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

ANNA ADAMASZEK, University of Copenhagen, Denmark
ARTUR CZUMAJ, University of Warwick, UK
MATTHIAS ENGLERT, University of Warwick, UK
HARALD RÄCKE, Technical University Munich, Germany

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 paging 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.

CCS Concepts: • **Theory of computation → Caching and paging algorithms**; *Design and analysis of algorithms*; *Online algorithms*;

Additional Key Words and Phrases: Online primal dual, knapsack cover inequalities

ACM Reference Format:
Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. 1111. An $\boldsymbol{O(\log k)}$-competitive algorithm for Generalized Caching. *ACM Trans. Algor.* 1, 1, Article 1 (January 1111), 18 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

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.

*A preliminary version of this paper appeared in proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA), 2012, pages 1681-1689.

Authors' addresses: Anna Adamaszek, Department of Computer Science (DIKU), University of Copenhagen, Denmark, anad@di.ku.dk; Artur Czumaj, Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry, UK, A.Czumaj@warwick.ac.uk; Matthias Englert, Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry, UK, M.Englert@warwick.ac.uk; Harald Räcke, Department for Informatics, Technical University Munich, Garching, Germany, raecke@in.tum.de.

1549-6325/1111/1-ART1 $15.00
https://doi.org/0000001.0000001

	uniform sizes	arbitrary sizes
uniform costs	$2H_k$-competitive [10] H_k-competitive [1, 12] lower bound H_k [10]	$O(\log^2 k)$-competitive [11] $O(\log k)$-competitive [4]
arbitrary costs	$O(\log k)$-competitive [3]	$O(\log^2 k)$-competitive [4] $\boldsymbol{O(\log k)}$**-competitive**

Table 1. An overview of the results for the online caching problem in the randomized setting. The new result is highlighted in bold.

This classical problem can be naturally extended to the *generalized caching problem*, by allowing pages to have non-uniform fetching costs and to have non-uniform sizes. 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. The framework of generalized caching has been motivated by applications in web caching and networking. The non-uniform sizes of the pages can correspond to the scenarios of caching web pages of different sizes, and the non-uniform costs of fetching a page can model scenarios in which the pages have different locations in a large network.

Various special cost models have been proposed in the literature. In the *bit model* [4, 11], each page p has $c_p = w_p$, and thus, for example, minimizing the fetching cost can correspond to minimizing the total traffic in the network. In the *fault model* [4, 11], for each page we have the fetching cost $c_p = 1$ and the size w_p may be arbitrary; in this case the fetching cost corresponds to the number of times a user has to wait for a page to be retrieved. In the *weighted caching* model [3, 11], for each page p we have the size $w_p = 1$ and the fetching cost c_p may be arbitrary; this models situations where some pages are more expensive to fetch than others because they may be on far away servers, or slower disks.

We consider the *online* version of the generalized caching problem. In this version as soon as a page is requested, it has to be loaded into the cache, and while processing the request we have no information about the sequence of pages which will be requested later.

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 [13] in their work that introduced the framework of competitive analysis. They show that well known paging rules like LRU (Least Recently Used) or FIFO (First In First Out) are k-competitive, and that this is the best competitive ratio that any deterministic algorithm can achieve.

Fiat et al. [10] 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 [12], and Achlioptas et al. [1] design 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. [8] 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. [3] 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 [11], and there was a sequence of results [2, 9, 11] that lead to the work of Bar-Noy et al. [5] which gives a 4-approximation for the offline problem.

For the online version, the first results consider special cases of the problem. Irani [11] shows that for the bit model and for the fault model in the deterministic case LRU is $(k + 1)$-competitive. Cao and Irani [7], and Young [14], extend this result to the generalized caching problem.

In the randomized setting, Irani [11] gives an $O(\log^2 k)$-competitive algorithm for both bit and fault models, but for the generalized caching problem no $o(n)$-competitive ratio was known until the work of Bansal et al. [4]. They show 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. Table 1 presents an overview of the results for the online caching problem in the randomized setting.

Since it has been known that no randomized $o(\log k)$-competitive online algorithm for generalized caching exists, the central open problem in this area was whether it is possible to design a randomized $O(\log k)$-competitive algorithm.

1.2 Result and Techniques

We present 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. [4]. This improved bound unifies all earlier results for special cases of the caching problem. It 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 [10].

Our approach is similar to the approach used by Bansal, Buchbinder, and Naor in their results on weighted caching [3] and generalized caching [4]. 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 [6]. However, using the framework as a 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 this specific problem. Note that this $O(\log k)$-factor is optimal, i.e., in general one needs to lose 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 [4] 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. [4] 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. In particular, our rounding algorithm maintains a distribution over only k^2 integral cache states. The approaches by Bansal et al. [3, 4] do not give a good bound on the number of states maintained by the rounding algorithm.

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 loses 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 of Bansal, Buchbinder, and Naor [4].

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 last time we load a page into the cache that remains in the cache at the end. In the general model, 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.

We begin by 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. 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 $(i + 1)$-st 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

$$\sum_{p \in B(t) \setminus \{p_t\}} w_p(1 - x(p, r(p, t))) \leq k - w_{p_t} \quad .$$

Rewriting the constraint gives

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

To shorten the notation, we define the total weight of a set of pages S as $W(S) := \sum_{p \in S} w_p$. Altogether, this results in the following IP formulation for the generalized caching problem.

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

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

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

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

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

Input: Current time step t, current variable assignments x and y, a minimal set $S \subseteq B(t)$.
Output: Terminates if S becomes non-minimal or constraint t, S in primal-LP is satisfied and returns the new assignments for x and y.

```
while $\sum_{p \in S \setminus \{p_t\}} \tilde{w}_p^S \cdot x(p, r(p,t)) < W(S) - k$ do {constraint for $t, S$ violated}
    infinitesimally increase $y_S(t)$
    for each $p \in S$ do
        $v := \sum_{\tau: r(p,\tau) = r(p,t), p \neq p_\tau} \sum_{S \subseteq B(\tau): p \in S, W(S) > k} \tilde{w}_p^S \, y_S(\tau) - c_p$
        {$v$ is a violation of the 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 any more}
    end for
end while
return {the primal constraint for step $t$ and set $S$ is fulfilled}
```

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. [4], we can assume without loss of generality that $x(p, i) \leq 1$. This results in the final LP formulation.

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

The dual of primal-LP is

$$
\begin{aligned}
\max \quad & \sum_t \sum_{S \subseteq B(t): W(S) > k} (W(S) - k) \, y_S(t) \\
\text{s.t. } \forall_{p,i} : \quad & \sum_{t: r(p,t) = i, p \neq p_t} \sum_{S \subseteq B(t): p \in S, W(S) > k} \tilde{w}_p^S \, y_S(t) \leq c_p \\
\forall_t \forall_{S \subseteq B(t): W(S) > k} : \quad & y_S(t) \geq 0 \ .
\end{aligned}
\qquad \text{(dual-LP)}
$$

Procedure 1 will be called by our online rounding algorithm to generate assignments for the LP variables. Note that, as the procedure will not be called for all violated constraints, 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. [4], whenever there is a violated constraint t, S in primal-LP, there is also a violated constraint $t, S' \subseteq S$ 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 any more because the value of $x(p, r(p,t))$ reaches 1 for some page $p \in S$ or the constraint t, S is not violated any more. 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. [4].

Theorem 2.1. *Let $x(p,i)$ be the final variable assignments generated by successive calls to Procedure 1 (with different subsets). The total cost $\sum_p \sum_i c_p x(p,i)$ is at most $O(\log k)$ times the cost of an optimal solution to the caching problem.*

Observe that our online algorithm generates the calls to Procedure 1. Sometimes it may succeed in constructing a rounded solution from an infeasible assignment to the $x(p,i)$'s. In this case it will not continue to call Procedure 1 and, hence, the final assignment to the $x(p,i)$'s may not be feasible. However, this is not a problem for the analysis.

3 THE ONLINE ALGORITHM

The online algorithm for the generalized caching problem works as follows. It computes primal and dual assignments x and y 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.

Procedure 2 online-step(t)

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

In addition to the fractional solutions x and y, the online algorithm maintains a probability distribution over valid cache states. Specifically, μ will be the uniform distribution over k^2 subsets — each subset specifying the set of pages that are not present in the cache. Some of the k^2 subsets may be identical. A randomized algorithm then chooses a random number r from $[1,\dots,k^2]$ and behaves according to the r-th subset, i.e., whenever the r-th subset changes it performs the corresponding operations. Note that this rounding approach differs substantially from the results by Bansal, Buchbinder and Naor [3, 4], since it constructs a distribution over a small number of cache states. The approaches in [3] and [4] do not allow to obtain a good upper bound on the number of states.

We will design the distribution μ in such a way that it closely mirrors the primal fractional solution x. In Section 3.1 we will show that each set in the support of μ is a complement of a valid cache state, i.e., the size constraints are fulfilled and the currently requested page is contained in the cache. In Section 3.2 we will show the way of updating μ in such way that 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 $O(\varepsilon)$.

The rounding algorithm loses only a constant factor, which gives us a $O(\log k)$-competitive algorithm for generalized caching.

Procedure 2 gives the outline of a single step of the online algorithm, whereγ is a scaling factor which is explained in the following.

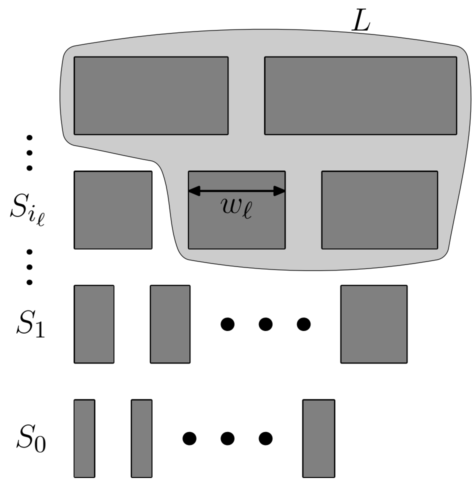

Fig. 1. The size classes S_i and the set of large pages L obtained from the set S.

3.1 Ensuring that Cache States are Valid

We will set up some constraints for the sets in the support of μ, which guarantee that the sets describe complements of valid 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 the scaling: we define $\bar{z}_p := \lfloor k \cdot z_p \rfloor / k$, which is simply the value of z_p 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 value of z_p 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 p that are fractional in the scaled solution, i.e., have $z_p < 1$ (or equivalently $\bar{z}_p < 1$). We divide these pages into *size classes* as follows. The class S_i contains pages whose size falls into the range $[2^i, 2^{i+1})$. See Figure 1 for an illustration.

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 values of $\bar{z}$ for the 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 $1 \leq \sum_{p \in L} \bar{z}_p < 2$ or $L = S$. The following claim shows that the second possibility only occurs when the weight of S is small compared to the size of the cache k or while the online algorithm is serving a request (for example when the online algorithm iterates through the while-loop of Procedure 2).

Claim 3.1. *After a step of the online algorithm, we either have* $1 \leq \sum_{p \in L} \bar{z}_p < 2$ *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 Procedure 2 finishes, the linear program enforces the following condition for the 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 values of $\bar{z}_p$ for the pages p 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 complement of 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:* $\lfloor \sum_{p\in S_i} \bar{z}_p \rfloor \leq \sum_{p\in S_i} D(p)$. *(class constraints)*
(4) $\lfloor \sum_{p\in L} \bar{z}_p \rfloor \leq \sum_{p\in L} D(p)$. *(large pages constraint)*

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

We 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 of pages that γ-mirrors the fractional solution x to the linear program, describes a complement of a valid 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 complement of a valid cache state we need to show that the page p_t which is accessed at time t is not contained in D, and that the size of all pages which 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 $x_{p_t} = x(p_t, r(p_t, t))$ has value 0. (It is set to 0 when Procedure 2 is called for time t, and it is not increased by Procedure 1.) Hence, we have $\bar{z}_{p_t} = 0$ and Property 1 in Definition 3.2 guarantees that p_t is not in D.

It remains to show that the size of all pages which are not in D sums up to at most k. 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 the integrality constraints we have

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

In order to obtain Equation 1 it suffices to show that

$$\sum_{p\in S\setminus\{p_t\}} w_p D(p) \geq W(S) - k \ .$$

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, $1 \leq \sum_{p\in L} \bar{z}_p < 2$ due to Claim 3.1. Recall that w_ℓ is the size of the smallest page in L, and that i_ℓ denotes the corresponding class-id.

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

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

where the third inequality follows from the large pages constraint, 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

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

Here the second inequality follows since $w_p \geq 2^i$ for $p \in S_i$, the third inequality follows from the class constraints, and 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

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

where the last inequality uses $\sum_{p\in L} \bar{z}_p \leq 2$. 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

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

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. □

3.2 Updating the Distribution Online

We will show how to update the distribution μ over subsets of pages online in such a way, that we can relate the update cost to the cost of our linear programming solution x. We show that in each step the subsets in the support of μ *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 the first and the last property.

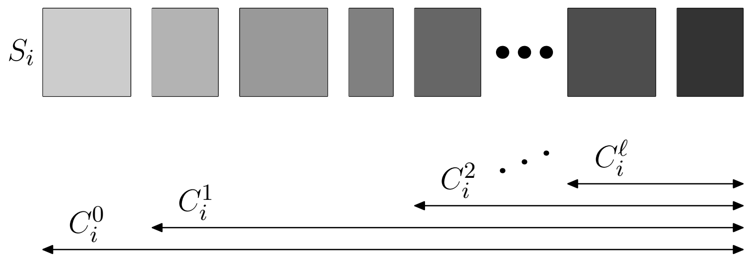

Fig. 2. Cost classes.

From these two distributions we obtain the 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 from Definition 3.2, 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, where some of the subsets may be identical.

In the following we assume that the values of $\bar{z}_p$ change in single steps by $1/k$. This is actually not true. Consider for example Line 1 of Procedure 2 where, after the time step t is increased, the variable $x(p_t, r(p_t, t))$ is set to 0. As x_{p_t} is a shorthand for $x(p_t, r(p_t, t))$, the value of x_{p_t}, and therefore also the value of $\bar{z}_{p_t}$, is set to 0. However, the drop in the value of $\bar{z}_{p_t}$ larger than $1/k$ can be simulated by several consecutive changes by a value of $1/k$.

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 of μ_1 have to fulfill. For each size class S_i we introduce *cost classes* $C_i^0, C_i^1, \ldots$ where $C_i^s = \{p \in S_i : c_p \geq 2^s\}$ (see Figure 2). Note that we have $S_i = C_i^0$.

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

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

B. For each page p

$$\sum\nolimits_D D(p) \cdot \mu_1(D) = \bar{z}_p \quad .$$

Note that the first set of constraints ensures that class constraints are fulfilled. This holds because $C_i^0 = S_i$. Hence, the left inequality for C_i^0 is exactly the class constraint for class S_i. The second set of constraints imply the integrality constraints. An example of a distribution that satisfies the constraints A and B is given in Figure 3.

Increasing $\bar{z}_p$. Suppose that for some page p the value of $\bar{z}_p$ increases by $1/k$ (see Figure 4 for an example). 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.

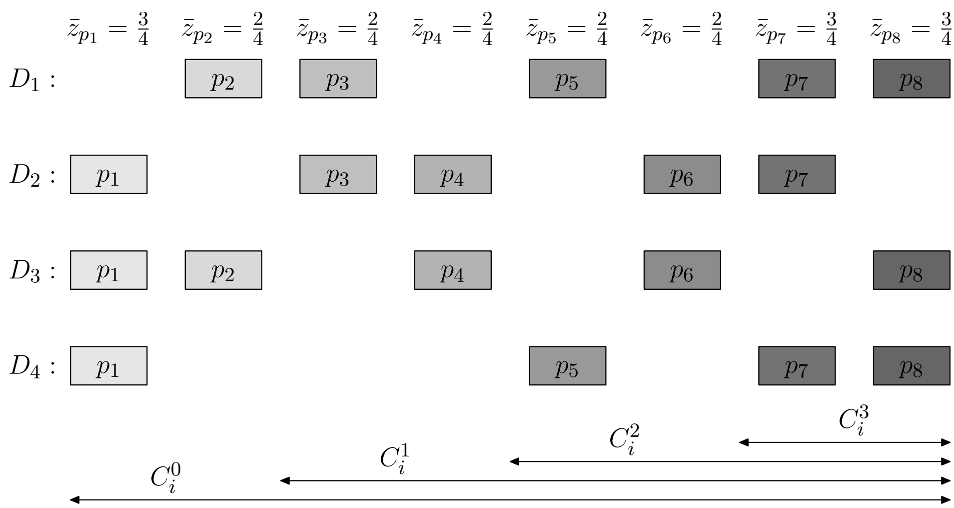

Fig. 3. An example of distribution μ_1 for the cache of size $k = 4$ and the set of pages $S_i = \{p_1, p_2, \ldots, p_8\}$. The values $\bar{z}_{p_i}$ are given at the top of the figure. μ_1 is a uniform distribution over 4 sets: $D_1 = \{p_2, p_3, p_5, p_7, p_8\}$, $D_2 = \{p_1, p_3, p_4, p_6, p_7\}$, $D_3 = \{p_1, p_2, p_4, p_6, p_8\}$, and $D_4 = \{p_1, p_5, p_7, p_8\}$. Constraints A and B are satisfied.

However, after this step some of the constraints

$$\left\lfloor \sum\nolimits_{p \in C_i^s} \bar{z}_p \right\rfloor \le \sum\nolimits_{p \in C_i^s} D(p) \le \left\lceil \sum\nolimits_{p \in C_i^s} \bar{z}_p \right\rceil$$

corresponding to the cost classes C_i^s for $s \le r$ may become violated. We repair the violated constraints step by step from $s = r$ to 0. We do that by moving the pages between the sets D in such a way, that while repairing the constraints for the cost class C_i^s we keep the following invariant: all but one of the sets D from the support of μ have the same number of pages from the set C_i^s, as they had before we increased the value of $\bar{z}_p$. The remaining set, which we denote by D^+, has one additional page. At the beginning $D^+ = D^*$.

Notice that $\sum_{p \in C_i^s} \bar{z}_p = \sum_D \sum_{p \in C_i^s} D(p) \cdot \mu_1(D)$ is equal to the average number of pages from C_i^s that a set in the support of μ_1 has. If the number of pages from C_i^s in the sets D in the support of μ_1 differs by at most one, each set has either $\lfloor \sum_{p \in C_i^s} \bar{z}_p \rfloor$ or $\lceil \sum_{p \in C_i^s} \bar{z}_p \rceil$ pages from C_i^s, and all the constraints for C_i^s are satisfied.

Fix s and assume that the constraints hold for all $s' > s$, but some are violated for s. Let $a := \lceil \sum_{p \in C_i^s} \bar{z}_p - \frac{1}{k} \rceil$, i.e., before increasing the value of $\bar{z}_p$ each set D contained at most a, and at least $a - 1$, pages from C_i^s. As the only set that has now a different number of pages from C_i^s is D^+, and some constraints for C_i^s are violated, it must be the case that D^+ has now $a + 1$ pages from C_i^s, and some set D' with positive support in μ_1 has $a - 1$ pages from C_i^s. The constraints for C_i^{s+1} are satisfied, so the number of pages in class C_i^{s+1} differs by at most 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 all the sets in the support of μ_1 have either $a - 1$ or a pages from C_i^s, and so all the constraints for C_i^s are satisfied. As we did not modify any pages from C_i^{s+1}, the constraints for values $s' > s$ remain satisfied. Now the set D' has one additional page, and it becomes the new set D^+.

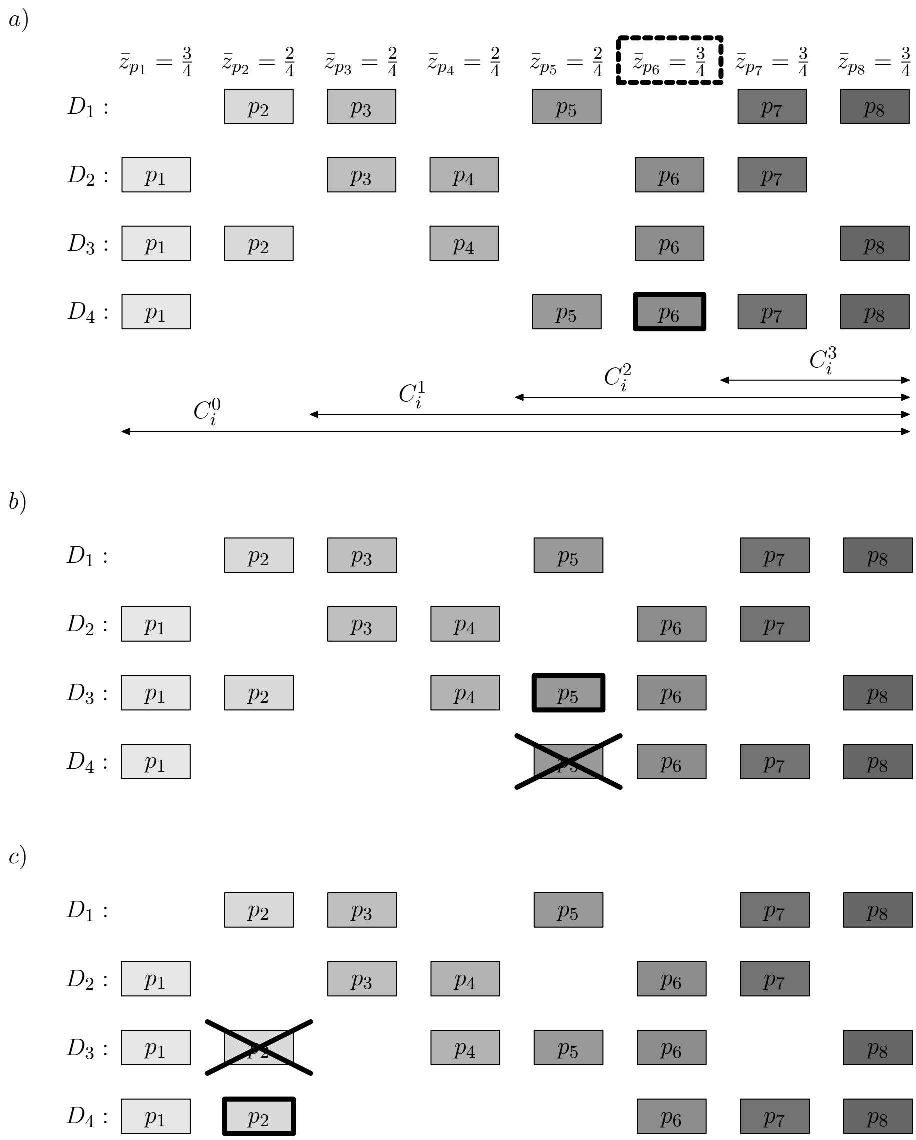

Fig. 4. In the setting as in Figure 3 the value of $\bar{z}_{p_6}$ increases by $1/k = 1/4$. *a)* To satisfy Constraint B, we add page p_6 to the set D_4. After this modification Constraint A is not satisfied for the cost class $C_i^2 = \{p_5, p_6, p_7, p_8\}$ and the set D_4. *b)* To satisfy Constraint A for the cost class C_i^2 we move the page p_5 from D_4 to D_3. Now Constraint A is satisfied for the cost classes C_i^3, C_i^2 and C_i^1, but it is violated for C_i^0 and the sets D_3, D_4. *c)* To satisfy Constraint A for the cost class C_i^0 we move the page p_2 from D_3 to D_4. Now all the constraints are satisfied.

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

Change of the set S. The class constraints depend on the set S that is dynamically changing. Therefore we have to check whether the constraints 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 for every cost class C_i^s that contains p and for every set D in the support of μ_1 the values of $\sum_{p\in C_i^s} \bar{z}_p$ and $\sum_{p\in C_i^s} D(p)$ change by exactly 1.

3.2.2 Maintaining Distribution μ_2. We will show how to maintain a distribution μ_2 over subsets of pages, such that each set D in the support of μ_2 fulfills the large pages constraint and does not contain any page p for which $\bar{z}_p = 0$. Note that as $\sum_{p\in L} \bar{z}_p < 2$, the large pages constraint can be reformulated as follows: if $\sum_{p\in L} \bar{z}_p \geq 1$ then each subset D in the support of μ_2 contains at least one page from the set of large pages L.

In the following we introduce an alternative way of thinking about this problem. A variable $\bar{z}_p$ can be in one of $k+1$ 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* (or *becomes active*) if the value of $\bar{z}_p$ 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* (or *becomes inactive*) if either the value of $\bar{z}_p$ decreases from ℓ/k to $(\ell-1)/k$, or when the value of $\bar{z}_p$ reaches 1.

Note that if for a page p we have $\bar{z}_p = 1$, the page p is not in the set S, and it only enters S once the value of $\bar{z}_p$ 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 the value $\bar{z}_p \neq 0$.*

The above observation says that if we guarantee that a set D in the support of μ_2 can contain only those pages from S which have an active point, we guarantee one of our constraints — the set D does not contain any page p for which $\bar{z}_p = 0$.

We assign priorities to the active points, according to the size of the corresponding page, where points corresponding to larger pages have higher priorities. Ties are broken first according to page-ids, and then to the point-numbers. At any time step, Q denotes the set of the k active points with highest priority, or all active points if there are less than k (see Figure 5). 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 p in L that have a value of $\bar{z}_p \neq 0$ is exactly the set of pages that have at least one point in Q.*

We assign to active points labels from the set $\{1, \ldots, k\}$, with the meaning that if a point q has label ℓ, then the ℓ-th set in the support of μ_2 contains the page corresponding to q. At each point in

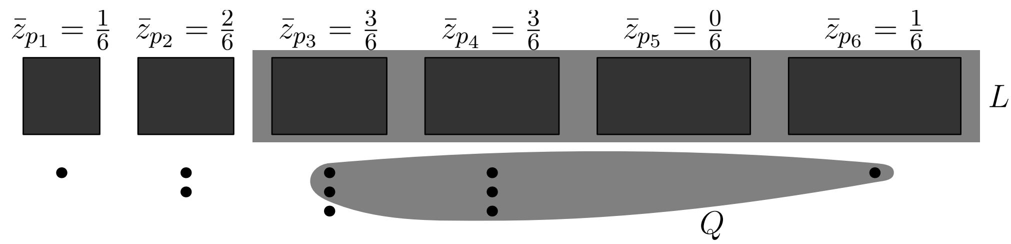

Fig. 5. Each page $p \in S$ with $\bar{z}_p = i/k$ has i corresponding points (here $k = 6$). The set of k points with highest priority (Q) and the set of large pages (L) are marked in grey.

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 $\bar{z}_p = 0$ is not contained in any set in the support of μ_2, because a page with this property does not have any active points.

Adding a label to a point q increases the expected cost of the online algorithm by at most $c_{p(q)}/k$, where $p(q)$ is the page corresponding to the point q. 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 can still amortize c_p/k against the LP-cost.

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

This means that we only have to show that there is a labeling scheme that on 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. [4] have monotone cost functions. Therefore, the following section gives an alternative proof for an $O(\log k)$-competitive algorithm for these cost models.

3.2.3 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 the highest 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 not been used 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 the 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 a lower priority than q may be added to the set Q (if there are at least k

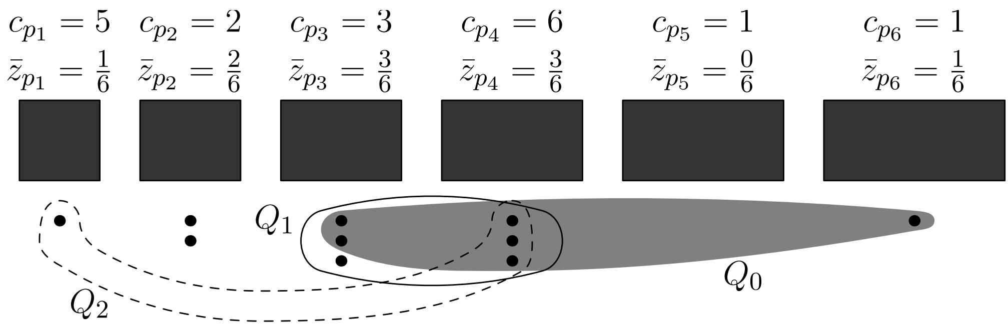

Fig. 6. Each page $p \in S$ with $\bar{z}_p = i/k$ has i corresponding points (here $k = 6$). The sets Q_i of points with the highest priority that correspond to the pages with cost at least 2^i have been marked. The sets Q_0 and Q_1 have k points each, and the set Q_2 has less than k points.

points in total). We give q' the same label that q had. This incurs a cost of $c_{p(q')}/k \le c_{p(q)}/k$, where the inequality holds due to the monotonicity of the cost function (note that this is the only place where monotonicity is used). Since we can amortize $c_{p(q)}/k$ against the cost of the LP-solution we are competitive.

3.2.4 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 the set 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 set of k active points with the highest 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 (see Figure 6). Note that $Q = Q_0$.

Essentially our goal is to have a labeling scheme with small re-labeling cost that guarantees that each set Q_i sees at least $|Q_i|$ different labels. Since $Q = Q_0$, this gives the desired result. However, for the case of general cost, it will not be sufficient any more 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, where the chosen label is from the set of all labels assigned to q (see Figure 7). 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 switch the label it commits to for free, i.e., no cost is charged to the online algorithm. Recall that if a point corresponding to a page p has several labels then all sets D corresponding to these labels contain the page p; therefore committing to a different label is for free as no change has to be made for any set D from the support of μ_2. The label to which a set Q_i commits for a point $q \in Q_i$ is denoted by $Q_i(q)$.

Appearance of a point q. Suppose that a point q_0 corresponding to a page p with $c_p \in [2^r, 2^{r+1})$ appears. This increases the cost of the LP by at least $\Omega(2^r/k)$. We assign an arbitrary label ℓ_0 to this point and, as ℓ_0 is the only label of q_0, we set $Q_s(q_0) = \ell_0$ for all subsets Q_s that contain q_0. Assigning a new label corresponds to evicting the page in some cache state D, and, consequently, induces an expected cost of at most $2^r/k$ for the online algorithm.

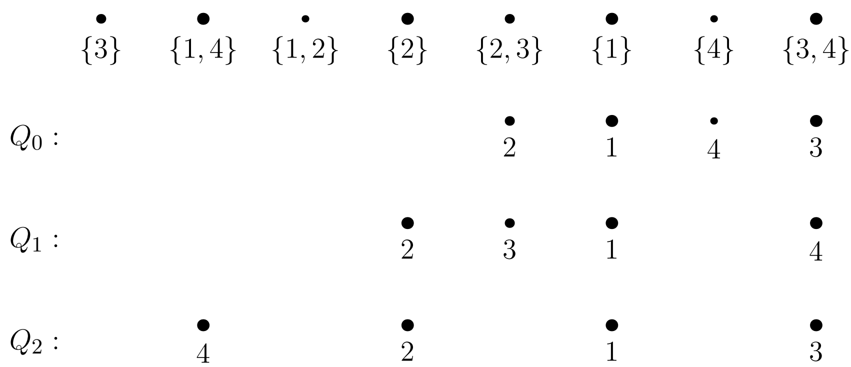

Fig. 7. Labelings for the sets Q_i ($k = 4$). The 8 active points are ordered increasingly with respect to the priority. The size of the dots corresponds do the cost of 1, 2 or 4 of the respective pages — larger dots represent larger costs. Each point has a set of labels assigned to it. Each set Q_i contains 4 points with the highest priorities amongst the points with cost at least 2^i. For each set Q_i we are given a valid labeling.

It remains to adjust the labeling so that the labeling of every set Q_i becomes valid again. The sets Q_s where $s > r$ are not affected by the appearance of q_0, and their labelings remain valid. We only have to fix the labelings for the sets Q_s where $s \leq r$. We will do this while only paying at most $O(2^s/k)$ for every $s \leq r$. This cost can be amortized against the cost of the LP.

Assume that labelings for all sets $Q_{s'}$ where $s' > s$ are valid, but the labeling for Q_s is violated. We want to fix it, paying only $O(2^s/k)$. 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*. The labeling is valid iff there exist no 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 at most one duplicate label, namely the label ℓ_0 that we assigned to q_0, and for which we have $Q_s(q_0) = \ell_0$.

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 assigning the label ℓ_{free} to one of the duplicate points q, and setting $Q_s(q) = \ell_{\text{free}}$. However, this would create a cost that depends on the cost of the page corresponding to the chosen point q. 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 switch the labels that Q_s commits to for the duplicate points, until the cost of one of the duplicate points q 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 assign the free label ℓ_{free} to a duplicate point q with cost at most 2^{s+1}, set $Q_s(q) = \ell_{\text{free}}$, and obtain a valid labeling for Q_s.

The process of switching the labels for the 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 the set Q_{s+1}. As the labeling for Q_{s+1} is valid, 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 set $Q_s(q') = \ell'$. Now, ℓ' may be the new duplicate label for the 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 has no duplicate points, or one of the duplicate points corresponds to a page with cost smaller than 2^{s+1}.

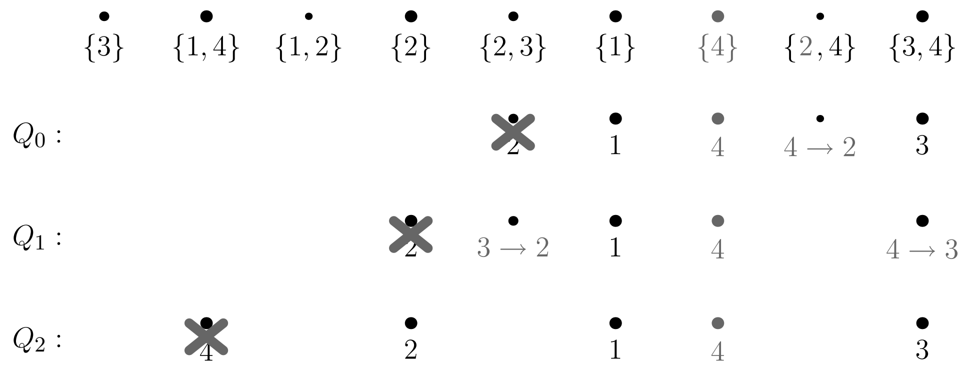

Fig. 8. In the setting as in Figure 7 a new point (marked in grey) arrives, which is assigned Label 4. Then the point with lowest priority disappears from each Q_i. Afterwards Q_2 contains k different labels. For Q_1 the rightmost element re-commits to Label 3, and afterwards the label of the leftmost element is changed to 2 at small cost. For Q_0 only the label of the third element is changed.

An example of fixing the labeling after adding a new point can be seen in Figure 8. As we only pay cost $2^{s+1}/k$ for fixing the labeling of Q_s, the total payment summed over all sets Q_s with $s \leq r$ is $O(2^r/k)$, which can be amortized to the cost of LP.

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 for which $s \leq r$. 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, as any point, when it becomes active, gets a label). Now, for each Q_s we have the same situation as in the case when a new 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 $O(2^s/k)$, and the total cost of $O(2^r/k)$ can be amortized to the cost of LP.

ACKNOWLEDGMENTS

This work is supported by the Centre for Discrete Mathematics and its Applicaitons (DIMAP) under EPSRC award EP/D063191/1 and by EPSRC grant EP/F043333/1.

REFERENCES

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. 2000. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science* 234, 1-2 (2000), 203–218. https://doi.org/10.1016/S0304-3975(98)00116-9

[2] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. 1999. Page Replacement for General Caching Problems. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 31–40. https://doi.org/10.1145/314500.314528

[3] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. 2012. A Primal-Dual Randomized Algorithm for Weighted Paging. *J. ACM* 59, 4 (2012), 19. https://doi.org/10.1145/2339123.2339126

[4] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. 2012. Randomized Competitive Algorithms for Generalized Caching. *SIAM J. Comput.* 41, 2 (2012), 391–414. https://doi.org/10.1137/090779000

[5] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. 2001. A Unified Approach to Approximating Resource Allocation and Scheduling. *J. ACM* 48, 5 (2001), 1069–1090. https://doi.org/10.1145/502102.502107

[6] Niv Buchbinder and Joseph Naor. 2009. Online Primal-Dual Algorithms for Covering and Packing. *Mathematics of Operations Research* 34, 2 (2009), 270–286. https://doi.org/10.1287/moor.1080.0363

[7] Pei Cao and Sandy Irani. 1997. Cost-Aware WWW Proxy Caching Algorithms. In *USENIX Symposium on Internet Technologies and Systems*. 193–206.

[8] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. 1991. New Results on Server Problems. *SIAM Journal on Discrete Mathematics* 4, 2 (1991), 172–181. https://doi.org/10.1137/0404017

[9] Edith Cohen and Haim Kaplan. 2002. Caching Documents with Variable Sizes and Fetching Costs: An LP-Based Approach. *Algorithmica* 32, 3 (2002), 459–466. https://doi.org/10.1007/s00453-001-0081-z

[10] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. 1991. Competitive Paging Algorithms. *Journal of Algorithms* 12, 4 (1991), 685–699. https://doi.org/10.1016/0196-6774(91)90041-V

[11] Sandy Irani. 2002. Page Replacement with Multi-Size Pages and Applications to Web Caching. *Algorithmica* 33, 3 (2002), 384–409. https://doi.org/10.1007/s00453-001-0125-4

[12] Lyle A. McGeoch and Daniel D. Sleator. 1991. A Strongly Competitive Randomized Paging Algorithm. *Algorithmica* 6, 6 (1991), 816–825. https://doi.org/10.1007/BF01759073

[13] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM* 28, 2 (1985), 202–208. https://doi.org/10.1145/2786.2793

[14] Neal E. Young. 2002. On-Line File Caching. *Algorithmica* 33, 3 (2002), 371–383. https://doi.org/10.1007/s00453-001-0124-5

Received February 2016; revised January 2018; accepted January 1111