Improved Space Bounds for Strongly Competitive Randomized Paging Algorithms*

Gabriel Moruz and Andrei Negoescu

Goethe University Frankfurt am Main. Robert-Mayer-Str. 11-15,
60325 Frankfurt am Main, Germany
{gabi,negoescu}@cs.uni-frankfurt.de

Abstract. Paging is a prominent problem in the field of online algorithms. While in the deterministic setting there exist simple and efficient strongly competitive algorithms, in the randomized setting a tradeoff between competitiveness and memory is still not settled. In this paper we address the conjecture in [2], that there exist strongly competitive randomized paging algorithms using $o(k)$ bookmarks, i.e. pages not in cache that the algorithm keeps track of. We prove tighter bounds for EQUITABLE2 [2], showing that it requires less than k bookmarks, more precisely $\approx 0.62k$. We then give a lower bound for EQUITABLE2 showing that it cannot both be strongly competitive and use $o(k)$ bookmarks. Our main result proves the conjecture that there exist strongly competitive paging algorithms using $o(k)$ bookmarks. We propose an algorithm, denoted PARTITION2, which is a variant of the PARTITION algorithm in [3]. While PARTITION is unbounded in its space requirements, PARTITION2 uses $\Theta(k/\log k)$ bookmarks.

1 Introduction

The paging problem is defined as follows. We have a two-level memory hierarchy consisting of a fast cache which can accommodate k pages, and a slow memory of infinite size. The input consists of requests to pages which are processed sequentially as follows. If the currently requested page is not in cache, a *cache miss* occurs and the requested page must be brought into cache. If the cache is full, a page must be evicted to accommodate the new one. The cost is given by the number of misses incurred.

Online algorithms in general and paging algorithms in particular are typically analyzed in the framework of *competitive analysis*[4,5]. An algorithm A is said to have a *competitive ratio* of c (or c-competitive) if its cost satisfies for any input $cost(A) \leq c \cdot cost(OPT) + b$, where b is a constant and $cost(OPT)$ is the cost of an optimal offline algorithm, i.e. an algorithm which is presented with the input in advance and processes it optimally; for randomized algorithms, $cost(A)$

* Partially supported by the DFG grants ME 3250/1-3 and MO 2057/1-1, and by MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation). A full version of the paper is available as technical report [1].

F.V. Fomin et al. (Eds.): ICALP 2013, Part I, LNCS 7965, pp. 757–768, 2013.
© Springer-Verlag Berlin Heidelberg 2013

is the expected cost of A. An algorithm achieving an optimal competitive ratio is *strongly competitive*. For paging, an optimal offline algorithm (OPT) evicts the page whose next request occurs the furthest in the future [6]. For comprehensive surveys on online algorithms in general and paging algorithms in particular, we refer the interested reader to [7,8].

Competitive ratio has often been criticized for its pessimistic quality guarantees. Especially in the deterministic setting, the empirically measured performance for practical algorithms is far below the theoretical guarantee of k provided by competitive analysis [9]. This gap is significantly smaller for randomized algorithms, since the best possible competitive ratio is H_k. Although using only the quality guarantees provided by competitive analysis is a naive way to distinguish good paging algorithms from bad ones, we have shown in [10] that ideas from competitive analysis for randomized algorithms can be successfully employed to design algorithms with good performance on real-world inputs. That is because an optimal randomized algorithm can be viewed as a collection of reasonable deterministic algorithms, and the algorithm designer can simply look for suitable algorithms in this collection.

Randomized paging algorithms have been well studied over the past two decades. In [11] a lower bound of H_k on the competitive ratio of randomized paging algorithms has been given[1]. Also in [11], a simple $(2H_k - 1)$-competitive algorithm, denoted MARK, has been proposed. In [12] it was shown that no randomized marking algorithm can achieve a competitive ratio better than $(2-\varepsilon)H_k$ for any $\varepsilon > 0$. The first strongly competitive paging algorithm, PARTITION, was proposed in [3]. While it is strongly competitive, its time and space usage are in the worst case proportional to the input size independent of the cache size, which is hopelessly high. More recent research focused on improving these bounds, especially the space requirements. Apart from the k pages in cache, a paging algorithm may store information about pages not in cache. In the literature, these "extra" pages are denoted *bookmarks*. An H_k-competitive algorithm, denoted EQUITABLE, using $O(k^2 \log k)$ bookmarks was proposed in [13]. A better version of EQUITABLE, denoted EQUITABLE2, improved this bound to $2k$ bookmarks [14]. This solved the open question in [8] that there exist H_k-competitive paging algorithms using $O(k)$ space. In [15] we proposed an algorithm, ONLINEMIN, which further improved EQUITABLE2 by reducing its runtime for processing a page from $O(k^2)$ to $O(\log k / \log \log k)$ while maintaining its space usage. Note that MARK and most deterministic algorithms use no bookmarks.

A distinct line of research for randomized paging algorithms considers fixed cache sizes ($k = 2$ and $k = 3$ to our best knowledge) to obtain tighter bounds than for general k. In [16], for $k = 2$, a $\frac{3}{2}$-competitive algorithm using only one bookmark was proposed. Still for $k = 2$, for randomized algorithms using no bookmarks lower and upper bounds on the competitive ratio of $\frac{37}{24} \approx 1.5416$ and ≈ 1.6514 respectively were given in [12,16]. In [14], strongly competitive randomized paging algorithms were proposed for $k = 2$ and $k = 3$, using 1 and 2 bookmarks respectively.

[1] $H_k = \sum_{i=1}^{k} 1/i$ is the kth harmonic number.

Our Contributions. This work focuses on the number of bookmarks needed by randomized algorithms to achieve the optimal competitive ratio of H_k. The best previously known result is $2k$ [14]. In [2] it was conjectured that there exist algorithms that use $o(k)$ bookmarks and are H_k-competitive. We first give a tighter analysis for EQUITABLE2 improving the amount of bookmarks from $2k$ to $\approx 0.62k$, which is the first solution using less than k bookmarks. We give a negative result showing that EQUITABLE2 cannot be H_k-competitive and use $o(k)$ bookmarks. Nonetheless, we show that it can trade competitiveness for space: if it is allowed to be (H_k+t)-competitive, it requires $k/(1+t)$ bookmarks.

We propose PARTITION2 which is a modification of the PARTITION algorithm. PARTITION2 improves the bookmark requirements of PARTITION from proportional to input size to $\Theta(k/\log k)$ and thus proves the $o(k)$ conjecture. For our analysis we provide a constructive equivalent between the two representations of the *offset functions* in [17] and [3]. Since offset functions are the key ingredient for optimal competitive paging algorithms, this may be of independent interest.

2 Preliminaries

Offset Functions and Layer Representation. For the paging problem it is possible to track online the exact minimal cost using *offset functions*. For a fixed input sequence σ and an arbitrary cache configuration C (i.e., a set of k pages), the *offset function* ω assigns to C the difference between the minimal cost of processing σ ending in configuration C and the minimal cost of processing σ. A configuration is called *valid* iff $\omega(C) = 0$. In [17] it was shown that the class of valid configurations $\mathcal{V}$ determines the value of ω on any configuration C by $\omega(C) = \min_{X\in\mathcal{V}}\{|C \setminus X|\}$. We can assume that OPT is always in a valid configuration. More precisely, if p is requested and there exists a valid configuration containing p, then the cost of OPT is 0; otherwise OPT pays 1 to process p.

In [17] it was shown for the paging problem that the offset function can be represented as a partitioning of the pageset in $k+1$ disjoint sets $L = (L_0|L_1|\ldots|L_k)$, denoted layers. An update rule for the layers when processing a page was also provided. Initially, the first k pairwise distinct requested pages are stored in layers $L_1, \ldots, L_k$, one page per layer, and L_0 contains the remaining pages. Upon processing page p, let L^p be the partitioning after processing p; we have[2]:

- $L^p = (L_0 \setminus \{p\}|L_1|\ldots|L_{k-2}|L_{k-1} \cup L_k|\{p\})$, if $p \in L_0$
- $L^p = (L_0|\ldots|L_{i-2}|L_{i-1} \cup L_i \setminus \{p\}|L_{i+1}|\ldots|L_k|\{p\})$, if $p \in L_i$, $i > 0$

This layer representation can keep track of all valid configurations. More specifically, a set C of k pages is valid iff $|C \cap L_i| \leq i$ holds for all $0 \leq i \leq k$ [17]. For a given L, denote by *support* $S(L) = L_1 \cup \ldots \cup L_k$. Also, a layer containing a single page is a *singleton*. Let r be the smallest index such that $L_r, \ldots, L_k$ are singletons. The pages in $L_r, \ldots, L_k$ are denoted *revealed*, the pages in support

[2] We use the layer representation introduced in [15], which is equivalent to the ones in [13,17].

which are not revealed are *unrevealed*, and the pages in L_0 are denoted *Opt-miss*. OPT faults on a request to p iff $p \in L_0$ and all revealed pages are (independent of the current request) in OPT's cache. If L has only revealed pages it is denoted a *cone* and we know the content of OPT's cache. We define the *signature* $\chi(L)$ as a k-dimensional vector $\chi = (x_1, \ldots, x_k)$, with $x_i = |L_i| - 1$ for each $i = 1, \ldots, k$.

Selection Process. In [15] we defined a priority-based selection process on L which is guaranteed to construct any valid configuration. Assuming that support pages have pairwise distinct priorities, we build a hierarchy of sets $C_0, \ldots, C_k$:

- $C_0 = \emptyset$
- C_i has the i pages in $C_{i-1} \cup L_i$ having the highest priorities, for all $i > 0$.

Note that, by definition, when constructing C_i there are $i + x_i$ candidates and i slots. Also, if L_i is singleton we have $x_i = 0$ and $C_i = C_{i-1} \cup L_i$; for singleton layers and only for singleton layers, all elements in both C_{i-1} and L_i make it to C_i and we say that no competition occurs. The outcome C_k contains k pages and is always a valid configuration.

Equitable, OnlineMin, and Forgiveness. The cache content of the Equitable algorithms [13,14] is defined by a probability distribution over the set of valid configurations. This distribution is achieved by OnlineMin using the previously introduced priority-based selection process, when priorities are assigned to support pages such that each permutation of the ranks of these pages is equally likely [15]. The cache content of OnlineMin is at all times the outcome C_k of the selection process. Nonetheless, the resulting probability distribution on cache configurations is the same as for Equitable [15], and in the rest of the paper we refer to this distribution and the associated algorithm as Equitable.

Note that the support size increases only when pages in L_0 are requested. As the number of Opt-miss requests may be very large, the support size and together with it the space usage of algorithms, such as Equitable, using it to decide their cache content may also be arbitrarily large. To circumvent this problem, the *forgiveness* mechanism is used. Intuitively, if the support size exceeds a given threshold, then the adversary did not play optimally and we can afford to use an approximation of the offset function with a layer representation bounded in size.

3 Better Bounds for Equitable2

There are two Equitable algorithms, Equitable [13] and Equitable2 [14][3]. For a fixed offset function, they have the same distribution as previously introduced. The difference between them is given by *forgiveness* mechanisms, which are used to approximate the current offset functions. In this section we focus on the Equitable2 algorithm using the forgiveness mechanism described in [14]

[3] In [14] Equitable2 is denoted K_Equitable. In this paper we use its original name.

which works as follows. Whenever the support size reaches $3k$ and an Opt-miss page is requested, the requested page is artificially inserted in L_1 and processed as a L_1 page. All pages in L_1 move to L_0 and the support size never exceeds $3k$. We give a tighter analysis and show that using the same forgiveness the algorithm uses less than k bookmarks and prove that $o(k)$ bookmarks is not possible. Finally, we show that it can trade competitiveness for space: if the algorithm uses $k/(1+t)$ bookmarks, then it is $(H_k + t)$-competitive, for $t \geq 0$.

To accommodate the selection process for OnlineMIN, all pages in support have pairwise distinct priorities, such that each priority ordering of the support pages is equally likely. We say that some page p has *rank* i in a set if its priority is the i'th largest among the elements in the given set.

In [13] an elegant potential function, based only on the current offset function, was introduced. Given the layer representation L, the potential $\Phi(L)$ is defined to be the cost of a so-called *lazy attack sequence*, that is, a sequence of consecutive requests to unrevealed pages until reaching a cone. The potential Φ is well defined because in the case of the Equitable distribution, all lazy attack sequences have the same overall cost for a given offset function [13].

Initially, we are in a cone and $\Phi = 0$. Upon a request to a page p in support, having cache miss probability $pb(p)$, by definition we have that $\Delta\Phi = -pb(p)$. On lazy requests OPT does not fault and thus $\Delta cost + \Delta\Phi = \Delta cost_{OPT} = 0$. Upon a request from L_0 both Equitable and OPT have cost 1 and it was shown that $\Delta\Phi \leq H_k - 1$ [13,14]. We thus have:

$$\Delta cost + \Delta\Phi \leq H_k \cdot \Delta cost_{OPT}.$$

If L is a cone, it is easy to verify that $\Delta\Phi = H_k - 1$ for a request in L_0. If the support size exceeds k, the difference in potential is smaller, i.e. $\Delta\Phi < H_k - 1$. This means that the algorithm pays less than its allowed cost and thus it can make *savings*, which can be tracked by a second potential function and pay for the forgiveness step when the support is large enough. While Φ is very convenient to use for requests in support, for arbitrary offset functions there is no known closed form for its exact actual value or for its exact change upon a request in L_0.

3.1 Approximation of Φ

The key ingredient to our analysis is to get a bound for $\Delta\Phi$ that is as tight as possible on requests in L_0. A tighter bound for this value implies larger savings, which in turn means that these savings can pay earlier (i.e. for a smaller support size) for a forgiveness step, which in the end means fewer bookmarks. We therefore analyze $\Delta\Phi$ for requests to pages in L_0 when no forgiveness step is applied. Note that Φ depends only on the signature $\chi = (x_1, \ldots, x_k)$ of the layer representation. We use $\chi = 0$ for the cone signature $(0|0|\ldots|0)$ and $\chi = e_i$ for the i-th unit vector $(0|\ldots|x_i = 1|\ldots|0)$. If $\chi = 0$ we have $\Phi = 0$. Otherwise, let i be the largest index such that $x_i > 0$. Since all lazy attack sequences have the same cost, we consider Φ as the cost of i consecutive requests, each of them to a page in the (current) first layer. For the layer representation L of the current

offset function, we let $cost_1(L)$ denote the probability of cache miss for a page p in L_1, i.e. $pb(p \notin C_k)$ in the selection process.

We start with the case when all layers are singletons except some layer L_i. The potential Φ for this particular case is given in Lemma 1. For some arbitrary values i, n, and γ, where $0 < i < \gamma \leq k$ consider the signatures $\chi = n \cdot e_i$ and $\chi' = n \cdot e_i + e_{\gamma-1}$; let L and L' be their corresponding layer representations. We define the difference in the cost for a request in L_1: $f(i, n, \gamma) = cost_1(\chi') - cost_1(\chi)$. In the special case $\gamma = k$ it represents $\Delta cost_1$ upon a request in L_0. The value for $f(i, n, \gamma)$ can be computed exactly and is given in Lemma 2, and in Lemma 3 we show that $f(i, n, \gamma)$ is an upper bound on $\Delta cost_1$ for a whole class of signatures. Lemma 4 provides an identity for approximating $\Delta\Phi$ for a request in L_0. The proofs for all these results are given in the full version.

Lemma 1. *Let $\chi = n \cdot e_i$ be the signature of L, where $n > 0$ and $0 < i < k$. We have $\Phi(\chi) = n \cdot (H_{i+n} - H_n)$.*

Lemma 2. *It holds that $f(i, n, \gamma) = \frac{1}{n+\gamma} \prod_{j=i}^{\gamma-1} \frac{j}{n+j}$.*

Lemma 3. *Consider a signature $\chi = (x_1| \ldots |x_k)$, and let i be the minimal index with $x_j = 0$ for all $j > i$. Also, let $\chi' = \chi + e_{\gamma-1}$, $i < \gamma \leq k$. For $n = x_1 + \cdots + x_i$, we have $cost_1(\chi') - cost_1(\chi) \leq f(i, n, \gamma)$.*

Lemma 4. *It holds that $\sum_{j=1}^{i} f(i-j+1, 1, \gamma-j+1) = H_\gamma - H_{\gamma-i} - \frac{i}{\gamma+1}$, for any i and γ with $i < \gamma$.*

Theorem 1. *For a request to a page $p \in L_0$ where no forgiveness is applied, let i be the largest index with $x_i > 0$; $i = 0$ if we are in a cone. We have that:*

$$H_{k-i} - H_1 \leq \Delta\Phi \leq H_k - H_1 - i/(k+1).$$

Proof. For $i = 0$, in a cone we have $\Delta\Phi = H_k - 1$ by Lemma 1. If $i > 0$, let L and L', and χ and $\chi' = \chi + e_{k-1}$ denote the layers and their corresponding signatures before and after the request to p respectively. We consider the cost of a sequence of i consecutive requests $p_1, \ldots, p_i$, each of these to pages in the current L_1. For each $j = 1, \ldots, i$ let χ^j and χ'^j denote the signatures before processing p_j. After the whole sequence is processed, we have $\chi = 0$ with $\Phi = 0$ and $\chi' = e_{k-i-1}$ with $\Phi' = H_{k-i} - H_1$ by Lemma 1. We get:

$$\Delta\Phi = H_{k-i} - H_1 + \sum_{j=1}^{i} \left(cost_1(\chi'^j) - cost_1(\chi^j) \right)$$

Since $cost_1(\chi'^j) - cost_1(\chi^j)$ is non-negative, the left inequality holds.

Now we bound $cost_1(\chi'^j) - cost_1(\chi^j)$ using Lemma 3. Before processing page p_j we have $x^j_{i-j+1} > 0$, $x^j_l = 0$ for all indices $l > i - j + 1$ and $\chi'^j = \chi^j + e_{\gamma-1}$ with $\gamma = k - j + 1$. Denoting $n^j = x^j_1 + \cdots + x^j_{i-j+1}$, and using the fact that f is decreasing in n, $n^j > 0$ for all $j \leq i$, and the result in Lemma 4, we get:

$$\Delta\Phi \leq \sum_{j=1}^{i} f(i-j+1, n_j, k-j+1) + H_{k-i} - H_1 \leq H_k - H_{k-i} - \frac{i}{k+1} + H_{k-i} - H_1 \, .$$

3.2 Competitiveness and Bookmarks

Having obtained a tighter bound on $\Delta\Phi$ for requests in L_0, we get improved savings using a second potential Ψ. To define $\Psi(L)$, we first introduce the concept of *chopped signature*. For some signature $\chi = (x_1|\dots|x_k)$, let i be the largest index such that $x_i > 0$. The chopped signature corresponding to χ is $\overline{\chi} = (\overline{x_1}|\dots|\overline{x_k})$, where $\overline{x_i} = x_i - 1$ and $\overline{x_j} = x_j$ for all $j \neq i$. If we are in a cone and $\chi = 0$ we define $\overline{\chi} = \chi$. We define Ψ as $\Psi(L) = \frac{1}{k+1}\sum_{i=1}^{k-1} i \cdot \overline{x_i}$. Note that $\Psi(L) = 0$ if $\chi = 0$ or $\chi = e_i$ and otherwise we have $\Psi(L) > 0$.

Fact 1. *For a request to page $p \in L_i$, $i > 0$, we have $\Delta\Psi = -\frac{1}{k+1}\sum_{j=i}^{k-1} \overline{x_j}$.*

To prove that EQUITABLE2 is H_k-competitive, it suffices to show that for each request $cost + \Phi + \Psi \leq H_k \cdot cost_{OPT}$, as both Φ and Ψ are non-negative.

Lemma 5. *If no forgiveness is done then $\Delta cost + \Delta\Phi + \Delta\Psi \leq H_k \cdot \Delta cost_{OPT}$.*

Proof. We first analyze the case for a request $p \in L_i$, with $i > 0$. We have $\Delta cost + \Delta\Phi = 0$ by the definition of Φ and $\Delta cost_{OPT} = 0$. By Fact 1 $\Delta\Psi \leq 0$ and we are done. For requests to pages in L_0, both the algorithm and OPT incur a cost of one, and thus $\Delta cost = 1$ and $\Delta cost_{OPT} = 1$. It remains to show that $\Delta\Psi + \Delta\Phi \leq H_k - 1$. We analyze separately the case when we are in a cone. In this case, by definition $\Delta\Psi = 0$, and by Lemma 1 we obtain $\Delta\Phi = H_k - 1$. In the following we assume we are not in a cone upon the L_0 request. Let i be the largest index with $x_i \neq 0$. By the update rule, we get that $x'_{k-1} = x_{k-1} + 1$ and $x'_j = x_j$ for all $j \neq k-1$. For the chopped signature $\overline{\chi'}$ this implies $\overline{x'_j} = \overline{x_j}$ for all $j \neq i$ and $\overline{x'_i} = \overline{x_i} + 1$, because $i \neq k$ as L_k is always singleton. It follows $\Delta\Psi = i/(k+1)$. On the other hand we have by Theorem 1 that $\Delta\Phi \leq H_k - H_1 - i/(k+1)$.

Theorem 2. EQUITABLE2 *is H_k-competitive and uses $2 + \frac{\sqrt{5}-1}{2} \cdot k$ bookmarks.*

Proof. If the support size reaches the threshold $k + x$, i.e. x bookmarks, we apply upon a request from L_0 the forgiveness mechanism from [14]. Recall that we move the requested page artificially into L_1, and then we process it as if it was requested from L_1. We have $\Delta cost = 1$ and $\Delta cost_{OPT} = 0$. Like in [14], we need to prove that $1 + \Delta\Phi + \Delta\Psi \leq 0$. Denote by χ the current signature, and let $x = \sum_{i=1}^{k} x_i$ be the number of bookmarks used by the algorithm. We have that $\Delta\Phi = -cost_1(\chi)$. We get that $1 + \Delta\Phi$ is the probability that a page in L_1 is in the algorithm's cache, which by the selection process of ONLINEMIN is at most $k/|S| = k/(x+k)$. Using the result in Fact 1 and the fact that $\sum_{j=1}^{k-1} \overline{x_j} = x - 1$, we need to ensure that: $\frac{k}{x+k} - \frac{x-1}{k+1} \leq 0$. Solving this inequality, we get x is at most $\frac{\sqrt{5}-1}{2}k + 2$. Therefore, EQUITABLE2 needs only $\frac{\sqrt{5}-1}{2}k + 2 \approx 0.62k$ bookmarks. The cases where no forgiveness occurs are covered by Lemma 5.

In Theorem 3 and Theorem 4 we show that EQUITABLE2 cannot be both H_k-competitive and use $o(k)$ bookmarks, but that it can trade competitiveness for bookmarks. The proofs of these results are provided in the full version.

Theorem 3. *If* EQUITABLE2 *uses* $t \leq k/4$ *bookmarks, it is not* H_k*-competitive.*

Theorem 4. *There exist implementations of* EQUITABLE2 *that use* $k/(1+c)$ *bookmarks and are* $(H_k + c)$*-competitive, for* $k > 1$ *and* $c \geq 1$.

4 Partition

In this section we prove the conjecture in [14] that there exists a strongly competitive paging algorithm using $o(k)$ bookmarks. We propose a variation of the PARTITION algorithm [3], denoted PARTITION2, using $\Theta(k/\log k)$ bookmarks. We also give a simple lower bound showing that for any H_k-competitive randomized paging algorithm, the number of pages having non-zero probability of being in cache is at least $k + k/H_k$. This leads to a lower bound of k/H_k bookmarks for all algorithms which store all non-zero probability pages, i.e. representation of the approximated offset function, and have a deterministic forgiveness step. Note that this bound holds for all known H_k-competitive algorithms.

4.1 Partition

We give a brief description of the PARTITION algorithm in [3]. A crucial difference compared to EQUITABLE is that while the distribution of the cache configurations depends only on the current offset function for EQUITABLE, PARTITION uses a more detailed representation of the offset function, which we denote in the following *set-partition*. It partitions the whole pageset into a sequence of disjoint sets $S_\alpha, S_{\alpha+1}, \ldots, S_{\beta-1}, S_\beta$ and each set S_i with $i < \beta$ has a *label* k_i. Initially $\beta = \alpha + 1$, S_β contains the first k pairwise distinct pages, the remaining pages are in S_α, and $k_\alpha = 0$. Throughout the computation S_β contains all revealed pages and S_α all Opt-miss pages. The set-partition is updated as follows:

- if $p \in S_\alpha$: $S_\alpha = S_\alpha \setminus \{p\}$, $S_{\beta+1} = \{p\}$, $k_\beta = k - 1$, and $\beta = \beta + 1$.
- if $p \in S_i$, with $\alpha < i < \beta$: $S_i = S_i \setminus \{p\}$, $S_\beta = S_\beta \cup \{p\}$, and $k_j = k_j - 1$, $(i \leq j < \beta)$. Additionally, if there are labels which become zero, let j be the largest index such that $k_j = 0$; we set $S_j = S_\alpha \cup \cdots \cup S_j$ and $\alpha = j$.
- if $p \in S_\beta$: nothing changes

In [3] it was shown that the following invariants on the labels hold: $k_\alpha = 0$ and $k_i > 0$ for all $i > 0$; $k_{\beta-1} = k - |S_\beta|$. Furthermore, it holds at all times that $k_i = (k_{i-1} + |S_i|) - 1$. The probability distribution of the cache content can be described as the outcome of the following selection process on the set-partition:

- $\mathcal{C}_\alpha = \emptyset$
- For $\alpha < i < \beta$ choose p uniformly at random from $\mathcal{C}_{i-1} \cup S_i$ and then set $\mathcal{C}_i = (\mathcal{C}_{i-1} \cup S_i) \setminus \{p\}$
- $\mathcal{C}_\beta = \mathcal{C}_{\beta-1} \cup S_\beta$.

Note that, whereas for the selection process of ONLINEMIN it holds that $|C_i| = i$ $(0 \leq i \leq k)$, for PARTITION we have that $|\mathcal{C}_j| = k_j$ $(\alpha \leq j \leq \beta - 1)$.

Lemma 6 ([3, Lemma 3]). *If p is requested from S_i, where $\alpha < i < \beta$, the probability that p is not in the cache of* PARTITION *is at most* $\sum_{i \le j < \beta} \frac{1}{k_j+1}$.

Apart from obeying the probability distribution, PARTITION must satisfy two constraints: it must not evict pages upon a cache hit and it must not evict more than one page upon a cache miss. For any set $\mathcal{C}_i$, the membership of a page to $\mathcal{C}_i$ is encoded with a marking system on pages as follows. If a page is in set S_i, where $\alpha < i < \beta$, it has either no mark or a series of marks $i, i+1, \ldots, j-1, j$. If p has no mark then $p \notin \mathcal{C}_i$ and otherwise it is in the selection sets $\mathcal{C}_i, \mathcal{C}_{i+1}, \ldots, \mathcal{C}_{j-1}, \mathcal{C}_j$. The cache of PARTITION is at all times $\mathcal{C}_\beta$, with $|\mathcal{C}_\beta| = k$. For a page $p \in S_i$ it suffices to store the value m_p of the highest mark or $i-1$ if p has no mark.

Initially there are only the two sets S_α and S_β and thus no marks. If the requested page $p \in S_\beta$ nothing changes. If $p \in S_\alpha$ first the set-partition is updated, where β is increased by 1 and we have to determine $\mathcal{C}_{\beta-1}$. A page q is chosen uniformly at random from the k elements $\mathcal{C}_{\beta-2} \cup S_{\beta-1}$ (the cache content before the request), and this element is the only one not receiving a $\beta - 1$ mark. The page q is replaced in the cache by the requested page p. We now turn to the case $p \in S_i$, where $\alpha < i < \beta$. If p is in cache then $m_p = \beta - 1$ and we are done. Otherwise let $j \le \beta - 1$ be the lowest index such that $p \notin \mathcal{C}_j$. We choose uniformly at random a page $q \in \mathcal{C}_j$ and set $m_p = m_q$ and $m_q = j - 1$, i.e. p steals the marks of q. We repeat this until $m_p = \beta - 1$. The page which loses its $\beta - 1$ mark is replaced in cache by p. Afterwards the set-partition is updated.

4.2 Partition2

PARTITION2 is a variant of PARTITION which uses (deterministic) forgiveness to reduce the space usage from arbitrarily high bookmarks to $O(k/\log k)$ bookmarks. A lower bound is provided which shows that this bound is asymptotically optimal for algorithms using deterministic forgiveness. Unlike previous works, when a forgiveness step must be applied, we distinguish between two cases and apply two distinct forgiveness rules accordingly. The first of them is the same one used by EQUITABLE2 and covers only a single request, and the second one is a *forgiveness phase* which spans consecutive requests. To apply the forgiveness step of EQUITABLE2, we provide an embedding of the set-partition into the layer representation of the offset function. Based on this embedding, we give a simple potential function which depends only on the signature of the offset function.

Layer Embedding. We provide an embedding of the set-partition into the layer representation of the offset functions, as used by EQUITABLE. The layers become ordered sets and contain pages and set identifiers, the latter of which we visualize by $\star$. The initialization does not change and no set identifiers are present. The update rule changes mainly for the case $p \in L_0$:

$$L_{k-1} = (L_{k-1}, L_k, \star),\ L_k = \{p\}.$$

In the case $p \in L_i$, upon the merge operation $L_{i-1} \cup L_i \setminus \{p\}$, we remove p from L_i and concatenate L_{i-1} with L_i without removing any set identifier. Upon

merging L_1 into L_0 we delete all set identifiers from the resulting layer L_0. The following fact follows inductively.

Fact 2. *For L_i, with $i > 0$ and $|L_i| = 1 + x_i$, it holds that L_i contains exactly x_i set identifiers. Moreover, if $x_i > 0$ then the last element in L_i is a set identifier.*

We describe how to obtain the sets of the set-representation. Let j be maximal such that $x_j > 1$. We have $S_\beta = L_{j+1} \cup \cdots \cup L_k$ and $S_\alpha = L_0$. A set $S_{\alpha+j}$, where $1 < j < \beta - \alpha$ consists of all pages between the $(j-1)$-th and the j-th set identifier; for $j = 1$, $S_{\alpha+1}$ consists of all support pages until the first set identifier. We say that each set $S_{\alpha+j}$, $0 < j < \beta - \alpha$, is *represented* by the j'th set identifier. As long as no pages are moved into S_α, the correspondence between the layer representation and the set-partition follows immediately from the update rules. Otherwise, by Lemma 7 and noticing that each L_i with $x_i > 0$ ends in a set delimiter, we obtain that p is in L_1 and moreover the pages moved to S_α correspond to $L_1 \setminus \{p\}$.

Lemma 7. *Let $S_a, S_{a+1}, \ldots, S_b$ be the sets whose identifiers are in layer L_i, $i \geq 0$. We have $k_b = i$, $k_{a+j} \geq i$ for $0 \leq j < b - a$.*

Proof. Due to space limitations, the proof is provided in the full version.

Lemma 8. *If p is requested from L_i, where $i > 0$, the probability that p is not in the cache of* Partition *is at most $\sum_{j \geq i} \frac{x_j}{j+1}$.*

Proof. If $p \in S_\beta$, then it is in a revealed layer L_i and thus $x_j = 0$ for all $j \geq i$ and the result holds. Let S_{i^*} be the set with $p \in S_{i^*}$, $\alpha < i^* < \beta$. Then by Lemma 6 we have the probability bounded by $\sum_{i^* \leq j^* < \beta} \frac{1}{k_{j^*}+1}$. All sets S^*_j, where $i^* \leq j^* < \beta$ have their identifier in some layer L_j with $j \geq i$ and using Lemma 7 we obtain $\frac{1}{k_{j^*}+1} \leq \frac{1}{j+1}$. Since each layer L_j contains exactly x_j identifiers the statement follows.

Forgiveness. Forgiveness is applied when the support size reaches a threshold of $k + 3t$ (we define t later) and a page in L_0 is requested. Depending on the support we have two kinds of forgiveness: *regular forgiveness* and an *extreme forgiveness mode*. The regular forgiveness is applied if $|L_1| + \cdots + |L_t| > 2t$ and is an adaptation of the forgiveness step of Equitable2. If a page p is requested from L_0 (equivalent to S_α), we first identify a page q satisfying that $q \in S_{\alpha+1} \cap L_1$. Note that there always exists such a page, since $k_{\alpha+1} \geq 1$ and $|S_1| = k_1 + 1$ and at least one of them is in L_1. We move q to L_0 and replace it, together with its marks, by p. Then we perform the set-partition and mark update where p is requested from $S_{\alpha+1}$. We stress that in terms of the layer representation (used by e.g. Equitable), we replace the requested page with an existing page in L_1, and replacing $q \in L_1$ by p and requesting p leads to the same offset function when the forgiveness step in [14] is applied. This has a cost of 1 for Partition and a cost of 0 for OPT. The support size decreases by $|L_1| - 1 \geq 0$.

The extreme forgiveness mode is applied if $|L_1| + \cdots + |L_t| \leq 2t$. We simply apply regular forgiveness for any page request in L_0 starting with the current one. This extreme forgiveness mode ends when reaching a cone.

Competitive Ratio. We use PARTITION with the forgiveness rule for $t = \lceil \frac{k}{\ln k} \rceil$ from the previous paragraph if $k > 10$ and denote the resulting algorithm PARTITION2. For $k \leq 10$ we use the regular forgiveness if the support size reaches $2k$.

Theorem 5. PARTITION2 *uses* $\Theta(\frac{k}{\log k})$ *bookmarks and is* H_k*-competitive.*

Proof. The space bound follows from the fact that the support size never exceeds $k + 3t$ for $k > 10$, where $t = \lceil \frac{k}{\ln k} \rceil$. It remains to show that PARTITION2 is still H_k-competitive. We use the following potential function on the layer embedding:

$$\Phi = \sum_{j=1}^{k-1} x_j \cdot (H_{j+1} - 1) \; .$$

We denote by *cost* the cost of PARTITION2 and by OPT the cost of the optimal offline algorithm. We have to show that $cost \leq H_k \cdot OPT$ holds after each request. In all cases except the extreme forgiveness we show that the following holds before and after each request: $\Phi + cost \leq H_k \cdot OPT$. This leads to $cost \leq H_k \cdot OPT$ since $\Phi \geq 0$. When applying the extreme forgiveness we assume that the potential inequality holds before the phase and show that it holds at the end of the phase, but not necessarily during the phase. For requests during the phase we argue directly that it always holds $cost \leq H_k \cdot OPT$.

Let p be the requested page. If $p \in L_0$ without forgiveness, $\Delta OPT = 1$ and x_{k-1} increases by 1, which implies that $\Delta\Phi + \Delta cost = H_k - 1 + 1 = 1 \cdot H_k$. If p is from some layer L_i, where $0 < i \leq k$, we use the bound on the cache miss probability from Lemma 8

$$\Delta\Phi + \Delta cost \leq -\sum_{j \geq i} \frac{x_j}{j+1} + \sum_{j \geq i} \frac{x_j}{j+1} \leq 0 \leq H_k \cdot \Delta OPT.$$

Now we analyze the cases where forgiveness occurs for $k > 10$. Assume that $|L_1| + \cdots + |L_t| \geq 2t+1$ which implies that $x_1 + \cdots + x_t \geq t+1$. We perform just one forgiveness step, yielding $\Delta cost = 1$ and $\Delta OPT = 0$. We show $\Delta\Phi \leq -1$:

$$\Delta\Phi = -\sum_{j=1}^{k-1} \frac{x_j}{j+1} \leq -\sum_{j=1}^{t} \frac{x_j}{t+1} = -\frac{t+1}{t+1} = -1.$$

Now assume that $x_{t+1} + \cdots + x_{k-1} \geq 2t$. Before we start the extreme forgiveness mode, we have that $\Phi \geq \sum_{j=t+1}^{k-1} x_j (H_{j+1} - 1) \geq 2t(H_{t+2} - 1)$. For $t = \lceil \frac{k}{\ln k} \rceil$ and $H_x \geq \ln x$ we obtain: $\Phi \geq \frac{2k}{\ln k}(\ln k - \ln\ln k - 1) \geq k$, if $k > 10$. Right before the phase starts we have $cost + \Phi \leq H_k \cdot OPT$, where $\Phi \geq k$ which is equivalent to $cost \leq H_k \cdot OPT - k$. Reaching the next cone implies at most $k-1$ unrevealed requests and thus the cost during this phase is bounded by $k-1$. This implies that $cost \leq H_k \cdot OPT$ holds. Since in a cone $\Phi = 0$ we also have at the end of the phase the invariant $cost + \Phi \leq H_k \cdot OPT$.

For the case $k \leq 10$ the analysis of the extreme forgiveness does not hold. In this case we use only the regular forgiveness step if we have k bookmarks. Using $x_1 + \cdots + x_{k-1} = k$ the same argument as before leads to $\Delta\Phi \leq -1$.

Lemma 9. *For any H_k-competitive algorithm A there exists an input such that the maximal number of pages with non-zero probability of being in A's cache is at lest $k + k/H_k$.*

Proof. Due to space limitations, the proof is provided in the full version.

References

1. Moruz, G., Negoescu, A.: Improved space bounds for strongly competitive randomized paging algorithms. Technical report, Goethe University Frankfurt am Main (2013)
2. Bein, W.W., Larmore, L.L., Noga, J.: Equitable revisited. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 419–426. Springer, Heidelberg (2007)
3. McGeoch, L.A., Sleator, D.D.: A strongly competitive randomized paging algorithm. Algorithmica 6(6), 816–825 (1991)
4. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 77–119 (1988)
5. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)
6. Belady, L.A.: A study of replacement algorithms for virtual-storage computer. IBM Systems Journal 5(2), 78–101 (1966)
7. Albers, S.: Online algorithms: a survey. Mathematical Programming 97(1-2), 3–26 (2003)
8. Borodin, A., El-Yaniv, R.: Online computation and competitive anlysis. Cambridge University Press (1998)
9. Young, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica 11(6), 525–541 (1994)
10. Moruz, G., Negoescu, A.: Outperforming LRU via competitive analysis on parametrized inputs for paging. In: Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1669–1680 (2012)
11. Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young, N.E.: Competitive paging algorithms. Journal of Algorithms 12(4), 685–699 (1991)
12. Chrobak, M., Koutsoupias, E., Noga, J.: More on randomized on-line algorithms for caching. Theoretical Computer Science 290(3), 1997–2008 (2003)
13. Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science 234(1-2), 203–218 (2000)
14. Bein, W.W., Larmore, L.L., Noga, J., Reischuk, R.: Knowledge state algorithms. Algorithmica 60(3), 653–678 (2011)
15. Brodal, G.S., Moruz, G., Negoescu, A.: Onmin: A fast strongly competitive randomized paging algorithm. In: Theory of Computing Systems (2012)
16. Bein, W.W., Fleischer, R., Larmore, L.L.: Limited bookmark randomized online algorithms for the paging problem. Information Processing Letters 76(4-6), 155–162 (2000)
17. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. In: Proc. 35th Symposium on Foundations of Computer Science, pp. 394–400 (1994)