Approximating 1-Wasserstein Distance with Trees

Anonymous authors
Paper under double-blind review

Abstract

Wasserstein distance, which measures the discrepancy between distributions, shows efficacy in various types of natural language processing (NLP) and computer vision (CV) applications. One of the challenges in estimating Wasserstein distance is that it is computationally expensive and does not scale well for many distribution comparison tasks. In this paper, we aim to approximate the 1-Wasserstein distance by the tree-Wasserstein distance (TWD), where TWD is a 1-Wasserstein distance with tree-based embedding and can be computed in linear time with respect to the number of nodes on a tree. More specifically, we propose a simple yet efficient L1-regularized approach to learning the weights of the edges in a tree. To this end, we first show that the 1-Wasserstein approximation problem can be formulated as a distance approximation problem using the shortest path distance on a tree. We then show that the shortest path distance can be represented by a linear model and can be formulated as a Lasso-based regression problem. Owing to the convex formulation, we can obtain a globally optimal solution efficiently. Moreover, we propose a tree-sliced variant of these methods. Through experiments, we demonstrated that the weighted TWD can accurately approximate the original 1-Wasserstein distance.

1 Introduction

Wasserstein distance, which is an optimal transport (OT)-based distance, measures the discrepancy between two distributions, and is widely used in natural language processing (NLP) and computer vision (CV) applications. For example, measuring the similarity between documents is a fundamental natural language processing task, and can be used for semantic textual similarity (STS) tasks (Yokoi et al., 2020). For CV tasks, because it is possible to obtain matching between samples using OT, it is used to determine the correspondence between two sets of local features (Sarlin et al., 2020).

One of the widely used application of the Wasserstein distance is Word Mover's Distance (WMD) (Kusner et al., 2015), which measures the similarity between documents by solving the optimal transport problem. Recently, Yokoi et al. (2020) proposed a more effective similarity measure based on the Word Rotator's Distance (WRD). The WMD and WRD are distance-based optimal transport, which can be estimated by solving linear programming with $O(n^3)$ order of computation, where n is the number of words in a document. Thus, using WMD and WRD for a large number of documents is challenging.

One of the well-known speedup techniques for OT problems is the use of the Sinkhorn algorithm (Cuturi, 2013), which solves the entropic regularized optimal transport problem. Using the Sinkhorn algorithm, we can estimate WMD and WRD with $O(n^2)$ computational cost. However, this is slow for most NLP and CV tasks. Instead of using linear programming and the Sinkhorn algorithm, we can speed up the computation by projecting word vectors into 1D space and solving the OT problem. This approach is known as the sliced Wasserstein distance (Rabin et al., 2011; Kolouri et al., 2016), and its computational complexity is $O(n \log n)$. Note that the result of the sliced Wasserstein distance can be different if we use another projection matrix, and thus the performance of the sliced Wasserstein distance is highly dependent on the projection matrix. To alleviate this problem, we average the sliced Wasserstein distances with different projection matrices.

Tree-Wasserstein distance (TWD) (Evans & Matsen, 2012; Le et al., 2019) computes the optimal transport on a tree, and it can be computed in $O(N)$, where N is the number of nodes in a tree. Specifically, we first

embedded the word vectors into a tree and computed the optimal transport based on a tree metric. Because we can obtain an analytical solution to the OT problem on a tree, it can be solved efficiently. Moreover, TWD includes sliced Wasserstein as a special case, and it empirically outperforms sliced Wasserstein (Le et al., 2019). The performance of TWD is highly dependent on tree construction. One of the widely used tree construction methods is based on the QuadTree (Indyk & Thaper, 2003). Another approach involves using a clustering-based tree algorithm (Le et al., 2019). Recently, Takezawa et al. (2021) proposed a continuous tree-construction approach. Although tree-based methods improve the computational complexity by a large margin, existing tree-based methods do not approximate the vanilla Wasserstein distance satisfactorily.

In this paper, we aimed to accurately approximate the 1-Wasserstein distance using TWD. Specifically, we propose a simple yet efficient sparse learning-based approach to train the edge weights of a tree. To this end, we first demonstrate that the 1-Wasserstein approximation problem can be formulated as a distance approximation problem using the shortest path distance on a tree. Then, we show that the shortest path distance can be represented by a linear model when we fix the tree structure. The weight estimation problem is then formulated as a non-negative Lasso-based optimization problem. Owing to the convex formulation, we can obtain a globally optimal solution efficiently. Moreover, we propose a tree-sliced variant of the proposed method. One of the key advantages of our formulation is that the optimization can be efficiently solved using an off-the-shelf solver. Through experiments, we evaluate the proposed algorithm for Twitter, BBCSport, and Amazon datasets and show that the 1-Wasserstein distance computed by linear programming can be accurately approximated by TWD, whereas QuadTree (Indyk & Thaper, 2003) and ClusterTree (Le et al., 2019) have high approximation errors.

Contribution: The contributions of our paper are summarized below:

- A Lasso-based weight estimation is proposed for TWD.
- An estimation of the weights for tree-sliced Wasserstein distance is proposed.
- We empirically demonstrate that the proposed method is capable of approximating the 1-Wasserstein distance accurately.

2 Related Work

Optimal Transport and Wasserstein distance: Measuring the distance between sets is an important research topic in machine learning, computer vision, and natural language processing. Among the early works, Earth Mover's Distance (EMD) is an optimal transport-based distance, equivalent to Wasserstein distance if we employ distance to compute the cost function and can be computed using linear programming. EMD has been particularly studied in the computer vision community because it can obtain matching between samples. For example, Sarlin et al. (2020) proposed the SuperGlue method, which determines the correspondence between two sets of local features using optimal transport. Liu et al. (2020) uses the optimal transport method for semantic correspondence. In NLP tasks, Kusner et al. (2015) proposed Word Mover's Distance (WMD), the first work to use Wasserstein distance for textual similarity tasks, and it is widely used in NLP tasks, including text generation evaluation (Zhao et al., 2019). Recently, Sato et al. (2022) re-evaluated WMD in various experimental setups and summarized the pros and cons of WMD. Another promising approach is based on the Word Rotater's Distance (WRD) (Yokoi et al., 2020), which normalizes the word vectors and solves WMD with adjusted probability mass, and it has been reported that the performance significantly improves over WMD. However, these approaches employ EMD, and the computational cost is very high; therefore, they cannot be used for large-scale distribution comparison tasks.

To speed up EMD and Wasserstein distance computation, Cuturi (2013) proposed the Sinkhorn algorithm, which solves the entropic regularized optimization problem and computes Wasserstein distance in quadratic order (i.e., $O(n^2)$). Kusner et al. (2015) proposed a relaxed Wasserstein Mover's distance (RWMD), which only considers the some to one constraint. Because RWMD computation is simple, it can efficiently compute the distance. Atasu & Mittelholzer (2019) proposed the approximate constrained transfer method, which iteratively adds constraints to RWMD and improves the performance of RWMD while maintaining the

computational speed. To solve large-scale OT problems, Genevay et al. (2016) formulated the optimal transport problem as a concave maximization problem using c-transform and then solved the problem with the stochastic gradient descent (SGD) and the stochastic average gradient (SAG). Mensch & Peyré (2020) proposed the online Sinkhorn algorithm, which solves OT problems using a stream of samples.

Another approach is based on the sliced Wasserstein distance (SWD) (Rabin et al., 2011; Kolouri et al., 2016), which solves the optimal transport problem in a projected one-dimensional subspace. Because it is known that one-dimensional optimal transport can be solved using sorting, SWD can be computed in $O(n \log n)$. Several extensions of SWD have been proposed. The generalized sliced-Wasserstein distance extends the 1D sliced Wasserstein distance for multidimensional cases (Kolouri et al., 2019). The max-sliced Wasserstein distance determines the 1D subspace that maximizes the discrepancy between two distributions and then computes the optimal transport in the subspace (Mueller & Jaakkola, 2015; Deshpande et al., 2019). The subspace robust Wasserstein distance (SRWD) is a general extension of the max-sliced Wasserstein distance for multi-dimensional subspaces (Paty & Cuturi, 2019; Lin et al., 2020).

Tree-Wasserstein distance (TWD) (Evans & Matsen, 2012; Le et al., 2019) uses tree-based embedding, whereas SWD uses one-dimensional embedding. Because a chain of a tree can represent one-dimensional embedding, TWD includes SWD as a special case. Le et al. (2019) reported that the TWD can empirically outperform SWD. TWD is also studied in the theoretical computer science community and can be computed using the QuadTree algorithm (Indyk & Thaper, 2003). The extension of TWD has also been studied, including unbalanced TWD (Sato et al., 2020; Le & Nguyen, 2021), supervised Wasserstein training (Takezawa et al., 2021), and tree barycenter (Takezawa et al., 2022). These approaches mainly focus on approximating 1-Wasserstein with tree construction, and the weight of the edges are set by a constant number. Recently, Backurs et al. (2020) proposed a flowtree tree that combines the QuadTree method and the cost matrix computed from vectors. They then showed that the flowtree outperformed QuadTree-based approaches. Moreover, they theoretically guarantee that the QuadTree and flowtree methods can approximate the nearest neighbor using the 1-Wasserstein distance. Dey & Zhang (2022) proposed an L1-embedding approach for approximating 1-Wasserstein distance for the persistence diagram. However, there are no learning-based approaches for approximating 1-Wasserstein distance for general setup. Thus, we focus on estimating the weight parameter of TWD from the data to approximate 1-Wasserstein distance.

Applications of Optimal transport: The optimal transport has been applied to various types of applications including barycenter estimation (Cuturi & Doucet, 2014; Benamou et al., 2015), generative modeling (Arjovsky et al., 2017; Gulrajani et al., 2017; Kolouri et al., 2018), domain adaptation (Courty et al., 2017; Shen et al., 2018), inverse optimal transport (Li et al., 2019), differentiable sorting (Cuturi et al., 2019; Blondel et al., 2020; Xie et al., 2020), and modeling of population dynamics (Bunne et al., 2022). Moreover, by using the Sinkhorn algorithm, we can solve a Wasserstein distance minimization problem in an end-to-end manner (Genevay et al., 2018). In this research direction, solving the Wasserstein distance minimization problem in mini-batch setup is an important research topic (Nguyen et al., 2022).

Many OT-based applications are formulated as a minimization problem of Wasserstein distance in dual (Arjovsky et al., 2017), a minimization problem using the Sinkhorn algorithm (Genevay et al., 2018), or a minimization problem with 1D slicing (Kolouri et al., 2018). Tree Wasserstein distance, which includes a sliced Wasserstein distance as a special case, has been applied to document classification (Le et al., 2019), topological data analysis (TDA) (Le et al., 2019), a neural architecture search (NAS) (Nguyen et al., 2021), and a barycenter estimation (Takezawa et al., 2022). Moreover, TWD is known as the UniFrac distance (Lozupone & Knight, 2005), which is a phylogenetic method for comparing microbial communities. Thus, our proposed weighting technique for TWD can be used for microbial community comparisons.

3 Preliminary

In this section, we introduce the optimal transport, Wasserstein distances, and tree-Wasserstein distances.

3.1 Optimal transport and Wasserstein Distance

Optimal transport: We computed the distance between two datasets $\{(\boldsymbol{x}_i, a_i)\}_{i=1}^n$ and $\{(\boldsymbol{x}'_i, b_i)\}_{j=1}^{n'}$, where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, $\boldsymbol{x}' \in \mathcal{X}' \subset \mathbb{R}^d$, and $\sum_{i=1}^n a_i = 1$ and $\sum_{j=1}^{n'} b_j = 1$ are the probability masses. For example, in the text similarity setup, $\boldsymbol{x}$ and $\boldsymbol{x}'$ are precomputed word-embedding vectors, and a_i is the frequency of the word $\boldsymbol{x}_i$ in a document. Let $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N'}$ denote the entire set of vectors, where N' denotes the number of vectors. The goal of this paper is to measure the similarity between two datasets $\{(\boldsymbol{x}_i, a_i)\}_{i=1}^n$ and $\{(\boldsymbol{x}'_i, b_i)\}_{j=1}^{n'}$.

In the Kantorovich relaxation of OT, admissible couplings are defined by the set of transport plans between two discrete measures, $\mu = \sum_{i=1}^n a_i \delta_{\boldsymbol{x}_i}$ and $\nu = \sum_{j=1}^{n'} b_j \delta_{\boldsymbol{x}'_j}$.

$$U(\mu, \nu) = \{\boldsymbol{\Pi} \in \mathbb{R}_+^{n \times n'} : \boldsymbol{\Pi} \mathbf{1}_{n'} = \boldsymbol{a}, \boldsymbol{\Pi}^\top \mathbf{1}_n = \boldsymbol{b}\}, \tag{1}$$

where $\delta_{\boldsymbol{x}_i}$ is the Dirac at position $\boldsymbol{x}_i$, $\mathbf{1}_n$ is the n-dimensional vector whose elements are ones, and $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^\top \in \mathbb{R}_+^n$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_{n'})^\top \in \mathbb{R}_+^{n'}$ are the probability masses of distributions μ and ν, respectively.

Then, the OT problem between two discrete measures μ and ν is given as

$$\min_{\boldsymbol{\Pi} \in U(\mu,\nu)} \quad \sum_{i=1}^n \sum_{j=1}^{n'} \pi_{ij} c(\boldsymbol{x}_i, \boldsymbol{x}'_j), \tag{2}$$

where π_{ij} is the i, j-th element of $\boldsymbol{\Pi}$, and $c(\boldsymbol{x}, \boldsymbol{x}')$ is a cost function (e.g., $c(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2$). Eq. (2) is equivalent to the Earth Mover's Distance and it can be solved by using linear programming with $O(n^3)$ computation ($n = n'$). To speed up the optimal transport computation, Cuturi (2013) proposed an entropic regularized optimal transport problem:

$$\min_{\boldsymbol{\Pi} \in U(\mu,\nu)} \quad \sum_{i=1}^n \sum_{j=1}^{n'} \pi_{ij} c(\boldsymbol{x}_i, \boldsymbol{x}'_j) - \epsilon H(\boldsymbol{\Pi}), \tag{3}$$

where $\epsilon \geq 0$ is the regularization parameter and $H(\boldsymbol{\Pi}) = -\sum_{i=1}^n \sum_{j=1}^{n'} \pi_{ij}(\log(\pi_{ij}) - 1)$ is the entropic regularization. The entropic regularized OT is equivalent to EMD when $\epsilon = 0$. The entropic regularization problem can be efficiently solved using the Sinkhorn algorithm with $O(nn')$ cost. Moreover, the Sinkhorn algorithm can be efficiently implemented on GPU, leading to a super-efficient matrix computation. Hence, the Sinkhorn algorithm is one of the most widely used OT solvers in the ML field.

Wasserstein distance: If we use distance function $d(\boldsymbol{x}, \boldsymbol{x}')$ for the cost $c(\boldsymbol{x}, \boldsymbol{x}')$ and $p \geq 1$, the p-Wasserstein distance of two discrete measures between two probability measures μ and ν is defined as

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \mathcal{U}(\mu,\nu)} \int_{\mathcal{X} \times \mathcal{X}'} d(\boldsymbol{x}, \boldsymbol{x}') \mathrm{d}\pi(\boldsymbol{x}, \boldsymbol{x}')^p \right)^{1/p}, \tag{4}$$

where $\mathcal{U}$ denotes a set of joint probability distributions (Peyré et al., 2019) (Remark 2.13):

$$\mathcal{U}(\mu, \nu) = \{\pi \in \mathcal{M}_+^1(\mathcal{X} \times \mathcal{X}') : P_{\mathcal{X}\sharp}\pi = \mu, P_{\mathcal{X}'\sharp}\pi = \nu\}, \tag{5}$$

and $\mathcal{M}_+^1(\mathcal{X})$ is the set of probability measures, $P_{\mathcal{X}\sharp}$ and $P_{\mathcal{X}'\sharp}$ are push-forward operators (see Remark 2.2 of (Peyré et al., 2019)). The p-Wasserstein distance of the two discrete measures $\mu = \sum_{i=1}^n a_i \delta_{\boldsymbol{x}_i}$ and $\nu = \sum_{j=1}^{n'} b_j \delta_{\boldsymbol{x}'_j}$ is defined as

$$W_p(\mu, \nu) = \left(\min_{\boldsymbol{\Pi} \in U(\mu,\nu)} \sum_{i=1}^n \sum_{j=1}^{n'} \pi_{ij} d(\boldsymbol{x}_i, \boldsymbol{x}'_j)^p \right)^{1/p}. \tag{6}$$

From the definition of the p-Wasserstein distance, the 1-Wasserstein distance of two discrete measures is given as:

$$W_1(\mu, \nu) = \min_{\boldsymbol{\Pi} \in U(\mu,\nu)} \sum_{i=1}^{n} \sum_{j=1}^{n'} \pi_{ij} d(\boldsymbol{x}_i, \boldsymbol{x}'_j), \tag{7}$$

which is equivalent to Eq. (2) when $c(\boldsymbol{x}, \boldsymbol{x}') = d(\boldsymbol{x}, \boldsymbol{x}')$. For the distance function, the Euclidean distance $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2$ and Manhattan distance $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_1$ are common choices. Note that Cuturi (2013) proposed the entropic regularized Wasserstein distance, although the entropic regularized formulation does not admit the triangle inequality, and thus, it is not a distance. Sinkhorn divergence was proposed to satisfy the metric axioms of the entropic regularized formulation (Genevay et al., 2018).

3.2 Tree-Wasserstein distance (TWD)

Although the Sinkhorn algorithm has quadratic time complexity, it is still prohibitive when dealing with large vocabulary sets. TWD has been gathering attention owing to its light computation, and is thus used to approximate the 1-Wasserstein distance (Indyk & Thaper, 2003; Evans & Matsen, 2012; Le et al., 2019).

We define $\mathcal{T} = (V, E)$, where V and E are the sets of nodes and edges, respectively. In particular, for a given tree $\mathcal{T}$, EMD with a tree metric $d_{\mathcal{T}}(x, x')$ is called TWD. TWD admits the following closed-form expression:

$$W_{\mathcal{T}}(\mu, \nu) = \sum_{e \in E} w_e |\mu(\Gamma(v_e)) - \nu(\Gamma(v_e))|, \tag{8}$$

where e is an edge index, $w_e \in \mathbb{R}_+$ is the edge weight of edge e, v_e is the e-th node index, and $\mu(\Gamma(v_e))$ is the total mass of the subtree with root v_e. The restriction to a tree metric is advantageous in terms of computational complexity, TWD can be computed as $O(N)$, where N is the number of nodes of a tree. It has been reported that TWD compares favorably with the vanilla Wasserstein distance and is computationally efficient. Note that the sliced Wasserstein distance can be regarded as a special case of TWD (Takezawa et al., 2022).

TWD can be represented in matrix form as (Takezawa et al., 2021)

$$W_{\mathcal{T}}(\mu, \nu) = \|\mathrm{diag}(\boldsymbol{w}) \boldsymbol{B} (\boldsymbol{a} - \boldsymbol{b})\|_1, \tag{9}$$

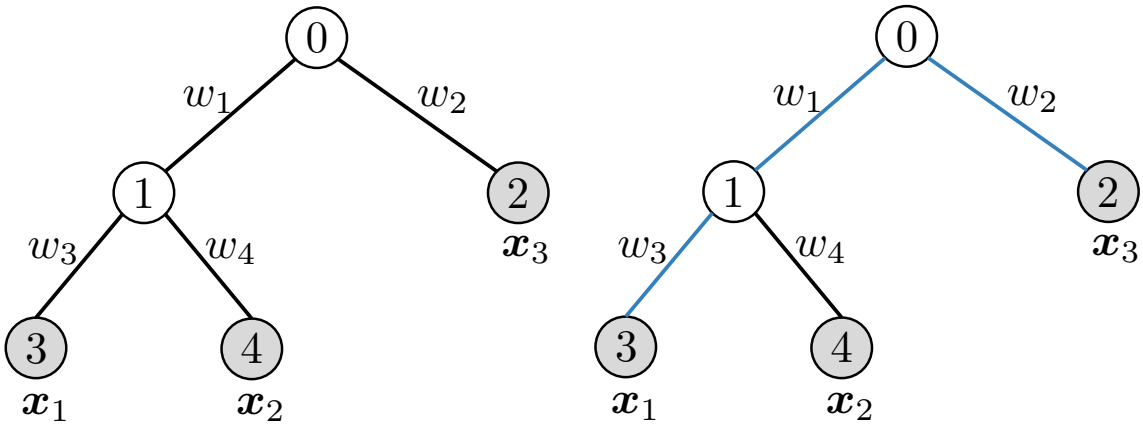
where $\boldsymbol{w} \in \mathbb{R}_+^N$ is the weight vector, $\boldsymbol{B} \in \{0, 1\}^{N \times N_{\mathrm{leaf}}}$ is a tree parameter, $[\boldsymbol{B}]_{i,j} = 1$ if the node i is the ancestor node of the leaf node j and zero otherwise, N is the total number of nodes of a tree, N_{leaf} is the number of leaf nodes, and $\mathrm{diag}(\boldsymbol{w}) \in \mathbb{R}_+^{N \times N}$ is a diagonal matrix whose diagonal elements are $\boldsymbol{w}$. Figure 1a shows an example of a tree embedding. In this tree, the total number of nodes is $N = 5$ and the number of leaf nodes is $N_{\mathrm{leaf}} = 3$. It is noteworthy that w_0 is the weight of the root node. Because $\boldsymbol{a}$ and $\boldsymbol{b}$ are probability vectors (i.e., $\boldsymbol{a}^\top \boldsymbol{1} = 1$), and the elements of the first row of $\boldsymbol{B}$ are all 1, the weight of the root node is ignored in $W_{\mathcal{T}}(\mu, \nu)$.

To compute TWD, the choice of tree structure and weight is crucial for approximating the Wasserstein distance. Indyk & Thaper (2003) proposed the QuadTree algorithm, which constructs a tree by recursively splitting a space to four regions (quad) and setting the weight parameter as $2^{-\ell(e)}$ ($\ell(e)$ is the depth of an edge e). For QuadTree, TWD is defined as

$$W_{\mathcal{T}}(\mu, \nu) = \sum_{e \in E} 2^{-\ell(e)} |\mu(\Gamma(v_e)) - \nu(\Gamma(v_e))|. \tag{10}$$

Le et al. (2019) proposed a clustering-based tree construction. More recently, Takezawa et al. (2021) proposed training a tree using continuous optimization in a supervised learning setup.

Previous research related to TWD has focused more on constructing a tree. However, learning edge weights in TWD has not been well examined. Thus, in this paper, we propose a weight estimation procedure for TWD to approximate Wasserstein distance.



(a) An illustration of tree embedding.

(b) Shortest path between $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$.

Figure 1: An illustration of tree embedding and the shortest path between $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ of the tree. All the input vectors are associated with the leaf nodes. The $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, and $\boldsymbol{b}_3$ of the tree are given as $\boldsymbol{b}_1 = (1, 1, 0, 1, 0)^\top$, $\boldsymbol{b}_2 = (1, 1, 0, 0, 1)^\top$, and $\boldsymbol{b}_3 = (1, 0, 1, 0, 0)^\top$, respectively. In this case, $d_{\mathcal{T}}(\boldsymbol{x}_1, \boldsymbol{x}_3) = w_3 + w_1 + w_2$.

4 Proposed method

In this section, we propose a tree weight estimation method using a non-negative Lasso to accurately approximate the 1-Wasserstein distance.

4.1 Tree weight estimation using Lasso

Because TWD is a 1-Wasserstein distance with a tree metric, it is natural to estimate the weight that can approximate the 1-Wasserstein distance with an arbitrary distance $d(\boldsymbol{x}, \boldsymbol{x}')$. Thus, we propose estimating the weight $\boldsymbol{w}$ by minimizing the error between the 1-Wasserstein distance with distance $d(\boldsymbol{x}, \boldsymbol{x}')$ and TWD. First, we show that TWD can approximate the vanilla Wasserstein distance arbitrarily well with some trees.

Proposition 1 *For any measure $\mu = \sum_{i=1}^{n} a_i \delta_{\boldsymbol{x}_i}$ and $\nu = \sum_{j=1}^{n'} b_j \delta_{\boldsymbol{x}'_j}$ and the distance function $d\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists a tree $\mathcal{T}$ such that $W_1(\mu, \nu) = W_{\mathcal{T}}(\mu, \nu)$ and $\boldsymbol{\Pi}^*(\mu, \nu) = \boldsymbol{\Pi}^*_{\mathcal{T}}(\mu, \nu)$ holds, where $\boldsymbol{\Pi}^*(\mu, \nu)$ is an optimal solution to Eq. (2) with cost $c = d$, and $\boldsymbol{\Pi}^*_{\mathcal{T}}(\mu, \nu)$ is an optimal solution with the cost function being the shortest path distance $d_{\mathcal{T}}$ on tree $\mathcal{T}$.*

The proof is provided in section A. This proposition indicates that TWD has sufficient expressive power to model the 1-Wasserstein distance.

Proposition 2 *(Le et al., 2019)*

$$W_{\mathcal{T}}(\mu, \nu) = \inf_{\pi \in \mathcal{U}(\mu, \nu)} \int_{\mathcal{X} \times \mathcal{X}'} d_{\mathcal{T}}(\boldsymbol{x}, \boldsymbol{x}') \mathrm{d}\pi(\boldsymbol{x}, \boldsymbol{x}'), \tag{11}$$

where $d_{\mathcal{T}}(\boldsymbol{x}, \boldsymbol{x}')$ is the length of the (unique) path between $\boldsymbol{x}$ and $\boldsymbol{x}'$ on a tree (i.e., the shortest path on the tree) and $\mathcal{U}$ is a set of joint probability distributions.

This proposition indicates that the 1-Wasserstein distance with an arbitrary distance metric and TWD are equivalent (i.e., $W_1(\mu, \nu) = W_{\mathcal{T}}(\mu, \nu)$) if $d(\boldsymbol{x}, \boldsymbol{x}') = d_{\mathcal{T}}(\boldsymbol{x}, \boldsymbol{x}')$. Thus, we aim to fit $d(\boldsymbol{x}, \boldsymbol{x}')$ using $d_{\mathcal{T}}(\boldsymbol{x}, \boldsymbol{x}')$, where the tree structure and edge weights are potential learning parameters. Figure 1b shows the shortest distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$. In this case, $d_{\mathcal{T}}(\boldsymbol{x}_1, \boldsymbol{x}_3) = \sum_{k \in \mathrm{Path}(\boldsymbol{x}_1, \boldsymbol{x}_3)} w_k = w_3 + w_1 + w_2$; Thus, a closed-form expression for the shortest path distance can be obtained as follows.

Algorithm 1 Sliced Weight Estimation with Trees

1: **Input:** The matrix $\boldsymbol{X}$, the regularization parameter $\lambda_1 \geq 0$, and a set of indices Ω.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: random.seed(i)
4: $\boldsymbol{B}_t :=$ QuadTree($\boldsymbol{X}$) or $\boldsymbol{B}_t :=$ ClusterTree($\boldsymbol{X}$)
5: Compute $\boldsymbol{z}_{i,j}^{(t)}$ from $\boldsymbol{B}_t$ and $d(\boldsymbol{x}_i, \boldsymbol{x}_j), (i,j) \in \Omega$
6: $\widehat{\boldsymbol{w}}_t := \operatorname{argmin}_{\boldsymbol{w} \in \mathbb{R}_+^{N_t}} \sum_{(i,j)\in\Omega} (d(\boldsymbol{x}_i, \boldsymbol{x}_j) - \boldsymbol{w}^\top \boldsymbol{z}_{i,j}^{(t)})^2 + \lambda \|\boldsymbol{w}\|_1$.
7: **end for**
8: **return** $\{(\boldsymbol{B}_t, \widehat{\boldsymbol{w}}_t)\}_{t=1}^T$

Proposition 3 *We denote by* $\boldsymbol{B} \in \{0,1\}^{N \times N_{leaf}} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_{N_{leaf}}]$ *and* $\boldsymbol{b}_i \in \{0,1\}^N$. *The shortest path distance between leaf* i *and leaf* j *can be represented as*

$$d_{\mathcal{T}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{w}^\top (\boldsymbol{b}_i + \boldsymbol{b}_j - 2\boldsymbol{b}_i \circ \boldsymbol{b}_j). \tag{12}$$

(Proof) *The shortest distance between leaf* i *and leaf* j *can be written as*

$$\begin{aligned} d_{\mathcal{T}}(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sum_{k \in \mathrm{Path}(\boldsymbol{x}_i, \boldsymbol{x}_j)} w_k \\ &= \sum_{k \in \mathcal{S}_i} w_k + \sum_{k' \in \mathcal{S}_j} w_{k'} - 2 \sum_{k \in \mathcal{S}_i \cap \mathcal{S}_j} w_k \\ &= \boldsymbol{w}^\top \boldsymbol{b}_i + \boldsymbol{w}^\top \boldsymbol{b}_j - 2\boldsymbol{w}^\top (\boldsymbol{b}_i \circ \boldsymbol{b}_j) \\ &= \boldsymbol{w}^\top (\boldsymbol{b}_i + \boldsymbol{b}_j - 2\boldsymbol{b}_i \circ \boldsymbol{b}_j), \end{aligned}$$

where $\mathcal{S}_i$ *includes the set of ancestor node indices of leaf* i *and node index of leaf* i. □

Given the closed-form expression in Proposition 3, we formulate the edge weight estimation problem as follows. First, we assumed that the tree was obtained using a tree construction algorithm such as QuadTree (Indyk & Thaper, 2003) and ClusterTree (Le et al., 2019), and we fixed the tree structure (i.e., fixed $\boldsymbol{B}$). We then propose the following optimization for learning $\boldsymbol{w}$:

$$\widehat{\boldsymbol{w}} := \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}_+^N} \sum_{i,j=1}^{m} (d(\boldsymbol{x}_i, \boldsymbol{x}_j) - \boldsymbol{w}^\top \boldsymbol{z}_{i,j})^2 + \lambda \|\boldsymbol{w}\|_1, \tag{13}$$

where $\boldsymbol{z}_{i,j} = \boldsymbol{b}_i + \boldsymbol{b}_j - 2\boldsymbol{b}_i \circ \boldsymbol{b}_j \in \mathbb{R}_+^N$, $\|\boldsymbol{w}\|_1$ is the L1-norm, and $\lambda \geq 0$ is a regularization parameter. This optimization problem is convex with respect to $\boldsymbol{w}$ and we can easily solve the problem using an off-the-shelf nonnegative Lasso solver. Note that $\boldsymbol{z}_{i,j}$ is a sparse vector with only $2H \ll N$ elements, where H is the tree depth. Thus, we can store the entire $\boldsymbol{z}$ in a sparse format and efficiently solve the Lasso problem. Moreover, thanks to the L1-regularization, we can make most of the weights zero; this corresponds to merging the nodes of the tree. Thus, with this optimization, we are not only able to estimate the tree's weight, but also compress its size.

We further propose a subsampled variant of the regression method:

$$\widehat{\boldsymbol{w}} := \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}_+^N} \sum_{(i,j)\in\Omega} (d(\boldsymbol{x}_i, \boldsymbol{x}_j) - \boldsymbol{w}^\top \boldsymbol{z}_{i,j})^2 + \lambda \|\boldsymbol{w}\|_1, \tag{14}$$

where Ω denotes a set of indices. In this paper, we randomly subsampled the indices.

Although our proposed method is specifically designed for the 1-Wasserstein distance approximation, it can also be used to approximate any distance using shortest path distance on a tree.

4.2 Tree-Slice extension

Thus far, we have focused on estimating the distance using a single tree. However, the approximation performance is highly dependent on the tree that is fixed before the weight estimation. To address this issue, we can use the tree-sliced Wasserstein distance (Le et al., 2019):

$$\bar{W}_{\mathcal{T}}(\mu,\nu) = \frac{1}{T}\sum_{t=1}^{T} W_{\mathcal{T}_t}(\mu,\nu), \tag{15}$$

where $W_{\mathcal{T}_t}(\mu,\nu)$ is TWD with the tth generated tree, and T is the total number of trees. In previous studies, TWD outperformed approaches based on a single tree. The matrix version of the tree-sliced Wasserstein distance can be written as

$$\bar{W}_{\mathcal{T}}(\mu,\nu) = \frac{1}{T}\sum_{t=1}^{T} \left\|\mathrm{diag}(\boldsymbol{w}_t)\boldsymbol{B}_t(\boldsymbol{a}-\boldsymbol{b})\right\|_1, \tag{16}$$

where $\boldsymbol{w}_t$ and $\boldsymbol{B}_t$ are the weight and parameters of the t-th tree, respectively.

Here, we extend the single-tree weight estimation to multiple trees. To this end, we separately estimate $\widehat{\boldsymbol{w}}_t$ from each tree and then average the estimates:

$$\bar{W}_{\mathcal{T}}(\mu,\nu) = \frac{1}{T}\sum_{t=1}^{T} \left\|\mathrm{diag}(\widehat{\boldsymbol{w}}_t)\boldsymbol{B}_t(\boldsymbol{a}-\boldsymbol{b})\right\|_1, \tag{17}$$

where

$$\widehat{\boldsymbol{w}}_t := \underset{\boldsymbol{w}\in\mathbb{R}_+^{N_t}}{\mathrm{argmin}} \sum_{(i,j)\in\Omega} (d(\boldsymbol{x}_i,\boldsymbol{x}_j) - \boldsymbol{w}^\top \boldsymbol{z}_{i,j}^{(t)})^2 + \lambda\|\boldsymbol{w}\|_1, \tag{18}$$

$\boldsymbol{z}_{i,j}^{(t)} \in \mathbb{R}_+^{N_t}$ is computed from $\boldsymbol{B}_t$, where N_t is the number of nodes in the t-th tree. We call the proposed methods with QuadTree and ClusterTree as Sliced-qTWD and Sliced-cTWD, respectively. Algorithm 1 is the pseudocode for the weight estimation procedure using trees. If we set $T = 1$, this corresponds to a single-tree case.

4.3 Computational cost

The computational cost of the proposed algorithm consists of three parts: computing a tree, estimating the weight, and computing TWD. In many Wasserstein distance applications, such as document retrieval, it is necessary to compute the tree and weight only once. In this paper, we employed QuadTree and ClusterTree, which are efficient methods of constructing trees. The computational complexity of QuadTree is linear with respect to the number of samples $O(dN_{\mathrm{leaf}}\log 1/\epsilon)$, where d is the number of input dimension and $\epsilon > 0$ is accuracy. For the ClusterTree (Le et al., 2019), we employed the farthest-point clustering (Gonzalez, 1985), and its computational complexity of $O(dHN_{\mathrm{leaf}}\log K)$, where H is the depth of tree and K is the number of clusters. For weight estimation, we employed the FISTA method (Beck & Teboulle, 2009), which has a efficient solver. Overall, when the number of vectors N_{leaf} is $10,000$ to $30,000$, the tree construction and weight estimation can be performed efficiently with a single CPU. For inference, TWD can be efficiently computed by using our tree-based approach, since its computational complexity is $O(N)$, where N is the total number of nodes of a tree. For the tree-sliced variant case, the computational cost of tree-sliced Wasserstein is T times larger than that of a single tree case.

5 Experiments

In this section, we evaluate our proposed methods using Twitter, BBCSport, and Amazon datasets.

Table 1: The results of 1-Wasserstein distance approximation. We report the averaged mean absolute error (MAE), the averaged Pearson's correlation coefficients (PCC), and the averaged number of nodes used for computation, respectively. For tree-sliced variants, we empirically fix the number of sliced as $T = 3$. For MAE and PCC computation, we computed the MAE and PCC between the 1-Wasserstein distance computed by EMD with Euclidean distance and the tree counterparts, and we run the algorithm 10 times by changing the random seed.

	Twitter			BBCSport			Amazon		
Methods	MAE	PCC	Nodes	MAE	PCC	Nodes	MAE	PCC	Nodes
QuadTree	0.486	0.702	5097.2	0.512	0.851	11463.1	0.573	0.596	34203.0
qTWD ($\lambda = 10^{-3}$)	0.053	0.880	4976.4	0.098	0.923	11016.4	0.072	0.789	32132.7
qTWD ($\lambda = 10^{-2}$)	0.053	0.879	4542.2	0.095	0.921	9968.5	0.067	0.783	28443.3
qTWD ($\lambda = 10^{-1}$)	0.051	0.873	4328.4	0.073	0.878	8393.8	0.051	0.732	16983.9
ClusterTree	8.028	0.674	5717.5	5.842	0.799	12217.7	8.074	0.759	33430.0
cTWD ($\lambda = 10^{-3}$)	0.039	0.885	5674.1	0.028	0.902	12145.3	0.036	0.785	32642.3
cTWD ($\lambda = 10^{-2}$)	0.038	0.881	5481.0	0.023	0.894	11031.8	0.033	0.778	28701.8
cTWD ($\lambda = 10^{-1}$)	0.036	0.867	4195.2	0.023	0.866	6680.7	0.028	0.768	9879.8
Sliced-QuadTree	0.463	0.796	15164.6	0.498	0.901	34205.7	0.552	0.759	102274.2
Sliced-qTWD ($\lambda = 10^{-3}$)	0.053	0.890	14702.7	0.100	0.945	32447.4	0.072	0.838	94737.4
Sliced-qTWD ($\lambda = 10^{-2}$)	0.052	0.892	13581.1	0.096	0.945	29966.5	0.067	0.849	85604.9
Sliced-qTWD ($\lambda = 10^{-1}$)	0.050	0.892	13026.3	0.073	0.925	25647.7	0.047	0.838	53689.9
Sliced-ClusterTree	7.989	0.804	17129.5	5.83	0.863	36644.2	8.071	0.822	100,251.3
Sliced-cTWD ($\lambda = 10^{-3}$)	0.035	0.929	17002.6	0.026	0.948	36405.2	0.035	0.888	97891.5
Sliced-cTWD ($\lambda = 10^{-2}$)	0.034	0.929	16414.7	0.021	0.943	33024.7	0.031	0.884	86068.8
Sliced-cTWD ($\lambda = 10^{-1}$)	0.029	0.930	12565.3	0.018	0.924	19947.2	0.022	0.870	29540.0

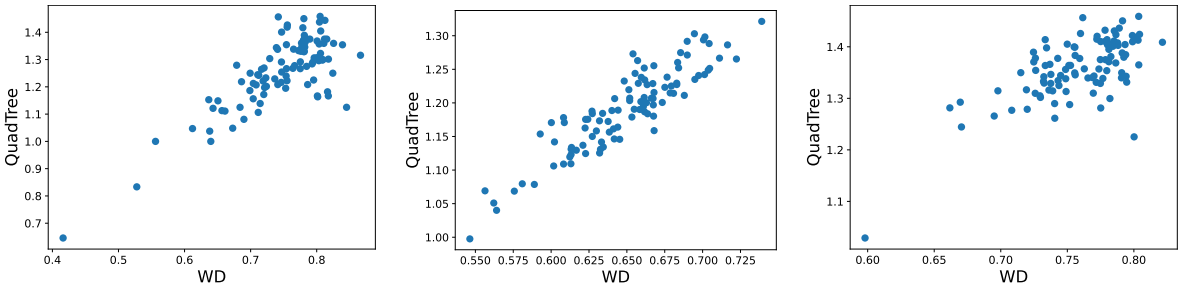
5.1 Setup

We evaluated the approximation ability of the proposed method. More specifically, we measured the error and correlation between the 1-Wasserstein distance with the Euclidean norm and tree-based methods. For the tree-based methods, we evaluated QuadTree (Indyk & Thaper, 2003) and ClusterTree (Le et al., 2019). For ClusterTree, we set the number of clusters $K = 5$ for all experiments. For the 1-Wasserstein distance, we used the Python Optimal Transport (POT) package [1]. For tree methods, we first construct a tree using an entire word embedding vector $\boldsymbol{X}$ and then compute the Wasserstein distance with the tree. For the proposed methods, we selected the regularization parameter from $\{10^{-3}, 10^{-2}, 10^{-1}\}$. qTWD and cTWD are both proposed methods, and qTWD and cTWD employ QuadTree and ClusterTree, respectively. We compared all the methods using the Twitter, BBCSport, and Amazon datasets [2]. We further evaluated the group-sliced TWD using QuadTree and ClusterTree. We randomly subsample 100 document pairs and computed the mean absolute error (MAE) and Pearson's correlation coefficient (PCC) between the 1-Wasserstein distance computed by EMD with Euclidean distance and the tree counterparts. Moreover, we reported the number of nonzero tree weights. In this experiment, we set the number of slices to $T = 3$ and the regularization parameters $\lambda = \{10^{-3}, 10^{-2}, 10^{-1}\}$. We used SPAMS to solve Lasso problems [3]. For all methods, we ran the algorithm ten times by changing the random seed. We evaluated all methods using Xeon CPU E5-2690 v4 (2.60 GHz) and Xeon CPU E7-8890 v4 (2.20GHz). Note that since the number of vectors $\boldsymbol{x}_i$ tends to be large for real-world applications, we only evaluated the subsampled training method.

5.2 Results of the approximation of 1-Wasserstein distance

Table 1 presents the experimental results for the Twitter, BBCSport, and Amazon datasets. As shown, qTWD and cTWD can obtain a small MAE and accurately approximate the original 1-Wasserstein distance. In contrast, QuadTree and ClusterTree had larger MAE values than the proposed methods. Although QuadTree has a theoretical guarantee, the MAE of QuadTree is one order of magnitude larger than that of

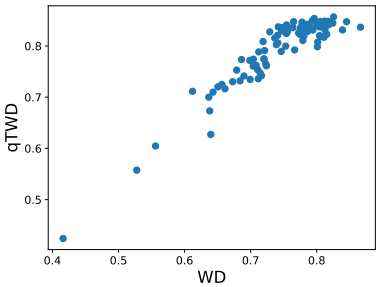
[1] https://pythonot.github.io/index.html
[2] https://github.com/gaohuang/S-WMD
[3] http://thoth.inrialpes.fr/people/mairal/spams/



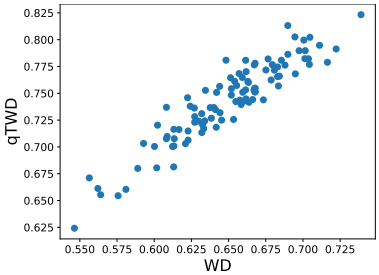
(a) Twitter, QuadTree ($\rho = 0.781$).

(b) BBCSport, QuadTree ($\rho = 0.904$).

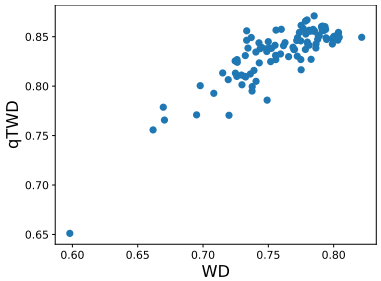
(c) Amazon, QuadTree ($\rho = 0.669$)



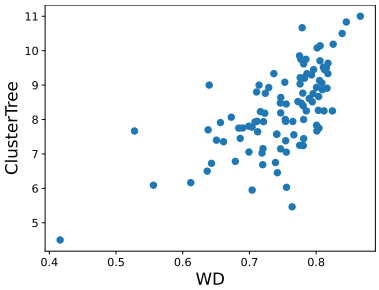
(d) Twitter, qTWD ($\rho = 0.930$).



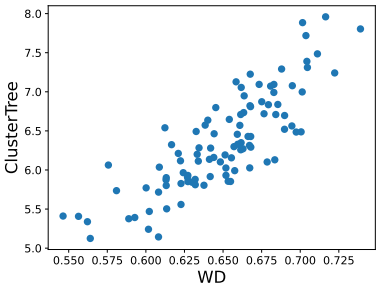
(e) BBCSport, qTWD ($\rho = 0.929$).



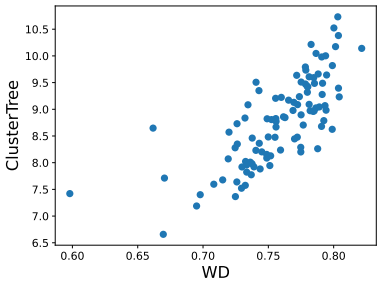
(f) Amazon, qTWD ($\rho = 0.848$).



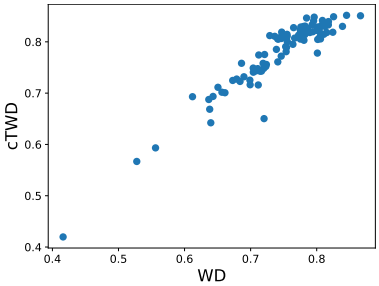
(g) Twitter, ClusterTree ($\rho = 0.651$).



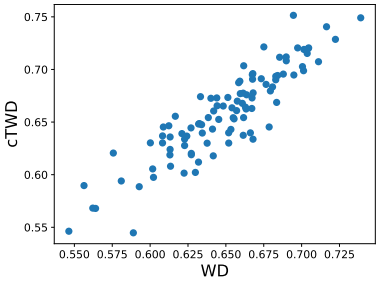
(h) BBCSport, ClusterTree ($\rho = 0.828$).



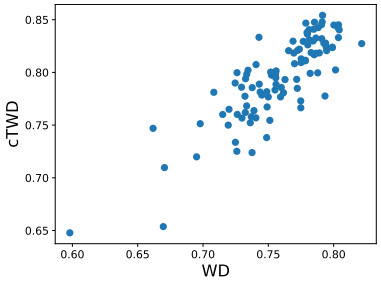
(i) Amazon, ClusterTree ($\rho = 0.747$)



(j) Twitter, cTWD ($\rho = 0.942$).



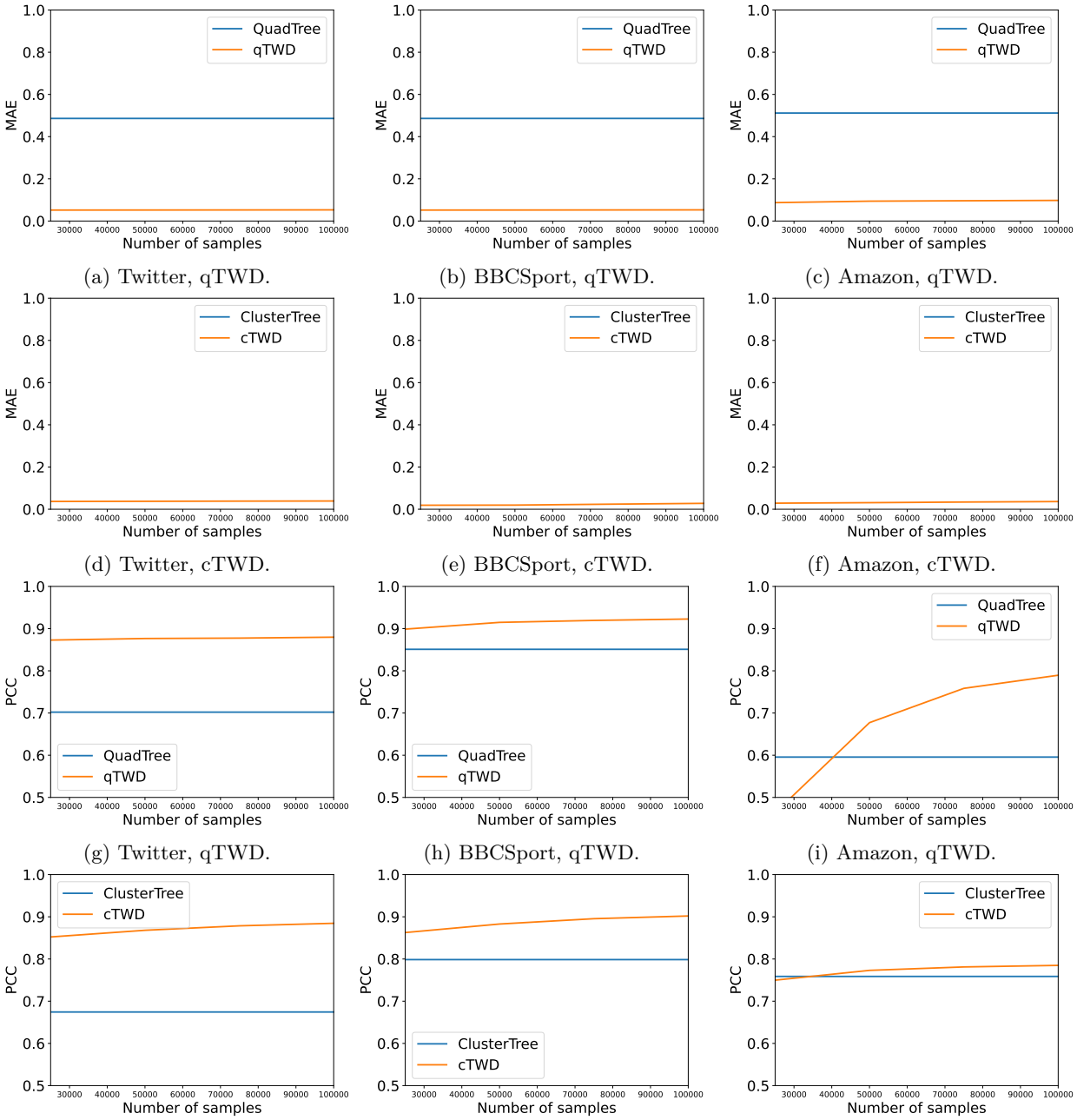
(k) BBCSport, cTWD ($\rho = 0.898$).



(l) Amazon, cTWD ($\rho = 0.841$).

Figure 2: Scatter plots of the Twitter, BBCSport, and Amazon datasets.

the proposed method. For ClusterTree, since ClusterTree is based on clustering and the tree construction is independent of the scaling of vectors, it cannot guarantee a good approximation of 1-Wasserstein. However, our proposed method can approximate 1-Wasserstein even if we use ClusterTree. For PCC, qTWD and cTWD outperformed the vanilla QuadTree and ClusterTree if we set a small λ. Moreover, even when using half of the nodes (i.e., $\lambda = 10^{-1}$), PCCs of qTWD and cTWD were comparable to QuadTree and ClusterTree. Figure 2 shows scatter plots for each method. We can see that all methods correlate with the 1-Wasserstein distance. However, it is evident that the proposed method has a similar scale to that of the original WD, whereas QuadTree and ClusterTree have larger values than the original WD. Moreover, Figure 3 shows the



(a) Twitter, qTWD. (b) BBCSport, qTWD. (c) Amazon, qTWD.

(d) Twitter, cTWD. (e) BBCSport, cTWD. (f) Amazon, cTWD.

(g) Twitter, qTWD. (h) BBCSport, qTWD. (i) Amazon, qTWD.

(j) Twitter, cTWD. (k) BBCSport, cTWD. (l) Amazon, cTWD.

Figure 3: MAE and PCC with respect to the number of training samples. In this experiments, we computed MAE and PCC for $m \in \{25,000, 50,000, 75,000, 100,000\}$. Note that MAEs of the ClusterTree are bigger than 1, and they do not appear in (d),(e), and (f).

effect of the training sample size used for training Eq. (14). In these experiments, the number of samples was changed to $25,000$, $50,000$, $75,000$, and $100,000$. We found that PCC could be improved by using more samples. In contrast, the MAE is small, even if we use a relatively small number of samples.

With the tree-sliced, as reported in (Le et al., 2019), Sliced-QuadTree and Sliced-ClusterTree outperform their non-sliced counterparts. Although there is no gain in MAE, our proposed methods, Sliced-qTWD and Sliced-cTWD can significantly improve the PCC values. Furthermore, because we use distance to retrieve

Table 2: The document classification results for Twitter, BBCSport, and Amazon datasets. We used the ten nearest neighbor classifiers. For the Sinkhorn algorithm, we set the regularization parameter with 10^{-2} and the maximum iteration with 100, respectively. We run the algorithms five times by changing the random seed and reporting the averaged classification accuracy.

Methods	Twitter	BBCSport	Amazon
WMD (Sinkhorn)	0.675	0.972	0.905
QuadTree	0.691	0.970	0.865
qTWD ($\lambda = 10^{-3}$)	0.697	0.967	0.847
Sliced-QuadTree	0.694	0.970	0.877
Sliced-qTWD ($\lambda = 10^{-3}$)	0.697	0.967	0.887
ClusterTree	0.682	0.901	0.873
cTWD ($\lambda = 10^{-3}$)	0.694	0.962	0.878
Sliced-ClusterTree	0.694	0.929	0.900
Sliced-cTWD ($\lambda = 10^{-3}$)	0.700	0.970	0.905

similar documents, we can obtain similar results using the Wasserstein distance. More interestingly, for the Amazon dataset, Sliced-cTWD ($\lambda = 10^{-1}$) has a PCC value of 0.870 with 29,540 nodes, whereas cTWD ($\lambda = 10^{-3}$) has a PCC value of 0.785 with 32,642.3 nodes. Thus, Sliced-cTWD significantly outperformed cTWD. Thus, to obtain a low approximation error, one strategy is to use the sliced version and prune unimportant nodes using Lasso.

5.3 Results of Document Classification

We evaluate the document classification experiments tasks with 1-Wasserstein distance (Sinkhorn algorithm), QuadTree, ClusterTree, qTWD, cTWD, and the sliced counterparts. For proposed method, we set the regularization parameter as $\lambda = 10^{-3}$. In this experiment, we used the 90% of samples for training and the rest of samples for the test. We then used the ten nearest neighbor classification method. We run five times by changing the random seed and report the average mean accuracy.

Table 2 shows the classification accuracy for each methods. The proposed weighting-based approaches compare favorably with the Word Mover's Distance (WMD) with the Sinkhorn algorithm. In particular, the proposed method can significantly gain the ClusterTree cases, while the improvements from the vanilla QuadTree method are modest.

5.4 Computation and memory costs

Table 3 summarizes the computational cost for training and test. For training, we measured the average computational cost using Xeon CPU E5-2690 v4 (2.60 GHz) for computing cost matrix, tree construction[4], training set generation for weight estimation, and the Lasso optimization. The training set generation process includes calculating the cost matrix and the $\boldsymbol{z}$ vectors. For the test, we compute the average time consumption for comparing the entire documents in training set with one document for the test. As can be observed, the training set generation for training is the most computationally expensive operation for our proposed method. The computational costs for computing cost matrix and tree construction are comparable. In contrast, the computational cost of the Lasso optimization is small. Overall, the computational cost of training increases if the number of training vectors N_{leaf} increases. For the test, the proposed method's computational speed is several orders of magnitude faster with GPUs. It is important to note that the proposed approximation approach cannot compute the exact 1-Wasserstein distance; the proposed approach may not be suited for the task that we should evaluate the 1-Wasserstein distance exactly.

Memory cost: Table 4 shows the required memory space for each method. Note that we report the size of dense matrices because dense matrices are more suited for GPU. However, since the tree matrix $\boldsymbol{B}$ is

[4]Due to our implementation of tree-construction, the computational complexities of QuadTree and ClusterTree are $O(dN_{\mathrm{leaf}})$ and $O(dN_{\mathrm{leaf}}HK)$, respectively.

Table 3: The time comparison of Twitter, BBCSport, and Amazon datasets. For training, we compute the time using Xeon CPU E5-2690 v4 (2.60 GHz). For the test, we run all the methods with an A6000 GPU. We run the algorithm five times by changing the random seed and report the averaged computational time. We compute the average time consumption for comparing the entire documents in training set with one document for the test. The number in the parenthesis of each dataset is the number of the word vectors used in this paper.

Dataset	Methods	Training (CPU)				Test (GPU)
		Cost matrix	Tree const.	Training set gen.	Lasso	
Twitter ($N_{\text{leaf}} = 4,489$)	Sinkhorn	3.0	n/a	n/a	n/a	0.159
	QuadTree	n/a	8.0	n/a	n/a	9.1×10^{-5}
	ClusterTree	n/a	6.0	n/a	n/a	8.8×10^{-5}
	qTWD ($\lambda = 10^{-3}$)	n/a	8.0	39.4	4.1	8.8×10^{-5}
	cTWD ($\lambda = 10^{-3}$)	n/a	6.0	49.8	11.7	9.0×10^{-5}
BBCsport ($N_{\text{leaf}} = 10,103$)	Sinkhorn	14.7	n/a	n/a	n/a	0.125
	QuadTree	n/a	24.9	n/a	n/a	8.7×10^{-5}
	ClusterTree	n/a	21.3	n/a	n/a	8.8×10^{-5}
	qTWD ($\lambda = 10^{-3}$)	n/a	24.9	95.6	4.5	8.7×10^{-5}
	cTWD ($\lambda = 10^{-3}$)	n/a	21.3	109.7	12.4	8.7×10^{-5}
Amazon ($N_{\text{leaf}} = 30,249$)	Sinkhorn	183.4	n/a	n/a	n/a	2.24
	QuadTree	n/a	162.2	n/a	n/a	1.1×10^{-4}
	ClusterTree	n/a	126.3	n/a	n/a	1.1×10^{-4}
	qTWD ($\lambda = 10^{-3}$)	n/a	162.2	368.8	5.6	1.2×10^{-4}
	cTWD ($\lambda = 10^{-3}$)	n/a	126.3	366.8	12.9	1.2×10^{-4}

Table 4: Memory usage of each method for test time. Note that we report the size of dense matrices because dense matrices are more suited for GPU. However, since the tree matrix $\boldsymbol{B}$ is sparse, the required memory space for $\boldsymbol{B}$ is negligible in practice. Thus, if we compute TWD with CPU, the required memory space is much less than that of Sinkhorn-based algorithms.

Methods	Twitter ($N_{\text{leaf}} = 4,489$)		BBCSport ($N_{\text{leaf}} = 10,103$)		Amazon ($N_{\text{leaf}} = 30,249$)	
	Cost matrix	Tree matrix	Cost matrix	Tree matrix	Cost matrix	Tree matrix
Sinkhorn	161.2 MB	n/a	816.6 MB	n/a	7320.0 MB	n/a
QuadTree	n/a	184.8 MB	n/a	929.7 MB	n/a	8149.3 MB
ClusterTree	n/a	203.4 MB	n/a	982.6 MB	n/a	8089.1 MB
qTWD ($\lambda = 10^{-3}$)	n/a	184.8 MB	n/a	929.7 MB	n/a	8149.3 MB
cTWD ($\lambda = 10^{-3}$)	n/a	203.4 MB	n/a	982.6 MB	n/a	8089.1 MB

sparse, the required memory space for $\boldsymbol{B}$ is negligible in practice. Thus, if we compute TWD with CPU, the required memory space is much less than that of Sinkhorn-based algorithms.

6 Conclusion

In this paper, we consider approximating the 1-Wasserstein distance with trees. More specifically, we first showed that the 1-Wasserstein distance approximation can be formulated as a distance-approximation problem. Then, we proposed a non-negative Lasso-based optimization technique for learning the weights of a tree. Because the proposed method is convex, a globally optimal solution can be obtained. Moreover, we proposed a weight estimation procedure for a tree-sliced variant of Wasserstein distance. Through experiments, we show that the proposed weight estimation method can significantly improve the approximation performance of the 1-Wasserstein distance for both QuadTree and clustered tree cases. Owing to L1-regularization, we can compress the tree size without losing approximation performance.

A potential limitation of our proposal is that the tree-based approach may not be suited for tasks that require the exact evaluation of the 1-Wasserstein distance. Moreover, since it needs to construct a tree from data, the proposed approach may not be suited for the end-to-end training of deep neural network models. Another limitation of the tree-based approach is that it may not be suited for a matching problems, such as a semantic correspondence detection task (Liu et al., 2020). In contrast, the tree-based approach is super

fast with reasonable approximation performance of the original 1-Wasserstein distance. Thus, the proposed tree-based approach is more suited when we compute a large number of the 1-Wasserstein distances, such as document retrieval and document classification tasks (i.e., k-nearest neighbor classifier).

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Kubilay Atasu and Thomas Mittelholzer. Linear-complexity data-parallel earth mover's distance approximations. In *ICML*, 2019.

Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. In *ICML*, 2020.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *ICML*, 2020.

Charlotte Bunne, Laetitia Papaxanthos, Andreas Krause, and Marco Cuturi. Proximal optimal transport modeling of population dynamics. In *AISTATS*, 2022.

Nicolas Courty, Rémi Flamary, Amaury Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. *NeurIPS*, 2017.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *NIPS*, 2013.

Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In *ICML*, 2014.

Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *NeurIPS*, 2019.

Ishan Deshpande, Yuan-Ting Hu, Ruoyu Sun, Ayis Pyrros, Nasir Siddiqui, Sanmi Koyejo, Zhizhen Zhao, David Forsyth, and Alexander G Schwing. Max-sliced Wasserstein distance and its use for GANs. In *CVPR*, 2019.

Tamal K Dey and Simon Zhang. Approximating 1-wasserstein distance between persistence diagrams by graph sparsification. In *ALENEX*, 2022.

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, 2012.

Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. *NIPS*, 2016.

Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with sinkhorn divergences. In *AISTATS*, 2018.

Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *NeurIPS*, 2017.

Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *3rd international workshop on statistical and computational theories of vision*, volume 2, pp. 5. Nice, France, 2003.

Soheil Kolouri, Yang Zou, and Gustavo K Rohde. Sliced wasserstein kernels for probability distributions. In *CVPR*, 2016.

Soheil Kolouri, Phillip E Pope, Charles E Martin, and Gustavo K Rohde. Sliced wasserstein auto-encoders. In *ICLR*, 2018.

Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde. Generalized sliced wasserstein distances. In *NeurIPS*, 2019.

Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *ICML*, 2015.

Tam Le and Truyen Nguyen. Entropy partial transport with tree metrics: Theory and practice. In *AISTATS*, 2021.

Tam Le, Makoto Yamada, Kenji Fukumizu, and Marco Cuturi. Tree-sliced approximation of wasserstein distances. *NeurIPS*, 2019.

Ruilin Li, Xiaojing Ye, Haomin Zhou, and Hongyuan Zha. Learning to match via inverse optimal transport. *Journal of machine learning research*, 20, 2019.

Tianyi Lin, Chenyou Fan, Nhat Ho, Marco Cuturi, and Michael Jordan. Projection robust wasserstein distance and riemannian optimization. *NeurIPS*, 2020.

Yanbin Liu, Linchao Zhu, Makoto Yamada, and Yi Yang. Semantic correspondence as an optimal transport problem. In *CVPR*, 2020.

Catherine Lozupone and Rob Knight. Unifrac: a new phylogenetic method for comparing microbial communities. *Applied and environmental microbiology*, 71(12):8228–8235, 2005.

Arthur Mensch and Gabriel Peyré. Online sinkhorn: Optimal transport distances from sample streams. *NeurIPS*, 2020.

Jonas W Mueller and Tommi Jaakkola. Principal differences analysis: Interpretable characterization of differences between distributions. *NIPS*, 2015.

Khai Nguyen, Dang Nguyen, Tung Pham, Nhat Ho, et al. Improving mini-batch optimal transport via partial transportation. In *ICML*, 2022.

Vu Nguyen, Tam Le, Makoto Yamada, and Michael A Osborne. Optimal transport kernels for sequential and parallel neural architecture search. In *ICML*, 2021.

François-Pierre Paty and Marco Cuturi. Subspace robust wasserstein distances. In *ICML*, 2019.

Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot. Wasserstein barycenter and its application to texture mixing. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pp. 435–446. Springer, 2011.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *CVPR*, 2020.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Fast unbalanced optimal transport on tree. In *NeurIPS*, 2020.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Re-evaluating word mover's distance. *ICML*, 2022.

Jian Shen, Yanru Qu, Weinan Zhang, and Yong Yu. Wasserstein distance guided representation learning for domain adaptation. In *AAAI*, 2018.

Yuki Takezawa, Ryoma Sato, and Makoto Yamada. Supervised tree-wasserstein distance. In *ICML*, 2021.

Yuki Takezawa, Ryoma Sato, Zornitsa Kozareva, Sujith Ravi, and Makoto Yamada. Fixed support tree-sliced wasserstein barycenter. *AISTATS*, 2022.

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. *NeurIPS*, 2020.

Sho Yokoi, Ryo Takahashi, Reina Akama, Jun Suzuki, and Kentaro Inui. Word rotator's distance. *EMNLP*, 2020.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M Meyer, and Steffen Eger. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. *EMNLP-IJCNLP*, 2019.

A Proof of Proposition 1

Let $\mathbf{\Pi}^* = \mathbf{\Pi}^*(\mu, \nu)$. If there exists (i, j, k) such that $\pi_{ij} > 0$ and $\pi_{jk} > 0$,

$$\pi'_{pq} = \begin{cases} \pi_{ij} - \min(\pi_{ij}, \pi_{jk}) & (p, q) = (i, j) \\ \pi_{jk} - \min(\pi_{ij}, \pi_{jk}) & (p, q) = (j, k) \\ \pi_{ik} + \min(\pi_{ij}, \pi_{jk}) & (p, q) = (i, k) \\ \pi_{pq} & \text{otherwise} \end{cases} \tag{19}$$

is no worse than $\mathbf{\Pi}^*$ because $d(i, j) + d(j, k) \geq d(i, k)$ due to the triangle inequality. We assume no tuples (i, j, k) exists such that $\pi_{ij} > 0$ and $\pi_{jk} > 0$ without loss of generality. If there exist $i_1, i_2, \cdots, i_{2n}, i_{2n+1} = i_1$ such that $\boldsymbol{P}_{i_1 i_2} > 0, \boldsymbol{P}_{i_3 i_2} > 0, \boldsymbol{P}_{i_3 i_4} > 0, \cdots, \boldsymbol{P}_{i_1 i_{2n}} > 0$, either

$$\pi'_{pq} = \begin{cases} \pi_{i_{2k+1} i_{2k+2}} - \varepsilon & (p, q) = (i_{2k+1}, i_{2k+2}) \\ \pi_{i_{2k+3} i_{2k+2}} + \varepsilon & (p, q) = (i_{2k+3}, i_{2k+2}) \\ \pi_{pq} & \text{otherwise} \end{cases} \tag{20}$$

or

$$\pi''_{pq} = \begin{cases} \pi_{i_{2k+1} i_{2k+2}} + \varepsilon & (p, q) = (i_{2k+1}, i_{2k+2}) \\ \pi_{i_{2k+3} i_{2k+2}} - \varepsilon & (p, q) = (i_{2k+3}, i_{2k+2}) \\ \pi_{pq} & \text{otherwise} \end{cases} \tag{21}$$

is no worse than $\mathbf{\Pi}^*$, where $\varepsilon = \min\{\pi_{i_{2k+1} i_{2k+2}} \mid k = 0, \cdots, n-1\} \cup \{\pi_{i_{2k+3} i_{2k+1}} \mid k = 0, \cdots, n-1\}$, and both supports of π'_{pq} and π''_{pq} do not have this cycle. Therefore, we assume the support of $\boldsymbol{P}^*$ has no cycle without loss of generality, i.e., the graph is a tree. Let $\mathcal{T}$ be the graph induced by the support of $\mathbf{\Pi}^*$ and let the weight of edge $e = (i, j)$ be $w_e = d(w_i, w_j)$. As the shortest path distance between u and v on tree $\mathcal{T}$ is no less than $d(u, v)$ by definition, $W_1(\mu, \nu) \leq W_{\mathcal{T}}(\mu, \nu)$ holds. However, any pair (i, j) on the support of $\mathbf{\Pi}^*$ is directly connected in $\mathcal{T}$. Therefore,

$$\begin{aligned} W_1(\mu, \nu) &= \min_{\mathbf{\Pi} \in U(\mu, \nu)} \sum_{i=1}^{n} \sum_{j=1}^{n'} \pi_{ij} d(\boldsymbol{x}_i, \boldsymbol{x}'_j), \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n'} \pi^*_{ij} d(\boldsymbol{x}_i, \boldsymbol{x}'_j) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n'} \pi^*_{ij} d_{\mathcal{T}}(\boldsymbol{x}_i, \boldsymbol{x}'_j) \\ &\geq \min_{\mathbf{\Pi} \in U(\mu, \nu)} \sum_{i=1}^{n} \sum_{j=1}^{n'} \pi_{ij} d_{\mathcal{T}}(\boldsymbol{x}_i, \boldsymbol{x}'_j) \\ &= W_{\mathcal{T}}(\mu, \nu). \end{aligned}$$

Therefore, $W_1(\mu, \nu) = W_{\mathcal{T}}(\mu, \nu)$ and $\boldsymbol{P}^*$ is an optimal solution for both $W_1(\mu, \nu)$ and $W_{\mathcal{T}}(\mu, \nu)$.