Lecture notes:
Probabilistic metric embeddings into high-dimensional quadtrees

Tal Wagner
Blavatnik School of Computer Science and AI
Tel Aviv University
talwag@tauex.tau.ac.il

May 16, 2026

Abstract

These are lecture notes for a class delivered by the author on May 14, 2026 as part of the graduate-level course "Approximation Algorithms for Big Data" at Tel Aviv University.

Note: These notes were not thoroughly checked for correctness. Please let me know if you spot any errors or inaccuracies. Review and citations of prior work are not intended to be comprehensive and complete, though please inform me of any significant oversight on that front as well.

Contents

1 Tree metrics

Let T be a tree, in the usual graph-theoretic sense, with n labeled leaves and positive edge weights. The associated tree metric on the leaves is defined such that the distance between a pair of leaves is the length of the unique path between them, where the "length" of a path is the sum of weights on its edges. We will denote the distance between two leaves x, y as $T(x, y)$.

Many algorithmic problems are easier on trees. This motivates us to want to embed more general metric spaces into tree metrics. Unfortunately, tree metrics are not expressive enough to accommodate general (nor Euclidean) metric spaces with small distortion. In particular, we have the following theorem:

Theorem 1.1 ([RR98])**.** *For every sufficiently large n, there is a metric space on n points such that embedding it into a tree metric requires distortion $\Omega(n)$.*

We won't go over the proof of this theorem, but only mention briefly that the "hard" metric space is the cycle metric C_n, i.e., the shortest-path metric on an unweighted cycle graph. Since C_n embeds with constant distortion into $\mathbb{R}^2$ (we won't prove that either, though the proof is not difficult and goes by placing the nodes evenly spaced on a circle), we have the following corollary.

Corollary 1.2. *For every sufficiently large n and $d \geq 2$, there is a Euclidean metric space on n points in $\mathbb{R}^d$ such that embedding it into a tree metric requires distortion $\Omega(n)$.*

2 Probabilistic embedding into tree metrics

To work around this roadblock, we introduce a more general definition of embedding: instead of embedding our metric space into a single tree metric, we embed it into a *distribution* over tree metrics. We require one side of the distortion bound to hold only in expectation over this distribution.

Definition 2.1. *Let (X, D) be a metric space and $b, c \geq 1$. We say that (X, D) embeds probabilistically into tree metrics if there is a distribution $\mathcal{T}$ over tree metrics such that each tree metric T in the support of $\mathcal{T}$ has n leaves that correspond to the points in X. Furthermore, for every x, y:*

- *$D(x, y) \leq b \cdot T(x, y)$ for every supported tree T in the support of $\mathcal{T}$.*
- *$\mathbb{E}_{T \sim \mathcal{T}}[T(x, y)] \leq c \cdot D(x, y)$.*

The distortion of the embedding is bc.

This notion was introduced [Bar96] in building on [Kar89, AKPW95]. In the definition, b is the *contraction* factor – the tree contracts distances by a factor up to b; and c is the (expected) *expansion* factor – the tree expands distances by a factor up to c, in expectation.

Remark: Normally you would see this definition with b normalized to $b = 1$ and only the expansion factor c. Such tree metrics are called "dominating" because the tree distance is never smaller than the original distance (i.e., no distance contraction). I keep b unspecified since we will need this in an application toward the end of the lecture.

[Bar96] proved an $O(\log^2 n)$ upper bound and an $\Omega(\log n)$ lower bound on the distortion. [Bar98] subsequently improved the upper bound to $O(\log n \log \log n)$. The optimal $O(\log n)$ bound was later achieved in [FRT03] with a construction now known as "FRT trees".

I should also mention that all these constructions, as well as the (different) one we will see, have an additional important property: the trees in the embedding are *hierarchically well-separated* (abbrev. HSTs). This means that the edge weights in each level in the tree are larger by a constant

multiplicative factor than the edge weights in the next level. Intuitively, they decompose the metric space into separate distance scales: the largest distance within any one subtree in level ℓ is small (up to constants) compared to any distance across two different subtrees in level ℓ. This scale-separation is helpful for many algorithmic applications in the metric space.

The construction and proof of FRT trees are very elegant and it is widely taught in graduate courses. You can find many great lecture notes on it online or just read the paper itself. We won't cover it here. Being designed for general metric spaces, the construction time is quadratic in n, which is too slow for our purposes. Rather, we will focus on a specific HST construction specialized to Euclidean (and ℓ_1) metrics: **probabilistic embedding into high-dimensional quadtrees**. The distortion won't be as good as FRT trees, but it will be quite good, and importantly, the construction time will be provably and practically efficient – nearly linear in n.

2.1 An aside: HSTs and combinatorial graph problems

Before we get into quadtrees, I want to only mention some satisfying connections of this topic to other topics in algorithms. Consider the *Min Bisection Problem:* the input in a simple graph $G(V, E)$ (undirected and unweighed) with an even number of nodes $n = |V|$. The goal is to find a cut with equal-sized sides and the minimum possible number of crossing edges. This is a classical NP-hard problem in combinatorial graph theory, seemingly not directly related to metric spaces.

At the time, the best approximation result for Min Bisection was $O(\log^2 n)$ and then $O(\log^{1.5} n)$ by [FK02]. Then, [Räc08] improved the approximation factor to $O(\log n)$ by uncovering connection between graph cuts and graph metrics that enabled utilizing FRT trees for solving Min Bisection. [AF09] provide a very clear exposition of this cut/metric duality in graphs.

[FK02]'s bounds for Min Bisection were based on Sparsest Cut, another NP-hard balanced graph partitioning problem. They showed that Min Bisection can be approximated up to $O(\log n)$ times the approximation factor for Sparsest Cut. Originally the best Sparsest Cut approximation algorithm result was $O(\log n)$ due to [LR99], and it was later improved to $O(\sqrt{\log n})$ by [ARV09]. Both of these classical results are also based on metric spaces and metric embeddings. Generally, there are deep connections between algorithmic metric space theory and combinatorial graph problems – this is what I wanted to point out here. If you are interested, all these results are covered in chapters 8.5 and 15 of the textbook [WS11], which is available online.

3 Probabilistic embedding into high-dimensional quadtrees

We now get to the main algorithm of this lecture: the randomly shifted high-dimensional quadtree. Everything we will see in this lecture from now on is an application of this algorithm.

3.1 Preliminaries: Metric space parameters, and embedding ℓ_2 into ℓ_1

We have mentioned in class that Euclidean metrics embed near-isometrically into ℓ_1. Furthermore, an n-point Euclidean metric can be efficiently embedded in ℓ_1 with dimension $O(\log n/\varepsilon^2)$ such that the distortion is $(1 \pm \epsilon)$ with high probability. This is a variant of JL sometimes called "$\ell_2 \to \ell_1$ JL". As we will see soon, our high-dimensional quadtrees interact more naturally with ℓ_1 than ℓ_2, so we might as well embed our ℓ_2 metrics into ℓ_1 for the purpose of this lecture. By "$\ell_2 \to \ell_1$ JL", this only proves a more general result.

The setting. For the rest of this lecture, we assume that we have an ℓ_1 metric space $X \subset \mathbb{R}^d$ with n points. By normalization, without loss of generality, we assume that the minimum distance

between any two points is 1. Under this normalization, we further assume that the coordinates of all points are in the range $[0, \Delta)$ for some $\Delta \geq 1$. Note that the only assumption we've made is that the coordinates have bounded (rather than infinite) range: the assumption that they are non-negative is without loss of generality by shifting, since the ℓ_1 distance is shift-invariant (i.e., adding some large number to all the coordinates to make them all non-negative does not change any distances). Observe that now the ℓ_1-diameter of our metric space is at most Δd, and since we have normalized the minimum distance to 1, the aspect ratio is at most Δd.

We will work explicitly with all three parameters n, d, Δ. However, for the sake of intuition, it is instructive to keep in mind a typical setting with just one parameter, n. For d, we will think of it as $d = O(\log n)$, with the excuse that our ℓ_1 metric space was actually obtained from an ℓ_2 metric space by $\ell_2 \to \ell_1$ JL (with ε set arbitrarily to 0.5; since we will deal throughout the lecture with worse-than-constant distortion, it doesn't matter). For Δ, we will think of it as $\Delta = n^{O(1)}$ and therefore $\log \Delta = O(\log n)$, with the excuse that each coordinates is represented in a computer as a single machine word, which in standard RAM model is assumed to have $O(\log n)$ bits.

3.2 Construction of the randomly shifted high-dimensional quadtree

The first step in the construction is a random shift of the points. We pick a random vector $s \in \mathbb{R}^d$ with i.i.d. coordinates sampled uniformly in $[0, \Delta)$. We replace each point x with its shifted analog $x + s$. This obviously doesn't change any of the distances: $\|(x + s) - (y + s)\|_1 = \|x - y\|_1$. This is the only randomness used in the construction.

In the second step, we build a high-dimensional quadtree on the points by imposing a hierarchy of nested grids. The top grid is just a hypercube with side-length 2Δ that encloses all of the points. Note that the side-length is large enough to enclose all of the points even after the random shift. This hypercube is the root of the quadtree. In the next level, we partition the hypercube into sub-hypercubes by splitting it along each axis exactly in the middle. This creates as many as 2^d sub-hypercubes, but we avoid the exponential dependence on d by creating only those sub-hypercubes that contain any points from our metric space. Thus, we create at most n sub-hypercubes, and they are the children of the root in the quadtree. We proceed recursively on each hypercube in the new level, partitioning it into sub-hypercube by halving it along each axis, and creating quadtree nodes for the non-empty sub-hypercubes. As for edge weights: we set the weight of every edge to be side-length of the hypercube it leads to. Figure 1 illustrates the process.

We do this until we have $L := \lceil \log_2(4\Delta d) \rceil$ levels. Actually, let's simplify and assume that both Δ and d are powers of 2, and thus $L = \log_2(4\Delta d)$. This does not lose generality since we can always round them up to the nearest power of 2, losing only constant factors that we suppress anyway.

Why this number of levels? Recall that we want each leaf in the quadtree to contain a single point from our metric space X. The root hypercube has side-length 2Δ. Each subsequent level of the quadtree halves the side-length. Therefore, if we build the quadtree with $L = \log_2(4\Delta d)$ levels, the side-length of the leaves will be $2\Delta/2^{L-1} = 1/d$. Therefore, the diameter of the leaf hypercubes will be 1. Here we recall that we are working in ℓ_1, not ℓ_2, so the ℓ_1-diameter of a d-dimensional hypercube is its side-length times d, not $\sqrt{d}$; and that are hypercubes are cartesian products of semi-open intervals, so our leaf hypercubes with side-length $1/d$ cannot fit any two points at distance 1. Since we have assumed by normalization that the minimal ℓ_1 distance between two points in our metric space X is 1, this guarantees that each leaf contains exactly one point from X.

Formal data structure and construction algorithm. To be concrete, let's specify the construction formally. The quadtree is a data structure where:

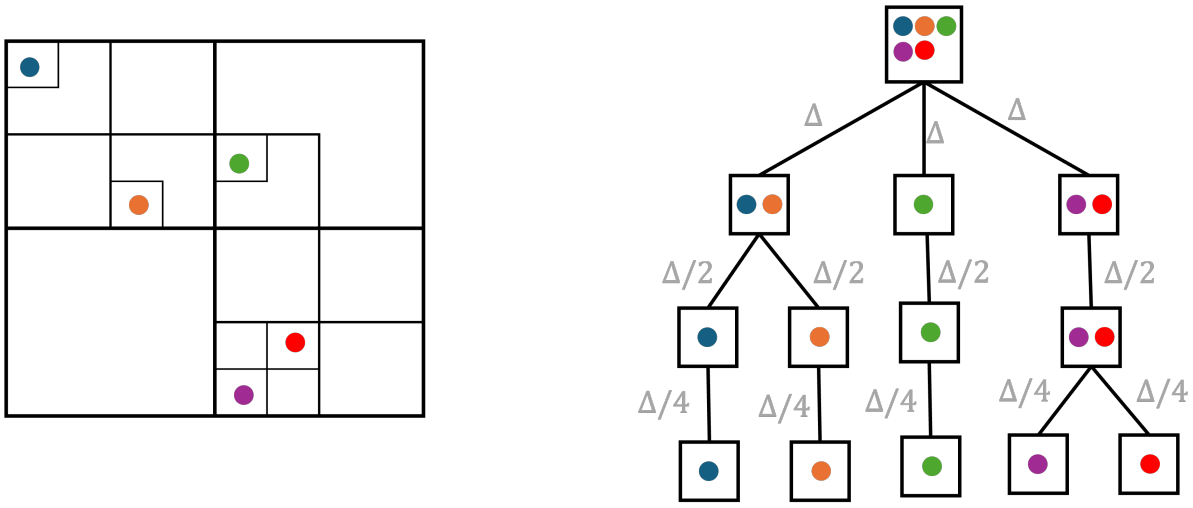

Figure 1: Illustration of quadtree construction.

- Each node v is associated with a hypercube $H_v \subset \mathbb{R}^d$, a "cluster" $X_v \subset X$ of the points in our metric space contained in H_v, and a list of edges E_v that lead to its children.
- Each edge e has a label $\tau_e \in \{0,1\}^d$ and a positive weight equal to the side-length of the hypercube below it.

The construction algorithm is:

1. Initialize the root node v_0 with hypercube $H_{v_0} = \times_{i=1}^{d}[0, 2\Delta)$, cluster $X_{v_0} = X$, and outgoing edges $E_v = \emptyset$.
2. Initialize a list `prevLevelNodes` that contains only v_0.
3. For $\ell = 1, \ldots, \log_2(4\Delta d) - 1$:
   - 3.1 Initialize an empty list `newLevelNodes`.
   - 3.2 For every node v in `prevLevelNodes`:
     - 3.2.1 Let $H_v = \times_{i=1}^{d}[a_i, b_i)$ denote the axis-intervals of its associated hypercube H_v. Furthermore, let $m_i = \frac{1}{2}(a_i + b_i)$ denote the midpoint of each axis-interval.
     - 3.2.2 Initialize an empty map $\phi(v)$ that maps length-d bitstrings to quadtree nodes.
     - 3.2.3 For every point $x \in X_v$:
       - 3.2.3.1 Compute the length-d bitstring $\tau(x) := (\mathbf{1}\{x_i \geq m_i\})_{i=1}^{d}$.
       - 3.2.3.2 If $\tau(x)$ doesn't exist in the key list of $\phi(v)$:
         - 3.2.3.2.1 Define the hypercube

$$H_x := \times_{i=1}^{d} I_i \quad \text{where} \quad I_i := \begin{cases} [a_i, m_i) & \tau_i(x) = 0, \\ [m_i, b_i) & \tau_i(x) = 1. \end{cases}$$

         - 3.2.3.2.2 Create a new tree node v_x with the associated hypercube $H_{v_x} = H_x$ and cluster $X_{v_x} = \{x\}$.
         - 3.2.3.2.3 Add the mapping $\phi(v)[\tau(x)] = v_x$ to $\phi(v)$.
       - 3.2.3.3 Else, let $v' = \phi(v)[\tau(x)]$. Add x to its associated cluster $X_{v'}$.

3.2.4 For every length-d bitstring τ in the keys of $\phi(v)$:

3.2.4.1 Let $v' = \phi(v)[\tau]$.

3.2.4.2 Add an edge to E_v with label $\tau_e = \tau$ that leads to the child node v'.

3.2.4.3 Add v' to `newLevelNodes`.

3.3 `prevLevelNodes` $\leftarrow$ `newLevelNodes`.

The construction is simple: we go level by level. In each level we split each hypercube from the previous level. In step 3.2.3 we iterate over the points in the hypercube, and in step 3.2.3.1 we determine which sub-hypercube the point belongs to. If this sub-hypercube has not been created yet, we create it in step 3.2.3.2. Then we add all of the newly created sub-hypercubes as children of our current hypercube, and move on.

Running time. The nodes in each level of the quadtree form a partition of the points in X. Therefore, in step 3.2.3, each point is encountered exactly once. For each point, step 3.2.3.1 takes time $O(d)$ to compute since it thresholds each coordinate in the point. These are the dominating computations running-time-wise, so processing every level (i.e., every iteration of the loop in step 3) takes $O(nd)$ time. The total time over L levels is $O(ndL) = O(nd \log(\Delta d))$.

Memory. The quadtree has at most n nodes in each level, so $O(nL)$ nodes and edges in total. The hypercube of each node and label of each edge can be stored in $O(d)$ machine words. This takes total memory $O(ndL)$. We won't actually need to store the cluster associated with each node; it suffices to store only the point associated with each leaf. We have n leaves and each is associated with one point, so the total extra space is $O(n)$ machine words. The clusters can be recovered since for every node v, the cluster X_v is exactly the points associated with the leaves in the subtree rooted by v.

Randomly shifted quadtrees as probabilistic embedding into trees. We have thus defined a probabilistic embedding into (quad)trees: sampling the initial random shift and then executing the above quadtree construction returns a randomized quadtree with leaves corresponding to the points in X. This adheres to Definition 2.1. Next we will analyze the distortion of this probabilistic embedding into trees.

Remark. You will often hear this construction referred to (including by me) as "randomly shifted grids". We shifted the points rather than grids for the convenience of presentation, but it of course makes no difference whether you shift the points or the grids (except that if we shift the points, we have to also remember to similarly shift all future points that we would encounter).

4 Distortion analysis

We now prove distortion bounds in line with Definition 2.1:

Theorem 4.1. *The above randomized embedding into quadtrees satisfies the following for every $x, y \in X$:*

- $\|x - y\|_1 \le O(d) \cdot T(x, y)$ *for every supported quadtree T.*
- $\mathbb{E}[T(x, y)] \le O(\log(\Delta d)) \cdot \|x - y\|_1$.

Thus, the distortion is $O(d \log(\Delta d))$. Recall that in our intuitive "typical" setting we have $d = O(\log n)$ and $\log \Delta = O(\log n)$, so in this setting the distortion is $O(\log^2 n)$.

4.1 Level numbering

Let us fix a way to number the levels that will make all subsequent analysis more convenient. We number the levels from top to bottom as $\log_2(2\Delta), \log_2(2\Delta) - 1, \ldots, \log_2(1/d)$. Thus, in every level ℓ, the side-length of the hypercubes in that level of the quadtree is just 2^ℓ. Figure 2 illustrates this.

Note that we indeed have $L = \log_2(4\Delta d)$ levels, because the number of levels numbered $\log_2(2\Delta), \log_2(2\Delta) - 1, \ldots, \log_2(1/d)$ is $\log_2(2\Delta) - \log_2(1/d) + 1 = \log_2(4\Delta d)$.

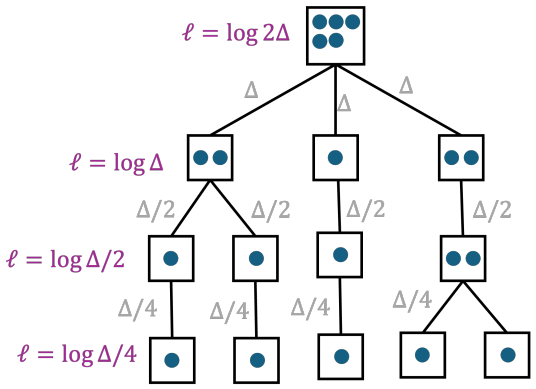

Figure 2: Level numbering in the quadtree.

4.2 Quadtree distances

Before getting to the formal proof, let's take a moment to just observe what the quadtree distances are. Take two points $x, y \in X$. In the quadtree, each is associated with a leaf. Denote those leaves v_x, v_y respectively. The quadtree distance $T(x, y)$ is the sum of edge weights on the unique path between v_x and v_y.

Those two points, like all points, start out at the same hypercube at the root, and at some point become separated as we go down the tree levels. Let ℓ^*_{xy} be the lowest level where x, y are still in the same hypercube, and let v^*_{xy} be the node associated with that hypercube. The distance $T(x, y)$ is the sum of lengths of the paths from v^*_{xy} to v_x and from v^*_{xy} to v_y. The length of each of those paths is $\Lambda := \sum_{\ell=\log(1/d)}^{\ell^*_{xy}-1} 2^\ell$. Therefore, $T(x, y) = 2\Lambda$. Therefore, $T(x, y)$ satisfies:

$$2^{\ell^*_{xy}} \le T(x, y) \le 2 \sum_{\ell=-\infty}^{\ell^*_{xy}-1} 2^\ell \le 2 \cdot 2^{\ell^*_{xy}}.$$

Thus, we have the very specific characterization, $T(x, y) = \Theta(2^{\ell^*_{xy}})$: the quadtree distance $T(x, y)$ is simply equal, up to a constant, to the exponent of the last level where x, y fall in the same hypercube. Our goal in proving Theorem 4.1 will be to relate this to their original ℓ_1 distance.

4.3 Deterministic distance contraction

We prove the first bullet in Theorem 4.1. On the one hand, by the above, we have $T(x, y) \ge 2^{\ell^*_{xy}}$. On the other hand, the hypercube that contains both x, y in level ℓ^*_{xy} has side-length $2^{\ell^*_{xy}}$, and therefore it has ℓ_1-diameter $2^{\ell^*_{xy}} d$. Therefore we must have $\|x - y\|_1 \le 2^{\ell^*_{xy}} d$. The bound follows.

4.4 Expected distance expansion

We prove the second bullet in Theorem 4.1. Let us denote by $Sep(x, y, \ell)$ the event that x, y are separated at level ℓ (i.e., they fall in different hypercubes at level ℓ of the quadtree). In order for this event to occur, it's enough for them to be separated at any axis. Therefore, by a union bound,

$$\Pr[Sep(x, y, \ell)] \le \sum_{i=1}^{d} \Pr[x, y \text{ are separated at axis } i \text{ at level } \ell].$$

The probabilities on the right-hand side are not hard to work out exactly. The hypercubes at level ℓ have side-length 2^ℓ. If $|x_i - y_i| \geq 2^\ell$, then it doesn't matter how we randomly shift the point, they will always fall in different hypercubes. If $|x_i - y_i| < 2^\ell$, then the probability they are separated at axis i at level ℓ is the probability that the grid boundary falls between them at axis i. Since the shift is uniformly random,

$$\Pr[x, y \text{ are separated at axis } i \text{ at level } \ell] = \min\{1, \frac{|x_i - y_i|}{2^\ell}\}.$$

Therefore,

$$\Pr[Sep(x, y, \ell)] \leq \sum_{i=1}^{d} \min\{1, \frac{|x_i - y_i|}{2^\ell}\} \leq \sum_{i=1}^{d} \frac{|x_i - y_i|}{2^\ell} = \frac{\|x - y\|_1}{2^\ell}.$$

Now, we observe that we can write the tree distance $T(x, y)$ as

$$T(x, y) = \sum_{\ell=\log(1/d)}^{\log(2\Delta)} \mathbf{1}\{Sep(x, y, \ell)\} \cdot 2 \cdot 2^\ell.$$

This should be easy to see: for each level ℓ, if x, y are separated in level ℓ, then it contributes two edges with weight 2^ℓ to their tree distance (one on the path $v^*_{xy} - v_x$ and one on the path $v^*_{xy} - v_y$). If x, y are not separated at level ℓ, it contributes nothing to their tree distance.

Now we finish the proof by taking expectation:

$$\begin{aligned}
\mathbb{E}[T(x, y)] &= \sum_{\ell=\log(1/d)}^{\log(2\Delta)} \Pr[Sep(x, y, \ell)] \cdot 2 \cdot 2^\ell \\
&\leq \sum_{\ell=\log(1/d)}^{\log(2\Delta)} \frac{\|x - y\|_1}{2^\ell} \cdot 2 \cdot 2^\ell \\
&= 2\|x - y\|_1 \sum_{\ell=\log(1/d)}^{\log(2\Delta)} 1 \\
&= 2\|x - y\|_1 \log(4\Delta d).
\end{aligned}$$

5 Approximate nearest neighbor search

The reason we try to embed metric spaces into simpler metric spaces – like probabilistic embedding into trees – is to get efficient algorithms. As an example, let's see how we can use our randomly shifted quadtree embeddings for ANN search.

Theorem 5.1. *Using the above randomized quadtree embeddings, we can get a solution to the c-approximate nearest neighbor problem in ℓ_1 with:*

- *Construction time and memory usage $O(nd \log(\Delta d))$.*
- *Query time $O(d \log(\Delta d))$.*
- *Approximation factor $c = O(d^2 \log(\Delta d))$.*

In our "typical setting", the construction time is near-linear: $\widetilde{O}(n)$; the query time is polylogarithmic: $O(\log^2 n)$; and the approximation is polylogarithmic, $c = O(\log^3 n)$. Compare this with the LSH-based ANN solution we saw in the last lecture: the construction time was $\widetilde{O}(n^{1+\rho})$, the query time was $\widetilde{O}(n^{\rho})$, and the approximation factor was any constant $c > 1$ we choose, with $\rho \approx 1/c$. So the above theorem yields better computational efficienty (construction time, memory, query time), but a considerably more crude approximation quality. Still, there are applications where this trade-off is beneficial.

Construction. The construction is simply to build a randomly shifted quadtree on X as we've done above. The construction time and memory usage have already been analyzed.

Query algorithm. Given a query point $z \in \mathbb{R}^d$, how are we going to return an approximate nearest neighbor from our quadtree? For simplicity, suppose that z has coordinates in $[0, \Delta)$. Nevertheless, it wasn't present when the quadtree was constructed. We only created tree nodes with hypercubes that contained points from X. So in some levels in the tree, the hypercubes that are supposed to contain z might not exist.

The query algorithm will be as follows. First we remember to also shift z by the same shift as all the points in the quadtree, i.e., $z \leftarrow z + s$. Then, we simulate inserting z into the quadtree. We start at the root. We know that z is in its associated hypercube because of our assumption about the coordinate range of z. We try to walk downward to the sub-hypercube that contains z. If it exists, we walk to it and repeat the process. If we get stuck, because the sub-hypercube that contains z doesn't exist, we simply return an arbitrary point from the cluster of the node where we got stuck. Concretely, this means that at that stage we just continue traversing the tree downward along arbitrary edges, until we reach a leaf, and then we return the point from X associated with the leaf we ended up reaching.

It should be clear that the query time is $O(dL) = O(d \log(\Delta d))$, because the query algorithm simply traverses the quadtree from the root downward to a leaf. The path has length L, and in each step we spend $O(d)$ time to determining the next node to go to.

Approximation analysis. Let x^* denote the true nearest neighbor of z in X. Let $\widetilde{x}$ denote the approximate nearest neighbor returned by our algorithm.

We define a "goodness" condition for the query which ensures our algorithm returns a good approximate nearest neighbor for it. It is called *padding*.

Definition 5.2 (padding)**.** *We say that z is* α-padded *if for every level ℓ, the hypercube that contains z also contains the entire ℓ_1-ball of radius $2^\ell \alpha$ around z.*

Padding means that z is far from the boundaries of the hypercube that contains it. Note how this addresses the "bad" case in the quadtree embedding: where things can go wrong in the embedding is if z is very near the boundary of its hypercube, since in that case, the nearest neighbor could be just across the boundary, but in a completely different branch in the tree. Padding ensures that this does not happen.

First, let's see that padding ensures that our algorithm returns a good approximate near neighbor.

Lemma 5.3. *If z is α-padded, then $\|z - \widetilde{x}\|_1 \le 2d\alpha^{-1}\|z - x^*\|_1$.*

Proof. Let ℓ^* be the lowest level where z and x^* are in the same hypercube. This means they are separated at level $\ell^* - 1$. By padding, x^* must lie outside the ball of radius $2^{\ell^*-1}\alpha$ around z, thus $\|z - x^*\| > 2^{\ell^*-1}\alpha$. On the other hand, the query algorithm returns some $\widetilde{x}$ from the same level-ℓ^* hypercube as z, whose ℓ_1-diameter is $2^{\ell^*}d$, and therefore $\|z - \widetilde{x}\|_1 \le 2^{\ell^*}d$. □

Our next task is to show that every query z is α-padded with constant probability with the best (i.e., largest) α we can.

Lemma 5.4. *Every z is α-padded with probability $2/3$ with $\alpha = 1/(6d\log(4\Delta d))$.*

Proof. Consider what can cause z to not be padded. Padding is violated by axis i at level ℓ is the boundary of the grid with side-length 2^ℓ hits at distance at most $2^\ell\alpha$ from z along axis i either to its left or to its right. Since the shift is uniformly random, this happens with probability $\frac{2\cdot 2^\ell\alpha}{2^\ell} = 2\alpha$. By a union bound over all d axes and all L levels, the probability that the padding of z is violated anywhere is at most $2\alpha dL$. Plugging $\alpha = 1/(6dL)$, the lemma follows. □

This concludes the proof of Theorem 5.1: by Lemma 5.4, our query z is α-padded with $\alpha = 1/\Theta(d\log(\Delta d))$ with probability $2/3$, and by Lemma 5.3, this implies that our quadtree reports a c-approximate nearest neighbor with $c = O(d/\alpha) = O(d^2\log(\Delta d))$.

6 Earth Mover Distance

Let's move on to a different set of applications where high-dimensional quadtrees are useful. They are about a different and far more complex notion of distance – the earth mover distance (EMD). It is also known (with some variations) as the Optimal Transport distance and the Wasserstein-1 distance. It is a very developed area in mathematics with many applications in computer science and specifically in machine learning. We mentioned some in class though I'll skip them here.

Let's define EMD. It is defined over a "ground" metric, which in our case will be ℓ_1 or ℓ_2 in $\mathbb{R}^d$. As before, let's work with ℓ_1. EMD is a notion of distance between two *sets* of points (or more generally, two distributions over points) instead of between just two points. So, every "point" in EMD is a distribution over points in $\mathbb{R}^d$.

Defining EMD. Let $X, Y \subset \mathbb{R}^d$ be two subsets of points in $\mathbb{R}^d$, each of size at most n. Each subset is endowed with a distribution over its points: a distribution $\mathcal{D}_X$ over X and a distribution $\mathcal{D}_Y$ over Y. For the purpose of this lecture, we will assume that both $\mathcal{D}_X$ and $\mathcal{D}_Y$ are uniform distributions; this already captures nearly all the difficulty.

Consider the undirected weighted bipartite graph between X and Y, where the edge between $x \in X$ and $y \in Y$ has weight equal to their distance in the ground metric, $\|x-y\|_1$. The EMD distance between X and Y is defined as the cost of the minimum-cost flow on this graph from X to Y with demands specified by $\mathcal{D}_X$ and $\mathcal{D}_Y$:

$$EMD(X,Y) := \min_F \sum_{x\in X,\, y\in Y} \|x-y\|_1 \cdot F(x,y),$$

where F ranges over all admissible flows. This is illustrated in Figure 3.

Intuitively, we start with mass $\mathcal{D}_X$ distributed across the points in X, and our goal is re-distribute

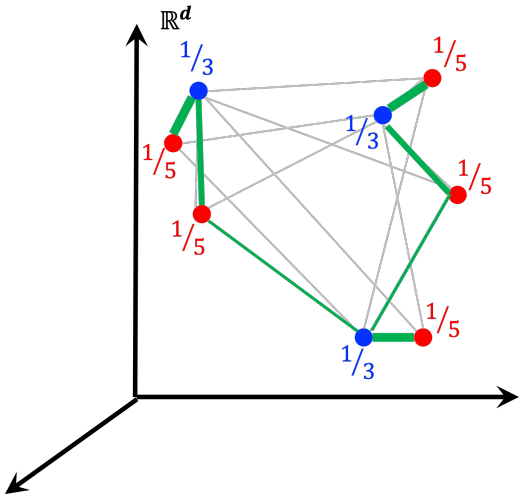

Figure 3: Illustration of EMD between a blue subset with 3 points and a red subset with 5 points.

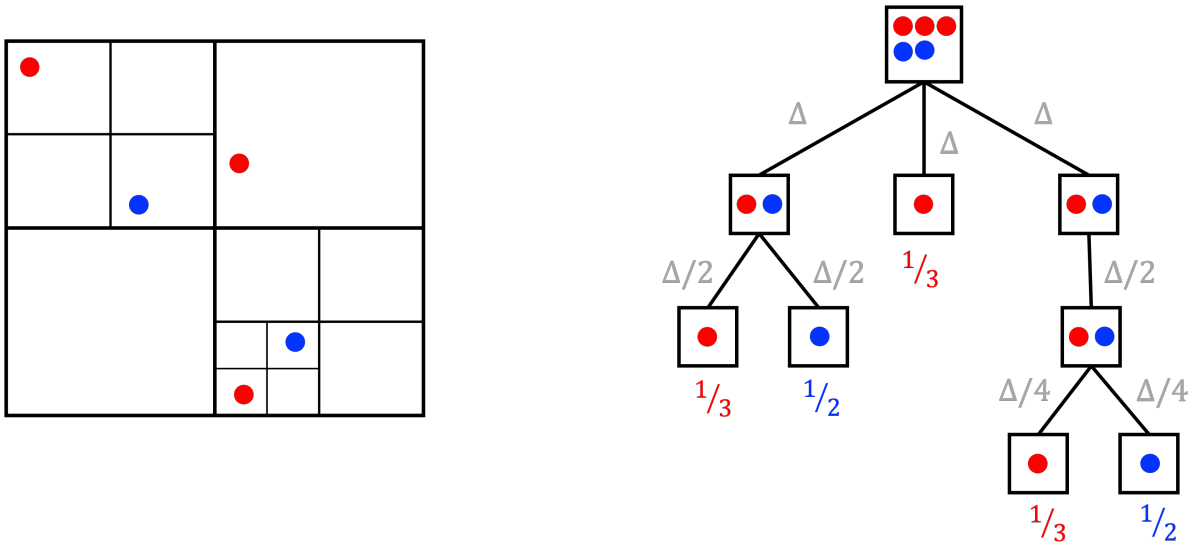

Figure 4: Illustration of tree-EMD over the quadtree metric as a ground metric.

it across the points in Y with mass $\mathcal{D}_Y$. Our goal is to carry out this re-distribution such that the total "ground" distance (ℓ_1 distance, in our case) traveled by the mass particles is as small as possible. This is where the name comes from: we imagine each mass particle as a grain of sand, and we want to take piles lying on the points in X and re-distribute them across Y.

It is not difficult to see that EMD is a metric, i.e., that it satisfies the triangle inequality. Suppose we had a third subset Z of points endowed with a distribution $\mathcal{D}_Z$. We could transport the mass from X to Z at cost $EMD(X,Z)$ and then from Z to Y at cost $EMD(Z,Y)$. This composed flow through Z is an admissible flow from X to Y with cost $EMD(X,Z)+EMD(Z,Y)$. The EMD is defined as the minimum cost flow from X to Y, hence by definition, $EMD(X,Y) \leq EMD(X,Z) + EMD(Z,Y)$.

Computing EMD. How do we even compute the EMD between one pair of subsets? Computing it exactly means solving a min-cost flow problem, which takes time super-linear in n. This is quite slow and non-scalable even for one distance computation, let alone more complex computations on the EMD metric space. So as usual, we turn to approximation.

7 EMD approximation with high-dimensional quadtrees

To start, let's see how we approximate the EMD in near-linear time. As reminder, we keep our assumptions throughout that the point coordinates are bounded in $[0, \Delta)$ and that the minimum ℓ_1 distance between any pair of points is 1.

Theorem 7.1. *Given X, Y, we can compute in time $O(nd\log(\Delta d))$ an estimate $\widetilde{EMD}(X,Y)$ that with probability $2/3$ approximates $EMD(X,Y)$ up to distortion $O(d\log(\Delta d))$.*

The proof is, of course, through our probabilistic embedding of the subsets X, Y into a quadtree. Note that once our subsets X, Y are embedded in the tree metric – which is a metric space – we can define the EMD distance between X and Y with the tree metric as the ground metric instead of ℓ_1. This is illustrated in Figure 4. We will henceforth denote EMD with the ℓ_1 ground metric as $EMD(X,Y)$ and EMD with the tree ground metric as $EMD_T(X,Y)$.

Our approximation algorithm simply embeds $X \cup Y$ into a randomized quadtree T, computes $EMD_T(X,Y)$, and returns it as the estimate $\widetilde{EMD}(X,Y)$. Our two goals toward proving Theorem 7.1 are to show that $EMD_T(X,Y)$ can be computed in near-linear time and that it is a good approximation for $EMD(X,Y)$.

Lemma 7.2. *$EMD_T(X,Y)$ can be computed in time $O(n \log(\Delta d))$.*

Proof. A greedy algorithm works here. We start with an informal overview. We take any leaf in the tree. If it has both $\mathcal{D}_X$-mass and $\mathcal{D}_Y$-mass, we match them as much as possible; this adds no cost for $EMD_T(X,Y)$, because the tree distance between a leaf and itself is zero. Then we are left with only one type of mass in the leaf – whichever was larger. For example, if the $\mathcal{D}_X$-mass at the leaf is $1/2$ and the $\mathcal{D}_Y$-mass is $1/3$, we can match $1/3$ mass at the leaf and we are left with an excess $1/6$ of $\mathcal{D}_X$-mass. This is illustrated in Figure 5.

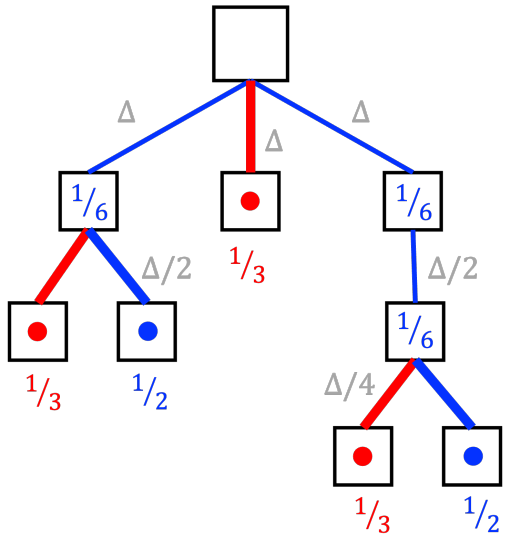

Figure 5: Illustration of greedily computing EMD with a ground tree metric.

We ship it upward to the parent of the leaf, because that excess mass has nowhere else to go. Now the leaf has no mass, so we remove it from the tree. We repeat this with another leaf, and so on, until we have removed all nodes from the tree. The masses must eventually even out at the root because the total $\mathcal{D}_X$-mass and total $\mathcal{D}_Y$-mass were equal (to 1) to begin with. So we get an admissible flow.

Now, formally: for every node v in the quadtree define two variables, $m_X(v)$ for its $\mathcal{D}_X$-mass and $m_Y(v)$ for its $\mathcal{D}_Y$-mass. Initially each leaf v has its $m_X(v)$ and $m_Y(v)$ set according to the point in $X \cup Y$ associated with it, and for each internal node v we set $m_X(v) = m_Y(v) = 0$. The algorithm is:

1. Initialize $C \leftarrow 0$.
2. While the tree has more than one node:
   - 2.1 Let v be a leaf. Let u be its parent. Let $w(v)$ be the edge weight between u and v.
   - 2.2 $\mu \leftarrow \min\{m_X(v), m_Y(v)\}$. *// maximum mass that can be matched inside the leaf.*
   - 2.3 $m_X(v) \leftarrow m_X(v) - \mu$. *// remove matched $\mathcal{D}_X$-mass.*
   - 2.4 $m_Y(v) \leftarrow m_Y(v) - \mu$. *// remove matched $\mathcal{D}_Y$-mass.*
   - 2.5 $m_X(u) \leftarrow m_X(u) + \max\{0, m_X(v) - m_Y(v)\}$. *// ship excess $\mathcal{D}_X$-mass to parent.*
   - 2.6 $m_Y(u) \leftarrow m_Y(u) + \max\{0, m_Y(v) - m_X(v)\}$. *// ship excess $\mathcal{D}_Y$-mass to parent.*
   - 2.7 $C \leftarrow C + w(v) \cdot \max\{m_X(v), m_Y(v)\}$. *// add cost of shipping excess mass to parent.*
   - 2.8 Delete v and its incident edge from the tree.
3. Return C.

We do $O(1)$ operations per tree node, and there are at most nL tree nodes, so the running time is $O(nL)$. Correctness is proven easily by induction on the number of nodes in the tree. □

Lemma 7.3. *With probability* $2/3$ *we have*

$$\Omega\left(\frac{1}{d}\right) \cdot EMD(X,Y) \leq EMD_T(X,Y) \leq O(\log(\Delta d)) \cdot EMD(X,Y).$$

Proof. We will use our distortion analysis from Theorem 4.1. Let F_1 denote the optimal (min-cost) flow for EMD with the ℓ_1 ground metric, and F_T the optimal (min-cost) flow for EMD_T with the quadtree ground metric. Thus we have,

$$EMD(X,Y) = \sum_{x,y} \|x-y\|_1 \cdot F_1(x,y) \quad \text{and} \quad EMD_T(X,Y) = \sum_{x,y} T(x,y) \cdot F_T(x,y).$$

Distance contraction: we have deterministically,

$$\begin{aligned}
EMD_T(X,Y) &= \sum_{x,y} T(x,y) \cdot F_T(x,y) \\
&\geq \Omega\left(\frac{1}{d}\right) \cdot \sum_{x,y} \|x-y\|_1 \cdot F_T(x,y) && \text{first bullet of Theorem 4.1} \\
&\geq \Omega\left(\frac{1}{d}\right) \cdot \sum_{x,y} \|x-y\|_1 \cdot F_1(x,y) && \text{optimality of } F_1 \text{ for } \ell_1\text{-EMD} \\
&= \Omega\left(\frac{1}{d}\right) \cdot EMD(X,Y).
\end{aligned}$$

Distance expansion: we have in expectation,

$$\begin{aligned}
\mathbb{E}\left[EMD_T(X,Y)\right] &= \mathbb{E}\left[\sum_{x,y} T(x,y) \cdot F_T(x,y)\right] \\
&\leq \mathbb{E}\left[\sum_{x,y} T(x,y) \cdot F_1(x,y)\right] && \text{optimality of } F_T \text{ for tree-EMD} \\
&= \sum_{x,y} \mathbb{E}[T(x,y)] \cdot F_1(x,y) \\
&\leq O(\log(\Delta d)) \cdot \sum_{x,y} \|x-y\|_1 \cdot F_1(x,y) && \text{second bullet of Theorem 4.1} \\
&= O(\log(\Delta d)) \cdot EMD(X,Y), && (1)
\end{aligned}$$

and therefore, by Markov's inequality, we have

$$\Pr[EMD_T(X,Y) \leq O(\log(\Delta d)) \cdot EMD(X,Y)] \geq 2/3.$$

□

Theorem 7.1 follows from Lemmas 7.2 and 7.3.

8 Embedding EMD into ℓ_1

An interesting consequence of the previous section is that the EMD metric itself, as a metric space, embeds into ℓ_1 with some distortion. The next theorem is based on results from [KT02, Cha02].

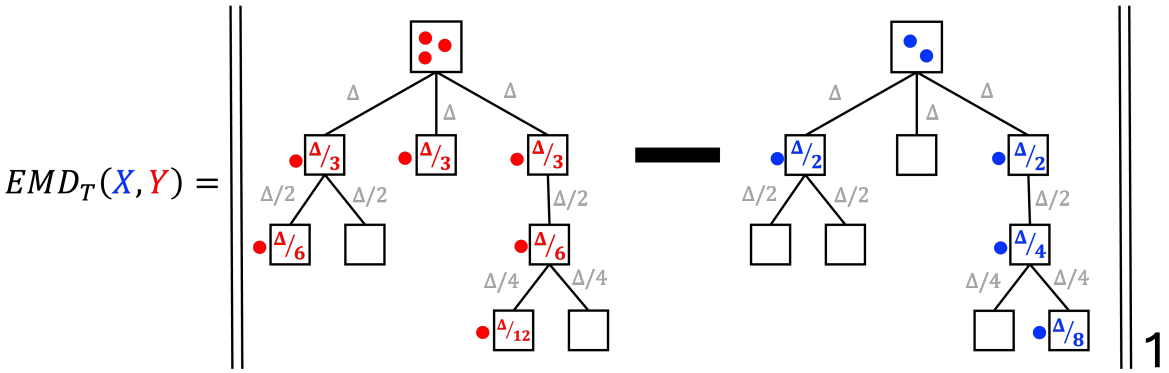

Figure 6: Illustration embedding tree-EMD into ℓ_1. Each tree node except for the root becomes a vector coordinate, and its value is the total mass of points in the hypercube associated with the node, times the weight of the edge leading to the node.

Theorem 8.1. *The EMD metric with ℓ_1 as the ground metric, over subsets of points with coordinates $[0, \Delta)$ and minimum distance 1, embeds into ℓ_1 with distortion $O(d \log(\Delta d))$.*

Note what Theorem 8.1 means: for every two subsets $X \subset \mathbb{R}^d$ we can define just a single real-valued vector $\varphi(X)$, such that if X, Y are subsets that satify the assumptions in the theorem, then $EMD(X,Y) \approx \|\varphi(X) - \varphi(Y)\|_1$, where "$\approx$" here is up to distortion $O(d \log(d\Delta))$ as stated in the theorem.

Let us highlight two special cases of Theorem 8.1 that more often appear in the literature. One is to restrict out points to have integer coordinates in $\{0, \ldots, \Delta - 1\}$. Theorem 8.1 implies that the ℓ_1-EMD distance on subsets of the integer lattice $[\Delta]^{\otimes d}$ embeds into ℓ_1 with distortion $O(d \log(\Delta d))$.

The second special case is the further specialization of the integer lattice to just the unit hypercube $\{0,1\}^d$, on which the ℓ_1 metric is the Hamming distance. Theorem 8.1 implies that the EMD on the d-dimensional hypercube endowed with the Hamming distance embeds into ℓ_1 with distortion $O(d \log d)$. This is known to be nearly optimal: [KN06] show that any embedding of EMD over the Hamming hypercube into ℓ_1 must have distortion $\Omega(d)$.

8.1 Embedding tree-EMD into ℓ_1

We will prove Theorem 8.1 slightly informally to avoid getting bogged down by measure-theoretic and other details, though the argument can be easily formalized. We first show that the tree-EMD metric of any fixed tree T embeds isometrically into ℓ_1.

Lemma 8.2. *Let T be a tree metric. For every subset of leaves X endowed with a distribution $\mathcal{D}_X$, there is a vector $\varphi_T(X) \in \mathbb{R}^{N_T - 1}$, where N_T is the number of nodes in T, such that for every two subsets X, Y it holds that*

$$EMD_T(X,Y) = \|\varphi_T(X) - \varphi_T(Y)\|_1.$$

I will only sketch the proof (also literally sketch it in a drawing, see Figure 6), because I think it is easier to stare and realize than to read. Each node v in the tree except for the root becomes a vector coordinate. Given a subset of leaves X, the value of the coordinate associated with v in the vector $\varphi_T(X)$ equals the total $\mathcal{D}_X$-mass of its leaves times the weight of the edge leading to it. This is what's illustrated (or sketched) in Figure 6.

A good way to see that $EMD_T(X,Y) = \|\varphi_T(X) - \varphi_T(Y)\|_1$ is to revisit the greedy tree-EMD computation algorithm from Lemma 7.2. Take a node v and let $w(v)$ the weight of the edge leading to it (in our quadtree we'd have $w(v) = 2^\ell$ where ℓ is the tree level where v is). Let $m_X(v)$ and $m_Y(v)$ be the total $\mathcal{D}_X$-mass and $\mathcal{D}_Y$-mass in v respectively. If we greedily match all possible mass inside v, which has zero cost, the excess flow that remains to handle is $|m_X(v) - m_Y(v)|$. We must ship it upward along the edge leading to v, and thus pay for it $w(v)|m_X(v) - m_Y(v)|$ in the tree-EMD cost. So, if we define $\varphi_T(X)$ to have $w(v) \cdot m_X(v)$ in its v-coordinate, and similarly define $\varphi_T(Y)$ with $w(v) \cdot m_Y(v)$ in its v-coordinate, then $|\varphi_T(X)_v - \varphi_T(Y)_v| = w(v)|m_X(v) - m_Y(v)|$ is precisely the contribution of node v to the total tree-EMD cost. Therefore,

$$\|\varphi_T(X) - \varphi_T(Y)\|_1 = \sum_v |\varphi_T(X)_v - \varphi_T(Y)_v| = EMD_T(X,Y).$$

This is not quite a formal proof, but it should be enough to convince yourselves that the proof can be completed.

8.2 Embedding a distribution over tree-EMDs into ℓ_1

Now we can prove Theorem 8.1 using Lemma 8.2. Note that Theorem 8.1 is a mathematical embeddability statement; we are not concerned with efficiency for this theorem, so we won't worry about bounding the embedding dimension; we'll even let it be infinite.

Our goal is to define a vector $\varphi(X)$ for every $X \subset \mathbb{R}^d$ such that $\|\varphi(X) - \varphi(Y)\|_1$ approximates $EMD(X,Y)$. What we have is a probabilistic embedding $\mathcal{T}$ into quadtrees. For every quadtree T in the support of $\mathcal{T}$, let $p(T)$ denote the probability mass of sampling it.[1] We define the vector $\varphi(X)$ as the (possibly infinite) concatenation of all the vectors $p(T) \cdot \varphi_T(X)$ for all of the quadtrees T in the support of $\mathcal{T}$. This yields Theorem 8.1 because,

$$\begin{aligned}
\|\varphi(X) - \varphi(Y)\|_1 &= \int_{\mathcal{T}} \|p(T) \cdot \varphi_T(X) - p(T) \cdot \varphi_T(Y)\|_1 dT && \text{by coordinate-additivity of } \ell_1 \\
&= \int_{\mathcal{T}} p(T) \cdot EMD_T(X,Y) && \text{Lemma 8.2} \\
&= \mathbb{E}_{T \sim \mathcal{T}}[EMD_T(X,Y)] && \text{by definition of expectation,}
\end{aligned}$$

and we already know that $\mathbb{E}_{T \sim \mathcal{T}}[EMD_T(X,Y]$ is an approximation of $EMD(X,Y)$ up to a factor of $O(d \log(\Delta d))$ from the proof of Theorem 7.1.

8.3 Application: Approximate nearest neighbor search for EMD

Even though above we treated Theorem 8.1 as a "mathematical" statement, it still has algorithmic implications. I will just mention one briefly. Consider the approximate nearest neighbor search problem where the metric that defines "nearest" is EMD. We have already seen in previous lectures that ANN can be quite difficult to do efficiently even in "simpler" metrics like ℓ_2 and ℓ_1, so you can imagine how complex it can get with EMD.

However, since we now have an embedding of EMD into ℓ_1, we could hope to implement some algorithmic form of the embedding and then use the LSH-based ANN algorithms we have seen for ℓ_1. This was done in [IT03] for fast image retrieval, since one common application of EMD is to define a measure of distance between images.

[1] Here we are being slightly informal because of continuity; to formalize this argument we'd need to either treat the distribution over quadtrees as properly continuous, or discretize it. Both can be made to work.

9 Improving the distortion of EMD with quadtrees

Our last result for this lecture will be an improvement to Theorem 7.1. The distortion in that theorem was $O(d \log(\Delta d))$. That linear dependence on d is an eyesore, so we are going to improve it to $O(\log n \cdot \log(\Delta d))$. The result is due to [AIK08]. A simplified presentation more in line with these lecture notes appears in [BDI+20].

We did say earlier in our "typical setting" that $d = O(\log n)$. So what's the point in "improving" d to $\log n$? There are two reasons. One is that the "typical" setting only works out if our ℓ_1 metric was originally an ℓ_2 metric embedded with $\ell_2 \to \ell_1$ JL. If the points are originally in ℓ_1, then d can be much larger than $\log n$, since there is no dimension reduction theorem for ℓ_1.

The second and more important reason is that we are now dealing with EMD. Early in this lecture we were still dealing with ℓ_2 and ℓ_1 metric spaces, where n was the overall number of points. However, in our EMD setting, n is the maximum number of points in a single "superpoint" $X \subset \mathbb{R}^d$. In an EMD metric space we would typically have some large number N of superpoints – for example, the number of superpoints in a dataset over which we need to solve EMD-ANN queries. Even if the ground metric is originally ℓ_2, in order to preserve all distances between the points contained in all superpoints, we need the reduced JL dimension to be $\log(Nn)$, not $\log n$. Thus, in the context of EMD, the improvement of d to $\log n$ is indeed a meaningful improvement.

Having motivated it, we state the final theorem of the lecture.

Theorem 9.1. *Given X, Y, we can compute in time $O(nd \log(\Delta d))$ an estimate $\widetilde{EMD}(X, Y)$ that with probability $2/3$ approximates $EMD(X, Y)$ up to distortion $O(\log n \cdot \log(\Delta d))$.*

The algorithm is, of course, yet again our quadtree. In fact, the algorithm is nearly identical to the one from Theorem 7.1: we probabilistically embed our $X \cup Y$ in a quadtree, compute the tree-EMD on the quadtree, and return the result as our estimate. Theorem 9.1 will follow from a somewhat different analysis of nearly the same algorithm.

The only difference in the algorithm is that we start the quadtree construction with a larger enclosing hypercube. We choose $R = \Theta(\Delta d)$, sample the random shift s with uniform i.i.d. coordinates in $[0, R)$, and start the quadtree construction with an enclosing hypercube of side-length $2R$, instead of 2Δ as in the previous sections. The number of levels is now $L' = \log(Rd) = O(\log(\Delta d))$, which is asymptotically the same as our previous number of levels $L = \log(4\Delta d)$.

We now prove Theorem 9.1. The proof of the expected distance expansion bound is the same as in Theorem 7.1. We showed there in Equation (1) that $\mathbb{E}[T(x, y)] \leq O(L) \cdot \|x - y\|_1$. Repeating the same argument with our larger quadtree, we get $\mathbb{E}[T(x, y)] \leq O(L') \cdot \|x - y\|_1$ with our new number of levels L'. Since L and L' are the same up to a constant, we get the same bound asymptotically,

$$\mathbb{E}[EMD_T(X, Y)] \leq O(\log(\Delta d)) \cdot EMD(X, Y).$$

Therefore, by Markov's inequality,

$$\Pr\left[EMD_T(X, Y) \leq O(\log(\Delta d)) \cdot EMD(X, Y)\right] \geq 0.9. \tag{2}$$

What we will do differently now is the contraction bound. Let $x \in X$ and $y \in Y$. Way back in Section 4.4, we defined $Sep(x, y, \ell)$ as the event that x, y are separated in level ℓ of the quadtree. We proved the upper bound $\Pr[Sep(x, y, \ell)] \leq 2^{-\ell}\|x - y\|_1$. Now we will prove a lower bound. For $Sep(x, y, \ell)$ to not occur, x, y must not be separated in any axis in level ℓ. Therefore,

$$1 - \Pr[Sep(x, y, \ell)] = \prod_{i=1}^{d} \left(1 - \Pr[x, y \text{ are separated at axis } i \text{ at level } \ell]\right).$$

In Section 4.4 we showed that

$$\Pr[x, y \text{ are separated at axis } i \text{ at level } \ell] = \min\{1, \frac{|x_i - y_i|}{2^\ell}\}.$$

If at any axis i we have $|x_i - y_i| \geq 2^\ell$ then x, y must be separated at level ℓ, hence $\Pr[Sep(x, y, \ell)] = 1$. Therefore we may assume that $|x_i - y_i| < 2^\ell$ for all $i = 1, \ldots, d$. Thus,

$$\Pr[x, y \text{ are separated at axis } i \text{ at level } \ell] = 2^{-\ell}|x_i - y_i|.$$

Plugging this above, and using the fact that $1 - a \leq e^{-a}$ for all $a \in \mathbb{R}$,

$$\begin{aligned}
\Pr[Sep(x, y, \ell)] &= 1 - \prod_{i=1}^{d} \left(1 - 2^{-\ell}|x_i - y_i|\right) \\
&\geq 1 - \prod_{i=1}^{d} e^{-2^{-\ell}|x_i - y_i|} \\
&= 1 - e^{-2^{-\ell}\sum_{i=1}^{d}|x_i - y_i|} \\
&= 1 - e^{-2^{-\ell}\|x-y\|_1}. \qquad (3)
\end{aligned}$$

Now, let $\eta = 1/(5n^2)$. Define $\ell(x, y, \eta)$ as

$$\ell(x, y, \eta) := \lfloor \log \left(\frac{\|x - y\|_1}{2 \ln(1/\eta)} \right) \rfloor.$$

We consider whether $\ell(x, y, \eta)$ is a level that exists in our quadtree. We argue that it cannot be too large for that, and this is where we need those extra levels we added to the top of the quadtree by starting with a larger enclosing hypercube. $\|x - y\|_1$ is at most the ℓ_1-diameter of our points, which is at most $d\Delta$. Therefore, $\ell(x, y, \eta)$ is at most $\Theta(\log(d\Delta/\ln(1/\eta)))$. Our enlarged quadtree starts at level $\Theta(\log(d\Delta))$, ensuring that $\ell(x, y, \eta)$ is not larger than the top level in the quadtree.

We now have two cases to handle. In the first case, $\ell(x, y, \eta)$ is smaller than the smallest level in the quadtree, which we recall is $\log(1/d)$. Together with our minimum distance assumption $\|x - y\|_1 \geq 1$, this implies that $d = O(\ln(1/\eta)) = O(\log n)$. By first bullet of Theorem 4.1, we have deterministically,

$$T(x, y) \geq \Omega\left(\frac{\|x - y\|_1}{d}\right) = \Omega\left(\frac{\|x - y\|_1}{\log n}\right). \qquad (4)$$

In the second case, $\ell(x, y, \eta)$ is a level in the quadtree. Plugging its value in Equation (3), we get

$$\Pr[Sep(x, y, \ell(x, y, \eta))] \geq 1 - e^{-2^{-\ell(x,y,\eta)}\|x-y\|_1} \geq 1 - \eta. \qquad (5)$$

If $Sep(x, y, \ell(x, y, \eta))$ occurs then, as per Section 4.2, the quadtree distance between x and y satisfies

$$T(x, y) \geq 2^{\ell(x,y,\eta)} \geq \frac{\|x - y\|_1}{2\log(1/\eta)} = \Omega\left(\frac{\|x - y\|_1}{\log n}\right). \qquad (6)$$

We take a union bound in Equation (5) over all pairs $x \in X, y \in Y$ for which the second case holds. By our choice $\eta = 1/(5n^2)$ we get that $Sep(x, y, \ell(x, y, \eta))$ occurs for all of them simultaneously with probability $4/5$. Under this event, by Equations (4) and (6), we have

$$\text{for all } x \in X \text{ and } y \in Y, \quad T(x, y) \geq \Omega\left(\frac{\|x - y\|_1}{\log n}\right).$$

Therefore, again denoting by F_T the optimal flow for tree-EMD and by F_1 the optimal flow for EMD, we have

$$\begin{aligned} EMD_T(X,Y) &= \sum_{x,y} T(x,y) \cdot F_T(x,y) \\ &\geq \Omega\left(\frac{1}{\log n}\right) \sum_{x,y} \|x-y\|_1 \cdot F_T(x,y) \\ &\geq \Omega\left(\frac{1}{\log n}\right) \sum_{x,y} \|x-y\|_1 \cdot F_1(x,y) \\ &= \Omega\left(\frac{1}{\log n}\right) \cdot EMD(X,Y). \end{aligned}$$

This occurs with probability 0.8. By a union bound, this occurs simultaneously with Equation (2) with probability 0.7. Together they imply Theorem 9.1.

References

[AF09] Reid Andersen and Uriel Feige, *Interchanging distance and capacity in probabilistic mappings*, arXiv preprint arXiv:0907.3631 (2009).

[AIK08] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer, *Earth mover distance over high-dimensional spaces.*, SODA, vol. 8, 2008, pp. 343–352.

[AKPW95] Noga Alon, Richard M Karp, David Peleg, and Douglas West, *A graph-theoretic game and its application to the k-server problem*, SIAM Journal on Computing **24** (1995), no. 1, 78–100.

[ARV09] Sanjeev Arora, Satish Rao, and Umesh Vazirani, *Expander flows, geometric embeddings and graph partitioning*, Journal of the ACM (JACM) **56** (2009), no. 2, 1–37.

[Bar96] Yair Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, Proceedings of 37th Conference on Foundations of Computer Science, IEEE, 1996, pp. 184–193.

[Bar98] ———, *On approximating arbitrary metrices by tree metrics*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998, pp. 161–168.

[BDI+20] Arturs Backurs, Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner, *Scalable nearest neighbor search for optimal transport*, International Conference on machine learning, PMLR, 2020, pp. 497–506.

[Cha02] Moses S Charikar, *Similarity estimation techniques from rounding algorithms*, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 2002, pp. 380–388.

[FK02] Uriel Feige and Robert Krauthgamer, *A polylogarithmic approximation of the minimum bisection*, SIAM Journal on Computing **31** (2002), no. 4, 1090–1118.

[FRT03] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, 2003, pp. 448–455.

[IT03] Piotr Indyk and Nitin Thaper, *Fast image retrieval via embeddings*, 3rd international workshop on statistical and computational theories of vision, vol. 1, Nice, France, 2003.

[Kar89] Richard M Karp, *A 2k-competitive algorithm for the circle*, Manuscript, August **5** (1989), no. 11.

[KN06] Subhash Khot and Assaf Naor, *Nonembeddability theorems via fourier analysis*, Mathematische Annalen **334** (2006), no. 4, 821–852.

[KT02] Jon Kleinberg and Eva Tardos, *Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields*, Journal of the ACM (JACM) **49** (2002), no. 5, 616–639.

[LR99] Tom Leighton and Satish Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, Journal of the ACM (JACM) **46** (1999), no. 6, 787–832.

[Räc08] Harald Räcke, *Optimal hierarchical decompositions for congestion minimization in networks*, Proceedings of the fortieth annual ACM symposium on Theory of computing, 2008, pp. 255–264.

[RR98] Yuri Rabinovich and Ran Raz, *Lower bounds on the distortion of embedding finite metric spaces in graphs*, Discrete & Computational Geometry **19** (1998), no. 1, 79–94.

[WS11] David P Williamson and David B Shmoys, *The design of approximation algorithms*, Cambridge university press, 2011.