A Primal-Dual Randomized Algorithm for Weighted Paging*

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

April 2, 2012

Abstract

The study the weighted version of classic online paging problem where there is a weight (cost) for fetching each page into the cache. We design a randomized $O(\log k)$-competitive online algorithm for this problem, where k is the cache size. This is the first randomized $o(k)$-competitive algorithm and its competitive ratio matches the known lower bound for the problem, up to constant factors. More generally, we design an $O(\log(k/(k-h+1)))$-competitive online algorithm for the version of the problem where the online algorithm has cache size k and it is compared to an optimal offline solution with cache size $h \leq k$.

Our solution is based on a two-step approach. We first obtain an $O(\log k)$-competitive fractional algorithm based on an online primal-dual approach. Next, we obtain a randomized algorithm by rounding in an online manner the fractional solution to a probability distribution on the possible cache states. We also give an online primal-dual randomized $O(\log N)$-competitive algorithm for the Metrical Task System problem (MTS) on a weighted star metric on N leaves.

1 Introduction

Caching is one of the earliest and most effective techniques of accelerating the performance of computing systems. Thus, vast amounts of effort have been invested in the improvement and refinement of caching techniques and algorithms. We consider the most basic theoretical model for caching known as two-level caching problem. Here, we are given a collection of n pages and a fast memory (cache) which can hold up to k of these pages. At each time step one of the pages is requested. If the requested page is already in the cache then this page is served instantaneously and no penalty is incurred, otherwise there is cache miss and the algorithm must bring the currently requested page into the cache, possibly evicting some other page(s). In the most basic version of the problem, referred to as paging or caching, all the pages are identical and the goal is to minimize the total number of cache misses.

The paging problem is most interesting in the online setting, where the requests arrive over time and the algorithm must make its current decision of which page to evict based on the input seen thus far, and without any knowledge of future requests. Indeed, paging is one of the most

*A preliminary version appeared in the Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (2007), pp. 507-517.

†Eindhoven University of Technology, The Netherlands. E-mail: *n.bansal@tue.nl*

‡Computer Science Department, Open University of Israel. Supported by ISF grant 954/11 and BSF grant 2010426. E-mail: *niv.buchbinder@gmail.com*

§Computer Science Department, Technion, Haifa 32000, Israel. Supported by ISF grant 954/11 and BSF grant 2010426. E-mail: *naor@cs.technion.ac.il*

extensively studied problems in online computation, and was already considered in the seminal paper of Sleator and Tarjan [27] on competitive analysis. In competitive analysis, one compared the worst case performance of an online algorithm to the optimum offline solution in hindsight. Formally, given a minimization problem P and an instance I of P, let $\mathrm{OPT}(I)$ denote the cost of the optimum offline solution for instance I. Then, we say that an online algorithm A for P has a competitive ratio of c, if for every instance I of P, it holds that

$$A(I) \leq c \cdot \mathrm{OPT}(I) + d,$$

where d is a universal constant independent of the instance I. Sometimes, the competitive ratio c may depend on parameters such as the length of the input sequence (though this will not be the case for the problems we consider in this paper). An algorithm with a competitive ratio of c is also referred to as c-competitive. If the algorithm A is randomized, then A is said to be c-competitive if for every instance I, it holds that

$$\mathbb{E}[A(I)] \leq c \cdot \mathrm{OPT}(I) + d.$$

Note that the expectation is only over the random choices of the online algorithm A, and we still consider the worst case over all possible instances I.

In this paper we consider the weighted version of the paging problem, often referred to as weighted paging or weighted caching in the literature. Here there is a collection of n pages, and a cache of size k, i.e. it can hold any subset of up to k pages at any time. Each page p has an associated weight $w(p)$ that indicates the cost of fetching this page into the cache. The weights model situations where some pages are more expensive to fetch than others because they may be on far away servers, or slower disks, and so on. Given a request sequence of pages, the goal is to minimize the total fetching cost incurred by the algorithm.

1.1 Previous Work

Paging and weighted paging: The unweighted paging problem is very well understood. In their seminal paper on competitive analysis, Sleator and Tarjan [27] showed that any deterministic algorithm is at least k-competitive, and also showed that the LRU (Least Recently Used) algorithm is exactly k-competitive. They also considered the more general (h, k)-paging problem where the online algorithm with cache size k is compared to an optimal offline solution with cache size h where $h \leq k$. They showed that any deterministic algorithm is at least $k/(k-h+1)$-competitive, and that LRU is exactly $k/(k-h+1)$-competitive. When randomization is allowed, Fiat et al. [19] designed the first randomized algorithm, the *marking algorithm*, which is $2H_k$-competitive against an oblivious adversary, where H_k is the k-th Harmonic number. They also showed that any randomized algorithm is at least H_k-competitive. Subsequently, McGeoch and Sleator [26] gave a matching H_k-competitive algorithm, and Achlioptas, Chrobak and Noga [1] gave another H_k-competitive algorithm that is easier to state and analyze. For (h, k)-paging, Young [28] gave a $2\ln(k/(k-h+1))$-competitive algorithm (omitting lower order terms from the competitive ratio) and showed that any algorithm is at least $\ln(k/(k-h+1))$-competitive. There has been extensive work on paging along several other directions, and we refer the reader to the excellent text by Borodin and El-Yaniv [9] for further details.

For the weighted paging problem, a k-competitive deterministic algorithm was obtained by Chrobak et al. [15]. This ratio exactly matches the known lower bound for the unweighted paging. Subsequently, Young [29] gave a best possible $k/(k-h+1)$-competitive deterministic algorithm for the more general (h, k)-weighted paging problem. Despite substantial interest [29, 9, 25, 3], no randomized algorithms with competitive ratio $o(k)$ were known for weighted

paging, except in some special cases. Irani [22] gave an $O(\log k)$-competitive algorithm for the two-weight case, i.e., when each page weight is either 1 or some fixed $M > 1$. In another direction, Blum, Furst and Tomkins [8] gave an $O(\log^2 k)$-competitive algorithm for the case of $n = k + 1$ pages. Later, Fiat and Mendel [20] gave an improved $O(\log k)$ competitive algorithm for the case of $n = k + c$ pages, where c is a constant. In particular, no $o(k)$-competitive algorithm was known (prior to our work) for the case of three distinct weights and large n. Another generalization of weighted paging is multi-size setting when pages have arbitrary sizes in addition to arbitrary weights. Recently, building up on the techniques in this paper, [5] give an $O(\log^2 k)$ for this problem. This improves and extends the previous results of Irani [23].

Connection to the k-server problem: Paging can also be viewed as a special case of the much more general and challenging k-server problem. In this problem, there are k servers located on some points in an n-point metric space. At each time step a request is placed at one of the points and the algorithm must move one of the servers to this point to serve the request. The goal is to minimize the overall distance traveled by the servers. The unweighted paging problem is precisely the k-server problem on a uniform metric space. The weighted paging problem is identical (up to an additive constant) to the k-server problem on a weighted star metric space, where the leaves correspond to pages, and the distance of leaf a from the center is $w(a)/2$, where $w(\cdot)$ denotes the page weights.

The k-server problem has a fascinating history (see for example [9]) and we only mention the main results here. It is known that any deterministic algorithm must be at least k-competitive on any metric space with more than k points. Fiat, Rabani and Ravid [21] gave the first algorithm for which the competitive ratio only depends on k. Their algorithm was $O((k!)^3)$-competitive. A breakthrough was achieved by Koutsoupias and Papadimitriou [24] who gave an almost tight $2k-1$ competitive algorithm. This is still the best known bound for general metric spaces, both in the deterministic and randomized case (if no dependence on n is allowed). The tight guarantee of k is also known for many special cases. In particular, Chrobak and Larmore [16] gave a k-competitive algorithm for trees. Recently, [4] gave an $\tilde{O}(\log^3 n \log^2 k)$-competitive[1] randomized algorithm for the k-server problem on a general metric space, building up on the ideas in this paper and in [18].

Metrical task systems: A closely related problem is the metrical task systems problem (MTS). In the MTS problem we are given a finite metric space $\mathcal{M} = (V, d)$, where $|V| = N$. We view the points of $\mathcal{M}$ as states in which the algorithm may be situated in. The distance between the points of the metric measures the cost of transition between the possible states. We use the k-server terminology and say that a server that serves the requests is moving between the states. Each task (request) r in a metrical task system is associated with a vector $(r(1), r(2), \ldots, r(N))$, where $r(i)$ denotes the cost of serving r in state $i \in V$. In order to serve request r in state i the server has to be in state i. Upon arrival of a new request, the state of the system can first be changed to a new state (paying the transition cost), and only then the request is served (paying for serving the request in the new state). The objective is to minimize the total cost which is the sum of the transition costs and the service costs.

The MTS model was formulated by Borodin, Linial and Saks [10] who gave tight upper and lower bound of $2N - 1$ for any deterministic online algorithm for the problem. They also designed a $2H_N$-competitive randomized algorithm for the uniform metric, and showed a lower bound of H_N for this metric. For the MTS problem on a weighted star, Blum et

[1] The $\tilde{O}$ notation hides some $\log \log$ factors.

al. [8] gave a randomized $O(\log^2 N)$-competitive algorithm. Fiat and Mendel [20] improved this bound to $O(\log N)$. This algorithm is based on the fact that a weighted star metric can be approximated well by a hierarchically well-separated tree (HST) with several "nice" properties. For general metrics Bartal et al. [7] designed a randomized $O(\log^5 N)$-competitive algorithm which is based on an algorithm for HST-s. Fiat and Mendel [20] improved this bound and designed an $O(\log^2 N \log\log N)$-competitive algorithm for general metrics.

1.2 Results and Techniques

Our main result is the following.

Theorem 1.1 (**Weighted Paging**). *There is an $O(\log k)$-competitive randomized algorithm for the online weighted paging problem. More generally, there is an $O(\log(k/(k-h+1)))$-competitive algorithm with respect to an optimal solution that has cache size $h \leq k$.*

A novel aspect of our algorithm is that unlike previous randomized algorithms for paging, it does not rely on "phases" to lower bound the cost of the optimum algorithm. Our algorithm is based on a two-step approach. First, we obtain an $O(\log k)$-competitive *fractional* algorithm, where it is allowed to maintain fractions of pages in the cache, as long as the sum of the page fractions does not exceed the cache size, k. The cost of fetching an ϵ fraction of a page is then ϵ times the weight associated with the page. This algorithm is designed by formulating the problem using a *covering* linear program with box constraints[2] and then applying the primal-dual framework of [11] developed for online packing and covering problems. However, directly applying the primal-dual framework of [11] only gives a (fractional) $O(\log n)$-competitive algorithm, and we need some additional ideas to obtain the improved $O(\log k)$-competitive algorithm.

Second, we transform this fractional algorithm to a randomized algorithm, by mapping the probablity distribution on pages given by the fractional solution to a probability distribution on integral cache states. We show that such a transformation is possible and can be maintained in an online manner, while incurring only an additional constant multiplicative factor loss. These two steps together yield the desired randomized $O(\log k)$-competitive algorithm.

We remark that the linear programming formulation we use was previously used by Bar-Noy et al. [6] in the context of designing approximation algorithms for a more general variant of caching, where the pages have both arbitrary weights and sizes, and also the amount of available cache might vary with time. Cohen and Kaplan [17] also used this formulation to give an alternative proof that Young's dual-greedy algorithm [29] is $k/(k-h+1)$-competitive (in fact, [29] used an equivalent linear program in the sense that it defines the same polytope even though this formulation is not a covering-packing linear program).

Our second result is the following.

Theorem 1.2 (**Metrical Task Systems**). *There is an $O(\log N)$-competitive randomized algorithm for the online Metrical Task Systems (MTS) problem on a metric defined by a weighted star on N leaves.*

This algorithm is designed along the same lines, but requires an additional initial idea. First, we define a new MTS model and show that for a weighted star metric, its cost is within a constant factor of that on the original MTS model. Then we formulate the new MTS model as a covering linear program, generate an online fractional solution to it using the online primal dual

[2] Box constraints are upper bounds on the values of the variables in the linear program.

method, and then show how to round it with no additional cost. Although we do not improve the best known algorithm for this problem (the result of [20] is tight up to $O(1)$ factors), our algorithm does not involve the use of HST-s and seems more natural. The algorithm can be viewed as an extension of the original $O(\log N)$-competitive algorithm for a uniform metric [10] to a weighted star metric.

The online primal-dual method: The primal-dual method is one of the fundamental design methodologies in the areas of approximation algorithms and combinatorial optimization. Recently, Buchbinder and Naor [11] have further extended the primal-dual method and have shown its wide applicability to the design and analysis of online algorithms. In further work, [12, 14] have shown more applications of the primal-dual method to online algorithms such as ski-rental, ad-auctions, routing and load balancing, network optimization problems, and more. See [12] for a survey. We use the primal-dual method here for both making online decisions as well as for performing the competitive analysis. We note that the connection between competitive analysis and linear programming duality was made explicit for the first time by Young [29] in his work on weighted paging, where an optimal deterministic k-competitive algorithm is obtained via a primal-dual approach.

2 Preliminaries

Let us recall the notation before we begin. There is a cache of size k and there are $n > k$ different pages denoted $1, \ldots, n$, and each page p has a fetching cost of w_p. For each time $t = 1, \ldots, T$, where T is the length of the input sequence, let p_t denote the page requested at t. If p_t is not in the cache, the algorithm must fetch it to the cache, possibly evicting some other page to make room for it and incurring a cost of w_{p_t}. Note that evicting a page does not incur any cost.

Cost Accounting. A simple observation is that instead of charging for fetching a page, we can charge for evicting it from the cache. Clearly, over the entire time horizon of the request sequence, the number of times any page is fetched can differ from the number of times it is evicted by at most 1. Thus, the overall cost for fetching versus evicting can differ by at most $k \cdot w_{\max}$, which is an additive constant independent of the length of the request sequence.

2.1 LP formulation for the weighted paging problem.

While it is perhaps more natural to view caching as a packing problem (as the pages must be packed into a cache of size k), it can also be viewed as a covering problem in the following way. For each page p, and integers $j = 1, \ldots$, let $t(p, j)$ denote the time of the jth request for p. Consider any two consecutive requests for page p at times $t(p, j)$ and $t(p, j + 1)$. Since page p must be present in the cache at both of these times, the algorithm pays for fetching p at time $t(p, j + 1)$ if and only if p was evicted during the time interval $[t(p, j) + 1, t(p, j + 1) - 1]$. Now, in the offline optimal solution, if page p is evicted in this interval, then we can assume without loss of generality that it is evicted at time $t(p, j) + 1$ (clearly, this assumption is not necessarily true in the online case).

The above motivates the following covering integer program. Let $x(p, j)$ be a $0 - 1$ variable, with value 1 indicating that page p is evicted from the cache during $[t(p, j) + 1, t(p, j + 1) - 1]$ (and hence at $t(p, j) + 1$). For each page p, let $r(p, t)$ denote the number of requests for page p

until time t (including t). For any time t, let $B(t) = \{p \mid r(p,t) \geq 1\}$ denote the set of pages that were ever requested until time t (including t).

We need to satisfy that at any time t, the page p_t must be present in the cache, and that the total space used by pages in the cache is at most k. As page p_t must be in the cache, this implies that at most $k-1$ space can be used by pages in $B(t) \setminus \{p_t\}$ at time t, i.e. $\sum_{p \in B(t) \setminus \{p_t\}} (1 - x(p, r(p,t))) \leq k-1$, which upon rearrangement, gives us that

$$\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \geq |B(t)| - k. \tag{1}$$

This gives the following exact covering formulation of the weighted paging problem (where we charge for evicting a page instead of fetching).

$$\begin{aligned} \min \quad & \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} w_p \cdot x(p,j) & & \\ & \sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \quad \geq \quad |B(t)| - k & \qquad & \forall t \in \{1, \ldots, T\} \\ & x(p,j) \quad \in \quad \{0,1\} & \qquad & \forall \ p, j \end{aligned}$$

The natural LP relaxation is obtained by relaxing the variables $x(p,j)$ in the above IP to take values between 0 and 1, yielding the following linear programming formulation:

$$\begin{aligned} \min \quad & \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} w_p \cdot x(p,j) & & \text{(LP-Paging)} \\ & \sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \quad \geq \quad |B(t)| - k & \qquad & \forall t \in \{1, \ldots, T\} \\ & 0 \leq x(p,j) \quad \leq \quad 1 & \qquad & \forall \ p, j \end{aligned} \tag{2}$$

In the dual program, there is a variable $y(t)$ for each time t, and a variable $z(p,j)$ for each page p and the jth time it is requested. The dual program is the following:

$$\max \quad \sum_{t=1}^{T} (|B(t)| - k)\, y(t) \ - \ \sum_{p=1}^{n} \sum_{j=1}^{r(p,T)} z(p,j)$$

For each page p and the jth time it is requested:

$$\begin{aligned} \left(\sum_{t=t(p,j+1)+1}^{t(p,j+1)-1} y(t) \right) - z(p,j) \leq w_p & & & \\ \text{for any } p, j: \qquad z(p,j) \geq 0 & & & \\ \text{for all } t: \qquad y(t) \geq 0 & & & \end{aligned} \tag{3}$$

2.2 Randomized Algorithms

Any randomized algorithm can be viewed as a probability distribution over deterministic algorithms. This is also the view that we will adopt in designing our randomized algorithms.

In particular, at each time step, we will explicitly specify the probability distribution on the deterministic states that the algorithm maintains. To do this, for each time step t and each real number $\eta \in [0,1)$, we specify a set $S(\eta,t)$ of pages such that: (i) the total number of pages in $S(\eta,t)$ is at most k and (ii) the currently requested page p_t is present in $S(\eta,t)$ for all $\eta \in [0,1)$. Upon arrival of a new request at time $t+1$, the sets $S(\eta,t+1)$ can possibly change and the cost incurred by the algorithm at this time step is the expected cost

$$\mathbb{E}_\eta\Big[C(S(\eta,t+1) \setminus S(\eta,t))\Big], \tag{4}$$

where $C(X)$, for a subset of pages X, denotes the total fetching cost of all pages in X.

Now, given this explicit description, consider the randomized algorithm that picks a random number $\eta \in [0,1)$, once and for all, and sets its cache state at each time t to $S(\eta,t)$. Clearly, the expected cost (by averaging over η) of this randomized algorithm is exactly equal to the cost given by (4). We remark that our algorithm does not necessarily run in polynomial time (this is not an issue for online algorithms) and we shall not concern ourselves with this aspect here.

Instead of working directly with probability distributions over deterministic states, it is much simpler to break this down into two steps:

- *Maintaining a fractional solution:* Instead of specifying an explicit probability distribution on cache states, we work with a probability distribution on pages in the cache at each time step, i.e. the probability of page p being present in the cache at time t, denoted by $\Pr(p,t)$. We call this the *fractional* view, and first design a *fractional* algorithm in this setting. Here, the fractional algorithm may keep fractions of pages as long as the total (fractional) number of pages present in the cache does not exceed the cache size k. The cost of fetching an ε fraction of a page p is then defined to be εw_p. We note that any feasible solution to the linear program (LP-Paging), gives a fractional algorithm as the variables indicate what fraction of a page is missing from the cache. More formally, at time t, $\Pr(p_t,t) = 1$, and for each $p \neq p_t$, $\Pr(p,t) = 1 - x(p,r(p,t))$.

- *Online rounding of the fractional solution:* In the second step, we map the above fractional algorithm into an algorithm that works with a probability distribution over cache states. To this end, we maintain online a probability distribution over cache states such that (i) it is consistent with the fractional solution in hand, and (ii) show how to map the changes in the fractional view over pages (at every time step t) to changes in the probability distribution over the cache states, so that the expected cost incurred is not much higher than the fractional cost. Note that this mapping is maintained in an online manner.

Clearly, a distribution on pages can be mapped consistently to several distributions over cache states. For example, suppose $k = 2$ and the fractional solution is $(1/2, 1/2, 1/2, 1/2)$ on four pages A, B, C, D. This solution is consistent with the distribution D_1 which assigns probability $1/2$ each to the two cache states (A,B) and (C,D), and it is also consistent with the distribution D_2 that assigns a probability of $1/6$ to each of the six cache states (A,B), $(A,C), \ldots, (C,D)$. It turns out that one has to choose the distribution over cache states carefully (among the possible consistent ones), in order to be able to perform the mapping online. The following example is instructive.

Suppose the fractional solution is $(1/2, \ldots, 1/2)$ on four pages A, B, C and D. The pages have weights $w_A = w_B = 1$ and $w_C = w_D = M \gg 1$. Suppose we map this fractional solution to the distribution D_1 (defined above). Now, suppose that at the next time step, the fractional solution changes to $(0, 1, 1/2, 1/2)$, i.e. the algorithm moves $1/2$ unit of probability mass from page A to B, incurring a fractional cost of $1/2$. We claim that it is impossible to modify the

distribution D_1 to be consistent with the new fractional solution without incurring $\Theta(M)$ cost. To see this, observe that the only distribution on cache states consistent with $(0, 1, 1/2, 1/2)$ is to assign a probability of $1/2$ each to the cache states (B, C) and (B, D)[3]. Thus, from the cache state (C, D) (in the distribution D_1), either C or D must be moved to make room for B, which incurs cost $\Theta(M)$.

To get around this problem, we restrict our mapping to distributions in a certain subclass. For example, in the scenario described above, we disallow the distribution D_1 for the fractional solution $(1/2, 1/2, 1/2, 1/2)$. In particular, in Section 5 we show how to maintain an online mapping from the fractional solution to probability distributions on cache states, such that any fractional change incurring cost c is mapped to a change on distributions on cache states with cost at most $10c$.

2.3 The Metrical Task System Problem

We consider the metrical task system (MTS) problem on a metric $\mathcal{M}$ defined by a weighted star. The leaves of the star are denoted by $\{1, 2, \ldots, N\}$, and the distance from the center of the star to leaf i is $d(i)$. There is a single server and the leaf in which the server is located defines the *state* of the system. The cost of moving the server from state i to state j is $d(i) + d(j)$. We can assume that the server is initially in state 1. Each task (request) r in a metrical task system is associated with a vector $(r(1), r(2), \ldots, r(N))$, where $r(i)$ denotes the cost of serving r in state (leaf) i. In order to serve request r in state i the server has to be in leaf i. Upon arrival of a new request, the state of the system can first be changed to a new state (paying the transition cost), and only then the request is served (paying for serving the request in the new state).

3 A Fractional Primal-Dual Algorithm for Weighted Paging

Our online paging algorithm produces fractional primal and dual solutions to LP-Paging. In the online case, the constraints of LP-Paging (corresponding to the requests to pages) are revealed one-by-one. Upon arrival of a constraint, the algorithm finds a feasible assignment to the (primal) variables that satisfies the constraint. Consider variable $x(p, j)$. In the offline case, we can assume without loss of generality that the value of $x(p, j)$ is determined at time $t(p, j) + 1$. However, this is not necessarily true in the online case; thus, we require that the values assigned to $x(p, j)$ by the online algorithm in the time interval $[t(p, j) + 1, t(p, j + 1) - 1]$ form a monotonically non-decreasing sequence.

We start with a high level description of the algorithm. Upon arrival of a new constraint at time t, if it is already satisfied, then the algorithm does nothing. Otherwise, the algorithm needs to satisfy the current constraint by increasing some of the primal variables in the constraint. Satisfying the constraint guarantees that there is enough space in the cache to fetch the new page. To this end, the algorithm starts increasing (continuously) the new dual variable $y(t)$. As a result, the variables $x(p, j)$ increase as an exponential function of $y(t)$, which means that pages are gradually evicted. When variable $x(p, j)$ reaches 1 (i.e. page p is completely evicted), it remains so from this time onwards. To satisfy the dual constraint corresponding to $x(p, j)$, the algorithm starts increasing the dual variable $z(p, j)$ at the same rate as $y(t)$. The algorithm is presented in a continuous fashion, but it can easily be implemented in a discrete fashion.

[3]This is because B must lie in every state, and since $k = 2$, this fixes everything else.

Fractional Paging Algorithm: At time t, when page p_t is requested:

- Set the new variable: $x(p_t, r(p_t, t)) \leftarrow 0$ (It can only be increased at times $t' > t$).
- If the primal constraint corresponding to time t is satisfied (i.e., $\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \geq |B(t)| - k$), then do nothing.
- Otherwise: increase primal and dual variables, until the primal constraint corresponding to time t is satisfied, as follows:

  a. Increase variable $y(t)$ continuously.

  b. For each variable $x(p, r(p,t)) < 1$ that appears in the (yet unsatisfied) primal constraint (2) corresponding to time t, increase $x(p, r(p,t))$ at rate:

$$\frac{dx(p, r(p,t))}{dy(t)} = \frac{\ln(1+k)}{w_p} \left(x(p, r(p,t)) + \frac{1}{k} \right) \tag{5}$$

  For each variable $x(p, r(p,t)) = 1$ that appears in the (yet unsatisfied) primal constraint (2) corresponding to time t, increase $z(p, r(p,t))$ at the same rate as $y(t)$.

Observation 3.1. *For every p and j, the value of $x(p,j)$ at the end of the algorithm (i.e. at time T) is given by*

$$x(p,j) = \frac{1}{k} \cdot \left(\exp\left(\frac{\ln(1+k)}{w_p} \left(\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j) \right) \right) - 1 \right). \tag{6}$$

Proof. Consider first $x(p,j)$ that is strictly less than 1 at time T. We note that $x(p,j)$ is only updated at times $t(p,j)+1, \ldots, t(p,j+1)-1$, and it increases according to (5) whenever some dual variable $y(t)$ such that $t \in [t(p,j)+1, \ldots, t(p,j+1)-1]$ is increased.

Now, consider the differential equation $dx/dy = c(x + 1/k)$. If y increases from a to b, we have that

$$\ln(x(b) + 1/k) - \ln(x(a) + 1/k) = c(b-a).$$

Variable $x(p,j)$ starts at 0 at time $t(p,j)+1$, increases according to (5), and any $y(t)$ is only increased at time t starting with value 0, implying that at the end of time $t(p,j+1)-1$:

$$\ln(x(p,j) + 1/k) - \ln(1/k) = \sum_{t'=t(p,j)+1}^{t(p,j+1)-1} \frac{1}{w_p} \cdot \ln(1+k) y(t),$$

which upon taking exponents of both sides and rearranging gives the claimed result. Assume next that the variable $x(p,j) = 1$. This means that it was raised in rate determined by the differential equation until it became 1. From this point on its derivative by the algorithm is zero. On the other hand, $z(p,j)$ is increased (starting from zero) by the same rate as $y(t)$ making the derivative of (6) zero as well. □

We next show:

Theorem 3.2. *The fractional paging algorithm is $2\ln(1+k)$-competitive.*

Remark 3.3. *It is possible to improve the competitive ratio of the algorithm from* $2\ln(1+k)$ *to approximately* $(1+o(1))\ln(1+k)$*, by redefining Equation (5) to be:*

$$\frac{dx(p,r(p,t))}{dy(t)} = \frac{\ln(1+k\ln k)}{w_p}\left(x(p,r(p,t)) + \frac{1}{k\ln k}\right)$$

It is not hard to verify in the proof of Theorem 3.2 that this change results in an algorithm with competitive ratio $\ln(1+k)$ *plus lower order terms (e.g.* $\ln\ln k$*). In the rounding phase, however, we anyway lose several constant factors, so we will not focus on optimizing the constants in Theorem 3.2.*

Proof of Theorem 3.2. It suffices to show the following. First, both primal and dual solutions are feasible, and second, the primal cost is at most $2\ln(1+k)$ times the dual cost. As the value of any feasible dual solution lower bounds the optimum LP value, the result is implied.

Feasibility: The primal solution generated by the algorithm is feasible by design since in each iteration, the variables $x(p,j)$ are increased until the primal constraint of time t is satisfied. Also, each variable $x(p,j)$ is never increased beyond 1, since whenever $x(p,j)$ reaches 1, the algorithm stops increasing it.

To see that the dual is not violated, note that as $x(p,j) \leq 1$, observation 3.1 implies that

$$x(p,j) = \frac{1}{k}\cdot\left(\exp\left(\frac{\ln(1+k)}{w_p}\left(\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j)\right)\right) - 1\right) \leq 1$$

which upon rearranging and taking logarithms equivalently implies that

$$\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j) \leq w_p,$$

and hence that the dual constraint corresponding to the variable $x(p,j)$ is never violated.

Bounding Primal and Dual Costs: We now bound the primal cost by relating it to the dual profit. We view the change in the primal and dual variables as a continuous process. At any time, let S denote the set of pages $p \in B(t)$ for which $x(p,r(p,t)) < 1$. Let S^c denote the remaining pages $B(t) \setminus S$. Let P be the primal cost, and let D be the dual profit. With this notation, the rate of dual and primal increase can be expressed as follows:

$$\begin{aligned}
\frac{dP}{dy(t)} &= \sum_{p\in B(t)\setminus\{p_t\}} w_p \cdot \frac{dx(p,r(p,t))}{dy(t)} \\
&= \ln(1+k)\cdot \sum_{p\in S\setminus\{p_t\}} \left(x(p,r(p,t)) + \frac{1}{k}\right) && (7)\\
&\leq \ln(1+k)\cdot\left((|B(t)| - k - |S^c|) + \frac{|S|-1}{k}\right) && (8)\\
&\leq 2\ln(1+k)\cdot(|S|-k) = 2\ln(1+k)\frac{dD}{dy(t)} && (9)
\end{aligned}$$

Equality (7) follows directly from Equation (5). Inequality (8) holds since the new primal constraint is unsatisfied yet and hence

$$\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) < |B(t)| - k.$$

As $x(p, r(p,t)) = 1$ for $p \in S^c$, this implies

$$\sum_{p \in S \setminus \{p_t\}} x(p, r(p,t)) < |B(t)| - k - |S^c| = |S| - k.$$

Inequality (9) follows by observing that $(|S| - 1)/k \leq |S| - k$. This holds by the simple fact that $(x-1)/k \leq x-k$ for any real number $x \geq k+1$, and observing that $|S| \geq k+1$. The latter holds as the primal constraint is not yet feasible, and hence $0 \leq \sum_{p \in S \setminus \{p_t\}} x(p, r(p,t)) < |S| - k$. Finally, the equality in (9) follows as $y(t)$ increases at rate 1, and each $z(p, r(p,t))$ also increases at rate 1 for each $p \in S^c$. Thus, the dual profit increases at rate $(|B(t)| - k) - |S^c| = |S| - k$.

Thus, the primal cost derivative is always at most $2\ln(1+k)$ times the derivative of the dual profit. Since our primal and dual solutions are feasible and the profit of any dual feasible solution is always a lower bound on the optimal solution, we conclude by weak duality that the algorithm is $2\ln(1+k)$-competitive. □

4 A Fractional Primal-Dual Algorithm for Bicriteria Weighted Paging

In this section, we give a fractional online algorithm with improved guarantees for the (h, k)-weighted paging problem. Here the online algorithm has a cache of size k, but we compare its performance to the offline optimum solution that has a cache of size h (where $h \leq k$). Again, the algorithm will be based on the primal-dual approach and we will perform a primal-dual analysis. However, there is one main difference from the previous section. We write a primal formulation, as previously, corresponding to a cache of size h, and then consider its dual which will act as a lower bound for the optimum solution. However, the online algorithm needs to be designed to work with a cache of size k (and not h like the primal or the dual). The algorithm is described below.

Fractional Paging Algorithm: At time t, when page p_t is requested:

- Set the new variable: $x(p_t, r(p_t, t)) \leftarrow 0$ (It can only be increased at times $t' > t$).
- If the primal constraint corresponding to time t is satisfied (i.e., $\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \geq |B(t)| - k$), then do nothing.
- Otherwise: increase primal and dual variables, until the primal constraint corresponding to time t is satisfied, as follows:
  a. Increase variable $y(t)$ continuously.
  b. Let $\eta = k/(k-h+1)$. For each variable $x(p, r(p,t)) < 1$ that appears in the (yet unsatisfied) primal constraint (2) corresponding to time t, increase $x(p, r(p,t))$ at rate:
  $$\frac{dx(p, r(p,t))}{dy(t)} = \frac{\ln(1+\eta)}{w_p} \left(x(p, r(p,t)) + \frac{1}{\eta} \right) \tag{10}$$
  For each variable $x(p, r(p,t)) = 1$ that appears in the (yet unsatisfied) primal constraint (2) corresponding to time t, increase $z(p, r(p,t))$ at the same rate as $y(t)$.

The algorithm is very similar to the previous one and there are only two differences. First, η replaces the value k in the previous algorithm. Second, even though the constraint in the primal LP (corresponding to the offline optimum) for time t is $\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \geq (|B(t)| - h)$, we only raise $y(t)$ if $\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) \geq (|B(t)| - k)$. As in Observation 3.1 it is easy to verify that,

Observation 4.1. *For every p and j, the value of $x(p,j)$ at the end of the algorithm (i.e. at time T) is given by*

$$x(p,j) = \frac{1}{\eta} \cdot \left(\exp\left(\frac{\ln(1+\eta)}{w_p} \left(\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j) \right) \right) - 1 \right). \tag{11}$$

Theorem 4.2. *The algorithm is $2\ln(1 + k/(k-h+1))$-competitive with respect to an optimal solution that has cache size h.*

Proof. The proof is the same as the proof of Theorem 3.2. We simply modify it to work with respect to the modified dual (that bounds any solution that has a cache of size h).

Feasibility: The primal solution generated by the algorithm is feasible by design since in each iteration, the variables $x(p,j)$ are increased until the primal constraint of time t is satisfied. Also, each variable $x(p,j)$ is never increased beyond 1, since whenever $x(p,j)$ reaches 1, the algorithm stops increasing it.

To see that the dual is not violated, note that as $x(p,j) \leq 1$, observation 4.1 implies that

$$\frac{1}{\eta} \cdot \left(\exp\left(\frac{\ln(1+\eta)}{w_p} \left(\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j) \right) \right) - 1 \right) \leq 1$$

which upon rearranging and taking logarithms equivalently implies that

$$\sum_{t'=t(p,j)+1}^{t(p,j+1)-1} y(t') - z(p,j) \leq w_p,$$

and hence that the dual constraint corresponding to the variable $x(p,j)$ is never violated.

Bounding Primal and Dual Costs: We now bound the primal cost by relating it to the dual profit. We view the change in the primal and dual variables as a continuous process. At any time, let S denote the set of pages $p \in B(t)$ for which $x(p, r(p,t)) < 1$. Let S^c denote the remaining pages $B(t) \setminus S$. Let P be the primal cost, and let D be the dual profit. With this notation, the rate of dual and primal increase can be expressed as follows:

$$\begin{aligned}
\frac{dP}{dy(t)} &= \sum_{p \in B(t) \setminus \{p_t\}} w_p \cdot \frac{dx(p, r(p,t))}{dy(t)} \\
&= \ln(1+\eta) \cdot \sum_{p \in S \setminus \{p_t\}} \left(x(p, r(p,t)) + \frac{1}{\eta} \right) && (12) \\
&\leq \ln(1+\eta) \cdot \left((|B(t)| - k - |S^c|) + \frac{|S|-1}{\eta} \right) && (13) \\
&\leq 2\ln(1+\eta) \cdot (|S| - h) = 2\ln(1+\eta) \frac{dD}{dy(t)} && (14)
\end{aligned}$$

Equality (12) follows directly from Equation (10). Inequality (13) holds since the new primal constraint is unsatisfied yet and hence $\sum_{p \in B(t) \setminus \{p_t\}} x(p, r(p,t)) < |B(t)| - k$. Moreover, as $x(p, r(p,t)) = 1$ for $p \in S^c$, this implies that

$$\sum_{p \in S \setminus \{p_t\}} x(p, r(p,t)) < |B(t)| - k - |S^c| = |S| - k \leq |S| - h$$

Inequality (14) follows by observing that $(|S|-1)/\eta = (|S|-1)\frac{k-h+1}{k} \leq |S| - h$. This holds by the simple fact that $(x-1) \cdot \frac{k-h+1}{k} \leq x - h$ for any real number $x \geq k+1$, and observing that $|S| \geq k+1$. The latter holds as the primal constraint is not yet feasible, and hence $0 \leq \sum_{p \in S \setminus \{p_t\}} x(p, r(p,t)) < |S| - k$. Finally, the equality in (14) follows as $y(t)$ increases at rate 1, and each $z(p, r(p,t))$ also increases at rate 1 for each $p \in S^c$. Thus, the dual profit increases at rate $(|B(t)| - h) - |S^c| = |S| - h$.

Thus, the primal cost derivative is always at most $2\ln(1+\eta)$ times the derivative of the dual profit. Since our primal and dual solutions are feasible and the profit of any dual feasible solution is always a lower bound on the optimal solution with cache size h, we conclude by weak duality that the algorithm is $2\ln(1 + \frac{k}{k-h+1})$-competitive. □

5 Rounding the Fractional Solution Online

In this section we show how to obtain a randomized online algorithm for the weighted paging problem from the fractional algorithm described in Section 3.

High level idea and basic definitions. As discussed previously, in order to obtain an actual randomized algorithm we need to specify a probability distribution, μ, over the various possible cache states and specify how this probability distribution evolves over time. More precisely, at

each time step t and for each real number $\eta \in [0,1)$, the probability distribution μ associates a subset $S(\eta,t)$ of pages that lies in the cache. When a new request arrives at time $t+1$, the sets $S(\eta,t+1)$ change and the cost incurred by the online algorithm is the cost of the fetched pages, averaged over η, i.e.,

$$\mathbb{E}_\eta\Big[C(S(\eta,t+1)\setminus S(\eta,t))\Big] \tag{15}$$

where $C(X)=\sum_{p\in X} w_p$. Of course, the probability distribution μ and its evolution over time will be closely related to the evolution of the fractional caching solution x.

Relating μ and fractional caching solution. Instead of specifying a probability distribution over cache states, it turns out to be more convenient to work with a probability distribution over sets of pages missing from the cache. This is because the fractional caching solution specifies a variable $x(p)$ for each page p denoting the probability that it is missing from the cache. So at time t, we maintain for each $D \subseteq B(t)$ the probability $\mu(D)$ that exactly the pages in D are missing from the cache.

At any time step t, let $x_p := x(p, r(p,t))$ denote the fraction of page p that is absent from the cache as specified by the fractional solution. The probability distribution μ, that we maintain later will be *consistent* with the fractional solution in the sense that the marginal probability that a page p is not in the cache will be exactly x_p. Formally,

Definition 5.1 (Consistent Distribution)**.** *A probability distribution μ on subsets $D \subseteq B(t)$ is consistent with respect to the fractional solution x on pages, if*

$$\forall p: \qquad \sum_{D\subseteq B(t)} \chi_D(p)\cdot\mu(D) = x_p, \tag{16}$$

where χ_D denotes the characteristic vector of the subset D, i.e., for each page p, $\chi_D(p)=1$ iff $p\in D$.

The support of the probability distribution μ that we maintain should only consist of *valid* complement of cache states. As the total size of pages in a cache can be at most k, this implies the following natural definition,

Definition 5.2 (Valid Distribution)**.** *A probability distribution μ is* valid*, if for any set $D \subseteq B(t)$ with $\mu(D)>0$, it holds that $\sum_{p\in B(t)}(1-\chi_D(p)) \le k$, or alternatively,*

$$\sum_{p\in B(t)} \chi_D(p) \ge |B(t)| - k.$$

Online Maintenance. We now discuss the issue of maintaining the probability distribution μ over time as the fractional solution changes. First, throughout this section we consider the (equivalent) cost version in which the fractional solution pays w_p for both fetching and evicting a page p. As the total amount of evictions and fetches can differ by at most 1 for any page, over the entire time horizon, this increases the cost of the fractional solution by a factor of two. So, in this accounting, the fractional cost of changing the fractional value x to x' is $\sum_p w_p|x'_p - x_p|$.

As we would like to ensure that the cost of modifying μ from time t to $t+1$ is not much more than the fractional cost incurred during that step, it motivates the following definition.

Definition 5.3 (β-simulation)**.** *Let t be some time during the execution, and suppose the fractional solution changes from x to x' while incurring a fractional cost of d. Let μ be a probability*

distribution that is consistent with respect to x and valid. A β-simulation *is a procedure of updating the probability distribution μ to a probability distribution μ' that is consistent with respect to x' and valid, while incurring a cost of at most βd, where $\beta > 0$. In particular, for each $\eta \in [0, 1)$, the simulation procedure specifies the way the set $S(\eta)$ changes.*

The existence of a β-simulation procedure thus implies that

$$\sum_t \mathbb{E}_\eta[C(S_{\eta,t} \oplus S_{\eta,t+1})] \leq \beta \cdot \sum_t \sum_p |x_p^t - x_p^{t+1}|, \tag{17}$$

where $A \oplus B$ denotes the symmetric difference of sets A and B.

The following observation is immediate.

Observation 5.4. *If there exists a β-simulation procedure with respect to x, and if there exists a c-competitive fractional algorithm, then there exists a randomized algorithm that is βc-competitive.*

Proof. By equation (17) the cost of the randomized algorithm can be bounded as

$$\sum_t \mathbb{E}_\eta[C(S_{\eta,t} \oplus S_{\eta,t+1})] \leq \beta \cdot \sum_t \sum_p |x_p^t - x_p^{t+1}| = \beta \cdot C_f \leq \beta c \cdot C^*,$$

where C_f and C^* denote the costs of the fractional solution and the offline optimal solution, respectively. □

By the above observation, our goal is now is to design a β-simulation for the weighted paging problem with β as small as possible. However, designing such a procedure is rather subtle. Indeed, we already described in Section 2.2 an example in which the rounding can get "stuck" if μ is not chosen carefully. In particular, even if μ is consistent with x at time t, if x changes to x' with fractional cost d, it could be the case that moving to any probability distribution μ' from μ, such that μ' is consistent with x', incurs a cost much higher than d. Thus, our approach is to guarantee that the probability distribution μ is not only valid and consistent with x, but rather satisfies an additional stronger property.

We then construct a β-simulation procedure with the following properties: (i) at each time t, the probability distribution μ is consistent, valid, and satisfies the stronger property with respect to x; (ii) when x changes to x', incurring a fractional cost d, the probability distribution μ can be modified to μ' which is consistent with x', valid, satisfies the stronger property, and also incurs a cost of at most βd, where $\beta > 0$. Designing such a simulation guarantees that the stronger property can always be maintained, which gives us the desired randomized algorithm by observation 5.4.

5.1 Obtaining randomized algorithm for weighted paging

For $i = 0, 1, 2, \ldots$, we define the cost class $S(i)$ to be the set of pages p of cost at least 2^i and less than cost 2^{i+1}. Formally, $S(i) = \{p \mid 2^i \leq w_p < 2^{i+1}\}$. Let $x_1, \ldots, x_n$ be the LP solution (produced by the fractional algorithm) at the current time step. We first define the stronger property on the probability distribution μ that we are going to maintain.

Definition 5.5 (Balanced Subsets)**.** *A probability distribution μ has* the balanced subsets *property with respect to x, if for any set $D \subseteq B(t)$ with $\mu(D) > 0$, the following holds for all $j \geq 0$:*

$$\left\lfloor \sum_{i \geq j} \sum_{p \in S(i)} x_p \right\rfloor \leq \sum_{i \geq j} \sum_{p \in S(i)} \chi_D(p) \leq \left\lceil \sum_{i \geq j} \sum_{p \in S(i)} x_p \right\rceil. \tag{18}$$

Note that the middle sum is simply the number of pages in D with cost at least 2^j.

Lemma 5.6. *Let μ be any probability distribution that is consistent with x and satisfies the balanced subset property, then any set D such that $\mu(D) > 0$ is a valid complement of a cache .*

Proof. By plugging $j = 0$ in the balanced subset property we get that at any time t,

$$\sum_{p \in B(t)} \chi_D(p) \;=\; \sum_{i \geq 0} \sum_{p \in S(i)} \chi_D(p) \geq \left\lfloor \sum_{i \geq 0} \sum_{p \in S(i)} x_p \right\rfloor = \left\lfloor \sum_{p \in B(t)} x_p \right\rfloor \geq |B(t)| - k$$

The first inequality follows by the balanced subset property (18). The second inequality follows as the fractional solution is feasible and hence $\sum_p x_p \geq |B(t) - k|$, moreover as $|B(t)| - k$ is an integer, the inequality in maintained upon taking floors. □

The β-simulation: We now describe a β-simulation procedure with $\beta = 10$ for the the problem. This amounts to showing the following:

Lemma 5.7. *Let x be any fractional caching solution, and let μ be an arbitrary probability distribution on cache states which is both consistent and has the balanced subsets property with respect to x. Then, given any other fractional caching solution x', there is a procedure to transform μ into a probability distribution μ' which is both consistent and has the balanced subsets property with respect to x', such that the cost of transforming μ to μ' is at most β times the fractional cost of transforming x to x', for $\beta = 10$.*

Before we prove the above lemma, we note that it readily implies our main result for weighted paging. In particular, as the LP solution gives a fractional $O(\log k)$ competitive algorithm, and as $\beta = O(1)$ in Lemma 5.7, together with Observation 5.4, we get that:

Theorem 5.8. *There is a randomized $O(\log k)$-competitive algorithm for the weighted paging problem.*

We now prove Lemma 5.7.

Proof. (Lemma 5.7) Let us decompose the change from x to x' into a sequence of steps where the marginal x_p of exactly one page p changes from x_p to x'_p. As the fractional cost of changing x to x' is equal to the sum of the fractional costs incurred at each step, it suffices to describe and prove the lemma for a single step.

So, let us fix some page p and assume that $x'_p = x_p + \varepsilon$ for some $\varepsilon > 0$ (the case when $x'_p = x_p - \varepsilon$ is analogous). Let i be the class of p, i.e. $w_p \in [2^i, 2^{i+1})$. Thus, the fractional cost of the change from x to x' is at least $\varepsilon 2^i$.

Construction of μ': We construct μ' as follows. Recall that μ is consistent with the fractional solution x, and hence for page p, $\sum_{D \subseteq B(t)} \chi_D(p) \cdot \mu(D) = x_p$. To be consistent with x', we need to transform μ to μ' such that $\sum_{D \subseteq B(t)} \chi_D(p) \cdot \mu'(D) = x'_p = x_p + \varepsilon$. To this end, choose an arbitrary collection[4] of sets $D \subseteq B(t)$ with total measure ε under μ such that $p \notin D$, and add the page p to them (i.e, $D \leftarrow D \cup \{p\}$). The resulting probability distribution μ' is consistent with x'. The cost incurred is at most $2^{i+1} \cdot \varepsilon$, as we evict p from an ε fraction of caches.

[4]As μ associates a set D with each real number $\eta \in [0, 1)$, we must specify this choice explicitly. To do this, we can pick the least indices η with total measure ε corresponding to sets that do not contain p.

Fixing the Balanced Set Property: However, the above probability distribution μ' may violate the balanced set property. We now fix this. Let us inspect the balance condition (18) more closely. First observe that it may only be violated for classes $\tilde{i}$ such that $\tilde{i} \leq i$. Moreover, this violation can occur for two reasons. First, the subsets D that changed to $D \cup \{p\}$ may cause the middle term of (18) to increase for some classes $\tilde{i}$ and the left and right terms remain unchanged. Second, the increase of x_p to $x_p + \varepsilon$ may cause the right and left terms to increase for some classes $\tilde{i}$, but the middle term remains unchanged for some sets D. We consider each of these two cases separately.

First Case: Let us consider the (harder) first case. We will fix the unbalanced property one class at a time starting from class i and going down to class 0. Let us consider class i. Let $s = \lceil \sum_{j \geq i} \sum_{p \in S(j)} x_p \rceil$ denote the upper bound on class $\geq i$ pages. Note that (in this first case) we are assuming that s does not increase when we replace x_p by $x'_p = x_p + \varepsilon$. As μ satisfied the balanced set property with respect to x, each D with $\mu(D) > 0$ contained exactly s or $s-1$ pages of classes $\geq i$. So, in the probability distribution μ', each subset D can contain exactly $s-1, s$ or $s+1$ pages belonging to classes $\geq i$. Let ε' be the measure of sets D in μ' that have exactly $s+1$ pages in classes $\geq i$. Note that $\varepsilon' \leq \varepsilon$ since we only added page p to ε measure of the probability distribution μ. As μ' is consistent by our assumption above,

$$\sum_{D \subseteq B(t)} \mu'(D) \sum_{k \geq i} \sum_{p \in S(k)} \chi_D(p) = \sum_{k \geq i} \sum_{p \in S(k)} x'_p \leq s.$$

As ε' measure of the sets in μ' have $s+1$ pages of classes $\geq i$, this implies that at least ε' measure of sets in μ' that contain $s-1$ pages of classes $\geq i$.

We arbitrarily[5] choose ε' measure of these sets and match them with the sets containing $s+1$ class $\geq i$ pages (again for explicitness we can choose the smallest lexicographic matching). Consider any pair of sets (D, D') that are matched. Since μ' satisfies condition (18) for class $i+1$, the number of pages in D and D' that lie in classes $i+1$ or higher differs by at most 1 (while the number of pages in class i or higher differs by 2). Hence, $D \setminus D'$ contains at least one page, p' of class i (if there are several such p', we can choose the page with the least index). We move this page from D to D' (i.e., $D' \leftarrow D' \cup \{p'\}$, $D \leftarrow D \setminus \{p'\}$). Note that both D and D' satisfy the balanced sets property (18) of class i after this procedure (they both have exactly s pages of class i or higher). As this procedure transfers one page of class i from ε' measure of the subsets the total cost incurred here is at most $(2\varepsilon') \cdot 2^{i+1} \leq 2^{i+2} \cdot \varepsilon$.

Second Case: Let us now consider the case when $\lceil \sum_{j \geq i} \sum_{p \in S(j)} x'_p \rceil$ increases to $s+1$. This is relatively easy. First, we split ε as $\varepsilon' + \varepsilon''$, where ε'' is such that $\sum_{j \geq i} \sum_{p \in S(j)} x_p + \varepsilon' = s$ (i.e. the fractional number of pages in classes $\geq i$ just reaches s). We apply the transformation of the previous case with $x'_p = x_p + \varepsilon'$, which by the balanced set condition and consistency ensures that every cache state with $\mu(D) > 0$ has exactly s class $\geq i$ pages. Now, we add p to ε'' measure of the sets D in μ, guaranteeing that cache states contains either s or $s+1$ of class $\geq i$ pages and guarantees that the probability distribution is consistent with x'.

Handling Smaller Classes: We now show that applying the above steps to class i results in an analogous scenario for classes $i-1$ and lower. In the above procedure for class i, page p is added to at most ε measure of sets D, and some page p' of class i is removed from a subset of the above sets and transferred to sets containing $s-1$ pages from classes $\geq i$. As the resulting

[5]To make this choice explicit, we can choose the relevant sets with the smallest indices η

probability distribution satisfies the balanced property for classes $\geq i$, and exactly ε measure of sets D gain a class i page, the resulting μ' may violate condition (18) with respect to class $i-1$ for at most ε measure of subsets D. This results in the same situation with respect to class $i-1$ that we previously considered for class i. More specifically, μ' satisfies the balanced subset property for classes $\geq i$, and at most ε measure of D in μ' violate (by at most 1) the balanced property for class $i-1$. So, we apply the above procedures sequentially to fix classes $i-1, i-2, \ldots, 0$ resulting in an additional cost of $\sum_{j=0}^{i-1} 2\varepsilon \cdot 2^{j+1} < 4\varepsilon 2^i$.

Thus, the total cost incurred by all the steps (i.e. adding page p to ε fraction of states and fixing the balanced property for classes $i, \ldots, 0$) is at most $(2+4+4)\varepsilon 2^i = 10\varepsilon 2^i$. □

6 The Metrical Task System Problem on a Weighted Star Metric

We consider in this section the metrical task system (MTS) problem on a metric $\mathcal{M}$ defined by a weighted star. The leaves of the star are denoted by $\{1, 2, \ldots, N\}$. The distance from node i to node j is $d(i,j) = d(i) + d(j)$. We present an $O(\log N)$-competitive online algorithm for the problem.

Let $\bar{\sigma} = r_1, r_2, \ldots, r_T$ denote the request sequence, where each r_t is a vector that defines the service cost in each of the possible states at time t. It is convenient henceforth to work with an equivalent continuous time MTS model [9, Sec. 9.1.1]. Here, the cost vector r_t is incurred uniformly during the time interval $[t, t+1)$ and the algorithm is allowed to change states at any time $t \in [1, n+1]$ (instead of just at discrete times). Formally, an algorithm in this model specifies the state of the algorithm Alg$[t]$ at each $t \in [1, n+1]$, by describing a sequence of states $s_0, \ldots, s_k$ and a sequence of transition times $t_1 < \ldots < t_k$. The algorithm is in state s_i during the time interval $[t_i, t_{i+1}]$.[6]

The service cost incurred by Alg for processing the ith request is:

$$\int_{t'=t}^{t+1} r_t(\text{Alg}[t'])dt' \tag{19}$$

It is known [9, Sec. 9.1.1] that any algorithm in the continuous model can be transformed to a discrete time algorithm without increasing the total cost. On the other hand, since the continuous time model is a relaxation of the discrete model, the optimal cost cannot increase. We use this model for our analysis.

Next, we change the way we charge the algorithm for transitions (movement) cost. Instead of charging the algorithm $d(i)+d(j)$ for moving from state i to state j, we charge the algorithm $2d(i)$ whenever the server moves out of state i to any other state. This changes the total movement cost of any solution (over the entire input sequence) by an additive term that is at most $\max_i\{d_i\}$, as this incurs an additional cost of at most $\max_i\{d(i)\}$ for moving out of the state initially at $t=1$. For notational convenience we scale down the movement cost by a factor of 2 (we also scale down the service cost by a factor of 2, so the instance remains identical up to scaling). So from now on we let $d(i)$ be the cost of moving from state i to any other state. We call the model defined so far the *original model* and summarize its properties:

- The model is continuous and thus states can be changed at non-integral times.

[6] Formally, the algorithm may change its state infinitely many times. However, such an algorithm is not of interest since its movement cost is infinite. Thus, we restrict our attention to the set of algorithms that change their state finitely many times.

- The service cost is charged by integrating in accordance with Equation (19).
- The movement cost from state i (to any other state) is $d(i)$.

Finally, before stating the algorithm, we define yet another model that we call the *new model*. We will show that obtaining a c-competitive for the new model results in a $4c$-competitive algorithm for the original model. The new model is also continuous, but as we explain later, it allows state transitions to occur only at certain specific times. The high level idea of the new model is to do away with the movement cost incurred due to state transitions and pay only for serving requests. To balance the costs, we restrict the algorithm and allow it to change its state only if certain conditions are fulfilled. For each state i we partition the time interval into phases. We permit the solutions to leave state i only at the end of a phase (of state i). The first phase of each state starts at time $t = 1$. Phase p of state i starts at time $\tau_{p-1}(i)$ and ends at the earliest time $\tau_p(i)$ for which the accumulated service cost at state i in the interval $[\tau_{p-1}(i), \tau_p(i)]$ is exactly $d(i)$[7]. Formally, $\tau_p(i)$ is the earliest time for which:

$$\int_{t=\tau_{p-1}(i)}^{\tau_p(i)} r_{\lfloor t \rfloor}(i)dt = d(i). \tag{20}$$

We are now ready to describe the new MTS model in its full generality. An algorithm in the new model is allowed to leave a state i only at the end of a phase of state i. The algorithm does not pay any transition cost when moving from one state to another. If the algorithm is ever in state i during a phase p, then it pays a cost $d(i)$. Note that the algorithm pays the full cost of the phase even if it was in state i only during part of the phase p. This can happen if the algorithm moves to state i from i' in the middle of the pth phase of i (and at the end of a phase of state i'). That is, let $x(i,p)$ be a 0/1 variable indicating that there exists some time $t \in [\tau_{p-1}(i), \tau_p(i)]$ for which $\text{Alg}[t] = i$. Let n_i be the number of phases of state i. Then, the total service cost of the algorithm is:

$$\sum_{i=1}^{N} \sum_{p=1}^{n_i} d(i)x(i,p).$$

As in the original model, an algorithm in the new model can be described as a sequence of states $s_0, \ldots, s_k$ and a sequence of transition times $t_1 < \ldots < t_k$ such that the algorithm is in state s_i during the time interval $[t_i, t_{i+1}]$. To summarize, the new model has the following properties:

- The model is continuous, but an algorithm is allowed to move out of state i only at the end of the current phase for state i.
- The algorithm incurs a service cost $d(i)$ for each state i and phase p, if there exists some $t \in [\tau_{p-1}(i), \tau_p(i)]$ for which $\text{Alg}[t] = i$.
- The algorithm does not pay any transition cost.

Given a sequence of requests $\bar{\sigma}$, let $\text{OPT}_n(\bar{\sigma})$ be the minimum offline cost of serving the sequence of requests in the new MTS model. Let $\text{OPT}_o(\bar{\sigma})$ be the minimum offline cost of serving the sequence of requests in the original MTS model. We prove the following two lemmas:

[7]The end of the time horizon is also defined as an end of a phase.

Lemma 6.1. *Let $\bar{\sigma}$ be a sequence of requests. Any solution S to $\bar{\sigma}$ in the original MTS model with cost C can be transformed into a solution S' in the new MTS model with cost at most $2C$. In particular,* $\mathrm{OPT}_n(\bar{\sigma}) \leq 2\mathrm{OPT}_o(\bar{\sigma})$.

Proof. Let S be a solution to the MTS problem in the original model. Let $s_0, \ldots, s_k$ be sequence of states of the solution and let $t_1 < \ldots < t_k$ be its sequence of transition times. The main issue is that this solution might not be valid in the new model since transition times are not necessarily at the end of phases. We therefore construct a new solution S' that is valid in the new model and prove that its cost in the new model is at most twice the cost of S in the original model.

Initially, S' starts from the same state s_0 as solution S. We build the transition sequence inductively until the end of the time horizon T. Suppose the current state of S' is s, and S' transits to s at time $t \in [\tau_{p-1}(s), \tau_p(s)]$ for some phase p of state s. S' transits at time $\tau_p(s)$ to the state s' such that $S[\tau_p(s)] = s'$ (i.e, the solution S is in s' at time $\tau_p(s)$). Note that if the solution S is at state s at time $\tau_p(s)$ then S' simply stays in state s (at least until time $\tau_{p+1}(s)$). Clearly, S' is a valid solution in the new MTS model by design as it changes its state only at the end of a phase. Also, it is easily seen that if solution S' is in state s at some time during $[\tau_{p-1}(s), \tau_p(s)]$, then it stays in state s at least until time $\tau_p(s)$.

Next, we prove that the cost of solution S' in the new model is at most twice the cost of S in the original model. Let $s'_0, s'_1, s'_2, \ldots, s'_m$ be the sequence of states of solution S' and let $t'_1 < t'_2 \ldots, < t'_m$ be the transition times of solution S'. Consider the cost of solution S' in the interval $[t'_i, t'_{i+1}]$. Let $p, p+1, p+\ell-1$ be the phases such that $\tau_p(s'_i) \in [t'_i, t'_{i+1})$. Note that we count only phases whose end is in the middle of the time interval, and not at time t'_{i+1} (even though t'_{i+1} is an end of phase). Therefore, ℓ might be zero. This means that the service cost of S' in the interval $[t'_i, t'_{i+1}]$ (hence the total cost of S') is $(\ell+1) \cdot d(s'_i)$.

Let us now lower bound the cost of solution S in the same time interval. We claim that S pays a total cost of at least $\max\{1, \ell\} d(s'_i)$ in the interval $[t'_i, t'_{i+1}]$ which will imply the result. Suppose $\ell = 0$, then it means that solution S is in state s'_i at time t'_i and is not at s'_i at time t'_{i+1}. Thus, in this interval it has to transit from state s'_i and pay at least $d(s'_i)$. Next, if $\ell \geq 1$, we claim that solution S pays at least $d(s'_i)$ in each of the intervals $[t'_p(s'_i), \tau_{p+1}(s'_i)], [\tau_{p+1}(s'_i), \tau_{p+2}(s'_i)], \ldots, [\tau_{p+\ell-1}(s'_i), t'_{i+1}(s'_i)]$. Consider first the last time intervals. We know that solution S is in state s'_i at time $\tau_{p+\ell-1}(s'_i)$ and is in state $s'_{i+1} \neq s'_i$ at time t'_{i+1}. Thus, in this interval it has to transit from state s'_i and pay at least $d(s'_i)$. Consider now other intervals $[\tau_{p+j}(s'_i), \tau_{p+j+1}(s'_i)]$. We know that solution S is in state s'_i at time $\tau_{p+j}(s'_i)$ and also at time $\tau_{p+j+1}(s'_i)$. If the solution S does not leave state s'_i during that time interval then by the definition of phases it pays exactly $d(s'_i)$ service cost. Otherwise, it transits during this time interval from state s'_i and pays $d(s'_i)$. Either way it pays at least $d(s'_i)$, which concludes our proof. □

Lemma 6.2. *Let $\bar{\sigma}$ be a sequence of requests. Any solution S to $\bar{\sigma}$ in the new MTS model with cost C is a feasible solution S' in the original MTS model with cost at most $2C$.*

Proof. Let $s_0, \ldots, s_k$ be sequence of states of solution S in the new model, and let $t_1 < \ldots < t_k$ be its sequence of transition times. We analyze the cost of this sequence in the original model. First, the service cost of the sequence in the original MTS model is never more than the service cost of the sequence in the new model. This follows by the following simple observation. Consider the cost of solution S in the interval $[t_i, t_{i+1}]$. Let $p, p+1, p+\ell-1$ be the phases such that $\tau_p(s_i) \in [t_i, t_{i+1})$. Then the service cost in the new model during this interval is $(\ell+1)d(s_i)$.

However, by the definition of phases the cost during interval $[t_i, t_{i+1}]$ in the original model is at most $(\ell+1)d(s_i)$ (otherwise there were more phases inside this interval).

Next, we claim that the transition cost of solution S in the original model is no more than the service cost of S in the new model. This follows since for every i, the service cost of S in the new model during time interval $[t_i, t_{i+1}]$ is at least $d(s_i)$ which is the transition cost (out of s_i) in the standard model at the end of the interval. Thus, the cost of the solution S in the original MTS model is at most twice its cost in the new MTS model. □

By Lemmas 6.1 and 6.2 a c-competitive algorithm in the new MTS model implies a $4c$-competitive algorithm in the original MTS model, and hence it suffices to consider the new MTS model.

6.1 The Algorithm

In this section we design a simple linear programming formulation for the problem in the new MTS model. Similarly to weighted paging the online algorithm generates a fractional solution to this linear program. We later transform this fractional solution to a randomized integral solution. Let $x(i,p)$ be a 0-1 variable indicating the event that the algorithm is in state i during the pth phase of state i. Let n_i be the number of phases of state i. An integer program for the problem is then the following:

$$(I) \qquad \min \quad \sum_{i=1}^{N} \sum_{p=1}^{n_i} d(i)x(i,p)$$

$$\sum_{i=1}^{N} \sum_{p \mid t \in [\tau_{p-1}(i), \tau_p(i)]} x(i,p) \geq 1 \quad \forall t \tag{21}$$

$$x(i,p) \in \{0,1\} \quad \forall i,p \tag{22}$$

While it may seem that this integer program contains an unbounded number of constraints (21), it is easy to see that we only need to consider (21) at times t that are the end times of a phase of some state. It can also be easily verified that any valid solution in the new MTS model defines a feasible solution to (I) with the same cost.

We obtain an LP relaxation to the problem by relaxing the variables $x(i,p)$ in the integer formulation and allow them to take values in $[0,1]$. Since the RHS in each of the constraints is 1, an optimal solution will never give the variables value above 1. Thus, we may remove the upper bound on the variables obtaining the following linear programming relaxation.

$$(P) \qquad \min \quad \sum_{i=1}^{N} \sum_{p=1}^{n_i} d(i)x(i,p)$$

$$\sum_{i=1}^{N} \sum_{p \mid t \in [\tau_{p-1}(i), \tau_p(i)]} x(i,p) \geq 1 \quad \forall t \tag{23}$$

$$x(i,p) \geq 0 \quad \forall i,p \tag{24}$$

A fractional solution to (P) defines for any time t and state i the marginal probability in which the algorithm state is i. The dual program has a variable $y(t)$ for each time t, and the

constraint requires that the total sum of the variables $y(t)$, taken over a phase p of state i, is at most $d(i)$. The dual program is the following.

$$(D) \qquad \max \quad \sum_{t=1}^{T} y(t)$$

$$\sum_{t \in [\tau_{p-1}(i), \tau_p(i)]} y(t) \leq d(i) \quad \forall i, p \tag{25}$$

$$y(t) \geq 0 \quad \forall t \tag{26}$$

Algorithm: We now describe an online primal-dual algorithm for the MTS problem. This is a special case of the online covering-packing algorithm described in [11], and the analysis of [11] already implies an $O(\log N)$-competitive factor for it. However, as the algorithm described in [11] is more general (and complicated), we provide here for completeness the algorithm and a proof of the competitive factor.

At each time period t (when the phase $p-1$ of some state j ends),

- Set $y(t) \leftarrow 0$ and $x(j,p) \leftarrow 0$,
- While the primal constraint of time t is not satisfied (i.e., $\sum_{i=1}^{N} \sum_{p \mid t \in [\tau_{p-1}(i), \tau_p(i)]} x(i,p) < 1$):
  1. Increase variable $y(t)$ continuously.
  2. For each i, p such that $t \in [\tau_{p-1}(i), \tau_p(i)]$ increase the variable $x(i,p)$ at rate

$$\frac{dx(i,p)}{dy(t)} = \frac{\ln(1+N)}{d(i)} \left(x(p,i) + \frac{1}{N} \right)$$

By reasoning as in Observation 3.1, it is easy to verify that

Observation 6.3. *For every i and p, the value of $x(i,p)$ at the end of the time horizon (i.e. at time T) is given by*

$$x(i,p) = \frac{1}{N} \left[\exp\left(\frac{\ln(1+N)}{d(i)} \sum_{t \in [\tau_{p-1}(i), \tau_p(i)]} y(t) \right) - 1 \right]. \tag{27}$$

Theorem 6.4. *The algorithm is $2\ln(1+N)$-competitive.*

Proof. It suffices to show the following. First, both primal and dual solutions are feasible, and second, the primal cost is at most $2\ln(1+N)$ times the dual cost. As the value of any feasible dual solution lower bounds the optimum LP value, the result is implied.

Feasibility: The primal solution generated by the algorithm is feasible by design since in each iteration, the variables $x(i,p)$ are increased until the primal constraint of time t is satisfied. Also, we never increase variable $x(i,p)$ to be more than 1, since whenever $x(i,p)$ reaches 1, the current constraint is already feasible.

To see that the dual is not violated, note that as $x(i,p) \leq 1$, observation 6.3 implies that

$$x(i,p) = \frac{1}{N}\left[\exp\left(\frac{\ln(1+N)}{d(i)}\sum_{t\in[\tau_{p-1}(i),\tau_p(i)]} y(t)\right) - 1\right] \leq 1$$

which upon rearranging and taking logarithms is equivalent to $\sum_{t\in[\tau_{p-1}(i),\tau_p(i)]} y(t) \leq d(i)$, and hence that the dual constraint (25) is never violated.

Bounding Primal and Dual Costs: We now bound the primal cost by relating it to the dual profit. We view the change in the primal and dual variables as a continuous process. At any time t, let P be the primal cost, and let D be the dual profit. With this notation, the rate of dual and primal increase can be expressed as follows:

$$\begin{aligned}
\frac{dP}{dy(t)} &= \sum_{i=1}^{N} \sum_{p \mid t\in[\tau_{p-1}(i),\tau_p(i)]} d(i)\cdot\frac{dx(i,p)}{dy(t)} \\
&= \ln(1+N)\sum_{i=1}^{N} \sum_{p \mid t\in[\tau_{p-1}(i),\tau_p(i)]} \left(x(i,p)+\frac{1}{N}\right) \\
&\leq 2\ln(1+N) = 2\ln(1+N)\frac{dD}{dy(t)}. \qquad (28)
\end{aligned}$$

Here (28) follows as $\sum_{i=1}^{N}\sum_{p \mid t\in[\tau_{p-1}(i),\tau_p(i)]} x(i,p) < 1$ and as the number of variables in any primal constraint is N.

Thus, the rate of change of the primal cost is always at most $2\ln(1+N)$ times the rate of change of the dual profit. Since our primal and dual solutions are feasible and the profit of any dual feasible solution is always a lower bound on the optimal solution, we conclude by weak duality that the algorithm is $2\ln(1+N)$-competitive. □

Rounding the fractional solution: This is a based on a standard approach. The algorithm maintains the invariant that it is in state i at time t (in phase p) with probability equal to $x(i,p)$. Suppose the phase p for state i ends and we have $x(i,p) = a$ at this time. When the new phase $p+1$ for state $i+1$ begins the algorithms starts with $x(i,p+1) = 0$ and increases the value of the variables $x(j,p_j)$ (including $j = i$), where p_j denote the current phase of j. Let a_j denote the amount of increase in $x(j,p_j)$ for state j. As $a = \sum_j a_j$, we can view this process as redistributing a among the other states (where state j gets a_j). If our randomized algorithm is in state i at this time, then the algorithm moves to state j with probability a_j/a, (note that $\sum_j a_j = a$), otherwise if it was in a different state $i' \neq i$, then it remains in the state i'. This maintains the invariant that the algorithm is in state j with probability $x(j,p_j)$. Also the probability that the algorithm ever enters a state j during the phase p_j of j is exactly $x(j,p_j)$, and hence the expected cost of the algorithm is exactly equal to the cost of the fractional solution.

7 Concluding Remarks

We give a randomized $O(\log k)$-competitive algorithm for the weighted paging problem based on the online primal-dual method. Subsequently, these ideas were extended to obtain poly-logarithmic competitive algorithms for the more general case where pages have both arbitrary

weights and sizes [5]. Recently, [2] refined this approach further to obtain the optimum $O(\log n)$ guarantee for the problem. Building up on these ideas in yet another direction, [4] gave an $\tilde{O}(\log^3 n \log^2 k)$ competitive algorithm for the k-server problem on a general metric space on n points. Some additional applications of the online primal-dual method are surveyed in [13].

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theory Computer Science*, 234(1-2):203–218, 2000.

[2] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An o(log k)-competitive algorithm for generalized caching. In *Symposium on Discrete Algorithms, SODA, to appear*, 2012.

[3] S. Albers. Online algorithms: A survey. *Mathematical Programming*, 97:3–26, 2003.

[4] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the Foundations of Computer Science*, pages 267–276, 2011.

[5] Nikhil Bansal, Niv Buchbinder, and Joseph (Seffi) Naor. Randomized competitive algorithms for generalized caching. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 235–244, 2008.

[6] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[7] 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.

[8] A. Blum, M. Furst, and A. Tomkins. What to do with your free time: algorithms for infrequent requests and randomized weighted caching. Manuscript 1996.

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

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

[11] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. In *Proceedings of the 13th Annual European Symposium on Algorithms*, pages 689–701, 2005.

[12] N. Buchbinder and J. Naor. Fair online load balancing. In *Proceedings of the 18th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 291–298, 2006.

[13] N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.

[14] Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proceedings of the 15th Annual European Symposium*, pages 253–264, 2007.

[15] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Math*, 4(2):172–181, 1991.

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

[17] Edith Cohen and Haim Kaplan. Lp-based analysis of greedy-dual-size. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 879–880, 1999.

[18] 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.

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

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

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

[22] S. Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002.

[23] Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 701–710, 1997.

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

[25] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[26] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

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

[28] Neal Young. On-line caching as cache size varies. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 241–250, 1991.

[29] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.