Pricing Online Metric Matching Algorithms on Trees

Aditya Krishnan

CMU-CS-18-112
August 2018

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
Anupam Gupta, Chair
Anil Ada

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science

Keywords: online algorithm, matching algorithm, metric matching, pricing mechanism, tree metric, monotone

Abstract

The online metrical matching problem is a well studied paradigm in online algorithms. The problem is defined on an underlying metric space with k special points called servers. Requests arrive in an online fashion on the metric space and the objective is to match requests to yet unmatched servers while minimizing the total cost of the matching. Research into posted price algorithms for online metrical matching was initiated in Cohen at al. (2015) as part of a line of research to study the use of posted price algorithms to minimize social cost in a setting with selfish, autonomous agents instead of requests. They gave a post-price algorithm that "mimics" the $\log(k)$-competitive algorithm for line metrics by Gupta and Lewi (2012). We investigate the post-price setting for tree metrics by characterizing the properties of post-price algorithms and discuss how to give a poly-$\log(k)$ competitive post-price algorithm on tree metrics.

Acknowledgments

I have a long list of people to grateful to. Firstly, I am very grateful to my advisor Prof. Anupam Gupta for his guidance, advice and motivation. I wouldn't have been able to complete this thesis without his support. I am especially grateful for his patience with me in times that could have easily led to frustration.

While Prof. Kirk Pruhs is neither my official advisor nor a member of my thesis committee, he has acted as a second advisor to me. I thank Prof. Pruhs for being a supportive mentor and guiding me through times when this problem seemed insurmountable.

More than learning about online algorithms, I learned how to be a better researcher from Prof. Gupta and Prof. Pruhs. I thank them for being helpful mentors.

I would also like to thank Prof. Anil Ada for being a selfless mentor and an invaluable voice of guidance in times when I've needed support as a young researcher. He has inspired me to pursue my further studies in theoretical computer science.

I would be remiss if I didn't acknowledge those who gave me emotional support and guidance to complete this thesis. My parents have been my pillar of support, not only for this year but throughout my life. They are an invaluable source of wisdom on how to maneuver the hardships of life. Thank you Amma and Appa, you have my sincerest gratitude for your unwavering support.

This problem has not only been technically challenging but completing it has been emotionally challenging too. Noshin Nova, thank you for your emotional support through this year and thank you for making my life richer.

My friends have been a great source of academic inspiration. Sidhanth Mohanty, Nicholas Sieger, Ainesh Bakshi and Roie Levin thank you for the many fun discussions. I would especially like to thank Sidhanth Mohanty for being the peer I am inspired by and for being an amazing friend.

Last but not least, I would like to thank Tracy Farbacher for bearing with all the problems and hiccups with my thesis.

Contents

1 Introduction **8**

1.1 Prior Related Work 8

1.2 Our Contribution 9

2 Notation and Preliminaries **10**

2.1 Graph Notation 10

2.2 Preliminary Definitions 11

2.3 Hierarchical Clusterings and Scaled Trees 11

3 Outline **12**

3.1 Priceable $\iff$ Monotone 12

3.2 poly$(\log(k))$-competitive Monotone Algorithm on Tree Metrics 13

3.2.1 Clustering the Input Tree Metric 13

3.2.2 HTMatch: A Divide and Conquer Algorithm 14

3.2.3 Conquer Algorithm on Spiders and Trees 14

3.2.4 poly$(\log(k))$-competitive Algorithm on Tree metrics 14

4 Pricing Monotone Algorithms **15**

4.1 Pricing Deterministic Monotone Algorithms 15

4.2 Randomized Algorithms 16

5 Embedding Trees into Scaled Trees **21**

6 Our Algorithm **22**

6.1 HTMatch: A Divide and Conquer Algorithm 22

6.2 Matching on Spiders 23

6.3 From Spiders to Trees 24

6.4 Analyzing our Algorithm 24

6.4.1 Monotonicity of HTMatch 25

7 Conclusion **26**

1 Introduction

Imagine we had to design a parking system that made available a set of parking spots for drivers while trying to minimize the amount of time and fuel spent by drivers in trying to find a parking spot. The goal of the system is to design a pricing system that achieves these objectives. There are several parking systems such as the San Francisco's SFPark and Calgary's ParkPlus system that have tried to achieve this.

The problem of centrally assigning drivers to parking spots to minimize time and wasted fuel is naturally modeled by the online metrical matching problem. Let $\mathcal{M} = (M, d_M : M \to \mathbb{R}^+)$ be a metric space and let $S : [k] \to M$ be a set of k special points, known as servers. For an online sequence of requests $\vec{r} = (r_1, ..., r_k)$ that arrive at points in M, upon the arrival of each request r_i, we must make an irrevocable matching of r_i to a server S denoted by $f(r_i) \in S$ such that $f : \vec{r} \to S$ is a bijection. The goal is to minimize the total sum of distances between each request and its paired server, $\sum_{i=1}^{k} d_M(r_i, f(r_i))$. We will refer to this sum as the cost of the algorithm on the sequence of requests $\vec{r}$.

In order to be implementable within the context of the parking system, online algorithms need to be posted-price algorithms. In this problem we again control a set of k servers S, but now we maintain a pricing function $p_i : S \to \mathbb{R}$ at every time step $i \in [k]$. At the arrival of each request r_i the requests are now automatically matched to whichever unused server $s := f(r_i)$ minimizes $d(r_i, s) + p(s)$. Again, the goal of the algorithm is to minimize the total sum of distances between each request and its paired server, $\sum_{i=1}^{k} d(r_i, f(r_i))$.

[CEFJ15] gave an $O(\log n)$-competitive randomized posted-price algorithm for a line metric. The goal of our research was to determine whether one can extend this result to a tree metric. Before stating our results, we will review the most relevant related results in the literature.

1.1 Prior Related Work

We start by summarizing results known about online algorithms for online metric matching in general metric spaces. There is a deterministic online algorithm that is $(2k-1)$-competitive for any metric space, and no deterministic online algorithm can achieve a better competitive ratio in a star metric [KP93, KMV94]. An $O(\log k)$-competitive randomized algorithm for $O(\log k)$-HST's (Hierarchically Separated Trees) is given in [MNP06]. By combining this result with results about randomly embedding metric spaces into HST's [Bar96, Bar98, FRT04], [MNP06] obtained an $O(\log^3 k)$-competitive randomized online algorithm. Following this general approach [BBGN14] later obtained an $O(\log^2 k)$-competitive randomized online algorithm by giving an $O(\log k)$-competitive randomized algorithm for 2-HST's.

For certain natural types of metric spaces better competitive ratios are achievable. For a line metric, an $O(k^{.59})$-competitive deterministic online algorithm was given in [ABN$^+$14]. Later an $O(\log^2 k)$-competitive deterministic online algorithm for a line was given in [NR17]. It is known that the competitive ratio of every deterministic algorithm for the line is at least 9.001 [FHK05]. Several different $O(\log k)$-competitive randomized online algorithms for a line are given in [GL12]

that leverage special properties of HST's constructed from a line metric. [GL12] also showed that the natural Harmonic algorithm, which always picks one of the two free servers on either side of a request, with the probability of picking each serving being proportional to its distance to the request, is $O(\log \Delta)$-competitive. Here Δ is the aspect ratio of the server locations.

Research into posted price algorithms for online metrical matching was initiated in [CEFJ15], as part of a line of research to study the used of posted price algorithms to minimize social cost. As a post-price algorithm is a valid online algorithm, one can not expect to obtain a better competitive ratio for post-price algorithms that what is achievable by online algorithms. So this research line has primarily focused on problems where the optimal competitive ratio achievable by an online algorithm is (perhaps approximately) known, and seeks to determine when a similar competitive ratio can be (again perhaps approximately) achieved by a post-price algorithm.

Within this line of research, two algorithmic design paradigms have emerged. The first design paradigm is what we will call *mimicry*. A post-price algorithm A *mimics* an online algorithm B if the probability that B will take a particular action is equal the the probability that a self-interested agent will choose this same action when the prices of actions are set using A. Mimicry is used in [CEFJ15] in the context of online metric matching when the metric space is a line. [CEFJ15] shows how to set prices to mimic the $O(\log k)$-competitive Harmonic algorithm for online metric matching on a line metric. In the context of minimizing makespan on related machines, [FFR17] shows how to mimic the $O(1)$-competitive algorithm Slow-Fit from [AAF+97, AKP+97]. For some problems it is not possible to mimic known online algorithms using posted prices. For such problems, another algorithmic design paradigm is what we will call *monotonization*. In the monotonization approach, one first seeks to characterize online algorithms that can be mimicked, and then designs such an online algorithm. In the known examples this characterization involves some sort of monotonicity property. Monotonization is used in [CEFJ15] to obtain an $O(k)$-competitive posted-price algorithm for the k-server problem on a line. Monotonization is used in [IMPS17] to obtain an $O(1)$-competitive posted-price algorithm for minimizing maximum flow time on related machines. Another algorithmic design approach is to just directly design a pricing algorithm, as in done for metrical task systems in [CEFJ15].

1.2 Our Contribution

We first observe that one can mimic the Permutation algorithm from [KP93, KMV94] to obtain a posted-price algorithm that is also $(2k-1)$-competitive. We then observe that, even for tree metrics, it is not possible to set prices to mimic any of the online algorithms that are based on HST's as HST's by their very nature lose too much information about the structure of the metric space. Thus in our search for a $\mathsf{poly}(\log(k))$-competitive post-price algorithm for a tree metric, we turn to the monotonization approach. We first identify a monotonicity property that characterizes mimicable online algorithms for a tree metric. We show that an online algorithm A on a tree metric is mimicable if and only if the probability that a server s handles the next request if it arrives at location r_1 is at least as great as the probability that s handles the next request it arrives at r_2 if r_1 is on the path between s and r_2 in the tree.

The bulk of the thesis is devoted to showing that priceable algorithms and monotone algorithms are equivalent. Our proofs fall short of giving a $\mathsf{poly}(\log(k))$ monotone algorithm for any request sequence. Instead, we discuss some techniques and approaches on how one might go about giving a a $\mathsf{poly}(\log(k))$-competitive algorithm that is monotone.

The approach we suggest starts by embedding the tree into what we call a scaled Tree (ST), which is a refinement of an HST that retains more information about the original metric space. Roughly speaking the root of a ST is a tree where the cost of each edge is the diameter of the metric space divided by a parameter α. Level i of a ST consists of a collection of trees, where there is a bijection between each tree on this level and the nodes in the next higher level of the ST. Further the cost of each edge on this level is the diameter of the tree metric space divided by α^i. There is a bijection between the vertices of the original metric space and the leaves of the ST. Our construction of a ST starts with a Low Diameter Decomposition (LDD) of the metric space. A LDD is a partition of the vertices of the metric space. In an ST there is one vertex on the top level for each partition in the LDD. The construction of the lower levels of the ST then proceeds recursively on the partitions in the LDD.

We then note that we can assume with any loss of generality that each tree in the ST is a spider. A spider is a tree with at most one vertex of degree greater than 2, which we call the root of the spider. Further, we will pick the node to be the root of the spider in a particular way based on the topology of the original tree metric. We then suggest a reduction from trees to spider graphs arguing that it is sufficient to give and analyze an algorithm on a spider. Finally, we discuss how one might go about analyzing the competitiveness of an algorithm designed by this approach and then argue that it is sufficient to show that the algorithm on the spider is monotone for the entire algorithm to be monotone.

2 Notation and Preliminaries

2.1 Graph Notation

We start by giving some graph notation in order to talk about the problem.

We denote the input tree metric by $T = (V, E, d_T : E \to \mathbb{R}^+)$. Notice that a request sequence is defined completely by the vertices it arrives on. Hence for the entirety of the thesis, when there isn't any ambiguity, let us assume that the input request sequence is a sequence of vertices given by $\langle u_1, \ldots, u_k \rangle$.

Given a rooted tree, $T = (V, E)$ with root $\rho \in V$, let the *height* of T be the maximum number of vertices on a path from ρ to a leaf.

Definition 2.1. *A spider graph $T = (V, E)$, with root $\rho \in V$ and unit edge weights, is a tree satisfying the property that no vertex other than ρ has degree greater than 1. We will refer to the degree d of the spider by the degree of its root and we will denote the height of (T, ρ) by H.*

2.2 Preliminary Definitions

In order to characterize posted-price algorithms on tree metrics, we first define monotone algorithms.

Definition 2.2 (Monotonicity)**.** *For a tree metric $T = (V, E, d_T)$, let $P_{u,v}$ be the vertices on the path from u to v for any two vertices $u, v \in V(T)$. Let a matching problem on T have a set of servers $\mathcal{S} : [k] \to V(T)$. For a given matching algorithm $\mathcal{A}$ and request sequence $\vec{r}$, let us denote $\mathcal{S}_i$ to be the set of servers after $i-1$ requests are matched. We say $\mathcal{A}$ is **monotone** at time i on input T, server set $\mathcal{S}_i$ and request sequence $\vec{r}$ if algorithm $\mathcal{A}$ has the property that:*

$$\forall s \in \mathcal{S}_i, u \in V \text{ and } v \in P_{u,s}, \quad \Pr[u \to_{\mathcal{A}} s] \leq \Pr[v \to_{\mathcal{A}} s]$$

Where $\Pr[u \to_{\mathcal{A}} s]$ is the probability that $\mathcal{A}$ matches a request on u to s.

We say an algorithm is monotone on T with servers $\mathcal{S}$ and request $\vec{r}$ if it is monotone at every time step $i \in [k]$ for all valid subsets $\mathcal{S}_i$ (valid as per all the possible random choices of algorithm $\mathcal{A}$ on the first i requests of $\vec{r}$).

We say an algorithm is monotone if it is monotone over all possible input tree metrics, servers on the metric and request sequences on the input.

2.3 Hierarchical Clusterings and Scaled Trees

Given a tree $T = (V, E, d_T)$ rooted at some node ρ, we embed it into a probability distribution over "scaled" trees. The ideas are fairly standard, but we give the details here for concreteness, since we need to be precise about the constants.

First, we define a *hierarchical clustering with parameter α* as a sequence $(\mathscr{C}_0, \mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_D)$ of partitions of V, with the following properties:

1. Each $\mathscr{C}_i = \{(U_1^i, v_1^i), ..., (U_{k_i}^i, v_{k_i}^i)\}$ where U_j^i is a subset of vertices, and vertex v_j^i belongs to U_j^i and is called its *root*. The partition $\mathscr{C}_0 = \{(U_1^0 = V, v_1^0 = \rho)\}$ is the trivial partition.
2. The sets $U_1^i, \ldots, U_{k_i}^i$ form a partition of the vertex set; that is, $\bigcup_j U_j^i = V$ and $U_{j_1}^i \cap U_{j_2}^i = \emptyset$ for all $j_1 \neq j_2$. Moreover, each $U_j^{(i)}$ induces a connected component of T.
3. The root vertex v_j^i is often denoted $\rho(U_j^i)$. If v is a root of its part in $\mathscr{C}_i$, then for every level $i' > i$, v is the root of its part in $\mathscr{C}_{i'}$.
4. The partition $\mathscr{C}_i$ is a refinement of the partition $\mathscr{C}_{i-1}$; i.e., for each set U_j^i there exists a set $U_{j'}^{i-1}$ such that $U_j^i \subseteq U_{j'}^{i-1}$. This induces a parent-child relationship between clusters in consecutive levels.
5. The *radius* of a cluster U is the maximum distance of any vertex in U from the root of U. The radius of any child cluster is at most $\frac{1}{\alpha}$ the radius of its parent.

This clustering induces a *scaled tree* $H(T)$ with the same edges as T but with different edge costs.

Definition 2.3. *[scaled Tree] Given a hierarchical clustering $\mathscr{C} = (\mathscr{C}_0, \ldots, \mathscr{C}_m)$ with parameter α of a tree metric $T = (V, E, d_T)$, a* scaled tree *$H(T)$ of T with respect to $\mathscr{C}$ has the same vertex and edge set as T but with different edge lengths. Indeed, for each edge $(u_i, u_j) \in E(H(T))$, let l be the smallest index such that u_i, u_j lie in different parts of the partition $\mathscr{C}_l$. Define the length of the edge (u_i, u_j) to be $d_{H(T)}(u_i, u_j) := \frac{\mathsf{radius}(T,\rho)}{\alpha^l}$.*

3 Outline

In this section we outline the layout of the thesis, the different sections and the various components we discuss. We compartmentalize the work into sections and abstract out the details as much as possible in order to understand each section separately and how they contribute to the result as a whole.

At the highest level, we break the problem into two components. Firstly, in Section 4, we show that all pricing algorithms on tree metrics have the monotonicity property and vice versa and additionally give an explicit algorithm to construct a pricing algorithm from a monotone one. The proof for the first component is self contained and independent of the second part in which we discuss how one could construct a $\mathsf{poly}(\log(k))$-competitive monotone algorithm for tree metrics.

3.1 Priceable $\iff$ Monotone

Showing that pricing algorithms are monotone is relatively simple. The converse is, although, more involved.

We first make the observation that every deterministic monotone algorithm can be expressed as a partition of the vertices of the tree with special properties. Specifically we show,

Lemma 3.1 (Deterministic Monotone Algorithms are Partitions)**.** *Let $\mathcal{A}$ be a deterministic monotone algorithm on a tree metric $T = (V, E, d_T)$ and let $\vec{r}$ be an arbitrary request sequence. At some arbitrary time $1 \le t \le k$, let $\mathcal{S} : [k - t] \to V$ be the set of servers after $t - 1$ requests have been matched by $\mathcal{A}$. Then, the algorithm $\mathcal{A}$ at time t induces a partition $\mathbf{D} = (Q_1, \ldots, Q_{k-t})$ of the vertices of the tree that satisfies the following properties:*

(a) There exists a server $s \in Q_i$ for each nonempty part $Q_i \in \mathbf{D}$ such that all requests that arrive in Q_i are matched to s_i under $\mathcal{A}$.
(b) Each part $Q_i \in \mathbf{D}$ is connected with respect to the edges E.

We then show that given such a special partition scheme, one can design a pricing scheme that mimics the matching induced by the partitioning. I.e. We can assign prices to servers such that for every vertex $u \in V(T)$ the vertex matches to the same server under the partitioning scheme as it does under the constructed pricing scheme.

Lemma 3.2 (Partitions can be Priced)**.** *Given tree metric $T = (V, E, d_T)$ with servers $\mathcal{S} : [k] \to V$ and a monotone partitioning $\mathcal{D} = (Q_1, \ldots, Q_k)$ of T, there is a pricing scheme $p : \mathcal{S} \to \mathbb{R}$ such that*

for all vertices $u \in V(T)$

$$\forall s \in S, \quad \Pr[u \to_{\mathcal{D}} s] = \Pr[u \to_{p} s]$$

Since a randomized monotone algorithm is a distribution over deterministic monotone algorithms, we get our final theorem using Lemma 3.2 from above.

Theorem 3.3 (Pricing Schemes exist for Randomized Monotone Algorithms)**.** *Let* $\mathcal{A}$ *be a randomized monotone algorithm on a tree metric* $T = (V, E, d_T)$ *and let* $\vec{r}$ *be an arbitrary request sequence. At some arbitrary time* $1 \leq t \leq k$*, let* $\mathcal{S} : [k - t] \to V$ *be the set of servers after* $t - 1$ *requests have been matched by* $\mathcal{A}$*.*

Then, there is an explicit distribution $\mathcal{P}$ *over pricing mechanisms* $\{p \mid p : \mathcal{S} \to \mathbb{R}\}$ *for time* t *that ensures for any request given by a vertex* $u \in V$ *and any server* $s \in \mathcal{S}$*:*

$$\Pr[u \to_{p} s] = \Pr[u \to_{\mathcal{A}} s]$$

where $p \sim P$

Our proof for Theorem 3.3 constructs the pricing mechanism given the distribution over servers for each vertex. We construct this by inducting on the size of the tree and constructing a distribution over the special kinds of partitions discussed above.

3.2 $\mathsf{poly}(\log(k))$-competitive Monotone Algorithm on Tree Metrics

The second component of the thesis is a discussion on constructing a $\mathsf{poly}(\log(k))$-competitive monotone algorithm on tree metrics. Given such an algorithm, one can construct a pricing scheme that mimics it using the procedure from Theorem 3.3. The exploration of this part is done in Sections 5 and 6.

Instead of giving an explicit distribution over servers to match to at every vertex on the input metric, we suggest constructing a kind of divide and conquer algorithm. We discuss approaches to two algorithms, one to divide and the other to conquer.

3.2.1 Clustering the Input Tree Metric

Our divide algorithm uses a clustering subroutine as described in Section 5 and outputs a hierarchical clustering, with parameter α, given by $\mathscr{C} = (\mathscr{C}_0, \ldots, \mathscr{C}_m)$ with the following guarantee:

Lemma 3.4 (Expected Stretch of Scaled Tree)**.** *Consider tree metric* $T = (V, E, d_T)$ *with root vertex* ρ *and* $R = \mathsf{radius}(T, \rho)$*. For some* $\alpha > 0$*, the scaled tree* $H(T)$ *with respect to the above hierarchical clustering* $\mathscr{C}$ *guarantees that for all* $(u, v) \in V$*,*

(1) $d_{H(T)}(u, v) \geq 1/\alpha \cdot d_T(u, v)$*, and*
(2) $\boldsymbol{E}\left[d_{H(T)}(u, v)\right] \leq c_\alpha(R) \cdot d_T(u, v)$*, where* $c_\alpha(R)$ *is such that* $\alpha^{c_\alpha(R)} = R$*.*

The expectation in property (2) is over the randomness of the low-diameter decomposition.

3.2.2 HTMatch: A Divide and Conquer Algorithm

We define our algorithm HTMatch in Section 6.1 that takes a conquer algorithm, which we will call TreeMatch, and the hierarchical clustering to output an algorithm on the input tree metric. The algorithm will run an instance of TreeMatch in every cluster at every level of the hierarchical clustering essentially matching a request to a cluster at every level until it matches it to a server at the lowest level of the clustering.

3.2.3 Conquer Algorithm on Spiders and Trees

In order to construct TreeMatch we suggest the procedure of first giving an algorithm on spider graphs, which are defined in Definition 2.1 and then showing a reduction from general trees to spider graphs.

In Section 6.2 we discuss techniques one could use to give an algorithm on spider graphs with unit edge weights. Our objective would be to bound the cost of a request sequence as a function of the height and degree of the spider graph as well as the number of times the optimal algorithm payed positive cost on $\vec{r}$.

Following that, in Section 6.3 we suggest how one could reduce general trees to spider graphs. The objective is to show that if we had a well-defined algorithm on spider graphs for all request sequences, then we can have a well-defined algorithm for general trees.

3.2.4 poly$(\log(k))$-competitive Algorithm on Tree metrics

In Section 6.4 we discuss how to analyze the divide and conquer algorithm HTMatch. Our objective is to show the following Lemma

Proposed Lemma 3.5 (Cost Bound on Scaled Trees)**.** *Let $H(T) = (V, E)$ be a scaled tree, with parameter α, set of servers $S : [k] \to V$ and hierarchical clustering $(\mathscr{C}_0 \ldots, \mathscr{C}_m)$. Let $\vec{r} = \langle r_1, \ldots, r_k \rangle$ be any online request sequence on $H(T)$.*

The algorithm HTMatch *on $H(T), \vec{r}$ is a* poly$(\alpha \cdot c_\alpha(R))$*-competitive algorithm. Where $c_\alpha(R)$ is such that $\alpha^{c_\alpha(R)} = R$*

After discussing how to show Lemma 3.5, we discuss how to show that HTMatch is a monotone algorithm in Section 6.4.1. We argue why it is sufficient to show that TreeMatch is a monotone algorithm for HTMatch to be a monotone algorithm. Thus, our objective is to show the following Lemma

Proposed Lemma 3.6 (Algorithm is Monotone)**.** *Given an input tree $T = (V, E)$ with servers $\mathcal{S} : [k] \to V$ and a request sequence $\vec{r} = \langle r_1, \ldots, r_k \rangle$,* TreeMatch *on $T, \mathcal{S}$ and $\vec{r}$ is a monotone matching algorithm.*

Combining Lemma 3.5 with Lemma 3.6 would give us that HTMatch is a poly$(\log(k))$-competitive monotone algorithm on scaled trees. We can combine this with Lemma 3.4 bounding the stretch of

the embedding of a tree into a scaled tree to give us a $\mathsf{poly}(\log(k))$-competitive monotone algorithm on a tree. Specifically, we propose the following Theorem

Proposed Theorem 3.7 ($\mathsf{poly}(\log(k))$-competitive Monotone Algorithm)**.** *For a tree metric $T = (V, E, d_T)$, the algorithm corresponding to building a scaled tree $H((T, \rho))$, with parameter $\alpha = \mathsf{poly}(\log(k))$ and radius R, and running* HTMatch *on $H(T)$ is monotone and a $\mathsf{poly}(\alpha \cdot c_\alpha(R))$-competitive algorithm. Where $c_\alpha(R)$ is such that $\alpha^{c_\alpha(R)} = R$*

4 Pricing Monotone Algorithms

In this section, we show that matching algorithms on trees that can be implemented as pricing-based algorithms are exactly those that satisfy the following monotonicity property. This result is analogous to results for scheduling proved in [IMPS17].

It is not difficult to show that any pricing-based algorithm must be monotone using our techniques in this section. The converse is more interesting: we give a procedure to price any monotone algorithm. We first show this correspondence for the deterministic setting, and then extend it to randomized algorithms.

4.1 Pricing Deterministic Monotone Algorithms

To price a deterministic monotone algorithm, the main idea is to find a suitable partition $\mathbf{D} = (Q_1, \ldots, Q_{k-t})$ of the vertices of the tree at time $1 \le t \le k$.

Lemma 3.1 (Deterministic Monotone Algorithms are Partitions)**.** *Let $\mathcal{A}$ be a deterministic monotone algorithm on a tree metric $T = (V, E, d_T)$ and let $\vec{r}$ be an arbitrary request sequence. At some arbitrary time $1 \le t \le k$, let $\mathcal{S} : [k - t] \to V$ be the set of servers after $t - 1$ requests have been matched by $\mathcal{A}$. Then, the algorithm $\mathcal{A}$ at time t induces a partition $\mathbf{D} = (Q_1, \ldots, Q_{k-t})$ of the vertices of the tree that satisfies the following properties:*

(a) There exists a server $s \in Q_i$ for each nonempty part $Q_i \in \mathbf{D}$ such that all requests that arrive in Q_i are matched to s_i under $\mathcal{A}$.
(b) Each part $Q_i \in \mathbf{D}$ is connected with respect to the edges E.

Proof. Let the partitioning of V given by $Q_1, ..., Q_{k-t}$ be defined as $Q_i := \{v : v \to_{\mathcal{A}} s_i\}$. Since $\mathcal{A}$ is a deterministic algorithm, these sets must be disjoint and their union must be V. For any $Q_i \neq \emptyset$, it follows that $s_i \in Q_i$. To see this, let $u \in Q_i$: since s_i lies on the path from v to s_i, it must be that $1 = \Pr[u \to_{\mathcal{A}} s_i] \le \Pr[s_i \to_{\mathcal{A}} s_i] \le 1$, and therefore $s_i \in Q_i$. Hence for any nonempty Q_i we define s_i to be its leader. Further, if $u, v \in Q_i$ then $\Pr(u \to_{\mathcal{A}} s_i) = \Pr(v \to_{\mathcal{A}} s_i) = 1$ and thus any vertex between u and v must also match to s_i. Thus these components are connected. □

We will call partitioning schemes that satisfy Lemma 3.1 *monotone partitions*. Monotone partitions also induce monotone deterministic algorithms, by simply matching any request in each part Q_i to its designated leader, and are hence synonymous with them.

Lemma 3.2 (Partitions can be Priced)**.** *Given tree metric $T = (V, E, d_T)$ with servers $\mathcal{S} : [k] \to V$ and a monotone partitioning $\mathcal{D} = (Q_1, \ldots, Q_k)$ of T, there is a pricing scheme $p : \mathcal{S} \to \mathbb{R}$ such that for all vertices $u \in V(T)$*

$$\forall s \in S, \quad \Pr[u \to_{\mathcal{D}} s] = \Pr[u \to_p s]$$

Proof. Recall that the deterministic algorithm induced by the monotone partitioning matches requests that arrive in part $Q_i$to server $s_i := S(i)$. The following pricing scheme $p : \mathcal{S} \to \mathbb{R}$ implements this algorithm.

```
DetPrice (D = {Q_1, ..., Q_k})
    Set p(s_1) = c for any c ∈ ℝ
    for any Q_j adjacent to a part already priced do
        Let the edge (u, v) ∈ E be such that u ∈ Q_i and v ∈ Q_j
        Set p(s_j) ← p(s_i) + d(u, s_i) − d(v, s_j)
    for any Q_j = ∅ do
        Set p(s_j) ← ∞
```

Observe that the price $p(s_j) := p(s_i) + d(u, s_i) - d(v, s_j)$ satisfies

- $d(v, s_j) + p(s_j) < d(v, s_i) + p(s_i)$
- $d(u, s_i) + p(s_i) < d(u, s_j) + p(s_j)$.

When all leaders have been priced, price any non-leader servers at ∞. To show that it implements $\mathcal{A}$, we show that if $v \to_{\mathcal{A}} s_i$ (i.e., $v \in Q_i$), then $d(v, s_i) + p(s_i) < d(v, s_j) + p(s_j)$ for all $j \neq i$.

Suppose otherwise that there exists an s_j and v with leader s_i satisfying that $d(v, s_i) + p(s_i) \geq d(v, s_j) + p(s_j)$. First, consider the case where Q_j and Q_i are adjacent parts. Then there exists an edge (u_i, u_j) that crosses over from Q_i to Q_j. But we know that $d(v, s_i) \leq d(v, u_i) + d(u_i, s_i)$ and that $d(v, s_j) = d(v, u_i) + d(u_i, s_j)$. But then by substituting into our assumption we have that $d(v, u_i) + d(u_i, s_i) + p(s_i) \geq d(v, u_i) + d(u_i, s_j) + p(s_j) \implies d(u_i, s_i) + p(s_i) \geq d(u_i, s_j) + p(s_j)$, a contradiction to the second observation of our pricing scheme above.

Thus Q_j cannot be adjacent to Q_i. However, let Q_k be the adjacent part to Q_j that crosses the path from s_i to s_j. Since these two parts are adjacent, there exists an edge (u_k, u_j) the crosses from Q_k to Q_j. We also know that $d(u_k, s_k) + p(s_k) \leq d(u, s_j) + p(s_j)$ by design of the pricing scheme: however, this implies that $d(v, s_k) + p(s_k) \leq d(v, u_k) + d(u_k, s_k) + p(s_k) \leq d(v, u_k) + d(u_k, s_j) + p(s_j) = d(v, s_j) + p(s_j)$. This means that v should have instead matched to s_k, a contradiction. □

4.2 Randomized Algorithms

Now that we can price monotone deterministic algorithms, we give an explicit algorithm that determines the probability distribution over monotone deterministic algorithms describing a monotone randomized algorithm. This algorithm combined with algorithm in Lemma 3.2 will give us an explicit algorithm for determining a pricing scheme for a randomized monotone algorithm. Specifically, we show the following Lemma

Lemma 4.1. *Let $\mathcal{A}$ be any monotone matching algorithm on a tree metric $T = (V, E, d_T : E \to \mathbb{R}^+)$. Additionally, let $\vec{r}$ be an arbitrary request sequence and let $\mathcal{S} : [k - j] \to V$ be the server set after some arbitrary number, say $1 \leq j \leq k$, requests have been matched by $\mathcal{A}$.*

There exists a distribution $\mathcal{P}$ over monotone partitions of T, such that $\Pr[v \to_{\mathcal{P}} s_i] = \Pr[v \to_{\mathcal{A}} s_i]$ for all vertices $v \in T$ and all servers $s_i \in \mathcal{S}$.

Proof. Without loss of generality we can assume that $j = 0$ since the value of j does not change our proof. Given a tree T with k servers, assign each vertex v a vector $\pi_v := (\pi_v(1), \ldots, \pi_v(k))$, such that $\pi_v(i) = \Pr[v \to_{\mathcal{A}} s_i]$. For the rest of the proof, we assume that each vertex has such a vector, and want to get a distribution over monotone partitions that is consistent with these probability values. I.e. our objective is to find a distribution $\mathcal{P}$ over monotone partitions of T such that for all vertices $u \in V(T)$ and servers $s_i \in \mathcal{S}$, $\Pr_{D \sim \mathcal{P}}[u \to_D s_i] = \pi_u(i)$

The proof is via induction on the number of vertices of the tree: at each step we contract a leaf (and its servers) into its neighbor, update the probability vectors for the vertices in the contracted tree, inductively find a distribution over partitions, and then extend these partitions to the original tree. Note that a vertex may have multiple co-located servers.

The base case is the case of a single vertex v containing all k servers on it. In this case, each possible partition of the singleton vertex tree contains just one part, containing that one vertex with one of the k servers chosen as a leader. Given that the partition corresponding to the part with leader s_i is denoted by $\mathbf{D}_i$, we assign partition $\mathbf{D}_i$ a probability of $\pi_v(i)$. This gives the desired distribution $\mathcal{P}$ over monotone partitions.

For the inductive step, consider a tree T. Pick some leaf u (with t co-located servers, say) and contract it into its neighbor vertex to get a smaller tree T'. The probability vector π' for each vertex in $v \in V(T')$ is the following:

$$\pi'_v(i) = \pi_v(i)$$

Let $\Pr_{\mathcal{P}}(D)$ denote the probability that partition D is chosen under the distribution $\mathcal{P}$. Let $\mathcal{D}_i^x(T) := \{\text{partition } D \text{ of } T \mid v \to_D s_i\}$ denote the set of all monotone partitions on T that cause vertex $x \in V(T)$ to be mapped to server $s_i \in \mathcal{S}$. We inductively assume that we have a probability distribution $\mathcal{P}'$ over monotone partitions of T' such that $\sum_{D \in \mathcal{D}_i^x(T')} \Pr_{\mathcal{P}'}(D) = \pi'_x(i) = \pi_x(i)$ for all vertices $x \in V(T')$ and servers $s_i \in \mathcal{S}$.

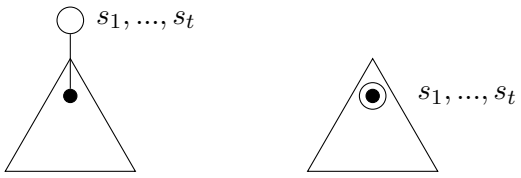

Figure 4.1: An example of a contraction

Given a partition F of T' in the support of $\mathcal{P}'$, we extend F to a partition on the tree T. If we extend partition F into, say, m partitions of T corresponding to $F_1, \ldots, F_m$, we distribute the probability $\Pr_{\mathcal{P}'}(D')$ among the m new partition(s). Doing this for all monotone partitions F of T' in the support of $\mathcal{P}'$ gives us a new distribution $\mathcal{P}$ over partitions of T. We additionally maintain

the invariant that for ever vertex and server pair given by $x \in V(T), s_i \in \mathcal{S}$ the probability that x is matched to s_i under $\mathcal{P}$ is $\pi_x(i)$. Specifically, we maintain the following equalities

$$\forall \text{partitions } F \text{ of } T' \quad \sum_{i=1}^{m} \Pr_{\mathcal{P}}(F_i) = \Pr_{\mathcal{P}'}(F) \tag{4.1}$$

$$\forall x \in V(T), s_i \in \mathcal{S} \quad \sum_{D \in \mathcal{D}_i^x(T)} \Pr_{\mathcal{P}}(D) = \pi_x(i) \tag{4.2}$$

While inducting, let us assume that we contracted the leaf $u \in V(T)$ with t co-located servers $s_1, \ldots, s_t$ onto its neighboring vertex $v \in V(T)$ so as to get T'. Let us consider a partition F of T' in the support of $\mathcal{P}'$. We need to extend F so as to get partitions of the tree T. There are two cases to consider when extending F:

Case 1: One of $s_1, \ldots, s_t$ is a leader of its part in F. I.e. F is a partition in $D_i^v(T')$ such that $i \in [t]$.

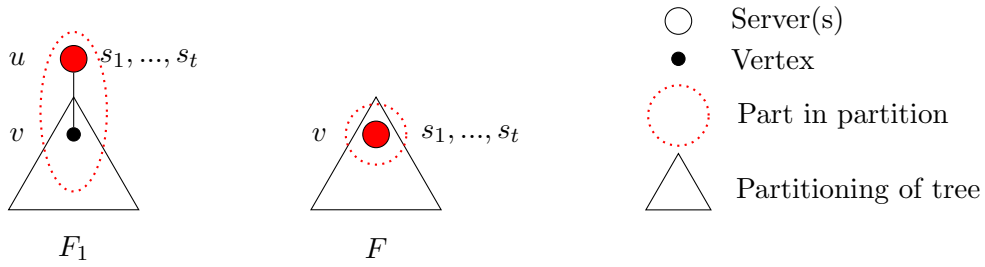

Figure 4.2: An example of a Case 1 extension. A red filled server is the leader of its part

In this case, only one new partition F_1 will be created from F and will inherit its probability: $\Pr_{\mathcal{P}}(F_1) = \Pr_{\mathcal{P}'}(F)$. Hence, if F is a partition in $D_i^v(T')$ such that $i \in [t]$, F extends to its corresponding partition in $D_i^v(T) \cap D_i^u(T)$. We satisfy Equation 4.1 by noticing that for a specific F_1 extended from F, no other partition F' of T' can be extended to give F_1. This can noted by noticing that any partition F' of T' that can be extended to F_1 must be in $D_i^v(T')$ (i.e. partitions in Case 1).

Case 2: The leader of the partition F is $s' \notin \{s_1, ..., s_t\}$. I.e. F is a partition in $D_i^v(T')$ such that $i \in [k] \backslash [t]$.

Let us assume $F \in D_i^v(T')$ such that $i \in [k] \backslash [t]$, then we will extend F into $t+1$ new partitions of T. For partitions $F_1, ..., F_t$, we will extend the partitioning such that u is in a singleton part and s_j is the leader of $F_j \in \{F_1, \ldots, F_t\}$. The partition $F_j \in \{F_1, \ldots, F_t\}$ will correspond to a partition in $D_i^v(T) \cap D_j^u(T)$. The final partitioning F_{t+1} created from F extends the part of s_i to include u and the t servers. I.e. F is extended to its corresponding partition in $D_i^v(T) \cap D_i^u(T)$. Examples of the two extensions have been given in Figure 4.4 and Figure 4.5 respectively.

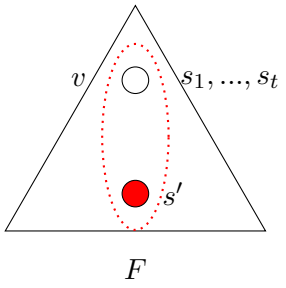

Figure 4.3: An example of a Case 2 partition in T'

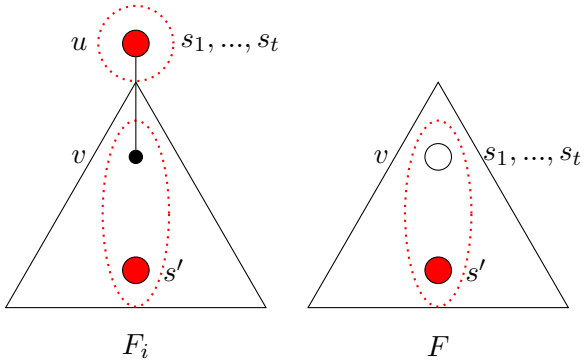

Figure 4.4: An example of Case 2 extension of F to F_i for $1 \leq i \leq t$

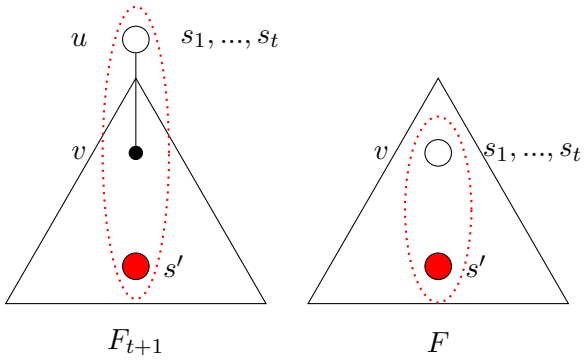

Figure 4.5: An example of Case 2 extension of F to F_{t+1}

We are left to show how we assign probabilities to partitions extended in Case 2. We will assign probabilities to partitions of T while simultaneously arguing that these probabilities will satisfy Equations 4.1 and 4.2.

If we show that Equation 4.1 is satisfied for all partitions of T' that we extend, since Equation 4.2 is satisfied for tree T' and distributions $\mathcal{P}'$ and π', the distribution $\mathcal{P}$ satisfies Equation 4.2 for all vertices in $V(T)\backslash\{u\}$ and servers in $\mathcal{S}$. Hence, we only need to show that $\mathcal{P}$ satisfies Equation 4.1 and Equation 4.2 for vertex $u \in V(T)$ which we contracted in the inductive step.

We first assign probabilities to partitions in $D_i^u(T)$ and prove that Equation 4.2 holds for vertex u and

servers $s_i \in \mathcal{S}\backslash\{s_1, \ldots, s_t\}$. To that end, for some $i \in [k]\backslash[t]$, we define $\Delta_i := \sum_{D \in D_i^v(\mathcal{P}')} \Pr_{\mathcal{P}'}(D) - \pi_u(i) = \pi_v(i) - \pi_u(i)$. Notice that by the monotonicity property, since $i \in [k]\backslash[t]$, we have $\pi_v(i) - \pi_u(i) \geq 0$.

Fix some $i \in [k]\backslash[t]$. For partitions $F \in D_i^v(T')$ (i.e. Case 2 partitions), we extend it to some $t+1$ partitions $F_1, \ldots, F_{t+1}$ such that $F_{t+1} \in D_i^u(T)$. Initially, we set $\Pr_{\mathcal{P}}(F_{t+1}) := \Pr_{\mathcal{P}'}(F)$for all such $F \in D_i^v(T')$. Then, we greedily remove Δ_i amount of mass from the probabilities of partitions in $D_i^u(T)$ without violating the constraint that all partitions in $D_i^u(T)$ must have non-negative probability. After completing this, we have that

$$\sum_{D \in D_i^u(T)} \Pr_{\mathcal{P}}(D) = \sum_{D \in D_i^v(T')} \Pr_{\mathcal{P}'}(D) - \Delta_i = \pi_u(i)$$

We now assign probabilities to partitions in $D_j^u(T)$ and prove that Equation 4.2 holds for vertex u and servers $s_j \in \{s_1, \ldots, s_t\}$. Specifically, for some $j \in [t]$, we want:

$$\sum_{D \in D_j^u(T)} \Pr_{\mathcal{P}}(D) = \sum_{F \in D_j^v(T')} \Pr_{\mathcal{P}}(F_1) + \sum_{i>t} \sum_{F \in D_i^v(T')} \Pr_{\mathcal{P}}(F_j) = \pi_u(j). \tag{4.3}$$

Notice that the probabilities in the first summation in (4.3) have already been assigned in Case 1 and sum to $\pi_v(j)$. So we must ensure that the probability we assign to partitions F_j such that $F \in D_i^v(T')$ satisfies both Equation 4.1 and Equation 4.3.

To that end, for some $j \in [t]$, let us define $\Gamma_j := \pi_u(j) - \sum_{F \in D_j^v(T')} \Pr_{\mathcal{P}}(F_1) = \pi_u(j) - \pi_v(j)$, which must be non-negative by the monotonicity property. We then greedily assign the Γ_j probability mass across all $\{F_j | F \in D_i^v(T'), i \notin \{1, ..., t\}\}$ such that we don't violate Equation 4.1 for any partition $F \in D_i^v(T')$ of server $i \in [k]\backslash[t]$ by assigning too much probability mass to F's extension F_j in T. Then by definition of Γ_j, for $j \in [t]$, we have satisfied equation 4.3.

By doing the above, we have satisfied Equation 4.2 for vertex u and all servers $i \in [k]$. Note that, if we show Equation 4.1 holds for $\mathcal{P}$, Equation 4.2 is maintained for every other vertex just by the inductive hypothesis.

Notice that we already showed Equation 4.1 for partitions $F \in D_i^v(T')$ where $i \in [t]$ as per the description for Case 1. It is left to argue that Equation 4.1 holds after greedily removing Δ_i from partitions in $D_i^u(T)$ for each $i \in [k]\backslash[t]$ and adding Γ_j mass to partitions in $\{F_j | F \in D_l^v(T'), l \in [k]\backslash[t]\}$ for each $j \in [t]$. Notice that in the process of adding the Γ values, we maintained the inequality

$$\sum_{l=1}^{t+1} \Pr_{\mathcal{P}}(F_l) \leq \Pr_{\mathcal{P}'}(F) \tag{4.4}$$

for all $F \in D_i^v(T')$ such that $i \in [k]\backslash[t]$.

In order to show equality in the above equation, first, notice that we can express the vector π_u as in the equation below by referring to the process by which we assigned probability values to partitions of T.

$$\pi_u = \pi_v + (\Gamma_1, ..., \Gamma_t, -\Delta_{t+1}, ..., -\Delta_k) \tag{4.5}$$

Since both π_u and π_v are probability distributions, $\sum_{i=1}^{k} \pi_u(i) = \sum_{i=1}^{k} \pi_v(i) = 1$. Thus,

$$\sum_{j=1}^{t} \Gamma_j = \sum_{i=t+1}^{k} \Delta_i$$

For partitions in $D_i^v(T')$ where $i \in [k] \backslash [t]$, inequality 4.4 combined with the equality above allows us to conclude that the amount of mass we removed is the same (the Δ values) is the same as the mass we added back to it (the Γ values) hence giving Equation 4.1 for these partitions. □

Thus we achieve our main theorem:

Theorem 3.3 (Pricing Schemes exist for Randomized Monotone Algorithms)**.** *Let $\mathcal{A}$ be a randomized monotone algorithm on a tree metric $T = (V, E, d_T)$ and let $\vec{r}$ be an arbitrary request sequence. At some arbitrary time $1 \leq t \leq k$, let $\mathcal{S} : [k - t] \to V$ be the set of servers after $t - 1$ requests have been matched by $\mathcal{A}$.*

Then, there is an explicit distribution $\mathcal{P}$ over pricing mechanisms $\{p \mid p : \mathcal{S} \to \mathbb{R}\}$ for time t that ensures for any request given by a vertex $u \in V$ and any server $s \in \mathcal{S}$:

$$\Pr[u \to_p s] = \Pr[u \to_{\mathcal{A}} s]$$

where $p \sim P$

Proof. Lemma 4.1 gives us that $\mathcal{A}$ can be decomposed into a distribution over monotone partitions, each of which is separately priceable by Lemma 3.2. By letting $\mathcal{P}$ be the distribution of pricing schemes induced by the distribution of monotone partitions given by Lemma 4.1, we have that $\Pr[r_i \to_{\mathcal{P}} s_j] = \Pr[r_i \to_{\mathcal{A}} s_j]$ □

5 Embedding Trees into Scaled Trees

In this section, we show how we embed the input tree metric into a scaled tree. Recall from Definition 2.3 that for a tree metric $T = (V, E, d_T)$, it is sufficient to give a hierarchical clustering $\mathscr{C} = (\mathscr{C}_0, \ldots, \mathscr{C}_m)$ with parameter α.

Given any tree T' rooted at ρ' and a parameter α, define its low-diameter decomposition as follows. Define $R := \mathsf{radius}(T', \rho')$. Let r.v. X be uniform in the range $(0, (R/\alpha)]$. Delete all edges (u, v) in T' such that for some integer $z \in \mathbb{Z}_{\geq 0}$,

$$d_{T'}(\rho', u) < z(R/\alpha) + X \leq d_{T'}(\rho', v).$$

Define the root of each connected component to be the vertex that used to be closest to the root ρ'. Note that the radius of each component is now smaller than R/α.

Now the hierarchical clustering $\mathscr{C}$ for T is easy. Let $\mathscr{C}_0 = (V, \rho)$. Perform the above low-diameter decomposition on T with root ρ to get clusters of radius at most $\mathsf{radius}(T, \rho)/\alpha$. This gives $\mathscr{C}_1$. Now repeat this operation on each connected component (with its root as defined above) to get the next level $\mathscr{C}_2$, and so on, until each part in the partition consists of only singletons.

Lemma 3.4 (Expected Stretch of Scaled Tree)**.** *Consider tree metric $T = (V, E, d_T)$ with root vertex ρ and $R = \mathsf{radius}(T, \rho)$. For some $\alpha > 0$, the scaled tree $H(T)$ with respect to the above hierarchical clustering $\mathscr{C}$ guarantees that for all $(u, v) \in V$,*

(1) $d_{H(T)}(u, v) \geq 1/\alpha \cdot d_T(u, v)$, and
(2) $\boldsymbol{E}\left[d_{H(T)}(u, v)\right] \leq c_\alpha(R) \cdot d_T(u, v)$, where $c_\alpha(R)$ is such that $\alpha^{c_\alpha(R)} = R$.

The expectation in property (2) is over the randomness of the low-diameter decomposition.

Proof. It suffices to show the properties for each edge (u, v) of T. For property (1), let l be the smallest index such that u, v lie in different clusters in $\mathscr{C}_l$, hence they belonged to the same component in $\mathscr{C}_{l-1}$. The radius of clusters in $\mathscr{C}_{l-1}$ is at most R/α^{l-1}, and hence $d_T(u, v) \leq R/\alpha^{l-1}$. Now Definition 2.3 prescribes the length of edge (u, v) in $H(T)$ to be $R/\alpha^l \geq d_T(u, v)/\alpha$. This proves property (1).

For property (2), let $\mathcal{E}_i$ be the event that (u, v) gets cut at level i. If it has been cut at some level $j < i$, it will be cut with probability zero. Else the radius of the level-$(i-1)$ component containing (u, v) is at most R/α^{i-1}. So the probability of the low-diameter decomposition cutting (u, v) at level i is $\Pr[\mathcal{E}_i] \leq d_T(u, v)/(R/\alpha^i)$. Hence

$$\mathbf{E}\left[d_{H(T)}(u, v)\right] = \sum_{i=1}^{c_\alpha(R)} \mathbf{Pr}\left[\mathcal{E}_i\right] \cdot (R/\alpha^i) \leq \sum_{i=1}^{c_\alpha(R)} \frac{d_T(u, v)}{R/\alpha^i} \cdot (R/\alpha^i) = c_\alpha(R) \cdot d_T(u, v).$$

Hence the proof. □

We record one final fact about the low-diameter decomposition procedure.

Fact 5.1 (Cluster Trees have Small Height)**.** *Consider the tree obtained by contracting the components obtained by running the low-diameter decomposition procedure on T rooted at r, and adding back the deleted edges. The height of this tree is at most $\alpha + 1$.*

6 Our Algorithm

In this section, we discuss a proposal of how one could give a $\mathsf{poly}(\log(k))$-competitive monotone algorithm for our matching problem on scaled trees. In order to give a competitive algorithm on the original metric, we first embed our tree metric T into a scaled tree $H(T)$ as described in Section 5.

6.1 HTMatch: A Divide and Conquer Algorithm

HTMatch is the overarching algorithm that uses a conquer algorithm, TreeMatch, defined for trees on the scaled tree. To that end, we describe HTMatch in the rest of Section 6.1.

Let $(\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_m)$ be the output of our hierarchical clustering as per the construction in Section 5. When a request r_t arrives at time step t, let $(U_{l-1}, v) \in \mathscr{C}_{l-1}$ be the cluster and root on level $l-1$ that r_t arrives at and let this be the lowest level (largest index) of the hierarchical clustering such that r_t has unused servers in its cluster. In the construction of the hierarchical clustering, the LDD subroutine is applied to every cluster. Consequently, let $\mathcal{U} = \{U_0, U_1, \ldots\}$ be the sub-clusters at level l formed by applying LDD to U_{l-1}. Without loss of generality, let U_0 be the cluster on level l in which r_t arrives and let us denote $U_v \in \{U_0, U_1, \ldots\}$ to be the cluster such that $v \in U_v$.

By the construction of the hierarchical clustering, the sub-clusters $\{U_0, U_1, \ldots\}$ induce a tree with edges of weight $\frac{\mathsf{radius}(T)}{\alpha^l}$. We construct the tree $T_{\mathcal{U}}$ induced by these clusters by assigning a vertex for each sub-cluster and then placing as many servers on the vertex as contained in the corresponding sub-cluster in $\mathcal{U}$.

The "conquer" algorithm $\mathsf{TreeMatch}$ for trees will be used to match r_t to a vertex $u \in V(T_{\mathcal{U}})$ corresponding to matching $r_t \to U$ such that $U \in \mathcal{U}$. By maintaining an instance of $\mathsf{TreeMatch}$ for every cluster at every level of the hierarchical clustering, this induces a recursive algorithm for matching r_t to a server because the path from r_t to U arrives at a sub-cluster of U from which we run the same process as if the request r_t arrived at this sub-cluster. We denote this recursive algorithm by $\mathsf{HTMatch}$.

6.2 Matching on Spiders

In this section, we discuss how one might approach giving an algorithm for spider graphs. Let us call this algorithm $\mathsf{SpiderMatch}$ for convenience. Assume we have some input spider graph metric $T = (V, E, d_T : E \to \mathbb{R}^+)$ with root $\rho \in V$, degree d, height H, servers $\mathcal{S} : [k] \to V$ and some request sequence $\vec{r}$ given by the sequence of vertices $\langle u_1, \ldots, u_k \rangle$.

In order to design a competitive algorithm, we first analyze the properties of the requests for which we pay positive cost. To that end, we first define some notation; let $\eta : V \to \mathbb{N}$ and $A_t : V \to \mathbb{N}$ be functions defined for each time step $t \in [k]$ such that $\eta(u)$ is the number of servers at vertex $u \in V$ before any requests arrived and $A_t(u)$ is the number of requests that have arrived at u by time t. We say a request u_t is non-colocated if $\eta(u_t) - A_t(u_t) \leq 0$.

By matching a non-collocated request that arrives at u_t to a server on some vertex $u \neq u_t$ we are creating a *hole* at u – essentially, we used up a server on u that could potentially be used to serve a request that arrives at u. In essence, the algorithm's objective is to find the right home for every non-colocated request that arrives. Whenever it sends the non-colocated request to a vertex, it creates a hole at this vertex on which another request might arrive causing us to send this new request to another vertex which will create a hole at this new vertex and so on. The objective in designing the algorithm is to bound the length of the sequence of holes created by every non-colocated request.

Let there be M non-colocated requests in sequence $\vec{r}$, our objective would be to bound the expected number of times we pay positive cost in terms of M, the degree d and the height H of the spider.

6.3 From Spiders to Trees

In this subsection we suggest a reduction from general trees to spider graphs. Given an input tree metric $T = (V, E)$ with root ρ, unit edge distances and servers $S : [k] \to V$, our objective is to give a spider graph $T_S = (V_S, E_S)$ with root ρ_S, servers $S' : [k'] \to V_S$ and mapping $\gamma : V_S \to V$ from the vertices of the spider to the tree.

For a request sequence $\vec{r} = \langle u_1, \ldots, u_k \rangle$ on T, we give an alternate request sequence $\vec{r'} = \langle u'_1, \ldots, u'_{k'} \rangle$ on T_S and use the matching on T_S of $\vec{r'}$ combined with the mapping γ to give a matching on T of $\vec{r}$.

Let us first define the spider T_S. For a server $s \in S$, let P_s be the path from ρ to s on T. For every server $s \in S$ such that $P_s = \rho, v_1, \ldots, v_s$ is not completely contained in $P_{s'}$ for every other $s' \in S$, we create a path $\rho_S, v'_1, \ldots, v'_s$ in T_S and set $\gamma(v'_i) := v_i$. Naturally, we set $\gamma(\rho_S) := \rho$

We now give the alternate request sequence $\vec{r'}$ and the corresponding matching on T. When a request arrives on vertex v in T, we pick a vertex v' in T_S such that $\gamma(v') = v$ and send a request there on T_S. Assuming SpiderMatch matches the request to s' in T_S, we match the request on T to $\gamma(v_{s'})$ where $v_{s'}$ is the vertex on T_S which contains s'. We then send a request to every $v' \in V_S$ such that $\gamma(v') = \gamma(v_{s'})$ so as to ensure no "copy" of s is unused on T_S.

Using this scheme, we know that any time when a server s' is used on T_S all the servers s" such that $\gamma(s") = \gamma(s')$ are used. And that if s' is unused then all servers s" with $\gamma(s") = \gamma(s')$ are unused. Also notice that whenever we pay positive cost on T, we pay positive cost on T_S. Hence analyzing the cost for SpiderMatch on spider graphs is sufficient to give a cost bound on trees.

6.4 Analyzing our Algorithm

In this section we discuss how one might go about analyzing the algorithm HTMatch which has the properties that we discussed in Sections 6.2 and 6.3. Our objective is to show the following Lemma

Proposed Lemma 3.5 (Cost Bound on Scaled Trees)**.** *Let $H(T) = (V, E)$ be a scaled tree, with parameter α, set of servers $S : [k] \to V$ and hierarchical clustering $(\mathscr{C}_0 \ldots, \mathscr{C}_m)$. Let $\vec{r} = \langle r_1, \ldots, r_k \rangle$ be any online request sequence on $H(T)$.*

The algorithm HTMatch *on $H(T), \vec{r}$ is a* poly$(\alpha \cdot c_\alpha(R))$*-competitive algorithm. Where $c_\alpha(R)$ is such that $\alpha^{c_\alpha(R)} = R$*

HTMatch maintains an instance of TreeMatch in every cluster at every level of the hierarchical clustering. For every level i and every cluster and root $(U, v) \in \mathscr{C}_i$ at that level, let $M^{\text{out}}(U)$ be the number of requests that were matched to cluster U by TreeMatch at level i. Similarly, we define $M^{\text{in}}(U)$ to be the number of non-colocated requests that arrived in U and for which the highest level (smallest index) they were non-colocated is level $i + 1$. Note that the holes created due to these $M^{\text{in}}(U)$ non-collocated requests never exit U.

For every level and for every cluster U at that level, we bound the cost incurred by the $M^{\text{in}}(U) + M^{\text{out}}(U)$ requests in U. Recall that in Section 6.2 we designed SpiderMatch such that every non-colocated request created a hole by matching to a vertex it did not arrive on and by doing so, created

a sequence of requests that matched to vertices they did not land on. Hence, it is sufficient to bound the cost of creating the sequence of holes, across levels, that non-colocated requests (requests contributing to $M^{\text{in}}(U)$) create.

Potentially, a proof for Lemma 3.5 would involve some recurrence that, using the cost bound for TreeMatch on a particular level, would bound the total cost across levels.

6.4.1 Monotonicity of HTMatch

Recall that we want to give an algorithm that is monotone because, as we will show in the next section, one can construct a pricing scheme that mimics a monotone algorithm. Our objective is to show the following Theorem

Proposed Lemma 3.6 (Algorithm is Monotone)**.** *Given an input tree $T = (V, E)$ with servers $\mathcal{S} : [k] \to V$ and a request sequence $\vec{r} = \langle r_1, \ldots, r_k \rangle$,* TreeMatch *on $T, \mathcal{S}$ and $\vec{r}$ is a monotone matching algorithm.*

We first argue that it is sufficient to show that TreeMatch is monotone in order to show HTMatch is monotone. For the sake of convenience, we denote TreeMatch with T and HTMatch by HT.

Let $H(T)$ be a scaled tree with parameter α and hierarchical clustering $(\mathscr{C}_0, \ldots, \mathscr{C}_m)$. Let v_1, v_2 be vertices in $H(T)$ and s be a server on $H(T)$ such that v_2 lies on the path from v_1 to s. We need to show that

$$\Pr[v_1 \to_{\mathsf{T}} s] \leq \Pr[v_2 \to_{\mathsf{T}} s]$$

If $v_1 = v_2$, then we are done. Else, let level $i+1$ be the highest level (smallest index) of the clustering in which v_1 and s lie in different clusters. Recall that there is an instance of SpiderMatch defined in each cluster, across levels, such that algorithm will match into lower and lower levels until it arrives at an individual server. Let $\mathcal{U}_s = (U_1^s, \ldots, U_m^s)$, $\mathcal{U}_1 = (U_1^1, \ldots, U_m^1)$ and $\mathcal{U}_2 = (U_1^2, \ldots, U_m^2)$ denote the sequence of clusters that s, v_1 and v_2 belong to respectively in the hierarchical clustering. We then have,

$$\begin{aligned}
\Pr[v_1 \to_{\mathsf{HT}} s] &= \prod_{\ell \geq i} \Pr[U_\ell^1 \to_{\mathsf{T}} U_\ell^s] \\
&\leq \prod_{\ell \geq i} \Pr[U_\ell^2 \to_{\mathsf{T}} U_\ell^s] \\
&= \Pr[v_2 \to_{\mathsf{HT}} s]
\end{aligned}$$

where the second line follows from the assumption that TreeMatch is monotone on all tree metrics and request sequences. One would still have to give a proof for the monotonicity of TreeMatch depending on the specific details of the algorithm. In which case, we would get the final concluding Theorem

Proposed Theorem 3.7 ($\mathsf{poly}(\log(k))$-competitive Monotone Algorithm)**.** *For a tree metric $T = (V, E, d_T)$, the algorithm corresponding to building a scaled tree $H((T, \rho))$, with parameter $\alpha =$*

$\mathsf{poly}(\log(k))$ *and radius* R*, and running* HTMatch *on* $H(T)$ *is monotone and a* $\mathsf{poly}(\alpha \cdot c_\alpha(R))$*-competitive algorithm. Where* $c_\alpha(R)$ *is such that* $\alpha^{c_\alpha(R)} = R$

Using Lemma 3.4 in conjunction with Theorem 4 from [Bar96] and Lemma 3.5 will give us that our algorithm is a $\mathsf{poly}(\alpha \cdot c_\alpha(R))$-competitive algorithm on tree metrics. For an input metric, Theorem 4 from [Bar96] argues that embedding the metric into another metric with at most α factor distortion in expectation combined with a β-competitive algorithm in the embedded metric gives an $\alpha \cdot \beta$-competitive algorithm in the original metric. Lemma 3.6 would concludes that the algorithm is monotone.

7 Conclusion

The result on the equivalence of monotone algorithms and pricing algorithms is useful. Monotonicity is a simple and intuitive property one would expect any competitive algorithm would have for the online metric matching problem. Yet, after several attempts at solving this problem, it is not completely clear to us how to design a competitive monotone algorithm.

While we are unable to give the details for the $\mathsf{poly}(\log(k))$-competitive monotone algorithm on trees, our work has indicated strongly that the Lemmas presented in Section 6 are true. The core roadblocks are in giving a competitive algorithm for the spider graph and arguing that it is monotone. Approaches we tried for the algorithm on spiders had two components; first, we maintained a non-trivial distribution over "holes" on the spider graph and updated this distribution as requests arrived. Second, we tried to argue that an actual matching algorithm can be faithful to this distribution.

An interesting question that arises from our approach to giving a $\mathsf{poly}(\log(k))$-competitive monotone algorithm is whether the monotonicity property admits a $\mathsf{poly}(k)$ runtime algorithm. I.e. Whether there is an algorithm that has polynomial runtime at each time step.

A natural extension of this problem is to characterize priceability for all graph metrics. I.e. what properties must a priceable algorithm have on general finite metrics? The monotonicity property is ill-defined for algorithms on general graph metrics because there could be several paths from a point to a server. Even if we defined the monotonicity property such that the path from the vertex of choice to server of choice is the shortest path, there are monotone algorithms that aren't priceable under this definition of monotonicity. One can construct simple counter-examples on cycles.

References

[AAF+97] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3), May 1997.

[ABN+14] Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A $o(n)$-competitive deterministic algorithm for online matching on a line. In *Workshop on Approximation and Online Algorithms*, pages 11–22, 2014.

[AKP+97] Yossi Azar, Bala Kalyanasundaram, Serge A. Plotkin, Kirk Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[Bar98] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *ACM Symposium on Theory of Computing*, pages 161–168, 1998.

[BBGN14] Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Naor. A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.

[CEFJ15] Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 73–91, 2015.

[FFR17] Michal Feldman, Amos Fiat, and Alan Roytman. Makespan minimization via posted prices. In *ACM Conference on Economics and Computation*, pages 405–422, 2017.

[FHK05] Bernhard Fuchs, Winfried Hochstättler, and Walter Kern. Online matching on a line. *Theoretical Computer Science*, 332(1-3):251–264, 2005.

[FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[GL12] Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *International Colloquium on Automata, Languages, and Programming*, pages 424–435, 2012.

[IMPS17] Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. Minimizing maximum flow time on related machines via dynamic posted pricing. In *European Symposium on Algorithms*, pages 51:1–51:10, 2017.

[KMV94] Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

[KP93] Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[MNP06] Adam Meyerson, Akash Nanavati, and Laura J. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 954–959, 2006.

[NR17] Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *Symposium on Foundations of Computer Science*, pages 505–515, 2017.