# Online $k$-Taxi via Double Coverage and Time-Reverse Primal-Dual[⋆]

Niv Buchbinder[1], Christian Coester[2], and Joseph (Seffi) Naor[3]

[1] Tel Aviv University, Israel
`niv.buchbinder@gmail.com`
[2] CWI, Amsterdam, Netherlands
`christian.coester@cwi.nl`
[3] Computer Science Department, Technion, Israel
`naor@cs.technion.ac.il`

**Abstract.** We consider the online $k$-taxi problem, a generalization of the $k$-server problem, in which $k$ servers are located in a metric space. A sequence of requests is revealed one by one, where each request is a pair of two points, representing the start and destination of a travel request by a passenger. The goal is to serve all requests while minimizing the distance traveled *without carrying a passenger*.

We show that the classic *Double Coverage* algorithm has competitive ratio $2^k - 1$ on HSTs, matching a recent lower bound for deterministic algorithms. For bounded depth HSTs, the competitive ratio turns out to be much better and we obtain tight bounds. When the depth is $d \ll k$, these bounds are approximately $k^d/d!$. By standard embedding results, we obtain a randomized algorithm for arbitrary $n$-point metrics with (polynomial) competitive ratio $O(k^c \Delta^{1/c} \log_\Delta n)$, where $\Delta$ is the aspect ratio and $c \geq 1$ is an arbitrary positive integer constant. The only previous known bound was $O(2^k \log n)$. For general (weighted) tree metrics, we prove the competitive ratio of Double Coverage to be $\Theta(k^d)$ for any fixed depth $d$, but unlike on HSTs it is not bounded by $2^k - 1$.

We obtain our results by a dual fitting analysis where the dual solution is constructed step-by-step *backwards* in time. Unlike the forward-time approach typical of online primal-dual analyses, this allows us to combine information from the past and the future when assigning dual variables. We believe this method can be useful also for other problems. Using this technique, we also provide a dual fitting proof of the $k$-competitiveness of Double Coverage for the $k$-server problem on trees.

**Keywords:** online algorithms · $k$-taxi · $k$-server · dual fitting.

## 1 Introduction

The $k$-taxi problem, proposed three decades ago as a natural generalization of the $k$-server problem by Fiat et al. [13], has gained renewed interest recently. In

---

[⋆] This research was supported in part by US-Israel BSF grant 2018352, by ISF grant 2233/19 (2027511) and by NWO VICI grant 639.023.812.

this problem there are $k$ servers, or taxis, which are located in a metric space containing $n$ points. A sequence of requests is revealed one by one to an online algorithm, where each request is a pair of two points, representing the start and destination of a travel request by a passenger. An online algorithm must serve each request (by selecting a server that travels first to its start and then its destination) without knowledge of future requests. The goal is to minimize the total distance traveled by the servers *without carrying a passenger*. The motivation for not taking into account the distance the servers travel with a passenger is that any algorithm needs to travel from the start to the destination, independently of the algorithm's decisions. Thus, the $k$-taxi problem seeks to only minimize the overhead travel that depends on the algorithm's decisions. While this does not affect the optimal (offline) assignment, it affects the competitive factor.

Besides scheduling taxi rides, the $k$-taxi problem also models tasks such as scheduling elevators (the metric space is the line), and other applications where objects need to be transported between locations.

The extensively studied and influential $k$-server problem is the special case of the $k$-taxi problem where for each request, the start equals the destination. A classical algorithm for the $k$-server problem on tree metrics is DoubleCoverage. This algorithm is described as follows. A server $s$ is called *unobstructed* if there is no other server on the unique path from $s$ to the current request. To serve the request, DoubleCoverage moves *all* unobstructed servers towards the request at equal speed, until one of them reaches the request. If a server becomes obstructed during this process, it stops while the others keep moving.

DoubleCoverage was originally proposed for the line metric, to which it owes its name, as there are at most two servers moving at once. For a line metric it achieves the optimal competitive ratio of $k$ [7], and this result was later generalized to tree metrics [8].

Given the simplicity and elegance of DoubleCoverage, it is only natural to analyze its performance for the $k$-taxi problem. Here, we use it only for bringing a server to the start vertex of a request.

### 1.1 Related Work and Known Results

For the $k$-server problem, the best known deterministic competitive factor on general metrics is $2k-1$ [17]; with randomization, on hierarchically well-separated trees (HSTs)[4] the best known bound is $O(\log^2 k)$ [4,5]. By a standard embedding argument, this implies a bound of $O(\log^2 k \log n)$ for $n$-point metrics, and it was also shown in [4] that a dynamic embedding yields a bound of $O(\log^3 k \log \Delta)$ for metrics with aspect ratio $\Delta$. In [18], a more involved dynamic embedding was proposed to achieve a polylog($k$)-competitive algorithm for general metrics.[5] Contrast these upper bounds with the known deterministic lower bound of $k$ [21] and the randomized lower bound of $\Omega(\log k)$ [12]. More information about the $k$-server problem can be found in [16].

---

[4] See Section 2 for an exact definition of HSTs.

[5] There is a gap in the version posted to the arXiv on February 21, 2018 [19,20].

Surprisingly, until recently very little was known about the $k$-taxi problem, in contrast to the extensive work on the $k$-server problem. Coester and Koutsoupias [9] provided a $(2^k - 1)$-competitive memoryless randomized algorithm for the $k$-taxi problem on HSTs against an adaptive online adversary. This result implies: (i) the existence of a $4^k$-competitive deterministic algorithm for HSTs via a known reduction [3], although this argument is non-constructive; (ii) an $O(2^k \log n)$-competitive randomized algorithm for general metric spaces (against an oblivious adversary). Both bounds currently constitute the state of the art. Coester and Koutsoupias also provided a lower bound of $2^k - 1$ on the competitive factor of any deterministic algorithm for the $k$-taxi problem on HSTs, thus proving that the problem is substantially harder than the $k$-server problem. However, large gaps still remain in our understanding of the $k$-taxi problem, and many problems remain open in both deterministic and randomized settings. For general metrics, an algorithm with competitive factor depending only on $k$ is known only if $k = 2$, and for the line metric only if $k \leq 3$ [9]. Both of these algorithms can be viewed as variants of DoubleCoverage.

The version of the problem where the start-to-destination distances also contribute to the objective function was called the "easy" $k$-taxi problem in [15,9], whereas the version we are considering here is the "hard" $k$-taxi problem. The easy version has the same competitive factor as the $k$-server problem [9]. The $k$-taxi problem was recently reintroduced as the Uber problem in [10], who studied the easy version in a stochastic setting.

### 1.2 Our Contribution

We provide the following bounds on the competitive ratio of DoubleCoverage for the $k$-taxi problem.

**Theorem 1.** *The competitive ratio of* DoubleCoverage *for the $k$-taxi problem is at most*

*(a)* $\left(c_{kd} = \sum_{h=1}^{\min\{k,d\}} \binom{k}{h}\right)$ *on HSTs of depth $d$.*
*(b)* $O(k^d)$*-competitive on general (weighted) tree metrics of depth $d$.*

We complement these upper bounds by the following lower bounds:

**Theorem 2.** *The competitive ratio of* DoubleCoverage *for the $k$-taxi problem is at least*

*(a)* $\left(c_{kd} = \sum_{h=1}^{\min\{k,d\}} \binom{k}{h}\right)$ *on HSTs of depth $d$.*
*(b)* $\Omega(k^d)$ *on (even unweighted) tree metrics of constant depth $d$.*

When the depth $d$ of the HST is at least $k$, the upper bound $c_{kd} = 2^k - 1$ exactly matches the lower bound of [9] that holds even for randomized algorithms against an adaptive online adversary. Note that for fixed $d$, $c_{kd}$ is roughly $k^d/d!$ up to a multiplicative error that tends to 1 as $k \to \infty$. The $\Omega(k^d)$ lower bound on general trees is hiding a constant factor that depends on $d$. Since the root in general trees can be chosen arbitrarily, $d$ is essentially half the hop-diameter.

By well-known embedding techniques of general metrics into HSTs [2,11], slightly adapted to HSTs of bounded depth (see Theorem 5 in Section 2), we obtain the following result for general metrics.

**Corollary 1.** *There is a randomized* $O(k^c \Delta^{1/c} \log_\Delta n)$*-competitive algorithm for the* $k$*-taxi problem for every* $n$*-point metric, where* $\Delta$ *is the aspect ratio of the metric, and* $c \geq 1$ *is an arbitrary positive integer. In particular, setting* $c = \left\lceil \sqrt{\frac{\log \Delta}{\log k}} \right\rceil$*, the competitiveness is* $2^{O\left(\sqrt{\log k \log \Delta}\right)} \log_\Delta n$*.*

Compared to the $O(2^k \log n)$ upper bound of [9], our bound has only a polynomial dependence on $k$ at the expense of some dependence on the aspect ratio. Since $c_{kd} \leq 2^k - 1$ for all $d$, we still recover the same $O(2^k \log n)$ competitive factor. The bounds in Corollary 1 actually hide another division by $(c-1)!$ if $c \leq k$. Therefore, whenever $\Delta$ is at most $2^{O(k^2)}$, our bound results in an improvement.

**Techniques.** For the $k$-server problem, there exists a simple potential function analysis of DoubleCoverage. The potential value depends on the relative distances of the server locations, which, in the $k$-taxi problem, can change arbitrarily by relocation requests even though the algorithm does not incur any cost. Therefore, such a potential cannot work for the $k$-taxi problem. In [9], the $2^k - 1$ upper bound for the randomized HST algorithm is proved via a potential function that is $2^k - 1$ times the minimum matching between the online and offline servers. As is stated there, the same potential can be used to obtain the same bound for DoubleCoverage when $k = 2$, but it fails already when $k = 3$. Nonetheless, they conjectured that DoubleCoverage achieves the competitive ratio of $2^k - 1$ on HSTs.

We are able to prove that this is the case (and give the more refined bound of $c_{kd}$) with a primal-dual approach (which still uses an auxiliary potential function as well). The primal solution is the output of the DoubleCoverage algorithm. A dual solution is constructed to provide a lower bound on the optimal cost. The typical way a dual solution is constructed in the online primal-dual framework is forward in time, step-by-step, along with the decisions of the online algorithm (see e.g. [6,14,1]). By showing that the objective values of the constructed primal and dual solutions are within a factor $c$ of each other, one gets that the primal solution is $c$-competitive and the dual solution is $1/c$-competitive. For the LP formulation of the $k$-taxi problem we consider, we show that a pure forward-time approach (producing a dual solution as well) is doomed to fail:

**Theorem 3.** *There exists no competitive online algorithm for the dual problem of the* $k$*-taxi LP as defined in Section 3, even for* $k = 1$*.*

Our main conceptual contribution is a novel way to overcome this problem by constructing the dual solution backwards in time. Our assignment of dual variables for time $t$ combines knowledge about both the future *and* the past: It incorporates knowledge about the future simply due to the time-reversal; knowledge about the past is also used because the dual assignments are guided by the movement of DoubleCoverage, which is a forward-time (online) algorithm. Our method can be seen as a restricted form of online dual fitting, which is more "local", and hence easier to analyze step by step, similarly to primal-dual algorithms. We believe that this time-reversed method of constructing a dual solution may be useful for analyzing additional online problems, especially when information about the future helps to construct better dual solutions. Using this technique, we also provide a primal-dual proof of the $k$-competitiveness of DoubleCoverage for the $k$-server problem on trees. To the best of our knowledge, a primal-dual proof of this classical result was not known before.

**Theorem 4 ([8]).** DoubleCoverage *is $k$-competitive for the $k$-server problem on trees.*

Due to space constraints, most proofs are omitted from this extended abstract and we focus on showing the upper bound for $k$-taxi on HSTs (Theorem 1(a)).

## 2 Preliminaries

**The $k$-Taxi Problem.** The $k$-taxi problem is formally defined as follows. We are given a metric space with point set $V$, where $|V| = n$. Initially, $k$ taxis, or servers, are located at points of $V$. At each time $t$ we get a request $(s_t, d_t)$, where $s_t, d_t \in V$. To serve the request, one of the servers must move to $s_t$ and the cost paid by the algorithm is the distance traveled by the server. Then, one of the servers from $s_t$ is relocated to the point $d_t$. There is no cost for relocating the server from $s_t$ to $d_t$. The goal is to minimize the cost.

Without loss of generality, we can split each request $(s_t, d_t)$ into two requests: a *simple* request and a *relocation* request. Thus, at each time $t$, request $(s_t, d_t)$ is either one of the following:

- *Simple request ($s_t = d_t$)*: a server needs to move to $s_t$, if there is no server there already. The cost is the distance traveled by the server.
- *Relocation request ($s_t = d_{t-1}$)*: a server is relocated from $s_t$ to $d_t$. There is no relocation cost.

We can then partition the time horizon into two sets $T_s, T_r$ (odd and even times). For times in $T_s = \{1, 3, 5, \ldots, 2T-1\}$ we have simple requests, and for times in $T_r = \{2, 4, 6, \ldots, 2T\}$ we have relocation requests. The $k$-server problem is the special case of the $k$-taxi problem without relocation requests.

**Trees and HSTs.** Consider now a tree $\mathbb{T} = (V, E)$ and let $\mathbb{r}$ denote its root. There is a positive weight function defined over the edge set $E$, and without loss of generality all edge weights are integral. The *distance* between vertices $u$ and $v$ is the sum of the weights of the edges on the (unique) path between them in $\mathbb{T}$, which induces a metric. The *combinatorial depth* of a vertex $v \in V$ is defined to be the number of edges on the path from $\mathbb{r}$ to $v$. The combinatorial depth of $\mathbb{T}$ is the maximum combinatorial depth among all vertices. At times it will be convenient to assume that all edges in $E$ have unit length by breaking edges into unit length parts called *short edges*. We then refer to the original edges of $\mathbb{T}$ as *long edges*. However, the combinatorial depth of $\mathbb{T}$ is still defined in terms of the long edges. We define the *weighted depth* of a vertex $u$ as the number of *short* edges on the path from $\mathbb{r}$ to $u$. For $u \in V$, let $V_u \subseteq V$ be the vertices of the subtree rooted at $u$. In trees where all leaves are at the same weighted/combinatorial depth (namely, HSTs, see below), we define the *weighted/combinatorial height* of a vertex $u$ as the number of short/long edges on the path from $u$ to any leaf in $V_u$. We let $v \prec u$ denote that $v$ is a child of $u$, and let $p(u)$ denote the parent of $u$.

*Hierarchically well-separated trees (HSTs)*, introduced by Bartal [2], are special trees that can be used to approximate arbitrary finite metrics. For $\alpha \geq 1$, an $\alpha$-HST is a tree where every leaf is at the same combinatorial depth $d$ and the edge weights along any root-to-leaf path decrease by a factor $\alpha$ in each step. The associated metric space of the HST is only the set of its leaves. Hence, for the $k$-taxi problem on HSTs, the requested points $s_t$ and $d_t$ are always leaves. Any $n$-point metric space can be embedded into a random $\alpha$-HST such that (i) the distance between any two points can only be larger in the HST and (ii) the expected blow-up of each distance is $O(\alpha \log_\alpha n)$ [11]. The latter quantity is also called the *distortion* of the embedding. The depth of the random HST constructed in the embedding is at most $\lceil \log_\alpha \Delta \rceil$, where $\Delta$ is the aspect ratio, i.e., the ratio between the longest and shortest non-zero distance. Choosing $\alpha = \Delta^{1/d}$, we obtain an HST of depth $d$ and with distortion $O\left(d\Delta^{1/d} \log_\Delta n\right)$.

**Theorem 5 (Corollary to [11]).** *Any metric with $n$ points and aspect ratio $\Delta$ can be embedded into a random HST of combinatorial depth $d$ with distortion $O\left(d\Delta^{1/d} \log_\Delta n\right)$.*

**The Double Coverage Algorithm.** We define DoubleCoverage in a way that will suit our definition of short edges of length 1 later. Consider the arrival of a simple request at location $s_t$. A server located at vertex $v$ is *unobstructed* if there are no other servers on the path between $v$ and $s_t$. If other servers on this path exist only at $v$, we consider only one of them unobstructed (chosen arbitrarily). Serving the request is done in several small steps, as follows:

```
DoubleCoverage (upon a simple request at $s_t$):
While no server is at $s_t$: all currently unobstructed servers move distance 1
towards $s_t$.
```

Upon a relocation request $(s_t, d_t)$, we simply relocate a server from $s_t$ to $d_t$.

For a given small step (i.e., iteration of the while-loop), we denote by $U$ and $B$ the sets of servers moving towards the root (**u**pwards in the tree) and away from the root (towards the **b**ottom of the tree), respectively.

**Observation 1.** In any small step:

- $B$ is either a singleton ($B = \{j\}$ for a server $j$) or empty ($B = \emptyset$).
- The subtrees rooted at servers of $U$ are disjoint and do not contain $s_t$. If $B = \{j\}$, then these subtrees and $s_t$ are inside the subtree rooted at $j$.

## 3 LP Formulations

We formulate a linear program (LP) for the $k$-taxi problem along with its dual that we use for the purpose of analysis. We assume for ease of exposition that all edges are short edges. As already mentioned, when considering the $k$-taxi problem on HSTs, requests appear only at the leaves. It is easy to see that in this case the upward movement cost (i.e., movement towards the root) is the same as the downward movement cost, up to an additive error of $k$ times the distance from the root to any leaf. The same is true of the $k$-server problem on general trees (but not for the $k$-taxi problem on general trees). Hence, for the $k$-taxi problem on HSTs and for the $k$-server problem, we can use an LP that only takes into account the upward movement cost. (A slightly different LP is needed for the $k$-taxi problem on general trees.)

The LP below is a relaxation of the problem as it allows for fractional values of the variables. For $u \in V$, let variable $x_{ut}$ denote the number of servers in $V_u$ after the request at time $t$ has been served. Variable $y_{ut} \geq 0$ denotes the number of servers that left subtree $V_u$ (moving upwards) at time $t$. For $u = \mathbb{r}$, $x_{\mathbb{r}t}$ is defined to be the *constant* $k$. (It is not a variable.) The primal LP is the following:

$$
\begin{aligned}
\min \quad & \sum_{t \in T_s} \sum_{u \neq \mathbb{r}} y_{ut} \\
& x_{ut} \geq \mathbb{1}_{\{u = s_t \text{ and } t \in T_s\}} + \sum_{v < u} x_{vt} && \forall u \in V, t \in T_s \cup T_r \\
& y_{ut} \geq x_{u,t-1} - x_{ut} && \forall u \neq \mathbb{r}, t \in T_s \\
& x_{ut} = x_{u,t-1} + \xi_{ut} && \forall u \neq \mathbb{r}, t \in T_r \\
& y_{ut} \geq 0 && \forall u \neq \mathbb{r}, t \in T_s,
\end{aligned}
$$

where

$$
\xi_{ut} := \begin{cases} -1 & \text{if } s_t \in V_u, d_t \notin V_u \\ 1 & \text{if } s_t \notin V_u, d_t \in V_u \\ 0 & \text{otherwise.} \end{cases}
$$

For technical reasons, we will add the additional constraint

$$
0 = x_{u,2T} - \bar{x}_{u,2T} \qquad \forall u \neq \mathbb{r}
$$

to the primal LP, where $\bar{x}_{u,2T}$ are *constants* specifying the configuration of DoubleCoverage at the last time step. Clearly, this affects the optimal value by only an additive constant. We will also view $x_{u0} = \bar{x}_{u0}$ as constants describing the initial configuration of the servers. The corresponding dual LP is the following.

$$\max \sum_{t\in T_s} \lambda_{s_t t} + \sum_{t\in T_r}\sum_{u\neq \mathbb{r}} \xi_{ut} b_{u,t-1} - \sum_{t\in T_s\cup T_r} k\lambda_{\mathbb{r}t} + \sum_{u\neq \mathbb{r}} \left(\bar{x}_{u0} b_{u0} - \bar{x}_{u,2T} b_{u,2T}\right)$$

$$\begin{aligned} \lambda_{ut} - \lambda_{p(u)t} &= b_{ut} - b_{u,t-1} & \qquad \forall u \neq \mathbb{r}, t \in T_s \cup T_r \\ b_{u,t-1} &\in [0,1] & \forall u \neq \mathbb{r}, t \in T_s \\ \lambda_{ut} &\geq 0 & \forall u \in V, t \in T_s \cup T_r \end{aligned}$$

We can use the same primal and dual formulation for the $k$-server problem, except that the set $T_r$ is then empty.[6]

### 3.1 Dual Transformation

The dual LP is not very intuitive. By a transformation of variables, we get a simpler equivalent dual LP, which we can interpret as building a mountain structure on the tree over time. This new dual LP has only one variable $A_{ut}$ for each vertex $u$ and time $t$. We interptet $A_{ut}$ as the *altitude* of $u$ at time $t$. We denote by $\Delta_t A_u = A_{ut} - A_{u,t-1}$ the change of altitude of vertex $u$ at time $t$. For a server $i$ of DoubleCoverage, we denote by $v_{it}$ its location at time $t$, and define similarly $\Delta_t A_i := A_{v_{it}t} - A_{v_{i,t-1}t-1}$ as the change of altitude of server $i$ at time $t$. The new dual LP is the following:

$$\max \quad \sum_{t\in T_s}\left[\Delta_t A_{s_t} - \sum_{i=1}^{k} \Delta_t A_i\right] - \sum_{t\in T_r}\sum_{i=1}^{k} \Delta_t A_{v_{it}}$$

$$A_{ut} - A_{p(u)t} \in [0,1] \qquad \forall u \neq \mathbb{r}, t+1 \in T_s \tag{1}$$

$$\Delta_t A_u \geq 0 \qquad \forall u \in V, t \in T_s \cup T_r \tag{2}$$

The constraints of the LP stipulate that altitudes are non-decreasing over time and (at time steps before a simple request) along root-to-leaf paths, with the difference in altitude of two adjacent nodes being at most 1. The objective function measures changes in the altitudes of request and server locations.

We define

$$D_t := \begin{cases} \Delta_t A_{s_t} - \sum_{i=1}^{k} \Delta_t A_i & t \in T_s \\ -\sum_{i=1}^{k} \Delta_t A_{v_{it}} & t \in T_r, \end{cases} \tag{3}$$

so that the dual objective function is equal to $D := \sum_{t\in T_s\cup T_r} D_t$.

The following lemma allows us to use this new LP for our analyses.

---

[6] We note that our LP for the $k$-server problem is different from LPs used in the context of polylogarithmically-competitive randomized algorithms for the $k$-server problem. In our context of deterministic algorithms for $k$-taxi (and $k$-server), we show that we can work with this simpler formulation.

**Lemma 1.** *The two dual LPs are equivalent. That is, any feasible solution to one of them can be translated (online) to a feasible solution to the other with the same objective function value.*

A proof of this lemma is given in the full version of our paper. It is based on a transformation of variables satisfying

$$\begin{aligned} A_{ut} - A_{p(u)t} &= b_{ut} \\ \Delta_t A_u &= \lambda_{ut}. \end{aligned}$$

Moreover, the comparably simple dual objective function is obtained by expanding the terms $\bar{x}_{u0} b_{u0} - \bar{x}_{u,2T} b_{u,2T}$ in the objective of the original dual to a telescoping sum.

## 4 The $k$-taxi Problem on HSTs

In this section we analyze DoubleCoverage on HSTs, proving Theorem 1(a).

Besides constructing a dual solution, our analysis will also employ a potential function $\Psi$ (that depends on the state of the system). The choice of $\Psi$ will be the only difference between the analyses of the $k$-server and $k$-taxi problems. The dual solution and potential will be such that for all $t \in T_s \cup T_r$,

$$cost_t^{\uparrow} + \Psi_t - \Psi_{t-1} \leq c \cdot D_t, \tag{4}$$

where $cost_t^{\uparrow}$ is the cost of movement *towards the root* by DoubleCoverage's servers while serving the $t$th request, $c$ is the desired competitive ratio, and $D_t$ is the increase of the dual objective function at time $t$, as given by (3). As discussed in Section 2 we may use in this case the dual of the program that only measures movement cost towards the root. Thus, summing (4) for all times will then imply that DoubleCoverage is $c$-competitive.

Recall that for a simple request ($t \in T_s$), DoubleCoverage breaks the movement of the servers into small steps in which the servers in $U \cup B$ move distance 1 towards the request. We will break the construction of a dual solution into these same small steps. We will denote by $\Delta\Psi$ the change of $\Psi$ during the step and by $\Delta D$ the contribution of the step to $D_t$. The cost paid by the servers (for moving towards the root) in the step is $|U|$. Using this notation, we satisfy (4) for simple requests if we show for each step that:

$$|U| + \Delta\Psi \leq c \cdot \Delta D. \tag{5}$$

In Section 4.1 we describe how we construct the dual solution by going backwards in time. We also mention a simple potential function to prove the $k$-competitiveness of $k$-server on trees. In Section 4.2 we describe a more involved potential function proving the competitiveness for $k$-taxi on HSTs.

### 4.1 Constructing the Dual Solution

As already mentioned, we break the construction of a dual solution into the same small steps that already partition the movement of DoubleCoverage. That is, we will define altitudes also for the times between two successive small steps. We will call a dual solution where altitudes are also defined for times between small steps an *extended dual solution.*

We will construct this dual solution by induction backwards in time. For a given point in time, let $A_u$ be the altitude of a vertex $u$ at this time as determined by the induction hypothesis, and let $v_i$ be the location of server $i$ at this time. We will denote by $A'_u$ and $v'_i$ the new values of these quantities at the next point in reverse-time. We denote by $\Delta A_u := A_u - A'_u$ and $\Delta A_i := A_{v_i} - A'_{v'_i}$ the change of the altitude of vertex $u$ and server $i$, respectively, in *forward-time* direction. For the update due to a small step when serving a simple request $s_t$, define

$$\Delta D := \Delta A_{s_t} - \sum_i \Delta A_i.$$

Thus, the sum of the quantities $\Delta D$ for all small steps corresponding to a simple request at time $t$ is precisely $D_t$. In reverse-time, we can think of $\Delta D$ as the amount by which the request's altitude *decreases plus* the amount by which the sum of server altitudes *increases.*

We will update altitudes so as to satisfy the following two rules:

(i) $\Delta A_u \geq 0$ for all $u \in V$ (constraint (2) is satisfied): In reverse-time, we only *decrease* the altitude of any vertex (or leave it unchanged).
(ii) $A_u - A_{p(u)} \in \{0, 1\}$ for all $u \neq \mathbb{r}$ at all times (constraint (1) is satisfied): The altitude of $u$ and $p(u)$ is the same, or the altitude of $u$ is higher by one than the altitude of $p(u)$. Overall, altitudes are non-increasing towards the root.

**Lemma 2.** *There exists a feasible extended dual solution satisfying:*

- *For a relocation request at time $t$: $D_t = 0$.*
- *For a small step where $B = \emptyset$: $\Delta D \geq 1$.*
- *For a small step where $B = \{j\}$: $\Delta D \geq 0$.*

*Proof.* For the base of the reverse time induction, let $A$ be some arbitrary constant and define the altitude of every vertex $u \in V$ at the time after the final request to be $A$. This trivially satisfies rule (ii).

**Relocation requests ($t \in T_r$):** We guarantee $D_t = 0$ by simply keeping all altitudes unchanged.

**Simple requests ($t \in T_s$):** Consider a small step of the simple request to $s_t$. In reverse-time, any server $i \in U$ moves from $v_i = p(v'_i)$ to $v'_i$ during the small step. Then, $\Delta A_i = A_{p(v'_i)} - A'_{v'_i}$. By rule (ii) of the induction hypothesis, we have $A_{p(v'_i)} - A_{v'_i} \in \{-1, 0\}$. Similarly, if $B = \{j\}$, then $j$ moves from $v_j$ to $v'_j = p(v_j)$ in reverse-time, and $\Delta A_j = A_{v_j} - A'_{p(v_j)}$.

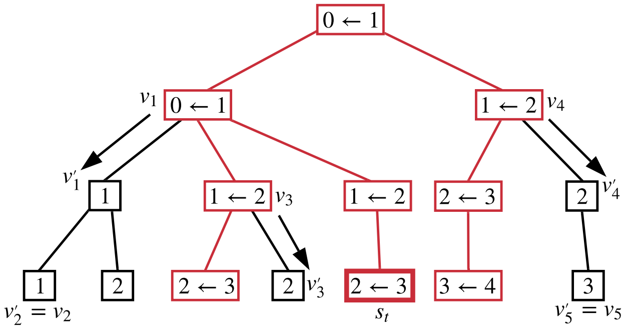

**Fig. 1.** Example of a dual update for a small step of a simple request when $B = \emptyset$. The current request is $s_t$, and the vertices colored red are the component $V'$. The arrows show the movement of servers (in reverse time direction). Numbers in boxes represent altitudes, where $b \leftarrow a$ means that the altitude is updated from $a$ to $b$ (in reverse time).

**Case 1: $B = \emptyset$:** If for at least one server $i \in U$ we have $A_{v'_i} - A_{p(v'_i)} = 1$, then we set $A'_u := A_u$ for all vertices. In this case, $\Delta_t A_i = -1$ for the aforementioned server $i$, and for all other servers in $i \in U$, $\Delta_t A_i \leq 0$. Overall, $\Delta D \geq 1$.
Otherwise, for all $i \in U$, $A_{v'_i} - A_{p(v'_i)} = 0$ meaning that every edge along which a server moves during this small step connects two vertices of the same altitude (an example of the update of the dual for this case is shown in Figure 1). Let $V' = V \setminus \bigcup_{i \in U} V_{v'_i}$ be the connected component containing $s_t$ when cutting all edges traversed by a server in this step. Notice that $V'$ does not contain $v'_i$ even for servers $i$ that are not moving during the step, since those are located in subtrees below the servers of $U$. For each $u \in V'$, we set $A'_u := A_u - 1$ (or $\Delta A_u = 1$), and otherwise we keep the altitudes unchanged. In particular, rule (i) is satisfied. Since for all cut edges $A_{v'_i} - A_{p(v'_i)} = 0$, then for these edges $A'_{v'_i} - A'_{p(v'_i)} = 1$, and rule (ii) also remains satisfied. As stated, servers only moved along edges connecting vertices of the same altitude (before the update), and all server positions $v'_i$ are outside the component $V'$, so $\Delta A_i = 0$ for each server. But the component contains the request $s_t$, so $\Delta A_{s_t} = A_{s_t} - A'_{s_t} = 1$. Overall, we get $\Delta D = 1$.

**Case 2: $B = \{j\}$:** If some server $i \in U$ moves in reverse-time to a vertex of higher altitude ($A_{p(v'_i)} - A_{v'_i} = -1$) *or* server $j$ moves to a vertex of the same altitude ($A_{p(v_j)} - A_{v_j} = 0$), then we set $A'_u := A_u$ for each $u \in V$. In this case, $\Delta A_i \in \{-1, 0\}$ for all $i \in U$, and $\Delta A_j \in \{0, 1\}$, and the aforementioned condition translates to the condition that $\Delta A_i = -1$ for some $i \in U$ *or* $\Delta A_j = 0$. Either case then guarantees that $\Delta D \geq 0$.
Otherwise, $j$ moves (in reverse-time) to a vertex of lower altitude ($A_{p(v_j)} - A_{v_j} = -1$) *and* all servers in $U$ move along edges of unchanging altitude ($A_{p(v'_i)} - A_{v'_i} = 0$) (an example of the update of the dual for this case is shown in Figure 2). Let

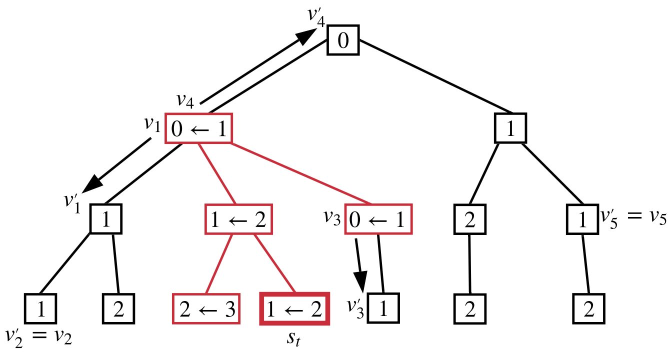

**Fig. 2.** Example of a dual update for a small step of a simple request when $B = \{j\}$. The current request is $s_t$, and the vertices colored red are the component $V'$. The arrows show the movement of servers (in reverse time direction). Numbers in boxes represent altitudes, where $b \leftarrow a$ means that the altitude is updated from $a$ to $b$ (in reverse time).

$V' = V_{v_j} \setminus \bigcup_{i \in U} V_{v'_i}$ be the connected component containing $s_t$ when cutting all edges traversed by servers in this step. We decrease the altitudes of all vertices in this component by 1 ($A'_u := A_u - 1$ for $u \in V'$) and leave other altitudes unchanged, satisfying rule (i). As $A_{p(v_j)} - A_{v_j} = -1$ and $A_{p(v'_i)} - A_{v'_i} = 0$ for $i \in U$, also rule (ii) is satisfied. Again, the locations $v'_i$ of any server $i$ (moving or not) are outside the component, so the update of altitudes does not affect $\Delta A_i$. Thus, $\Delta A_j = A_{v_j} - A_{p(v_j)} = 1$ and, for each server $i \neq j$, $\Delta A_i = 0$. But the altitude of $s_t$ is decreasing by 1 in reverse-time, so $\Delta_t A_{s_t} = 1$. Overall, we get that $\Delta D = 0$. ☐

**Potential Function Requirements.** Based on Lemma 2, we conclude that the following requirements of a potential function $\Psi$ are sufficient to conclude inequality (4) (resp. (5)) and therefore $c$-competitiveness of DoubleCoverage.

**Observation 2.** DoubleCoverage is $c$-competitive if there is a potential $\Psi$ satisfying:

- For a relocation request at time $t$: $\Psi_t = \Psi_{t-1}$.
- In a single step of a simple request, where no server is going downwards: $|U| + \Delta\Psi \le c$.
- In a single step of a simple request, where there is a server moving downwards: $|U| + \Delta\Psi \le 0$.

In the $k$-server problem there are no relocation requests, and one can satisfy the two requirements involving simple requests in Observation 2 with the potential function $\Psi = -\sum_{i<j} d_{\mathrm{lca}(i,j)}$, where the sum is taken over all pairs of servers $\{i, j\}$ and $d_{\mathrm{lca}(i,j)}$ denotes the weighted depth (distance from the root $\mathbb{r}$) of the least common ancestor of $i$ and $j$.

### 4.2 Finalizing the Analysis for $k$-Taxi on HSTs

We show that the competitive ratio of the $k$-taxi problem on HSTs of combinatorial depth $d$ is $c_{kd} = \sum_{h=1}^{k \wedge d} \binom{k}{h}$, which is the number of non-empty sets of at most $d$ servers. One can prove by induction on $k$ that for $k \geq 0$ and $d \geq 1$,

$$c_{kd} = k + \sum_{i=0}^{k-1} c_{i,d-1}. \tag{6}$$

For a given point in time, fix a naming of the servers by the numbers $0, \ldots, k-1$ such that their heights $h_0 \leq \cdots \leq h_{k-1}$ are non-decreasing. If we consider a (small) step, we choose this numbering such that $h_0 \leq \cdots \leq h_{k-1}$ holds both *before* and *after* the step. Since it is not possible that a server is strictly higher than another server before the step and then strictly lower afterwards, such a numbering exists. Let $U, B \subseteq \{0, \ldots, k-1\}$ be the sets of servers that move upwards (i.e., towards the root) and downwards (away from the root), respectively. Note that by our earlier observation, $B$ is either a singleton $\{j\}$ (one server moves downwards) or the empty set (no server moves downwards).

Let $\alpha_\ell$ denote the weighted height of the node layer at combinatorial height $\ell$ in the HST (i.e., the distance of these vertices from the leaf layer). Thus, $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_d$. We use the following potential function at time $t$:

$$\Psi_t = \sum_{i=0}^{k-1} \sum_{\ell=0}^{d-1} c_{i\ell} \cdot \max\{\alpha_\ell, h_{it} \wedge \alpha_{\ell+1}\},$$

where $h_{0t} \leq \cdots \leq h_{k-1,t}$ are the weighted heights of the servers. We next show that $\Psi$ satisfies the requirements of Observation 2 with $c = c_{kd}$.

**A relocation request, $t \in T_r$:** Since all requests are at the leaves and thus the height of the moving server is 0, we have $\Psi_t = \Psi_{t-1}$.

**A small step of a simple request, $t \in T_s$:** Let $\ell_i$ be such that the edge traversed by server $i$ during the step is located between node layers of combinatorial heights $\ell_i$ and $\ell_{i+1}$. Thus, the weighted height of $i$ lies in $[\alpha_{\ell_i}, \alpha_{\ell_i+1}]$. Then

$$\Delta\Psi = \sum_{i \in U} c_{i\ell_i} - \sum_{j \in B} c_{j\ell_j}. \tag{7}$$

**Case 1: $B = \emptyset$:**

$$|U| + \Delta\Psi \leq k + \sum_{i=0}^{k-1} c_{i\ell_i} \leq k + \sum_{i=0}^{k-1} c_{i,d-1} = c_{kd}.$$

The first inequality follows from (7) and since $|U| \leq k$. The second inequality follows from the definition of $c_{kd}$. The final equation is due to (6).

**Case 2: $B = \{j\}$:** In this case $U \subseteq \{0, \dots, j-1\}$ and $\ell_i \le \ell_j - 1$ for each $i \in U$. Therefore,

$$|U| + \Delta\Psi \le j - c_{j\ell_j} + \sum_{i \in U} c_{i\ell_i} \le j - c_{j\ell_j} + \sum_{i=0}^{j-1} c_{i,\ell_j - 1} = 0.$$

The first inequality follows from (7) and since $U \subseteq \{0, \dots, j-1\}$. The second inequality follows from the definition of $c_{kd}$. The equation is due to (6).

## 5 The $k$-taxi Problem on Weighted Trees

We briefly summarize the main ways in which the proof of part (b) of Theorem 1 differs from that of part (a). There are two reasons why our analysis for HSTs fails on general weighted trees:

1. The costs of movement towards and away from the root no longer need to be within a constant of each other. E.g., if relocation repeatedly brings servers closer to the root, then most cost would be incurred while moving away from the root.
2. The potential is no longer constant under relocation requests, because servers can be relocated to and from internal vertices, affecting their height.

To address the first issue, we use an LP formulation that measures movement cost both towards and away from the root. This is achieved in the primal LP by introducing additional variables $z_{ut} \ge 0$ for downwards movement, replacing $y_{ut}$ by $y_{ut} + z_{ut}$ in objective function, and replacing the constraints $y_{ut} \ge x_{u,t-1} - x_{ut}$ by $y_{ut} - z_{ut} = x_{u,t-1} - x_{ut}$. The only change this causes in the (transformed) dual is that the constraint $A_{ut} - A_{p(u)t} \in [0, 1]$ becomes $A_{ut} - A_{p(u)t} \in [-1, 1]$.

To address the second issue, we eliminate the potential function from our proof. Instead, we construct a dual solution that bounds the cost of DOUBLE-COVERAGE in each step, but it may violate the constraints $A_{ut} - A_{p(u)t} \in [-1, 1]$. However, it will still satisfy $A_{ut} - A_{p(u)t} \in [-c, c]$ for some $c$. Thus, dividing all dual variables by $c$ yields a feasible dual solution, and $c$ is our competitive ratio. A complete proof is given in the full version of our paper.

## References

1. Azar, Y., Buchbinder, N., Chan, T.H., Chen, S., Cohen, I.R., Gupta, A., Huang, Z., Kang, N., Nagarajan, V., Naor, J., Panigrahi, D.: Online algorithms for covering and packing problems with convex objectives. In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016. pp. 148–157. IEEE Computer Society (2016)
2. Bartal, Y.: Probabilistic approximation of metric spaces and its algorithmic applications. In: In 37th Annual Symposium on Foundations of Computer Science. pp. 184–193. FOCS '96 (1996)

3. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in on-line algorithms. Algorithmica **11**(1), 2–14 (1994)
4. Bubeck, S., Cohen, M.B., Lee, Y.T., Lee, J.R., Madry, A.: k-server via multiscale entropic regularization. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018. pp. 3–16. ACM (2018)
5. Buchbinder, N., Gupta, A., Molinaro, M., Naor, J.S.: k-servers with a smile: Online algorithms via projections. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019. pp. 98–116. SIAM (2019)
6. Buchbinder, N., Naor, J.: The design of competitive online algorithms via a primal-dual approach. Foundations and Trends in Theoretical Computer Science **3**(2-3), 93–263 (2009)
7. Chrobak, M., Karloff, H., Payne, T., Vishwanathan, S.: New results on server problems. SIAM J. Discrete Math. **4**(2), 172–181 (1991)
8. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for k servers on trees. SIAM Journal on Computing **20**(1), 144–148 (1991)
9. Coester, C., Koutsoupias, E.: The online *k*-taxi problem. In: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019. pp. 1136–1147. ACM (2019)
10. Dehghani, S., Ehsani, S., Hajiaghayi, M., Liaghat, V., Seddighin, S.: Stochastic k-Server: How Should Uber Work? In: 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). pp. 126:1–126:14 (2017)
11. 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)
12. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. J. Algorithms **12**(4), 685–699 (1991)
13. Fiat, A., Rabani, Y., Ravid, Y.: Competitive k-server algorithms (extended abstract). In: 31st Annual Symposium on Foundations of Computer Science. pp. 454–463. FOCS '90 (1990)
14. Gupta, A., Nagarajan, V.: Approximating sparse covering integer programs online. Math. Oper. Res. **39**(4), 998–1011 (2014)
15. Kosoresow, A.P.: Design and analysis of online algorithms for mobile server applications. Ph.D. thesis, Stanford University (1996)
16. Koutsoupias, E.: The k-server problem. Computer Science Review **3**(2), 105–118 (2009)
17. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)
18. Lee, J.R.: Fusible HSTs and the randomized k-server conjecture. In: Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science. pp. 438–449. FOCS '18 (2018)
19. Lee, J.R.: Fusible HSTs and the randomized k-server conjecture (Feb 2018), arXiv:1711.01789v2
20. Lee, J.R.: Personal Communication (2019)
21. Manasse, M., McGeoch, L., Sleator, D.: Competitive algorithms for on-line problems. In: Proceedings of the twentieth annual ACM Symposium on Theory of Computing. pp. 322–333. STOC '88, ACM (1988)