An Algorithmist's Toolkit Brown CSCI 2952T – Spring 2026

Lecture 11: Distance, Routing Sparsification via Tree Embeddings

April 15, 2026 Lecturer: D Ellis Hershkowitz Scribe: Anand A., Aamen M., Phuoc P.V.L.

Generally, today's lecture will be about saying that any problem you have on a graph you can solve on a tree, and then we'll use that to do a problem called "oblivious routing".

1 Probabilistic Tree Embeddings with $O(\log n \log \Delta)$ distortion

Given any metric, one can find a "good enough" embedding into a tree metric, which would be the goal of this section. An embedding, intuitively, maps a set along with its metric into a *larger* set and metric, which means embedding into a subset, while trying to preserve (in a certain ratio, called *distortion*) the distance between the mapped points.

Definition 1 (Embedding)**.** *Recall an* ***embedding*** *of a metric (V, d) into metric (V', d') is a function $f : V \to V'$. f has* ***distortion*** *α if:*

$$d(u, v) \leq d'(f(u), f(v)) \leq \alpha \cdot d(u, v) \, \forall u, v \in V.$$

The embedding f is ***isometric*** *if $\alpha = 1$.*

A tree metric can be embedded exactly into a tree with distance between two points being their shortest path.

Definition 2 (Tree metric)**.** *A metric (V, d) is a* ***tree metric*** *if it embeds isometrically into (V, d_T) where d_T is the shortest path metric of some edge-weighted tree that is a superset of V.*

In general, it is impossible to embed any metric into a tree metric isometrically, because we can have cycles in graphs that will disrupt any tree embedding. In Figure 1, we can see that whichever edge being removed would turn the pair of end nodes of this edge into one with distance $n - 1$, hence the distortion is also $n - 1$. However, it has been studied extensively how we can do the embedding with low distortion, specifically $O(\log n)$, where n is the number of points in V. In this lecture, we will show a *probabilistic* algorithm to do so.

Definition 3. *An* ***α-distortion probabilistic tree embedding*** *of (V, d) is a distribution $\mathcal{T}$ on trees such that*

1. $d(u, v) \leq d_T(u, v) \, \forall T \in \mathcal{T} \, \forall u, v$
2. $\mathbb{E}_{T \sim \mathcal{T}}[d_T(u, v)] \leq \alpha \cdot d(u, v) \, \forall u, v$

Let Δ be the diameter of the set of points. The notion of α-distortion probabilistic tree embedding first appeared in [1], which showed a probabilistic embedding with distortion $O(\log n \log \Delta)$. The

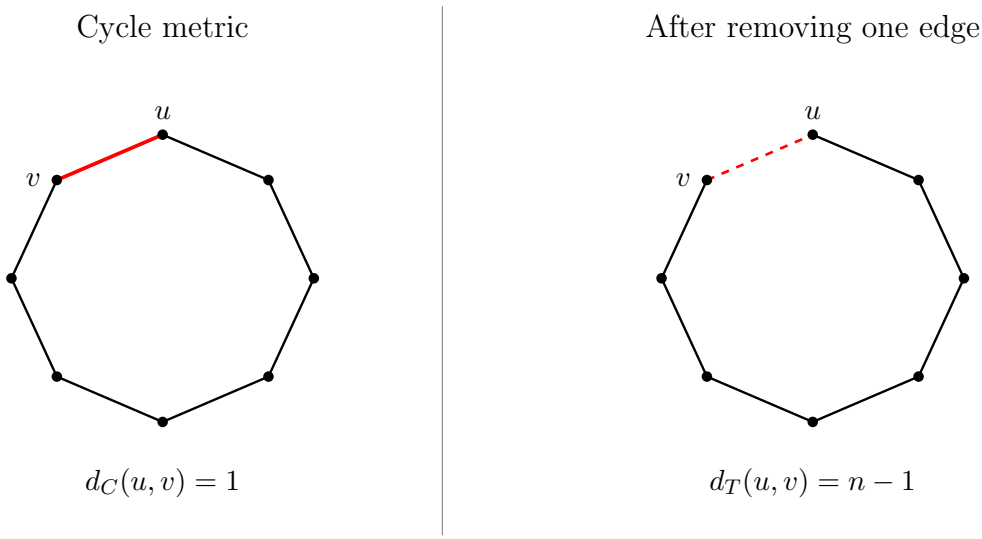

Figure 1: Deterministic tree embedding for a cycle.

dependency on diameter is undesirable, and is removed in [3] into an $O(\log n)$-distortion embedding. In this section, we will explain the tree embedding that achieves $O(\log n \log \Delta)$ distortion as a warm-up first.

Theorem 1 (Warm-up)**.** *Every metric (V, d) where $|V| = n$ has a probabilistic tree embedding $\mathcal{T}$ with distortion $O(\log n)$.*

Structure of the output tree embedding. As the first step of understanding this algorithm, we first see what the resulting tree would look like. The outcome would be a rooted tree with $\log \Delta$ layers (where the layers are 1-indexed, so the root is in the first layer), with every node at the i-th layer would have edge weight to any of their children being $\Delta/2^i$. That means, we start from the distance between the root and any of its children being $\Delta/2$, and half the edge weight layer by layer as we go down the tree. The leaves nodes of the tree would be exactly corresponding to each point in the original set. An illustration for this tree is in Figure 2.

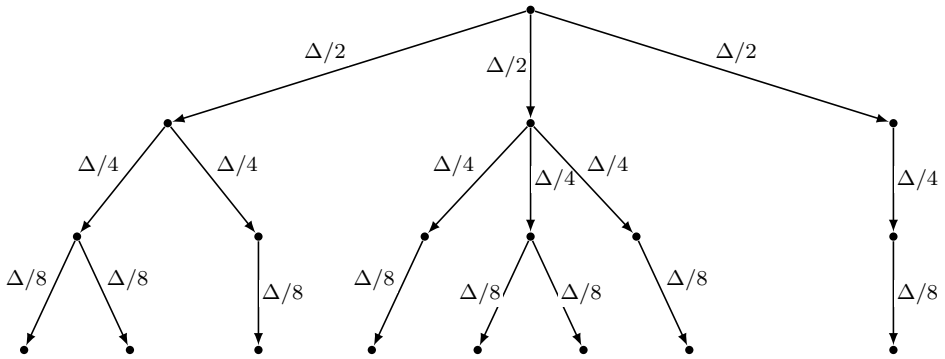

Figure 2: Output tree of the FRT [3] algorithm.

High-level idea of how to create this tree. The tree is created by iteratively clustering the set of points, where in each layer, a cluster would be further *refined* into smaller sub-clusters. The

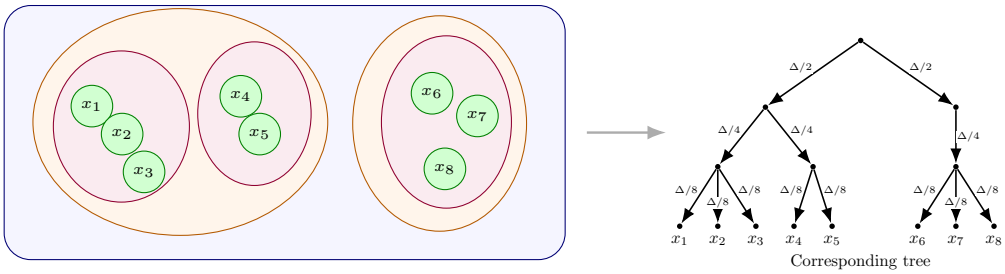

Figure 3: A set of eight points with a hierarchical clustering and the corresponding tree.

clusters created in the i-th step would have diameter $\Delta/2^i$. In the final step, the set of points should be fully partitioned, that is every point is in its own cluster for this last layer. An illustration for this clustering into tree algorithm is in Figure 2, which gives one more intuition: for each two leaves, its lowest common ancestor is exactly the cluster that contains both of the original two points.

Claim 1. *Let T be any tree produced by the hierarchical clustering procedure illustrated in Figure 1. Then for every $u, v \in V$,*

$$d(u, v) \leq d_T(u, v).$$

Hence the embedding is non-contractive, i.e. it satisfies property 1 of Definition 3.

Proof. Fix $u, v \in V$, and let i be the first level at which u and v lie in different clusters. Equivalently, at level $i-1$ they still lie in the same cluster, but when that cluster is refined at level i, they are separated. Since u and v belong to the same cluster at level $i-1$, and every cluster at that level has diameter at most $\Delta/2^{i-1}$, we get

$$d(u, v) \leq \frac{\Delta}{2^{i-1}}. \tag{1}$$

In the tree $\mathcal{T}$, the lowest common ancestor of the leaves corresponding to u and v is exactly the node corresponding to their common cluster at level $i-1$. Therefore, in the tree, the path from u to v must go from u up to that ancestor and then back down to v, and contains at least the two edges from level $i-1$ down to level i Each such edge, one on the side of u and one on the side of v, has length $\Delta/2^i$, so

$$d_T(u, v) \geq 2 \cdot \frac{\Delta}{2^i} = \frac{\Delta}{2^{i-1}}. \tag{2}$$

Combining inequalities (1) and (2), we get $d(u, v) \leq d_T(u, v)$ as required. □

The end-goal: What do we want in each layer? Turns out the previously described tree embedding just needs one more property from the clustering algorithm in each layer, to achieve the $O(\log n \log \Delta)$ distortion. On the i-th tree layer, recall that each cluster would have diameter $D = \Delta/2^i$. For two points u and v in a cluster, with distance being $d(u, v)$, we want the probability that they get separated in the next step to be at most:

$$\log n \cdot \frac{2}{D} \cdot d(u, v).$$

If you recall from previous lectures, this is in fact just a *low diameter decomposition*. Many algorithms for separation parameter $O(\log n)$ had been provided [1, 2, 4], and in the previous lecture the algorithm explained is in [2]. Now the rest of the work is to prove that the tree embedding has distortion $O(\log n)$, using the mentioned property.

Claim 2. *Using low diameter decomposition with separation $O(\log n)$, the tree embedding in Algorithm 1 achieves $O(\log n)$ distortion.*

Proof. Consider two points u and v. If the first time they are separated at level i, then by summing up maximally all edges on two paths from the common ancestor to u and v, we obtain $d_T(u, v)$ to be at most:

$$2\left(\frac{\Delta}{2^i} + \frac{\Delta}{2^{i+1}} + \ldots + \frac{\Delta}{2^{\lceil \log_2 \Delta \rceil}}\right) \leq 2\frac{\Delta}{2^{i-1}}.$$

Recalling the probability that u and v get separated at level i given that they do not get separated at level $i-1$ is $\log n \frac{2^{i+1}}{\Delta} d(u, v)$, we also have the probability that u and v being first separated at level i is at most $\frac{2^{i+1}}{\Delta}$. Let this event be denoted by A_i, we can calculate the expected embedded distance of u and v by the law of total expectation:

$$\begin{aligned}
E\left[d_T(u, v)\right] &= \sum_{i=1}^{\log_2 \Delta} \Pr\left[A_i\right] \cdot E\left[d_T(u, v) \mid A_i\right] \\
&\leq \sum_{i=1}^{\log_2 \Delta} \log n \frac{2^{i+1}}{\Delta} d(u, v) \cdot 2\frac{\Delta}{2^{i-1}} \\
&= \log_2 \Delta \cdot \log n \cdot 8d(u, v) \\
&= O(\log \Delta \log n) d(u, v).
\end{aligned}$$

□

2 Improved algorithm to get $O(\log n)$ distortion

The work [3] provides a more optimal distribution that achieves $O(\log n)$ distortion. The main thing to focus on this algorithm is a better analysis on the *separation* events of any two points u, v. The previous protocol treats every layer independently, hence the separation probability actually adds up and results in the $\log \Delta$ loss.

Cluster radius in each layer. The algorithm introduces a starting radius $\beta \in [1, 2)$, that would double as we go up the tree. At the i-th level (but from the leaves now), a cluster would have radius $\beta 2^i$. For any two points u, v and a given center w, it is only possible for a cluster centered at w with radius $\beta 2^i$ to separate u and v at an interval of values for β. More concretely, for a cluster of radius $\beta 2^i$, centered at w to separate u and v, one needs the radius to lie between $d(u, w)$ and $d(v, w)$, which is an interval of size $|d(u, w) - d(v, w)|$. By the triangle inequality, we have:

$$|d(u, w) - d(v, w)| \leq d(u, v). \tag{3}$$

How the clustering works. The algorithm fixes the same permutation π for every layer, where π permutes the set of points. In each layer, a point u would belong to the cluster centered by the first point w in the permutation with distance from u less than $\beta 2^i$, or formally the first point w in the permutation with $d(u,w) \leq \beta 2^i$. We also care about when this point would separate u from another point v for the first time, which only happens when:

$$d(u,w) \leq \beta 2^i < d(w,v),$$

and furthermore, this should be the *first time* that $\beta 2^i$ exceeds $d(u,w)$, which only happens *once* for each value of β. Hence for each value of β, we now have a more fine-grained analysis of the probability that two points u and v are separated, by analyzing each other center (with different positions in the permutation) differently. In fact, since $\beta \in [1,2)$, there is a single value of i that would be the smallest number such as $\beta 2^i \geq d(u,w)$, for *any* value of β.

Probability that u and v are first separated by a center w, at the corresponding unique level i. At this level i, the center w would be able to separate u and v if and only if w is the first point in the permutation with distance from u being less than $\beta 2^i$. That means any point that has distance from u being less than $d(u,w)$ should arrive later in the permutation than w. Suppose w is the j-th away point from u, by simple math we get the probability that w arrive before every closer points is $\frac{1}{j}$. From inequality (3), we further have the range of β such that w separates u and v at level i is at most $\frac{d(u,v)}{2^i}$ in size, hence multiplying the two probabilities together we get:

$$\begin{aligned}&\Pr[w \text{ separates } u,v \text{ at level } i] \cdot \Pr[w \text{ is the first to separate } u,v] \\ &\leq \frac{1}{j} \cdot \frac{d(u,v)}{2^i}.\end{aligned}$$

Note also that when u and v are separated at level i, by a similar proof as Claim 2, we have the tree distance of u and v is at most:

$$d_T(u,v) \leq 2\left(\beta + 2\beta + \ldots + 2^i\beta\right) \leq 2^{i+2}\beta \leq 2^{i+3}.$$

Hence the expected tree distance of u and v if they are separated by w at most:

$$\frac{1}{j} \cdot \frac{d(u,v)}{2^i} 2^{i+3} = \frac{1}{j} \cdot 8d(u,v).$$

We have finally proven the $O(\log n)$ distortion by simply summing up over all j, and by:

$$\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{n} \approx \log n.$$

3 Oblivious Routing

Recall: Given graph $G = (V,E)$, a **demand** is a function $D : V \times V \to \{0,1\}$. Paths $\mathcal{P}$ **route** D if $\forall u,v$ such that $D(u,v) = 1$, there exists a path $u \rightsquigarrow v$ path $\in \mathcal{P}$. The **congestion** of e with respect to $\mathcal{P}$ is $\text{Con}_{\mathcal{P}}(e) := |\{P \in \mathcal{P} : e \in P\}|$. The congestion of $\mathcal{P}$ is $\text{Con}_{\mathcal{P}} := \max_e \text{Con}_{\mathcal{P}}(e)$.

Algorithm 1: FRT probabilistic tree embedding

Input: A finite metric space (V, d)

Output: A random rooted tree metric T with leaves V

`// Initialize`

Rescale so that the minimum nonzero distance is at least 1;
$\Delta \leftarrow \max_{u,v \in V} d(u, v)$;
$L \leftarrow \lceil \log_2 \Delta \rceil$;
Sample $\beta \sim \text{Unif}[1, 2)$;
Sample a uniformly random permutation $\pi = (v_1, \ldots, v_n)$ of V;

`// Top partition`

$\mathcal{P}_{L+1} \leftarrow \{V\}$; Create a root node r_V corresponding to the cluster V;
for $k \leftarrow L$ **to** 0 **do**
    $\mathcal{P}_k \leftarrow \emptyset$;
    $U \leftarrow V$; `// points not yet assigned at level` k
    for $j \leftarrow 1$ **to** n **do**
        $C \leftarrow \{x \in U : d(x, v_j) < \beta \cdot 2^{k-2}\}$;
        if $C \neq \emptyset$ **then**
            Add C to $\mathcal{P}_k$;
            $U \leftarrow U \setminus C$;

for $k \leftarrow L$ **to** 0 **do**
    foreach $C \in \mathcal{P}_k$ **do**
        Let $D \in \mathcal{P}_{k+1}$ be the unique cluster with $C \subseteq D$;
        Create a node r_C corresponding to C;
        Add an edge (r_D, r_C) of length 2^{k-1};

Identify each singleton cluster $\{v\} \in \mathcal{P}_0$ with the leaf v;
return T;

Minimum Congestion Routing Problem: given demand D and collection of paths $\mathcal{P}$, find paths $\mathcal{P}_D \in \mathcal{P}$ minimizing $\text{Con}(\mathcal{P})$.

α-competitive randomized algorithm: an algorithm so that

$$\mathbb{E}(\text{Con}_{\mathcal{P}_D}(e)) \leq \alpha \cdot OPT_D \quad \forall D, e \in E$$

where OPT_D denotes optimum congestion for a given demand D

We can get a 1-competitive algorithm using what we already know - just by writing down the problem as an LP!

Let X_P be the probability that we pick a given path $P = (s, ..., t) \in \mathcal{P}_D$.

We want to minimize Z (the max congestion of any edge) such that:

$$Z \geq \sum_{P: e \in P} X_P \quad \forall e \in E$$

and

$$1 = \sum_{P=(s,..,t)} X_P \quad \forall s,t : D(s,t) = 1$$

$$X_P \in [0,1] \quad \forall P$$

where the first constraint just states that Z is the max congestion and the second constraint says that our sampling is from a probability distribution.

The problem with this strategy is that if we are adding or removing new edges to the network in an actual application, we'll re-calculate our LP every time, which will be very slow and not able to keep up with changes in the network. So we want something a little more robust.

Theorem 2. *Every graph has an $\tilde{O}(\log n)$-competitive oblivious routing algorithm.*

Recall that $\tilde{O}(f(n))$ means we're hiding poly-log$(f(n))$ factors, so here this is equivalent to $O(\log n \cdot poly(\log\log n))$.

Corollary. Given $(V,E,w), \exists$ spanning tree $T^* \subseteq G$ s.t. $\sum_{\{u,v\}\in E} d_{T^*}(u,v) \leq \tilde{O}(\log n)\cdot\sum_{u,v\in E} w(\{u,v\})$.

This corollary follows from linearity of expectation:

$$\mathbb{E}_{T\sim\mathcal{T}}\left[\sum_{\{u,v\}\in E} d_T(u,v)\right] \leq \tilde{O}(\log n)\cdot\sum_{\{u,v\}\in E} w(\{u,v\}).$$

(Since a given tree achieves this in expectation, there must be one tree that achieves it.)

Definition 4. *We say an oblivious routing algorithm (ORA) is **tree-based** if there exists a distance over spanning trees $\mathcal{T}$ such that*

$$\mathcal{P}_{st} := T(s,t) \text{ for } T \sim \mathcal{T}$$

That is, you pick a tree from your distribution of trees (PTE), and route along that tree. Do this a few times for different sampled trees.

Definition 5. *For a spanning tree T, the **load on edge** e is $L_T(e) := |\delta_G(T\{e\})|$. Intuitively, if you delete that edge, you're left with two connected components in the spanning tree: treat this as a cut. The load is the number of edges in the original graph that cross this cut.*

Definition 6. *If $\mathcal{T}$ is a distribution over spanning trees, the **load on** e is the expected load $L_{\mathcal{T}}(e) := \mathbb{E}_{T\sim\mathcal{T}}[L_T(e)]$. We also say for the overall PTE that the **load of** $\mathcal{T}$ is the* $\max_e L_{\mathcal{T}}(e)$.

Claim: Any tree-based ORA with distribution $\mathcal{T}$ is $L(\mathcal{T})$-competitive.

Proof. Fix a demand D and a tree and edge from the distribution, $T \in \mathcal{T}$, $e \in T$, and let

$$D(T\{e\}) := \sum_{u\in T\{e\}}\sum_{v\notin T\{e\}} D(u,v) + D(v,u).$$

Note that the optimal routing $OPT_D \geq D(T\{e\})/L_T(e)$ (by an averaging argument). We can rearrange this to get an upper bound on $D(T\{e\})$.

Now we're pretty much done:

$$\begin{aligned}\mathbb{E}[\mathrm{Con}_{\mathcal{P}_D}(e)] &= \sum_{T\in\mathcal{T}} P(T)\cdot D(T\{e\}) \\ &\leq \sum_{T\in\mathcal{T}} P(T)\cdot OPT_D \cdot L_T(e) \\ &= OPT_D \cdot L_{\mathcal{T}}(e) \\ &\leq OPT_D \cdot \max_e L_{\mathcal{T}}(e) \\ &= OPT_D \cdot L(\mathcal{T}).\end{aligned}$$

. □

now, to prove theorem 2, it remains to show that

$$\exists \mathcal{T} \ni L(\mathcal{T}) \leq \tilde{O}(log(n))$$

this can be done by setting up an LP.

Proof. We let $x_T = Pr_T(\mathcal{T})$ and $z = L(\mathcal{T})$, and set as our goal to minimize z subject to constraints:

$$z \geq \sum_T x_T L_{T(e)} \quad \forall e \in E$$

$$\sum_T x_T = 1$$

$$x_T \geq 0 \quad \forall T$$

Forming the dual of our problem, we let our variables be $w_e \forall e \in E$ and Γ where our goal now is to maximize Γ subject to:

$$w_e \geq 0 \quad \forall e \in E$$

$$\sum_{(u,v)\in E} d_{T(u,v)} \geq \Gamma \sum_{e\in E} w_e \quad \forall T$$

but by previous corollary, we have a spanning tree T that satisfies

$$\sum_{\{u,v\}\in E} d_T(u,v) \leq \tilde{O}(\log n)\cdot \sum_{e\in E} w(e)$$

and thus, by applying this inequality to second constraint we get:

$$\Gamma \sum_{e\in E} w_e \leq \tilde{O}(\log n)\cdot \sum_{e\in E} w(e)$$

and hence that

$$\Gamma \leq \tilde{O}(\log n)$$

by noting our initial setop of our LP, this is equivalent to

$$L(\mathcal{T}) \leq \tilde{O}(log(n))$$

but by previous claim, we have shows that

$$\begin{aligned}\mathbb{E}[\mathrm{Con}_{\mathcal{P}_D}(e)] &\leq OPT_D \cdot L(\mathcal{T}) \\ &\leq OPT_D \cdot \tilde{O}(log(n))\end{aligned}$$

which is by definition, $\tilde{O}(log(n))$-competitive. □

References

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

[2] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.

[3] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

[4] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 320–330, 1991.