Article:
Buchbinder, N., Coester, C. and Naor, J. (2023) Online k-taxi via Double Coverage and time-reverse primal-dual. Mathematical Programming, 197 (2). pp. 499-527. ISSN: 0025-5610

https://doi.org/10.1007/s10107-022-01815-6

Mathematical Programming (2023) 197:499–527
https://doi.org/10.1007/s10107-022-01815-6

FULL LENGTH PAPER

Series B

Online k-taxi via Double Coverage and time-reverse primal-dual

Niv Buchbinder[1] · Christian Coester[2] · Joseph Naor[3]

Received: 28 April 2021 / Accepted: 16 March 2022 / Published online: 13 May 2022

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 Δ is the aspect ratio and $c \geq 1$ is an arbitrary positive integer constant. The 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, and in contrast to 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 both the past and the future when

An extended abstract of this article appeared in IPCO 2021.
This work is supported in part by US-Israel BSF Grant 2018352 and by ISF Grant 2233/19 (2027511)
Most of the work was done while C. Coester was at CWI in Amsterdam, supported by the NWO VICI Grant 639.023.812.

✉ Christian Coester
christian.coester@gmail.com

Niv Buchbinder
niv.buchbinder@gmail.com

Joseph Naor
naor@cs.technion.ac.il

1 Department of Statistics and Operations Research, Tel Aviv University, Tel Aviv, Israel

2 University of Sheffield, Sheffield, UK

3 Computer Science Department, Technion, Haifa, Israel

assigning dual variables. We believe this method can also be useful 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.

Mathematics Subject Classification 68W27 Online algorithms; streaming algorithms · 68W40 Analysis of algorithms

1 Introduction

The k-taxi problem, proposed three decades ago as a natural generalization of the k-server problem by Fiat et al. [14], has gained renewed interest recently. In 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 [8], and this result was later generalized to tree metrics [9].

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 $2k - 1$ [18]; with randomization, on hierarchically well-separated trees (HSTs)[1] the best known bound is $O(\log^2 k)$ [5, 6]. 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 [5] that a dynamic embedding yields a bound of $O(\log^3 k \log \Delta)$ for metrics with aspect ratio Δ. In [19], a more involved dynamic embedding was proposed to achieve a polylog(k)-competitive algorithm for general metrics.[2] Contrast these upper bounds with the known deterministic lower bound of k [22] and the randomized lower bound of $\Omega(\log k)$ [13]. More information about the k-server problem can be found in [17].

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 [10] 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 only existential and does not provide a way to construct the algorithm; (ii) an $O(2^k \log n)$-competitive randomized algorithm for general metric spaces (against an oblivious adversary). Very recently, Bubeck et al. [4] gave an $O(n^2 \log^2 k \log n)$-competitive algorithm, thus improving upon the $O(2^k \log n)$ bound whenever n is subexponential in k. All these bounds currently constitute the state of the art. Coester and Koutsoupias [10] 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 $k \leq 3$ [10]. 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 [10, 16], 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 [10]. The k-taxi problem was recently reintroduced as the Uber problem in [11], 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*

[1] See Sect. 2 for an exact definition of HSTs.

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

(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 $c_{kd} = 2^k - 1$ also matches exactly the lower bound from [10] 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 on general trees can be chosen arbitrarily, d is essentially half the hop-diameter.

By well-known embedding techniques of general metrics into HSTs [2, 12], slightly adapted to HSTs of bounded depth (see Theorem 6 in Sect. 2), we obtain the following result for general metrics.

Corollary 3 *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* Δ *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 from [10], 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 also recover the same $O(2^k \log n)$ competitive factor. The bounds in Corollary 3 actually hide another division by $(c-1)!$ if $c \leq k$. Therefore, whenever Δ is at most $2^{O(k^2)}$ our bound yields 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 [10], 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 $k = 3$ (see Appendix 1). 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 auxilary 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. [1, 7, 15]). 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 are considering, we show that a pure forward-time approach (producing a competitive dual solution as well) is doomed to fail:

Theorem 4 *There exists no competitive online algorithm for the dual problem of the k-taxi LP as defined in Sect. 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 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 dual fitting suitable for online algorithms that is more "local" and hence easier to be analyzed 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 has not been known before.

Theorem 5 [9] DoubleCoverage *is k-competitive for the k-server problem on trees.*

1.3 Organization

Section 2 defines the necessary notation and terminology. In Sect. 3, we provide a linear programming formulation of the k-taxi problem on trees and the corresponding dual, which we then transform to a more intuitive equivalent dual LP. In Sect. 4 we prove the upper bound for k-taxi on HSTs (Theorem 1(a)) and, as a byproduct, for k-server on general trees (Theorem 5). The upper bound for k-taxi on general trees (Theorem 1(b)) is given in Sect. 5. The lower bounds (Theorem 2) are proved in Sect. 6. Corollary 3 follows directly from Theorem 1(a) and Theorem 6 in Sect. 2. Appendix 1 contains a proof of the limitation of forward-time setting of dual variables (Theorem 4).

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 $s_t, d_t \in V$. The request is served by moving one of the servers to s_t (unless there is already a server at s_t). The cost paid by the algorithm is the distance traveled by the server to s_t. 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.

Note that any request (s_t, d_t) can be split into two requests: first (s_t, s_t), then (s_t, d_t). This does not affect the optimal (offline) cost, and for the online algorithm the problem only becomes harder because it has to decide which server to send to s_t before the point d_t is revealed. We can therefore assume without loss of generality that requests are split in this way. Thus, at each time t, request (s_t, d_t) is either a simple or a relocation request as per the following definition:

- *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* ($d_{t-1} = s_t \neq d_t$): a server already present at s_t is relocated to d_t. There is no relocation cost.

We can then partition the time horizon $\{1, 2, \ldots, T\}$ into two sets T_s and T_r corresponding to simple and relocation requests, i.e., $T_s = \{t = 1, \ldots, T : s_t = d_t\}$ and $T_r = \{t = 1, \ldots, T : d_{t-1} = s_t \neq d_t\}$, and cost is incurred only during time steps in T_s. 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 the 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 denote by $v \prec u$ that v is a child of u. We write $p(u)$ for 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 α-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 α 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 α-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 $O(\alpha \log_\alpha n)$ [12]. 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 Δ 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 6 [Corollary to [12]] *Any metric with n points and aspect ratio Δ 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 7 *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 on trees along with its dual that we use for the purpose of analysis. We assume for ease of exposition that all edges are short edges. The formulation is a relaxation of the problem as it allows for fractional values of the variables[3]. 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 the subtree V_u (moving upwards) at time t. The variable $z_{ut} \geq 0$ denotes the number of servers that enter V_u (moving downwards) at time t. For $u = \mathbb{r}$, $x_{\mathbb{r}t}$ is defined to be the *constant* k. (It is not a variable.) The first set of constraints ensures that the number of servers at a node u is at least the number of servers in its children with an additional server if the request is for vertex u. The second set of requests measures the movement cost (upwards and downwards) from each node u at any simple request. Finally, the third set of constraint "implements" the relocation request by removing one server from each subtree that contains s_t and not d_t and adding a server to subtrees that contains d_t but not s_t.

[3] In general, for any metric, the offline k-taxi problem can be solved in polynomial time by a reduction to a min-cost flow problem.

The dual variables corresponding to the primal constraints appear in parenthesis left of the constraints. The primal LP is the following:

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

where

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

For technical reasons, before we construct the dual LP we will add the additional constraint

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

to the primal LP, where $\bar{x}_{u,T}$ are *constants* specifying the configuration of DOUBLE-COVERAGE 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:

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

$$\begin{aligned}
(x_{ut}) \quad & \lambda_{ut} - \lambda_{p(u)t} = b_{ut} - b_{u,t-1} && \forall u\neq\mathbb{r}, t\in T_s\cup T_r \quad (1) \\
(z_{ut}, y_{ut}) \quad & b_{u,t-1}\in[-1,1] && \forall u\neq\mathbb{r}, t\in T_s \quad (2) \\
& \lambda_{ut}\geq 0 && \forall u\in V, t\in T_s\cup T_r \quad (3)
\end{aligned}$$

We can use the same primal and dual formulation for the k-server problem, except that the set T_r is then empty.[4]

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).

[4] 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.

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, and thus the coefficient of the variables z_{ut} in the primal objective function become zero. The only change to the dual linear program as a result of this is that constraint (2) becomes

$$b_{u,t-1} \in [0,1] \qquad \forall u \neq \mathbb{r}, t \in T_s \qquad \text{when measuring only upward cost.} \tag{4}$$

3.1 Dual transformation

The dual LP is not very intuitive. By a transformation of variables, we can obtain a simpler equivalent dual LP. In Appendix 1 we also give an alternative (but less intuitive) primal LP that would directly yield the new dual derived in this section.

The new dual LP has only one variable A_{ut} for each vertex u and time t. We can interpret this new dual as building a mountain structure on the tree over time, where we view 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:

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

$$A_{ut} - A_{p(u)t} \in [-1,1] \qquad \forall u \neq \mathbb{r}, t+1 \in T_s \tag{5}$$

$$\Delta_t A_u \geq 0 \qquad \forall u \in V, t \in T_s \cup T_r \tag{6}$$

The first constraint of the LP means that altitudes vary with "slope" at most 1 between adjacent vertices (at time steps before a simple request). The second constraint stipulates that the altitude of each node can only increase over time. The objective function measures changes in the altitudes of request and server locations. When measuring only movement towards the root, constraint (5) becomes

$$A_{ut} - A_{p(u)t} \in [0,1] \qquad \forall u \neq \mathbb{r}, t+1 \in T_s. \tag{7}$$

This corresponds to the additional requirement that altitudes are non-decreasing along root-to-leaf paths.

We define

$$D_t := \begin{cases} \Delta_t A_{s_t} - \sum_{i=1}^{k} \Delta_t A_i & t \in T_s \\ -\sum_{i=1}^{k} \Delta_t A_{v_{it}} & t \in T_r, \end{cases} \tag{8}$$

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.

Lemma 8 *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.*

Proof We refer to the first LP as the original LP and the second LP as the new LP. Given a solution A_{ut} to the new LP let

$$\lambda_{ut} := \Delta_t A_u := A_{ut} - A_{u,t-1}$$
$$b_{ut} := A_{ut} - A_{p(u)t}.$$

It is easy to see that feasibility for the new LP implies feasibility for the original LP; in particular, $\lambda_{ut} - \lambda_{p(u)t} = A_{ut} - A_{u,t-1} - A_{p(u)t} + A_{p(u),t-1} = b_{ut} - b_{u,t-1}$. For the other direction, a feasible solution λ_{ut}, b_{ut} to the original LP can be transformed by setting

$$A_{ut} := \sum_{\tau=1}^{t} \lambda_{u\tau} + \sum_{v \neq \mathbb{r} | u \in V_v} b_{v0}.$$

Using this definition, we have:

$$A_{ut} - A_{p(u)t} = \sum_{\tau=1}^{t} \left(\lambda_{u\tau} - \lambda_{p(u)\tau}\right) + b_{u0} = b_{ut} \tag{9}$$
$$\Delta_t A_u = A_{ut} - A_{u,t-1} = \lambda_{ut} \tag{10}$$

Again, feasibility for the new LP follows from the constraints of the original LP.

Finally, we consider the value of the objective function. Notice that Eqs. (9) and (10) hold for both directions of the transformation, so we can use them in the following. Let $\bar{x}_{ut}$ denote the number of servers i with $v_{it} \in V_u$. Using this, we can write

$$\begin{aligned}
\sum_{u \neq \mathbb{r}} \bar{x}_{ut} b_{ut} &= \sum_{u \neq \mathbb{r}} \sum_{i | v_{it} \in V_u} b_{ut} \\
&= \sum_{i} \sum_{u \neq \mathbb{r} | v_{it} \in V_u} b_{ut} \\
&= \sum_{i} \left(A_{v_{it}t} - A_{\mathbb{r}t}\right) \\
&= -kA_{\mathbb{r}t} + \sum_{i} A_{v_{it}t},
\end{aligned}$$

where the penultimate equation follows from Eq. (9). Therefore,

$$\sum_{u \neq \mathbb{r}} \left(\bar{x}_{u,t-1} b_{u,t-1} - \bar{x}_{ut} b_{ut}\right) = k\Delta_t A_{\mathbb{r}} - \sum_{i} \Delta_t A_i = k\lambda_{\mathbb{r}t} - \sum_{i} \Delta_t A_i. \tag{11}$$

Thus, for $t \in T_s$ we get

$$D_t = \Delta_t A_{s_t} - \sum_i \Delta_t A_i = \lambda_{s_t t} - k\lambda_{\mathbb{r}t} + \sum_{u \neq \mathbb{r}} \left(\bar{x}_{u,t-1} b_{u,t-1} - \bar{x}_{ut} b_{ut} \right). \tag{12}$$

For a relocation request at time $t \in T_r$, most servers stay in their old position (hence $\Delta_t A_i = \Delta_t A_{v_{it}}$ for these servers), except for the relocated server i^*, which moves from $v_{i^*,t-1} = s_t$ to $v_{i^*t} = d_t$, so for this server $\Delta_t A_{v_{i^*t}} = A_{d_t,t} - A_{d_t,t-1}$ and $\Delta_t A_{i^*} = A_{d_t,t} - A_{s_t,t-1}$. We therefore have

$$D_t = -\sum_i \Delta_t A_{v_{it}} = A_{d_t,t-1} - A_{s_t,t-1} - \sum_i \Delta_t A_i \qquad \text{(if } t \in T_r\text{)}.$$

Using (9), we can rewrite

$$A_{d_t,t-1} - A_{s_t,t-1} = \sum_{u | s_t \notin V_u, d_t \in V_u} b_{u,t-1} - \sum_{u | s_t \in V_u, d_t \notin V_u} b_{u,t-1} = \sum_{u \neq \mathbb{r}} \xi_{ut} b_{u,t-1}.$$

Recalling (11), we get for $t \in T_r$ that

$$D_t = \sum_{u \neq \mathbb{r}} \xi_{ut} b_{u,t-1} - k\lambda_{\mathbb{r}t} + \sum_{u \neq \mathbb{r}} \left(\bar{x}_{u,t-1} b_{u,t-1} - \bar{x}_{ut} b_{ut} \right). \tag{13}$$

Combining (12) and (13) and noticing that the terms $\bar{x}_{u,t-1} b_{u,t-1} - \bar{x}_{ut} b_{ut}$ telescope over time for each $u \neq \mathbb{r}$, the dual objective value of the new LP can be rewritten as

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

which is precisely the objective value of the original dual. □

4 The k-taxi problem on HSTs

In this section we analyze DoubleCoverage on HSTs, proving part (a) of Theorem 1. As a byproduct we also give a primal-dual proof of the k-competitiveness of DoubleCoverage for the k-server problem on trees (Theorem 5).

Besides constructing a dual solution, our analysis will also employ a potential function Ψ. The choice of Ψ will be the only difference between the analyses for the k-server and k-taxi problem. The dual solution and potential will be such that for all $t \in T_s \cup T_r$,

$$cost_t^{\uparrow} + \Psi_t - \Psi_{t-1} \leq c \cdot D_t, \tag{14}$$

where $cost_t^{\uparrow}$ is the cost of movement *towards the root* by DOUBLECOVERAGE's servers while serving the tth request, c is the desired competitive ratio, and D_t is the increase of the dual objective function at time t, as given by (8). As discussed in Sect. 2 we may use in this case the dual of the program that only measures movement cost towards the root. Thus, summing (14) 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 Ψ during the step and by ΔD 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 (14) for simple requests if we show for each step that

$$|U| + \Delta\Psi \leq c \cdot \Delta D. \tag{15}$$

In Sect. 4.1 we describe how we construct the dual solution by going backwards in time. In Sect. 4.2 we design a simple potential function that proves the k-competitiveness of k-server on trees. In Sect. 4.3 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 ΔD for all small steps corresponding to a simple request at time t is precisely D_t. In reverse-time, we can think of ΔD 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:

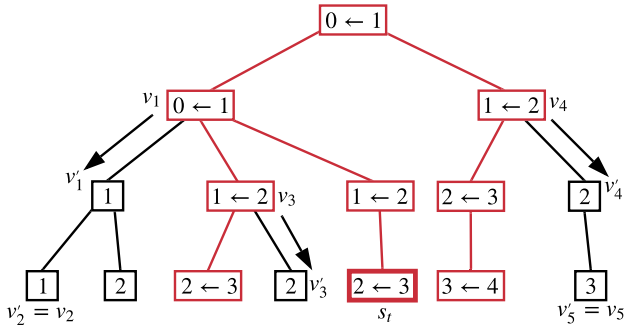

Fig. 1 Example of a dual update for a small step of a simple request when $B = \emptyset$. The current request is s_t, and the vertices colored red are the component V'. The thick arrows show the movement of servers (in reverse time direction). Numbers in boxes represent altitudes, i.e., dual variables: We use the notation $b \leftarrow a$ to denote that the altitude of the vertex is changed from a to b (in reverse time)

(i) $\Delta A_u \geq 0$ for all $u \in V$ (constraint (6) 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 (7) 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.

Recall, that 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 (see also, Observation 7).

Lemma 9 *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 $\Delta A_j = A_{v_j} - A'_{p(v_j)}$.

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 $i \in U$ we have $\Delta_t A_i \leq 0$. Overall, $\Delta D \geq 1$.
Otherwise, we have $A_{v'_i} - A_{p(v'_i)} = 0$ for all $i \in U$, meaning that every edge along which a server moves during this small step connects two vertices of the same altitude

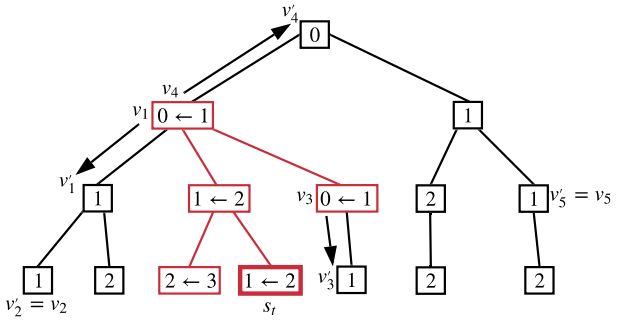

Fig. 2 Example of a dual update for a small step of a simple request when $B = \{j\}$. The current request is s_t, and the vertices colored red are the component V'. The thick arrows show the movement of servers (in reverse time direction). Numbers in boxes represent altitudes, i.e., dual variables: We use the notation $b \leftarrow a$ to denote that the altitude of the vertex is changed from a to b (in reverse time)

(an example of the update of the dual for this case is shown in Fig. 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 Fig. 2). Let $V' = V_{v_j} \setminus \bigcup_{i \in U} V_{v'_i}$ 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 ΔA_i. Thus, $\Delta A_j = A_{v_j} - A_{p(v_j)} = 1$ and, for each server $i \neq j$, we have $\Delta A_i = 0$. 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 9, we conclude that the following requirements of a potential function Ψ are sufficient to conclude inequality (14) (resp. (15)) and therefore c-competitiveness of DOUBLECOVERAGE.

Observation 10 DOUBLECOVERAGE *is c-competitive if there is a potential* Ψ *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:* $|U| + \Delta\Psi \leq c$.
- *In a single step of a simple request, where there is a server moving downwards:* $|U| + \Delta\Psi \leq 0$.

4.2 Finalizing the analysis for *k*-server on trees

In the k-server problem there are no relocation requests, and hence we only need to satisfy the two requirements involving simple requests in Observation 10. We define the following 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.

Case 1: $B = \emptyset$: In this case, the depth of the least common ancestor decreases by 1 only for $i \in U$ and j which is located in the subtree of i. The total number of servers that are located in the subtrees below servers of U is precisely $k - |U|$, and hence the potential function in this case grows by $k - |U|$. Thus, $|U| + \Delta\Psi = k$, satisfying Observation 10.

Case 2: $B = \{j\}$: In this case, all the servers of U are located in the subtree of j. For each server $i \in U$, the depth of the least common ancestor with j increases by 1, therefore contributing $-|U|$ to $\Delta\Psi$. There may be more servers in the subtree of j that do not belong to U. For each such server i', there is a server $i \in U$ on its path to j. The depth of the least common ancestor of i' and i decreases by 1, while the depth of the least common ancestor of i' and j increases by 1. Thus, the total contribution of i' to $\Delta\Psi$ is 0, and likewise for servers outside the subtree of j. Hence, $|U| + \Delta\Psi = 0$, satisfying Observation 10.

4.3 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. We need the following recurrence relation for c_{kd}.

Lemma 11 *For $k \geq 0$ and $d \geq 1$,*

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

Proof The statement follows by induction on k. For the induction step, we have

$$\begin{aligned} k + \sum_{i=0}^{k-1} c_{i,d-1} &= 1 + c_{k-1,d-1} + \left(k - 1 + \sum_{i=0}^{k-2} c_{i,d-1} \right) \\ &= 1 + c_{k-1,d-1} + c_{k-1,d} \\ &= \sum_{h=1}^{k \wedge d} \binom{k-1}{h-1} + \sum_{h=1}^{(k-1) \wedge d} \binom{k-1}{h} \\ &= \sum_{h=1}^{k \wedge d} \binom{k}{h} = c_{kd}. \end{aligned}$$

□

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). We sometimes write a sum over elements of B, so such a sum is either 0 or a single term.

Let α_ℓ denote the weighted height of the node layer at combinatorial height ℓ 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:

$$\Psi_t = \sum_{i=0}^{k-1} \sum_{\ell=0}^{d-1} c_{i\ell} \cdot \max\{\alpha_\ell, h_{it} \wedge \alpha_{\ell+1}\}, \tag{16}$$

where $h_{0t} \leq \cdots \leq h_{k-1,t}$ are the weighted heights of the servers. We next show that Ψ satisfies the requirements of Observation 10 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 ℓ_i be such that the edge traversed by server i during the step is located between the node layers of combinatorial heights ℓ_i and ℓ_{i+1}. Thus, the weighted height of i lies in $[\alpha_{\ell_i}, \alpha_{\ell_i+1}]$ during the step. Then

$$\Delta\Psi = \sum_{i \in U} c_{i\ell_i} - \sum_{j \in B} c_{j\ell_j}. \tag{17}$$

There are two cases to be considered.

Case 1: $B = \emptyset$:

$$|U| + \Delta\Psi \leq k + \sum_{i=0}^{k-1} c_{i\ell_i} \tag{18}$$

$$\leq k + \sum_{i=0}^{k-1} c_{i,d-1} = c_{kd}. \tag{19}$$

Inequality (18) follows from (17) and since $|U| \leq k$. Inequality (19) follows from the definition of c_{kd}. The final equation is due to Lemma 11.

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

$$|U| + \Delta\Psi \leq j - c_{j\ell_j} + \sum_{i \in U} c_{i\ell_i} \tag{20}$$

$$\leq j - c_{j\ell_j} + \sum_{i=0}^{j-1} c_{i,\ell_j - 1} = 0. \tag{21}$$

Inequality (20) follows from (17) and since $U \subseteq \{0, \ldots, j-1\}$. Inequality (21) follows from the definition of c_{kd}. The equation is due to Lemma 11.

5 The k-taxi problem on weighted trees

In this section we analyze DoubleCoverage on general (weighted) trees, proving part (b) of Theorem 1. Specifically, we prove the following theorem.

Theorem 12 *The competitive ratio of* DoubleCoverage *on weighted trees of depth d is at most*

$$\begin{cases} 4d - 1 & \text{if } k = 2 \\ \dfrac{2k(k-1)^d - 3k + 2}{k-2} = O(k^d) & \text{if } k \geq 3. \end{cases}$$

There are two reasons why our analysis for HSTs fails on general weighted trees:

1. In the k-server problem (with only simple requests), for each edge the cost of traversing it in one direction is equal to the cost of traversing it in the other direction, up to an additive constant. In the k-taxi problem, when relocation requests are allowed, this is no longer true. However, in an HST, where relocation requests are allowed only between leaves that are all at the same distance from the root, it is true that the total upward movement is always the same as the downwards movement up to an additive constant of wk, where w is the distance from the root to any leaf. In an arbitrary weighted tree 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. Relocation requests can be done arbitrarily and incur no cost for the algorithm as well as the dual solution. Hence, a potential function satisfying inequality (14) must be constant under relocation requests. However, on arbitrary trees a relocation can affect the height of a server, and thus the potential (16) is no longer constant under relocation requests on arbitrary trees.

To address the first issue, we use the LP formulations that measure movement cost both towards and away from the root. As discussed in Sect. 3, the only change in the dual is to replace $A_{ut} - A_{p(u)t} \in [0, 1]$ by $A_{ut} - A_{p(u)t} \in [-1, 1]$. To address the second issue, we will eliminate the potential function from our proof. Instead, we will construct a dual solution that bounds the cost of DOUBLECOVERAGE 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.

5.1 Proof of Theorem 12

We now turn to a more detailed description of the analysis. The proof uses the same notation and the observations as in Sect. 4. Let d be the depth of the tree. For $i = 1, \ldots, d$, let

$$m_i := \begin{cases} -2(d-i)-1 & \text{if } k = 2 \\ \frac{-2(k-1)^{d-i+1}+k}{k-2} & \text{if } k \geq 3 \end{cases}$$

$$M_i := \begin{cases} 2(d+i)-1 & \text{if } k = 2 \\ \frac{2k(k-1)^d - 2(k-1)^{d-i+1} - k}{k-2} & \text{if } k \geq 3. \end{cases}$$

These quantities have been chosen to satisfy the following lemma:

Lemma 13 *The values m_i and M_i satisfy the following:*

(a) $m_1 < m_2 < \cdots < m_d = -1 < M_1 < M_2 < \cdots < M_d$
(b) $M_i - m_i$ *is a constant independent of* i.
(c) $M_1 + (j-1)m_1 \geq j$ *for all* $j = 1, \ldots, k$.
(d) $(j-1)m_{i+1} - m_i \geq j$ *for all* $j = 1, \ldots, k$ *and* $i = 1, \ldots, d-1$.

Proof Properties (a) and (b) are straightforward to check. For properties (c) and (d), since $m_1 < m_{i+1} < 0$ it suffices to show these for the case $j = k$. For $k = 2$, they are easily verified. For $k \geq 3$, we have

$$\begin{aligned} M_1 + (k-1)m_1 &= \frac{2k(k-1)^d - 2(k-1)^d - k}{k-2} + (k-1)\frac{-2(k-1)^d + k}{k-2} \\ &= \frac{k^2 - 2k}{k-2} = k \\ (k-1)m_{i+1} - m_i &= (k-1)\frac{-2(k-1)^{d-i} + k}{k-2} - \frac{-2(k-1)^{d-i+1} + k}{k-2} \\ &= \frac{k^2 - 2k}{k-2} = k. \end{aligned}$$

□

Recall that we break the long edges of the original tree into short edges of unit length. Requests arrive only in the subset of V that are endpoints of long edges. For a node $u \in V$ (which might lie in the middle of a long edge), define its depth d_u to be be the minimal number of long edges of a path of long edges that starts at the root $\mathbb{r}$ and includes u. We will construct a dual solution that satisfies constraint (6), but instead of (5) it will only satisfy $A_{ut} - A_{p(u)t} \in [m_{d_u}, M_{d_u}]$. We use again the terminology and notation from before. To satisfy these (relaxed) constraints, we impose the following two rules on the extended dual solution that we will be constructing:

(i) $\Delta A_u \geq 0$ for all $u \in V$: Altitudes can only decrease in reverse-time.
(ii) $A_u - A_{p(u)} \in [m_{d_u}, M_{d_u}]$ for all $u \neq \mathbb{r}$ at all times: Adjacent altitudes are not too different.

We will show for all $t \in T_s \cup T_r$ that

$$cost_t \leq D_t, \tag{22}$$

where $cost_t$ is the movement cost by DoubleCoverage to serve the tth request.
The competitive ratio: A feasible dual solution can then be obtained by dividing all altitudes by

$$c := \max\{M_i, |m_i| : i = 1, \ldots, d\} = M_d = \begin{cases} 4d - 1 & \text{if } k = 2 \\ \frac{2k(k-1)^d - 3k + 2}{k-2} = O(k^d) & \text{if } k \geq 3. \end{cases}$$

As this also divides the dual objective value by c, it implies that DoubleCoverage is c-competitive, proving Theorem 12.
Constructing a dual solution satisfying (22): As before, we proceed by induction backwards in time, and divide into several cases. We start with some arbitrary fixed altitude A for each vertex at the time after the final request.
A relocation request, $t \in T_r$: We keep all altitudes unchanged. Observe that (22) is satisfied with both sides equal to 0.

A simple request, $t \in T_s$: We break the movement again into small steps where DOUBLECOVERAGE's servers move by distance 1 and employ the earlier notation. For a given step, the cost of DOUBLECOVERAGE is $|U| + |B|$. Thus, we obtain (22) if we can show for the step that

$$|U| + |B| \leq \Delta D.$$

Case 1: $B = \emptyset$: Let $\delta := \min_{i \in U} M_{d_{v_i'}} - A_{v_i'} + A_{p(v_i')}$ and let the minimum be achieved for $i^* \in U$. By rule (ii) of the induction hypothesis, we have $\delta \geq 0$. Therefore, the following reverse-time update of altitudes satisfies rules (i) and (ii):

$$\begin{aligned} A'_u &:= A_u - \delta && \text{for } u \in V \setminus \cup_{i \in U} V_{v_i} \\ A'_u &:= A_u && \text{for } u \notin V \setminus \cup_{i \in U} V_{v_i} \end{aligned}$$

We have

$$\begin{aligned} \Delta D &= \Delta A_{s_t} - \sum_{i \in U} \Delta A_i \\ &= \delta + \sum_{i \in U} \left(A_{v_i'} - A_{p(v_i')} \right) && (23) \\ &\geq M_{d_{v_{i^*}'}} + \sum_{i \in U \setminus \{i^*\}} m_{d_{v_i}} && (24) \\ &\geq M_1 + (|U| - 1) m_1 && (25) \\ &\geq |U| = |U| + |B|. && (26) \end{aligned}$$

Equation (23) follows since the request location s_t lies in the component where altitudes are changed by δ and the updated server positions v_i' are all outside of it. Inequality (24) follows from rule (ii) of the induction hypothesis. Inequalities (25) and (26) follow by Lemma 13.

Case 2: $B = \{j\}$: Let $\delta \geq 0$ be the minimum of the set $\{A_{v_j} - A_{p(v_j)} - m_{d_{v_j}}\} \cup \{M_{d_{v_i'}} - A_{v_i'} + A_{p(v_i')} \mid i \in U\}$. We modify the altitudes as follows.

$$\begin{aligned} A'_u &:= A_u - \delta && \text{for } u \in V_{v_j} \setminus \cup_{i \in U} V_{v_i} \\ A'_u &:= A_u && \text{for } u \notin V_{v_j} \setminus \cup_{i \in U} V_{v_i} \end{aligned}$$

Again, rules (i) and (ii) are obeyed. We have

$$\begin{aligned} \Delta D &= \Delta A_{s_t} - \Delta A_j - \sum_{i \in U} \Delta A_i \\ &= \delta + A_{p(v_j)} - A_{v_j} + \sum_{i \in U} \left(A_{v_i'} - A_{p(v_i')} \right) \\ &= \begin{cases} -m_{d_{v_j}} + \sum_{i \in U} \left(A_{v_i'} - A_{p(v_i')} \right) \text{ if } \delta = A_{v_j} - A_{p(v_j)} - m_{d_{v_j}} \\ M_{d_{v_{i^*}}} + A_{p(v_j)} - A_{v_j} + \sum_{i \in U \setminus \{i^*\}} \left(A_{v_i'} - A_{p(v_i')} \right) \quad textif \delta = M_{d_{v_{i^*}'}} - A_{v_i'} + A_{p(v_{i^*}')} \text{ for } i^* \in U \end{cases} \end{aligned}$$

$$\geq \begin{cases} -m_{d_{v_j}} + \sum_{i \in U} m_{d_{v_i'}} \\ M_{d_{v_{i^*}}} - M_{d_{v_j}} + \sum_{i \in U \setminus \{i^*\}} m_{d_{v_i}} \end{cases}$$

$$\geq \begin{cases} -m_{d_{v_j}} + |U| m_{d_{v_j}+1} \\ M_{d_{v_j}+1} - M_{d_{v_j}} + (|U|-1) m_{d_{v_j}+1} \end{cases} \tag{27}$$

$$= -m_{d_{v_j}} + |U| m_{d_{v_j}+1} \tag{28}$$

$$\geq |U| + 1 = |U| + |B|. \tag{29}$$

Inequality (27) follows by Lemma 13(a) and since $d_{v_i'} \geq d_{v_j} + 1$ for all $i \in U$. Equation (28) follows by Lemma 13(b). Finally, inequality (29) follows by Lemma 13(d) if $d_{v_j} < d$; if $d_{v_j} = d$, then $|U| = 0$ and the inequality holds due to Lemma 13(a), which states that $m_d = -1$. This concludes the proof of inequality (22), and thereby Theorem 12.

6 Lower bounds

In this section we show lower bounds on the competitive ratio of DoubleCoverage, proving Theorem 2. Our inductive proofs crucially require internal tree vertices of degree greater than k.

6.1 Lower bound for depth d Trees

We will prove the following theorem, implying part (b) of Theorem 2.

Theorem 14 *The competitive ratio of* DoubleCoverage *for the k-taxi problem on unweighted tree metrics of depth d is at least*

$$4 \sum_{h=1}^{d-1} \binom{k+h-2}{h} + 2\binom{k+d-2}{d} + 1.$$

Notice that for $d = 1$, the lower bound is equal to $2k - 1$, and for $k = 2$ it is equal to $4d - 1$, and both of these cases match exactly our upper bound from Theorem 12. For constant d, the lower bound is at least $\Omega(k^d)$ as $k \to \infty$, matching our upper bound for weighted trees up to a constant depending on d.

We prove the lower bound on the k-ary tree of depth d, where each edge has length 1. We call a pair of online and offline configurations a *situation*. Consider a situation with the following properties:

- The location of j online servers matches that of j offline servers.
- Of the remaining online servers, at least one is located at a vertex x, and none is in the subtree below x.
- Of the remaining offline servers, at least one is located at a child y of x.

We call such a situation a *j-match around* (x, y).

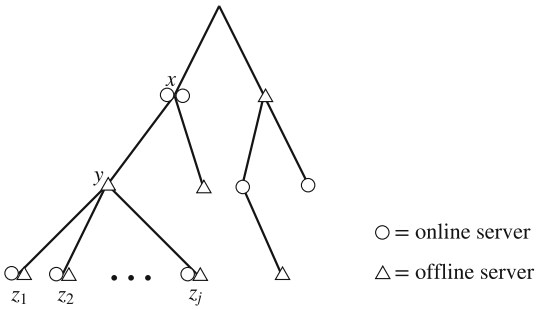

Fig. 3 A j-match around (x, y) (not all edges depicted)

Lemma 15 *Suppose the current situation is a j-match around (x, y) and let h be the height of y. There exists a request sequence on which* DOUBLECOVERAGE *suffers cost $2\binom{j+h}{h} - 1$, the offline algorithm suffers cost 0 and the resulting situation differs from the original one only in that there is one online server less at x and instead there is now an online server at y.*

Proof We proceed by induction on h.

If $h = 0$, we issue a single simple request at y. Upon this request, DOUBLECOVERAGE moves a single server from x to y for cost $1 = 2\binom{j}{0} - 1$. The request is free for the offline algorithm and the difference between the resulting and original situation is as desired.

If $h \geq 1$, we can assume that the j pairs of matching online and offline servers are located at j children $z_1, \ldots, z_j$ of y, as in Fig. 3. Indeed, this can be achieved by issuing j relocation requests from the locations of the j pairs of matching servers to the locations $z_1, \ldots, z_j$. Given a j-match of this form, we first issue a request at y, which moves the j online servers from $z_1, \ldots, z_j$ up to y and one online server from x down to y, overall incurring cost $j + 1$. Notice that the new situation is a 1-match around (y, z_i) for each $i = 1, \ldots, j$. Next, we will apply the induction hypothesis j times; we will maintain the invariant that before its ℓth application, the situation is an ℓ-match around (y, z_i) for each $i = \ell, \ldots, j$. By applying the ℓth application to the ℓ-match around (y, z_ℓ), the invariant is indeed maintained. After the last application of the induction hypothesis, we have a situation that differs from the original one in that an online server got removed from x and added to y, as desired. The offline cost of the sequence is 0 and the cost of DOUBLECOVERAGE is

$$j + 1 + \sum_{\ell=1}^{j} \left[2\binom{\ell + h - 1}{h - 1} - 1 \right] = 1 + 2 \sum_{\ell=1}^{j} \binom{\ell + h - 1}{h - 1} = 2\binom{j + h}{h} - 1,$$

where the last equation follows from the identity

$$\sum_{\ell=0}^{j} \binom{\ell + h - 1}{h - 1} = \binom{j + h}{h}.$$

□

We call a situation $(h, \uparrow)$-*situation* (resp. $(h, \downarrow)$-*situation*) if the location of $k-1$ online servers matches that of $k-1$ offline servers, the last online server is at a node x at height h and the last offline server is at the parent (resp. a child) of x. We say a *situation transforms to another situation at cost* c if there exists a request sequence that leads from the first to the second situation, incurring cost c for DOUBLECOVERAGE and cost 0 for the offline algorithm.

Lemma 16 *Let* $b_h = 2\binom{k+h-1}{h+1}$.

(a) *For* $h = 0, \ldots, d-2$, *any* $(h, \uparrow)$*-situation transforms to a* $(h+1, \uparrow)$*-situation at cost* b_h.
(b) *Any* $(d-1, \uparrow)$*-situation transforms to to a* $(d, \downarrow)$*-situation at cost* b_{d-1}.
(c) *For* $h = 2, \ldots, d$, *any* $(h, \downarrow)$*-situation transforms to a* $(h-1, \downarrow)$*-situation at cost* b_{h-2}.

Proof (a) Denote by x the vertex where the unmatched online server is located and by y the parent of x where the unmatched offline server is located. We first issue some relocation requests, so that one of the matching server pairs is at the parent z of y and the other $k-2$ matching server pairs are at distinct siblings $x_1, \ldots, x_{k-2}$ of x (see Fig. 4, left) Now request y. This results in all online servers moving to y for cost k. The offline servers are still at $z, y, x_1, \ldots, x_{k-2}$. Notice that for each $i = 1, \ldots, k-2$, the current situation is a 1-match around (y, x_i). We will apply Lemma 15 $k-2$ times: Before the ℓth application, the current situation is an ℓ-match around (y, x_i) for all $i = \ell, \ldots, k-2$. We can maintain this invariant by applying Lemma 15 to the ℓ-match around (y, x_ℓ). After all these applications of Lemma 15, there are two online servers at y (one of them matching the offline server at y) and the others are matching the offline servers at $x_1, \ldots, x_{k-2}$. Since y is at height $h+1$, and the last offline server at the parent z of y, we are now in a $(h+1, \uparrow)$-situation. The total online cost of the transformation is

$$k + \sum_{\ell=1}^{k-2} \left[2\binom{\ell+h}{h} - 1 \right] = 2 \sum_{\ell=0}^{k-2} \binom{\ell+h}{h} = 2\binom{k+h-1}{h+1} = b_h.$$

(b) The proof is identical to case (a) except that vertex z is now a child from y distinct from $x, x_1, \ldots, x_{k-2}$. (Notice that y is the root, since x is at height $d-1$.)
(c) In this case, the vertex y where the unmatched offline server is located is a child of the vertex x. We first issue some relocation requests so that the $k-1$ matched server pairs are at children $z_1, \ldots, z_{k-1}$ of y (see Fig. 4, right). Then we request y, forcing all online servers to move to y at cost k. The new situation is a 1-match around (y, z_i) for each $i = 1, \ldots, k-1$. We now apply Lemma 15 $k-2$ times, similarly to before, and after all these applications we reach a $k-1$-match around (y, z_{k-1}). Since y is at height $h-1$, this is also a $(h-1, \downarrow)$-situation. The total cost is obtained by the same calculation as in case (a), with h replaced by $h-2$ since the vertices z_i are at height $h-2$.

□

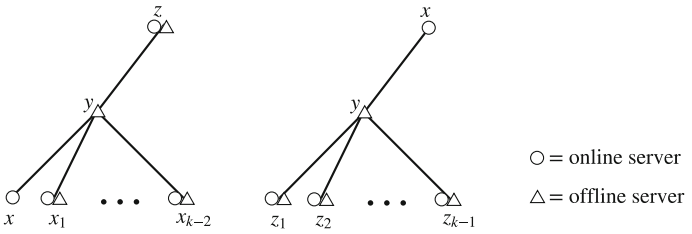

Fig. 4 A $(h, \uparrow)$-situation (left) and $(h, \downarrow)$-situation (right), if x is at height h

Proof of Theorem 14 From a situation where DOUBLECOVERAGE and the offline algorithm are in the same configuration, the offline algorithm can pay cost 1 to reach a $(0, \uparrow)$-situation. Now we successively apply all cases of Lemma 16, so that we eventually reach a $(h-1, \downarrow)$-situation. After one more request to the unmatched offline server, DOUBLECOVERAGE pays an additional cost 1, and the two algorithms are again in the same configuration. While the total cost of the offline algorithm was only 1, the cost of DOUBLECOVERAGE, and therefore a lower bound on its competitive ratio, is

$$2\sum_{h=0}^{d-2} b_h + b_{d-1} + 1 = 4\sum_{h=1}^{d-1}\binom{k+h-2}{h} + 2\binom{k+d-2}{d} + 1.$$

□

6.2 Lower bound for HSTs

We now prove that our analysis on HSTs is exactly tight for any depth by giving a matching lower bound of c_{kd} on the competitive ratio of DOUBLECOVERAGE, which yields part (a) of Theorem 2. We remark that for $d = 1$, the lower bound $c_{k1} = k$ already follows from the known lower bound on the k-server problem, and for $d \geq k$, the lower bound $c_{kd} = 2^k - 1$ follows from [10], where it was shown that even randomized algorithms against adaptive adversaries cannot achieve a better competitive ratio on HSTs of depth $d \geq k$.

For $\alpha \in \mathbb{N}$, let $T_{\alpha d}$ be an HST of depth d with edge lengths $\alpha^{d-1}, \alpha^{d-2}, \ldots, \alpha^0$ along each root-to-leaf path and where each internal vertex has sufficiently many (i,e., at least $k+1$) children. Let $W_{\alpha d} := \sum_{h=0}^{d-1} \alpha^h = \frac{\alpha^d - 1}{\alpha - 1}$ be the distance from the root to any leaf. The lower bound in Theorem 2 for HSTs follows from the following lemma by letting $\alpha \to \infty$.

Lemma 17 *For all $k \in \mathbb{N}_0$, $d \in \mathbb{N}_0$, $\alpha \in \mathbb{N}$, any initial configuration of k servers in $T_{\alpha d}$ and any leaf ℓ of $T_{\alpha d}$, there exists a request sequence with the following properties when being served:*

(a) *The upwards movement cost of* DOUBLECOVERAGE *is at least* $(\alpha - 1)^{d-1} c_{kd}$.
(b) *The upwards movement cost of an optimal offline algorithm is at most* $W_{\alpha d}$.
(c) *The cost of an algorithm with an additional* $(k+1)$*st server at* ℓ *is* 0.

(d) *If* DOUBLECOVERAGE *had an additional* $(k+1)$*st server sufficiently far away from the root on an extra edge incident to the root, this server would move distance* $W_{\alpha d}$ *towards the root.*

Proof We proof the lemma by induction on d. For $d = 0$, the empty request sequence trivially yields the result since $c_{k0} = W_{\alpha 0} = 0$.

Consider now the case $d \geq 1$. Denote by $S_0, S_1, \ldots, S_k$ different depth-$(d-1)$-subtrees, where S_0 is the one containing ℓ. Notice that each S_i is a copy of $T_{\alpha,d-1}$. We first issue relocation requests to ensure that for each $i = 1, \ldots, k$, there is a server at some leaf ℓ_i in S_i. We then issue a request at ℓ. To serve this request, the offline algorithm moves its server from ℓ_k to ℓ for upwards movement cost $W_{\alpha d}$, and it will suffer no additional cost for the remainder of the request sequence. In particular, this will ensure that an algorithm with an additional server at ℓ would suffer 0 cost, as required. DOUBLECOVERAGE moves its k servers to the root for cost $kW_{\alpha d}$ and then moves one server down to ℓ to serve the request. We will construct the remainder of the request sequence so that there are times $t_1 < \cdots < t_k$ such that at time t_i, the following holds:

- There are i online servers whose positions match those of i offline servers.
- The remaining $k - i$ online servers are at the root.
- The remaining $k - i$ offline servers are at $\ell_i, \ldots, \ell_{k-1}$.

Notice that initially, these properties are satisfied for $i = 1$. For $i < k$, the requests between times t_i and t_{i+1} are as follows: First we issue i relocation requests so that the i matching server pairs are in S_i. Now we issue the request sequence from the induction hypothesis applied to the subtree S_i, with i instead of k servers and with ℓ replaced by the vertex ℓ_i where the extra offline server is located. By property (c) of the induction hypothesis, this incurs no additional cost for the offline algorithm. By property (d) of the induction hypothesis, it will cause one of the online servers from the root of $T_{\alpha d}$ to move towards the root of S_i by distance $W_{\alpha,d-1}$. Thus, we can run the sequence from the induction hypothesis $\alpha - 1$ times and the server will move distance $\alpha^d - 1$ from the root of $T_{\alpha d}$ towards the root of S_i and hence it still has not reached S_i. In the end, we request the $i + 1$ offline server locations in S_i repeatedly until DOUBLECOVERAGE has a server at all these locations. Notice that the properties for time t_{i+1} are now satisfied. This completes the description of the request sequence.

The upwards movement cost of DOUBLECOVERAGE during the $\alpha - 1$ invocations of the induction hypothesis between times t_i and t_{i+1} is at least $(\alpha - 1)^{d-1} c_{i,d-1}$ due to property (a) of the induction hypothesis. Thus, the upwards movement cost during the entire request sequence is at least

$$kW_{\alpha d} + (\alpha - 1)^{d-1} \sum_{i=1}^{k-1} c_{i,d-1} \geq (\alpha - 1)^{d-1} c_{kd},$$

where we have used $W_{\alpha d} \geq (\alpha - 1)^{d-1}$ and the recurrence from Lemma 11. This proves property (a). As announced, the offline algorithm does not incur any additional cost beyond moving from ℓ_k to ℓ in the beginning, and therefore (b) and (c) are also satisfied. Property (d) holds because such an additional online server "above the root"

would be pulled down only during the initial stage where the k online servers move upwards by distance $W_{\alpha d}$ each. □

A limitation of previous techniques

A.1 Forward-time primal dual

We show that there exists no finitely competitive online algorithm for the dual problem, meaning that a forward-time construction of a dual solution (without knowing the future) would not have allowed us to obtain our result.

Proof of Theorem 4 Consider the tree containing only two leaves that are at distance 1 from the root. Consider an arbitrary deterministic online algorithm for the dual LP. (If the algorithm is randomized, then the same proof works by replacing altitudes with expected altitudes, so this is without loss of generality.) We will construct a request sequence for which the dual algorithm fails to achieve any positive objective value even though the optimal dual objective value (which is equal to the optimal objective value of the k-taxi problem) tends to infinity. Fix $k = 1$, and denote the single DOUBLECOVERAGE server by i. Note that this server is always located at the destination of the last request. The following proof easily extends to greater values of k by relocating *all* servers to the same leaf before issuing a simple request at the other leaf.[5]

Consider the first time t for which no request has been issued yet. Denote the two leaves by u and w such that $A_{u,t-1} \leq A_{w,t-1}$.

If the server is at w just before time t: We first issue a relocation request from $s_t = w$ to $d_t = u$ and then a simple request at $s_{t+1} = w$. At time t, the objective value changes by

$$D_t = -\Delta_t A_u = A_{u,t-1} - A_{ut},$$

and at time $t + 1$, it changes by

$$D_{t+1} = \Delta_{t+1} A_w - \Delta_{t+1} A_i = A_{w,t+1} - A_{wt} - A_{w,t+1} + A_{ut} = A_{ut} - A_{wt}.$$

[5] If one does not want to relocate several servers to the same vertex, one can also expand the two leaves to subtrees of small diameter and relocate the servers to the same subtree.

So overall, the objective value changes by

$$D_t + D_{t+1} = A_{u,t-1} - A_{wt} \leq A_{u,t-1} - A_{w,t-1} \leq 0,$$

where the first inequality is due to constraint (6) and the other inequality follows by definition of u and w.
If the server is at u just before time t: Then we issue a simple request at $s_t = w$. The objective value changes by

$$D_t = \Delta_t A_w - \Delta_t A_i = A_{wt} - A_{w,t-1} - A_{wt} + A_{u,t-1} = A_{u,t-1} - A_{w,t-1} \leq 0.$$

Thus, the dual objective value never increases, but any 1-taxi algorithm has to pay a constant movement cost for each simple request. □

Notice that the request at time t is chosen based on the altitudes at time $t-1$. When altitudes are constructed backwards in time, such an adversarial request sequence would not be well-defined.

A.2 Matching potential

Coester and Koutsoupias [10] gave a *randomized* algorithm for the k-taxi problem on HSTs which also achieves competitive ratio $2^k - 1$. The proof of competitiveness of this algorithm is given by a potential function argument, where the potential function is $2^k - 1$ times the value of a minimum matching between the online and offline servers. In a sense, the algorithm can be viewed as the randomized analogue of DoubleCoverage. Therefore, it is unsurprising that the same potential can be used to prove the same competitive ratio for DoubleCoverage when $k = 2$, as stated in [10]. However, they also mention that this potential fails for $k = 3$. Indeed, consider the depth-1-HST with four leaves a, b, c, d that are at distance 1 from the root $\mathbb{r}$. Consider the configuration with online servers at a, b, c and offline servers at b, c, d. The minimum matching has value 2. After a simple request at leaf d, DoubleCoverage has one server at d and the other two servers reside at the root $\mathbb{r}$, while the offline configuration is unchanged. Observe that the new minimum matching still has value 2. Thus, the online algorithm incurred cost for this request, but the potential and offline cost remain unchanged. Therefore, this potential cannot be used to prove competitiveness of DoubleCoverage on HSTs, even for $k = 3$. We were unable to find a pure potential proof that proves competitiveness of DoubleCoverage beyond the case $k = 2$.

B The transformed dual's dual

Instead of the primal LP defined in Sect. 3, we could have also defined the following different primal LP. It is less intuitive than the primal of Sect. 3, but it directly yields the altitude LP as its dual.

For $t \in T_s$ and $u \neq \mathbb{r}$, we use variables y_{ut} and z_{ut} for the numbers of servers leaving and entering subtree V_u at time t, as before. For $u = \mathbb{r}$, we view $y_{\mathbb{r}t}$ and $z_{\mathbb{r}t}$ as the constant 0. We also use variables x_{ut}, but with a different meaning than before: Now, x_{ut} denotes the number of servers *at* vertex u at time t that are *not* currently serving a simple request. Thus, $x_{ut} + \mathbb{1}_{\{u=s_t \text{ and } t\in T_s\}}$ is the total number of servers at vertex u at time t. We no longer use variables for the number of servers within a subtree. The following LP models the k-taxi problem on trees.

$$\min \quad \sum_{t\in T_s} \sum_{u\neq \mathbb{r}} (y_{ut} + z_{ut})$$

$$\begin{array}{lll}
(A_{u,t-1}) & y_{ut} - z_{ut} - \sum_{v\prec u} (y_{vt} - z_{vt}) = x_{u,t-1} + \mathbb{1}_{\{u=s_{t-1} \text{ and } t-1\in T_s\}} - x_{ut} - \mathbb{1}_{\{u=s_t\}} & \forall u \in V, t \in T_s \\
(A_{u,t-1}) & x_{ut} = x_{u,t-1} + \mathbb{1}_{\{u=d_t\}} & \forall u \in V, t \in T_r \\
 & y_{ut}, z_{ut} \geq 0 & \forall u \neq \mathbb{r}, t \in T_s \\
 & x_{ut} \geq 0 & \forall u \in V, t \in T_s \cup T_r
\end{array}$$

In the first constraint, the LHS and RHS are two different ways of writing the number of servers leaving vertex u at time $t \in T_s$. In particular, on the RHS we subtract the new number of servers at u, namely $x_{ut} + \mathbb{1}_{\{u=s_t\}} = x_{ut} + \mathbb{1}_{\{u=s_t \text{ and } t\in T_s\}}$, from the old number $x_{u,t-1} + \mathbb{1}_{\{u=s_{t-1} \text{ and } t-1\in T_s\}}$. Since $x_{ut} \geq 0$, it is guaranteed that there is at least one server at the requested location of each simple request. The second constraint guarantees that for $t \in T_r$, the server that was previously serving the simple request at $s_{t-1} = d_{t-1} = s_t$ is now located at d_t instead, and other servers remain at their old locations.

Before we construct the dual LP we will add the additional constraint

$$(A_{u,T}) \qquad 0 = x_{u,T} + \mathbb{1}_{\{u=s_t \text{ and } t\in T_s\}} - \bar{x}_{u,T} \qquad \forall u \neq \mathbb{r}$$

to the primal LP, where $\bar{x}_{u,T}$ are *constants* specifying the configuration of DOUBLECOVERAGE at the last time step. As before, this affects the optimal value by only an additive constant. We again also view $x_{u0} = \bar{x}_{u0}$ as constants describing the initial configuration of the servers (with the altered meaning of the variables x_{ut}). We can write the second primal constraint slightly more complicated by adding $\mathbb{1}_{\{u=s_{t-1} \text{ and } t-1\in T_s\}} - \mathbb{1}_{\{u=s_t\}}$ to its RHS, which is 0 because every relocation request at time $t \in T_r$ is preceded by a simple request at $s_t = s_{t-1}$. Then we obtain the following corresponding dual:

$$\max \quad \sum_{t\in T_s} \left[A_{s_t t} - A_{s_t,t-1}\right] + \sum_{t\in T_r} \left[A_{d_t,t-1} - A_{s_t,t-1}\right] + \sum_{u\in V} \left[\bar{x}_{u0} A_{u0} - \bar{x}_{u,T} A_{u,T}\right]$$

$$\begin{array}{ll}
A_{ut} - A_{p(u)t} \in [-1, 1] & \forall u \neq \mathbb{r}, t+1 \in T_s \\
\Delta_t A_u \geq 0 & \forall u \in V,\ t \in T_s \cup T_r
\end{array}$$

Expanding the term $\sum_{u\in V} \left[\bar{x}_{u0} A_{u0} - \bar{x}_{u,T} A_{u,T}\right]$ to a telescoping sum, we exactly recover the dual objective function from before.

References

1. Azar, Y., Buchbinder, N., Hubert 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, FOCS '96, pp. 184–193, (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., Buchbinder, N., Coester, C., Sellke, M.: Metrical service systems with transformations. In 12th innovations in theoretical computer science conference, ITCS 2021, vol. 185, pp. 21:1–21:20, (2021)
5. Bubeck, S., Cohen, M.B., Lee, Yin 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)
6. Buchbinder, N., Gupta, A., Molinaro, M., Naor, J(Seffi).: 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)
7. Buchbinder, N., Naor, J.: The design of competitive online algorithms via a primal-dual approach. Found. Trends Theor. Comput. Sci. **3**(2–3), 93–263 (2009)
8. Chrobak, M., Karloff, H., Payne, T., Vishwanathan, S.: New results on server problems. SIAM J. Discrete Math. **4**(2), 172–181 (1991)
9. Chrobak, M., Larmore, L.L.: An optimal on-line algorithm for k servers on trees. SIAM J. Comput. **20**(1), 144–148 (1991)
10. 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)
11. 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)
12. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**, 485–497 (2004)
13. 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)
14. Fiat, A., Rabani, Y., Ravid, Y.: Competitive k-server algorithms (extended abstract). In 31st Annual symposium on foundations of computer science, FOCS '90, pp. 454–463, (1990)
15. Gupta, A., Nagarajan, V.: Approximating sparse covering integer programs online. Math. Oper. Res. **39**(4), 998–1011 (2014)
16. Kosoresow, A.P.: Design and analysis of online algorithms for mobile server applications. PhD thesis, Stanford University, (1996)
17. Koutsoupias, E.: The k-server problem. Comput. Sci. Rev. **3**(2), 105–118 (2009)
18. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)
19. Lee, J.R.: Fusible HSTs and the randomized k-server conjecture. In Proceedings of the 59th Annual IEEE symposium on foundations of computer science, FOCS '18, pp. 438–449, (2018)
20. Lee, J.R.: Fusible HSTs and the randomized k-server conjecture. arXiv:1711.01789, February (2018)
21. Lee, J.R.: Personal Communication (2019)
22. Manasse, M., McGeoch, L., Sleator, D.: Competitive algorithms for on-line problems. In Proceedings of the twentieth annual ACM symposium on theory of computing, STOC '88, pp. 322–333. ACM, (1988)