

# Equitable Revisited

Wolfgang Bein<sup>1</sup>, Lawrence L. Larmore<sup>1</sup>, and John Noga<sup>2</sup>

<sup>1</sup> Center for the Advanced Study of Algorithms, School of Computer Science,  
University of Nevada Las Vegas, Nevada 89154, USA. \*\*

bein@cs.unlv.edu larmore@cs.unlv.edu

<sup>2</sup> Computer Science Department, California State University,  
Northridge, CA 91330-8281, USA. jnoga@csun.edu

**Abstract.** The randomized  $k$ -paging algorithm EQUITABLE given by Achlioptas *et al.* is  $H_k$ -competitive and uses  $O(k^2 \log k)$  memory. This competitive ratio is best possible. The randomized algorithm RMARK given by Fiat *et al.* is  $(2H_k - 1)$ -competitive, but only uses  $O(k)$  memory. Borodin and El Yaniv [6] list as an open question whether there exists an  $H_k$ -competitive randomized algorithm which requires  $O(k)$  memory for  $k$ -paging. In this paper we answer this question in the affirmative by giving a modification of EQUITABLE.

*Keywords:* Design of Algorithms; Online Algorithms; Paging; Randomization.

## 1 Introduction

In the  $k$ -paging problem we consider a two-level memory system, consisting of fast memory, also referred as the *cache*, which can hold  $k$  memory units commonly called *pages*, and an area of slow memory capable of holding a much larger number of pages. If a page is needed in fast memory this is called a *page request*. Such a request causes a *hit* if the page is already in the cache at the time of the request. In the case of a *miss* (*i.e.* when the page is not in the cache) the requested page must be brought into the cache while a page in the cache must be evicted to make room for the new page. We assume that the total cost of copying a new page and ejecting a page is one, and all other costs are zero. Thus, given a sequence of requests,  $\varrho$ , the cost of an algorithm  $\mathcal{A}$  on  $\varrho$  will be denoted  $cost_{\mathcal{A}}(\varrho)$  and is simply the number of misses algorithm  $\mathcal{A}$  incurs on  $\varrho$ . An online paging algorithm must make decisions about such evictions as the request sequence of pages is presented to the algorithm. Online algorithms are analyzed in terms of *competitiveness*, a measure of the performance that compares the decision made online with the optimal offline solution for the same problem, where the lowest possible competitiveness is best. We say that a randomized online algorithm  $\mathcal{A}$  for the  $k$ -paging problem is  $C$ -competitive if there is a constant  $K$  such that, given any request sequence  $\varrho$ ,  $cost_{\mathcal{A}}(\varrho) \leq C \cdot cost_{opt}(\varrho) + K$ , where  $cost_{opt}(\varrho)$  is the optimal cost for sequence  $\varrho$  and  $cost_{\mathcal{A}}(\varrho)$  is the (expected) cost of the randomized algorithm.

---

\*\* Research of these authors supported by NSF grant CCR-0312093.

For paging the optimal cost can be computed using Belady’s algorithm [5], which replaces the page whose next request is latest. We also refer to this algorithm as OPT. A closely related concept is that of a work function which, when given a configuration, returns the minimum cost of serving the request sequence and ending in this configuration. In the case of paging a configuration is simply the set of  $k$  pages which are in the cache. Koutsoupias and Papadimitriou [8, 9] give a complete characterization of the work function for paging.

We note that the competitive ratio is one aspect of an online algorithm, the memory requirement is another. It is typical to measure the memory requirement of an algorithm by the maximum number of pages remembered by the algorithm: any paging algorithm knows which  $k$  pages are currently in the cache and beyond that, an algorithm might store information about pages which are not currently in the cache. Those pages are called *bookmarks*. Thus the total memory requirement of a paging algorithm is  $k$  plus the number of bookmarks. We mention that if an online algorithm for the problem permits no bookmarks, it is called *trackless*, see [3].

It is well known that the deterministic competitiveness of the  $k$ -paging problem is  $k$ , which can be achieved by the trackless algorithm LRU [11]. The randomized competitiveness of the  $k$ -paging problem is known to be  $H_k = \sum_{i=1}^k 1/i$ . The lower bound result is due to [7]. The same paper also gives a  $(2H_k - 1)$ -competitive algorithm called RMARK which uses only  $O(k)$  memory; this algorithm is also trackless. An algorithm with best possible competitive ratio of  $H_k$  was first described in [10]; however their algorithm, PARTITION, is unbounded in its memory requirement. Later, [1] constructed another algorithm, EQUITABLE, with best possible ratio  $H_k$  and a bounded memory requirement of  $O(k^2 \log k)$  pages. The technique of decreasing memory used in [1] is called *forgiveness*. Under certain conditions, the algorithm simply assumes that OPT’s cache is in a given configuration, despite the fact that it may not be. Informally speaking, the trick is that by this time the algorithm has enough “savings” to cover the cost of this “mistake.” On the other hand Bein and Larmore [3] have shown that it is not possible for a trackless algorithm to achieve  $H_k$ -competitiveness if  $k = 2$ . Borodin and El-Yaniv list in [6] the open question whether there exists an  $H_k$ -competitive randomized algorithm which requires only  $O(k)$  memory.

In this paper we answer this question in the affirmative. In the next section we review EQUITABLE as needed for our paper. In section 3 we improve this technique so that the algorithm (still a version of EQUITABLE) makes a forgiveness step sooner, and is never required to remember more than  $3k$  pages. The basic idea is to not just forgive to one configuration but to use a rather more elaborate scheme.

## 2 EQUITABLE

We assume that the reader is familiar with the definitions and notation of [1] although some definitions and results of that paper will be stated here for clarity and self-containedness. We remind the reader that many randomized online

algorithms are given in a so-called distributional form, as is the algorithm EQUITABLE of [1]. For randomized online algorithms against an oblivious adversary, the distribution model is equivalent to the more standard behavioral model (see *e.g.* [6]). For the  $k$ -cache problem, a *distributional algorithm* is essentially a state transition diagram, where each state is a probabilistic distribution of configurations. More precisely, a *configuration* is an unordered  $k$ -tuple of pages, which represents a possible cache configuration, and we denote by  $\mathcal{S}^k$  be the set of all possible cache configurations. A transition from one state to the next is a deterministic transition to a new distribution. The cost of a move can be defined by the transport distance between the distributions.

In analyzing the competitiveness of online algorithms it is often useful to know the optimal solution for each request sequence. To this end, the *work function*,  $\omega^\varrho : \mathcal{S}^k \rightarrow \mathbb{N}$ , associated with a sequence of requests  $\varrho$  is defined as follows:  $\omega^\varrho(S)$  is the minimum cost of servicing  $\varrho$  while ending in  $S$ . Note that if the last request  $r$  does not belong to  $S$  one can define  $\omega^\varrho(S) = 1 + \min_{S \in \mathcal{S}^k} \omega^\varrho(S + r - x)$ . Note also that the optimal cost of servicing request sequence  $\varrho$  is  $\min(\omega^\varrho)$ . The superscript  $\varrho$  is usually dropped if it is clear from the context.

Given work function  $\omega$  and request  $r$ , one can obtain the work function after service of request  $r$  – denoted as  $(\omega \wedge r)$  – in the following way:  $(\omega \wedge r)(S) = 1 + \min_{S \in \mathcal{S}^k} \omega^t(S + r - x)$  if  $r \notin S$  and simply  $(\omega \wedge r)(S) = \omega(S)$  if  $r \in S$ .

Let  $\mathcal{T}$  be a set of configurations. We say that a work function is coned-up from  $\mathcal{T}$ , if for every configuration  $S$  there is a  $T \in \mathcal{T}$  such that  $\omega(S) = \omega(T) + \|S, T\|$ , where the last term denotes the cost of changing the cache  $S$  to the cache  $T$ , namely the number of pages in  $S$  which are not in  $T$ . If  $\mathcal{T}$  is a singleton, we call  $\omega$  a *cone*.

In this paper (as in [1]) it is convenient to normalize work functions in the following way: If  $\min_{S \in \mathcal{S}^k} \{\omega(S)\} = \text{offset} > 0$  we lower the function by subtracting the constant *offset* and then storing that value; such functions are called *offset functions*. A notation system for offset functions was introduced by Koutsoupias and Papadimitriou [8, 9]; we briefly summarize that concept here. To define any offset function, suppose  $\emptyset = S_0 \subset S_1 \subset S_2 \subset \dots \subset S_k$  are sets of pages, and let  $m_i = |S_i|$ . An offset function  $\omega$  is then defined as follows:

1. We say that  $S \in \mathcal{S}^k$  is in the *support* of  $\omega$ , which we call  $\text{supp}(\omega)$ , if and only if  $|S \cap S_i| \geq i$  for all  $1 \leq i \leq k$ . We have  $\omega(S) = 0$  for all  $S \in \text{supp}(\omega)$ .
2. For any  $T \in \mathcal{S}^k$ ,  $\omega(T) = \min_{S \in \text{supp}(\omega)} \|S, T\|$ .

For convenience we also define  $L_i = S_i - S_{i-1}$ , called *layers*. We write either  $S_1|S_2|\dots|S_k|$  or equivalently  $L_1|L_2|\dots|L_k|$  to represent an offset function. If the initial set of pages in the cache is  $\{s_1, s_2, \dots, s_k\}$  then  $S_i = \{s_1, s_2, \dots, s_i\}$ . Given an offset function  $S_1|S_2|\dots|S_k|$  the offset function resulting from a request to a page  $r$  is

$$\begin{array}{ll} \{r\}|S_2 + r|S_3 + r|\dots|S_k + r| & \text{if } r \notin S_k, \\ \{r\}|S_1 + r|S_2 + r|\dots|S_{i-1} + r|S_{i+1}|S_{i+2}|\dots|S_k| & \text{if } r \in S_i \text{ and } i < k, \\ \{r\}|S_1 + r|S_2 + r|\dots|S_{k-1} + r| & \text{if } r \in S_k, \end{array}$$

where the offset is increased by one in the first case. We will call requests of

the first type *new* requests and requests of the second type *lazy*. We refer to  $S_k$  as the set of *active pages*. Thus, any algorithm which is based upon the offset function must use at least  $m_k - k$  bookmarks, and we say that the algorithm has to “keep track of” these pages.

The concept of *forgiveness* is defined as replacing the current work function  $\omega$  with another function  $\omega'$  where  $\omega(S) \geq \omega'(S)$  for all  $S \in \mathcal{S}^k$ . Note that if the algorithm makes a forgiveness step, the resulting function is no longer an accurate calculation of optimal cost but instead an underestimate. By a slight abuse of terminology we also refer to this estimate as an offset function.

As mentioned above, the algorithm EQUITABLE is defined using the distributional model. It is a *stable distribution* algorithm, *i.e.*, its distribution depends only on the offset function and not on any other information that could be computed from the sequence of requests thus far. If  $\omega$  is the current offset function, the distribution  $\pi^\omega$  will be by the procedure below. (We will omit the superscript  $\omega$  from functions when the current offset function is clear from context.)

1. For any  $S \notin \text{supp}(\omega)$ ,  $\pi(S) = 0$ .
2. For any  $S \in \text{supp}(\omega)$ ,  $\pi(S) > 0$ . The following random process selects a member of  $\text{supp}(\omega)$ :
  - Initialize  $S$  to be the empty set.
  - Let  $T = S_k$ .
  - Execute the following loop until  $|S| = k$ :
    - (a) Select  $x \in T$  uniformly at random.
    - (b) Delete  $x$  from  $T$ .
    - (c) If  $S + x$  is a subset of any member of  $\text{supp}(\omega)$  let  $S = S + x$ .
  - For any  $S \in \text{supp}(\omega)$ , define  $\pi(S)$  to be the probability that  $S$  is selected in the previous procedure.

For an offset function  $\omega$  and page  $x$ , define  $p_x^\omega$  as the probability that the page  $x$  is in the set  $S$  selected in the previous procedure. Equivalently let  $p_x = \sum_{\{S \in \mathcal{S}^k | x \in S\}} \pi(S)$  which is the probability that  $x$  is in the cache. Note that if  $p_x > 0$  then  $x \in S_k$ . Proofs of the following observations can be found in [1].

**Observation 1** *If  $x \in L_i$  and  $y \in L_j$ , where  $i \leq j$ , then  $p_x \geq p_y$ . If  $i = j$  then  $p_x = p_y$ .*

**Observation 2** *For any offset function, all sequences of lazy requests ending when  $|S_k| = k$  have the same cost for EQUITABLE.*

We define the function  $\Phi$  as the cost EQUITABLE incurs on a sequence of lazy requests ending when  $|S_k| = k$ .

### 3 Forgiveness Revisited

For any given offset function the algorithm EQUITABLE2 will maintain the same distribution as EQUITABLE. The difference between the algorithms is the forgiveness step. If  $|S_k| = 3k$  and  $r \notin S_k$  is requested then EQUITABLE2 moves

to offset function  $\{r\}|L_1|L_2|\dots|L_{k-1}|$  and charges OPT zero. One way to view this step is that the requested page is moved into  $L_k$  prior to being requested. The motivation for this choice is that when the set of active pages is large, a request in  $L_k$  and a request outside  $S_k$  have nearly the same cost to EQUITABLE (if  $|S_k| = 3k$  then a page in  $L_k$  will have cost between  $2/3$  and  $1$ ). If we can show that the small additional cost to the online algorithm can be covered, why not move the requested page into  $L_k$ , have it requested, and thereby reduce the size of the set of active pages?

Most of the notation of [1] is used, but the definition of  $\Psi$  is different. Recall that  $m_i = |S_i|$  and define  $\Psi = \Psi(\omega)$  as follows:

$$\Psi = \sum_{i=2}^k \left( \frac{m_i}{i} + H_{i-1} - H_{m_i-1} - 1 \right).$$

Similar to [1] we wish to show that for every request

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

where  $cost$  denotes the expected cost to the algorithm at that step and  $cost_{opt}$  is the optimal cost at that step.

In the discussion below, unprimed variables denote the values before a given request. Primed variables are the values after that request. Recall that  $m_i \geq i$ .

**Lemma 1.** *On a lazy request  $r \in L_j$ ,  $cost + \Delta\Phi + \Delta\Psi \leq H_k \cdot cost_{opt}$ .*

*Proof.* Since  $\Phi$  is the cost for EQUITABLE to serve a lazy sequence of requests ending in a cone, on a lazy request  $cost + \Delta\Phi = 0$ . For a lazy request  $cost_{opt} = 0$  by definition. So it suffices to show that  $\Delta\Psi < 0$ . We have

$$\begin{aligned} \Delta\Psi &= \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_i-1} - \frac{m_i}{i} + H_{m_i-1} \right) \\ &= \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} - H_{m_{i-1}} + H_{m_i-1} \right) \\ &= \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} + \sum_{\ell=m_{i-1}+1}^{m_i-1} \frac{1}{\ell} \right) \\ &\leq \sum_{i=2}^j \left( \frac{m_{i-1} + 1 - m_i}{i} + \sum_{\ell=m_{i-1}+1}^{m_i-1} \frac{1}{i} \right) \\ &= 0. \end{aligned}$$

**Lemma 2.** *On a request  $r \notin S_k$ , if forgiveness does not occur, then  $\Delta\Phi \leq \sum_{i=2}^k \frac{1}{m_i}$ .*

*Proof.* If  $k = 1$  then  $\Phi = \Phi' = 0$  which shows that the lemma is true for  $k = 1$ . Assume  $k > 1$  and that the lemma is true for  $k - 1$ .

For every page  $s \neq r$ ,  $p_s \geq p'_s$ . Since  $p_r = 0$  and  $p'_r = 1$ ,  $\sum_{s \in S_k} p_s - p'_s = 1$ , there must be an item  $x \in S_k$  for which  $p_x - p'_x \leq 1/m_k$ . Without loss of generality,  $x \notin S_1$  because every item  $y \in S_2$  will have  $p_y - p'_y \leq p_x - p'_x$ . Let this page  $x \in S_j$  be the first item in the lazy request sequence which defines  $\Phi$  and  $\Phi'$ . Define the following offset functions:

$$\begin{aligned}\omega &= S_1|S_2|\dots|S_k| \\ \omega' &= r|S_2+r|\dots|S_k+r| \\ \omega \wedge x &= x|S_1+x|S_2+x|\dots|S_{j-1}+x|S_{j+1}|\dots|S_k| \\ \omega' \wedge x &= x|xr|S_2+r+x|\dots|S_{j-1}+r+x|S_{j+1}+r|\dots|S_k+r| \\ \omega_{dropx} &= S_1|S_2|\dots|S_{j-1}|S_{j+1}|\dots|S_k| \\ \omega'_{dropx} &= r|S_2+r|\dots|S_{j-1}+r|S_{j+1}+r|\dots|S_k+r|.\end{aligned}$$

Now we notice that

$$\begin{aligned}\Delta\Phi &\leq \frac{1}{m_k} + \Phi(\omega' \wedge x) - \Phi(\omega \wedge x) \\ &= \frac{1}{m_k} + \Phi(\omega'_{dropx}) - \Phi(\omega_{dropx}) \\ &\leq \frac{1}{m_k} + \sum_{i=2}^{k-1} \frac{1}{m_i} \\ &= \sum_{i=2}^k \frac{1}{m_i}\end{aligned}$$

where the third line follows from the inductive hypothesis.

**Lemma 3.** *On a request  $r \notin S_k$ , if forgiveness does not occur, then*

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

*Proof.* Since  $r \notin S_k$ ,  $m'_i = m_i + 1$ . Given lemma 2, it follows that

$$\begin{aligned}cost + \Delta\Phi + \Delta\Psi &\leq 1 + \sum_{i=2}^k \frac{1}{m_i} + \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_i-1} - \frac{m_i}{i} + H_{m_i-1} \right) \\ &= 1 + \sum_{i=2}^k \frac{1}{m_i} + \sum_{i=2}^k \left( \frac{m_i+1}{i} - H_{m_i} - \frac{m_i}{i} + H_{m_i-1} \right) \\ &= \sum_{i=1}^k \frac{1}{i} \\ &= H_k \cdot cost_{opt}.\end{aligned}$$

**Lemma 4.** *On a request outside  $S_k$  when forgiveness occurs.*

$$cost + \Delta\Phi + \Delta\Psi \leq 0.$$

*Proof.* A forgiveness step occurs when  $S_k = 3k$  and a page  $r \notin S_k$  is requested. The forgiveness step can be seen as placing  $r$  in  $L_k$  and then requesting  $r$ . We can easily compute  $\Delta\Psi$  during a forgiveness step. However, it is easier to compute  $cost + \Delta\Phi$  when  $r$  is added to  $L_k$  and on the request separately.

When  $r$  is placed into  $L_k$  the distribution must be adjusted and the lazy potential changes. The transportation cost necessary to adjust the distribution is  $p'_r$ . Since the cost on all lazy sequences is the same, we can compute the change in potential by considering the sequence which begins with a page  $x \in L_k$ . From Observation 1  $cost + \Delta\Phi = p_x - p'_x + p'_r = p_x \leq k/m_k$ .

When  $r$  is requested  $cost + \Delta\Phi = 0$  because  $\Phi$  is the lazy potential. So it suffices to show that  $k/m_k + \Delta\Psi$  is no more than 0. We have

$$\begin{aligned} cost + \Delta\Phi + \Delta\Psi &\leq \frac{k}{m_k} + \sum_{i=2}^k \left( \frac{m'_i}{i} - H_{m'_i-1} - \frac{m_i}{i} + H_{m_i-1} \right) \\ &= \frac{1}{3} + \sum_{i=2}^k \left( \frac{m_{i-1} + 1 - m_i}{i} - H_{m_{i-1}} + H_{m_i-1} \right) \\ &\leq \frac{1}{3} + \left( \frac{k - m_k}{k} - H_{k-1} + H_{m_k-1} \right) \\ &\leq -\frac{5}{3} + H_{3k-1} - H_{k-1} \\ &\leq 0. \end{aligned}$$

The second inequality above holds because the  $\Delta\Psi$  is decreasing in  $m_\ell$  for all  $\ell < k$ . So the worst case occurs when  $m_\ell = \ell$ .

**Theorem 1.** *EQUITABLE2 is an  $H_k$ -competitive,  $O(k)$  memory, randomized algorithm for the  $k$ -paging problem.*

*Proof.* Since we perform a forgiveness step when the set of active pages has size  $3k$  and a new page is requested, the set of pages we keep track of is never greater than  $3k$ . Lemmas 1, 3, and 4 show that  $cost + \Delta\Phi + \Delta\Psi \leq H_k \cdot cost_{opt}$  on every request type. Noting that  $\Phi + \Psi$  is initially 0 and never negative, summing over every request shows that  $EQUITABLE(\varrho) \leq H_k \cdot OPT(\varrho)$  for any request sequence  $\varrho$ .

## 4 Conclusion

We have given an  $H_k$ -competitive randomized online algorithm for the  $k$ -paging problem which keeps track of only  $3k$  pages. For large  $k$ , Lemma 4 can be improved to  $\alpha k$  where  $\alpha \approx 2.2572$  satisfies  $\alpha^2 - \alpha - \alpha \log(\alpha) = 1$ .

Bein *et al.* [2, 4] show (using a different method than is used here) that for the 2-paging problem three pages suffice. For  $k = 3$ , the technique used in this paper can be used to calculate that 7 pages suffice. We also note that we have not proven  $3k$  to be the minimum number of pages needed for any particular  $k$ . In fact, we conjecture that a stronger result holds than the upper bound from this paper:

*Conjecture 1.* There exists a randomized online algorithm for paging which is  $H_k$ -competitive and uses  $o(k)$  bookmarks (i.e.  $k + o(k)$  memory).

## References

1. Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoret. Comput. Sci.*, 234:203–218, 2000.
2. Wolfgang Bein, Rudolph Fleischer, and Lawrence L. Larmore. Limited bookmark randomized online algorithms for the paging problem. *Inform. Process. Lett.*, 76:155–162, 2000.
3. Wolfgang Bein and Lawrence L. Larmore. Trackless online algorithms for the server problem. *Inform. Process. Lett.*, 74:73–79, 2000.
4. Wolfgang Bein, Lawrence L. Larmore, and Rüdiger Reischuk. Knowledge state algorithms: Randomization with limited information. *Arxiv: archive.org/cs/0701142*, 2007.
5. Laszlo A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Syst. J.*, 5:78–101, 1966.
6. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
7. Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12:685–699, 1991.
8. Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science (FOCS)*, pages 394–400. IEEE, 1994.
9. Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. *SIAM J. Comput.*, 30:300–317, 2000.
10. Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.
11. Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.