Algorithmica (1994) 11: 525–541

Algorithmica

The k-Server Dual and Loose Competitiveness for Paging[1]

N. Young[2]

Abstract. *Weighted caching* is a generalization of *paging* in which the cost to evict an item depends on the item. We study both of these problems as restrictions of the well-known *k-server problem*, which involves moving servers in a graph in response to requests so as to minimize the distance traveled.

We give a deterministic on-line strategy for weighted caching that, on any sequence of requests, given a cache holding k items, incurs a cost within a factor of $k/(k-h+1)$ of the minimum cost possible given a cache holding h items. The strategy generalizes "least recently used," one of the best paging strategies in practice. The analysis is a primal–dual analysis, the first for an on-line problem, exploiting the linear programming structure of the k-server problem.

We introduce *loose competitiveness*, motivated by Sleator and Tarjan's complaint [ST] that the standard competitive ratios for paging strategies are too high. A k-server strategy is *loosely* $c(k)$-*competitive* if, for any sequence, for *almost all* k, the cost incurred by the strategy with k servers *either* is no more than $c(k)$ times the minimum cost *or* is insignificant.

We show that certain paging strategies (including "least recently used," and "first in first out") that are k-competitive in the standard model are loosely $c(k)$-competitive provided $c(k)/\ln k \to \infty$ and both $k/c(k)$ and $c(k)$ are nondecreasing. We show that the marking algorithm, a randomized paging strategy that is $\Theta(\ln k)$-competitive in the standard model, is loosely $c(k)$-competitive provided $k - 2\ln\ln k \to \infty$ and both $2\ln k - c(k)$ and $c(k)$ are nondecreasing.

Key Words. On-line algorithms, k-Server problem, Linear programming, Approximation algorithms, Paging, Caching, Competitive analysis.

The body of this paper consists of four sections. In Section 1, the introduction, we describe background, our results, and related work. In Section 2 we give our weighted caching strategy. In Section 3 we show loose competitiveness of various paging strategies. We conclude with comments about further research in Section 4.

1. Introduction.

Many real problems must be solved *on-line*—decisions that restrict possible solutions must be made before the entire problem is known. Generally, an optimal solution cannot be guaranteed if a problem must be solved

[1] This paper is the journal version of "On-line Caching as Cache Size Varies," which appeared in the *Proceedings of the 2nd Annual ACM–SIAM Symposium on Discrete Algorithms* (1991). Details beyond those in this paper may be found in "Competitive Paging and Dual-Guided Algorithms for Weighted Caching and Matching," which is the author's thesis and is available as Technical Report CS-TR-348-91 from the Computer Science Department at Princeton University.

[2] University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742, USA. young@umiacs.umd.edu. This research was performed while the author was at the Computer Science Department, Princeton University, Princeton, NJ 08544, USA, and was supported by the Hertz Foundation.

Received May 16, 1990; revised November 15, 1990, and January 5, 1992. Communicated by Prabhakar Raghavan.

on-line. Thus a natural question for such a problem is whether a strategy exists that guarantees an approximately optimal solution.

In this paper we study the *k-server* problem [M], [MMS]. Various definitions of the problem exist in the literature; we take the following definition, which is technically convenient and essentially equivalent to the other definitions: A complete directed graph with edge lengths $d(u, v)$, a number, k, of identical, mobile servers, and a sequence r of requests, each to some node, is given. In response to the first request, all servers are placed on the requested node. In response to each subsequent request v, if no server is on v, some server must be chosen to move from its current node u to v at a cost of $d(u, v)$. A strategy for solving the problem is *on-line* if it chooses the server to move independently of later requests. The goal is to minimize the total cost.

As many authors (e.g., Chrobak and Larmore [CL]) have pointed out, the k-server problem is an abstraction of a number of practical on-line problems, including linear search, paging, font caching, and motion planning for two-headed disks.

We focus on two special cases of the k-server problem: the weighted caching problem [MMS], in which $d(u, v) = w(u)$ for $u \neq v$ (the cost to move a server from a node depends only on the node), and the paging problem [ST], in which the cost to move a server is uniformly 1.

Traditionally, paging is described as the problem of managing a *fast memory*, or *cache*, capable of holding k items: items are requested; if a requested item is not in the fast memory, it must be placed in the fast memory, possibly evicting some other item to make room. The goal is to minimize the *fault rate*—the number of evictions per request. Weighted caching is similar, except that the cost to evict an item depends on the item.

For these two problems, for technical reasons and without loss of generality, we replace the assumption that all servers begin on the first requested node with the assumption that initially no servers reside on nodes, and, in response to any request, any server that has not yet served a request may be placed on the requested node at no cost.

Following a number of authors [ST], [BLS], [MMS], we are interested in strategies that are *competitive*, that is, strategies that on any sequence incur a cost bounded by some constant times the minimum cost possible for that sequence. Formally,

- r denotes an arbitrary sequence of requests.
- X denotes some on-line k-server strategy.
- k denotes the number of servers given to the on-line strategy.
- Opt denotes the (off-line) strategy producing a minimum cost solution.
- h denotes the number of servers given to Opt.
- $\mathscr{C}_r(X, k)$ denotes the (expected) cost incurred by the (randomized[3]) strategy X with k servers on request sequence r.

---

[3] We implicitly assume that the input requests are independent of the random choices made by the strategy; for other models see [BDB$^+$].

- A strategy X is *c-competitive* for a given r, h, and k, when

$$\mathscr{C}_r(X, k) \leq c \cdot \mathscr{C}_r(\text{OPT}, h) + b,$$

  where b depends on the initial positions of the optimal and on-line servers, but is otherwise independent of r.
- A strategy X is *$c(h, k)$-competitive* when X is $c(h, k)$-competitive for *all* r, h, and k. $C(h, k)$ is then called a *competitive ratio* of X.

Note that the competitiveness of a strategy is unrelated to its computational complexity.

Before we describe our results, here is a summary of the strategies relevant to our work.

- LRU, FIFO, and FWF are, respectively, the "least recently used," "first in first out," and "flush when full" paging strategies. LRU moves the server from the least recently requested, served node. FIFO, which can be obtained from LRU by ignoring served requests, moves the least recently moved server. FWF evicts all items from the fast memory (removes all servers from the graph at a cost of k) when the fast memory is full and the requested item is not in the fast memory.
- MARK is the marking algorithm, a randomized paging strategy. MARK may be described as follows: If the requested node has no server, mark all servers if none are marked, and then move and unmark a marked server chosen uniformly at random; if the requested node has a server, unmark that server.
- BALANCE is the balance algorithm, a k-server strategy. In response to request r, BALANCE moves the server from served node u minimizing $d(u, r) + W(u)$, where $W(u)$ denotes the net distance traveled so far by the server on u. BALANCE generalizes FIFO.

1.1. A Primal–Dual Strategy for Weighted Caching. In Section 2 we introduce and analyze GREEDYDUAL, a new, primal–dual, deterministic, on-line weighted-caching strategy that is (optimally) $k/(k - h + 1)$-competitive. Figure 1 contains a direct description of GREEDYDUAL.

GREEDYDUAL is of practical interest because it generalizes LRU, one of the best paging strategies, to weighted caching. GREEDYDUAL also generalizes BALANCE for weighted caching, and thus FIFO.[4]

GREEDYDUAL is of theoretical interest because its analysis is the first primal–dual analysis[5] of an on-line algorithm and because the analysis, which shows an (optimal) competitive ratio of $k/(k - h + 1)$, is the first to show a ratio less than

---

[4] The natural generalization of LRU for weighted caching can be obtained by ignoring the $L[\cdot]$ values and lowering as much as possible in the RELABEL step. BALANCE, as it specializes for weighted caching, can be obtained by ignoring the $H[\cdot]$ values and lowering as little as possible in the RELABEL step.

[5] We assume familiarity with linear programming primal–dual techniques. For an introduction, see [PS].

GREEDYDUAL
  Maintain a pair of real values $L[s] \leq H[s]$ with each server s.
  In response to each request, let v be the requested node;
    if some server s is on v **then**
      let $H[s] \leftarrow w(v)$
    else if some server s has yet to serve a request **then**
      let $L[s] \leftarrow H[s] \leftarrow w(v)$
      Place s on v.
    else
RELABEL: Uniformly lower $L[s]$ and $H[s]$ for all s so that

$$\min_s L[s] \leq 0 \leq \min_s H[s].$$

      Move any server s such that $L[s] \leq 0$ to v.
      let $L[s] \leftarrow H[s] \leftarrow w(v)$

Fig. 1. The weighted caching algorithm GREEDYDUAL.

k when $h < k$ for any k-server problem more general than paging. A consequence of this reduced ratio is that GREEDYDUAL has a constant competitive ratio provided h is any fraction of k.

We feel that the primal–dual approach, well developed for exact optimization problems, is also important for approximation problems, including on-line problems, because primal–dual considerations help reveal combinatoric structure, especially how to bound optimal costs. The primal–dual approach also has the potential to unify the arguably *ad hoc* existing on-line analyses. For instance, the analyses of LRU and FIFO [ST], of BALANCE for weighted caching [CKPV], and of MARK [FKL⁺] can all be cast as closely related primal–dual analyses. The primal–dual approach can also reveal connections to existing optimization theory. For these reasons, we take pains to make explicit the primal–dual framework behind our analysis.

Here is a sketch of our primal-dual approach. The k-server problem has a natural formulation as an integer linear program (IP) that is essentially a minimum-weight matching problem. Relaxing the integrality constraints of IP yields a linear program (LP) (which, incidentally, has optimal integer solutions). GREEDYDUAL implicitly generates a solution to the dual program (DP) of LP. The dual solution serves two purposes: GREEDYDUAL uses the structural information that the solution provides about the problem instance to guide its choices, and GREEDYDUAL uses the cost of the dual solution as a lower bound on $\mathscr{C}_r(\text{OPT}, h)$ to certify competitiveness.

Related work includes the following. Sleator and Tarjan [ST] show that LRU and FIFO are $k/(k - h + 1)$-competitive, and that this ratio is optimal for deterministic, on-line paging strategies. A similar analysis shows that FWF is also $k/(k - h + 1)$-competitive.

Fiat *et al.* [FKL$^+$], [Y1] introduce and analyze MARK, showing that it is $2H_k$-competitive ($H_k \approx \ln k$), and showing that no randomized paging strategy is better than H_k-competitive when $h = k$. McGeoch and Sleator [MS] subsequently give an H_k-competitive randomized paging strategy. Young [Y2] shows that MARK is roughly $2 \ln k/(k - h)$-competitive when $h < k$ and that no randomized strategy is better than roughly $\ln k/(k - h)$-competitive.

Manasse *et al.* [MMS] show that BALANCE is k-competitive for the general problem provided only $k + 1$ distinct nodes are requested, and that no deterministic algorithm is better than $k/(k - h + 1)$-competitive in *any* graph with at least $k + 1$ distinct nodes.

Chrobak *et al.* [CKPV] show that BALANCE is k-competitive for weighted caching. Independently of their analysis of BALANCE, Chrobak *et al.* [CKPV] formulate the k-server problem as an integral-capacity, minimum-cost, maximum-flow problem and use this formulation to give a polynomial-time algorithm to find a minimum-cost solution.

The primal–dual approach has been used extensively for exact optimization problems [PS], and is used implicitly in a number of recent analyses of approximation algorithms. Goemans and Williamson [GW] explicitly use the approach for finding approximate solutions to NP-hard connectivity problems.

1.2. A More Realistic k-Server Model. In Section 3 we give the second contribution of this paper: *loose competitiveness.* Loose competitiveness is motivated by Sleator and Tarjan's [ST] complaint that (when $h = k$) the competitive ratios of paging strategies are too high to be of practical interest. We have done simulations that suggest that, in practice, good paging strategies usually incur a cost within a small constant factor of minimum. The graph in Figure 2 plots competitive ratio $\mathscr{C}_r(X, k)/\mathscr{C}_r(\text{OPT}, k)$ versus k for a number of paging strategies on a typical sequence.[6]

We would like to keep the worst-case character of competitive analysis but somehow show more realistic competitive ratios.

- A strategy X is *loosely $c(k)$-competitive* when, for all $d > 0$, for all $n \in \mathcal{N}$, for any request sequence r, only $o(n)$ values of k in $\{1, \ldots, n\}$ satisfy

$$\mathscr{C}_r(X, k) \geq \max\{c(k) \cdot \mathscr{C}_r(\text{OPT}, k), \mathscr{C}_r(\text{OPT}, 1)/n^d\} + b,$$

  where b depends only on the starting configurations of X and OPT, and the $o(n)$ is independent of r.

That is, X is loosely $c(k)$-competitive when, for any sequence, at all but a vanishing fraction of the values of k in any range $\{1, \ldots, n\}$, either X is $c(k)$-competitive in the usual sense, or the cost to X with k servers is insignificant (or both). For instance, if a paging strategy is loosely $3 \ln \ln k$-competitive, then, for any fixed

[6] The input sequence, traced by Sites and Agarwal [SA], consists of 692,057 requests to 642 distinct pages of 1024 bytes each. The sequence 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.

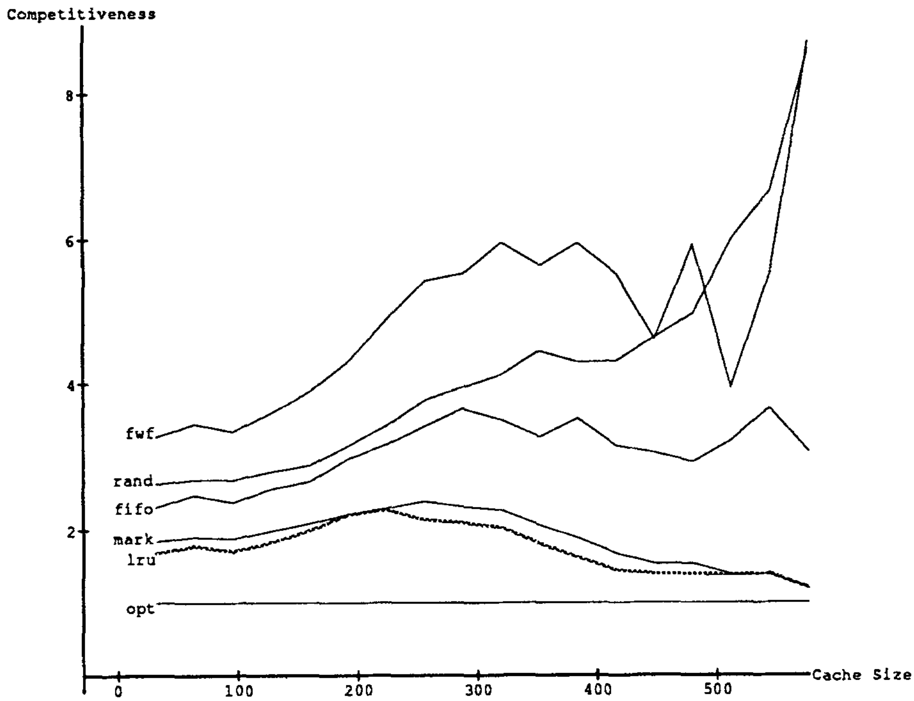

Fig. 2. Competitiveness ($\mathscr{C}_r(\cdot, k)/\mathscr{C}_r(\text{Opt}, k)$) versus k for typical r.

$d > 0$, on any sequence, for almost any choice of k in any range $\{1, 2, \ldots, n\}$, the fault rate will be either at most $1/n^d$ or at most $3 \ln \ln k$ times the minimum possible using a cache of size k (Figure 3).

This model is realistic provided input sequences are not critically correlated with k and provided we are only concerned about being near-optimal when the cost is significant. Both criteria are arguably true for most paging applications.

- A paging strategy is *conservative* if it moves no server from a node until all servers have been placed on the graph, and it moves servers at most k times during any consecutive subsequence requesting k or fewer distinct items.

Lru, Fifo, Mark, and even Fwf are conservative. Any conservative paging strategy is $k/(k - h + 1)$-competitive [Y1].

The results we obtain are as follows: Any conservative paging strategy is loosely $c(k)$-competitive provided $c(k)/\ln k \to \infty$ and both $c(k)$ and $k/c(k)$ are nondecreasing; Mark is loosely $c(k)$-competitive provided $c(k) - 2 \ln \ln k \to \infty$ and both $c(k)$ and $2 \ln k - c(k)$ are nondecreasing.

Loose competitive ratios are thus shown to be exponentially lower than standard competitive ratios.

Borodin *et al.* [BIRS] give a related work, in which the possible request sequences are quantified by the degree to which they exhibit a certain kind of locality of reference, and competitive ratios are considered as a function of this

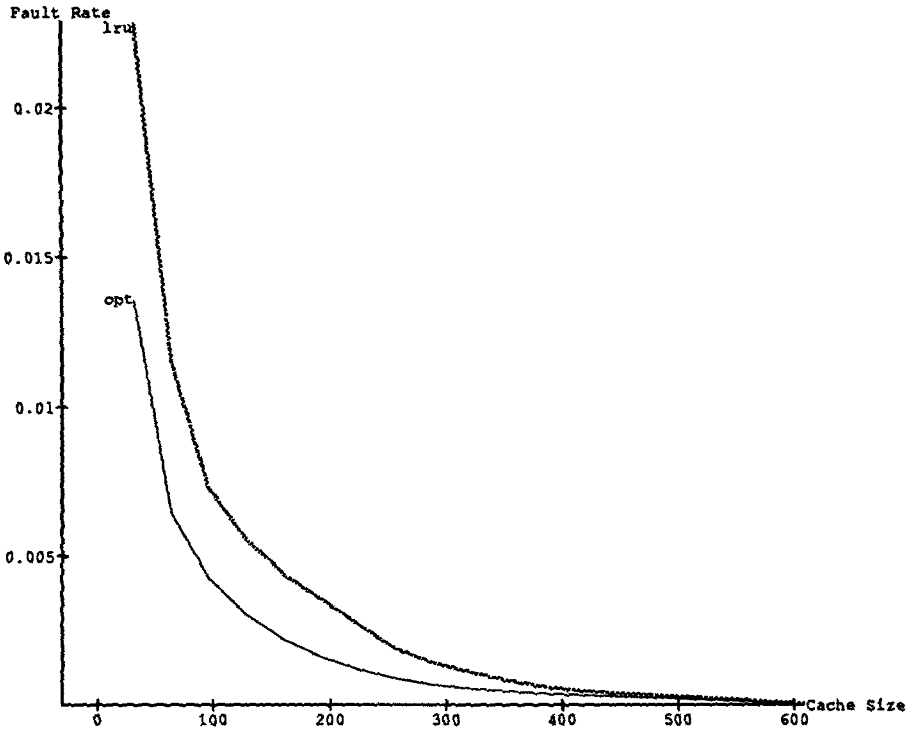

Fig. 3. Fault rate ($\mathscr{C}_r(\cdot, k)/|r|$) versus k for typical r.

parameter. The work is extended by Irani *et al.* [IKP]. The ratios shown in their model are, in most cases, much higher than the loose competitive ratios established in this paper.

2. GreedyDual. In this section we develop and analyze GreedyDual. We first develop a linear programming framework for the general k-server problem, and then we present and analyze GreedyDual as a primal–dual algorithm within this framework.

2.1. The k-Server Dual. Fix a request sequence $r_0, r_1, \ldots, r_N$, so that request i is to node r_i.

We next define IP, an integer linear program whose feasible solutions correspond to solutions of the k-server problem given by r. The variables of IP are $\{x_{ij}: 0 \leq i < j \leq N\}$, where $x_{ij} \in \{0, 1\}$ is 1 if and only if the request served by the server of request j before serving request j is request i.

After defining IP, we construct its fractional relaxation LP, and the dual DP of LP.

- IP(k) (or just IP, if k is determined by context) denotes the integer linear program

$$\text{minimize} \quad \sum_{0 \le i < j \le N} d(r_i, r_j) x_{ij}$$

$$\text{subject to} \quad \begin{cases} x(\text{out}(0)) \le k, & \\ x(\text{out}(i)) \le 1 & (1 \le i \le N-1), \\ x(\text{in}(i)) = 1 & (1 \le i \le N), \\ x_{ij} \in \{0, 1\} & (0 \le i < j \le N), \end{cases}$$

  where out(i) denotes the set $\{(i, j): i < j \le N\}$, in(i) denotes the set $\{(j, i): 0 \le j < i\}$, and $x(S) = \sum_{(i,j) \in S} x_{ij}$.

  For the weighted caching and paging problems (where initially no servers reside on the graph, and each server is allowed to serve its first request by being placed on the requested node at no cost), IP is defined as above, but we stipulate that request 0 is to an artificial node that is never requested again and that is at distance 0 to all later requests. With this stipulation the initial conditions for the general problem reduce to the initial conditions for weighted caching and paging.
- LP(k) (or just LP) denotes the relaxation of IP (obtained by replacing each constraint $x_{ij} \in \{0, 1\}$ with the constraint $0 \le x_{ij} \le 1$).
- DP(h) (or just DP) denotes the dual of LP(h):

$$\text{maximize} \quad -ha_0 - \sum_{1 \le i \le N-1} a_i + \sum_{1 \le i \le N} b_i$$

$$\text{subject to} \quad \begin{cases} b_j - a_i \le d(i, j) & (0 \le i < j \le N), \\ a_i \ge 0 & (0 \le i \le N). \end{cases}$$

- $\|(a, b)\|_h$ denotes the cost, $-ha_0 - \sum_{i \ge 1} a_i + \sum_i b_i$, of a feasible solution to DP(h).

Note that the dual constraints are independent of h, so that a dual solution is feasible independently of h.

By duality, for any feasible dual solution (a, b),

$$\mathscr{C}_r(\text{OPT}, h) \ge \|(a, b)\|_h.$$

Incidentally, a standard transformation shows that IP is equivalent to a minimum-weight, bipartite, perfect matching problem,[7] and thus that LP has

[7] It may be useful for the reader, in understanding LP, to study the equivalent minimum-weight perfect matching problem, so we briefly outline it here. Construct a weighted bipartite graph $G = (U, W, E)$, with $U = \{A_0, A_1, \ldots, A_{N-1}\}$, $W = \{B_1, \ldots, B_N\}$, $E = \{(A_i, B_j) \in U \times W: i < j\}$, and $w(A_i, B_j) = d(r_i, r_j)$. Each solution x to IP corresponds to the subset $\{e: x_e = 1\}$ of E. The cost of x equals the net weight of edges in the subset. Such subsets are exactly those such that every vertex in W touches one edge in the subset, every vertex in U except A_0 touches at most one edge, and A_0 touches at most k edges. We can leave the problem in this form, or we can convert it into a true perfect matching problem by duplicating A_0 with its edges k times and adding k copies of a new node B_∞ with zero-cost edges from every A_i.

optimal integer solutions, so that, for given r and h, the above bound is tight for some (a, b).

2.2. The Algorithm. Here are the definitions and notations specific to GREEDY-DUAL:

- A request *has a server* if the request has been served and the server has not subsequently served any other request.
- The notation i^- denotes the most recent request (up to and including request i) that resulted in the server of request i moving. We define $0^- = 0$.
- S denotes the set $\{i\colon \text{request } i \text{ has a server}\}$. (More correctly, S is a multiset, as 0 occurs in S once for each server on node r_0. Any $i > 0$ can occur only once.)
- (a, b) denotes a feasible dual solution maintained by GREEDYDUAL.

GREEDYDUAL responds to each request as follows. If the requested node has a server, it does nothing. Otherwise, it uniformly raises a subset of the dual variables enough to account for the cost of moving some server, but not so much that feasibility is violated. It then moves a server whose movement cost can be accounted for. The full algorithm is given in Figure 4.

2.3. Analysis of the Algorithm. A simple proof by induction on n shows that every $b_i \geq b_{i+1}$, that $b_{i+1} \leq w(r_i)$ for $i \in S$ (two facts that we use again later), and that the RELABEL step can in fact be performed. All other steps can be seen to be well defined by inspection, and, clearly, GREEDYDUAL produces an appropriate sequence of server movements. This establishes the correctness of GREEDYDUAL.

GREEDYDUAL(r, k)
Comment: Moves servers in response to requests $r_0, r_1, \ldots, r_N$, maintaining (a, b), a dual solution, and S, a multiset containing the currently served requests, such that (a, b) is feasible and the distance traveled by servers is at most $k/(k - h + 1)\|(a, b)\|_h - \sum_{i \in S} b_{i^-+1}$.
In response to request 0:
  let $a_{i-1} \leftarrow b_i \leftarrow 0$ for $i = 1, \ldots, N$
  let $S \leftarrow$ the multiset containing request 0 with multiplicity k
In response to each subsequent request $n > 0$:
  if node r_n has a server **then**
   STAY: **choose** $i \in S$ such that $r_i = r_n$
      let $S \leftarrow S \cup \{n\} - \{i\}$, satisfying request n
  else
  RELABEL: Uniformly raise the dual variables in the set

$$\{a_i\colon 0 \leq i \leq n-1,\ i \notin S\} \cup \{b_i\colon 1 \leq i \leq n\}$$

   so that $(\forall i \in S)\ b_{i+1} \leq w(r_i)$ but $(\exists i \in S)\ b_{i^-+1} \geq w(r_i)$.
  MOVE: **choose** $i \in S$ such that $b_{i^-+1} \geq w(r_i)$
     let $S \leftarrow S \cup \{n\} - \{i\}$, satisfying request n

Fig. 4. GREEDYDUAL as a primal–dual algorithm.

To establish that GreedyDual is $k/(k-h+1)$-competitive, we show two invariants: that the dual solution (a, b) is feasible, and that the distance traveled by servers is bounded by $k/(k-h+1)\|(a,b)\|_h - \sum_{i \in S} b_{i^-+1}$. Since every b_i is nonnegative and $\|(a,b)\|_h$ is a lower bound on $\mathscr{C}_r(\text{Opt}, h)$, this gives the result.

Lemma 2.1. GreedyDual *maintains the invariant that* (a, b) *is feasible.*

Proof. By induction on n. Clearly, (a, b) is initially feasible. The only step that changes (a, b) is Relabel. Clearly, Relabel maintains that every a_i is nonnegative. Thus the only dual constraint that Relabel might violate is of the form

$$b_j - a_i \leq d(r_i, r_j)$$

for some $0 \leq i < j \leq N$.

By inspection of the Relabel step, such a constraint can only be violated if $i \in S$ and $j \leq n$. In this case, $a_i = 0$ and $r_i \neq r_j$ because i has a server, so the constraint reduces to $b_j \leq w(r_i)$. Since $b_{i+1} \leq w(r_i)$ after the step, and $b_{i+1} \geq b_j$ (since $j > i$ and we have already established that every $b_i \geq b_{i+1}$), the constraint is maintained. □

Lemma 2.2. GreedyDual *maintains the invariant that the net distance traveled by servers is bounded by*

$$\frac{k}{k-h+1}\|(a,b)\|_h - \sum_{i \in S} b_{i^-+1}.$$

Proof. By induction on n. Clearly, the invariant is initially true.

The Stay step leaves the net distance and the bound unchanged.

The Relabel step also leaves the net distance and the bound unchanged. If $0 \notin S$, that the bound remains unchanged can be seen by inspecting the definition of $\|(a,b)\|_h$, and noting that when the dual variables are raised, $n-k$ of the a_i's, including a_0, and n of the b_i's increase. Consequently, $\|(a,b)\|_h$ is increased by $k-h+1$ times as much as any individual term, and, in the bound, the increase in the minuend exactly counterbalances the increase in the subtrahend.

If $0 \in S$, the bound remains unchanged because the constraint $b_1 \leq w(r_0) = 0$ ensures that the raise is degenerate—that the dual variables are in fact unchanged.

The Move step increases the distance traveled by $w(r_i)$, and increases the bound by $b_{i^-+1} - b_{n^-+1}$. Since $n^- = n$, $b_{n+1} = 0$, and $b_{i^-+1} \geq w(r_i)$, the bound is increased by at least $w(r_i)$, and the invariant is maintained. □

Corollary 2.3. GreedyDual *is* $k/(k-h+1)$*-competitive.*

Note that in order to implement GreedyDual, only the values $L[s_i] = w(r_i) - b_{i+1}$ and $H[s_i] = w(r_i) - b_{i^-+1}$ (for each server s_i of a request $i \in S$) need to be maintained, and that the artificial first request may be dropped, instead placing the servers on nodes when they first truly serve a request. We leave it to the reader to verify that these modifications lead to the direct description of GreedyDual given in Figure 1.

3. Loose Competitiveness. In this section we give our analyses of loose competitiveness of paging strategies. The theorems and lemmas in this section, except as noted, first appeared in [Y2] and [Y1].

Recall that a k-server strategy is *loosely $c(k)$-competitive* if, for any d, for any n, for any request sequence r, only $o(n)$ values of $k \in \{1, \ldots, n\}$ satisfy

$$\mathscr{C}_r(X, k) \geq \max\{c(k)\mathscr{C}_r(\text{Opt}, k), \mathscr{C}_r(\text{Opt}, 1)/n^d\} + b.$$

The following terminology is essentially from Fiat *et al.*'s [FKL$^+$], [Y1], [Y2] analysis of the marking algorithm. Given a sequence r and a positive integer k:

- The *k-phases* of r are defined as follows. The first k-phase is the maximum prefix of r containing requests to at most k distinct nodes. In general, the ith k-phase is the maximum substring[8] of r beginning with the request, if any, following the $(i-1)$st k-phase and containing requests to at most k distinct nodes.

  Thus the $(i+1)$st k-phase begins with the (new) request that would cause Fwf to flush its fast memory for the ith time.
- $\mathscr{P}_r(k)$ denotes the number of k-phases, minus 1.
- A *new request* (for a given k) in a k-phase (other than the first) is a request to a node that is not requested previously in the k-phase or in the previous k-phase. Thus in two consecutive k-phases, the number of distinct nodes requested is k plus the number of new requests in the second k-phase.
- $\mathscr{N}_r(k)$ denotes the *average* number of new requests per k-phase of r other than the first.

  Thus the total number of new requests in r for a given k is $\mathscr{N}_r(k) \cdot \mathscr{P}_r(k)$.

Our analysis has two parts. In the first part (Theorem 3.2) we show that, for any sequence, few values of k yield both a large number of k-phases and a low average number of new requests per k-phase.

In the second part we show (Lemma 3.4) that, for the paging strategies that interest us, for a given sequence and k, the cost incurred by the strategy is proportional to the number of k-phases, and the competitiveness is inversely related to the average number of new requests per k-phase. Consequently (Corollary 3.5), by the first part of the analysis, few values of k yield both a high cost and a high competitiveness.

The key technical insight (Lemma 3.1) for the first part of the analysis is that if, for a given k, the average number of new requests per k-phase is low, then, for k' just slightly larger than k, the number of k'-phases is a fraction of the number of k-phases.

[8] By "substring" we mean a subsequence of consecutive items.

LEMMA 3.1. *Fix a sequence r. For any k, and any* $k' \geq k + 2\mathcal{N}_r(k)$,

$$\mathcal{P}_r(k') \leq \tfrac{3}{4}\mathcal{P}_r(k).$$

PROOF. Let $p_0, \ldots, p_{\mathcal{P}_r(k)}$ denote the k-phase partitioning of r.

At least half (and thus at least $\lceil \mathcal{P}_r(k)/2 \rceil$) of the $\mathcal{P}_r(k)$ k-phases $p_1, \ldots, p_{\mathcal{P}_r(k)}$ have a number of new requests not exceeding $2\mathcal{N}_r(k)$. Denote these by $p_{i_1}, \ldots, p_{i_{\lceil \mathcal{P}_r(k)/2 \rceil}}$.

If we modify the k-phase partitioning of r by joining $p_{i_j - 1}$ and p_{i_j} for odd j, we obtain a coarser partitioning of r into at most $\mathcal{P}_r(k) - \lceil \mathcal{P}_r(k)/4 \rceil$ pieces. In the coarser partitioning, each piece resulting from a join references at most $k + 2\mathcal{N}_r(k) \leq k'$ distinct nodes, while each piece remaining unchanged from the k-phase partioning references at most $k \leq k'$ distinct nodes.

If we now consider the k'-phase partitioning, we find that each k'-phase must contain the final request of at least one of the pieces in the coarser partition, because if a k'-phase begins at or after the beginning of a subsequence of requests to at most $k + 2\mathcal{N}_r(k)$ distinct nodes, it will continue at least through the end of the subsequence.

Thus $\mathcal{P}_r(k') \leq \mathcal{P}_r(k) - \lceil \mathcal{P}_r(k)/4 \rceil \leq \tfrac{3}{4}\mathcal{P}_r(k)$. □

From this we show that there are not too many values of k yielding both a low average number of new requests per k-phase and a significant number of k-phases:

THEOREM 3.2. *For any* $\varepsilon > 0$, $M > 0$, *and any sequence r, the number of k satisfying*

$$\mathcal{N}_r(k) \leq M \quad \text{and} \quad \mathcal{P}_r(k) \geq \varepsilon\mathcal{P}_r(1) \tag{1}$$

is $O(M \ln 1/\varepsilon)$.

PROOF. Let s be the number of k satisfying the condition. We can choose $l = \lceil s/\lceil 2M \rceil \rceil$ such k so that each chosen k differs from every other by at least $2M$. Then we have $1 \leq k_1 \leq k_2 \leq \cdots \leq k_l$ such that, for each i,

$$\mathcal{N}_r(k_i) \leq M, \tag{2}$$

$$k_{i+1} - k_i \geq 2M, \tag{3}$$

and

$$\mathcal{P}_r(k_l) \geq \varepsilon\mathcal{P}_r(1). \tag{4}$$

Then, for any i, by (2) and (3), $k_{i+1} \geq k_i + 2\mathcal{N}_r(k_i)$, so, by Lemma 3.1, $\mathcal{P}_r(k_{i+1}) \leq \tfrac{3}{4}\mathcal{P}_r(k_i)$. Inductively, $\mathcal{P}_r(k_l) \leq (\tfrac{3}{4})^{l-1}\mathcal{P}_r(1)$.

This, and (4), imply $(\frac{3}{4})^{l-1} \geq \varepsilon$, so

$$\left\lceil \frac{s}{\lceil 2M \rceil} \right\rceil - 1 = l - 1 \leq \ln_{4/3} \frac{1}{\varepsilon}.$$

This implies the bound on s. □

This establishes the first part of the analysis.

We begin the second part by showing that OPT's cost per k-phase is at least proportional to the average number of new requests per k-phase.

LEMMA 3.3. *For arbitrary paging request sequence r, and arbitrary k, h > 0,*

$$\frac{\mathscr{C}_r(\text{OPT}, h)}{\mathscr{P}_r(k)} \geq \frac{k - h + \mathscr{N}_r(k)}{2}. \tag{5}$$

(We use only the case $k = h$, but prove the general case.)

PROOF. Let m_i $(1 < i \leq \mathscr{P}_r(k))$ denote the number of new requests in the ith k-phase, so that $\mathscr{N}_r(k)\mathscr{P}_r(k) = \sum_{i>1} m_i$. During the $(i-1)$st and ith k-phases, $k + m_i$ distinct nodes are referenced. Consequently, any strategy for r with h servers makes at least $k + m_i - h$ server movements during the two phases. Thus the total cost for the strategy is at least

$$\max\left\{\sum_{i\geq 1} (k - h + m_{2i+1}), \sum_{i \geq 1} (k - h + m_{2i})\right\} \geq \frac{(k - h + \mathscr{N}_r(k))\mathscr{P}_r(k)}{2}. \qquad \square$$

We next show that the strategies that interest us incur a cost proportional to k times the number of k-phases, and (using the above lemma) that the strategies have competitiveness inversely related to the average number of new requests per k-phase.

LEMMA 3.4. *Let X denote any conservative paging strategy. Let* MARK *denote the marking algorithm. Then*

$$\mathscr{P}_r(k) \geq \frac{\mathscr{C}_r(X, k)}{k}, \tag{6}$$

$$\mathscr{N}_r(k) \leq 2k \frac{\mathscr{C}_r(\text{OPT}, k)}{\mathscr{C}_r(X, k)}, \tag{7}$$

$$\mathscr{N}_r(k) \leq k \exp\left(1 - \frac{1}{2} \frac{\mathscr{C}_r(\text{MARK}, k)}{\mathscr{C}_r(\text{OPT}, k)}\right). \tag{8}$$

PROOF. Bound (6) follows directly from the definition of conservativeness.

Bound (7) follows from bound (6) and bound (5) of Lemma 3.3, applied with $h = k$.

Finally, we prove bound (8). Fix a request sequence r, and let m_i $(1 < i \le \mathscr{P}_r(k))$ denote the number of new requests in the ith k-phase. Fiat *et al.* [FKL$^+$] show[9] that $\mathscr{C}_r(\text{MARK}, k) \le \sum_i m_i(H_k - H_{m_i} + 1)$.

Since $H_a - H_b = \sum_{i=a+1}^{b}(1/i) \le \int_a^b(1/x)\,dx = \ln(a/b)$, letting $f(m) = m(1 + \ln(k/m))$, $\mathscr{C}_r(\text{MARK}, k) \le \sum_i f(m_i)$. Since f is convex, $\mathscr{C}_r(\text{MARK}, k) \le \mathscr{P}_r(k) f(\mathscr{N}_r(k))$. Applying bound (3.3), $\mathscr{C}_r(\text{MARK}, k) \le 2\mathscr{C}_r(\text{OPT}, k) f(\mathscr{N}_r(k))/\mathscr{N}_r(k)$, which is equivalent to bound (8). □

We have established (Theorem 3.2) that for any sequence there are few values of k yielding many k-phases and a low average number of new requests per k-phase.

We have established (Lemma 3.4) that for the strategies we are interested in, the cost they incur with k-servers on a sequence is proportional to the number of k-phases, while the competitiveness is inversely related to the average number of new requests per k-phase.

Finally, we combine the two parts to show that, for any sequence, there are few values of k for which our strategies incur high cost and high competitiveness. This establishes loose competitiveness.

COROLLARY 3.5. *Let X denote any conservative paging strategy and let $c\colon \mathscr{N}^+ \to \mathscr{R}^+$ be a nondecreasing function. X is loosely $c(k)$-competitive provided that $k/c(k)$ is nondecreasing and*

$$\frac{c(k)}{\ln k} \to \infty, \tag{9}$$

while MARK *is loosely $c(k)$-competitive provided $2 \ln k - c(k)$ is nondecreasing and*

$$c(k) - 2 \ln \ln k \to \infty. \tag{10}$$

PROOF. Let X denote either any conservative paging stragegy, in which case we assume condition (9) and that $k/c(k)$ is nondecreasing, or MARK, in which case we instead assume condition (10) and that $2 \ln k - c(k)$ is nondecreasing.

We show that, for any $d > 0$, $n > 0$, and request sequence r, the number of

[9] MARK moves a server chosen uniformly at random from those on nodes not yet requested in the current k-phase. Briefly, the analysis of MARK classifies nonnew requests within a phase into *repeat* requests (to nodes already requested this phase) and *old* requests (to nodes requested in the previous phase but not yet in this phase); the expected cost for the ith old request is bounded by $m/(k - i + 1)$ because at least $k - m - i + 1$ of the $k - i + 1$ nodes requested in the previous phase but not in this phase are served, each with equal probability.

violators $k \in \{1, \ldots, n\}$ is $o(n)$, where a violator is a k such that

$$\mathscr{C}_r(X, k) \geq \max\left\{c(k)\mathscr{C}_r(\text{Opt}, k), \frac{\mathscr{C}_r(\text{Opt}, 1)}{n^d}\right\}.$$

Let k be a violator. Then bound (6) implies

$$\mathscr{P}_r(k) \geq \frac{\mathscr{C}_r(X, k)}{k} \geq \frac{\mathscr{C}_r(\text{Opt}, 1)}{n^{d+1}} = \frac{1}{n^{d+1}}\mathscr{P}_r(1). \tag{11}$$

Bound (7) and the monotonicity of $k/c(k)$ imply

$$\mathscr{N}_r(k) \leq 2k\frac{\mathscr{C}_r(\text{Opt}, k)}{\mathscr{C}_r(X, k)} \leq \frac{2k}{c(k)} \leq \frac{2n}{c(n)}. \tag{12}$$

Since each violator k satisfies (11) and (12), by Theorem 3.2, the number of violators is $O((\ln n^{d+1})n/c(n))$. This is $o(n)$ by assumption (9).

If $X = \text{Mark}$, then bound (8) and the monotonicity of $2 \ln k - c(k)$ imply, for each violator k, that

$$\begin{aligned} \mathscr{N}_r(k) &\leq k \exp\left(1 - \frac{1}{2}\frac{\mathscr{C}_r(\text{Mark}, k)}{\mathscr{C}_r(\text{Opt}, k)}\right) \\ &\leq k \exp\left(1 - \frac{c(k)}{2}\right) \\ &\leq n \exp\left(1 - \frac{c(n)}{2}\right), \end{aligned} \tag{13}$$

so that by bounds (11) and (13) and Theorem 3.2 the number of violators is $O((\ln n^{d+1})n \exp(1 - c(n)/2))$. This is $o(n)$ by assumption (10). □

4. Concluding Remarks. We conclude in this section with comments about further avenues of research.

Historically, the role of duality in solving optimization problems is well explored: dual solutions are used to guide the construction of primal solutions and to certify optimality. For on-line problems such as the k-server problem, duality can serve a similar role; the differences are that the solutions we seek are approximate, and that the problem we want to solve is on-line. For on-line problems, it seems natural to seek a sequence of *closely related* dual solutions, one for each prefix of the request sequence.

For those interested in extending our approach to the general k-server problem we give the following brief hints. Solutions with monotonic b_i's are not sufficient to give good bounds: add constraints $b_{i+1} \le b_i$ to the dual problem and reformulate the primal; in the new primal request sequences can be much cheaper than in the old. Raising *all* of the b_i's is probably not a good idea: consider a request sequence with requests from two infinitely separate metric spaces; a b_i should change only when a request is made to the metric space of r_i. Finally, a promising experimental approach: if r is a *worst-case* sequence for a k-competitive algorithm X, and the bound $\mathscr{C}_r(X, k) \le k\|(a, b)\|_k$ is sufficient to establish competitiveness, then (a, b) must be optimal; thus by examining *optimal* dual solutions for worst-case sequences, we may discover the special properties of the (generally nonoptimal) dual solutions that we seek for such an analysis. A similar technique has been tried for potential functions, but in that case each experiment is much less informative: it reveals only a single number, not an entire dual solution.

There is a suggestive similarity between potential function and primal–dual techniques [Y1]. Briefly, both can be viewed as transforming the costs associated with operations so that a sum of local inequalities gives the necessary global bound. This connection might yield some insight into the special nature of primal–dual analyses for on-line problems.

Open questions remain concerning loose competitiveness for paging. Theorem 3.2 of [Y1] is shown to be tight, and consequently the analysis of loose competitiveness for FWF is shown to be tight. No lower bounds on the loose competitive ratios of LRU, FIFO, or MARK have been shown.

Finally, two challenges: Find a randomized algorithm for weighted caching that is better than k-competitive, and show reduced loose competitiveness for a weighted-caching algorithm. A possible hint: The concept of "new requests" used in analyzing MARK and showing loose competitiveness of paging strategies may be captured by an algorithm that mimics GREEDYDUAL, but increases each b_i at only half the rate that GREEDYDUAL does, and increases each a_i only as much as necessary to maintain the dual constraints.

References

[BDB+] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Proc. 22nd Annual ACM Symp. on Theory of Computing*, Baltimore, MD, May 1990, pages 379–386. Also in *Algorithmica*, **11**:2–14, January 1994.

[BIRS] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *Proc. 23rd Annual ACM Symp. on Theory of Computing*, New Orleans, LA, May 1991, pages 249–259.

[BLS] A. Borodin, M. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Proc. 19th Annual ACM Symp. on Theory of Computing*, New York, May 1987, pages 373–382. Also to appear in *J. Assoc. Comput. Mach.*

[CKPV] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. new results on server problems. *SIAM J. Discrete Math.*, **4**(2):172–181, May 1991.

[Cl] M. Chrobak and L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, **20**(1):144–148, February 1991.

[FKL+] 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. *J. Algorithms*, to appear.

[GW] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *Proc. 3rd Annual ACM–SIAM Symp. on Discrete Algorithms*, Orlando, FL, January 1992, pages 317–324.

[IKP] S. Irani, A. R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *Proc. 3rd Annual ACM–SIAM Symp. on Discrete Algorithms*, Orlando, FL, January 1992, pages 228–236.

[M] L. A. McGeoch. Algorithms for Two Graph Problems. Ph.D. Thesis, Carnegie Mellon University, 1987.

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

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

[PS] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

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

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

[Y1] N. Young. Competitive paging and dual-guided algorithms for weighted caching and matching. (Thesis) Technical Report CS-TR-348-91, Computer Science Department, Princeton University, October 1991.

[Y2] N. Young. On-line caching as caches size varies. *Proc. 2nd Annual ACM–SIAM Symp. on Discrete Algorithms*, San Francisco, CA, January 1991, pages 241–250.