More on Randomized On-line Algorithms for Caching: Simplicity vs Competitiveness

Marek Chrobak[1] Elias Koutsoupias[2] John Noga[3]

Technical Report UCR-CS-01-02
Department of Computer Science
University of California
Riverside

[1]Department of Computer Science, University of California, Riverside, CA 92521. Email: marek@cs.ucr.edu. Research supported by NSF grant CCR-9988360.
[2]Department of Computer Science, University of California, Los Angeles, CA 90095. Email: elias@cs.ucla.edu.
[3]Department of Computer Science, University of California, Riverside, CA 92521. Email: jnoga@cs.ucr.edu. Research supported by NSF grant CCR-9988360.

Abstract

We address the tradeoff between the competitive ratio and the resources used by randomized on-line algorithms for caching. Two algorithms reported in 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 $(2 - \epsilon)H_k$.

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 $37/24$. The best known upper bound is 2. We reduce this gap by giving a trackless algorithm with ratio $\frac{1}{4}(3 + \sqrt{13}) \approx 1.651$. We also give some upper and lower bounds for *finite-state* algorithms.

0.1 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. Online algorithms are commonly evaluated using the performance measure called the *competitive ratio*. An on-line algorithm $\mathcal{A}$ 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. 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, 5]), in which all distances are equal to one. 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 [6]) 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.* [6] 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 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 which item in the cache was requested. LRU, FIFO and RMARK are trackless, while algorithms PARTITION and EQUITABLE from [11, 1] are not. From purely practical standpoint, non-trackless algorithms are of limited interest as cache replacement strategies, as they cannot be realistically implemented. (However, they may be useful as page replacement strategies for swapping pages between the disk and random-access memory.) 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 $37/24 \approx 1.5416$. They also provide computational results, using linear programming software, showing 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.

The main result of this paper, presented in Section 0.3, is that no marking algorithm can achieve competitive ratio $(2 - \epsilon)H_k$ for $\epsilon > 0$. Note that 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. Thus our lower bound applies even to marking algorithms that are not necessarily simple, including those that store a complete history of the past computation.

Other mark-based (or phase-based) randomized algorithms have been studied in 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 strong evidence that this gap is an inherent feature of the phase-based approach, and that in order to obtain tight bounds a different framework is necessary.

Next, we consider trackless algorithms for $k = 2$. For this case we give a trackless algorithm with competitive ratio $\frac{1}{4}(3 + \sqrt{13}) \approx 1.6514$, improving the previously known upper bound of 2.

None of the two restrictions discussed above, trackless or marking, prevents an algorithm from storing and processing 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 propose another model that puts an explicit bound on the memory of the algorithm. An *m-state algorithm* is 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 for $k = 2$ mentioned in the previous paragraph has 6 states. For two states we give a lower bound of 2. For three states we give an algorithm with competitive ratio $5/3 \approx 1.667$.

0.2 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. We will assume that the initial configuration X_0 is fixed. As explained earlier in the introduction, an on-line caching algorithm $\mathcal{A}$ needs to respond to every request p before the next request is issued. If a fault 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 fault has unit cost.

Online algorithms. 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 fault occurred or not. We will charge $\mathcal{A}$ a cost of 1 for each such swap. It is easy to see that, without loss of generality, $\mathcal{A}$ will never bring an item into the cache unless it is the current request.

Competitiveness. Denote by $cost_{\mathcal{A}}(\varrho)$ the cost of $\mathcal{A}$ on a request sequence ϱ, and by $opt(\varrho)$ the optimal (off-line) cost on ϱ. We will say that $\mathcal{A}$ is *c-competitive* if there is a constant $a > 0$ such that for every request sequence ϱ

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

In all 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.

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. A randomized *behavioral* algorithm can make random choices at each step of the computation. A randomized *distribution* algorithm 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. However, this equivalence does not extend to some special classes of algorithms. For example, it is easy to see that behavioral randomized memoryless algorithms are not equivalent to probability distributions on deterministic memoryless algorithms. For the above reason, all algorithms considered in this paper are randomized behavioral algorithms.

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 0.3. 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*. In general, 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 r be the last request. Then there is an integer $a \geq 0$ a finite set of items $Y = \{y_1, \ldots, y_m\}$, such that $\omega(r, y_i) = a$ for all $y_i \in Y$, $\omega(r, y) = a + 1$ for any $y \notin Y$, and $\omega(x, y) = a + 2$ for any $x, y \notin Y \cup \{r\}$. We call pairs (r, y), for $y \in Y$, the *support* of ω. Sometimes, informally, we may 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 $\langle r|y_1 y_2 \ldots y_m\rangle$. Then 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. When some $y_i \in Y$ is requested, the adversary cost is 0 and the offset function changes to $\langle y_i|r\rangle$. When some $y \notin Y$ is requested, the adversary cost is 1 and the offset function changes to $\langle y|ry_1 \ldots y_m\rangle$. Easy verification of these rules is left to the reader. See [7] for details.

0.3 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 $\epsilon > 0$. Thus RMARK is essentially optimal among marking algorithms.

We first define a certain random walk $\{\mu_t\}$ and prove a technical lemma about its distribution. This random walk 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: $\nu_0 = 0$, 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 the expected maximum of ν after $3\delta/2$ steps is at most $\sum_{i=1}^{\infty} i(2/3)^{i+1} = 4$. 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. So we will restrict our consideration to $t \leq \delta$.

We now describe a method to implement μ and ν which will make is easier to compare their maximums. At step t, we change μ and ν as follows: (1) with probability $\min\left\{\frac{2}{5}, p_t\right\}$ both processes increase their position, (2) with probability $\min\left\{\frac{3}{5}, 1 - p_t\right\}$ both processes decrease their position, (3) if $p_t \leq \frac{2}{5}$ then increase ν with probability 1 and decrease μ with probability $\frac{2}{5} - p_t$, and (4) if $p_t > \frac{2}{5}$ then decrease ν with probability 1 and increase μ with probability $p_t - \frac{2}{5}$.

An event $\mu_t > \nu_t$ can occur only if there is a $s < t$ for which $\nu_s = \mu_s$ and $p_s > \frac{2}{5}$. Intuitively this means that at some step s enough white balls have been chosen to increase the probability of choosing a red ball at the next step above $\frac{2}{5}$.

We now show that this event occurs with probability $o(1/\delta)$. 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 $s \geq \delta/4$. Again, there is a constant $a > 0$ such that $\Pr[\nu_s \leq -2s/5] \leq 2e^{-as}$ (see [13]). Summing over $s = \delta/4, \ldots, \delta$, we get $\Pr[\mu_t > \nu_t] = o(1/\delta)$. Since $\mu_t \leq \delta/2$ for all t, this completes the proof of inequality (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: The proof is based on Yao's minimax principle (see, for example, [4]). We show a probability distribution on request sequences on which any on-line marking algorithm $\mathcal{A}$ pays $2H_k - o(\log k)$ times more than a clairvoyant adversary.

The adversary uses a set X of $k + \delta$ items, where $\delta = o(\log k)$ and $\delta = \omega(1)$. 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 are 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 at the beginning of phase i are not in the adversary cache. The probability that Δ_{i+1} intersects $F_i \cup \Delta_i$ in a given phase is $O(1/\delta)$. For such phases, serve faults by evicting items from Δ_i. At the end of such phases, restore the invariant by evicting items from Δ_{i+1}. It is easy to see that this can be done at a cost of at most δ.

We now restrict our attention to 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 cache which are in Δ_{i+1} choose the one which will be requested latest (in this phase).

We fault on $\delta/2$ items in F_i, plus possibly some faults in Δ_{i+1}. If on each fault in F_i we had at least one previously requested item from Δ_{i+1} in the cache, then the adversary would incur exactly $\delta/2$ faults and would have exactly $\delta/2$ items from Δ_i in its cache, as needed. It remains to estimate the number of faults in Δ_{i+1}. The number of these faults will be precisely the number of times the adversary cannot serve a request 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}, up to time t. 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 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 $\delta/2+O(1)+\delta O(1/\delta) = \delta/2+O(1)$. This completes the proof. □

0.4 Trackless Algorithms for $k = 2$

Recall that a *trackless* on-line algorithm is defined as follows: at each step, it 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.

0.4.1 A New Upper Bound

We now present a trackless algorithm for 2 servers that we call TRACK2. The algorithm maintains two types of marks associated with the 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 the other item in the cache.

Algorithm TRACK2: Suppose the current 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:

1. If the cache is (x, y), we go to $(z, \dot{x})$ or $(z, \dot{y})$, each with probability $\frac{1}{2}$.
2. If the cache is $(x, \dot{y})$, we go to (z, x) with probability p and to $(z, \ddot{y})$ with probability $1 - p$. (We assume that $p \geq \frac{1}{2}$.)
3. If the cache is $(x, \ddot{y})$, we go to (z, x) with probability 1.

We now introduce some notation for distributions. Notation x^α means that x is in the cache with probability α. If the last request is x and for $i = 1, \ldots, m$ y_j is in the cache with probability p_i, then we represent this distribution by $(x, y_1^{p_1} y_2^{p_2} \ldots y_m^{p_m})$, where the y_j are listed in order, from most recent to least recent. Some of the y_j in this notation may be marked. For example, $(x, y^\alpha \dot{y}^\beta \dot{z}^\gamma)$ is a distribution in which the last request was x and the other item in the cache is y (with no mark) with probability α, $\dot{y}$ (y with the mark) with probability β, and $\dot{z}$ with probability γ.

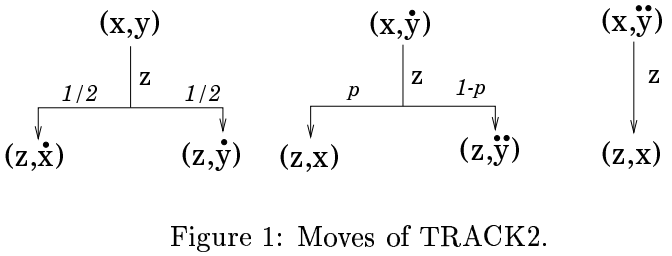

Figure 1: Moves of TRACK2.

Lemma 2 *At each step the distribution has the form*

$$\left(x\,,\, y_1^\alpha \dot{y}_1^\beta \dot{y}_2^\beta \ddot{y}_2^{\gamma_2} \ddot{y}_3^{\gamma_3}\right)$$

where $\alpha \geq \gamma_2$, *and* $\beta + \gamma_2 \geq \gamma_3$.

Proof: The proof is by induction. The lemma trivially holds 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. After requesting y_j, $j = 1, 2, 3, \ldots$, the distribution is

$$\begin{aligned} y_1 &\rightarrow \left(y_1\,,\, x^{\alpha+(1+p)\beta+\gamma_2+\gamma_3}\ddot{y}_2^{(1-p)\beta}\right) \\ y_2 &\rightarrow \left(y_2\,,\, x^{(1+p)\beta+\gamma_2+\gamma_3}\dot{x}^{\frac{1}{2}\alpha}\dot{y}_1^{\frac{1}{2}\alpha}\ddot{y}_1^{(1-p)\beta}\right) \\ y_j &\rightarrow \left(y_j\,,\, x^{2p\beta+\gamma_2+\gamma_3}\dot{x}^{\frac{1}{2}\alpha}\dot{y}_1^{\frac{1}{2}\alpha}\ddot{y}_1^{(1-p)\beta}\ddot{y}_2^{(1-p)\beta}\right), \quad \text{for } j \geq 3 \end{aligned}$$

The verification of the desired inequalities is straightforward. □

As explained in Section 0.2, we use standard notation $\langle x|yz..\rangle$ for offset functions, where r is the last request, and the support is (x, y), (x, z), ... The items $x, y, z \ldots$, are listed in order from most to least recently requested.

Theorem 2 *For $p = \frac{1}{2}(5 - \sqrt{13})$, Algorithm* TRACK2 *has competitive ratio $\frac{1}{4}(3 + \sqrt{13}) \approx 1.6514$.*

Proof: Each configuration is determined by the current distribution of TRACK2 and the current offset function. By Lemma 2, the distribution of TRACK2 has the form

$$\left(x\,,\, y_1^{\alpha}\dot{y}_1^{\beta}\dot{y}_2^{\beta}\ddot{y}_2^{\gamma_2}\ddot{y}_3^{\gamma_3}\right).$$

Assuming that $y_1, \ldots, y_m$ are in the support, we define the potential of this configuration by

$$\Phi \;=\; \begin{cases} (2-p)\beta + \gamma_2 + \gamma_3 & m = 1 \\ (2-p/2)\alpha + (2-p)\beta + \gamma_3 & m = 2 \\ (2-p/2)\alpha + 2(2-p)\beta + \gamma_2 & m = 3 \\ (2-p/2)\alpha + 2(2-p)\beta + \gamma_2 + \gamma_3 & m \geq 4 \end{cases}$$

By routine inspection, one can show that this is the "lazy" potential function, equal to the maximum cost of TRACK2 if the adversary repeats requests on x and some y_j, for $j \leq m$. Thus, by definition, Φ works on requests in the support.

For a request outside the support, assume (without loss of generality) that the requested point had probability zero. Then, depending on the support of the original configuration, we get

$$\Delta\Phi \;=\; -\tfrac{1}{2}p\alpha + (4p - p^2 - 2)\beta + \tfrac{1}{2}(2-p)(\gamma_2+\gamma_3) + \begin{cases} \alpha+\beta & m = 1 \\ \beta+\gamma_2 & m = 2 \\ \gamma_3 & m = 3 \end{cases}$$

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

$$\begin{aligned} \Delta\Phi &\leq \left(1 - \tfrac{1}{2}p\right)(\alpha + \gamma_2 + \gamma_3) + (4p - p^2 - 1)\beta \\ &= 1 - \tfrac{1}{2}p + (5p - p^2 - 3)\beta \end{aligned}$$

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 $\Delta cost + \Delta\Phi \leq \frac{1}{4}(3 + \sqrt{13})\Delta opt$, completing the proof. □

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

0.4.2 Finite State Algorithms

We now consider finite-state algorithms. An *m-state algorithm* is a probabilistic automaton with m memory states and actions (evictions) associated with transitions. The inputs are "fault" or "hit a", where a 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 evict. By $h^a_{s,t}$ we denote the probability that $\mathcal{A}$ moves from state s to t if item a is hit, and by $f^a_{s,t}$ we denote the probability that in state s, on a fault, $\mathcal{A}$ evicts item a and moves to state t. Note that $\sum_j h^k_{i,j} = 1$ for all pairs (i,k) and $\sum_{j,k} f^k_{i,j} = 1$ for all i.

Two States

As described above, a trackless 2-state algorithm $\mathcal{A}$ can be described by the values of $h^k_{i,j}$ and $f^k_{i,j}$.

Theorem 3 *No 2-state trackless algorithm can have a competitive ratio smaller than* 2.

Proof: Fix a 2-state algorithm $\mathcal{A}$ and assume that the cache initially contains items $\{x, y\}$. Consider the following four scenarios:
S_1: $\mathcal{A}$ is in state 1 and receives the request sequence $(zx)^*$,
S_2: $\mathcal{A}$ is in state 1 and receives the request sequence $(zy)^*$,
S_3: $\mathcal{A}$ is in state 2 and receives the request sequence $(zx)^*$, and
S_4: $\mathcal{A}$ is in state 2 and receives the request sequence $(zy)^*$.

If we define C_i to be $\mathcal{A}$'s expected cost on S_i and utilize the fact that the algorithm is trackless then we can find the following equalities:

$$\begin{aligned} C_1 &= 1 + f^1_{1,1}C_1 + f^1_{1,2}C_2 & \qquad C_2 &= 1 + f^2_{1,1}C_2 + f^2_{1,2}C_4 \\ C_3 &= 1 + f^1_{2,1}C_1 + f^1_{2,2}C_3 & \qquad C_4 &= 1 + f^2_{2,1}C_2 + f^2_{2,2}C4 \end{aligned}$$

Performing a bit of algebra yields:

$$\begin{aligned} C_1 &= \frac{1+f^1_{1,2}-f^1_{2,2}}{1-f^1_{1,1}-f^1_{2,2}-f^1_{1,2}f^1_{2,1}+f^1_{1,1}f^1_{2,2}} & \qquad C_2 &= \frac{1+f^2_{1,1}-f^2_{2,1}}{1-f^2_{1,1}-f^2_{2,2}-f^2_{1,1}f^2_{2,1}+f^2_{1,1}f^2_{2,2}} \\ C_3 &= \frac{1-f^1_{1,1}+f^1_{2,1}}{1-f^1_{1,1}-f^1_{2,2}-f^1_{1,2}f^1_{2,1}+f^1_{1,1}f^1_{2,2}} & \qquad C_4 &= \frac{1-f^2_{1,1}+f^2_{2,1}}{1-f^2_{1,1}-f^2_{2,2}-f^2_{1,1}f^2_{2,1}+f^2_{1,1}f^2_{2,2}} \end{aligned}$$

Since the optimal cost on all four scenarios is 1, $\mathcal{A}$ cannot have a competitive ratio smaller than $\max_i C_i$. Minimizing this value over all valid choices of $f^k_{i,j}$ yields the lower bound 2. □

Three States

We now present an RMARK-type algorithm for 2 servers that we call BRMARK (better RMARK). It works as follows: We maintain 1 or 2 marks assigned to items in the cache. First, mark the requested item z. If z is in the cache (hit), we are done. If z is not in the cache (miss), we have two cases:

1. If the two items in cache are marked, unmark them both and evict each with probability 1/2.
2. If just one item in the cache is marked, then:
   (a) With probability p do this: evict the unmarked item. Keep the mark on the other item with probability q, and erase it with probability $1-q$.
   (b) With probability $1-p$ evict the marked item.

We assume $\frac{1}{2} \leq p \leq 1/(2-q)$. Note that for $p = 1/2$ we get RANDOM (independently of q), and for $p = q = 1$ we get MARK.

As before, x^α means that x is in the cache with probability α. When x is marked, we denote it $\dot{x}$.

Lemma 3 *At each step the distribution is given by*

$$\left(\dot{x}\,,\, \dot{y}^\alpha y^\beta y_1^{\gamma_1} \dots y_i^{\gamma_i} \dots\right)$$

where the probabilities satisfy the following inequalities: (i) $\alpha + 2\beta \leq 1$, *and* (ii) $\alpha + \beta \geq \gamma_1 \geq \gamma_2 \dots$.

Proof: The proof is by induction. Requesting x does not change anything, so we can assume the request is either y or some y_j.

Case 1: The request is on y. The new distribution is

$$\left(\dot{y}\,,\, \dot{x}^{\alpha'} x^{\beta'} y_1^{\gamma'_1} \dots y_i^{\gamma'_i} \dots\right)$$

for $\alpha' = \alpha + \beta + pq(1-\alpha-\beta)$, $\beta' = p(1-q)(1-\alpha-\beta)$, and $\gamma'_i = (1-p)\gamma_i$ for each i. We have $\alpha' + 2\beta' - 1 = (1-\alpha-\beta)(p(2-q)-1) \leq 0$, because $p \leq 1/(2-q)$. Thus (i) holds. Clearly $\gamma'_{i+1} \geq \gamma'_i$. Also, $\alpha' + \beta' - \gamma'_i = p + (1-p)(\alpha+\beta-\gamma_i) \geq 0$, completing the proof of (ii).

Case 2: The request is on some y_j. The new distribution is

$$\left(\dot{y}_j\,,\, \dot{x}^{\alpha'} x^{\beta'} y^{\gamma'} y_1^{\gamma'_1} \dots y_i^{\gamma'_i} \dots\right)$$

for $\alpha' = pq(1-\alpha-\gamma_j) + \gamma_j$, $\beta' = p(1-q)(1-\alpha-\gamma_j) + \frac{1}{2}\alpha$, $\gamma' = (1-p)\beta + \frac{1}{2}\alpha$, and $\gamma'_i = (1-p)\gamma_i$ for each $i \neq j$. We have $\alpha' + 2\beta' - 1 = (1-\alpha-\gamma_j)(p(2-q)-1) \leq 0$ because $p \leq 1/(2-q)$. Thus (i) holds. That the γ'_i are decreasing is obvious. Also, $\alpha' + \beta' - \gamma' = p(1-\alpha) + (1-p)\gamma_j - (1-p)\beta \geq p(1-\alpha) - (1-p)\beta \geq (1-p)(1-\alpha-\beta) \geq 0$, and $\gamma' - \gamma'_i = \frac{1}{2}\alpha + (1-p)(\beta-\gamma_i) \geq (1-p)(\alpha+\beta-\gamma_i) \geq 0$. □

Theorem 4 *If $p = \frac{3}{4}$ and $q = \frac{2}{3}$, then algorithm BRMARK has competitive ratio $\frac{5}{3} \approx 1.667$.*

Proof: We only sketch the proof idea and leave the calculations to the reader. By Lemma 3, all configurations are of the form $\langle \dot{x} \,|\, \dot{y}^{\alpha} y^{\beta} y_1^{\gamma_1} \dots y_{\ell}^{\gamma_{\ell}} \rangle y_{\ell+1}^{\gamma_{\ell+1}} \dots$, for some $\ell \geq 0$. We define the potential of this configuration by

$$\Phi \;=\; \begin{cases} \frac{1}{p}(1-\alpha-\beta) & \text{for } \ell = 0 \\ \frac{1}{p}\big(1-\gamma_{\ell}+\alpha(p-\frac{1}{2})\big) & \text{for } \ell > 0 \end{cases}$$

Note that p and q satisfy the conditions: (i) $p = \frac{1}{2-q} \geq \frac{1}{2}$, and (ii) $p(1+q(2p-1)) = 1$. By equality (i), p and q satisfy the condition of BRMARK. Note also that, by Lemma 3, for $j > 0$ we have

$$1-\alpha-\beta \;\leq\; 1-\gamma_j+\alpha(p-\tfrac{1}{2}) \tag{3}$$

We need to show that $\Delta cost + \Delta\Phi \leq C \cdot \Delta opt$ for each move. We will ignore the z_i outside the support. Refer to the proof of Lemma 3 for the detailed description of transitions. The rest of the proof is by careful analysis of the following cases: (1) request on y, (2) request is on y_j, for $j \leq \ell$, (3) request is on y_j for $j > \ell$ (outside support) and $\ell = 0$, (4) request is on y_j for $j > \ell$ (outside support), and $\ell > 0$.

We show the derivation for Case 3. In that case we can assume that $\gamma_j = 0$. The new configuration is $\left\langle \dot{y}_j \,|\, \dot{x}^{\alpha'} x^{\beta'} y^{\gamma'} \right\rangle$, for $\alpha' = pq(1-\alpha)$, $\beta' = p(1-q)(1-\alpha) + \frac{1}{2}\alpha$, $\gamma' = (1-p)\beta + \frac{1}{2}\alpha$. Then

$$\begin{aligned} \Delta\Phi &= \frac{1}{p}\big[\big(1-\gamma'+\alpha'(p-\tfrac{1}{2})\big) - (1-\alpha-\beta)\big] \\ &= \frac{1}{2p}[(\alpha + p(2p-1)q(1-\alpha) + 2p\beta] \\ &\leq \frac{1}{2p}[(\alpha + p(2p-1)q(1-\alpha) + p(1-\alpha)] \\ &= \frac{1}{2p}[(1-p(1+(2p-1)q))\alpha + p(1+(2p-1)q)] \;=\; \tfrac{2}{3}, \end{aligned}$$

and thus $\Delta cost + \Delta\Phi \leq \frac{5}{3} = C \cdot \Delta opt$. □

The analysis of BRMARK is tight. Start from a configuration $\langle \dot{x} | \dot{y}^1 \rangle$ and request z. We have three different adversary strategies (ignoring symmetric strategies) that force ratio $\frac{5}{3}$:

Strategy S1: Repeat lazy calls on x and z. BRMARK will pay $1 + \frac{1}{2}\sum_{i=0}^{\infty}(1-p)^i = \frac{5}{3}$, and the adversary cost is 1.

Strategy S2: Request another item t, and then repeat requests on x and t. The optimal cost is 2 and BRMARK pays $\frac{5}{2} - \frac{q}{2} + pq + \frac{1}{2p} = \frac{10}{3}$, so the ratio is $\frac{5}{3}$.

Strategy S3: Let $x = z_0$, $y = z_1$, $z = z_2$. Requests are given in pairs, $z_3 z_2 z_4 z_3 z_5 z_4 \dots$, where $z_3, z_4, \dots$, are new items. The configuration type will converge to $\left\langle \dot{z}_{2j} | \dot{z}_{2j+1}^{\alpha} z_{2j+1}^{\beta} \right\rangle$, where α and β

satisfy: $\alpha = \alpha\frac{1}{2} + \beta(1-p) + [1-(1-\alpha)pq - \alpha\frac{1}{2} - \beta(1-p)]pq$, $\beta = (1-\alpha)\frac{1}{2}pq + [1-(1-\alpha)pq - \alpha\frac{1}{2} - \beta(1-p)]p(1-q)$. After substituting p, q and solving, we obtain the ratio $\frac{5}{3}$.

0.5 Final Comments

What is the optimal competitive ratio of trackless algorithms for $k = 2$? We know now that this ratio is between 1.54 and 1.651. Our preliminary analysis indicates that algorithm TRACK2 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 haven't been able to find a better potential function.

Another interesting question is whether the optimal ratio for trackless algorithms can be achieved with some fixed number of states.

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 $\epsilon > 0$.

Bibliography

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

[6] 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.

[7] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. In *Proc. 35th Symp. Foundations of Computer Science*, pages 394–400, 1994.

[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*, pages 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*, pages 218–235, 2000.