Competitively Pricing Parking in a Tree

Max Bender[1(✉)], Jacob Gilbert[1], Aditya Krishnan[2], and Kirk Pruhs[1]

[1] University of Pittsburgh, Pittsburgh, PA 15260, USA
{mcb121,jmg264}@pitt.edu, kirk@cs.pitt.edu
[2] Johns Hopkins University, Baltimore, MD 21218, USA
aditya.krishnan94@gmail.com

Abstract. Motivated by demand-responsive parking pricing systems we consider posted-price algorithms for the online metrical matching problem and the online metrical searching problem in a tree metric. Our main result is a poly-log competitive posted-price algorithm for online metrical searching.

1 Introduction

Since 2011 SFpark has been San Francisco's system for managing the availability of on-street parking [2,3,28]. 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, to lower prices in underutilized areas, and to raise 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 demand-responsive parking pricing systems, for example Calgary has had the ParkPlus system since 2008 [1].

The problem of centrally assigning drivers to parking spots to minimize time and fuel usage is naturally modeled by the online metrical matching problem. The setting for online metrical matching consists of a collection of k servers (the parking spots) located at various locations within a metric space. The algorithm then sees an online sequence of requests over time that arrive at various locations in the metric space (the drivers arriving to look for a parking spot). In response to a request, the online algorithm must match the request (car) to some server (parking spot) that has not been previously matched; Conceptually we interpret this matching as the request (car) moving to the location of the matched server (parking spot). The objective goal is to minimize the aggregate distance traveled by the requests (cars).

A. Krishnan—This work was done in part while this author was a student at Carnegie Mellon University advised by Anupam Gupta.

K. Pruhs—Supported in part by NSF grants CCF-1421508, CCF-1535755, CCF-1907673, CCF-2036077 and an IBM Faculty Award.

X. Chen et al. (Eds.): WINE 2020, LNCS 12495, pp. 220–233, 2020.
https://doi.org/10.1007/978-3-030-64946-3_16

We also consider what we call the online metrical search problem, which is an important special case of the online metrical matching problem. This is a promise problem in that the adversary is constrained to guarantee that there is an optimal matching for which only one edge has positive cost. It is useful to conceptually think of online metrical search as the following parking problem: the setting consists of many parking spots at various locations in a metric space and a single car that is initially parked at some location in the metric space. Over time the parking spots are decommissioned one by one until only one parking spot is left in commission. If at any time the car is not parked at an in-commission parking spot, then the car must move to a parking spot that is still in commission. The objective is to minimize the aggregate distance traveled by the car. The optimal solution is to move the car directly to the last remaining parking spot.

The online metrical search problem is a special case of the online metrical matching problem because the parking spots can be viewed as servers and the decommissioning of a parking spot can be simulated by the arrival of a request at the location of that parking spot. So a lower bound on the competitive ratio for the online metrical search problem for a particular metric space also gives a lower bound for the online metrical matching problem on the metric space. Conversely it seems that in terms of the optimal competitive ratio, online metric search is no easier than metric matching. In particular, there is no known example of a metric space where the optimal competitive ratio for online metrical matching is known to be significantly greater than the optimal competitive ratio for online metrical search on that metric space. For example on a line metric, the online metrical search problem is better known as the "cow path problem", and the optimal deterministic competitive ratio is known to be 9 [13], while the best known lower bound on the deterministic competitive ratio for online metrical matching on a line metric is 9.001 [18], worse only by a minuscule factor.

In order to be implementable within the context of SFpark, online algorithms must be posted-price algorithms. In this setting, posted-price means that before each request arrives, the online algorithm sets a price on each unused server (parking spot) without knowing the location where the next request will arrive. Furthermore, each request is assumed to be a selfish agent who moves to the available server (parking spot) that minimizes the sum of the price of and distance to that server. The objective remains to minimize the aggregate distance traveled by the requests. So conceptually the objective of the parking pricing agency is minimizing social cost, not maximizing revenue.

Research into posted-price algorithms for online metrical matching was initiated in [14] as part of a line of research to study the use of posted-price algorithms to minimize social cost in online optimization problems. As a posted-price algorithm is a valid online algorithm, one can not expect to obtain a better competitive ratio for posted-price algorithms than 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 whether a similar competitive ratio can

be (again perhaps approximately) achieved by a posted-price algorithm. The higher level goal is to determine the increase in social cost that is necessitated by the restriction that an algorithm has to use posted prices to incentivize selfish agents, instead of being able to mandate agent behavior.

An $O(\log \Delta)$-competitive randomized posted-price algorithm for metric matching on a line metric is given in [14] where Δ is the ratio of the distance between the furthest two servers and the distance between the closest two servers. No $o(\log k)$-competitive (not necessarily posted-price) algorithm is known for online metric matching on a line metric. So arguably, on a line metric there is a posted-price algorithm that is nearly as competitive as the best known centralized online algorithm.

Our original research goal was to determine whether posted-price algorithms can be similarly competitive with a centralized online algorithm for tree metrics for online metrical matching. In order to be more specific about our goal, we need to review a bit. A tree metric is represented by a tree $T = (V, E)$ with positive real edge weights where the distance $d_T(u, v)$ between vertices $u, v \in V$ is the shortest path between vertices u and v in T. There is a deterministic online algorithm that is $(2k - 1)$-competitive for online metric matching in any metric space, and no deterministic online algorithm can achieve a better competitive ratio for online metric searching in a tree metric [21,22]. An $O(\log k)$-competitive randomized algorithm for online metric matching in $O(\log k)$-HST's (Hierarchically Separated Trees) is given in [25]. By combining this result with results about randomly embedding metric spaces into HST's [10,11,16,25] obtained an $O(\log^3 k)$-competitive randomized algorithm for online metric matching in a general metric space. Following this general approach, [9] later obtained an $O(\log^2 k)$-competitive randomized algorithm for online metrical search in an arbitrary metric by giving an $O(\log k)$-competitive randomized algorithm for 2-HST's. No better results are known for tree metrics, so all evidence points to tree metrics as being as hard as general metrics for online metrical matching. Thus, more specifically our original research goal was to determine whether there is poly-log competitive randomized posted-price algorithm for the online metrical matching problem on a tree metric. Before stating our progress toward this goal, it will be useful to review the literature a bit more.

1.1 Prior Related Work

The most obvious algorithmic design approach for posted-price problems is to directly design a pricing algorithm from scratch, as is done for metrical task systems in [14], but this is not the most common approach in the literature. Two less direct algorithmic design paradigms have emerged in the literature. The first algorithmic design paradigm is what we will call *mimicry*. A posted-price algorithm A *mimics* an online algorithm B if the probability that B will take a particular action is equal the probability that a self-interested agent will choose this same action when the prices of actions are set using A. For example, [14] shows how to set prices to mimic the $O(\log \Delta)$-competitive Harmonic algorithm for online metric matching on a line metric from [19]. As another example, [17]

shows how to set prices to mimic the $O(1)$-competitive algorithm Slow-Fit from [7,8] for the problem of minimizing makespan on related machines. However, for some problems it is not possible to mimic known competitive algorithms using posted prices. For such problems, another algorithmic design paradigm is what we will call *monotonization*. In the monotonization algorithm design approach, one first seeks to characterize the online algorithms that can be mimicked, and then designs such an online algorithm. In the examples in the literature, this characterization involves some sort of monotonicity property. For example, monotonization is used in [14] to obtain an $O(k)$-competitive posted-price algorithm for the k-server problem on a line metric, and in [15] to obtain an $O(k)$-competitive posted-price algorithm for the k-server problem on a tree metric. Since no deterministic algorithm can be better than k-competitive for the k-server problem in any metric [24], this shows that in these settings, there is minimal increase in social cost necessitated by the use of posted-prices. As another example, monotonization is used in [20] to obtain an $O(1)$-competitive posted-price algorithm for minimizing maximum flow time on related machines.

For online metric matching on a line metric, better competitive ratios are achievable. An $O(k^{.59})$-competitive deterministic online algorithm was given in [4]. Subsequently several different $O(\log n)$-competitive randomized online algorithms for a line are given in [19]; these algorithms leverage special properties of HST's constructed from a line metric. As already mentioned, [19] also showed that the natural Harmonic algorithm is $O(\log \Delta)$-competitive. An $O(\log^2 k)$-competitive deterministic online algorithm was given in [26], and this was later improved to $O(\log k)$ in [27]. Super-constant lower bounds for various types of algorithms are given in [5,23]. More generally, the algorithm for online metric matching given in [26] has the property that for every metric space, its competitive ratio is at most $O(\log^2 k)$ times the optimal competitive ratio achievable by any deterministic algorithm on that metric space.

1.2 Our Contribution

There is no hope to mimic any of the online algorithms for online metrical matching that are based on HST's as HST's by their very nature lose too much information about the structure of a tree metric. Therefore we adopt the monotonization approach. In Sect. 2 we identify a monotonicity property that characterizes mimicable algorithms for online metrical matching in tree metrics. Roughly speaking this property says that if a request were to have arrived on the route to its desired server, then the probability that the request would still have been matched to this server can not decrease. Thus we reduce finding a post-priced algorithm to finding a monotone algorithm.

In Sect. 3 we give an algorithm `TreeSearch` for the online metrical search problem on a tree metric. The algorithm is based on the classic multiplicative weights algorithm for online learning from experts [6]. Conceptually there is one expert E^ℓ for each leaf ℓ of the tree T. Expert E^ℓ always recommends that the car/request travels toward the leaf ℓ. Thus expert E^ℓ pays a cost of one whenever a parking spot on the path from the root to ℓ is decommissioned, a

cost of zero when other parking spots are decommissioned, and an infinite cost if there are no remaining parking spots on the path from the root to ℓ. Let π_t^ℓ be the probability that the multiplicative weights algorithm has associated with expert E^ℓ right before request r_t arrives. Let v_t^ℓ be the location of the car just before request r_t arrives if the advice of expert E^ℓ had always been followed. The algorithm `TreeSearch` maintains the invariant that right before request r_t arrives, the probability that the car is at a vertex v is $\sum_{\ell:v_t^\ell=v} \pi_t^\ell$, the sum of the probabilities of the experts that recommend that the car should be parked at v. The most technically difficult part of the algorithm design process was maintaining this invariant. We then upper bound the expected number of jumps made by the `TreeSearch` algorithm, where a jump is a movement of the car by a positive amount. Finally, we show how to extend `TreeSearch` to be a monotone algorithm `TreeMatch` for online metrical matching on a tree metric.

In Section algorithm for online metric searching on a tree metric. Before any requests arrive, an algorithm `GroveBuild` embeds the tree metric into what we will call a grove, which is a refinement of an HST that retains more information about the topology of the original metric space. It is probably easiest to explain what a grove is by explaining the difference in how one is constructed in comparison to how an HST is constructed. The construction of each starts with a Low Diameter Decomposition (LDD) of the metric space. A LDD is a partition $\mathcal{P} = \{P_1, \ldots, P_n\}$ of the vertices of the metric space where each part is connected and the diameter of each part is an α factor smaller than the diameter of the whole metric space. The top of the HST consists of a star where the center of the star is the root of the HST, and there is one child of the root for each part P_i. In contrast, the top of a grove consists of the tree that remains after collapsing each part to a single vertex. For both an HST and a grove, the construction then proceeds recursively on each part. So intuitively the key difference is that groves retain information about the distances between parts in the LDD that the HST instead discards. See Fig. 1 for a comparison of an HST and a grove constructed from the same LDD.

We then give a monotone algorithm `GroveMatch` for online metrical matching on a tree metric that utilizes the algorithm `TreeMatch` on each tree in the grove constructed from the tree metric. We show that `GroveMatch` is poly-log competitive (more precisely $O(\log^6 \Delta \log^2 n)$-competitive) on metric search instances by induction on the levels of the grove. This is an extension of a similar induction argument in [25] that shows that a $O(\log n)$-competitive algorithm for a star (or a complete unit metric) can be extended to an algorithm for a $O(\log n)$-HST with the loss of a poly-log factor in the competitiveness. However, our situation is complicated by the fact the possible ways that a request can potentially move within a grove is more complicated than the possible ways a request can move within an HST, and thus the induction is more complicated as the induction depends on when the request is moving "up" and when the request is moving "down" in trees within the grove. The bound on the number of jumps made by `TreeSearch` translates to a bound on the number of recursive calls made by `GroveMatch`. There is not a lot of wiggle room in our analysis, and thus both the

algorithm design and algorithm analysis process are necessarily quite delicate. For example, if `TreeSearch` made just 1% more jumps than the bound that we can show, then the resulting competitiveness of `GroveMatch` would not be poly-logarithmic. One consequence of this delicateness is that we can not use a black box LDD construction to build our grove, we need to construct our LDD in a way that tightly controls the variance of random properties of our grove.

Due to space requirements, proofs have mostly been removed. See [12] for the full paper with complete proofs.

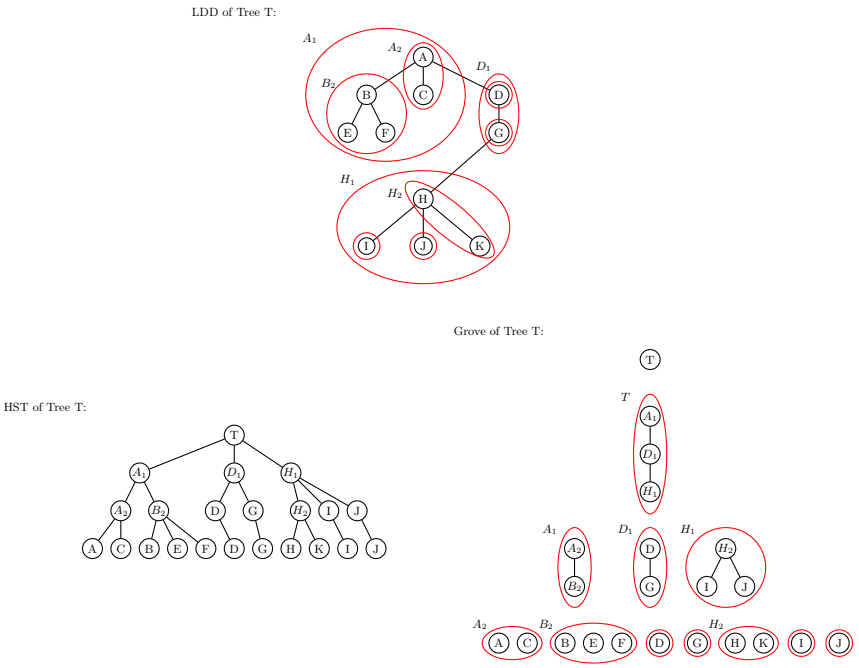

Fig. 1. An example of a LDD, the corresponding HST, and the corresponding grove.

2 Pricing Monotone Algorithms

In this section, we show that an algorithm for the online metrical matching can be implemented as a posted-price algorithm if and only if the algorithm satisfies the following monotonicity property. We note that monotonicity does not have a natural interpretation within the context of online metrical searching, which explains why we give a monotone algorithm for online metrical matching, even though we only analyze its competitiveness for online metrical search.

Definition 1. *An algorithm A for online metric matching is monotone if for every instance, every request r_t in that instance, every possible sequence R of*

random events internal to A prior to r_t's arrival, and all vertices u, v, s where v is on the path from u to s it is the case that: $\boldsymbol{Pr}[A_R(r_t) = s \mid E_R \text{ and } r_t = u] \leq \boldsymbol{Pr}[A_R(r_t) = s \mid E_R \text{ and } r_t = v]$ *where $A_R(r_t)$ is the event that A matches r_t to s, and E_R is the event that the past random events internal to A are equal to R.*

Theorem* 1. *Any algorithm A for the online metrical matching problem can be implemented as a posted-price algorithm if and only if A is monotone.*

3 The Algorithm `TreeMatch`

In Subsect. 3.1 we define algorithm `TreeSearch` for the metric search problem on a tree $T = (V, E)$ rooted at vertex a ρ. The distance metric on T will not be of interest to us in this section. We will use the interpretation of a car moving when its parking spot is decommissioned, as introduced earlier, as we think that this interpretation is more intuitive. The description of `TreeSearch` in Subsect. 3.1 uses a probability distribution $q_t^{\sigma}(\tau)$ that is complicated to define, so its exact definition is postponed until Subsect. 3.2, in which we also show that it achieves our goal of matching the experts distribution. Finally in Subsect. 3.3, we show how to convert `TreeSearch` into a monotone algorithm `TreeMatch` for online metrical matching that is identical to `TreeSearch` on online metrical search instances.

3.1 Algorithm Description

We start with some needed definitions and notation.

Definition 2. *A parking spot s_i in the collection S of parking spots is a leaf-spot if there are no other parking spots in the subtree rooted at s_i. Let $L(T) = \{\ell_1, ..., \ell_d\}$ denote the collection of leaf-spots. Let H be the maximum initial number of parking spots in T on the path from the root ρ to a leaf-spot in $L(T)$. For $\sigma \in [d]$, define $T_\sigma \subseteq V$ as the set of parking spots on the path from the root ρ to ℓ_σ, inclusive. We define T_σ to be alive if there is still an in-commission parking spot in T_σ, and dead otherwise. A T_σ is killed by r_t if r_t is the last parking spot to be decommissioned in T_σ. Let $\mathcal{A}_t = \{\sigma \in [d] \mid T_\sigma$ is alive just before the arrival of $r_t\}$. For a vertex $v \in V$, let $L(v)$ denote the collection of leaf-spots that are descendants of v in T. Let c_t be the location of the car just before the arrival of request r_t.*

Algorithm `TreeSearch`: The algorithm has two phases: the prologue phase and the core phase. The algorithm starts in the prologue phase and transitions to the core phase after the first time m when there is no available parking space on the path from the new parking spot c_{m+1} to the root ρ, inclusive. The algorithm then remains in the core phase until the end. In the prologue phase, whenever the car is not parked at a vertex with an in-commission parking spot, the following actions are taken:

1. If there is an in-commission parking spot at c_t then no action is taken.
2. Else if there is an in-commission parking spot on the path between c_t and the root ρ, inclusive, then the car moves to the first in-commission parking spot on this path nearest to c_t.
3. Else the car moves to the root ρ and enters the core phase to determine where to go from there. So for analysis purposes, the movement to ρ counts as being part of the prologue phase, and the rest of the movement counts as being in the core phase.

If the car is at the root ρ and the algorithm is just transitioning into the core phase, then a live T_τ is picked uniformly at random from $\mathcal{A}_{t+1}$, an internal variable γ is set to be τ, and the car moves to the first in-commission parking spot on the path from ρ to ℓ_τ. Subsequently in the core phase, when a parking spot r_t is decommissioned then:

1. If the car is not parked at r_t, that is if $c_t \neq r_t$, then no action is taken.
2. Else the car moves to the first in-commission parking spot in T_τ with probability $q_t^\gamma(\tau)$ and sets γ to be τ. ($q_t^\gamma(\tau)$ is defined in the next subsection.)

Intuitively γ stores the last random choice of the algorithm.

3.2 The Definition of $q_t^\sigma(\tau)$

In this section we only consider times in the core phase. We conceptually divide up the tree T into three regions. Given vertex v and time t, we let z_t^v be the number of in-commission parking spots on the path from v to ρ, inclusive, just before decommission r_t. We then define the regions as follows:

1. The *root region* is the set of all vertices v such that $z_t^v = 0$. Note that this region is connected, and no decommissioning can occur in this region since there are no parking spots left.
2. The *frontier region* is the set of all vertices v such that $z_t^v = 1$. A decommissioning r_t is called a frontier decommissioning if r_t is in the frontier region.
3. The *outer region* is the set of all vertices v such that $z_t^v > 1$. A decommissioning r_t is called a outer decommissioning if r_t is in the outer region.

Observe that these regions have no dependence on random events internal to the algorithm. Further observe that step 2 of the core phase in algorithm `TreeSearch` maintains the invariant that the car is always parked at a spot in the frontier region. This means that any outer decommissionings will not move the car from its current parking spot.

Definition 3. *Let r_m be the last decommissioning handled in the prologue phase of `TreeSearch`. Define $\mathcal{X}_t = \mathcal{A}_t \cap L(r_t)$ to be the collection of σ's such that T_σ is alive and contains r_t and define $\mathcal{Y}_t = \mathcal{A}_t \setminus \mathcal{X}_t = \mathcal{A}_t \cap \overline{L(r_t)}$ to be the collection of σ's such that T_σ is alive and doesn't contain r_t. Define $\mathcal{F}_t = \mathcal{X}_t \cap \overline{\mathcal{A}_{t+1}}$ to be the collection of σ's such that T_σ is killed by r_t. Let n_t^σ denote the number of frontier decommissionings strictly before time t from T_σ. Define $w_t^\sigma = (1-\epsilon)^{n_t^\sigma}$ for each*

$\sigma \in [d]$. Define $W_t(\mathcal{J}) = \sum_{\sigma \in \mathcal{J}} w_t^\sigma$ for any $\mathcal{J} \subseteq \{1, ..., d\}$. Define π_t^σ as the probability the experts algorithm would give to expert σ, that is $\pi_t^\sigma = \frac{w_t^\sigma}{\sum_{\tau \in [d]} w_t^\tau}$. Define $\tilde{\pi}_t^\sigma$ as π_t normalized amongst all experts in $\mathcal{A}_t$, that is $\tilde{\pi}_t^\sigma = \frac{w_t^\sigma}{\sum_{\tau \in \mathcal{A}_t} w_t^\tau}$ if $\sigma \in \mathcal{A}_t$, and 0 otherwise. Define p_t^σ as the probability that $\gamma = \sigma$ right before time t.

We are now ready to define $q_t^\sigma(\tau)$. Note that by the definition of `TreeSearch`, $q_t^\sigma(\tau)$ is only used for $\sigma \in \mathcal{X}_t$ since the algorithm only reaches step 2 of the core phase when $r_t \in T_\gamma$. We show in Lemma 1 that this definition of $q_t^\sigma(\tau)$ indeed defines a probability distribution over $\tau \in [d]$. We then show in Lemma 2 that the definition of $q_t^\sigma(\tau)$ guarantees that our desired invariant $p_t^\sigma = \tilde{\pi}_t^\sigma$ holds.

Definition 4

$$q_t^\sigma(\tau) = \begin{cases} \frac{\epsilon w_t^\tau}{(1-\epsilon)W_t(\mathcal{X}_t \setminus \mathcal{F}_t) + W_t(\mathcal{Y}_t)} & \text{if } \tau \in \mathcal{Y}_t \text{ and } \sigma \in \mathcal{X}_t \setminus \mathcal{F}_t \\ \frac{w_t^\tau}{(1-\epsilon)W_t(\mathcal{X}_t \setminus \mathcal{F}_t) + W_t(\mathcal{Y}_t)} & \text{if } \tau \in \mathcal{Y}_t \text{ and } \sigma \in \mathcal{F}_t \\ \frac{1 - \sum_{\varsigma \in \mathcal{Y}_t} q_t^\sigma(\varsigma)}{|\mathcal{X}_t \setminus \mathcal{F}_t|} & \text{if } \tau \in \mathcal{X}_t \setminus \mathcal{F}_t \\ 0 & \text{if } \tau \in \mathcal{F}_t \text{ or } \tau \in \overline{\mathcal{A}_t} \end{cases}$$

Lemma* 1. *For all times t in the core phase and for all $\sigma \in \mathcal{X}_t$, $q_t^\sigma(\tau)$ forms a distribution over $\tau \in [d]$.*

Lemma* 2. *For all times t during the core phase and for all $\sigma \in [d]$, $p_t^\sigma = \tilde{\pi}_t^\sigma$.*

Definition 4 and Lemmas 1 and 2 give us the following bound on the cost:

Theorem* 2. *During the prologue phase, $\sum_{t=1}^{m} \mathbf{1}^{\mathsf{TM}}(t) \leq H$ and during the core phase, $\boldsymbol{E}\left[\sum_{t=m+1}^{k-1} \mathbf{1}^{\mathsf{TM}}(t)\right] \leq (1+\epsilon)H + \frac{\ln d}{\epsilon}$ where $\mathbf{1}^{\mathsf{TM}}(t)$ is an indicator random variable that is 1 if `TreeSearch` moves the car to a new parking spot on the decommissioning r_t and 0 otherwise.*

3.3 Monotonicity

We show that any neighbor algorithm for online metrical search can be extended to a monotone algorithm for online metrical matching, where a neighbor algorithm has the property that if it moves the car to a parking spot s_i with positive probability then it must be the case that there is no in-commission parking spot on the route to s_i. As `TreeSearch` is obviously a neighbor algorithm, it then follows that it can be extended to a monotone algorithm for online metrical matching, which we will call `TreeMatch`.

Lemma* 3. *Let A be a neighbor algorithm for online metrical search. Then there exists a monotone algorithm B for online metrical matching on a tree metric that is identical to A for online metrical search instances.*

4 The GroveMatch Algorithm

In Subsect. 4.1 we describe an algorithm GroveBuild that builds a grove G from a tree metric T with distance metric d_T before any request arrives. We assume without loss of generality that the minimum distance in T is 1. In Subsect. 4.2 we then give an algorithm GroveMatch for online metrical matching on a tree metric that utilizes the algorithm TreeMatch on each tree in the grove constructed by GroveBuild, and we prove some basic properties of the grove G. In Subsect. 4.3 we show that GroveMatch is a monotone online metrical matching algorithm on a tree metric, and is $O(\log^6 \Delta \log^2 n)$-competitive for online metrical search instances.

4.1 The GroveBuild Algorithm

Definition 5. *A grove G is either: a rooted tree X consisting of a single vertex, or an unweighted rooted tree X with a grove $X(v)$ associated with each vertex $v \in X$. The tree X is the canopy of the grove G. Each $X(v)$ is a subgrove of X. The canopy of a subtree $X(v)$ is a child of X. Trees in G are descendants of X.*

GroveBuild Description: GroveBuild is a recursive algorithm that takes as input a tree metric T, a designated root ρ of T, positive real R, a positive real α and a positive integer d. In the initial call to GroveBuild, T is the original tree metric, ρ is an arbitrary vertex in T, R is the maximum distance Δ between ρ and any other vertex in T, d is 1, and α is a parameter to be determined later in the analysis.

If T consists of a single vertex v, then the recursion ends and the algorithm outputs a rooted tree consisting of only the vertex v. We call this tree a leaf of the grove. Otherwise the algorithm's first goal is to partition the vertices of T into parts $P_1, \ldots, P_k$, and designate one vertex ℓ_i of each partition P_i as being the leader of P_i. To accomplish this, the algorithm sets partition P_1 to consist of the vertices in T that are within a distance z of ρ, where z is selected uniformly at random from the range $[0, \frac{R}{\alpha}]$. The leader ℓ_1 is set to be ρ. To compute P_i and ℓ_i after the first $i-1$ parts and leaders are computed the algorithm takes the following steps. Let ℓ_i be a vertex such that $\ell_i \notin \cup_{j=1}^{i-1} P_j$ and for each vertex v on the path (ℓ_i, ρ) it is the case that $v \in \cup_{j=1}^{i-1} P_j$. So ℓ_i is not in but adjacent to the previous partitions. Then P_i consists of all vertices $v \in T - \cup_{j=1}^{i-1} P_j$ that are within distance $\frac{R}{\alpha}$ from ℓ_i in T. So P_i intuitively is composed of vertices that are not in previous partitions and that are close to ℓ_i.

The tree X at this point in the recursion has a vertex for each part in the partition of T. There is an edge between vertices/parts P_i and P_j in X if and only if there is an edge (v, w) in T such that $v \in P_i$ and $w \in P_j$. We identify this edge in X with the edge $(v, w) \in T$. The root of X is the vertex/part P_1. The tree X is at depth d in the grove. The grove $X(P_i)$ associated with vertex P_i in X is the result of calling GroveBuild on the subtree of T induced by the

vertices in P_i, with ℓ_i designated as the root, parameter R decreased by an α factor, parameter α unchanged, and parameter d incremented by 1.

So from here on, let G denote the grove built by `GroveBuild` on the original tree metric T.

Definition 6

- *For an edge $(u, v) \in T$, let $\delta(u, v)$ be the depth in the grove G of the tree X that contains (u, v). Note that each edge in T occurs in exactly one tree in G.*
- *For an edge $(u, v) \in T$, define $d_G(u, v)$ to be $\frac{\Delta}{\alpha^{\delta(u,v)-1}}$.*
- *For vertices $u_0, u_h \in T$, connected by the simple path $(u_0, u_1, \ldots, u_h)$ in T, define $d_G(u_0, u_h)$ to be $\sum_{i=0}^{h-1} d_G(u_i, u_{i+1})$. Obviously d_G forms a metric on the vertices of T.*

Lemma* 4. *Recall that $d_T(u, v)$ is the shortest path distance between two vertices u, v of tree T. For all vertices $u, v \in T$, we have that $d_G(u, v) \geq d_T(u, v)$ and $\boldsymbol{E}[d_G(u, v)] \leq \alpha(1 + \log \Delta) \cdot d_T(u, v)$.*

Corollary* 1. *An algorithm $\boldsymbol{B}$ that is c-competitive for online metric matching on T with distance metric d_G is $O(c \cdot \alpha \log \Delta)$-competitive for online metric matching on T with distance metric d_T.*

4.2 `GroveMatch` Description

We now describe an algorithm `GroveMatch` for online metrical matching for tree metrics.

`GroveMatch` Description: Conceptually within `GroveMatch`, a separate copy `TreeMatch`(X) of the online metric matching algorithm `TreeMatch` will be run on each tree X in the grove G constructed by the algorithm `GroveBuild`. In order to accomplish this, we need to initially place servers at the vertices in X. We set the number of servers initially located at each vertex $x \in X$ to the number of servers in T that are located at vertices $v \in T$ such that $v \in x$ (recall that each vertex in a tree in the grove G corresponds to a collection of vertices in T).

When a request r_t arrives at a vertex v in T, the algorithm `GroveMatch` calls the algorithm `TreeMatch` on a sequence $(X_1, x_1), (X_2, x_2), \ldots$ where each X_i is a tree of depth i in G and x_i is a vertex in X_i. Initially X_1 is the depth 1 tree in G, and x_1 is the vertex in X_1 that contains v. Assume that `TreeMatch` has already been called on $(X_1, x_1), (X_2, x_2), \ldots (X_{i-1}, x_{i-1})$, then the algorithm `GroveMatch` processes (X_i, x_i) in the following manner. First, `TreeMatch`(X_i) is called to respond to a request at x_i. Let y_i be the vertex in X_i that `TreeMatch`(X_i) moved this request to. If X_i is a leaf in G, then `TreeMatch`(X_i) sets $y_i = x_i$, and `GroveMatch` moves request r_t to the unique vertex in T corresponding to x_i. If X_i is not a leaf in G, then X_{i+1} is set to be the canopy of the grove $X_i(y_i)$, and $x_{i+1} = \arg\min_{w \in T : w \in X_{i+1}} d_T(v, w)$ or equivalently x_{i+1} is the first vertex in X_{i+1} that one encounters if one walks in T from v to the vertices of X_{i+1}.

Lemma* 5. *Consider a tree X at depth δ with root ρ in grove G. For any vertex v in X, the number of hops in X between ρ and v is at most $\alpha+1$. Furthermore, by the time that* `TreeMatch`(X) *enters its core phase, it must be the case that for every descendent tree Y of X in G there will be no future movement of the car on edges in Y while* `TreeMatch`(Y) *is in its prologue phase.*

4.3 `GroveMatch` Analysis

We now analyze `GroveBuild` and `GroveMatch` under the assumption that $\alpha = (\ln n)(\log_\alpha^2 \Delta)$ and $\epsilon = \frac{1}{\log_\alpha \Delta}$.

Lemma 6. *The algorithm* `GroveMatch` *is $O(\log n \log^3 \Delta)$-competitive for online metrical search instances with the metric d_G.*

Proof. If `GroveMatch` directs a request to traverse an edge $(u, v) \in T$, we will say that the cost of this traversal is charged to the unique tree in G that contains (u, v). Define $P(\delta)$ to be the charge incurred by a tree X of depth δ in G and all subgroves $X(v)$ of X during the prologue phase of `TreeMatch`(X). Define $C(\delta)$ to be the charge incurred by a tree X of depth δ in G and all subgroves $X(v)$ of X during the core phase of `TreeMatch`(X).

Recall that the distance under the d_G metric of ever edge in X is $\frac{\Delta}{\alpha^{\delta-1}}$ and by Lemma 5 there are at most $\alpha + 1$ vertices on the path from any leaf to the root of X. This gives us that the distance in X under d_G from the root to any leaf is at most $\alpha\frac{\Delta}{\alpha^{\delta-1}} = \frac{\Delta}{\alpha^{\delta-2}}$ and that the diameter of X is at most $2\frac{\Delta}{\alpha^{\delta-2}}$. The only subgroves $X(v)$ of X that incur costs during the prologue phase of `TreeMatch`(X) are those subgroves for which v is traversed by the car on its path to the root of X. Thus we obtain the following recurrence:

$$P(\delta) \leq (\alpha + 1)\left(P(\delta + 1) + C(\delta + 1)\right) + \frac{\Delta}{\alpha^{\delta-2}}. \tag{1}$$

Note that once the core phase begins in `TreeMatch`(X), by Lemma 5 all instances of `TreeMatch`(Y) on any tree Y that is a descendent of X in G can incur no most costs in their prologue phase. By Theorem 2 the core phase cost on X is at most $(1 + \epsilon)(\alpha + 1) + \frac{\ln n}{\epsilon}$ times the diameter of X, which is at most $2\frac{\Delta}{\alpha^{\delta-2}}$. Thus we obtain the following recurrence:

$$C(\delta) \leq \left(C(\delta + 1) + 2\frac{\Delta}{\alpha^{\delta-2}}\right)\left((1 + \epsilon)(\alpha + 1) + \frac{\ln n}{\epsilon}\right) \tag{2}$$

We expand the recurrence relation for $C(\delta)$ first. Treating $((1 + \epsilon)(\alpha + 1) + \frac{\ln(n)}{\epsilon})$ as a constant Z, and expanding $C(\delta)$ we obtain:

$$C(\delta) \leq \left(C(\delta + 1) + 2\frac{\Delta}{\alpha^{\delta-2}}\right) Z \leq \frac{2\Delta \log_\alpha \Delta}{\alpha^{\delta-1}} \left(\frac{Z}{\alpha}\right)^{\log_\alpha(\Delta)} \leq \frac{2e^4 \Delta \log_\alpha \Delta}{\alpha^{\delta-1}}$$

Now expanding the recurrence relation for $P(\delta)$ we obtain:

$$P(\delta) \leq (\alpha+1)\left(P(\delta+1)+C(\delta+1)\right)+\frac{\Delta}{\alpha^{\delta-2}} \leq \frac{3e^5\Delta\log_\alpha^2\Delta}{\alpha^{\delta-2}}$$

Hence the cost of the algorithm `GroveMatch` is $O\left(\frac{\Delta}{\alpha^{\delta-2}}\log^2\Delta\right)$. However, note that `TreeMatch` only pays positive cost on X if for any optimal solution there is at least one request that such a solution must pay positive cost for in X. The reason for this is that if `TreeMatch`(X) moves the car out of a vertex v in X, then there are no in-commission parking spots left in v, and therefore every algorithm would have to move the car out of v. Since every edge in X has distance $\frac{\Delta}{\alpha^{\delta-1}}$, this gives us that `GroveMatch` must be $O(\alpha\log^2\Delta) = O(\log n\log^3\Delta)$ competitive on the metric d_G.

Together with Corollary 1, Lemma 6 gives us the following theorem:

Theorem 3. *`GroveMatch` is $O(\log^6\Delta\log^2 n)$-competitive for online metrical search instances.*

Lemma* 7. *`GroveMatch` is a monotone algorithm for online metrical matching.*

Acknowledgements. We thank Anupam Gupta for his guidance, throughout the research process, that was absolutely critical to obtaining these results. We thank Amos Fiat for introducing us to this posted-price research area, and for several helpful discussions.

References

1. Calgar ParkPlus Homepage. https://www.calgaryparking.com/parkplus
2. SFpark Homepage. http://sfpark.org/
3. SFpark Wikipedia page. https://en.wikipedia.org/wiki/SFpark
4. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: A $o(n)$-competitive deterministic algorithm for online matching on a line. In: Workshop on Approximation and Online Algorithms, pp. 11–22 (2014)
5. Antoniadis, A., Fischer, C., Tönnis, A.: A collection of lower bounds for online matching on the line. In: Bender, M.A., Farach-Colton, M., Mosteiro, M.A. (eds.) LATIN 2018. LNCS, vol. 10807, pp. 52–65. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77404-6_5
6. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta-algorithm and applications. Theory Comput. **8**, 121–164 (2012)
7. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)
8. Azar, Y., Kalyanasundaram, B., Plotkin, S.A., Pruhs, K., Waarts, O.: On-line load balancing of temporary tasks. J. Algorithms **22**(1), 93–110 (1997)
9. Bansal, N., Buchbinder, N., Gupta, A., Naor, J.: A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. Algorithmica **68**(2), 390–403 (2014)

10. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: Symposium on Foundations of Computer Science, pp. 184–193 (1996)
11. Bartal, Y.: On approximating arbitrary metrics by tree metrics. In: ACM Symposium on Theory of Computing, pp. 161–168 (1998)
12. Bender, M.E., Gilbert, J., Krishnan, A., Pruhs, K.: Competitively pricing parking in a tree. arXiv, abs/2007.07294 (2020)
13. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
14. Cohen, I.R., Eden, A., Fiat, A., Jez, L.: Pricing online decisions: beyond auctions. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 73–91 (2015)
15. Cohen, I.R., Eden, A., Fiat, A., Jez, L.: Dynamic pricing of servers on trees. In: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. LIPIcs, vol. 145, pp. 10:1–10:22 (2019)
16. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**(3), 485–497 (2004)
17. Feldman, M., Fiat, A., Roytman, A.: Makespan minimization via posted prices. In: ACM Conference on Economics and Computation, pp. 405–422 (2017)
18. Fuchs, B., Hochstättler, W., Kern, W.: Online matching on a line. Theor. Comput. Sci. **332**(1–3), 251–264 (2005)
19. Gupta, A., Lewi, K.: The online metric matching problem for doubling metrics. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7391, pp. 424–435. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31594-7_36
20. Im, S., Moseley, B., Pruhs, K., Stein, C.: Minimizing maximum flow time on related machines via dynamic posted pricing. In: European Symposium on Algorithms, pp. 51:1–51:10 (2017)
21. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
22. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theor. Comput. Sci. **127**(2), 255–267 (1994)
23. Koutsoupias, E., Nanavati, A.: The online matching problem on a line. In: Solis-Oba, R., Jansen, K. (eds.) WAOA 2003. LNCS, vol. 2909, pp. 179–191. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24592-6_14
24. Manasse, M., McGeoch, L., Sleator, D.: Competitive algorithms for on-line problems. In: ACM Symposium on Theory of Computing, pp. 322–333 (1988)
25. Meyerson, A., Nanavati, A., Poplawski, L.J.: Randomized online algorithms for minimum metric bipartite matching. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 954–959 (2006)
26. Nayyar, K., Raghvendra, S.: An input sensitive online algorithm for the metric bipartite matching problem. In: Symposium on Foundations of Computer Science, pp. 505–515 (2017)
27. Raghvendra, S.: Optimal analysis of an online algorithm for the bipartite matching problem on a line. In: Symposium on Computational Geometry. LIPIcs, vol. 99, pp. 67:1–67:14 (2018)
28. Shoup, D., Pierce, G.: SFpark: Pricing Parking by Demand (2013). https://www.accessmagazine.org/fall-2013/sfpark-pricing-parking-demand/