Towards the Randomized k-Server Conjecture: A Primal-Dual Approach

(Extended Abstract)

Nikhil Bansal * Niv Buchbinder † Joseph (Seffi) Naor ‡

Abstract

Recently, Coté et al. [10] proposed an approach for solving the k-server problem on Hierchically Separated Trees (HSTs). In particular, they define a problem on a uniform metric, and show that if an algorithm with a certain refined guarantee exists for it, then one can obtain polylogarithmic (in diameter) competitive factors for the k-server problem on HSTs by solving this problem recursively. By designing such an algorithm for a two point metric, they obtained a logarithmic competitive algorithm for well-separated *binary* HSTs. Extending their result to uniform metrics on arbitrarily many points would imply a poly-logarithmic competitive algorithm for k-server on general HSTs (and hence general metrics) and is thus of major interest.

Here, we design such an algorithm for any uniform metric, provided the instance satisfies a certain "convexity" property. Even though this does not give a result for k-server, convexity seems to be a very natural property, and we give evidence that instances arising in the Coté et al. [10] reduction from k-server essentially possess this property, suggesting that this might be a promising approach. Already, our setting is general enough to model the finely competitive paging problem proposed by Blum et al. [4], who motivated it as a first step towards achieving a polylog(k) competitive algorithm for k-server. Our result implies an $r + O(\log k)$-competitive algorithm for finely competitive paging, resolving the main open problem of [4].

Our results are based on an extension of the primal-dual framework for online algorithms developed by Buchbinder and Naor [7]. The original approach works for problems whose offline version can be expressed as a packing or a covering linear program, possibly with box constraints. The online nature of the problem is modeled by revealing the constraints one by one and the requirement that variables can only be increased over time.

Here, we consider more general types of constraints, where terms can be both positive and negative. Moreover, we allow the variables to both increase and decrease. This versatility allows us to model problems such as predicting with expert advice, which could not be modeled earlier. To show the simplicity and generality of this approach, we give an alternate $O(\log k)$-competitive algorithm for weighted paging with a very simple proof. We also give an alternate primal-dual approach to design regret minimization algorithms for the problem of online prediction with expert advice. Our results suggest the possibility of a more general primal-dual framework for online problems beyond covering and packing LPs.

1 Introduction

The k-server problem is one of the most central and well studied problems in competitive analysis and is considered by many to be the "holy grail" problem in the field. In the k-server problem, there is a distance function d defined over an n-point metric space and k servers located at the points of the metric space. At each time step, an online algorithm is given a request at one of the points of the metric space, and it is served by moving a server to the requested point. The cost is defined to be the distance traveled by the server. Thus, the goal of an online algorithm is to minimize the total sum of the distances traveled by the servers so as to serve a given sequence of requests. The k-server problem can model many problems, and one of the most widely studied problems is *paging*. In this problem there is a cache that can hold up to k pages out of a universe of n pages. At each time step a page is requested; if the page is already in the cache then no cost is incurred, otherwise it must be brought into the cache (possibly evicting some other page) at a cost of one unit. Paging is the special case of k-server on a uniform metric. In

*IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. E-mail: *nikhil@us.ibm.com*

†Microsoft Research, Cambridge, MA. E-mail: *nivbuchb@microsoft.com*

‡Computer Science Dept., Technion, Haifa, Israel. E-mail: *naor@cs.technion.ac.il*. Research supported in part by ISF Grant 1366/07. Part of this research was done while visiting Microsoft Research and IBM T. J. Watson Research Center.

their seminal paper on competitive analysis, Sleator and Tarjan [19] gave k-competitive algorithms for paging, and also showed that this is the best possible for any deterministic algorithm.

The k-server problem in its full generality was first posed by Manasse et al. [16]. They conjectured that there is a k-competitive online algorithm in any metric space and for any value of k. This is called the *k-server conjecture*. Fiat et al. [12] were the first to prove that there exists an online algorithm for any metric space with competitive ratio which depends only on k. Following a sequence of results, a major breakthrough was achieved by Koutsoupias and Papadimitriou [15] who showed that the work function algorithm is $2k-1$ competitive. We note that for special metrics such as the uniform metric, lines, and more generally trees [9], a competitive factor of k is known for the k-server problem.

The competitive ratio for k-server seems to substantially improve if randomization is allowed. For example, Fiat et al. [13] gave a randomized $2H_k$-competitive algorithm for paging, where H_k is the k-th Harmonic number. They also showed that any randomized algorithm is at least H_k-competitive. In fact, no better lower bound is known for any other metric space. The *randomized k-server conjecture* states that there is an $O(\log k)$-competitive randomized algorithm for the k-server problem in an arbitrary metric space against an *oblivious* adversary. However, despite much interest, the randomized case remains poorly understood and $o(k)$ upper bounds are known for very few special cases. In addition to paging, poly-logarithmic competitive algorithms are known only for general metrics on $n = k+O(\text{polylog}k)$ points (via results for Metrical Task Systems), for certain well-separated spaces [17], and for weighted paging [2]. Even for the seemingly simple case of n uniformly spaced points on a line, the best known result is an $O(n^{2/3})$-competitive algorithm [11], which is $o(k)$-competitive only for $n = o(k^{3/2})$. Given our current lack of understanding, a major breakthrough would be to even resolve a weaker variant of the randomized k-server conjecture – obtaining a $\text{polylog}(k, n, \Delta)$-competitive algorithm for general metrics, where Δ is the diameter. Since a general metric space can be embedded into a probability distribution over Hierarchically Separated Trees (HSTs) with logarithmic distortion of the distances, it suffices to consider the k-server problem on HSTs to obtain the latter bound.

Very recently, Coté et al. [10] proposed an approach for obtaining non-trivial results for HSTs. They defined a problem on uniform metrics, which we call the *Allocation problem*, and showed that an algorithm with a guarantee of a certain refined form for it could be used as a building block to recursively solve the k-server problem on an HST. Coté et al. [10] were able to solve the Allocation problem of a metric space with two points, allowing them to obtain an $O(\log \Delta)$-competitive algorithm for well separated *binary* HSTs. Given the relative simplicity of uniform metrics, such an approach seems quite promising; we note that it has already been applied very successfully to the related (but easier) *metrical task systems* (MTS) problem [3, 14].

We build on the approach of Coté et al. [10] and design an algorithm for the Allocation problem on any uniform metric, provided the instance satisfies a certain "convexity" property. We call this problem the *Allocation-C* problem. Although we show that convexity does not hold directly in the Coté et al. [10] reduction, we do conjecture that it holds in an aggregate sense, allowing for the use of the Allocation-C problem to solving the k-server problem. Our ideas already yield a rather versatile approach allowing us to apply our techniques to a bunch of problems, defined formally in the next subsection. We then explain the approach in more detail.

1.1 Problem Definitions and Preliminaries.

We now define and elaborate on the problems we consider in the paper.

Metrical Task Systems and Unfair MTS. The MTS problem was introduced by Borodin et al. [6] as a generalization of various online problems. There is a server which can be in any one of n states $1, 2, \ldots, n$, and a metric d which defines the cost of moving between the states. At time step t, a task specified by a cost vector $c_t = (c_t(1), \ldots, c_t(n))$ arrives, representing the cost of processing the task in each of the states. Given a task, the server can choose to move to some other state j from its current state i while paying a cost of $d_{ij} + c_t(j)$. The goal is to minimize the total cost. Borodin et al. [6] gave a tight deterministic $2n-1$ competitive algorithm for any metric, and a $\Theta(\log n)$-competitive randomized algorithm for the uniform metric.

Later on, in a breakthrough paper, Bartal et al. [3] obtained the first polylogarithmic competitive algorithm for general metrics by recursively solving the problem on HSTs. To do this, [3] defined a more general problem called the unfair MTS problem on a uniform metric. It is similar to MTS, but has an extra unfairness parameter $r \geq 1$. Here, given vector c_t, the online algorithm (unfairly) pays $r \cdot c_t(i)$ in state i, while the offline algorithm pays $c_t(i)$. The movement costs are the same for both offline and online. Using techniques from online learning, [3] gave an algorithm, that given any $\epsilon > 0$, incurs a processing cost of $(1+\epsilon)$ times the total optimum cost, and a movement cost of

$O(\log(n/\epsilon))$ times the total optimum cost. This implies an $r + O(\log n)$ competitive algorithm for unfair MTS (in contrast, the usual MTS algorithm only gives an $O(r \log n)$ guarantee). Such a guarantee allows a solution of MTS on an HST recursively by solving it for uniform metrics. The idea is that each state in the unfair MTS problem represents a subtree of an HST and r models the competitive ratio achieved within this subtree. Applying the algorithm of [3] recursively on an HST, intuitively, $r + O(\log n)$ becomes the unfairness parameter for the next level of the HST (and so on), thus implying an $O(\ell \log n)$-competitive algorithm for an HST of depth ℓ. Bartal et al. [3] further refined their approach to remove dependence on the diameter of the HST. Currently, the best known guarantee for the MTS problem on an HST is $O(\log n \log\log n)$, which is almost optimal, due to Fiat and Mendel [14].

Finely Competitive Paging. Motivated by the unfair MTS approach, Blum, Burch and Kalai [4] proposed an unfair version of k-server on uniform metrics called finely competitive paging. Indeed, they view this problem as a possible first step towards achieving a polylogarithmic (in k) competitive randomized algorithm for k-server. Finely competitive paging is the following variation on the usual paging problem. Upon a request for page p, the algorithm need not necessarily fetch p into the cache. Instead it can "rent" it. However, renting only satisfies the current request, in particular, if p is requested at the very next time step, then again the algorithm must either fetch or rent p. The offline algorithm can rent at cost $1/r$, while the online algorithm pays cost 1.[1] Blum et al. [4] gave a $6(2.8r + 2\ln k)$-competitive algorithm for this problem (in contrast, the marking algorithm [13] is $O(r \log k)$-competitive). Blum et al. [4] state their main open question as designing an $r + O(\log k)$-competitive algorithm for the problem, or even a weaker $(1+\varepsilon)r + (1+\frac{1}{\varepsilon})O(\log k)$-competitive algorithm.

Allocation Problem. Recently, Coté et al. [10] gave the first completely formal approach to solving the k-server problem on HSTs by recursively solving a problem on the uniform metric. We call their problem the Allocation problem, defined as follows: there is a uniform metric on n points and there are up to k available servers. At time step t, the total number of available servers $k(t) \leq k$ is specified, and a request arrives at some point i_t. The request is specified by a $(k+1)$-dimensional vector $\vec{h}^t = (h^t(0), h^t(1), \ldots, h^t(k))$, where $h^t(j)$ denotes the cost when serving the request using j servers. Upon receiving a request, the algorithm may choose to move additional servers to the requested point and then serve it. The cost is divided into two parts. The *Move-Cost* incurred for moving the servers, and the *Hit-Cost*, $h^t(j)$, determined by the cost vector. The goal is to minimize the total cost. In addition, the cost vectors at any time are guaranteed to satisfy the following *monotonicity* property: for any $0 \leq j \leq k-1$, the costs satisfy $h^t(j) \geq h^t(j+1)$. That is, serving a request with less resources can only increase the cost.

Given an instance of the Allocation problem, let Optcost be its cost. Coté et al. [10] showed that if there is an online algorithm that incurs a hit cost of $(1+\varepsilon)$Optcost and a move cost of polylog$\cdot$Optcost, then there is a polylogarithmic competitive algorithm for the k-server problem on general metrics. For completeness, we describe their reduction, the formal statement of the above result, and its proof (with a somewhat different presentation) in Section 6. At a high level, each node in the HST runs an instance of the Allocation problem on the uniform metric formed by its children. In the Allocation problem, the cost vector at a node i at time t specifies the incremental cost incurred by the optimal k-server solution on the instance restricted to the subtree rooted at i.

Intuitively, the monotonicity condition seems very natural (having more servers can only help), however it is rather non-trivial to prove, since we are interested in the incremental cost at each time step rather than the total cost. Coté et al. [10] were able to show that monotonicity holds using quasi-convexity of the work function. Coté et al. [10] showed how to solve the Allocation Problem on a two point space, and gave a $(1, O(1))$ approximation for it. This allowed them to obtain an $O(\log D)$-competitive algorithm for a binary HST with $\alpha = \Omega(\log D)$, where D is the diameter of the HST. Thus, the main question is whether a similar result holds for the Allocation problem on a uniform metric with an arbitrary number of nodes. However, it is not clear how to extend their ideas to more than two points[2]. As they point out, for two points the problem is similar to MTS on a line (and their algorithm uses this structure critically).

Allocation-C Problem. We consider a restricted version of the Allocation problem, where the cost vectors satisfy the following additional *convexity* property.

[1] Blum et al. [4] use the version here interchangeably with the version where the online does not have the ability to rent at all. We discuss the implications of this issue in more detail in Section 4.

[2] Even if one could obtain a weaker O(polylog,polylog) competitive algorithm for the Allocation problem, this would imply a sub-linear $2^{\log^{1-\gamma} D}$ guarantee for k-server which would already be very interesting, given our current state of knowledge. But such an algorithm is not known either. Extending the result to more than two points seems to be the major difficuly.

for any j, $0 \leq j \leq k-2$, and time t:

$$h^t(j+1) - h^t(j+2) \leq h^t(j) - h^t(j+1).$$

That is, the additional marginal benefit from any additional server is decreasing. If we consider the vectors arising from the Coté et al. [10] reduction, this says that the reduction in cost (at any time), if an additional server is available, is lesser if there are more servers to begin with. While this condition seems quite natural, somewhat surprisingly we show in Section 5 that it does not always hold in the the reduction instance. Thus, an algorithm for the Allocation-C problem cannot be used directly in the reduction of Coté et al. [10]. However, as we discuss in detail later, the convexity property seems to hold in a more aggregate sense, which might suffice for solving the k-server problem. In Section 4 we show that finely competitive paging is a special case of the Allocation-C problem.

The Allocation-C problem can also be viewed as a natural resource allocation problem which may be of independent interest. Consider a project manager who runs projects in multi-site locations. The manager can use up to k resources (workers) to execute the projects. Each new project p is located in one of the n locations and can be executed by $i \leq k$ workers with cost (time) $h^p(i)$. We refer to the execution cost as the *service cost*. We note that both monotonicity and convexity are reasonable assumptions on service cost. Serving a request with more resources takes less time (cost), i.e., the monotonicity assumption. The convexity property requires that the marginal contribution of adding a resource is decreasing. (E.g., the reduction in time when executing the project with two workers instead of one is larger than the reduction in time when executing the project with three workers instead of two). In addition, there is a uniform cost in *moving* resources from one site to another. In the online setting, projects arrive one-by-one, and the algorithm has to execute all of them in an online fashion while minimizing the sum of service costs and movement costs.

1.2 Our Results and Techniques.

Our main result is stated in the following theorem:

THEOREM 1.1. *Let* OPT *be the optimal offline cost for the Allocation-C problem. For any $\varepsilon \leq 1$ there exists an online algorithm for it such that:*

- *The service cost is at most* $(1+\varepsilon)$OPT.
- *The movement cost is at most* $O(\log \frac{k}{\varepsilon})$OPT.

As a special case, we resolve the main open question of [4]:

THEOREM 1.2. *There is a* $\big((1+\varepsilon)r + O(\log \frac{k}{\varepsilon})\big)$*-competitive algorithm for the finely paging problem, for any $\varepsilon \leq 1$.*

In fact, as $r < k$ for all cases of interest (recall that r models the competitive ratio of a subtree in the HST), by setting $\varepsilon = 1/k$, we obtain an $r + O(\log k)$ guarantee which is the best possible.

To obtain these results, we work with an extension of the primal-dual framework for designing online algorithms developed by Buchbinder and Naor [7]. The approach of [7] applies to online problems that can be expressed as a covering/packing linear program, possibly with box constraints. In the online case constraints arrive sequentially over time, and each constraint needs to be satisfied upon arrival. Variables can only be increased over time, modeling the fact that decisions made in the past cannot be revoked. The main result of [7] is a (fractional) online algorithm with a logarithmic competitive ratio. The fractional algorithm can often be converted to a randomized algorithm, however, this part is problem specific. The primal-dual framework developed for covering/packing problems has been instrumental in viewing previous work on online algorithms in a unified way, as well as for obtaining new results. This includes, among others, the classic ski rental problem, the online set-cover problem, graph optimization problems, machine scheduling, ad auctions, weighted paging, and more. All these problems can now be solved using the basic recipe developed for online covering/packing problems, or simple tweaks thereof.

Extended Primal Dual Framework. Despite its success, there are two main limitations to this framework. First, the variables can only be increased over time, and this might be problematic for several natural classes of problems, e.g., online learning from expert advice. For this problem the variables naturally correspond to the probability mass on experts, which may both increase or decrease over time[3]. Second, requiring the constraints to be non-negative might be too restrictive for many problems, e.g., ruling out very simple constraints of the type $a_i \geq a_j$ or $a_i + a_j \geq a_k$.

The key technical contribution of this paper is to be able to work with more general forms of constraints with both *positive* and *negative* terms, and also allow variables to both increase and decrease over time. This allows the formulation of linear programs that seem very powerful and interesting.

New Simple Proofs. In addition to our main result for the Allocation-C problem, we also consider three

[3] At least we do not know how to write an LP for this problem where this is not the case.

(P)	$\min \sum_{p=1}^{n} \sum_{t=1}^{T} w_p \cdot z_{p,t} + \sum_{t=1}^{T} \infty \cdot y_{p_t,t}$		(D)	$\max \sum_{t=1}^{T} \sum_{S} (\lvert S \rvert - k) a_{S,t}$
$\forall t$ and $S \subseteq [n]$ with $\lvert S \rvert > k$	$\sum_{p \in S} y_{p,t} \geq \lvert S \rvert - k$		$\forall t$ and $p \neq p_t$	$\sum_{S: p \in S} a_{S,t} - b_{p,t+1} + b_{p,t} \leq 0$
$\forall t$ and page p	$z_{p,t} \geq y_{p,t-1} - y_{p,t}$		$\forall t$ and p	$b_{p,t} \leq w_p$
$\forall t$ and page p	$z_{p,t}, y_{p,t} \geq 0$		$\forall t$ and p and $\lvert S \rvert > k$	$a_{t,S}, b_{p,t} \geq 0$

Figure 1: Primal and dual LP formulations for the weighted caching problem.

other very well studied problems, and reprove known results for them via the extended primal-dual framework. First, we consider the *weighted caching* problem, and describe a new $O(\log k)$-competitive algorithm for it. This is a very natural and simple algorithm, and its proof turns out to be extremely simple. Second, we consider the *Unfair MTS* problem on a uniform metric, and reprove the $r + O(\log n)$-competitive algorithm for it [3], which was the key building block for MTS on general metrics. Third, we show how to derive optimal low regret algorithms for the classic problem of *predicting with expert advice* (one can also extend the techniques here to consider more fancy versions of expert problems, but we do not address this here). As we shall see, our algorithms for these problems are quite natural and simple, and have the same underlying analysis techniques. We believe that these techniques should find many other applications.

We note that at a high level, both online algorithms and learning from expert advice require dealing with an uncertain future based on the input observed thus far. This suggests the possibility of an underlying theory that unifies these two areas, and believe that our work might be step in this direction. In fact, the recent book by Cesa-Bianchi and Lugosi [8] on prediction and learning calls for the derivation of a general theory allowing a unified analysis of both types of problems.

Finally, we note that recently Shalev-Shwartz and Singer (e.g. [18]) have proposed a primal-dual approach for designing and analyzing algorithms for prediction from expert advice. However, their focus and techniques seem very different from ours. Their approach seems specifically tailored towards analyzing regret, and cannot model other types of costs such as "movement" costs that arise in typical online problems. At a high level, they show how weight update strategies for experts can be viewed as some form of gradient descent. Our approach, on the other hand, is the typical algorithmic approach of raising both the dual and primal together, while maintaining (approximate)-feasibility, and using the dual to pay for the primal.

Organization. The paper is organized as follows. In section 2, we study weighted caching, unfair MTS, and learning from expert advice using our extended primal-dual approach. This section serves as a gentle warm up to the primal-dual method and also develops most of the techniques that we need later. In section 3 we prove our main result for the Allocation-C problem. In section 4 we show how Allocation-C problem implies our result for finely competitive paging. In section 5 we discuss the applicability of the Allocation-C problem to the k-server problem. In particular, we show that convexity does not hold directly in the Coté et al. [10] reduction. However, we claim that it does hold in an aggregate sense, and state a conjecture and a possible approach such that the Allocation-C problem can be used for the k-server problem. Finally, in section 6 we describe the reduction of Coté et al. [10] for completeness, and to obtain explicit bounds on the guarantee for k-server given a bi-criteria guarantee for the Allocation problem.

2 Introducing the Basic Technique

In this section we demonstrate our technique by deriving optimal algorithms for three different well studied problems using similar ideas.

2.1 Weighted Caching. We describe a new $O(\log k)$-competitive algorithm for weighted caching, and give a much simpler proof than [2]. In the weighted caching problem there is a cache of size k, and pages $\{1, \ldots, n\}$ associated with weights $w_1, \ldots, w_n$. The weights denote the cost of fetching the pages into the cache. Pages are requested online and the goal is to minimize the total weight of pages we fetch into the cache. We describe a monotonic fractional $O(\log k)$-competitive algorithm for the problem. It can be converted to a randomized online algorithm with a constant factor loss in the competitive factor using the reduction in [2].

The primal and the dual LP formulations for the weighted caching problem appear in Figure 1. Note that this formulation is different from the one used by [2] as it contains both positive and negative variables. The variables $y_{p,t}$ denote the fraction of page p missing from the cache at time t. Let p_t denote the page requested at time t. The primal LP constraint states that at any time t, if we take any set S of pages, $|S| > k$, then the total number of pages outside the cache is at

least $|S| - k$. The variables $z_{p,t}$ denote the fraction of page p that is fetched at time t. The first term in the objective function is the fetching cost and the second term enforces the requirement that page p_t must be in the cache at time t. (Alternately, we could have just added the constraint $y_{p_t,t} = 0$, but the above provides a more unified view useful for later results). The dual of the LP consists of variables for each set S and time t, and variables for each page p and time t.

Algorithm. For each page p and time t, the following relation between primal and dual variables is maintained:

$$y_{p,t} := \frac{1}{k} \cdot \left(\exp\left(\frac{b_{p,t+1}}{w_p} \cdot \ln(1+k) \right) - 1 \right). \tag{2.1}$$

Consider a request for page p_t at time t. Initially, we set $y_{p,t} = y_{p,t-1}$ for all p, and hence $b_{p,t+1}$ is initialized to $b_{p,t}$. We set $y_{p_t,t} = b_{p_t,t+1} = 0$. Let $S = \{p : y_{p,t} < 1\}$. We start increasing $a_{S,t}$ and $b_{p,t+1}$ for all pages $p \in S \setminus \{p_t\}$ at the same rate. By (2.1), increasing $b_{p,t+1}$ causes these pages to be gradually evicted from the cache. The eviction of pages continues until the primal constraint for set S is satisfied. If some $y_{p,t}$ reaches the value of 1 during this process, we redefine $S = S \setminus \{p\}$, and continue on.

Analysis. We observe that both the primal and dual solutions are feasible at all times. In particular, $b_{p,t+1}$ never exceeds w_p for any page p, as (2.1) implies that $y_{p,t} = 1$ when $b_{p,t+1} = w_p$, and when this happens p is removed from S. Moreover, note that satisfying the primal constraint for the particular set $S = \{p : y_{p,t} < 1\}$, directly implies that the primal constraints for all sets are satisfied. Finally, the dual constraint is satisfied as $a_{S,t}$ and $b_{p,t+1}$ increases at the same rate for every $p \in S \setminus \{p_t\}$.

It remains to analyze the cost. Let P and D be the values of the primal and dual solutions produced by the algorithm. We prove that throughout the algorithm the derivative of P is at most $O(\log k)$ times the derivative of D. Suppose that at some point the algorithm increases the variable $a_{S,t}$, then:

$$\frac{\partial D}{\partial a_{S,t}} = |S| - k.$$

In the primal cost, let us consider eviction costs instead of fetching costs. (Clearly, this can add at most k times the maximum page weight to the overall total cost.) Since,

$$\frac{dy_{p,t}}{db_{p,t+1}} = \frac{\ln(1+k)}{w_p} \left(y_{p,t} + \frac{1}{k} \right),$$

and $b_{p,t+1}$ is being raised at the same rate as $a_{S,t}$, the derivative of the eviction cost in the primal is

$$\begin{aligned} &\sum_{p \in S \setminus \{p_t\}} w_p \cdot \frac{dy_{p,t}}{db_{p,t+1}} \\ &= \left(\sum_{p \in S \setminus \{p_t\}} \ln(1+k) \left(y_{p,t} + \frac{1}{k} \right) \right) \\ &\leq \ln(1+k) \left((|S| - k) + \frac{(|S|-1)}{k} \right) \\ &\leq 2\ln(1+k) \cdot (|S| - k) = 2\ln(1+k) \cdot \frac{\partial D}{\partial a_{S,t}}. \end{aligned}$$

The second step follows as $\sum_{p \in S} y_{p,t} < (|S| - k)$, and the last step follows as $(x-1)/k \leq x - k$ for $x \geq k+1$.

2.2 The Unfair MTS Problem. In this section we design a fractional $r + O(\log n)$-competitive algorithm for the unfair MTS problem in a uniform metric. The algorithm can be converted to a randomized online algorithm with a constant factor loss on the movement cost using the reduction in [2].

It is well known that we may consider only elementary tasks in which only single coordinate is ε and the rest of the coordinates are 0 . The primal and dual LP formulations for the problem appear in Figure 2. The variables $y_{i,t}$ denote the server fraction in leaf i at time t. Let p_t denote the location that has cost ε at time t. The primal LP constraint states that at any time t the number of servers is exactly 1. The variables $z_{i,t}$ denote the fraction of server in leaf i that arrives at time t. The first term in the objective function is the movement cost and the second term is the service cost. Note that the online service cost (in the unfair setting) is r times the offline cost. The dual of the LP consists of variables a_t for each time t, and variables for each leaf i and time t.

Algorithm: For each leaf i and time t, the following relation between primal and dual variables is maintained:

$$y_{i,t} := \frac{\gamma}{n} \left(\exp\left(\ln(1 + \frac{n}{\gamma}) b_{i,t+1} \right) - 1 \right). \tag{2.2}$$

The first observation is that:

$$\frac{dy_{i,t}}{db_{i,t+1}} = \ln(1 + \frac{n}{\gamma}) \cdot \left(y_{i,t} + \frac{\gamma}{n} \right)$$

Consider the request at leaf p_t at time t. Initially, we set $y_{i,t} = y_{i,t-1}$ for all i, and hence $b_{i,t+1}$ is initialized to $b_{i,t}$. As a result all the dual constraints for states $i \neq p_t$ are tight. We next start increasing a_t and $b_{i,t+1}$ at the same rate for all $i \neq p_t$. This keeps the dual constraints tight. However, this increases $y_{i,t}$ for all $i \neq p_t$. Since

(P)	$\min \sum_{i,t} z_{i,t} + \sum_{t=1}^{T} \varepsilon \cdot y_{p_t,t}$	(D)	$\max \sum_{t=1}^{T} a_t$
$\forall t$	$\sum_i y_{i,t} = 1$	$\forall t$ and $i \neq p_t$	$a_t + b_{i,t} - b_{i,t+1} \leq 0$
$\forall t$ and leaf i	$z_{i,t} \geq y_{i,t-1} - y_{i,t}$	$\forall t$	$-\varepsilon + a_t + b_{p_t,t} - b_{p_t,t+1} \leq 0$
$\forall t$ and leaf i	$z_{i,t}, y_{i,t} \geq 0$	$\forall t$ and i	$b_{i,t} \leq 1$

Figure 2: Primal and dual LP formulations for the unfair MTS problem.

we should keep $\sum_i y_{i,t} = 1$, we ensure that

$$\sum_{i \neq p_t} \frac{dy_{i,t}}{da_t} = -\frac{dy_{p_t,t}}{da_t}$$

Since

$$\frac{dy_{i,t}}{da_t} = \frac{dy_{i,t}}{db_{i,t+1}} = y_{i,t} + \frac{\gamma}{n} \qquad \text{for} \quad i \neq p_t$$

and

$$\frac{dy_{p_t,t}}{da_t} = \frac{dy_{p_t,t}}{db_{p_t,t+1}} \cdot \frac{db_{p_t,t+1}}{da_t} = (y_{p_t,t} + \frac{\gamma}{n}) \frac{db_{p_t,t+1}}{da_t}$$

We obtain that

$$\begin{aligned} -\frac{db_{p_t,t+1}}{da_t} &= \frac{\sum_{i \neq p_t} \left(y_{i,t} + \frac{\gamma}{n}\right)}{y_{p_t,t} + \frac{\gamma}{n}} \\ &\leq \frac{\gamma + \sum_{i \neq p_t} y_{i,t}}{y_{p_t,t}} \end{aligned} \tag{2.3}$$

Note, that $y_{p_t,t}$ is decreasing throughout the process and the rest of the $y_{i,t}$ variables are increasing. We stop the process if $y_{p_t,t+1} = 0$ or when the dual constraint of request $y_{p_t,t+1}$ becomes tight.

Analysis:. We observe that both primal and dual solutions are feasible at all times. In particular, $b_{i,t+1}$ never exceeds 1 for any leaf i, as (2.2) implies that $y_{i,t} = 1$ when $b_{i,t+1} = 1$, and when this happens we stop increasing a_t. Similarly, all primal constraints are satisfied. The dual constraint is satisfied as a_t and $b_{i,t+1}$ increase at the same rate for every $i \neq p_t$. The dual constraint of $y_{p_t,t}$ is satisfied by construction.

It remains to analyze the cost. To analyze the movement cost we analyze the derivatives of the primal and dual costs with respect to the increase in a_t. The derivative of the dual solution is simply 1. The derivative of the movement cost in the primal is:

$$\begin{aligned} \sum_{i \neq p_t} \frac{dy_{i,t}}{db_{i,t+1}} &= \sum_{i \neq p_t} \ln\left(1 + \frac{n}{\gamma}\right)\left(y_{i,t} + \frac{\gamma}{n}\right) \\ &\leq (1+\gamma) \cdot \ln(1 + \frac{n}{\gamma}) \end{aligned}$$

The inequality follows since $\sum_i y_{i,t} = 1$. Next we bound the service cost. If the iteration ends when $y_{p_t,t} = 0$ then the service cost is zero. Otherwise, we know that the dual constraint of the variable $y_{p_t,t}$ is tight. Therefore, we get:

$$\varepsilon = -\Delta b_{p_t,t+1} + a_t$$

Plugging in Inequality 2.3 (on the derivative), we get:

$$\varepsilon \leq a_t \cdot \left(\frac{\gamma + \sum_{i \neq p_t} y_{i,t}}{y_{p_t,t}} + 1\right),$$

and so the service cost is at most:

$$\begin{aligned} r \cdot \varepsilon y_{p_t,t} &\leq r \cdot a_t y_{p_t,t} \cdot \left(\frac{\gamma + \sum_{i \neq p_t} y_{i,t}}{y_{p_t,t}} + 1\right) \\ &= r \cdot a_t \cdot (1+\gamma) \end{aligned}$$

Since the change in the dual during the iteration is a_t we are done.

Thus, for any $\gamma < 1$, the algorithm is $(1+\gamma) \cdot r$-competitive for service cost, and is $(1+\gamma)\ln(1+\frac{n}{\gamma})$-competitive for movement cost. Since the interesting values of r in the unfair MTS problem satisfy $r < n$, we get by setting $\gamma = 1/n$ an $(r + O(\log n))$-competitive algorithm.

2.3 Predicting from Expert Advice. We now consider the classic problem of predicting from expert advice. The most basic version is the following. We are given n experts. At the beginning of time step t, the algorithm chooses an expert i_{t-1}. Then the cost vector $\vec{c}_t = (c_{1,t}, \ldots, c_{n,t})$ indicating the cost for each expert is revealed and the algorithm incurs a cost of $c_{i_{t-1},t}$. An equivalent problem is to maintain a distribution on the experts, and then the cost is defined to be the expected cost. We consider the usual setting of regret minimization. Here the performance of the online algorithm is compared with the performance of the best fixed expert (over time). We adopt the primal-dual approach, but indeed the algorithm that we get and analyze is essentially the well studied *weighted majority* algorithm (in disguise). We obtain the optimum regret $O(T \log n)^{1/2})$, where T is the number of time steps. As regret scales with cost, we assume that $\max_{i,t} c_{i,t} \leq 1$.

The primal and dual LP formulations for the problem of finding the best *fixed* or *static* expert appear in

(P)	$\min \sum_{t=1}^{T} \sum_{i=1}^{n} c_{i,t} \cdot y_{i,t-1} + \sum_{t=1}^{T} \sum_{i=1}^{n} \infty \cdot z_{i,t}$	(D)	$\max \sum_{t=0}^{T} a_t$
$\forall t \geq 0$	$\sum_{i=1}^{n} y_{i,t} = 1$	$\forall i$ and $t = 0$	$a_0 + \beta_{i,1} \leq 0$
$\forall t \geq 1$ and expert i	$z_{i,t} \geq y_{i,t} - y_{i,t-1}$	$\forall t \geq 1$ and i	$a_t - \beta_{i,t} + \beta_{i,t+1} \leq c_{i,t}$
$\forall t$ and expert i	$z_{i,t}, y_{i,t} \geq 0$	$\forall t \geq 1$ and i	$b_{i,t} \leq \infty$

Figure 3: The primal and dual LP formulations for the experts problem.

Figure 3. The variables $y_{i,t}$ denote the fraction we assign to expert i at time t. The primal LP constraint states that at any time t the sum of the fractions add up to exactly 1. The first term in the objective function is the cost of the experts and the second term enforces the condition that the LP cannot change its distribution over time. (Again, we could have removed this term and just put the constraints $y_{i,t} = y_{i,t-1}$ for all i and t, but the above view is more useful.) Thus, the LP finds the best static distribution on experts, or equivalently the best single expert. The dual of the LP consists of variables a_t for each time t, and variables for each expert i and time t.

Algorithm. Note that the primal program is now irrelevant since it captures the static experts problem. The online algorithm however can move between experts, but it compares itself to the dual of the static expert problem. The algorithm is going to maintain a relation between the primal and dual variables. At each step, $y_{i,t}$ will be a function of $\beta_{i,t+1}$. We define:

$$y_{i,t} := \exp\left(-\varepsilon \cdot \beta_{i,t+1}\right),$$

where $\varepsilon \leq 1$ will be a parameter in our algorithm. For each $1 \leq i \leq n$, we initialize $\beta_{i,1} = \frac{\log n}{\varepsilon}$, and hence $y_{i,0} = 1/n$. We also set $a_0 = -\frac{\log n}{\varepsilon}$ so that the first dual constraint is maintained.

Suppose a new cost vector $\vec{c}_t$ arrives at time t. The algorithm updates $\beta_{i,t+1}$ by setting

$$\beta_{i,t+1} \leftarrow c_{i,t} + \beta_{i,t}.$$

This update decreases $y_{i,t}$ resulting in a violation of constraint $\sum_{i=1}^{n} y_{i,t} = 1$. Then, the algorithm increases a_t from 0 (and correspondingly decreasing $\beta_{i,t+1}$ while maintaining the dual constraint $\beta_{i,t+1} = c_{i,t} + \beta_{i,t} - a_t$, until $\sum_{i=1}^{n} y_{i,t} = 1$ is satisfied.

Analysis. Clearly, by design, the algorithm guarantees that $\sum_i y_{i,t} = 1$. Similarly, by design, the algorithm also satisfies the dual constraints. Finally, we need to ensure that the variables $\beta_{i,t}$ are non-negative at all times. This is true since the function $\exp\left(-\varepsilon \cdot \beta_{i,t+1}\right) = 1$ when $\beta_{i,t+1} = 0$, and thus there is no reason for the algorithm to decrease any $\beta_{i,t+1}$ (by increasing a_t) beyond this value.

It remains to bound the online cost and relate it to the dual profit. At any time t we keep a distribution on the experts and therefore:

$$\begin{aligned} 1 &= \sum_{i=1}^{n} \exp\left(-\varepsilon \cdot \beta_{i,t+1}\right) \\ &= \sum_{i=1}^{n} \exp\left(-\varepsilon \cdot [c_{i,t} + \beta_{i,t} - a_t]\right) \\ &= \sum_{i=1}^{n} y_{i,t-1} \cdot \exp\left(\varepsilon \cdot [a_t - c_{i,t}]\right). \end{aligned}$$

Rearranging the expression and using $\sum_{i=1}^{n} y_{i,t-1} = 1$, we get:

$$\exp\left(-\varepsilon \cdot a_t\right) = \sum_{i=1}^{n} y_{i,t-1} \exp\left(-\varepsilon \cdot c_{i,t}\right). \tag{2.4}$$

Observe first that $a_t \leq \max_i c_{i,t} \leq 1$, otherwise the left-hand side above cannot be equal to the convex combination of $\exp\left(-\varepsilon \cdot c_{i,t}\right)$. Next, we use the fact that for any $0 \leq x \leq 1$ and $0 \leq \epsilon \leq 1$, $1 - \varepsilon x \leq \exp(-\varepsilon x) \leq 1 - \frac{\varepsilon}{1+2\varepsilon} x$. Using these bounds we get:

$$\begin{aligned} 1 - \varepsilon \cdot a_t &\leq \exp\left(-\varepsilon \cdot a_t\right) \\ &= \sum_{i=1}^{n} y_{i,t-1} \exp\left(-\varepsilon \cdot c_{i,t}\right) \\ &\leq \sum_{i=1}^{n} y_{i,t-1} \left(1 - \frac{\varepsilon}{1+2\varepsilon} \cdot c_{i,t}\right) \end{aligned}$$

Denote by D the value of the dual solution. Rearranging, we get:

$$\sum_{i=1}^{n} y_{i,t-1} c_{i,t} \leq (1+2\varepsilon) \cdot a_t$$

Next, summing up over all times we get:

$$\begin{aligned} \sum_{t=1}^{T} \sum_{i=1}^{n} c_{i,t} y_{i,t-1} &\leq (1+2\varepsilon) \left(\sum_{t=1}^{T} a_t\right) \\ &= (1+2\varepsilon) \left(\sum_{t=0}^{T} a_t\right) - (1+2\varepsilon) a_0 \\ &= (1+2\varepsilon) D + (1+2\varepsilon) \frac{\log n}{\varepsilon}. \end{aligned}$$

We set $\varepsilon = \sqrt{\frac{\log n}{T}}$ (and use the standard doubling trick when T is unknown) to obtain the guarantee on the regret.

3 The Allocation-C Problem

In this section we present our main result - an algorithm for the Allocation-C problem. To this end we define another problem called *caching with costs* and reduce the allocation-C problem to it.

Caching with Costs. A uniform metric on ℓ points is given. At each time step t, we are given the number of available servers $k(t)$, and a cost vector $c^t = (c^t_1, \ldots, c^t_\ell)$. After the cost vector is revealed, the algorithm is allowed to move servers between the points of the metric. Then, for each point i, a cost of c^t_i is incurred if no server is present at i at time t. Again, let $\sum_i c^t_i$ denote the *hit cost* at time t and let the cost of moving the servers be the *movement cost*.

LEMMA 3.1. *Any instance I of the Allocation-C problem can be reduced to an instance I' of the caching with costs problem. Given any solution to I, there is a solution to I' with similar or better cost, and vice versa. It is possible to convert I' to I in an online fashion.*

The lemma implies that a c-competitive algorithm for I' implies a c-competitive algorithm for I.

Proof. Given an instance of the Allocation-C problem on n points, consider the following instance of the Caching with Costs problem. We replace each point i, $i = 1, \ldots, n$, by k points $(i,1), \ldots, (i,k)$. This gives us $\ell = nk$. When a request $\vec{h}^t_i$ arrives at i in the Allocation-C problem, we have the associated vector $\vec{c}^t = (c^t_{1,1}, \ldots, c^t_{n,k})$ defined as follows: For $1 \le j \le k$, $c^t_{i,j} = (h^t_{i,j-1} - h^t_{i,j})$, and is 0 otherwise. For notational convenience, let $\vec{c}^t_i$ denote the vector $(c^t_{i,1}, \ldots, c^t_{i,k})$. One can view $\vec{c}^t$ as the collection of $\vec{c}^t_i$ for $i = 1, \ldots, n$.

By the monotonicity property, the vector $\vec{c}^t_i$ is non-negative, and the convexity property implies that $\vec{c}^t_i$ is monotonically decreasing. We claim that any solution to the caching with costs problem implies a solution to the Allocation-C problem that is at least as good, and vice versa. The idea is that whenever there are k_i servers at point i in the Allocation-C problem, we have one server each at points $(i,1), \ldots, (i,k_i)$ in the Caching with Costs problem. In particular, if there are k_i servers at i at time t in Allocation-C and we incur a hit cost of h^t_{i,k_i}, then the hit cost in the Caching with Costs problem is

$$\sum_{j=k_i+1}^{k} c^t_{i,j} = \sum_{j=k_i+1}^{k} (h^t_{i,j-1} - h^t_{i,j}) = h^t_{i,k_i} - h^t_{i,k}.$$

We can assume that $h^t_{i,k} = 0$ and hence the caching problem also pays h^t_{i,k_i}. (Otherwise, any solution incurs this cost, and a harder instance can be obtained by subtracting $h^t_{i,k}$ from each entry $h^t_{i,j}$. This would still maintain the monotonicity and convexity properties). Moreover, whenever a server is moved in the Allocation-C problem, say from i (with k_i servers) to j (with k_j servers), then after the reduction we move a server from point (i,k_i) to point (j,k_j+1). Both these steps incur a cost of 1.

Conversely, suppose the reduced problem has k_i servers located at a subset S of $\{(i,1), \ldots, (i,k)\}$. Then the hit cost is $\sum_{(i,j)\notin S} c^t_{(i,j)}$, which by the monotonicity of c^t_i is at least $\sum_{j=k_1+1}^{k} c^t_{(i,j)}$, which is equal to the hit cost in the Allocation-C problem. Note that this is where we use the convexity property of h^t or equivalently the monotonicity property of c^t. Finally, the movement cost in the allocation problem never exceeds that in the Caching with Costs problem, as The caching problem always pays a cost of 1 for moving a server. ■

We further simplify the problem using a standard reduction ([5, Lemma 9.1]).

LEMMA 3.2. *Given any cost vector $c = (c_1, \ldots, c_\ell)$, we can replace it by a sequence of c_i/ε vectors ϵ_i, for $i = 1, \ldots, \ell$, where ε is infinitesimally small and vector ϵ_i has cost ε at position i and 0 everywhere else.*

We skip the formal proof here, but the idea is that the reduction can only help the offline algorithm by giving it more flexibility to change its allocation, while harming the online algorithm as less information is revealed to it. Similarly, when the number of available servers $k(t)$ at time t changes from $k(t-1)$ and a cost vector c^t arrives, we break it into two steps. First, the number of servers changes (and a 0 cost vector arrives), and only then the cost vector c^t arrives.

3.1 An LP formulation for the Caching with Costs Problem.

Consider the following formulation. For each time step we have the indicator variable $y_{i,t}$ which is 1 if the server is absent at point i, and 0 otherwise. Without loss of generality, we assume that we only need to pay for moving a server to a point, i.e., we pay $y_{i,t-1} - y_{i,t}$ if $y_{i,t} < y_{i,t-1}$, but not if $y_{i,t} > y_{i,t-1}$. By Lemma 3.2 above, let p_t denote the location that has non-zero cost c_{p_t} at time t. Let T be the time horizon.

$$(P) \qquad \min \quad \sum_{t=1}^{T} c_{p_t,t} \cdot y_{p_t,t} + \sum_{i=1}^{n} \sum_{t=1}^{T+1} z_{i,t}$$

For each location i with (without) a server initially:

$$y_{i,0} \quad = \quad 0 \ (1)$$

For any time $t \geq 1$ and each subset of locations S:

$$\sum_{i:i\in S} y_{i,t} \quad \geq \quad |S| - k(t) \tag{3.5}$$

For each location i and time $1 \leq t \leq T+1$:

$$z_{i,t} \quad \geq \quad y_{i,t-1} - y_{i,t} \tag{3.6}$$

For each i and t: $y_{i,t}, z_{i,t} \geq 0$

Consider the dual of this program. We associate the variable $\alpha_{t,S}$ with constraints of the type (3.5), and variables $\beta_{i,t}$ with variables of type (3.6). We will also have variables γ_i for each initial location. This gives the following dual program.

$$(D) \qquad \max \sum_{t=1}^{T}\sum_{S}(|S|-k(t))\alpha_{t,S} + \sum_{i|\text{ no server at } i} \gamma_i$$

For each location i and time $1 \leq t \leq T$:

$$\sum_{S:i\in S} \alpha_{t,S} + \beta_{i,t} - \beta_{i,t+1} \leq c_{i,t} \tag{3.7}$$

For each location i and time $t = 0$: $\gamma_i \geq \beta_{i,1}$

For each i and time $1 \leq t \leq T+1$:

$$\beta_{i,t} \leq 1 \tag{3.8}$$

For each i and time t and set S: $\beta_{i,t}, a_{t,S} \geq 0$

3.2 The Algorithm. We first give a fractional algorithm for the problem. In Section 3.4 we will show how to round this solution in an online manner to obtain a randomized algorithm. Our rounding will preserve the hit cost and increase the movement cost by a factor of 2.

The algorithm maintains both (feasible) primal and dual solutions and also maintains a connection between the primal and dual variables. Specifically, we maintain at all times:

$$y_{i,t} = \frac{\varepsilon}{1+k}\left(\exp\left(\ln(1+\frac{1+k}{\varepsilon})\cdot\beta_{i,t+1}\right) - 1\right). \tag{3.9}$$

Note that $y_{i,t} = 0$ for $\beta_{i,t+1} = 0$ and $y_{i,t} = 1$ for $\beta_{i,t+1} = 1$. Note also that k is an upper bound on any $k(t)$. Another observation is that:

$$\frac{dy_{i,t}}{d\beta_{i,t+1}} = \ln\left(1+\frac{1+k}{\varepsilon}\right)\cdot\left(y_{i,t}+\frac{\varepsilon}{1+k}\right). \tag{3.10}$$

The algorithm works as follows. Initially, we set $\beta_{i,1} = \gamma_i = 1$ for all non-servers and $\beta_{i,1} = \gamma_i = 0$ for all servers. This is a feasible solution for the dual program at time 0.

Suppose we get a hit cost c_{i_t} at location i_t. We start by setting $b_{i,t+1} = b_{i,t}$ for all i. Let S be the set of points i such that $y_{i,t} < 1$. If the primal constraint (3.5) of S is not tight, we decrease first $\beta_{i_t,t+1}$ until the constraint becomes tight, or $y_{i_t,t}$ is zero. Next, if the primal constraint of S is tight, we start increasing $\alpha_{t,S}$ with rate 1, and $\beta_{i,t+1}$ for each $i \neq i_t$ also with rate 1. We then decrease $\beta_{i_t,t+1}$ with rate that is exactly enough to keep the primal constraint tight, by the following equality:

$$-\frac{dy_{i_t,t}}{d\beta_{i_t,t+1}}\cdot\frac{d\beta_{i_t,t+1}}{d\alpha_{t,S}} = \sum_{i\neq i_t, i\in S}\frac{dy_{i,t}}{d\alpha_{t,S}} = \sum_{i\neq i_t, i\in S}\frac{dy_{i,t}}{d\beta_{i,t+1}}.$$

Plugging in Equality 3.10 we get that:

$$-\frac{d\beta_{i_t,t+1}}{d\alpha_{t,S}} = \frac{\sum_{i\neq i_t, i\in S}\left(y_{i,t}+\frac{\varepsilon}{1+k}\right)}{y_{i_t,t}+\frac{\varepsilon}{1+k}}. \tag{3.11}$$

We continue with this rate until $y_{i_t,t} = 0$ or the dual constraint (3.7) of $y_{i_t,t}$ is tight. Next, we handle changes in $k(t)$. If $k(t)$ increases, so $k(t) > k(t-1)$ (we have more servers), then the algorithm simply sets $y_{i,t} = y_{i,t-1}$ for all i and does nothing else. Note that this keeps the primal and dual feasible and both primal cost and dual profit are 0. If $k(t) < k(t-1)$, then the algorithm initializes $y_{i,t} = y_{i,t-1}$ and sets S to be the set of all locations i such that $y_{i,t} < 1$. It starts increasing $a_{t,S}$ for this set S and $\beta_{i,t+1}$ for all $i \in S$ until the primal constraint (3.5) of the set is tight again.

3.3 Analysis. In this section we prove Theorem 1.1 that states the performance of the algorithm.

Proof. First, we show that the primal and dual solutions are feasible. The primal solution is feasible by design, since the algorithm always ensures the covering constraints, and sets $z_{i,t}$ to keep the constraints feasible. Similarly, the dual constraint (3.7) is satisfied by design. Constraint (3.8) is satisfied since i is removed from S whenever $\beta_{i,t+1}$ reaches 1, and hence it does not increase beyond. Similarly, $\beta_{i,t+1}$ is never decreased below 0, as $y_{i,t}$ never falls below 0. We are now ready to bound the cost incurred by the algorithm.

Bounding the movement cost. Without loss of generality we will pay movement cost only when we increase $y_{i,t}$ and not while decreasing. This may add only additive term to the competitive ratio. We increase variables $y_{i,t}$ only when the primal constraint of some

set S is tight or violated. In this case we also increase $\alpha_{t,S}$ and get some dual profit. There are two cases, one in which we decrease $y_{i_t,t}$ and increase all other $y_{i,t}$ for $i \in S$ and the other when $k(t)$ decreases and we increase all $y_{i,t}$. We analyze the first case and show that the movement cost derivative can be bounded by the derivative of the dual profit, the second case is similar. The derivative of the dual profit is simply $|S| - k(t)$. The derivative of the movement is at most:

$$\begin{aligned} & \sum_{i \neq i_t, i \in S} \frac{dy_{i,t}}{d\beta_{i,t+1}} \\ = \quad & \ln\left(1 + \frac{1+k}{\varepsilon}\right) \cdot \sum_{i \neq i_t, i \in S} \left(y_{i,t} + \frac{\varepsilon}{1+k}\right) \\ (3.12) \leq \quad & 2\left(|S| - k(t)\right) \ln\left(1 + \frac{1+k}{\varepsilon}\right) \end{aligned}$$

where (3.12) follows since we increase $y_{i,t}$ only when their sum is at most $|S|-k(t)$, and since $|S| \geq k(t)+1$ we have $(|S|-1)/(1+k) \leq (|S|-1)/(1+k(t)) \leq (|S|-k(t))$.

Bounding the hit cost. Consider times in which we incur a hit cost. For analysis purpose we separate two cases. One in which the primal constraint (3.5) of S is tight, and a second in which the primal constraint is not tight. Consider the first case. We decrease $y_{i_t,t}$ until either it is zero, or the dual constraint (3.7) of it is tight. We partition further the cost $c_{i_t,t}$ and assume that we only increase a single $\alpha_{t,S}$. This only make us pay more, since $y_{i_t,t}$ will decrease more later (and we don't take this into account). If $y_{i_t,t}$ becomes zero we have no hit cost. Otherwise, we know that:

$$-\Delta\beta_{i_t,t+1} + \alpha_{t,S} = c_{i_t,t} \tag{3.13}$$

However, by equality (3.11) we have an upper bound:

$$\begin{aligned} -\frac{d\beta_{i_t,t+1}}{d\alpha_{t,S}} \quad &= \quad \frac{\sum_{i \neq i_t, i \in S}\left(y_{i,t} + \frac{\varepsilon}{1+k}\right)}{y_{i_t,t} + \frac{\varepsilon}{1+k}} \\ (3.14) \qquad &\leq \quad \frac{\epsilon(|S| - k(t)) + \sum_{i \neq i_t, i \in S} y_{i,t}}{y_{i_t,t}}. \end{aligned}$$

Combining (3.13) and (3.14) we get that:

$$\begin{aligned} & c_{i_t,t} y_{i_t,t} \\ \leq \quad & \left(\epsilon(|S| - k(t)) + \sum_{i \neq i_t, i \in S} y_{i,t}\right) \alpha_{t,S} + \alpha_{t,S} y_{i_t,t} \\ = \quad & \alpha_{t,S} \cdot \left(\epsilon(|S| - k(t)) + \sum_{i \in S} y_{i,t}\right) \\ = \quad & \alpha_{t,S}(|S| - k(t))\left(1 + \epsilon\right). \end{aligned}$$

Consider now cases in which we get hit cost but the primal constraint of the set $S = \{y_{i,t} | y_{i,t} < 1\}$ is not tight. In general we keep the primal constraint of S tight at all times except when $k(t)$ is changing. When $k(t)$ decreases (the constraint is more difficult to satisfy) we increase $\alpha_{t,S}$ gain dual profit, but pay no hit cost. The constraint of S may become un-tight only when $k(t)$ increases (the constraint becomes easier). In this case the algorithm does nothing. However, later, when there some cost arrives at point i_t we decrease $\beta_{i_t,t+1}$ and so $y_{i_t,t}$ until it becomes zero or until the constraint of the set S becomes tight again. If $y_{i_t,t}$ is not zero when S becomes tight, then we incur hit cost, but have no dual profit in the current iteration. To pay for such an event we look for for a previous decrease in $k(t)$ in which we gained dual profit, but did not pay any hit cost. In general, $k(t)$ is increasing and decreasing throughout the execution, so we can match each increase to a previous decrease except for a constant number of the increases that depends on the initial and final values of $k(t)$. We claim that each decrease of $k(t)$ can pay for a later increase.

To see that, suppose $k(t)$ has increased and the primal constraint is not tight, then instead of decreasing the variable $y_{i_t,t}$ at the time we incur the hit cost, we can instead pretend that this hit cost arrived at the time when we decrease $k(t)$ and decrease $y_{i_t,t}$ at that time. We do this using the same process as in the previous case when the constraint is tight. Note that we exactly keep equality 3.13 as this is exactly the amount we are allowed to decrease $\beta_{i_t,t+1}$. Note also that as in the case of the tight constraint, we increase all y's by increasing $\alpha_{t,S}$ and decrease one i_t by decreasing $\beta_{i,t+1}$. ■

3.4 Converting the Fractional Solution into a Randomized Algorithm.

We show that the fractional solution can be mapped to a distribution on cache states in an online manner. Moreover, this can be done such that the hit cost incurred in both solutions is the same, and the movement cost increases by at most a factor of 2. We use the same technique as in [4].

Given any fractional solution $y_{i,t}$ at time t, suppose we have a distribution on cache states such that page i is present in the cache with probability $x_{i,t} = 1 - y_{i,t}$. The expected hit cost of the online algorithm at time t is exactly the same as that of the fractional solution.

Now, suppose the distribution changes at time $t+1$. Suppose that $x_{i,t}$ decreases by ε and $x_{j,t}$ is increasing by ε. The fractional cost in this case is ε (and any move can always be decomposed into such moves with the same total cost). Remove a server from location i from ε fraction of the configurations and add a server to location j in ε fraction of the configurations that do not have a server in location j. This is done with cost ε.

After this step there might be at most ε configurations with $k-1$ servers, call these deficient configurations, and at most ε fraction of the configurations with $k+1$ servers, call these excess configurations. Note that the measure of deficient configurations is exactly the same as that of excess configurations. We arbitrarily match the deficient and excess configurations. In each such matching there exists a location in which the excess configuration has a server and the deficient configuration does not have a server. Thus, we add a server to the location in the deficient configuration and remove a server from the excess configuration, and pay a cost of another ε. Thus, our online algorithm pays at most 2ε.

4 Finely-Competitive Paging

Recall that in finely competitive paging, the offline algorithm is allowed to "rent" a page at cost $1/r$, while the online algorithm can only rent at cost 1. Both the offline and the online algorithms pay 1 to fetch a page.

Remark: In their original formulation, [4] do not allow the online algorithm to rent a page, even though in their analysis they allow it to rent at cost 1. This is fine for obtaining a guarantee of the form $O(r + \log k)$ as in [4], since the ability to rent or not can only change the online algorithm's cost by a factor of 2 (if the online algorithm is not allowed to rent, it can simulate renting p' by evicting some page p, fetching in p', then evicting p', and fetching p again). However, as we show next, this (rather technical) distinction matters if one is interested in a guarantee of the form $(1+\varepsilon)r + O(\log k)$. In particular, we show that if the online algorithm cannot rent, then one cannot hope to get a guarantee better than $2r + O(\log k)$.

Suppose we have a cache of size k and $n = k+1$ pages. The request sequence is composed of $m \gg 1$ phases. In each phase pages 1 through k are requested cyclically for, say, a 100 times, followed by a single request to page $k+1$. An offline algorithm that is allowed to rent a page at cost $1/r$, will keep pages 1 to k in memory during all m phases and rent $k+1$ once in each phase, thus incurring a total cost of m/r. However, any optimal solution for an algorithm that cannot rent pages must incur two page faults in each phase, one for $k+1$, and another for some other page in $1, \ldots, k$, since it is very non-optimal to keep $k+1$ in the cache between two consecutive requests for it. Thus, its payment is $2m$. Since r is allowed to be arbitrarily large, say $r \gg \log k$, one cannot hope for a guarantee better than $2r + O(\log k)$.

We show that finely competitive paging is equivalent to a special case of the Allocation-C problem. If a request for page p arrives at time t in the paging instance, we give a cost vector $c = (c_0, \ldots, c_k) = (1/r, 0, 0, \ldots, 0)$ at time t in the Allocation-C instance. It is easily seen that any (possibly fractional) solution to the paging instance is also a solution with the same cost for the allocation-C problem, and vice versa. In both problems, the algorithm pays $1/r$ if page p is not fetched into the cache at time t, and otherwise pays 0. Moreover, the cost of fetching a page is identical to movement cost in the allocation-C problem.

Let H^* and M^* denote the optimum hit cost and movement costs for the fractional allocation-C problem. By Theorem 1.1, there is an online algorithm for the allocation-C problem that incurs a hit cost of $(1+\varepsilon)(H^*+M^*)$ and a movement cost of $O(\log(k/\varepsilon))(H^*+M^*)$. Now, consider the online version of finely competitive paging where the online algorithm pays 1 (instead of $1/r$) to rent a page. Using the same solution as determined by the Allocation-C problem, the fractional paging algorithm incurs a rental cost of $r(1+\varepsilon)(H^*+M^*)$ and a fetching cost of $O(\log(k/\varepsilon))(H^*+M^*)$. Since H^*+M^* is a lower bound on the optimum cost, and the rounding process for the fractional online algorithm only incurs an O(1) multiplicative cost in the fetching cost and no extra cost for the renting cost, this implies an $r(1+\varepsilon)+O(\log(k/\varepsilon))$-competitive algorithm for finely competitive paging.

5 How Restrictive is the Convexity Assumption

We now come back to our main motivation for studying the allocation Problem. If the allocation problem obtained in the Coté et al. [10] reduction from k-server were to satisfy the convexity property, then by Theorem 1.1 we would be done. We investigate this possibility here.

The Coté et al. [10] reduction and its connection to Theorem 1.1 are described in detail in Section 6, and we discuss only the relevant part here. Let i be a non-leaf node in the HST, and let $T(i)$ denote the subtree rooted at i. Then, there is an instance of the allocation problem running on the uniform metric formed by the children of i; call this problem $A(i)$. Let ρ be a k-server request sequence ρ (and assume wlog that all requests in ρ are at leaves in $T(i)$). Let $\text{Optcost}(i,j,t)$ denote the optimum cost of serving the requests until time t with j servers. To make Optcost well defined, assume that at $t=0$, the servers are at the first j nodes $1, \ldots, j$ in $T(i)$. Let $\Delta\text{Optcost}(i,j,t) = \text{Optcost}(i,j,t) - \text{Optcost}(i,j,t-1)$.

Suppose at time t, the request arrives at some leaf in the subtree $T(c)$, for some child c of i. In the reduction of Coté et al. [10], in the allocation problem $A(i)$, the cost vector

$$h(t) = (\Delta\text{Optcost}(c,0,t), \ldots, \Delta\text{Optcost}(c,k,t))$$

arrives at c. Recall that $\text{Optcost}(c, j, t)$ is the incremental optimum cost of the "j-server" solution for the instance restricted to $T(c)$. Intuitively, we would expect that $\Delta\text{Optcost}(c, j, t) \geq \Delta\text{Optcost}(c, j+1, t)$ for any j, i.e. the monotonicity property of $h(t)$. Indeed, why would the optimum solution with $j+1$ serves incur a higher cost for a request with j servers? Surprisingly, the proof of this is non-trivial and was shown using quasi-convexity of the work function. Roughly speaking, the trouble is that we care about the incremental cost at each step (and not the aggregate cost).

Now, our convexity condition in addition requires that for any j, the following holds:

$$\begin{aligned} & \Delta\text{Optcost}(c, j, t) - \Delta\text{Optcost}(c, j+1, t) \\ \geq \; & \Delta\text{Optcost}(c, j+1, t) - \Delta\text{Optcost}(c, j+2, t). \end{aligned}$$

That is, the (incremental) cost benefit for the optimum solution, when the number of servers increases from j to $j+1$, must be at least as large as the cost benefit when the number of servers increases from $j+1$ to $j+2$. Even though quite natural, it turns out that the convexity property does not hold at every step. Consider c to be a star with, say, $\ell = 3$ leaves and suppose requests arrive at 1,2,3 repeatedly in that order. With 1 server, any solution always pays 1 at each step, and hence the vector $\Delta\text{Optcost}(c, 1) = (1, 1, 1, 1, 1, 1, 1, \ldots)$. With 2 servers, the solution will pay 1 at every alternate time step, and hence $\Delta\text{Optcost}(c, 2) = (0, 1, 0, 1, 0, 1, 0, \ldots)$. With 3 servers, clearly the cost is 0 at all times, and hence $\Delta\text{Optcost}(c, 3) = (0, 0, 0, 0, 0, \ldots)$. By comparing these vectors at each co-ordinate t, note that convexity does not hold at $t = 2, 4, 6, \ldots$.

However, the above example is rather unsatisfying. The convexity condition seems to be violated due to the "discretness" of the distances. For example, convexity would be true if we could view $\Delta\text{Optcost}(c, 2)$ as $(1/2, 1/2, 1/2, 1/2 \ldots)$ instead of alternating 0s and 1s. In fact, we are unable to construct stronger counter-examples to convexity. We conjecture that all violations to the convexity property are of the above type. That is, if we could aggregate the costs over few time steps (instead of looking at each individual time step), the problem would essentially disappear.

CONJECTURE 5.1. *For any HST $T(c)$ rooted at c and any $t_1 \leq t_2$:*

$$\begin{aligned} & \sum_{t=t_1}^{t_2} (\Delta\text{Optcost}(c, j, t) + \Delta\text{Optcost}(c, j+2, t)) \\ \geq \; & 2\left(\sum_{t=t_1}^{t_2} \Delta\text{Optcost}(c, j+1, t)\right) - O(1) \cdot D(T(c)) \end{aligned}$$

where $D(T(c))$ is the diameter of $T(c)$.

If true, this conjecture (or even a weaker version of it with the $O(1)$ term replaced by $O(\text{polylog})$) should suffice by adapting the Coté et al [10] approach. Instead of giving a cost vector at every time step, aggregate enough cost vectors such that the effect of the $O(1)$ term becomes negligible. This is acceptable, as the costs vectors at node c are used by the allocation problem running at i, the parent of c, to determine how to distribute the servers among c and its siblings. This distribution is affected by costs of the order of $\text{D}(T(i)) = \alpha\text{D}(T(c))$. Thus, if α is chosen treasonably (poly-logarithmic) large, one can subsume the effect of the $O(1) \cdot \text{D}(T(c))$ term.

In fact, one can easily show the following weaker variant of Conjecture 5.1.

PROPOSITION 5.1. *For any HST, $T(c)$ rooted at c, and times t_1, t_2:*

$$\begin{aligned} & \sum_{t=t_1}^{t_2} (\Delta\text{Optcost}(c, j, t) + \Delta\text{Optcost}(c, j+2, t)) \\ \geq \; & 2\left(\sum_{t=t_1}^{t_2} \Delta\text{Optcost}(c, j+1, t)\right) - O(k) \cdot D(T(c)). \end{aligned}$$

This follows by viewing a k-server solution as a flow of k units, and observing that by averaging flows of value k and $k+2$, corresponding to k and $k+2$ server solutions, gives a flow corresponding to a valid $k+1$-server solution. The $O(k) \cdot \text{D}$ term is incurred due to the cost paid to make flows consistent at the boundaries t_1 and t_2. We skip the details as this proposition does not seem too useful. In particular, due to the $O(k)$ term, it is not clear how to use this in the current framework unless $\alpha > k$.

6 Allocation Problem to k-server

Here we will show the following result.

THEOREM 6.1. *If there is an $(1+\varepsilon, \log(k/\varepsilon))$-competitive algorithm for the Allocation Problem for any $\varepsilon > 0$, then there is an $\tilde{O}(\log^2 D \log n \log^2 k)$-competitive algorithm for any metric space on n points with diameter D. Here $\tilde{O}$ notation ignores some $\log\log$ terms.*

We note that all the ideas we describe here are either implicit or explicit in the work of Coté et al. [10]. However, our presentation is somewhat different and we give the result here for completeness.

Given a metric G on n points, we first embed it into a probability distribution over α-HSTs, incurring a loss of factor $O(\alpha \log n)$. We will specify α later. Thus we focus on a particular HST T. Before we begin, we

give some notation. The root of T is denoted by r. For an arbitrary node j of T, let $T(j)$ denote the subtree rooted at j. Let $\Delta(j)$ (resp. $\delta(j)$) denote the length of the edge connecting j to its parent (resp. child). So, $\Delta(j) = \alpha\delta(j)$ unless j is a leaf or the root. Let $\rho = (\rho(1), \rho(2), \ldots)$ be the request sequence where $\rho(t)$ denotes the leaf requested at time t. For each node j, we will also have a vector $\kappa(j) = (\kappa(j,0), \kappa(j,1), \ldots)$, where $\kappa(j,t)$ denotes the number of servers available in $T(j)$ at time t. We allow $\kappa(j,t)$ to be fractional (this makes the presentation somewhat easier, it is straightforward to view this as probability distribution on integer vectors). Throughout, we use $k \cdot \vec{1}$ to indicate the constant vector with each entry k.

Consider a subtree $T(j)$, and suppose we are given a vector $\kappa(j)$. We define $\text{Optcost}(j, \kappa(j), t)$ to be the optimum cost of serving the request sequence $\rho \cap T(j)$ until time t subject to the constraint that $\kappa(j,t')$ servers are available at any time t'. As usual serving a request means that we should place at least one unit of server at the requested point, and the cost of the solution is simply the total movement cost of the servers. If the number of servers $\kappa(j,t)$ at time t changes by $\eta = \kappa(j,t) - \kappa(j,t-1)$, then we require η units of servers to enter (or leave as the case may be) at node j and travel to some leaves.

Remark: In the definition of Optcost, we do not pay the cost $\Delta(j)|\kappa(j,t) - \kappa(j,t-1)|$ due to servers entering or leaving $T(j)$. Strictly speaking, Optcost is well-defined only when the starting configuration of servers is specified. However, this is not important for our purposes, and we will always assume that number of servers initially is 0.

For a vector $\kappa(j)$, let us define $g(\kappa(j), t) = \sum_{t'=1}^{t} |\kappa(j,t') - \kappa(j,t'-1)|$. The fact below follows directly from the definition of Optcost.

LEMMA 6.1. *Let j be a node in T with children $j_1, \ldots, j_c$, then for any request sequence ρ.*

$$\begin{aligned}(6.15)\ &\text{Optcost}(j, \kappa(j), t) \\ &= \min_{\kappa(j_1),\ldots,\kappa(j_c):\sum_{i=1}^{c}\kappa(j_i)=\kappa} \left(\sum_{i=1}^{c} \text{Optcost}(j_i, \kappa(j_i), t) \right. \\ &+ \left. \delta(j) \sum_{i=1}^{c} g(\kappa(j_i), t) \right).\end{aligned}$$

Proof. The condition $\sum_{i=1}^{c} \kappa(j_i) = \kappa$ ensures consistency between the number of servers in $T(j)$ and its subtrees $T(j_i)$. $\text{Optcost}(j_i, \kappa(j_i), t)$ measures the movement cost within $T(j_i)$ and $g(\kappa(j_i), t)$ measures the cost between these subtrees. ■

We view the allocation problem as follows[4]: There is a star rooted at j with children $j_1, \ldots, j_c$. Let δ denote the root to leaf distance. At each time step, we are given $\kappa(t) \leq k$ that specifies the number of available servers at time t and cost vectors $h_t^i = (h_t^i(0), h_t^i(1), \ldots, h_t^i(k))$ for each child j_i. Upon receiving the vectors, the algorithm can change the number of serves on leaves from $k_1, \ldots, k_c$ to $k'_1, \ldots, k'_c$ incurring a movement cost of $\sum_{i=1}^{c} |k'_i - k_i|\delta$ and incurs a hit-cost of $\sum_{i=1}^{c} h_t^i(k'_i)$. We will allow k'_i to be fractional, in which case we interpret the cost $h_t^i(k'_i)$ by considering the piecewise linear extension h_t^i to $[0, k]$. We are now ready to describe the k-server algorithm.

The k-server Algorithm. Each non-leaf node j in T solves an allocation problem $A(j)$ defined on j and its children $j_1, \ldots, j_c$. The instance $A(j)$ is defined as follows:

Cost-Vectors: At time t, node j_i receives the cost vector $h_{j_i}^t$ where its m-th entry is given by

$$h_{j_i}^t(m) = \text{Optcost}(j_i, m \cdot \vec{1}, t) - \text{Optcost}(j_i, m \cdot \vec{1}, t-1).$$

That is, the incremental optimum cost of the m-server solution for $T(j_i)$ with request sequence $\rho \cap T_{j_i}$.

Available servers $\kappa(j,t)$: The number of available servers is determined recursively. The root r always has k servers, i.e. $\kappa(r) = k \cdot \vec{1}$. For every other node j, the vector $\kappa(j)$ is determined by the solution to the allocation problem running at the root of j.

Analysis. Before we prove theorem 6.1, we state the following key lemma of Coté et al. [10] . This lemma relates the optimum cost incurred in a subtree with vector κ to the cost vectors arriving at j in the allocation problem. Its proof follows by quasi-convexity of the work function. In their paper, Coté et al, prove this lemma for integer vectors κ, but their proof goes through verbatim for fractional values of $\kappa(t)$.

LEMMA 6.2. *For any vector $\kappa(j)$, and node j of an α-HST, the following relations hold.*

$$\begin{aligned}(6.16)\ &-2\frac{\alpha}{\alpha-1}\delta(j)g(\kappa(j),t) \\ &\leq \text{Optcost}(j, \kappa(j), t) - \sum_{t'=1}^{t} h_j^{t'}(\kappa(j,t')) \\ &< \frac{\alpha}{\alpha-1}\delta(j)g(\kappa(j),t).\end{aligned}$$

Remark: Note that $\sum_{t'=1}^{t} h_j^{t'}(\kappa(j,t'))$ is exactly the hit cost that would be incurred in the allocation problem

[4] This view of the allocation problem is equivalent to that stated in the introduction, up to perhaps a factor 2 difference in movement costs.

(running on the root of j) if j has $\kappa(j,t')$ servers at $t' = 1,\ldots,t$.

Proof. (Theorem 6.1). We prove the following claim by induction on the depth of the HST ℓ.

CLAIM 6.1. *Let $T(j)$ be an α-HST with root j and let its depth be ℓ. Suppose we are given a vector $\kappa(j)$. Consider the k-server algorithm on $T(j)$ obtained by executing the algorithm above. The value of the solution produced is no more than $\beta_\ell \cdot \text{Optcost}(j,\kappa(j),\infty)$, where β_ℓ satisfies the recurrence $\beta_0 = 1$ and $\beta_\ell = \gamma\beta_{\ell-1} + O(\log(k/\varepsilon))$ where*

$$\gamma = (1+\varepsilon)\left(1+\frac{3}{\alpha}\right) + O\left(\frac{\log(k/\varepsilon)}{\alpha}\right).$$

The claim is clearly true for $\beta = 0$ (i.e. a single point space). Suppose it is true for HSTs of depth $\ell-1$, and let $T(j)$ be some HST of depth ℓ. Given $\kappa(j)$, consider some optimum solution for $T(j)$ that achieves value $\text{Optcost}(j,\kappa(j),\infty)$, and let $\kappa^*(j_i)$ be optimum vectors for the children j_i of j corresponding to this solution. For notational convenience, let us denote $\kappa = \kappa(j)$, $\delta = \delta(j)$ and $\kappa_i^* = \kappa^*(j_i)$. We also denote the total cost $\text{Optcost}(j,\kappa(j),\infty)$ by $\text{Optcost}(j,\kappa(j))$, and for any node i define

$$\text{Hitcost}(i,\kappa(i)) = \sum_t h_i^t(\kappa(i,t)).$$

By (6.15) and (6.16), we have

$$\begin{aligned}\text{Optcost}(j,\kappa) &= \sum_i (\text{Optcost}(j_i,\kappa_i^*) + \delta g(\kappa_i^*)) \\ &\geq \sum_i \left(\text{Hitcost}(j_i,\kappa_i^*) + \delta(1-\frac{2}{\alpha-1})g(\kappa_i^*)\right)\end{aligned}$$

In particular, this implies that

$$\begin{aligned}(6.17)\qquad &\sum_i (\text{Hitcost}(j_i,\kappa_i^*) + \delta g(\kappa_i^*)) \\ &\leq \frac{\alpha-1}{\alpha-3}\text{Optcost}(j,\kappa) \\ &\leq \left(1+\frac{3}{\alpha}\right)\text{Optcost}(j,\kappa).\end{aligned}$$

Consider the allocation problem $A(j)$ running at j. It computes the vectors κ_i for the children j_i of j, and since this is a $(1+\varepsilon, O(\ln(k/\varepsilon)))$-competitive algorithm, and by 6.18, we get

$$\begin{aligned}&\sum_i \text{Hitcost}(j_i,\kappa_i) \\ &\leq (1+\varepsilon)\left(\sum_i \text{Hitcost}(j_i,\kappa_i^*) + \delta\sum_i g(\kappa_i^*)\right) \\ (6.18)\leq\ &(1+\varepsilon)\left(1+\frac{3}{\alpha}\right)\text{Optcost}(j,\kappa)\end{aligned}$$

and

$$\begin{aligned}&\delta(\sum_i g(\kappa_i)) \\ &\leq O(\ln(k/\varepsilon))\left(\sum_i \text{Hitcost}(j_i,\kappa_i^*) + \delta\sum_i g(\kappa_i^*)\right) \\ (6.19)\leq\ &O(\ln(k/\varepsilon))\text{Optcost}(j,\kappa)\end{aligned}$$

Now, by the design of the algorithm, it solves the allocation problem on children $T(j_1),\ldots,T(j_c)$ with vectors $\kappa_1,\ldots,\kappa_c$. By the induction hypothesis, the k-server cost for $T(j_1),\ldots,T(j_c)$ is at most $\beta_{\ell-1}(\sum_i \text{Optcost}(j_i,\kappa_i))$ and movement cost across among $j_1,\ldots,j_c$ is $\delta\sum_i g(\kappa_i)$. Thus, the k-server cost for the subtree $T(j)$ is

$$\begin{aligned}&\beta_{\ell-1}\left(\sum_i \text{Optcost}(j_i,\kappa_i)\right) + \delta\sum_i g(\kappa_i) \\ \leq\ &\beta_{\ell-1}\left(\sum_i \text{Hitcost}(j_i,\kappa_i) + \sum_i \frac{\delta}{\alpha-1}g(\kappa_i)\right) \\ +\ &\delta\sum_i g(\kappa_i) \\ \leq\ &\beta_{\ell-1}\left((1+\varepsilon)\left(1+\frac{3}{\alpha}\right) + O\left(\frac{\log(k/\varepsilon)}{\alpha}\right)\right)\text{Optcost}(j,\kappa) \\ +\ &O(\log(k/\varepsilon))\text{Optcost}(j,\kappa).\end{aligned}$$

The first inequality follows by (6.16) and the second one follows by (6.18) and (6.19). Let $\gamma = (1+\varepsilon)(1+\frac{3}{\alpha}) + O(\frac{\log(k/\varepsilon)}{\alpha})$. Thus, we get the recurrence $\beta_\ell \leq \gamma\beta_{\ell-1} + O(\log(k/\varepsilon))$. Since $\beta_0 = 1$, we obtain that

$$\beta_\ell = O(\log(k/\varepsilon))\left(\frac{\gamma^{l+1}-1}{\gamma-1}\right).$$

Setting $\varepsilon = 1/(4\ell)$ and $\alpha = O(\ell\log(4k\ell))$, we get that $\gamma \leq (1+\frac{1}{2\ell})$ and hence $\beta_\ell = O(\ell\log(k\ell))$. Thus we have an $O(\alpha\log n\ell\log(k\ell)) = O(\log n\ell^2\log(k\ell))$ competitive algorithm for general metrics. As $\ell = O(\log_\alpha D) = \tilde{O}(\log D)$ this implies the result. ∎

6.1 Rounding the fractional k-server solution.

Note that each node runs the online procedure above, and this produces a fractional k-server solution for the HST. We do the following simple simulation procedure to convert this into an integral solution.

Consider time t, we call a configuration valid, if for every node and time t there are either $\lfloor\kappa_v[t]\rfloor$ or $\lceil\kappa_v[t]\rceil$ servers in subtree rooted at v. This definition automatically ensures that there is at most 1 server at any leaf, and there are exactly k servers in the tree. Now when the fractional distribution changes, we apply the technique that we use for (uniform) paging starting from the root, i.e. if ε mass moves from p in the left

subtree to q in the right, we remove p arbitrarily from ε fractional of configurations containing it, and add q arbitrarily to some ε fraction of configurations that do not contain q. Now, in the left subtree some subtrees might have 1 fewer node than what is allowed, or one more node than what is allowed, but the number of such nodes with excess or deficit is the same, we recursively balance them. Similarly for the right subtree. As long as $\alpha < 2$, it can be shown that the cost of this simulation is $O(1)$.

References

[1] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Unfair metrical task systems on hsts and applications. In *Manuscript, submitted to SODA 2010*.

[2] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. A primal-dual randomized algorithm for weighted paging. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 507–517, 2007.

[3] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proceedings of the 29th Annual ACM Symposium on Theory of computing*, pages 711–719, 1997.

[4] Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *IEEE Symposium on Foundations of Computer Science*, pages 450–458, 1999.

[5] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

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

[7] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *13th Annual European Symposium on Algorithms*, 2005.

[8] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. 2006.

[9] M. Chrobak and L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[10] A. Coté, A. Meyerson, and L. Poplawski. Randomized k-server on hierarchical binary trees. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 227–234, 2008.

[11] B. Csaba and S. Lodha. A randomized on-line algorithm for the k-server problem on a line. *Random Structures and Algorithms*, 29(1):82–104, 2006.

[12] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.

[13] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991.

[14] Amos Fiat and Manor Mendel. Better algorithms for unfair metrical task systems and applications. *SIAM J. Comput.*, 32(6):1403–1422, 2003.

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

[16] M. Manasse, L.A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[17] Steven S. Seiden. A general decomposition theorem for the k-server problem. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 86–97, 2001.

[18] S. Shalev-Shwartz and Y. Singer. A primal-dual perspective of online learning algorithms. *Machine Learning Journal*, 69(2/3):115 – 142, 2007.

[19] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.