Finely-Competitive Paging

Avrim Blum[*] Carl Burch[†] Adam Kalai[‡]

Abstract

We construct an online algorithm for paging that achieves an $O(r + \log k)$ competitive ratio when compared to an offline strategy that is allowed the additional ability to "rent" pages at a cost of $1/r$. In contrast, the competitive ratio of the Marking *algorithm for this scenario is $O(r \log k)$. Our algorithm can be thought of in the standard setting as having a "fine-grained" competitive ratio, achieving an $O(1)$ ratio when the request sequence consists of a small number of working sets, gracefully decaying to $O(\log k)$ as this number increases.*

Our result is a generalization of the result in Bartal et al. [2] that one can achieve an $O(r + \log n)$ ratio for the unfair n-state uniform-space Metrical Task System problem. That result was a key component of the polylog(n) competitive randomized algorithm given in that paper for the general Metrical Task System problem. One motivation of this work is that it may be a first step toward achieving a polylog(k) randomized competitive ratio for the much more difficult k-server problem.

1. Introduction

Paging is a classic online problem. In this problem, an online algorithm controls a cache of k pages and is faced with a sequence of memory requests $\rho_1, \rho_2, \ldots$. When an item outside the current cache is requested, the algorithm incurs a page fault and (in the classic paging problem) must load the requested page into the cache, evicting some other page of its choice. The goal of the algorithm is to minimize the number of page faults it makes.

Within the framework of competitive analysis, Fiat et al. [6] (see also [4]) describe a randomized algorithm called Marking and show that it achieves an $O(\log k)$ competitive ratio. The competitive ratio is the worst-case ratio (over request sequences) of the expected number of page faults by the online algorithm to the number of page faults incurred by the optimal offline algorithm that knows the entire sequence in advance. Fiat et al. show, moreover, that any paging algorithm must have a competitive ratio of at least $\Omega(\log k)$. We consider in this paper the question of whether one can perform better than Marking in a somewhat subtle sense. In particular, we consider the following question: suppose that on a page fault, the *offline* algorithm is allowed the additional power to "rent" the requested page at a cost of only $1/r$ (think of $r = \log k$), compared with the cost of 1 for actually loading the page into the cache. Renting means that the memory request is serviced but the requested page is *not* brought into the cache and the offline cache is *not* modified. So, for instance, if the offline algorithm rents a page and then the same page is requested again, the offline algorithm incurs another page fault. The *online* algorithm is given no such privilege. (Technically, it is convenient to allow the online algorithm to rent for a cost of 1; this can help the online algorithm by at most a factor of two.) The question we examine is, what competitive ratio can be achieved in this scenario? For instance, can one still achieve an $O(\log k)$ competitive ratio when $r = \log k$? This question can be thought of as an application of Seiden's [14] notion of an "unfair" competitive ratio — unfair because the offline algorithm has this extra ability to rent cheaply — to the paging problem.

For this harder unfair problem, no algorithm can achieve a competitive ratio less than r (consider a sequence where each request is to a new page), nor can any algorithm achieve a competitive ratio less than $O(\log k)$. Marking achieves competitive ratio $O(r \log k)$. We consider the question of whether one can achieve ratio $O(r+\log k)$. The main result of this paper is that we can, using an algorithm based on the Randomized Weighted-Majority algorithm WMR from machine learning [12] (also called Hedge [8]) together with a notion of phases similar to Marking.

[*]Carnegie Mellon University. E-mail: `avrim+@cs.cmu.edu`. Supported in part by NSF grant CCR-9732705.

[†]Carnegie Mellon University. E-mail: `cburch+@cs.cmu.edu`. Supported in part by an NSF graduate fellowship.

[‡]Carnegie Mellon University. E-mail: `akalai+@cs.cmu.edu`. Supported in part by an NSF graduate fellowship.

1.1. Motivation

Because the problem stated above is not obviously self-motivating, we begin by presenting two motivations, one from paging and another from the k-server problem.

Finely-competitive paging: Request sequences in practice often consist of a core working set of frequently requested pages, together with occasional assorted memory requests, where this working set may change over time. Suppose that, in hindsight, the request sequence can be partitioned into time periods $t_1, t_2, \ldots, t_m$ with the following property. Each time period t_i has a working set W_i ($|W_i| \leq k$), such that there are at most o_i requests to pages *outside* W_i during the time period. In this scenario, one offline strategy in our "unfair" model is in each time period to load the current working set into the cache and to rent the requests outside the current working set. This has a cost of

$$\frac{1}{r}(o_1 + \cdots + o_m) \\ +|W_1| + |W_2 \setminus W_1| + \cdots + |W_m \setminus W_{m-1}| \,.$$

Taking $r = \log k$, an algorithm with unfair competitive ratio $O(r + \log k)$ must pay at most $O(\log k)$ times this, or

$$O\left((o_1 + \cdots + o_m) + (\log k)(|W_1| \right. \\ \left. +|W_2 \setminus W_1| + \cdots + |W_m \setminus W_{m-1}|)\right) \,.$$

So, if the sequence involves only a few working sets or if their differences are small compared to the o_i, the online algorithm is only a small (constant) factor from the optimal service sequence.

Here is a simple concrete example. Suppose that the request sequence repeatedly cycles over a fixed set of $k+1$ pages. In that case, the deterministic LRU algorithm has competitive ratio k (it faults on every request) and Marking has competitive ratio $O(\log k)$ (in expectation, it makes $O(\log k)$ page faults per cycle). However, our algorithm in this case is required to have an $O(1)$ ratio because we can view this sequence as having a single fixed working set of size k, with one additional request per cycle. In other words, in the unfair model, the offline algorithm could simply incur a cost of $1/r = 1/\log k$ per cycle by renting.

In a sense, this goal can be viewed as follows. The motivation of the competitive ratio measure itself is to allow the online algorithm to perform worse on "harder" sequences but to require it to perform better on "easier" ones. Unfairness provides a more fine-grained measure, in which we split the offline cost into an "easy" component (the rentals) and a "hard" component (the loads). We require the algorithm to be constant-competitive with respect to the easy component and only allow an $O(\log k)$ ratio with respect to the hard component.

Because of the working set phenomenon, researchers have tried designing cache systems that in a certain sense add such a renting ability. One practical implementation is to reserve the main cache for the supposed working set while adding a second, smaller cache of potential working-set candidates [10].

In Section 5 we give results of a simple experiment on paging traces made available by Fiat and Rosen [7] showing that these traces do, in a sense, contain working sets of this flavor.

The k-server problem: The question of the best possible randomized competitive ratio for the k-server problem of Manasse, McGeoch, and Sleator [13] remains a major open question. Koutsoupias and Papadimitriou's proof [11] that the work-function algorithm achieves an $O(k)$ competitive ratio was a breakthrough result, especially given that $\Omega(k)$ is a lower bound for deterministic algorithms. However, it is conceivable that a randomized algorithm could achieve a polylog(k) ratio. Hope that this might be possible comes from the Bartal et al. result [2] giving a polylog(n) competitive ratio for the simpler n-state Metrical Task System (MTS) Problem. At the core of this result is an algorithm for achieving an $O(r + \log n)$ ratio for the r-unfair uniform-space MTS problem. Essentially, the "r" term is the competitive ratio of a recursive application of the algorithm run on a subspace, which is being abstracted to a single point in the current space. The uniform-space MTS problem is roughly equivalent to the paging problem in a domain of $k+1$ pages total. Thus, our result can be thought of as a generalization of this $O(r+\log n)$ bound to the paging problem. The motivation is that this could potentially be one step toward achieving a polylog(k) bound for the more general k-server problem. Of course, there are many additional issues involved in attempting to construct such a recursive k-server algorithm. We discuss some of these issues in Section 6.

2. A warmup: The case of $k+1$ pages

We can view the special case of the paging problem in which there are only $k+1$ distinct pages requested as an MTS problem on a uniform space of $k+1$ points. For the r-unfair version of this problem, Bartal et al. [2] prove that a randomized work-function-based algorithm achieves competitive ratio $r + O(\log k)$. Blum and Burch [3] prove that Herbster and Warmuth's simpler "experts" algorithm Variable-Share [9] achieves a similar bound. We begin by showing that an even simpler algorithm (simpler to describe and to analyze), also achieves an $O(r + \log k)$ bound, though it has somewhat worse constants than the others. We call this algorithm the *Phased Randomized Weighted-Majority* algorithm (Phased-WMR). It is this algorithm that we then extend to the general paging problem.

The Phased-WMR algorithm, in the context of the $(k + 1)$-page paging problem, works as follows. Each phase proceeds until every one of the $k + 1$ pages has had r requests. At the beginning of the phase, we associate to each page a weight w_i which is initialized to 1. The weights w_i define a probability distribution $p(i) = w_i/W$, where $W = \sum_j w_j$; this is our probability over pages *not* to have in the cache. (For example, initially all weights are 1 and so each page is equally likely to be the one outside the cache.) When a page is requested, we multiply the page's weight by β ($\beta < 1$ is a parameter of the algorithm) and readjust our probability distribution accordingly. Notice that this effectively increases the probability that the page is in the cache. Operationally, if $\mathbf{p}$ is the probability distribution before the request and $\mathbf{p}'$ is the distribution after, the readjustment can be implemented by the rule, "on a request to page j not in the cache, rent with probability $p'(j)/p(j)$ and load with probability $(p(j) - p'(j))/p(j)$; in the latter case we evict each page i with probability $(p'(i) - p(i))/(p(j) - p'(j))$." Note that the expected cost to our algorithm for this request to j is simply $p(j)$.

In the terminology of the machine learning literature, we could think of having an "expert" associated to each of the $k + 1$ subsets of k pages advocating that the cache contain these k pages, and we could think of the Phased-WMR algorithm as the standard Randomized Weighted Majority algorithm WMR [12], with the small modification that we reinitialize the algorithm periodically at phase boundaries.

A well-known theorem of Littlestone and Warmuth [12, 5] states that the expected loss incurred by WMR is at most

$$\frac{L \ln(1/\beta) + \ln n}{1 - \beta}, \qquad (1)$$

where L is the loss of the best expert in hindsight and n is the number of experts. This formula assumes that if an algorithm has probability distribution $\mathbf{p}$ among experts, and the experts receive loss vector ℓ, then the expected loss of the algorithm is $\mathbf{p} \cdot \ell$ (e.g., if expert j receives loss of 1 and the rest receive loss of 0, then the algorithm's expected loss is $p(j)$). As noted above, this is exactly the case in our setting. Therefore, in our context, this implies that the expected cost of the Phased-WMR algorithm per phase is at most $1 + (r \ln(1/\beta) + \ln(k + 1))/(1 - \beta)$. (The "1+" is the initialization cost for choosing a random page at the beginning of the phase.) Now, noting that the offline algorithm must pay at least 1 per phase, either to evict a page or to rent a page r times, we have the following theorem.

Theorem 1 *The competitive ratio of the Phased-WMR algorithm for the r-unfair $(k + 1)$-page paging problem is at most*

$$1 + \frac{r \ln(1/\beta) + \ln(k + 1)}{1 - \beta}.$$

Notice that the bound of Theorem 1 is of the desired $O(r + \log k)$ form. For $\beta = 3/4$, we get approximately $1 + 1.15r + 4 \ln(k+1)$. As $\beta \to 1$, the bound approaches $1 + (1 + \varepsilon/2)r + \frac{1}{\varepsilon} \ln k$.

For the general paging problem, we extend the Phased-WMR algorithm to have one "expert" for every *sequence* of pages marked in the previous phase, which the expert believes should be the order in which pages are evicted during the current phase. The two difficulties that this approach entails are (1) there are now many more experts, and (2) the possible cost for switching between two different experts has increased from 1 to k. We deal with the first issue by considering "pools" of multiple experts. The second issue involves substantially more effort.

3. The general case: Phases and the offline cost

We begin the description of the general case by defining the notion of "phase" that the online algorithm uses and proving a lower bound for the offline cost based on this notion. Then in Section 4 we describe how the algorithm behaves within each phase and prove an upper bound on the expected online cost. Because our online algorithm is not a "lazy" algorithm, we separately analyze its expected number of page faults (the easier part of the analysis) and its expected cost for modifying its probability distribution over caches (the harder analysis). To define the initial state of our problem, we assume the cache is empty before the first request occurs.

Like the Marking algorithm, we divide the request sequence into phases. We say that page j is *marked* when it has accumulated at least r requests within the phase. The phase reaches its end when k pages become marked.

Let M_i denote the set of pages marked in phase i. (Define M_0 to be the empty set.) Also, let $\ell_{i,j}$ denote the number of requests to page j in phase i. We define m_i as the number of pages marked in phase i but not in the previous phase ($|M_i \setminus M_{i-1}|$). Finally, we define o_i as the total offline cost for renting pages outside $M_{i-1} \cup M_i$; that is, $o_i = \frac{1}{r}\sum_{j \notin M_{i-1}\cup M_i} \ell_{i,j}$.

As in the standard analysis of Marking, this use of phases gives a convenient lower bound on the offline player's cost.

Lemma 2 *If $cost_{\mathrm{OPT}}(\sigma)$ is the optimal offline cost for the task sequence, then we have*

$$cost_{\mathrm{OPT}}(\sigma) \geq \frac{1}{2}\sum_i (m_i + o_i) \ .$$

Proof. Consider two phases $i-1$ and i. Notice that for all but the k pages in the offline cache at the beginning of phase $i-1$, the offline algorithm must either load the page into its cache, at a cost of 1, or service all requests to that page (if any) by renting, at a cost of at least $(\ell_{i-1,j} + \ell_{i,j})/r$. Therefore, any offline algorithm must pay at least

$$cost_{\mathrm{OPT}}(\sigma_{i-1}\sigma_i) \geq \left(\sum_j \min\left\{1, \frac{\ell_{i-1,j}+\ell_{i,j}}{r}\right\}\right) - k$$

in these two phases. For pages j marked in phases $i-1$ or i, we know $\ell_{i-1,j} + \ell_{i,j} \geq r$; for other pages j, we know $\ell_{i,j} < r$ since j is not marked in phase i. These facts imply

$$\begin{aligned}
&\left(\sum_j \min\left\{1, \frac{\ell_{i-1,j}+\ell_{i,j}}{r}\right\}\right) - k \\
\geq\ & \left(\sum_{j\in M_{i-1}\cup M_i} 1\right) + \left(\sum_{j\notin M_{i-1}\cup M_i} \frac{\ell_{i,j}}{r}\right) - k \\
=\ & (k + m_i) + o_i - k = m_i + o_i \ .
\end{aligned}$$

Also note that the offline algorithm must pay at least $m_1 + o_1$ in the first phase. Let σ_i represent the sequence of requests in phase i. Then we get the following.

$$\begin{aligned}
2cost_{\mathrm{OPT}}(\sigma) &\geq cost_{\mathrm{OPT}}\left((\sigma_1\sigma_2)(\sigma_3\sigma_4)\cdots\right) \\
&\quad + cost_{\mathrm{OPT}}\left(\sigma_1(\sigma_2\sigma_3)(\sigma_4\sigma_5)\cdots\right) \\
&\geq ((m_2+o_2) + (m_4+o_4) + \cdots) \\
&\quad + ((m_1+o_1) + (m_3+o_3) + \cdots) \\
&= \sum_i (m_i + o_i) \ .
\end{aligned}$$

■

4. The online algorithm

We now describe a randomized online algorithm whose expected cost in each phase i is $O(r + \log k)$ more than the offline bound of $\frac{1}{2}(m_i + o_i)$ given in Lemma 2. To describe the algorithm, we use $p_t(j)$ to denote the probability that page j is in the cache after servicing the tth request.

We divide the description and analysis of the algorithm into two parts. First, we describe how the algorithm determines the probabilities $p_t(j)$, and we use this to bound the expected number of page faults incurred by the algorithm. We then describe how the algorithm loads and ejects pages to maintain these probabilities, and we bound the additional cost incurred by those operations.

4.1. The online cache probabilities and expected number of page faults

The algorithm determines the probabilities $p_t(j)$ based on a weighted average over a collection of "experts". At the beginning of phase i, the algorithm has M_{i-1} in its cache, and initializes one expert for each of the $k!$ permutations of the pages in M_{i-1}. Each expert behaves like a deterministic version of the Marking algorithm, where the given permutation determines the order in which unmarked pages are thrown out, and pages are considered marked when they have received r requests. Specifically, the expert for permutation $\mathcal{P}$ behaves as follows:

- Any page requested at least r times in this phase is considered marked.

- On a page fault, rent the requested page if it is not yet marked. Otherwise, load the requested page into the cache, throwing out the first (according to permutation $\mathcal{P}$) unmarked page that is still in the cache.

Notice that at the end of the phase, each expert has exactly M_i in its cache, maintaining our initial assumption.

Each expert is initialized with a weight of 1 and we use the WMR algorithm with $\beta = 1/2$ to update the weights; that is, we multiply the weight of an expert by β whenever it incurs a page fault. The probabilities $p_t(j)$ are determined in the natural way from these weights. Specifically, $p_t(j)$ is the result of dividing the total weight on experts having page j in their cache by the total weight on all the experts. If we select a cache according to a distribution matching these probabilities, then our algorithm's expected number of page faults will match the expected cost to WMR.

Lemma 3 *By combining these experts using* WMR*, the online algorithm's expected number of page faults in phase i is at most* $(2.8r + 2\ln k)m_i + (1.4r)o_i$.

Proof. For concreteness, let us first consider the case $m_i = 0$. In this case, none of the experts will recommend loading any pages and the algorithm will have $M_{i-1} = M_i$ in its cache throughout the phase. Thus it pays a total of ro_i, meeting the desired bound.

In the general case, a "good" expert is one in which the m_i pages of M_{i-1} that were *not* marked come first in its permutation, and the $k - m_i$ marked pages of M_{i-1} come last. There are $m_i!(k-m_i)!$ of these good experts, and each one makes at most $2rm_i + ro_i$ page faults in the phase for the following reason. For each of the m_i pages $j \in M_i \setminus M_{i-1}$, it incurs a total of r page faults until the page is finally marked and brought into the cache. For each of the m_i pages $j \in M_{i-1} \setminus M_i$, it incurs at most r page faults after throwing it out (since these pages do not become marked). Finally, the expert always rents pages $j \notin M_{i-1} \cup M_i$, and the total cost for these is ro_i.

The formula in equation (1) for the loss of the WMR algorithm can be generalized to the case where we have a "pool" of many good experts. In this case, the bound becomes

$$\frac{L\ln(1/\beta) + \ln(n/n_{good})}{1-\beta}\,, \qquad (2)$$

where L is an upper bound on the loss of any expert in the pool, n_{good} is the number of experts in the pool, and n is the total number of experts. In our case, $L = 2rm_i + ro_i$ and $n/n_{good} = k!/(m_i!(k-m_i)!) = \binom{k}{m_i}$. If we choose $\beta = 1/2$ and maintain probabilities $p_t(j)$ according to the expert weights as above, then the total expected number of page faults of our algorithm is at most

$$\begin{aligned} &1.4(2rm_i + ro_i) + 2\ln\binom{k}{m_i} \\ &\quad\leq\quad 1.4(2rm_i + ro_i) + 2m_i \ln k \\ &\quad=\quad (2.8r + 2\ln k)m_i + (1.4r)o_i\,. \end{aligned}$$

■

For the purpose of analyzing the movement cost, it is helpful to modify the algorithm described above in two ways. First of all, when an expert has been determined to be "bad" — that is, if a page it has thrown out becomes marked — we give it an infinite penalty, setting its weight to 0. Second, when an expert throws out some page j, we penalize it for all requests to that page that have occurred so far in the phase; i.e., we penalize it as if it had thrown the page out at the very beginning. Notice that the first modification only helps the algorithm, and the second modification has already been factored in to the upper bound L on the loss of the "good" experts calculated in the proof above.

These two modifications allow us to write the probability that a given page is in the cache of the algorithm in terms of the *number* of requests to each page so far in the phase, without reference to the order in which those requests occurred. This is useful in proving the lemma below.

Lemma 4 *If there is a request to page j at time t, then $p_{t+1}(j) \geq p_t(j)$ and for all $j' \neq j$, $p_{t+1}(j') \leq p_t(j')$.*

Proof sketch. The easy part of the lemma is the statement that when a request is made to page j, the probability that j is in the cache increases. That happens because WMR penalizes all experts that do not have j in their cache and does not penalize those that do. Furthermore, if page j becomes marked by this request, then $p_{t+1}(j) = 1$. The harder part is the statement about pages $j' \neq j$ because of the possibility of correlations among pages.

To analyze these pages, we can directly write out a formula for $p(j')$ in terms of the requests so far. In particular, let m be the number of pages marked so far that were not in the cache at the start of the current phase (so all experts of nonzero weight have evicted exactly m pages), and for pages j_i that *were* in the cache at the start of the phase, define l_{j_i} to be the number of hits to that page during this phase or ∞ if there were r hits. Then we have

$$1 - p(j') = \frac{\beta^{l_{j'}} \sum_{\{j_1,j_2,\ldots,j_{m-1}\neq j'\}} \beta^{l_{j_1}+l_{j_2}+\cdots+l_{j_{m-1}}}}{\sum_{\{j_1,j_2,\ldots,j_m\}} \beta^{l_{j_1}+l_{j_2}+\cdots+l_{j_m}}}.$$

It can then be verified that this is an increasing function of all $j \neq j'$, and of m.

■

4.2. Moving between probabilities

At any point in time, our algorithm maintains a probability distribution q over caches (experts), which induces page probabilities $p(j)$ over pages. The section above describes one distribution q using the WMR algorithm. However, notice that for the purpose of computing the expected number of page faults (as in Lemma 3), any two distributions over caches that induce the same page probabilities are equivalent. Therefore, we are free to deviate from the instructions given by the WMR algorithm so long as we are faithful to the page probabilities $p(j)$. This is important for the next part of our analysis, where we bound the expected cost incurred by moving between probability distributions.

In particular, we now examine the following question. Given a distribution q over caches that induces probabilities $p(j)$ over pages, and given a new target set of page probabilities $p'(j)$ that satisfies $\sum_j p'(j) = k$, we want to move to some new distribution q' over caches that induces p'. At a minimum, any algorithm must load an expected $\sum_{p'(j)>p(j)} (p'(j) - p(j))$ number of pages to move from the page probabilities p to p'. Achieving this is easily possible in a setting where there are only $k+1$ pages total, but it is harder in general. In this section, we show a method for achieving an expected cost of at most $2\sum_{p'(j)>p(j)} (p'(j) - p(j))$.

A simple example will help illustrate the difficulty and the algorithm. Say that $k = 2$ and initially our cache is $[A, B]$ with probability $1/2$ and $[C, D]$ with probability $1/2$. This induces page probabilities p; say we want to convert this to a new distribution p' as follows.

page	A	B	C	D
p	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
p'	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$

If we momentarily forget about the cache capacity of k, we can easily move to a new cache distribution $\hat{q}$ consistent with p': we can simply evict B with probability $1/2$ if our cache is $[A, B]$ and load A with probability $1/2$ if our cache is $[C, D]$. So $\hat{q}$ is the following.

cache	$[A]$	$[A, B]$	$[C, D]$	$[A, C, D]$
$\hat{q}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

The $[A, C, D]$ possibility, unfortunately, exceeds the size limit of $k = 2$. However, there is (and there must be) a cache that has a vacancy, in this case $[A]$. We rebalance by adding page D to the small cache and evicting D from the large cache. This new cache distribution is now only over legal caches, and we use this for q'.

cache	$[A, D]$	$[A, B]$	$[C, D]$	$[A, C]$
q'	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

In other words, the strategy in this case is: "if our cache is $[A, B]$ then with probability $1/2$ do nothing and with probability $1/2$ evict B and load D; if our cache is $[C, D]$ then with probability $1/2$ do nothing and with probability $1/2$ evict D and load A." This strategy seems a bit strange because $p'(D) = p(D)$ yet we sometimes evict or load D, but this is necessary in this situation. As you can see, the expected number of page loads in this example is $1/2$, which equals $2\sum_{p'(j)>p(j)} (p'(j) - p(j))$.

Our strategy, in general, is as follows. To move from a set of probabilities p to p', for any page j with $p'(j) < p(j)$, we evict j from our cache (if present) with probability $1 - p'(j)/p(j)$. Next, for pages with $p'(j) > p(j)$, we add them to a cache not containing j with probability $(p'(j) - p(j))/(1 - p(j))$. This gives us a cache distribution $\hat{q}$ with the correct probabilities p' and loading cost $\sum_{p'(j)>p(j)} (p'(j) - p(j))$, but it may create caches that are too large.

Fortunately, the expected number of pages in the cache is $\sum p'(j) = k$. Thus, if there are caches with more than k pages, there must be caches with fewer than k pages. Take a cache with more than k pages and one with fewer than k pages, and some page that is in the larger but not the smaller. We can evict the page from the larger cache and load it in to the smaller cache in such a way as to not change p'. If the two caches do not have equal probabilities, we cannot immediately reduce the probability of both of the original caches to 0. However, one of the two caches will end with probability 0, and thus we are always making discrete progress in decreasing the total excess and shortage in cache sizes, over all caches with nonzero probability. Furthermore, the total probability of performing a load in the rebalancing step is no more than the probability of loading a page in the increase step, since each load required for a rebalance originates from an increased probability. The expected number of loads is no more than $2\sum_{p'(j)>p(j)} (p'(j) - p(j))$.

Lemma 5 *Given a probability distribution q on caches, this implies page probabilities p. Given a new set of page probabilities p', we can move to a new probability distribution q' on caches with expected cost $2\sum_{p'(j)>p(j)} (p'(j) - p(j))$.*

4.3. Bounding the online movement cost

The final step to showing that our algorithm achieves the required bound is to use Lemmas 4 and 5 to show that its cost for maintaining the page probabilities $p_t(j)$ is at most its expected number of page faults, which we have already bounded in Lemma 3.

Lemma 6 *Using the movement strategy given in Lemma 5, the expected cost for the algorithm of Section 4.1 for maintaining its probability distribution is at most twice its expected number of page faults.*

Proof. Consider the expert weights before receiving a request to page j. Let p be the page probabilities before the request and p' be the page probabilities after the request. Since j is the only page whose probability of being in the cache increases (Lemma 4), the expected cost from Lemma 5 is at most $2\,(p'(j) - p(j))$. This is clearly at most $2(1 - p(j))$, which is twice the probability of incurring a page fault. ∎

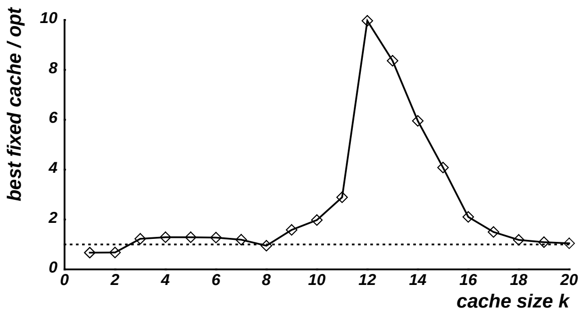

Figure 1. fgrep trace

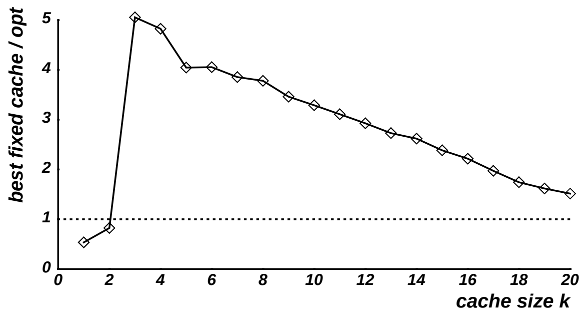

Figure 2. gzip trace

Combining Lemmas 3 and 6 to bound the total expected online cost, and using Lemma 2 as our bound on the offline cost, we conclude with our competitive ratio of $O(r + \log k)$.

Theorem 7 *There is an algorithm whose r-unfair competitive ratio for the paging problem is* $6(2.8r + 2\ln k)$.

5. Experiments

As described in Section 1.1, the performance measure considered here can be viewed as a kind of fine-grained competitive ratio for the standard (no-rental) paging problem. In particular, we can think of our goal as being to simultaneously achieve constant-competitiveness with respect to the number of page faults of the best *fixed* cache (working set) in hindsight, a slightly worse ratio compared to the best partition into two working sets in hindsight, and so on up to an $O(\log n)$-competitive ratio with respect to OPT.

To get some sense of the interestingness of this performance measure, we examined page trace data from Fiat and Rosen [7].[1] For each trace, and for each choice of cache size k, we considered two quantities: (1) the number of page faults incurred by the best fixed cache in hindsight (i.e., this is just the number of requests to pages not in that set), and (2) the number of page faults incurred by the optimal offline page replacement policy (OPT). The ratio of these two quantities is plotted in Figures 1 and 2 for two of these traces. The point to notice from these traces is that depending on the cache size, the ratio of these quantities can vary substantially, and for many cache sizes it might be better to have a small competitive ratio compared to the best single cache, rather than a large ratio compared to OPT.[2] This suggests that having both properties simultaneously would be a desirable quality for an online algorithm.

[1] `http://www.math.tau.ac.il/~rosen/results.html`.

[2] It is actually possible for the best fixed cache to do better than OPT, because OPT is required to bring the requested page into its cache on a page fault.

6. Conclusions

This paper presents an algorithm for achieving an $O(r + \log k)$ competitive ratio for the r-unfair paging problem, which we can view as achieving a fine-grained form of competitive ratio in the standard paging setting. The main technique we use for doing so is the WMR approach from online machine learning, though a number of technical issues must be addressed in order to make it work. In particular, in the standard machine learning setting, one need not worry about "costs" for switching between experts as we have here. Moreover, the diameter of the space (the maximum possible cost for switching between two experts) is k, so the generic bound of [3] cannot be used here. A drawback of this approach is that the resulting algorithm is not time-efficient in its straightforward implementation, though it appears possible to improve on this somewhat.

An interesting question is whether an algorithm for the unfair scenario can be used to get improved bounds for the k-server problem [13] if we probabilistically approximate a space using the Hierarchically Separated Trees of Bartal [1]. Bartal et al. [2] determine how to do so for the MTS problem, but there are several challenges to extending this to the k-server problem. Even assuming that the game is being played on a metric space of $poly(k)$ points, and that the HST for the space is balanced, it is still not clear how to manage the recursion. In particular, unlike in the MTS problem, there are varying numbers of servers that can be placed in each subspace by both the online and offline algorithms. This means, for instance, that the abstraction would have to consider what it means to have multiple servers at a single point in the uniform space.

This paper demonstrates an algorithm with an $O(r + \log k)$ ratio, but it is not as good as one may hope. We would like an algorithm (preferably simple and efficient) whose ratio is $r + O(\log k)$, as Bartal et al. [2] demonstrate for the uniform-space MTS problem. A slightly simpler goal is a ratio of $(1+\varepsilon)r + (1 + 1/\varepsilon)O(\log k)$

(as in Theorem 1 for the case of $k + 1$ points). An algorithm with either ratio would provide additional hope for application of Bartal's HST approximation to the k-server problem.

References

[1] Y. Bartal. On approximating arbitrary metrics by tree metrics. In *Proc ACM Symposium on Theory of Computing*, pages 161–168, May 1998.

[2] Y. Bartal, A. Blum, C. Burch, and A. Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proc ACM Symposium on Theory of Computing*, pages 711–719, 1997.

[3] A. Blum and C. Burch. On-line learning and the metrical task system problem. In *Proc ACM Workshop on Computational Learning Theory*, pages 45–53, 1997.

[4] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *J of the ACM*, 39(4):745–763, 1992.

[5] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. In *Proc ACM Symposium on Theory of Computing*, pages 382–391, 1993.

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

[7] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: beating the LRU standard? In *Proceedings of the 8th Symposium on Discrete Algorithms*, pages 63–72, 1997.

[8] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J Comp Syst Sci*, 55(1):119–139, 1997.

[9] M. Herbster and M. Warmuth. Tracking the best expert. *J Machine Learning*, 32(2):286–294, 1998.

[10] L. John and A. Subramanian. Design and performance evaluation of a cache assist to implement selective caching. In *Proc International Conference on Computer Design*, pages 610–518, October 1997.

[11] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *J of the ACM*, 42(5):971–983, 1995.

[12] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[13] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for server problems. *J Algorithms*, 11:208–230, 1990.

[14] S. Seiden. Unfair problems and randomized algorithms for metrical task systems. *Information and Computation*, 1998. To appear.