Tree Structure for the Categorical Wasserstein Weisfeiler-Lehman Graph Kernel

Anonymous authors
Paper under double-blind review

Abstract

The Wasserstein Weisfeiler-Lehman (WWL) graph kernel is a popular and efficient approach, utilized in various kernel-dependent machine learning frameworks for practical applications with graph data. It incorporates optimal transport geometry into the Weisfeiler-Lehman graph kernel, to mitigate the information loss inherent in aggregation strategies of graph kernels. While the WWL graph kernel demonstrates superior performances in many applications, it suffers a drawback in its computational complexity, i.e., at least $\mathcal{O}(n_1 n_2)$, where n_1, n_2 denote the number of vertices on input graphs. Consequently, it hinders the practical applicability of WWL graph kernel, especially in large-scale settings. In this paper, we propose the Tree Wasserstein Weisfeiler-Lehman (TWWL) graph kernel, which leverages *tree structure* to scale up the exact computation of the WWL graph kernel for graph data with categorical node labels. In particular, the computational complexity of TWWL graph kernel is $\mathcal{O}(n_1 + n_2)$, enabling for its applications with large-scale graphs. Numerical experiments demonstrate that performances of the proposed kernel compare favorably with those baseline kernels, while its computation is several-order faster than the classic WWL graph kernel, paving ways for its applications in large-scale datasets where the WWL kernel is computationally prohibited.

1 Introduction

Graph data are a popular structure and play a fundamental role in many applications (Barabási & Oltvai, 2004; Pržulj, 2007; Barabási & Pósfai, 2016; Petric Maretic et al., 2019; Xu et al., 2019; Nguyen et al., 2021; Chen et al., 2022; Vincent-Cuaz et al., 2022; Le et al., 2022; Kong et al., 2023). To incorporate such structured data into machine learning frameworks, several graph kernels (Vishwanathan et al., 2010; Borgwardt et al., 2020; Kriege et al., 2020; Nikolentzos et al., 2021) have been proposed in the literature, which measure similarity of input graphs by aggregating their local features, e.g., histogram kernels (Nikolentzos et al., 2021) compare multisets counting on nodes or edges, path-based kernels (Borgwardt & Kriegel, 2005; Feragen et al., 2013) accumulate similarities along shortest paths, and graphlet kernels (Pržulj, 2007; Shervashidze et al., 2009) evaluate the frequency of small subgraph structures. Despite their effectiveness, these kernels face certain significant challenges. A primary limitation is their limited scalability to the number of nodes in input graphs. For examples, the shortest path kernel relies on the pair-wise shortest paths between all graph nodes, which typically requires $\mathcal{O}(n^3)$, where n is the number of nodes in a graph. For the k-node graphlet kernel, it requires about $\mathcal{O}(n^k)$ induced subgraphs, forcing users either to perform an exhaustive count or to approximate it by sampling methods, yet both of which become prohibitive even for a moderate k.

To overcome this challenge, and to enhance the expressiveness of graph kernels, Shervashidze et al. (2011) proposed the Weisfeiler-Lehman (WL) kernel. By coupling iterative neighborhood aggregation with relabeling, the WL kernel achieves neighborhood-aware representations while retaining computational efficiency. Moreover, Togninalli et al. (2019) leverage optimal transport geometry to enrich the representational power of the WL kernel to propose the Wasserstein Weisfeiler-Lehman (WWL) kernel. Notice that several traditional graph kernels, e.g., graphlet kernels, shortest-path kernels, and WL subtree kernel (Shervashidze et al., 2011), typically enumerate substructures and then aggregate them by frequency counting or averaging. Such pooling approach collapses the empirical distribution of substructures to its first-order statistics,

eliminating information about how different patterns co-occur or how their attributes are distributed. The WWL kernel overcomes this information loss by comparing the entire distributions of node embeddings via the Wasserstein distance (Peyré & Cuturi, 2020), thus retaining intricate structural information.

Although the WWL graph kernel strengthens the ability to capture complex graph structures, the Wasserstein distance within its computation process brings up concerns about its scalability (i.e., super cubic computational complexity). Since this challenge stems from the inherent difficulty in optimal transport (OT) theory, various acceleration strategies have been explored in many domains. There are two main acceleration strategies for the computation of Wasserstein distance: (i) the sliced Wasserstein distance (SWD) (Bonneel et al., 2015), which relies on one-dimensional projections for supports of input measures, and exploits the closed-form expression of univariate OT; and (ii) an entropic regularization approach (Cuturi, 2013), which can be solved efficiently by the celebrated Sinkhorn algorithm (Sinkhorn, 1964). Extensive research has been conducted related to these approaches. Perez et al. (2024) proposed to utilize the SW approach to *approximate* the WWL kernel, to scale up its computation, which is applicable to large-scale datasets with continuous node attributes. In the Sinkhorn algorithm, various proposals have been introduced in the literature to further reduce its computational cost, including a greedy strategy (Altschuler et al., 2017), kernel matrix sparsification (Li et al., 2023), and low-rank factorization of the coupling (Scetbon et al., 2021). Additionally, Tang et al. (2024) proposed to consider a Newton-type method with sparse Hessian matrix to accelerate its convergence.

In contrast to the above strategies, by restricting the space to a tree structure, it has been shown that the Wasserstein distance can be expressed in closed form (Indyk & Thaper, 2003; Le et al., 2019b). This variant is commonly referred to as the Tree Wasserstein distance (TWD). The computational efficiency of the TWD has been leveraged in various applications. Le et al. (2019b) considered the TWD as a generalization of the univariate Wasserstein distance, and then proposed the tree-sliced Wasserstein distance as a variant of the SWD. As the choice of a tree structure plays a crucial role in the TWD, Yamada et al. (2022) proposed an algorithmic approach for estimating the edge weights to approximate the Wasserstein distance. Lin et al. (2025) addressed the issue on how to construct a tree by embedding features into a multiscale hyperbolic space via diffusion geometry, decoding the latent feature hierarchy into an explicit tree, and defining the TWD on the learned tree. Acceleration algorithms leveraging the TWD have also been proposed for the Wasserstein barycenter problem (Le et al., 2019a; Takezawa et al., 2022) and for metric learning using Wasserstein singular vectors (Düsterwald et al., 2025).

Recent work by Perez et al. (2024) introduced an approximation method that speeds up the WWL kernel when continuous node attributes are involved, and generic Sinkhorn-based accelerations mentioned above can be also applied to the same scenario. Exploiting an effective acceleration approach to graphs with categorical node labels would therefore position the WWL kernel as a flexible and scalable solution applicable to a diverse range of scenarios, including large-scale graph datasets. In this paper, we address the acceleration of the WWL graph kernel for datasets with categorical node labels by utilizing the TWD. Specifically, we first show that the labels generated by the WL algorithm inherently exhibit a tree structure, and that the resulting path metric on the tree exactly preserves the Hamming distance between the labels. This results in a closed-form expression to calculate the WWL kernel for categorical node labels. One notable advantage of our approach over the Sinkhorn approach is its predictable runtime. Whereas the Sinkhorn solver requires a quadratic complexity over the number of iterations to converge, the computational cost of our approach is fixed once the WL algorithm is completed.

Contribution. Our contribution is three-fold:

1. We present that the relationship among generated labels by the WL algorithm forms a tree structure.
2. We show that the proposed algorithm, *Tree Wasserstein Weisfeiler-Lehman* algorithm, is an exact and accelerated method for computing the WWL graph kernel designed for graph datasets with categorical node labels.
3. Numerical experiments demonstrate that the computational complexity of the proposed algorithm outperforms the original WWL kernel algorithm without sacrificing its performances. Moreover, our

algorithm makes it feasible to apply the WWL kernel to large-scale categorical-label graph datasets that are computationally prohibited for the original WWL graph kernel algorithm.

Organization. The paper is organized as follows: Section 2 is devoted to notations in this paper. In Section 3, we review the related works that are necessary to construct our algorithm. Our algorithm and its properties are described in Section 4. Section 5 presents the numerical experiments, and we conclude the paper in Section 6.

2 Notations

In this section, we introduce the notations used in subsequent sections. Let $\mathcal{G} = (V, E)$ be an undirected graph, where V and E denote the set of vertices and edges, respectively. We write the neighborhood of a vertex v by $\mathcal{N}_v \subset V$. We denote a set of initial categorical labels by Σ_0, and say that a graph is *labeled* if each of its vertices is assigned a label from Σ_0.[1] Let $\boldsymbol{\mathcal{G}} \coloneqq \{\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_N = (V_N, E_N)\}$ be a set of graphs, N be the number of graphs, n_i be the number of vertices in $\mathcal{G}_i$, $n \coloneqq \sum_{i=1}^{N} n_i$ be the total number of vertices in $\boldsymbol{\mathcal{G}}$, and $\bar{n} \coloneqq \frac{n}{N}$ be the average number of vertices per graph. Throughout this paper, we assume that all graphs in $\boldsymbol{\mathcal{G}}$ are labeled and share the same label set Σ_0. Furthermore, we write $\mathcal{V} \coloneqq \bigcup_{i=1}^{N} V_i$. A function $l_0 : \mathcal{V} \to \Sigma_0$ is used to denote the initial label of vertices.

A sequence of vertices $(v_1, \ldots, v_k)$ in a graph $\mathcal{G}$ is called a *walk* if an edge exists between every pair of consecutive vertices. If all vertices in the walk are distinct, the walk is called a *path*. A sequence $(v_1, \ldots, v_k)$ is called a *cycle* if its first and last vertices coincide, the vertices $v_1, \ldots, v_{k-1}$ are distinct, and for every pair of consecutive vertices there exists an edge between them. A graph is *connected* if, for every pair of vertices there exists a path. We denote an edge weight function by $w : E \to \mathbb{R}_{\geq 0}$.

A connected graph without cycles is called a *tree*, and is denoted by $\mathcal{T}$. Note that for any pair of vertices in a tree, there exists a unique path connecting them. This is because the existence of two distinct paths would necessarily imply the presence of a cycle, contradicting the definition of a tree. Unless it leads to ambiguity, we use $\mathcal{T}$ to denote the set of its nodes, and $E(\mathcal{T})$ to denote the set of edges in the tree. We write $\mathcal{P}(u, v)$ as a set of edges that constitute the path between $u, v \in \mathcal{T}$. Given a tree $\mathcal{T}$ with a weight function w, the *path metric* between $u, v \in \mathcal{T}$ is defined as $d_{(\mathcal{T},w)}(u, v) \coloneqq \sum_{e \in \mathcal{P}(u,v)} w_e$. We denote a tree rooted at r by $\mathcal{T}_r$, and a set of nodes in a subtree rooted at u by $\Gamma_u \coloneqq \{v \in \mathcal{T}_r \mid u \in \mathcal{P}(v, r)\}$. For an edge $e \in E(\mathcal{T}_r)$, we write $u_e \in \mathcal{T}_r$ for the node closer to the root and $v_e \in \mathcal{T}_r$ for the node farther away from the root node.

Figure 1 illustrates the notations of path metric, subtree, and edge endpoints introduced above. The path metric between vertices y and z can be calculated as $d_{(\mathcal{T}_r,w)}(y, z) = w_1 + w_2 + w_3$, the subtree rooted at y is indicated by the region outlined in blue, and Γ_y denotes the set of vertices contained within the region. For the edge e that joins vertices x and z, the endpoint closer to the root is $u_e = x$, whereas the endpoint farther from the root is $v_e = z$.

3 Related work

Our algorithmic approach accelerates the categorical WWL graph kernel, which combines the WL algorithm with the Wasserstein distance, by leveraging the tree structure over labels within the WL procedure. In this section, we review the necessary background and related works that support our approach.

3.1 Wasserstein distance

The Wasserstein distance (Villani, 2009; Peyré & Cuturi, 2020) is a metric between probability measures supported on a given metric space. Let (Ω, d) be a metric space and μ, ν be probability measures supported

[1] For graphs without node labels, we regard them assigned with dummy labels to each vertex.

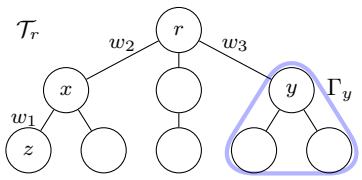

Figure 1: An example of tree notations.

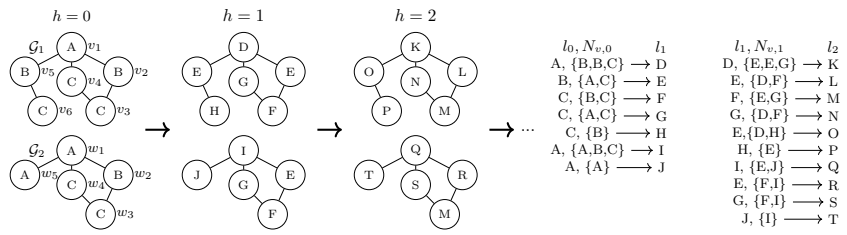

Figure 2: An example of the WL algorithm for given graphs with $\Sigma_0 = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$.

on Ω. The L^p-Wasserstein distance for $p \geq 1$ is defined as

$$W^p(\mu, \nu) \coloneqq \left(\inf_{\pi \in \Pi(\mu,\nu)} \int d(x,y)^p \pi(dx, dy) \right)^{1/p},$$

where π is a probability measure on $\Omega \times \Omega$ whose marginals are μ and ν, called a *coupling*, and $\Pi(\mu, \nu)$ is the set of all couplings of μ and ν. In this paper, we focus on the L^1-Wasserstein distance, which we denote by W_d, i.e., optimal transport with ground cost d.

3.2 Weisfeiler-Lehman algorithm

Shervashidze et al. (2011) proposed to incorporate the concept of the Weisfeiler-Lehman test (Weisfeiler & Leman, 1968) into graph kernels. The WL test is a procedure for graph isomorphism problem that iteratively refines vertex labels based on their neighborhood structure. This approach yields features that are both highly expressive, by capturing latent graph structure, and computationally efficient, thereby addressing the scalability issues that hindered various graph kernels.

The WL algorithm iteratively assigns a new label to each vertex, treating the resulting sequence of labels as the feature of the vertex. Let $H \in \mathbb{N}$ denote the maximum number of iterations, Σ_h denote the set of all labels generated at the h-th iteration, and $\Sigma \coloneqq \bigcup_{h=0}^{H} \Sigma_h$ denote the set of all labels. At the h-th iteration in the WL algorithm, the new label for vertex v is generated as follows, based on its previous label and the set of labels in its neighborhood:

$$l_h(v) \coloneqq \mathrm{hash}\left(l_{h-1}(v), \mathcal{N}_{v,h-1}\right), \tag{1}$$

where we write $\mathcal{N}_{v,h-1} \coloneqq \{l_{h-1}(u) \mid u \in \mathcal{N}_v\}$ and handle $\mathcal{N}_{v,h-1}$ in its sorted form. The hash function (Cormen et al., 2009) is defined as a function commonly used in computer science that returns the same output for identical inputs and returns the distinct value for different inputs. A general description of hash functions is provided in Appendix A. Lemma 1 immediately follows from the definition of the hash function.

Lemma 1. *For every $\sigma \in \Sigma_h$, there exists a unique label sequence $(\sigma_0, \ldots, \sigma_{h-1})$, $\sigma_i \in \Sigma_i$ that generates σ.*

Figure 2 illustrates the iterative process of the WL algorithm on labeled graphs $\boldsymbol{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2\}$. In this figure, the new labels generated at each step are represented by Latin characters that are not used in the previous steps, reflecting the injectivity of the hash function. Vertices v_2 and v_5 start with identical labels and the same multiset of neighbor labels, meaning that they remain indistinguishable after the first iteration. However, as structural information from farther neighborhoods is propagated, their local contexts diverge. This divergence leads to them being assigned distinct labels at $h = 2$. This example demonstrates how the WL algorithm effectively propagates structural information to eventually differentiate vertices that initially appear identical from a local viewpoint.

3.3 Weisfeiler-Lehman subtree kernel

The WL subtree kernel, introduced by Shervashidze et al. (2011), is a member of a graph kernel family that leverages the WL algorithm. Based on the label set Σ generated by the WL algorithm, let $c(\sigma, \mathcal{G})$ denotes

the count of a label σ in $\mathcal{G}$. Then, the WL subtree feature for $\mathcal{G}$ is defined as

$$\phi(\mathcal{G}) \coloneqq \left(c(\sigma_1, \mathcal{G}), \ldots, c(\sigma_{|\Sigma|}, \mathcal{G})\right).$$

We use $\langle \cdot, \cdot \rangle$ to denote an inner product, then the WL subtree kernel on two graphs $\mathcal{G}_1, \mathcal{G}_2$ is defined as

$$k_{\mathrm{WLsubtree}}(\mathcal{G}_1, \mathcal{G}_2) \coloneqq \langle \phi(\mathcal{G}_1), \phi(\mathcal{G}_2) \rangle.$$

The WL subtree kernel is positive semidefinite and can be computed in $\mathcal{O}(Hm)$, where m is the total number of edges in $\mathcal{G}_1$ and $\mathcal{G}_2$.

3.4 Wasserstein Weisfeiler-Lehman graph kernel

Most classical graph kernels aggregate substructures by simple pooling approach, which reduces the empirical distribution to first-order statistics and discards information about pattern co-occurrence and attribute variation. To reflect this information in the comparison, Togninalli et al. (2019) proposed the WWL kernel. The key idea is to treat the multiset of WL node labels as an empirical distribution over the label space and to compare similarity between two graphs via the Wasserstein distance on that space for the WWL kernel.

For labeled graphs, the ground distance between each pair of vertices $v \in V_i,\ w \in V_j$ is defined by the Hamming distance between their WL label sequences:

$$d_{\mathrm{Ham}}(v, w) \coloneqq \sum_{h=0}^{H} \rho\left(l_h(v), l_h(w)\right), \quad \text{where} \quad \rho(a, b) = \begin{cases} 1 & a \neq b, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Due to the uniqueness of the label sequence that generates a label $\sigma \in \Sigma_H$ and the definition of the hash function, the Hamming distance d_{Ham} can be regarded as a metric on Σ_H. Assigning a uniform probability measure to each vertex induces empirical measures on that label space. Let μ, ν be probability measures on Σ_H for $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively. The Wasserstein distance $W(\mathcal{G}_1, \mathcal{G}_2)$ is then computed by

$$W(\mathcal{G}_1, \mathcal{G}_2) \coloneqq W_{d_{\mathrm{Ham}}}(\mu, \nu) = \inf_{\pi \in \Pi(\mu, \nu)} \int d_{\mathrm{Ham}}(x, y) \pi(dx, dy), \tag{3}$$

and the corresponding kernel is defined by

$$k_W(\mathcal{G}_1, \mathcal{G}_2) \coloneqq e^{-\lambda W(\mathcal{G}_1, \mathcal{G}_2)}, \tag{4}$$

where λ is a positive hyperparameter. Togninalli et al. (2019) employed either the network simplex algorithm (Orlin, 1997) to compute the Wasserstein distance or the Sinkhorn algorithm (Sinkhorn, 1964; Cuturi, 2013) to approximate it.

3.5 Tree Wasserstein distance

The tree Wasserstein distance was initially introduced by Indyk & Thaper (2003), and implicitly utilized in *UniFrac* (Lozupone & Knight, 2005), which quantifies the similarity between two microbial communities on a reference phylogenetic tree, then was rigorously formalized by Evans & Matsen (2012). When the ground space is a weighted tree, the 1-order Wasserstein distance admits a closed form that can be evaluated in $\mathcal{O}(|E|)$.

Theorem 1 ((Evans & Matsen, 2012; Le et al., 2019b)). *Let $d_{\mathcal{T}}$ be a path-length metric on $\mathcal{T}$ and μ, ν be probability measures on $\mathcal{T}$. Then, the Wasserstein distance can be written as follows:*

$$W_{d_{\mathcal{T}}}(\mu, \nu) = \sum_{e \in E(\mathcal{T})} w_e \left| \mu(\Gamma(v_e)) - \nu(\Gamma(v_e)) \right|.$$

Computing this expression requires only a single post-order traversal of the tree. When the tree degenerates to a path, the formula reduces to the standard univariate Wasserstein distance, highlighting TWD as a natural generalization.

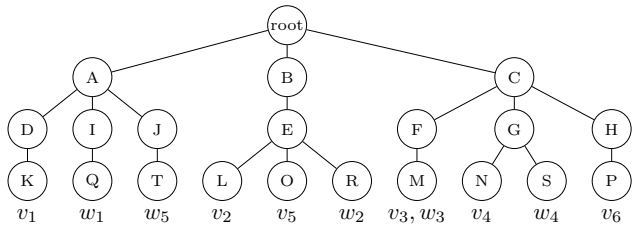

Figure 3: An illustration of a label tree based on graphs in Figure 2.

4 Main Algorithm

We propose the *Tree Wasserstein Weisfeiler-Lehman* (TWWL) algorithm that enables us to calculate the Wasserstein distance based on the Hamming distance precisely, efficiently and without approximation by regularization. In our algorithm, we focus on the relationship between $l_h(v)$ and $l_{h-1}(v)$ in Equation 1.

Definition 1 (Parent-Child relationship among labels)**.** For $\sigma \in \Sigma_h$ and $\tau \in \Sigma_{h+1}$, if there exists a label set $S \subset \Sigma_h$ such that $\tau = \mathrm{hash}(\sigma, S)$, then σ is considered a parent label of τ, or equivalently, τ is a child label of σ.

It is clear from the property of the hash function that the following holds.

Lemma 2. *Let u, v be arbitrary vertices in $\mathcal{V}$. If there exists $h_0 < H$ such that $l_{h_0}(u) \neq l_{h_0}(v)$, then $\forall h \in \{h_0, \ldots, H\}$, $l_h(u) \neq l_h(v)$ holds.*

The TWWL algorithm yields a tree structure for labels, called the *Label Tree*, and applies the tree Wasserstein distance. The steps for constructing a label tree are described in Algorithm 1. The label tree is constructed to satisfy the following properties: (i) except for the root node, each node is in one-to-one correspondence with a label in Σ; (ii) for every label in Σ_0, there is an edge connecting it to the root node; (iii) an edge exists between any two labels that satisfy the relation defined in Definition 1; (iv) each edge is assigned a weight of 1/2.

Figure 3 illustrates the label tree constructed from the example in Figure 2. Taking the label tree properties into consideration, one can intuitively verify that the Hamming distance between v_1 and w_1, as defined in Equation 2 with $H = 2$, coincides with the length of the shortest path on the tree. The following two Lemmas provide a formal justification for these claims.

Lemma 3. *A label tree obtained by Algorithm 1 is a tree.*

Proof. We begin by proving the connectivity of the label tree, i.e., that there exists a walk between any $\sigma, \tau \in \Sigma$. By Lemma 1 and the Property (iii) of the label tree, there exist label sequences $(\sigma_0, \ldots, \sigma)$, $(\tau_0, \ldots, \tau)$ that form a path in the label tree. The Property (ii) implies that there exist edges connecting the root to σ_0 and to τ_0. Therefore, a walk exists between σ and τ.

Next, we show that the label tree contains no cycle. Suppose, for the sake of contradiction, that there exists a cycle $(\sigma, \tau_1, \ldots, \tau_k, \sigma)$ in the label tree for some label $\sigma \in \Sigma$. Without loss of generality, assume that σ is the deepest vertex in the cycle. Then, both τ_1 and τ_k must be parent nodes of σ. By a property of the hash function, $\tau_1 = \tau_k$ holds. However, this equality contradicts the definition of a cycle, which requires that the intermediate vertices are distinct. Therefore, no cycle exists in the label tree. □

Lemma 4. *The mapping $\phi : \Sigma_H \to \mathcal{T}$ obtained by Algorithm 1 is distance-preserving.*

Proof. We denote $\phi(\sigma)$ simply as σ. We show that for any $\sigma_H, \tau_H \in \Sigma_H$, $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = d_{(\mathcal{T}, 1/2)}(\sigma_H, \tau_H)$ holds. Lemma 1 implies that there exist label sequences $(\sigma_0, \ldots, \sigma_H)$, $(\tau_0, \ldots, \tau_H)$ that satisfy the parent-child relationship. From Lemma 2, we consider the following three cases:

Case 1: $\forall h \in \{0, \ldots, H\}$, $\sigma_h = \tau_h$. Then, we have both $d_{\mathrm{Ham}}(\sigma_H, \tau_H)$ and $d_{(\mathcal{T}, 1/2)}(\sigma_H, \tau_H)$ are 0.

Algorithm 1 Tree Structure for TWWL

Input: $\boldsymbol{\mathcal{G}}$: a set of undirected graphs, where each vertex v in every graph $\mathcal{G}_i$ is associated with a label $l_0(v) \in \Sigma_0$; H: the maximum number of WL algorithm.

Output: Label tree $\mathcal{T}$ rooted at **root**, and a mapping $\phi : \Sigma_H \to \mathcal{T}$

```
struct NODE
    hash: STRING
    depth: INTEGER
    Γ: VECTOR OF N DOUBLES
    children: VECTOR OF TUPLE {DOUBLE, NODE}
end struct
root ← NODE(_, −1, 0_N, [ ])
que ← empty queue
for G_i in 𝒢 do
    for v in V_i do
        if l_0(v) is included in root.children then
            child ← GET(root.children, l_0(v))
        else
            child ← NODE(l_0(v), 0, 0_N, [ ])
            PUSH(root.children, (1/2, child))
        end if
        ENQUEUE(que, (child, i, v))
    end for
end for
while !ISEMPTY(que) do
    (node, i, v) ← DEQUEUE(que)
    node.Γ_i ← node.Γ_i + 1/n_i
    if node.depth = H then
        CONTINUE
    end if
    h ← node.depth
    if l_{h+1}(v) is included in node.children then
        child ← GET(node.children, l_{h+1}(v))
    else
        child ← NODE(l_{h+1}(v), h + 1, empty list, 0_N)
        PUSH(node.children, child)
    end if
    ENQUEUE(que, (child, i, v))
end while
```

Case 2: $\sigma_0 \neq \tau_0$. Then, it follows from Lemma 2 that $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = H + 1$. Additionally, Lemma 3 implies that the sequence $(\sigma_H, \ldots, \sigma_0, \mathrm{root}, \tau_0, \ldots, \tau_H)$ is a unique path between σ_H and τ_H. Thus, $d_{(\mathcal{T},1/2)}(\sigma_H, \tau_H) = H + 1$.

Case 3: $\exists\, h_0 \in \{1, \ldots, H\}$, $\sigma_{h_0-1} = \tau_{h_0-1}$ and $\sigma_{h_0} \neq \tau_{h_0}$. Then, by Lemma 2, $d_{\mathrm{Ham}}(\sigma_H, \tau_H) = H + 1 - h_0$ is satisfied and the sequence $(\sigma_H, \ldots, \sigma_{h_0}, \sigma_{h_0-1} = \tau_{h_0-1}, \sigma_{h_0}, \ldots, \sigma_H)$ is a unique path between σ_H and τ_H. This implies $d_{(\mathcal{T},1/2)}(\sigma_H, \tau_H) = H + 1 - h_0$.

Therefore, Lemma 4 holds for all cases. □

The above results imply that the Wasserstein distance for categorical WWL graph kernel can be computed efficiently as follows.

Proposition 1. *Suppose that (Σ_H, d_{Ham}) is a metric space and μ, ν are measures supported on Σ_H. Let $(\mathcal{T}, d_{\mathcal{T}}, \phi)$ denote the outputs of Algorithm 1, namely, a label tree, path-length metric, and mapping from Σ_H*

to $\mathcal{T}$, respectively. Then, the Wasserstein distance in Equation 3 can be written as

$$W_{d_{Ham}}(\mu,\nu) = \sum_{e\in E(\mathcal{T})} \frac{1}{2} \left|\phi_\sharp\mu(\Gamma(v_e)) - \phi_\sharp\nu(\Gamma(v_e))\right|,$$

where we let $\phi_\sharp\mu$ denote the push-forward measure of μ by ϕ.

Proof. Since $W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu) = \sum_{e\in E(\mathcal{T})} \frac{1}{2} \left|\phi_\sharp\mu(\Gamma(v_e)) - \phi_\sharp\nu(\Gamma(v_e))\right|$ holds by Theorem 1, we show that $W_{d_{\mathrm{Ham}}}(\mu,\nu) = W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu)$.

First, let π be a coupling of μ and ν. Note that ϕ is injective and measurable by Lemma 4, the mapping $\phi\times\phi$ is a measurable from Σ_H^2 to $\mathcal{T}^2$, and $(\phi\times\phi)_\sharp\pi$ is a push-forward measure on $\mathcal{T}^2$ that is also a coupling of $\phi_\sharp\mu$ and $\phi_\sharp\nu$. Since the mapping ϕ is distance-preserving, we have

$$\begin{aligned}
&\int_{\Sigma_H^2} d_{\mathrm{Ham}}(x,y)\pi(dx,dy) = \int_{\mathcal{T}^2} d_\mathcal{T}(a,b)((\phi\times\phi)_\sharp\pi)(da),\\
&\Rightarrow \inf_{\pi\in\Pi(\mu,\nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(x,y)\pi(dx,dy) \geq \inf_{\gamma\in\Pi(\phi_\sharp\mu,\phi_\sharp\nu)} \int_{\mathcal{T}^2} d_\mathcal{T}(a,b)\gamma(da,db) = W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu).
\end{aligned}$$

Therefore, it follows that $W_{d_{\mathrm{Ham}}}(\mu,\nu) \geq W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu)$.

Next, we show that $W_{d_{\mathrm{Ham}}}(\mu,\nu) \leq W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu)$. Since the map ϕ restricted to its image $\phi(\Sigma_H)$ is bijective, its inverse map $\phi^{-1}:\phi(\Sigma_H)\to\Sigma_H$ exists, and ϕ^{-1} is distance-preserving.

Let γ denotes a coupling of $\phi_\sharp\mu$ and $\phi_\sharp\nu$, then

$$\begin{aligned}
\int_{\mathcal{T}^2} d_\mathcal{T}(a,b)\gamma(da,db) &\geq \int_{\phi(\Sigma_H)^2} d_\mathcal{T}(a,b)\gamma(da,db), \quad \text{since } \phi^{-1} \text{ is distance-preserving,}\\
&= \int_{\Sigma_H^2} d_{\mathrm{Ham}}(x,y)((\phi^{-1}\times\phi^{-1})_\sharp\gamma)(dx,dy).
\end{aligned}$$

$((\phi^{-1}\times\phi^{-1})_\sharp\gamma)$ is a measure on Σ_H^2, and is a coupling of μ and ν. Consequently, it leads to

$$\inf_{\gamma\in\Pi(\phi_\sharp\mu,\phi_\sharp\nu)} \int_{\mathcal{T}^2} d_\mathcal{T}(a,b)\gamma(da,db) \geq \inf_{\pi\in\Pi(\mu,\nu)} \int_{\Sigma_H^2} d_{\mathrm{Ham}}(x,y)\pi(dx,dy) = W_{d_{\mathrm{Ham}}}(\mu,\nu).$$

This means $W_{d_\mathcal{T}}(\phi_\sharp\mu,\phi_\sharp\nu) \geq W_{d_{\mathrm{Ham}}}(\mu,\nu)$ holds. □

Proposition 1 shows that the TWWL algorithm exhibits lower time complexity than the approach described in Togninalli et al. (2019). Given that $\mathcal{G}_1$ and $\mathcal{G}_2$ are labeled graphs with n_1 and n_2 vertices, respectively. The time complexity of the Sinkhorn algorithm for Equation 3 requires $\mathcal{O}(n_1 n_2)$. The time complexity of the TWWL algorithm is determined by the number of edges in the label tree and the computation of Γ. The number of edges is $\mathcal{O}(n_1+n_2)$ even in the worst case, and Γ can be computed in $\mathcal{O}(n_1+n_2)$ time beforehand. Consequently, the overall time complexity of our algorithm is $\mathcal{O}(n_1+n_2)$. As the number of vertices in the given graphs increases, our algorithm becomes significantly faster than the existing approach.

Equation 4 has been shown to be positive definite for every $\lambda > 0$ by Togninalli et al. (2019). The alternative expression based on the Tree Wasserstein distance offers an independent verification through Proposition 2 in Le et al. (2019b).

5 Experiments

In this section, we conduct two numerical experiments to validate the scalability and the performances of our approach. The experimental code has been provided as in supplementary material for review, and will be made publicly available upon acceptance.

Table 1: Summary of datasets.

	N	$\bar{n}$	avg$\{\lvert E_i \rvert\}$	$\sharp$classes	node labels
MUTAG	188	17.93	19.79	63/125	✓
PTC-MR	344	14.29	14.69	192/152	✓
ENZYMES	600	32.63	62.14	100/100/100/100/100/100	✓
PROTEINS	1113	39.06	72.82	663/450	✓
DD	1178	284.32	715.66	691/487	✓
NCI1	4110	29.87	32.30	2053/2057	✓
COLLAB	5000	74.49	2457.78	2600/775/1625	-
REDDIT-BINARY	2000	429.62	497.75	1000/1000	-
REDDIT-MULTI-5K	4999	508.51	594.87	1000/1000/1000/1000/999	-
REDDIT-MULTI-12K	11929	391.40	456.89	767/2592/1000/1094/902/1205 513/999/1243/1092/522	-
DBLP-v1	19456	10.48	19.65	9530/9926	✓
github-stargazers	12725	113.79	234.64	5917/6808	-

We evaluate the proposed methods on graph classification task using real-world datasets summarized in Table 1. For datasets without node labels, we assign dummy labels based on the degree of each vertex. All datasets were downloaded from Morris et al. (2020). The details of the hyperparameter settings are described in Appendix B.

We compare the proposed method with both traditional graph kernels, and a family of WL graph kernels as its baselines. We benchmark our algorithm against the Wasserstein Weisfeiler-Lehman (WWL) kernel with the Sinkhorn algorithm (Togninalli et al., 2019), Weisfeiler-Lehman (WL) subtree kernel (Shervashidze et al., 2011), Weisfeiler-Lehman Optimal Assignment (WL-OA) kernel (Kriege et al., 2016), and traditional graph kernels such as Shortest Path (SP) Kernel (Borgwardt & Kriegel, 2005), and Graphlet (GL) sampling kernel (Pržulj, 2007). Excluding our algorithmic approach and the WWL kernel, we utilize the graph kernel implementations provided by Siglidis et al. (2020). When comparing the execution times of our algorithm against WWL kernels, all code is implemented entirely in Julia.

5.1 Runtime Comparison

In the first experiment, we benchmark the computation time required for calculating the pairwise Wasserstein distances on real-world datasets. For each combination of datasets and methods, we conduct 10 independent trials and report the average runtime (in seconds) and the corresponding standard deviation. We denote linear programming solver by LP, implemented by Huangfu & Hall (2018), and note that our algorithm provides the same calculated values with the solver. Furthermore, since the computation time of the Sinkhorn algorithm varies depending on the entropic regularization term γ, we assess the time for the Sinkhorn algorithm with both 0.01 and 1.0. The parameter H for the WL algorithm is fixed at 5 for simplicity. For further details, additional runtime results for the proposed method with varying values of H are placed in Appendix B.3.

Table 2 shows that TWWL significantly reduces runtime compared to either the Sinkhorn algorithm or linear programming solver. Notably, TWWL completes computations within practical time limits even in challenging scenarios, such as a dataset with numerous graphs (e.g., DBLP-v1), a dataset with high average vertex counts per graph (e.g., REDDIT-MULTI-5K), and a dataset with both features (e.g., REDDIT-MULTI-12K). For some datasets, such as NCI1, DD, and larger datasets, it was frequently observed that the Sinkhorn algorithm failed to converge within the maximum number of iterations for any value of γ. A key advantage of our algorithmic approach is its reliability to produce stable solutions.

Table 2: Runtime performance of the Wasserstein distance computation fixed at $H = 5$. Entries labeled "TLE (Time Limit Exceeded)" indicate that a single trial exceeded 24 hours in runtime.

METHOD	MUTAG	PTC-MR	ENZYMES	PROTEINS	DD	NCI1
LP	33.89 ± 1.68	85.98 ± 5.73	882.56 ± 69.52	4597.12 ± 157.77	TLE	$34\,775.82 \pm 337.00$
WWL (0.01)	3.15 ± 0.04	4.54 ± 0.01	37.01 ± 0.02	147.26 ± 0.52	9390.61 ± 28.49	2116.45 ± 6.92
WWL (1.00)	0.27 ± 0.01	0.47 ± 0.00	5.33 ± 0.16	24.89 ± 0.23	1283.43 ± 9.48	229.97 ± 0.59
TWWL	0.03 ± 0.00	0.08 ± 0.00	0.67 ± 0.12	2.38 ± 0.19	26.93 ± 0.40	24.33 ± 0.56

	COLLAB	**REDDIT-B**	**REDDIT-M5**	**REDDIT-M12**	DBLP-v1	**github-stargazers**
LP	TLE	TLE	TLE	TLE	TLE	TLE
WWL (0.01)	7721.67 ± 37.48	TLE	TLE	TLE	1798.11 ± 5.92	TLE
WWL (1.00)	1653.68 ± 27.69	8457.94 ± 469.58	TLE	TLE	945.74 ± 8.84	$25\,820.89 \pm 65.90$
TWWL	44.18 ± 0.32	38.92 ± 0.32	333.53 ± 3.02	5535.76 ± 25.07	328.89 ± 0.56	952.02 ± 2.11

Table 3: Classification accuracies on datasets. Entries labeled "MLE (Memory Limit Exceeded)" indicate that the computation failed with an error that requested more than 256GB of memory.

METHOD	MUTAG	PTC-MR	ENZYMES	PROTEINS	DD	NCI1
SP	81.84 ± 8.62	61.00 ± 6.75	40.67 ± 6.54	$\mathbf{75.74 \pm 2.75}$	$\mathbf{79.80 \pm 2.37}$	73.26 ± 1.95
GL	77.02 ± 9.70	55.82 ± 4.15	29.83 ± 5.24	68.92 ± 3.45	72.49 ± 4.52	59.61 ± 2.35
WL	85.00 ± 10.51	62.27 ± 11.90	54.33 ± 6.95	75.65 ± 2.86	79.20 ± 3.66	85.18 ± 1.51
WL-OA	85.56 ± 6.92	62.24 ± 5.88	$\mathbf{60.67 \pm 5.62}$	75.21 ± 2.58	79.46 ± 2.29	86.16 ± 1.35
WWL	87.72 ± 6.91	65.12 ± 7.27	60.00 ± 5.50	74.58 ± 2.54	TLE	86.37 ± 1.40
TWWL	$\mathbf{88.27 \pm 6.16}$	$\mathbf{66.00 \pm 6.83}$	59.83 ± 4.74	74.94 ± 2.98	77.17 ± 3.51	$\mathbf{86.57 \pm 1.27}$

	COLLAB	REDDIT-B	REDDIT-M5	REDDIT-M12	DBLP-v1	github-stargazers
SP	$\mathbf{81.34 \pm 2.47}$	$\mathbf{88.70 \pm 1.93}$	52.33 ± 2.62	$\mathbf{44.10 \pm 1.14}$	MLE	$\mathbf{68.06 \pm 1.17}$
GL	TLE	TLE	TLE	TLE	TLE	TLE
WL	78.74 ± 2.51	75.05 ± 2.07	50.79 ± 1.46	39.65 ± 1.33	92.97 ± 0.52	65.19 ± 1.03
WL-OA	TLE	88.60 ± 2.33	TLE	TLE	TLE	TLE
WWL	TLE	TLE	TLE	TLE	MLE	TLE
TWWL	80.28 ± 1.67	85.75 ± 2.87	$\mathbf{54.47 \pm 2.48}$	42.90 ± 0.95	$\mathbf{93.37 \pm 0.47}$	65.55 ± 0.73

5.2 Performance Comparison

In this experiment, we evaluate the performances of TWWL on classification task. The experiments are evaluated using 10-fold cross-validation and support vector machines (Cortes & Vapnik, 1995) as the classifier. Since the TWWL algorithm is specifically designed to accurately and efficiently compute the Wasserstein distance, it is expected that our algorithm achieves comparable performance to the WWL kernel. Our aim is to verify that the precise Wasserstein distance does not adversely affect performance when compared to the Sinkhorn algorithm.

Table 3 shows that our algorithm achieves classification accuracy competitive with the WWL kernel. Notably, TWWL can scale to large datasets for which the Sinkhorn algorithm is computationally prohibited due to the runtime and/or memory requirements. This result suggests that our proposed approach can serve as a drop-in replacement for the Sinkhorn algorithm in the WWL kernel in the context of categorical node labels.

6 Conclusion

In this paper, we proposed an exact and efficient algorithm to compute the Wasserstein Weisfeiler-Lehman graph kernel for datasets with categorical node labels, especially for large-scale application domains. We revealed that labels generated by the Weisfeiler-Lehman algorithm inherently yield a tree structure, and that the proposed algorithm provides the same result with the original problem in the WWL graph kernel. The computational complexity of the proposed algorithm scales linearly with the number of vertices in a graph, which is considerably more efficient than the quadratic computational cost required by the Sinkhorn algorithmic approach. Numerical experiments demonstrate that our algorithm achieves performances comparable to the Sinkhorn algorithm, and its efficient computation enables the WWL kernel to be applied to large-scale datasets that were previously infeasible, thereby establishing it as an alternative for datasets with categorical node labels.

A promising direction for future work is to extend the proposed approach to datasets with continuous node attributes, e.g., using clustering approach as in the bag-of-visual-word approach in computer vision. Our method is based on the property that the space formed by label set and the Hamming distance can be represented as a tree structure while preserving its topological property. However, this property does not generally hold for continuous data, necessitating approximations. One possible approach is to develop a graph kernel based on the tree-sliced Wasserstein distance (Le et al., 2019b). While existing work (Perez et al., 2024) accelerates the WWL kernel by leveraging the sliced Wasserstein distance, a systematic comparison of these strategies, focusing on both computational efficiency and performance, would be valuable. In addition, there have been numerous studies on approximating Euclidean structures with trees, an investigation into optimal tree construction strategies may yield informative insights and enhance the effectiveness of our approach for continuous data.

References

Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1961–1971, December 2017.

Albert-László Barabási and Zoltán N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, February 2004. doi: 10.1038/nrg1272.

Albert-László Barabási and Márton Pósfai. *Network Science*. Cambridge University Press, 2016.

Nicolas Bonneel, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. Sliced and radon wasserstein barycenters of measures. *Journal of mathematical imaging and vision*, 51(1):22–45, January 2015. doi: 10.1007/s10851-014-0506-3.

Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends® in Machine Learning*, 13(5-6): 531–712, December 2020. doi: 10.1561/2200000076.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining*, pp. 74–81, 2005. doi: 10.1109/ICDM.2005.132.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, June 2005. doi: 10.1093/bioinformatics/bti1007.

Samantha Chen, Sunhyuk Lim, Facundo Memoli, Zhengchao Wan, and Yusu Wang. Weisfeiler-lehman meets gromov-wasserstein. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 3371–3416, July 2022.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3 edition, July 2009.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, September 1995. doi: 10.1007/bf00994018.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, volume 2, pp. 2292–2300, December 2013.

Asim Kumar Debnath, Ross L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, February 1991. doi: 10.1021/jm00106a046.

Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, July 2003. doi: 10.1016/S0022-2836(03)00628-4.

Kira Michaela Düsterwald, Samo Hromadka, and Makoto Yamada. Fast unsupervised ground metric learning with tree-wasserstein distance. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Steven N. Evans and Frederick A. Matsen. The phylogenetic Kantorovich–Rubinstein metric for environmental sequence samples. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 74 (3):569–592, June 2012. doi: 10.1111/j.1467-9868.2011.01018.x.

Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pp. 216–224, December 2013.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T. H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. POT: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

Tony Hansen and Donald E. Eastlake 3rd. US secure hash algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, May 2011.

Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, January 2001. doi: 10.1093/bioinformatics/17.1.107.

Qi Huangfu and Julian Hall. Parallelizing the dual revised simplex method. *Mathematical programming computation*, 10(1):119–142, March 2018. doi: 10.1007/s12532-017-0130-5.

Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *International workshop on statistical and computational theories of vision*, 2003.

Lemin Kong, Jiajin Li, Jianheng Tang, and Anthony Man-Cho So. Outlier-robust gromov-wasserstein for graph data. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 24781–24803, December 2023.

Nils M. Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 1623–1631, December 2016.

Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied network science*, 5(1):1–42, December 2020. doi: 10.1007/s41109-019-0195-3.

Tam Le, Viet Huynh, Nhat Ho, Dinh Phung, and Makoto Yamada. Tree-wasserstein barycenter for large-scale multilevel clustering and scalable bayes, October 2019a.

Tam Le, Makoto Yamada, Kenji Fukumizu, and Marco Cuturi. Tree-sliced variants of wasserstein distances. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 12304–12315, December 2019b.

Tam Le, Truyen Nguyen, Dinh Phung, and Viet Anh Nguyen. Sobolev transport: A scalable metric for probability measures with graph metrics. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151, pp. 9844–9868, March 2022.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on knowledge discovery from data*, 1(1):2, March 2007. doi: 10.1145/1217299.1217301.

Mengyu Li, Jun Yu, Tao Li, and Cheng Meng. Importance sparsification for sinkhorn algorithm. *Journal of Machine Learning Research*, 24(247):1–44, 2023.

Ya-Wei Eileen Lin, Ronald R. Coifman, Gal Mishne, and Ronen Talmon. Tree-wasserstein distance for high dimensional data with a latent feature hierarchy. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Catherine Lozupone and Rob Knight. UniFrac: a new phylogenetic method for comparing microbial communities. *Applied and Environmental Microbiology*, 71(12):8228–8235, December 2005. doi: 10.1128/AEM.71.12.8228-8235.2005.

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 Workshop on Graph Representation Learning and Beyond*, July 2020.

Vu Nguyen, Tam Le, Makoto Yamada, and Michael A. Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 8084–8095, July 2021.

Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *The journal of artificial intelligence research*, 72:943–1027, November 2021. doi: 10.1613/jair.1.13225.

James B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical programming*, 78(2):109–129, August 1997. doi: 10.1007/bf02614365.

Shirui Pan, Xingquan Zhu, Chengqi Zhang, and Philip S. Yu. Graph stream classification using labeled and unlabeled graphs. In *Proceedings of the 29th IEEE International Conference on Data Engineering*, pp. 398–409, April 2013. doi: 10.1109/icde.2013.6544842.

Raphaël Carpintero Perez, Sébastien Da Veiga, Josselin Garnier, and Brian Staber. Gaussian process regression with sliced wasserstein weisfeiler-lehman graph kernels. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, pp. 1297–1305, May 2024.

Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. GOT: An optimal transport framework for graph comparison. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 13899–13910, December 2019.

Gabriel Peyré and Marco Cuturi. Computational optimal transport, 2020.

Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2): e177–e183, January 2007. doi: 10.1093/bioinformatics/btl301.

Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. Karate club: An API oriented open-source python framework for unsupervised learning on graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pp. 3125–3132, October 2020.

Meyer Scetbon, Marco Cuturi, and Gabriel Peyré. Low-rank sinkhorn factorization. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pp. 9344–9354, July 2021.

Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, volume 5, pp. 488–495, April 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011. doi: 10.5555/1953048.2078187.

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*, 21(54): 1–5, 2020.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, June 1964.

Yuki Takezawa, Ryoma Sato, Zornitsa Kozareva, Sujith Ravi, and Makoto Yamada. Fixed support tree-sliced wasserstein barycenter. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, pp. 1120–1137, March 2022.

Xun Tang, Michael Shavlovsky, Holakou Rahmanian, Elisa Tardini, Kiran Koshy Thekumparampil, Tesi Xiao, and Lexing Ying. Accelerating sinkhorn algorithm with sparse newton iterations. In *Proceedings of the 12th International Conference on Learning Representations*, 2024.

Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 6439–6449, December 2019.

Cedric Villani. *Optimal Transport: Old and New*. Springer, December 2009. doi: 10.1007/978-3-540-71050-9.

Cédric Vincent-Cuaz, Rémi Flamary, Marco Corneli, Titouan Vayer, and Nicolas Courty. Semi-relaxed gromov-wasserstein divergence and applications on graphs. In *Proceedings of the 10th International Conference on Learning Representations*, 2022.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(40):1201–1242, 2010.

Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and information systems*, 14(3):347–375, March 2008. doi: 10.1007/s10115-007-0103-5.

Boris Weisfeiler and Andrei Leman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

Hongteng Xu, Dixin Luo, and Lawrence Carin. Scalable gromov-wasserstein learning for graph partitioning and matching. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 3052–3062, December 2019.

Makoto Yamada, Yuki Takezawa, Ryoma Sato, Han Bao, Zornitsa Kozareva, and Sujith Ravi. Approximating 1-wasserstein distance with trees. *Transactions on Machine Learning Research*, September 2022.

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*, pp. 1365–1374. Association for Computing Machinery, 2015. doi: 10.1145/2783258.2783417.

A Hash Functions

In many areas of computer science, a hash function is used to map arbitrary length data (e.g., text data, image, or binary data) to a fixed length. This output, commonly called a hash value, serves as a compact representation of the original data. In general, hash functions are used for two main purposes. One is for data uniqueness verification and tamper detection in cryptographic security, and the other is as table keys for processing large volumes of data or for detecting differences between two data in non-cryptographic domains. SHA-2 (Hansen & 3rd, 2011) is one of the well-known standards designed by the National Security Agency and standardized by the National Institute of Standards and Technology as a hash function for cryptographic security.

The properties required of a hash function depend on its intended application, but to be considered a good hash function, it must satisfy at least the following two properties:

- Deterministic: for the same input, the hash function always produces the same hash value.
- Collision resistance: it should be computationally infeasible to find two different inputs that yield the same hash value.

The hash function in Equation 1 is assumed to satisfy these two properties. A well-known SHA-256 implementation can be used to compute Equation 1. Alternatively, Julia's built-in `Base.hash` function is used in our implementation.

B Details on Experiments

This appendix details the datasets and hyperparameter settings used in the numerical experiments of Section 5. All datasets are publicly available through TUDatasets repository (Morris et al., 2020). Every graph dataset contains the adjacency information and a graph-level label. Some datasets additionally provide optional information such as node labels/attributes or edge labels/attributes. In our experiments, we exploited only the node-label information when it was available. For datasets without node labels (COLLAB, REDDIT-BINARY, REDDIT-MULTI-5K, REDDIT-MULTI-12K, github-stargazers), we assigned pseudo node labels based on its degree.

B.1 Dataset Descriptions

Datasets used in our experiments can be categorized into two types: chemoinformatics and bioinformatics datasets, and social network datasets. The main characteristics of each dataset are summarized as follows.

MUTAG: This is a chemoinformatics dataset consisting of 188 aromatic and heteroaromatic compounds (Debnath et al., 1991). Each graph models a compound, with nodes corresponding to atoms and edges to the chemical bonds between them. The task is to predict the mutagenic effect of the compound, categorized into two classes.

PTC-MR: This dataset originates from the Predictive Toxicology Challenge (Helma et al., 2001, PTC), a competition to promote the development of machine learning models for predicting chemical toxicity. The PTC-MR dataset contains chemical compounds tested for carcinogenicity in male rats (MR). Each compound is modeled as a graph, where nodes denote atoms and edges denote the chemical bonds connecting them. The task is to predict compound carcinogenicity, which is a binary classification problem.

ENZYMES: This is a bioinformatics dataset constructed from the BRENDA (Braunschweig Enzymes Database), comprehensive enzymes and metabolic information repository (Borgwardt et al., 2005). Each graph represents an enzyme, where nodes are its secondary structure elements (SSEs), and edges are created between nodes if they are neighbors along the amino acid sequence or are spatially close. The task is to classify enzymes into six classes based on their catalytic activity.

PROTEINS: This dataset is used as a benchmark for graph classification task of predicting whether a protein is an enzyme or not (Borgwardt et al., 2005). A protein is represented by a graph as follows. Nodes in the

graph correspond to SSEs, and they are connected by an edge if they are neighbors along the amino acid sequence or are spatially close.

DD: This dataset consists of 1178 protein structures where the task is to classify them into enzymes and non-enzymes (Dobson & Doig, 2003). Each graph represents a protein, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.

NCI1: This dataset consists of chemical compounds screened for their effectiveness against non-small cell lung cancer, with the task being to classify each compound as active or not (Wale et al., 2008). Nodes in each graph represent atoms, and edges represent chemical bonds between them.

COLLAB: This is a scientific collaboration dataset of researchers (Leskovec et al., 2007). Each graph represents an ego-network of a researcher. In the graph, nodes represent researchers and edges indicate co-authorship. This dataset is widely used as a benchmark for graph classification tasks of predicting the researcher's academic field among three classes: High Energy Physics, Condensed Matter Physics, and Astro Physics.

REDDIT-BINARY/MULTI-5K/MULTI-12K: These are social network dataset created from Reddit, an online discussion platform where users participate in discussions within topic-specific threads called subreddits (Yanardag & Vishwanathan, 2015). Each graph represents the user interaction structure of a single discussion thread. In each graph, nodes represent users, and an edge connects two nodes if one user replied to another's comment. The collection includes three variants with different classification challenges:

- *REDDIT-BINARY* is a binary classification task to distinguish threads from QA-style subreddits versus discussion-style subreddits.
- *REDDIT-MULTI-5K* is a 5-class classification task among five subreddits: worldnews, videos, AdviceAnimals, aww and mildlyinteresting.
- *REDDIT-MULTI-12K* expands the task to a larger set of subreddits: AskReddit, AdviceAnimals, atheism, aww, IAmA, mildlyinteresting, Showerthoughts, videos, todayilearned, worldnews, and TrollXChromosomes.

DBLP-v1: This is a scientific collaboration dataset derived from DBLP, a comprehensive bibliography database of computer science publications (Pan et al., 2013). Each graph represents a paper in DBLP, where nodes denote either a paper or a keyword, and edges represent one of three types of relationships: a citation relationship between two papers, a link between a paper and its corresponding keyword, or a link between keywords from the same paper. The task is to classify each paper into one of two classes: database and data mining field or computer vision and pattern recognition field.

github-stargazers: This is a social network dataset derived from GitHub, the software development platform (Rozemberczki et al., 2020). It consists of social networks of developers who starred popular machine learning and web development repositories. Each graph represents the community of developers who starred a repository, where nodes represent users, and edges denote follower relationships. The goal is to predict whether a social network belongs to a machine learning or web development repository.

B.2 Hyperparameter Settings

The following hyperparameter settings were used in the experiments of Section 5. For a regularization parameter C in the support vector classifier, we search over the range $\{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. The number of iterations H for the WL algorithm is chosen from $\{1, \ldots, 7\}$. The parameter λ of the WWL and TWWL kernel is selected from $\{10^{-4}, 10^{-3}, \ldots, 10^{1}\}$. For the Sinkhorn algorithm, we select the entropic regularization parameter from $\{0.01, 0.05, 0.1, 0.2, 0.5, 1, 10\}$ and fix the maximum number of iterations to 1000, which is the default setting in the commonly used implementation (Flamary et al., 2021). In the second experiment, hyperparameter optimization is performed on the training data via grid search over the predetermined range of values.

Table 4: TWWL runtime performance of the Wasserstein distance computation as the parameter H is increased.

DATASET	$H = 1$	$H = 2$	$H = 3$	$H = 4$	$H = 5$	$H = 6$	$H = 7$
MUTAG	0.007 ± 0.004	0.010 ± 0.001	0.015 ± 0.001	0.022 ± 0.001	0.032 ± 0.001	0.042 ± 0.001	0.052 ± 0.001
PTC-MR	0.036 ± 0.066	0.040 ± 0.057	0.038 ± 0.001	0.056 ± 0.001	0.076 ± 0.001	0.096 ± 0.001	0.181 ± 0.213
ENZYMES	0.08 ± 0.07	0.20 ± 0.06	0.36 ± 0.09	0.51 ± 0.09	0.70 ± 0.16	0.84 ± 0.11	1.04 ± 0.15
PROTEINS	0.24 ± 0.09	0.64 ± 0.10	1.19 ± 0.16	1.83 ± 0.14	2.37 ± 0.16	2.98 ± 0.23	3.47 ± 0.18
DD	6.20 ± 0.31	11.61 ± 0.47	16.67 ± 0.43	21.74 ± 0.17	26.72 ± 0.18	31.83 ± 0.28	36.96 ± 0.36
NCI1	5.47 ± 0.45	10.19 ± 0.53	15.56 ± 0.54	20.70 ± 0.38	25.22 ± 0.49	30.20 ± 0.30	34.81 ± 0.31
COLLAB	18.08 ± 0.53	25.97 ± 0.85	34.77 ± 0.63	43.69 ± 0.56	52.24 ± 0.66	60.90 ± 0.71	69.66 ± 0.58
REDDIT-B	5.50 ± 0.14	13.55 ± 0.14	22.36 ± 0.32	30.62 ± 0.57	38.74 ± 0.45	47.08 ± 0.54	55.27 ± 0.49
REDDIT-M5	66.58 ± 1.29	138.67 ± 3.30	207.05 ± 2.08	388.21 ± 53.25	554.52 ± 2.83	686.86 ± 2.78	817.82 ± 1.12
REDDIT-M12	326.64 ± 1.28	835.47 ± 1.66	1476.17 ± 3.76	2155.87 ± 6.07	5631.26 ± 16.50	7246.46 ± 32.74	8688.33 ± 30.20
DBLP-v1	139.52 ± 0.85	187.52 ± 0.69	236.03 ± 0.93	285.69 ± 1.27	333.33 ± 0.75	382.69 ± 1.34	431.27 ± 0.85
github-stargazers	208.34 ± 0.19	398.80 ± 0.54	598.15 ± 0.52	794.32 ± 1.12	987.71 ± 1.21	1184.01 ± 1.37	1377.23 ± 3.00

B.3 Further results on runtime comparison

To evaluate how the computational cost of our algorithm scales with the parameter H, we supplement the runtime study in Section 5.1 with additional experiments. Whereas the experiment in Section 5.1 fixed $H = 5$, Table 4 reports the runtime results of the TWWL algorithm for a range of H values. In the TWWL algorithm, the extra cost caused by increasing H is proportional to the number of newly generated labels, whose maximum is bounded by the total number of vertices n in a dataset. Consequently, the overall complexity grows approximately linearly in H, a trend clearly confirmed by Table 4.