Dynamic Pricing of Servers on Trees

Ilan Reuven Cohen
TU Eindhoven and CWI, Netherlands
ilanrcohen@gmail.com

Alon Eden
Tel Aviv University
alonarden@gmail.com

Amos Fiat
Tel Aviv University
fiat@tau.ac.il

Łukasz Jeż
University of Wrocław
lje@cs.uni.wroc.pl

Abstract

In this paper we consider the k-server problem where events are generated by selfish agents, known as *the selfish k-server problem*. In this setting, there is a set of k servers located in some metric space. Selfish agents arrive in an online fashion, each has a request located on some point in the metric space, and seeks to serve his request with the server of minimum distance to the request. If agents choose to serve their request with the nearest server, this mimics the greedy algorithm which has an unbounded competitive ratio. We propose an algorithm that associates a surcharge with each server independently of the agent to arrive (and therefore, yields a truthful online mechanism). An agent chooses to serve his request with the server that minimizes the distance to the request *plus* the associated surcharge to the server.

This paper extends [9], which gave an optimal k-competitive dynamic pricing scheme for the selfish k-server problem on the line. We give a k-competitive dynamic pricing algorithm for the selfish k-server problem on tree metric spaces, which matches the optimal online (non truthful) algorithm. We show that an α-competitive dynamic pricing scheme exists on the tree if and only if there exists α-competitive online algorithm on the tree that is *lazy*, *local*, and *monotone*. Given this characterization, the main technical difficulty is coming up with such an online algorithm.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Algorithmic mechanism design

Keywords and phrases Online algorithms, Online mechanisms, k-server problem, Online pricing

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2019.10

Funding *Ilan Reuven Cohen*: Partially supported by the ERC consolidator grant 617951
Alon Eden: Partially supported by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 337122, by the Israel Science Foundation (grant numbers 317/17 and 1841/14)
Amos Fiat: Partially supported by the Israel Science Foundation (grant number 1841/14)
Łukasz Jeż: Partially supported by Polish National Science Centre grant 2016/22/E/ST6/00499

1 Introduction

Online algorithms were designed to deal with cases where the input arrives piecemeal over time and consists of a sequence of events. Problems such as paging, online matching, online scheduling, etc., are all examples of such problems.

This paper, belongs to a thread of recent research where events are selfish and the goal is to set surcharges on the various decisions that can be made by the agent with some desirable goal in mind such as minimizing social cost, makespan, completion time, flow time, sum of completion times, etc. (See Section 1.1 for some examples.) The prices may change over time, but must be known to the selfish agent upon arrival so that the agent can make an informed decision. Truthfulness is immediate in such settings, the agent gets asked no questions and therefore cannot lie about anything. The agent simply takes the utility maximizing (disutility minimizing) option available.

Specifically, in the dynamic pricing scheme for the k-server problem that we consider, the mechanism sets a surcharge on each server *prior* to an arrival of the next request. The agent that issues the request greedily chooses the server which minimizes the distance between the server and request *plus* the surcharge for the server. Note that the mechanism may update the surcharge of the servers based on *past* requests.

This paper extends the dynamic pricing results obtained for the k-server problem in [9] and deals with servers on a tree rather than restricted to a line. Although the basic idea is the same: use dynamic pricing to "nudge" selfish agents to act as though they were under the control of a centralized online algorithm, the tree metric is much more challenging to deal with than the line.

We show that any α-competitive online algorithm on the tree that is simultaneously *(i) lazy*: moves at most one server, *(ii) local*: a request at a point occupied by one or more servers is served by one of these servers, and *(iii) monotone*: the set of points serviced by a server is contiguous, can be converted into a dynamic posted pricing scheme for the selfish k-server problem on the tree with a competitive ratio of α. These properties were defined and in fact proved for the line [9], but they extend naturally to trees; cf. Section 2.2 for formal definitions. Thus, the main challenge in this paper is to give a k-competitive k-server algorithm for the tree that is lazy, local, and monotone.

In the work of Cohen et al. [9], the main idea for obtaining an algorithm with those properties on a line is to run a simulation of the Double Cover (DC) algorithm and serve each request (at point) r with a server that is adjacent to r (i.e., there are no intermediate servers on its path to r) and that can be matched to a simulated Double Cover server which serves r in a min cost matching. This maintains the competitive ratio and ensures laziness, locality and monotonicity. Generalizing this idea to trees is not immediate. In particular, choosing an arbitrary server adjacent to the request which can also be matched to a simulated server in a min cost matching results in non-monotonicity, which cannot be priced. This means that one needs a deeper understanding of the tree topology in deciding which of the servers is to serve the request (We explain this in detail in Section 2.2).

1.1 Related Work

1.1.1 Dynamic Pricing Schemes and Online Mechanisms

Lavi and Nisan [18] initiated the study of competitive analysis of incentive compatible online auctions. In particular, they give an incentive compatible on-line auction for many identical items with a tight competitive ratio. They consider both revenue and social welfare targets.

Awerbuch, Azar, and Myerson [1] give a general scheme that produces posted prices for general combinatorial auctions, with a competitive ratio equal to the logarithm of the ratio between highest and lowest prices, times the underlying competitive ratio for the combinatorial auction.

Although not explicitly stated as a pricing scheme, [14] effectively gives a dynamic pricing scheme for 2 servers in any metric space. Dynamic pricing was used in the context of packets with values and deadlines [12] with the goal of maximizing social welfare. Dynamic subsidies were introduced in [6] in the context selfish agents and facility locations. In [9] selfish agent versions were introduced for metrical task systems [4], for the k-server problem [19] on the line, and for metrical matching [15] on the line, and appropriate dynamic pricing schemes were described for reducing social cost. Dynamic pricing for scheduling selfish agents on related machines to minimize makespan were studied in [11]. In [13] dynamic prices were used to give a good approximation to the maximal flow time. In [10] dynamic prices were used to approximate the sum of weighted completion times. Many problems and extensions remain open.

1.1.2 The k-server problem

The k-server problem was introduced by Manasse et al. [19] as a far reaching generalization of various online problems. The best-studied of those is the paging (caching) problem, which corresponds to k-server problem on a uniform metric space. Sleator and Tarjan [20] gave several k-competitive algorithms for paging and proved that this is the best possible ratio for any deterministic algorithm.

The famous *k-server conjecture* of Manasse et al. [19] hypothesizes that the k-server problem is no harder in other metric spaces, i.e., that k is the optimal ratio for deterministic algorithms in general metrics. A lower bound of k holds in any metric space of at least $k+1$ points [19], and a nearly matching upper bound of $2k-1$ was given for the Work Function Algorithm (WFA) by Koutsoupias and Papadimitriou [17], which remains the best known algorithm for general metrics. The conjecture has been settled (exactly) for several special metrics. In particular, Chrobak et al. [7] gave an elegant k-competitive algorithm for the line metric, called Double Coverage (DC), which was later extended and shown to be k-competitive for all tree metrics [8]. Additionally, Bartal and Koutsoupias have shown that WFA is k-competitive for the line, the star, and all metric spaces with $k+2$ points [3].

Moreover, Bansal et al. [2] have recently shown that the exact competitive ratio of the DC algorithm, which we simulate by dynamic pricing scheme, when it uses k servers but the offline optimum uses only $h \leq k$ servers is $\frac{k(h+1)}{k+1}$. (For such setting, the general lower bound is $\frac{k}{k-h+1}$ [19], which is matched only for the special case of paging [20].)

Most results on the k-server problem can be found in the survey by Koutsoupias [16]. Due to our focus, we ignore the randomized variant, on which there is significant recent progress [5].

1.2 Roadmap to this Paper

The next section, Section 2 gives the model and sufficient condition to give of competitive pricing algorithms on trees. We show that any algorithm that is *lazy*, *local*, and *monotone* can be used to derive a dynamic pricing scheme, and that a dynamic pricing scheme implies that such an algorithm must exist. Section 3 gives an algorithm that is clearly lazy, local and monotone, but it remains to show that all points on the tree are associated with some server, *i.e.*, that the algorithm is well defined. This is shown in Section 4. In Section C (in the Appendix) we show that the algorithm of Section 3 can be implemented in polynomial time. The Appendix also contains full proofs of various claims.

2 The Model and Preliminaries

2.1 The Selfish k-server problem

In this problem, there is a set of k-servers located in some metric space defined by an undirected weighted tree $T = (V, E, w)$. A sequence of selfish requests $\sigma = \langle \sigma_1, \sigma_2, \ldots, \rangle$ arrives online, where each request is issued at some point in the metric space. Before an arrival of each request, a dynamic pricing scheme sets a surcharge (price) on each server, and the arriving request chooses to be served by the server s that minimizes the sum of the distance of s from the request and the surcharge on s; the server s is then moved to the request. The dynamic pricing scheme's objective is to minimize the total distance moved by all servers.

Formally, given a request sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_T \rangle$, each of the requests must be served by one of the k servers, let $\ell = \langle \ell_1, \ell_2, \ldots, \ell_T \rangle$ denote the *solution sequence*, where $\ell_i \in \{1, \ldots, k\}$ is the index of the server which serves the i-th request. Define the *event prefix* $\sigma^{\prec t}$ to be the sequence of events up to but not including event t: $\sigma^{\prec t} = \langle \sigma_1, \sigma_2, \ldots, \sigma_{t-1} \rangle$. The servers location after request t is: $s_i(\sigma^{\prec t+1}) = s_i(\sigma^{\prec t})$ for $i \neq \ell_t$ and $s_{\ell_t}(\sigma^{\prec t+1}) = \sigma_t$. Let $s_i(\sigma^{\prec 1})$ denote the initial server location.

The cost of serving σ by the solution sequence ℓ is

$$\mathbf{COST}(\sigma, \ell) = \sum_{t=1}^{T} \mathsf{dist}(\sigma_t, s_{\ell_t}(\sigma^{\prec t})).$$

In the selfish setting, the server that serves the request σ_t in step t is chosen so as to minimize the distance of σ_t to the server's current location *plus* the surcharge function $c\colon \sigma^{\prec t} \times \{1, \ldots, k\} \mapsto \mathbb{R}^+$ (i.e., c depends only on past events). The chosen server is:

$$\ell_t^c \in \arg\min_i \mathsf{dist}\left(\sigma_t, s_i(\sigma^{\prec t})\right) + c(\sigma^{\prec t}, i).$$

Let $\ell^c = \langle \ell_1^c, \ldots, \ell_t^c \rangle$ be the (solution) sequence of server indices chosen by the selfish requests σ, and let $\ell^* = \langle \ell_1^*, \ldots, \ell_t^* \rangle$ be the servers that minimize the total cost for σ. A pricing scheme c is α-competitive if for any σ:

$$\frac{\mathbf{COST}(\sigma, \ell^c)}{\mathbf{COST}(\sigma, \ell^*)} \leq \alpha.$$

2.2 A Sufficient Condition for Competitive Pricing Algorithms on trees

In this paper, we focus on tree metrics, where given a weighted tree $T = (V, E, w)$, we define a tree metric space to include the vertices of T along with all points along the edges of T (see Fig. 3a in Appendix 5). Given two points $a, b \in T$, we denote by $\mathcal{P}[a, b]$ the [unique] path between a and b including both endpoints. We use $\mathsf{dist}(a, b)$ to denote the distance between a and b defined by the metric. We also use $\mathcal{P}(a, b]$ to denote the path from a to b that is open at a and closed at b.

We avoid reasoning about prices by describing how any online algorithm of a certain form can be converted into a dynamic pricing scheme that nudges the [upcoming] selfish agent do exactly as the online algorithm.

We use the following three properties. We say that an online algorithm is

1. *lazy* if it moves at most one server,
2. *local* if some point p has one or more servers on it, then a request at p will be served by one of these servers.
3. *monotone* if, for any two requests that the algorithm would service by the same server (for the next request to arrive), it is also true that a request at any point along the (tree) path connecting the requests would also be serviced by the same server.

▷ Observation 1. Any algorithm that is local and monotone has the following property: if server i, at s_i serves a request at r then there is no other server along the path $\mathcal{P}(s_i, r]$.

The following lemma shows that any α-competitive algorithm that satisfies the above three properties can be translated into a dynamic pricing scheme with the same competitive ratio. We sketch the proof below for a "degenerate" case, and we defer the full proof to Appendix B.

▶ **Lemma 2.** *Given a lazy, local, and monotone online algorithm for the k-server problem on tree metrics, with a competitive ratio of α, there is a dynamic pricing scheme for the k-server problem on tree metrics, with the same competitive ratio.*

Proof sketch. Just before the arrival of some request σ_t (and after serving $\sigma^{\prec t}$), every server s has an associated subtree T_s of points such that for every point $p \in T_s$ if the next request were made at p, then s would serve it; we say that s is *responsible* for T_s (breaking ties lexicographically in case multiple servers are at a request's location). These subtrees partition the whole tree metric, i.e., they are disjoint and their union is the entire tree.

First, we set the price for servers for which $T_s = \emptyset$ at ∞. Next, we observe that when setting the surcharges it is sufficient to consider just the endpoints of the subtrees. We say that two non-empty subtrees, T_s and $T_{s'}$, are *touching* at an endpoint p if there is no server s'' such that in the paths from s to p and from s' to p in T contain a point $q(\neq p) \in T_{s''}$. Note that there may be many mutually touching subtrees.

Consider a maximal collection of non-empty subtrees $T_{s_1}, T_{s_2}, \ldots, T_{s_k}$, which pairwise touch at an endpoint p. (Clearly, p belongs to one of those subtrees.) The key observation is that a selfish agent requesting service at p must be indifferent between choosing any of the servers $s_1, \ldots, s_k$. This induces a set of linear equations giving the difference in the surcharges, $c(s_i) - c(s_j)$,

$$\begin{aligned} \mathrm{dist}(s_i, p) + c(s_i) &= \mathrm{dist}(s_j, p) + c(s_j) \quad \text{for all } 1 \le i < j \le k \\ \Rightarrow c(s_i) - c(s_j) &= \mathrm{dist}(s_j, p) - \mathrm{dist}(s_i, p) \quad \text{for all } 1 \le i < j \le k. \end{aligned} \tag{1}$$

The relationship of subtrees "touching" can itself be described as a tree, so the equations above (1) can all be simultaneously satisfied. Any solution gives the prices we need. ◀

The above argument is incomplete, as when subtrees touch at tree vertices, or at at a server's location, the selfish request may deviate from the prescribed behavior of the algorithm. This issue can be treated easily by "nudging" the subtrees to avoid these phenomena. More on this in Appendix B.

We remark that it not necessarily true that a lazy, monotone, and local algorithm can be obtained from a pricing scheme. In particular, price all servers but one at ∞, this is a pricing scheme (albeit a terrible competitive ratio) but contradicts locality.

How to find a lazy, local and monotone algorithm.

Any non-lazy algorithm can be trivially transformed into a lazy algorithm simply by delaying the motion of a server that is not serving a request. However, this may contradict

Observation 1, so to preserve monotonicity we must compromise locality. Rather than simply follow the simulation. We do as in [9][1], one may move any server matched to the simulated server in a min cost matching — this is guaranteed to preserve the competitive ratio. We show below that Locality and Monotonicity can be preserved by choosing an appropriate matching.

Given an online algorithm A and a set of requests $\overline{\sigma}$, let $\mathsf{cost}(A, \overline{\sigma})$ be the cost of A for serving $\overline{\sigma}$.

▶ **Lemma 3** ([9], Lemma 4.3)**.** *Let* ON *be an online algorithm, let* $\mathsf{on}_i^{\prec t}$ *be the location of server* i *after* ON *serves requests* $\sigma^{\prec t}$*, and let* LAZY *be an algorithm that serves request* σ_t *by the server* ℓ *which is matched to* σ_t *in an arbitrary min-cost matching between* $\{\mathsf{on}_i^{\prec t+1}\}_{i \in [k]}$ *and* $\mathsf{s}^{\prec t}$*, where the latter is a vector of locations of* LAZY*'s servers after serving* $\sigma^{\prec t}$*. Then* $\mathsf{cost}(\mathsf{LAZY}, \sigma^{\prec t}) \leq \mathsf{cost}(\mathsf{ON}, \sigma^{\prec t})$ *for every* t*.*

The above lemma suggests a natural approach to find an algorithm with the three desired properties. The approach is to simulate an algorithm that does not satisfy these properties (in our case, the Double Cover algorithm discussed in Section 2.4), and whenever the simulated algorithm serves the request with one of its *simulated servers*, choose a *real server* that is matched to the simulated server in a min-cost matching. While this solution produces a lazy and local algorithm with the same competitive ratio, it is not a-priori clear *if such a server can be chosen in a way that results in a monotone algorithm.* We show that for the Double Cover algorithm, this can indeed be done.

2.3 Characterization of min-cost matching on trees

We now give a full characterization of min-cost matchings on trees. As mentioned, the matching between two sets of points P and Q ($|P| = |Q|$) in a tree metric T is more involved than in a line, as given a point $p \in P$, there can be multiple points in Q local to p that can be matched to p in a min-cost matching between P and Q. Figure 1 contains a simple example.

In order to characterize the min-cost matching we use the following definition to "cut" a tree T at point x to two trees: $T_x(p), \overline{T}_x(p)$, where $p \in T_x(p)$. Formally,

▶ **Definition 4.** *Given a tree* T *and two distinct points* $p, x \in T$*, let* $T_x(p)$ *be the subtree that contains* p *and does not contain* x *when splitting* T *into two subtrees at point* x*. Let* $\overline{T}_x(p)$ *be* $T \setminus T_x(p)$*.*

We define the lowest common ancestor of two points p and q in the tree when rooted at point r.

▶ **Definition 5.** *The* ***lowest common ancestor*** *of two points* p, q *with respect to a point* r*, as* $\mathsf{LCA}_r(p,q) = \mathsf{argmax}_{x \in T}\{\mathsf{dist}(x,r) : x \in \mathcal{P}(p,r) \cap \mathcal{P}(q,r)\}$*.*

The following Lemma gives necessary and sufficient conditions for a point $p \in P$ to be matched to $q \in Q$ in some min cost matching.

▶ **Lemma 6.** *Let* P *and* Q *be two sets of points in* T *such that* $|P| = |Q|$*, and let* $p \in P$ *and* $q \in Q$*. Then there exists a min-cost matching* $\mathcal{M} : P \to Q$ *that matches* p *to* q *if and only if the following holds — when considering every point* $x \neq q$ *on the path from* p *to* q*,* $\left|\overline{T}_x(q) \cap P\right| > \left|\overline{T}_x(q) \cap Q\right|$*.*

[1] Originally shown for the line, but the proof works for any metric space, which we show in Appendix A for completeness.

The following structural lemma is used in our proofs (we defer both proofs to Appendix D).

▶ **Lemma 7.** *Let P, Q be two sets of points in T ($|P| = |Q|$). For points $q, r \in T$, let $T_r(q)$ be a sub-tree such that $|T_r(q) \cap P| > |T_r(q) \cap Q|$. Then there exists $p \in T_r(q) \cap P$ such that for all $x \in \mathcal{P}(p, r)$, $\left|\overline{T}_x(r) \cap P\right| > \left|\overline{T}_x(r) \cap Q\right|$.*

2.4 The Double Cover algorithm

In order to achieve an optimal deterministic bound, our surcharge algorithm simulates the Double Cover (DC) algorithm on trees [8]. In [8], the following was shown.

▶ **Theorem 8** ([8]). *The Double Cover algorithm is k-competitive.*

The algorithm roughly works as follows: When a request is issued at some point r, move all the servers that "see" r (have no other server on the path to r) at the same speed until either (i) a server d is blocked by another server c that moves towards r, in which case d no longer "sees" r and will cease moving towards r (and all servers that see r will continue moving towards r), or (ii) a server d reached r's position, in which case, the servers stop moving, and d serves r.

We use the following notation throughout the paper. The locations of the Double Cover servers, $\mathsf{dc}_i(\sigma^{\prec t}) \in M$, $i = 1, \dots, k$, determine the "area of responsibility" for every Double Cover server: should some request occur at point $p \in M$, there is at least one server i at $\mathsf{dc}_i(\sigma^{\prec t})$ that will be used by the Double Cover algorithm to serve the request at p. If the time t and requests $\sigma^{\prec t} = \sigma_1, \dots, \sigma_{t-1}$ are fixed, we can simplify notation as follows:

$$
\begin{aligned}
s_i &= s_i(\sigma^{\prec t}), \qquad i = 1, \dots, k, \\
\mathsf{S} &= \langle s_1, \dots, s_k \rangle \\
\mathsf{dc}_i &= \mathsf{dc}_i(\sigma^{\prec t}), \\
\mathsf{DC} &= \langle \mathsf{dc}_1, \dots, \mathsf{dc}_k \rangle \\
\mathsf{dc}_i(r) &= \mathsf{dc}_i(\sigma^{\prec t} r) \qquad r \in T, \\
\mathsf{DC}(r) &= \langle \mathsf{dc}_1(r), \dots, \mathsf{dc}_k(r) \rangle.
\end{aligned}
$$

In [9], we showed that for the line metric, exactly one of the two adjacent *real* servers to the request can be matched to the simulated server at the request (Lemma 4.2 in [9]). Moreover, if we use DC on the line as ON, serving the request σ_t using the adjacent real server that is matched to σ_t recovers monotonicity (Lemma 4.4 in [9]). For the case where the underlying metric is a tree, this is much more involved, as there can be multiple adjacent real servers that can be matched to σ_t in a min cost matching, and choosing the wrong one might result in a violation of monotonicity, as shown in Figure 1. In Section 3, we define a binary relation $\succ_r$ on pairs of servers that can serve a request at point r such that if $i \succ_r j$, then server i cannot cause a monotonicity issue with respect to server j (more on that in the relevant section). Since $\succ_r$ is a strict order(see Lemma 16), there exists a server that is maximal with respect to $\succ_r$, and using this server would not cause monotonicity issue.

The following property on the movement of the double cover servers on trees that is used to prove the correctness of our algorithm. We defer the proof of the Lemma to Appendix **??**.

▶ **Lemma 9.** *For any DC server dc_i, and any point $r \in T$: If dc_i does not serve the request at r ($\mathsf{dc}_i(r) \neq r$), then for any $p \notin T_r(\mathsf{dc}_i)$ we have $\mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(p)] \subseteq \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(r)]$.*

Proof. Consider the trail of a DC server moving in response to a request. Observe that every point along the trail was closer to the (former) location of the DC server than to the (former) location of any other DC server. That is:

$$\text{For all } \mathsf{dc}_j, r \in T, \text{for every } q \in \mathcal{P}(\mathsf{dc}_j, \mathsf{dc}_j(r)], \mathsf{dist}(\mathsf{dc}_j, q) < \mathsf{dist}(\mathsf{dc}_z, q) \text{ for all } z \neq j. \quad (2)$$

Let $\mathsf{dc}_j(r,t)$ be the position of server j after a movement of at most t units for a request r, or the maximum movement the server can make if it is blocked before moving t unites. Let $t_j(r)$ be the distance traversed by dc_j for the request r, i.e., $t_j(r) = \mathsf{dist}(\mathsf{dc}_j, \mathsf{dc}_j(r))$. Since $p \notin T_r(\mathsf{dc}_i)$, the following holds:

$$\text{For all } \mathsf{dc}_j \in T_r(\mathsf{dc}_i),\ t' \leq t_j(r) : \mathcal{P}[\mathsf{dc}_j, \mathsf{dc}_j(p,t')] \subseteq \mathcal{P}[\mathsf{dc}_j, \mathsf{dc}_j(r,t')]. \quad (3)$$

We will prove that $t_i(p) \leq t_i(r)$ and by (3) the condition holds. Let b be the DC server that blocks i, i.e. $\mathsf{dc}_b(r, t_i(r)) \in \mathcal{P}(\mathsf{dc}_i(r, t_i(r)), r)$, and let $y = \mathsf{dc}_b(r, t_i(r))$.

Case 1: $\mathsf{dc}_b \in T_r(\mathsf{dc}_i)$ and $t_b(p) \geq t_i(r)$. By (3), $\mathsf{dc}_b(p, t_i(r)) = y \in \mathcal{P}(\mathsf{dc}_i(p, t_i(r)), p)$, so dc_b block dc_i at $t_i(r)$ when the request is at p.

Case 2: $\mathsf{dc}_b \in T_r(\mathsf{dc}_i)$ and $t_b(p) < t_i(r)$. Let dc_ℓ the server which blocked dc_b, by (2) we have $\mathsf{dc}_\ell(p, t_b(p)) \notin \mathcal{P}(dc_b, y)$. Hence, $\mathsf{dc}_\ell(p, t_b(p)) \in \mathcal{P}(y,p) \subseteq \mathcal{P}(\mathsf{dc}_i(p, t_b(p)), p)$ so dc_ℓ block dc_i at $t_b(p) < t_i(r)$ when the request is at p.

Let $x = \mathsf{LCA}_p(r, \mathsf{dc}_b)$ and $t_b^x = \mathsf{dist}(t_b, x)$. Note that if $\mathsf{dc}_b \notin T_r(\mathsf{dc}_i)$ then $t_b^x \leq t_i(r)$.

Case 3: $\mathsf{dc}_b \notin T_r(\mathsf{dc}_i)$ and $t_b(p) \geq t_b^x$. Hence, $\mathsf{dc}_b(p, t_b^x) = x$ and $x \in \mathcal{P}(r,p) \subseteq \mathcal{P}(dc_i(p, t_b^x), p)$ so dc_b blocks dc_i at $t_b^x \leq t_i(r)$ when the request is at p.

Case 4: $\mathsf{dc}_b \notin T_r(\mathsf{dc}_i)$ and $t_b(p) < t_b^x$. Let dc_ℓ the server which blocked dc_b. By (2), $\mathsf{dc}_\ell(p, t_b(p)) \notin \mathcal{P}(dc_b, x)$ hence $\mathsf{dc}_\ell(p, t_b(p)) \in \mathcal{P}(x,p) \subseteq \mathcal{P}(dc_i(p, t_b^x), p)$ so dc_ℓ blocks dc_i at $t_b(p) < t_i(r)$ when the request is at p.

◀

3 An Algorithm for Dynamic Pricing on Trees

We now present a lazy, local and monotone k-competitive algorithm. This is a “new" (optimal) algorithm for the k-server problem on trees. As mentioned, our goal is to find a region for each server, such that for any request in the region, there exists a min cost matching which matches the server to the dc server at the request (*after* the movement of the dc servers). Note that, for some requests more than one server can be matched to the request. Figure 1 contains a simple example. Moreover, the figure shows that the naïve approach that matches an arbitrary “adjacent” real server to the DC server serving the request produces non-monotonicity. We need to select the real server to move more carefully—this is the purpose of the precedence relation, $\succ_r$.

Recall the the definition of a lowest common ancestor (LCA) (Definition 5). We now define the precedence relation that is used to determined which of the servers in the min-cost matching to the DC server that serves the request can be used to serve the request. Roughly speaking, a server i *precedes* server j with respect to point r ($i \succ_r j$) if, when inspecting the LCA of i and j with respect to point r, there is a DC server ℓ that comes from j's subtree and leaves the LCA towards r. The intuition behind this definition is as follows. Suppose we choose j as the server that serves r (when j is in the min-cost matching to the DC server that serves r). If the request is at a point r' further away from r, DC server ℓ might not leave the LCA, preventing server j from being in a min-cost matching to the DC server that serves the request at r', which might result in non-monotonicity. This situation is exactly the one depicted in Figure 1.

▶ **Definition 10.** *We say that server $i \succ_r j$ (i has higher priority than j with respect to r) if (i) $\mathsf{LCA}_r(s_i, s_j) \neq s_j$, and (ii) there exists some DC server ℓ such that:*

$$\mathsf{LCA}_r(s_i, s_j) \in \mathcal{P}[\mathsf{dc}_\ell, \mathsf{dc}_\ell(r)] \quad \text{and} \quad \mathsf{dc}_\ell \in T_{\mathsf{LCA}_r(s_i,s_j)}(s_j).$$

▶ **Definition 11.** *We define*

$$\mathsf{MC}(r) = \{\ell : \exists \text{ min-cost matching } \mathcal{M} : \mathsf{S} \to \mathsf{DC}(r) \text{ such that } \mathcal{M}(s_\ell) = r\}$$

to be the set of servers that can be matched to the DC server serving the next request located at r.

▶ **Definition 12.** *A point $r \in T$ is **ℓ-colorable** for some server ℓ:*

1. *There is no server $j \neq \ell$ such that $s_j \in \mathcal{P}(s_\ell, r]$.*
2. *$\ell \in \mathsf{MC}(r)$.*
3. *There is no server j such that $j \in \mathsf{MC}(r)$ and $j \succ_r \ell$.*

The intuition behind the above definition is that Property 2 ensures that the conditions for Lemma 3 hold and thus the algorithm is k-competitive. If Property 1 did not hold then the algorithm would not be local. Finally, Property 3 ensures that the algorithm is monotone and well-defined, as we will show. See Figure 2 in Section 5 for illustrations of the various definitions made above.

Our algorithm is described in Algorithm 1. We remark that it is not obviously poly-time. In particular, it may not be clear how R_i's can be computed efficiently. However, we describe how to implement the algorithm in poly-time in Appendix C.

Algorithm 1 The Local Regions algorithm (see Fig. 3 in Appendix 5) for illustration.

Input: A tree metric T, initial servers locations $\langle s_1(\emptyset), \ldots s_k(\emptyset)\rangle \in M^k$, and an online sequence of requests $\overline{\sigma} \in T^*$.

1. After serving $\sigma^{\prec t}$, before the current request σ_t is revealed:
   - a. Initialize the forest $F^0 \leftarrow T$
   - b. For $i = 1, \ldots, k$:
     - i. $C_i \leftarrow \{p \in F^{i-1} : p \text{ is } i\text{-colorable}\}$
       # C_i is the set of points that are i-colorable in the current forest F^{i-1}.
     - ii. $R_i \leftarrow \{\, p \in C_i \;:\; \text{for all } q \in \mathcal{P}(p, s_i),\ q \in C_i \,\}$
       # R_i is the monotone region of C_i around the location of server i.
     - iii. $F^i \leftarrow F^{i-1} \setminus R_i$
       # F_i is the remaining forest after removing R_i.
2. Let σ_t be the current request, and let $\ell \in [k]$ be the server such that $\sigma_t \in R_\ell$
   - Serve σ_t with server ℓ
   - $\mathsf{dc}_{t+1} \leftarrow \mathrm{DC}(\mathsf{dc}_t, \sigma_t)$

We say that our algorithm is *well defined* if for every sequence $\sigma^{\prec t}$, for every point $x \in T$, there exists a server i such that $x \in R_i$.

▶ **Theorem 13.** *There exists a dynamic pricing scheme for the selfish k-server problem on trees with an optimal competitive ratio of k.*

Proof. Assuming Algorithm 1 is lazy, local, monotone and well defined, it can be simulated by a pricing scheme by Lemma 2 and it is k-competitive by Lemma 3, because a point $r \in T$

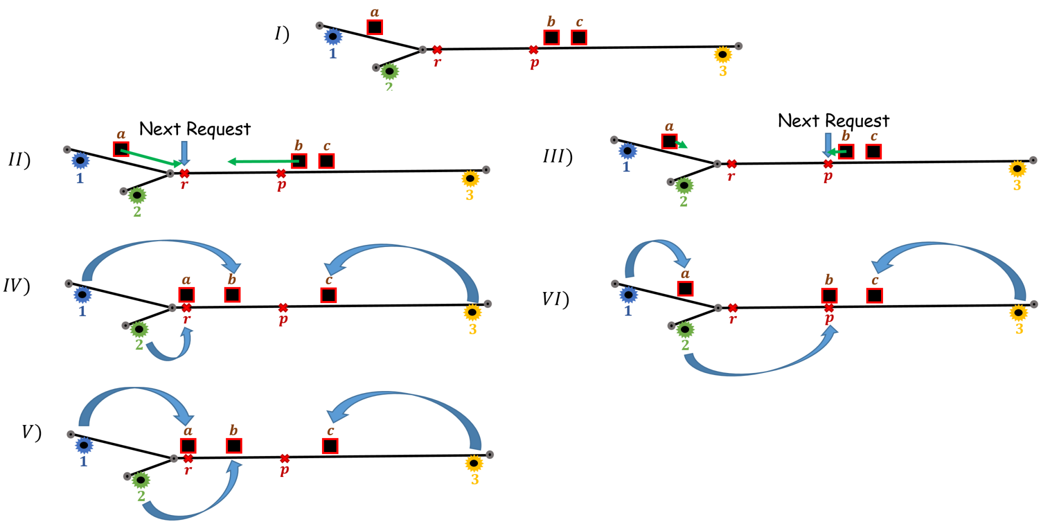


Figure 1 In order to maintain double cover's (DC) competitive ratio, we want to serve each request with a real server that "sees" the request (has no intermediate real servers along the path to the request), and is matched to a DC server that serves the request in a min cost matching between the *real* servers and the *simulated* DC servers. Unfortunately, choosing an arbitrary real server that is matched to the DC server might violate monotonicity. In the figure above DC servers are depicted by squares, namely a, b, c, and real servers by circles, namely $1, 2, 3$. Figure I depicts the initial configuration. We consider two possible locations of the next request, r, p. If the next request is at r, depicted in Figure II, then after the DC servers move, server a which served the request can either be matched to the green(2) server (Figure IV), or to the blue(1) server (Figure V) in the min-cost matching. If one chooses to serve the request with the blue(1) server, then it violates monotonicity. This is since if the next request in the initial configuration is on p (Figure III) instead, then the unique min-cost matching matches the green(2) server to server b. Finally, note that in the initial configuration r *is not* blue(1) colorable. According to Definition 12, properties 1 and 2 hold for the blue(1) server, but property 3 does not since $(2) \in MC(r)$ and $(2) \succ_r (1)$ (DC server a traverses $\mathsf{LCA}_r(1,2)$ and 'arrives' from the blue(1) server subtree).

is served by server ℓ only if r is in R_ℓ, and therefore r is ℓ-colorable, which implies $\ell \in \mathsf{MC}(r)$. The ℓ-colorability (property 1) of r further implies locality of the algorithm, whereas its laziness follows by definition. The monotonicity of the algorithm follows by step 1(b)ii of Algorithm 1, since the region contains only points p such that all other points on the path from p to the server are also in the region of the server[2]. To conclude the proof, Lemma 14 below implies the algorithm is well-defined. ◀

4 Algorithm 1 is Well Defined

In this section, we show that Algorithm 1 is well defined, i.e. that every point in the tree would be in some server's region, concluding the proof of Theorem 13. To help the reader in following this section, various figures, depicting important lemmas of this section, are presented in Figure 4 of Section 5.

▶ **Lemma 14** (Well-Defined Lemma)**.** *For any sequence σ, Algorithm 1 is well-defined.*

[2] We note that C_i itself might not be continuous, and therefore, step 1(b)ii is needed in order to ensure monotonicity.

We use the following observation:

▷ Observation 15 (See Figure 4a). From the definition, we observe that for every r, p, q in T ($r \neq p$):

(1) For $q \in T_r(p)$, we have: $x \in T_r(p) \iff r \notin \mathcal{P}[x, q]$.

(2) For $q \notin T_r(p)$, we have: $x \in T_r(p) \Rightarrow r \in \mathcal{P}[x, q]$.

In order to prove Lemma 14, we first show that the relation $\succ_r$ is a strict partial order.

▶ **Lemma 16.** *$\succ_r$ is a strict partial order relation for every $r \in T$.*

Proof. In order to show that $\succ_r$ is a strict partial order relation, we need to show it is irreflexive and transitive. (Note that these two properties imply asymmetry.) Since it is clear that $\succ_r$ is irreflexive ($\mathsf{LCA}_r(s_j, s_j) = s_j$ for every $r \in T$ and j), we show that it is transitive.

Assume that $i \succ_r j$ and $j \succ_r \ell$, we prove that $i \succ_r \ell$. Let $L_{i,j} = \mathsf{LCA}_r(s_i, s_j)$ and $L_{j,\ell} = \mathsf{LCA}_r(s_j, s_\ell)$ and $L_{i,\ell} = \mathsf{LCA}_r(s_i, s_\ell)$. Let $\mathsf{dc}_{i,j}$ and $\mathsf{dc}_{j,\ell}$ be the respective dc servers which order the servers, *i.e.*, $L_{i,j} \in \mathcal{P}[\mathsf{dc}_{i,j}, \mathsf{dc}_{i,j}(r)]$ and $\mathsf{dc}_{i,j} \in T_{L_{i,j}}(s_j)$, and $L_{j,\ell} \in \mathcal{P}[\mathsf{dc}_{j,\ell}, \mathsf{dc}_{j,\ell}(r)]$ and $\mathsf{dc}_{j,\ell} \in T_{L_{j,\ell}}(s_\ell)$.

Case 1. $L_{i,j} \in \mathcal{P}[L_{j,\ell}, r]$, hence $L_{i,\ell} = L_{i,j}$ and $T_{L_{i,j}}(s_j) = T_{L_{i,j}}(s_\ell)$, and therefore $L_{i,\ell} \in \mathcal{P}[\mathsf{dc}_{i,j}, \mathsf{dc}_{i,j}(r)]$ and $\mathsf{dc}_{i,j} \in T_{L_{i,\ell}}(s_\ell)$. By Definition 10 $i \succ_r \ell$.

Case 2. $L_{j,\ell} \in \mathcal{P}[L_{i,j}, r]$, hence $L_{i,\ell} = L_{j,\ell}$ and therefore $L_{i,\ell} \in \mathcal{P}[\mathsf{dc}_{j,\ell}, \mathsf{dc}_{j,\ell}(r)]$ and $\mathsf{dc}_{j,\ell} \in T_{L_{i,\ell}}(s_\ell)$. By Definition 10 $i \succ_r \ell$. ◀

This allows us to conclude that every point in the tree T is colorable by some server.

▶ **Lemma 17.** *For any $r \in T$, there exist j such that r is j-colorable.*

Proof. Consider a point $r \in T$. Recall that $\mathsf{MC}(r)$ is the set of servers the can be matched to r in a min-cost matching between S and $\mathsf{DC}(r)$. Since $\succ_r$ is a strict order relation (by Lemma 16), there is a server $\ell \in \mathsf{MC}(r)$ that is maximal with respect to $\succ_r$ in $\mathsf{MC}(r)$, i.e., such that for every server $j \in \mathsf{MC}(r)$, $j \not\succ_r \ell$. Hence, there is a server ℓ for which Properties 2 and 3 of ℓ-colorability hold.

Let ℓ be a server for which Properties 2 and 3 hold. If there is no other server in $\mathcal{P}(s_\ell, r]$, then Property 1 holds as well and r is ℓ-colorable. Otherwise, we claim that every for server in $\mathcal{P}(s_\ell, r]$ Properties 2 and 3 hold. Since for the closest server to r in $\mathcal{P}(s_\ell, r]$, j, Property 1 holds as well, it follows that r is j-colorable.

Let ℓ be a server in $\mathsf{MC}(r)$ which is maximal with respect to $\succ_r$. Let j be a server for which $s_j \in \mathcal{P}(s_\ell, r]$. Since the path from s_j to r is a subpath of the path from s_ℓ to r, and since for every $x \in \mathcal{P}[s_j, r]$, $\overline{T}_x(s_j) = \overline{T}_x(s_\ell)$, the characterization of Lemma 6 holds for s_j and r as well, hence, $j \in \mathsf{MC}(r)$.

Now assume that j is not maximal with respect to $\succ_r$, that is, there exists some server $j' \in \mathsf{MC}(r)$ such that $j' \succ_r j$. By Definition 10, $\mathsf{LCA}_r(s_{j'}, s_j) \neq s_j$, and there exists some server ℓ' such that

$$\mathsf{LCA}_r(s_{j'}, s_j) \in \mathcal{P}[\mathsf{dc}_{\ell'}, \mathsf{dc}_{\ell'}(r)] \quad \text{and} \quad \mathsf{dc}_{\ell'} \in T_{\mathsf{LCA}_r(s_{j'}, s_j)}(s_j).$$

Let $x := \mathsf{LCA}_r(s_{j'}, s_j)$. Since $x \neq s_j$, and since $s_\ell \in \overline{T}_{s_j}(r)$, it must be the case that $\mathsf{LCA}_r(s_{j'}, s_\ell) = x$. Therefore, $\mathsf{LCA}_r(s_{j'}, s_\ell) \in \mathcal{P}[\mathsf{dc}_{\ell'}, \mathsf{dc}_{\ell'}(r)]$. Since when splitting the tree at x, the subtree containing s_ℓ is also the subtree containing s_j, we also have that $\mathsf{dc}_{\ell'} \in T_x(s_\ell)$ which implies that $j' \succ_r \ell$ as well, in contradiction to ℓ's maximality. Therefore, it must be the case that j is maximal as well. This implies that r is j-colorable by some server j which concludes the proof. ◀

A subtree $\tilde{T}$ is *fully-colorable* if for any point $p \in \tilde{T}$ there exists a server ℓ such that p is ℓ-colorable and $s_\ell \in \tilde{T}$. Since Algorithm 1 preserves monotonicity, it follows that a server would color points only in the subtree containing this server. Therefore, in order to prove that Algorithm 1 is well-defined we need to show that not only the original tree T is *fully-colorable* (Lemma 17), but also that every $\tilde{T} \in F^{i-1}$ is fully-colorable as well.

For the sake of proving this property (Corollary 23), we characterize properties of the min-cost matching $\mathsf{MC}(p)$ and the relation $\succ_p$. First, we now show that for any server ℓ the region in which ℓ is in the min-cost matching is monotone.

▶ **Lemma 18** (See Figure 4b). *For any server ℓ and two points r, p in T such that $p \notin T_r(s_\ell)$, the following holds—if $\ell \in \mathsf{MC}(p)$ then $\ell \in \mathsf{MC}(r)$.*

Proof. We will show that for any point $x \in \mathcal{P}[s_\ell, r]$, if $\mathsf{dc}_j(r) \in T_x(s_\ell)$ then $\mathsf{dc}_j(p) \in T_x(s_\ell)$:

First, we observe that $\mathsf{dc}_j(r) \neq r$ (dc_j does not serve request at r), since $r \notin T_x(s_\ell)$ and $\mathsf{dc}_j(r) \in T_x(s_\ell)$. Then, we observe that $\mathsf{dc}_j \in T_x(s_\ell)$, since $\mathcal{P}(\mathsf{dc}_j(r), x) \subseteq \mathcal{P}(\mathsf{dc}_j, x)$. By Lemma 9, we have $\mathcal{P}[\mathsf{dc}_j, \mathsf{dc}_j(p)] \subseteq \mathcal{P}[\mathsf{dc}_j, \mathsf{dc}_j(r)]$, since $x \notin \mathcal{P}(\mathsf{dc}_j, \mathsf{dc}_j(r))$ ($\mathsf{dc}_j(r) \in T_x(s_\ell)$), we have $x \notin \mathcal{P}(\mathsf{dc}_j, \mathsf{dc}_j(p))$ and we have $\mathsf{dc}_j(p) \in T_x(s_i)$.

We get that for every x in $\mathcal{P}[s_\ell, r]$, if $\mathsf{dc}_j(r) \in T_x(s_\ell)$, then $\mathsf{dc}_j(p) \in T_x(s_\ell)$, which implies $|T_x(s_\ell) \cap \mathsf{dc}(p)| \geq |T_x(s_\ell) \cap \mathsf{dc}(r)|$. Since $\ell \in \mathsf{MC}(p)$, for any $x \in \mathcal{P}[s_\ell, r]$ we have $|T_x(s_\ell) \cap S| > |T_x(s_\ell) \cap \mathsf{dc}(p)|$. Which together yields that the condition of Lemma 6 hold also for $\mathsf{dc}(r)$, and therefore $\ell \in \mathsf{MC}(r)$. ◀

Which yields the following lemma which will be used to prove Lemma 22.

▶ **Lemma 19** (See Figure 4c). *For any two servers b, ℓ and a points x in T such that $b \in \mathsf{MC}(x)$ and $s_\ell \notin T_x(s_b)$ we have for any $p \in \mathcal{P}(s_b, x)$ that $\ell \notin \mathsf{MC}(p)$.*

Proof. Assume towards a contradiction that there exists $p \in \mathcal{P}(s_b, x)$ such that $\ell \in \mathsf{MC}(p)$. Consider a point $y \in \mathcal{P}(x, p)$ which isn't a tree vertex, and in which at most a single DC server will arrive if the request is issued at this point (there exists such a point due to the continuity of the metric space). According to Lemma 18, $\ell, b \in \mathsf{MC}(y)$.

Therefore, by Lemma 7 we have:

$$|T_y(s_b) \cap \mathsf{DC}(y)| < |T_y(s_b) \cap \mathsf{S}|\,, \text{ and}$$
$$|T_y(s_\ell) \cap \mathsf{DC}(y)| < |T_y(s_\ell) \cap \mathsf{S}|\,.$$

Since y is not a tree node, $T = T_y(s_\ell) \cup T_y(s_b) \cup \{y\}$. Moreover, there is at most one $\mathsf{DC}(y)$ server at y (by y's selection), so overall there are more real servers than $\mathsf{DC}(y)$ servers, a contradiction. ◀

The following is an important property of the strict partial order $\succ_r$ we defined over the servers.

▶ **Lemma 20** (See Figure 4d). *For any two servers ℓ, j, a point r such that $s_j \in T_r(s_\ell)$, and any point $p \notin T_r(s_\ell)$: If $j \succ_p \ell$, then $j \succ_r \ell$.*

Proof. First, since $s_j \in T_r(s_\ell)$ then $\mathsf{LCA}_r(s_\ell, s_j) \in T_r(s_\ell)$, therefore we have that $\mathsf{LCA}_r(s_\ell, s_j) = \mathsf{LCA}_p(s_\ell, s_j)$. Second, $j \succ_p \ell$ therefore there exists dc_i such that $\mathsf{dc}_i \in T_{\mathsf{LCA}_p(s_\ell, s_j)}(s_\ell)$, and $\mathsf{LCA}_p(s_\ell, s_j) \in \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(p)]$. Clearly, if the request is on r and dc_i serves point r then $\mathsf{LCA}_r(s_\ell, s_j) \in \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(r)]$. If dc_i does not serves point r, by Lemma 9 we have $\mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(p)] \subseteq \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(r)]$, and again $\mathsf{LCA}_r(s_\ell, s_j) \in \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(r)]$. In either case $\mathsf{LCA}_r(s_\ell, s_j) \in \mathcal{P}[\mathsf{dc}_i, \mathsf{dc}_i(r)]$ and by Definition 10 we have $j \succ_r \ell$. ◀

We now prove the main technical lemma used in proving that the algorithm is monotone. The lemma roughly shows the following. Let $r \in T$ be some point that is ℓ colorable by some server ℓ, and let j be another server on the 'same side' of ℓ with respect to r. Let p be a point on the other side of ℓ and j with respect to r. The lemma states that if p is j-colorable, then it is also ℓ-colorable (see Figure 4e for a visual depiction).

The significance of this lemma is the following—suppose r is a point that the algorithm decided should be served by some server ℓ (which obviously means r is ℓ-colorable). Since we want our algorithm to be monotone, this immediately disconnects all the points further away from r from the servers that are on the same side as ℓ with respect to r. This would be a problem if there was such a point p that can be served only by servers on the same side as ℓ, but not ℓ itself. The lemma basically shows this situation cannot happen.

▶ **Lemma 21** (See Figure 4e). *For any two servers ℓ, j and two points r, p in T such that $s_j, s_\ell \in \overline{T}_r(p)$: If r is ℓ-colorable and p is j-colorable, then p is ℓ-colorable.*

Proof. Assume for contradiction that p is not ℓ-colorable. We consider the following cases

Case 1. $\ell \in \mathsf{MC}(p)$. By the definition of ℓ-colorable, we have that there is a server i such that $i \in \mathsf{MC}(p)$ and $i \succ_p \ell$. If $s_i \in \overline{T}_r(p)$, then by Lemma 18, $i \in \mathsf{MC}(r)$, and by Lemma 20, $i \succ_r \ell$, Hence r is not ℓ-colorable, a contradiction. Otherwise, $s_i \in T_r(p)$. Let $x = \mathsf{LCA}_p(s_\ell, s_i)$. Note that $r \in \mathcal{P}[s_\ell, p], r \in \mathcal{P}[s_j, p]$ and $r \notin \mathcal{P}[s_i, p]$ by Observation 15. We get that $\mathcal{P}[s_i, p] \cap \mathcal{P}[s_\ell, p] = \mathcal{P}[s_i, p] \cap \mathcal{P}[r, p] = \mathcal{P}[s_i, p] \cap \mathcal{P}[s_j, p]$, hence $\mathsf{LCA}_p(s_j, s_i) = \mathsf{LCA}_p(s_\ell, s_i) = x$. In addition, $T_x(s_\ell) = T_x(r) = T_x(s_j)$, and since $i \succ_p \ell$ we get $i \succ_p j$ by Definition 10. Recall that, $i \in \mathsf{MC}(p)$, therefore p not j-colorable, a contradiction.

Case 2. $\ell \notin \mathsf{MC}(p)$. By Lemma 6, there exists a point x on the path from s_ℓ to p such that

$$|T_x(s_\ell) \cap \mathsf{S}| \le |T_x(s_\ell) \cap \mathsf{DC}(p)| \,. \tag{4}$$

Let x be the closest point to r for which (4) holds. Since $j \in \mathsf{MC}(p)$, by Lemma 6, for every point y on the path from s_j to p, $|T_y(s_j) \cap \mathsf{S}| > |T_y(s_j) \cap \mathsf{DC}(p)|$, and hence, $x \in \mathcal{P}[s_\ell, \mathsf{LCA}(s_\ell, s_j)] \subseteq \mathcal{P}[s_\ell, r]$. Moreover, since r is ℓ-colorable, $\ell \in \mathsf{MC}(r)$, so Lemma 6 implies that

$$|T_x(s_\ell) \cap \mathsf{S}| > |T_x(s_\ell) \cap \mathsf{DC}(r)| \,. \tag{5}$$

Therefore, combining (4) and (5) yields $|T_x(s_\ell) \cap \mathsf{DC}(r)| < |T_x(s_\ell) \cap \mathsf{DC}(p)|$, and there must be a server dc_a such that $\mathsf{dc}_a \in T_x(s_\ell)$ and $\mathsf{dc}_a(r) \notin T_x(s_\ell) \Rightarrow x \in \mathcal{P}[\mathsf{dc}_a, \mathsf{dc}_a(r)]$. In addition, we have

$$\left|\overline{T}_x(r) \cap \mathsf{S}\right| > \left|\overline{T}_x(r) \cap \mathsf{DC}(p)\right| , \tag{6}$$

since x is the closest point to p for which (4) holds. Combining (4) and (6) yields that in $\widehat{T} = \overline{T}_x(r) \setminus T_x(s_\ell)$ we have $\left|\widehat{T} \cap \mathsf{S}\right| > \left|\widehat{T} \cap \mathsf{DC}(p)\right|$. Notice that for every $b \neq a$ such that $\mathsf{dc}_b \in \overline{T}_x(r)$, we have that $\mathsf{dc}_b(r) \in \overline{T}_x(r)$ since only a single DC server can cross point x. Since $\left|\widehat{T} \cap \mathsf{DC}(p)\right| = \left|\widehat{T} \cap \mathsf{DC}\right|$, by Lemma 9, we get $\left|\widehat{T} \cap \mathsf{DC}(p)\right| = \left|\widehat{T} \cap \mathsf{DC}(r)\right|$. Therefore, $\left|\widehat{T} \cap \mathsf{S}\right| > \left|\widehat{T} \cap \mathsf{DC}(r)\right|$, and Lemma 7 implies that there exists $s_i \in \widehat{T}$ such that for all $z \in \mathcal{P}[s_i, x]$, we have

$$|T_z(s_i) \cap \mathsf{S}| > |T_z(s_\ell) \cap \mathsf{DC}(r)| \,. \tag{7}$$

In addition, (7) holds also for $z \in (x, r)$ by (5), hence, $i \in \mathsf{MC}(r)$. Moreover, since $x = \mathsf{LCA}_r(s_i, s_\ell)$, $x \in \mathcal{P}[\mathsf{dc}_a, \mathsf{dc}_a(r)]$ and $\mathsf{dc}_a \in T_x(s_\ell)$, we also have $i \succ_r \ell$, which combined with $i \in \mathsf{MC}(r)$ is a contradiction to r being ℓ-colorable. ◀

The main lemma to show the property *fully-colorable* is the following:

▶ **Lemma 22.** *For a fully-colorable sub-tree $\tilde{T}$, let $r, p \in \tilde{T}$ be two points and ℓ a server in $\tilde{T}$ such that $p \notin T_r(s_\ell)$. If we have that*

- *r is ℓ-colorable, and*
- *for all servers a such that $s_a \in \tilde{T}$ where p is a-colorable, we have $s_a \in T_r(s_\ell)$,*

then for any $x \in \mathcal{P}(r, p]$, x is ℓ-colorable.

Proof. First, by Lemma 21 we have that p is ℓ-colorable as well. Assume for the purpose of contradiction that it is not true, let $x \in \mathcal{P}(r, p)$ be the closet point to p such that x is not ℓ-colorable. Since $\tilde{T}$ is fully-colorable, there exists a server b, such that $s_b \in \tilde{T}$ and x is b-colorable. Note that, if $s_b \in T_r(s_\ell)$, then $s_b, s_\ell \in \overline{T}_r(x)$, and since r is ℓ-colorable, by Lemma 21, x is ℓ colorable, a contradiction. Let $L = \mathsf{LCA}_r(p, s_b)$

Case 1. One of the following two holds: (i) $x \notin \mathcal{P}(s_b, s_\ell)$, (ii) $x = L$. In this case, $s_b, s_\ell \in \overline{T}_x(p)$ and x is b-colorable. Therefore, by Lemma 21, p is b-colorable, a contradiction.

Case 2. $x \in \mathcal{P}(s_b, s_\ell)$, and $x \neq L$, which implies $s_\ell \notin T_x(s_b)$, and $b \in \mathsf{MC}(x)$ (since x is b-colorable). Therefore, by Lemma 19, we have $\ell \notin \mathsf{MC}(y)$ for any $y \in \mathcal{P}(s_b, x)$, however since $x \neq L$, there exist $z \in \mathcal{P}(x, s_b) \cap \mathcal{P}(x, p)$, on one hand z is ℓ-colorable (by our choice of x), on the other hand $\ell \notin MC(z)$ (since $z \in \mathcal{P}(s_b, x)$), a contradiction. ◀

The above lemma implies the following corollary which yields that Algorithm 1 is well-defined.

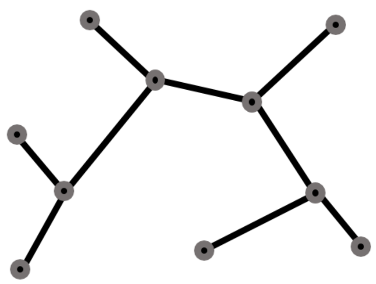
▶ **Corollary 23.** *For a fully-colorable subtree $\tilde{T}$, and i a server such that $s_i \in \tilde{T}$, then for all subtrees $\hat{T} \in \tilde{T} \setminus R_i$ we have that $\hat{T}$ is fully-colorable tree.*

Proof. Let p be the point in $\hat{T}$ for which this does not hold, since $\tilde{T}$ is fully-colorable, let j be the server such that $s_j \in \tilde{T}$ and p is j-colorable. Let $r = \mathsf{argmin}_x\{\mathsf{dist}(p, x) : x \in \mathcal{P}(s_i, p) \cap R_i\}$ be the closest point to p in R_i. Observe that $r \notin \mathcal{P}(s_j, s_i)$ since otherwise $\mathcal{P}(s_j, p) \subseteq \mathcal{P}(s_j, r) \cup \mathcal{P}(r, p)$, where $\mathcal{P}(s_j, r) \cap R_i = \emptyset$ and $\mathcal{P}(r, p) \cap R_i = \emptyset$. Therefore, $\mathcal{P}(s_j, p) \cap R_i = \emptyset$, and thus $s_j \in \hat{T}$, a contradiction. Hence, by Observation 151, $s_j \in T_r(s_i)$. Finally, By Lemma 22, the entire $\mathcal{P}(r, p)$ is i-colorable, a contradiction for $p \notin R_i$. ◀

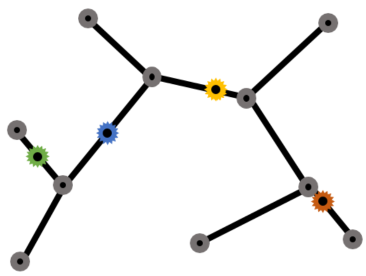
Using this corollary, we can now prove the Well-Defined Lemma.

Proof of Well-Defined Lemma [Lemma 14]. In order for Algorithm 1 to be well-defined, each point in T should be in the R_ℓ region of some server ℓ. We will show that each subtree $\tilde{T} \in F^i$ after iteration i in the run of the algorithm execution is fully-colorable. The initial tree, T is fully-colorable by Lemma 17. After each iteration i, every subtree in F^i is fully-colorable by Corollary 23 (Note that, R_i is a subregion of a single subtree of F^{i-1}). Therefore, eventually a sub-tree would contain a single server and it is fully-colored by this server, which yields that $F^k = \emptyset$ as needed. ◀

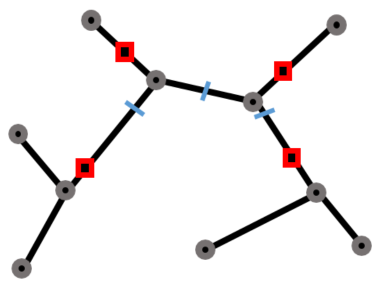
5 Figures



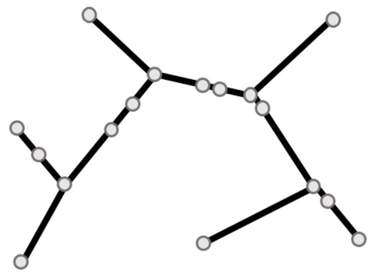
(a) A tree metric.



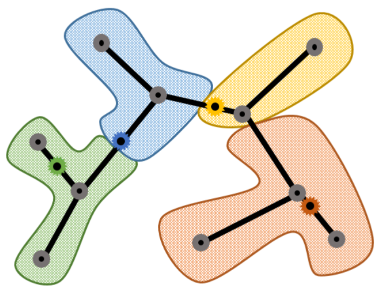
(b) Servers' locations $\mathbf{s}^{\prec t}$.



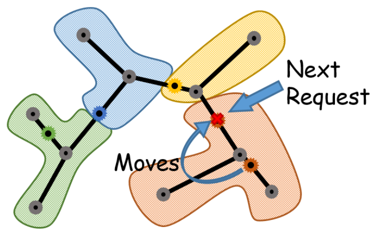
(c) DC servers' locations $\mathsf{dc}^{\prec t}$ with boundary points $B^{\prec t}$.



(d) The critical tree graph $T_c^{\prec t}$.

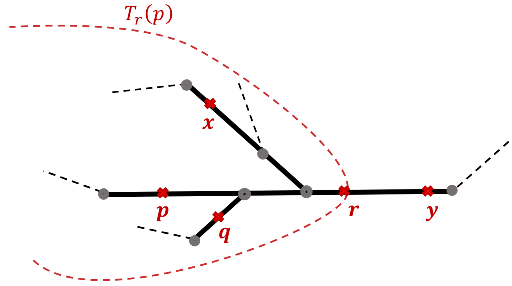


(e) The coloring of the tree as produced by ColorRegion. Notice that the tree is colored irrespective of the next request.

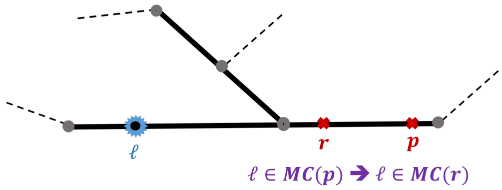


(f) When (next) request σ_t occurs, it is serviced by the server in whose region it is located.

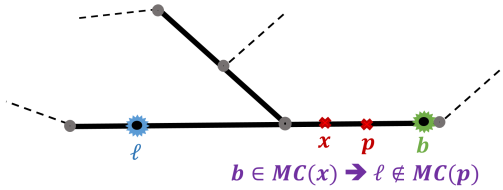
Figure 3 Key ingredients for Algorithm 1.



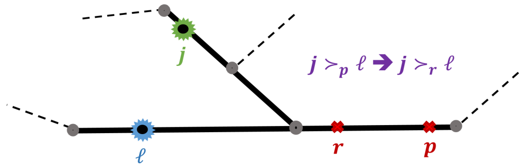
(a) Observation 15.



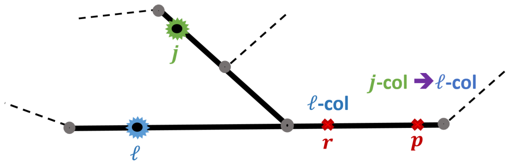
(b) Lemma 18.



(c) Lemma 19.



(d) Lemma 20.



(e) Lemma 21.

Figure 4 A visual depiction of the lemmas used in order to prove the Well-Defined Lemma.

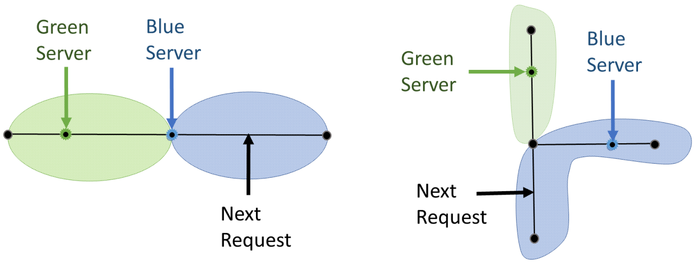


Figure 5 Issues with the naïve pricing algorithm. In the example on the left, the range served by the blue server has the blue server on its left end. The open interval up to the blue server is served by the green server. By setting the surcharges as in the naïve algorithm, a selfish request (the next request) in the blue zone is indifferent between moving the green and blue servers, so we have no guarantee that selfish agents emulate the online algorithm. The figure on the right shows a similar problem where the green and blue regions touch, and, again, by setting the prices naïvely, selfish agents may choose to move either the green or the blue agent in response to a request. In both cases, a solution to this problem is to break the tie by "pushing" the boundary between the green and blue regions slightly "away" from the blue region. See Figure 6 for details.

References

1 Baruch Awerbuch, Yossi Azar, and Adam Meyerson. Reducing truth-telling online mechanisms to online optimization. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 503–510, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780616.

2 Nikhil Bansal, Marek Eliáš, Lukasz Jez, Grigorios Koumoutsos, and Kirk Pruhs. Tight bounds for double coverage against weak adversaries. *Theory Comput. Syst.*, 62(2):349–365, 2018. URL: https://doi.org/10.1007/s00224-016-9703-3, doi:10.1007/s00224-016-9703-3.

3 Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theor. Comput. Sci.*, 324(2-3):337–345, 2004.

4 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

5 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16, 2018. URL: https://doi.org/10.1145/3188745.3188798, doi:10.1145/3188745.3188798.

6 Niv Buchbinder, Liane Lewin-Eytan, Joseph (Seffi) Naor, and Ariel Orda. Non-cooperative cost sharing games via subsidies. *Theor. Comp. Sys.*, 47(1):15–37, July 2010. URL: http://dx.doi.org/10.1007/s00224-009-9197-3, doi:10.1007/s00224-009-9197-3.

7 Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

8 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

9 Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January*

4-6, 2015, pages 73–91. SIAM, 2015. URL: `http://dx.doi.org/10.1137/1.9781611973730`, `doi:10.1137/1.9781611973730.7`.

10 Alon Eden, Michal Feldman, Amos Fiat, and Tzahi Taub. Truthful prompt scheduling for minimizing sum of completion times. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 27:1–27:14, 2018. URL: `https://doi.org/10.4230/LIPIcs.ESA.2018.27`, `doi:10.4230/LIPIcs.ESA.2018.27`.

11 Michal Feldman, Amos Fiat, and Alan Roytman. Makespan minimization via posted prices. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 405–422, 2017. URL: `http://doi.acm.org/10.1145/3033274.3085129`, `doi:10.1145/3033274.3085129`.

12 Amos Fiat, Yishay Mansour, and Uri Nadav. Efficient contention resolution protocols for selfish agents. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 179–188. SIAM, 2007. URL: `http://dl.acm.org/citation.cfm?id=1283383.1283403`.

13 Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Clifford Stein. Minimizing maximum flow time on related machines via dynamic posted pricing. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 51:1–51:10. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. URL: `https://doi.org/10.4230/LIPIcs.ESA.2017.51`, `doi:10.4230/LIPIcs.ESA.2017.51`.

14 Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Inf. Process. Lett.*, 39(2):85–91, 1991.

15 Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993. URL: `https://doi.org/10.1006/jagm.1993.1026`, `doi:10.1006/jagm.1993.1026`.

16 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. URL: `https://doi.org/10.1016/j.cosrev.2009.04.002`, `doi:10.1016/j.cosrev.2009.04.002`.

17 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

18 Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, EC '00, pages 233–241, New York, NY, USA, 2000. ACM. `doi:10.1145/352871.352897`.

19 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

20 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. `doi:10.1145/2786.2793`.

A Proof of Lemma 3

Proof of Lemma 3. Given two sets of points P, Q such that $|P| = |Q|$, let $w(P, Q)$ be the weight of the min-cost matching between P and Q.

Let $\mathsf{cost}_t(\mathsf{LAZY})$ and $\mathsf{cost}_t(\mathsf{ON})$ be the respective cost of algorithms LAZY and ON when serving request σ_t. We show that for every t,

$$\mathsf{cost}_t(\mathsf{LAZY}) + \Delta\Phi \le \mathsf{cost}_t(\mathsf{ON}), \tag{8}$$

for a non-negative potential function $\Phi = w(\mathsf{S}, \mathsf{on})$, where S and on are the current locations of the servers of LAZY and ON respectively. To prove (8), it suffices to consider the moves of ON and LAZY independently, in this order.

Fix some min-cost matching $\mathcal{M} : \mathsf{S} \to \mathsf{on}$. We keep $\mathcal{M}$ fixed as ON moves its servers. Clearly, when ON moves a server ℓ by distance d, the cost of $\mathcal{M}$ does not increase by more than d. Hence, the same holds for the min-cost matching. Thus Φ increases by at most d, and (8) holds.

Once ON is done with its moves, we analyze the move of LAZY. Note that at this point $\sigma_t \in \mathsf{on}$, *i.e.*, ON has one of its servers at σ_t. Let $\mathcal{M}'$ be the updated min-cost matching after ON moves, and let ℓ' be some server of LAZY that is matched to σ_t. Upon the move of ℓ' to σ_t, the cost of $\mathcal{M}'$ is decreased by $\mathsf{dist}(s_{\ell'}, \sigma_t)$. Since the cost of the min-cost matching after ℓ' moves is no bigger than that of $\mathcal{M}'$, Φ decreases by at least $\mathsf{dist}(s_{\ell'}, \sigma_t)$ as well, which is exactly $\mathsf{cost}_t(\mathsf{LAZY})$. Therefore, $\mathsf{cost}_t(\mathsf{LAZY}) + \Delta\Phi \le 0$, and (8) holds. ◀

B Full Argument for Lemma 2

The proof sketch of Lemma 2 shows that one can set surcharges where for the incoming agent there exists a server that minimizes the distance + surcharge *and* this is the same server that the algorithm would choose. Whenever this server can be matched (in a min cost matching) to the DC server that served the request, Lemma 3 implies that the competitive ratio achieved is optimal. This is enough for a truthful online algorithm with optimal competitive ratio if we can break ties for the agent. However, our goal is to let the agents break ties for themselves.

We first notice the are two scenarios where an agent can have more than one disutility minimizing server — (i) either the transition between the responsibility area of server j and adjacent server i is the location of server i (left side of Figure 5). In this case, setting prices using Equation (1) will result in both server i *and* server j being the disutility minimizing servers for the responsibility area of agent i. (ii) the responsibility area of agent i contains a tree vertex x from which starts the responsibility area of agent j (right side of Figure 5, i is blue and j is green). In this case, if a request is made in the responsibility area of agent i but on the other side of x than server i itself (i.e., in $\overline{T}_x(s_i)$), then both server i *and* server j are the disutility minimizing servers for this request.

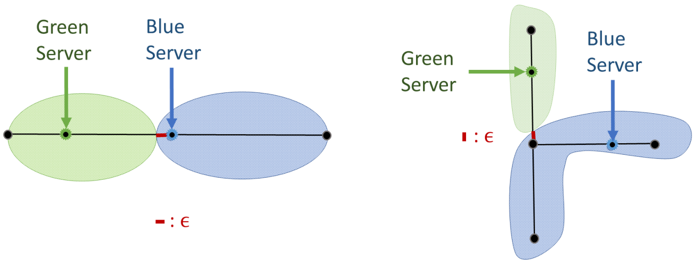


Figure 6 Modifying the regions for which the DC servers are responsible by pushing their boundaries away from real servers and tree vertices. This prevents indifference between different real servers except for isolated points. The boundaries are pushed by small amounts such that even their sum over all regions and all steps is arbitrarily small, thus having no effect on the competitive ratio. See Appendices A and B for the full argument, which uses a potential function.

To resolve this issue, we "nudge" the responsibility area of agent i slightly to the direction of the responsibility area of agent j by an exponentially decreasing tiny ϵ (see Figure 6). We inspect the proof of Lemma 3 to see why this does not change the competitive ratio. Since we do not necessarily use the server that minimizes the min cost matching at the nudged areas, Equation (8) does not hold if the request is in the nudged area. We notice though that this equation is violated by at most $k\epsilon$. To see this, we first move ON to the request. Using the same argument as in Lemma 3, we see that Equation (8) still holds after doing this.

We now move LAZY. Assume LAZY moves some server ℓ'. If the request would have been in the border between two responsibility areas before the nudge, then the cost of the min cost matching would have decreased by at least $\mathsf{dist}(s_{\ell'}, \sigma_t)$ and this would have paid for the cost of moving ℓ'. We notice that if the location of a request in DC moves by ϵ, the locations of all servers change by at most ϵ. Therefore, using the same matching in the nudged area as we would have used in the border before the nudge increases the cost of the min cost matching by at most $k\epsilon$. Hence, moving ℓ' decreases the cost of the min cost matching by at least $\mathsf{dist}(s_{\ell'}, \sigma_t) - k\epsilon$, violating Equation (8) by at most $k\epsilon$.

As we can let ϵ exponentially decay (say by a factor of two at each step t), summing Equation (8) for all t's yields that the cost of LAZY is at most $2k\epsilon$ larger than the cost of ON. As ϵ is arbitrarily small, so is the difference between LAZY and ON, which thus have the same competitive ratio.

C Implementation in Polynomial Time

Algorithm 1 as defined in Section 3 is continuous in the sense that every point is considered when deciding which set of points should be in the region R_i of some server i. In this secion, we show that one can discretize the metric space in a way that only polynomially many points (in the number of servers and vertices of the tree) are considered when determining the regions of each server.

Consider a point $p \in T$, such that there exist $1 \le i < j \le k$ such that

$$\mathsf{dc}_i(\sigma^{\prec t} \parallel p) = \mathsf{dc}_j(\sigma^{\prec t} \parallel p)$$

(where $\|$ denotes concatenation), then p is called a *boundary point*. That is, a boundary point is a point for which, if a request occurs in p, two DC servers will serve the request. Define the set of all boundary points for Double Cover just before event t arrives (see Fig. 3c in Appendix 5):

$$B^{\prec t} = \left\{p \mid \exists 1 \le i < j \le k \text{ such that } \mathsf{dc}_i(\sigma^{\prec t} \parallel p) = \mathsf{dc}_j(\sigma^{\prec t} \parallel p)\right\}.$$

▶ **Definition 24.** *Given a tree metric $T = (V, E, \mathsf{dist})$, a set of requests $\sigma^{\prec t}$, and the current locations of the servers $\mathsf{S}^{\prec t}$, we define the critical tree graph $T_c^{\prec t}$ by subdividing the edges of the tree (V, E) at all the server locations and boundary points, and retaining the distance function* dist*, see Fig. 3 in Appendix 5. Formally:*

- *Define the vertex set of the critical tree graph $T_c^{\prec t}$ to be the set $V_c^{\prec t}$, the union of the following point sets on the tree metric*
  - *Vertices of the tree T.*
  - *Server locations $\left\{\mathsf{S}_\ell^{\prec t}\right\}_{\ell=1,\dots,k}$.*
  - *The set of boundary points $B^{\prec t}$.*
- *The edge set of $T_c^{\prec t}$ is denoted by $E_c^{\prec t}$. There is an edge $(p, q) \in E_c^{\prec t}$ (where $p \in V_c^{\prec t}$ and $q \in V_c^{\prec t}$) if p and q lie along the same edge of T, and there is no intermediate point $r \in V_c^{\prec t}$ between them. The weight of the edge $(p, q) \in E_c^{\prec t}$ is the distance between p and q in the tree metric T.*

The intuition behind the critical graph is that the vertices of the graph are exactly the points in the metric space where the sets of valid colors ($\{\ell : p \text{ is } \ell\text{-colorable}\}$) change.

▶ **Lemma 25.** *Let $e = \{v_1, v_2\}$ be some edge of $T_c^{\prec t}$, and let ℓ be some server such that $v_1 \in \mathcal{P}[s_\ell, v_2]$ and v_1 is ℓ-colorable. The edge e is ℓ-colorable iff there exists some point p along the edge, excluding the endpoints, such that $\ell \in \mathsf{MC}(p)$.*

Proof. By definition, if e is ℓ-colorable, then for every p along the edge, p is ℓ-colorable, and therefore, $\ell \in \mathsf{MC}(p)$.

Now assume that there exists some p along the edge e such that $\ell \in \mathsf{MC}(p)$. Since there exists some min-cost matching such that s_ℓ is matched to the DC server that serves p, and since p cannot be a vertex of T, by Lemma 6,

$$|T_p(s_\ell) \cap \mathsf{S}| > |T_p(s_\ell) \cap \mathsf{DC}(p)|. \tag{9}$$

Since there are no servers and no tree vertices along edge e, for every point $q \in \mathcal{P}[v_1, v_2] \setminus \{v_1, v_2\}$,

$$|T_q(s_\ell)| = |T_p(s_\ell)|. \tag{10}$$

For a given $q \in \mathcal{P}[v_1, v_2] \setminus \{v_1, v_2\}$ let

$$d_1(q) = |T_q(v_1) \cap \mathsf{DC}(q)| \, (= |T_q(s_\ell) \cap \mathsf{DC}(q)|)$$

be the set of DC servers in the subtree containing v_1 when splitting T at point q *after* serving a request at q. Let i be the index of the DC server that serves all the requests along the edge e, excluding its endpoints (there must be a unique such DC server since there are no boundary points along e). Notice that for every $j \neq i$, $\mathcal{P}[\mathsf{dc}_j, \mathsf{dc}_j(q)] \cap \mathcal{P}[v_1, v_2] \setminus \{v_1, v_2\} = \emptyset$. Otherwise, there would have been a point q along e which is closer to server j than server i, which implies the existence of a boundary point along e.

Since there are no tree vertices along e, we get that for every $q, q' \in \mathcal{P}[v_1, v_2] \setminus \{v_1, v_2\}$, $d_1(q) = d_1(q')$. Therefore, for every such point q,

$$|T_q(s_\ell) \cap \mathsf{DC}(q)| = d_1(q) = d_1(p) = |T_p(s_\ell) \cap \mathsf{DC}(p)| . \tag{11}$$

Combining (9), (10) and (11) yields that for every $q \in \mathcal{P}[v_1, v_2] \setminus \{v_1, v_2\}$, $|T_q(s_\ell) \cap \mathsf{S}| > |T_q(s_\ell) \cap \mathsf{DC}(q)|$. Therefore, $\left|\overline{T}_q(s_\ell) \cap \mathsf{S}\right| < \left|\overline{T}_q(s_\ell) \cap \mathsf{DC}(q)\right|$, and there exists some point $q' \in \mathcal{P}[q, v_2]$ such that

$$\left|\overline{T}_{q'}(s_\ell) \cap \mathsf{S}\right| \leq \left|\overline{T}_{q'}(s_\ell) \cap \mathsf{DC}(q)\right| \Rightarrow \left|\overline{T}_{q'}(q) \cap \mathsf{S}\right| \leq \left|\overline{T}_{q'}(q) \cap \mathsf{DC}(q)\right| .$$

Since there are no servers in $\mathcal{P}[p, v_2]$ (there are no servers along every edge e of $T_c^{\prec t}$), for every server j such that $s_j \in T_q(v_2)$, q' is on the path from s_j to q, and by Lemma 6, $j \notin \mathsf{MC}(q)$. By definition, this implies that for every point q along edge e, and every j such that $s_j \in \overline{T}_{v_2}(q)$, q is not j-colorable. Since by Lemma 17 every point is colorable by some server, we get that for every q along e, q is ℓ'-colorable by some server ℓ' such that $s_{\ell'} \in \overline{T}_q(v_2) \Rightarrow s_{\ell'} \in \overline{T}_{v_1}(v_2)$. By Lemma 21, since v_1 is ℓ-colorable, we get that every q along the edge e is ℓ-colorable, which implies that e is ℓ-colorable, as desired. ◀

▶ **Lemma 26.** *Let e be some edge $\{v, v'\} \in E_c^{\prec t}$ such that $\mathsf{color}(v) = j$ and $\mathsf{color}(v') = j'$. There exists $i \in \{j, j'\}$ such that all points in $\mathcal{P}[v, v'] \setminus \{v, v'\}$ are i-colorable which can be determined by inspecting a single point in $\mathcal{P}[v, v'] \setminus \{v, v'\}$.*

Proof. consider some edge $e = \{v, v'\} \in E_c^{\prec t}$ such that $\mathsf{color}(v) = j$ and $\mathsf{color}(v') = j'$. Let p be a point between v and v'. By Lemma 17, it is colorable by some server ℓ. Since there are no servers along x, ℓ must be located either in $\overline{T}_v(p)$ or in $\overline{T}_{v'}(p)$. Assume without loss of generality that $\ell \in \overline{T}_v(p)$. By Lemma 21, p is j-colorable, which implies that $j \in \mathsf{MC}(p)$. By Lemma 25, x is j-colorable. ◀

▶ **Lemma 27.** *Determining R_i at every iteration i in Step 1b of Algorithm 1 can be done in polynomial time.*

Proof. Consider the graph $T_c^{\prec t}$. This graph has at most $2k - 1 + |V|$ vertices — k servers, at most $k - 1$ boundary points, and $|V|$ original vertices. The boundary points can of course be computed in polynomial time. Consider iteration i of Step 1b of Algorithm 1. To determine R_i, one can start at s_i, which is obviously in R_i, and then expend R_i using any tree traversal algorithm (that runs in linear time) on $T_c^{\prec t}$. The traversal does not go further down the tree if the vertex/edge currently considered is not i-colorable.

To check if a point $r \in T$ is i-colorable can be done in poly-time: Property 1 of Definition 12 can easily be checked. As for properties 2 and 3, Computing $\mathsf{MC}(r)$ can be done in poly-time using the characterization in Lemma 6. Therefore, property 2 can immediately be checked. For Property 3, one should consider each server $j \in \mathsf{MC}(r)$, and check that $j \not\prec_r i$, which again can be done in poly-time.

From the above, it is clear that determining whether a vertex in $T_c^{\prec t}$ is i-colorable can be done in poly-time. As for an edge, by Lemma 26, checking whether the edge is i-colorable can be done by inspecting an arbitrary point in the edge, and checking whether this point is i-colorable, which again, can be done in poly-time. Therefore, the tree-traversal can be made in poly-time, and so does determining R_i. ◀

D Missing Proofs of Section 4

Proof of Lemma 6. $\Leftarrow$: Let $p \in P$ and $q \in Q$ be two points such that there exists a point $x \in \mathcal{P}(p,q)$ such that $|\overline{T}_x(q) \cap P| \leq |\overline{T}_x(q) \cap Q|$ and let $\mathcal{M} : P \to Q$ be a matching such that $\mathcal{M}(p) = q$. Since p is matched to a server in $T_x(q)$, $|\overline{T}_x(q) \cap P - \{p\}| < |\overline{T}_x(q) \cap Q|$, and there must be a server $\hat{p} \in T_x(q) \cap P$ that is matched to a server $\hat{q} \in \overline{T}_x(q) \cap Q$. Let $y = \mathsf{LCA}_x(\hat{p}, q)$. Since $\hat{p}$ and q are both in $T_x(q)$, $y \neq x$. Consider the matching $\mathcal{M}'$ in which p is matched to $\hat{q}$, $\hat{p}$ is matched to q, and for every $\tilde{p} \in P \setminus \{p, \hat{p}\}$, $\mathcal{M}'(\tilde{p}) = \mathcal{M}(\tilde{p})$. We have

$$\begin{aligned}
\mathsf{dist}(p,q) + \mathsf{dist}(\hat{p},\hat{q}) &= \mathsf{dist}(p,x) + \mathsf{dist}(x,y) + \mathsf{dist}(y,q) + \\
&\quad\ \mathsf{dist}(\hat{p},y) + \mathsf{dist}(y,x) + \mathsf{dist}(x,\hat{q}) \\
&> \mathsf{dist}(p,x) + \mathsf{dist}(x,\hat{q}) + \mathsf{dist}(\hat{p},y) + \mathsf{dist}(y,q) \\
&\geq \mathsf{dist}(p,\hat{q}) + \mathsf{dist}(\hat{p},q),
\end{aligned}$$

where that first equality is due to the fact that the path from x to y is contained in both the path from p to q and the path from $\hat{q}$ to $\hat{p}$, the first strict inequality is due to dropping non-zero terms, and the last inequality follows from the triangle inequality. Therefore, $\mathcal{M}'$ is a matching of a strictly smaller cost than that of $\mathcal{M}$, and $\mathcal{M}$ cannot be a min-cost matching.

$\Rightarrow$: Assume that the condition holds for p, q, let $\mathcal{M}$ be a matching. Let $x = \mathsf{LCA}_q(p, \mathcal{M}(p))$.

Case 1. $x \neq q$, therefore $|\overline{T}_x(q) \cap P| > |\overline{T}_x(q) \cap Q|$. Hence, there exists $\hat{p} \in \overline{T}_x(q)$ s.t. $\mathcal{M}(\hat{p}) \notin \overline{T}_x(q)$. Let $\hat{q} = \mathcal{M}(\hat{p})$, and $q' = \mathcal{M}(p)$. Note that $\mathsf{dist}(p,q') = \mathsf{dist}(p,x) + \mathsf{dist}(x,q')$ and $\mathsf{dist}(\hat{p},\hat{q}) = \mathsf{dist}(\hat{p},x) + \mathsf{dist}(x,\hat{q})$. Consider the matching $\mathcal{M}'$ in which p is matched to $\hat{q}$, $\hat{p}$ is matched to q', and for every $\tilde{p} \in P \setminus \{p, \hat{p}\}$, $\mathcal{M}'(\tilde{p}) = \mathcal{M}(\tilde{p})$.

$$\begin{aligned}
\mathsf{dist}(p,\hat{q}) + \mathsf{dist}(\hat{p},q') &\leq \mathsf{dist}(p,x) + \mathsf{dist}(x,\hat{q}) + \mathsf{dist}(\hat{p},x) + \mathsf{dist}(x,q') \\
&= \mathsf{dist}(p,q') + \mathsf{dist}(\hat{p},\hat{q}),
\end{aligned}$$

where the inequality is by the triangle inequality. Therefore, $\mathcal{M}'$ is also a min-cost matching. Let $x' = \mathsf{LCA}_q(p, \mathcal{M}'(p))$ then $\mathsf{dist}(p,x') > \mathsf{dist}(p,x)$ since $x' \notin \overline{T}_x(q)$, therefore we can repeat this process until $x = q$ (**Case 2**).

Case 2. $x = q$, hence $\mathcal{P}(p,q) \subseteq \mathcal{P}(p, \mathcal{M}(p))$. Let $\hat{q} = \mathcal{M}(p)$ and let $\hat{p}$ be such that $q = \mathcal{M}(\hat{p})$. Consider the matching $\mathcal{M}'$ in which p is matched to q, $\hat{p}$ is matched to $\hat{q}$, and for every $\tilde{p} \in P \setminus \{p, \hat{p}\}$, $\mathcal{M}'(\tilde{p}) = \mathcal{M}(\tilde{p})$.

$$\begin{aligned}
\mathsf{dist}(p,q) + \mathsf{dist}(\hat{p},\hat{q}) &= \mathsf{dist}(p,\hat{q}) - \mathsf{dist}(q,\hat{q}) + \mathsf{dist}(\hat{p},\hat{q}) \\
&\leq \mathsf{dist}(p,\hat{q}) + \mathsf{dist}(\hat{p},q)
\end{aligned}$$

where the last inequality is by the triangle inequality. Therefore, $\mathcal{M}'$ is also min cost matching and $\mathcal{M}'(p) = q$ as needed. ◀

Proof of Lemma 7. Let v be the closest vertex to r in $T_r(q)$ (recall that $r \notin T_r(q)$, so $v \neq r$). If there exists $p \in \mathcal{P}[v,r) \cap P$, let $p \in \mathcal{P}[v,r) \cap P$ be the closest such point to r. In this case, the condition holds for p since for all $x \in \mathcal{P}(p,r)$, $\overline{T}_x(r) \cap P = T_r(q) \cap P$.

If there is no such p, then

$$\left|(\overline{T}_v(r) - \{v\}) \cap P\right| = |T_r(q) \cap P| > |T_r(q) \cap Q| \geq \left|(\overline{T}_v(r) - \{v\}) \cap Q\right|.$$

By the pigeonhole principle, there exists $v' \in \overline{T}_v(r)$ such that $|T_v(v') \cap P| > |T_v(v') \cap Q|$. Therefore, by repeating above process, we find $\hat{p} \in P \cap T_v(v')$ for which the condition holds for all $x \in \mathcal{P}(\hat{p}, v)$. Since the condition holds for every $x \in \mathcal{P}(v,r)$ (as $\overline{T}_x(r) \cap P = T_r(q) \cap P$), the lemma follows. ◀

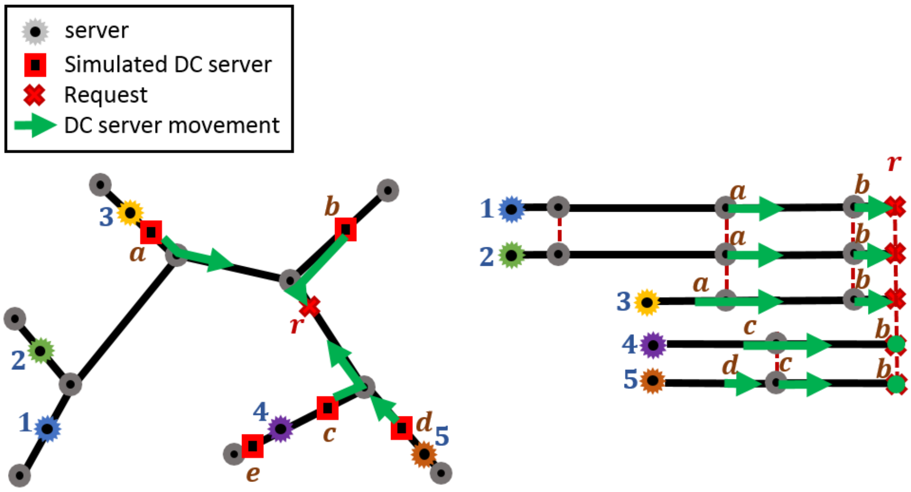


Figure 2 Servers and DC servers are denoted by numbers and letters respectively. Points on the tree are said to be colorable by some set of servers. Colorability of a point r is determined by simulating the double cover (DC) algorithm for a request at r. When DC processes a request, multiple DC servers move towards the request, and one or more arrive to serve it. Imagine a server were to look along the tree towards r when the DC servers were in motion in response to a request at r. Such a server may see a trail left by (at most one) DC server in motion towards r. Different servers may see trails of different DC servers. Two servers see the same trails beyond (above) their lowest common ancestor (when the tree is rooted at r) but for a DC server that traverses their lowest common ancestor, they may observe different trails. We say that server i has higher priority than server j with respect to r, if the trail of the DC server that traverses the lowest common ancestor of i and j is contained in the trail seen by server j (of the same DC server). On the left the movement of the DC servers relative to the real server positions is depicted. On the right, all paths from real servers to r are depicted, with dashed lines indicating vertices seen by more than one real server. In this example, $1 \succ_r 3$ since that trail that server 1 sees of DC server a is contained in the trail that server 3 sees of DC server a. Similarly, $2 \succ_r 3$ (because of a), $5 \succ_r 4$ (because of c), and $4, 5 \succ_r 1, 2, 3$ (because of b). Notice that $\succ_r$ is not defined for all pairs of servers; For example, both $1 \not\succ_r 2$ and $2 \not\succ_r 1$. Subsequent to the motion of the DC servers, there are several min cost matching between real servers and DC servers. In one such matching server 1 is matched to server b, in another such min matching server 2 is matched to server b, in a third such min matching server 3 is matched to server b. Therefore, $\mathsf{MC}(r) = \{1, 2, 3\}$. Since $1 \not\succ_r 2$, $3 \not\succ_r 2$, $2 \not\succ_r 1$ and $3 \not\succ_r 1$. We get that r is $1, 2$-colorable. r is not 3-colorable since $1 \succ_r 3$.