Data-Dependent LSH for the Earth Mover's Distance

Rajesh Jayaram
Google Research
USA
rkjayaram@google.com

Erik Waingarten
University of Pennsylvania
USA
ewaingar@seas.upenn.edu

Tian Zhang
University of Pennsylvania
USA
tianzh@seas.upenn.edu

ABSTRACT

We give new data-dependent locality sensitive hashing schemes (LSH) for the Earth Mover's Distance (EMD), and as a result, improve the best approximation for nearest neighbor search under EMD by a quadratic factor. Here, the metric $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ consists of sets of s vectors in $\mathbb{R}^d$, and for any two sets x, y of s vectors the distance $\mathsf{EMD}(x, y)$ is the minimum cost of a perfect matching between x, y, where the cost of matching two vectors is their ℓ_p distance. Previously, Andoni, Indyk, and Krauthgamer gave a (data-independent) locality-sensitive hashing scheme for $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ when $p \in [1, 2]$ with approximation $O(\log^2 s)$. By being data-dependent, we improve the approximation to $\tilde{O}(\log s)$.

Our main technical contribution is to show that for any distribution μ supported on the metric $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, there exists a data-dependent LSH for dense regions of μ which achieves approximation $\tilde{O}(\log s)$, and that the data-independent LSH actually achieves a $\tilde{O}(\log s)$-approximation outside of those dense regions. Finally, we show how to "glue" together these two hashing schemes without any additional loss in the approximation.

Beyond nearest neighbor search, our data-dependent LSH also gives optimal (distributional) sketches for the Earth Mover's Distance. By known sketching lower bounds, this implies that our LSH is optimal (up to $\mathrm{poly}(\log \log s)$ factors) among those that collide close points with constant probability.

CCS CONCEPTS

• **Theory of computation → Nearest neighbor algorithms**.

KEYWORDS

Earth Mover's Distance, Locality Sensitive Hashing, Approximate Nearest Neighbor Search

ACM Reference Format:
Rajesh Jayaram, Erik Waingarten, and Tian Zhang. 2024. Data-Dependent LSH for the Earth Mover's Distance. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649666

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649666

1 INTRODUCTION

In the approximate nearest neighbor problem (ANN), we are given a set P of n points in a metric space (X, d_X), and the goal is to build a data structure that, upon receiving a query point $q \in X$, can quickly return a point $p \in P$ such that $d(p, q) \le c \cdot \min_{x \in X} d(q, x)$, for some approximation factor $c \ge 1$. The goal is to minimize c while answering queries as fast as possible—ideally, significantly faster than a linear scan. Nearest neighbor search is a fundamental problem in computer science, with applications in areas such as machine learning, data mining, information retrieval, computer vision, and many others. In this paper, we study approximate nearest neighbor search for the Earth Mover's Distance (EMD), also known as the Optimal Transport or Wasserstein-1 metric.

Let (X, d_X) be a "ground metric" (which, for us, will be $\mathbb{R}^d$ with the ℓ_p-norm for $p \in [1, 2]$). Given two collections of s elements from the ground metric, i.e., two multi-sets $x = \{x_1, \ldots, x_s\}, y = \{y_1, \ldots, y_s\} \subset X$ of size s, the Earth Mover's distance (EMD) between x and y is

$$\mathsf{EMD}(x, y) = \min_{\substack{\pi : [s] \to [s] \\ \text{bijection}}} \sum_{i=1}^{n} d_X(x_i, y_{\pi(i)}).$$

We will write $\mathsf{EMD}_s(X, d_X)$ to denote the metric space of size-s subsets of the (X, d_X) under the Earth Mover's distance. Computational aspects of EMD have long been studied within the theoretical computer science literature [1–4, 6, 7, 9, 13, 18, 23, 25, 26, 31, 37–39, 41, 45, 47, 52–54]. It is a central problem in algorithms, since it is a geometric version of bipartite matching. In addition, the Earth Mover's distance, and in particular nearest neighbor search under EMD, has gained immense popularity in natural language processing and machine learning [17, 19, 46, 50], where it is a popular measure of distance between sets of embeddings (such as Word2Vec or GloVe [49]).

The canonical approach for approximate nearest neighbor search is to employ *locality sensitive hashing* (LSH). These are randomized hash functions which partition the underlying metric space into hash buckets such that closer points are more likely to collide. An ANN data structure can then restrict its search to the hash buckets which the query maps to. By now, the theory of LSH for basic metrics like ℓ_1/ℓ_2 is well understood; the best c-approximations have query time $n^{1/c}$ for ℓ_1, and query time $n^{1/(2c^2-1)+o(1)}$ for ℓ_2 [5, 8, 10, 14, 40], leading to highly sublinear n^{ϵ}-time algorithms which achieve constant-factor (i.e., $1/\sqrt{\epsilon}$ or $1/\epsilon$) approximations.

Despite its popularity in theory and practice, LSH functions for EMD are not nearly as accurate as for ℓ_p spaces. This is because computing EMD, unlike ℓ_p, is significantly more computationally complex (for example, it does not decompose into a sum across coordinates). Computing EMD exactly requires solving a min-cost bipartite matching problem, achieved classically by the Hungarian

algorithm (in $O(s^3)$ time), and only recently in $O(s^{2+o(1)})$ time [29]. In addition, a simple heuristic like greedily generating a matching achieves a poor $\Omega(n^{0.58\cdots})$ approximation [51]. This makes EMD difficult to reason about, and computations involving EMD especially challenging for sublinear algorithms which are limited in their computational abilities. The typical approach in sublinear algorithms is to *embed* EMD into a "simpler" metric (usually ℓ_1) and use LSH in the simpler metric. Indyk [39] gave such an embedding of EMD into ℓ_1 with distortion $O(d \log \Phi)$ (where Φ is the aspect ratio and should be read as $\text{poly}(s)$, as there is a simple reduction to this case), leading to a $O(d \log s)$-approximation. This was later improved by [6], who gave a (randomized) embedding resulting in a LSH with approximation $O(\log s \log(d\Phi))$ (i.e., $O(\log^2 s)$ as it will also suffice to consider $d = \text{poly}(s)$). However, despite significant and recent focus from the sublinear algorithms community [16, 19, 20, 24, 27, 31], to date no further improvements to the $O(\log^2 s)$-approximation of [6] have been made. Our main result is a nearly quadratic improvement in this approximation with the same runtime.

Theorem 1 (Main Result—Informal version of Theorem 8). *For any constant $\epsilon > 0$ and $p \in [1, 2]$, there is a data structure for nearest neighbor search in $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, with approximation $\tilde{O}(\log s)$, pre-processing time $n^{1+\epsilon} \cdot \text{poly}(sd)$, and query time $n^{\epsilon} \cdot \text{poly}(sd)$.*

With regards to the runtime, note that in nearest neighbor search the primary goal is to have query time that significantly sublinear in n, which is the number of data points. In the context of EMD, the parameter s (along with d) is the description size of a single point in the metric space; in fact, it takes $O(sd)$ time to simply read a query. Thus, polynomial query time dependencies on s, d are generally acceptable, however exponential dependency on sd would be undesirable.

The key component of Theorem 1 is a new *data-dependent* locality-sensitive hash family for EMD, which, as we expand on next, is a relatively new algorithmic primitive for sublinear algorithms in geometric spaces [8, 10–12, 14]. We believe these data-dependent hash families are of independent interest, as they give rise to new and space optimal sketches for EMD in a distributional setting (see Section 8). Specifically, our LSH scheme gives a $\tilde{O}(\log s)$ approximation for this problem, nearly matching a $\Omega(\log s)$ lower bound of [6]. In particular, this implies a $\Omega(\log s)$-approximation lower bound for *any* LSH family where close points collide with constant probability (Theorem 10), which is a property our LSH family satisfies.

Data-Dependent (Locality-Sensitive) Hashing for EMD**.** As we further expand on in Section 1.1, the traditional guarantees of LSH are "data-independent," or "data-oblivious." In particular, LSH guarantees that, for *any* pair of points x, y from the metric, x and y tend to collide if they are close, and separate if they are far. One could imagine—and first successfully implemented in [8]—that the hash function be specifically tailored to the dataset P, and that doing so would improve the approximation. In data-dependent LSH, the dataset is still arbitrary and worst-case; yet, by exploiting properties of an arbitrary dataset, one may improve on the best approximations. Put succinctly, we show that every dataset of $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ has special structure to exploit algorithmically which we cannot capture with known (data-independent) LSH.

Sketching for Sets of Vectors. By now, there are various techniques for dealing with computationally "simple" objectives of high-dimensional vectors in sublinear regimes. For example, for ℓ_p-norms we now have an essentially complete understanding of sketching (i.e., communication complexity), locality-sensitive hashing, and metric embeddings [9, 14, 21, 35, 44, 48]. This work, as well as recent developments in geometric streaming [28, 30–32, 34] and parallel algorithms [33, 42], aims to develop sketching techniques (which were initially designed for a single high-dimensional vector) to support objectives over entire collections of high-dimensional vectors. In particular, an important technical contribution of this paper is to generalize the probabilistic tree embeddings of [31] (which were designed for streaming algorithms) to obtain an improved data-dependent LSH family for nearest neighbor search. We believe that the LSH families developed in this paper are an important step towards closing the gap in our understanding between sketching for individual vectors and sketching for sets of vectors.

1.1 Overview of Contributions and Techniques

We now overview the techniques involved in proving Theorem 1, and additionally state our formal results for data-dependent LSH (Theorem 3) and nearest neighbor search (Theorem 8). At a high level, this work can be seen within a progression of works, starting with [6] and continuing with [19, 31], on sketching for EMD via probabilistic tree embeddings. We aim to explain this progression, as it will highlight our main ideas (and the limitations of prior work).

(Data-Independent) LSH for EMD**.** An LSH for a metric space (X, d_X) is a hash family $\mathcal{H}$ which is so-called (r, cr, p_1, p_2)-sensitive. For a threshold $r \geq 0$, an approximation $c \geq 1$, and $0 < p_2 < p_1 < 1$, the guarantees are:

(1) **Close Points Collide**: $\mathbf{Pr}_{\boldsymbol{h} \sim \mathcal{H}}[\boldsymbol{h}(x) = \boldsymbol{h}(y)] \geq p_1$ for every $x, y \in X$ with $d_X(x, y) \leq r$.
(2) **Far Points Separate**: $\mathbf{Pr}_{\boldsymbol{h} \sim \mathcal{H}}[\boldsymbol{h}(x) = \boldsymbol{h}(y)] \leq p_2$ for every $x, y \in X$ with $d_X(x, y) \geq cr$.

The seminal work of [38, 40] designed such LSH families for several metric spaces (like $(\mathbb{R}^d, \ell_p)$ for $p \in [1, 2]$) and showed how to use them for c-approximate nearest neighbor with query time and space complexity governed by the gap between p_1 and p_2 (see Theorem 5). Using [6], one may construct an LSH for EMD with an arbitrary threshold r, approximation $c = O(\log^2 s)$, and constant $0 < p_2 < p_1 < 1$ (resulting in a theorem like Theorem 1, although with approximation $O(\log^2 s)$). The (data-independent) LSH for EMD crucially relies on a probabilistic metric embedding $\psi : \mathbb{R}^d \to \mathbf{T}$ from the ground metric of the EMD into a randomized tree metric [1], which satisfies that for any $x, y \in \mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ **(i)** $\mathsf{EMD}_{\mathbf{T}}(\psi(x), \psi(y)) \geq \mathsf{EMD}(x, y)$ with high probability, and **(ii)** $\mathbb{E}_{\mathbf{T}}[\mathsf{EMD}_{\mathbf{T}}(\psi(x), \psi(y))] \leq O(\log^2 s) \cdot \mathsf{EMD}(x, y)$.

The reason for embedding $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ into $\mathsf{EMD}_s(\mathbf{T}, d_{\mathbf{T}})$ is that EMD over tree-metrics is a much simpler metric. In particular,

[1] For applications in sublinear algorithms such as ours, it is important that the embeddings themselves can be efficiently stored and efficiently evaluated. Thus, the classical works on probabilistic tree embeddings [22, 36] are not applicable.

the greedy algorithm is optimal for EMD over trees, and as a consequence there is a folklore *isometric* embedding of $\mathsf{EMD}_s(\mathbf{T}, d_{\mathbf{T}})$ into ℓ_1 [25, 39] (see Fact 7.1), thereby embedding a *set* of vectors in a tree into a single vector in ℓ_1. Finally, after applying this embedding into ℓ_1, one can apply the classic LSH functions for ℓ_1 [40] (denoted as $\boldsymbol{\phi}$ below) to obtain a LSH function for $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$. This process is shown in the diagram below, where the names of the embeddings are shown on top of the arrows, and the distortion of those embeddings is shown below:

$$\begin{array}{ccccccc} \mathbb{R}^d & \xrightarrow{a\mapsto\psi(a)} & \mathbf{T} & & & & \\ & \searrow & & & & & \\ \mathsf{EMD}(\mathbb{R}^d,\ell_p) & \xrightarrow[O(\log^2 s)]{x\mapsto\psi(x)} & \mathsf{EMD}(\mathbf{T},d_{\mathbf{T}}) & \xrightarrow[\text{isometric}]{\text{folklore}} & \ell_1 & \xrightarrow{\boldsymbol{\phi}} & \{\text{hash buckets}\} \end{array} \tag{1}$$

Since the second mapping is isometric, the distortion of the entire embedding into ℓ_1 is $O(\log^2 s)$, thus the resulting LSH for $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ is a $O(\log^2 s)$ factor larger than the distortion incurred by the LSH $\boldsymbol{\phi}$ for ℓ_1.

Recently, [31] improved the probabilistic tree embedding of [6] by being data-dependent. They show that, for an arbitrary subset Ω of m vectors in $(\mathbb{R}^d, \ell_p)$, there exists a probabilistic tree embedding $\psi_\Omega : (\Omega, \ell_p) \to (\mathbf{T}_\Omega, d_{\mathbf{T}_\Omega})$ which depends on Ω, and that embeds Ω obtaining guarantees **(i)** and **(ii)** above as achieved by [6], except with an expected distortion of $\tilde{O}(\log(ms))$, where $m = |\Omega|$ (see Lemma 4.3).[2] This means that, if we wanted to use the above embedding for nearest neighbor search, there are two immediate challenges here:

- **Challenge 1**: In nearest neighbor search, the input is an arbitrary dataset $x_1, \ldots, x_n \in \mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, where each x_i is a subset of $(\mathbb{R}^d, \ell_p)$ of s vectors. The natural choice is $\Omega = \bigcup_{i=1}^n x_i$. The resulting (data-dependent) probabilistic tree $(\mathbf{T}_\Omega, d_{\mathbf{T}_\Omega})$, and composition of the maps (with an LSH for ℓ_1), would give an LSH family for $\mathsf{EMD}(\Omega, \ell_p)$. By construction, each $x_1, \ldots, x_n$ is inside $\mathsf{EMD}(\Omega, \ell_p)$, so dataset vectors can be hashed. However, the approximation increases to $\tilde{O}(\log(ns))$, which is far from the claimed $\tilde{O}(\log s)$-bound, and may be worse than the $O(\log^2 s)$ approximation of [6].
- **Challenge 2**: Even if we set Ω to all vectors used by $x_1, \ldots, x_n$, a crucial component of LSH involves applying the hash functions to the (unknown) query point. In particular, the data structure will hash the dataset during preprocessing, and in the future, a query comes (which was unknown during preprocessing) and needs to be hashed as well.

Warm-Up: Overcoming Challenge 2. We first show, as a warm-up and independent contribution, that the second challenge can be overcome by making [31] dynamic (Theorem 6 below, there is a reduction to d, Φ being $\mathrm{poly}(s)$). The data structure sets $\Omega = \bigcup_{i=1}^n x_i$, generates a tree embedding $(\mathbf{T}_\Omega, d_{\mathbf{T}_\Omega})$, and constructs a hash function to the dataset $x_1, \ldots, x_n$. Then, when a query point $y \in \mathsf{EMD}(\mathbb{R}^d, \ell_p)$ comes, we first *update* the tree to $(\mathbf{T}_{\Omega\cup y}, d_{\mathbf{T}_{\Omega\cup y}})$ (and corresponding hash functions) and identify the (few) dataset points x_i whose hash value changes. This allows the algorithm to maintain a view consistent with having preprocessed the dataset with the tree $(\mathbf{T}_{\Omega\cup y}, d_{\mathbf{T}_{\Omega\cup y}})$.

[2] Similarly to Footnote 1, it is especially important that the embeddings be efficiently stored and evaluated.

Theorem 2 (Dynamic and Data-Dependent Probabilistic Tree Embedding). *For a fixed $d \in \mathbb{N}$ and $p \in [1, 2]$, there is a data structure that maintains maintains a set $\Omega \subset [\Delta]^d$ of m vectors and an non-contracting embedding $\varphi : (\Omega, \ell_p) \to \mathbf{T}$, with expected distortion $\tilde{O}(\log(md\Delta))$ for any pair $x, y \in \Omega$. Moreover, it supports the following operations in expected time $O(d \log(d\Delta))$*

- **Query**: *Given a vector $x \in \Omega$, return the weighted path from the root of $\mathbf{T}$ to $\varphi(x)$*
- **Insertions/Deletions**: *Add or remove vectors from the set Ω, and also return the updated weighted paths of every vector $v \in \Omega$ whose path weights changed from the insertion/deletion.*

Main Task: Addressing Challenge 1. The above dynamic embedding still suffers a $\tilde{O}(\log(ns))$ distortion. We now proceed with the main technical component, of designing a data-dependent LSH for $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$. It turns out that for nearest neighbor search, it suffices to tailor (and relax) the second condition of LSH to an arbitrary fixed distribution (see Definition 3.2 and Theorem 5 for how data-dependent hashing implies nearest neighbor search). Specifically, a hash family $\mathcal{H}$ is (r, cr, p_1, p_2)-sensitive for a distribution μ supported on a metric (X, d_X) whenever:

(1) **Close Points Collide**: $\mathbf{Pr}_{\boldsymbol{h}\sim\mathcal{H}}[\boldsymbol{h}(x) = \boldsymbol{h}(y)] \geq p_1$ for every $x, y \in X$ with $d_X(x, y) \leq r$.
(2) **Far Points Separate on Average**: For any $x \in X$, the probability over $\boldsymbol{h} \sim \mathcal{H}$ and $\boldsymbol{y} \sim \mu$ that $\boldsymbol{h}(x) = \boldsymbol{h}(\boldsymbol{y})$ and $d_X(x, \boldsymbol{y}) \geq cr$ is at most p_2.

The only difference is the second condition (2) above, where one considers any $x \in X$ and ensures that a sampled point $\boldsymbol{y} \sim \mu$ far from x collides with probability at most p_2 (see Section 3, for comparison with [14]). It sufficient for us to prove the following theorem, which by a reduction from approximate near neighbors to data-dependent LSH (Theorem 5) implies Theorem 1 by setting p_2 to $1/10$ and $p_1 = 1 - \epsilon$.

Theorem 3 (Data-Dependent Hashing for EMD (Theorem 7 + Lemma 5.1)). *For any $s, d \in \mathbb{N}$, $p \in [1, 2]$, a threshold $r > 0$, and any $0 < p_2 < p_1 < 1$, there exists a data structure with the following guarantees:*

- **Preprocessing**: *The data structure receives sample access to a distribution μ supported on $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, and in time $\mathrm{poly}(sd/((1-p_1)p_2))$, initializes a draw $\boldsymbol{h}$ from a hash family $\mathcal{D}$ (which depends on μ) and is (r, cr, p_1, p_2)-sensitive for μ (see Definition 3.2), with*

$$c = \tilde{O}\left(\log s \cdot \frac{\log^2(1/p_2)}{1 - p_1}\right).$$

- **Query**: *Given any $q \in \mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, the data structure computes $\boldsymbol{h}(q)$ in time $\mathrm{poly}(sd)$.*

The above is our main technical theorem, and most of the work is devoted to that proof. Similarly to before, it will suffice via a simple reduction, to consider EMD over the hypercube $\{0, 1\}^d$ with ℓ_1 distance, where $d \leq \mathrm{poly}(s)$ and the threshold $r = \omega(s)$ (see Lemma 5.1). Then, the construction of the data-dependent hashing

scheme from Theorem 3 can be split into three parts, which we now describe.

Step 1: The SampleTree Embedding. We will use the data-dependent probabilistic trees of [31] on the union Ω of a small number of $m = \text{poly}(s)$ samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$ (i.e., $\Omega = \bigcup_{i=1}^m \boldsymbol{y}_i$, which is a subset of $s \cdot m$ vectors in $\{0,1\}^d$). Composing the data-dependent probabilistic tree $\mathbf{T}_\Omega$ (which we will refer to as $\mathbf{T}$) with the isometric embedding defines an embedding of $\mathsf{EMD}_s(\Omega)$ into ℓ_1, we aim to *extend* the embedding to the entire space $\mathsf{EMD}_s(\{0,1\}^d)$:[3]

$$\begin{array}{ccccc} \mathsf{EMD}_s(\Omega) & \xrightarrow[\tilde{O}(\log s)]{[31]} & \mathsf{EMD}_s(\mathbf{T}, d_\mathbf{T}) & \xrightarrow[\text{isometric}]{\text{folklore}} & \ell_1 \qquad (2) \\ \uparrow & & & & \uparrow \\ \mathsf{EMD}_s(\{0,1\}^d) & \xrightarrow{\text{desired new map in Section 7.1}} & & & \ell_1 \qquad (3) \end{array}$$

In the above diagram, (2) has expected $\tilde{O}(\log s)$-distortion from $\mathsf{EMD}_s(\Omega)$ to ℓ_1 from [31] on the samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$. We then define the extension (3) of (2), which is a natural "hybrid" of [31] and [6], that we call $\textsc{SampleTree}(\mu, m)$ in Section 7.1. It is not too difficult to show that the embedding (3) given by $\textsc{SampleTree}(\mu, m)$ is non-contracting with high-probability (Lemma 7.3).

The more subtle argument, however, is upper bounding the expansion. On the one hand, suppose $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$ are two arbitrary points, and all vectors in $x \cup y$ happened to be in Ω, then (3) inherits the $\tilde{O}(\log s)$ expected distortion from $(\mathbf{T}, d_\mathbf{T})$. On the other hand, if all vectors of $x \cup y$ are very far from Ω, then the root-to-leaf paths of vectors in x and y in $\mathbf{T}$ are mostly disjoint from those of Ω. This means $\mathsf{EMD}_\mathbf{T}(x, y)$ is effectively always using the data-independent weights, and similarly to the analysis of [6], incurs distortion $O(\log^2 s)$.

Step 2(a): Extensions on Chamfer Neighborhoods. Our notion of representation in $\mathsf{EMD}_s(\{0,1\}^d)$ will consider the *Chamfer Distance*, which is an (assymmetric) measure capturing dissimilarity of subsets in $\mathbb{R}^d$. Formally, given two subsets of vectors x, z in $\{0,1\}^d$, we use the Chamfer distance from x to z in $\{0,1\}^d$ with ℓ_1 distance,

$$\mathsf{Chamfer}(x, z) = \sum_{a \in x} \min_{b \in z} \|a - b\|_1.$$

Chamfer lower bounds $\mathsf{EMD}(\cdot,\cdot)$, since it relaxes the bijection condition $\pi : x \to z$, and is much simpler to reason about. In the context of the extension (3), it captures, for any point $x \in \mathsf{EMD}_s(\{0,1\}^d)$, how far x is from Ω (and from the data-dependent edge weights in $\textsc{SampleTree}(\mu, m)$), which leads to the following idea.

> **Key Idea 1**: We demonstrate that, with a last modification to $\textsc{SampleTree}(\mu, m)$, all points x, y in a Chamfer neighborhoods of radius $\mathsf{EMD}(x, y) \cdot \text{poly}(\log s)$ (for arbitrary constant power) around Ω still maintain a $\tilde{O}(\log s)$ expected distortion, and this will suffice for the remainder of the argument.

Specifically, in Lemma 7.2 (using Lemma 6.2), we argue that in $\textsc{SampleTree}(\mu, m)$, if in addition to taking m samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$ and letting $\Omega = \bigcup_{i=1}^m \boldsymbol{y}_i$, we let

$$\widehat{\Omega} = \textsc{Nbr}(\Omega) = \left\{ b' \in \{0,1\}^d : \exists b \in \Omega, \|b - b'\|_1 \le 1 \right\},$$

where $|\widehat{\Omega}| \le \text{poly}(s)$ (recall m is $\text{poly}(\log s)$ and d is $\text{poly}(s)$), and define data-dependent weights with respect to $\widehat{\Omega}$, then we have the crucial conclusion:

$$\mathop{\mathbb{E}}_{\mathbf{T}} [\mathsf{EMD}_\mathbf{T}(x, y)] \le \tilde{O}(\log s) \cdot \mathsf{EMD}(x, y) \left(1 + \log\left(\frac{\mathsf{Chamfer}(x, \Omega)}{\mathsf{EMD}(x, y)} + 1\right)\right). \quad (4)$$

Step 2(b): Locally Dense and non-Locally Dense Points. Given the analysis of $\textsc{SampleTree}(\mu, m)$, we may compose (3) with a LSH for ℓ_1 to obtain a hash family which always satisfies the "p_2-property" (i.e., far points separate) because the $\textsc{SampleTree}(\mu, m)$ is non-contracting, but only satisfies the "p_1-property" on close pairs points x, y where x is locally-dense with respect to μ (see Lemma 5.4). As mentioned, the important property of "locally-dense" is that, if we consider $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$ (where m is only $\text{poly}(\log s)$), then setting $\Omega = \bigcup_{i=1}^m \boldsymbol{y}_i$ satisfies $\mathsf{Chamfer}(x, \Omega) \le r \cdot \log^{10} s$ in expectation (we used 10 as an arbitrary setting of the $\text{poly}(\log s)$ to illustrate the point-to-come).

Now divide μ into two regions: the locally-dense points, and the remainder. The $\textsc{SampleTree}(\mu, m)$ embedding composed with an LSH for ℓ_1 handles the locally-dense region. The remaining region is handled by the following observation.

We consider a point x and sample from a (weak) data-independent LSH of [6], $\mathcal{H}$, which is $(r, \tilde{c}r, p_1, p_2)$-sensitive with $\tilde{c} = O(\log^2 s)$. Then, the "p_1-property" still holds for any pair of points x, y, since $\mathsf{EMD}(x, y) \le r$ implies that $\boldsymbol{h}(x) = \boldsymbol{h}(y)$ with probability at least p_1. Moreover, the "p_2-property" on points which are *not locally-dense for* μ follow from the following

> **Key Idea 2:** Suppose x is *not locally-dense* for μ. Then if we sample $\boldsymbol{y} \sim \mu$, the point $\boldsymbol{y}$ is likely to satisfy $\mathsf{EMD}(x, \boldsymbol{y}) \ge r \cdot \log^{10} s$; otherwise, taking $m - 1$ additional samples to define Ω (which includes $\boldsymbol{y}$) would satisfy $\mathsf{Chamfer}(x, \Omega) \le \mathsf{EMD}(x, \boldsymbol{y}) \le r \cdot \log^{10} s$. Thus if $\boldsymbol{y} \sim \mu$ satisfies $\mathsf{EMD}(x, \boldsymbol{y}) \ge r \cdot \log^{10} s$, then we can use the (weaker) data-independent LSH $\mathcal{H}$. Note that, $\mathsf{EMD}(x, \boldsymbol{y}) \ge \log^{10} s \cdot r$ is much larger than $\tilde{c}r$, so x and $\boldsymbol{y}$ collide in $\mathcal{H}$ with probability at most p_2, since $\log^{10} s \gg \tilde{c} = \log^2 s$.

In Section 5.2, we execute the above idea. We define a collection of (data-independent) LSH families which appear to be weak ($O(\log^2 s)$-approximation). These LSH families always satisfy the "p_1-property" (Lemma 6.1), but not a good "p_2-property." Then, we connect failure of the p_2-property on these LSH families to the expected Chamfer distance to a randomly sampled collection Ω. Namely, we consider a point $x \in \mathsf{EMD}_s(\{0,1\}^d)$, and we assume that the hash families from Lemma 6.1 fail to separate randomly sampled points from μ. In Section 5.2, we call these points "locally-dense" (Definition 5.3), and show in Lemma 6.2 that these are points whose expected Chamfer distance to Ω is at most $r \cdot \text{poly}(\log s)$.

Step 3: Gluing LSH for Locally-Dense and Non-Locally Dense Regions. The final step involves a "gluing" operation, which uses various hash families (for different regions of μ) to define a single data-dependent LSH family for all μ. Up to now, we have constructed:

[3] In both cases, $\mathsf{EMD}_s(\Omega)$ and $\mathsf{EMD}_s(\{0,1\}^d)$ refers to $\mathsf{EMD}_s(\Omega, \ell_1)$ and $\mathsf{EMD}_s(\{0,1\}^d, \ell_1)$, respectively. Furthermore, the map $\psi : \mathbb{R}^d \to \mathbf{T}$ is implicit in the notation, so we write $d_\mathbf{T}(a, b)$ for $d_\mathbf{T}(\psi(a), \psi(b))$ and $\mathsf{EMD}_\mathbf{T}(x, y)$ for $\mathsf{EMD}_\mathbf{T}(\psi(x), \psi(y))$.

- A hash family coming from SampleTree(μ, m), which always has a good "p_2-property," but only has a good "p_1-property" on points x which are locally-dense for μ.
- A collection of data-independent LSH families, $\mathcal{H}(\tau, \ell)$ for (fixed) threshold $\tau > 0$ and each $\ell \in \{0, \dots, L\}$ for $L = O(\log d)$ in Lemma 5.2. Here, the level ℓ corresponds to a level of the (data-independent) tree embedding, which is then embedded into ℓ_1, and thereafter hashed via a ℓ_1 LSH (see Definition 6.1 for full details).

In Section 5.3, we glue these hash families together, and obtain a data-dependent LSH which is (r, cr, p_1, p_2)-sensitive for μ (proving Theorem 3). The gluing proceeds as follows: for a fixed threshold $\tau > 0$ (which depends on the parameters r, p_1 and p_2 which we wish to obtain), we sample hash functions $\boldsymbol{h}_1, \dots, \boldsymbol{h}_L$ where $\boldsymbol{h}_\ell \sim \mathcal{H}(\tau, \ell)$ for each $\ell \in \{0, \dots, L\}$ and $L = O(\log d)$, as well as a hash function $\boldsymbol{h}_*$ resulting from SampleTree(μ, m). Importantly, the hash families $\mathcal{H}(\tau, \ell)$ are initialized to be $(r, \tilde{c}r, p_1/L, \tilde{p}_2)$-sensitive for an approximation $\tilde{c}$ (which will be a large poly$(\log s)$), and an appropriate value of $\tilde{p}_2$ for Step 2 to go through (i.e., failure of the "$\tilde{p}_2$-property" for $\boldsymbol{h}_1, \dots, \boldsymbol{h}_L$ implies a bounded Chamfer distance to Ω). Our final key observation is as follows:

> **Key Idea 3**: For a hash family $\mathcal{H}$, distribution μ, point x, and a draw $\boldsymbol{h} \sim \mathcal{H}$, the point x can check whether (a stronger version of) its own "p_2-property" holds given $\boldsymbol{h}$. In particular, one hashes the point $\boldsymbol{h}(x) = u$, and for the (now fixed) $\boldsymbol{h}$, one can computes the probability that $\boldsymbol{y} \sim \mu$ satisfies $\boldsymbol{h}(\boldsymbol{y}) = u$ by simply looking at the probability mass of points which hash to the bucket u (if μ is the uniform distribution, this is just proportional to the size of the hash bucket). If this probability mass is at most p_2, then the "p_2-property" necessarily holds for x conditioned on $\boldsymbol{h}$.

The above check is for a stronger "p_2-property", since we are not also checking whether $\boldsymbol{y} \sim \mu$ is far from x. Note that if this 'p_2-property" holds for some $\ell \in \{0, \dots, L\}$, then we can hash x to this bucket and make significant progress by reducing the size of the dataset. Given the above observation, the gluing proceeds by letting

$$\boldsymbol{h}(x) = (\boldsymbol{\ell}(x), \boldsymbol{h}_{\boldsymbol{\ell}(x)}(x)),$$

where $\boldsymbol{\ell}(x)$ is the smallest $\ell \in \{0, \dots, L\}$ where the above "p_2-property" check succeeds for x with the hash function $\boldsymbol{h}_\ell$. If it always fails, then $\boldsymbol{\ell}(x) = *$, thereby signifying that the hash output will be determinined by the output of the SampleTree data-dependent LSH. Since each $\mathcal{H}(\tau, \ell)$ collides close points with probability p_1/L, we can union bound over the L levels to ensure that a close pair of points collide in all L draws with probability at least p_1, thus $\boldsymbol{\ell}(x) = \boldsymbol{\ell}(y)$ for a close pair (x, y) with probability at least p_1. Using this, the "p_1-property" follows immediately whenever $\ell \neq *$. On the other hand, if $\ell = *$, this indicates a failure of the p_2 property for each of the data-independent families, which as we have shown implies a bounded Chamfer distance from x to a random sample Ω, which in turn implies that x is locally dense and therefore the p_1 property holds for x under the SampleTree LSH $\boldsymbol{h}_*$ (and thus holds for the full "glued" hash function). Finally, for the "p_2 property", if $\ell(x) \neq *$ then by definition of $\ell(x)$ we have split x from all but a p_2 fraction of μ, and otherwise the hash of x is determined by SampleTree, which always satisfies the desired "p_2 property". Putting together the above arguments will complete the proof of the Theorem 7.

2 PRELIMINARIES

Notation. For any integer $n \geq 1$, we write $[n] = \{1, 2, \dots, n\}$, and for two integers $a, b \in \mathbb{Z}$, write $[a : b] = \{a, a+1, \dots, b\}$. For $a, b \in \mathbb{R}$ and $\epsilon \in (0, 1)$, we use the notation $a = (1 \pm \epsilon)b$ to denote the containment of $a \in [(1-\epsilon)b, (1+\epsilon)b]$. We will use boldface symbols to represent random variables and functions, and non-boldface symbols for fixed values (potentially realizations of these random variables) for instance $\boldsymbol{f}$ vs, f.

We denote the metric space consisting of multi-sets of s points in a ground metric space (X, d), where the distance between sets is the Earth Mover's Distance metric, by $\mathsf{EMD}_s(X, d)$. When the distance over X is understood by context, we use $\mathsf{EMD}(x, y)$ to denote the distance function of $\mathsf{EMD}_s(\mathbb{R}^d, \ell_1)$ or $\mathsf{EMD}_s\{0, 1\}$.

Given a rooted tree $T = (V(T), E(T))$, all edges of T will be directed from parent to child, so an edge $(u, v) \in E(T)$ denotes an edge from the parent u to the child v. We will often abuse notation and write $u \in T$ to denote that $u \in V(T)$. Given a rooted tree with weighted edges $T = (V(T), E(T), W(T))$, we abuse the notation T to denote the tree metric $(V(T), d_T)$ where, for $u, v \in V(T)$, $d_T(a, b)$ is defined as the length of the shortest weighted path between u, v. We use EMD_T to denote the metric function of $\mathsf{EMD}_s(V(T), d_T)$.

Remark 4 (On Embedding ℓ_p into ℓ_1). *For the remainder of the paper, we will prove all our upper bounds for the case that the ground metric is $(\mathbb{R}^d, \ell_1)$. To extend this to general $p \in [1, 2]$, we can use well-known (data-independent) embeddings from $(\mathbb{R}^d, \ell_p)$ into $(\mathbb{R}^{d'}, \ell_1)$ with $d' = O(d)$ [43], which preserve all distances up to $(1 \pm \epsilon)$ for any arbitrary constant $\epsilon > 0$. This embedding will only increase the runtime by a multiplicative factor of $O(d)$, and increase the space by an additive $O(d^2)$.*

3 NEAREST NEIGHBORS, EMBEDDINGS, AND DATA-DEPENDENT HASHING

In this section, we define the approximate *near* neighbor search problem and data-dependent hashing.

Definition 3.1 (Approximate Near Neighbor). *Let (X, d_X) be a metric space, $r > 0$ be a threshold, and $c > 1$ be an approximation. By a standard reduction (see [38]), it will suffice to solve the (c, r)-approximate near neighbor problem:*

- **Preprocessing**: *We receive a dataset $P \subset X$ of n points to preprocess into a data structure.*
- **Query**: *A query is specified by any point $q \in X$, and a query is* correct *whenever the following occurs. If there exists a point $p \in P$ with $d_X(p, q) \leq r$, the data structure outputs a point $\widehat{p} \in P$ with $d_X(\widehat{p}, q) \leq cr$.*

A data structure solves the (c, r)-approximate near neighbor problem if, for every (fixed) dataset $P \subset X$ and query $q \in X$, following preprocessing of P, the data structure answers correctly on q with probability at least $9/10$ over the construction of the data structure.

We remark that he definition of data-dependent hashing (Definition 3.2) that we obtain in this paper is slightly more stringent

than the one presented in [15]. The one subtlety is that, because our hashing family *depends* on the dataset, one must instantiate it to the desired dataset before using it.

Definition 3.2 (Data-Dependent Hashing). *For a metric (X, d_X), a distribution μ over X, and a threshold $r > 0$, we say that a distribution $\mathcal{D}$ over maps $h : X \to U$ is (r, cr, p_1, p_2)-sensitive for distribution μ if*

- **Close Points Collide**: *For any two points $x, y \in X$ with $d_X(x, y) \le r$, we have*
$$\Pr_{\boldsymbol{h}\sim\mathcal{D}}[\boldsymbol{h}(x) = \boldsymbol{h}(y)] \ge p_1.$$
- **Far Points Separate on Average**: *For any point $x \in X$, we have*
$$\Pr_{\substack{\boldsymbol{h}\sim\mathcal{D}\\ \boldsymbol{y}\sim\mu}}\left[\begin{array}{c} d_X(x,\boldsymbol{y}) > c\cdot r,\\ \boldsymbol{h}(x) = \boldsymbol{h}(\boldsymbol{y})\end{array}\right] \le p_2.$$

Definition 3.3 (Data Structure for Data-Dependent Hashing). *For a metric (X, d_X), a data structure for data-dependent hashing with a (r, cr, p_1, p_2)-sensitive family satisfies:*

- **Preprocessing**: *The data structure preprocesses the description of a distribution μ supported on X, and maintains a draw of $\boldsymbol{h}$ from a (r, cr, p_1, p_2)-sensitive family for μ.*
- **Query**: *Given any point $q \in X$, the data structure outputs the value of $\boldsymbol{h}(q)$.*

We let $I_{\mathrm{h}}(n)$ denote the time of instantiating the data structure with a distribution μ supported on n points, and let $Q_{\mathrm{h}}(n)$ denote the worst-case query time.

Theorem 5 (Data-Dependent Hashing to Approximate Near Neighbors). *Let (X, d_X) be a metric, $r > 0$ be a threshold, $c > 1$ be an approximation, and $p_1, p_2 \in (0, 1)$ be two parameters, where $\rho \in \mathbb{R}$ is the parameter*
$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)}.$$
Suppose there is a data structure for data-dependent hashing with a (r, cr, p_1, p_2)-sensitive family with preprocessing time $I_{\mathrm{h}}(n)$ and query time $Q_{\mathrm{h}}(n)$. Then, there exists a data structure for the (c, r)-approximate near neighbor problem which satisfies:

- **Preprocessing Time**: *The data structure preprocesses a size-n dataset in time at most*
$$O\left(n^{\rho}/p_1 \cdot \log_{1/p_2} n \cdot (I_{\mathrm{h}}(n) + n \cdot Q_{\mathrm{h}}(n))\right),$$
and therefore its space complexity is at most that amount.
- **Query Time**: *A query is answered in time at most*
$$O\left(n^{\rho}/p_1 \cdot \log_{1/p_2} n \cdot Q_{\mathrm{h}}(n)\right).$$

4 DYNAMIC AND DATA-DEPENDENT PROBABILISTIC TREE EMBEDDINGS

In this Section, we describe the dynamic, data-dependent probabilistic tree embedding from Theorem 2. For simplicity in this Section, we consider vectors which have integer coordinates $x \in [\Delta]^d = \{1, 2, \ldots, \Delta\}^d$.

Theorem 6 (Dynamic and Data-Dependent Probabilistic Tree Embedding). *For a fixed $d \in \mathbb{N}$ and $p \in [1, 2]$, there is a data structure supporting the following:*

- **Maintenance**: *The data structure maintains a set $\Omega \subset [\Delta]^d$ of m vectors, as well as a rooted probabilistic tree metric $\mathbf{T}$ (whose distribution depends on Ω), along with a non-contracting embedding $\varphi : (\Omega, \ell_p) \to \mathbf{T}$, such that for any $x, y \in \Omega$:*
$$\mathop{\mathbb{E}}_{\mathbf{T}}[d_{\mathbf{T}}(\varphi(x), \varphi(y))] \le \tilde{O}(\log(md\Delta)) \cdot \|x - y\|_p.$$
- **Query**: *In time $O(d \log(d\Delta))$, we may query a vector $x \in \Omega$ and obtain the weighted path from the root to $\varphi(x)$ in $\mathbf{T}$.*
- **Insertions/Deletions**: *In $O(d \log(d\Delta) + \log^2(d\Delta))$ expected time, the algorithm can add or remove vectors from the set Ω, and returns the updated weighted paths of every vector in Ω whose path changed from the insertion/deletion.*

We show the data structure for Theorem 6 for the Hamming cube $\{0, 1\}^d$ with ℓ_1 metric (where we note that this sets $\Delta = 2$). This proof will already contain the major ideas, and subsequent sections will utilize the main definition of the QuadTree sub-routine specified below. It can be easily generalized to $([\Delta]^d, \ell_p)$ for $p \in [1, 2]$. See the full version for the analysis of the data structure and generalization to $([\Delta]^d, \ell_p)$.

We consider a subset $\Omega \subset \{0, 1\}^d$ of n vectors in the Hamming cube. For any (multi-)set of indices $\vec{i} = (i_1, i_2, \ldots, i_t) \in [d]^t$, define the projection $p_{\vec{i}} : \{0, 1\}^d \to \{0, 1\}^t$ which maps a vector $x \in \{0, 1\}^d$ to $p_{\vec{i}}(x) = (x_{i_1}, x_{i_2}, \ldots, x_{i_t})$. For any $t \in \mathbb{N}$, we consider the hash family $\mathcal{H}_{t,d}$ given by
$$\mathcal{H}_{t,d} = \{p_{\vec{i}} \,:\, \vec{i} \in [d]^t\}. \tag{5}$$

The construction of the (static) data-dependent probabilistic tree metric $\mathbf{T}$ is described by the algorithm QuadTree (in Figure 1). We also allow QuadTree to take an additional scaling parameter ξ, which is not needed here and will be used in Section 7.

Definition 4.1. *For any subset $\Omega \subset \{0, 1\}^d$ and any draw of $\mathbf{T}$ generated from an execution of QuadTree(Ω) (in Figure 1), we have the following:*

- *For every vector $x \in \{0, 1\}^d$, there is a unique root-to-leaf path in $\mathbf{T}$, given by the sequence of nodes $v_0(x), \ldots, v_{L+1}(x)$, inductively defined by $v_0(x) = v_0$ and*
$$v_\ell(x) \text{ is unique child } v_u \text{ of } v_{\ell-1}(x) \text{ with } x \in \textbf{\textit{Elms}}(v_u).$$
- *The mapping $\varphi : \{0, 1\}^d \to \mathbf{T}$ sends $x \in \{0, 1\}^d$ to $v_{L+1}(x)$, and since the path for each $x \in \{0, 1\}^d$ is unique, we abuse notation and associate $x \in \{0, 1\}^d$ with its leaf $x = v_{L+1}(x) \in \mathbf{T}$.*
- *The tree metric $(\mathbf{T}, d_{\mathbf{T}})$ is specified by the edge weights in $\boldsymbol{w}(\cdot, \cdot)$, and for any $x, y \in \{0, 1\}^d$, the distance $d_{\mathbf{T}}(\varphi(x), \varphi(y))$ is the sum of edge-weights $\boldsymbol{w}$ on the path from $\varphi(x)$ to $\varphi(y)$ in $\mathbf{T}$.*

Whenever we generate $\mathbf{T}$ from QuadTree(Ω) and we consider $x, y \in \Omega$, the edge weight $\boldsymbol{w}(v_{\ell-1}(x), v_\ell(x))$ always falls into the first case in Step 4, —we will call these edges "data-dependent". When one of x or y is not in Ω, then at least one edge along the path $\varphi(x)$ to $\varphi(y)$ in $\mathbf{T}$ falls in the second case of Step 4 and has $\boldsymbol{w}(\cdot, \cdot)$ set to $d/2^\ell \cdot \xi$—we will call these edges "data-independent."

[4] As noted earlier, the symbol ξ in the description of $\boldsymbol{w}(v, v_u)$ can be arbitrary in this section. Specifically, in this section, an edge of whose weight depends on ξ will never be evaluated. Looking ahead, we have placed the parameter ξ as it will become important in Section 7, where ξ will be set to $O(\log s)$.

Subroutine $\textsc{QuadTree}(\Omega, \xi)$

Input: A subset of vectors $\Omega \subset \{0,1\}^d$, and a scaling parameter ξ (if unspecified, set $\xi = 1$).
Output: A probabilistic weighted tree $\mathbf{T}$.

(1) Initialize a root node v_0 at depth 0. We will let $L = O(\log d)$ (for a large enough constant, say 2) denote the depth of the tree, and we define the notation which will indicate, for a node v, let $\textbf{Elms}(v)$ be the subset of $\{0,1\}^d$ which will embed into the subtree at v. Initially, $\textbf{Elms}(v_0) = \{0,1\}^d$.
(2) For each $\ell = 0, 1, \ldots, L-1$, sample a random hash function $\boldsymbol{\phi}_\ell \sim \mathcal{H}_{2^\ell}$, and let $\boldsymbol{\phi}_L : \{0,1\}^d \to \{0,1\}^d$ be the identity mapping $\boldsymbol{\phi}(x) = x$.
(3) For each $\ell = 0, \ldots, L$:
   - For every node v at depth ℓ, and every $u \in \{0,1\}^{2^\ell}$, we initialize a child node v_u to v (at depth $\ell+1$). We create the edge (v, v_u), and set
   $$\textbf{Elms}(v_u) = \textbf{Elms}(v) \cap \{x \in \{0,1\}^d \mid \boldsymbol{\phi}_\ell(x) = u\}.$$
   The nodes at depths $L+1$ are leaves of $\mathbf{T}$.
(4) For every edge $(v, v_u) \in \mathbf{T}$ where v is at depth ℓ, we assign the weight
$$\boldsymbol{w}(v, v_u) = \begin{cases} \mathop{\mathbb{E}}_{\substack{\boldsymbol{c}\sim\textbf{Elms}(v)\cap\Omega \\ \boldsymbol{c}'\sim\textbf{Elms}(v_u)\cap\Omega}} [\|\boldsymbol{c} - \boldsymbol{c}'\|_1] & \textbf{Elms}(v_u) \cap \Omega \neq \emptyset \\ d/2^\ell \cdot \xi & \text{otherwise.}[4] \end{cases}$$

Figure 1: The Data-Dependent QuadTree Embedding.

Fact 4.2 (Distances in $\mathbf{T}$ from $\textsc{QuadTree}(\Omega)$). *Let $\Omega \subset \{0,1\}^d$ be any subset and let $\mathbf{T}$ be drawn from $\textsc{QuadTree}(\Omega)$. For any $x, y \in \{0,1\}^d$ and $\ell \in \{0, \ldots, L+1\}$, we let $\textsc{Split}_\ell(x,y)$ denote the indicator variable*
$$\textsc{Split}_\ell(x,y) = \mathbf{1}\{v_\ell(x) \neq v_\ell(y)\},$$
and note that we may write
$$d_{\mathbf{T}'}(x,y) = \sum_{\ell=1}^{L+1} \textsc{Split}_\ell(x,y) \cdot \left(w_x + w_y\right) \tag{6}$$
where $w_x = \boldsymbol{w}(v_{\ell-1}(x), v_\ell(x))$, $w_y = \boldsymbol{w}(v_{\ell-1}(y), v_\ell(y))$.

We use the following lemma from [31].

Lemma 4.3 (Follows from Lemma 3.6 (with $i_0 = 0$) and Lemma 3.4 from [31]). *For any set $\Omega \subseteq \{0,1\}^d$ of m vectors, and any two $a, b \in \Omega$, we have that, whenever $\mathbf{T}$ is generated from $\textsc{QuadTree}(\Omega)$, we have*
$$\mathop{\mathbb{E}}_{\mathbf{T}}[d_{\mathbf{T}}(a,b)] \leq \tilde{O}(\log(m) + \log(d)) \cdot \|a - b\|_1$$
Moreover, we have $d_{\mathbf{T}}(a,b) \geq \|a-b\|_1$ deterministically.

Note that Lemma 4.3 immediately implies that a draw of $\mathbf{T}$ from $\textsc{QuadTree}(\Omega)$ satisfies the desired distortion guarantees.

Definition 4.4. *For any set $\Omega \subset \{0,1\}^d$, let $\mathbf{T}$ be generated from an execution of $\textsc{QuadTree}(\Omega)$. We let $\mathbf{T}'$ denote the tree metric whose vertex set, edge set, and mapping $\varphi : \{0,1\}^d \to \mathbf{T}$ is the same as in $\mathbf{T}$; however, we modify the weights as follows:*

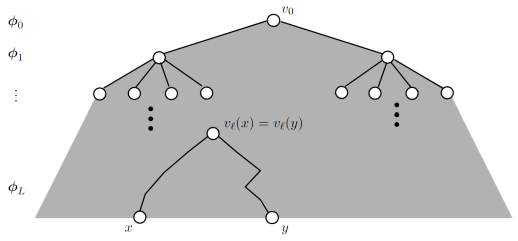

Figure 2: Tree Embedding $\mathbf{T}$ Sampled from $\textsc{QuadTree}(\Omega)$. The root node is v_0 and the tree is generated by the maps $\boldsymbol{\phi}_0, \ldots, \boldsymbol{\phi}_L$. Displayed are two vectors x, y which map to the leaves of the tree, whose lowest common ancestor is $v_\ell(x) = v_\ell(y)$.

- *For each node $v \in \mathbf{T}'$, if $\textbf{Elms}(v) \cap \Omega \neq \emptyset$, we sample what we call a representative $\textbf{Rep}(v) \sim \textbf{Elms}(v_u) \cap \Omega$.*
- *For each edge $(v, v_u) \in \mathbf{T}'$ where v is at depth ℓ, we let*

$$\boldsymbol{w}'(v, v_u) = \begin{cases} \|\textbf{Rep}(v) - \textbf{Rep}(v_u)\|_1 & \textbf{Elms}(v_u) \cap \Omega \neq \emptyset \\ d/2^\ell \cdot \xi & \text{otherwise} \end{cases}.$$

We similarly consider the tree metric $(\mathbf{T}', d_{\mathbf{T}'})$, and we have
$$d_{\mathbf{T}'}(x,y) = \sum_{\ell=1}^{L+1} \textsc{Split}_\ell(x,y) \cdot \left(w'_x + w'_y\right) \tag{7}$$
where $w'_x = \boldsymbol{w}'(v_{\ell-1}(x), v_\ell(x))$, $w'_y = \boldsymbol{w}'(v_{\ell-1}(y), v_\ell(y))$.

Data Structure for Dynamic, Data-Dependent Probabilistic Trees. We can now describe the data structure which maintains the tree $\mathbf{T}'$, which samples $\mathbf{T}$ from $\textsc{QuadTree}(\Omega)$ and uses the modified edge weights in Definition 4.4. The data structure will maintain the following information:

- We store the sampled functions $\boldsymbol{\phi}_0, \ldots, \boldsymbol{\phi}_L$ (by storing the set of indices sampled for each $\ell \in \{0, \ldots, L\}$), and note that it suffices to store the *set* of indices samples, which has size at most d always. This takes time $O(Ld)$ during initialization.
- We also maintain the set Ω, as well as the subtree of $\mathbf{T}'$ of nodes v for which $\textbf{Elms}(v) \cap \Omega$ is non-empty. For each such node v, we maintain the set $\textbf{Elms}(v) \cap \Omega$, as well as the sample $\textbf{Rep}(v) \sim \textbf{Elms}(v) \cap \Omega$.

This completes the description of the data structure. Note that, each vector $x \in \Omega$ is naturally mapped to a leaf. Then, when updating the set Ω by inserting or deleting a vector $x \in \{0,1\}^d$, we proceed by:

- **Insertion:** We insert x to Ω and find the leaf $\varphi(x)$, considering the root-to-leaf path given by nodes $v_0(x), \ldots, v_{L+1}(x)$, where one may need to initialize new nodes if $v_\ell(x)$ was not stored in the stored subtree. For each node $v = v_\ell(x)$, with probability $1/|\textbf{Elms}(v) \cap \Omega|$ (note that Ω now includes one more vector), we update $\textbf{Rep}(v)$ to x; otherwise, do not update $\textbf{Rep}(v)$. If the data structure updates $\textbf{Rep}(v)$, every vector in $\textbf{Elms}(v) \cap \Omega$ has its weighted path modified and its change is reported.
- **Deletion:** We delete x to Ω and find the leaf $\varphi(x)$, considering the root-to-leaf path given by nodes $v_0(x), \ldots, v_{L+1}(x)$, where one may need to initialize new nodes if $v_\ell(x)$ was

not stored in the stored subtree. For each node $v = v_\ell(x)$, if $\textbf{Rep}(v) = x$, we update $\textbf{Rep}(v)$ by re-sampling from $\textbf{Elms}(v) \cap \Omega$ or removing v if empty. If the data structure updates $\textbf{Rep}(v)$, every vector in $\textbf{Elms}(v) \cap \Omega$ has its weighted path modified and its change is reported.

We claim the above data structure leads to the proof of Theorem 6. We refer the reader to the full version of this paper for the analysis.

5 LOCALITY SENSITIVE HASH FAMILY FOR EMD

We will proceed by **(i)** reducing data-dependent hashing over $(\mathbb{R}^d, \ell_p)$ to that of the hypercube $\{0,1\}^d$, and then **(ii)** giving a data-dependent hashing scheme for EMD over the hypercube.

5.1 Reduction to Data-Dependent LSH over the Hypercube

By Remark 4, it suffices to consider data-dependent hashing for EMD over $(\mathbb{R}^d, \ell_1)$. In the following Lemma, we reduce the problem further to data-depedent hashing for EMD over the Hamming Cube $\{0,1\}^t$.

Lemma 5.1 (Reduction to the Data-Dependent Hashing on $\mathsf{EMD}_s(\{0,1\}^t)$). *For any parameters $s, \tau \geq 0$ and $c > 3$, $\delta \in (0,1)$:*

- *Suppose that there exists a data structure for data-dependent hashing over $\mathsf{EMD}_s(\{0,1\}^t)$ which is $(r, cr/3, p_1, p_2)$-sensitive for the parameter settings*

$$t = \Theta(s^2 c^2 \log(1/\delta)) \qquad \text{and} \qquad r = \frac{t}{1.99c} \geq \omega(s),$$

which has initialization time $I_{\mathsf{h}}(n)$ and query time $Q_{\mathsf{h}}(n)$.
- *Then, there exists a data structure for data-dependent hashing over $\mathsf{EMD}_s(\mathbb{R}^d, \ell_1)$ which is $(\tau, c\tau, p_1 - \delta, p_2 + \delta)$-sensitive with initialization time $I_{\mathsf{h}}(n) + n \cdot \mathrm{poly}(sd)$ and query time $Q_{\mathsf{h}}(n) + \mathrm{poly}(sd)$.*

5.2 Three Crucial Ingredients for LSH for EMD over the Hypercube

We will refer to "points" as the size-s tuples of vectors in $\{0,1\}^d$, and "elements" to the points in $\{0,1\}^d$ which will be in the tuples. We will need three ingredients

(1) The first ingredient is Lemma 5.2, which specifies a sequence of hash families whose hash functions maps points (i.e., size-s tuples of $\{0,1\}^d$) to buckets. These hash families are parametrized by a so-called "level" ℓ, and we will let ℓ vary among L possible levels,[5] for $L = \Theta(\log d)$. As we will see, these data-independent hash families have a good "p_1"-property.
(2) The second ingredient is Definition 5.3, a point being "locally-dense" with respect to a distribution μ over points. We define "local-density" as all hash families defined in Lemma 5.2 failing to have the "p_2"-property; however, the important consequence will be "many" of its elements a_i have a non-trivial fraction of "nearby" elements from points in μ
(3) The final ingredient will be the data-dependent hash family which fills in the gap. For all points, the data-dependent hash family always has the desired "p_2"-property, but it may not have the "p_1"-property. However, we will prove that the data-dependent hash family has the property "p_1"-property whenever a point is locally-dense.

We now formally state the lemmas and definitions which capture the three ingredients.

Lemma 5.2. *For any parameter $\ell \in \{0, \ldots, L\}$ and any $\tau > 0$, we define a hash family $\mathcal{H}(\tau, \ell)$ (in Definition 6.1). The hash family $\mathcal{H}(\tau, \ell)$ satisfies that, for any two $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$,*

$$\Pr_{\boldsymbol{h} \sim \mathcal{H}(\tau,\ell)} [\boldsymbol{h}(a) \neq \boldsymbol{h}(b)] \leq \frac{\mathsf{EMD}(x,y)}{\tau}.$$

In addition, there is a data structure which maintains a draw of $\boldsymbol{h} \sim \mathcal{H}(\tau, \ell)$ while supporting queries of $\boldsymbol{h}(x)$ in initialization and query time $O(sd)$.

Definition 5.3. *Let μ denote a distribution supported on $\mathsf{EMD}_s(\{0,1\}^d)$. For parameters $\alpha, \tau > 0$, we say that a point $x \in \mathsf{EMD}_s(\{0,1\}^d)$ is (α, τ)-locally-dense with respect to μ if for all $\ell \in \{0, \ldots, L\}$,*

$$\Pr_{\substack{\boldsymbol{u} \sim \mu \\ \boldsymbol{h} \sim \mathcal{H}(\tau,\ell)}} [\boldsymbol{h}(x) = \boldsymbol{h}(\boldsymbol{u})] \geq \alpha.$$

Lemma 5.4. *Let μ be a distribution supported on $\mathsf{EMD}_s(\{0,1\}^d)$, and fix any $\alpha, \tau > 0$. Then for any $\gamma > 0$ and $\delta \in (0,1)$, there exists a hash family $\mathcal{H}(\mu, \tau, \gamma, \delta)$ with the following properties:*

- **Close Points Collide**: *Let $\boldsymbol{h} \sim \mathcal{H}(\mu, \tau, \gamma, \delta)$. For any points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$, if x is (α, τ)-locally-dense, then*

$$\Pr_{\boldsymbol{h}} [\boldsymbol{h}(x) \neq \boldsymbol{h}(y)] \leq \lambda \cdot \frac{\mathsf{EMD}(x,y)}{\gamma} \left(1 + \log\left(\frac{\tau + s}{\mathsf{EMD}(x,y)} + 1\right)\right)$$

Where $\lambda = C_1 \log\left(\frac{sd}{\delta\alpha}\right) \left(\log\log\left(\frac{sd}{\delta\alpha}\right)\right)^{C_2}$ for absolute constants C_1, C_2.
- **Far Points Separate**: *Let $\boldsymbol{h} \sim \mathcal{H}(\mu, \tau, \gamma, \delta)$. For any points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$,*

$$\Pr_{\boldsymbol{h}} [\boldsymbol{h}(x) = \boldsymbol{h}(y)] \leq \exp\left(-\frac{\mathsf{EMD}(x,y)}{\gamma}\right) + \delta.$$

In addition, there is a data structure which maintains a draw $\boldsymbol{h} \sim \mathcal{H}(\mu, \tau, \gamma, \delta)$ while supporting queries of $\boldsymbol{h}(x)$ which has initialization time $n \cdot \mathrm{poly}(sd/\alpha)$ (where μ is supported on n points) and query time $\mathrm{poly}(sd)$.

5.2.1 Main Theorems for Data Dependent Hashing and Nearest Neighbor Search. With the above ingredients set in place, we are ready to state the data-dependent hash family for EMD. The remainder of the section is devoted to proving the main theorem below.

Theorem 7 (Data-Dependent Hashing for EMD). *Fix any $0 < p_2 < p_1 < 1$. There exists a data structure for a (r, cr, p_1, p_2)-sensitive data-dependent hashing family for $\mathsf{EMD}_s(\{0,1\}^d)$ where $r > s$ for an approximation $c > 1$ which is*

$$c = \tilde{O}\left(\frac{1}{1 - p_1} \cdot \log\left(\frac{1}{p_2}\right) \cdot \log\left(\frac{sd}{p_2}\right)\right)$$

[5] These levels will correspond to the depth of the quadtree embeddings.

The data structure has initialization time $I_{\mathsf{h}}(n) \le n \cdot \mathrm{poly}(sd/((1-p_1)p_2))$ *and query time* $\mathrm{poly}(sd)$.

Our main result, for Data-Dependent LSH for $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ for any $p \in [1,2]$ (Theorem 3) follows from combining Theorem 7 with Lemma 5.1. Setting $p_1 = 1-\epsilon$ and $p_2 = \Theta(1)$ in Theorem 7, and then applying Theorem 5, we obtain our main result on nearest neighbor search under the Earth Mover's Distance.

Theorem 8 (Approximate Nearest Neighbor Search for EMD). *For any* $s, d \in \mathbb{N}$, $p \in [1,2]$ *and* $\epsilon \in (0,1)$, *there exists a data structure for approximate nearest neighbor search over* $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ *with approximation* $c = \tilde{O}(\frac{\log s}{\epsilon})$ *satisfying the following guarantees:*

- **Preprocessing Time**: *The data structure preprocesses a dataset* P *of* n *points in* $\mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$ *in time* $n^{1+\epsilon} \cdot \mathrm{poly}(sd\epsilon^{-1})$.
- **Query Time**: *For a vector* $q \in \mathsf{EMD}_s(\mathbb{R}^d, \ell_p)$, *we output a* c*-approximate nearest neighbor of* q *in* P *in time* $n^{\epsilon} \cdot \mathrm{poly}(sd)$.

5.3 The Hash Family $\mathcal{D}$ in Theorem 7

Now that we have stated all of the preliminary ingredients, we show how to construct the data-dependent hash family $\mathcal{D}$ stated in Theorem 7. Check the full version for the analysis. Let $\lambda = \tilde{O}\left(\log(\frac{sd}{\delta\alpha})\right)$ be the parameter defined in Lemma 5.4. We now instantiate the following parameters

$$\tau = \frac{4\cdot(L+1)\cdot r}{1-p_1}, \quad \alpha = \frac{(1-p_1)\cdot p_2}{6}, \quad c = \log\left(\frac{3}{p_2}\right)\cdot\frac{\gamma}{r},$$

$$\gamma = \frac{\left(\lambda \log\left(\frac{20(L+1)}{1-p_1}\right) + \log(\frac{1}{1-p_1})\right)\cdot\tau}{L+1}, \quad \delta = \frac{p_2}{3}$$

To sample a hash function $\boldsymbol{h} \sim \mathcal{D}$, we first sample a hash functions $\boldsymbol{h}_0, \dots, \boldsymbol{h}_L$, where $\boldsymbol{h}_\ell \sim \mathcal{H}(\tau,\ell)$ for each $\ell \in \{0,\dots,L\}$, and we next sample a hash function $\boldsymbol{h}_* \sim \mathcal{H}(\mu,\tau,\gamma,\delta)$. In order to evaluate $\boldsymbol{h}$ on a point $z \in \mathsf{EMD}_s(\{0,1\}^d)$, we first check whether there exists an index $\ell \in \{0,\dots,L\}$ for which

$$\Pr_{\boldsymbol{u}\sim\mu}\left[\boldsymbol{h}_\ell(z) = \boldsymbol{h}_\ell(\boldsymbol{u})\right] \le \frac{p_2}{3}. \tag{8}$$

If so, then we define $\boldsymbol{\ell}(z)$ to be the smallest index $\ell \in \{0,\dots,L\}$ where (8) holds. If no such index exists, we set $\boldsymbol{\ell}(z) = *$. The final hash function $\boldsymbol{h} \sim \mathcal{D}$ then evaluates:

$$\boldsymbol{h}(z) = \left(\boldsymbol{\ell}(z), \boldsymbol{h}_{\boldsymbol{\ell}(z)}(z)\right).$$

6 INGREDIENTS 1 AND 2: THE HASH FAMILY $\mathcal{H}(\tau,\ell)$ AND LOCALLY-DENSE POINTS

In this section, we give the first ingredient and prove Lemma 5.2. We will first define the hash family $\mathcal{H}(\tau,\ell)$, and derive the main consequence of locally-dense points.

6.1 Hash Family $\mathcal{H}(\tau,\ell)$

As in Subsection 5.2, the term "points" is used to denote size-s tuples of vectors in $\{0,1\}^d$. Each of the s vectors in $\{0,1\}^d$ is referred to as an "element" of the point. We let $L = O(\log d)$, and we will refer to $\ell \in \{0,\dots,L\}$ as the "levels."

Definition 6.1 (The Hash Family $\mathcal{H}(\tau,\ell)$). *For* $\tau > 0$ *and* $\ell \in \{0,\dots,L\}$, *the hash family* $\mathcal{H}(\tau,\ell)$ *is specified by the following sampling procedure. A draw of a hash function* $\boldsymbol{h} \sim \mathcal{H}(\tau,\ell)$ *proceeds by:*

1. *First, we sample* $\boldsymbol{\phi} \sim \mathcal{H}_{2^\ell}$ *(as in Section 4) by sampling* 2^ℓ *coordinates* $\boldsymbol{i}_1, \dots, \boldsymbol{i}_{2^\ell} \sim [d]$ *and letting* $\boldsymbol{\phi}\colon \{0,1\}^d \to \{0,1\}^{2^\ell}$ *be*
$$\boldsymbol{\phi}(a) = (a_{\boldsymbol{i}_1}, a_{\boldsymbol{i}_2}, \dots, a_{\boldsymbol{i}_{2^\ell}}) \in \{0,1\}^{2^\ell}.$$
2. *Then, for each* $u \in \{0,1\}^{2^\ell}$ *and each* $k \in [s]$, *we let* $\mathbf{C}_{u,k} \sim \mathrm{Ber}(d/(\tau 2^{\ell+1}))$.
3. *For a point* $x \in \mathsf{EMD}_s(\{0,1\}^d)$, *and* $u \in \{0,1\}^{2^\ell}$ *and* $k \in [s]$, *we let* $\boldsymbol{\chi}(x,u,k) \in \{0,1\}$ *be*
$$\boldsymbol{\chi}(x,u,k) = \mathbf{1}\{\text{at least } k \text{ elements } a \in x \text{ satisfy } \boldsymbol{\phi}(a) = u\}.$$

With those definitions, we let

$$\boldsymbol{h}(x) = \left(\mathbf{C}_{u,k}\cdot\boldsymbol{\chi}(x,u,k) : u \in \{0,1\}^{2^\ell}, k \in [s]\right) \in \{0,1\}^{\{0,1\}^{2^\ell}\times[s]}.$$

Data Structure Guarantees for $\boldsymbol{h} \sim \mathcal{H}(\tau,\ell)$. It is important to note that, for each $x \in \mathsf{EMD}_s(\{0,1\}^d)$, we may compute $\boldsymbol{h}(x)$ in time $O(sd)$. This is because the vectors $\boldsymbol{h}(x)$ have at most s non-zero coordinates. We may identify the at-most-s non-zero entries of $\boldsymbol{\chi}(x,u,k)$ in $O(sd)$ time, and we can generate and store the corresponding Bernoulli random variables $\mathbf{C}_{u,k}$ with a constant-time overhead per access.

We leave the proof of correctness, which will complete the proof of Lemma 5.2, to the full version.

6.2 Locally-Dense Points

In this section, we give the main consequence of locally-dense points, which will become a crucial ingredient in Lemma 5.4. We will let μ denote a distribution over points in $\mathsf{EMD}_s(\{0,1\}^d)$ and refer to the hash families $\mathcal{H}(\tau,\ell)$ defined in Definition 6.1.

Lemma 6.2. *Let* μ *denote a distribution on* $\mathsf{EMD}_s(\{0,1\}^d)$. *If, for parameters* $\alpha, \tau > 0$, *a point* $x \in \mathsf{EMD}_s(\{0,1\}^d)$ *is* (α,τ)*-locally dense with respect to* μ, *then as long as* $m = \omega(\log(sd)/\alpha)$,

$$\mathop{\mathbb{E}}_{\boldsymbol{y}_1,\dots,\boldsymbol{y}_m\sim\mu}\left[\mathsf{Chamfer}\left(x, \bigcup_{i=1}^m \boldsymbol{y}_i\right)\right] \le (\tau+s)\cdot\mathrm{polylog}(sd/\alpha).$$

7 INGREDIENT 3: SampleTree

In this section, we sketch the proof of Lemma 5.4, which gives the final ingredient of the data-dependent hashing scheme for $\mathsf{EMD}_s(\{0,1\}^d)$, and leads us to Theorem 7.

7.1 The SampleTree Embedding and Hash Family Construction

In this section, we specify the construction of the hash family $\mathcal{H}(\mu,\tau,\gamma,\delta)$. We will do so by first specifying the SampleTree embedding, and then concatenating it with a locality-sensitive hash function in ℓ_1. In particular, we first describe an algorithm, SampleTree, which takes as input a distribution μ supported on $\mathsf{EMD}_s(\{0,1\}^d)$ and a parameter m (which, as per Lemma 6.2, will be set to $\omega(\log(sd)/\alpha)$), and outputs a weighted tree $\mathbf{T}$ from an execution to QuadTree in Figure 1.

To describe SampleTree algorithm, we introduce the notations of neighborhood. For any element $e \in \{0,1\}^d$, let the neighborhood

Subroutine SampleTree(μ, m)

Input: A distribution μ supported on $\mathsf{EMD}_s(\{0,1\}^d)$, and positive integer m.
Output: A weighted tree $\mathbf{T}$.

(1) Take m random i.i.d. samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$. Let $\Omega = \bigcup_{i=1}^m \boldsymbol{y}_i \subset \{0,1\}^d$ and $\widehat{\Omega} = \textsc{Nbr}(\Omega)$.
(2) Run and return QuadTree($\widehat{\Omega}, \xi$) (Figure 1) where we set $\xi = \Theta(\log(msd/\delta))$.

Figure 3: The SampleTree Algorithm.

of e be

$$\textsc{Nbr}(e) \coloneqq \left\{p \in \{0,1\}^d \mid \|e - p\|_1 \le 1\right\}$$

We extend the above notation so that we can apply it to a set of elements as we do in Figure 3. For any set $\Omega \subseteq \{0,1\}^d$, let the neighborhood of Ω be

$$\textsc{Nbr}(\Omega) \coloneqq \left\{p \in \{0,1\}^d \mid \exists e \in \Omega, \|e - p\|_1 \le 1\right\}$$

The SampleTree sub-routine (in Figure 3) specifies a tree metric $\mathbf{T}$, and a natural association of any element $a \in \{0,1\}^d$ to a leaf in $\mathbf{T}$ (each element $a \in \{0,1\}^d$ maps to a unique leaf in $\mathbf{T}$, since the final hash function $\boldsymbol{\phi}_{L+1} \colon \{0,1\}^d \to \{0,1\}^d$ is set to the identity). Thus, we let $\mathsf{EMD}_s(\mathbf{T})$ denote the metric space on size-s tuples of leaves in $\mathbf{T}$. We let $x, y \in \mathsf{EMD}_s(\mathbf{T})$, with $x = (x_1, \ldots, x_s)$ and $y = (y_1, \ldots, y_s)$ where $x_1, \ldots, x_n, y_1, \ldots, y_n$ are leaves in $\mathbf{T}$, and

$$\mathsf{EMD}_{\mathbf{T}}(x, y) = \min_{\substack{\pi \colon [s] \to [s] \\ \text{bijection}}} \sum_{i=1}^{s} d_{\mathbf{T}}(x_i, y_{\pi(i)}),$$

where $d_{\mathbf{T}}(\cdot,\cdot)$ denotes the length of the shortest path between two leaves. We thus have the following (straight-forward) association of points $x \in \mathsf{EMD}_s(\{0,1\}^d)$ to points in $\mathsf{EMD}_s(\mathbf{T})$: if the point $x \in \mathsf{EMD}_s(\{0,1\}^d)$ is specified by the s elements $x_1, \ldots, x_s \in \{0,1\}^d$, we consider the point $x' \in \mathsf{EMD}_s(\mathbf{T})$ given by the s-tuple of mapped elements $x_1, \ldots, x_s$ which are leaves in $\mathbf{T}$. We abuse notation and refer to $x \in \mathsf{EMD}_s(\{0,1\}^d)$ and $x \in \mathsf{EMD}_s(\mathbf{T})$ for clarity—these are in bijective correspondence and should be clear from context whether we will use the sampled tree $\mathbf{T}$, or the original representation in $\{0,1\}^d$.

Data Structure Guarantees for SampleTree. It is important to note (and similarly to Definition 6.1) that the running time of naively executing SampleTree will incur exponential-in-d factors, since Line 3 of QuadTree iterates through $u \in \{0,1\}^{2^\ell}$ (where ℓ may be as high as poly(d)). Therefore, the total number of edges in $\mathbf{T}$ will incur exponential-in-d factors. However, the number of edges of $\mathbf{T}$ whose weight depends on the sample $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m \sim \mu$ is only $ms \cdot L$, as there are at most s elements in each of the m points $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ and these go down L edges; the rest of the edges have weights are $\xi \cdot d/2^\ell$, which only depend on the depth ℓ and thus be (implicitly) maintained. Even though 2^ℓ may be larger than d, it suffices to maintain the subset of sampled coordinates from $[d]$ (which takes $O(d)$ space). We thus have the following two facts, which we will use to implicitly compute the embedding of points in $\mathsf{EMD}_s(\{0,1\}^d)$ into ℓ_1.

Fact 7.1 ((Folklore) Isometric Embedding of a Tree Metric into ℓ_1). *Let $\mathbf{T}$ be any (rooted) weighted tree with k edges and depth $L+1$:*

- *There exists a map $\psi_{\mathbf{T}} \colon \mathsf{EMD}_s(\mathbf{T}) \to \mathbb{R}^k$ which is an isometric embedding into ℓ_1, i.e., for any $x, y \in \mathsf{EMD}_s(\mathbf{T})$, $\mathsf{EMD}_{\mathbf{T}}(x, y) = \|\psi_{\mathbf{T}}(x) - \psi_{\mathbf{T}}(y)\|_1$ (implicit in Section 4 of [25]).*
- *For $x \in \mathsf{EMD}_s(\mathbf{T})$, the vector $\psi_{\mathbf{T}}(x) \in \mathbb{R}^k$ has $(L+1) \cdot s$ non-zero entries.*

Note that, the data structure may then provide access to the root-to-leaf path specified by an element $a \in \{0,1\}^d$ to the leaf of $\mathbf{T}$ where it mapped to. In order to maintain a draw $\mathbf{T}$ from SampleTree(μ, m), the data structure may first read μ (supported on n points) and take m samples in $O(mn)$ time and then store the data-dependent weights in $O(msdL)$ time. Given a point $x \in \mathsf{EMD}_s(\{0,1\}^d)$, one may then evaluate the sparse representation of $\psi_{\mathbf{T}}(x)$ by obtaining its root-to-leaf path in poly(sd) time as well.

Expansion and Contraction of SampleTree. Given the above description of SampleTree and the corresponding embedding that it produces into ℓ_1, we state two lemmas below which bound the expansion and contraction of the SampleTree embedding. The proof of these two lemmas will constitute the bulk of the remainder of the section, and assuming the two lemmas, the proof of Lemma 5.4 follows by concatenation with an ℓ_1 locality-sensitive hash function.

Lemma 7.2 (Expansion of SampleTree). *Consider any pair of points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$, and suppose that x is (α, τ)-locally dense with respect to μ. Then, as long as $m = \omega(\log(sd)/\alpha)$,*

$$\mathop{\mathbb{E}}_{\mathbf{T}}[\mathsf{EMD}_{\mathbf{T}}(x, y)] \le \mathsf{EMD}(x, y) \cdot \tilde{O}(\log(msd/\delta)) \left(1 + \log\left(\frac{\tau + s}{\mathsf{EMD}(x, y)} + 1\right)\right).$$

over a draw of $\mathbf{T}$ from SampleTree(μ, m),

Lemma 7.3 (Non-Contraction of SampleTree). *For any $\delta \in (0,1)$, consider executing QuadTree (in Figure 1) with the parameter*

$$\xi = \Omega(\log(msd/\delta)).$$

Then, for any pair of points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$, over a draw of $\mathbf{T}$ from SampleTree(μ, m),

$$\Pr_{\mathbf{T}}[\mathsf{EMD}_{\mathbf{T}}(x, y) < \mathsf{EMD}(x, y)] \le \delta.$$

The proofs of Lemma 7.2 and Lemma 7.3 are in the full version.

7.1.1 Proof of Lemma 5.4 assuming Lemma 7.2 and Lemma 7.3. In order to prove Lemma 5.4, we make use of Lemmas 7.2 and 7.3 in order to embed into ℓ_1, and utilize a locality-sensitive hash function in ℓ_1. In particular, classic works on locality-sensitive hashing [38, 40] give, for any parameter $\gamma > 0$, a distribution over hash functions $\boldsymbol{\phi} \colon \mathbb{R}^k \to U$ which satisfies, for any $x, y \in \mathbb{R}^k$

$$\Pr_{\boldsymbol{\phi}}[\boldsymbol{\phi}(x) \ne \boldsymbol{\phi}(y)] \le \frac{\|x - y\|_1}{\gamma} \tag{9}$$

$$\Pr_{\boldsymbol{\phi}}[\boldsymbol{\phi}(x) = \boldsymbol{\phi}(y)] \le \exp\left(-\frac{\|x - y\|_1}{\gamma}\right). \tag{10}$$

Furthermore, it is simple to construct a data structure which maintains a description of a hash function $\boldsymbol{\phi}$ which is generated "on-demand," such that, if the vector $x \in \mathbb{R}^k$ is sparse and written as its sparse representation, the data structure can output $\boldsymbol{\phi}(x)$ in

time which is linear in the description of x. Given these guarantees, check both required properties of Lemma 5.4 whenever we let $\boldsymbol{h} \sim \mathcal{H}(\mu, \tau, \gamma, \delta)$ denote the concatenation of

$$\boldsymbol{h} : x \in \mathsf{EMD}_s(\{0,1\}^d) \xmapsto{\mathrm{Id}} x \in \mathsf{EMD}_s(\mathbf{T}) \xmapsto{\psi_{\mathbf{T}}} \psi_{\mathbf{T}}(x) \in \mathbb{R}^k \xmapsto{\boldsymbol{\phi}} \boldsymbol{\phi}(\psi_{\mathbf{T}}(x)) \in U,$$

where the first (identity) map $x \in \mathsf{EMD}_s(\{0,1\}^d)$ to $x \in \mathsf{EMD}_s(\mathbf{T})$ is the natural association of the elements of x as vectors in $\{0,1\}^d$ to elements of x as leaves of $\mathbf{T}$, the second map $\psi_{\mathbf{T}}$ is the map from Fact 7.1, and the third is the LSH for ℓ_1 specified in (9) and (10). We set $m = \mathrm{poly}(\log(sd)/\alpha)$, thus $\xi = \Theta(\log(sd/(\delta\alpha))$ when invoking Lemma 7.2 and Lemma 7.3.

- **Close Points Collide:** Given any points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$, if x is (α, τ)-locally dense with respect to μ, we have:

$$\begin{aligned}
&\Pr_{\boldsymbol{h}\sim\mathcal{H}(\mu,\tau,\gamma,\delta)}[\boldsymbol{h}(x) \neq \boldsymbol{h}(y)] \\
&= \mathop{\mathbb{E}}_{\mathbf{T}}\left[\Pr_{\boldsymbol{\phi}}[\boldsymbol{\phi}(\psi_{\mathbf{T}}(x)) \neq \boldsymbol{\phi}(\psi_{\mathbf{T}}(y))]\right] \\
&\overset{(9)}{\leq} \mathop{\mathbb{E}}_{\mathbf{T}}\left[\frac{\|\psi_{\mathbf{T}}(x) - \psi_{\mathbf{T}}(y)\|_1}{\gamma}\right] \\
&\overset{(7.1)}{\leq} \mathop{\mathbb{E}}_{\mathbf{T}}\left[\frac{\mathsf{EMD}_{\mathbf{T}}(x,y)}{\gamma}\right] \\
&\overset{(7.2)}{\leq} \frac{\mathsf{EMD}(x,y)}{\gamma} \cdot \tilde{O}\left(\log\left(\frac{sd}{\delta\alpha}\right)\right) \cdot \left(1 + \log\left(\frac{\tau+s}{\mathsf{EMD}(x,y)} + 1\right)\right),
\end{aligned}$$

  where the last inequality simplified $m = \mathrm{poly}(\log(sd)/\alpha)$.
- **Far Points Separate:** For points $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$, we use a union bound and Lemma 7.3 to upper bound the probability that $\boldsymbol{h}(x) = \boldsymbol{h}(y)$. Namely, we have

$$\begin{aligned}
&\Pr_{\boldsymbol{h}\sim\mathcal{H}(\mu,\tau,\gamma,\delta)}[\boldsymbol{h}(x) = \boldsymbol{h}(y)] \\
&\leq \mathop{\mathbb{E}}_{\mathbf{T}}\left[\Pr_{\boldsymbol{\phi}}[\boldsymbol{\phi}(\psi_{\mathbf{T}}(x)) = \boldsymbol{\phi}(\psi_{\mathbf{T}}(y))] \mid \mathsf{EMD}_{\mathbf{T}}(x,y) \geq \mathsf{EMD}(x,y)\right] \\
&\qquad + \Pr_{\mathbf{T}}[\mathsf{EMD}_{\mathbf{T}}(x,y) < \mathsf{EMD}(x,y)] \\
&\leq \mathop{\mathbb{E}}_{\mathbf{T}}\left[\exp\left(-\frac{\mathsf{EMD}_{\mathbf{T}}(x,y)}{\gamma}\right) \mid \mathsf{EMD}_{\mathbf{T}}(x,y) \geq \mathsf{EMD}(x,y)\right] + \delta \\
&\leq \exp\left(-\frac{\mathsf{EMD}(x,y)}{\gamma}\right) + \delta,
\end{aligned}$$

  where above, we similarly use Fact 7.1 to embed $\mathsf{EMD}_s(\mathbf{T})$ into ℓ_1 isometrically, the expression (10) for $\boldsymbol{\phi}$, and finally Lemma 7.3.

8 DATA-DEPENDENT HASHING AND SKETCHING LOWER BOUNDS

We now show that the data-dependent LSH (Definition 3.2) construction from Theorem 7 has an approximation factor of $\tilde{O}(\log s)$ which is best possible (up to the $\mathrm{poly}(\log\log s)$ factors) when p_1 and p_2 are constant. We do this by reducing data-dependent LSH to sketching lower bounds, and apply the lower bound on [6].

Definition 8.1 (EMD Sketching and Distributional EMD Sketching). *For every $s, d \in \mathbb{N}$ and every $r > 0$ and $c > 1$, we consider the communication complexity of the following partial function, whose inputs are sets $x, y \in \mathsf{EMD}_s(\{0,1\}^d)$ which satisfies:*

$$F(x,y) = \begin{cases} 1 & \mathsf{EMD}(x,y) \leq r \\ 0 & \mathsf{EMD}(x,y) > cr \end{cases}.$$

In the EMD sketching communication problem, we assume that a player Alice receives as input $x \in \mathsf{EMD}_s(\{0,1\}^d)$ and Bob receives an input $y \in \mathsf{EMD}_s(\{0,1\}^d)$, and they must design a public-coin communication protocol Π whose outputs align with F with probability at least $2/3$, and which minimizes the communication.

*Furthermore, we define the distributional version of the EMD sketching problem to be the same as above, but when there is "far" distribution μ, known to both Alice and Bob, such that the inputs (x, y) satisfy that either **(1)** x, y are arbitrary such that $\mathsf{EMD}(x,y) \leq r$ and the protocol should output 1, or **(2)** the inputs $x, y \sim \mu$ are drawn independently from μ and whenever $\mathsf{EMD}(x,y) \geq cr$ the algorithm should output 0 . Whenever $\mathsf{EMD}(x,y) \leq r$ or $\mathsf{EMD}(x,y) > cr$, then the communication protocol must be correct with probability $2/3$ over it's own randomness, and over the randomness of $x, y \sim \mu$ (if this inputs come from case **(2)**), and the output is allowed to be arbitrary if $r < \mathsf{EMD}(x,y) \leq cr$.*

Theorem 4.1 and Theorem 4.8 of [6] indicates the following:

Theorem 9 (Theorem 4.1 and Lemma 4.8 of [6]). *For any $d \in \mathbb{N}$ and $1 \leq c \leq d$, there exists a distribution μ on $\mathsf{EMD}_s(\{0,1\}^d)$ with $s = 2^{\Theta(d)}$ with the following properties:*

- *If $\boldsymbol{x}, \boldsymbol{y} \sim \mu$ are drawn independently, then $\mathsf{EMD}(\boldsymbol{x}, \boldsymbol{y}) \geq sd/100$ with probability at least $1 - 2^{-\Omega(d)}$.*
- *Another distribution ρ supported on pairs $\mathsf{EMD}_s(\{0,1\}^d) \times \mathsf{EMD}_s(\{0,1\}^d)$ for which $(\boldsymbol{x}, \boldsymbol{y}) \sim \rho$ satisfies $\mathsf{EMD}(\boldsymbol{x}, \boldsymbol{y}) \leq sd/(100c)$ with probability at least $1 - 2^{-\Omega(d/c)}$.*

For any function $f : \mathsf{EMD}_s(\{0,1\}^d) \to \{0,1\}$,

$$\Pr_{\boldsymbol{x},\boldsymbol{y}\sim\mu}[f(\boldsymbol{x}) = f(\boldsymbol{y})] + \Pr_{(\boldsymbol{x},\boldsymbol{y})\sim\rho}[f(\boldsymbol{x}) \neq f(\boldsymbol{y})] \geq 1 - 2^{-\Omega(d/c)}.$$

Since any data-dependent LSH for EMD can easily been seen to solve the distributional variant of sketching EMD, by constructing the LSH dependending on the known "far" distribution μ. It yields the following.

Theorem 10. *Consider any fixed constants $0 < p_2 < p_1 < 1$, and suppose there exists some $c > 1$ such that, for all $s, d \in \mathbb{N}$ and $r > 0$, there is a data-dependent LSH which is (r, cr, p_1, p_2)-sensitive for $\mathsf{EMD}_s(\{0,1\}^d)$. Then, $c = \Omega(\log s)$.*

REFERENCES

[1] Pankaj Agarwal, Kyle Fox, Debmalya Panigrahi, Kasturi Varadarajan, and Allen Xiao. 2017. Faster algorithms for the geometric transportation problem. In *Proceedings of the 33rd International Symposium on Computational Geometry (SOCG '2017)*.

[2] Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. 2022. Deterministic, near-linear ϵ-approximation algorithm for geometric bipartite matching. In *Proceedings of the 54nd ACM Symposium on the Theory of Computing (STOC '2022)*.

[3] Pankaj K. Agarwal and R. Sharathkumar. 2014. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proceedings of the 46th ACM Symposium on the Theory of Computing (STOC '2014)*. 555–564.

[4] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. 2009. Efficient sketches for earth-mover distance, with applications. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2009)*.

[5] Alexandr Andoni and Piotr Indyk. 2006. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2006)*. 459–468.

[6] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. 2008. Earth mover distance over high-dimensional spaces. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA '2008)*. 343–352.

[7] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. 2009. Overcoming the ℓ_1 Non-Embeddability Barrier: Algorithms for Product Metrics. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA '2009)*. 865–874.

[8] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. 2014. Beyond Locality-Sensitive Hashing. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA '2014)*. 1018–1028. Available as arXiv:1306.1547.

[9] Alexandr Andoni, Robert Krauthgamer, and Ilya Razenshteyn. 2015. Sketching and Embedding are Equivalent for Norms. In *Proceedings of the 47th ACM*

Symposium on the Theory of Computing (STOC '2015). 479–488. Available as arXiv:1411.2577.

[10] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. 2017. Optimal Hashing-based Time–Space Trade-offs for Approximate Near Neighbors. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA '2017)*. Available as arXiv:1608.03580.

[11] Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya Razenshteyn, and Erik Waingarten. 2018. Data-dependent Hashing via Non-linear Spectral Gaps. In *Proceedings of the 50th ACM Symposium on the Theory of Computing (STOC '2018)*.

[12] Alexandr Andoni, Assaf Naor, Aleksandar Nikolov, Ilya Razenshteyn, and Erik Waingarten. 2018. Hölder Homeomorphism and Approximate Nearest Neighbors. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2018)*.

[13] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In *Proceedings of the 46th ACM Symposium on the Theory of Computing (STOC '2014)*.

[14] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal Data-Dependent Hashing for Approximate Near Neighbors. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC '2015)*. 793–801. Available as arXiv:1501.01062.

[15] Alexandr Andoni and Ilya Razenshteyn. 2016. Tight Lower Bounds for Data-Dependent Locality-Sensitive Hashing. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG '2016)*. 9:1–9:11. Available as arXiv:1507.04299.

[16] Alexandr Andoni and Hengjie Zhang. 2023. Sub-quadratic (1+\eps)-approximate Euclidean Spanners, with Applications. (2023).

[17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International conference on machine learning*. PMLR, 214–223.

[18] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2020. Scalable nearest neighbor search for optimal transport. In *Proceedings of the 37th International Conference on Machine Learning (ICLM '2020)*.

[19] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. 2020. Scalable nearest neighbor search for optimal transport. In *International Conference on machine learning*. PMLR, 497–506.

[20] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. 2023. A Near-Algorithm for the Chamfer Distance. *Advances in Neural Information Processing Systems* (2023).

[21] Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, and D. Sivakumar. 2004. An information statistics approach to data stream and communication complexity. *J. Comput. System Sci.* 68, 4 (2004), 702–732.

[22] Yair Bartal. 1998. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC '1998)*.

[23] Artūrs Bačkurs and Piotr Indyk. 2014. Better Embeddings for Planar Earth-Mover Distance over Sparse Sets. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry* (Kyoto, Japan) *(SOCG'14)*. Association for Computing Machinery, New York, NY, USA, 280–289. https://doi.org/10.1145/2582112.2582120

[24] Lorenzo Beretta and Aviad Rubinstein. 2023. Approximate Earth Mover's Distance in Truly-Subquadratic Time. *arXiv preprint arXiv:2310.19514* (2023).

[25] Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC '2002)*. 380–388.

[26] Moses Charikar, Beidi Chen, Christopher Ré, and Erik Waingarten. 2023. Fast Algorithms for a New Relaxation of Optimal Transport. In *Proceedings of the 36rd Annual Conference on Learning Theory (COLT '2023)*.

[27] Moses Charikar, Beidi Chen, Christopher Ré, and Erik Waingarten. 2023. Fast Algorithms for a New Relaxation of Optimal Transport. In *The Thirty Sixth Annual Conference on Learning Theory*. PMLR, 4831–4862.

[28] Moses Charikar and Erik Waingarten. 2022. Polylogarithmic Sketches for Clustering. In *International Colloquium on Automata, Languages and Programming (ICALP '2022)*. 38:1–38:20.

[29] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2022')*.

[30] Xi Chen, Vincent Cohen-Addad, Rajesh Jayaram, Amit Levi, and Erik Waingarten. 2023. Streaming Euclidean MST to a constant factor. In *Proceedings of the 55nd ACM Symposium on the Theory of Computing (STOC '2023)*. 156–169.

[31] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. 2022. New streaming algorithms for high dimensional EMD and MST. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 222–233.

[32] Xiaoyu Chen, Shaofeng H.-C. Jiang, and Robert Krauthgamer. 2023. Streaming Euclidean Max-Cut: Dimension vs Data Reduction. In *Proceedings of the 55nd ACM Symposium on the Theory of Computing (STOC '2023)*. 170–182.

[33] Artur Czumaj, Guichen Gao, Shaofeng H-C Jiang, Robert Krauthgamer, and Pavel Veselỳ. 2023. Fully Scalable MPC Algorithms for Clustering in High Dimension. *arXiv preprint arXiv:2307.07848* (2023).

[34] Artur Czumaj, Shaofeng H.-C. Jiang, Robert Krauthgamer, Pavel Veselý, and Mingwei Yang. 2022. Streaming facility location in high dimension via geometric hashing. In *Proceedings of the 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2022')*. 450–461.

[35] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry (SoCG '2004)*. 253–262.

[36] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. System Sci.* 69, 3 (2004), 485–497.

[37] Emily Fox and Jiashuai Lu. 2023. A deterministic near-approximation scheme for geometric transportation. In *Proceedings of the 64th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2023')*.

[38] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of Computing* 8, 1 (2012), 321–350.

[39] Piotr Indyk. 2004. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th ACM Symposium on the Theory of Computing (STOC '2004)*. 373–380.

[40] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC '1998)*. 604–613.

[41] Piotr Indyk and Nitin Thaper. 2003. Fast Color Image Retrieval via Embeddings. In *Workshop on Statistical and Computational Theories of Vision (at ICCV)*.

[42] Rajesh Jayaram, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. 2024. Massively Parallel Algorithms for High-Dimensional Euclidean Minimum Spanning Tree. *Proceedings of the 35nd ACM-SIAM Symposium on Discrete Algorithms (SODA '2024)* (2024).

[43] William B Johnson and Gideon Schechtman. 1982. Embedding lpm into l 1 n. *Acta Mathematica* 149 (1982), 71–85.

[44] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA '2010)*.

[45] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitry Paramonov. 2019. Preconditioning for the geometric transportation problem. In *Proceedings of the 35th International Symposium on Computational Geometry (SoCG '2019)*.

[46] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on Machine Learning (ICML '2015)*.

[47] Andrew McGregor and Daniel Stubbs. 2013. Sketching earth-mover distance on graph metrics. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 274–286.

[48] Ryan O'Donnell, Yi Wu, and Yuan Zhou. 2014. Optimal Lower Bounds for Locality-Sensitive Hashing (Except When q is Tiny). *ACM Transactions on Computation Theory* 6, 1 (2014), 5.

[49] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP '2014)*. 1532–1543.

[50] Gabriel Peyré and Marco Cuturi. 2019. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning* 11, 5–6 (2019), 355–607.

[51] Edward M. Reingold and Robert E. Tarjan. 1981. On a greedy heuristic for complete matching. *SIAM J. Comput.* 10, 4 (1981), 676–681.

[52] R. Sharathkumar and Pankaj K. Agarwal. 2020. A near-ϵ-approximation algorithm for bipartite geometric matching. *J. ACM* 67, 3 (2020), 18:1–18:19.

[53] Jonah Sherman. 2017. Generalized preconditioning and undirected minimum cost flow. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA '2017)*.

[54] Arman Yousefi and Rafail Ostrovsky. 2014. Improved Approximation Algorithms for Earth-Mover Distance in Data Streams. *arXiv preprint arXiv:1404.6287* (2014).

Received 13-NOV-2023; accepted 2024-02-11