Pricing Online Metric Matching Algorithms on Trees

Aditya Krishnan

May 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

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

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 giving a poly-$\log(k)$ competitive post-price algorithm on tree metrics.

Acknowledgments

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 would also like to thank Prof. Anil Ada for being a selfless mentor and an invaluable voice of guidance.

My friends and family have been a pillar of support and I thank them for without them, I certainly wouldn't have been able to complete this work.

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 **4**

1.1 Prior Related Work 4

1.2 Our Contribution 5

2 Notation and Preliminaries **7**

2.1 Problem statement and preliminaries 7

2.2 Clustering 7

2.3 The Experts Problem 8

3 Clustering a tree **8**

4 Our Algorithm **10**

4.1 HTMatch 11

4.2 Matching on spiders 11

4.3 From spiders to trees 18

4.4 TreeMatch 18

4.5 Analyzing our algorithm 18

5 Pricing Monotone Algorithms **20**

5.1 Deterministic Algorithms 20

5.2 Randomized Algorithms 22

5.2.1 Case 1 23

5.2.2 Case 2 24

1 Introduction

Since 2011 SFpark has been San Francisco's system for managing the availability of on-street parking. The goal of the system is to reduce the time and fuel wasted by drivers searching for an open space. The system monitors parking usages via sensors embedded in the pavement, and distributes this information in real-time to drivers via SFpark.org and phone apps. SFpark periodically adjusts parking meter pricing to manage demand, lowering prices in underutilized areas, and raising prices in overutilized areas. Prices can range from a minimum of 25 cents to a maximum of 7 dollar per hour during normal hours, with a 18 dollars per hour cap for special events such as baseball games or street fairs. Several other cities in the world have similar systems. For example, Calgary's ParkPlus system has been using a similar demand based pricing model since 2008 [SFpb, SP13, SFpa].

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 $S : [n] \to M$ be a set of n special points, known as servers. For an online sequence of requests $R = (r_1, ..., r_n)$ 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 $f : R \to S$ is a bijection. The goal is to minimize the total sum of distances between each request and its paired server, $\sum_{i=1}^{n} d_M(r_i, f(r_i))$. We will refer to this sum as the cost of the algorithm on the sequence of requests R.

In order to be implementable within the context of SFpark, online algorithms need to be posted-price algorithms. In this problem we again control a set of n servers S, but now we maintain a pricing function $p : S \to \mathbb{R}$. 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, $\sum_{i=1}^{n} d(r_i, f(r_i))$.

[CEFJ15] gave an $O(\log n)$-competitive randomined 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]. More generally, the algorithm given in [NR17] has the property that for every metric space, its competitive ratio is at most $O(\log^k)$ times the optimal competitive ratio achievable by any deterministic algorithm on that metric space. It known that the competitive ratio of every deterministic algorithm for the line is at least 9.001 [FHK05]. Several different $O(\log n)$-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 $O(\log n)$-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 polylog-competitive post-price algorithm for a tree metric, we turn to the monotonization approach. We first identify a monotonicity property that characterizes mimiacable 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 the bulk of the paper is devoted to developing an $(\log^? n)$-competitive monotone online algorthm. The algorithm starts by embedding the tree into what we will call a Hierarchical Tree (HT), which is a refinement of an HST that retains more information about the original metric space. Roughly speaking the root of a HT 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 HT 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 HT. Further the cost of each edge on this level is the diameter of the tree metric space divided by α^i. As in an HST, there is a bijection between the vertices of the original metric space and the leaves of the HT. As with an HST, our construction of a HT 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 HST the number of children of the root is equal to the number of partitions in the LDD, and the least-common-ancestor of vertices from different partitions in the LDD is the root. In an HT there is one vertex on the top level for each parition in the LDD. So the top level of a HT retains information about the distances between partitions in the LDD that an HST loses. The construction of the lower levels of the HT then proceeds recursively on the partitions in the LDD, as does the construction of the lower levels of an HST. Further, our analysis can not use a black box LDD construction. We need to construct our LDD in a special way that tightly controls the variance of random properties of our HT that affect the competitiveness of our algorithm.

We then note that we can assume with any loss of generality that each tree in the HT is a spider. A spider is a tree with at most one vertex of degree greater than 2, which we call the head of the spider. Further, we will pick the node to be the head of the spider in a particular way based on the topology of the original tree metric. We then give a monotone algorithm for a spider (with edges of equal cost) that is based on the classic multiplicative weights online learning algorithm [AHK12]. We provide an analysis to show that roughly speaking the competitiveness of this algorithm on a spider is linear in the length of the legs and the log of the number of vertices. We then show how to recursively apply this algorithm to obtain a monotone algorithm for a HT. Finally, we show by induction on the levels of the HT that the competitiveness of the algorithm is $O(\log^? n)$. The induction is an extension of the one in [MNP06] that a $O(\log n)$-competitive algorithm for a star can be extended to an algorithm for a $O(\log n)$-HST with the loss of a log factor in the competitiveness. There is not a lot of wiggle room in our analysis. For example, if the algorithm for a spider has 1% higher cost than our algorithm, then the resulting competitiveness on a HT would no longer be poly-logarithmic. So we have to be quite precise in our analysis, which causes some of the analysis to be more complicated than would be necessary if constant factors weren't so important.

We finish by noting that there is no poly-log competitive monotone online algorithm for a general metric space ??????

2 Notation and Preliminaries

2.1 Problem statement and preliminaries

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

Definition 2.1 (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 $\mathcal{S}_i : [n] \to V(T)$ that are unused after $i \in [n]$ requests have arrived, then, we say a matching algorithm $\mathcal{A}$ is **monotone** when it satisfies the following property:*

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

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

In order to converse about the problem, we must first define some terms and quantities. A request r_i is *collocated* with respect to an algorithm $\mathcal{A}$, if there exists $s \in S$ such that $s = r_i$ and s is yet unmatched to under $\mathcal{A}$. A non-collocated request $r_i \in R$, by definition, is matched to a server $s \in S$ under $\mathcal{A}$, such that $s \neq r$. In doing so, r_i has created a *hole* at s. Intuitively, a hole represents a missing server s; where $\mathcal{A}$ will have to pay some positive cost to match a request $r_j = s$ such that $j > i$, the optimal matching may pay nothing.

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 2.2. *A spider graph $T = (V, E)$, with root $\rho \in V$, 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 Clustering

Given a metric space $T = (V, E)$ with metric d_T and parameter α, a hierarchical clustering of the space is a sequence $(C_0, C_1, ..., C_M)$ satisfying the following:

1. Each $C_i = \{(U_1^{(i)}, v_1^{(i)}), ..., (U_{k_i}^{(i)}, v_{k_i}^{(i)})\}$ partitions the space; that is, $\cup_j U_j^i = V$ and $U_{j_1}^i \cap U_{j_2}^i = \emptyset$ for all $j_1 \neq j_2$.

2. Each $v_j^{(i)}$, known as the root of $U_j^{(i)}$ and denoted $\rho(U_j^{(i)})$, is contained in $U_j^{(i)}$, and for every $U \ni v$ in levels greater than i the root of U is v.

3. C_{i+1} is the collection of low diameter decompositions of all components in C_i. This induces a parent-child relationship between clusters in consecutive levels, where the radius $r(U)$ of any child cluster U is at most $\frac{1}{\alpha}$ the radius of the parent.

4. Each $U_j^{(i)}$ is a connected component of T.

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

Definition 2.3. *[Hierarchal Tree] Given a hierarchal clustering $C = (C_0, \ldots, C_m)$ with parameter α of a tree metric $T = (V, E, d_T)$; a* ***hierarchal tree*** *$H(T)$ of T with respect to C is formed by having the same vertex and edge set as T but with different edge distances. For each edge $(u_i, u_j) \in E(H(T))$, the distance $d_{(u_i,u_j)}$ is defined as follows: let l be the smallest index such that for $(U_i, \_), (U_j, \_) \in C_l$, we have $u_i \in U_i, u_j \in U_j$. Then, we define $d_{(u_i,u_j)} = \frac{\mathsf{radius}((T,\rho))}{\alpha^l}$.*

2.3 The Experts Problem

Recall the *experts* problem, where at each time step t, we produce a probability vector $p^t \in \triangle_N$, and the adversary produces a cost vector $c^t \in [0,1]^N$. The loss of our algorithm is $c^t \cdot p^t$; the algorithm is then allowed to update p^t to get p^{t+1} before the next cost vector c^{t+1} arrives. The loss of the algorithm over τ time steps is $\sum_{t=1}^{\tau} c^t \cdot p^t$.

Theorem 2.4 (see, e.g., [AHK12])**.** *For any $\varepsilon \in [0, 1/2]$, there exists a deterministic algorithm such that, for any expert i,*

$$\sum_{t=1}^{\tau} \langle c^t, p^t \rangle \leq (1+\varepsilon) \sum_{t=1}^{\tau} (c^t)_i + \frac{\ln N}{\varepsilon}$$

In other words, the loss of the algorithm is at most $(1+\varepsilon)$ times the loss of the best expert in hindsight, plus $\varepsilon^{-1} \ln N$.

3 Clustering a tree

Our overarching algorithm constructs a hierarchal tree, from a hierarchal clustering, of our original tree $T = (V, E, d_T)$ and then runs an algorithm recursively on every level of the hierarchal clustering. In this section, we construct our hierarchal tree and clustering.

We first give an algorithm $\mathsf{LDD}((T, v), \alpha)$ that partitions a tree T into connected clusters and their roots $(U_1, v_1), \ldots, (U_k, v_k)$ such that $\mathsf{radius}((U_i, v_i)) \leq \frac{\mathsf{radius}((T,v))}{\alpha}$ for all i. To construct the hierarchal clustering, we apply this algorithm recursively on each cluster and its root (U_i, v_i).

Let $\mathcal{U}(a, b)$ be the uniform distribution in the interval $[a, b]$ such that $a, b \in \mathbb{R}$

Algorithm 1: Low Diameter Decomposition

```
LDD ((T, $\rho$), $\alpha$):
    $X \sim \mathcal{U}(0, \frac{\mathsf{radius}(T,v)}{\alpha})$
    $U \leftarrow \{w \mid d_T(v, w) \leq X\}$
    $C \leftarrow \{U\}$
    while $\exists (u, v) \in E(T), U \in C$ such that $u \in U$ and $\forall W \in C,\ v \notin W$ do
        $V_{\bar{C}} \leftarrow V(T) \setminus \{w \mid \exists U \in C$ such that $w \in U\}$
        $U \leftarrow \{w \mid d_T(v, w) \leq \frac{\mathsf{radius}(T)}{\alpha}$ and $w \in V_{\bar{C}}\}$
        $C \leftarrow C \cup \{(U, v)\}$
  return $C$
```

Algorithm 2: Hierarchal Clustering

```
HC ((T, $\rho$), $\alpha$):
    if $|V(T)| = 1$ then
        return $(\{T, V(T)\})$
    else
        $C \leftarrow$ LDD$((T, \rho), \alpha)$
        $S \leftarrow \{$HC$((U, \rho_u), \alpha) \mid (U, \rho_u) \in C\}$
        $d \leftarrow \max_{\mathcal{C}_i \in S}$ sequence length$(\mathcal{C}_i)$
        $\mathcal{C} = (C_0, \ldots, C_d)$
        for $i = 1$ to $d$ do
            $C_i \leftarrow \bigcup_{(C_1^j, \ldots, C_m^j) \in S} C_i^j$
  return $\mathcal{C}$
```

The algorithm $\texttt{HC}((T, \rho), \alpha)$ builds a hierarchal clustering $\mathcal{C}$ of tree T rooted at ρ using the LDD subroutine. Given the original tree metric $T = (V, E, d_T)$, we build the hierarchal tree as described in Definition 2.3 from the hierarchal clustering returned by $\texttt{HC}((T, v), \alpha)$ where $v \in V(T)$ is a vertex such that $\forall u \in V(T)$, $d_T(v, u) \leq \frac{\mathsf{diameter}(T)}{2}$ and α is a parameter to be set later.

Lemma 3.1. *For a tree metric $T = (V, E, d_T)$, with root $\rho \in V(T)$, and some $\alpha > 0$, the hierarchal tree $H(T)$ constructed by the hierarchal clustering $(C_0, C_1, \ldots, C_m) \leftarrow \texttt{HC}((T, \rho), \alpha)$ has two properties:*

(1) $\forall u, v \in V(T),\ d_{H(T)}(u, v) \geq \beta_1 \cdot d_T(u, v)$

(2) $\forall u, v \in V(T),\ \boldsymbol{E}\left[d_{H(T)}(u, v)\right] \leq \beta_2 \cdot d_T(u, v)$

Where $\beta_1, \beta_2 > 0$ are the dilation and contraction factors of the embedding respectively and the expectation in property (2) is over the randomness of the LDD *subroutine.*

Proof. Let $R = \mathsf{radius}(T, \rho)$ and let $d_\alpha(R)$ correspond to the value such that $\alpha^{d_\alpha(R)} = R$.

In order to show the two properties, let $u_0, u_k \in V(T)$ be two vertices and let $P_{u,v} = (u_0, u_1, \ldots, u_k)$ be the path from u_0 to u_k. Since $d_{H(T)}(u_0, u_k) = \sum_{i=0}^{k} d_{H(T)}(u_i, u_{i+1})$, it is sufficient to argue property (1) and (2) for any edge $(u, v) \in E(T)$.

Let us first show property (1). Let l be the smallest index such that for $i \neq j$, $(U_i, \_), (U_j, \_) \in C_l$ and $u \in U_i, v \in U_j$. I.e. let (u, v) be *cut* at level l in the hierarchal clustering. Since l is smallest such index, $d_T(u, v) \leq \frac{R}{\alpha^{l+1}}$. Since (u, v) was cut at level l, we have $d_{H(T)}(u, v) = \frac{R}{\alpha^l}$. Setting $\beta_1 \leftarrow \alpha$ proves property (1).

We now show property (2). Let A_i be the event that (u, v) gets cut at level i and $A_{<i}$ be the event that (u, v) is cut at level j such that $j < i$. Then, by the definition of expectation, we have

$$\begin{aligned}
\mathbf{E}\left[d_{H(T)}(u, v)\right] &= \sum_{i=1}^{d_\alpha(R)} \mathbf{Pr}\left[A_i\right] \cdot \frac{R}{\alpha^i} \\
\mathbf{Pr}\left[A_i\right] &= \mathbf{Pr}\left[A_i | \bar{A}_{<i}\right] (1 - \mathbf{Pr}\left[A_{<i}\right]) \leq \mathbf{Pr}\left[A_i | \bar{A}_{<i}\right] \\
\mathbf{Pr}\left[A_i | \bar{A}_{<i}\right] &= \frac{d_T(u, v)}{\frac{R}{\alpha^i}} \\
\therefore \mathbf{E}\left[d_{H(T)}(u, v)\right] &\leq d_\alpha(R) \cdot d_T(u, v)
\end{aligned}$$

We thus get that $\beta_2 \leq d_\alpha(R)$ and $\beta_1 \geq \alpha$. □

We will eventually use Lemma 3.1 in conjunction with Theorem 4 from [Bar96] to show that our algorithm is competitive on the original metric T. Given an algorithm on an embedded space, the Theorem from [Bar96] gives a bound on the competitiveness of the induced algorithm on the original metric space as a function of the competitiveness on the embedded space and the distortion of the embedding.

Fact 3.2. *The height of the tree induced by* `LDD`$((T, \rho), \alpha)$*, given by connecting the clusters outputted by* `LDD`*, is at most* $\alpha + 1$

The above fact follows from the fact that $\mathsf{radius}((T, \rho)) \leq R$ and $\mathsf{radius}((U, v)) \leq \frac{R}{\alpha}$ for every $(U, v) \in C$.

4 Our Algorithm

In this section, we give a monotone algorithm for our matching problem on hierarchal trees. In order to give a competitive algorithm on the original metric, we first embed our tree metric T into a hierarchal tree $H(T)$ as described in Section 3 and then use the algorithm described in this section.

In order to give an algorithm for hierarchal trees, we first give an algorithm, SpiderMatch, for a restricted class of hierarchal trees where the tree induced by every level of the clustering is a spider (i.e. hierarchal spiders). We then give an algorithm TreeMatch that is a reduction from general hierarchal trees to hierarchal spiders.

We prove the following Theorems in this section: (PUT IN SpiderMatch/TreeMatch ANALYSIS THM, MONOTONE THM, MASTER THM)

4.1 HTMatch

We define our algorithm HTMatch by giving an algorithm for the spider/tree induced by the sub-clusters formed by performing LDD on a cluster. Let $(C_0, C_1, \ldots, C_m)$ be the output of our hierarchal clustering as per Algorithm 2. When a request r_t arrives at time step t, let $(U_{l-1}, v) \in C_{l-1}$ be the cluster and root on level $l-1$ that r_t arrives at and let this be the lowest level of the hierarchal clustering such that r_t has unused servers in its cluster. In the construction of the hierarchal 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 hierarchal 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 $\mathcal{U}$.

We give an algorithm TreeMatch for trees that matches r_t to a vertex $u_i \in V(T_{\mathcal{U}})$ corresponding to matching $r_t \to U_i$ such that $U_i \in \mathcal{U}$. By maintaining an instance of TreeMatch for every cluster at every level of the hierarchal clustering, this induces a recursive algorithm for matching r_t to a server because the path from r_t to U_i arrives at a sub-cluster of U_i from which we run the same process as if the request r_t arrived at this sub-cluster. We denote this recursive algorithm by HTMatch .

4.2 Matching on spiders

We now give our algorithm, SpiderMatch, for a restricted class of hierarchal trees: hierarchal spiders. Let (T, ρ) be a spider with root ρ, degree d and height H. We give the algorithm in two parts; the sub-routine SpiderStates maintains a desired distribution over the "states" of the matching algorithm SpiderMatch, updates this distribution when a new request arrives and then outputs this desired distribution to SpiderMatch which updates its "states" so as to remain faithful to the distribution over states maintained by SpiderStates.

Let us label the d root to leaf paths by $\mathcal{L} = \{l_1, \ldots, l_d\}$ such that the first vertex in each path $l \in \mathcal{L}$ is a neighbor of ρ. We define three values for each vertex $u \in V(T)$ using which we determine how to match requests. Let $\eta(u)$ be the number of servers at u before any requests arrive. Let $A_t(u)$ be the number of requests that arrived at u before time t and let $O_t(u)$ be the number of requests before time t that arrived on a vertex in the path from u to the leaf but were matched to u. At every time step t, we define the function $f_t : V(T) \to \mathbb{R}$ such that for vertex $u \in V(T)$, $f_t(u) = \eta(u) - A_t(u) - O_t(u)$.

In order to define the algorithm, we need to define some terms to converse about regions of the tree that are important to the algorithm. We call a path $l \in \mathcal{L}$ *alive* at time t if there exists a vertex $u \in l$ such that $f_t(u) > 0$. Similarly, we call the path l *dead* if for all vertices $u \in l$ we have $f_t(u) \leq 0$. At every time step t, let us define two disjoint subsets of the vertices of T: $\mathcal{R}_t, \mathcal{F}_t$. $\mathcal{R}_t$ is the set of all vertices u such that for all vertices v in the path from u to ρ, we have $f_t(v) \leq 0$. We define $\mathcal{F}_t$ to be the set of vertices u such that u is the first vertex on path $l_u \in \mathcal{L}$ from ρ such that $f_t(u) > 0$. We denote $\mathcal{F}_t$ to be the frontier at time t.

SpiderStates is the workhorse of the algorithm. Our overarching technique is to maintain a distribution over *holes* - a hole is created by sending a non-collocated request to a vertex with an unmatched server. For all non-collocated requests that arrive on root-reachable vertices, SpiderStates, picks with some distribution, a root-reachable vertex with unused servers to create a hole. For non-collocated requests that arrive on non root-reachable vertices, SpiderStates creates a hole towards the root. The objective of SpiderStates is to maintain a distribution of holes on the root-reachable vertices with unused servers and update this distribution as requests arrive on vertices with holes. In essence, the algorithm's objective is to find the right home for every non-collocated request that arrives. Whenever it sends it to the wrong vertex, it creates a hole. So, the objective is to bound the sequence of holes created by a single non-collocated request before it reaches its intended vertex.

We give some notation before describing SpiderStates:

u_v	root of $T_{\mathcal{U}}$
u_{r_t}	vertex on which r_t arrives
$\mathcal{L}_t$	set of alive paths, at time t, out of all d paths from root
l_t	path in $\mathcal{L}_t$ on which request arrived
M_t	number of requests r_i such that $f_i(u_{r_i}) \leq 0$ for $i < t$
$\mathsf{MW}(\vec{c}, \vec{p}, \epsilon)$	call to Multiplicative Weights to update probability vector $\vec{p}$ for cost vector $\vec{c}$ with $\epsilon \in [0, 1/2]$ parameter for multiplicative update
p^t	probability vector in $\Delta^{\binom{d}{M_t}}$ for Multiplicative Weights algorithm with $\binom{d}{M_t}$ experts
$(\tilde{p}^t)_S$	probability that SpiderMatch created M_t holes at the vertices in $\{\mathcal{F}_t(l) \mid l \in S\}$ such that $S \in \binom{\mathcal{L}_t}{M_t}$

Remark 4.1. SpiderStates *will maintain the invariant that at any point in time t, every vertex $u \in V(T_{\mathcal{U}})$ at the frontier that has a hole has exactly one non-collocated request that matched to a server on u. This way, if a non-collocated request arrives on u at time $t' > t$, there has to be $N_S(u)$ collocated requests that arrived on u by time t'. This would mean that, at time t',* OPT *too would not have any unmatched servers at u.*

Algorithm 3: SpiderStates

```
SpiderStates ($T_{\mathcal{U}}$, $u_v$, $u_{r_t}$)
    case 1: $u_{r_t} \neq \mathcal{F}_t(l_t)$ and $f_t(u_{r_t}) \geq 1$
        Indicate SpiderMatch to match $r_t \to u_{r_t}$

    case 2: $u_{r_t} = \mathcal{F}_t(l_t)$ and $f_t(u_{r_t}) > 1$
        Indicate SpiderMatch to match $r_t \to u_{r_t}$

    case 3: $u_{r_t} = \mathcal{F}_t(l_t)$ and $f_t(u_{r_t}) = 1$ (* There was a hole at $u_{r_t}$ *)
        Let $S \in \binom{d}{M_t}$
        Initialize $c^t \in \mathbb{R}^{\binom{d}{M_t}}$ such that $(c^t)_S = \begin{cases} 1 & \text{if } l_t \in S \\ 1 & \text{if } \exists \text{ dead path} \in S \\ 0 & \text{o/w} \end{cases}$
        $(p^{t+1})_S \leftarrow \mathsf{MW}(c^t, p^t, \epsilon)$
        $\forall S \in \binom{\mathcal{L}_t}{M_t}$, $(\tilde{p}^{t+1})_S \leftarrow \frac{(p^{t+1})_S}{\sum_{T \in \binom{\mathcal{L}_t}{M_t}} (p^{t+1})_S}$
        $\forall S \mid \exists$dead path $\in S$, $(\tilde{p}^{t+1})_S \leftarrow 0$

    case 4: $r_t$ is a request such that $u_{r_t} \in \mathcal{R}_t$ (* New non-collocated request *)
        $M_{t+1} \leftarrow M_t + 1$
        if $M_{t+1} = |\mathcal{L}_t|$ then
            Exit SpiderStates, SpiderMatch and play the greedy algorithm on $T_{\mathcal{U}}$
        else
            if No collocated requests have arrived then
                Initialize $\tilde{p}^{t+1} \in \Delta^{\binom{d}{M_{t+1}}}$
                $\forall S \in \binom{d}{M_{t+1}}$, $(p^t)_S \leftarrow \frac{1}{\binom{d}{M_{t+1}}}$
                $\forall S \in \binom{d}{M_{t+1}}$, $(\tilde{p}^{t+1})_S \leftarrow (p^t)_S$ (* All paths are alive *)
            else
                $\tilde{p}^{t+1} \leftarrow$ Run instance of SpiderStates with all $M_t + 1$ non-collocated requests
                arriving at start at root.

    case 5: $r_t$ is a non-collocated request such that $u_{r_t} \notin \mathcal{R}_t, u_{r_t} \neq \mathcal{F}_t(l_t)$
        $u_l$: closest vertex to $u_{r_t}$ on path to $u_v$ such that $f_t(u_l) \geq 1$
        if $u_l \notin \mathcal{F}_t$ or $f_t(u_l) > 1$ then
            Indicate SpiderMatch to match $r_t \to u_l$
        else
            $\mathcal{R}_{t+1} \leftarrow \mathcal{R}_t \cup u_l$
            $\mathcal{F}_{t+1} \leftarrow \mathcal{F}_t \backslash \{u_l\}$
            $\tilde{p}^{t+1} \leftarrow$ Run instance of SpiderStates with all $M_t$ non-collocated requests arriving
            at start at root and with $l_t \leftarrow l_t \backslash \{u_l\}$ and without $r_i$ such that $u_{r_i} = u_l$.
    return $\tilde{p}^{t+1}$
```

We give an observation about the distributions $\tilde{p}^t$ from Algorithm 3:

Observation 4.2. *For request r_t that arrived on vertex $u_{r_t} \in V(T_{\mathcal{U}})$, satisfying the conditions for* ***case 3*** *of* SpiderMatch*:*
(1) For any $S \in \binom{\mathcal{L}_t}{M_t}$ such that $l_t \in S$, $(\tilde{p}^{t+1})_S < (\tilde{p}^t)_S$
(2) For any $S \in \binom{\mathcal{L}_t}{M_t}$ such that $l_t \notin S$, $(\tilde{p}^{t+1})_S > (\tilde{p}^t)_S$
This follows from the Multiplicative Weights *algorithm. For weights $(w^t)_S$ of set S at time t, we have $(w^{t+1})_S = (w^t)_S(1 - \epsilon \cdot (c^t)_S)$. Since $(w^{t+1})_S < (w^t)_S$ for $l_t \in S$ and $(w^{t+1})_S = (w^t)_S$, using cost vectors in* ***case 3*** *of* SpiderStates*, the observation follows.*

SpiderStates is the algorithm that maintains a desired distribution over states of SpiderMatch. The state, at time t, was given by the set $S \in \binom{\mathcal{L}_t}{M_t}$ on which the algorithm created the M_t holes at time t. SpiderMatch will exist in exactly one of these states, hence, it is left to show that SpiderMatch can transition between states with the "right" probabilities when SpiderStates enters **case 3**, **case 4**, **case 5**. I.e. at time t, given some $\tilde{p}^{t+1}$, if SpiderStates returned from **case 3**, **4**, or **5**, we need to show that the algorithm can transition to a state such that the probability SpiderMatch is in state S at time $t+1$ is $(\tilde{p}^{t+1})_S$ given that distributions $\tilde{p}^1, \dots, \tilde{p}^t$ describe the probability that SpiderMatch was in state S' with probability $(\tilde{p}^i)_{S'}$ at time i for all $i \leq t$.

We define some notation for SpiderMatch, for request r_t that arrived at vertex u_{r_t} on path l_t at time t:

$\mathcal{X}(t)$	$\{S \subseteq \mathcal{L}_t \mid l_t \in S\}$ such that $S \subseteq \mathcal{L}_t$ and $\lvert S \rvert = M_t$
$\mathcal{Y}(t)$	$\{S \subseteq \mathcal{L}_t \mid l_t \notin S\}$ such that $S \subseteq \mathcal{L}_t$ and $\lvert S \rvert = M_t$
$\mathcal{U}_{T\Delta_1}$	Denotes $\{S \in \mathcal{U} \mid \lvert (T\Delta S) \cup \{l_t\} \rvert \leq 2\}$ for collection of sets $\mathcal{U}$
$\mathcal{U}_{\subset T}$	Denotes $\{S \in \mathcal{U} \mid S \subset T\}$ for collection of sets $\mathcal{U}$
$G(A, B, E, c)$	Bipartite flow network with vertex sets A, B, source and sink s and t, edges E and edge-flow capacities $c : E \to \mathbb{R}^+$

We notice that if $M_{t+1} = M_t$ (i.e. **case 3**, **5** of SpiderStates), SpiderMatch, can transition only from states $S \in \mathcal{X}(t)$ since this corresponds to moving the hole from $F_t(l_t)$. Additionally, it can only transition to states $T \in \binom{\mathcal{L}_{t+1}}{M_t}$ such that $T = S$ or $T\Delta S = \{l_t, l\}$, corresponding to moving the hole to some $\mathcal{F}_t(l)$. If $M_{t+1} = M_t + 1$ (i.e. **case 4** of SpiderStates), SpiderMatch, can transition from states $S \in \binom{\mathcal{L}_t}{M_t+1}$ and can only transition to $T \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}$ such that $S \subset T$ corresponding to moving the new non-collocated request to a path that doesn't have a hole. Our objective in SpiderMatch is to achieve the following for the $M_{t+1} = M_t$ and $M_{t+1} = M_t + 1$ case respectively:

$$\forall T \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}, \sum_{S \in \mathcal{X}(t)_{T\Delta_1}} (\tilde{p}^t)_S \mathbf{Pr}\left[\text{SpiderMatch transitions } S \to T \mid (\tilde{p}^t)_S\right] = (\tilde{p}^{t+1})_T$$

$$\forall T \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}, \sum_{S \in \binom{\mathcal{L}_t}{M_t}_{\subset T}} (\tilde{p}^t)_S \mathbf{Pr}\left[\text{SpiderMatch transitions } S \to T \mid (\tilde{p}^t)_S\right] = (\tilde{p}^{t+1})_T$$

Algorithm 4: SpiderMatch

```
SpiderMatch ($T_{\mathcal{U}}$, $u_v$, $u_{r_t}$)
    Run SpiderStates($T_{\mathcal{U}}, u_v, u_{r_t}$)
    Let $S_{\mathrm{A}} \in \binom{\mathcal{L}_t}{M_t}$ be state of SpiderMatch at time $t$
    case 1: SpiderStates indicated $r_t \to u_{r_t}$
        Match $r_t \to u_{r_t}$
    case 2: $u_{r_t} \in \mathcal{R}_t$ and no collocated requests arrived:
        Match uniformly among paths in $\mathcal{L}_t \backslash S_A$
    case 3: SpiderStates returned from case 3, case 4, case 5
        We define a matching of $r_t$ for every possible state $T \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}$
        if $S_A \in \mathcal{Y}(t)$ then
            match $r_t \to u_{r_t}$
        else
            case $M_t = M_{t+1}$
                $A \leftarrow \mathcal{X}(t)$, $B \leftarrow (\mathcal{X}(t+1) \cup \mathcal{Y}(t+1))$
                $E = \begin{cases} (s, S_a) & \text{such that } S_a \in A \\ (S_b, t) & \text{such that } S_b \in B \\ (S_a, S_b) & \text{such that } S_a \in \mathcal{X}(t)_{S_b \Delta_1},\ S_b \in B \end{cases}$
            case $M_{t+1} = M_t + 1$
                $A \leftarrow \mathcal{X}(t) \cup \mathcal{Y}(t)$, $B \leftarrow (\mathcal{X}(t+1) \cup \mathcal{Y}(t+1))$
                $E = \begin{cases} (s, S_a) & \text{such that } S_a \in A \\ (S_b, t) & \text{such that } S_b \in B \\ (S_a, S_b) & \text{such that } S_a \in (\mathcal{X}(t) \cup \mathcal{Y}(t))_{\subset S_b},\ S_b \in B \end{cases}$
            $c = \begin{cases} (\tilde{p}^t)_{S_a} & \text{for } (s, S_a) \\ (\tilde{p}^{t+1})_{S_b} & \text{for } (S_b, t) \\ \infty & \text{for } (S_a, S_b) \end{cases}$
            Compute Max-Flow($G(A, B, E, c)$)
            Flow $f((S_a, S_b)) = (\tilde{p}^t)_{S_a} \mathbf{Pr}\left[\text{SpiderMatch transitions } S_a \to S_b \mid (\tilde{p}^t)_{S_a}\right]$
```

Lemma 4.3. *The bipartite flow network in Algorithm 4 has max-flow* $\sum_{S \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}} (\tilde{p}^{t+1})_S = 1$

Proof. Since the edges between vertices in A, B have capacity ∞, it is sufficient to show that the subgraph induced by A, B is connected since this implies that the min-cut of this graph must be $\sum_{S \in \binom{\mathcal{L}_t}{M_t}} (\tilde{p}^t)_S = \sum_{S \in \binom{\mathcal{L}_{t+1}}{M_{t+1}}} (\tilde{p}^{t+1})_S$. This in conjunction with the Max-Flow Min-Cut Theorem proves the result.

Pick two sets $B_1, B_2 \in B$. To show that the subgraph is connected, it is sufficent to argue that there is a path between these two vertices, using vertices only in A, B, because every vertex $a \in A$ has at least one edge to B.

We case on whether $M_{t+1} = M_t$ or $M_{t+1} = M_t + 1$. Let us consider the former case first – We induct on $|B_1 \Delta B_2|$. In the base case, let $B_1 \Delta B_2 = \{b_1, b_2\}$, such that $b_1 \in B_1, b_2 \in B_2$. If $l_t \in B_1$

and $l_t \notin B_2$, then the set $B_1 \in A$ and has an edge to B_1 and B_2. If $l_t \notin B_1, B_2$, then the set $(B_1 \backslash \{b_1\}) \cup \{l_t\} \in A$ and has an edge to B_1 and B_2. If $l_t \in B_1, B_2$, then $B_1 \in A$ which has an edge to the set $X = (B_1 \backslash \{l_t\}) \cup \{b_2\} \in B$ which has an edge to $(X \backslash \{b_1\}) \cup \{l_t\} = B_2 \in A$ which has an edge to $B_2 \in B$. The inductive case follows by performing the same case analysis and process by picking any two dissimilar elements $b_1, b_2 \in B_1 \Delta B_2$.

If $M_{t+1} = M_t + 1$, we induct on $|B_1 \backslash B_2|$. In the base case where we have $|B_1 \backslash B_2| = 1$, let this be $b_1 \in B_1$. We then have an edge from B_1 to $(B_1 \backslash \{b_1\}) \in A$ which has an edge to B_2. If $|B_1 \backslash B_2| = k$, pick $b_1 \in B_1, b_2 \in B_2$ to be a pair of elements such that $b_1 \notin B_2$ and $b_2 \notin B_1$. $(B_1 \backslash \{b_1\}) \in \binom{\mathcal{L}_t}{M_t}$ since $\mathcal{L}_t \subseteq \mathcal{L}_{t+1}$. But this set has an edge to $(B_1 \backslash \{b_1\}) \cup \{b_2\}$ which has $k-1$ dissimilar elements with B_2. □

Lemma 4.3 shows that SpiderMatch is a well defined algorithm and can match requests $r_1, \ldots, r_t$ such that the $\tilde{p}^t_S$ is the probability that there are M_t holes at $\{\mathcal{F}_t(l) \mid l \in S\}$ for all $S \in \binom{\mathcal{L}_t}{M_t}$.

So as to give a bound on the competitive ratio of SpiderMatch on $T_{\mathcal{U}}$, we must first give some lower bound on OPT. To that end, we make the following observation

Observation 4.4. *At time t, the optimum matching must have had at least M_t non-collocated requests over the request sequence $R = \langle r_1, \ldots, r_t \rangle$ on the rooted tree $T_{\mathcal{U}}, \rho$ where M_t is as per notation for* SpiderStates$((T_{\mathcal{U}}, \rho), R)$.

Proof. Notice that $M_{t+1} = M_t + 1$ only when SpiderStates enters **case 4** which requires that $u_{r_t} \in \mathcal{R}_t$. By the definition of $\mathcal{R}_t$, we have that $N^{(t)}_M(u_{r_t}) = N_S(u_{r_t}) = 0$. A request is matched to a server with probability 1 under SpiderStates either when it is collocated (**case 1, 2**) or, as in **case 5**, because a non-collocated request arrived deeper in the path $l(u_{r_t})$ and created a series of holes till u_{r_t}. Hence, the hole at u_{r_t} due to this earlier non-collocated request is not yet accounted for by M_t. This combined with Remark 4.1 gives us the observation. □

SpiderMatch pays positive cost when SpiderStates enters **case 3**, **4**, **5** since the request is non-collocated. Our main theorem gives a bound on the expected number of times SpiderMatch enters **case 3** and **4** based on $\max_t M_t$

Theorem 4.5. *Let (T, ρ) be a rooted spider, with height H and degree d, let $\langle r_1, \ldots, r_t \rangle$ be any request sequence with M non-collocated requests arriving in $\mathcal{R}_0$ at the start of the request sequence and all r_i such that $i > M$ are collocated requests (as per OPT). Let $\mathcal{P} = \langle p^1, \ldots, p^t \rangle$, $\tilde{\mathcal{P}} = \langle \tilde{p}^1, \ldots, \tilde{p}^t \rangle$ and $\langle c^1, \ldots, c^t \rangle$ be the probability vectors and cost vectors given by running* SpiderStates(T). *Let $\mathbf{1}^{\mathsf{SM}}(i)$ be the indicator function which is 1 if* SpiderMatch(T) *pays positive cost to match r_i or 0 otherwise.* SpiderMatch(T) *has the following two properties:*

$$\forall i > M, \ \boldsymbol{E}_{\tilde{\mathcal{P}}}\left[\mathbf{1}^{\mathsf{SM}}(i)\right] \leq \vec{c}_i \cdot \vec{p}_i \tag{4.1}$$

$$\sum_i \vec{c}_i \cdot \vec{p}_i \leq M \cdot \left((1+\epsilon)H + \frac{\ln(d)}{\epsilon} + 1\right) \tag{4.2}$$

Proof. Since all non-collocated requests arrive at the start and in $\mathcal{R}_0$, we can assume that for every r_i such that $i > M$, the request arrived at the frontier because otherwise it is definitely collocated.

We first prove (1). By definition of $\tilde{p}^i$ in Algorithm 3, we know that

$$\mathbf{E}_{\tilde{\mathcal{P}}}\left[\mathbf{1}^{\mathsf{SM}}(i)\right] = \sum_{S\in\mathcal{X}(i)} (\tilde{p}^i)_S = \sum_{S\in\mathcal{X}(i)} \frac{(p^i)_S}{\sum_{T\in\binom{\mathcal{L}_i}{M}}(p^i)_T}$$

Let $Q = \sum_{S\in\binom{d}{M}\ \mid\ \exists \text{ dead leg in } S}(p^i)_S$. Let δ_S be defined for all $S \in \binom{\mathcal{L}_i}{M}$ such that $(\tilde{p}^i)_S = (p^i)_S + \delta_S$. Since $\tilde{p}$ and p are probability distributions, we know $\sum_{S\in\binom{\mathcal{L}_i}{M}} \delta_S = Q$. By definition of c^i in Algorithm 3 we have,

$$\begin{aligned} c^i \cdot \vec{p}_i &= \sum_{S\in\binom{d}{M}\ \mid\ \exists \text{ dead leg in } S} (p^i)_S + \sum_{S\in\mathcal{X}(i)} (p^i)_S \\ &= \sum_{S\in\binom{\mathcal{L}_i}{M}} \delta_S + \sum_{S\in\mathcal{X}(i)} (p^i)_S \\ &\geq \sum_{S\in\mathcal{X}(i)} \delta_S + \sum_{S\in\mathcal{X}(i)} (p^i)_S = \sum_{S\in\mathcal{X}(i)} (\tilde{p}^i)_S \end{aligned}$$

Now we show (2). The Multiplicative Weights guarantee from [AHK12] gives us that

$$\forall S \in \binom{d}{M},\ \sum_i c^i \cdot \vec{p}_i \leq (1+\epsilon)\sum_i (c^i)_S + \frac{\ln\binom{d}{M}}{\epsilon}$$

Let us choose S to be the last M paths that die. At most H is paid for the first path in S to die. Then, since these are the last M living paths, at most $(M-1)H$ is paid before they all die. The result follows. $\square$

The proof for Theorem 4.5 assumes that all the non-collocated requests arrive at the start. This is an overestimate of the cost. If a non-collocated request arrived at time i in $\mathcal{R}_i$ after some collocated requests arrived, then SpiderStates continues in state $i+1$ as though r_i arrived at the start (as per **case 4**) and hence the cost incurred on $r_{i+1}, \ldots, r_t$ is the same. Moreover, the cost on $r_1, \ldots, r_i$ is strictly lower sine there were fewer non-collocated requests and hence fewer holes.

When a non-collocated request r_i doesn't arrive in $\mathcal{R}_i$ or the frontier, it creates at most H holes before reaching $\mathcal{R}_j$ at some time $j > i$. But, by the construction of **case 5** in Algorithm 3, at time j we continue as though r_i arrived at the start as a non-collocated request at $\mathcal{R}_0$. Apart from the additional H holes SpiderMatch must pay for for the request to reach $\mathcal{R}_j$, the cost to SpiderMatch is upper bounded by the same argument as above if r_i arrived at $\mathcal{R}_0$ at the start.

4.3 From spiders to trees

4.4 TreeMatch

The next algorithm, `TreeMatch`, will transform general trees into spiders and use the previously created SPIDERMATCH algorithm to match requests. The algorithm is as follows: given a tree $T = (V, E, d_T)$ with root q, define $L : V \to \mathbb{N}$ to be the number of leaf servers descended from v. We will create a spider graph T' of degree $d = L(q)$, the number of leaf servers in the graph. Each leg is a copy of the path from the root q to the corresponding leaf server. Whenever a request arrives at $v \in T$, we will send a request to T' at any copy v' of v. Let $s' \in T'$ be the server that T' uses to handle the request at v', where $s \in T$ is the original version of s'. After handling the request at v', we will send $L(s) - 1$ more requests, one at each available copy of s, to ensure that there are no more copies of s remaining in T'. In T, we will use s to handle the request of v.

Note that because we are sending the extra requests, we are ensuring that whatever server we match to in T' will be available in T. Since we will have to pay for matching v' to s' in T', it follows that the expected cost of our algorithm on T' is at least the expected cost of the algorithm on T.

4.5 Analyzing our algorithm

Lemma 4.6. *For any hierarchal tree $H(T) = (V, E)$, with parameter α, hierarchal clustering $(C_0 \ldots, C_m)$, set of servers $S : [n] \to V$ and online request sequence $\vec{r} = \langle r_1, \ldots, r_n \rangle$, the algorithm HTMatch on $H(T), \vec{r}$ is a $O(\alpha^2 \cdot d_\alpha^3)$-competitive algorithm. Where $d_\alpha(R)$, abbreviated with d_α, denotes the number i such that $\frac{R}{\alpha^i} = 1$.*

Proof. HTMatch maintains an instance of SpiderMatch in every cluster at every level. For every level i and every cluster $(U, v) \in C_i$, let $M^{\text{out}}(U)$ be the number of non-collocated requests at level i that were matched to cluster U from the algorithm at level i. Similarly, we define $M^{\text{in}}(U)$ to be the number of non-collocated requests (that arrived at root-reachable clusters with no unmatched servers) that arrived in U for which the lowest level they were non-collocated is level $i + 1$. Note that the holes created due to these non-collocated requests never exited U.

We bound the cost incurred by the sequence of holes created by all the $M^{\text{in}}(U) + M^{\text{out}}(U)$ non-collocated requests at every level and for every cluster U at that level. Notice that every non-collocated request in $M^{\text{out}}(U)$ is created by a non-collocated request at a higher level. Hence, it is sufficient to bound the cost of creating the sequence of holes, across levels, that requests in $M^{\text{in}}(\cdot)$ create. By Observation 4.4 we know that the number of non-collocated requests over $\vec{r}$ on T_U (the tree induced by the clustering of U at level $i + 1$) for OPT is $M^{\text{in}}(U)$.

We know that the expected number of holes created by M non-collocated requests on a spider T with degree d and height H that arrive on root-reachable vertices is $M \cdot (\alpha(1 + \epsilon)H + \frac{\ln d}{\epsilon})$ by Theorem 4.5. In addition, if a non-collocated request arrives deeper on one of the paths of the spider (i.e. not on a root-reachable vertex), it will create at most H extra holes until it reaches a root-reachable vertex. Since the algorithm is oblivious to the request that creates the hole, we can assume that each of the M requests creates $\alpha(1 + \epsilon)H + \frac{\ln d}{\epsilon}$ holes in expectation.

By these arguments, it is sufficient to bound the cost incurred by a single non-collocated request r in $M^{\text{in}}(U)$ where (U, v) is a cluster with radius R. Let U be at level $i-1$ and let the subsclusters of U be given by $\{U_1, \ldots, U_l\}$ such that $v \in U_v$ is the root cluster. In the worst case, we assume that r arrived deep in a path from the root cluster in a non root-reachable cluster and hence had to pay for the extra H holes (on level i) to get to a root-reachable cluster. Let the cost for r to reach a root-reachable cluster in a rooted clustering, (U, v), with radius R be $U(R)$. Similarly, let $D(R)$ be the cost incurred in creating the sequence of holes once it reaches a root-reachable cluster. Let $A(R) = D(R) + U(R)$ be the total cost incurred by r for creating the sequence of holes across levels. Let $d_\alpha(R)$, sometimes abbreviated with d_α, denote the number i such that $\frac{R}{\alpha^i} = 1$. We assume $\alpha = \log^c(n)$ and note that $d_\alpha(R) \le \frac{\log(n)}{c \log\log(n)}$. We then show a bound for $D(R)$

$$\begin{aligned} D(R) &\le \left(\alpha(1+\epsilon) + \frac{\log(n)}{\epsilon}\right) D\left(\frac{R}{\alpha}\right) + R\left(\alpha(1+\epsilon) + \frac{\log(n)}{\epsilon}\right) \\ &\le d_\alpha(R) \frac{R}{\alpha^{d_\alpha(R)}} \left(\alpha(1+\epsilon) + \frac{\log(n)}{\epsilon}\right)^{d_\alpha(R)+1} \end{aligned}$$

By setting $\epsilon \le \frac{1}{d_\alpha(R)}$

$$\begin{aligned} &\le d_\alpha \frac{R}{\alpha^{d_\alpha}} \left(\alpha + (\log(n))^{c-1} + \log^2(n)\right)^{d_\alpha+1} \\ &\le d_\alpha R \left(\alpha + (\log(n))^{c-1} + \log^2(n)\right) \sum_{i=0}^{d_\alpha} \binom{d_\alpha}{i} \left(\frac{(\log^{c-1}(n) + \log^2(n))}{\alpha}\right)^i \end{aligned}$$

For $c \ge 3$

$$\begin{aligned} &\le d_\alpha^2 R \left(\alpha + \log^{c-1}(n) + \log^2(n)\right) \\ &\le 3 d_\alpha(R)^2 R\alpha \end{aligned}$$

Next we will bound $U(R)$. Recall that $U(R)$ is simply the cost of bringing r up to the root cluster, where it will pay $\frac{R}{\alpha}$ for each of the $\mathsf{height}(U)$ clusters it goes through. From Fact 3.2, we know that $\mathsf{height}(U) \le \alpha + 1$ Thus, we have that

$$\begin{aligned} U(R) \le (\alpha+1)A\left(\frac{R}{\alpha}\right) + R &\le (\alpha+1)U\left(\frac{R}{\alpha}\right) + (\alpha+1)D\left(\frac{R}{\alpha}\right) + R \\ &\le (\alpha+1)U\left(\frac{R}{\alpha}\right) + R(3d_\alpha(R)^2\alpha + 1) \\ &\le O(R\alpha d_\alpha(R)^3) \end{aligned}$$

Thus $A(R) \le U(R) + D(R) \le R\alpha d_\alpha(R)^3$. Since OPT must pay at least $\frac{R}{\alpha}$ for r, $\mathsf{HTMatch}$ is $O(\alpha^2 d_\alpha^3)$-competitive. ☐

Lemma 4.7. *$\mathsf{HTMatch}$ is a monotone matching algorithm.*

Proof. Given points p_1, p_2 and available server s where p_2 lies on the path from p_1 to s, we need to show that

$$P(\text{request } r \text{ matches to } s \mid r = p_1) \leq P(\text{request } r \text{ matches to } s \mid r = p_2).$$

If $p_1 = s$, then $P(\text{request } r \text{ matches to } s \mid r = p_1) = 1$ and we are done. If $p_1 \neq s$, let level C_i be the first level of the clustering in which p_1 and s lie in different clusters. Because of rule 2 above, they will lie in different clusters for the remainder of the levels. Recall that the algorithm is defined in each cluster, such that algorithm will start at C_0 and continue matching into lower and lower levels until it arrives at an individual server. On each level, the algorithm is clearly monotone since the probability of matching to any given server is equal over all vertices within a connected component consisting only of vertices not containing open servers. Let $(U^\ell_{k_\ell})$ denote the sequence of partitions that s belongs to, $(U^\ell_{k'_\ell})$ denote the sequence of partitions that p_1 belongs to, and $(U^\ell_{k^*_\ell})$ the sequence of p_2. Note that by construction of the algorithm

$$\begin{aligned}
&P(\text{request } r \text{ matches to } s \mid r = p_1) \\
&\quad = \prod_{\ell \geq i} P(\text{request } r \text{ matches to } U^\ell_{k_\ell} \ni s \mid U^\ell_{k'_\ell} \ni r) \\
&\quad \leq \prod_{\ell \geq i} P(\text{request } r \text{ matches to } U^\ell_{k_\ell} \ni s \mid U^\ell_{k^*_\ell} \ni r) \\
&\quad = P(\text{request } r \text{ matches to } s \mid r = p_2)
\end{aligned}$$

where the third line follows from the third by the monotonicity of each level as pointed out above. □

We give our final concluding theorem

Theorem 4.8. *For a tree metric $T = (V, E, d_T)$, the algorithm corresponding to building a hierarchal tree $H((T, \rho))$, with parameter $\alpha = \log^3(n)$, and running HTMatch is monotone and a $O\left(\frac{\log^{13}(n)}{\log\log(n)}\right)$ competitive algorithm.*

Using Lemma 3.1 in conjunction with Theorem 4 from [Bar96] and Lemma 4.6 gives us that this algorithm is a $\alpha^3 d_\alpha^4$-competitive algorithm. Setting $\alpha = \log^3(n)$ and $d = \frac{\log(n)}{3\log\log(n)}$ in conjunction with Lemma 4.7 gives us the above theorem.

5 Pricing Monotone Algorithms

5.1 Deterministic Algorithms

In this section we describe the process of transforming monotone algorithms on trees into pricing schemes. Here, a **monotone** algorithm is an algorithm which satisfies the following:

- For any $u, v \in V$ and $s \in S$ where v lies on the path from u to s, it holds that $\mathbb{P}(u \to s) \leq \mathbb{P}(v \to s)$.

A **partitioning scheme** on a tree is a set of disjoint subsets of the tree whose union is the whole tree. Deterministic monotone algorithms and partitioning schemes share the following relation.

Lemma 5.1. *Any monotone deterministic algorithm $\mathcal{A}$ induces a partitioning scheme P that also satisfies:*

1. *In each nonempty part $A \in P$ there is a single server designated as leader.*
2. *Each part is connected with respect to the tree.*

Proof. Let the sets $Q_1, ..., Q_k$ be defined by $Q_i = \{v : v \to_{\mathcal{A}} s_i\}$ and set $P = \{Q_i : Q_i \neq \emptyset\}$. 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 $1 = \mathbb{P}(u \to_{\mathcal{A}} s_i) \leq \mathbb{P}(s_i \to_{\mathcal{A}} s_i) \leq 1$, 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 $\mathbb{P}(u \to_{\mathcal{A}} s_i) = \mathbb{P}(v \to_{\mathcal{A}} s_i) = 1$ and thus any vertex between u and v must also match to s_i. Thus these components are connected. $\square$

We will call partitioning schemes that satisfy Lemma 5.1 **monotone partitions**. Note that monotone partitions also induce monotone deterministic algorithms and are hence synonymous with them.

We now show how to transform a monotone partition $P = \{Q_1, ..., Q_k\}$ into a pricing scheme $p : S \to \mathbb{R}$. Set the price of the leader s_1 of Q_1 to be any constant $p(s_1) = c$. We then proceed iteratively as follows: let Q_j be any part with leader s_j adjacent to a part already priced. Then there exists a $v \in Q_j$ adjacent to a $u \in V$ such that u's leader s_i has already been priced.

We set the price $p(s_j) = p(s_i) + d(u, s_i) - d(v, s_j)$ to satisfy

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

When all leaders have been priced, price any non leaders at ∞. This pricing scheme acts exactly as the monotone partition.

Theorem 5.2. *For any monotone partition scheme P, the pricing scheme p described above simulates P. That is, if $v \to_P s_i$, then $d(v, s_i) + p(s_i) < d(v, s_j) + p(s_j)$ for all $j \neq i$.*

Proof. 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. $\square$

5.2 Randomized Algorithms

Now that we can price monotone deterministic algorithms, it remains to show that any monotone randomized algorithm is can be written as a distribution over monotone deterministic algorithms. To do this, we will show that we can define a distribution $\mathcal{P}$ over monotone partitions that simulates the algorithm. For the remainder of this section, we will assume our algorithm $\mathcal{A}$ defines for each vertex v a distribution $(v_1 := \mathbb{P}(v \to_{\mathcal{A}} s_1), ..., v_k := \mathbb{P}(\to_{\mathcal{A}} s_k))$ that determines its probability of matching to each server. We will use the notation $\mathbb{P}_T(D)$ to denote the probability that partition D is chosen on graph T and let $D_i^v = \{D | v \to_D s_i\}$.

Now we show by induction on the number of vertices in the tree T that any monotone algorithm on a tree induces a distribution over monotone partitions of T. Here, this means we are defining $\mathcal{P}$ such that $\sum_{D \in D_i^v} \mathbb{P}_T(D) = v_i$ for all vertices $v \in T$. Our induction will proceed by contracting leaves of the tree and any servers located on those leaves, building the partition on the smaller tree, and then extending those partitions back onto the larger tree.

When contracting downwards, the leaf vertex (server(s)) will inherit the distribution of the vertex onto which they're contracted. Thus if $s_1, \ldots, s_t$ are leaf servers and v is their neighbour, after contraction all of $s_1, \ldots, s_t$ will have distribution $v_1, \ldots, v_k$

Lemma 5.3. *For any monotone algorithm $\mathcal{A}$ on a single tree T there exists a distribution $\mathcal{P}$ over monotone partitions of the tree such that $\mathbb{P}(v \to_{\mathcal{P}} s_i) = \mathbb{P}(v \to_{\mathcal{A}} s_i)$ for all vertices $v \in T$ and all servers s_i.*

Proof. The base case of this induction will be the final contraction where all servers are collocated on one vertex v. In this case, each possible partition will just be that one point with one server chosen as a leader. For each such partition A_i with leader s_i, we will just assign the partition the probability v_i.

Let us call the original tree T and let us assume we are contracting t colocated servers on vertex u which is a leaf and is adjacent to some vertex v. Let us call the smaller tree T' where we contract the servers onto v.

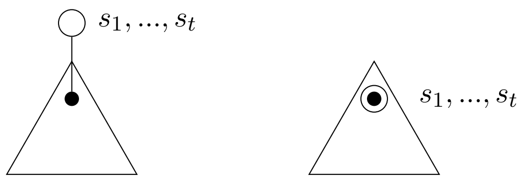

Figure 5.1: An example of a contraction

As in Figure 1, we will extend every partition on the contracted graph T' by uncontracting the edge to obtain the old graph T and defining new partition(s) with new probabilities. For any partitions $F_1, ...$ of T created from a partition F of T', we will maintain the constraint that

$$\sum_i \mathbb{P}_T(F_i) = \mathbb{P}_{T'}(F) \tag{5.3}$$

Subject to the above constraint we must assign probabilities to the new partitions such that $\mathbb{P}(v \to_{\mathcal{P}} s_i) = v_i$.

In particular, there are only two cases to consider for the partition F:

5.2.1 Case 1

One of $s_1, ..., s_t$ is a leader of F.

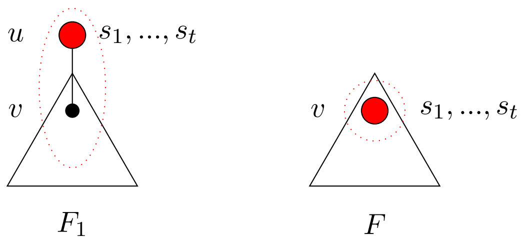

Figure 5.2: An example of a case 1 contraction: here, the red dotted circle denotes a part in a partitioning with a red filled circle as its leader

In this case, only one new partition F_1 will be created from F and will inherit its probability: $\mathbb{P}_T(F_1) = \mathbb{P}_{T'}(F)$ so as to satisfy constraint (5.3).

5.2.2 Case 2

The leader of F is $s' \notin \{s_1, ..., s_t\}$.

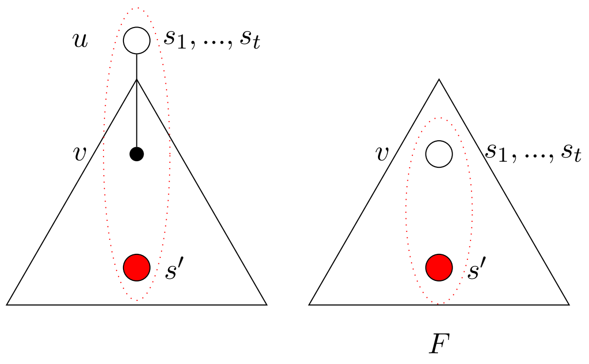

Figure 5.3: An example of case 2

We will extend F into $t+1$ new partitions. For partitions $F_1, ..., F_t$, we will extend the partitioning to make s_i the leader of its own singleton part.

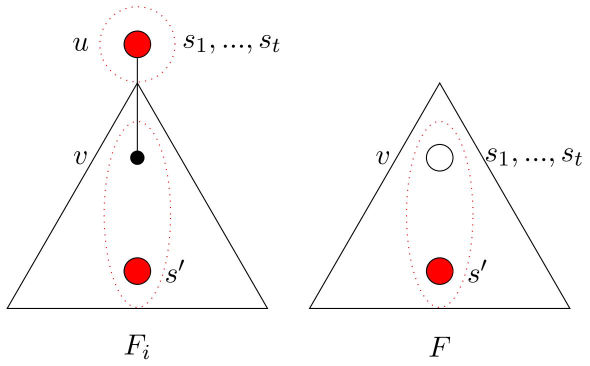

Figure 5.4: An example of case 2 F_i for $1 \leq i \leq t$

The final partitioning F_{t+1} created from F extends the partition of s' to include u:

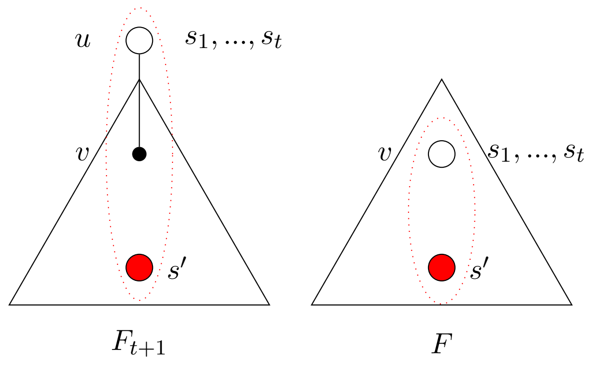

Figure 5.5: An example of case 2 F_i for $1 \leq i \leq t$

Notice that the way we're extending the partitions of T' to T and with constraint 5.3, we only need to make sure we're satisfying the distribution for u since all other vertices' distributions remain satisfied under our extension.

By our inductive constraint we know that for any $i \notin \{1, ..., t\}$ we get a distribution over partitions of T' such that $\sum_{D \in D_i^v} \mathbb{P}_{T'}(D) = v_i$, where $v_i \geq u_i$ by monotonicity. We will let $-\Delta_i = u_i - \sum_{D \in D_i^v} \mathbb{P}_{T'}(D)$.

Notice that each $D \in D_i^v$ gets extended to an equivalent partition in D_i^u.

Initially, we set $\mathbb{P}_T(F_{t+1}) = \mathbb{P}_{T'}(F)$, but then greedily remove Δ_i across all $F_{t+1} \in D_i^u$ without violating the constraint that all $F_{t+1} \in D_i^u$ must have non-negative probability. After completing this, we have that $\sum_{D \in D_i^u} \mathbb{P}_T(D) = u_i$ for all $i \notin \{1, ..., t\}$.

We now only need to show that we satisfy that $\sum_{D \in D_j^u} \mathbb{P}_T(D) = u_j$ for all $j \in \{1, ..., t\}$. Specifically, we want for each s_j:

$$\sum_{D \in D_j^u} \mathbb{P}_T(D) = \sum_{F \in D_j^v} \mathbb{P}_T(F_1) + \sum_{i>t} \sum_{F \in D_i^v} \mathbb{P}_T(F_j) = u_j. \tag{5.4}$$

The probabilities in the first summation in (5.4) have already been assigned in Case 1, so we must ensure that the probability we assign to each F_j satisfies both (5.3) and $\sum_{F \in D_j^v} \mathbb{P}_T(F_j) = u_j$.

Let us define $\Delta_j = u_j - \sum_{F \in D_j^v} \mathbb{P}_T(F_1) = u_j - v_j$, which must be positive by monotonicity. We then greedily assign the Δ_j probability mass across all $\{F_j | F \in D_i^v, i \notin \{1, ..., t\}\}$ such that we don't violate (5.3) by assigning too much probability mass to some F_j.

By doing the above, we have satisfied that $u_i = \sum_{D \in D_i^u} \mathbb{P}_T(D)$ for all i. Note that the distribution of every other vertex was maintained just by the inductive hypothesis and (5.3). It is left to argue that we didn't violate (5.3) by greedily assigning the Δ_is and Δ_js. To this end, fix some $\ell \in \{1, ..., t\}$ and consider the distribution $\vec{p_\ell}$ over servers to match to if a request landed on s_ℓ, Consider the same distribution if a request landed on v. Call the distribution vector $\vec{p_v}$. From before, the first t Δs were the increase in probability of matching to the servers on $s_1, \dots, s_t$ (for a request on u) by uncontracting the edge, while the last $k - t$ Δs were the reduction in probability of u matching to some $s_j \notin \{s_1, ..., s_t\}$. Then

$$\vec{p_\ell} = \vec{p_v} + (\Delta_1, ..., \Delta_t, -\Delta_{t+1}, ..., -\Delta_k).$$

Since both $\vec{p_v}$ and $\vec{p_\ell}$ are probability distributions, their L_1 norm is 1. Thus,

$$\sum_{\ell=1}^{t} \Delta_\ell = \sum_{i=t+1}^{k} \Delta_i.$$

Thus, we know that assigning the Δs greedily the way we have will not violate constraint (5.3). ☐

Combining Lemma 5.3 and Theorem 5.2 gives us that every monotone randomized algorithm is priceable.

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.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory OF Computing*, 8:121–164, 2012.

[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.

[SFpa] *SFpark Homepage.*

[SFpb] *SFpark Wikipedia page.*

[SP13] Donald Shoup and Greg Pierce. *SFpark: Pricing Parking by Demand*, 2013.