Draft

An $O(\log k)$-competitive algorithm for Generalized Caching*

Anna Adamaszek† Artur Czumaj† Matthias Englert†
Harald Räcke‡

In the generalized caching problem, we have a set of pages and a cache of size k. Each page p has a size $w_p \geq 1$ and fetching cost c_p for loading the page into the cache. At any point in time, the sum of the sizes of the pages stored in the cache cannot exceed k. The input consists of a sequence of page requests. If a page is not present in the cache at the time it is requested, it has to be loaded into the cache incurring a cost of c_p.

We give a randomized $O(\log k)$-competitive online algorithm for the generalized caching problem, improving the previous bound of $O(\log^2 k)$ by Bansal, Buchbinder, and Naor (STOC'08). This improved bound is tight and of the same order as the known bounds for the classic problem with uniform weights and sizes. We use the same LP based techniques as Bansal et al. but provide improved and slightly simplified methods for rounding fractional solutions online.

*Research supported by the Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, EPSRC award EP/D063191/1, and EPSRC grant EP/F043333/1.

†Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick. {A.M.Adamaszek, A.Czumaj, M.Englert}@warwick.ac.uk.

‡Institut für Informatik, Technische Universität München. raecke@informatik.tu-muenchen.de.

1 Introduction

We consider the *generalized caching* problem in the online setting using the framework of competitive analysis. In the basic two-level caching problem we are given a collection of n *pages* and a *cache* (a fast access memory). The cache has a limited capacity and can store up to k pages. At each time step a request to a specific page arrives and can be served directly if the corresponding page is in the cache; in that case no cost is incurred. If the requested page is not in the cache, a page fault occurs and in order to serve the request the page must be fetched into the cache possibly evicting some other page and a cost of one unit is incurred. The goal is to design an algorithm that specifies which page to evict in case of a fault such that the total cost incurred on the request sequence is minimized.

This classic problem can be naturally extended to the *Generalized Caching Problem* that we study, by allowing pages to have non-uniform fetching costs and to have non-uniform sizes. This is well motivated by caching of web pages, as these typically have non-uniform size and also non-uniform fetching cost due to their different locations in the network (for a more detailed discussion, see e.g., [BBN08, Ira02]).

In the general model we are given a collection of n pages. Each page p is described by a fetching cost $c_p \geq 0$ and a size $w_p \geq 1$. The cache has limited capacity and can only store pages up to a total size of at most k. In this paper, we present a randomized $O(\log k)$-competitive algorithm for this problem, which is asymptotically optimal as already for the problem with uniform page sizes and uniform fetching costs there is a lower bound of $\Omega(\log k)$ on the competitive ratio of any randomized online algorithm [FKL+91].

1.1 Related work

The study of the caching problem with uniform sizes and costs (*the paging problem*) in the online setting has been initiated by Sleator and Tarjan [ST85] in their work that introduced the framework of competitive analysis. They show that well-known paging rules like LRU or FIFO are k-competitive, and that this is the best competitive ratio that any deterministic algorithm can achieve.

Fiat et al. [FKL+91] extend the study to the randomized setting and design a randomized $2H_k$-competitive algorithm, where H_k is the k-th Harmonic number. They also prove that no randomized online paging algorithm can be better than H_k-competitive. Subsequently, McGeoch and Sleator [MS91], and Achlioptas et al. [ACN00] designed randomized online algorithms that achieve this competitive ratio.

Weighted caching, where pages have uniform sizes but can have arbitrary cost, has been studied extensively because of its relation to the k-server problem. The results for the k-server problem on trees due to Chrobak et al. [CKPV91] yield a tight deterministic k-competitive algorithm for weighted caching. The randomized complexity of weighted caching has been resolved only recently, when Bansal et al. [BBN07] designed a randomized $O(\log k)$-competitive algorithm.

The caching problem with non-uniform page sizes seems to be much harder. Already the offline-version is NP-hard [Ira02], and there was a sequence of results [Ira02, AAK99, CK99] that lead to the work of Bar-Noy et al. [BNBYF+01] which gives a 4-approximation for the offline problem.

For the online version, the first results consider special cases of the problem. Irani [Ira02] introduces the *Bit Model* in which each page p has a cost $c_p = w_p$ (i.e., minimizing fetching cost

corresponds to minimizing network traffic), and the *Fault Model* in which the fetching costs are uniform and only the page sizes vary (i.e., $c_p = 1$ for every page p). She shows that in the deterministic case LRU is $(k+1)$-competitive for both models. Cao and Irani [CI97], and Young [You02], extend this result to the generalized caching problem.

In the randomized setting, Irani [Ira02] gives an $O(\log^2 k)$-competitive algorithm for both models, but for the generalized caching problem no $o(n)$-competitive ratio was known until the recent work of Bansal et al. [BBN08]. They showed how to obtain a competitive ratio of $O(\log^2 k)$ for the general model, and also a competitive ratio of $O(\log k)$ for the Bit Model and the Fault Model.

1.2 Result and Techniques

We give a randomized $O(\log k)$-competitive online algorithm for the generalized caching problem, improving the previous bound of $O(\log^2 k)$ by Bansal et al. [BBN08]. This improved bound unifies all earlier results for special cases of the caching problem: it is asymptotically tight and of the same order as the known bounds for the classical problem with uniform weights and sizes.

Our approach is similar to the approach used by Bansal et al. in their results on weighted caching [BBN07] and generalized caching [BBN08]. In both these papers the authors first formulate a packing/covering linear program that forms a relaxation of the problem. They can solve this linear program in an *online*-manner by using the online primal-dual framework for packing and covering problems introduced by Buchbinder and Naor [BN05]. However, using the framework as black-box only guarantees an $O(\log n)$-factor between the cost of the solution obtained online and the cost of an optimal solution. They obtain an $O(\log k)$-guarantee by tweaking the framework for the special problem using ideas from the primal-dual scheme. Note that this $O(\log k)$-factor is optimal, i.e., in general one needs to loose a factor of $\Omega(\log k)$ when solving the LP online.

In the second step, they give a randomized rounding algorithm to transform the fractional solution into a sequence of integral cache states. Unfortunately, this step is quite involved. In [BBN08] they give three different rounding algorithms for the general model and the more restrictive Bit and Fault Models, where in particular the rounding for the Fault Model is quite complicated.

We use the same LP as Bansal et al. [BBN08] and also the same algorithm for obtaining an online fractional solution. Our contribution is a more efficient and also simpler method for rounding the fractional solution online. We first give a rounding algorithm for monotone cost models (where $w_p \geq w_{p'}$ implies $c_p \geq c_{p'}$) and then extend it to work for the General Model. Our rounding algorithm only looses a constant factor and, hence, we obtain an $O(\log k)$-competitive algorithm for generalized caching.

2 The Linear Program

This section describes an LP for the generalized caching problem. It also shows how to generate good variable assignments which are used in the rounding algorithm of the next section. Although there are some minor notational differences, this largely follows the work by Bansal, Buchbinder, and Naor [BBN08] and is only included to make the paper more self contained.

We assume that cost is incurred when a page is evicted from the cache, not when it is loaded into the cache. This means we will not pay anything for the pages remaining in the cache at the end.[1]

We begin with describing an integer program IP for the generalized caching problem. The IP has variables $x_p(i)$ indicating if page p has been evicted from the cache after the page has been requested for the i-th time, i.e., if $x_p(i) = 1$, page p was evicted after the i-th request to page p and has to be loaded back into the cache when the page is requested for the $(i+1)$-th time. The total cost is then $\sum_p \sum_i c_p x_p(i)$.

Let $B(t)$ denote the set of pages that have been requested at least once until and including time t and let $r(p,t)$ denote the number of requests to page p until and including time t. In a time step t in which page p_t is requested, the total size of pages other than p_t in the cache can be at most $k - w_{p_t}$. Thus, we require $\sum_{p \in B(t) \setminus \{p_t\}} w_p (1 - x_p(r(p,t))) \leq k - w_{p_t}$. Rewriting this constraints gives $\sum_{p \in B(t) \setminus \{p_t\}} w_p x_p(r(p,t)) \geq \sum_{p \in B(t)} w_p - k$. To shorten notation, we define the total weight $W(S) := \sum_{p \in S} w_p$ of a set of pages S. Altogether, this results in the following IP formulation for the generalized caching problem:

$$\begin{aligned} \min \quad & \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall t: \quad & \textstyle\sum_{p \in B(t) \setminus \{p_t\}} w_p \, x_p(r(p,t)) \geq W(B(t)) - k \\ \forall p, i: \quad & x_p(i) \in \{0,1\} \end{aligned} \tag{IP 1}$$

To decrease the integrality gap of our final LP relaxation, we add additional, redundant constraints to this IP.

$$\begin{aligned} \min \quad & \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall t \, \forall S \subseteq B(t) \text{ with } W(S) > k: \quad & \\ & \textstyle\sum_{p \in S \setminus \{p_t\}} w_p \, x_p(r(p,t)) \geq W(S) - k \\ \forall p, i: \quad & x_p(i) \in \{0,1\} \end{aligned} \tag{IP 2}$$

Unfortunately, even the relaxation of this IP formulation can have an arbitrarily large integrality gap. However, in an integral solution any $w_p > W(S) - k$ cannot give any additional advantage over $w_p = W(S) - k$ for a constraint involving set S. Therefore, it is possible to further strengthen the constraints without affecting an integral solution. For this, we define $\tilde{w}_p^S := \min\{W(S) - k, w_p\}$. Relaxing the integrality constraint, we obtain an LP. As shown in Observation 2.1 of Bansal et al. [BBN08], we can assume without loss of generality that $x_p(i) \leq 1$. This results in the final LP

[1] We may assume that the last request is always to a page of size k and zero cost. This does not change the cost of any algorithm in the original cost model. However, it does ensure that the cost in our alternate cost model matches the cost in the original model.

Procedure 1 fix-set(S, t, x, y)

Input: The current time step t, current variable assignments x and y, a minimal set S.

Output: New assignments for x and y. Return if S becomes non-minimal or constraint t, S in primal-LP is satisfied.

while $\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \cdot x_p(r(p,t)) < W(S) - k$ **do**
  // constraint for set S and time t is not fulfilled
  infinitesimally increase $y_S(t)$
  for each $p \in S$ **do**
    $v := \sum_{t' | \tau \in (r(p,t),t]} \sum_{S \ni p} \tilde{w}_p^S \cdot y_S(t') - c_p$
    // v violates dual constraint for $x_p(r(p,t))$
    if $v \geq 0$ **then** $x_p(r(p,t)) := \frac{1}{k} \exp\left(\frac{v}{c_p}\right)$
    if $x_p(r(p,t)) = 1$ **then**
      return *// S is not minimal anymore*
  end for
end while
return
// the primal constraint for step t and set S is fulfilled

formulation.

$$\begin{aligned} \min \quad & \textstyle\sum_p \sum_i c_p \, x_p(i) \\ \text{s.t. } \forall t \, \forall S \subseteq B(t) \text{ with } W(S) > k: & \\ & \textstyle\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \, x_p(r(p,t)) \geq W(S) - k \\ \forall p, i: \quad & x_p(i) \geq 0 \end{aligned} \tag{primal-LP}$$

The dual of primal-LP is

$$\begin{aligned} \max \quad & \textstyle\sum_{t=1}^{n} \sum_{S \subseteq B(t)} (W(S) - k) \, y_S(t) \\ \text{s.t. } \forall t, p: \quad & \textstyle\sum_{\tau \,|\, t \in (r(p,\tau),\tau]} \sum_{S \ni p} \tilde{w}_p^S \, y_S(\tau) \leq c_p \\ \forall t, S: \quad & y_S(t) \geq 0 \ . \end{aligned} \tag{dual-LP}$$

Procedure 1 will be called by our online rounding algorithm to generate assignments for the LP variables. Note that the variable assignments will not necessarily result in a feasible solution to primal-LP but will have properties which are sufficient to guarantee that our rounding procedure produces a feasible solution. We assume that all primal and dual variables are initially zero.

For a time step t, we say a set of pages S is *minimal* if, for every $p \in S$, $x_p(r(p,t)) < 1$. We note that by Observation 2.1 of Bansal et al. [BBN08], whenever there is a violated constraint t, S in primal-LP, there also is a violated constraints $t, S' \subseteq S$ for a minimal set S'. The idea behind Procedure 1 is that it is called with a minimal set S. The procedure then increases the primal (and dual) variables of the

current solution in such a way that one of two things happen: either the set S is not minimal anymore because $x_p(r(p,t))$ reaches 1 for some page $p \in S$ or the constraint t, S is not violated anymore. At the same time, the following theorem guarantees that the primal variables are not increased too much, that is, that the final cost is still bounded by $O(\log k)$ times the cost of an optimal solution. Its proof follows exactly the proof of Theorem 3.1 from Bansal et al. [BBN08].

Theorem 2.1 *The total cost incurred by successive calls to the Procedure 1 is at most $O(\log k)$ times the cost of an optimal solution to the caching problem. In other words, let OPT be the cost of an optimal solution to the caching problem, then $\sum_{p=1}^{n}\sum_{t=1}^{n} c_p x_p(t) \leq O(\log k)\cdot\mathsf{OPT}$, where the $x_p(t)$ are the final variable assignments generated by calls to Procedure 1.*

3 The Online Algorithm

The online algorithm for the generalized caching problem works as follows. It computes primal and dual assignments for LPs primal-LP and dual-LP, respectively, by repeatedly finding violated primal constraints and passing the constraint together with the current primal and dual solution to Procedure 1. Algorithm 2 gives the outline of a single step of the online algorithm.

Note that the primal "solution" may not be feasible but may only fulfill a subset of the constraints, which, however, will be sufficient for our rounding procedure.

In addition to these fractional solutions, the online algorithm maintains a probability distribution μ over cache states. More concretely, μ will be the uniform distribution over k^2 subsets—each subset D specifying a set of pages that are currently evicted from the cache. A randomized algorithm then chooses a random number r from $[1,\ldots,k^2]$ and behaves according to the r-th subset, i.e., whenever the r-th subset changes it performs the corresponding operations.

We will design μ in such a way that it closely mirrors the primal fractional solution x. Then we can use results from the previous section to show that each set in the support of μ is a valid complement of a cache-state, i.e., it fulfills size constraints and has the currently requested page in the cache.

Algorithm 2 online-step(t)

1: $x(p_t, r(p_t,t)) := 0$ // *put page p_t into the cache*
2: // *buffer constraints may be violated*
3: $S := \{p \in B(t) \mid \gamma \cdot x(p, r(p,t)) < 1\}$
4: **while** constraint for S is violated **do**
5: fix-set(S, t, x, y) // *change the current solution*
6: adjust distribution μ to mirror new x
7: // *recompute S*
8: $S := \{p \in B(t) \mid \gamma \cdot x(p, r(p,t)) < 1\}$
9: **end while**
10: // *buffer constraints are fulfilled*

We will show the following

1. Each set D in the support of the distribution μ is a valid complement of a cache-state.

2. A change in the fractional solution of the LP that increases the fractional cost by ϵ is accompanied by a change in the distribution μ with (expected) cost at most $O(\epsilon)$.

3.1 Ensuring that cache-states are valid

In order to show the first part we will set up some constraint for the sets D that will guarantee that they describe valid complements of cache-states. In order to define these constraints we introduce the following notation.

Let t denote the current time-step and and set $x_p := x(p, r(p,t))$. Let $\gamma \geq 2$ denote a scaling factor to be chosen later, and define $z_p := \min\{\gamma x_p, 1\}$. The variable z_p is a scaling of the primal fractional solution x_p. We also introduce a rounded version of this: we define $\bar{z}_p := \lfloor k \cdot z_p \rfloor / k$, which is simply the z_p-value rounded down to the nearest multiple of $1/k$. Note that due to the way the LP-solution is generated $z_p > 0$ implies that $z_p \geq \gamma/k$. Therefore, rounding down can only change the z_p-value by a small factor. More precisely, we have $\bar{z}_p \geq (1 - 1/\gamma) \cdot z_p$.

We use S to denote the set of pages that are fractional in the scaled solution, i.e., have $z_p < 1$ (or equivalently $\bar{z}_p < 1$). We divide these pages into classes as follows. The class S_i contains pages whose size falls into the range $[2^i, 2^{i+1})$.

We construct a set $L \subseteq S$ of "large pages" by selecting pages from S in decreasing order of size (ties broken according to page-id) until either the $\bar{z}$-values of selected pages add up to at least 1, or all pages in S have been selected. We use w_ℓ to denote the size of the smallest page in L, and i_ℓ to denote its class-id. Note that this construction guarantees that either $1 \leq \sum_{p \in L} \bar{z}_p \leq 2$ or $L = S$. The following claim shows that the second possibility only occurs when the weight of S is small or while the online algorithm is serving a request (for example when the online algorithm iterates through the while-loop of Algorithm 2).

Claim 3.1 *After a step of the online algorithm, we either have* $1 \leq \sum_{p \in L} \bar{z}_p \leq 2$ *or* $W(S) \leq k$.

Proof. If $W(S) \leq k$ there is nothing to prove. Otherwise, we have to show that we do not run out of pages during the construction of the set L. Observe that after the while-loop of Algorithm 2 finishes, the linear program enforces the following condition for subset S:

$$\sum_{p \in S} \min\{W(S) - k, w_p\} \cdot x_p \geq W(S) - k \ .$$

In particular, this means that $\sum_{p \in S} x_p \geq 1$ and hence $\sum_{p \in S} \bar{z}_p \geq (1 - 1/\gamma)\gamma \geq 1$, as $\gamma \geq 2$. Since the $\bar{z}$-values of pages in S sum up to at least 1 we will not run out of pages when constructing L. ■

Let D denote a subset of pages that are evicted from the cache. With a slight abuse of notation we also use D to denote the characteristic function of the set, i.e., for a page p we write $D(p) = 1$ if p belongs to D and $D(p) = 0$ if it does not. We are interested whether at time t the set D describes a valid cache state.

Definition 3.2 *We say that a subset D of pages* γ-mirrors *the fractional solution x if:*

1. $\forall p \in B(t) : \bar{z}_p = 0$ *implies* $D(p) = 0$ *(i.e., p is in the cache).*

2. $\forall p \in B(t) : \bar{z}_p = 1$ *implies* $D(p) = 1$ *(i.e., p is evicted from the cache).*

3. *For each class* S_i*:* $\lfloor \sum_{p \in S_i} \bar{z}_p \rfloor \leq \sum_{p \in S_i} D(p)$.

4. $\lfloor \sum_{p \in L} \bar{z}_p \rfloor \leq \sum_{p \in L} D(p)$.

Here $\bar{z}$ is the solution obtained after scaling x by γ and rounding down to multiples of $1/k$.

We will refer to the constraints in the first two properties as *integrality constraints*, to the constraints in the third property as *class constraints*, and the constraint in the fourth property is called the *large pages constraint*.

Lemma 3.3 *A subset D of pages that γ-mirrors the fractional solution x to the linear program, describes a valid complement of a cache state for $\gamma \geq 16$.*

Proof. Let D denote a set that mirrors the fractional solution x. In order to show that D is a valid complement of a cache-state we need to show that the page p_t that is accessed at time t is not contained in D, and we have to show that the size of all pages that are not in D sums up to at most k.

Observe that the fractional solution is obtained by applying Procedure 1. Therefore, at time t the variable $x_{p_t} \triangleq x(p_t, r(p_t, t))$ will be 0. (It is set to 0 when Algorithm 2 is called for time t, and it is not increased by Procedure 1 until time $t+1$.) Hence, we have $\bar{z}_{p_t} = 0$ and therefore Property 1 guarantees that p_t will not be in D.

It remains to show that there are enough pages evicted in D. This means we have to show

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p D(p) \geq W(B(t)) - k \ . \tag{1}$$

Because of Property 1 and Property 2 we have

$$\begin{aligned}
&\sum_{p \in B(t) \setminus \{p_t\}} w_p D(p) \\
&\qquad = \sum_{p \in B(t) \setminus S} w_p D(p) + \sum_{p \in S \setminus \{p_t\}} w_p D(p) \\
&\qquad = \sum_{p \in B(t) \setminus S} w_p + \sum_{p \in S} w_p D(p) \\
&\qquad = W(B(t)) - W(S) + \sum_{p \in S} w_p D(p) \ .
\end{aligned}$$

Therefore it suffices to show $\sum_{p \in S} w_p D(p) \geq W(S) - k$, in order to obtain Equation 1. For the case that $W(S) \leq k$ this is immediate, since the left hand side is always non-negative. Therefore in the following we can assume that $W(S) > k$, and, hence, $1 \leq \sum_{p \in L} \bar{z}_p \leq 2$ due to Claim 3.1.

If $2^{i_\ell} \geq W(S) - k$, then

$$\begin{aligned}
\sum_{p \in S} w_p D(p) &\geq \sum_{p \in L} w_p D(p) \\
&\geq 2^{i_\ell} \sum_{p \in L} D(p) \geq 2^{i_\ell} \geq W(S) - k \ ,
\end{aligned}$$

where the third inequality follows from Property 4, and the fact that $\sum_{p\in L}\bar{z}_p \geq 1$.

In the remainder of the proof we can assume $2^{i_\ell} < W(S)-k$. We have

$$\begin{aligned}
\sum_{p\in S} w_p D(p) &\geq \sum_{i\leq i_\ell}\sum_{p\in S_i} w_p D(p) \\
&\geq \sum_{i\leq i_\ell} 2^i \cdot \sum_{p\in S_i} D(p) \\
&\geq \sum_{i\leq i_\ell} 2^i \cdot \Big(\sum_{p\in S_i}\bar{z}_p - 1\Big) \\
&= \frac{1}{2}\sum_{i\leq i_\ell}\sum_{p\in S_i} 2^{i+1}\bar{z}_p - \sum_{i\leq i_\ell} 2^i \\
&\geq \frac{1}{2}\sum_{i\leq i_\ell}\sum_{p\in S_i} w_p\bar{z}_p - 2^{i_\ell+1} \\
&\geq \frac{\gamma}{4}\sum_{i\leq i_\ell}\sum_{p\in S_i} \tilde{w}_p^S x_p - 2(W(S)-k) \ .
\end{aligned} \tag{2}$$

Here the second inequality follows since $w_p \geq 2^i$ for $p\in S_i$; the third inequality follows from Property 3; the fourth inequality holds since $w_p \leq 2^{i+1}$ for $p\in S_i$. The last inequality uses the fact that $\bar{z}_p \geq (1-1/\gamma)\gamma x_p \geq \gamma/2\cdot x_p$ for every $p\in S$, and that $w_p \geq \tilde{w}_p^S$.
Using the fact that $\bar{z}_p \geq \gamma/2\cdot x_p$ we get

$$\begin{aligned}
\frac{\gamma}{4}\sum_{p\in L\setminus S_{i_\ell}} \tilde{w}_p^S x_p &\leq \frac{1}{2}\sum_{p\in L}\tilde{w}_p^S\bar{z}_p \\
&\leq \frac{1}{2}(W(S)-k)\sum_{p\in L}\bar{z}_p \leq W(S)-k \ ,
\end{aligned}$$

where the last inequality uses the fact that $\sum_{p\in L}\bar{z}_p \leq 2$. Adding the inequality $0 \geq \frac{\gamma}{4}\sum_{p\in L\setminus S_{i_\ell}} \tilde{w}_p^S x_p - (W(S)-k)$ to Equation 2 gives

$$\begin{aligned}
\sum_{p\in S} w_p D(p) &\geq \frac{\gamma}{4}\sum_{p\in S}\tilde{w}_p^S x_p - 3(W(S)-k) \\
&\geq (\gamma/4-3)(W(S)-k) \geq W(S)-k \ ,
\end{aligned}$$

for $\gamma \geq 16$. Here the second inequality holds because after serving a request the online algorithm guarantees that the constraint $\sum_{p\in S}\tilde{w}_p^S x_p \geq W(S)-k$ is fulfilled for the current set S (if the set has at least weight k, of course). ■

3.2 Updating the distribution in an online-manner

We have to show how to update the distribution μ over cache-complements in such a way that we can relate the update cost to the cost of our linear programming solution. We show that in each step the subsets in the support *mirror* the current linear programming solution. Then Lemma 3.3 guarantees that we have a distribution over complements of valid cache states.

However, directly ensuring all properties in Definition 3.2 leads to a very complicated algorithm. Therefore, we partition this step into two parts. We first show how to maintain a distribution μ_1 over subsets D that fulfill the first three properties in Definition 3.2 (i.e., the integrality constraints and the class constraints). Then we show how to maintain a distribution μ_2 over subsets that fulfill Property 1 and Property 4.

From these two distributions we obtain our distribution μ as follows. We sample a subset D_1 from the first distribution and a subset D_2 from the second distribution, and compute $D = D_1 \cup D_2$ (or $D = \max\{D_1, D_2\}$ if D is viewed as the characteristic function of the set).

Clearly, if both sets D_1 and D_2 fulfill Property 1 then the union fulfills Property 1. Furthermore, if one of D_1, D_2 fulfills one of the properties 2, 3, or 4, then the corresponding property is fulfilled by the union as these properties only specify a lower bound on the characteristic function D.

We will construct μ_1 and μ_2 to be uniform distributions over k subsets. Then the combined distribution μ is a uniform distribution over k^2 subsets (note however that some of the subsets may actually be identical).

In the following we assume that the $\bar{z}_p$-values change in single steps by $1/k$. This is actually not true. Consider, for example Line 1 of Algorithm 2, for a page p_t that is requested at time t (after t is increased). As x_{p_t} is a shorthand for the variable $x(p_t, r(p_t, t))$, these two statements may cause a drop in the variable $\bar{z}_{p_t}$ that is larger than $1/k$. However, we can simulate larger changes by several consecutive changes of value $1/k$.

3.2.1 Maintaining distribution μ_1

In order to be able to maintain the distribution μ_1 at a small cost we strengthen the conditions that the sets D in the support have to fulfill. For each size class S_i we introduce cost classes $C_i^0, C_i^1, \ldots$, where $C_i^s = \{p \in S_i : c_p \geq 2^s\}$. For the subsets D in the support of μ_1 we require

A. For each subset D, for each size class S_i, and for all cost classes C_i^s

$$\left\lfloor \sum_{p \in C_i^s} \bar{z}_p \right\rfloor \leq \sum_{p \in C_i^s} D(p) \leq \left\lceil \sum_{p \in C_i^s} \bar{z}_p \right\rceil \ .$$

B. For each page p

$$\sum_D D(p) \cdot \mu_1(D) = \bar{z}_p \ .$$

Note that the second constraint above ensures that the integrality constraints are fulfilled, and that the first set of constraints ensures that the class constraints are fulfilled (because $C_i^0 = S_i$).

Increasing $\bar{z}_p$. Suppose that the $\bar{z}_p$ value for some page p increases by $1/k$. Assume that $p \in S_i$ and $c_p \in [2^r, 2^{r+1})$, i.e., $p \in C_i^0, \ldots, C_i^r$. As we have to satisfy the property $\sum_D D(p)\mu_1(D) = \bar{z}_p$, we have to add the page p to a set D^* in the distribution μ_1, which currently does not contain p (i.e., we have to set $D^*(p) = 1$ for this set). We choose this set arbitrarily.

However, after this step the constraints $\lfloor \sum_{p \in C_i^s} \bar{z}_p \rfloor \leq \sum_{p \in C_i^s} D(p) \leq \lceil \sum_{p \in C_i^s} \bar{z}_p \rceil$, for $s \leq r$ may be violated for some sets. We repair the violated constraints step by step from $s = r$ to 0.

Fix s and assume that the constraints hold for all $s' > s$ but some are violated for s. We first assume that the change of $\bar{z}_p$ did not change the value of $a := \lceil \sum_{p \in C_i^s} \bar{z}_p \rceil$. Then D may now have $a+1$ pages

from C_i^s, which is not allowed (for all other sets the constraint for s did not change and is still valid). However, $\sum_{p\in C_i^s} \bar{z}_p = \sum_D \sum_{p\in C_i^s} D(p)\cdot\mu_1(D)$ is equal to the average number of pages from C_i^s that a set in the support of μ_1 has. This is at most a and D has $a+1$ pages. Therefore there must exist a set D' with positive support in μ_1 that has at most $a-1$ pages from C_i^s. The constraint for $s+1$ tells us that the number of pages in class C_i^{s+1} differs at most by 1 between D and D'. Hence, there must exist a page in $C_i^s \setminus C_i^{s+1}$ that is in D but not in D'. We move this page to D' which incurs an expected cost of at most $2^{s+1}/k$. Now, the constraint for s is satisfied and the constraints for values $s' > s$ are still satisfied.

For the case that a increased, observe that D^* is now the only set with $a = \lceil \sum_{p\in C_i^s} \bar{z}_p \rceil$ pages from cost class C_i^s. Furthermore, note that after the change $a > \sum_{p\in C_i^s} \bar{z}_p$ as otherwise there would not have been a change. Therefore, there cannot be a set with less than $a-1 = \lfloor \sum_{p\in C_i^s} \bar{z}_p \rfloor$ pages. Hence, all constraints are fulfilled.

Performing the above procedure incurs expected cost of $2^{s+1}/k$ for s from r to 0. In total we have expected cost $O(2^r/k)$. The increase of $\bar{z}_p$ increases the LP-cost by at least $2^r/k$. Therefore, the cost in maintaining the distribution μ_1 can be amortized against the increase in LP-cost.

Decreasing $\bar{z}_p$. When the $\bar{z}_p$-value for a page p with $c_p \in [2^r, 2^{r+1})$ decreases then we have to delete the page p from a set D in the support of μ_1 that currently contains it. The analysis for this case is completely analogous to the case of an increase in $\bar{z}_p$. The resulting cost of $O(2^r/k)$ can be amortized against the LP-cost. (At a loss of a constant factor we can amortize $O(2^r/k)$ against the LP-cost when $\bar{z}_p$ increases and the same value when $\bar{z}_p$ decreases.)

Change of set S. The class constraints depend on the set S that is dynamically changing. Therefore we have to check whether they are fulfilled if a page enters or leaves the set S. When a page p with $c_p \in [2^r, 2^{r+1})$ increases its $\bar{z}_p$-value to 1 we first add it to the only set in the support of μ_1 that does not contain it. This induces an expected cost of at most $2^{r+1}/k$. Then we fix Constraint A, as described in the procedure for increasing a $\bar{z}_p$-value. This also induces expected cost $O(2^r/k)$. After that we remove the page from the set S. Constraint A will still be valid because every cost-class that contains p changes $\sum_{p\in C_i^s} \bar{z}_p$ and $\sum_{p\in C_i^s} D(p)$ by exactly 1.

3.2.2 Maintaining distribution μ_2.

For this part we only have to guarantee that a set in the support of μ_2 fulfills the large pages constraint and does not evict any page for which $\bar{z}_p = 0$.

In the following we introduce an alternative way of thinking about this problem. A variable $\bar{z}_p$ can be in $k+1$ different states $\{0, 1/k, 2/k, \ldots, 1-1/k, 1\}$. We view the $k-1$ non-integral states as *points*. We say that the *ℓ-th point for page p appears* (*becomes active*) if the variable $\bar{z}_p$ changes from $(\ell-1)/k$ to ℓ/k. Points can disappear for two reasons: Suppose the ℓ-th point for page p is active. We say that it *disappears (becomes inactive)* if either the $\bar{z}_p$-value of p decreases form ℓk to $(\ell-1)/k$, *or* when the $\bar{z}_p$-value reaches 1.

Note that a $\bar{z}_p$-value of 1 means that the page is not in the set S, and that it only enters S once its $\bar{z}_p$-value is decreased to 0 again. The appearance of a point for page p corresponds to a cost of

c_p/k of the LP-solution. At a loss of a factor of 2 we can also amortize c_p/k against the cost of the LP-solution when a point for page p disappears.

Observation 3.4 *The set of pages with active points is the set of pages in S with non-zero $\bar{z}_p$-value.*

The above observation says that if we guarantee to only evict pages that have an active point we guarantee our first constraint that no page with $\bar{z}_p = 0$ is evicted.

We assign priorities to the points, according to the size of the corresponding page (large pages have highest priorities). Ties are broken according to page-ids and point-numbers (in this order). Let at any point in time the set Q denote the set of the k *active* points with largest priority (or all active points if there are less than k). The following observation follows directly from the definition of Q and L as we used the same tie-breaking mechanisms for both constructions.

Observation 3.5 *For any time-step, the set of pages in L that have a non-zero $\bar{z}_p$-value is exactly the set of pages that have at least one point in Q.*

We assign labels from $\{1,\ldots,k\}$ to active points, with the meaning that if a point a has label ℓ, then the ℓ-th set in the support of μ_2 has the page corresponding to a evicted. At each point in time the ℓ-th set consists of pages for which one of the corresponding points has label ℓ. In general we will allow a point to have several labels. Note that this definition of the sets in the support of μ_2, directly ensures that a page that has $\bar{z}_p = 0$ cannot be evicted in any set, because a page with this property does not have any active points.

Adding a label to a point a increases the expected cost of the online algorithm by at most c_p/k, where p is the page corresponding to a. Deleting a label is for free, and in particular if a point disappears (meaning its labels also disappear) the online algorithm has no direct cost while we still can amortize c_p/k against the LP-cost.

The following observation forms the basis for our method of maintaining μ_2.

Observation 3.6 *If the points in Q have different labels, then all sets in the support of the distribution μ_2 fulfill the large pages constraint (Property 4 in Definition 3.2).*

This means that we only have to show that there is a labeling scheme that on the one hand has a small re-labeling cost, i.e., the cost for re-labeling can be related to the cost of the LP-solution, and that on the other hand guarantees that at any point in time no two points from Q have the same label. We first show that a very simple scheme exists if the cost-function is monotone in the page-size, i.e., $w_p \leq w_{p'}$ implies $c_p \leq c_{p'}$ for any two pages p, p'. Note that the bit-model and the fault-model that have been analyzed by Bansal et al. [BBN08] form monotone cost functions. Therefore, the following section gives an alternative proof for an $O(\log k)$-competitive algorithm for these cost models.

Maintaining μ_2 for monotone cost

We show how to maintain a labeling of the set Q such that all labels assigned to points are different. Assume that currently every point in the set Q has a single unique label.

Appearance of a point q. Suppose that a new point q arrives. If q does not belong to the k points with largest priority, it will not be added to Q and we do not have to do anything.

If the set Q currently contains strictly less than k points, then the new point will be contained in the new set Q, but at least one of the k labels has been unused before and we can label q with it. In the new set Q all points have different labels. The online algorithm paid a cost of $c_{p(q)}/k$, where $p(q)$ denotes the page corresponding to point q.

If Q already contains k pages, then upon appearance of q, a point q' with lower priority is removed from Q and q is added. We can assign the label of q' to the new point q, and then all points in the new set Q have different labels. Again the online algorithm pays a cost of $c_{p(q)}/k$.

In all cases the online algorithm pays at most $c_{p(q)}/k$ whereas the LP-cost is $c_{p(q)}/k$.

Disappearance of a point q. Now, suppose that a point q in the current set Q is deleted. This means that a point q' with smaller priority than q may be added to the set Q (if there are at least k points in total). We give q' the label that q had. This incurs a cost of $c_{p(q')}/k \leq c_{p(q)}/k$, where the inequality holds due to the monotonicity of the cost function. Since we can amortize $c_{p(q)}/k$ against the cost of the LP-solution we are competitive.

Maintaining μ_2 for general cost.

We want to assign labels to points in Q in such a way that we are guaranteed to see k different labels (if Q contains at least k points). In the last section we did this by always assigning different labels to points in Q. For the case of general cost functions we proceed differently.

Let Q_i denote the k active points with largest priority that correspond to pages with cost at least 2^i (in case there are less than k such points, Q_i contains all active points corresponding to pages with cost at least 2^i).

Essentially our goal is to have a labeling scheme with small re-labeling cost that guarantees that a set Q_i sees at least $|Q_i|$ different labels. Since $Q = Q_0$ this gives the result. However, for the case of general cost, it will not be sufficient anymore to assign unique labels to points, but we will sometimes be assigning several different labels to the same point. At first glance, this may make a re-labeling step very expensive in case a point with a lot of labels disappears.

To avoid this problem we say that a set Q_i has to commit to a unique label for every point q contained in it (and the chosen label must be from the set of labels assigned to q). The constraint for Q_i is that it commits to different labels for all points contained in it. If a point currently has labels ℓ and ℓ', then a set Q_i may either commit to ℓ or ℓ', but furthermore during an update operation it may change the label it commits to for free (i.e., no cost is charged to the online-algorithm). (Recall that if a point for a page p has several labels then all sets corresponding to these labels have the page p evicted; therefore committing to a different label is for free as no page has to be evicted from any set.)

Appearance of a point q. Suppose that a point q corresponding to a page p with $c_p \in [2^r, 2^{r+1})$ appears. We assign an arbitrary label to this point and make all subsets Q_s that contain q commit to this label for q. Sets Q_s, $s > r$ are not affected by the appearance of q at all, and their constraints remain valid. We only have to fix the condition for sets Q_s, $s \leq r$,.

Assume that the condition holds for all sets $Q_{s'}$, $s' > s$ but is violated for s. We call a label ℓ a *duplicate label* for Q_s if there exist two points in Q_s for which Q_s commits to ℓ. We call the corresponding points *duplicate points*. We call a label ℓ *free* for Q_s if currently there is no point in Q_s for which Q_s commits to ℓ. When we start processing Q_s there exists exactly one duplicate label namely the label that we assigned to q and to which Q_s committed.

Since the total number of labels is k and there are at most k points in Q_s there must exist a free label ℓ_{free}. We could fix the condition for Q_s by re-labeling one of the duplicate points with ℓ_{free}. However, this would create a cost that depends on the cost of the page corresponding to the chosen point. This may be too large, as our aim is to only pay $O(2^s/k)$ for fixing the condition for set Q_s. Therefore, we will successively change the labels that Q_s commits to until the cost of one of the duplicate points is in $[2^s, 2^{s+1})$. During this process we will maintain the invariant that there are at most two duplicate points for Q_s. Hence, in the end we can re-label a duplicate point with cost at most 2^{s+1} and arrive at a set Q_s that fulfills the condition that it commits to a different label for each of its points.

The process for changing the commitments of set Q_s is as follows. Suppose that currently ℓ denotes the duplicate label and that the two duplicate points both correspond to pages with cost at least 2^{s+1}. This means that both points are in set Q_{s+1}. As the constraint for Q_{s+1} is fulfilled we know that Q_{s+1} commits to different labels for these points. One of these labels must differ from ℓ. Let q' denote the duplicate point for which Q_{s+1} commits to a label $\ell' \neq \ell$. We switch the commitment of set Q_s for point q' from ℓ to ℓ'. (Now, ℓ' may be the new duplicate label for set Q_s)

The above process can be iterated. With each iteration the number of points in the intersection of Q_s and Q_{s+1} for which both sets commit to the same label increases by one. Hence, after at most k iterations we either end up with a set Q_s that fulfills the condition (i.e., has no duplicate points) or one of the duplicate points corresponds to a page with cost at most 2^{s+1}.

As we only pay cost $2^{s+1}/k$ for fixing Q_s the total payment summed over all sets Q_s, $s \leq r$ is $O(2^r/k)$, which can be amortized to the LP-cost.

Disappearance of a point q. Now, suppose that a point q corresponding to a page p with $c_p \in [2^r, 2^{r+1})$ is deleted. Then a new point may enter the sets Q_s, $s \leq r$ (the only case for which this does not happen is when Q_s already contains all active points corresponding to pages with cost at least 2^s). For each Q_s we commit to an arbitrary label for this point (recall this doesn't induce any cost). Now, for each Q_s we have the same situation as in the case when a point appears. The set either fulfills its condition or has exactly two duplicate points. As before we can fix the condition for set Q_s at cost $O(2^s/k)$.

References

[AAK99] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 31–40, 1999.

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[BBN07] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 507–517, 2007.

[BBN08] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 235–244, 2008.

[BN05] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing problems. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 689–701, 2005.

[BNBYF+01] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[CI97] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

[CK99] Edith Cohen and Haim Kaplan. Lp-based analysis of greedy-dual-size. In *SODA*, pages 879–880, 1999.

[CKPV91] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[FKL+91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[Ira02] Sandy Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002.

[MS91] Lyle A. McGeoch and Daniel D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[You02] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.