GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

Institut für Informatik

Fachbereich Informatik und Mathematik

Improved Space Bounds for Strongly Competitive Randomized Paging Algorithms

Gabriel Moruz, Andrei Negoescu

2013

Frankfurter Informatik-Berichte

Institut für Informatik • Robert-Mayerst.11-15 • D-60054 Frankfurt am Main, Germany

ISSN 1868-8330

Improved Space Bounds for Strongly Competitive Randomized Paging Algorithms *

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

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

ABSTRACT

Paging is one of the prominent problems 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. Bein et al. [4] conjectured that there exist strongly competitive randomized paging algorithms, using $o(k)$ bookmarks, i.e. pages not in cache that the algorithm keeps track of. Also in [4] the first algorithm using $O(k)$ bookmarks ($2k$ more precisely), EQUITABLE2, was introduced, proving in the affirmative a conjecture in [7].

We prove tighter bounds for EQUITABLE2, 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. Nonetheless, we show that it can trade competitiveness for space. More precisely, if its competitive ratio is allowed to be $(H_k + t)$, then it requires $k/(1 + t)$ 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 by McGeoch and Sleator [13]. While classical PARTITION is unbounded in its space requirements, PARTITION2 uses $\Theta(k/ \log k)$ bookmarks. Furthermore, we show that this result is asymptotically tight when the forgiveness steps are deterministic.

General Terms

Online algorithms, Paging, Randomized algorithms

1. INTRODUCTION

*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).

Paging is a prominent and well studied problem in the field of online algorithms. We are provided with 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 as follows. If the currently requested page is in the cache, we say that a *cache hit* occurs and the algorithm proceeds to the next page. Otherwise, a *cache miss* occurs and the requested page must be brought into cache. Additionally, if the cache was full, a page in cache must be evicted to accommodate the new one. The cost of the algorithm is given by the number of cache misses incurred.

Online algorithms in general and paging algorithms in particular are typically analyzed in the framework of *competitive analysis*[11, 15]. An algorithm A is said to have a *competitive ratio* of c (or c-competitive) if its cost satisfies for any input $cost(A) \leq c \cdot cost(OPT) + O(1)$, where $cost(OPT)$ 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, $cost(A)$ is the expected cost of A. An algorithm achieving an optimal competitive ratio is said to be *strongly competitive*. For paging, an optimal offline algorithm was proposed decades ago; upon a cache miss, it evicts the page in cache whose request occurs the furthest in the future [6]. In the remainder of the paper, we refer to this algorithm as OPT. For comprehensive surveys on online algorithms in general and paging algorithms in particular, we refer the interested reader to [2, 7].

Competitive ratio has been often criticized for its too 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 [16]. 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 [14] 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.

Related work. Randomized competitive paging algorithms have been extensively studied over the past two decades. In [10] a lower bound of H_k on the competitive ratio of randomized paging algorithms has been given[1]. Also in [10], a simple $(2H_k - 1)$-competitive algorithm, denoted MARK, has been proposed. In [9] it was shown that no randomized Marking algorithm can achieve a competitive ratio better than $(2 - \varepsilon)H_k$ for any $\varepsilon > 0$, meaning that MARK is essentially optimal.

The first strongly competitive paging algorithm, denoted PARTITION, was proposed in [13]. While it achieves the optimal competitive ratio of H_k, its time and space requirements are in the worst case proportional to the input size independently of the cache size, which makes them hopelessly high. More recent research has focused on improving these bounds, especially the space requirements. In the literature, a *bookmark* refers to a page outside the cache that the algorithm keeps track of; in particular, an algorithm is denoted *trackless* if it stores no bookmarks at all. In [1], an H_k-competitive algorithm, denoted EQUITABLE, was proposed, using only $O(k^2 \log k)$ bookmarks. Using a better version of EQUITABLE, denoted EQUITABLE2, this bound was further improved in [5] to $2k$ bookmarks. This solved the open question in [7] that there exist H_k-competitive paging algorithms using $O(k)$ space. In [8] we proposed an algorithm, denoted 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 requirements.

A distinct line of research for randomized paging algorithms consists of considering fixed small cache sizes ($k = 2$ and $k = 3$ to our best knowledge) to obtain tighter bounds than for general k. In [3], for $k = 2$, a $\frac{3}{2}$-competitive algorithm using only one bookmark was proposed. Also in [3], for trackless randomized algorithms a lower bound on the competitive ratio of $\frac{37}{24} \approx 1.5416$ was given. Still for $k = 2$, a trackless algorithm having an upper bound of ≈ 1.6514 was introduced in [9]. Finally, in [5], strongly competitive randomized paging algorithms were proposed for $k = 2$ and $k = 3$, using 1 and 2 bookmarks respectively.

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 $2k$ [5]. In [5] 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 achieve a competitive ratio of H_k using $o(k)$ 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 an algorithm PARTITION2 which is a modification of the PARTITION algorithm. PARTITION2 improves the bookmarks requirements 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 [13] and [12]. Since offset functions are the key ingredient in the design and analysis of optimal competitive algorithms for paging, this may be of independent interest. Finally, we show that k/H_k is a lower bound on the number of bookmarks for any strongly competitive algorithm which uses a deterministic approximation of the offset function. This makes PARTITION2 asymptotically optimal within this class.

2. PRELIMINARIES

In this section we give a brief introduction concerning offset functions for paging, the EQUITABLE algorithms, and forgiveness as a space bounding technique.

Offset Functions. In competitive analysis the cost approximation of the optimal offline algorithm plays an important role. 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 [12] 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.

Layer Representation. In [12] it was shown for the paging problem that the actual offset function can be represented as a partitioning of the pageset in $k + 1$ disjoint sets $L = (L_0|L_1| \dots |L_k)$, 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 $L_1, \dots, L_k$, one page per layer, and L_0 contains the remaining pages. Upon processing page p, let $L^p = (L^p_0|L^p_1 \dots |L^p_k)$ be the partitioning after processing p; L^p is obtained from L as follows[2]:

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

This layer representation can be used to keep track of all valid configurations. More specifically, a set C of k pages is valid iff $|C \cap L_i| \le i$ holds for all $0 \le i \le k$ [12].

For a given L, denote by *support* $S(L) = L_1 \cup \dots \cup L_k$. Also, we call a layer containing a single page a *singleton*. Let r be the smallest index such that $L_r, \dots, L_k$ are singletons. The pages in $L_r, \dots, L_k$ are denoted *revealed*, the pages in support which are not revealed are *unrevealed*, and the pages

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

[2] We use the layer representation introduced in [8], which is equivalent to the ones in [1, 12].

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 OPTs cache. If L consists only of revealed pages it is denoted a *cone* and we know the content of OPT's cache. Although the layer representation is not unique it has an unique *signature*. The signature $\chi(L)$ is defined 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 [8] we defined a priority based selection process on L which is guaranteed to construct a valid configuration. Assume that pages in the support have pairwise distinct priorities. Our selection process builds a hierarchy of sets $C_0, \ldots, C_k$ as follows:

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

Note that, by definition, when cunstructing 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. In particular, if the priorities are the negated timestamps of the next requests (in the future) for the support pages, then C_k is identical to OPT's cache.

Equitable and OnlineMin. The cache content of the EQUITABLE algorithms [1, 5] is defined by a probability distribution over the set of all valid configurations. The cache configuration depends solely on the current offset function. This distribution is achieved by the ONLINEMIN algorithm 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. The cache content of ONLINEMIN is at all times the outcome C_k of the selection process. Since we use in the remainder of the paper only this selection process, we do not describe the selection process for EQUITABLE. Nonetheless, the resulting probability distribution on cache configurations is the same as for EQUITABLE, and in the rest of the paper we refer to this distribution and the associated algorithm as EQUITABLE.

Forgiveness. Note that the support size increases only when pages in L_0 are requested, and may decrease only when pages in L_1 are requested. As the amount 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 which is bounded in size.

3. BETTER BOUNDS FOR EQUITABLE2

There are two EQUITABLE algorithms, EQUITABLE [1] and EQUITABLE2 [5][3]. For a fixed offset function (both use an approximation of the actual offset function), they have the same distribution as previously introduced. The difference between them is given by the forgiveness mechanism used. In this section we focus on the EQUITABLE2 algorithm using the forgiveness mechanism described in [5] which works as follows. Whenever the support size reaches a threshold value and an Opt-miss page is requested, the requested page is artificially inserted in L_1 and processed as a L_1 page. This way, all pages in L_1 move to L_0 and the support size never exceeds the designated threshold. The threshold in [5] is set to $3k$, i.e. the algorithm uses $2k$ bookmarks. We give a tighter analysis and show that using the same forgiveness the algorithm uses less than k bookmarks. We also give lower bounds showing that it can not achieve $o(k)$ bookmarks while preserving its H_k competitive ratio. Finally, we show that it can trade competitiveness for space. More specifically, if the algorithm is allowed to be $(H_k + t)$-competitive, it can be implemented using $k/(1 + t)$ bookmarks, where t is an arbitrary non-negative value, e.g a function in k.

To accommodate the selection process for ONLINEMIN previously introduced, 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.

Potential. In [1] 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 [1].

Initially, we are in a cone and thus $\Phi = 0$. 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 $\Delta\Phi \leq H_k - 1$ [1, 5]. Since upon revealed requests both algorithms never fault and the offset function does not change we have:

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

If L is a cone, it is easy to verify that a request in L_0 leads to $\Delta\Phi = H_k - 1$. If the support size is strictly larger than 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*. These savings can be tracked by a second potential function and are used to pay for the forgiveness step when the support size becomes large enough. While Φ is very comfortable 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

[3] In [5] EQUITABLE2 is denoted K_EQUITABLE. In this paper we use its original name.

upon a request in L_0.

3.1 Approximation of Φ

The key ingredient to our analysis is to get a bound as tight as possible for $\Delta\Phi$ on requests in L_0. That is because 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 get that Φ is 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 a simple case, where all layers are singletons except some layer L_i. The potential Φ for this particular case is easy to calculate and is given in Lemma 1.

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

PROOF. Let p be a page in L_1. Since there is no competition in the selection process for C_j, where $j \neq i$, we have that $p \in C_{i-1}$ independent of its priority and $p \in C_k$ iff $p \in C_i$. For the selection in C_i we have i slots and $i + n$ candidates. All these candidates have the same probability to be selected in C_i, since all layers $L_1, \ldots, L_i$ are singleton and thus no competition steps happened; note that this argument holds only if $x_1 = \cdots = x_{i-1} = 0$. This means that the probability of a cache miss is $\frac{n}{i+n}$. Updating the layers leads to $\chi = n \cdot e_{i-1}$. Repeating the argument we obtain:

$$\Phi = \frac{n}{i+n} + \frac{n}{i-1+n} + \cdots + \frac{n}{1+n} = n(H_{i+n} - H_n) ,$$

and the claim holds. $\square$

For some arbitrary values i, n, and κ, where $0 < i < \kappa \leq k$ consider the signatures $\chi = n \cdot e_i$ and $\chi' = n \cdot e_i + e_{\kappa-1}$; let L and L' be their corresponding layer representations. We define the difference in the cost for a request in L_1:

$$f(i, n, \kappa) = cost_1(\chi') - cost_1(\chi) .$$

In the special case $\kappa = k$ it represents $\Delta cost_1$ upon a request in L_0. The value for $f(i, n, \kappa)$ can be computed exactly and is given in Lemma 2.

LEMMA 2. *$f(i, n, \kappa) = \frac{1}{n+\kappa} \prod_{j=i}^{\kappa-1} \frac{j}{n+j}$.*

PROOF. If $i = \kappa - 1$, we have $cost_1(\chi) = n/(\kappa - 1 + n)$ and $cost_1(\chi') = (n+1)/(\kappa+n)$, and the result immediately follows. For the remainder of the proof we assume $i < \kappa - 1$. Consider a priority assignment for χ' and a request to some page p in L'_1. By the selection process for ONLINEMIN, the value of $f(i, n, \kappa)$ is given by the probability that $p \in C'_i$ and $p \notin C'_{\kappa-1}$, since if $p \in C'_{\kappa-1}$ then $p \in C'_k$ and the probability that $p \in C'_i$ is the probability of a cache hit in χ, i.e. if $p \in C_i$ then $p \in C_k$. The scenario $p \in C'_i$ and $p \notin C'_{k-1}$ happens when p has rank i (i.e. has the i'th highest priority) among the $n + i$ pages in $L'_1 \cup \cdots \cup L'_i$ and all pages in $L'_{i+1}, \ldots, L'_{\kappa-1}$ have greater priorities than p. There are $(n + i - 1)!$ possibilities that p has rank i among the $n + i$ pages in $L'_1 \cup \cdots \cup L'_i$. For each of these, there are $i \cdot (i + 1) \cdot \ldots \cdot (\kappa - 1)$ possibilities that all the $\kappa - i$ pages in $L'_{i+1}, \ldots, L'_{\kappa-1}$ have priorities higher than p. We get that:

$$f(i, n, \kappa) = \frac{(n + i - 1)! \prod_{j=i}^{\kappa-1} j}{(n + \kappa)!} = \frac{1}{n + \kappa} \prod_{j=i}^{\kappa-1} \frac{j}{n + j} ,$$

which concludes the proof. $\square$

We are now ready to move to a more general case. In Lemma 3 we show that $f(i, n, \kappa)$ is an upper bound on $\Delta cost_1$ for a whole class of signatures.

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 $\chi' = \chi + e_{\kappa-1}$, $i < \kappa \leq k$. For $n = x_1 + \cdots + x_i$, we have*

$$cost_1(\chi') - cost_1(\chi) \leq f(i, n, \kappa).$$

PROOF. Let $g(i, n, \kappa) = cost_1(\chi') - cost_1(\chi)$. Similar to the proof of Lemma 2, the value of $g(i, n, \kappa)$ is given by the probability that a request $p \in L'_1$ is in C'_i and not in $C'_{\kappa-1}$. Intuitively, the proof is based on the observation that the fact that p must have exactly rank i among the $n + i$ pages in $L'_1 \cup \cdots \cup L'_i$ is necessary but not sufficient, whereas in the proof of Lemma 2 this fact was necessary and sufficient.

Assume $i < \kappa - 1$. By the definition of the selection process, if $p \in C'_i$ then the priority of p is compared against the priorities of all pages in $L'_1 \cup \cdots \cup L'_i$, because $p \in L'_1$; note that this doesn't necessarily hold if $p \in L'_j$, with $j > 1$. This immediately means that p must necessarily have rank i in $L'_1 \cup \cdots \cup L'_i$. The number of permutations where p has rank i among the $n + i$ pages is $(n + i - 1)!$. However, it may not hold that for all of them we have $p \in C'_i$. Let $j_1, \ldots, j_t$ be indices smaller than i such that $x'_{j_l} \neq 0$ for all j_l. To have $p \in C'_i$, the priority of p must also be among the largest j_1 in $L'_1 \cup \cdots \cup L'_{j_1}$, among the largest j_2 in $C'_{j_1} \cup L'_{j_1+1} \cdots \cup L'_{j_2}$ and so on; in short, p must overcome t selection processes, instead of one as in the proof of Lemma 2. The set P of permutations on the $(n + i)$ pages in the first i layers where p has rank i and $p \in C'_i$ has size at most $(n + i - 1)!$. Recall that all $\kappa - i$ elements in $L'_{i+1} \cup \cdots \cup L'_{\kappa-1}$ must have higher priorities than p. For each permutation in P there are $i \cdot (i + 1) \cdot \ldots \cdot (\kappa - i)$ possibilities to do so. In total, we get that:

$$g(i, n, \kappa) = \frac{|P| \prod_{j=i}^{\kappa-1} j}{(n + \kappa)!} \leq \frac{(n + i - 1)! \prod_{j=i}^{\kappa-1} j}{(n + \kappa)!} = f(i, n, \kappa) .$$

If $i = \kappa - 1$, we have that $g(i, n, \kappa)$ is the probability that $p \in C_i$ and $p \notin C'_i$. Let q be an arbitrary page in L'_i. Then $g(i, n, \kappa)$ is bounded by the probability that q has rank $(i+1)$ in $L'_1 \cup \cdots \cup L'_i$ and rank i in $L'_1 \cup \cdots \cup L'_i \setminus \{q\}$. Using a similar reasoning, there are $(n+i-1)! \cdot i$ possibilities for this scenario to occur, which concludes the proof. $\square$

Lemma 4 provides a useful identity for approximating $\Delta\Phi$ for a request in L_0.

LEMMA 4. *For any i and κ with $i < \kappa$, it holds that* $\sum_{j=1}^{i} f(i-j+1, 1, \kappa-j+1) = H_\kappa - H_{\kappa-i} - \frac{i}{\kappa+1}$.

PROOF. We first note that:

$$f(i,1,\kappa) = \frac{1}{\kappa+1} \cdot \frac{i}{i+1} \cdot \frac{i+1}{i+2} \cdots \frac{\kappa-1}{\kappa} = \frac{i}{\kappa(\kappa+1)} .$$

Denoting by $S(i,\kappa) = \sum_{j=1}^{i} f(i-j+1, 1, \kappa-j+1)$ and using that $i/(\kappa(\kappa+1)) = i/\kappa - i/(\kappa+1)$, we have:

$$\begin{aligned} S(i,k) &= f(i,1,\kappa) + \cdots + f(1,1,\kappa-(i-1)) \\ &= \frac{i}{\kappa} - \frac{i}{\kappa+1} + \cdots + \frac{1}{\kappa-(i-1)} - \frac{1}{\kappa-(i-2)} \\ &= \frac{1}{\kappa} + \cdots + \frac{1}{\kappa-(i-2)} + \frac{1}{\kappa-(i-1)} - \frac{i}{\kappa+1} , \end{aligned}$$

which concludes the proof. $\square$

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:

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

Since $cost_1(\chi'^j) - cost_1(\chi^j)$ is non-negative, the left inequation 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 $\chi'^j = \chi^j + e_{\kappa-1}$ with $\kappa = k-j+1$. Denoting $n^j = x^j_1 + \cdots + x^j_{i-j+1}$, we get:

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

The inequations stem from the fact that f is decreasing in n and $n^j > 0$ for all $j \leq i$, and the equality is the result in Lemma 4. $\square$

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 $\chi = (x_1|\ldots|x_k)$, let i be the largest index such that $x_i > 0$. The chopped signature corresponding to χ is $\overline{\chi} = (\overline{x_1}|\ldots|\overline{x_k})$, 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 $\overline{\chi} = \chi$. Ψ is defined 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 $\Psi(L) > 0$.

LEMMA 5. *For a request to page $p \in L_i$, $i > 0$, it holds:*

$$\Delta\Psi = -\frac{1}{k+1} \sum_{j=i}^{k-1} \overline{x_j}.$$

PROOF. Before the request each $\overline{x_j}$ with $j \geq i$ contributes with $\frac{j \cdot \overline{x_j}}{k+1}$ and after the request with $\frac{(j-1) \cdot \overline{x_j}}{k+1}$ leading to a difference of $-\frac{\overline{x_j}}{k+1}$. $\square$

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. We do so by proving for each step the inequation is preserved by considering the differences in costs and potentials.

LEMMA 6. *If no forgiveness is applied it holds,*

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

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 Lemma 5 $\Delta\Psi \leq 0$ 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 $\Delta\Phi \leq H_k - H_1 - i/(k+1)$. $\square$

THEOREM 2. EQUITABLE2 *is H_k-competitive and requires* $2 + \frac{\sqrt{5}-1}{2} \cdot k$ *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 [5]. Recall that we move the requested page artificially into L_1. This step does not increase OPT's overall cost. Then we process it as if it was requested from L_1. We have $\Delta cost = 1$ and $\Delta cost_{OPT} = 0$. Like in [5], 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 Lemma 5 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 inequation, we get $x \geq (1-k+\sqrt{5k^2+6k+1})/2$, which is at most $\frac{\sqrt{5}-1}{2}k + c$ for $c \geq 2$. Therefore, EQUITABLE2 needs only $\frac{\sqrt{5}-1}{2}k+c \approx 0.62k$ bookmarks. The cases where no forgiveness occurs are covered by Lemma 6. $\square$

Lower bound. We now show in Theorem 3 that EQUITABLE2 can not achieve $o(k)$ bookmarks and be H_k-competitive.

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

PROOF. For easiness of exposition we assume that k is divisible by 4. It suffices to build an input sequence which starts and ends in a cone where the cost of EQUITABLE2 using t bookmarks exceeds $H_k \cdot OPT$ for arbitrary large k. This sequence consists of three phases.

In the first phase we bring t additional pages into layer L_i (no forgiveness occurs), where the index $i > 0$ is determined later. To do so, we request a page in L_0 leading to $\chi = e_{k-1}$ followed by $k - i - 1$ requests from L_{i+1}. The resulting signature is $e_i = (0|\dots|x_i = 1|\dots|0)$. We repeat this step $t-1$ more times and obtain the signature $\chi_i = t \cdot e_i$ which by Lemma 1 has the potential $\Phi_i = t(H_{t+i} - H_t)$. By Theorem 1, each request in L_0 increases Φ by at least $H_{k-i} - 1$, leading to a total amount of potential increases $\Phi_+ = t * H_{k-i} - t$. Since Φ decreases upon lazy requests the total cost of EQUITABLE during this phase is

$$t + \Phi_+ - \Phi_i = t \cdot (H_{k-i} - H_{t+i} + H_t).$$

The second phase starts with a request from L_0 which forces EQUITABLE2 to apply forgiveness. This leads to $\chi = t \cdot e_i + e_{k-1}$ whereas the signature used by EQUITABLE2 is $\chi_{Eq} = t \cdot e_{i-1}$. This means that page $q \in L_1$ in the (original) layer representation is for sure not in cache. We request q. We can repeat the last request type $i - 1$ additional times which leads to a total cost in the second phase of i whereas OPT pays 1. In the third phase we bring the (original) offset function to a cone, and repeat revealed requests (if needed) such that EQUITABLE also reaches a cone and we can repeat our attack. The third phase incurs no cost for OPT. Choosing $i = (k - t)/2$ we need to show:

$$\frac{tH_t + 0.5(k-t)}{t+1} > H_k.$$

Setting $t = k/4$, we get:

$$1.5 + \cdot H_{k/4} - H_k - \frac{H_k}{k/4} > 0.$$

For the value $k = 200$ the left side is about 0.0036. The term $H_{k/4} - H_k$ is increasing in k. To see this let $k = k + 4$. We obtain a difference of $\frac{4}{k} - \frac{1}{k+1} - \frac{1}{k+2} - \frac{1}{k+3} - \frac{1}{k+4} > 0$. On the other hand $\frac{H_k}{k/4}$ is decreasing in k. We conclude that the inequation is true for $k \geq 200$. $\square$

Trading competitiveness for space. We now show that EQUITABLE2 can achieve $o(k)$ bookmarks at the expense of competitiveness. This result is given in Theorem 4.

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

PROOF. Again, we consider two functions Φ and Ψ, both initially set to zero, and for each request we prove that:

$$\Delta cost + \Delta\Phi + \Delta\Psi \leq (H_k + c)\Delta cost_{OPT}\ .$$

As before, Φ is the cost of a lazy sequence of requests in the support ending in a cone. However, Ψ is defined differently: $\Psi = \frac{c}{k+1}\sum_{j=1}^{k-1} j \cdot \overline{x_j}$.

For requests in L_0 when no forgiveness step is applied, we have $\Delta cost = 1$, $\Delta cost_{OPT} = 1$, and, by Theorem 1, we get $\Delta\Phi \leq H_k - H_1 - i/(k+1)$, where i is the largest index having $x_i > 0$. Also, similarly to Lemma 6, we get $\Delta\Psi \leq \frac{ci}{k+1}$, which, using $i < k$, leads to $1 + \Delta\Phi + \Delta\Psi \leq H_k + c$.

For pages in support, we analyze the request to a page $p \in L_i$. By definition of Φ, we have $\Delta cost + \Delta\Phi = 0$. The result in Lemma 5 can be adapted straightforward to obtain $\Delta\Psi = -\frac{c}{k+1}\sum_{j=i}^{k-1} \overline{x_j}$. Altogether, we get $\Delta cost + \Delta\Phi + \Delta\Psi \leq 0$.

For requests in L_0 , when forgiveness must be applied, we use the same forgiveness mechanism from [5], where the requested page is artificially inserted in L_1 and processed as a page in L_1. Again, in this case, the algorithm is charged a cost of 1, and OPT is charged 0. We have that $1 + \Delta\Phi$ is the probability of a cache hit for a page in L_1, which is at most $\frac{k}{x+k}$, where $x = \sum_{j=1}^{k} x_j$ is the amount of bookmarks allowed. Using $\Delta\Psi = -\frac{c}{k+1}(x - 1)$, we need to ensure that $\frac{k}{x+k} \leq \frac{cx}{k+1}$. Solving the inequation, we get that it holds for $x \geq -\frac{k}{2} + \frac{\sqrt{c^2k^2+4kc}}{2c}$. Enforcing $x = k/(1 + c)$, the result follows. $\square$

We note that the result in Theorem 4 gives a range of algorithms whose performance is between the classic EQUITABLE and MARKING algorithms, with respect to competitiveness and space usage; in particular, the interesting values for c are such that $c = \omega(1)$ and $c < H_k - 1$. That is because, classic EQUITABLE is H_k-competitive but uses $\Theta(k)$ bookmarks, while MARKING uses no bookmarks, but is $2H_k - 1$ competitive.

4. PARTITION

In this section we prove in the affirmative the conjecture in [5] that there exists a strongly competitive paging algorithm using $o(k)$ bookmarks. We propose a variation of the PARTITION algorithm [13], that we call PARTITION2, which uses $O(k/\log k)$ bookmarks. We furthermore 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 must be 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 with bounded space usage, i.e. depending only on k.

4.1 Partition

In this section we give a brief description of the PARTITION algorithm in [13]. A crucial difference between PARTITION and EQUITABLE is that while the distribution of the cache configurations depends only on the current offset function for EQUITABLE, PARTITION is defined on a special, more detailed, representation of the offset function, which we denote in the following *set-partition*. We show in Observation 1 that the offset function alone does not suffice to determine the probability distribution for the cache of PARTITION[4]. 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 (pages which are in OPT's cache independent of the future requests) and S_α all the pages which are not in OPT's cache. Upon a request to page p the set-partition is updated as follows. If $p \in S_\beta$ nothing changes. If $p \in S_\alpha$ the following assignments are done:

$$S_\alpha = S_\alpha \setminus \{p\},\ S_{\beta+1} = \{p\},\ k_\beta = k-1,\ \beta = \beta+1.$$

The last case covers $p \in S_i$, where $\alpha < i < \beta$:

$$S_i = S_i \setminus \{p\},\ S_\beta = S_\beta \cup \{p\},\ k_j = k_j - 1\ (i \le j < \beta).$$

Additionally, if there are labels which become zero, let j be the largest index such that $k_j = 0$; the following assignments are performed:

$$S_j = S_\alpha \cup \cdots \cup S_j,\ \alpha = j.$$

In [13] it was shown that the following invariants on the labels hold: $k_\alpha = 0$ and $k_i > 0$ for all $i > 0$; $k_\beta = k - |S_{\beta-1}|$. Furthermore, it holds at all times that:

$$k_i = (k_{i-1} + |S_i|) - 1.$$

Probability distribution of cache configurations. 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 set $\mathcal{C}_i = (\mathcal{C}_{i-1} \cup S_i) \setminus \{p\}$
- $\mathcal{C}_\beta = \mathcal{C}_{\beta-1} \cup S_\beta$.

[4]Previous work [1] gave a simplified and intuitive description of PARTITION, but which is not fully accurate.

Note that, whereas for the selection process of ONLINEMIN the size of C_i is given by i, for PARTITION we have that $|\mathcal{C}_i| = k_i$. The following result was given in [13, Lemma 3].

LEMMA 7. *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*

$$\sum_{i \le j < \beta} \frac{1}{k_j + 1}.$$

Cache replacement. Apart from obeying the cache distribution previously introduced, PARTITION must satisfy two constraints, namely 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 do nothing. 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.

OBSERVATION 1. *The probability distribution of* PARTITION *does not depend on the offset function alone.*

PROOF. To illustrate the claim, we give two scenarios leading to the same offset function where there exist a page having different probabilities of being in cache. In the first scenario, we start with the cone $L^1 = (p_1|\ldots|p_{k-1}|q_1)$ and request two pages from L_0, namely q_2 and q_3. Since upon a request in L_0 PARTITION evicts a page uniformly at random from cache, the probability that q_1 is in cache after processing q_3 is $(k-1)^2/k^2$. In the second scenario, we start with offset function $L^2 = (p_1|\ldots|p_{k-1}|q_2)$ and we request q_1 and q_3, both of which are in L_0. This leads to the same layer representation of the offset function as in the first scenario, but the probability that q_1 is in cache is now only $(k-1)/k$, which concludes the proof. □

4.2 Partition2

In this section we describe the PARTITION2 algorithm. As implied by its name, it 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 first 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. In the following 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 $p \in L_0$:

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

Upon the merge operation $L_{i-1} \cup L_i \setminus \{p\}$ in the case $p \in L_i$ 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. An example is given in Figure 1. The following fact follows inductively.

FACT 1. *For L_i, with $i > 0$ and $|L_i| = 1 + x_i$, it holds*

- *L_i contains exactly x_i set identifiers,*
- *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 8 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 8. *Let $S_a, S_{a+1}, \ldots, S_b$ be the sets whose identifiers are in layer L_i, $i \geq 0$. We have:*

$$k_b = i,\ k_{a+j} \geq i \text{ for } 0 \leq j < b - a\ .$$

Req	Offset function	
-	$L = (7, 8, 9\|1\|2\|3\|4\|5\|6)$	
	$S = \{7, 8, 9\}_0\{1, 2, 3, 4, 5, 6\}$	$(\alpha = 1, \beta = 2)$
9	$L = (7, 8\|1\|2\|3\|4\|5, 6, \star\|9)$	
	$S = \{7, 8\}_0\ \{1, 2, 3, 4, 5, 6\}_5\ \{9\}$	$(\alpha = 1, \beta = 3)$
6	$L = (7, 8\|1\|2\|3\|4, 5, \star\|9\|6)$	
	$S = \{7, 8\}_0\ \{1, 2, 3, 4, 5\}_4\ \{9,6\}$	$(\alpha = 1, \beta = 3)$
8	$L = (7\|1\|2\|3\|4, 5, \star\|9, 6, \star\|8)$	
	$S = \{7\}_0\ \{1, 2, 3, 4, 5\}_4\ \{9, 6\}_5\ \{8\}$	$(\alpha = 1, \beta = 4)$
1	$L = (7\|2\|3\|4, 5, \star\|9, 6, \star\|8\|1)$	
	S=$\{7\}_0\ \{2, 3, 4, 5\}_3\ \{9, 6\}_4\ \{8, 1\}$	$(\alpha = 1, \beta = 4)$
9	$L = (7\|2\|3\|4, 5, \star, 6, \star\|8\|1\|9)$	
	$S = \{7\}_0\ \{2, 3, 4, 5\}_3\ \{6\}_3\ \{8, 1, 9\}$	$(\alpha = 1, \beta = 4)$
6	$L = (7\|2\|3, 4, 5, \star, \star\|8\|1\|9\|6)$	
	$\{7\}_0\ \{2, 3, 4, 5\}_3\ \{\}_2\ \{8, 1, 9, 6\}$	$(\alpha = 1, \beta = 4)$
3	$L = (7\|2, 4, 5, \star, \star\|8\|1\|9\|6\|3)$	
	$S = \{7\}_0\ \{2, 4, 5\}_2\ \{\}_1\ \{8, 1, 9, 6, 3\}$	$(\alpha = 1, \beta = 4)$
5	$L = (7, 2, 4\|8\|1\|9\|6\|3\|5)$	
	$\{7, 2, 4\}_0\ \{8, 1, 9, 6, 3, 5\}$	$(\alpha = 3, \beta = 4)$

Figure 1: Example for the layer embedding of the set-representation.

PROOF. We show that the invariant remains true after each update of the set-partition. Let p be the currently requested page; also let L and L' be the layer representation and S and S' the corresponding set-partition before and after processing p respectively.

If page $p \in S_\beta$ nothing (except a shift of the revealed layers in L) changes. If $p \in S_\alpha$ we also have $p \in L_0$. Page $q \in L_k$ followed by a new set identifier (representing the set $S_{\beta'-1}$) is appended to L_{k-1} and $L'_k = \{p\}$. All sets except for $S_{\beta'-1}$ are not affected. The set-partition update rule assigns $k_{\beta'-1} = k - 1$. Since the identifier of $S_{\beta'-1}$ is the rightmost element in L'_{k-1}, the result holds.

Now we turn to the case $p \in S_{i^*}$, where $\alpha < i^* < \beta$. Let L_i be the layer containing p. if L_i is singleton, then for all sets S_{j^*}, $j^* \geq i^*$ we have that both k_{j^*} and its corresponding layer index decrease by 1. Since the relevant parameters for the remaining sets don't change, the result holds. If L_i is not singleton, by construction L_i ends in a set identifier; this set identifier represents a set S_{j^*}, $j^* \geq i^*$. By inductive hypothesis, we get $k_{j^*} = i$. By the update rules, $k'_{j^*} = i - 1$ and it is the last set identifier in L'_{i-1}. All other set identifiers in L_i represent sets having labels at least i, which might decrease by at most 1. All these identifiers are moved to L'_{i-1} and the result follows. □

LEMMA 9. *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 7 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 8 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. $\square$

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 of the offset function (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 [5] is applied. This has a cost of 1 for PARTITION and a cost of 0 for OPT. The size of the support 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 and bookmarks. 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 apply the 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 on the layer representation of the offset function:

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

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 inequation holds before the phase and show that it holds at the end of the phase, but not necessary 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 9

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

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 have to show that $\Delta\Phi \leq -1$.

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

Now assume that $x_{t+1} + \cdots + x_{k-1} \geq 2t$. Before we start the extreme forgiveness mode, we have that

$$\Phi \geq \sum_{j=t+1}^{k-1} x_j(H_{j+1} - 1) \geq 2t(H_{t+2} - 1)$$

By the choice of $t = \lceil \frac{k}{\ln k} \rceil$ and the approximation $H_x \geq \ln x$ we obtain

$$\Phi \geq \frac{2k}{\ln k}(\ln k - \ln\ln k - 1) \geq k, \text{ if } k > 10.$$

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 $\Delta\Phi \leq -1$. $\square$

LEMMA 10. *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 least* $k+k/H_k$.

PROOF. We assume that A is H_k-competitive and the number of pages with non-zero probability is always less than $k + k/H_k$. We start in a cone $(p_1|p_2|\ldots|p_k)$ and request $q_1, q_2, \ldots, q_\alpha$, where $\alpha = k/H_k$ and all q_i have never been requested before. Thus OPT and A perform each α page faults. The resulting work function has the signature $(0|\ldots|0|\alpha|0)$ and the support has size $k+\alpha$. By our assumption there exists at least one page from the support on which A faults with probability 1. Since for the next $k-1$ requests the support does not change we can force $k-1$ page faults on A each with cost 0 for OPT. Afterwards we continue the request sequence to reach a cone and repeat our attack. We conclude that A is not H_k competitive

$$\frac{\text{cost}(A)}{\text{cost}(OPT)} = \frac{k-1+\alpha}{\alpha} = 1 + \frac{k-1}{k/H_k} > H_k \ ,$$

and the proof follows. $\square$

5. CONCLUSIONS

We have shown that PARTITION2 improves the bookmark complexity from $O(k)$ to $O(k/\log k)$ and thus proved the conjecture that there exist H_k-competitive randomized paging algorithms using $o(k)$ bookmarks. This is the best possible for algorithms using deterministic forgiveness techniques and store the whole representation of the (approximated) offset function. One possible direction to improve this bound is to use randomization at the forgiveness step. The more LRU-like distribution of PARTITION and its simple potential in the layer embedding seems to be the more promising candidate.

We stress that the forgiveness used for PARTITION2 does not lead to $o(k)$ bookmarks for the distribution of EQUITABLE. Nonetheless, EQUITABLE is interesting due to its $O(\log k)$ runtime and the elegant potential definition. Moreover, the priority-based selection process in [8] gives an alternate approach to analyzing the EQUITABLE distribution by employing elementary combinatorics.

6. REFERENCES

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

[2] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1–2):3–26, 2003.

[3] W. W. Bein, R. Fleischer, and L. L. Larmore. Limited bookmark randomized online algorithms for the paging problem. *Information Processing Letters*, 76(4-6):155–162, 2000.

[4] W. W. Bein, L. L. Larmore, and J. Noga. Equitable revisited. In *Proc. 15th Annual European Symposium on Algorithms*, pages 419–426, 2007.

[5] W. W. Bein, L. L. Larmore, J. Noga, and R. Reischuk. Knowledge state algorithms. *Algorithmica*, 60(3):653–678, 2011.

[6] L. A. Belady. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive anlysis*. Cambridge University Press, 1998.

[8] G. S. Brodal, G. Moruz, and A. Negoescu. Onlinemin: A fast strongly competitive randomized paging algorithm. *Theory of Computing Systems, Special issue of the 9th Workshop on Approximation and Online Algorithms*, 2013.

[9] M. Chrobak, E. Koutsoupias, and J. Noga. More on randomized on-line algorithms for caching. *Theoretical Computer Science*, 290(3):1997–2008, 2003.

[10] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[11] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[12] E. Koutsoupias and C. H. Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symposium on Foundations of Computer Science*, pages 394–400, 1994.

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

[14] G. Moruz and A. Negoescu. Outperforming LRU via competitive analysis on paramtrized inputs for paging. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1669–1680, 2012.

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

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