Kriege et al., 2016), Wasserstein Weisfeiler-Lehman (WWL) kernel, and Fused Gromov-Wasserstein (FGW) distance. Among them, WL-OA and WWL are extension models of the WL kernel, and FGW uses an indefinite kernel matrix $\exp\left(-\gamma FGW\left(\mathcal{G}_1, \mathcal{G}_2\right)\right)$ that we mentioned above. For models other than WWL and FGW, we use GraKel, a library that provides implementations of several well-established graph kernels (Siglidis et al., 2020). For another

category, we use Graph Isomorphism Network (GIN) (Xu et al., 2019), which theoretically has the same expressive power as the WL test. Next, we introduce the parameter settings of each model in the experiment: (1) for the WL, WL-OA, and WWL, we adjust the number of WL iterations within $\{1, 2, \ldots, 10\}$; (2) for the WWL, to evaluate the performance of extracting structural information, we choose the case of categorical embeddings; (3) for the WWLS, we adjust the number of WL iterations within $\{1, 2, \ldots, 7\}$, and when using IMDB-B and IMDB-M, we set $\{1, 2\}$ due to their large node degrees; (4) for the FGW, we use the shortest path and Hamming distance; we adjust the parameter α of FGW within $\{0.2, 0.5, 0.8\}$; (5) for all graph kernels based on the exponential kernel, we adjust the parameter γ within $\{0.0001, 0.001, 0.01, 0.1, 1, 10\}$; (4) for the GIN, we choose pairs of the number of hidden layers and hidden dimension from the range of $\{1, 2, 3, 4, 5\}$ and $\{32, 64, 128\}$, respectively. Also, we cross-validate the parameter C of SVM within $\{0.001, 0.01, 0.1, 1, 10, 100, 1000\}$. In the first experiment, we prepare 10 fixed random states and perform 10-fold nested cross-validation (using 9 folds for training with parameter adjustment, 1 for testing) on each state; thus, the overall average of 100 classification tests is used as the result for each model. For the GIN, we also perform ten times 10-fold cross-validation. For each training, we run 200 epochs for parameter adjustments and then choose the best parameter pair for testing. In the second experiment, we use 1-NN to classify graphs. As comparison methods, we choose WWL and FGW, which are also the same distance methods. We set the same conditions as the first classification experiment for the number of WL iterations and the settings of FGW. In the same way, we perform ten times 10-cross validation and let the average of 100 tests be the result for each model.

5.2 Results and discussion

Table 1 and Table 2 summarize the results of the first classification experiment with graph kernels, respectively. The first and second-best results in Tables 1 and Table 2 are shown in **red** and **purple**, respectively. Table 3 and Table 4 summarize the results of the graph classification experiment with the distance method, respectively. The best result in Table 3 and Table 4 are shown in **bold**. We analyze the results of these two experiments in the following.

The first experiment compares WWLS as a graph kernel method with other well-known graph kernel methods. As Table 1 and Table 2 show, WWLS yields the first or second-best results in datasets other than PTC-FM: WWLS gives the best results in MUTAG, PTC-MR, and IMDB-M; WWLS gives the second-best results in BZR, PROTEINS, ENZYMES, and IMDB-B. We find that WWLS outperforms WL-based methods overall and even rivals GIN in these mainstream datasets. The second experiment compares WWLS as a distance method between two graph structures with WWL and FGW. Both WWL and FGW use the OT framework. As Table 3 and Table 4 show, WWLS performs better than WWL and FGW as a distance method except for IMDB-B. The accuracy trend of WWLS on the dataset is similar to that of WWL. This indicates that WWLS obtains similar structural information to WWL; however, in terms of accuracy, WWLS captures more structural information than WWL. From these observations, WWLS is superior to traditional methods to measure the distance between graph structures.

6 Conclutions

Traditional WL-based algorithms use hash values instead of WL subtrees, which results in the loss of important graph structure information and limits the ability to measure the differences between graph structures. The proposed WWLS successfully defines a valid distance between nodes by directly using WL subtrees as node features and measuring node dissimilarity using approximate tree edit distance. In addition, we use the OT framework to define a valid distance between graphs by reflecting the node distance at the graph level. As an avenue of future study, WWLS needs to reduce the computation time. Although the results are higher than WL-based methods in the above experiments, constructing the WL subtree and the searching tree is relatively time-consuming. Hence, it needs to consider an approach to omit the tree construction and directly obtain the substructure obtained by WWLS. Since the complete subtrees of the WL subtree have various structural information, it also needs to study which substructures are most important for graph classification.

REFERENCES

Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. *Journal of the ACM (JACM)*, 50(4):429–443, 2003.

László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*, pp. 684–697, 2016.

Muhammet Balcilar, Pierre Héroux, Benoit Gaüzère, Pascal Vasseur, Sébastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *IEEE international conference on data mining (ICDM)*, pp. 8–pp, 2005.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems (NeurIPS)*, 26:2292–2300, 2013.

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations (ICLR)*, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Daiji Fukagawa, Tatsuya Akutsu, and Atsuhiro Takasu. Constant factor approximation of edit distance of bounded height unordered trees. In *International Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 7–17. Springer, 2009.

Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

Minos Garofalakis and Amit Kumar. Xml stream processing using tree-edit distance embeddings. *ACM Transactions on Database Systems (TODS)*, 30(1):279–332, 2005.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (ICKD)*, pp. 855–864, 2016.

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

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *International World Wide Web Conference (WWW)*, pp. 2704–2710, 2020.

Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. *Advances in Neural Information Processing Systems (NeurIPS)*, 29:1623–1631, 2016.

Haibin Ling and Kazunori Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 29(5): 840–853, 2007.

Ronny Luss and Alexandre d'Aspremont. Support vector machine classification with indefinite kernels. *Mathematical Programming Computation*, 1(2):97–118, 2009.

Facundo Memoli. On the use of Gromov-Hausdorff Distances for Shape Comparison. In M. Botsch, R. Pajarola, B. Chen, and M. Zwicker (eds.), *Eurographics Symposium on Point-Based Graphics*. The Eurographics Association, 2007.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *2009 IEEE 12th international conference on computer vision (ICCV)*, pp. 460–467, 2009.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (ICKD)*, pp. 701–710, 2014.

Gabriel Peyré, Marco Cuturi, and Justin Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning (ICML)*, pp. 2664–2672, 2016.

Kaspar Riesen. Structural pattern recognition with graph edit distance. In *Advances in computer vision and pattern recognition*. Springer, 2015.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research (JMLR)*, 12(9), 2011.

Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *Journal of Machine Learning Research (JMLR)*, 21(54):1–5, 2020.

Reza Takapoui and Stephen Boyd. Linear programming heuristics for the graph isomorphism problem. *arXiv preprint arXiv:1611.00711*, 2016.

Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler–lehman graph kernels. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6436–6446. Curran Associates, Inc., 2019.

Titouan Vayer, Laetitia Chapel, Rémi Flamary, Romain Tavenard, and Nicolas Courty. Fused gromov-wasserstein distance for structured objects. *Algorithms*, 13(9):212, 2020.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer, 2009.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research (JMLR)*, 11:1201–1242, 2010.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *International World Wide Web Conference (WWW)*, pp. 2022–2032, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (ICKD)*, pp. 974–983, 2018.

A APPENDIX

A.1 SCHWARTZ-ZIPPEL LEMMA

Lemma 1 (Schwartz-Zippel lemma (Agrawal & Biswas, 2003))**.** *Let F be a filed. Let $p(x_1, x_2, \ldots, x_n)$ be a polynomial of total degree d. Suppose that f the non-zero polynomial and S is a finite subset of F. Let $x'_1, x'_2, \ldots, x'_n$ be chosen at random uniformly and independently from S. Then the probability that $p(x'_1, x'_2, \ldots, x'_q) = 0$ is less than or equal to $\frac{d}{|S|}$, that is* $\Pr(p(x'_1, x'_2, \ldots, x'_q) = 0) \leq \frac{d}{|S|}$.

A.2 DATASET INFORMATION

Table 5 shows the specific information of the dataset we used in our experiments.

Table 5: Dataset Details table

Name	Graphs	Classes	Avg. Nodes	Avg. Edges	Node Labels
MUTAG	188	2	17.93	19.79	✓
PTC-FM	349	2	14.11	14.48	✓
PTC-MR	344	2	14.29	14.69	✓
BZR	405	2	35.75	38.36	✓
PROTEINS	1113	2	39.06	72.82	✓
ENZYMES	600	6	32.63	62.14	✓
IMDB-B	1000	2	19.77	96.53	-
IMDB-M	1500	3	13.00	65.94	-

A.3 NUMBER OF COMPLETE SUBTREES

For several datasets, we show the relationship between the number of complete subtree types and the number of WL iterations in Table 6.

Table 6: Number of complete subtree types for each WL iteration number

WL iterations	MUTAG	PTC-FM	PTC-MR	ENZYMES	IMDB-B
1	29	105	114	55	1771
2	50	191	209	106	3476
3	71	277	304	157	5181
4	92	363	399	208	-
5	113	449	494	259	-
6	134	535	589	-	-