Competitive Analysis of Randomized Paging Algorithms

Dimitris Achlioptas* Marek Chrobak† John Noga‡

October 4, 1996

Abstract

The paging problem is defined as follows: we are given a two-level memory system, in which one level is a fast memory, called *cache*, capable of holding k items, and the second level is an unbounded but slow memory. At each given time step, a request to an item is issued. Given a request to an item p, a *miss* occurs if p is not present in the fast memory. In response to a miss, we need to choose an item q in the cache and replace it by p. The choice of q needs to be made *on-line*, without the knowledge of future requests. The objective is to design a replacement strategy with a small number of misses.

In this paper we use competitive analysis to study the performance of randomized on-line paging algorithms. Our goal is to show how the concept of work functions, used previously mostly for the analysis of deterministic algorithms, can also be applied, in a systematic fashion, to the randomized case. We present two results: we first show that the competitive ratio of the marking algorithm is exactly $2H_k - 1$. Previously, it was known to be between H_k and $2H_k$. Then we provide a new, H_k-competitive algorithm for paging. Our algorithm, as well as its analysis, is simpler than the known algorithm by McGeoch and Sleator. Another advantage of our algorithm is that it can be implemented with complexity bounds independent of the number of past requests: $O(k^2 \log k)$ memory and $O(k^2)$ time per request.

1 Introduction

The paging problem is defined as follows: we are given a two-level memory system, in which one level is a fast memory (that we refer to as *cache*) capable of holding k items, and the second level is an unbounded but slow memory. At each given time step, a request to an item is issued. Given

*Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 3G4, Canada. Email: optas@cs.toronto.edu. Research supported by an NSERC fellowship.

†Department of Computer Science, University of California, Riverside, CA 92521, USA. Email: marek@cs.ucr.edu. Research supported by the NSF grant CCR-9503498.

‡Department of Mathematics, University of California, Riverside, CA 92521, USA. Email: jnoga@cs.ucr.edu. Research supported by the NSF grant CCR-9503498.

a request to an item p, a *miss* occurs if p is not present in the fast memory. In response to a miss, we need to move p from the slow memory into the cache. In order to make room for p, one of the items residing currently in the cache, say q, needs to be evicted. The choice of q is made *on-line*, i.e., before the next request is issued, and a strategy for making such choices will be referred to as an *on-line algorithm*.

The cost function associated with the paging problem is the number of misses. In general, no on-line algorithm can always achieve a minimum cost on all request sequences. Therefore, in order to evaluate various on-line algorithms, one needs to design a performance measure that takes into account the on-line nature of the problem. In this paper we use the competitive analysis approach: a given on-line algorithm is said to be *c-competitive*, if on every request sequence its cost is bounded (asymptotically) by c times the optimal cost for this sequence.

Paging is a classical on-line problem and has been extensively studied in the literature on competitive on-line algorithms. It can be viewed as a special case of the k-server problem (see, for example, [10, 12, 2]), in which all distances are equal to one. In the deterministic case, it is known that the well-known LRU (least recently used) strategy is k-competitive, and that no better competitiveness is possible (see [15]).

In this paper we concentrate on the randomized version of the paging problem. It is relatively easy to show (see [6]) that no randomized on-line algorithm can be better than H_k-competitive, where H_k is the k-th harmonic number. Two algorithms have been proposed for this problem in the past. Fiat et al [6] gave a simple marking algorithm (called MARK) and proved that it is $2H_k$-competitive. Subsequently, McGeoch and Sleator [13] presented another algorithm, called PARTITION, and proved that it is H_k-competitive, and thus optimal.

Work functions have played an important role in the analysis of on-line problems. However, up until recently, this technique has been used mostly in the analysis of deterministic algorithms. The only example that we are aware of, where work functions have been applied to the randomized case is [5], where some optimal randomized algorithms for the page migration problem were developed based on this technique. One of our goals was to show how work functions can also be applied in the competitive analysis of randomized algorithms for the paging problem. A work function approach usually starts with some characterization of work functions for a given problem. Then, the properties of work functions are used to design and analyze an on-line algorithm. Koutsoupias and Papadimitriou [9] presented a simple, elegant characterization of work functions for paging. In an earlier work, McGeoch and Sleator in [13] gave an equivalent characterization of the behavior of an optimal algorithm, although their formulation did not explicitly involve work functions.

Analysis of MARK. Algorithm MARK of [6], even though not optimal, is of its own interest. It is simpler, faster and more space-efficient than PARTITION. MARK can be implemented with $O(k)$ memory and in $O(1)$ time per request, while PARTITION may need as much as $\Omega(n)$ memory, where n is the number of past requests. Fiat et al [6] provided an upper bound of $2H_k$ on the competitive ratio of MARK. The trivial lower bound is H_k. Thus it would be interesting to know the exact competitive ratio of this algorithm. Our first result is a proof that the competitive ratio of MARK is $2H_k - 1$.

A new optimal algorithm. The result of McGeoch and Sleator [13] gives a tight bound on the optimal competitive ratio in the randomized case. However, the algorithm PARTITION from [13] is somewhat counterintuitive, and both the correctness and competitiveness proofs are rather difficult. In the second part of the paper, we present an alternative H_k-competitive algorithm for paging, called EQUITABLE, that we believe to be simpler and more natural. In fact, our algorithm follows quite naturally from the concepts of work functions, stable algorithms, and some basic game-theoretic principles. We believe that those ideas can be extended to other on-line problems. We also show that our algorithm can be implemented in space $O(k^2 \log k)$ and time $O(k^2)$ per request. Unlike PARTITION, these bounds are independent of the length of the request sequence.

2 Preliminaries

Throughout the paper, by k we denote the cache size. By a *cache configuration*, or simply *configuration*, we will mean a k-tuple of items representing the cache content. We will assume that the initial configuration is fixed, and we call it X_0. As explained in the introduction, an on-line paging algorithm $\mathcal{A}$ needs to respond to every request p before the next request is issued. If a miss occurs, that is, if p is not in the cache, $\mathcal{A}$ must decide which item q should be evicted from the cache to make room for p. Each miss has unit cost.

Mathematically, it is convenient to define an on-line algorithm as a function $\mathcal{A}(\varrho)$ that to a given request sequence ϱ assigns the configuration after serving ϱ. In order to ensure that the requests are satisfied, if r is the last request in ϱ then we require that $r \in \mathcal{A}(\varrho)$. Note that this definition allows $\mathcal{A}$ to swap an arbitrary number of items in the cache at each step, whether a miss occurred or not. However, we will charge $\mathcal{A}$ a cost of 1 for each such swap. It is easy to see that in this case, without loss of generality, $\mathcal{A}$ will never bring an item into the cache unless it has been currently requested.

Denote by $cost_{\mathcal{A}}(\varrho)$ the cost of $\mathcal{A}$ on request sequence ϱ, and by $opt(\varrho)$ the optimal cost on ϱ. We will say that $\mathcal{A}$ is *c-competitive* if there is a constant a such that on every request sequence ϱ

$$cost_{\mathcal{A}}(\varrho) \quad \leq \quad c \cdot opt(\varrho) + a \ . \tag{1}$$

In our algorithms the additive constant a will be zero. The *competitive ratio* of $\mathcal{A}$ is the minimum c for which $\mathcal{A}$ is c-competitive. (In our applications this minimum is well-defined.)

There are several ways to define a randomized algorithm. One can define a randomized algorithm as a probability distribution on the set of all deterministic algorithms for a given problem.

Another way is to view a randomized algorithm as an algorithm that at every step chooses its move from a probability distribution on the set of possible moves. In the theory of multi-stage games these two approaches are sometimes called, respectively, *mixed strategies* and *behavior strategies* (see, for example, [11]). The definitions of cost and competitiveness extend naturally to randomized algorithms, independently of which of the two above definitions is being used. If $\mathcal{A}$ is a randomized algorithm then $cost_{\mathcal{A}}(\varrho)$ denotes the expected cost of $\mathcal{A}$ on ϱ, and inequality (1) remains unchanged. It is quite easy to show that these approaches are equivalent, in the sense that an algorithm of each type can be transformed into one of the other type without increasing its (expected) cost on any request sequence.

Yet another way is to consider an algorithm that at each request chooses, deterministically, its probability distribution on the configuration set. In that case $\mathcal{A}(\varrho)$ is a probability distribution on the set of all possible configurations. We call it a *distribution-based* algorithm. The cost of a move can be defined by a so-called transport distance between the distributions. It can be shown that this approach is equivalent to the other two (see [3, 5]).

Algorithm MARK is defined as a behavior algorithm, while our H_k-competitive algorithm EQUITABLE is easier to define using the distribution-based approach. However, for the sake of completeness and for cost estimation, we also show how to "implement" EQUITABLE as a behavior algorithm.

In analyzing the competitiveness of on-line algorithms it is often useful to know the optimal solution for each request sequence. A *work function* is a function from the set of possible configurations to real numbers which gives the optimal cost to serve a sequence of requests and end in a particular configuration. Specifically, the work function ω associated with a sequence of requests ϱ is defined as follows: $\omega(X)$ is the minimum cost of servicing ϱ, starting from the initial configuration X_0 and ending in X. Note that, for convenience, in this definition we do not insist that the last request r belongs to X. In such a case, define $\omega(X) = 1 + \min_{x\in X}\omega(X + r - x)$. In other words, we allow an optimal algorithm to swap r out of the cache after the request has been satisfied and before the next request was issued. The optimal cost of servicing a request sequence with associated work function ω is simply $\min(\omega)$.

Suppose that the current work function is ω and r is the new request. What is the new work function, after serving r? Denote this new, updated, work function by $\omega\wedge r$. It is straightforward to see that this function is

$$\omega\wedge r(X) \quad = \quad \begin{cases} \omega(X) & \text{if } r \in X \\ 1 + \min_{x\in X}\omega(X + r - x) & \text{if } r \notin X \ . \end{cases}$$

With each work function ω we can associate an *offset function* $\omega - \min(\omega)$. Instead of keeping track of work functions, it is more convenient to deal with offset functions. If ω is the current work function and μ is the current offset function then by μ^r we will denote the offset function after request r, that is $\mu^r = \omega\wedge r - \min(\omega\wedge r)$. The value $\min(\omega\wedge r) - \min(\omega) = \min(\mu\wedge r)$ is the optimal cost associated with serving the request for r.

Let K be a set of configurations and ω a work function. We say that ω is *coned-up* from K, if for every configuration X there is a $Y \in K$ such that $\omega(X) = \omega(Y) + |X - Y|$. In other words, the value of ω on all configurations is uniquely determined by its values in K. If K is a singleton, $K = \{X\}$, then we call ω a *cone on* X.

Potential argument. In order to prove competitiveness of a given algorithm $\mathcal{A}$, we will use amortized analysis. A potential function Φ assigns a real number to a given offset function and a configuration of $\mathcal{A}$. (To simplify notation, throughout the paper we will omit the function arguments when defining potentials.) We consider each move separately. Each move is described by a current offset function ω, a configuration of $\mathcal{A}$, and a request r. Denote by $\Delta cost_{\mathcal{A}}$, Δopt and $\Delta\Phi$, the cost of $\mathcal{A}$, the optimal cost, and the change of the potential in this move. Our goal is to show that for every move

$$\Delta cost_{\mathcal{A}} + \Delta\Phi \quad \leq \quad c \cdot \Delta opt \ . \tag{2}$$

Inequality (2) implies c-competitiveness of $\mathcal{A}$ by simple summation over the whole request sequence. This method works for randomized algorithms as well, the only difference being that the potential now depends on the current offset function and the distribution of $\mathcal{A}$, while $\Delta cost_{\mathcal{A}}$ is the expected cost of $\mathcal{A}$ in the given move.

Characterization of work functions. Koutsoupias and Papadimitriou [9] gave the following elegant characterization of the work functions for the paging problem with the cache of size k.

Lemma 1 *Every offset function is coned up from the set of configurations for which its value is zero. Moreover, if ω is the current offset function and r is the last request then there is a sequence of sets $L_1, L_2, \ldots, L_k$, with $L_1 = \{r\}$, such that $\omega(X) = 0$ iff $|X \cap \bigcup_{i \leq j} L_i| \geq j$ for all $1 \leq j \leq k$.*

The sets L_i are called the *layers* of ω. Note that the above representation is not always unique. Configurations for which the value of ω is zero will be called *valid* and the collection of all of all valid configurations will be denoted by $V(\omega)$.

The set $S(\omega) = \bigcup_{i \leq k} L_i$ will be called the *support* of ω. By Lemma 1, we can identify ω with its sequence of layers, and write $\omega = (L_1|L_2|\ldots|L_k)$. We always have $|L_1| = 1$. Let i be the largest number for which $L_1, \ldots, L_i$ are singletons. All items in $L_1 \cup \ldots \cup L_i$ are called *revealed*. We can assume that an optimal algorithm has all revealed items in the cache, since any optimal algorithm can be modified to one with this property without raising its cost. By $N(\omega)$ we denote the set of non-revealed items in $S(\omega)$.

Koutsoupias and Papadimitriou [9] also give a method for updating the layers after a request. Let $\omega = (L_1|\ldots|L_k)$. Suppose that r is a new request. Then

$$\omega^r \;=\; \begin{cases} (r|L_2|\ldots|L_{j-1}|L_j \cup L_{j+1} - r|L_{j+2}|\ldots|L_k) & \text{if } r \in L_j \text{ and } j < k \\ (r|L_2|\ldots|\ldots|L_{k-1}) & \text{if } r \in L_k \\ (r|L_1 \cup L_2|L_3|\ldots|L_k). & \text{if } r \notin S(\omega) \quad . \end{cases} \tag{3}$$

To simplify notation, in the equations above and throughout the paper, we will often omit braces in the notation for sets. We will also write $X + x$ (or $X - x$) when adding an item x to a set X (respectively, removing x from X). If $Z \subseteq X$ for some $X \in V(\omega)$, then we will also use notation ω^Z for $\omega^{z_1 \ldots z_p}$, where $z_1 \ldots z_p$ is an arbitrary permutation of Z. (It is easy to see that this is well-defined.)

Example: Let $k = 3$, and suppose that originally items a, b, and c are in the cache. So the initial configuration is $X_0 = \{a, b, c\}$ and the initial work function is the cone on X_0, represented by $(a|b|c)$ (the order of a, b, c in this representation is arbitrary). Consider the request sequence d, e, b. The corresponding sequence of offset functions is

$$(a|b|c) \;\rightarrow\; (d|a, b|c) \;\rightarrow\; (e|a, b, d|c) \;\rightarrow\; (b|e|a, c, d) \;\; .$$

The optimal cost of this sequence is 2. ♠

3 Analysis of Mark

Algorithm Mark maintains up to k marks on the items that are in the cache. At the beginning, Mark initializes all items in the cache as marked. Suppose that an item p is requested. If p is in the cache, we mark it, unless it is already marked. If p is not in the cache, we first check if all items in the cache are already marked, and if so, we unmark them all. Then we mark p and swap it with a random, uniformly chosen, unmarked item in the cache.

For the purpose of the analysis, we will use two different marks: *amber* and *blue*. The amber marks are the same as the ones used by Mark. The blue marks are given to the amber items when they are unmarked by Mark, and they are erased when a new set of amber marks is changed to blue. For convenience, we will refer to non-marked items as *white* items.

Let a and b denote the numbers of amber and blue items. The ranges of a, b are: $1 \leq a \leq k$, $2 \leq b \leq k$ or $b = 0$, and $a + b \geq k$. Also, $a = k$ implies $b = 0$, $a = 1$ implies $b = k$, and $b > 1$ implies $a + b > k$. The following fact follows directly from the definition of Mark.

Fact 1 *The amber items are in the cache. Each blue item is in the cache with probability* $(k - a)/b$.

Lemma 2 *The following invariant is preserved by* Mark*: Let* $\omega = (L_1|\ldots|L_k)$ *be the current offset function, and denote by* L_{k+1} *the set of items outside* $S(\omega)$*. Then for each* $i > 1$*:*

(a) If L_i contains an amber item then L_{i-1} contains only amber items;

(b) If L_i contains a blue item, then L_{i-1} contains only amber or blue items.

Say that $x \in L_i$ *occurs before* $y \in L_j$ if $i < j$. An alternative way to state the above lemma is that a blue item cannot occur before an amber item, and a white item cannot occur before a blue item.

Proof: The proof is by induction on the number of requests. The lemma holds trivially at the beginning, when our state is a cone on a set X_0 and all items in X_0 are amber. For the inductive step, assume that lemma holds for $\omega = (L_1|L_2|...|L_k)$, let $r \in L_j$ and consider ω^r. We can think about the rules (3) as being combinations of the following operations: (i) joining two consecutive layers, (ii) removing an item from a layer, and (iii) creating a new first layer colored amber. All these operations preserve invariants (a) and (b). □

By the above lemma, each state of the computation is uniquely characterized by a triple (ω, a, b) where ω is the current offset function and a, b are the numbers of amber and blue items.

Theorem 1 *The competitive ratio of algorithm* MARK *is* $2H_k - 1$.

Proof:[lower bound] We will show a cycle in which the optimal cost is 1, and the cost of MARK is $2H_k - 1$.

Start from a state in which the offset function is a cone on $X = \{x_1, \ldots, x_k\}$, but MARK has 1 amber mark on x_1, the last request. The remaining $k-1$ items are uniformly distributed among k blue items, including $x_2, \ldots, x_k$ and one additional item $y \notin X$. Request a new item $x_0 \notin X + y$. Then $\omega^{x_0} = (x_0|x_1, x_2|x_3| \ldots |x_k)$, and MARK now has amber marks on x_0, x_1 and the same set of k blue marks as before. From now on we request $x_2, \ldots, x_{k-1}$, and we reach an offset function $(x_{k-1}| \ldots |x_2|x_0|x_1, x_k)$, in which MARK will have k amber marks on $x_0, \ldots, x_{k-1}$. Then request x_k, which is now white, going back to a state equivalent to the one we started from. The optimal cost is 1, and the cost of MARK is

$$1 + \frac{2}{k} + \frac{2}{k-1} + \ldots \frac{2}{3} + 1 \quad = \quad 2H_k - 1 \ .$$

To complete the proof it remains to show that the state from which we started the cycle can be reached from an initial state. We leave this as an exercise. □

Proof:[upper bound] Now we will prove that Algorithm MARK is $(2H_k - 1)$-competitive. We divide the computation into *phases*, where each phase starts when $a = k$ and a non-amber item is requested. Let ω be a given offset function at the beginning of a phase, and let s be the number of layers in ω that contain white items. We use the following potential function:

$$\Phi \quad = \quad s(H_k - H_s + 1)$$

For a fixed phase starting at ω, let l and t denote the numbers of white requests in this phase which are in the support, and outside the support, respectively. Let s' be the new value of s, at the beginning of the new phase. Also, let $\Delta cost$ be MARK's cost, $\Delta\Phi$ be the potential change, and Δopt be the optimal cost in this phase. We have $l \leq s$ because the number of white requests from the support cannot be greater than the number of white layers. After the first request in a phase there is one level consisting of only one amber item. For each non-amber request inside the support the number of levels with only amber items increases by one. Therefore, $s' \leq t = \Delta opt$ and if $l = 0$, $s' \leq t - 1$.

Furthermore, we claim that $\Delta cost \leq (l+t)(H_k - H_{l+t} + 1)$. MARK will fault on each of the $l + t$ white requests and may fault on blue requests. The probability that MARK will fault on the ith blue request given that j white requests have already been made is $j/(k - i + 1)$. Since this probability is increasing in j, the cost of faulting on the ith blue request is no more than $(l+t)/(k-i+1)$. Summing these estimates yields $\Delta cost \leq (l+t)(H_k - H_{l+t} + 1)$.

Now, we proceed as follows:

$$
\begin{aligned}
\Delta cost + \Delta\Phi &\leq (l+t)(H_k - H_{l+t} + 1) + s'(H_k - H_{s'} + 1) - s(H_k - H_s + 1) \\
&\leq (l+t)(H_k - H_{l+t} + 1) + s'H_k - l(H_k - H_l + 1) \\
&\leq (2H_k - 1)t - (l+t)H_{l+t} + lH_l + 2t - (t - s')H_k \\
&\leq (2H_k - 1)t \\
&= (2H_k - 1)\Delta opt \ .
\end{aligned}
$$

If $l > 0$, the last inequality holds because $2t - (t+l)H_{l+t}$ is decreasing in t. If $l = 0$, the last inequality holds because in this case $t - s' \geq 1$ and $2t - tH_t$ is maximized when $t = 1$. This implies the $(2H_k - 1)$-competitiveness of MARK, as explained in the previous section. □

Note that extending any sequence by appending requests from the support until each layer is a singleton and all items in the support are amber, will not increase the optimal cost and can only increase the cost of MARK. Thus, we can assume that the final potential is zero, and therefore the additive constant in (1) is 0.

4 A New Optimal Algorithm

A randomized algorithm is said to be *stable* if its probability distribution at each step does not depend upon anything other than the current offset function. Thus, a stable algorithm can be described as a function $\mathcal{A}(\omega)$ that describes the distribution of $\mathcal{A}$ if the current offset function is ω. Naturally, for stable randomized algorithms the potential function will depend only on the current offset function.

Partition, the algorithm proposed by McGeoch and Sleator in [13], can be described as distribution-based and stable. Its probability distribution is defined by the following k-round tournament: Initially, each $x \in L_i$ is given rank $i+1$. At step $i = k, k-1, \ldots, 1$, pick uniformly $k - i + 1$ items from among those that have rank $i + 1$, and decrease their rank to i. At the end, the k winners of rank 1, are chosen as the configuration. Call an arbitrary ranking valid if (a) each item of rank $i \geq 2$ belongs to $L_{i-1} \cup \ldots \cup L_k$, and (b) there are exactly $k - i + 1$ items of rank $\leq i$ in $L_i \cup \ldots \cup L_k$. McGeoch and Sleator prove that the above tournament generates the uniform probability distribution on all valid rankings, and use this fact to prove that Partition is H_k-competitive.

Our approach is different. First, assume that we are looking for a stable algorithm. Thus, we need to specify a probability distribution $P(\omega)$ on the set of configurations which are used by this algorithm when the current offset function is ω. How can we derive this distribution? Suppose that the requests are generated by a so-called *lazy adversary*, that is one that starting at ω only makes requests that do not increase the optimal cost. In other words, the requests are in the support of ω. We can assume that the adversary never requests the revealed items, and thus each lazy sequence consists of at most $k - 1$ requests, and the final offset function is a cone. We refer to such request sequences as adversary *lazy strategies*.

What is the probability distribution that would best protect us against such an adversary? Clearly, the ideal distribution is one for which all such lazy strategies have the same on-line cost. (We leave it to the reader to verify that Partition does not satisfy this property for $k \geq 3$.) View the problem as a two-person zero-sum game. Let our player determine his configuration. Let the adversary determine his lazy strategy, item by item. Using the fact that the value of an optimal mixed strategy against a pure (deterministic) strategy which appears with nonzero probability in some optimal mixed strategy gives the value of the game, it is not hard to see that the adversary should pick the first request uniformly from the support. It seems reasonable that the optimal distribution for our player would match the optimal adversary strategy. This yields the following algorithm.

Algorithm Equitable. The algorithm is defined by a probability distribution $P_X(\omega)$ of being in a configuration X when the current offset function is ω. Select X as follows: Let $X = \emptyset$. While $|X| < k$ select an item x uniformly from $S(\omega^X) - X$ and add this item to X. $P_X(\omega)$ is the probability of selecting X using the above random process.

In other words, $P_X(\omega)$ is the probability of reaching the cone on X if, starting at ω, at each step a request is chosen uniformly from the support. For simplicity, we will use the same notation $P(\omega)$ for the distribution of Equitable and for the random process that generates this distribution. Only valid configurations have positive probability of being used by Equitable.

Denote the probability that Equitable has x in the cache when in ω by $P_x(\omega)$. Obviously,

$P_x(\omega) = 0$ for $x \notin S(\omega)$ and $P_x(\omega) = 1$ for $x \in S(\omega) - N(\omega)$. Let M be any set such that $N(\omega) \subseteq M \subseteq S(\omega)$. We can as well assume $P(\omega)$ initializes X to $S(\omega) - M$ instead of $\emptyset$. Therefore

$$P_x(\omega) \quad = \quad \frac{1}{|M|} \sum_{z \in M} P_x(\omega^z) \ . \tag{4}$$

Lemma 3 *If the current offset function is ω, and the request is r, then the cost of* EQUITABLE *on this request is $1 - P_r(\omega)$, the probability that r is not in the cache.*

Proof: It is sufficient to show how EQUITABLE can be "implemented" as a behavior algorithm $\mathcal{B}$ that has the following properties: (i) if a requested item is in the cache, then $\mathcal{B}$ does not move, and (ii) if a request is not in the cache, then $\mathcal{B}$ only swaps one item. By "implement" we mean that at each step $\mathcal{B}$ induces the same probability distribution as EQUITABLE.

Given an offset function ω, a configuration X of $\mathcal{B}$, and a request r, we want to define how $\mathcal{B}$ will serve r. $\mathcal{B}$ orders X randomly as follows: a permutation $x_1 x_2 ... x_k$ is chosen with the same probability that this would be the order the items of X were chosen by $P(\omega)$, given that the set X was chosen. Let j be the largest integer such that $X + r - x_j \in V(\omega^r)$. Then $\mathcal{B}$ replaces x_j in the cache by r.

Note that if $r \in X$ then $x_j = r$ and no actual swap is needed. If $r \notin S(\omega)$ then we have $j = k$. In the last case, if $r \in S(\omega) - X$, then there is a unique p such that $r \in S(\omega^{X_{p-1}}) - S(\omega^{X_p})$. Equivalently, both x_p and r are in the last layer of $\omega^{X_{p-1}}$. By the choice of p, we have $X - x_p + r \in V(\omega^r)$ and $X - x_i + r \notin V(\omega^r)$ for $i > p$. Therefore $j = p$, and we conclude that j is the index for which x_j and r are in the last layer of $\omega^{X_{j-1}}$. Additionally, we have $x_i \in S(\omega^{r + X_{i-1}}) - X_{i-1} - r$ for all $i < j$.

$\mathcal{B}$ clearly satisfies conditions (i) and (ii) above, and it generates only configurations in $V(\omega^r)$, so it remains to show that it induces the same distribution on $V(\omega^r)$ as EQUITABLE. The proof is by induction on the number of requests. It is certainly true in the initial state, so assume that it holds for some offset function ω. We will show that it also holds for ω^r. We can assume that when $P(\omega^r)$ chooses its configuration Y, it initializes $Y = \{r\}$ and then it selects the remaining $k-1$ items $y_1, \ldots, y_{k-1}$ of Y. Denote $Y_i = \{y_1, \ldots, y_i\}$ for each i. Recall that each y_{i+1} is uniformly distributed in $S(\omega^{Y_i + r}) - Y_i - r$.

We can also think of $\mathcal{B}$ as generating a sequence of $k-1$ items other than r, namely the $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{j+1}, \ldots, x_k$. For convenience, rename this sequence $z_1, \ldots, z_{k-1}$. Let $Z_i = \{z_1, \ldots, z_i\}$ for each $i \le k-1$, and $Z = Z_{k-1} + r$. We want to show that random variables Y and Z have the same distribution. In order to do so, it is sufficient to prove the following claim: for each i, z_{i+1} is uniformly distributed in $S(\omega^{Z_i + r}) - Z_i - r$. We break the proof into two cases.

Case A: $r \notin S(\omega)$. Then $j = k$, and $z_i = x_i$ for each $i \le k-1$. Then, since $S(\omega^{r+X_i}) - X_i - r = S(\omega^{X_i}) - X_i$ for $i \le k-2$, both processes make the same random choices at each step, and the claim follows.

Case B: $r \in S(\omega)$. The proof is by induction on i. It holds vacuously for $i = 0$. If $z_{i+1} = x_{i+1}$ then $i + 1 < j$, $Z_i = X_i$ and x_{i+1} was chosen uniformly from $S(\omega^{X_i}) - X_i$. But x_{i+1} is in $S(\omega^{r+X_i}) - X_i - r$ because $X + r - x_j \in V(\omega^r)$. Since $S(\omega^{r+X_i}) - X_i - r \subseteq S(\omega^{X_i}) - X_i$, we obtain that x_{i+1} is distributed uniformly in $S(\omega^{X_i+r}) - X_i - r$. In the other case, if $z_{i+1} = x_{i+2}$ then x_{i+2} was chosen uniformly from $S(\omega^{X_{i+1}}) - X_{i+1} = S(\omega^{r+Z_i}) - Z_i - r$, completing the proof. □

Example: Let $\omega = (a|b, c|d|e, f)$. The following chart summarizes the distribution of EQUITABLE:

X	$\{a,b,c,d\}$	$\{a,b,c,e\}$	$\{a,b,c,f\}$	$\{a,b,d,e\}$	$\{a,b,d,f\}$	$\{a,c,d,e\}$	$\{a,c,d,f\}$
$P_X(\omega)$	$\frac{1}{10}$	$\frac{3}{20}$	$\frac{3}{20}$	$\frac{3}{20}$	$\frac{3}{20}$	$\frac{3}{20}$	$\frac{3}{20}$

To illustrate how to obtain the above numbers, consider the probability of $X = \{a, b, c, e\}$. In the process used to define EQUITABLE, X can be generated in 24 different orders. The permutations (a, b, c, e), (a, c, b, e), (b, a, c, e), (b, c, a, e), (c, a, b, e), and (c, b, a, e) have probabilities $\frac{1}{6}\cdot\frac{1}{5}\cdot\frac{1}{4}\cdot\frac{1}{3} = \frac{1}{360}$. The permutations (a, b, e, c), (a, c, e, b), (b, a, e, c), and (c, a, e, b) have probabilities $\frac{1}{6}\cdot\frac{1}{5}\cdot\frac{1}{4}\cdot\frac{1}{2} = \frac{1}{240}$. The permutations (a, e, b, c), (a, e, c, b), (b, e, a, c), and (c, e, a, b) have probabilities $\frac{1}{6}\cdot\frac{1}{5}\cdot\frac{1}{3}\cdot\frac{1}{2} = \frac{1}{180}$. The permutations (b, c, e, a) and (c, b, e, a) have probabilities $\frac{1}{6}\cdot\frac{1}{5}\cdot\frac{1}{4}\cdot\frac{1}{1} = \frac{1}{120}$. Permutations (b, e, c, a) and (c, e, b, a) have probabilities $\frac{1}{6}\cdot\frac{1}{5}\cdot\frac{1}{3}\cdot\frac{1}{1} = \frac{1}{90}$. Permutations (e, a, b, c), (e, a, c, b), (e, b, a, c), and (e, c, a, b) have probabilities $\frac{1}{6}\cdot\frac{1}{4}\cdot\frac{1}{3}\cdot\frac{1}{2} = \frac{1}{144}$. Permutations (e, b, c, a) and (e, c, b, a) have probabilities $\frac{1}{6}\cdot\frac{1}{4}\cdot\frac{1}{3}\cdot\frac{1}{1} = \frac{1}{72}$. So the probability of X is $\frac{3}{20}$.

What is the probability $P_c(\omega)$? Item c can be generated in a number of ways. With probability $\frac{1}{6}$, it can be generated in the first step. With probability $\frac{1}{6}\cdot\frac{1}{5}$ it can be generated after b. By considering all possible sequences ending at c, we obtain $P_c(\omega) = \frac{7}{10}$. ♠

Lemma 4 *The Commutativity Lemma: For each offset function ω and any two items $x, y \in S(\omega)$*

$$P_x(\omega) + P_y(\omega^x) \quad = \quad P_y(\omega) + P_x(\omega^y) \ .$$

Proof: The proof is by induction on $n = |N(\omega)|$. If $n = 0$, then the lemma holds vacuously. In the inductive step, suppose that $\omega = (L_1|...|L_k)$, and that the lemma holds for every offset function with fewer non-revealed items, in particular for each ω^z where $z \in N(\omega)$.

The lemma is obvious if x, y are in the same layer. So we can assume that $x \in L_i$, $y \in L_j$, for some $1 \le i < j \le k$. If x is revealed then the lemma is clearly true. So assume $x, y \in N(\omega)$. Note that $\omega^{pq} = \omega^{qp}$ for all items $p \in S(\omega) - L_k$ and $q \in S(\omega)$. Then, if $j < k$, using (4) and the inductive assumption, we have

$$\begin{aligned} P_x(\omega) + P_y(\omega^x) &= \frac{1}{n}\sum_{z\in N}[P_x(\omega^z) + P_y(\omega^{zx})] \\ &= \frac{1}{n}\sum_{z\in N}[P_y(\omega^z) + P_x(\omega^{zy})] = P_y(\omega) + P_x(\omega^y) \ . \end{aligned}$$

Suppose now $j = k$, and let $\ell = |L_k|$. If $z \in L_k$ then $P_y(\omega^{zx}) = 0$ and $P_x(\omega^z) = P_x(\omega^y)$. Therefore, after using (4) and the inductive assumption for ω^z when $z \in N - L_k$, we have

$$
\begin{aligned}
P_x(\omega) + P_y(\omega^x) &= \frac{1}{n} \sum_{z \in N - L_k} [P_y(\omega^z) + P_x(\omega^{zy})] + \frac{1}{n} \sum_{z \in L_k} P_x(\omega^z) \\
&= P_y(\omega) + \frac{1}{n} \sum_{z \in N - L_k} P_x(\omega^{yz}) + \frac{1}{n} \sum_{z \in L_k} P_x(\omega^y) \\
&= P_y(\omega) + \frac{n-\ell}{n} P_x(\omega^y) + \frac{\ell}{n} P_x(\omega^y) \;=\; P_y(\omega) + P_x(\omega^y) \; .
\end{aligned}
$$

In the last step we applied (4) to $P_x(\omega^y)$ with $M = N(\omega^y) = N - L_k$. □

Lemma 5 *For any offset function ω, the cost of* EQUITABLE *is the same against all lazy adversary strategies for ω.*

Proof: The proof is by induction on $n = |N(\omega)|$. The base case, $n = 0$, is trivial. For the inductive step, let $\omega = (L_1 | \ldots | L_k)$, and suppose we are given two lazy strategies $x\alpha$ and $y\beta$. Let L_i be the first non-revealed layer. Each lazy strategy strategy must contain an item from L_i, and thus, by Lemmas 3 and 4, we can assume that $x, y \in L_i$. Then EQUITABLE has the same cost on x and y, and ω^x and ω^y are identical, up to a symmetry. Since, by the inductive assumption, the cost of EQUITABLE on α and β is the same, we are done. □

Theorem 2 *Algorithm* EQUITABLE *is H_k-competitive.*

Proof: Let $\Phi(\omega)$ be cost of EQUITABLE in a lazy adversary strategy for $\omega = (L_1 | \ldots | L_k)$. This is well defined because of Lemma 5. By its very definition, Φ satisfies inequality (2) when $r \in S(\omega)$.

For $r \notin S(\omega)$, we proceed by induction on k. The base case, $k = 1$, is trivial: $\Delta\Phi = 0 = H_1 - 1$. For $k > 1$, since $\sum_{x \in S(\omega)} P_x(\omega) = \sum_{x \in S(\omega^r)} P_x(\omega^r) = k$, $S(\omega^r) = S(\omega) + r$, and $P_r(\omega^r) = 1$, we have $\sum_{x \in S(\omega)} [P_x(\omega) - P_x(\omega^r)] = 1$. We conclude that there exists an $x \in S(\omega)$ such that

$$
P_x(\omega) - P_x(\omega^r) \;\leq\; \frac{1}{|S(\omega)|} \;\leq\; \frac{1}{k} \; . \tag{5}
$$

We can assume that $x \in L_i$ with $i \neq 1$, for if $x \in L_1$ then $P_x(\omega) = 1$ and $P_x(\omega^r) = P_y(\omega^r)$ for any $y \in L_2$. Let μ be the same as ω^x but with the first layer $\{x\}$ removed. Then μ is an offset function for the cache with $k - 1$ items. We have $\Phi(\mu) = \Phi(\omega^x)$ and, similarly, $\Phi(\omega^{rx}) = \Phi(\mu^r)$. Therefore

$$
\begin{aligned}
\Delta\Phi &= \Phi(\omega^r) - \Phi(\omega) \;=\; [\Phi(\omega^{rx}) + 1 - P_x(\omega^r)] - [\Phi(\omega^x) + 1 - P_x(\omega)] \\
&\leq \Phi(\mu^r) - \Phi(\mu) + \frac{1}{|S(\omega)|} \;\leq\; H_{k-1} - 1 + \frac{1}{k} \;=\; H_k - 1,
\end{aligned}
$$

completing the proof. □

4.1 Time and Space Complexity

Let the behavior version of EQUITABLE, as defined in the proof of Lemma 3, be denoted by $\mathcal{B}$. As defined, the space required by $\mathcal{B}$ is $O(n)$, where n is the number of past requests. We now show

how to improve the space complexity to $O(k^2 \log k)$. Let $M = \lceil 5k^2 H_k \rceil$. Modify $\mathcal{B}$ as follows: if the current offset function ω satisfies $|S(\omega)| = M$, and if X is $\mathcal{B}$'s configuration, then $\mathcal{B}$ executes the following *forgiveness* step: it replaces the offset function ω by the work function $\chi_X - k + 1$, where χ_X is the cone on X. (See [5, 4] for other examples of the forgiveness method.) Note that $\chi_X - k + 1 \leq \omega$, so $\mathcal{B}$ cannot benefit from this action. $\mathcal{B}$ does not incur any cost, but the new offset function is χ_X, which means that the optimal cost of this move is $\Delta opt = -k + 1$.

In order to prove that $\mathcal{B}$ is still H_k-competitive, we use a more subtle potential argument. Define a *stage* to be the sequence of steps in-between two forgiveness moves. Let Φ be the potential introduced in Theorem 2. Initially and after each forgiveness move, let $\Psi = 0$. For each move (other than forgiveness) define the quantity $\Delta\Psi = H_k \Delta opt - \Delta cost - \Delta\Phi$. Since Φ and EQUITABLE satisfy (2), $\Delta\Psi \geq 0$ for all moves in the stage. Therefore, $\Psi \geq 0$ at all times. We will refer to each $\Delta\Psi$ as the *savings* at a given step, and let Ψ stand for the total savings up to a given step of a given stage. By summing over all requests, if Φ and Ψ satisfy

$$\Delta cost + \Delta\Phi + \Delta\Psi \quad \leq \quad H_k \cdot \Delta opt$$

at each step, then $\mathcal{B}$ is H_k competitive. By the very definition of Ψ, the inequality is satisfied for each step other than forgiveness. In the forgiveness step we have $\Delta cost = 0$, $\Delta\Phi \leq 0$, $\Delta opt = -k+1$ and $\Delta\Psi = -\Psi$, so it suffices to show the following claim: When $|S(\omega)| = M$ then $\Psi \geq H_k(k-1)$. To prove the claim, consider a stage ending at ω. For each $m = k, \ldots, M-1$ there was a step in this stage when the offset function μ satisfied $|S(\mu)| = m$ and the new request was $r \notin S(\mu)$. At that time, from the proof of Theorem 2, we know that $\Delta\Psi$ satisfied $\Delta\Psi \geq \frac{1}{k} - \frac{1}{m}$. So at the end of the stage we have

$$\begin{aligned} \Psi \quad &\geq \quad \sum_{m=k}^{M-1} \left(\frac{1}{k} - \frac{1}{m} \right) \quad = \quad \frac{M}{k} - H_{M-1} + H_{k-1} - 1 \\ &\geq \quad kH_k - H_k + 4kH_k - H_{5k^2 H_k} \;\geq\; H_k(k-1), \end{aligned}$$

where the last inequality uses the fact that $\ln k \leq H_k \leq \ln k + 1$.

We now show how $\mathcal{B}$ can be implemented in $O(k^2)$ time per request. Let ω be the current offset function, X be the current configuration for $\mathcal{B}$, and r be the request. Updating ω can be accomplished in time $O(k \log k)$ using appropriate data structures to represent the layers of the offset function. Given an order for the items of X, finding the index j described in Lemma 3 can be easily accomplished in $O(k)$ time once we note that we only need to keep track of the locations of each $x_i \in X$ and the cardinalities of the L_i and not the L_i themselves. So, it suffices to show an $O(k^2)$ method for finding an ordering of X which is consistent with Lemma 3. This will be done by induction on the number of requests since the last time ω was a cone. When ω is a cone, each permutation appears with equal probability. This can be accomplished in time $O(k)$ by uniformly selecting k items from X one at a time. Assume that X is ordered, and that Z is defined as in the proof of Lemma 3. Arrange the elements of Z in the order $\{z_1, ..., z_{i-1}, r, z_i, ...z_{k-1}\}$ with probability proportional to the probability that Z would be chosen by $P(\omega^r)$ in this order.

Finding the probabilities for each of these k orders takes time $O(k)$, giving the overall time $O(k^2)$ for ordering Z. An argument analogous to the proof of Lemma 3 shows that using this method for updating the ordering yields the same distribution at each step as that of EQUITABLE. We conclude this section with the following theorem.

Theorem 3 *Algorithm* EQUITABLE *can be implemented in* $O(k^2 \log k)$ *memory and* $O(k^2)$ *time per request.*

5 Final Comments

Both PARTITION from [13], and the naive implementation of EQUITABLE are very time and space consuming. They both use $O(n)$ space, where n is the number of past requests. We have shown that EQUITABLE can be implemented in space $O(k^2 \log k)$ and time $O(k^2)$ per step, independent of n. Algorithm MARK, although not optimally competitive, uses only $O(k)$ memory and $O(1)$ time for each request. One problem that we leave open is whether there is a simple H_k-competitive algorithm for paging which requires only $O(k)$ memory. Raghavan and Snir in [14] investigated a memory versus randomization trade-off in on-line algorithms. Is there also a trade-off between memory and competitiveness?

It would also be interesting to extend the applications of work functions to other approaches to competitive analysis. The competitive analysis has been criticized for being overly pessimistic. Addressing this problem, the most recent work on the competitive analysis of paging moves towards relaxing the definition of competitiveness. Koutsoupias and Papadimitriou [9] used work functions to analyze paging under two refined notions of competitiveness. The first one restricts the adversary to using a distribution chosen from a given set of distributions. The second compares two classes of algorithms by allowing the server to choose an algorithm from the first set, the adversary to choose an algorithm from the second set, and considering the worst case ratio of costs of the two algorithms.

There is also some recent research in this direction that does not involve work functions. In [8] the paging problem is considered in the case where the requests come from a Markov process. In [16] both the paging and weighted cache problems are considered as linear programming problems and the resulting dual problem is investigated. Several results about the so-called loose competitiveness of these problems are proven. In [1] and [7] the paging problem is addressed with the assumption that after any particular request the next request is likely to be from a small set of nearby items.

We believe that the work function approach can be used to simplify and possibly refine the results from these papers, and ultimately lead to a more systematic and uniform treatment of different approaches to competitive analysis of paging.

References

[1] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proc. 23rd ACM Symposium on Theory of Computing*, pages 249–259, 1991. To appear in *Journal of Computer and System Sciences.*

[2] M. Chrobak and L. L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[3] M. Chrobak and L. L. Larmore. Metrical service systems: Randomized strategies. manuscript, 1992.

[4] M. Chrobak and L. L. Larmore. Generosity helps or an 11-competitive algorithm for three servers. *Journal of Algorithms*, 16:234–263, 1994. Also in Proceedings of ACM/SIAM Symposium on Discrete Algorithms, 1992, 196-202.

[5] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook. Page migration algorithms using work functions. Technical Report YALE/DCS/RR-910, Department of Yale University, 1992. Submitted for journal publication.

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

[7] S. Irani, A. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. In *3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 228–236, 1992.

[8] A. Karlin, S. Phillips, and P. Raghavan. Markov paging. In *Proc. 33rd IEEE Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[9] E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. In *Proc. 25th Symposium on Foundations of Computer Science*, pages 394–400, 1994.

[10] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. In *Proc. 25th Symposium on Theory of Computing*, pages 507–511, 1994.

[11] H. Kuhn. Extensive games and the problem of information. In H. Kuhn and A. Tucker, editors, *Contributions to the Theory of Games*, pages 193–216. Princeton University Press, 1953.

[12] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. Also in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 322-333.

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

[14] P. Raghavan and M. Snir. Memory versus randomization in online algorithms. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 687–703. Springer-Verlag, 1989.

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

[16] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 1991. To appear. Rewritten version of "On-line caching as cache size varies", in The 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, 241-250, 1991.