Chapter 27
On-Line Caching as Cache Size Varies

Neal Young*

Abstract

We consider the competitiveness of on-line strategies using k servers versus the optimal off-line strategy using $h \leq k$ servers for the paging, weighted cache, and k-server problems.

We show that when $h < k$ the competitiveness of the marking algorithm, a randomized paging strategy, is no more than $2(\ln \frac{k}{k-h} - \ln\ln \frac{k}{k-h} + \frac{1}{e-1})$ when $\frac{k}{k-h} \geq e$, and at most 2 otherwise. We show this is roughly within a factor of two of optimal.

Intuitively, we say a paging strategy is *loosely* $C(k)$-*competitive* if, for any sequence, at most cache sizes the fault rate of the paging strategy is at most an insignificant amount above $C(k)$ times the fault rate of the optimal strategy. We show that LRU, FWF, and FIFO are loosely $C(k)$-competitive provided $C(k)/\ln k \to \infty$ and that the marking algorithm is loosely $C(k)$-competitive provided $C(k) - 2\ln\ln k \to \infty$.

We formulate the off-line version of the k-server problem as a linear program and examine the dual to derive a lower bound on the performance of the optimal algorithm. We use this bound to show that for the weighted cache problem the balance algorithm is $\frac{k}{k-h+1}$-competitive.

1 Introduction

In the most general version of the k-server problem, one is given a directed graph with edge lengths and k *servers*. Initially, the servers are positioned on nodes of the graph. A sequence $r_1, r_2, \ldots, r_n$ of requests for service at nodes is given and after each request a server must be moved to the node if a server is not already present. The goal is to minimize the total length travelled by the servers. In the *standard* version of the problem, the edge weights $d(i,j)$ satisfy $d(i,j) = d(j,i)$, $d(i,i) = 0$, and $d(i,j) \leq d(i,k) + d(k,j)$.

A special case of the k-server problem is the *weighted cache problem*. In this case the nodes of the graph have non-negative weights, and the cost to service a request to a node with no server is taken to be the weight of the node.

A special case of the weighted cache problem is the *paging problem*. In this case, all of the nodes have weight one. For this problem, we adopt traditional terminology: nodes are referred to as *pages*, served nodes are said to be *in the cache*, and a request to an unserved node is a *fault*.

If all of the requests of a k-server problem are known in advance, the optimal allocation of servers may be found. This is called the *off-line* version of the problem. In the *on-line* version, servers must be moved without knowledge of requests beyond the next.

Competitive analysis was introduced for on-line paging and list management algorithms by Sleator and Tarjan [10]. One strategy is said to be c-competitive against another if, on any sequence, c times the cost of the former is within an additive constant (independent of the sequence) of the latter. The *competitiveness* of the one strategy versus the other is the infimum of such c.

In this paper, the competitiveness of an on-line strategy will generally refer to the competitiveness using k servers versus the optimal off-line algorithm with $h \leq k$ servers.

For randomized strategies, different extensions of competitiveness are possible, see[1]; in this paper we use the oblivious adversary model, in which the cost of a randomized strategy on a sequence is taken to be the expected cost over all random choices of the strategy.

1.1 Previous Results.

Previous efforts have determined bounds on the minimum competitiveness of a deterministic or randomized strategy for special cases of the k-server problem.

For any deterministic on-line algorithm, any standard k-server graph, and any k and $h \leq k$, in [6] it is shown how to construct a sequence of requests so that the on-line algorithm with k servers pays at least $\frac{k}{k-h+1}$ times as much as the optimal off-line algorithm with h servers. For some metric spaces, slightly better lower bounds can be shown; for example, when $h = 2$ a lower bound of 2 can be shown for any k and any deterministic on-line k-server algorithm for servicing requests on the line [8].

For the paging problem known bounds are as follows. For deterministic algorithms, matching upper and lower bounds of $\frac{k}{k-h+1}$ were established by [10]. For the randomized case with $h = k$, an upper bound of $2H_k$ and a lower bound of H_k were established by [4]. (H_k,

*Computer Science Department, Princeton University, Princeton, NJ 08544, USA. Research supported by the Hertz Foundation.

Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, 241-250 (1991)

the kth Harmonic Number, equals $1+\frac{1}{2}+\cdots+\frac{1}{k} \approx \ln k$.) This upper bound was lowered to H_k by [7].

For the weighted cache problem fewer results have been obtained. In [3], a simple "memoryless", randomized, $\frac{k}{k-h+1}$-competitive strategy, and a matching lower bound for memoryless randomized strategies, are given. In [2] the balance algorithm, a deterministic k-server algorithm which moves the server which has incurred the least net cost, is shown to be k-competitive when $h = k$. Currently no randomized algorithm is known to have lower competitiveness than $\frac{k}{k-h+1}$. In [5] a lower bound above H_k is given on the competitiveness (with $h = k$) of any randomized strategy for a three node k-server problem which is easily translated into a weighted cache problem.

Many other special cases and generalizations of the k-server problem have been studied; for a more extensive bibliography, see [1].

1.2 Our Results. We give three results in three sections. The sections may be read in any order.

In section 2, we generalize the argument of [4] to show that, when $h < k$, the competitiveness of the marking algorithm, a randomized paging strategy, is no more than $2(\ln \frac{k}{k-h} - \ln\ln \frac{k}{k-h} + \frac{1}{2})$ when $\frac{k}{k-h} \geq e$ and at most 2 otherwise. We also show that this is roughly within a factor of two of optimal: any randomized paging strategy has competitiveness at least $\ln \frac{k}{k-h} - \ln\ln \frac{k}{k-h} - \frac{3}{k-h}$ when $\frac{k}{k-h} \geq e$.

In the full paper we briefly discuss a generalization of the marking algorithm, called the (h,k)-marking algorithm, that is within a factor of two of optimally competitive, and a corresponding generalization of the lower bound of [4] to the case $h < k$.

In section 3, we give a relaxed model for competitive analysis of paging strategies. The model is intended to incorporate the assumptions that input sequences are not strongly dependent on cache size and competitiveness is only important when the fault rate is significant. Specifically, a paging strategy is *loosely $C(k)$-competitive* if for any c, as $n \to \infty$, for any sequence requesting at least n distinct pages, the number of cache sizes $k \in \{1,...,n\}$ such that the fault rate of the paging strategy exceeds $1/n^c$ plus $C(k)$ times the optimal is $o(n)$.

We show that LRU, FWF, and FIFO are loosely $C(k)$-competitive provided $C(k)/\ln k \to \infty$. We show that the marking algorithm, a randomized paging strategy, is loosely $C(k)$-competitive provided $C(k) - 2\ln\ln k \to \infty$.

In section 4, we consider the off-line k-server problem as an integer linear program. By considering the dual, we establish a lower bound on the cost of the optimal off-line algorithm. The goal of this technique, which we call the *dual bounding technique*, is, by establishing a good lower bound on the off-line algorithm, to derive a competitive on-line algorithm. The lower bounds in [10] and [4] are special cases of our bound.

We apply the bound to derive the *greedy strategy*, which for the weighted cache problem turns out to be a variant of the balance algorithm, and to show that the greedy strategy and consequently the balance algorithm are $\frac{k}{k-h+1}$-competitive for the weighted cache problem.

Amotz Bar-Noy and Baruch Schieber [8] have independently shown a deterministic weighted cache algorithm to be $\frac{k}{k-h+1}$-competitive.

In the full paper we also have a small section describing the application of the dual bounding technique to the list management problem.

2 Randomized On-Line Paging

In this section, we consider the competitiveness of randomized paging strategies. We extend the arguments of Fiat *et al.* to show that when $h < k$ the competitiveness of the marking algorithm is no more than $2(\ln \frac{k}{k-h} - \ln\ln \frac{k}{k-h} + \frac{1}{2})$ when $\frac{k}{k-h} \geq e$, and at most 2 otherwise. We modify the lower bound of [10] on the competitiveness of any deterministic strategy to show that this is roughly within a factor of two of optimal.

2.1 The Marking Algorithm. The marking algorithm of [4], with a cache of size k, partitions the input sequence into k-phases — subsequences of requests extending as long as possible while containing requests to at most k distinct pages. Within a phase, when a page must be removed from the cache, the marking algorithm chooses a page uniformly at random from those not yet requested during the phase and swaps that page out.

In [4], the following argument is given showing that the marking algorithm with a cache of size k is $2H_k$-competitive when $h = k$. In a phase, we classify the requests into *new* requests, which are requests to pages not requested previously in this phase or the previous, *old* requests, which are requests to pages not requested yet in this phase but requested in the previous, and *repeated* requests, which are requests to pages which have been requested this phase.

Suppose there are m new requests and $k - m$ old requests. At any point in the phase all pages requested in the previous phase but not yet in this phase are equally likely to be in the cache. Before the ith old request, at most $m + i - 1$ pages have been requested this phase, so that at least $k-m-(i-1)$ of the $k-(i-1)$ pages requested in the previous phase but not yet in this phase remain in the cache. Thus the probability that the page requested at the ith old request will cause a

fault is at most $\frac{m}{k-(i-1)}$. In addition, each of the m new requests causes a fault, so the expected cost during the phase is bounded by

$$m + \sum_{i=1}^{k-m} \frac{m}{k-(i-1)} = m + m(H_k - H_m).$$

Within any two consecutive phases, the off-line strategy with a cache of size $h = k$ makes at least m faults. This gives a lower bound of $\frac{\overline{m}}{2}$ times the number of phases on the cost of the off-line algorithm. (Here $\overline{m}$ is taken to be the average number of new requests per phase.) As a consequence, the marking algorithm is shown to be $2H_k$-competitive.

We extend this result in the following theorem.

THEOREM 2.1. *The competitiveness of the marking algorithm with a cache of size k versus any algorithm with a cache of size $h < k$ is no more than*

$$2\left(\ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h} + \frac{1}{2}\right)$$

when $\frac{k}{k-h} \geq e$, and no more than 2 otherwise.

Proof. As a first step at extending the above analysis due to [4] to the case $h < k$, note that any algorithm with a cache of size h must make at least $k + m - h$ faults in any two consecutive phases with the second phase having m new requests. This is because $k + m$ distinct pages are requested during the two phases, and at most h are in the cache at the start of the first of the two phases. Consequently, if the average number of new pages per phase is $\overline{m}$, the number of faults to the optimal algorithm with a cache of size h is at least $\frac{k+\overline{m}-h}{2}$. The competitiveness of the marking algorithm is thus bounded by two times

$$\frac{\overline{m} + \overline{m}(H_k - H_{\overline{m}})}{k + \overline{m} - h}. \tag{2.1}$$

It remains only to bound (2.1). We omit the argument in this abstract, noting only that we consider the continuous upper bound $f(\overline{m}) = \frac{\overline{m}}{k+\overline{m}-h}(\ln\frac{k}{\overline{m}} + 1)$ and use analytic techniques. □

2.2 A Lower Bound on the Competitiveness.

Next we generalize the lower bound of Sleator and Tarjan[10] on the competitiveness of any deterministic strategy to show that any randomized paging strategy has competitiveness at least roughly half that of the marking algorithm.

THEOREM 2.2. *The competitiveness (versus an oblivious adversary) of any randomized paging algorithm with a cache of size k versus any algorithm with a cache of size $h < k$ is at least*

$$\ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h} - \frac{3}{k-h}$$

when $\frac{k}{k-h} \geq e$.

Note that for $\frac{k}{k-h} \leq e$ the analysis of the marking algorithm shows that its competitiveness is at most 2.

Proof. We give an (oblivious) adversary argument, in which the adversary knows the probability distribution of pages in the cache at any point.

We generate a sequence of requests in segments. During each segment, we first request $k-h+m$ pages not previously requested. These, together with the h pages in the cache of the optimal strategy before the beginning of the first request of the segment, yield $k + m$ pages, called *candidate* pages. Next we generate $h-1$ requests, $r_1, r_2, \ldots, r_{h-1}$, each to the candidate node least likely to be in the cache of the on-line strategy. Once a request to a page r_i has been made, we interrupt the remaining request sequence as necessary with repeated requests to r_i to keep it in the cache with probability at least $1-\epsilon$.

This generates $k - h + m$ requests to new pages followed by at least $h - 1$ requests to at most $h - 1$ candidate pages. The optimal strategy can keep the $h-1$ candidate pages requested in the latter part of the segment in its cache, and use the remaining space to service the requests in the former part of the segment, thus incurring a cost of at most $k - h + m$.

Before request r_i is made, the expected number of pages $r_1, \ldots, r_{i-1}$ in the cache of the on-line strategy is at least $(1-\epsilon)(i-1)$, so the expected number of the $k + m - i + 1$ remaining candidate pages in the cache is at most $k - (1-\epsilon)(i-1)$, the expected number not in the cache is at least $m - \epsilon(i-1)$, and the probability of a fault is at least $\frac{m-\epsilon(i-1)}{k+m-i+1}$. It follows that the expected number of faults can be made arbitrarily close to $k - h + m + m(H_{k+m} - H_{k+m-h+1})$, and a lower bound on the competitiveness is given by

$$\max_{m=1,2,\ldots} 1 + \frac{m}{k-h+m}(H_{k+m} - H_{k+m-h+1}). \tag{2.2}$$

(Note that the above analysis assumes a bounded number of repeated requests to the r_i. Since each such request costs the on-line algorithm at least ϵ, if necessary we can end the request sequence after enough repeated requests have occurred for the on-line algorithm to incur cost sufficient to obtain the same bound.)

It remains only to prove that (2.2) is at least $\ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h} - \frac{3}{k-h}$ when $\frac{k}{k-h} > e$. We omit this part of the proof in this abstract, remarking only that

the bound is proven by taking $m = \lfloor (k-h)(\ln \frac{k}{k-h} - 1) + 1 \rfloor$. □

3 Competitiveness and Fault Rate

In this section, we restrict our attention to competitiveness when $h = k$, unless otherwise stated.

The marking algorithm has substantially lower competitiveness than any deterministic algorithm. Is it possible that some randomized algorithm might improve over deterministic strategies in practice? A typical graph of competitiveness vs. cache size for a real sequence appears in figure 1. This sequence, traced by Dick Sites of Digital Equipment Corporation [9], consists of 692,057 requests to 642 distinct pages, and was generated by two X-windows network processes, a "make" (program compilation), and a disk copy running concurrently. The requests include data reads and writes and instruction fetches. For the purposes of this paper the page size was taken to be 1024 bytes.

This graph shows competitiveness of LRU of less than two and one half, independent of the cache size. There is little room for improvement over LRU.

Motivated by these results, we ask if there is any sense in which we can effectively show constant competitiveness (independent of cache size) for any on-line paging strategy. We relax the standard model of competitiveness for paging by considering a range of cache sizes for a given sequence and allowing violations of the competitive ratio at a vanishing fraction of the cache sizes and when the fault rate is insignificant. Specifically, a paging strategy is *loosely $C(k)$-competitive* if for any c, as $n \to \infty$, for any sequence requesting at least n distinct pages, the number of cache sizes $k \in \{1, ..., n\}$ such that the fault rate of the paging strategy exceeds $1/n^c$ plus $C(k)$ times the optimal is $o(n)$. (The constants in the $o(n)$ may depend on c.)

In this section we show that LRU, FWF, and FIFO are loosely $C(k)$-competitive provided $C(k)/\ln k \to \infty$. We also show that the marking algorithm is loosely $C(k)$-competitive provided $C(k) - 2\ln\ln k \to \infty$. The argument we use is somewhat indirect; first we show that under the assumption that competitiveness is high for many cache sizes on a sequence, fault rate decreases rapidly as the cache size is increased. The results of this argument will be used to show loose competitiveness.

In general, we will be assuming some fixed sequence, and our notation will not reflect this dependence explicitly. For instance, for a paging strategy X, X_k will denote the (expected) number of faults made by X on the sequence. We will take the (standard) competitiveness of X with a cache of size k on a sequence to be the ratio $\frac{X_k - k}{\mathrm{OPT}_k}$, for k less than the number of distinct requests in the sequence, and 1 otherwise. The *stable fault rate* — the fault rate not counting the up to k faults to fill the cache initially — is $X_k - k$ divided by the number of requests.

3.1 High competitiveness and low fault rate.

In order to obtain a link between competitiveness and stable fault rate, we use the notion of a *phase*, central to the marking algorithm and the analysis of the flush-when-full paging strategy. FWF, with a cache of size k, implicitly partitions the input sequence into *k-phases* — subsequences of requests extending as long as possible while containing requests to at most k distinct pages. FWF flushes it cache at times corresponding to phase boundaries.

Within a phase, FWF makes k faults. On the other hand, one can show (e.g. [4]) that any algorithm with a cache of size k makes at least one fault for every phase. It follows that FWF is k-competitive when $h = k$. More generally, as is also shown in [4], if there are m *new* requests — requests to pages not requested previously in this phase or the previous phase — in a phase other than the first, then any strategy with a cache of size k makes at least m faults in this phase and the previous. This is because at the beginning of the previous phase, the strategy has at most k of the $k+m$ pages which will be requested in the two phases in its cache. It follows that if the average number of new pages per phase is $\overline{m}$, the optimal strategy makes at least $\frac{\overline{m}}{2}$ faults per phase. It follows that the competitiveness of FWF is bounded by $\frac{2k}{\overline{m}}$.

LRU and FIFO also have at most k faults per phase, and a similar argument applies to them. In [4], it is shown that in a phase with m new pages, the marking algorithm has an expected number of faults bounded by $m(H_k - H_m + 1) \leq m(\ln k - \ln m + 1)$, so that the competitiveness of the marking algorithm is at most $2(\ln k - \ln \overline{m} + 1)$. (Note the use of the bound $H_a - H_b \leq \ln \frac{a}{b}$, which follows from an integration argument.)

In each of these cases, high competitiveness is obtained only for small $\overline{m}$, and fault rate is roughly correlated with the number of phases. Next we make precise the following argument. Competitiveness is high only when the number of new pages per phase is small. When the number of new pages is small, a small increase in the cache size will result in a large decrease in the number of phases. Thus if competitiveness is high for many cache sizes for a given sequence, then the number of phases is decreasing rapidly as the cache size increases. This in turn ensures a rapid decrease in the page fault rate.

Given a sequence (implicit in the context of the

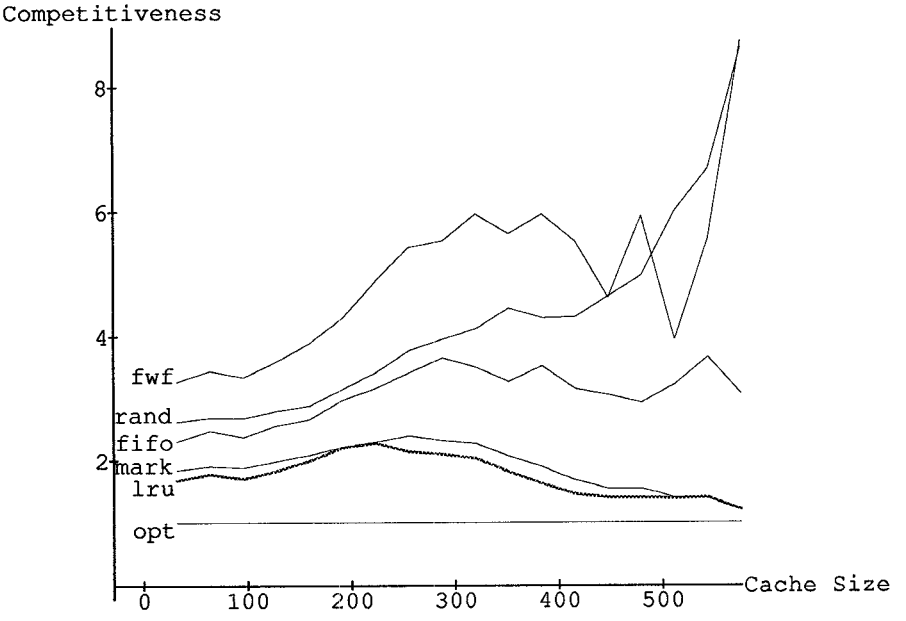

Figure 1: Typical Competitiveness vs. Cache Size

notation),

- P_k denotes the number of k-phases other than the first in the sequence. That is, we break our sequence into k-phases as described above, and P_k is the number of such phases, minus 1.
- $\overline{m}_k$ denotes the average number of new pages per k-phase other than the first. That is, we sum the number of new pages in the phase over all k-phases with new pages, and divide by P_k.
- X_k, for any paging strategy X, denotes the (expected) number of faults made by X with a cache of size k on the sequence. Thus X_k/X_1 is an upper bound on the fault rate of X with a cache of size k (in fact it is exactly the fault rate for a sequence with no immediately repeated requests).

The next lemma is the key to the argument. The lemma states that if the average number of new pages per phase is small for a given cache size, then increasing the cache size by a small amount decreases the number of phases substantially.

LEMMA 3.1. *Fix a sequence. For any cache size k, $P_{k+\lfloor 2\overline{m}_k \rfloor} \leq \frac{3}{4} P_k$.*

Proof. Let $\alpha_0, \ldots, \alpha_{P_k}$ denote the k-phase partitioning of the sequence.

At least half (and thus at least $\lceil P_k/2 \rceil$) of the P_k k-phases $\alpha_1, \ldots, \alpha_{P_k}$ have a number of new pages not exceeding $2\overline{m}_k$ (and thus not exceeding $\lfloor 2\overline{m}_k \rfloor$). Denote these by $\alpha_{i_1}, \ldots, \alpha_{i_{\lceil P_k/2 \rceil}}$.

If we modify the k-phase partitioning by joining every other such phase with the phase immediately preceding, i.e. we join α_{i_j-1} and α_{i_j} for odd j, we obtain a coarser partitioning of the sequence into at most $P_k - \lceil P_k/4 \rceil$ pieces. In the coarser partitioning, pieces resulting from a join reference at most $k + \lfloor 2\overline{m}_k \rfloor$ distinct pages, while the other pieces continue to reference at most k distinct pages.

If we now consider the $k + \lfloor 2\overline{m}_k \rfloor$-phase partitioning, we find that each $k + \lfloor 2\overline{m}_k \rfloor$-phase contains the final request of at least one of the pieces in the coarser partition. This is because if a $k + \lfloor 2\overline{m}_k \rfloor$-phase begins at or after the beginning of a subsequence of requests to at most $k + \lfloor 2\overline{m}_k \rfloor$ distinct pages, it will continue at least through the end of the subsequence.

Thus $P_{k+\lfloor 2\overline{m}_k \rfloor} \leq \frac{3}{4} P_k$. □

It is worth noting that we have a construction for sequences for which $\frac{P_{k+\alpha(k)}}{P_k} \geq \frac{1}{4}$ for all k, and for which the number of new pages per k-phase is no more than $3\alpha(k)$. The construction holds for any integer valued α such that $\alpha(k + \alpha(k)) \leq 2\alpha(k)$ and $1 \leq \alpha(k) \leq k$.

pages inductively

Having established this lemma, the rest of the argument is straightforward. First, we argue that if, for

some fixed sequence, over a range of cache sizes there are many cache sizes with a small average number of new pages per phase, then the number of phases is decreasing rapidly over the range.

LEMMA 3.2. *Given M, and any sequence such that the number of cache sizes k in the range $L, L+1, ..., U$ with $\overline{m}_k \le M$ is N,*

$$P_U \le \left(\frac{3}{4}\right)^{\frac{N}{2M}-1} P_L.$$

Proof. From the N values of k we can choose at least $q = \lceil \frac{N}{2M} \rceil$ values $k_1, k_2, \ldots, k_q$ such that $k_{i+1} - k_i \ge \lfloor 2M \rfloor \ge \lfloor 2\overline{m}_{k_i} \rfloor$ for $i = 1, 2, \ldots, M-1$.

By the previous lemma and the monotonicity of P_k, we have $P_{k_{i+1}} \le \frac{3}{4} P_{k_i}$ for $i = 1, \ldots, q-1$. Consequently

$$P_U \le P_{k_q} \le \left(\frac{3}{4}\right)^{q-1} P_{k_1} \le \left(\frac{3}{4}\right)^{\frac{N}{2M}-1} P_L.$$

□

A deterministic paging strategy with cache size k is *conservative* if, for any k and any sequence of requests, during any contiguous subsequence of requests to only k distinct pages, the strategy makes at most k faults. A strategy is *monotone* if increasing the cache size cannot increase the page fault rate. The reader may verify that FWF, LRU, and FIFO are conservative and monotone.

The basic bounds which we will be using in the next proofs are (for a conservative, monotone paging strategy X and a given sequence)

$$\frac{X_k - k}{\mathrm{OPT}_k} \le \frac{2k}{\overline{m}_k}, \tag{3.3}$$

$$\frac{\mathrm{MARK}_k - k}{\mathrm{OPT}_k} \le 2(\ln k - \ln \overline{m}_k + 1). \tag{3.4}$$

THEOREM 3.1. *Let X be any conservative, monotone paging strategy. Let β be any function such that $\beta(k)$ and $\frac{k}{\beta(k)}$ are non-decreasing. Fix any sequence of requests, and let X_k denote the number of faults by X with a cache of size k on the sequence.*

If there are N cache sizes k in a range $1, ..., U$ with $\frac{X_k - k}{\mathrm{OPT}_k} \ge \beta(k)$, then

$$\frac{N}{U} \le \frac{4}{\ln \frac{4}{3}} \frac{\ln \frac{X_1 - 1}{X_U - U} + \ln U + \ln \frac{4}{3}}{\beta(U)}.$$

Proof. The bound (3.3) implies that for every k with $\frac{X_k - k}{\mathrm{OPT}_k}$ at least $\beta(k)$, we have $\overline{m}_k \le \frac{2k}{\beta(k)}$.

Thus taking $M = \frac{2U}{\beta(U)}$, we have N cache sizes k in the range $1, \ldots, U$ such that $\overline{m}_k \le \frac{2k}{\beta(k)} \le M$, so the previous lemma and the inequalities $X_U \le U P_U + U$ and $X_1 \ge P_1 + 1$ give

$$\frac{X_U - U}{U} \le P_U \le \left(\frac{3}{4}\right)^{\frac{N}{2M}-1} P_1 \le \left(\frac{3}{4}\right)^{\frac{N}{4U}\beta(U)-1} (X_1 - 1).$$

Taking logarithms and rewriting gives the result. □

Next we give the corresponding result for the marking algorithm.

THEOREM 3.2. *Let β be any function with $\beta(k)$ and $\frac{k}{e^{\beta(k)/2}}$ non-decreasing. Fix any sequence of requests, and let MARK_k denote the expected number of faults by the marking algorithm with a cache of size k on the sequence.*

If there are N cache sizes k in a range $1, ..., U$ with $\frac{\mathrm{MARK}_k - k}{\mathrm{OPT}_k} \ge \beta(k)$, then

$$\frac{N}{U} \le \frac{2}{\ln \frac{4}{3}} \frac{\ln \frac{\mathrm{MARK}_1 - 1}{\mathrm{MARK}_U - U} + \ln U + \ln \frac{4}{3}}{e^{\frac{1}{2}\beta(U)-1}}.$$

Proof. The bound (3.4) implies that each cache size k with $\frac{\mathrm{MARK}_k - k}{\mathrm{OPT}_k}$ at least $\beta(k)$ has $\overline{m}_k \le k e^{1 - \frac{1}{2}\beta(k)}$.

The rest of the proof is as for the previous theorem. □

3.2 Loose Competitiveness. We have established that for any conservative, monotone paging strategy, and for the marking algorithm, the fraction of cache sizes in a range exceeding a bound is bounded by roughly the logarithm of one over the fault rate divided by a term which grows with the bound. This implies that for any sequence either the fault rate drops off rapidly or competitiveness is low at most cache sizes. We use this in the next two corollaries to put upper bounds on the loose competitiveness of these paging strategies.

COROLLARY 3.1. *Let X be any conservative, monotone paging strategy, and $C(k)$ any function such that $\frac{C(k)}{\ln k} \to \infty$ and $C(k)$ and $\frac{k}{C(k)}$ are non-decreasing. Then X is loosely $C(k)$-competitive.*

Proof. Fix c, n, and a sequence with R requests to at least n distinct pages. We must show that the number of cache sizes $k \in \{1, ..., n\}$ such that

$$\frac{X_k}{R} > C(k)\frac{\mathrm{OPT}_k}{R} + \frac{1}{n^c} \tag{3.5}$$

is $o(n)$.

The idea is that if the fault rate doesn't decrease to $1/n^c$ with a cache size of $o(n)$, then theorem 3.1 applies to show that the number of cache sizes with high competitiveness is small.

We assume without loss of generality that $X_n/R \geq 1/n^c$; otherwise we can choose a smaller n and get a larger fraction of cache sizes violating (3.5).

A technical consideration is that in our definition of loose competitiveness we are concerned with X_k rather than $X_k - k$. Thus we must exclude the (vanishing) fraction of cache sizes for which X_k nears k. Thus we consider the range of cache sizes $1, ..., U$, where $U = \left\lfloor n\left(1 - \frac{2}{C(n)}\right)\right\rfloor$.

The number of violations of (3.5) with $U < k \leq n$ is $o(n)$, since $n - U$ is $o(n)$. For $k \leq U$

$$k \leq n - \frac{2n}{C(n)} \leq n - \frac{2k}{C(k)}$$

so $\frac{k}{n-k} \leq \frac{1}{2}C(k)$ and for k also violating (3.5)

$$\frac{X_k - k}{\mathrm{OPT}_k} \geq C(k) - \frac{k}{\mathrm{OPT}_k} \geq C(k) - \frac{k}{n-k} \geq \frac{1}{2}C(k).$$

(Note the use of $\mathrm{OPT}_k \geq n - k$.)

Thus we can use theorem 3.1 to bound the number N of violations with $k \leq U$.

$$\begin{aligned} N &= U\frac{N}{U} \\ &\leq UO(1)\frac{\ln \frac{X_1-1}{X_U-U} + \ln U}{C(U)/2} \\ &\leq O(1)U\frac{\ln \frac{R}{X_U} + \ln(1 + \frac{U}{X_U-U}) + \ln U}{C(U)} \\ &\leq O(1)U\frac{\ln n^c + 2\ln n}{C(U)} \\ &= O(1)U\frac{\ln n}{C(U)} \\ &\leq O(1)n\frac{\ln n}{C(n)}. \end{aligned}$$

This last step follows from $\frac{U}{C(U)} \leq \frac{n}{C(n)}$. Since $\frac{\ln n}{C(n)}$ is $o(1)$, the final expression is $o(n)$. □

Similarly for the marking algorithm:

COROLLARY 3.2. *Let $C(k)$ be any function such that $C(k) - 2\ln\ln k \to \infty$ and $C(k)$ and $\frac{k}{e^{C(k)/2}}$ are non-decreasing. Then the marking algorithm is loosely $C(k)$-competitive.*

Proof. Fix c, n, and a sequence with R requests to at least n distinct pages. Again we must show that the number of cache sizes $k \in \{1, ..., n\}$ such that

$$(3.6) \qquad \frac{\mathrm{MARK}_k}{R} > C(k)\frac{\mathrm{OPT}_k}{R} + \frac{1}{n^c}$$

is $o(n)$.

We have to be a bit more careful about avoiding the fraction of cache sizes near n, because we want to preserve the constant 2 in the statement of the corollary.

As in the previous proof we assume without loss of generality that $MARK_n/R \geq 1/n^c$.

Let $\alpha(k) = (C(k) - 2\ln\ln k)/2$, so $\alpha(k) \to \infty$.

Let $U = \left\lfloor n\left(1 - \frac{1}{\alpha(n)+1}\right)\right\rfloor$. The number of violations of (3.6) occuring with $k > U$ is $o(n)$, since $n - U$ is $o(n)$.

For $k \leq U$ we have $\frac{k}{\mathrm{OPT}_k} \leq \frac{k}{n-k} \leq \frac{U}{n-U} \leq \alpha(n)$, so for k also violating (3.6)

$$\frac{\mathrm{MARK}_k - k}{\mathrm{OPT}_k} \geq C(k) - \frac{k}{\mathrm{OPT}_k} \geq C(k) - \alpha(n).$$

We apply theorem 3.2 to bound the number N of violations with $k \leq U$:

$$\begin{aligned} N &\leq O(1)U\frac{\ln \frac{\mathrm{MARK}_1-1}{\mathrm{MARK}_U-U} + \ln U}{e^{\frac{1}{2}(C(U)-\alpha(n))-1}} \\ &\leq O(1)U\frac{\ln \frac{R}{\mathrm{MARK}_U} + \ln(1 + \frac{U}{\mathrm{MARK}_U-U}) + \ln U}{e^{\frac{1}{2}(C(U)-\alpha(n))}} \\ &\leq O(1)U\frac{\ln n}{e^{\frac{1}{2}(C(U)-\alpha(n))}} \\ &\leq O(1)ne^{\frac{1}{2}(2\ln\ln n - C(n) + \alpha(n))}. \end{aligned}$$

This last step follows since $\frac{U}{e^{\frac{1}{2}C(U)}} \leq \frac{n}{e^{\frac{1}{2}C(n)}}$. The final expression is $o(n)$ since $2\ln\ln n - C(n) + \alpha(n) = -2\alpha(n) + \alpha(n) = -\alpha(n)$. □

(It is worth noting that the previously mentioned construction can be applied to construct a sequence such that the competitiveness of FWF for cache sizes in the range $\lceil n/2\rceil, ..., n$ is at least $(d\log n)/6$, while the fault rate with a cache of size n is bounded below by $1/(4n^{2d+1})$. Thus FWF is not loosely $O(\log k)$-competitive, and for FWF the first corollary is tight.)

4 A K-Server Dual Bound

It was observed in [2] that the off-line version of the k-server problem can be formulated as a minimum-cost maximum flow problem. In this section we first formulate the problem directly as a linear program and derive the dual program. (The derivation of the dual program may be skipped on first reading.) We then give a direct interpretation of the dual and a direct proof that the cost of any feasible solution to the dual gives a lower bound to the cost of the off-line algorithm.

We develop some general terminology for deriving feasible solutions to the dual. We show that a sequential greedy method of generating a solution yields a lower bound which we use to derive a deterministic on-line

k-server strategy called the greedy strategy, which we show is optimally competitive for the weighted cache problem. Finally, we show that for the weighted cache problem, the balance algorithm and the greedy strategy correspond on sequences with no requests to served nodes, so that the balance algorithm is also $\frac{k}{k-h+1}$-competitive.

4.1 A K-Server Linear Program. We are given a directed n vertex graph, with edge lengths $d(i,j)$, and a sequence of requests $r_1, r_2, \ldots, r_N$.

We do not assume $d(i,j) = d(j,i)$ or that the edge lengths satisfy the triangle inequality. Rather, we give a lower bound which will apply to any *lazy* allocation of servers. (An allocation of servers is *lazy* if a server is moved to a node only when the node is requested.) It is well known that for a weighted cache or standard k-server problem (i.e. a k-server problem on an undirected graph with edge weights satisfying the triangle inequality), any allocation can be converted to a lazy allocation without increasing the cost, so that for these problems the lower bound will apply to any allocation of servers.

For simplicity, we assume that $d(i,i) = 0$ and $d(i,j) \geq 0$, although the method extends to the general case.

Given an optimal lazy off-line solution using h servers, let $X_{i,t}$ denote the amount of server on node i at time t. Then

$$\sum_i X_{i,t} = h \quad (\forall t), \tag{4.7}$$

$$X_{r_t,t} = 1 \quad (\forall t), \tag{4.8}$$

$$X_{i,t-1} - X_{i,t} \geq 0 \quad (\forall t > 1, i \neq r_t), \tag{4.9}$$

$$X_{i,t} \geq 0 \quad (\forall t, i).$$

Furthermore, the cost of the allocation is

$$\sum_{t>1,i} (X_{i,t-1} - X_{i,t}) d(i, r_t).$$

It is not hard to reformulate the above linear program as a min-cost max-flow problem, as in [2], and thus show that there are always optimal integer solutions.

4.2 Derivation of a K-Server Lower Bound. To obtain a lower bound for this cost, we consider the above set of constraints and cost as a linear program (the *primal*), and formulate the dual linear program. The cost of any feasible point in the dual serves as a lower bound to the cost of the optimal primal solution, and thus to the cost of the optimal off-line server allocation.

In particular, given any allocation of the $X_{i,t}$ satisfying the above constraints, we consider the following sequence of bounds:

$$\begin{aligned}
&\sum_{t>1,i} (X_{i,t-1} - X_{i,t}) d(i, r_t) \\
&\geq \sum_t -S_t \left(\sum_i X_{i,t} \right) + \sum_t R_t X_{r_t,t} \\
&\quad + \sum_{t>1, i \neq r_t} \Psi_{i,t} (X_{i,t-1} - X_{i,t}) \qquad (4.10) \\
&\geq \sum_t -hS_t + \sum_t R_t \\
&= \sum_t R_t - hS_t. \qquad (4.11)
\end{aligned}$$

The expression (4.11) is the dual cost. The S_t, R_t, and $\Psi_{i,t}$ are the dual variables.

Bound (4.10) will hold provided the coefficient of each variable $X_{i,t}$ in the cost is greater than or equal to the coefficient of the variable in the bound (since $X_{i,t} \geq 0$). Considering this for each $X_{i,t}$ yields the dual constraints (4.12).

Bound (4.10) will in turn be bounded by (4.11) (given (4.7), (4.8), and (4.9)) provided each $\Psi_{i,t}$ is non-negative. This gives the dual constraints (4.13).

This yields the following: the cost of the optimal off-line allocation of h servers servicing the sequence of requests is bounded below by $\sum_t R_t - hS_t$ provided

$$\begin{aligned}
&[t < N]\, d(i, r_{t+1}) - [t > 1]\, d(i, r_t) \\
&\geq [i = r_t]\, R_t - S_t \\
&\quad + [t < N, i \neq r_{t+1}]\, \Psi_{i,t+1} \\
&\quad - [t > 1, i \neq r_t]\, \Psi_{i,t} \quad (\forall t, i), \qquad (4.12)
\end{aligned}$$

$$\Psi_{i,t} \geq 0 \quad (\forall t > 1, i \neq r_t). \tag{4.13}$$

(For a boolean expression b, $[b]$ denotes 1 if b is true and 0 otherwise.) We can add these constraints so that the $\Psi_{i,t}$ telescope, and the bounds (4.12) are slightly simplified. (For details see the full paper.)

This yields:

$(\forall i, t, t' : i = r_t \text{ or } t = 1, \nexists s, t \leq s < t', r_{s=1} = i)$

$$[t' < N]\, d(i, r_{t'+1}) \geq [r_t = i]\, R_t - S_t - \cdots - S_{t'}.$$

4.3 A K-Server Lower Bound. We are now in a position to give a direct interpretation of the dual constraints and lower bound. We give the bound as a theorem, and we give an alternate, more intuitive proof than the derivation.

THEOREM 4.1. *Given requests $r_1, \ldots, r_N$ to nodes in a directed graph with edge costs $d(i,j)$ such that $d(i,j) \geq 0$ and $d(i,i) = 0$, the cost of lazily servicing*

the requests with h servers is at least

$$\sum_{t=1}^{N} R_t - hS_t,$$

provided that the S_t and R_t meet the dual constraints: $(\forall t, t' : 1 \le t \le t' \le N, \nexists s : t \le s < t', r_{s+1} = r_t)$

$$\begin{aligned} R_t - S_t - \cdots - S_{t'} &\le \begin{cases} d(r_t, r_{t'+1}) & (t' < N) \\ 0 & (t' = N) \end{cases} \\ -S_1 - \cdots - S_{t'} &\le 0 \end{aligned}$$

Proof. We maintain a potential for each node as requests for service are received. Initially, all potentials are zero. At each t from 1 to N, we do the following. We raise the potential of the requested node r_t to R_t, then decrease the potential of every node by S_t. Thus at time t' the potential of node i is $R_t - S_t - S_{t+1} - \cdots - S_{t'}$ if i was last requested at time t, and is $-S_1 - \cdots - S_{t'}$ if i was not previously requested.

The *dual constraints* are satisfied if and only if for each time t and node i, the potential of i does not exceed the distance from the node i to the node r_{t+1} next requested (or zero if the current request is the last).

Suppose we have some lazy allocation of h servers satisfying the requests. When the allocation moves a server from a node i at time t to a node r_{t+1} at time $t+1$, or when the allocation leaves a server on $i = r_{t+1}$ from time t to $t+1$, or when t equals N, we charge to the allocation the potential of the node i at time t.

The dual constraints ensure that a lazy allocation is charged no more than it pays to move servers. Consideration shows that the charge to the allocation can be viewed as, at each time t, crediting the allocation by R_t and then debiting the allocation by S_t for each node with a server. This shows that the amount charged is $\sum_t R_t - hS_t$. □

Although this proof is sufficient for the theorem, that the bound is a linear programming dual bound is also important. For instance, that a linear program and its dual have equal cost optimal solutions, together with the fact that the original primal program always has an optimal integer solution, imply that with the right choice of R_t and S_t the above bound can be made tight.

4.4 The Greedy Lower Bound. Given any solution satisfying the dual constraints for a fixed sequence, we can modify it as follows. Let t_i, a function of t and i, denote $\max\{s \le t : r_s = i\}$. If t_i is defined, and we raise S_t and R_{t_i} by some amount Δ, we increase the potential (see the proof of theorem 4.1) on i at times t_i through $t-1$ by Δ, and leave other potentials of i unchanged. For large enough $\Delta \ge 0$, the potential at some time s between t_i and $t-1$ inclusive will meet its upper bound $d(i, r_{s+1})$. This value of Δ is called the *slack* of i at time t. If the slack is non-zero, we say the node is *pending*.

If we raise S_t, and R_{t_i} for each node i pending at time t, by Δ, with Δ not exceeding the minimum slack of the pending nodes, the weights continue to satisfy the dual constraints. We call this a *pivot at time t by* Δ. As a consequence of such a pivot, we add Δ times the number of pending nodes minus $h\Delta$ to the lower bound.

This suggests the following bound. Start by setting all dual variables to zero, so that the dual constraints are satisfied. Consider the requests sequentially. When a request r_t comes in at time t, if $k+1$ nodes are pending, pivot fully (taking Δ to be the minimum slack of the pending nodes), so that following the pivot k or fewer nodes are pending at time t.

Each pivot raises one S_t and $k+1$ R_{t_i} by Δ and raises the lower bound by $(k-h+1)\Delta$. As a consequence we maintain that the lower bound $\sum_t R_t - hS_t$ equals $(k-h+1)\sum_t S_t$, and we have the equality $\sum_t R_t - S_t = k\sum_t S_t$. In order to obtain a $\frac{k}{k-h+1}$-competitive on-line algorithm, then, we need an on-line algorithm whose cost is bounded within a constant of $\sum_t R_t - S_t$.

4.5 The Greedy Strategy. Recalling the proof of theorem 4.1, note that when the off-line allocation moves a server, we charge it the potential of the node the server moves from. Suppose we have an assignment of the R_t and the S_t which gives a tight lower bound. In that case, the off-line allocation moves a server from a node only when the potential on the node equals the cost of moving the server. Thus if the potential on a node at a time t reaches its upper bound, that is an indication that the off-line allocation might move its server from the node at time t.

With respect to the greedy bound, when a node ceases to pend as the result of a pivot at time t, this suggests that at the time $s \le t$ where the potential of the node now meets its upper bound as the result of the pivot, the off-line allocation may have moved a server from that node. These considerations give us the greedy strategy, a deterministic on-line k-server strategy.

The strategy is simply to keep the pending nodes served as the greedy bound is calculated on-line. When a node r_t is requested, following a possible pivot, at time t, k or fewer nodes will be pending, so we can move a server from a non-pending node to a pending node if necessary.

When a node i is requested at time t, it is subsequently pending at times $t, t+1, t+2$, etc. until at some point either it is again requested, or it ceases to

pend as a result of a pivot. If i is requested again while pending, the algorithm incurs no cost. If i is requested after it ceases to pend, R_t has been raised enough so that at some time $t + j \geq t$ the potential of i meets its upper bound. This means that for some $r_{t+j+1} \neq i$, $R_t - S_t - \cdots - S_{t+j} = d(i, r_{t+j+1})$. Thus $R_t - S_t \geq d(i, r_{t+j+1})$.

For the weighted cache problem this is sufficient, for $d(i, r_{t+j+1}) = w(i)$ equals the cost incurred in moving from i. Thus for this problem, the cost incurred by the greedy strategy in moving from requested nodes is bounded by $\sum_t R_t - S_t$. The remaining cost incurred by the algorithm is at most $k \max_i w(i)$. As discussed previously, this implies that for the weighted cache problem the greedy strategy is $\frac{k}{k-h+1}$-competitive.

Note that for the standard k-server problem deterministic on-line $\frac{k}{k-h+1}$-competitive algorithms are not generally possible. For instance, on the 5 node cycle graph, if the next request to a 2-server algorithm is always to the node adjacent to the previous request and furthest from the other server, it is possible to show that the algorithm can not be simultaneously 1-competitive versus the optimal algorithm with 1 server and 2-competitive versus the optimal algorithm with 2 servers.

4.6 The Balance Algorithm. We can describe the greedy strategy for the weighted cache problem directly as follows. Maintain a value for each server. (The value of a server will correspond to the slack at the served node.) When a node is requested that is not served, subtract the minimum server value from all server values. Move a zero-valued server and set its value to the weight of the node to which it moves. When a node is requested that is served, reset the value of the server to the weight of the node.

If we modify the above strategy by ignoring requests to served nodes, we obtain the balance algorithm. As the balance algorithm does not change state when a served node is requested, for the competitive analysis of the balance algorithm it suffices to consider sequences of requests which only request unserved nodes. On such sequences, the balance algorithm corresponds to the above strategy. It follows that the balance algorithm is $\frac{k}{k-h+1}$-competitive for the weighted cache problem.

References

[1] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 379–386, 1990.

[2] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 291–300, 1990.

[3] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 359–368, 1990.

[4] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. Technical Report CMU-CS-88-196, Computer Science Department, Carnegie Mellon University, 1988. (To appear in *Algorithmica*).

[5] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 301–309, 1990.

[6] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 322–333, 1988.

[7] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. Technical Report CMU-CS-89-122, Computer Science Department, Carnegie Mellon University, 1989. (To appear in *Algorithmica*).

[8] Baruch Schieber. Personal communication. 1990.

[9] R. L. Sites and A. Agarwal. Multiprocessor cache analysis using ATUM. In *Proc. 15th IEEE Int. Symp. on Computer Architecture*, pages 186–195, 1988.

[10] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, February 1985.