A HIGHER PRECISION ALGORITHM FOR COMPUTING THE 1-WASSERSTEIN DISTANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the problem of computing the 1-Wasserstein distance $\mathcal{W}(\mu, \nu)$ between two d-dimensional discrete distributions μ and ν that are within the unit hypercube. Let A (resp. B) be the support of μ (resp. ν). There are several algorithms that estimate $\mathcal{W}(\mu, \nu)$ within an additive error of ε. However, when $\mathcal{W}(\mu, \nu)$ is small, the additive error ε dominates, leading to noisy results. Consider any additive approximation algorithm with execution time $T(n, \varepsilon)$. We propose an algorithm that runs in $O(T(n, \varepsilon/d) \log n)$ time and boosts the accuracy of estimating $\mathcal{W}(\mu, \nu)$ from ε to an expected additive error of $\min\{\varepsilon, (d \log_{\sqrt{d}/\varepsilon} n)\mathcal{W}(\mu, \nu)\}$. For the special case where every point in $A \cup B$ has a mass of $1/n$ (also called the Euclidean Bipartite Matching problem), we describe an algorithm to boost the accuracy of any additive approximation algorithm from ε to an expected additive error of $\min\{\varepsilon, (d \log\log n)\mathcal{W}(\mu, \nu)\}$ in $O(T(n, \varepsilon/d) \log\log n)$ time.

1 INTRODUCTION

Given two discrete probability distributions μ (with support A) and ν (with support B) inside a d-dimensional unit hypercube in the Euclidean space with $\max\{|A|, |B|\} = n$, the 1-Wasserstein distance $\mathcal{W}(\mu, \nu)$ (also called the Earth Mover's distance) between them is the minimum cost required to transport mass from ν to μ. The special case where $|A| = |B| = n$ and the mass at each point of $A \cup B$ is $1/n$ is called the Euclidean Bipartite Matching (EBM) problem. In machine learning applications, one can improve a model μ by using its Earth Mover's distance from a distribution ν built on real data. Consequently, it has been extensively used in generative models (Deshpande et al. (2018); Genevay et al. (2018); Salimans et al. (2018)), robust learning (Esfahani & Kuhn (2018)), supervised learning (Luise et al. (2018); Janati et al. (2019)), and parameter estimation (Liu et al. (2018); Bernton et al. (2019)).

One can find the 1-Wasserstein distance between μ and ν in $O(n^3 \log n)$ time (Edmonds & Karp (1972); Orlin (1988)), which is computationally expensive. Therefore, there has been substantial focus on the design of approximation algorithms that estimate $\mathcal{W}(\mu, \nu)$ within an additive error of ε in $n^2 \mathrm{poly}(d, \log n, 1/\varepsilon)$ time. We refer to the transport cost corresponding to such approximations as an ε-*close* transport cost. When $\mathcal{W}(\mu, \nu)$ is significantly smaller than ε, however, the cost produced by such algorithms will be unreliable as it is dominated by the error parameter ε.

To get a higher accuracy in this case, one can compute a relative approximation of the 1-Wasserstein distance. For a parameter $\alpha > 1$, an α-*relative approximation* is a cost that is within $\alpha\mathcal{W}(\mu, \nu)$. There has been considerable effort in the design of relative approximation algorithms; however, many such methods suffer from curse of dimensionality; i.e, their execution time grows exponentially in d. Furthermore, they rely on fairly involved data structures that have good asymptotic execution times but are difficult to implement and have large hidden constants within them making them impractical (Agarwal & Sharathkumar (2014); Fox & Lu (2020); Agarwal et al. (2022a;b)). The only exception to this are algorithms designed by using a simple hierarchical quadtree-based paradigm. The classical algorithm in this paradigm is based on a d-dimensional quadtree and returns an $O(d \log n)$-approximation of the 1-Wasserstein distance in $O(nd)$ time and has been used in various machine learning and computer vision applications (Gupta et al. (2010); Backurs et al. (2020)). In the case of the Euclidean Bipartite Matching, Agarwal & Varadarajan (2004) and Indyk

(2007) generalized the algorithm in the hierarchical framework to achieve an approximation ratio of $O(d^2 \log 1/\varepsilon)$ in $\tilde{O}(n^{1+\varepsilon})$ time[1].

In this paper, we use any additive approximation algorithm as a black-box within the hierarchical framework for relative approximation algorithms, resulting in better guarantees for both additive and relative approximations. To our knowledge, this is the *first* result that combines the power of additive and relative approximation techniques, leading to improvement in both settings.

1.1 Problem Definition

In the 1-Wasserstein problem, we are given two discrete distributions μ and ν. Let A and B be the points that define μ and ν, respectively. For the distribution μ (resp. ν), suppose each point $a \in A$ (resp. $b \in B$) has a probability of μ_a (resp. ν_b) associated with it, where $\sum_{a \in A} \mu_a = \sum_{b \in B} \nu_b = 1$. Let $G(A, B)$ denote the complete bipartite graph where, for any pair of points $a \in A$ and $b \in B$, there is an edge from a to b of cost $\|a - b\|$, i.e, the Euclidean distance between a and b.

For each point $a \in A$ (resp. $b \in B$), we assign a weight $\eta(a) = -\mu_a$ (resp. $\eta(b) = \nu_b$). We refer to any point $v \in A \cup B$ with a negative (resp. positive) weight as a *demand point* (resp. *supply point*) with a demand (resp. supply) of $|\eta(v)|$. Given any subset of points $V \subseteq A \cup B$, the weight $\eta(V)$ is simply the sum of the weights of its points; i.e, $\eta(V) = \sum_{v \in V} \eta(v)$. For any edge $(a, b) \in A \times B$, let the cost of transporting a supply of β from b to a be $\beta\|a - b\|$. In this problem, our goal is to transport all supplies from supply points to demand points with a minimum cost. More formally, a transport plan is a function $\sigma : A \times B \to \mathbb{R}_{\geq 0}$ that assigns a non-negative value to each edge of $G(A, B)$ indicating the quantity of supply transported along the edge. The transport plan σ is such that the total supplies transported into (resp. from) any demand (resp. supply) point $a \in A$ (resp. $b \in B$) is equal to $-\eta(a)$ (resp. $\eta(b)$). The cost of the transport plan σ, denoted by $w(\sigma)$, is given by $\sum_{(a,b) \in A \times B} \sigma(a, b)\|a - b\|$. The goal of this problem is to find a minimum-cost transport plan.

If two points $a \in A$ and $b \in B$ are co-located (i.e, they share the same coordinates), then due to the metric property of the Euclidean distances, if $\eta(b) = -\eta(a)$, we can match the supplies to the demands at zero cost and remove the points from the input. Otherwise, if $\eta(b) \neq -\eta(a)$, we replace the two points with a single point of weight $\eta(a) + \eta(b)$. By definition, if the weight of the newly created point is negative (resp. positive), we consider it a demand (resp. supply) point. In our presentation, we always consider A and B to be the point sets obtained after replacing all the co-located points. Observe that the total supply $U = \eta(B)$ need not be 1. However, it is easy to see that $\eta(B) = -\eta(A)$; i.e, the problem instance defined on $A \cup B$ is *balanced*. We say that a transport plan σ is an ε-close transport plan if $w(\sigma) \leq \mathcal{W}(\mu, \nu) + \varepsilon U$.

In many applications, the distributions μ and ν are continuous or large (possibly unknown) discrete distributions. In such cases, it might be impossible to compute $\mathcal{W}(\mu, \nu)$. Instead, one can draw two sets A and B of n samples each from μ and ν, respectively. Each point $a \in A$ (resp. $b \in B$) is assigned a weight of $\eta(a) = -1/n$ (resp. $\eta(b) = 1/n$). One can approximate the 1-Wasserstein distance between the distributions μ and ν by simply solving the 1-Wasserstein problem defined on $G(A, B)$. This special case where every point has the same demand and supply is called the *Euclidean Bipartite Matching* (EBM) problem. A *matching* M is a set of vertex-disjoint edges in $G(A, B)$ and has a cost $1/n \sum_{(a,b) \in M} \|a - b\|$. For the special case of the EBM problem, the optimal transport plan is simply a minimum-cost matching of cardinality n.

For any point set P in the Euclidean space, let $C_{\max}(P) := \max_{(a,b) \in P \times P} \|a - b\|$ denote the distance of its farthest pair and $C_{\min}(P) := \min_{(a,b) \in P \times P, a \neq b} \|a - b\|$ denote the distance of its closest pair. The *spread* of the point set, denoted by $\Delta(P)$, is the ratio $\Delta(P) = C_{\max}(P)/C_{\min}(P)$. When P is obvious from the context, we simply use $C_{\min}$, $C_{\max}$, and Δ to denote the distance of its closest and farthest pair and its spread.

1.2 Related Work

Relative Approximations: In fixed dimensional settings, i.e., $d = O(1)$, there is extensive work on the design of near-linear time $(1 + \varepsilon)$-relative approximation for 1-Wasserstein and EBM problems.

[1] $\tilde{O}()$ hides $\mathrm{poly}(d, \log n, 1/\varepsilon)$ factors in the execution time.

The execution time of these algorithms are $\Omega(n(d\varepsilon^{-1}\log n)^d)$ (Khesin et al. (2019); Raghvendra & Agarwal (2020); Fox & Lu (2020); Agarwal et al. (2022a)). A recent algorithm presented by Agarwal et al. (2022b) improved the dependence on d slightly and achieved an execution time of $n(d\varepsilon^{-1}\log\log n)^d$. Nonetheless, the exponential dependence on d make it unsuitable for higher dimensions.

For higher dimensions, a quad-tree based greedy algorithm provides an expected $O(d\log n)$ approximation in $O(nd)$ time. This algorithm constructs a randomly-shifted recursive partition of space by splitting each cell into 2^d cells with half the side-length. For every cell of the quad-tree, the algorithm moves excess supply present inside its child to meet any demand present inside another child. Agarwal & Varadarajan (2004) combined a different hierarchical partition with an exact solver to get an expected $O(d^2\log 1/\varepsilon)$-approximation in $\tilde{O}(n^{1+\varepsilon})$ time. Indyk (2007) combined the hierarchical greedy framework with importance sampling to estimate the cost of the Euclidean bipartite matching within a constant factor of the optimal. Additionally, there are $\tilde{O}(nd)$ time algorithms that approximate the matching cost with a factor of $O(\log^2 n)$ (Andoni et al. (2008)).

There are also exact and approximation algorithms that run in $\tilde{O}(n^2)$ time; however, these algorithms rely on several black-box reductions and at present, there are no usable implementations of these algorithms (Agarwal & Sharathkumar (2014); Sherman (2017)). The lack of fast exact and relative approximations that are also implementable have motivated machine learning researchers to design additive approximation algorithms which we discuss next.

Additive Approximations: Cuturi (2013) introduced a regularized version of the optimal transport problem that produces an ε-close transport plan, which can be solved using the Sinkhorn method. For input points within the unit hypercube, such an algorithm produces an ε-close transport plan in $\tilde{O}(n^2d/\varepsilon^2)$[2] time (Lin et al. (2019)). One can also adapt graph theoretic approaches including the algorithm by Gabow & Tarjan (1989) to obtain an ε-close solution in $O(n^2\sqrt{d}/\varepsilon + nd/\varepsilon^2)$ time for points within the unit hypercube (Lahn et al. (2019)). Some of the additive approximation methods, including the Sinkhorn method, are also highly parallelizable. For instance, the algorithm by Jambulapati et al. (2019) has a parallel depth of $\tilde{O}(1/\varepsilon)$; see also Altschuler et al. (2017; 2019); Blanchet et al. (2018); Quanrud (2018); Dvurechensky et al. (2018); Guo et al. (2020)

1.3 Our Results

Let $T(n,\varepsilon)$ be the time taken by an ε-additive approximation algorithm on an input of n points in the unit hypercube. In Theorem 1.1 and Theorem 1.2, we present new algorithms that improve the accuracy of any additive approximation algorithm for the 1-Wasserstein problem and the EBM problem, respectively.

Theorem 1.1. *There exists a randomized algorithm that, given two discrete distributions μ and ν in the d-dimensional unit hypercube with a spread of $\Delta = n^{O(1)}$ and a parameter $\varepsilon > 0$, computes a transport plan with an expected additive error of $\min\{\varepsilon, (d\log_{\sqrt{d}/\varepsilon} n)\mathcal{W}(\mu,\nu)\}$ in $O(T(n,\varepsilon/d)\log_{\sqrt{d}/\varepsilon} n)$ time; here, $\mathcal{W}(\mu,\nu)$ is the 1-Wasserstein distance between μ and ν.*

Theorem 1.2. *There exists an algorithm that, given two sets of samples A and B each from distributions μ and ν in the d-dimensional unit hypercube , where $|A| = |B| = n$, and a parameter $\varepsilon > 0$, computes, with high probability, a matching that is within an expected additive error of $\min\{\varepsilon, (d\log\log n)w^*\}$ from the optimal matching cost w^* in $O(T(n,\varepsilon/d)\log\log n)$ time.*

Typical additive approximation algorithms run in $T(n,\varepsilon) = \tilde{O}(n^2)$ time and compute an ε-close transport plan for any arbitrary cost function; i.e, they make no assumption about the distance between points. The inputs to our algorithm, on the other hand, are point sets in the Euclidean space. Therefore, one can use an approximate dynamic nearest neighbor data structure to improve the execution time of such additive approximation algorithms. In particular, the algorithm by Lahn et al. (2019) executes in $O(1/\varepsilon)$ phases, where in each phase, they execute one iteration of Gabow-Tarjan's algorithm. As shown by Agarwal & Sharathkumar (2014), one can use a $\frac{1}{\sqrt{\varepsilon}}$-approximate dynamic nearest neighbor data structure with a query/update time of $O(n^\varepsilon + d\log n)$ (Andoni et al.

[2]The execution time of Sinkhorn algorithm as well as other additive approximations depend on the diameter C of the point set. In the case of d-dimensional unit hypercube, $C = \sqrt{d}$.

(2014)) to execute each iteration of Gabow-Tarjan's algorithm in $\tilde{O}(n^{1+\varepsilon})$ time. Combining with the algorithms from Theorem 1.1 and Theorem 1.2, we obtain the following relative approximation algorithms. The details are provided in Appendix D.

Theorem 1.3. *There exists a randomized algorithm that, given two discrete distributions μ and ν in the d-dimensional unit hypercube in Euclidean space with a spread of $\Delta = n^{O(1)}$ and a parameter $\varepsilon > 0$, computes a $O(d/\varepsilon^{3/2})$-approximate transport plan in $O(d^2 n^{1+\varepsilon}/\varepsilon)$ time.*

Theorem 1.4. *There exists an algorithm that, given two sets of samples A and B each from distributions μ and ν in the d-dimensional unit hypercube in Euclidean space, where $|A| = |B| = n$, and a parameter $\varepsilon > 0$, computes, with high probability, an $O(\frac{d}{\sqrt{\varepsilon}} \log \frac{d}{\varepsilon})$-approximate matching in $O(dn^{1+\varepsilon} \log \frac{d}{\varepsilon})$ time.*

In contrast to our results, for the EBM problem, Agarwal & Varadarajan (2004) compute an $O(d^2 \log \frac{1}{\varepsilon})$-approximate matching in the same running time. Therefore, our algorithm computes a more accurate EBM when $d > \frac{1}{\sqrt{\varepsilon}}$. For instance, consider the case where $d = \sqrt{\log n}$. For any arbitrarily small constant $\varepsilon > 0$, the algorithm of Theorem 1.4 will run in $\tilde{O}(n^{1+\varepsilon})$ time and return an $O(\sqrt{\log n})$-approximation. In contrast, all previous methods that achieve sub-logarithmic approximation require $\Omega(n^{5/4})$ time (Agarwal & Sharathkumar (2014)).

We also note that all of our algorithms extend to any ℓ_p norm in a straight-forward way[3]. For simplicity, we restrict our presentation only to the Euclidean case.

Overview of the Algorithm: Our algorithm uses the hierarchical greedy paradigm. In our presentation, we refer to a hypercube as a *cell*. For any cell $\square$, let $V_\square$ be the set of all points of $A \cup B$ that lies inside $\square$. Unlike in the quadtree greedy algorithm, instead of splitting each cell $\square$ into 2^d cells, we split it into $\min\{|V_\square|, (4\sqrt{d}/\varepsilon)^d\}$ cells. Thus, the height of the resulting tree T reduces from $O(\log n)$ to $O(\log_{\sqrt{d}/\varepsilon} n)$. For any cell $\square$ of T and any child $\square'$ of $\square$, we move any excess supply or demand inside $\square'$ to its center. Let $\mathcal{A}_\square$ (resp. $\mathcal{B}_\square$) be a set consisting of the center points of all children of $\square$ with excess demand (resp. supply). For any child $\square'$ of $\square$ with excess demand (resp. supply), we assign a weight of $\eta(V_{\square'})$ to its center point in $\mathcal{A}_\square$ (resp. $\mathcal{B}_\square$). Using an additive approximation algorithm, we compute an (ε/d)-close transport cost between $\mathcal{A}_\square$ and $\mathcal{B}_\square$ in $T(|V_\square|, \varepsilon/d)$ time. We report the sum of the transport costs computed at all cells of T as an approximate 1-Wasserstein distance. This simple algorithm guarantees improvement in the quality of the solutions produced by both additive and relative approximation algorithms.

From the stand-point of relative approximation algorithms, Agarwal & Varadarajan (2004) as well as Indyk (2007) have utilized a very similar hierarchical framework to design approximation algorithms. However, unlike our algorithm, they used an exact solver that takes $\Omega(|V_\square|^3)$ time. As a result, to obtain near-quadratic execution time, they needed every cell to be divided into at most $|V_\square|^{2/3}$ children, i.e., $1/\varepsilon^d \leq n^{2/3}$ or $d = O(\log_{1/\varepsilon} n)$. This also forces the height of the tree to be $O(d \log_{1/\varepsilon} n)$ leading to an $O(d^2 \log_{1/\varepsilon} n)$-factor approximation. We replace the $\Omega(n^3)$ time exact solver in their algorithm with a $T(|V_\square|, \varepsilon) = \tilde{O}(|V_\square|^2)$ time additive approximation algorithm. Therefore, each instance (regardless of the number of non-empty children) can be solved in $\tilde{O}(|V_\square|^2)$ time. As a result, we are able to improve the approximation factor from $O(d^2 \log_{1/\varepsilon} n)$ to $O(d \log_{\sqrt{d}/\varepsilon} n)$ and also remove restrictions on the dimension. Our algorithm now works for any dimension!

Technical Challenge: By using an additive approximation algorithm to solve each instance, we have an increased error that may be difficult to bound. We use the following observation to overcome this challenge. In Section 2.2, we show that for any point set with spread Δ, an additive approximation algorithm can be used to compute a 2-relative approximation in $T(n, 1/\Delta)$ time. The algorithm guarantees that the spread of the point set at each cell $\square$ is $O(d/\varepsilon)$, and as a result, we get a 2-relative approximation by using an additive approximation algorithm in $O(T(n, \varepsilon/d))$ time.

Improvements for EBM: For the EBM problem, as mentioned in Theorem 1.2, we obtain an improvement in the approximation ratio as follows: Instead of dividing each cell into a fixed number

[3]Owing to a higher complexity of LSH in arbitrary ℓ_p norms, the approximation factor in Theorem 1.3 and Theorem 1.4 is slightly higher for arbitrary ℓ_p norms.

$\min\{n, (4\sqrt{d}/\varepsilon)^d\}$ of children, we divide them into $\min\{n, n^{d/2^i}\}$ children at each level i; here, level of any cell in the tree is equal to the length of the path from the root to this cell. By doing so, we reduce the height of the tree to $O(\log\log n)$. To analyse the running time, we show that the number of remaining unmatched points over all level i cells is $\tilde{O}(n^{1-\frac{1}{2^i}})$. Since there are only sub-linearly many points remaining, we can afford a larger spread of $O(\frac{d}{\varepsilon} n^{\frac{1}{2^i}})$ and the resulting execution time of the additive approximation will continue to be $O(T(n, \frac{\varepsilon}{d}))$ per level.

Limitations: Although our algorithms work in any dimension, we would like to note that the relative approximation factor grows linearly in the dimension. Therefore, the improvement achieved by our algorithm degrades as dimension increases.

2 PRELIMINARIES

We begin by introducing notations that are used in presenting our algorithm. For any cell $\square$, we denote the side-length of $\square$ by $\ell_\square$ and the *center* of $\square$ by $c_\square$. For a parameter ℓ, let $\mathbb{G}(\square, \ell)$ denote a grid that partitions $\square$ into smaller cells with side-length ℓ. Recall that $V_\square$ denotes the subset of $A \cup B$ that lies inside $\square$. We say that $\square$ is *non-empty* if $V_\square$ is non-empty; i.e, $\square$ contains at least one point of $A \cup B$. We define the weight of $\square$ to be $\eta(V_\square)$ and denote it by $\eta(\square)$. For each cell $\square$, we call it a *deficit cell* if $\eta(\square) < 0$, a *surplus cell* if $\eta(\square) > 0$, and a *neutral cell* if $\eta(\square) = 0$.

In this section, we provide a simple transformation of the input for achieving an additive approximation of the 1-Wasserstein distance. Furthermore, for point sets with a spread of Δ, we show how an additive approximation algorithm can be used to obtain a $(1+\varepsilon)$-relative approximation of the 1-Wasserstein problem in $T(n, \varepsilon/\Delta)$ time.

2.1 ADDITIVE APPROXIMATION IN EUCLIDEAN SPACE

For any $\varepsilon > 0$, given an additive approximation algorithm that runs in $T(n, \varepsilon)$ time, we present an algorithm to compute an ε-close transport cost for distributions inside the d-dimensional unit hypercube $\square^*$ in $O(\min\{T(n, \frac{\varepsilon}{2}), n + T((\frac{2\sqrt{d}}{\varepsilon})^d, \frac{\varepsilon}{2})\})$ time.

The algorithm executes in two steps. In the first step, the algorithm builds a grid $\mathbb{G}(\square^*, \frac{\varepsilon}{2\sqrt{d}})$ on the unit hypercube $\square^*$ and constructs a transport plan σ_1 as follows. For each non-empty neutral cell of the grid, σ_1 arbitrarily transports all supplies to demands within the cell. Similarly, for any deficit (resp. surplus) cell, σ_1 arbitrarily transports supplies from (resp. to) all supply (resp. demand) points inside the cell to (resp. from) some arbitrary demand (resp. supply) points within the cell.

In the second step, the algorithm constructs a set of demand points $\mathcal{A}$ and supply points $\mathcal{B}$ as follows: For any deficit cell $\square$ (resp. surplus cell $\square'$), the point $c_\square$ (resp. $c_{\square'}$) is added to $\mathcal{A}$ (resp. $\mathcal{B}$) with a weight of $\eta(\square)$ (resp. $\eta(\square')$). Note that $\mathcal{A} \cup \mathcal{B}$ is a balanced instance for the 1-Wasserstein problem. The algorithm computes an $\frac{\varepsilon}{2}$-close transport plan σ_2 on the instance $\mathcal{A} \cup \mathcal{B}$ in $T(|\mathcal{A}| + |\mathcal{B}|, \varepsilon/2)$ time (See Figure 1). The algorithm returns $w(\sigma_1) + w(\sigma_2)$ as an ε-close transport cost on $A \cup B$.

We provide a discussion on the accuracy of the algorithm in Appendix A. The following lemma follows from the fact that $|\mathcal{A}| + |\mathcal{B}|$ is bounded by $\min\{2n, (2\sqrt{d}/\varepsilon)^d\}$.

Lemma 2.1. *There exists an algorithm that, given two point sets A and B in the d-dimensional unit hypercube and a value $\varepsilon > 0$, computes an ε-close transport cost in $O(\min\{T(n, \frac{\varepsilon}{2}), n + T((\frac{2\sqrt{d}}{\varepsilon})^d, \frac{\varepsilon}{2})\})$ time.*

Instead of transporting supplies inside cells arbitrarily, our algorithm in Section 3 recursively applies the same algorithm in each cell and obtains a higher accuracy.

2.2 RELATIVE APPROXIMATION FOR LOW SPREAD POINT SETS

In this section, we show that an ε-additive approximation algorithm can be used to obtain a $(1+\varepsilon)$-relative approximation algorithm for the 1-Wasserstein problem in $T(n, \varepsilon/\Delta)$ time; here, Δ is the spread of the points in $A \cup B$. Since relative approximations are scale invariant, without loss of generality, we assume that the input has $C_{\max} = 1$ and $C_{\min} = 1/\Delta$.

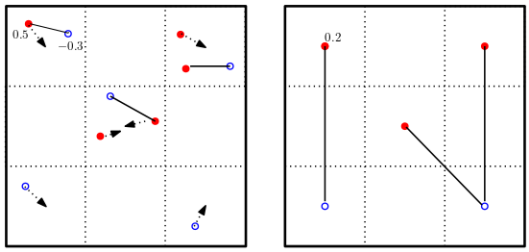


Figure 1: (a) The algorithm transports supplies (red disks) to demands (blue circles) within each cell and create an instance by moving any excess supplies or demands to the center of the corresponding cells, (b) An $\varepsilon/2$-close transport plan is computed on the new problem instance.

Suppose $-\eta(A) = \eta(B) = U$. The minimum distance between any two points $(a, b) \in A \times B$ is $1/\Delta$. Therefore, the cost of transporting a supply of U is at least U/Δ. We obtain a $(1+\varepsilon)$-relative approximate transport plan by simply executing an additive approximation algorithm with an additive error of $U(\varepsilon/\Delta)$ in time $T(n, \varepsilon/\Delta)$.

Lemma 2.2. *There exists an algorithm that, given two point sets A and B in the d-dimensional unit hypercube with a spread of Δ, computes a $(1+\varepsilon)$-approximate transport plan in $T(n, \varepsilon/\Delta)$ time.*

3 An $O(d \log_{\sqrt{d}/\varepsilon} n)$-approximation algorithm for 1-Wasserstein problem

In this section, we present an algorithm satisfying the bounds claimed in Theorem 1.1. We begin by defining a hierarchical partitioning and a tree T associated with it. Each node in the tree corresponds to a non-empty cell in our hierarchical partition and we do not distinguish between the two. Our tree partitions each cell of side-length ℓ into $\lceil 4\sqrt{d}/\varepsilon \rceil^d$ many cells of side-length at most $(\varepsilon/4\sqrt{d})\ell$. We construct our hierarchical partition in a randomly-shifted fashion as follows.

Hierarchical Partitioning: First, we pick a point ξ uniformly at random from the unit hypercube $[0,1]^d$ and set $\square^* = [-1,1]^d + \xi$. Note that $\square^*$ is a hypercube of side-length 2 containing all points in $A \cup B$. We designate $\square^*$ as the root of T. Let $\kappa = \lceil 4\sqrt{d}/\varepsilon \rceil$. For any cell $\square$, if only one point of $A \cup B$ lies inside $\square$, we designate $\square$ as a leaf cell in T. For any non-leaf cell $\square$, we construct its children by partitioning $\square$ using a grid $\mathbb{G}_\square = \mathbb{G}(\square, \ell_\square/\kappa)$ into $\lceil 4\sqrt{d}/\varepsilon \rceil^d$ cells and create a child node for each non-empty cell of this grid. For any non-leaf cell $\square$ of T, we denote the set of children of $\square$ by $\mathsf{C}[\square]$. Assuming the spread $\Delta = n^{O(1)}$, the height of T, h, is $O(\log_{\sqrt{d}/\varepsilon} n)$.

Similar to quadtrees, our hierarchical partitioning can be seen as a sequence of grids $\langle \mathbb{G}_0, \mathbb{G}_1, \ldots, \mathbb{G}_h \rangle$, where $\mathbb{G}_0$ is the root and each grid $\mathbb{G}_i$ refines the cells of grid $\mathbb{G}_{i-1}$. For each grid $\mathbb{G}_i$, we denote the cell-side-length of $\mathbb{G}_i$ by ℓ_i. Next, we describe our algorithm.

Computing an approximate 1-Wasserstein distance: In this section, we present our algorithm for computing an approximate cost of an optimal transport plan on $A \cup B$.

1 - Creating an instance at each cell: For each cell $\square$ of T, we create an instance $\mathcal{I}_\square$ of the 1-Wasserstein problem as follows. If $\square$ is a deficit or surplus cell, we add the point $c_\square$ with a weight $-\eta(\square)$ to $\mathcal{I}_\square$. Additionally, if $\square$ is a leaf cell, then it contains a single point u of $A \cup B$, which we add to $\mathcal{I}_\square$. Otherwise, $\square$ is a non-leaf cell and for any child $\square' \in \mathsf{C}[\square]$, if $\square'$ is a deficit or surplus cell, we add $c_{\square'}$ with a weight $\eta(\square')$ to $\mathcal{I}_\square$, which represents the excess demands or supplies of the points in $V_{\square'}$. The instance $\mathcal{I}_\square$ is balanced and has a spread of $O(d/\varepsilon)$.

2 - Estimating the 1-Wasserstein distance: In this step, for each cell $\square$, using the algorithm from Lemma 2.2, we compute a 2-approximate transport plan $\sigma_\square$ on $\mathcal{I}_\square$. Our algorithm then reports the total cost of the transport plans computed at all cells of T, i.e, $w := \sum_{\square \in T} w(\sigma_\square)$, as an approximate 1-Wasserstein distance.

Retrieving an approximate transport plan: We retrieve an approximate transport plan σ on the point set $A \cup B$ by processing grids $\langle \mathbb{G}_h, \dots, \mathbb{G}_0 \rangle$ in decreasing order of their level. First, for each non-empty cell $\square$ of $\mathbb{G}_h$, $\square$ is a leaf cell and $V_\square$ contains only one point. We map the only point in $V_\square$ to $c_\square$. For some $i < h$, assume (inductively) that after processing the non-empty cells of the grid $\mathbb{G}_{i+1}$, the following conditions (i)–(iii) hold for the current transport plan σ within any cell $\square$ in $\mathbb{G}_{i+1}$: (i) if $\square$ is a neutral cell, then σ transports every supply to some demand point inside $\square$, (ii) if $\square$ is a deficit (resp. surplus) cell, then σ transports all supplies (resp. demands) inside $\square$ to (resp. from) some demand (resp. supply) point within $\square$, and, (iii) if $\square$ is a deficit (resp. surplus) cell, the excess demand (resp. supply) is mapped to $c_\square$. Given this, we show how to process any non-empty cell $\square$ of $\mathbb{G}_i$ so that (i)–(iii) holds for $\square$.

Recollect that $\sigma_\square$ is a transport plan computed by our algorithm on $\mathcal{I}_\square$. By condition (iii), the excess supplies or demands at any child $\square'$ of $\square$ is mapped to $c_{\square'}$. Therefore, for any pair of children $\square_1, \square_2 \in \mathsf{C}[\square]$, where $\square_1$ is a surplus cell and $\square_2$ is a deficit cell, the transport plan σ transports $\sigma_\square(c_{\square_1}, c_{\square_2})$ supplies from $c_{\square_1}$ to $c_{\square_2}$. In addition, for any child $\square_1$ (resp. $\square_2$) of $\square$, suppose $\square_1$ (resp. $\square_2$) is a surplus (resp. deficit) cell. If $\sigma_\square(c_{\square_1}, c_\square) > 0$ (resp. $\sigma_\square(c_{\square_2}, c_\square) > 0$), then we map the supplies (resp. demands) from $c_{\square_1}$ (resp. $c_{\square_2}$) to $c_\square$. It is easy to confirm that after processing $\square$, (i)–(iii) holds for $\square$. From triangle inequality, $w(\sigma)$ is upper-bounded by the total cost of the transport plans computed for each cell of T; i.e, $w(\sigma) \leq \sum_{\square \in T} w(\sigma_\square)$.

Efficiency: For any i, let C_i denote the set of non-empty cells of T at level i. For each cell $\square \in \mathsf{C}_i$, let $n_\square$ be the number of points in $\mathcal{I}_\square$. Since the spread of the points in $\mathcal{I}_\square$ is $O(d/\varepsilon)$, executing the algorithm from Lemma 2.2 on $\mathcal{I}_\square$ takes $O(T(n_\square, \varepsilon/d))$ time. Since $\mathcal{I}_\square$ contains at most one point for each non-empty child of $\square$, $n_\square \leq \min\{|V_\square|, (4\sqrt{d}/\varepsilon)^d\}$. Therefore, $\sum_{\square \in \mathsf{C}_i} n_\square \leq \sum_{\square \in \mathsf{C}_i} |V_\square| = n$. Since $T(n, \varepsilon) = \Omega(n)$, the running time of our algorithm on cells at level i is $O(\sum_{\square \in \mathsf{C}_i} T(n_\square, \varepsilon/d)) = O(T(n, \varepsilon/d))$. Summing over all levels, the running time of our algorithm is $O(T(n, \varepsilon/d) \log_{\sqrt{d}/\varepsilon} n)$.

When the dimension is a small constant, we get an improved running time as follows. For each level i of T, there are at most n non-empty cells at level i and the instance created at each cell has a size of at most $(4\sqrt{d}/\varepsilon)^d$. Therefore, $O(\sum_{\square \in \mathsf{C}_i} T(n_\square, \varepsilon/d)) = O(nT((4\sqrt{d}/\varepsilon)^d, \varepsilon/d)) = n(\frac{d}{\varepsilon})^{O(d)}$ and the overall running time will be improved to $n(\frac{d}{\varepsilon})^{O(d)} \log_{\sqrt{d}/\varepsilon} n$.

Below, assuming $\varepsilon \leq 1/2$, we show that the cost w computed by the algorithm is an $O(d \log_{\sqrt{d}/\varepsilon} n)$-approximation of the cost of the optimal transport on $A \cup B$.

Quality of Approximation: We analyze the approximate 1-Wasserstein distance computed by our algorithm as follows. After scaling the points to have a unit diameter, the algorithm of Lemma 2.2 obtains a 2-approximate transport plan by computing a $(1/\Delta)$-close transport plan. When scaled back to original, for the root cell $\square^*$ of T, $\sigma_{\square^*}$ is an $(\varepsilon/4\sqrt{d})$-close transport plan on $\square^*$. The remaining demands and supplies are recursively transported within the children. Since the diameter of each child of $\square^*$ is $\varepsilon/2$, similar to Section 2.1, we can argue that our algorithm reports an ε-close 1-Wasserstein distance. Next, we show that the reported cost is an $O(d \log_{\sqrt{d}/\varepsilon} n)$-approximation of the 1-Wasserstein distance.

Recall that for each level i, C_i denotes the set of non-empty cells of T at level i. We show that the expected distortion of mapping all points to the centers of the leaf cells is $O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*)$, where the expectation is over the choice of the random shift of the hierarchical partitioning. Additionally, for each $i < h$, we show that $\mathbb{E}[\sum_{\square \in \mathsf{C}_i} w(\sigma_\square)] = O(d) w(\sigma^*)$; here, σ^* is an optimal transport plan on $A \cup B$. The result then follows since T has $O(\log_{\sqrt{d}/\varepsilon} n)$ levels.

For any level $i < h$, we bound $\mathbb{E}[\sum_{\square \in \mathsf{C}_i} w(\sigma_\square)]$ in two steps as follows. In the first step, we assign a budget to every edge (a, b) with $\sigma^*(a, b) > 0$ and show that the total budget assigned to all such edges is (in expectation) at most $O(d) w(\sigma^*)$. In the second step, we redistribute this budget to the cells of level i in a way that the budget received by any cell $\square$ is at least $w(\sigma_\square)/2$. Details of these steps are provided in Appendix B. Theorem 1.1 follows from combining the two steps.

4 AN $O(d \log\log n)$-APPROXIMATION ALGORITHM FOR EBM PROBLEM

Given two sets of samples A and B from distributions μ and ν, respectively, where $|A| = |B| = n$ and each point $a \in A$ (resp. $b \in B$) has a weight $\eta(a) = -1/n$ (resp. $\eta(b) = 1/n$), we present an approximate algorithm for the EBM problem satisfying the bounds claimed in Theorem 1.2. For simplicity in presentation, we scale all the $\eta(\cdot)$ values from $-1/n$ (resp. $1/n$) to -1 (resp. 1) and show that our algorithm computes a matching that is within an additive error of $n \min\{\varepsilon, (d \log\log n)(w(M^*)/n)\}$ from the optimal matching cost $w(M^*)$.

Input Transformation: We transform input points such that (1) all coordinates are positive integers bounded by $n^{O(1)}$, (2) an optimal matching on the transformed points is a $(1+\varepsilon)$-approximate matching with respect to the original points, and (3) the cost of the optimal matching is $O(d^{3/2} n \log n/\varepsilon)$. Similar transformations have been applied in several papers in the literature (Agarwal et al. (2017); Lahn & Raghvendra (2021)). We describe this transformation in Appendix C.1. As before, we can match and remove any co-located points $a \in A$ and $b \in B$.

Next, assuming $T(n, \varepsilon) = O(\frac{n^k}{\varepsilon}\mathrm{poly}(d, \log n, \log \frac{1}{\varepsilon}))$ for some $k \geq 1$, we describe our EBM algorithm. Our algorithm is easily adaptable to any additive approximate algorithm with a running time of $T(n, \varepsilon) = O(n^k \varepsilon^{-t}\mathrm{poly}(d, \log n, \log \frac{1}{\varepsilon}))$, where $k \geq 1$ and t is a fixed constant.

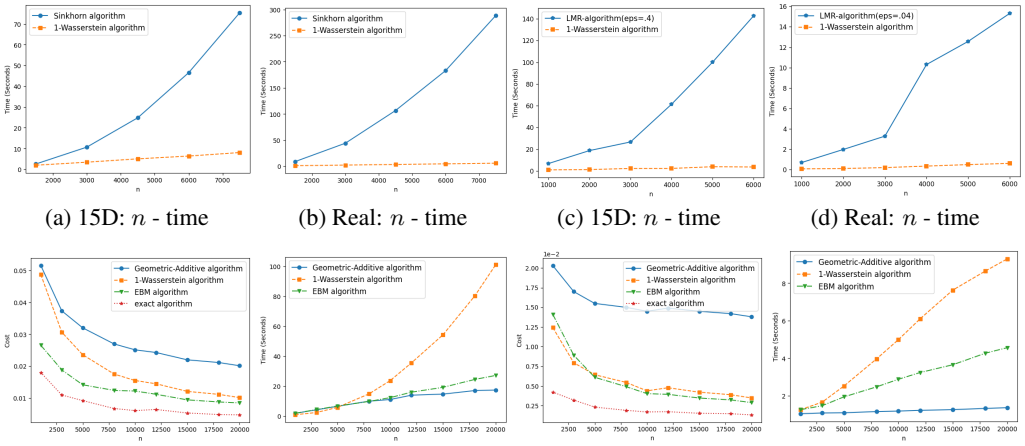
Overview of the algorithm: Similar to Section 3, our EBM algorithm constructs a hierarchical partitioning and the associated tree T, and executes the algorithm from Lemma 2.2 on the instance created for each cell of T. In contrast to Section 3, the tree T constructed by our EBM algorithm has a lower height of $O(\log\log n)$, resulting in an improved approximation factor. The hierarchical partitioning of this section differs from the one in Section 3 in two ways. First, we partition the root cell into a grid $\mathbb{G}_1$ with cell-side-length of $\Theta(d^{5/2} n \log n/\varepsilon^2)$ as the first level. The grid $\mathbb{G}_1$ may result in a high branching factor at the root; however, we show that, with probability at least $1 - \varepsilon/\sqrt{d}$, no edges of an optimal matching will cross $\mathbb{G}_1$. Therefore, with that probability, all cells of $\mathbb{G}_1$ are neutral cells and the problem instance for the root is an empty instance; i.e, the branching factor of the root will not impact the running time of our algorithm. Second, for any cell $\square$ of level i, instead of splitting $\square$ into a fixed number $\min\{n, (4\sqrt{d}/\varepsilon)^d\}$ of children, we divide $\square$ into $\min\{n, n^{d/2^i}\}$ children. Although this results in a higher spread of $\tilde{O}(n^{1/2^i})$ for the problem instance $\mathcal{I}_\square$, we show that the expected number of remaining unmatched points over all cells of level i is $\tilde{O}(n^{1-1/2^i})$. Therefore, the total execution time of our algorithm remains $T(n, \varepsilon/d)$ per level. These modifications result in a tree T of height $O(\log\log n)$. We describe the details below.

Hierarchical Partitioning: Define $\delta := 5d^2\varepsilon^{-1}\log n$. Similar to Section 3, we define a cell $\square^*$ as a randomly-shifted hypercube that contains all points of $A \cup B$ and has a side-length of $2\max\{C_{\max}, \ell_1/\varepsilon\}$, where $\ell_1 = \frac{\sqrt{d}}{\varepsilon}\delta n$. We designate $\square^*$ as the root of T ($\square^*$ is at level 0 of T). Define a grid $\mathbb{G}_1 := \mathbb{G}(\square^*, \ell_1)$. We add each non-empty cell of $\mathbb{G}_1$ to the tree as the children of $\square^*$. We construct the hierarchical partitioning in a recursive fashion as follows. For any non-root cell $\square$ of T, if $\square$ contains only one point of $A \cup B$, then we designate $\square$ as a leaf cell. Otherwise, let $\square$ be a cell of level i. Define the grid $\mathbb{G}_\square = \mathbb{G}(\square, \delta n^{1/2^i})$ and add the non-empty cells of $\mathbb{G}_\square$ to T as the children of $\square$. For simplicity in presentation, we assume $n^{1/2^i}$ is an integer. For any cell $\square$, denote the set of children of $\square$ in T by $\mathsf{C}[\square]$. The height of T, denoted by h, is $O(\log\log n)$.

Similar to Section 3, our hierarchical partitioning is also a sequence of grids $\langle \mathbb{G}_0, \mathbb{G}_1, \ldots, \mathbb{G}_h \rangle$, where $\mathbb{G}_0$ is the root cell $\square^*$, $\mathbb{G}_1$ has a cell-side-length of $\ell_1 = \frac{\sqrt{d}}{\varepsilon}\delta n$, and for each $2 \leq i \leq h$, the cell-side-length of $\mathbb{G}_i$ is $\ell_i = \delta n^{1/2^{i-1}}$.

Computing an approximate matching cost: To estimate the matching cost, similar to Section 3, our algorithm creates an instance of the 1-Wasserstein problem for each cell of the tree T. Using the algorithm from Lemma 2.2, our algorithm computes a 2-approximate transport plan for the instance created for each cell and returns the total cost of such transport plans as an approximate matching cost. This completes the description of the algorithm.

We describe the details of retrieving a matching in Appendix C.2, the quality of approximation in Appendix C.3, and the efficiency of our algorithm in Appendix C.4.



(a) 15D: n - time (b) Real: n - time (c) 15D: n - time (d) Real: n - time

(e) 15D: n - cost (f) 15D: n - time (g) Real: n - cost (h) Real: n - time

Figure 2: (a) and (b) Comparison with the Sinkhorn algorithm, (c) and (d) Comparison with the LMR algorithm, and (e)–(h) Comparison with the Geometric-Additive algorithm

5 EXPERIMENTS

In this section, we conduct experiments to show that our algorithms from Section 3 and Section 4 improve the accuracy of the additive approximation algorithms. We test an implementation of our algorithm, written in Python, on discrete probability distributions derived from real-world and synthetic data sets. All tests are executed on a computer with a 2.50 GHz Intel Core i7 processor and 8GB of RAM using a single computation thread.

Datasets: We test our algorithms on two sets of n samples taken from synthetic (distribution) and real-world data sets. For each dataset, we use our algorithms to compute the minimum-cost matching between these samples and present our results averaged over 10 executions. To generate the synthetic data, we sample from a uniform distribution from inside a 2-dimensional unit square placed on a random plane in 15-dimensional space (15D). For a real-world dataset, we use the *Adult Census Data* (UCI repository), which is a point cloud in $\mathcal{R}^6$ with continuous features for $35,000$ individuals, divided into two categories by income (Dua & Graff (2017)). See Appendix E for the results of our experiments on additional datasets.

Results: In our first experiment, we compare our algorithm with existing additive approximation schemes, namely the Sinkhorn method (Cuturi (2013)) (resp. LMR algorithm (Lahn et al. (2019))). We use the Sinkhorn (resp. LMR) algorithm as a black-box within our algorithm from Section 3 (1-Wasserstein algorithm) and compare its execution time with the standard Sinkhorn (resp. LMR) implementation. We set the parameters of the Sinkhorn and LMR algorithms in such a way that the error produced by them matches with the error produced by our 1-Wasserstein algorithm. As shown in Figure 2 (a)–(d), our algorithm runs significantly faster than both the Sinkhorn and the LMR algorithms while producing solutions of similar quality. In our second experiment, we also compare the accuracy as well as the computation time of our algorithms (1-Wasserstein algorithm and our EBM algorithm from Section 4) with the additive approximation algorithm from Section 2.1 (Geometric-Additive algorithm). For this experiment, our algorithms use the Sinkhorn algorithm as a black-box. The results of this experiment are shown in Figure 2 (e)–(h). We observe that our algorithms achieves a better accuracy (see Figure 2 (a) and (c)), especially on real-world data sets. As expected, the execution time of our algorithms increase slightly (see Figure 2 (d)). Furthermore, we observe that as the sample size increases, the costs returned by our algorithms converge to the optimal cost, whereas the Geometric-Additive approach does not.

Finally, we highlight the scalability of our algorithms. For an input of 3 million points drawn from a 2 dimensional uniform distribution on the unit square, our 1-Wasserstein algorithm runs in 593 seconds and computes an approximate transport cost of 0.0093. Furthermore, for an input of 1.5 million points for the 15D dataset, our 1-Wasserstein algorithm computes an approximate cost of 0.0304 in 608 seconds.

REFERENCES

Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. *In Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pp. 555–564, 2014.

Pankaj K. Agarwal and Kasturi R. Varadarajan. A near-linear constant-factor approximation for Euclidean bipartite matching? *In Proceedings of the 20th annual symposium on Computational geometry*, pp. 247–252, 2004.

Pankaj K. Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi R. Varadarajan, and Allen Xiao. Faster algorithms for the geometric transportation problem. In *Proc. 33rd International Symposium on Computational Geometry*, pp. 7:1–7:16, 2017.

Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear ε-approximation algorithm for geometric bipartite matching. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1052–1065, 2022a.

Pankaj K Agarwal, Sharath Raghvendra, Pouyan Shirzadian, and Rachita Sowle. An improved ε-approximation algorithm for geometric bipartite matching. In *18th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022b.

Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. *Advances in neural information processing systems*, 30, 2017.

Jason Altschuler, Francis Bach, Alessandro Rudi, and Jonathan Niles-Weed. Massively scalable sinkhorn distances via the nyström method. *Advances in neural information processing systems*, 32, 2019.

Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, volume 8, pp. 343–352, 2008.

Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1018–1028. SIAM, 2014.

Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Scalable nearest neighbor search for optimal transport. *In International Conference on Machine Learning*, pp. 497–506, 2020.

Espen Bernton, Pierre E. Jacob, Mathieu Gerber, and Christian P. Robert. On parameter estimation with the wasserstein distance. *Information and Inference: A Journal of the IMA*, 8(4):657–676, 2019.

Jose Blanchet, Arun Jambulapati, Carson Kent, and Aaron Sidford. Towards optimal running times for optimal transport. *arXiv preprint arXiv:1810.07717*, 2018.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26*, pp. 2292–2300, 2013.

Ishan Deshpande, Ziyu Zhang, and Alexander G Schwing. Generative modeling using the sliced wasserstein distance. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3483–3491, 2018.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Pavel Dvurechensky, Alexander Gasnikov, and Alexey Kroshnin. Computational optimal transport: Complexity by accelerated gradient descent is better than by sinkhorn's algorithm. In *International conference on machine learning*, pp. 1367–1376. PMLR, 2018.

Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1):115–166, 2018.

Kyle Fox and Jiashuai Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. In *Proc. 36th Annual Symposium on Computational Geometry*, pp. 45:1–45:18, 2020.

H. N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18: 1013–1036, October 1989. ISSN 0097-5397. doi: 10.1137/0218069. URL `http://portal.acm.org/citation.cfm?id=75795.75806`.

Aude Genevay, Gabriel Peyre, and Marco Cuturi. Learning generative models with sinkhorn divergences. *In International Conference on Artificial Intelligence and Statistics*, pp. 1608–1617, 2018.

Wenshuo Guo, Nhat Ho, and Michael Jordan. Fast algorithms for computational optimal transport and wasserstein barycenter. *In International Conference on Artificial Intelligence and Statistics*, pp. 2088–2097, 2020.

Rishi Gupta, Piotr Indyk, and Eric Price. Sparse recovery for earth mover distance. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1742–1744. IEEE, 2010.

P. Indyk. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 39–42, 2007.

Arun Jambulapati, Aaron Sidford, and Kevin Tian. A direct $\tilde{O}(1/\epsilon)$ iteration parallel algorithm for optimal transport. *arXiv preprint arXiv:1906.00618*, 2019.

Hicham Janati, Marco Cuturi, and Alexandre Gramfort. Wasserstein regularization for sparse multi-task regression. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1407–1416. PMLR, 2019.

Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. Preconditioning for the geometric transportation problem. *arXiv preprint arXiv:1902.08384*, 2019.

Nathaniel Lahn and Sharath Raghvendra. A faster algorithm for minimum-cost bipartite matching in minor-free graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 569–588. SIAM, 2019.

Nathaniel Lahn and Sharath Raghvendra. An $O(n^{5/4})$ time ε-approximation algorithm for RMS matching in a plane. *In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms*, pp. 869–888, 2021.

Nathaniel Lahn, Deepika Mulchandani, and Sharath Raghvendra. A graph theoretic additive approximation of optimal transport. *In Advances in Neural Information Processing Systems 32*, pp. 13813–13823, 2019.

Tianyi Lin, Nhat Ho, and Michael I. Jordan. On the efficiency of the sinkhorn and greenkhorn algorithms and their acceleration for optimal transport. *arXiv preprint arXiv:1906.01437*, 2019.

Huidong Liu, G. U. Xianfeng, and Dimitris Samaras. A two-step computation of the exact gan wasserstein distance. *In International Conference on Machine Learning*, pp. 3159–3168, 2018.

Giulia Luise, Alessandro Rudi, Massimiliano Pontil, and Carlo Ciliberto. Differential properties of sinkhorn approximation for learning with wasserstein distance. *Advances in Neural Information Processing Systems*, 31, 2018.

James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the Twentieth annual ACM symposium on Theory of Computing*, pp. 377–387, 1988.

Kent Quanrud. Approximating optimal transport with linear programs. *arXiv preprint arXiv:1810.05957*, 2018.

Sharath Raghvendra and Pankaj K. Agarwal. A near-linear time ε-approximation algorithm for geometric bipartite matching. *J. ACM*, 67(3):18:1–18:19, 2020. doi: 10.1145/3393694. URL https://doi.org/10.1145/3393694.

Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. Improving gans using optimal transport. *In International Conference on Learning Representations*, 2018.

Jonah Sherman. Generalized preconditioning and undirected minimum-cost flow. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 772–780, 2017.

A MISSING DETAILS OF SECTION 2.1

In this section, we analyze the additive error of the 1-Wasserstein distance computed in Section 2.1. First, we define some notations that are used in the analysis.

For a transport plan σ, the *residual graph* is a graph $\mathcal{G}(A, B)$ on the point set $A \cup B$ such that for any pair of points $(a, b) \in A \times B$, (1) if $\sigma(a, b) > 0$, there is a *backward edge* from a to b of capacity $\sigma(a, b)$ and (2) if $\sigma(a, b) < \min\{-\eta(a), \eta(b)\}$, then there is a *forward edge* from b to a with a capacity $\min\{-\eta(a), \eta(b)\} - \sigma(a, b)$. An *alternating path* is any simple path P in $\mathcal{G}(A, B)$; i.e, a path P where the edges alternates between forward edges and backward edges. Any demand point $a \in A$ (resp. supply point $b \in B$) is called a *free point* with respect to σ if the total supplies transported into a (resp. from b) in σ is less than $-\eta(a)$ (resp. $\eta(b)$). An *augmenting path* P with respect to σ is simply an alternating path from a free supply point to a free demand point. Let $\beta(P)$ denote the amount by which P increases the total supplies transported by σ.

Recall that the algorithm of Section 2.1 computes a transport plan in two steps. In the first step, it computes a transport plan σ_1 by arbitrarily transporting supplies to demands within each non-empty cell of the grid. Suppose U' supplies are transported in this step. In the second step, for each surplus (resp. deficit) cell $\square$, the algorithm moves the excess supplies (resp. demands) of the points inside $\square$ to the center point $c_\square$ and creates the input instance $\mathcal{A} \cup \mathcal{B}$. Then, it computes an $\varepsilon/2$-close transport plan σ_2 on $\mathcal{A} \cup \mathcal{B}$, which transports $U - U'$ supplies. The algorithm reports $w(\sigma_1) + w(\sigma_2)$ as an approximate 1-Wasserstein distance.

Since the diameter of each cell of the grid is $\varepsilon/2$, each edge carrying a positive supply in σ_1 has a length of at most $\varepsilon/2$; therefore, $w(\sigma_1) \leq \varepsilon U'/2$. Next, we bound the cost of σ_2. Let σ^* be any optimal transport plan on $A \cup B$. Let $\mathcal{P}$ denote the set of augmenting paths in the symmetric difference $\sigma^* \oplus \sigma_1$ with respect to σ_1. For any $P \in \mathcal{P}$, suppose P is an augmenting path with a supply point $b_P \in B$ and a demand point $a_P \in A$ as its endpoints. Note that a_P (resp. b_P) lies inside a deficit (resp. surplus) cell of the grid, since a_P (resp. b_P) is a free point with respect to σ_1.

Consider a transport plan σ_2^* constructed as follows. For each augmenting path $P \in \mathcal{P}$, let $\square_{a_P}$ (resp. $\square_{b_P}$) denote the cell of the grid containing a_P (resp. b_P). The transport plan σ_2^* transports $\beta(P)$ supplies from $c_{\square_{b_P}}$ to $c_{\square_{a_P}}$ (see Figure A(a)). Since the augmenting paths in $\mathcal{P}$ transport all excess supplies from the surplus cells to the deficit cells, σ_2^* is a valid transport plan for $\mathcal{A} \cup \mathcal{B}$. Furthermore, since σ_2 is an $\varepsilon/2$-close transport plan, $w(\sigma_2) \leq w(\sigma_2^*) + \frac{\varepsilon}{2}(U - U')$. Next, we bound $w(\sigma_2^*)$. Since the diameter of each cell is $\varepsilon/2$,

$$w(\sigma_2^*) = \sum_{P \in \mathcal{P}} \beta(P) \|c_{\square_{a_P}} - c_{\square_{b_P}}\| \leq \sum_{P \in \mathcal{P}} \beta(P)(\|a_P - b_P\| + \frac{\varepsilon}{2}).$$

The total supplies transported by σ_2^* is $\sum_{P \in \mathcal{P}} \beta(P) = U - U'$. Furthermore, for each augmenting path P, by the triangle inequality, $\|a_P - b_P\| \leq \sum_{(u,v) \in P} \|u - v\|$. Therefore,

$$w(\sigma_2^*) \leq \frac{\varepsilon}{2}(U - U') + \sum_{P \in \mathcal{P}} \sum_{(u,v) \in P} \beta(P) \|u - v\|.$$

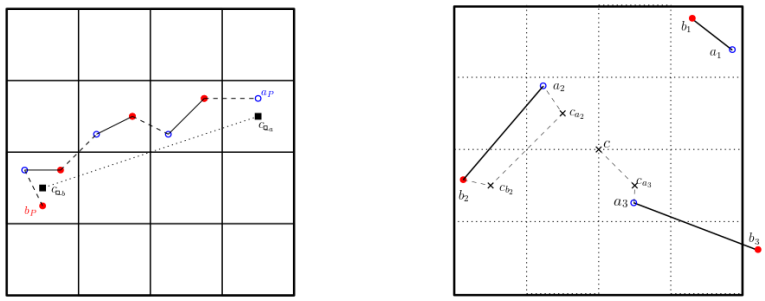
Since $\mathcal{P}$ is the set of augmenting paths in $\sigma^* \oplus \sigma_1$, the total cost of edges of the paths in $\mathcal{P}$ is at most $w(\sigma^*) + w(\sigma_1)$. Therefore,

$$w(\sigma_2^*) \leq \frac{\varepsilon}{2}(U - U') + w(\sigma^*) + w(\sigma_1).$$

Combining with $w(\sigma_1) \leq \frac{\varepsilon}{2} U'$ and $w(\sigma_2) \leq \frac{\varepsilon}{2}(U - U') + w(\sigma_2^*)$,

$$w(\sigma_1) + w(\sigma_2) \leq 2w(\sigma_1) + w(\sigma^*) + \varepsilon(U - U') \leq w(\sigma^*) + \varepsilon U.$$

Thus, the reported cost is an ε-close 1-Wasserstein distance.



(a) (b)

Figure 3: (a) An example of an augmenting path from a point b_P to a point a_P. In σ_2^*, we add a flow from $c_{\square_b}$ to $c_{\square_a}$. (b) An example showing edges of first case (a_1, b_1), second case (a_2, b_2), and third case (a_3, b_3). The budget assigned to them are shown as dashed lines.

B Missing details of Section 3

In this section, we provide the details of the discussion on the quality of approximation in Section 3 and show that the cost computed by our 1-Wasserstein algorithm is an $O(d \log_{\sqrt{d}/\varepsilon} n)$-approximation of the optimal transport cost on the point set $A \cup B$. For any level i of the tree T, recall that C_i denotes the set of all cells at level i, the grid $\mathbb{G}_i$ defines level i of T, and ℓ_i is the cell-side-length of $\mathbb{G}_i$. In the following, first in Lemma B.1, we show that the total cost of the transport plans computed at all cells at level h is $O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*)$.

Lemma B.1. $\mathbb{E}[\sum_{\square \in \mathsf{C}_h} w(\sigma_\square)] = O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*)$.

Proof. For any non-empty cell $\square \in \mathsf{C}_h$, $\sigma_\square$ moves the supplies or demands of the only point in $V_\square$ to $c_\square$. Since the diameter of the cells at level h is $\sqrt{d}\ell_h$ and by the linearity of expectation,

$$\mathbb{E}[\sum_{\square \in \mathsf{C}_h} w(\sigma_\square)] = \sum_{\square \in \mathsf{C}_h} \mathbb{E}[w(\sigma_\square)] \leq \sum_{\square \in \mathsf{C}_h} \sqrt{d}\eta(\square)\mathbb{E}[\ell_i] = 2\sqrt{d}U\mathbb{E}[\ell_h].$$

Next, we bound $\mathbb{E}[\ell_h]$. For any level i, $\ell_i = 2(\frac{\varepsilon}{4\sqrt{d}})^i$. Let $C_{\min}$ denote the minimum distance between any pair of points in $A \cup B$. Suppose j is the lowest integer with $2\sqrt{d}(\frac{\varepsilon}{4\sqrt{d}})^j < C_{\min}$; i.e, the diameter of any cell with side-length $2(\frac{\varepsilon}{4\sqrt{d}})^j$ is less than $C_{\min}$ and no two points of $A \cup B$ can lie inside the same cell of a grid with cell-side-length $2(\frac{\varepsilon}{4\sqrt{d}})^j$. Therefore, $h \leq j$. Since $\Delta(A \cup B) = n^{O(1)}$, we have $j = \lceil \log_{\frac{4\sqrt{d}}{\varepsilon}} \frac{2\sqrt{d}}{C_{\min}} \rceil = O(\log_{\sqrt{d}/\varepsilon} n)$. Let (u, v) be the closest pair of points in $A \cup B$. Then, for any i, the probability $\Pr(h = i)$ is at most the probability that the edge (u, v) crosses the grid $\mathbb{G}_i$, which is at most $\frac{\sqrt{d}\|u-v\|}{\ell_i}$. Thus,

$$\mathbb{E}[\ell_h] = \sum_{i=1}^{j} \ell_i \Pr(h = i) \leq \sum_{i=1}^{j} \ell_i \frac{\sqrt{d}C_{\min}}{\ell_i} = O(\sqrt{d}C_{\min} \log_{\frac{\sqrt{d}}{\varepsilon}} n).$$

Since $w(\sigma^*) \geq UC_{\min}$,

$$\mathbb{E}[\sum_{\square \in \mathsf{C}_h} w(\sigma_\square)] \leq 2\sqrt{d}U\mathbb{E}[\ell_h] = O(dUC_{\min} \log_{\sqrt{d}/\varepsilon} n) = O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*).$$

□

Next, we describe the two steps of our analysis to show that for each level $i < h$, the expected cost of the transport plans computed at all cells of level i is $O(d)w(\sigma^*)$.

Assigning Budgets: For a level $i < h$, consider the grids $\mathbb{G}_i$ and $\mathbb{G}_{i+1}$ (this is defined since $i < h$). For any edge $(a, b) \in A \times B$ with $\sigma^*(a, b) > 0$, we define its *budget* $\phi_i(a, b)$ as follows:

- *Case 1: both a and b are inside the same cell of $\mathbb{G}_{i+1}$*: We set $\phi_i(a,b) = 0$.
- *Case 2: both a and b are inside the same cell of $\mathbb{G}_i$ but in different cells $\square_a$ and $\square_b$ of $\mathbb{G}_{i+1}$*: We set $\phi_i(a,b) = (\sqrt{d}\ell_{i+1} + \|c_{\square_a} - c_{\square_b}\|)\sigma^*(a,b)$. Intuitively, this is the cost of moving from b to a via $c_{\square_b}$ and $c_{\square_a}$.
- *Case 3: a and b lie inside different cells $\square_a$ and $\square_b$ of $\mathbb{G}_i$*: We set $\phi_i(a,b) = (\sqrt{d}\ell_i)\sigma^*(a,b)$. Intuitively, this is the cost of moving a to $c_{\square_a}$ and b to $c_{\square_b}$.

Figure A(b) shows an example of each of the above three cases. The following lemma relates the expected budget on any edge to its Euclidean cost.

Lemma B.2. *For any edge $(a,b) \in A \times B$, $\mathbb{E}[\phi_i(a,b)] = O(d)\|a-b\|\sigma^*(a,b)$.*

Proof. The probability of a and b lying inside the same cell of $\mathbb{G}_i$ but on different cells of $\mathbb{G}_{i+1}$ is no more than the probability of (a,b) crossing $\mathbb{G}_{i+1}$. Therefore,

$$\begin{aligned}\mathbb{E}[\phi_i(a,b)] \leq \Pr((a,b) \text{ crosses } \mathbb{G}_{i+1})(\sqrt{d}\ell_{i+1} + \|c_{\square_a} - c_{\square_b}\|)\sigma^*(a,b) \\ + \Pr((a,b) \text{ crosses } \mathbb{G}_i)\sqrt{d}\ell_i\sigma^*(a,b).\end{aligned}$$

By triangle inequality, $\|c_{\square_a} - c_{\square_b}\| \leq \sqrt{d}\ell_{i+1} + \|a-b\|$. In addition, for any $j \leq h$, the probability that the edge (a,b) crosses the grid $\mathbb{G}_j$ is $\Pr((a,b) \text{ crosses } \mathbb{G}_j) \leq \min\{1, \frac{\sqrt{d}\|a-b\|}{\ell_j}\}$. Thus,

$$\begin{aligned}\mathbb{E}[\phi_i(a,b)] &\leq \Pr((a,b) \text{ crosses } \mathbb{G}_{i+1})(2\sqrt{d}\ell_{i+1} + \|a-b\|)\sigma^*(a,b) \\ &\quad + \Pr((a,b) \text{ crosses } \mathbb{G}_i)\sqrt{d}\ell_i\sigma^*(a,b) \\ &\leq \|a-b\|\sigma^*(a,b) + \left(\frac{\sqrt{d}\|a-b\|}{\ell_{i+1}}2\sqrt{d}\ell_{i+1} + \frac{\sqrt{d}\|a-b\|}{\ell_i}\sqrt{d}\ell_i\right)\sigma^*(a,b) \\ &\leq (1+3d)\|a-b\|\sigma^*(a,b).\end{aligned}$$

□

Using the linearity of expectation, the following is a straightforward corollary of Lemma B.2.

Corollary B.3. $\mathbb{E}[\sum_{(a,b)\in A\times B}\phi_i(a,b)] = O(d)w(\sigma^*)$.

Redistribution of Budgets to cells: In this section, we distribute the total budget on all edges of σ^* to non-empty cells of $\mathbb{G}_i$ and show that the budget assigned to any cell $\square \in \mathbb{G}_i$ is at least $w(\sigma_\square)/2$. For any cell $\square$ at level i of T, we define a budget $\phi_\square$ as follows. For any edge $(a,b) \in A \times B$ with $\sigma^*(a,b) > 0$, if

- *both a and b lies inside $\square$:* We assign the entire budget of (a,b), i.e, $\phi_i(a,b)$, to $\phi_\square$.
- *only a or b is inside $\square$:* We assign half the budget of (a,b), i.e, $\phi_i(a,b)/2$, to $\phi_\square$.

It is easy to observe that the sum of the budget of all cells at level i of T is equal to the sum of the budget of all edges carrying a positive flow in σ^*.

For any cell $\square$, the following lemma relates the cost of $\sigma_\square$ to the budget assigned to $\square$.

Lemma B.4. *For any cell $\square$ of T, $w(\sigma_\square) \leq 2\phi_\square$.*

Proof. To prove the lemma, we show that there exists a transport plan σ' on the instance $\mathcal{I}_\square$ that has a cost no more than $\phi_\square$. Assuming this and considering that $\sigma_\square$ is a 2-approximate transport plan on $\mathcal{I}_\square$, we get $w(\sigma_\square) \leq 2w(\sigma') \leq 2\phi_\square$, as desired. Let $\square$ be a cell at level i of T. Recall that $V_\square = (A \cup B) \cap \square$. For any point $u \in V_\square$, let $\square_u$ be the child of $\square$ that contains u. For any edge (u,v) with $\sigma^*(u,v) > 0$, consider the following cases:

- If $u, v \in V_\square$ and $\square_u \neq \square_v$, then σ' transports $\sigma^*(u,v)$ supplies from $c_{\square_u}$ to $c_{\square_v}$. Note that the edge (u,v) contributes to $\phi_\square$ by $\sigma^*(u,v)(\sqrt{d}\ell_{i+1} + \|c_{\square_u} - c_{\square_v}\|)$, which is greater than the cost of transporting $\sigma^*(u,v)$ supplies from $c_{\square_u}$ to $c_{\square_v}$.

- If $u \in V_\square$ and $v \notin V_\square$, then if u is a supply point, then σ' transports $\sigma^*(u,v)$ supplies from $c_{\square_u}$ to $c_\square$. Otherwise, u is a demand point and σ' transports $\sigma^*(u,v)$ supplies from $c_\square$ to $c_{\square_u}$. Note that (u,v) contributes to $\phi_\square$ by $\sigma^*(u,v)\frac{\sqrt{d}\ell_i}{2}$, which is at least the cost of transporting $\sigma^*(u,v)$ supplies between $c_{\square_u}$ and $c_\square$.
- If $u, v \in V_\square$ and $\square_u = \square_v$ or if $u, v \notin V_\square$, then σ' do not transport any supplies to/from the points u and v.

By the discussion above, the cost of σ' is no more than $\phi_\square$. However, σ' might not be a feasible transport plan for the instance $\mathcal{I}_\square$. In that case, we can improve σ' as follows. If there exists two edges $(a,b'),(a',b) \in A \times B$ with positive flow in σ^* such that $a, b \in V_\square$ and $a', b' \notin V_\square$, then σ' transports supplies from $c_\square$ to $c_{\square_a}$ and from $c_{\square_b}$ to $c_\square$, which can be replaced with a direct flow from $c_{\square_b}$ to $c_{\square_a}$ without increasing the cost of σ'. Similarly, if there exists any pair of points c_u and c_v where σ' transports flow both from c_u to c_v and from c_v to c_u, then cancelling the two flows with each other will not increase the cost of σ'. The obtained transport plan is a feasible solution for the instance $\mathcal{I}_\square$ with a cost $w(\sigma') \leq \phi_\square$. $\square$

Combining Lemma B.1, Corollary B.3, and Lemma B.4, we get the following.

$$\mathbb{E}[\sum_{\square \in T} w(\sigma_\square)] = 2\mathbb{E}[\sum_{\square \in T} \phi_\square] = 2\mathbb{E}[\sum_{i=0}^{h} \sum_{(a,b)\in A\times B} \phi_i(a,b)] = O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*).$$

Corollary B.5. $\mathbb{E}[\sum_{\square \in T} w(\sigma_\square)] = O(d \log_{\sqrt{d}/\varepsilon} n) w(\sigma^*)$.

C Missing details of Section 4

C.1 Input Transformation

Let $A' \cup B'$ be the input point set. Similar to Agarwal et al. (2022b), we transform $A' \cup B'$ into another point set $A \cup B$ such that

(T1) The coordinates of the points in $A \cup B$ are positive integers bounded by $n^{O(1)}$,

(T2) Any optimal matching with respect to $A \cup B$ is a $(1+\varepsilon)$ approximation for $A' \cup B'$, and,

(T3) The cost of the optimal matching of A and B is at least $5\sqrt{d}n/\varepsilon$ and at most $5d^{3/2}n \log n/\varepsilon$.

The details of this transformation are as follows. Suppose the hierarchical greedy paradigm described in Section 1, when applied to the input points $A' \cup B'$, returns a value θ which is, in expectation, an $O(d \log n)$-approximation of the cost of the optimal matching on $A' \cup B'$. Using θ, we can transform any point $p' = (p_{x_1}, \ldots, p_{x_d}) \in A' \cup B'$ into another point $p = (\lceil \frac{5d^{3/2} n \log n}{\varepsilon\theta} p_{x_1} \rceil, \ldots, \lceil \frac{5d^{3/2} n \log n}{\varepsilon\theta} p_{x_d} \rceil)$ to obtain a set of points $A \cup B$. If any pair of points $(a,b) \in A \times B$ map to the same location, we match the two and remove them from $A \cup B$. The resulting point set satisfies (T1)–(T3).

C.2 Retrieving an approximate bipartite matching

In this section, we show how to retrieve a matching M on the point set $A \cup B$ using an identical approach as described in Section 3. We process the grids $\langle \mathbb{G}_h, \ldots, \mathbb{G}_1 \rangle$ in the decreasing order of their level. For any cell $\square$ of $\mathbb{G}_h$, we map the unique point in $V_\square$ to the center point $c_\square$. Inductively, assume that for a level $i < h$, after processing all non-empty cells of $\mathbb{G}_{i+1}$, conditions (i)–(iii) defined in Section 3 hold for each cell of that grid. For each non-empty cell $\square$ of $\mathbb{G}_i$, we show how to process $\square$ so that conditions (i)–(iii) hold for $\square$ as well. For any two children $\square_1, \square_2 \in \mathsf{C}[\square]$, we match $\sigma_\square(c_{\square_1}, c_{\square_2})$ many unmatched points that are mapped to $c_{\square_1}$ to the unmatched points that are mapped to $c_{\square_2}$. Additionally, for any child $\square_1 \in \mathsf{C}[\square]$ with $\sigma_\square(c_{\square_1}, c_\square) > 0$, we map $\sigma_\square(c_{\square_1}, c_\square)$ unmatched points from $c_{\square_1}$ to $c_\square$.

C.3 QUALITY OF APPROXIMATION

In this section, we show that the cost returned by the algorithm is within an additive error of $\min\{\varepsilon, (d \log\log n) w(M^*)\}$ from the optimal matching cost. Consider any cell $\square$ of $\mathbb{G}_1$. Our algorithm first computes some matching $M_1^{\square}$ by matching the points within each child of $\square$. Then, it moves the unmatched points to the centers of the children of $\square$ and compute a matching $M_2^{\square}$ on this instance. Let $w_{\square}$ be the cost of the matching computed inside $\square$ after transforming the points back to the original ones. To show that the algorithm reports a ε-close matching cost, we show that for each non-empty cell $\square$ of $\mathbb{G}_1$, $w(M_1^{\square}) + w(M_2^{\square})$ is within an additive error of $|V_{\square}|\ell_1\varepsilon$ from the optimal matching cost on $V_{\square}$. Therefore, when transformed back to the original point sets in the unit hypercube, the additive error would be at most $|V_{\square}|\varepsilon$, as desired.

We use an identical discussion as Section A. Let $m_1^{\square}$ be the size of the matching $M_1^{\square}$. Since the diameter of each child of $\square$ is $\sqrt{d}\ell_2$, $w(M_1^{\square}) \leq m_1^{\square}\sqrt{d}\ell_2$. Furthermore, since the spread of the points in the instance created in the second step, as described above, is at most $\sqrt{d}\ell_1/\ell_2$, the matching $M_2^{\square}$ is obtained by computing a $\frac{1}{\sqrt{d}\ell_1/\ell_2}$-close transport plan on points that are scaled by a factor of at most $\frac{1}{\sqrt{d}\ell_1}$, which adds an additive error of $(|V_{\square}| - m_1^{\square})\frac{\ell_1}{\ell_1/\ell_2}$. Finally, the cost of moving the unmatched points inside the children of $\square$ to their centers is at most $(|V_{\square}| - m_1^{\square})\sqrt{d}\ell_2$. Since $\ell_1 = \frac{\sqrt{d}}{\varepsilon}\delta n$ and $\ell_2 = \delta\sqrt{n}$, and considering the discussion in Section A, the additive error of our algorithm would be at most

$$2m_1^{\square}\sqrt{d}\ell_2 + (|V_{\square}| - m_1^{\square})\ell_2 + (|V_{\square}| - m_1^{\square})\sqrt{d}\ell_2 \leq |V_{\square}|\ell_1\varepsilon.$$

Therefore, it suffices to show that the matching M is an $O(d \log\log n)$-approximate matching. To do so, using an argument identical to Section 3, we can show that the total cost of the transport plans computed at all cells of level i (in expectation) is at most $O(d)w(M^*)$.

Lemma C.1. $\mathbb{E}[\sum_{\square \in \mathsf{C}_i} w(\sigma_{\square})] \leq O(d)w(M^*)$.

Summing over all levels of T, we get the following corollary.

Corollary C.2. $\mathbb{E}[w(M)] \leq \sum_{i=1}^{h-1} \mathbb{E}[\sum_{\square \in \mathsf{C}_i} w(\sigma_{\square})] \leq O(d \log\log n)w(M^*)$.

C.4 EFFICIENCY OF OUR ALGORITHM

To analyze the running time of our algorithm, we first show that, with probability at least $1 - \frac{\varepsilon}{\sqrt{d}}$, no edges of an optimal matching will cross $\mathbb{G}_1$ and consequently, each child of $\square^*$ contains an equal number of points of A and B. In other words, all children of $\square^*$ are neutral cells and the problem instance $\mathcal{I}_{\square^*}$ is an empty instance.

Lemma C.3. *Let M^* be an optimal matching on the point set $A \cup B$. Then, with probability at least $1 - \varepsilon/\sqrt{d}$, no edge of M^* crosses $\mathbb{G}_1$.*

Proof. Recall that $\ell_1 = 5d^{5/2}n \log n/\varepsilon^2$. For any pair of points $(a, b) \in A \times B$, the probability that the edge (a, b) crosses the grid $\mathbb{G}_1$ is upper-bounded as follows.

$$\Pr((a, b) \text{ crosses } \mathbb{G}_1) \leq \frac{\sqrt{d}\|a - b\|}{5d^{5/2}n \log n/\varepsilon^2}. \tag{1}$$

Let $X_{(a,b)}$ be an indicator random variable where $X_{(a,b)} = 1$ if (a, b) crosses $\mathbb{G}_1$ and $X_{(a,b)} = 0$ otherwise. Clearly, $\mathbb{E}[X_{(a,b)}] = \Pr((a, b) \text{ crosses } \mathbb{G}_1)$. Define $X = \sum_{(a,b)\in M^*} X_{(a,b)}$ to be the number of edges of M^* crossing the grid $\mathbb{G}_1$. By linearity of expectation and Equation 1,

$$\mathbb{E}[X] = \sum_{(a,b)\in M^*} \mathbb{E}[X_{(a,b)}] \leq \sum_{(a,b)\in M^*} \frac{\sqrt{d}\|a - b\|}{5d^{5/2}n \log n/\varepsilon^2} = \frac{w(M^*)}{5d^2 n \log n/\varepsilon^2} \leq \frac{\varepsilon}{\sqrt{d}},$$

where the last inequality follows from property (T3) in the input transformation. Therefore, with probability at least $1 - \varepsilon/\sqrt{d}$, $X = 0$ and no edge of M^* crosses the grid $\mathbb{G}_1$. $\square$

Therefore, by constructing $O(\log_{\sqrt{d}/\varepsilon} n)$ randomly-shifted hierarchical partitions, with high probability, in at least one partition, all children of the root are neutral cells. Assuming such partition, we show that for each $0 < i \leq h$, the algorithm requires $O(T(n, \varepsilon/d))$ time to process all cells at level i. Summing over all levels, the total running time of our algorithm is $O(T(n, \varepsilon/d) \log \log n)$.

Recall that C_i denotes the set of cells of T at level i. For any cell $\square$, recall that $n_\square$ denotes the number of points in $\mathcal{I}_\square$. Define $n_i = \sum_{\square \in \mathsf{C}_i} n_\square$. For any cell $\square \in \mathsf{C}_i$, the spread of the points in $\mathcal{I}_\square$ is $O(\frac{d}{\varepsilon} n^{1/2^i})$. Therefore, since $T(n, \varepsilon) = \tilde{O}(n^k/\varepsilon)$ with $k \geq 1$, the execution time of our algorithm over all cells of level i is $\sum_{\square \in \mathsf{C}_i} T(n_\square, \frac{\varepsilon}{dn^{1/2^i}}) \leq T(n_i, \frac{\varepsilon}{dn^{1/2^i}})$. Additionally, since $n_i \leq n$, we have $\mathbb{E}[n_i^k] \leq n^{k-1}\mathbb{E}[n_i] = \left(\frac{\mathbb{E}[n_i]}{n}\right) n^k$. Therefore,

$$\sum_{\square \in \mathsf{C}_i} T(n_\square, \frac{\varepsilon}{dn^{1/2^i}}) \leq T\left(n_i, \frac{\varepsilon}{dn^{1/2^i}}\right) \leq \frac{n^{1/2^i}\mathbb{E}[n_i]}{n} T\left(n, \frac{\varepsilon}{d}\right). \tag{2}$$

Next, we bound $\mathbb{E}[n_i]$. For any cell $\square$ and any child $\square' \in \mathsf{C}[\square]$, the weight of $c_{\square'}$ in $\mathcal{I}_\square$ is the minimum number of matching edges connecting a point inside $\square'$ to a point out of it in any maximum matching. Summing over all cells of level i, n_i is upper-bounded by the number of edge of M^* crossing the grid $\mathbb{G}_{i+1}$. The following lemma bounds this number as a function of $w(M^*)$ and ℓ_{i+1}.

Lemma C.4. *For any $0 < j < h$, the expected number of matching edges in M^* crossing $\mathbb{G}_j$ is at most $\sqrt{d}w(M^*)/\ell_j$.*

Proof. For any edge $(a, b) \in M^*$, the probability that the edge (a, b) crosses the grid $\mathbb{G}_j$ is $\Pr((a, b) \text{ crosses } \mathbb{G}_j) \leq \frac{\sqrt{d}\|a-b\|}{\ell_j}$. Let $X_j(a, b)$ be an indicator random variable such that $X_j(a, b) = 1$ if (a, b) crosses $\mathbb{G}_j$ and $X_j(a, b) = 0$ otherwise. Thus, $\mathbb{E}[X_j(a, b)] \leq \frac{\sqrt{d}\|a-b\|}{\ell_j}$. Define X_j to be a random variable indicating the number of edges of M^* crossing $\mathbb{G}_j$; i.e, $X_j = \sum_{(a,b)\in M^*} X_j(a, b)$. Using the linearity of expectation, the expected value of X_j is

$$\mathbb{E}[X_j] = \sum_{(a,b)\in M^*} \mathbb{E}[X_j(a, b)] \leq \frac{\sqrt{d}w(M^*)}{\ell_j}.$$

□

By invoking Lemma C.4 with $j = i + 1$, we get $\mathbb{E}[n_i] \leq \sqrt{d}w(M^*)/\ell_{i+1}$. By definition, $\ell_{i+1} = \delta n^{1/2^i}$ and from property (3) of the input transformation, $w(M^*) \leq 5d^{3/2} n \log n/\varepsilon$. Plugging into Equation 2, the time taken to process all cells of level i is

$$\frac{\mathbb{E}[n_i]}{n^{1-\frac{1}{2^i}}} T\left(n, \frac{\varepsilon}{d}\right) \leq \frac{\sqrt{d}w(M^*)}{\ell_{i+1} n^{1-\frac{1}{2^i}}} T\left(n, \frac{\varepsilon}{d}\right) \leq T\left(n, \frac{\varepsilon}{d}\right),$$

as desired.

D An Improved Relative Approximation Algorithm

In Lemma 2.2, we show that any ε-additive approximation algorithm for the 1-Wasserstein problem with a running time of $T(n, \varepsilon)$ can be used to obtain a $(1 + \varepsilon)$-relative approximation algorithm in $T(n, \varepsilon/\Delta)$ time; here, Δ is the spread of the input points. Both of our algorithms in Section 3 and Section 4 use this lemma to obtain a 2-approximate transport plan for the instance created for each cell. Plugging the existing additive approximation algorithms, such as the algorithm by Lahn et al. (2019), in Lemma 2.2 results in a quadratic running time for our algorithms for 1-Wasserstein and EBM problems. In this section, using the ideas mentioned in (Agarwal & Sharathkumar (2014); Lahn & Raghvendra (2019)), we show that one can use an approximate dynamic nearest neighbor data structure (ANN) to improve the running time of our algorithms.

For demand points A and supply points B, let w^* denote the cost of the optimal transport plan on $A \cup B$. Let (α, ε)-*approximate transport plan* on $A \cup B$ be a transport plan σ satisfying $w(\sigma) \leq$

$\alpha w^* + \varepsilon$. Using an identical discussion as in Section 2.2, when points have a unit diameter, by setting $\varepsilon = 1/\Delta$, the additive error of the transport plan will be at most w^*, and as a result, any $(\alpha, 1/\Delta)$-approximate transport plan is an $(\alpha + 2)$-relative approximate transport plan. In this section, we modify the algorithm by Lahn et al. (2019) (we refer to this algorithm by the LMR-Algorithm) to obtain an (α, ε)-approximate transport plan in $\tilde{O}(n\Phi(n, \alpha)/\varepsilon)$ time; here, $\tilde{O}$ hides factors of $\mathrm{poly}(\log n)$.

Overview of the LMR-Algorithm: For input points $A \cup B$ of size n and diameter C, the LMR-Algorithm computes an ε-close transport plan on $A \cup B$ as follows. Initially, the distances are scaled by $4/\varepsilon$ and rounded down so that the distance between any two points is an integer bounded by $O(C/\varepsilon)$. The LMR-Algorithm maintains a transport plan $\sigma : A \times B \to \mathbb{R}_{\geq 0}$ and a set of integer dual weights $y : A \cup B \to \mathbb{Z}_{\geq 0}$ for the points satisfying the following feasibility conditions.

$$\begin{aligned} &y(b) - y(a) \leq d(a, b) + 1, && \text{if } \sigma(a, b) < \min\{\eta(a), \eta(b)\}, \\ &y(b) - y(a) \geq d(a, b), && \text{if } \sigma(a, b) > 0. \end{aligned} \tag{3}$$

The LMR-Algorithm runs in phases, where in each phase, similar to the algorithm by Gabow & Tarjan (1989), it executes a Hungarian search procedure followed by an execution of the DFS procedure to find a maximal set of vertex-disjoint admissible augmenting paths. The algorithm then augments the transport plan σ using the set of computed augmenting paths. The algorithm guarantees to increase the dual weight of each free point of B at each phase by at least 1. Additionally, each free point of A will have a zero dual weight at each phase. Therefore, after $\frac{4C}{\varepsilon} + 2$ phases, no point of B can be a free point, because otherwise, if there exists a free point $b' \in B$ and a free point $a' \in A$, $y(b') - y(a') > d(a', b') + 1$, which violates the feasibility conditions. Therefore, after $\frac{4C}{\varepsilon} + 2$ phases, the LMR-Algorithm computes a complete ε-close transport plan (see Lemma 2.3 in (Lahn et al. (2019))).

Implementation of the LMR-Algorithm using the Partial DFS: As discussed by Lahn & Raghvendra (2019), each phase of the Gabow-Tarjan's algorithm can be implemented using an execution of the partial DFS procedure on the admissible graph. In this implementation, in each phase, the algorithm iteratively initiates a partial DFS procedure from each free point b of B and maintains the search path $Q = \langle b = u_0, \ldots, u_k \rangle$. To process the point u_k, if u_k is a free point of A, the algorithm has found an admissible augmenting path Q. In this case, it stops the partial DFS, augments the transport plan along Q as described in Section 2.1 in (Lahn et al. (2019)), and initiates the next partial DFS. Otherwise, if there is an admissible outgoing edge (u_k, v) from u_k, if v has not been processed yet, the algorithm adds v to the end of Q as u_{k+1} and starts processing v. Finally, if there are no admissible outgoing edges from u_k, the algorithm sets $y(u_k) \leftarrow y(u_k) + 1$, marks u_k as processed, remove u_k from Q, and continue the search from u_{k-1}.

Implementation of each phase using Approximate Nearest Neighbor data structure: Next, we discuss how to use an α-approximate dynamic nearest neighbor data structure (α-ANN) to implement the partial DFS procedures more efficiently while introducing a factor of α in the approximation factor. Note that the dual weight of each point is an integer between 0 and $O(C/\varepsilon)$; i.e, there are only $O(C/\varepsilon)$ unique dual weights. Similar to Agarwal & Sharathkumar (2014), we define a set of α-feasibility conditions as follows.

$$\begin{aligned} &y(b) - y(a) \leq \alpha d(a, b), && \text{if } \sigma(a, b) < \min\{\eta(a), \eta(b)\}, \\ &y(b) - y(a) \geq d(a, b), && \text{if } \sigma(a, b) > 0. \end{aligned} \tag{4}$$

Any complete α-feasible transport plan is an α-approximate transport plan. Next, we describe the modified implementation of the partial DFS. We construct, for any $y' \in [0, \lfloor \frac{4C}{\varepsilon} + 1 \rfloor]$, an α-ANN for the subset of points of A with dual weight y'. Suppose the partial DFS procedure has maintained a path $Q = \langle u_0, \ldots, u_k \rangle$ and is processing u_k. If u_k is a free point of A, then we have found an augmenting path Q. We augment the transport plan along Q and add every point of A in Q to the α-ANN. Otherwise, if $u_k \in A$ is not a free point, then we simply check all the neighbors of u_k to find an admissible outgoing edge. Otherwise, $u_k \in B$. For any $y' \in [0, \lfloor \frac{4C}{\varepsilon} + 1 \rfloor]$, we query the α-ANN corresponding to the points of A with dual weight y' to find an approximate nearest neighbor a of u_k. If $\frac{y(u_k) - y'}{\alpha} \leq d(a, u_k) \leq y(u_k) - y'$ (successful query), we consider (a, u_k) as an admissible edge and add a as u_{k+1} to the search path Q. We remove a from the α-ANN and continue the search. Otherwise, the returned point a has distance $d(a, u_k) > y(u_k) - y'$ (unsuccessful query). If

all $O(C/\varepsilon)$ queries are unsuccessful, we set $y(u_k) \leftarrow y(u_k) + 1$, mark u_k as processed, and remove it from Q.

Next, we give a discussion on the running time of this algorithm. While processing any point $b \in B$, for any successful query returning a point a, two cases might happen: (1) the algorithm removes the neighbor point a from Q without finding any augmenting paths, or (2) the algorithm finds an augmenting path. The first case happens once for each point of A in each phase and requires $O(nC/\varepsilon)$ queries in total. The second case requires at most one query for each edge of each augmenting path, and by Lemma 2.4 in (Lahn et al. (2019)), requires $O(nC^2/\varepsilon^2)$ queries in total. Furthermore, in each phase, the search from b might result in at most $O(C/\varepsilon)$ unsuccessful queries and $O(nC^2/\varepsilon^2)$ queries in total. Processing all points of A requires searching for admissible edges among the edges with a positive flow, which by Lemma 2.4 in (Lahn et al. (2019)) is at most $O(nC^2/\varepsilon^2)$ over all phases. Finally, in each iteration, each point undergoes a dual-weight update at most once and there are $O(nC/\varepsilon)$ updates in total. Therefore, if $\Phi(\alpha, n)$ denotes the query/update time of an α-ANN on n points, this algorithm computes an (α, ε)-approximate transport plan in $O(\frac{nC^2\Phi(\alpha,n)}{\varepsilon^2})$ time.

Extensions of our algorithms: In our algorithms, as in Section 2.2, we assume $C = 1$. Plugging $\varepsilon = 1/\Delta$, we get the following lemma.

Lemma D.1. *There exists an algorithm that, given two point sets A and B in the d-dimensional unit hypercube with a spread of Δ, computes an $O(\alpha)$-approximate transport plan in $O(n\Delta^2\Phi(n, \alpha))$ time.*

First, we extend our 1-Wasserstein algorithm from Section 3 using the α-ANN data structure. Recall that for each cell $\square$, our algorithm creates instances $\mathcal{I}_\square$ with a spread $\Delta \leq d/\varepsilon$. Recall that $n_\square$ denotes the number of points in the instance $\mathcal{I}_\square$. Using Lemma D.1, we compute an $O(\alpha)$-approximate transport cost on $\mathcal{I}_\square$ in $O(d^2 n_\square \varepsilon^{-2}\Phi(n, \alpha))$ time. For any level of the hierarchical partitioning, the total size of instances created at all cells of that level is at most n. Therefore, summing over all cells of the tree, we get the following result.

Theorem D.2. *There exists a randomized algorithm that, given two discrete distributions μ and ν in the d-dimensional unit hypercube with a spread of $\Delta = n^{O(1)}$ and a parameter $\varepsilon > 0$, computes a $O(d\alpha \log_{\sqrt{d}/\varepsilon} n)$-approximate transport plan in $O(d^2 n\Phi(n, \alpha)\varepsilon^{-2} \log_{\sqrt{d}/\varepsilon} n)$ time.*

Next, we discuss the extension of our EBM algorithm. To do so, first, we modify our hierarchical partitioning to obtain T', as follows. Define $\delta' := 10d^2\varepsilon^{-1} \log n$. Similar to Section 4, we define the root cell of T' as a randomly-shifted cell $\square^*$ that contains all points of $A \cup B$ and has a side-length of $2 \max\{C_{\max}, \ell_1/\varepsilon\}$; here, $\ell_1 = \delta' n$. For any cell $\square$ of T', if $\square$ contains at most $n^{\varepsilon/2}$ points of $A \cup B$, then we designate $\square$ as a leaf cell. Otherwise, let $\square$ be a cell of level i. Define the grid $\mathbb{G}_\square = \mathbb{G}(\square, \delta' n^{(2/3)^i})$ and add the non-empty cells of $\mathbb{G}_\square$ to T' as the children of $\square$. The height of T' is $h = O(\log \frac{d}{\varepsilon})$. Using a similar proof as in Lemma C.3, we can show that with probability at least $1/2$, no cell of level 1 is a surplus or deficit cell.

For any non-root cell $\square$ of level i of T', we construct an instance $\mathcal{I}_\square$ identical to the one in Section 4. The spread of the points in $\mathcal{I}_\square$ is $O(\sqrt{d}\frac{\ell_i}{\ell_{i+1}}) = O(\sqrt{d}n^{\frac{1}{3}\cdot(\frac{2}{3})^i})$. Additionally, by invoking Lemma C.4 on level i cells, we have $\mathbb{E}[n_i] = O(\frac{\sqrt{d}w^*}{\ell_{i+1}}) = O(n^{1-(\frac{2}{3})^{i+1}})$. Therefore, plugging Lemma D.1 as the approximate solver of the instances, we get the following theorem.

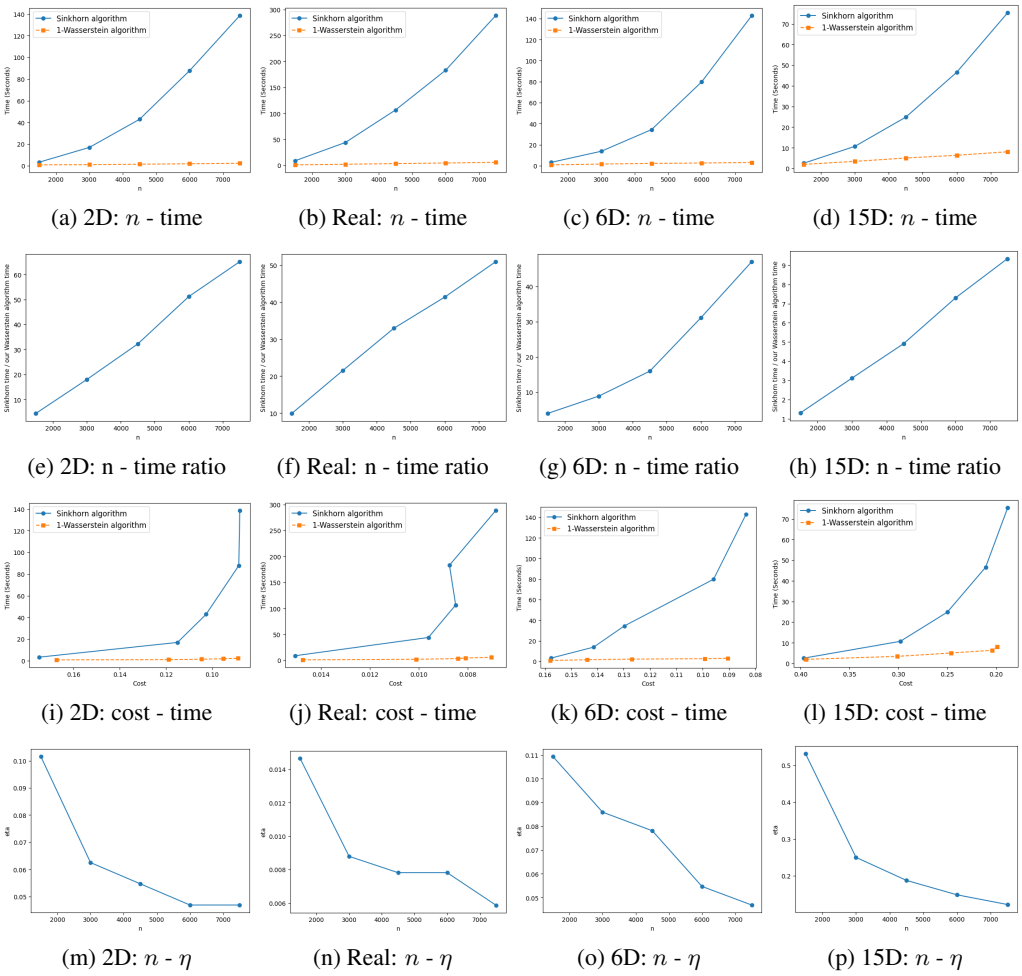
Theorem D.3. *There exists an algorithm that, given two sets of samples A and B each from distributions μ and ν, where $|A| = |B| = n$, and a parameter $\varepsilon > 0$, computes, with high probability, an $O(d\alpha \log \frac{d}{\varepsilon})$-approximate matching in $O(dn\Phi(n, \alpha) \log \frac{d}{\varepsilon} + n^{1+\varepsilon})$ time.*

E Additional Experiments

In this section, we show the results of our additional experiments. In addition to the two datasets (15D and Real) introduced in Section 5, we use three more datasets in our experiments: (i) a uniform distribution inside a unit square (Uniform), (ii) 3-dimensional Gaussian mixture with 2 mean points centered at $(0.24, 0.24, 0.24)$ and $(0.72, 0.72, 0.72)$ inside a unit cube (Bimodal), and (iii) a uniform distribution from inside a 2-dimensional unit square placed on a random plane in 6-dimensional space (6D).

E.1 COMPARISON WITH SINKHORN ALGORITHM

In this experiment, we want to determine, for any distribution and any fixed sample set drawn from that distribution, how long it takes for the Sinkhorn algorithm to compute a transport plan with a comparable cost to ours. Using a binary search, we identify, for each sample set, the value of the regularization parameter η in the Sinkhorn algorithm that produces a cost that is nearly identical to the cost computed by our algorithm. Figure 4 depicts the results of this experiment.



(a) 2D: n - time (b) Real: n - time (c) 6D: n - time (d) 15D: n - time

(e) 2D: n - time ratio (f) Real: n - time ratio (g) 6D: n - time ratio (h) 15D: n - time ratio

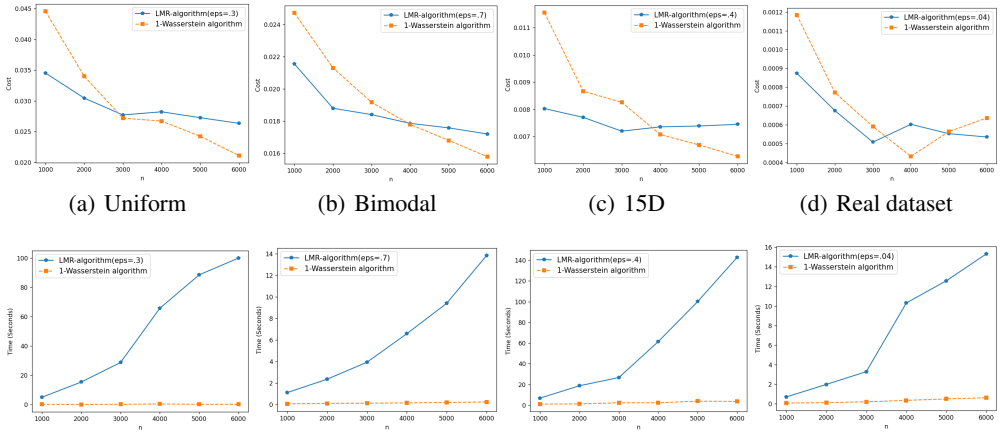
(i) 2D: cost - time (j) Real: cost - time (k) 6D: cost - time (l) 15D: cost - time

(m) 2D: n - η (n) Real: n - η (o) 6D: n - η (p) 15D: n - η

Figure 4: Comparison of our 1-Wasserstein algorithm with Sinkhorn algorithm when the regularization parameter η in Sinkhorn algorithm is set to compute a similar cost to our algorithm. (a)-(d) Sample size and execution time, (e)-(h) Sample size and the ratio of the execution time of Sinkhorn and our Wasserstein algorithm, (i)-(l) Cost and execution time, (m)-(n) Sample size and value of the regularization parameter η in Sinkhorn algorithm.

E.2 COMPARISON WITH THE LMR ALGORITHM

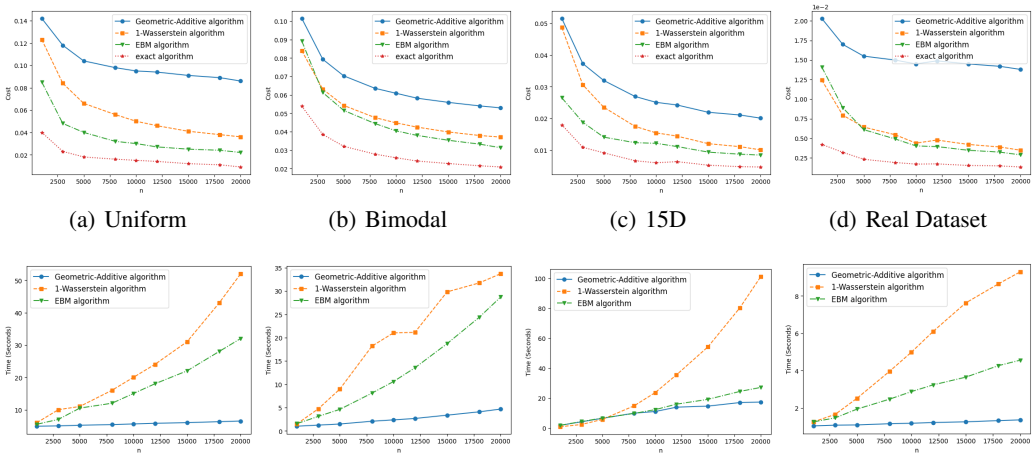
In this experiment, we use the LMR algorithm as the blackbox solver in our 1-Wasserstein algorithm. We compare our result with the result of applying the LMR algorithm on the entire pointset. The LMR algorithm takes an additive approximation parameter, ε, as its input parameter that affect its running time and output quality. For each dataset, we choose an additive approximation parameter for the LMR algorithm such that the cost returned by that algorithm is nearly equal to the cost computed by our 1-Wasserstein algorithm. Figure 5 shows the cost and running time plots of the executions of our algorithm and that of the LMR algorithm for the four data sets we use in our experiments, i.e. Uniform, Bimodal, 15D-Uniform and the Real dataset.



(a) Uniform (b) Bimodal (c) 15D (d) Real dataset

(e) Uniform - time (f) Bimodal - time (g) 15D - time (h) Real dataset - time

Figure 5: (a)-(d) comparing the cost computed by our 1-Wasserstein algorithm vs cost returned by LMR algorithm on Uniform, Bimodal, 15-D Uniform, and Real dataset. (e)-(h) comparing the running time of our 1-Wasserstein algorithm vs the running time of LMR algorithm on Uniform, Bimodal, 15-D Uniform, and Real dataset.



(a) Uniform (b) Bimodal (c) 15D (d) Real Dataset

(e) Uniform-Time (f) Bimodal-Time (g) 15D-Time (h) Real Dataset-Time

Figure 6: (a)–(d) estimated 1-Wasserstein distance, and (e)–(h) execution times of the algorithms

We can see in Figure 5 that while computing an almost same cost, the running time of our algorithm grows drastically slower than the LMR algorithm in all the cases.

E.3 Comparison with the Geometric-Additive algorithm

In this experiment, we compared the performance of our 1-Wasserstein and EBM algorithms to the Geometric Additive algorithm from Section 2.1. The results are illustrated in Figure 6.