More on Randomized On-line Algorithms for Caching

Marek Chrobak* Elias Koutsoupias† John Noga‡

Abstract

We address the tradeoff between the competitive ratio and the resources used by randomized on-line algorithms for caching. Two algorithms reported in the literature that achieve the optimal ratio H_k require a lot of memory and perform extensive computation at each step. On the other hand, a very simple algorithm called RMARK has competitive ratio $2H_k - 1$, within a factor of 2 of the optimum. A natural question that arises here is whether there is a tradeoff between simplicity and the competitive ratio. In particular, is it possible to achieve a competitive ratio better than $2H_k - 1$ with a simple algorithm like RMARK?

We first consider *marking algorithms* that are natural generalizations of RMARK, and we prove that, for any $\epsilon > 0$, there is no randomized marking algorithm for caching with competitive ratio $(2 - \epsilon)H_k$. Thus RMARK is essentially optimal among marking algorithms.

Another model of simple caching algorithms is that of *trackless algorithms*. These are algorithms that do not store any information about items that are not in the cache. It is known that, for $k = 2$, there is no randomized trackless algorithm for caching with ratio better than $37/24 \approx 1.5416$. The trivial upper bound is 2, achieved even by deterministic algorithms LRU and FIFO. We reduce this gap by giving a trackless randomized algorithm with competitive ratio $\frac{1}{4}(3 + \sqrt{13}) \approx 1.6514$.

1 Introduction

In the *caching problem* we have a two-level memory system consisting of a cache of size k and an unbounded main memory. At each step, a request to an item is issued. If a requested item is not in the cache, a *fault* occurs. On a fault, the requested item needs to be brought into the cache and thus one of the cached items needs to be evicted. The choice of the evicted item is made *on-line*, i.e., before the next request is issued. Our objective is to minimize the number of faults.

It is quite easy to see that no on-line caching algorithm can achieve a minimum cost on all request sequences. On-line algorithms are commonly evaluated using the performance measure called the

*Department of Computer Science, University of California, Riverside, CA 92521. Email: marek@cs.ucr.edu. Research supported by NSF grant CCR-9988360.

†Department of Computer Science, University of California, Los Angeles, CA 90095. Email: elias@cs.ucla.edu. Research supported by NSF grant CCR-0105752.

‡Department of Computer Science, California State University, Northridge, CA 91330. Email: jnoga@ecs.csun.edu. Research supported by NSF grant CCR-9988360.

competitive ratio. An on-line algorithm $\mathcal{A}$ is said to be *c-competitive* if, on every request sequence ϱ, its cost is, asymptotically, at most c times the optimal cost for this sequence. More precisely, for each ϱ, $\mathcal{A}$ must satisfy

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

where $cost_{\mathcal{A}}(\varrho)$ is the cost of $\mathcal{A}$ on ϱ, $opt(\varrho)$ is the optimal (off-line) cost on ϱ, and a is a constant independent of ϱ. The *competitive ratio* of $\mathcal{A}$ is the smallest c for which $\mathcal{A}$ is c-competitive.

Caching 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, [8, 10, 6]) in a uniform metric space. In the deterministic case, it has been established that several well-known strategies, including LRU and FIFO, are k-competitive, and that no better competitiveness is possible (see [12]).

In this paper we concentrate on *randomized* algorithms for caching. It is relatively easy to show (see [7]) that no randomized on-line algorithm can be better than H_k-competitive, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th harmonic number. Fiat *et al.* [7] gave a simple algorithm called RMARK which is $2H_k$-competitive. RMARK works as follows. Each requested item is marked. On a fault, the algorithm evicts a random, uniformly chosen non-marked item from the cache (in case when all cached items are marked, they are all unmarked first). Later, Achlioptas *et al.* [1] proved the competitive ratio of RMARK is exactly $2H_k - 1$.

Two algorithms with the optimal ratio H_k were reported in the literature. The first algorithm, called PARTITION, was discovered and analyzed by McGeoch and Sleator [11], the other, called EQUITABLE, appeared in [1]. Both algorithms store a large amount of information about past requests and they perform extensive computation at each step. Thus it is natural to ask whether there is a simple algorithm like RMARK with competitive ratio equal or close to H_k. Or, is there a tradeoff between simplicity and the competitive ratio?

Capturing the intuitive notion of simplicity with a formal mathematical definition is itself an interesting and challenging problem. The intuition tells us that RMARK is simple, while PARTITION and EQUITABLE are not. One way to address this question would be to simply limit the memory and running time of the algorithm. One step in this direction was made in [1]. Algorithm PARTITION from [11] uses $O(n+k)$ memory, where n is the number of requests. In [1], the authors show that their algorithm EQUITABLE can be implemented with only $O(k^2)$ memory, so its memory size is independent of the number of requests.

Another natural restriction is that a simple algorithm should not keep track of any information associated with the items that are not currently in the cache. This concept has been introduced by Bein and Larmore [3] who proposed the term *trackless* for algorithms that satisfy this property. In the context of caching, a trackless algorithm does not know the "identities" of the requested items. When a fault occurs, it only knows that the requested item is not in the cache. On a hit, it knows the cache location of the requested item. LRU, FIFO and RMARK are trackless, while algorithms PARTITION and EQUITABLE from [11, 1] are not. From a purely practical standpoint, non-trackless algorithms are of limited interest as cache replacement strategies, as they cannot be realistically implemented. Bein *et al.* [3, 2] proved that there is no on-line randomized trackless algorithm for caching for $k = 2$ with competitive ratio smaller than $37/24 \approx 1.5416$. Using linear programming software, they were also able to show that this competitive ratio must be at least $8453/5458 \approx 1.5487$.

Our results. We first consider *marking algorithms*. Similar to RMARK, such algorithms maintain a set of marks on some items in the cache. Each requested item is marked, and whenever a fault occurs with all cached items being marked, the algorithm unmarks all items. The only restriction posed on the algorithm is that on a fault only non-marked items can be evicted. In particular, RMARK is a marking algorithm that evicts a random, uniformly chosen, non-marked item.

Our definition of marking algorithms does not involve any assumptions on the probability distribution of evictions, the running time, nor on the information about the past maintained by the algorithm. Note that this definition covers some algorithms that are not necessarily simple, including those that store and use a complete history of the past computation.

The main result of this paper, presented in Section 3, is that no marking algorithm can achieve competitive ratio $(2-\epsilon)H_k$ for $\epsilon > 0$. Thus RMARK is essentially optimal among marking algorithms.

Other mark-based (or phase-based) randomized algorithms have been studied in the literature (see [14], for example), typically giving upper bounds on the competitive ratio that are a factor of 2 away from the corresponding lower bound. Our result provides a strong evidence that this gap is an inherent feature of the phase-based approach, and that in order to obtain tighter bounds a different framework is necessary.

Next, we consider trackless algorithms for $k = 2$. For this case we give a trackless algorithm $\mathsf{TL2}$ with competitive ratio $\frac{1}{4}(3 + \sqrt{13}) \approx 1.6514$, substantially improving the trivial upper bound of 2 achieved by LRU, FIFO and RMARK.

2 Preliminaries

Throughout the paper, by k we denote the cache size. By a *cache configuration*, or simply *configuration*, we will mean a set of k items representing the cache content.

Phases. Any request sequence can be decomposed into *phases* as follows. The first phase starts at the beginning of the request sequence, and each other phase starts on the first request after the previous phase. Each phase is a longest sequence of consecutive requests that contains at most k distinct requests. (Thus all phases, except possibly the last, will contain *exactly* k distinct items.)

The relationship between the phase decomposition and marking should be clear: a marking algorithm will keep marks on the items requested in the current phase. Thus we can alternatively define a marking algorithm as an algorithm that never evicts items from the current phase.

Randomized algorithms. There are two ways to define a randomized algorithm [4]. A randomized *behavioral* algorithm can make random choices at each step of the computation. A randomized *distribution* algorithm (also called a *mixed strategy*) is simply a probability distribution on the deterministic algorithms. In the general setting, when no restrictions are placed on the algorithms, the two models are equivalent, that is, a randomized algorithm of one type can be converted into an algorithm of the other type without increasing the expected cost. This equivalence also holds for marking algorithms (since we do not impose any restriction on the algorithm's memory). However, it does not extend to other special classes of on-line algorithms. For example, it is easy to see that behavioral randomized memoryless algorithms are not equivalent to probability distributions on deterministic memoryless algorithms [4]. Our trackless algorithm TL2 is a behavioral algorithm.

Optimal cost. For competitive analysis, we must be able to keep track of the optimal cost during the computation. There are two basic ways to do so. In the *adversary method*, we view the computation as a game between our algorithm and an adversary who must serve all requests with his own cache. We will use this approach in the lower bound proof in Section 3. The proof is obtained by showing an adversary strategy in which the ratio between our algorithm's cost and the adversary cost is at least the claimed lower bound.

Another method to keep track of the optimal cost is to use *work functions*. A work function ω at a given step determines, for each cache configuration, the optimal cost of serving past requests so that this configuration is reached. We will use work functions in the upper bound proofs for $k = 2$.

Work functions for caching were characterized by Koutsoupias and Papadimitriou in [9]. For $k = 2$, work functions have a very simple form. Let x be the last request. Then there is an integer $a \geq 0$ and a finite set of items Y with $x \notin Y$ such that

$$\begin{aligned}
\omega(x, y) &= a && \text{for} \quad y \in Y \\
\omega(x, v) &= a + 1 && \text{for} \quad v \notin Y \\
\omega(u, y) &= a + 1 && \text{for} \quad u \neq x \ \&\ y \in Y \\
\omega(u, v) &= a + 2 && \text{for} \quad u, v \notin Y \cup \{x\}
\end{aligned}$$

The set of pairs (x, y), for $y \in Y$, is called the *support* of ω. Sometimes, informally, we also refer to Y as the support set.

For convenience, we will offset a from ω and assume that ω is 0 in the support. This function is called an *offset function* and is denoted by $\langle x | y_1 y_2 \dots y_m \rangle$, where $Y = \{y_1, y_2, \dots, y_m\}$. The value of a represents the optimal cost on past requests and ω represents the current state of the adversary (all possible configurations with their differential costs). Offset functions can be updated as follows. On request x, the offset function does not change and the optimal cost is 0. When some $y_i \in Y$ is requested, the cost of (x, y_i) remains 0, the cost of (x, z) and (y_i, z), for $z \neq x, y_i$, is 1, and all other configurations have cost 2. Thus the adversary cost is 0 and the offset function changes to $\langle y_i | x \rangle$. By a similar argument, when some $v \notin Y \cup \{x\}$ is requested, the adversary cost is 1 and the offset function changes to $\langle v | x y_1 \dots y_m \rangle$. See [9] for details.

3 Lower Bound for Randomized Marking Algorithms

Recall that an on-line caching algorithm is called *marking* if it never evicts items that have been requested in the current phase. We prove in this section that no marking algorithm can have competitive ratio $(2-\epsilon)H_k$, for any $\epsilon > 0$.

We first define a certain random process $\{\mu_t\}$ and prove a technical lemma about its distribution. This random process starts at 0, and proceeds according to the following rules. Let $\delta > 0$ be an even integer, and suppose we start with $\delta/2$ red balls and δ white balls in an urn. At each time step a ball is selected without replacement. If it is red we increase our position by 1, if it is white we decrease our position by 1. More formally, let $\mu_0 = 0$, and for $t = 1, 2, \ldots, 3\delta/2$ let $\mu_{t+1} = \mu_t + 1$ with probability $p_t = (\delta - \mu_t - t)/(3\delta - 2t)$ and $\mu_{t+1} = \mu_t - 1$ with probability $1 - p_t$.

Lemma 1 $\mathrm{Exp}[\max_t \mu_t] = O(1)$.

Proof: To prove the lemma, we compare μ to a standard biased random walk ν. Let $\nu_0 = 0$, and for $t \geq 0$ let $\nu_{t+1} = \nu_t + 1$ with probability $\frac{2}{5}$ and $\nu_{t+1} = \nu_t - 1$ with probability $\frac{3}{5}$. It is known (see [13]) that the probability that ν will ever reach the point $i > 0$ (even in an arbitrarily large number of steps) is $(2/3)^i$. Therefore $\mathrm{Exp}[\max_t \nu_t] \leq \sum_{i=1}^{\infty} i(2/3)^{i+1} = 4$. So to prove the lemma it suffices to show the following inequality:

$$\mathrm{Exp}[\max_t \mu_t] \quad \leq \quad \mathrm{Exp}[\max_t \nu_t] + O(1). \tag{2}$$

The maximum of μ must occur sometime during the first δ steps, since after this point at least $\delta/2$ white balls must have been selected. Thus we can restrict our consideration to $t \leq \delta$.

We now present a different (but equivalent) way to describe μ and ν, which will make it easier to compare their maxima. At step t, we draw a random number $x \in [0,1]$ and the two processes change their positions as follows:

$$(\mu_{t+1}, \nu_{t+1}) \quad = \quad \begin{cases} (\mu_t + 1, \nu_t + 1) & 0 \leq x < \min(p_t, \frac{2}{5}) \\ (\mu_t - 1, \nu_t + 1) & p_t \leq x < \frac{2}{5} \\ (\mu_t + 1, \nu_t - 1) & \frac{2}{5} \leq x < p_t \\ (\mu_t - 1, \nu_t - 1) & \max(p_t, \frac{2}{5}) \leq x \leq 1 \end{cases}$$

It is important to note that although this way of describing the processes causes the two random processes to be correlated, it does not change their individual distributions. In particular, the expected values of $\max_t \mu_t$ and $\max_t \nu_t$ remain the same.

We first show that, in this new process, $\Pr[\exists t : \mu_t > \nu_t] = o(1/\delta)$. If the event $\mu_t > \nu_t$ occurs for some t, then there is an $s < t$ for which $\nu_s = \mu_s$ and $p_s > \frac{2}{5}$. If $\nu_s = \mu_s$ and $p_s > \frac{2}{5}$ then $\nu_s < -(\delta + s)/5$. Since $|\nu_s| \leq s \leq \delta$, we get $\nu_s \leq -2s/5$ and $s \geq \delta/4$. There is a constant $a > 0$ such that $\Pr[\nu_s \leq -2s/5] \leq 2e^{-as}$ (see [13]). Note that this only requires that ν is a biased random walk and does not require independence from μ. Summing over $s = \delta/4, \ldots, \delta$, we get $\Pr[\exists t : \mu_t > \nu_t] = o(1/\delta)$, as desired.

Now, using the inequalities $\max_t \mu_t \leq \max_t \nu_t + \max_t(\mu_t - \nu_t)$ and $\mu_t - \nu_t \leq \delta$ which hold with probability 1, and the bound from the previous paragraph, we have $\text{Exp}[\max_t \mu_t] \leq \text{Exp}[\max \nu_t] + \text{Exp}[\max_t(\mu_t - \nu_t)] \leq \text{Exp}[\max \nu_t] + \delta \Pr[\exists t : \mu_t > \nu_t] \leq \text{Exp}[\max \nu_t] + O(1)$. This completes the proof of (2). □

Theorem 1 *For any $\epsilon > 0$, no randomized on-line marking algorithm for caching can be $(2-\epsilon)H_k$-competitive if k is large enough.*

Proof: We show a probability distribution on request sequences for which (i) the expected cost of any on-line marking algorithm $\mathcal{A}$ is at least $2H_k - o(\log k)$ times more than the expected adversary cost, and (ii) the expected adversary cost is unbounded. Using Yao's minimax principle (see, for example, [4]), these two properties imply the theorem.

Let δ be an even integer such that $\delta = o(\log k)$ and $\delta = \omega(1)$. The adversary uses a set X of $k+\delta$ items. The request sequence consists of phases, with each phase having exactly k distinct requests. Let Δ_i denote the δ items not requested in phase i. In phase i, we first make a random, uniformly chosen request from Δ_{i-1}, followed by $k-1$ requests, each chosen uniformly and at random from those points in X which have not yet been requested in this phase.

We estimate $\mathcal{A}$'s cost in a phase. The first request is a fault, and for $j > 1$ the probability that $\mathcal{A}$ will fault on the jth request in this phase is $\delta/(k+\delta-j+1)$, since there are $j-1$ marked items in $\mathcal{A}$'s cache and, among the remaining $k+\delta-j+1$ items that are candidates for the next request, δ of them are not in the cache. Thus the expected cost incurred by $\mathcal{A}$ in each phase is $1+\delta(H_{k+\delta-1} - H_\delta)$.

We now show that the adversary can serve each phase with an expected cost of $\delta/2 + O(1)$. This will imply the theorem, because

$$\frac{1+\delta(H_{k+\delta-1} - H_\delta)}{\delta/2 + O(1)} \quad = \quad 2H_k - o(\log k),$$

by the choice of δ.

The adversary maintains the invariant that, at the beginning of phase i, the adversary's cache contains exactly $\delta/2$ items from Δ_i. To preserve this invariant during phase i the adversary uses information about the future, namely about Δ_{i+1}.

Let F_i be the $\delta/2$ items in $X - \Delta_i$ that are not in the adversary's cache at the beginning of phase i. In phases where Δ_{i+1} intersects $F_i \cup \Delta_i$, serve the faults evicting arbitrary items, and at the end of such phases, restore the invariant by loading or evicting the appropriate number of items from Δ_{i+1}. The probability that Δ_{i+1} intersects $F_i \cup \Delta_i$ in a given phase is $O(1/\delta)$ and the cost is $O(\delta)$.

We now consider phases where Δ_{i+1} does not intersect $F_i \cup \Delta_i$. When a fault occurs, serve the request by evicting an item from Δ_{i+1}. Whenever possible, choose an arbitrary item from Δ_{i+1} which has already been requested in this phase; otherwise, from the items in the adversary's cache which are in Δ_{i+1} choose the one which will be requested latest in this phase. By this strategy, after the phase the adversary will have $\delta/2$ items from Δ_{i+1} in the cache.

The adversary faults on the $\delta/2$ items in F_i, plus possibly on some items in Δ_{i+1}. It remains to estimate the number of faults in Δ_{i+1}. The number of these faults is precisely the number of times the adversary cannot serve a request in F_i with a previously requested item from Δ_{i+1}. We claim that this number is $O(1)$.

We consider the sequence of the $3\delta/2$ requests in $\Delta_{i+1} \cup F_i$. For $t = 1, 2, \ldots, 3\delta/2$, denote by μ_t the difference between the number of requests in F_i and the number of requests in Δ_{i+1} in the first t steps. Then the number of faults in Δ_{i+1} is the maximum of μ_t. Think of items in F_i as red balls and items in Δ_{i+1} as white balls. Then μ is the same random process as the one defined earlier in this section and, by Lemma 1, we get that the expected cost on Δ_{i+1} is $O(1)$.

Summarizing, the expected cost of the adversary in a phase is no more than $\delta/2 + O(1) + O(\delta)O(1/\delta) = \delta/2 + O(1)$. This completes the proof. □

4 A Trackless Algorithms for $k = 2$

Recall that a *trackless* on-line algorithm is defined as follows: at each step, the algorithm is told whether a fault occurred or not. If a hit occurred, it knows the cache location that contains the requested item. It does not have access to any other information.

We now present a trackless algorithm for 2 servers that we call TL2. The algorithm maintains two types of marks associated with the cached item that is not the last request. We represent cache configurations by *ordered pairs*, with the last request listed first. The possible cache configurations are (x, y), $(x, \dot{y})$ and $(x, \ddot{y})$, where x is the last request and y is the other item in the cache. The number of dots over an item is an estimate of the uncertainty about whether this item should be be in the cache at a given time. This number represents – roughly, but not exactly – the number of faults after this item was last requested.

Algorithm TL2: Let p be a parameter, $p \in [\frac{1}{2}, 1]$, whose value will be specified later. Suppose the items in the cache are x, y, where x is the last request, and that we request z. If $z = x$, nothing happens. If $z = y$, the new configuration is (y, x). If z is not in the cache, we have three cases:

(tl2a) If the cache is (x, y), we go to $(z, \dot{x})$ or $(z, \dot{y})$, each with probability $\frac{1}{2}$.

(tl2b) If the cache is $(x, \dot{y})$, we go to (z, x) with probability p and to $(z, \ddot{y})$ with probability $1 - p$.

(tl2c) If the cache is $(x, \ddot{y})$, we go to (z, x) with probability 1.

As explained in Section 2, we use notation $\langle x | y_1 y_2 \ldots y_m \rangle$ for offset functions, where x is the last request, and the support is (x, y_1), … ,(x, y_m). The items $x, y_1 \ldots, y_m$, are listed in order from most to least recently requested.

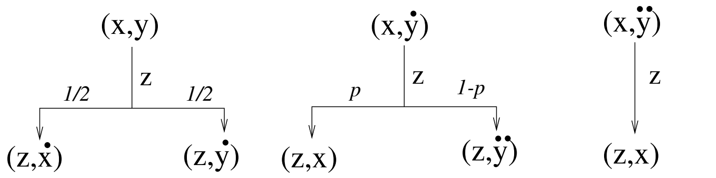

Figure 1: The moves of Algorithm TL2 on a fault.

We also introduce some notation for probability distributions on the algorithm's configurations. Notation x^α means that x is in the cache with probability α. If the last request is x and y_j is in the cache with probability p_j, for $j = 1, \ldots, k$ (and other items have probability 0), then we represent this distribution by $(x, y_1^{p_1} y_2^{p_2} \ldots y_k^{p_k})$, where the y_j are listed in order, from most to least recently requested. Some of the y_j in this notation may be marked. For example, $(x, y^\alpha \dot{y}^\beta \ddot{z}^\gamma)$ is the distribution in which the last request is x and the other item in the cache is y (with no mark) with probability α, $\dot{y}$ (y with the single mark) with probability β, and $\ddot{z}$ (z with the double mark) with probability γ.

For $0 \leq \alpha, \beta, \gamma, \delta \leq 1$, define

$$\Phi_j \;=\; \Phi_j(\alpha, \beta, \gamma, \delta) \;=\; \begin{cases} (2-p)\beta + \gamma + \delta & j = 1 \\ (2-p/2)\alpha + (2-p)\beta + \delta & j = 2 \\ (2-p/2)\alpha + 2(2-p)\beta + \gamma & j = 3 \\ (2-p/2)\alpha + 2(2-p)\beta + \gamma + \delta & j \geq 4 \end{cases} \tag{3}$$

We now state a lemma needed for the analysis of Algorithm TL2. The proof of this lemma is given in the appendix.

Lemma 2 (a) *Let x be the last request and $y_1, y_2, \ldots$ be the other items listed from most to least recently requested. Then the distribution of* TL2 *has the form*

$$\left(x\,,\, y_1^\alpha \dot{y}_1^\beta \dot{y}_2^\beta \ddot{y}_2^\gamma \ddot{y}_3^\delta\right) \tag{4}$$

where $\alpha \geq \gamma$, and $\beta + \gamma \geq \delta$.

(b) *For any $j \geq 1$, the expected cost of Algorithm* TL2 *starting from distribution* (4) *on the request sequence $(y_j x)^*$ equals Φ_j.*

Theorem 2 *For $p = \frac{1}{2}(5 - \sqrt{13})$, Algorithm* TL2 *has competitive ratio* $\frac{1}{4}(3 + \sqrt{13}) \approx 1.6514$.

Proof: The proof is by amortized analysis, using a potential argument. Each state is determined by the current distribution of TL2 and the current offset function. By Lemma 2, the distribution of TL2 has the form $\left(x, y_1^\alpha \dot{y}_1^\beta \dot{y}_2^\beta \ddot{y}_2^\gamma \ddot{y}_3^\delta\right)$. Using the constraints on $\alpha, \beta, \gamma, \delta$ in Lemma 2.a, we get $\Phi_j \leq \Phi_{j+1}$ for all j. If the support is $y_1, \ldots, y_m$, we take the potential of this state to be $\Phi = \Phi_m = \max_{j \leq m} \Phi_j$.

We now consider one step of the computation. Denote by $\Delta cost$, Δopt and $\Delta\Phi$, respectively, the cost of $\mathsf{TL2}$, the optimal cost, and the potential change in this move. We need to show that

$$\Delta cost + \Delta\Phi \quad \leq \quad \tfrac{1}{4}(3+\sqrt{13})\Delta opt. \tag{5}$$

The theorem follows from (5) by routine amortization.

Observe that Φ is the "lazy potential" function, equal to the maximum cost of $\mathsf{TL2}$ if the adversary repeats requests on x and some y_j, for $j \leq m$. Thus Φ satisfies (5) for requests in the support, since then $\Delta cost \leq -\Delta\Phi$ and $\Delta opt = 0$.

For a request outside the support, assume (without loss of generality) that the requested point y_j, with $j > m$, had probability zero. The new offset function is $\langle y_j | x y_1 \ldots y_m \rangle$ and the distribution is

$$\left(y_j \,,\, x^{2p\beta+\gamma+\delta} \dot{x}^{\frac{1}{2}\alpha} \dot{y}_1^{\frac{1}{2}\alpha} \ddot{y}_1^{(1-p)\beta} \ddot{y}_2^{(1-p)\beta} \right)$$

(See (8) in the appendix.) Then, depending on the support of the original configuration, we get

$$\begin{aligned}
\Delta\Phi &= \Phi_{m+1}(\,2p\beta+\gamma+\delta, \tfrac{1}{2}\alpha, (1-p)\beta, (1-p)\beta\,) \;-\; \Phi_m(\alpha,\beta,\gamma,\delta) \\
&= -\tfrac{1}{2}p\alpha + (4p-p^2-2)\beta + \tfrac{1}{2}(2-p)(\gamma+\delta) + \begin{cases} \alpha+\beta & m=1 \\ \beta+\gamma & m=2 \\ \delta & m=3 \end{cases}
\end{aligned}$$

By Lemma 2, $\Delta\Phi$ is maximized for $m=1$, so we get

$$\begin{aligned}
\Delta\Phi &\leq \left(1-\tfrac{1}{2}p\right)(\alpha+\gamma+\delta) + (4p-p^2-1)\beta \\
&= 1-\tfrac{1}{2}p + (5p-p^2-3)\beta
\end{aligned}$$

Note that p is the root of $5p-p^2-3=0$ in $[0,1]$. Our cost is at most 1 and the optimal cost is 1, so $\Delta cost + \Delta\Phi \leq 2 - \frac{1}{2}p = \frac{1}{4}(3+\sqrt{13})\Delta opt$, completing the proof of (5). $\Box$

The analysis is tight. If we start from (x,y) and request $zxzx...$, the cost of Algorithm $\mathsf{TL2}$ is $\frac{1}{2}(4-p) = \frac{1}{4}(3+\sqrt{13})$ and the optimal cost is 1. Another strategy is to request $uzxzx....$ Then the cost is $\frac{1}{2}(5+3p-p^2) = \frac{1}{2}(3+\sqrt{13})$, and the optimal cost is 2, so the ratio is also $\frac{1}{4}(3+\sqrt{13})$. Therefore our analysis above is tight.

5 Final Comments

What is the optimal competitive ratio of trackless algorithms for $k=2$? We know now that this ratio is between 1.5416 and 1.6514. We believe that Algorithm $\mathsf{TL2}$ from Section 4 can be further improved as follows: in state (x,y) on a fault, instead of evicting each item with equal probability, evict y with probability $q > \frac{1}{2}$. Unfortunately, the lazy potential function does not work well for this extension (the resulting formulas give the optimal value for q equal $\frac{1}{2}$), and so far we have not been able to find a better potential function.

For $k \geq 3$, no upper bounds better than $2H_k - 1$ are known for trackless algorithms. It is also not known whether it is possible to obtain the optimal ratio H_k with a trackless algorithm. We conjecture that there exists a trackless algorithm with competitive ratio $(2-\epsilon)H_k$, for some $\epsilon > 0$.

None of the two restrictions discussed in the paper, trackless or marking, prevents an algorithm from storing large amounts of information about the past. A trackless algorithm, for example, can remember whether a fault or a hit occurred at each step of the computation. It is not known to what degree such information can help in reducing the competitive ratio. To investigate this question, we can consider another model that puts an explicit bound on the memory of the algorithm. Define an *m-state algorithm* to be a probabilistic automaton with m memory states and actions (evictions) associated with transitions. The inputs are "fault" or "hit j", where j is a cache location. On a hit, the automaton specifies the new state. On a fault, the automaton specifies the new state and the item to be evicted. Our trackless algorithm TL2 for $k = 2$ can be implemented with only five states. We can show (see [5]) that it is not possible to achieve a ratio better than 2 with two states, but we can achieve ratio $5/3 \approx 1.667$ with three states. It would be interesting to determine whether the optimal ratio for trackless algorithms can be achieved with some fixed number of states.

Acknowledgements. We would like to thank the anonymous referees for insightful comments that helped us improve the presentation of the paper.

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234:203–218, 2000.

[2] Wolfgang Bein, Rudolph Fleischer, and Lawrence L. Larmore. Limited bookmark randomized online algorithms for the paging problem. *Information Processing Letters*, 76:155–162, 2000.

[3] Wolfgang Bein and Lawrence L. Larmore. Trackless online algorithms for the server problem. *Information Processing Letters*, 74:73–79, 2000.

[4] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[5] Marek Chrobak, Elias Koutsoupias, and John Noga. More on randomized on-line algorithms for caching: simplicity vs competitiveness. Technical Report UCR-CSE-01-02, Department of Computer Science and Engineering, University of California, Riverside, 2001.

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

[7] Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[8] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42:971–983, 1995.

[9] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[10] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[11] Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

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

[13] Frank Spitzer. *Principles of Random Walks.* Springer Verlag, 1976.

[14] Neal E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000.

A Proof of Lemma 2

(a) The proof is by induction. The lemma holds trivially for the initial distribution (x, y^1). Suppose the current distribution satisfies the lemma. Requesting x does not change anything, so we can assume the request is on some y_j. The table below shows the distribution achieved on each possible request from each possible current configuration:

Probability	Current	After y_1	After y_2	After y_j, $j \geq 3$
α	(x, y_1)	(y_1, x)	$(y_2, \dot{x}^{\frac{1}{2}}\dot{y}_1^{\frac{1}{2}})$	$(y_j, \dot{x}^{\frac{1}{2}}\dot{y}_1^{\frac{1}{2}})$
β	$(x, \dot{y}_1)$	(y_1, x)	$(y_2, x^p\ddot{y}_1^{1-p})$	$(y_j, x^p\ddot{y}_1^{1-p})$
β	$(x, \dot{y}_2)$	$(y_1, x^p\ddot{y}_2^{1-p})$	(y_2, x)	$(y_j, x^p\ddot{y}_2^{1-p})$
γ	$(x, \ddot{y}_2)$	(y_1, x)	(y_2, x)	(y_j, x)
δ	$(x, \ddot{y}_3)$	(y_1, x)	(y_2, x)	(y_j, x)

The distribution after requesting y_j is obtained by combining the configurations in the column of y_j, weighted by the row probabilities. The resulting distributions are:

$$\text{after } y_1 \quad : \quad \left(y_1\,,\, x^{\alpha+(1+p)\beta+\gamma+\delta}\ddot{y}_2^{(1-p)\beta}\right) \tag{6}$$

$$\text{after } y_2 \quad : \quad \left(y_2\,,\, x^{(1+p)\beta+\gamma+\delta}\dot{x}^{\frac{1}{2}\alpha}\dot{y}_1^{\frac{1}{2}\alpha}\ddot{y}_1^{(1-p)\beta}\right) \tag{7}$$

$$\text{after } y_j \quad : \quad \left(y_j\,,\, x^{2p\beta+\gamma+\delta}\dot{x}^{\frac{1}{2}\alpha}\dot{y}_1^{\frac{1}{2}\alpha}\ddot{y}_1^{(1-p)\beta}\ddot{y}_2^{(1-p)\beta}\right), \quad \text{for } j \geq 3 \tag{8}$$

In each case, this new distribution has the form as claimed in the lemma.

It remains to show that $\alpha' \geq \gamma'$ and $\beta' + \gamma' \geq \delta'$, where $\alpha', \beta', \gamma', \delta'$ are the parameters of the new distribution. The verification of these inequalities is routine. For example, for $j \geq 3$ we have $\alpha' = 2p\beta + \gamma + \delta$, $\beta' = \frac{1}{2}\alpha$, $\gamma' = (1-p)\beta$, and $\delta' = (1-p)\beta$. Then $\beta' + \gamma' \geq \delta'$ is trivial, and $\alpha' \geq \gamma'$ follows from $p \geq \frac{1}{2}$.

(b) We have four cases. On y_1 the cost of Algorithm $\mathsf{TL2}$ is $\beta + \gamma + \delta$ and its distribution changes to (6). Then, on x, the cost is $(1-p)\beta$, and the distribution changes to (x, y_1^1), and from now on Algorithm $\mathsf{TL2}$ incurs no cost. So the total cost is $\beta + \gamma + \delta + (1-p)\beta = (2-p)\beta + \gamma + \delta$.

On y_2 the cost is $\alpha + \beta + \delta$ and the distribution changes to (7). Starting at (7), we apply the formula for $j = 1$, so the cost is $(\alpha + \beta + \delta) + (2-p)\frac{1}{2}\alpha + (1-p)\beta = (2-p/2)\alpha + (2-p)\beta + \delta$.

On y_3 the cost is $\alpha + 2\beta + \gamma$ and the distribution changes to (8). Starting at (8), we apply the formula for $j = 1$, so the cost is $(\alpha + 2\beta + \gamma) + (2-p)\frac{1}{2}\alpha + 2(1-p)\beta = (2-p/2)\alpha + 2(2-p)\beta + \gamma$.

On y_j the cost is $1 = \alpha + 2\beta + \gamma + \delta$ and the distribution changes to (8). Starting at (8), we apply the formula for $j = 1$, so the cost is $(\alpha+2\beta+\gamma+\delta)+(2-p)\frac{1}{2}\alpha+2(1-p)\beta = (2-p/2)\alpha+2(2-p)\beta+\gamma+\delta$.