COMPETITIVE PAGING AND DUAL-GUIDED ON-LINE WEIGHTED CACHING AND MATCHING ALGORITHMS

Neal Young

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE, PRINCETON UNIVERSITY

October, 1991

© Copyright by Neal Young 1991
All Rights Reserved

Abstract

This thesis presents research done by the author on competitive analysis of on-line problems.

An on-line problem is a problem that is given and solved one piece at a time. An on-line strategy for solving such a problem must give the solution to each piece knowing only the current piece and preceding pieces, in ignorance of the pieces to be given in the future.

We consider on-line strategies that are *competitive* (guaranteeing solutions whose costs are within a constant factor of optimal) for several combinatorial optimization problems: paging, weighted caching, the k-server problem, and weighted matching.

We introduce variations on the standard model of competitive analysis for paging: allowing randomization, allowing *resource-bounded* lookahead, and *loose competitiveness*, in which performance over a range of fast memory sizes is considered and noncompetitiveness is allowed provided the fault rate is insignificant. Each variation leads to substantially better competitive ratios.

We present a general technique for competitive analysis of linear optimization problems: competitive analyses are obtained by using linear programming duality to obtain bounds on the optimal cost. The technique is implicit in previous work on paging, weighted caching, and weighted matching. We generalize the implicit previous use of the technique, obtaining the *greedy dual algorithm for weighted caching*. The strategy generalizes the least-recently-used and first-in-first-out algorithms for paging and the balance algorithm for weighted caching. The analysis strengthens a previous analysis of the balance algorithm for weighted caching.

We explore the linear programming dual of the k-server problem, showing that the k-server problem is a special case of on-line minimum-weight matching, revealing close relationships between on-line weighted caching and assignment algorithms, and showing how duality can yield potential function analyses.

Acknowledgements

Thanks to Lori Ferguson, Rick and Claire Kenyon, and Anthony Tomasic for their friendship.

This research was supported by the Hertz Foundation, by Princeton University, by Digital Equipment Corporation's Systems Research Center in Palo Alto, by NSF Grants DCR-86-05962 and CCR-89-20505, and by the ONR Grant N00014-87-K-0467.

Contents

Chapter 1

Introduction

In this thesis we present research done by the author on competitive analysis of on-line problems.

In Section 1.1 of this chapter we introduce on-line problems and competitive analysis.

In Section 1.2 we briefly summarize the thesis. In the succeeding sections we summarize the concepts and results relevant to the individual chapters.

1.1 On-Line Problems and Competitive Analysis

An on-line problem is a problem that is given and solved one piece at a time. An on-line strategy for solving such a problem must give the solution to each piece knowing only the current piece and preceding pieces, in ignorance of the pieces to be given in the future.

We use the following terminology from standard competitive analysis:

- A strategy is *c-competitive* if the cost of the solution produced by the strategy is bounded by $c \cdot \mathrm{opt} + b$, where opt is the cost of the optimal solution and b depends only on the starting configurations of the on-line and optimal (off-line) strategies.
- The *competitive ratio*, or *competitiveness*, of the strategy is the infimum of such c.

In this thesis we consider competitive on-line strategies for several combinatorial optimization problems: paging, weighted caching, the k-server problem, and weighted matching.

1.2 Summary

In Chapter 2, we introduce new variations on the standard model of competitive analysis for paging: allowing randomization, allowing *resource-bounded* lookahead, and *loose competitiveness*, in which competitive ratios over a range of fast memory sizes are considered and noncompetitiveness is allowed provided the fault rate is insignificant. All of these variations lead to substantially reduced competitive ratios.

In the remaining chapters we present and study a general technique for competitive analysis of linear optimization problems: we obtain competitive analyses by using linear programming duality to obtain bounds on the optimal cost. The technique, which we refer to as the *dual bounding technique*, is implicit in previous work on paging, weighted caching, and weighted matching.

In Chapter 3, we present and analyze the *greedy dual algorithm*, a new, deterministic, on-line strategy for weighted caching. The strategy generalizes the least-recently-used, first-in-first-out, and marking algorithms for paging and the balance algorithm for weighted caching. We analyze the strategy with the dual bounding technique. The analysis strengthens and generalizes a previous analysis of the balance algorithm for weighted caching.

In Chapter 4, we explore the linear programming dual of the k-server problem. We show that the k-server problem is a special case of the assignment (minimum-cost, perfect, bipartite matching) problem, so that the dual problem is a special case of the well-studied assignment dual.

We show how existing paging strategies implicitly use the dual, and how the greedy dual algorithm generalizes this implicit use of duality.

We present a direct, intuitive interpretation of the k-server dual, and use it to give an intuitive analysis of the balance algorithm for weighted caching. We discuss how duality transformations are similar to potential function analyses, and derive a potential function analysis of the balance algorithm for weighted caching from the duality analysis.

We discuss how the greedy dual algorithm is related to a previously analyzed on-line assignment algorithm, and we discuss the structure of optimal k-server dual solutions.

In Chapter 5, we use the dual bounding technique to reproduce the analyses of several existing weighted matching strategies. The purpose is to study the general applicability of

the technique.

We conclude the thesis with Chapter 6, in which we make some final comments, consider directions for future work, and summarize our results.

1.3 Paging

In this section we introduce concepts and results relevant to Chapter 2. We define the paging problem and relevant terminology, and summarize the pertinent paging strategies. We summarize old results on deterministic paging strategies, our new results on variations, and related work by other authors.

- The *paging* problem is as follows: One is given a collection of items (representing *pages*), a *fast memory* (or equivalently a *cache*) capable of holding a fixed number k of these items, and a sequence of requests for items. In response to each request, the requested item must be moved into the fast memory if it is not already present. If k items are already in the fast memory, some item (or items) must be evicted to make room for the new item. The problem is to choose which items to evict to minimize the number of evictions.

- A paging strategy is *on-line* if a strategy chooses which item to evict without knowledge of future requests.

- A *schedule* satisfying a sequence of requests is an appropriate sequence of evictions.

- The *cost of the schedule* is the number of evictions.

- The *cost of a strategy* on a sequence for a given k is the cost of the schedule produced by the strategy, unless the strategy is randomized. In this case the cost of the schedule is a random variable, and the cost of the strategy refers to the expected cost of the schedule.

- $\mathcal{C}_r(X,k)$ denotes the cost of strategy X, using a fast memory of size k, on sequence r.

We consider the following paging algorithms:

OPT — Belady's algorithm [Bel66], which yields an optimal (minimum cost) schedule for paging by evicting the item whose next request is further in the future.

LRU — Least-recently-used, which evicts the item that has been requested least recently.

FIFO — First-in-first-out, which evicts the item that has been in the fast memory the longest.

FWF — Flush-when-full, which evicts all items when space is needed.

MARK — The marking algorithm [FKL+88], which evicts an item chosen uniformly at random from the set of items not in the fast memory of FWF when space is needed.

- A *conservative* paging strategy is one that, with a fast memory of size k,
  - incurs no evictions before $k+1$ distinct items have been requested, and
  - incurs at most k evictions during any subsequence of requests to at most k distinct items.

The reader may verify that all of the above strategies except OPT are on-line, and all are conservative.

Throughout this thesis, X refers to a conservative, on-line paging strategy and r to a sequence of $r_0 r_1 \cdots r_N$ of items, each of which represents a request. If "r_i" is used to denote an item which is the subject of some request, it is intended that the request in question is the ith. For instance, if $r = aba$, then, in "Let r_i denote the last item requested," i could (in principle) be taken to be either 0 or 2; it is intended that i be taken to be 2.

Generally, k refers to the fast memory size of an on-line strategy, and h to the fast memory size of OPT. The competitive ratio of a strategy generally depends on k and h. Generally we assume $h \leq k$, sometimes restricting to the special case $h = k$.

1.3.1 Previous Results

The first competitive analysis for paging, given by Sleator and Tarjan [ST85], showed that LRU and FIFO have competitive ratio $k/(k-h+1)$ and that no deterministic, on-line paging strategy has a better competitive ratio. Sleator and Tarjan also showed that the least-frequently-used paging strategy, for instance, is not competitive.

1.3.2 New Results

Randomization can help on-line algorithms. When $h = k$, deterministic, on-line strategies are at best k-competitive, whereas MARK is $2H_k$-competitive.[1] On the other hand, no randomized, on-line strategy is less than H_k-competitive.

For $h < k$, MARK is roughly $2 \ln \frac{k}{k-h}$-competitive, and no randomized, on-line strategy is less than roughly $\ln \frac{k}{k-h}$-competitive.

Resource-bounded lookahead helps on-line algorithms:

- A strategy is *on-line with resource-bounded lookahead* l if its choices depend only on the past requests and the maximal prefix of the future requests for which it will incur l evictions.

A natural adaptation of MARK taking advantage of resource-bounded lookahead l is $2(\ln(k/l) + 1))$-competitive when $h = k$, while a deterministic version of this algorithm is in turn $\max\{2k/(k - h + l), 2\}$-competitive. These ratios are within a factor of about 2 of optimal. These results further explore the the trade-off, implicit in Sleator and Tarjan's analysis, between knowing the future and having a larger fast memory.

Relaxing the model helps on-line algorithms:

- A strategy X has *loose competitive ratio* $c(k)$ if, for any sequence r and numbers n and d, for all but $o(n)$ values of $k \in \{1, \ldots, n\}$,

$$\mathcal{C}_r(X, k) \leq \max\{c(k) \cdot \mathcal{C}_r(\text{OPT}, k), \epsilon \cdot \mathcal{C}_r(\text{OPT}, 1)\} + b,$$

  where $\epsilon = 1/n^d$ and b depends only on the starting configurations of X and OPT.

Intuitively, if X is loosely $c(k)$-competitive, then, for almost any fast memory size k, the competitive ratio is at most $c(k)$ or the cost is insignificant.

For nondecreasing $c(k)$, conservative paging strategies such as LRU, FIFO, and FWF have loose competitive ratio $c(k)$ provided $c(k)/\ln k$ is unbounded and $k/c(k)$ is nondecreasing, while MARK has loose competitive ratio $c(k)$ provided $c(k) - 2 \ln \ln k$ is unbounded and $2 \ln k - c(k)$ is nondecreasing. FWF is not loosely $c(k)$-competitive for any $c(k)$ which is $O(\ln k)$.

[1] $H_k = 1 + 1/2 + \cdots + 1/k \approx \ln k$.

1.3.3 Related Results

MARK is similar in spirit to the randomized on-line algorithm given by Borodin, Linial, and Saks for metrical task systems [BLS87].

The analysis of MARK when $h = k$, and the corresponding H_k lower bound on competitiveness for randomized paging strategies, were discovered in the summer of 1988 by three independent groups: Fiat, Karp, and Luby; McGeoch and Sleator; and the author. The results were published jointly [FKL+88].

McGeoch and Sleator [MS89] subsequently developed the partitioning algorithm, a randomized, on-line, H_k-competitive paging algorithm.

Kalyanasundaram and Pruhs [KP91] consider a form of lookahead for on-line assignment (min-cost, bipartite, perfect matching). Instead of receiving the input in individual pieces, the on-line algorithm receives the pieces in groups. If the assignment problem instance arrives in t groups, they give an on-line algorithm with competitive ratio $2t - 1$.

Ron Graham, in his talk "How Much of the Future is Worth Knowing?" at the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms in January of 1991, mentioned results concerning lookahead in the k-server problem with excursions. The gist of his talk was that, in contrast to the k-server problem without excursions, lookahead can sometimes help the competitive ratio of on-line algorithms with excursions.

Borodin, Irani, Raghavan, and Schieber [BIRS91] considered relaxing the standard model by restricting request sequences to paths in an "access graph". Their relaxation is intended to study locality of reference in paging. They characterize resulting competitive ratios in terms of the structure of the access graph: if the access graph is a tree of $k + 1$ nodes, for instance, the competitive ratio of LRU is shown to be the number of leaves in the tree, minus 1.

Our model for randomized strategies assumes the requests are independent of the specific random choices of the strategy. This is false in some situations: if the time to service a request can influence later requests, for instance. More appropriate models for this situation, and general relations between randomized and deterministic competitiveness, were considered by Ben-David, Borodin, Karp, Tardos, and Wigderson [BDBK+90].

1.4 The Weighted Caching and K-Server Problems

In this section we introduce two generalizations of the paging problem: the weighted caching and k-server problems, which we study in Chapter 3. We enumerate the weighted caching strategies that we study in the chapter, and we summarize relevant previous results, our new results, and related work by other authors.

- The *weighted caching problem* is a generalization of the paging problem in which the cost to evict an item r_i is a nonnegative function $w(r_i)$ of the item.
- The *k-server problem* is a further generalization in which the cost is a nonnegative function $d(r_i, r_j)$ of the item r_i evicted *and* the item r_j requested, and in which the fast memory is assumed to be initially full. Except for the special case of weighted caching, d is assumed to be metric[2].

The k-server problem may be transformed as follows into a network problem: Given n items, a fast memory of size k, and a request sequence r, form a complete, directed graph with the n items as vertices and with the length $d(r_i, r_j)$ of edge (r_i, r_j) equal to the cost of evicting item r_i from the fast memory and bringing in item r_j. Initially, place k *servers* on the vertices corresponding to the items initially in the fast memory. Transform each subsequent request for an item into a request for *service* at the corresponding vertex of the graph: when a vertex r_i is requested, if no server resides on vertex r_i, then a server must be moved to vertex r_i from some vertex r_j at a cost of $d(r_i, r_j)$. This is the standard form of the k-server problem.

For simplicity, we assume that all servers start on the first requested node, r_0.[3] When we consider paging or weighted caching as restrictions of the k-server problem, when we wish to allow the on-line strategy to start with an empty fast memory, we assume that request 0 is to an artificial vertex $\mathbf{o}$ *s.t.* $d(\mathbf{o}, \cdot) \equiv 0$.

We consider the following weighted caching and k-server algorithms:

OPT — The algorithm that produces an optimal k-server or weighted caching schedule.

[2]Symmetric, satisfying both the triangle inequality and $d(r_i, r_i) = 0$.

[3]This assumption gives the on-line strategy less of a disadvantage than the usual assumption that the strategy may choose the starting positions of the servers. The assumption is, however, without loss of generality, in that it increases the cost of OPT by at most an additive constant.

BALANCE — The balance algorithm [McG87, MMS88, MMS90, CKPV90] for k servers. In response to request r, BALANCE moves the server s which minimizes $W_s + d(s, r)$, where W_s is the distance traveled by s so far, and $d(s, r)$ is the length of the edge from the node served by s to r.

GREEDYDUAL — The greedy dual algorithm [You91, You] for weighted caching. The algorithm maintains values on the servers. Initially the value of a server is the weight of the node it serves. When an unserved vertex is requested, the server values are decreased by the minimum server value, some zero-valued server is moved, and its value is raised to the weight of its new vertex. When a served vertex is requested, the server value is reset anywhere between its current value and the weight of its vertex.

1.4.1 Previous Results

The k-server problem is a natural abstraction of a number of on-line scheduling problems and has been studied a great deal in the past several years. McGeoch [McG87], and by Manasse, McGeoch, and Sleator [MMS88, MMS90] give a 2-competitive, *residue*-based 2-server algorithm. Irani and Rubinfeld [IR] present a 10-competitive variant of BALANCE for 2 servers. Chrobak and Larmore [CL] present a 2-competitive 2-server algorithm based on the concept of the *closure* of a metric space. Berman, Karloff, and Tardos show that HARMONIC (a natural, memoryless, randomized algorithm which moves a server with probability inversely proportional to the cost of the edge it must traverse) is competitive for 3 servers.

Currently the most general analyses are the following: Coppersmith, Doyle, Raghavan, and Snir [CDRS90] use properties of random walks to show that HARMONIC is k-competitive in a large class of graphs. Chrobak and Larmore [CL91a] give a natural, k-competitive, deterministic algorithm for k servers on trees.[4] Both of these analyses show that the respective algorithms are k-competitive for weighted caching.

Manasse, McGeoch, and Sleator [MMS88, MMS90] show that in any metric space (or graph with symmetric edge weights satisfying the triangle inequality) with at least $k + 1$ distinct points, no deterministic, on-line strategy is better than $(k/(k - h + 1))$-competitive.

[4] "On trees" means that the distances in the graph are shortest path distances in some spanning tree of the graph.

BALANCE has been considered previously by a number of authors. McGeoch [McG87], and Manasse, McGeoch, and Sleator [MMS88, MMS90] show that BALANCE is k-competitive (when $h = k$) for the general k-server problem provided the number of distinct vertices requested is $k + 1$. Chrobak, Karloff, Payne, and Vishwanathan [CKPV90] show that BALANCE is k-competitive (when $h = k$) for weighted caching.

1.4.2 New Results

GREEDYDUAL, a new algorithm that generalizes LRU, FWF, MARK, and BALANCE, is $(k/(k-h+1))$-competitive for weighted caching. This result is the first result we know of showing reduced competitiveness when $h < k$ for any problem other than paging.

GREEDYDUAL is motivated by the discovery of a general technique, called the *dual bounding technique*, implicit in the analyses of the strategies it generalizes.

The technique stems from an answer to the question "How can we obtain a bound on the optimal cost in order to show competitiveness?" The answer is "Formulate the problem of finding the optimal solution as a linear program, so that each solution to the problem yields a feasible solution to the linear program of equal cost. The cost of any feasible solution to the dual of this linear program is a bound on the optimal cost." The technique is to produce such a lower bound and correlate it with the on-line algorithm to show competitiveness.

1.4.3 Related Results

Karlin, Manasse, McGeoch, and Owicki [KMMO90] considered randomized algorithms for 2 servers on isosceles triangles: 3-node graphs with nonnegative symmetric edge weights 1, d, and d, with $d \geq 1/2$. Lower and upper bounds on the optimal competitiveness of randomized, on-line strategies were shown. For integer $d > 1$ the lower bound exceeds H_2.

To see the relevance to weighted caching, note that any server problem with requests to the vertices of a triangle is equivalent to a 3-node, nonnegatively weighted caching problem.[5]

[5]Given a triangle with sides of length l_1, l_2, and l_3, choose w_1, w_2, and w_3 to satisfy the 3 linear equations $l_i = \sum_{j \neq i} w_j/2$ for $i = 1, 2, 3$. Then the triangle inequalities $l_i \leq \sum_{j \neq i} l_j$ for $i = 1, 2, 3$ are equivalent to the inequalities $w_j \geq 0$ for $j = 1, 2, 3$. Instead of charging l_i for a server to traverse edge i of the triangle, imagine charging $w_j/2$ when a server enters or leaves vertex j. The charges are equivalent, and the latter is equivalent to a 3-node weighted caching problem with nonnegative weights w_1, w_2, and w_3. Note that this construction may be reversed to obtain a triangular server problem from any 3-node, nonnegatively-weighted caching problem.

Thus, for some server problems, randomized algorithms can not be as competitive as for the paging problem.

More recently, lower bounds for randomized algorithms for the general k-server problem have been studied by Karloof, Rabani, and Ravid [KRR91].

GreedyDual appears to be closely related to Chrobak and Larmore's algorithm for k servers on trees [CL91a] as the algorithm applies to the weighted caching problem.

Duality has been used to obtain lower bounds in other contexts. Yao [Yao82] observed that, as a consequence of Von Neumann's min-max theorem for zero-sum games, a lower bound on the complexity of a randomized algorithm may be obtained by fixing an input distribution and showing a lower bound on the expected complexity of any deterministic algorithm for that distribution. (The min-max theorem may be viewed as a special case of linear programming duality.)

Lovász [Lov89] used duality directly, but in a similar way, to give lower bounds on randomized communication complexity.

1.5 The K-Server Dual

In this section we introduce the concepts relevant to Chapter 4. We summarize the relevant k-server strategies, old results, and our new results.

The following k-server strategies are considered:

K-Phase — Roughly, K-Phase develops a nonoptimal solution to the dual of the k-server problem on-line and uses complementary slackness conditions to determine its choices.

Perm for k servers — Perm, an on-line assignment algorithm, interpreted as a k-server strategy via the reduction (see Section 4.1) of the k-server problem to assignment. Perm for k servers is the following: keep servers on k vertices that could currently be covered by Opt (with $h = k$) if the current request were the last.

Work$_\lambda$ — The "work" algorithm. In response to the current request r, let S denote the set of possible states of Opt's k servers after serving r, let C_s denote the cost incurred by Opt given that its servers are in state $s \in S$, let w denote the current state of

the on-line algorithm's servers, and let $d(w, s)$ denote the cost of moving servers from state w to s. WORK$_\lambda$ moves its servers into a state $s \in S$ minimizing $\lambda d(w, s) + C_s$.

We mention WORK because WORK$_0$, for the k-server problem, is PERM for k servers.

1.5.1 Previous Results

Chrobak *et al.* [CKPV90] formulated the problem of finding an optimal k-server schedule as an integral capacity min-cost max-flow problem, and conversely gave a linear-time reduction from the assignment (min-cost, bipartite, perfect matching) problem to the off-line k-server problem.

Kalyanasundaram and Pruhs [KP91] showed that a $c(k)$-competitive (when $h = k$) k-server algorithm implies a $(2c(k) - 1)$-competitive on-line assignment algorithm for assignments in $2k$-node bipartite graphs. Kalyanasundaram and Pruhs, and independently Khuller, Mitchell, and V. Vazirani [KMV90], discovered PERM, which is $(2n-1)$-competitive in $2n$-node bipartite graphs.

A number of researchers, including McGeoch and Sleator (stemming from their work on *residues*); Chrobak and Larmore; and Karloff, have conjectured that WORK$_1$ is k-competitive [McG91]. The generalization to WORK$_\lambda$ is due to Chrobak and Larmore [CL91b].

1.5.2 New Results

An optimal k-server schedule may be found by formulating the problem as an assignment (minimum-weight bipartite matching) problem. The intuition for this formulation is simple: each request represents a demand for a server from previous requests and a supply of a server to later requests; the cost of supplying the jth demand with a server from the ith request, where $i < j$, is the distance from the ith requested vertex to the jth requested vertex. The formulation, which is described in detail in Chapter 4, maintains the on-line interpretation of the problem.

K-PHASE is a new algorithm, invented by generalizing the implicit use of duality in LRU. For paging, LRU is a special case of K-PHASE. For weighted caching, BALANCE is obtained from K-PHASE by ignoring served requests, while GREEDYDUAL is essentially a

combination of K-PHASE and BALANCE. (GREEDYDUAL generalizes both.) For the general problem, K-PHASE is not competitive.

Lower bounds obtained from dual solutions are similar to amortized analyses with potential functions, in that each can be viewed as transforming the costs of the original problem and then applying simple, "local" lower bounds. BALANCE for weighted caching has a nice, intuitive analysis using duality which illustrates this similarity, and which leads directly to a potential function analysis.

K-PHASE may be thought of as an approximation to PERM for k servers, in that each may be viewed as developing a solution to the dual problem on-line, and using complementary slackness conditions to guide its choices. The difference is that K-PHASE develops a nonoptimal dual solution.

The formulation of the k-server problem as an assignment problem, together with the understanding of dual solutions implicit in the analysis of K-PHASE, provide some suggestive insight into the structure of optimal solutions to the general k-server dual.

1.6 Duality Analyses of Weighted Matching Strategies

In this section we introduce the weighted matching algorithms considered in Chapter 5, and summarize their analyses, reproduced in the chapter with the dual bounding technique.

- A *matching* is a subset of the edges of a graph such that each vertex in the graph adjoins at most one edge of the subset.
- A *perfect matching* is a matching such that each vertex adjoins exactly one edge.
- The *cost of a matching* in a weighted graph is the sum of the weights of the edges in the matching.
- An *assignment* is a perfect matching in a bipartite graph. Abusing terminology, if the bipartite graph $G = (U, W, E)$ with vertex set $U \cup W$ and edge set $E \subseteq U \times W$ has $|U| \neq |W|$, we also call a matching of size $\min\{|U|, |W|\}$ an assignment.
- A *metric graph* is a complete, or complete bipartite, graph with symmetric edge weights satisfying the triangle inequality: the weight of any edge is at most the weight

of any path connecting the endpoints.

Matchings and weighted matchings are well-studied.
For maximum-weight matching problems, we assume the edge weights are nonnegative.

We consider the following three off-line matching algorithms:

MIN — An algorithm that produces a min-cost perfect matching.

MAX — An algorithm that produces a max-weight matching.

GREEDYMAX2 — The greedy heuristic for max-weight matching [Avi83], which generates a matching by starting with the empty matching and repeatedly adding a maximum-weight edge not adjacent to any edge currently in the matching.

- The *on-line assignment problem* in a weighted bipartite graph $G = (U, W, U \times W)$ is as follows. The vertices of U are presented in some order. When a vertex is presented, costs of all adjoining edges are revealed, and some such edge must be added to the matching.

We consider the following deterministic, on-line assignment algorithms:

GREEDYMAX3 — The greedy algorithm for on-line, maximum-weight, bipartite matching [KVV90, KP91], which adds the max-weight edge that adjoins the presented vertex but is not adjacent to any edge already in the matching.

PERM — The permutation algorithm for on-line assignment [KP91, KMV90], which adds an edge adjoining the presented vertex to maintain the following invariant: the set of vertices in W adjoining some edge in the matching may be matched to the set P of presented vertices by a matching that is minimum-cost among maximum matchings in $G_P = (P, W, P \times W)$.

1.6.1 Reproduced Results

The following results are reproduced using the dual bounding technique:

- GreedyMax2 produces a matching within a factor of two of maximum in any graph. This is a result by Avis [Avi83]; a special case of the result is mentioned by Karp, Vazirani, and Vazirani [KVV90].
- GreedyMax3 produces an assignment within a factor of three of maximum in any complete, bipartite, metric graph. This is a result by Kalyanasundaram and Pruhs [KP91].
- Perm produces an assignment within a factor of $2n - 1$ of minimum in any complete, $2n$-node, bipartite, metric graph. This is a result by Kalyanasundaram and Pruhs [KP91] and Khuller, Mitchell, and V. Vazirani [KMV90].

Chapter 2

Paging

In this chapter, we consider three variants on the standard model for competitive analysis of paging strategies: allowing randomization, allowing resource-bounded lookahead, and loose competitiveness.

2.1 Partitioning a Sequence into K-Phases

We begin by explaining how to break a sequence of requests into k-phases. K-phases serve as an intermediate step in our competitive analyses for paging, providing lower bounds on optimal schedule costs and upper bounds on on-line schedule costs.

- The *k-phase partitioning* (a partitioning into k-phases, generally NOT a partitioning of size k) of a paging request sequence r is defined as follows. The first k-phase is the maximal prefix of r containing requests to at most k distinct items. In general, the ith k-phase is the maximal prefix of r_i containing requests to at most k distinct items, where r_i denotes r with the first $i-1$ k-phases removed.

  Thus, the ith k-phase starts exactly with the request that causes FWF with an (initially empty) fast memory of size k to flush its memory for the $i-$1st time.

  Generally, "a k-phase", or "a phase" will refer to one that is nonempty. Thus, each k-phase (except the last) of a sequence contains requests to k distinct items, and the last contains requests to at least 1 and at most k distinct items.

abacd·EaFeab·CDeb

Figure 1: A 4-phase partitioning, with new requests in capitals.

- The *new requests* within a k-phase (other than the first) are requests to items that were not requested yet in this phase or the previous. Note that at least the first request of every phase (other than the first) is new.
- $\mathcal{N}_r(k)$ denotes the average number of new requests per k-phase other than the first.
- $\mathcal{P}_r(k)$ denotes the number of (nonempty) k-phases other than the first.

The following lemma characterizes optimal schedule costs sufficiently for all of our competitive analyses for paging.

Lemma 2.1.1 ([You91, You])

$$\mathcal{C}_r(\textsc{Opt}, h)/\mathcal{P}_r(k) \geq k - h + 1 \quad (1)$$

$$\mathcal{C}_r(\textsc{Opt}, h)/\mathcal{P}_r(k) \geq (k - h + \mathcal{N}_r(k))/2 \quad (2)$$

$$\mathcal{C}_r(\textsc{Opt}, h)/\mathcal{P}_r(h) \leq \mathcal{N}_r(h) \quad (3)$$

Proof: (1) In the optimal schedule for r with a fast memory of size h, after the first request of the ith ($i > 1$) k-phase, at most $h - 1$ of the k distinct items requested in the previous phase remain in the fast memory. Thus, at least $k - h + 1$ evictions occur after the first request of the $i - 1$st phase and before the second request of the ith.[1]

(2) Let m_i ($i > 1$) denote the number of new requests in the ith k-phase. Then, during that k-phase and the previous, $k + m_i$ distinct items were requested. Consequently, any schedule for r with a fast memory of size h has at least $k - h + m_i$ evictions in these two

[1] This argument is essentially due to Sleator and Tarjan [ST85].

phases. Thus one can argue that the total number of evictions is at least

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

(3) Any (off-line) schedule that chooses to evict any item that will not be requested during the current phase incurs a cost of at most m_i in the ith phase. □

With this lemma, the problem of finding a competitive schedule for r is essentially reduced to the sub-problem of finding a competitive schedule for each k-phase of r. For example:

Lemma 2.1.2 ([You91, You]) *For any conservative paging strategy X, sequence r, and $k \geq h$,*

$$\mathcal{C}_r(X,k) \leq \frac{k}{k-h+1}\mathcal{C}_r(\textsc{Opt},h).$$

Proof: Any schedule produced by a conservative paging strategy has no evictions in the first phase and at most k evictions in each subsequent phase, while, by (1) of lemma 2.1.1, for each subsequent phase the optimal schedule incurs[2] a cost of at least $k-h+1$. □

Corollary 2.1.3 ([ST85]) Lru, Fifo, *and* Fwf *are $k/(k-h+1)$-competitive.*

Recall that Sleator and Tarjan [ST85] showed such strategies are optimally competitive:

Lemma 2.1.4 ([ST85]) *No deterministic paging strategy is better than $(k/(k-h+1))$-competitive.*

2.2 Randomized Paging

The proof of bound (3) essentially shows that the (off-line) strategy that evicts items that will not be requested during the current phase, thus incurring an eviction only in response to a new request, is 2-competitive.

[2]Note that the $k-h+1$ evictions might not be incurred *during* the phase. Let a_i denote the number of evictions incurred by the on-line strategy in the ith phase, and let $b_i = k-h+1$ for $1 < i \leq \mathcal{P}_r(k)$. Then the on-line cost is $\sum_i a_i$, the optimal cost is at least $\sum_i b_i$ (by lemma 2.1.1), and $a_i \leq b_i k/(k-h+1)$. Thus, the on-line cost is at most $k/(k-h+1)$ times the optimal cost.

This suggests that the goal of an on-line algorithm should be, to the extent possible, to evict the items that will not be requested during the phase. Deterministic algorithms such as LRU attempt this by distinguishing between items in the fast memory that have been requested this phase, and items that have not, and evicting the latter. Of course, items that have not yet been requested this phase may be requested later during the phase, so this strategy is imperfect. Nonetheless, with each eviction, any on-line algorithm following this strategy gets hold of one more of the items that will be requested in the phase, thus incurring at most k evictions before it has all of the items that will be requested during the phase in the fast memory.

Consider the two request sequences **ABCA** and **ABCB** (each with 2-phase partitioning of the form **AB**·**CX**). If an on-line strategy using a fast memory of size 2 is deterministic, it will incur 2 evictions on one of these sequences and 1 on the other. The optimal schedule will incur 1 eviction. Thus, no deterministic strategy can be better than 2-competitive on these two sequences. From the standpoint of the discussion in the previous paragraph, the deterministic algorithm pays 2 in the worst case because it must commit to a guess about which item will not be requested during the phase, and in the worst case it will be wrong.

From this perspective, a natural thing to try is not committing to either guess, instead choosing randomly. If, after receiving requests **AB**, the strategy evicts **A** or **B** with equal probability in response to request **C**, the strategy will incur an expected cost of 1.5 on either sequence. Thus a randomized strategy can be 1.5-competitive on the two sequences.

MARK, the randomized paging strategy, may be described as follows:

MARK

Maintain phases explicitly. When room for a new item is required, evict an item that has not yet been requested during the current phase uniformly at random.

Next we show the following:

Lemma 2.2.1 ([FKL+88]) MARK *is* $2H_k$*-competitive when* $h = k$.

Proof: We show that in a k-phase with m new requests, the expected number of evictions by MARK is bounded by $m(H_k - H_m + 1)$. Bound (2) of lemma 2.1.1, applied with $h = k$, implies that the optimal schedule incurs a cost of at least $m/2$ for the phase, so this shows

the result.[3]

An *old* request is a request to an item requested in the previous phase but not yet in this phase. A *repeat* request is a request to an item requested previously in this phase. There are $k - m$ old requests, and an arbitrary number of repeat requests.

Just before the ith old request, at most $m + i - 1$ distinct items have been requested this phase, so at least $k - m - i + 1$ of the $k - i + 1$ items requested in the previous phase but not yet in this phase remain in the fast memory. Each such item is in the memory with equal probability, so the probability that the ith old request will cause an eviction is at most $m/(k - i + 1)$.

Thus the expected number of evictions in response to old requests is at most

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

In addition, each of the m new requests causes an eviction, while none of the repeat requests causes an eviction. This gives the result. □

For $h < k$, one can show

Lemma 2.2.2 ([You91]) *When $h < k$,* MARK *is*

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

if $k/(k-h) > e$ and 2-competitive otherwise.

Proof: The previous proof showed that the expected number of evictions in a phase with m new requests is at most $m(H_k - H_m + 1)$.

The bound $H_k - H_m \le \ln\frac{k}{m}$ follows from an integration argument:

$$H_k - H_m = \sum_{i=m+1}^{k} \frac{1}{i} = \int_m^k \frac{dx}{\lceil x \rceil} \le \int_m^k \frac{dx}{x} = \ln\frac{k}{m}.$$

By (2) of lemma 2.1.1, the optimal schedule with a fast memory of size h incurs a cost of at least $(k - h + m)/2$ for the phase. Thus the competitiveness is bounded by $2\times$

$$\max_m f(m) = \frac{m(\ln k - \ln m + 1)}{k + m - h}.$$

[3] Again, note that the cost is not necessarily incurred during the phase. See the footnote in the proof of lemma 2.1.2.

Next we use elementary analytic techniques to bound $f(m)$.

First, $f'(m) = \frac{k-h}{(k+m-h)^2}\left(\ln\frac{k}{m} - \frac{m}{k-h}\right)$, so f has a single maximum at $m = m^*$ with $m^* = (k-h)\ln\frac{k}{m^*}$.

From $f(m) = \frac{m}{k+m-h}(\ln\frac{k}{m}+1)$ and $m^* = (k-h)\ln\frac{k}{m^*}$ we derive the equality:

$$\begin{aligned} f(m^*) &= \frac{(k-h)\ln\frac{k}{m^*}\left(\ln\frac{k}{m^*}+1\right)}{k-h+(k-h)\ln\frac{k}{m^*}} \\ &= \ln\frac{k}{m^*} \\ &= \ln\frac{k}{k-h} - \ln\ln\frac{k}{m^*} \\ &= \ln\frac{k}{k-h} - \ln f(m^*) \end{aligned} \tag{4}$$

Applying (4), the monotonicity of ln, and assuming WLOG that $f(m^*) \geq 1$, yields

$$\begin{aligned} f(m^*) &= \ln\frac{k}{k-h} - \ln\left(\ln\frac{k}{k-h} - \ln\left(\ln\frac{k}{k-h} - \ln f(m^*)\right)\right) \\ &\leq \ln\frac{k}{k-h} - \ln\left(\ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h}\right) \\ &\leq \ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h} + \frac{1}{2}. \end{aligned}$$

The last inequality follows from the general inequality (for $x \geq 1$) $\ln(x - \ln x) - \ln x = \ln(1 - \frac{\ln x}{x}) \geq \ln(1-\frac{1}{e}) \geq -\frac{1}{2}$. □

In fact, MARK is within approximately a factor of two of optimally competitive:

Lemma 2.2.3 ([You91]) *The competitive ratio of any randomized, on-line paging strategy is at least H_k when $h = k$,*[4] *and at least*

$$\ln\frac{k}{k-h} - \ln\ln\frac{k}{k-h} - \frac{2}{k-h}$$

when $h < k$ and $k/(k-h) \geq e$.

(Note that when $k/(k-h) \leq e$ the analysis of MARK shows that its competitive ratio is at most 2.)

[4] An alternate proof of the lower bound when $h = k$ is given by Fiat *et al.* [FKL+88]. The advantage of this proof is that it generalizes nicely to $h < k$.

Proof: We adapt Sleator and Tarjan's [ST85] lower bound on deterministic, on-line paging strategies.

Let X denote the on-line strategy, fix $\epsilon > 0$ arbitrarily small, and let m denote a positive integer to be determined later.

We generate a sequence of requests in segments. Each segment is generated as follows.

Generating an Adversarial Segment

1. Mark the h items currently in OPT's fast memory.
2. Request and mark $k - h + m$ items not previously requested.
3. Unmark the last such item requested.
4. For $i = 1, \ldots, h - 1$
   (a) While some unmarked[5] item has probability less than $1 - \epsilon$ of being in X's fast memory, request such an item.
   (b) Request and unmark a marked item least likely to be in X's fast memory.

If X is competitive, only a bounded number of requests can be generated in step (4a), otherwise X incurs an unbounded cost on a subsequence of requests to less than h items. Thus, if X is competitive, the above method in fact generates a finite length segment of requests.

OPT can service the segment, incurring at most $k - h + m$ evictions, by choosing to evict any item except the at most h items that will be unmarked by the end of the segment. By following such a strategy, after the $k - h + m$ requests in step (2), the fast memory will contain the h items that will be unmarked by the end of the segment; no others will be subsequently requested.

X, in addition to the cost of $k - h + m$ incurred in response to step (2), will incur an expense for each of the $h - 1$ requests to marked items. Before the request to a marked item in the ith iteration of the loop, the expected number of unmarked items in the X's memory is at least $i(1 - \epsilon)$, so the expected number of the marked items in the fast memory is at most $k - i(1 - \epsilon)$. Since there are $k + m - i$ of these, some marked item is not in the fast memory with probability at least $1 - (k - i(1 - \epsilon))/(k + m - i)$. Thus, the probability of an eviction in the ith iteration of the loop is at least $(m - i\epsilon)/(k + m - i)$.

[5] By "unmarked", we mean previously marked during the segment, but not currently marked.

It follows that the expected number of evictions can be made arbitrarily close to

$$\begin{aligned} & k-h+m+\frac{m}{k+m-1}+\frac{m}{k+m-2}+\cdots+\frac{m}{k+m-h+1} \\ = \;& k-h+m+m(H_{k+m-1}-H_{k+m-h}), \end{aligned}$$

and a lower bound on the competitiveness is given by

$$\max_{m=1,2,\ldots} 1+\frac{m}{k-h+m}(H_{k+m-1}-H_{k+m-h}). \tag{5}$$

When $h=k$, expression (5) is maximized at H_k when $m=1$. It remains only to prove that expression (5) is at least $\ln\frac{k}{k-h}-\ln\ln\frac{k}{k-h}-\frac{2}{k-h}$ when $h<k$ and $\frac{k}{k-h}\geq e$.

Let $x=\ln\frac{k}{k-h}$, so $x\geq 1$, and $m=\lfloor y=(k-h)(x-1)+1\rfloor$, so $m\geq 1$. Then

$$\begin{aligned} & 1+\frac{m}{k-h+m}(H_{k+m-1}-H_{k+m-h}) \\ \geq \;& 1+\left(1-\frac{k-h}{k-h+m}\right)\times\left(H_{k-1}-H_{k+m-h-2}-\frac{2}{k+m-h-1}\right) \\ \geq \;& 1+\left(1-\frac{k-h}{k-h+m}\right)\times\left(\ln\frac{k}{k+m-h-1}-\frac{2}{k-h}\right) \\ \geq \;& 1+\left(1-\frac{k-h}{k-h+y-1}\right)\times\left(\ln\frac{k}{k+y-h-1}-\frac{2}{k-h}\right) \\ = \;& 1+\left(1-\frac{1}{x}\right)\left(\ln\frac{k}{(k-h)x}-\frac{2}{k-h}\right) \\ = \;& 1+\left(1-\frac{1}{x}\right)\left(x-\ln x-\frac{2}{k-h}\right) \\ \geq \;& x-\ln x-\frac{2}{k-h}. \end{aligned}$$

The first inequality follows from the expansion of H_{k+m} and $H_{k+m-h+1}$. The second follows from $H_a-H_b\geq\ln((a+1)/(b+1))$ (which follows from an integration argument) and from $m\geq 1$. The third follows from $y-1\leq m\leq y$. The rest are relatively straightforward. □

2.3 Resource-Bounded Lookahead

It is easy to see that allowing the on-line algorithm to see the next l requests in deciding which item to evict does not help it in the worst case: for any on-line algorithm with lookahead l, construct an equally competitive on-line algorithm with lookahead 1 as follows. On request sequence $r_1r_2r_3\cdots$, simulate the algorithm with lookahead l on the request

sequence $r_1^l r_2^l \cdots$ (where x^l represents x repeated l times), and have the algorithm with lookahead 1 mimic the actions of the simulated algorithm on the first request of each r_i^l.

Instead, we consider *resource-bounded lookahead.*

In this model, the paging strategy is given a *lookahead queue*, the contents of which it knows. The strategy may *either* service the request at the head of the queue (provided there is one) *or* add an additional request (if there is one) to the end of the queue.

At a given moment, the contents of the queue form a subsequence of the entire request sequence. The strategy is *on-line with resource-bounded lookahead* l provided it never incurs more than l evictions on any such subsequence.[6]

A more intuitive description of resource-bounded lookahead is as follows: Imagine that the resource being measured by the cost to the on-line algorithm is time. Specifically, if the algorithm incurs cost c to handle a request given at time t, then the next request is given at time $t + c$. In this interpretation, the on-line algorithm is allowed to look ahead into the future as many requests as it wants; it is on-line with resource-bounded lookahead of l if it never looks more than l time units into the future.

MARK$^{[l]}$, *the marking algorithm with resource-bounded lookahead of* l, is an adaptation of MARK that uses resource-bounded lookahead. MARK$^{[l]}$ mimics MARK but, at the beginning of each phase, adds requests to the end of the queue until either k distinct items or l new requests are in the queue (or there are no more requests). Subsequently, instead of marking all items in the fast memory, it marks only those items not requested in the lookahead queue. Finally, when an item must be evicted during the phase, a marked item is evicted uniformly at random.

Lemma 2.3.1 *When* $h = k$, MARK$^{[l]}$ *is* $2(\ln(k/l) + 1)$*-competitive.*[7]

[6] This is not a very realistic notion of lookahead, but it is theoretically interesting — it leads to reduced competitive ratios. The challenge, of course, is to find a model which is both realistic and interesting in this sense; we present resource-bounded lookahead as a small step in that direction.

More realistic alternatives might be considering loose competitiveness of strategies with regular lookahead, assuming an *average* (rather than consistent) resource-bounded lookahead of l, or assuming that the sequence is fixed by an adversary but the lookahead is stochastic. None of these alternatives seems very promising on preliminary consideration.

[7] When $h < k$, essentially the proof of the lemma shows that the competitive ratio is bounded by

$$\max\{2, \max_{m=l,l+1,\ldots} m(H_k - H_m + 1)/(k - h + m)\}.$$

The ratio in the expression (see the proof of lemma 2.2.3) is unimodal, with a single maximum around $m^* \approx (k-h)\ln(k/(k-h))$. For l larger than m^*, the ratio is maximized when $m = l$. For l smaller, the

Proof: Consider the k-phase partitioning of an arbitrary request sequence. At the beginning of a given phase, if m, the number of new requests in the phase, is less than l, then all items requested in the phase will be in the lookahead queue. Thus, m items will be marked and subsequently evicted during the phase, while (from lemma 2.1.1) the optimal schedule has at least $m/2$ evictions for the phase.

Otherwise, an analysis essentially the same as for MARK shows that the expected number of faults in the phase is at most $m(H_k - H_m + 1)$. Again the optimal schedule has at least $m/2$ evictions for the phase. Since $m \geq l$, and $H_a - H_b \leq \ln(a/b)$, this gives the result. □

Let DMARK$^{[l]}$ denote any deterministic version of MARK$^{[l]}$: MARK$^{[l]}$ with the random choices replaced by arbitrary deterministic choices.

Lemma 2.3.2 DMARK$^{[l]}$ *is* $\max\{2k/(k-h+l), 2\}$*-competitive.*

Proof: As in the previous proof, either $m < l$, in which case at most m evictions are incurred, while the optimal schedule incurs a cost of $(k-h+m)/2$ for the phase, or $m \geq l$, in which case at most k evictions are incurred, while the optimal schedule has at least $(k-h+l)/2$ evictions for the phase. □

These upper bounds seem weak, in that the algorithms seem not to take full advantage of the lookahead. Yet the following lemma shows that they are roughly within a factor of 2 of optimally competitive:

Lemma 2.3.3 *A (randomized) on-line paging strategy with resource-bounded lookahead l using a fast memory of size k can be simulated by a (randomized) on-line paging strategy using a fast memory of size $k + l - 1$, so that on any sequence the cost of the standard strategy is no more than the cost of the simulated strategy.*

Proof: The simulated strategy cannot ask for an item to be added to the queue if there are currently l distinct items in the queue and not in the fast memory, as it risks having a request to an item not in the fast memory or in the queue added to the queue, which would cause $l + 1$ distinct items to be in the queue and not in the fast memory, which would in turn violate the assumption that it is resource-bounded lookahead l.

ratio is maximized independently of l at $m = m^*$. It seems likely that a better analysis, either of MARK$^{[l]}$ or a variant, could be given when $h < k$ and $l < m^*$.

Thus, at most $k + l - 1$ distinct items are either in the fast memory or in any but the final spot of the queue. Simulate the strategy by keeping these items in our fast memory: service each request at the instant it ceases to be the final request in the queue, evicting an item when the simulated strategy evicts it. □

Corollary 2.3.4 *No randomized, on-line strategy with resource-bounded lookahead of l is better than*

$$\left(\ln \frac{k+l-1}{l-1} - \ln\ln \frac{k+l-1}{l-1} - \frac{2}{l-1}\right) - \textit{competitive}$$

when $h = k$.

No deterministic, on-line strategy with resource-bounded lookahead of l is better than

$$\frac{k+l-1}{k+l-h} - \textit{competitive}.$$

Proof: Follows directly from lemmas 2.1.4, 2.2.3, and 2.3.3. □

2.4 Loose Competitiveness

Figure 2 shows graphs of $\mathcal{C}_r(X,k)/\mathcal{C}_r(\textsc{Opt},k)$ versus k for a number of paging strategies X on a typical sequence[8] r. For large k, the ratio is not near k or even H_k.

Such simulations led us to consider *loose competitiveness*. Recall that a paging strategy X is *loosely $c(k)$-competitive* if, for any d, as $n \to \infty$, for any request sequence r, the number of $k \in \{1, ..., n\}$ such that

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

is $o(n)$, where b depends only on k and the initial server locations in the schedules produced by Opt and X for k servers serving r.

In this section we show that conservative paging strategies (including Lru, Fifo, and Fwf) are loosely $c(k)$-competitive provided $c(k)/\ln k \to \infty$ and both $c(k)$ and $k/c(k)$ are

[8]The input sequence, traced by Dick Sites [SA88], 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.

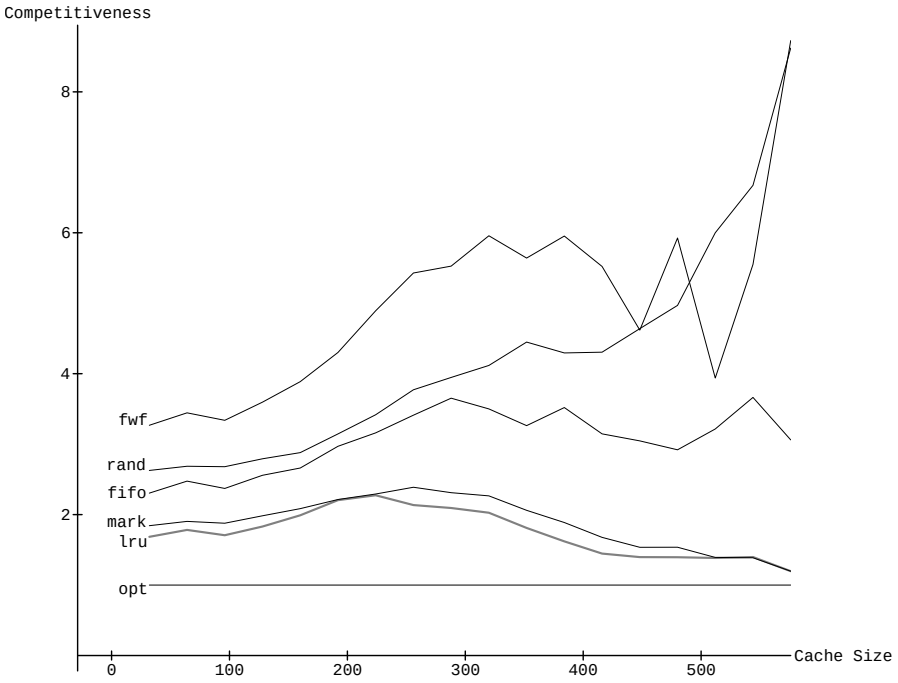

Figure 2: Typical competitiveness vs. fast memory size.

nondecreasing. We show that 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. We also give a tight lower bound for FWF: FWF is not loosely $c(k)$-competitive if $c(k)$ is $O(\ln k)$.

2.4.1 Upper Bounds for LRU, FIFO, FWF, and MARK

First, we summarize the bounds useful for the analysis:

Lemma 2.4.1 ([You91, You]) *Let X denote any conservative paging strategy. Then*

$$\mathcal{P}_r(k) \geq \mathcal{C}_r(X,k)/k \tag{6}$$

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

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

(Recall that $\mathcal{P}_r(k)$ and $\mathcal{N}_r(k)$ are, respectively, the number of k-phases of r (other than the first) and the average number of new requests per k-phase (other than the first) of r.)

Proof: (6) Follows directly from the definition of conservativeness.

(7) Follows from the definition of conservativeness and bound (2) of lemma 2.1.1.

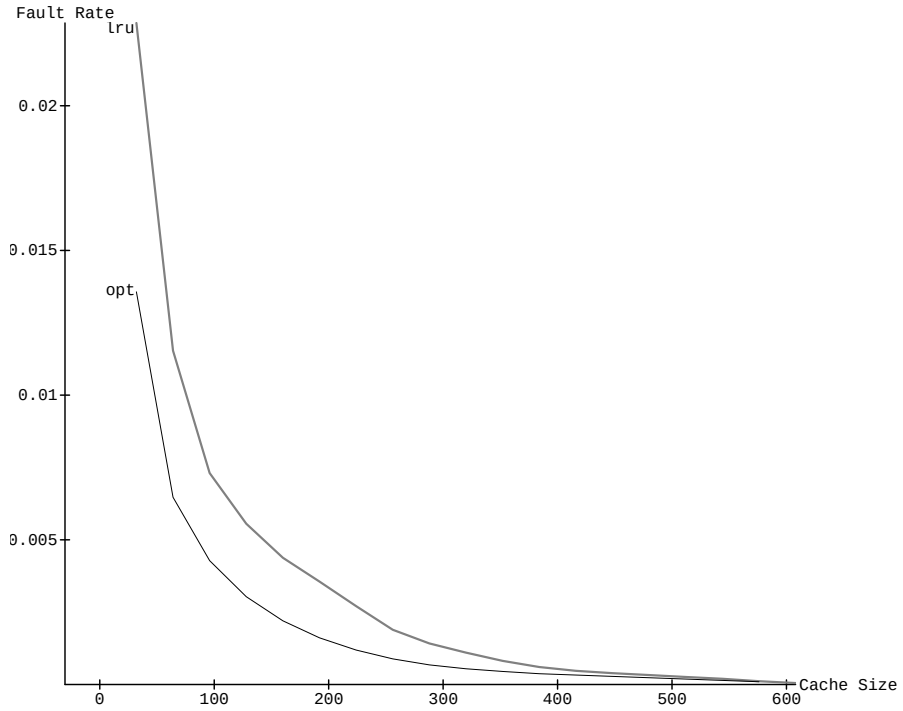

Figure 3: Typical fault rate vs. fast memory size.

(8) Follows from bound (2) and from the proof in lemma 2.2.1 that, in a k-phase with m new requests, MARK incurs a cost of at most $m(H_k - H_m + 1)$. □

Note that, for the on-line paging strategies we consider, high cost in an absolute sense implies high $\mathcal{P}_r(k)$, while high cost relative to the optimal schedule implies low $\mathcal{N}_r(k)$.

The essential observation for the analysis is that if $\mathcal{N}_r(k)$ is low, then a small increase in k will result in a large decrease in the number of phases:

Lemma 2.4.2 ([You91, You]) *Fix a sequence r. For any k, and any $k' \geq k + 2\mathcal{N}_r(k)$,*

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

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 $p_1, ..., p_{\mathcal{P}_r(k)}$ have a number of new nodes 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, pieces resulting from a join reference at most $k + 2\mathcal{N}_r(k) \leq k'$ distinct nodes, while the other pieces reference at most 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 $\mathcal{P}_r(k') \leq \mathcal{P}_r(k) - \lceil \mathcal{P}_r(k)/4 \rceil \leq \frac{3}{4}\mathcal{P}_r(k)$. □

Thus, there are few k with high $\mathcal{P}_r(k)$ and low $\mathcal{N}_r(k)$:

Theorem 2.4.3 *For any $\epsilon > 0, M > 0$, and any sequence r, the number of $k = 1, 2, \ldots$ such that*

$$\mathcal{N}_r(k) \leq M \text{ and } \mathcal{P}_r(k) \geq \epsilon \mathcal{P}_r(1) \tag{9}$$

is $O(M \ln \frac{1}{\epsilon})$.

Proof:

Let s be the number of k violating the condition.

We can choose $l = \lceil s/\lceil 2M \rceil \rceil$ such k so that each is at distance $2M$ from the preceding. Then we have $1 \leq k_1 \leq k_2 \leq \ldots \leq k_l$ such that for each i

$$\begin{aligned} \mathcal{N}_r(k_i) &\leq M, & (10)\\ k_{i+1} - k_i &\geq 2M, \text{ and} & (11)\\ \mathcal{P}_r(k_l) &\geq \epsilon \mathcal{P}_r(1). & (12) \end{aligned}$$

Then for any i, by (10) and (11), $k_{i+1} \geq k_i + 2\mathcal{N}_r(k_i)$, so, by lemma 2.4.2, $\mathcal{P}_r(k_{i+1}) \leq (3/4)\mathcal{P}_r(k_i)$. Inductively, $\mathcal{P}_r(k_l) \leq (3/4)^l \mathcal{P}_r(1)$.

This, and (12), imply $(3/4)^l \geq \epsilon$, so

$$\lceil s/\lceil 2M \rceil \rceil = l \leq \ln_{4/3} \frac{1}{\epsilon}.$$

This implies the bound on s. □

Recall that high absolute cost implies high $\mathcal{P}_r(k)$, while high relative cost implies low $\mathcal{N}_r(k)$. The preceding theorem thus implies that there are few k such that both absolute and relative costs are high, which shows loose competitiveness:

Corollary 2.4.4 *Let X denote any conservative paging strategy and $C : \mathcal{N}^+ \to \mathcal{R}^+$ a nondecreasing function.*

X is loosely $c(k)$-competitive provided that $k/c(k)$ is nondecreasing and

$$\frac{c(k)}{\ln k} \to \infty, \tag{13}$$

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

$$c(k) - 2\ln\ln k \to \infty. \tag{14}$$

Proof: Let X denote either any conservative paging strategy, in which case we assume condition (13) and that $k/c(k)$ is nondecreasing, or MARK, in which case we instead assume condition (14) 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 *violators* $k \in \{1, \ldots, n\}$ is $o(n)$, where a violator is a k such that

$$\mathcal{C}_r(X,k) \geq \max\{c(k)\mathcal{C}_r(\text{OPT},k), \mathcal{C}_r(\text{OPT},1)/n^d\}.$$

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

$$\mathcal{P}_r(k) \geq \frac{\mathcal{C}_r(X,k)}{k} \geq \frac{\mathcal{C}_r(\text{OPT},1)}{n^{d+1}} = \frac{1}{n^{d+1}}\mathcal{P}_r(1). \tag{15}$$

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

$$\mathcal{N}_r(k) \leq \frac{2k\mathcal{C}_r(\text{OPT},k)}{\mathcal{C}_r(X,k)} \leq \frac{2k}{c(k)} \leq \frac{2n}{c(n)}. \tag{16}$$

Since each violator k satisfies (15) and (16), by theorem 2.4.3, the number of violators is $O\left(\left(\ln n^{d+1}\right) n/c(n)\right)$. This is $o(n)$ by assumption (13).

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

$$\begin{aligned}
\mathcal{N}_r(k) &\leq k\exp\left(1 - \frac{\mathcal{C}_r(\text{MARK},k)}{2\mathcal{C}_r(\text{OPT},k)}\right) \\
&\leq k\exp(1 - c(k)/2) \\
&\leq n\exp(1 - c(n)/2),
\end{aligned}$$

so that by theorem 2.4.3 the number of violators is $O\left(\left(\ln n^{d+1}\right) n\exp(1 - c(n)/2)\right)$. This is $o(n)$ by assumption (14).

□

2.4.2 A Lower Bound for FWF

In the remainder of this section, we show that corollary 2.4.4 is tight for FWF.

First, we prove theorem 2.4.5, which shows that lemma 2.4.2 is qualitatively tight in a very strong sense: for any $n > 0$ and appropriately monotonic and smooth $\alpha : \mathcal{N} \to \mathcal{N}$, there exists a request sequence r such that, for *any* $k \leq n$, the average number of new requests per k-phase of r is at most $5\alpha(k)$, while the number of $(k+\alpha(k))$-phases is at least $1/8$ the number of k-phases.

A consequence of this theorem is that the bound provided by theorem 2.4.3 is tight in the worst case.

Since the costs of FWF and OPT for a given k on a given sequence are essentially captured by the average number of new requests per phase and the number of phases, as a corollary we show the lower bound for FWF.

Theorem 2.4.5 *Fix any $n > 0$, and any function $\alpha : \mathcal{N} \to \mathcal{N}$ such that, for all $k \in \mathcal{N}$, $\alpha(k) \leq \alpha(k + \alpha(k)) \leq 2\alpha(k)$ and $1 \leq \alpha(k) \leq k$.*

Then there exists a request sequence r such that for $k = 1, \ldots, n$

$$\mathcal{N}_r(k) \leq 5\alpha(k) \quad \textit{and} \quad \frac{\mathcal{P}_r(k + \alpha(k))}{\mathcal{P}_r(k)} \geq \frac{1}{8}.$$

Proof: Fix n and α as described above.

Let $k_0 = 1$, and $k_{i+1} = k_i + \alpha(k_i)$ for $i = 0, 1, \ldots$.

The proof is in two parts:

I. We inductively construct sequences $s_0, s_1, \ldots$ such that, for $j \leq i$,

$$\begin{aligned} \mathcal{P}_{s_i}(k_j) &= 2^{i-j} - 1, \text{ and} \\ \mathcal{N}_{s_i}(k_j) &= \alpha(k_j). \end{aligned}$$

II. We show that for any k, i, j *s.t.* $k < k_j < k_i$

$$\begin{aligned} \mathcal{P}_{s_i}(k + \alpha(k))/\mathcal{P}_{s_i}(k) &\geq 1/4 - 2^{j-i-1}, \text{ and} \\ \mathcal{N}_{s_i}(k) &\leq 5\alpha(k). \end{aligned}$$

Any s_i such that $k_{i-2} > n$ then proves the theorem, since $k \leq n$ implies $k < k_{i-2} < k_i$, which in turn implies $\mathcal{P}_{s_i}(k + \alpha(k))/\mathcal{P}_{s_i}(k) \geq 1/4 - 1/8 = 1/8$.

(I.) Each request in each s_i is to a member of $\mathcal{N}$, but may be represented either by the member itself or by the special symbol '$\mathbf{x}$', which represents a unique request to a new item. To interpret such a sequence as a request sequence per. se., for instance in order to perform a k-phase partitioning, replace each special symbol with a request to a unique item: a member of $\mathcal{N}$ not requested anywhere else in the sequence.

Inductively Constructing s_i

1. Let $s_0 = \mathbf{x}$.
2. For $i = 0, 1, 2, \ldots$
   (a) Let $s_i' \leftarrow s_i$.
   (b) Scan s_i' from left to right. After $\alpha(k_{i+1}) - \alpha(k_i)$ special symbols have been scanned, continue scanning, replacing each occurrence of the special symbol with a request to the smallest element of $\mathcal{N}$ not requested anywhere else in the sequence.
   (c) Let $s_{i+1} = s_i, s_i'$.

Inductively, one can verify the following for $j < i$:

- Each s_i consists of 2^i requests to k_i distinct items and has $\alpha(k_i)$ special symbols.
- The first subsequence of 2^j requests of s_i is a copy of s_j.
- Each subsequent (nonoverlapping) subsequence of length 2^j is a copy of s_j, except that each special symbol might be replaced by a request to an item that is not requested in the preceding subsequence of length 2^j.

Now, considering the successive nonoverlapping subsequences of length 2^j in s_i, each requests k_j distinct items, and each but the first begins with a request to an item not in the preceding subsequence. Thus, the successive subsequences exactly form the k_j-phase partitioning of s_i. Thus, $1 + \mathcal{P}_{s_i}(k_j) = 2^{i-j}$. (Recall that $\mathcal{P}_r(k)$ is the number of nonempty k-phases in r other than the first.)

Furthermore, the new requests in each k_j-phase consist exactly of the $\alpha(k_j)$ requests which correspond to special symbols in s_j. Thus, $\mathcal{N}_{s_i}(k_j) = \alpha(k_j)$.

(II.) Fix k and i *s.t.* $k < k_{i-1} < k_i$.

Find j *s.t.* $k_{j-1} \leq k < k_j = k_{j-1} + \alpha(k_{j-1})$.

First we bound $\mathcal{P}_{r_i}(k+\alpha(k))/\mathcal{P}_{r_i}(k)$: Note that $k + \alpha(k) \leq k_j + \alpha(k_j) = k_{j+1}$, so

$$\frac{\mathcal{P}_{s_i}(k+\alpha(k))}{\mathcal{P}_{s_i}(k)} \geq \frac{\mathcal{P}_{r_i}(k_{j+1})}{\mathcal{P}_{r_i}(k_{j-1})} = \frac{2^{i-j-1}-1}{2^{i-j+1}-1} \geq \frac{1}{4} - 2^{j-i-1}.$$

Next we bound $\mathcal{N}_{r_i}(k)$: Consider any k-phase (other than the first) with m new requests. The number of distinct items requested in the k-phase and the previous is $k+m$. Since the 2 k-phases are contained in a subsequence formed by at most 3 consecutive k_j-phases, and there are at most $k_j + 2\alpha(k_j)$ distinct items requested in any 3 consecutive k_j-phases, it follows that $m \leq k_j - k + 2\alpha(k_j)$. Since $k_j - k \leq k_j - k_{j-1} = \alpha(k_{j-1})$ and $\alpha(k_j) = \alpha(k_{j-1} + \alpha(k_{j-1})) \leq 2\alpha(k_{j-1})$, it follows that $m \leq 5\alpha(k_{j-1}) \leq 5\alpha(k)$.

□

Corollary 2.4.6 FWF *is not loosely $c(k)$-competitive for any $c(k)$ which is $O(\ln k)$.*

Proof: We show that for any fixed $b > 0$ and even n, there exists a sequence r with $\mathcal{C}_r(\text{OPT}, n)$ arbitrarily large such that for k between $n/2$ and n,

$$\mathcal{C}_r(\text{FWF}, k) \geq (1/n)^{3b+3} \cdot \mathcal{C}_r(\text{OPT}, 1), \text{ and} \tag{17}$$

$$\mathcal{C}_r(\text{FWF}, k) \geq (b \ln k/10) \cdot \mathcal{C}_r(\text{OPT}, k). \tag{18}$$

Thus, for any $c(k)$ which is $O(\ln k)$, and any n, there is a sequence r and a d such that, for all k between $n/2$ and n,

$$\mathcal{C}_r(\text{FWF}, k) - \max\{c(k)\mathcal{C}_r(\text{OPT}, k), \mathcal{C}_r(\text{OPT}, 1)/n^d\}$$

is arbitrarily large. (Take b *s.t.* $2c(k) \leq b \ln k$ and take $d \geq 3b + 4$.)

Next we exhibit the sequence satisfying (17) and (18) for k between $n/2$ and n. Note that $\mathcal{C}_r(k, \text{FWF}) = k\mathcal{P}_r(k)$ and $\mathcal{C}_r(k, \text{OPT}) \leq \mathcal{N}_r(k)\mathcal{P}_r(k)$, so to show (17) and (18) it suffices to show

$$k\mathcal{P}_r(k) \geq (1/n)^{3c+3} \cdot \mathcal{P}_r(1), \text{ and} \tag{19}$$

$$k \geq (c \ln k/10) \cdot \mathcal{N}_r(k). \tag{20}$$

Applying theorem 2.4.5 with $\alpha(k) = \min\{k, a\}$, where $a = n/(c \ln n)$, we obtain a sequence r such that for $k < n$, $\mathcal{N}_r(k) \leq 5 \min\{k, a\}$ and $\mathcal{P}_r(k + \min\{k, a\}) \geq \mathcal{P}_r(k)/8$. The former implies (20) for k between $n/2$ and n.

It remains only to show that the latter implies (19) for k between $n/2$ and n. We omit the details, noting that first one shows $\mathcal{P}_r(a) \geq (1/8)^{\log_2 a} \mathcal{P}_r(1)$, and then $\mathcal{P}_r(k) \geq (1/8)^{k/a} \mathcal{P}_r(a)$ for $k \geq a$, and the result follows. □

Chapter 3

Deterministic Weighted Caching

Section 3.1 of this chapter introduces *timings*, which are useful for obtaining lower bounds on k-server schedule costs, much as k-phases were useful for paging.

Section 3.2 introduces GreedyDual, a weighted caching algorithm, and presents a competitive analysis of GreedyDual using timings.

3.1 The K-Server Dual: Timings

- A *timing* for a k-server request sequence $r = r_0 r_1 \cdots r_N$ is a real value B_∞, a real value A_0, and a pair of real values $\{A_i, B_i\}$ for $i = 1, \ldots, N$ such that

$$A_i - B_j \leq \begin{cases} d(r_i, r_j) & 0 \leq i < j \leq N \\ 0 & 0 \leq i \leq N, j = \infty \end{cases}$$

- The *cost of the timing for k servers* is

$$k(A_0 - B_\infty) + \sum_{i=1}^{N} A_i - B_i.$$

Timings seem a bit mysterious as introduced in this chapter; Chapter 4 discusses how timings represent the linear programming dual of a natural formulation of the k-server problem as an assignment problem, and presents a more intuitive interpretation of timings.

For the purposes of this chapter, think of a timing as a transformation of the original k-server problem specified by r. Any schedule for r is still a schedule for the transformed

problem, but the costs associated with the various server movements are transformed, so that the cost of the schedule is also changed. Specifically, when a server leaves a node r_i, or remains on r_i after the final request, the schedule is credited (has its cost reduced) by A_i, when a server enters a node r_i, the schedule is charged an additional B_i, and when a server remains on any node after the final request, the schedule is charged an additional B_∞.

In this manner, the cost of any schedule for k servers will be decreased by exactly the cost of the timing for k servers. For instance, when a server enters and leaves a node r_i, the transformation results in an additional charge of $-A_i+B_i$ to the schedule, which contributes one term to the cost of the timing.

On the other hand, it is not hard to see that the transformation leaves the cost of the schedule nonnegative. This is because the individual charges to the schedule in the transformed cost (either $d(r_i, r_j) - A_i + B_j$ to move a server from r_i to r_j or $-A_i + B_\infty$ to leave a server on r_i after the final request) remain nonnegative due to the constraints on the values of the timing.

Thus, the cost of the schedule is at least the cost of the timing. This shows the following lemma.

Lemma 3.1.1 ([You91, You]) *The cost of the optimal schedule for k servers satisfying a request sequence r is at least the cost for k servers of any timing for r.*

It will follow as a consequence of results in Section 4.1 that the lower bound provided by the lemma can, with the right timing, always be made tight.

3.2 The Greedy Dual Strategy

GreedyDual, the greedy dual strategy for weighted caching described in the introductory chapter, may be described more carefully as follows.

GREEDYDUAL

Each server has a varying amount of *credit*. In response to request 0, all servers are placed on r_0 with no credit. In response to each subsequent request j to node r_j,

1. If node r_j has no server:
   (a) Each server's credit is increased equally until some server has enough credit to move to r_j. (If a server is currently on r_i, it must have $d(r_i, r_j) = w(r_i)$ credit to move to r_j.)
   (b) One such server serves request j, giving up all its credit.
2. If node r_j has at least one server:
   (a) One such server serves request j.
   (b) Unless the server has not yet moved, it gives up an arbitrary amount (possibly none) of its credit.

We have the following lemma

Lemma 3.2.1 ([You91, You]) GREEDYDUAL *is* $\frac{k}{k-h+1}$*-competitive:*

$$\mathcal{C}_r(\text{OPT}, h) \geq \frac{k-h+1}{k}\mathcal{C}_r(\text{GREEDYDUAL}, k) - (k-h)w(r_0).$$

Proof: Let P_i, for $i = 0, 1, \ldots, N$, denote the amount credited to each server in response to request i. Let $P_{N+1} = 0$.

For $i = 0, \ldots, N$, let $i' = \min\{j > i : r_j \text{ served by server of } r_i\}$ if the minimum is well-defined, otherwise let $i' = N + 1$.

For $i = 0, \ldots, N$ and $j = 1, \ldots, N+1$, let

$$\begin{aligned} B_j &= P_0 + \cdots + P_{j-1}, \\ A_i &= P_0 + \cdots + P_{i'}. \end{aligned}$$

Let $B_\infty = B_{N+1}$.

We prove the lemma in three parts:

I. The on-line cost is at most kB_∞.

II. $\{A_0, B_\infty\} \cup \{A_i, B_i : i = 1, \ldots, N\}$ is a timing.

III. The cost of the timing simplifies to $(k+1-h)B_\infty - (k-h)w(r_0)$.

From these and lemma 3.1.1, the on-line algorithm is $\frac{k}{k-h+1}$-competitive.

(I.) The distance moved by servers is bounded by the total amount credited to servers, which is kB_∞.

(II.) The quantity $A_i - B_j$ is nonpositive if $j > i'$, and otherwise may be written $P_j + \cdots + P_{i'}$. Thus it suffices to verify

$$P_j + \cdots + P_{i'} \leq \begin{cases} w(r_i) & \text{for } 0 \leq i < j \leq N,\ r_i \neq r_j, \\ 0 & \text{for } 0 \leq i < j \leq N,\ r_i = r_j,\ i' = j, \\ 0 & \text{for } 0 \leq i < j \leq N,\ r_i = r_j,\ i' \neq j, \\ 0 & \text{for } 0 \leq i \leq N,\ j = N+1. \end{cases}$$

The first case holds because the amount credited to the server of request i in response to requests $i+1$ through i', inclusive, is at most $w(r_i)$.

The second case holds because, since $r_i = r_j$ and $i' = j$, nothing was credited to servers in response to request j, so $P_j = 0$.

In the third case, the server of request i was on node $r_j = r_i$ at request j but some other server served it. Since no server is ever moved to a node that is served, this happens only if the two requests are to node r_0 and the two servers have not yet moved. Since the server of r_i is never credited after the ith request until it moves (which can't happen before request i') we have $P_j + \cdots + P_{i'} \leq P_{i+1} + \cdots + P_{i'} = 0$.

In the final case, since $i' \geq j$, $P_j + \cdots + P_{i'} = P_{N+1} = 0$.

(III.) Note that $A_i - B_i - P_i = P_{i+1} + \cdots + P_{i'}$, which is the amount credited to the server of request i in response to requests $i+1$ through i', inclusive. Let R_s denote the set of requests (other than request 0) served by the server s, and let $i_s = \min R_s$. The total credited to a server s is

$$\begin{aligned} & P_0 + \cdots + P_{N+1} \\ &= P_0 + \cdots + P_{i_s} + \sum_{i \in R_s} P_{i+1} + \cdots + P_{i'} \\ &= w(r_0) + \sum_{i \in R_s} A_i - B_i - P_i, \end{aligned}$$

so the total, kB_∞, credited to all servers may also be written

$$kw(r_0) + \sum_{i \in \cup_s R_s} A_i - B_i - P_i = kw(r_0) + \left(\sum_{i=1}^{N} A_i - B_i \right) - B_\infty.$$

(Note that $P_0 = P_{N+1} = 0$.) Thus, $\sum_{i=1}^{N} A_i - B_i = (k+1)B_\infty - kw(r_0)$, which gives the desired simplification.

□

Note that if we wish to allow the on-line strategy to start with an empty fast memory, so that r_0 is an artificial vertex of weight 0, the algorithm and analysis simplify slightly.

The reader may verify that FIFO and LRU, and BALANCE as it applies to weighted caching, are special cases of GREEDYDUAL. Also, MARK is a special case of GREEDYDUAL in the sense that it is obtained by breaking ties in step (1b) uniformly at random and having the server give up all its credit in step (2b).

Chapter 4

The K-Server Dual

Preceding chapters have concentrated on presenting results and their proofs. This chapter is a more intuitive exploration of the relationships between the results, via timings. The goal of the chapter is to gain insight into the general k-server problem, and to record some suggestive partial or preliminary results.

In section 4.1 of this chapter, we show how the k-server problem is a special case of the assignment problem, so that the problem of finding a maximum-cost timing is a special case of the dual of assignment, and we discuss how nonoptimal timings may be improved.

In Section 4.2 we define k-phase timings, which generalize k-phases, and work an example.

In Section 4.3, we introduce K-Phase, a k-server strategy which uses k-phase timings explicitly in the way that Lru uses k-phases implicitly. K-Phase is central to this thesis, in that it relates Lru, Balance, and Perm (the on-line assignment algorithm), and is a stepping stone to GreedyDual.

In Section 4.4, we present an intuitive interpretation of timings, an intuitive analysis of Balance for weighted caching obtained therewith, and a potential function analysis obtained from the intuitive analysis.

Finally, in Section 4.5, we discuss how Perm may be interpreted as a k-server strategy, and how Perm relates to K-Phase. We also discuss the structure of the optimal timings implicit in Perm.

4.1 The K-Server Problem as Assignment

The k-server problem is a special case of the assignment problem.

Intuitively, each k-server request represents a demand for one server from previous requests and a supply of up to one server to later requests, with the exception that the first request represents no demand and a supply of up to k servers for later requests. If the server of request i next serves request j, this corresponds to the ith supply being assigned to the jth demand, at a cost of $d(r_i, r_j)$. To balance the supply and demand, an artificial demand, ∞, is added, which costs nothing to supply. Specifically,

- $G^N = (U^N, W^N, U^N \times W^N)$, given a k-server request sequence $r = r_0 r_1 \cdots$ and $N \geq 0$, is the *supply-demand graph for* r: a complete, bipartite graph with edge lengths d, where

$$\begin{aligned} U^N &= \{0, \ldots, N\} \text{ are the } \textit{supply} \text{ vertices,} \\ W^N &= \{1, \ldots, N, \infty\} \text{ are the } \textit{demand} \text{ vertices, and} \\ d(i,j) &= \begin{cases} \infty & i \geq j \\ d(r_i, r_j) & 0 \leq i < j \leq N \\ 0 & 0 \leq i \leq N, j = \infty. \end{cases} \end{aligned}$$

- F_k is the function

$$F_k(i) = \begin{cases} 1 & i \notin \{0, \infty\} \\ k & i \in \{0, \infty\}. \end{cases}$$

- An *F_k-factor* of G^N is any subset of the edges of G^N such that nodes 0 and ∞ have degree at most k in the subset, and every remaining node has degree 1 in the subset.

- The *cost* of an F_k-factor is the sum of the lengths of its edges.

We leave it to the reader to verify that the schedules of k servers satisfying the first N requests of r correspond one-to-one with the finite-cost F_k-factors in G^N, and that the correspondence preserves cost. In this chapter we identify the problems of optimally scheduling the k servers and finding the minimum-cost F_k-factor.

4.1.1 The Assignment Dual

The problem of finding a minimum-cost F_k-factor of G^N is an instance of the Hitchcock problem [PS82, p.143].[1] The standard form of the dual problem is to find a maximum-cost *potential* (assignment of weights to the vertices) of G^N such that

$$\alpha_u + \beta_w \leq d(u,w) \quad \text{for } (u,w) \in U \times W,$$

where α_u is the weight of $u \in U^N$ and β_w is the weight of $w \in W^N$, and the cost of the potential is

$$k(\alpha_0 + \beta_\infty) + \sum_{i=1}^{N} \alpha_i + \beta_i.$$

The essential properties of the dual are that the cost of any potential is a lower bound on the cost of any F_k-factor, and equality is achieved if both are optimal. Potentials for G^N are trivially isomorphic to timings for r: from a potential we can obtain an equal cost timing, or vice versa, by negating the weights of the supply vertices. Thus, in the remainder of this chapter we consider only timings, and we refer to "timings for G^N".

4.1.2 Stretching a Timing

Given a timing, when and how can it be improved? Here we summarize the necessary technique, obtained by considering the equivalent Hitchcock problem as described in Papadimitriou and Steiglitz [PS82]. Given a timing,

- The *slack of an edge* (u,w) is $d(u,w) - A_u + B_w$.
- The *admissible edges* are the edges with slack 0.
- The *admissible neighbors of a vertex* are the neighbors of the vertex along admissible edges.
- The *weight,* $|v|$*, of a vertex* v in G^N is 1, unless the vertex is 0 or ∞, in which case the weight is k.
- The *weight,* $|S|$*, of a subset* S of the vertices is the sum of the weights of the vertices in the subset.

[1] It is straightforward to transform a minimum-cost F_k-factor problem into an assignment problem by splitting the nodes 0 and ∞ each into k identical nodes. We use the "nonsplit" form for syntactic simplicity.

- To *stretch a subset U^* of the supply vertices by $\epsilon > 0$*, given a timing, is to increase the weight of each vertex that is in U^*, or is an admissible neighbor of U^*, by ϵ.
- To *stretch a subset U^* of the supply vertices* is to stretch U^* by the largest $\epsilon > 0$ possible without violating any constraints.

Stretching a timing corresponds to a step of the primal-dual algorithm (see the alpha-beta procedure in Papadimitriou and Steiglitz [PS82]) for finding a maximum-cost potential.

If an F_k-factor exists among the admissible edges, the cost of the F_k-factor equals the cost of the timing, so both are optimal. Otherwise, some subset of U^N exists such that the weight of the subset is greater than the weight of the admissible neighbors of the subset.[2] That is, if U^* is the subset of U^N, and W^* is the set of admissible neighbors of U^*, then $|U^*| > |W^*|$. In that case, stretching U^* by ϵ increases the cost of the timing, $k(A_0 - B_\infty) + \sum_{i=1}^{N} A_i - B_i$, by $\epsilon(|U^*| - |W^*|)$.

How does stretching change the set of admissible edges? Stretching U^* as much as possible adds at least one admissible edge from U^* to a vertex not in W^*. Any admissible edges from vertices not in W^* to U^* become inadmissible. The admissibility of any other edge is unchanged.

4.2 K-Phase Timings

To reproduce and generalize k-phases with timings, we introduce some new terminology.

- *The zero timing* for any G^N is the timing for G^N with all values 0.
- *The pending nodes* of a timing are the supply nodes whose only admissible neighbor is the demand node ∞. Intuitively, these are the nodes that complementary slackness conditions suggest are currently covered (that is, the supply corresponding to the request is not used by a demand other than ∞) in the optimal schedule.
- *Extending a timing* for G^N yields a timing for G^{N+1}. Extending $\{A_0, B_1, \ldots, A_N, B_\infty\}$ yields $\{A_0, B_1, \ldots, A_N, B_{N+1}, A_{N+1}, B_\infty\}$, where B_{N+1} and A_{N+1} are taken to be B_∞. Extending violates no constraints and leaves the cost unchanged. Newly admissible

[2] This is a consequence of the alpha-beta procedure of Papadimitriou and Steiglitz [PS82]. We review in Section 4.5 how to find such a U^*.

r:	**o**	A	B	A	C	D	·	**E**	A	**F**	E	A	B	·	**C**	**D**	E	B	∞
A:	0	0	1	1	1	1		1	1	2	2	2	2		2	2	2	2	
B:		0	0	0	0	0		0	1	1	1	1	1		1	2	2	2	2

Figure 4: The 4-phase timing for **oABACDEAFEABCDEB**

edges in G^{N+1} are $(N+1, \infty)$ and edges of the form $(u, N+1)$ such that $d(r_u, r_{N+1}) = 0$ and (u, ∞) is admissible. Admissibility of other edges is unchanged. The new supply node $N+1$ is pending, and all other pending nodes continue to pend, except pending nodes i *s.t.* $d(r_i, r_{N+1}) = 0$.

- *The k-phase timing* for a request sequence $r_0 r_1 \cdots r_N$ is defined inductively (and on-line) as follows. The k-phase timing for r_0 is the zero timing. Given a k-phase timing for $r_0 \cdots r_{N-1}$, in response to request r_N, the timing is extended, and if the weight of the pending nodes exceeds k (the weight of demand node ∞, the only admissible neighbor of the pending nodes), the pending nodes are stretched. This yields the k-phase timing for $r_0 \cdots r_N$.

An important invariant satisfied by the k-phase timing for any request sequence $r_0 \cdots r_N$ is that the cost for k servers simplifies:

$$k(A_0 - B_\infty) + \sum_{i=1}^{N} A_i - B_i = B_\infty. \tag{21}$$

This invariant is easily proved by induction; the key observation is that each stretch operation raises equally B_∞ and the cost for k servers. From (21), the cost for h servers of a k-phase timing simplifies to

$$(k - h + 1)B_\infty - (k - h)A_0. \tag{22}$$

Next we try an example: we develop the 4-phase timing for paging request sequence **ABACDEAFEABCDEB**. Prepend to the sequence the artificial request **o** with $d(\mathbf{o}, \cdot) \equiv 0$, and assume that all servers start on **o**.

Begin with the timing $\{A_0 = 0, B_\infty = 0\}$ for G^0. The weight of supply node 0 (corresponding to the artificial request **o**) is k.

In response to the first request **A**, extend the timing. The new (zero-cost) edge $(0, 1)$ is admissible, so supply node 0, of weight k, ceases to pend, but supply node 1, of weight 1, is now pending.

In response to the second request **B**, extend the timing again. The timing is still a zero timing. Supply nodes 1 and 2 are now pending, with combined weight 2.

In response to the third request **A**, extend the timing a third time. Since the length of edge $(1,3)$ is 0, supply node 1 ceases to pend. Supply nodes 2 and 3 are now pending.

In response to each of the fourth and fifth requests **CD**, extend the timing again without stretching. This yields the zero timing for G^5. Supply nodes 2, 3, 4, and 5 are now pending, with net weight 4.

In response to the sixth request **E**, again extend the timing. The new supply node 6 is now pending, so the combined weight of the pending nodes is 5, which exceeds k by 1. Stretch the timing, increasing A_2 through A_6, B_∞, and the cost of the timing by 1. This yields the timing $\{A_0 = 0, A_1 = 0, B_1 = 0, A_2 = 1, B_2 = 0, A_3 = 1, B_3 = 0, A_4 = 1, B_4 = 0, A_5 = 1, B_5 = 0, A_6 = 1, B_6 = 0, B_\infty = 1\}$. Supply nodes 2 through 5 cease to pend, as their respective edges to demand node 6 have their slack reduced to 0. This timing is of cost 1 for 4 servers.

Continuing this example reveals the following general pattern. After request 1, or after any stretch, one weight-1 supply node is pending. Subsequently, with each request, the corresponding supply node is added to the set of pending nodes. If a supply node corresponding to a request to the same node was pending, that node ceases to pend, otherwise the number (and weight) of the pending nodes increases by 1. If the number is $k + 1$, a stretch occurs, and all nodes except the node corresponding to the last request cease to pend.

Each stretch raises the cost of the timing for k servers by 1. More generally, each stretch raises the cost of the timing for h servers by $k - h + 1$.

Thus, the k-phase timing, for a paging request sequence, corresponds to the k-phase partitioning of the sequence, and gives a proof that the optimal schedule for h servers costs at least $k - h + 1$ times the number of k-phases. (See lemma 2.1.1.)

The complete 4-phase timing for the example is shown in figure 4.

4.3 LRU, K-Phase Timings, and BALANCE

The essential property of LRU, from the point of view of the proof of lemma 2.1.2, is the following: once a node is requested within a phase, it remains for the duration of the phase.

To restate this property in terms of timings, we need to clarify our terminology a bit. Specifically, for the following discussion we imagine that, as requests come in, we maintain the k-phase timing on-line as discussed above: extending and possibly stretching the timing in response to each request. Then a node (in the original graph[3]) r_i is *pending* provided the supply node i is pending in the current timing.

From the point of view of the k-phase timing, this property may now be restated simply as follows: keep the pending nodes served.

For definiteness, we describe K-PHASE, an on-line algorithm that keeps the pending nodes served:

K-PHASE

Maintain a timing (in fact the k-phase timing) for the current request sequence on-line. In response to each request r_j,

1. Extend the timing. (The pending nodes r_i *s.t.* $d(r_i, r_j) = 0$ cease to pend.)
2. If the request was to a nonpending node and the weight of the pending supply nodes is now $k+1$, stretch the pending supply nodes. (This causes at least one pending node r_i to cease pending, so that $A_i - B_{j'} = d(r_i, r_{j'})$, for some j' *s.t.* $i < j' \leq j$ with $d(r_i, r_{j'}) \neq 0$.)
3. If the request was to a nonserved node, move a server from a nonpending node.

For paging, K-PHASE simplifies as follows:

[3]The terminology is awkward, since there are two types of nodes in context: nodes in the original graph in which requests occur, and nodes in the supply-demand graph. In general, we refer to the latter type as supply or demand nodes to disambiguate.

K-PHASE *for Paging*

Maintain marks on the nodes. (Pending nodes are those that are not marked.) Initially, the first k distinct nodes requested are served and not marked.[4] Other nodes are marked. In response to each subsequent request,

1. If the request was to a marked node, unmark the requested node.
2. If $k+1$ nodes are unmarked, mark all the nodes except the requested node.
3. If the request was to a nonserved node, move a server from a marked node.[5]

Note that in order to implement K-PHASE for paging, it suffices to maintain marks on the servers of the marked nodes.

For weighted caching, we also obtain some simplification. Since the B_j are nondecreasing with j in a k-phase timing, if a node r_i ceases to pend as the result of a stretch, for weighted caching, it must be that the edge $(i, i+1)$ became admissible in the stretched timing. Thus, in order to keep track of the pending nodes, it suffices to maintain, for each pending node r_i, the slack in the current timing of the edge $(i, i+1)$ (if it exists) in the supply-demand graph. If and only if this slack is reduced to 0 or node r_i is again requested does supply node i cease to pend.

Thus, to implement K-PHASE for weighted caching, it suffices to keep track of the slack on the edge $(i, i+1)$ for each node i:

K-PHASE *for Weighted Caching*

Maintain values on the nodes. Initially, all servers reside on node r_0, which has value $w(r_0)$. In response to each request,

1. If the request was to a node with value 0, and k other nodes have positive value, reduce all the positive values by the minimum positive value.
2. Raise the value of the requested node to its weight.
3. If the request was to a nonserved node, move a server from a node with value 0.

Note that in order to implement K-PHASE for weighted caching, it suffices to maintain the (nonnegative) values of the served nodes with the servers.

It is not hard to verify that K-PHASE is $\frac{k}{k-h+1}$-competitive for weighted caching. The essential observation is that the cost in moving a server from a node r_i is bounded by

[4] As in the example, we assume r_0 is to an artificial vertex $\mathbf{o}$ *s.t.* $d(\mathbf{o}, \cdot) \equiv 0$.

[5] Note that MARK is obtained by breaking ties uniformly at random in this step. The lower bound (2) used for the analysis of MARK is slightly different; to reproduce that lower bound using timings, stretch at the beginning of each phase by 1/2 rather than 1.

$A_i - B_{i+1}$, so that the net cost is bounded by $k(A_0 - B_1) + \sum_{i=1}^{N} A_i - B_{i+1}$. One then uses equation (21), $B_1 = 0$, and $A_N = B_\infty$ to simplify this bound to kB_∞; with the simplification (22) of the cost for a k-phase timing, this gives the result.[6] (We omit the details; this algorithm is a special case of GreedyDual, which is analyzed in full in lemma 3.2.1.)

If one modifies K-Phase for weighted caching by ignoring requests to served nodes, one obtains the following algorithm:

K-Phase *for Weighted Caching, Ignoring Served Requests*

Maintain values on the nodes. Initially, all servers reside on node r_0, which has value $w(r_0)$. In response to each request to a nonserved node,

1. If k nodes have positive value, reduce all the positive values by the minimum positive value.
2. Move a server from a node with value 0 and raise the value of the node to its weight.

In fact, this is Balance. To see this, imagine associating the values of served nodes with the servers. Then the values of servers are always decreased uniformly; the value of a server is otherwise changed only when it moves to a new node, at which point it is increased by the weight of the node. Thus, the value of one server minus the value of another is exactly the net work that would have been done by the former server if it were to serve the next request minus the corresponding quantity for the latter. By moving a minimum-value server one is thus moving the server that will have done the least work.

Note that Fifo is a special case of Balance.

Algorithm GreedyDual is obtained from K-Phase by relaxing step (2), so that, if the node requested is served, the value of the server may be reset anywhere between its current value (obtaining Balance) and the weight of the node (obtaining K-Phase). For paging, Balance generalizes Fifo, while K-Phase generalizes Lru. Lru is considered the better paging strategy, so perhaps for weighted caching K-Phase is preferable to Balance.

4.4 Duality Yields a Potential Function

In this section, we give an intuitive formulation of timings, use it implicitly to give a direct analysis of Balance for weighted caching, and finally show how this analysis may be recast

[6] Note that if $w(r_0) = 0$, the additive term may be taken to be 0.

as a potential function analysis.

An amortized analysis, or more specifically a potential function analysis, (see [Tar85, BM85, ST85, CL91a]), bounds the cost of a sequence of operations. Usually, such an analysis is in contrast with a worst-case analysis, which bounds the cost of the sequence by summing the worst-case costs of the individual operations.

A typical competitive analysis with a potential function is of the following form:

Competitive Analysis with a Potential Function

Given X producing a solution in response to $r = r_0 r_1 \cdots r_N$:

1. Define a potential Φ which is a function of the states of X and of OPT.
2. Show that, in response to each request r_i,
$$a \cdot x_i \leq b \cdot o_i + \Phi_i - \Phi_{i-1},$$
where $a > 0$, x_i and o_i are the costs incurred by X and OPT, respectively, in response to the ith request, and Φ_i is the value of the potential function after X and OPT have responded.
3. Sum the inequalities, showing that the cost incurred by X is bounded by
$$(b \cdot \text{opt} + \Phi_N - \Phi_0)/a$$
where "opt" is OPT's cost.
4. Show that $\Phi_N - \Phi_0$ is appropriately bounded.

One may think of a potential function analysis as transforming OPT's costs: In response to the ith request, OPT's cost is changed to $o'_i = o_i + (\Phi_i - \Phi_{i-1})/b$. The analysis then shows that OPT's overall cost is not substantially increased under the transformation, and gives a worst-case (per operation) bound on the transformed costs. That is, $b \cdot o'_i \geq a \cdot x_i$ is shown for each i.

Linear programming duality may be viewed similarly. The bound obtained by exhibiting a dual solution may often be viewed as follows: The dual solution gives a *dual transformation* of any primal solution. The dual transformation changes any primal solution's cost by an amount independent of the primal solution. The lower bound on the primal solution is then obtained by giving a simple, "localized" lower bound on the cost of the transformed primal solution, and taking into account the amount by which the transformation changed the cost.[7]

[7] For examples, consider the proof of lemma 3.1.1 or the discussion of the duals of weighted matching problems in Chapter 5.

The intuitive analysis of BALANCE for weighted caching which we give in this section uses duality in a manner very similar to a potential function argument: OPT's cost is transformed by a dual transformation, and it is shown that under the transformed cost, the cost incurred by OPT's server after it serves the ith request is at least the cost incurred by BALANCE's server after it serves the same request. The difference is that the transformation on OPT's cost increases OPT's overall cost, but by a predictable amount, and the subsequent bound correlates the costs of OPT and BALANCE to move from, rather than respond to, r_i.

Generally a dual solution can be interpreted as yielding any of a number of dual transformations. For instance, the proof of lemma 3.1.1 and the space-time formulation of timings present two different transformations for the k-server dual.

If we can view the dual problem via a dual transformation which does not increase OPT's overall cost, and which gives a "localized" lower bound of the form "in response to the ith request, OPT's cost is bounded by a (positive) constant times the on-line algorithm's cost," then we have obtained a duality transformation which is essentially a potential function.

It would seem that one can always find such a transformation; the question is how simple it and the resulting potential function are.

4.4.1 A Space-Time Formulation of Timings

Next we describe an intuitive interpretation of the k-server dual.

Let $T : \{0, \ldots, N\} \rightarrow \mathcal{R}$ be an arbitrary function. Imagine that the requests of a sequence $r = r_0 r_1 \cdots r_N$ occur at specified times: request i occurs at time $T(i)$.

Now, instead of charging the servers only to move through space (*i.e.* in the graph), imagine that the servers, except for the one about to service the next request, are also charged to move through time. Specifically, if the server of request i next services request $j > i$, then imagine that instead of incurring a cost $d(r_i, r_j)$, the server incurs a cost of

$$d_T(i, j) = d(r_i, r_j) + T(j-1) - T(i).$$

Additionally, if a server last services request i, then imagine that instead of incurring no cost to remain on the node until the final request, the server incurs a cost of

$$d_T(i, N+1) = T_N - T_i.$$

In charging to move through time as well as space, the cost of any given schedule[8] for h servers is increased by exactly $(h-1)(T(N)-T(0))$.

Next, let $M_i = \min_{j:i<j\leq N+1} d_T(i,j)$, so M_i is a lower bound on the cost in time and space incurred by the server of the ith request until serving another request, if it serves another one, or until time $T(N)$, if not. Then the net cost, in space and time, of *any* schedule for h servers is at least $hM_0 + \sum_{i=1}^{N} M_i$, and the cost in space alone is at least

$$(h-1)(T(0)-T(N)) + hM_0 + \sum_{i=1}^{N} M_i.$$

T is an alternate representation of a timing for r, and the above quantity is the cost of the timing for h servers. The correspondence between this representation of the timing and that given in lemma 3.1.1 is given by $B_j = T(j-1) - T(0)$, $A_i = T(i) - T(0) + M_i$, and $B_\infty = T(N) - T(0)$.

4.4.2 An Intuitive Analysis of BALANCE

The space-time view of timings gives a nice analysis of BALANCE for weighted caching:

Lemma 4.4.1 BALANCE *is* $\frac{k}{k-h+1}$*-competitive for weighted caching.*

Proof: Consider the schedule produced by BALANCE for a weighted caching request sequence $r = r_1 \cdots r_N$. Assume for simplicity that no request is to a node served by BALANCE (such requests are ignored by BALANCE and can be omitted without increasing the cost to OPT), and that BALANCE handles the first k requests by placing successive servers on the requested nodes at no cost. (This decreases the cost of BALANCE by at most an additive constant.)

The schedule assigns a path in the request graph to each server. Imagine that the servers simultaneously start at their respective initial vertices and begin traversing their paths at unit speed (so that the time to traverse an edge is its length), not pausing at vertices. When a server finishes traversing its path, imagine that it continues moving (anywhere) in the graph until all servers have finished traversing their paths. Call the distance traveled by the servers, including this extra movement, the *cost in space* of BALANCE.

[8] Recall that all servers are assumed to begin on r_0.

Let $T(0)$ be the starting time, and let $T(i)$ denote the time that a server arrives at r_i in response to request i, so that $T(i)$ is also the net distance traveled by the server by the time it arrives.

The definition of BALANCE implies that for weighted caching T is monotonic: if $T(i+1) < T(i)$ then the server of request $i+1$ would have been chosen as the server of request i instead, since the net distance traveled by the server after serving the ith request would have been less than that of the server actually chosen to serve the request.

Next, consider changing the charges for any schedule to service r by charging servers to move through space (in the graph) and, in addition, from time $T(0)$ until $T(N)$, charging each server, *except the one which will next service a request*, at a unit rate to exist in time.

Specifically, for moving a server from r_i to r_j, imagine charging $d_T(i,j) = d(r_i, r_j) + T(j-1) - T(i)$, instead of $d(r_i, r_j)$, and for leaving a server on r_i for the remainder of the schedule, imagine charging $d_T(i, N+1) = T(N) - T(i)$.

Clearly the cost in time and space (*i.e.* the cost under d_T) of any schedule for h servers is exactly $(h-1)(T(N) - T(0))$ plus the cost in space alone (*i.e.* the cost under d).

Since k servers are moving at each time from $T(0)$ until $T(N)$, the cost in space of BALANCE is $k(T(N) - T(0))$.

Let B_n denote the set of k requests which have servers remaining on them after the nth request. We claim that for any i and j, with $0 \le i < j \le N+1$

$$d_T(i,j) \ge \begin{cases} w(r_i) & i \notin B_N \text{ and} \\ T(N) - T(i) & i \in B_N. \end{cases} \tag{23}$$

(We postpone the verification of this claim until the end of the proof.)

Thus, the cost in space and time incurred by OPT's server in the interval between serving the ith request and serving its next request (if any, and otherwise until time $T(N)$) *is at least the cost in space incurred by the corresponding server of* BALANCE.

Consequently, the cost in space and time of OPT is at least the cost in space of BALANCE, and the cost in space of OPT is at least the cost in space of BALANCE, minus the cost in time of OPT. Thus,

$$\mathcal{C}_r(\text{OPT}, h) \ge (k - h - 1)(T(N) - T(0)). \tag{24}$$

Since $\mathcal{C}_r(\text{BALANCE}, k)$ is bounded by the cost in space of BALANCE, this gives the result.

It remains only to verify the claim (23). Recall that $0 \leq i < j \leq N+1$, T is monotonic, and we have assumed there are no requests nodes served by BALANCE.

There are four cases:

i. If $i \notin B_N$, $j \leq N$, (23) becomes $d(r_i, r_j) + T(j-1) - T(i) \geq w(r_i)$.
This is clear if $d(r_i, r_j) = w(r_i)$. Otherwise, $r_i = r_j$, so the server of r_i moved at time $i' < j$ to service a request. Thus, $T(j-1) \geq T(i') = T(i) + w(r_i)$.

ii. If $i \notin B_N$, $j = N+1$, (23) becomes $T(N) - T(i) \geq w(r_i)$.
Since $i \notin B_N$, the server of r_i moved at time $i' \leq N$ to service a request. Thus, $T(N) \geq T(i') = T(i) + w(r_i)$.

iii. If $i \in B_N$, $j \leq N$, (23) becomes $d(r_i, r_j) + T(j-1) - T(i) \geq T(N) - T(i)$.
Since $i \in B_N$, request j is not to r_i. Thus, $d(r_i, r_j) = w(r_i)$ and this case reduces to $w(r_i) + T(j-1) \geq T(N)$. This follows from $w(r_i) + T(i) \geq T(N)$, which is true because otherwise the server of r_i would have served r_N.

iv. If $i \in B_N$, $j = N+1$, (23) becomes $T(N) - T(i) \geq T(N) - T(i)$.

□

4.4.3 A Potential Function Analysis of BALANCE

Next we sketch how to transform the analysis in lemma 4.4.1 of BALANCE for weighted caching into a potential function analysis.

Consider the schedule produced by BALANCE for weighted-caching request sequence $r = r_1 r_2 \cdots$, assuming that BALANCE handles the first k requests by placing successive servers on the requested nodes at no cost. Assume also that BALANCE receives no requests to served nodes.

Let $T(i)$ be defined as in the the proof of lemma 4.4.1: $T(i)$ is distance traveled by the server of the ith request from its first request until serving the ith request.

The fundamental bound (case (i) of claim (23)) from that analysis is

$$d(r_i, r_j) \geq w(r_i) + T(i) - T(j-1) \quad \text{for } 0 < i < j.$$

What we want is a series of values $\Phi_k, \Phi_{k+1}, \ldots$ such that, for $j = k+1, k+2, \ldots$, the quantity $\Phi_j - \Phi_k$ is bounded, and

$$(k-h+1)d(r_{j^-}, r_j) \leq kd(r_{j^*}, r_j) + \Phi_j - \Phi_{j-1}, \tag{25}$$

where j^- and j^* denote, respectively, the requests last served by the servers of BALANCE and OPT that served request j.

Observing that $d(r_{j^-}, r_j) = w(r_{j^-})$ and $d(r_{j^*}, r_j) \geq w(r_{j^*}) + T(j^*) - T(j-1)$, we try $\Phi_k = 0$ and (for $j > k$)

$$\Phi_j = \Phi_{j-1} + (k-h+1)w(r_{j^-}) - k(w(r_{j^*}) - T(j-1) + T(j^*)).$$

We know this satisfies (25); if we can show it is appropriately bounded we will have the desired potential function.

Next we simplify Φ_n. O_n denotes the set of i such that OPT's server of the ith request does not serve another request until after request n, so $|O_n| = h$. Similarly, B_n denotes the corresponding set for BALANCE.

The simplification is a consequence of the following three equations:

$$\begin{aligned}
\sum_{j=k+1}^{n} w(r_{j^-}) &= \sum_{j \in B_n} T(j), \\
\sum_{j=k+1}^{n} w(r_{j^*}) &= \sum_{j=1}^{n} w(r_j) - \sum_{j \in O_n} w(r_j) \\
&= \sum_{j \in B_n} T(j) + w(r_j) - \sum_{j \in O_n} w(r_j), \quad \text{and} \\
\sum_{j=k+1}^{n} T(j-1) - T(j^*) &= \left(\sum_{j=k+1}^{n} T(j) - T(j^*) \right) - T(n) \\
&= \left(\sum_{j \in O_n} T(j) \right) - T(n).
\end{aligned}$$

From these equations it is straightforward to show

$$\begin{aligned}
\Phi_n &= \sum_{j=k+1}^{n} (k-h+1)w(r_{j^-}) - k(w(r_{j^*}) - T(j-1) + T(j^*)) \\
&= \sum_{j \in B_n} (h-1)(T(n) - T(j) - w(r_j)) - (k-h+1)w(r_j) \\
&\quad - \sum_{j \in O_n} k(T(n) - T(j) - w(r_j)).
\end{aligned}$$

This gives the desired potential function. This isn't too bad: Consider fixing a j and letting $n = j, j+1, \ldots$. The quantity $T(j) + w(r_j) - T(n)$ is $w(r_j)$ when BALANCE's server serves r_j, and, as n increases, the term decreases uniformly with the corresponding terms for other j. When it becomes 0, the server leaves the node to serve its next request.

To see that Φ_n is nonpositive, rewrite it:

$$\begin{aligned}
\Phi_n &= -\sum_{j \in B_n \cap O_n} (k-h+1)(T(n) - T(j)) \\
&\quad + \sum_{j \in B_n - O_n} (h-1)(T(n) - T(j) - w(r_j)) \\
&\quad - \sum_{j \in O_n - B_n} k(T(n) - T(j) - w(r_j)) \\
&\quad - \sum_{j \in B_n - O_n} (k-h+1) w(r_j) \\
&\leq -\sum_{j \in B_n - O_n} (k-h+1) w(r_j).
\end{aligned}$$

The inequality follows because for $j \leq n$, $T(n) \geq T(j)$, $T(n) \geq T(j) + w(r_j)$ (if $j \in B_n$), and $T(n) \leq T(j) + w(r_j)$ (if $j \notin B_n$).

Is this potential function analysis in fact a duality transformation, as discussed at the beginning of this subsection? Yes. Briefly, we have

$$\begin{aligned}
(k-h+1) d(r_{j^-}, r_j) &\leq k d(r_{j^*}, r_j) + \Phi_j - \Phi_{j-1} \quad \text{with} \\
\Phi_j - \Phi_{j-1} &= (k-h+1) d(r_{j^-}, r_j) - k(w(r_{j^*}) - T(j-1) + T(j^*)),
\end{aligned}$$

and we want a duality transformation d_Φ on the edge costs so that

$$(k-h+1) d(r_{j^-}, r_j) \leq k d_\Phi(i,j).$$

It is straightforward to derive

$$\begin{aligned}
d_\Phi(i,j) &= d(r_i, r_j) - T(i) - w(r_i) + T(j-1) + \frac{k-h+1}{k} w(r_{j^-}) \\
&= d_T(i,j) - w(r_i) + \frac{k-h+1}{k} w(r_{j^-})
\end{aligned}$$

as the desired duality transformation. We can view this as the transformation d_T modified so that a little bit of the increase in the overall cost of OPT under d_T is given back at each step, so that the overall cost is not increased by the modified transformation.

In sum, d_Φ is a duality transformation, analogous to d_T, for interpreting the lower bound given by a dual solution. The interpretation corresponding to d_Φ is as follows. Given an arbitrary server schedule for r for h servers, imagine changing the cost of serving r_j with the server which last served r_i from $d(r_i, r_j)$ to $d_\Phi(i, j)$. This transformation does not increase the overall cost for h servers, and the transformed costs satisfy $kd_\Phi(i, j) \geq (k - h + 1)d(r_{j-}, r_j)$. Thus, BALANCE is $k/(k - h + 1)$-competitive. The transformation d_Φ is in some sense equivalent to a potential function analysis with Φ.

4.5 K-Phase Timings, PERM, and Optimal Timings

In this section we briefly describe how PERM, the on-line assignment algorithm, yields an on-line k-server algorithm closely related to K-PHASE. In doing so, we also describe a method for generating optimal timings which produces timings with some suggestive structure.

The transformation of the k-server problem to an F_k-factor problem in Section 4.1 preserves the on-line nature of the problem: in response to request r_i with $(i > 0)$, the costs of edges adjacent to demand vertex i in G^N are revealed, and if edge (i, j) is added to the F_k-factor, a server is moved from r_i to r_j.

This on-line F_k-factor problem reduces (by splitting supply vertex 0 into k identical vertices) to the on-line assignment problem as defined in Section 1.6. PERM, the on-line assignment algorithm, reduces in this way to PERM for k servers, which, in response to each request, moves a server so that the servers cover vertices that would be covered by OPT after the request if it was the last request.

To describe this algorithm more constructively, we define *alternating paths*: Given a partial F_k-factor (a subset of edges so that each vertex i has degree at most $F_k(i)$ in the subset) among the admissible edges of a timing for G^N,

- An *alternating path* is a path in G^N whose odd edges are admissible and whose even edges are in the partial F_k-factor.
- An *augmenting path* is an odd-length alternating path, with each endpoint i of degree less than $F_k(i)$ in the partial factor.
- To *augment* the factor by an augmenting path is to remove the even edges of the path from the factor and add the odd edges, so that the degree of each endpoint in the

partial factor is increased by 1.

Now the algorithm may be described more constructively as follows:

PERM *for K-Servers*

Maintain an optimal timing and F_k-factor in the current G^j. In response to request r_j,

1. Extend the timing. (The timing ceases to be optimal; the factor becomes partial.)
2. While there is no augmenting path (necessarily from demand node j to supply node j),
   (a) Let U^* denote the set of supply vertices reachable from supply node j by alternating paths.
   (b) Stretch U^*.
3. Augment the partial factor by the augmenting path. (The timing and the factor are now optimal.)
4. Move a server from r_i such that (i, ∞) is not an edge in the current F_k-factor.

It is fairly easy to show that in each stretch operation, $|U^*| = |W^*| + 1$, so that invariant (21) and the simplification of the cost (22) also hold for the optimal dual solution produced in the above algorithm, so that the cost for h servers of the solution is

$$(k - h + 1)\mathcal{C}_r(k, \text{OPT}) - (k - h)A_0.$$

Note the relation between this algorithm and K-PHASE. K-PHASE may be viewed as maintaining a nonoptimal timing (the k-phase timing) and a (very) partial F_k-factor consisting of the k admissible edges (i, ∞) such that ∞ is the only admissible neighbor of i. In response to each request, the timing is stretched just enough so that the partial factor may be maintained, and a server is moved from r_i such that (i, ∞) is no longer an edge in the partial factor.

Chapter 5

Duality Analyses of Weighted Matching Strategies

In this chapter we use the dual bounding technique to analyze the maximum-weight matching heuristic GreedyMax2, the on-line weighted matching algorithm GreedyMax3, and the on-line assignment algorithm Perm. The purpose is to explore the general applicability of the dual bounding technique.

The dual problem of finding an assignment in a weighted, bipartite graph with edge weights is to find a maximum-cost potential (weighting of the vertices) such that the weight of any edge is at least the sum of the weights of its endpoints. The cost of the potential is the sum of the weights. (If we are abusing the term "perfect matching", and the bipartite graph has one part larger than the other, then the weights on the larger part must be nonnegative in the dual problem.) Again, by duality, the cost of any potential is a lower bound on the cost of any perfect matching.

To see the lower bound directly in terms of a dual transformation, imagine that the edge weights $d(i,j)$ are modified to $d(i,j) - \pi_i - \pi_j$, where π_x denotes the weight of vertex x. This reduces the cost of the matching by (at least) $\sum_x \pi_x$, but leaves the cost nonnegative.

The dual problem of finding a maximum-cost matching in a weighted, bipartite graph with nonnegative edge weights is to find a minimum-cost nonnegative potential such that such that the length of any edge is *at most* the sum of the weights of its endpoints. The cost of any such potential is an upper bound on the cost of any matching.

To see this upper bound directly, again imagine that the edge weights $d(i,j)$ are modified to $d(i,j) - \pi_i - \pi_j$. This reduces the cost of the matching by $\sum_x \pi_x$, and leaves the cost nonpositive.

For nonbipartite graphs, the above bounds also hold, and are useful, even though they cannot necessarily be made tight.[1]

The remainder of this chapter consists simply of the three analyses.

Lemma 5.0.1 ([Avi83]) *The cost of the matching produced by* GreedyMax2 *in an arbitrary, nonnegatively weighted graph is within a factor of 2 of maximum.*

Proof: If GreedyMax2 adds an edge (i,j) to the matching, let $\pi_i = \pi_j = d(i,j)$. Let all other $\pi_i = 0$.

For any edge (i,j), $d(i,j) \leq \max\{\pi_i, \pi_j\} \leq \pi_i + \pi_j$: clearly this holds for any matched edge, and the only reason an edge remains unmatched is some adjacent edge, say (i,k), was matched first, in which case $d(i,j) \leq d(i,k) = \pi_i$.

The cost of the matching is $\sum_i \pi_i/2$. Since $\sum_i \pi_i$ is an upper bound on the maximum cost of a matching, the cost of the matching is within a factor of 2 of maximum. □

Lemma 5.0.2 ([KP91]) *In any metric, bipartite graph, the cost of the matching produced by* GreedyMax3 *is within a factor of 3 of maximum.*

Proof: When GreedyMax3 adds an edge (i,j) to the matching as a result of the presentation of vertex j, let $\pi_i = 2d(i,j)$ and $\pi_j = d(i,j)$.

For any edge (i,j), $d(i,j) \leq \pi_i + \pi_j$: If (i,j) is matched this is clear. Otherwise, suppose j was presented and matched to k. If i was not yet matched at that point, then $d(i,j) \leq d(j,k) = \pi_j$. Otherwise, suppose i was already matched to h. When h was presented, k was not yet matched, so $d(k,h) \leq d(i,h)$. Thus,

$$d(i,j) \leq d(i,h) + d(h,k) + d(k,j) \leq 2d(i,h) + d(k,j) = \pi_i + \pi_j.$$

The cost of the matching is $\sum_i \pi_i/3$. Since $\sum_i \pi_i$ is an upper bound on the maximum cost of a matching, the cost of the matching is within a factor of 3 of maximum. □

[1] The primal problems require more constraints in the nonbipartite case to ensure there are optimal solutions corresponding to matchings. Thus, the dual problems allow a larger class of solutions, and possibly tighter bounds. See [PS82, p.255].

The next analysis is a bit more complicated, and uses terminology ("stretching" a dual solution, "alternating" and "augmenting" paths) from Section 4.5.

Lemma 5.0.3 ([KP91]) *In any $2n$-node, metric, bipartite graph, the assignment produced by* PERM *is within a factor of $2n-1$ of minimal.*

Proof: PERM, when given the graph $G = (U, W, U \times W)$, may be described as follows:

PERM

At any given time, let P denote the subset of W presented so far. Imagine maintaining an optimal potential and a minimum-cost matching[2] in the subgraph $G_P = (U, P, U \times P)$. Initially, the potential is the zero potential. In response to the presentation of a vertex j,

1. $P \leftarrow P \cup \{j\}$.
2. Extend the potential by adding $\pi_j = 0$, and consider the previous minimum-cost matching as a partial matching in the new G_P.
3. While there is no augmenting path (necessarily from node j) in G_P,
   (a) Let S denote the set of vertices reachable from vertex j by alternating paths.
   (b) Stretch S. (Modify the potential by increasing the weights on vertices in $S \cap P$, and decreasing those on vertices in $S \cap U$, by ϵ, where $\epsilon > 0$ is as large as possible so that no constraints are violated.)
4. Augment the partial matching by the augmenting path. (The matching and the factor are now optimal.)
5. Assign i to j in the assignment, where i is the other endpoint of the augmenting path.

To bound the cost of the resulting assignment, note that the cost of edge (i, j), where i, the other endpoint of the augmenting path, is assigned to j after j is presented, is bounded by the cost of the augmenting path. The even edges of the path (if any) form a subset of the previous minimum cost assignment, of cost no more than the previous potential. The odd edges of the path form a subset of the current minimum cost assignment, of cost no more than the current potential.

Since the cost of the potential is only increased during the course of the algorithm, and the first augmenting path consists of one edge, after $j \leq n$ vertices are presented and assigned, the cost of the assignment is bounded by $2j - 1$ times the cost of the current potential. □

[2] To distinguish between this minimum-cost matching and the assignment being produced by PERM, we refer to the former as a matching and the latter as the assignment.

Chapter 6

Conclusion

We conclude with some remarks, directions for future work, and a summary of the thesis.

6.1 Remarks

Competitive analysis for on-line problems is not a new area. For instance, competitive ratios of on-line bin-packing strategies have been considered for years (*e.g.* Coffman, Garey, and Johnson [CGJ83]).

Can competitive analyses be made more realistic and practical? Existing models are useful in that they provide a theoretical framework for judging on-line algorithms. For a problem for which empirical data is not extensive, for instance in designing multi-processors or configuring network communication, competitive analysis can provide guidelines, or even guarantees of good performance [KMRS88]. Nonetheless, competitive ratios are worst-case estimates, and, if high, may not be representative of typical behavior. An analogue exists here to NP-completeness, which is evidence that a problem is hard in the worst case, but not necessarily in the typical case. In other words, the worst-case pessimism that existing models for competitive analysis sometimes suffer from is representative of a more fundamental problem in current theoretical computer science: how to (when possible) obtain theoretically rigorous models for analyzing typical rather than worst-case or average-case behavior. Perhaps on-line problems are a good arena in which to explore this issue.[1]

[1] One alternative model for paging might be the following: Assume the input sequence is generated by a finite-state markov process. When the process enters a state, it requests an item depending only on the state.

How useful is the dual-bounding technique? Generally, duality helps understand structure, so it is not surprising duality is useful for analyzing optimization heuristics. Further, half of obtaining a competitive analysis is (at least implicitly) obtaining a useful bound on the optimal cost; the other half is correlating this bound with an on-line solution. At the very least, explicitly using duality can simplify the former: for linear optimization problems, unlike more general problems, obtaining a bound on all instances is simply a dual problem of exhibiting a single instance.

Perhaps the interesting refinement of the question is whether the latter — correlating the lower bound with an on-line algorithm — is made easier by considering duality. Perhaps on-line algorithms use duality in a consistent way. Competitive algorithms always have potential function analyses, the problem is finding and analyzing the potential function; perhaps, for linear optimization problems, the class of interesting potential functions is essentially a subset of the duality transformations, as discussed in Section 4.4, and the special structure of these problems can aid the search for potential functions.

6.2 Future Work

6.2.1 Randomized Weighted Caching

Currently no randomized weighted caching algorithm with competitiveness below $k/(k-h+1)$ is known, nor is a lower bound suggesting there is not room for substantial improvement with randomized strategies. MARK may be viewed as using randomization to take advantage of degeneracy in k-phase timings to reduce competitiveness. For weighted caching, optimal timings have a somewhat simpler structure than for the general k-server problem. Perhaps a better understanding of this structure could lead to progress on randomized weighted caching.

This might be appropriate for analyzing adaptive deterministic algorithms; from preliminary consideration, it would seem that such algorithms could in this model be as competitive as *barely random algorithms*: algorithms that use a number of random bits bounded independently of the length of the input. (MARK can be made barely random by randomly ordering the k servers, and breaking ties according to the random ordering instead of uniformly at random each time.)

Prabhakar Raghavan spoke at the DIMACS Workshop on On-Line Algorithms (hosted by Lyle McGeoch and Danny Sleator at Rutgers University in February 1991) of a model for analyzing investment strategies. The assumption was that the input would satisfy certain characteristics (*e.g.* certain investments would be more volatile than others, but all would, in the long run, have similar net growth), but, subject to that restriction, the worst-case input would be considered.

6.2.2 Lookahead

The obvious practical notion of lookahead does not yield reduced competitiveness, while resource-bounded lookahead yields reduced competitiveness but is obviously not practical. The challenge here is to find a compromise: a notion of lookahead for paging which yields some advantage yet is arguably realistic.[2]

6.2.3 Loose Competitiveness

Lower bounds on the loose competitiveness of LRU, FIFO, and MARK are open, and might require only a clever modification of the lower bound to FWF.

No work has yet been done on loose competitiveness for general k-server or the weighted caching algorithms.[3]

The key to showing loose competitiveness for paging was the idea of k-phases, particularly the more general lower bound on the optimal solution in terms of the number of new requests per phase. Of course we speculate that linear programming duality might be useful in developing the more general lower bounds needed to show loose competitiveness for k-server problems more general than paging.

6.2.4 The K-Server Dual

The structure of optimal timings should be explored. In particular,

- We conjecture that PERM for k servers (WORK_0) is k-competitive, or even $\frac{k}{k-h+1}$-competitive, for weighted caching, although it is easily shown not competitive for the

[2]One should keep in mind that in practice on-line paging strategies may have costs close enough to optimal that the improvement from lookahead is marginal, or at least impossible to model in a realistic *and* reasonably general way. In that case, we can only hope that our theoretical model provides heuristic evidence about the relative merits of various algorithms, or suggestions or guidelines about how to design good algorithms. In this light, resource-bounded lookahead is not so bad.

[3]One might want to focus first on $\frac{k}{k-h+1}$-competitive strategies, where they exist. For any such strategy, and any k, the cost of the schedule for k servers produced by the on-line strategy is within a factor of 2 of the cost of the optimal schedule for $k/2$ servers. Thus if the cost of the on-line schedule for k servers is high in comparison to the cost of the optimal schedule with k servers, then the cost of the optimal schedule with k servers is low in comparison to the cost of the optimal schedule with $k/2$ servers. Thus one can show, for instance, that if, for every k in some range, the cost of the on-line schedule for k servers is at least $c > 2$ times the cost of the optimal schedule for k servers, then the cost of the optimal schedule with k servers shrinks by a factor of $2/c$ every time k is doubled. This can show a weak variant of loose competitiveness: either the cost of the strategy rapidly becomes insignificant, or the competitive ratio for *some* k is less than c.

general k-server problem. A proof of this might follow similar lines as the analysis of K-PHASE, if the structure of the optimal timings discussed in Section 4.5 were better understood. For instance, the monotonicity of the B_j, an essential property of k-phase timings, seems to hold for the optimal timing implicit in PERM for k servers provided $w(r_0) = 0$.

- We know that the cost for h servers of the optimal timing for k servers discussed in Section 4.5 may be expressed

$$(k - h + 1)\mathcal{C}_r(\text{OPT}, k) - (k - h)A_0.$$

A better understanding of the A_0 term might yield bounds on the relative costs of optimal schedules for different numbers of servers on a given sequence.

It would be nice if some connection could be made between the combinatorial approach we have taken for competitive analysis and the *residue*-based approach (see McGeoch [McG87] and Manasse, McGeoch, and Sleator [MMS90]) or if further relations could be established with potential functions. A starting point for investigating further the connections with potential functions might be Chrobak and Larmore's [CL91a] potential function analysis of their algorithm for k servers on trees, particularly the case when the tree is a "star," (A tree with one nonleaf — this special case corresponds to weighted caching) because of the similarity of the tree-based algorithm when the tree is a star and GREEDYDUAL.

The relations between duality and potential functions are preliminary and warrant further investigation. As of this writing we have not attempted to transform the other uses of the dual bounding technique into potential function analyses; this would be a natural thing to attempt.

We wonder if the partitioning paging strategy of McGeoch and Sleator [MS89] can be viewed as a dual-guided algorithm.

6.3 Summary

We have seen a number of variations on the standard model for competitive analysis of paging strategies: allowing randomized strategies, allowing resource-bounded lookahead,

and loose competitiveness. Each led to substantially reduced, but not constant, competitive ratios.

The dual bounding technique — using a dual solution to bound the optimal cost and obtain a competitive analysis — was introduced and shown to be implicit in the work on paging; by recognizing this, a new, optimally competitive, deterministic, on-line algorithm was given for weighted caching, a more general problem. The algorithm subsumes well-known existing algorithms, and the analysis strengthens a previous analysis.

The structure of the linear programming dual of the k-server problem was explored. The k-server problem was seen to be a special case of the assignment problem, and on-line algorithms for weighted caching and paging were shown to be related, via duality, to an on-line assignment algorithm. An intuitive analysis, using duality, of a weighted caching strategy was given, and duality considerations were shown to lead to a potential function.

The dual bounding technique was applied to analyze an existing heuristic and an existing on-line algorithm for maximum-cost matching, and an on-line assignment algorithm.

Many fundamental topics remain to be explored:

- Practical and interesting models for lookahead.
- Lower bounds on loose competitiveness for LRU, FIFO, and MARK.
- Reduced competitiveness for weighted caching and k-server problems through loose competitiveness and/or randomization.
- The structure of optimal weighted caching and k-server dual solutions.
- The general applicability of the dual bounding technique and its relation to potential functions and residues.
- Models yielding better analyses of typical behavior when worst-case behavior is far from typical.

Bibliography

[AACS89] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. Research report RC 15118 (#67337), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY, October 1989.

[ALM90] Sanjeev Arora, Tom Leighton, and Bruce Maggs. On-line algorithms for path selection in a nonblocking network. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 149–158, May 1990. Baltimore, MD.

[Avi83] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[BDBK+90] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 379–386, May 1990. Baltimore, MD. *Algorithmica*, to appear.

[Bel66] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[BGRS90] M. Bern, D. H. Greene, A. Raghunathan, and M. Sudan. On-line algorithms for locating checkpoints. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 359–368, May 1990. Baltimore, MD.

[BIRS91] Allan Borodin, Sandy Irani, Prabhaker Raghavan, and Baruch Schieber. Competitive paging with locality of reference. In *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pages 249–259, May 1991. New Orleans, LA.

[BKT90] P. Berman, H. Karloff, and G. Tardos. A competitive 3-server algorithm. In *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 280–290, January 1990. San Francisco, CA.

[BLS87] A. Borodin, M. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th Annual ACM Symp. on Theory of Computing*, pages 373–382, May 1987. New York, NY. JACM, to appear.

[BM85] J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Comm. ACM*, 28(4):404–411, April 1985.

[BS89] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Tech. Rep. CMU-CS-89-201, Department of Computer Science, Carnegie Mellon University, 1989.

[CCF85] A. Calderbank, E. Coffman, and L. Flatto. Sequencing problems in two-server systems. *Math. Operations Research*, 10:585–598, 1985.

[CDRS90] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 369–378, May 1990. Baltimore, MD.

[CGJ83] E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM Journal of Computing*, 12:227–258, 1983.

[CKPV90] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 291–300, January 1990. San Francisco, CA.

[CKPV91] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, May 1991.

[CL] Marek Chrobak and Lawrence L. Larmore. A new approach to the server problem. *SIAM J. Discrete Math.* To appear.

[CL91a] M. Chrobak and L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Computing*, 20(1):144–148, February 1991.

[CL91b] Marek Chrobak and Lawrence L. Larmore. Server problems and on-line games. In Lyle A. McGeoch and Daniel D. Sleator, editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 11–64. American Mathematical Society, 1991.

[FKL+88] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. Tech. Rep. CMU-CS-88-196, Computer Science Department, Carnegie Mellon University, 1988. *J. Algorithms*, to appear.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. In *Proc. 31st Annual Symp. on Foundations of Comp. Sci.*, volume II, pages 454–469, October 1990. St. Louis, MO.

[GL88] A. Gyarfas and J. Lehel. On-line first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988.

[Gro91] Eddie Grove. The harmonic on-line server algorithm is competitive. In *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pages 260–266, May 1991. New Orleans, LA.

[GS53] D. Gale and F. M. Stewart. Infinite games with perfect information. In W. H. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games Vol. II, Annals of Mathematical Studies*, volume 28, pages 245–266. Princeton University Press, 1953.

[Hen76] W. J. Hendricks. An account of self-organizing systems. *SIAM Journal of Computing*, 5(4):715–723, December 1976.

[IR] Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*. To appear.

[IRWS91] S. Irani, N. Reingold, J. Westbrook, and D. Sleator. Randomized competitive algorithms for the list update problem. In *Proc. 2nd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 251–260, January 1991. San Francisco, CA.

[Kah91] S. Kahan. A model for data in motion. In *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pages 267–277, May 1991. New Orleans, LA.

[KMMO90] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 301–309, January 1990. San Francisco, CA.

[KMRS88] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[KMV90] S. Khuller, S. Mitchell, and V. Vazirani. On-line algorithms for weighted matching and stable marriages. Tech. Rep. TR 90-1143, Department of Computer Science, Cornell University, 1990.

[KP91] Bala Kalyanasundaram and Kirk Pruhs. On-line weighted matching. In *Proc. 2nd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 234–240, January 1991. San Francisco, CA.

[KRR91] H. Karloff, Y. Rabani, and Y. Ravid. Lower bounds for randomized k-server and motion planning algorithms. In *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pages 278–288, May 1991. New Orleans, LA.

[KVV90] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proc. 22nd Annual ACM Symp. on Theory of Computing*, pages 352–358, May 1990. Baltimore, MD.

[Lov89] László Lovász. Communication complexity: A survey. Tech. Rep. CS-TR-204-89, Computer Science Department, Princeton University, February 1989.

[McG87] L. A. McGeoch. *Algorithms for Two Graph Problems*. PhD thesis, Carnegie Mellon University, 1987.

[McG91] Lyle McGeoch. Personal communication. 1991.

[MMS88] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symp. on Theory of Computing*, pages 322–333, May 1988. Chicago, IL.

[MMS90] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[MS89] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. Tech. Rep. CMU-CS-89-122, Computer Science Department, Carnegie Mellon University, 1989. *Algorithmica*, to appear.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Riv76] R. Rivest. On self-organizing sequential search heuristics. *Comm. ACM*, 19(2):63–67, February 1976.

[RS89] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. In *16th International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 687–703. Springer-Verlag, July 1989. Revised version available as an IBM Research Report.

[RT81] E. Reingold and R. Tarjan. On a greedy heuristic for complete matching. *SIAM Journal of Computing*, 10:676–681, 1981.

[RW90] N. Reingold and J. Westbrook. Randomized algorithms for the list update problem. Tech. Rep. YALEU/DCS/TR-804, Department of Computer Science, Yale University, June 1990.

[SA88] R. L. Sites and A. Agarwal. Multiprocessor cache analysis using ATUM. In *Proc. 15th IEEE Int. Symp. on Computer Architecture*, pages 186–195, 1988. Honolulu, HI.

[Sch90] Baruch Schieber. Personal communication. 1990.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, February 1985.

[Tar85] R. E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Math.*, 6:306–318, 1985.

[Yao82] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 17th Annual Symp. on Foundations of Computer Science*, pages 80–91, November 1982. Chicago, IL.

[You] Neal Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, to appear in a special issue on on-line algorithms.

[You91] Neal Young. On-line caching as cache size varies. In *Proc. 2nd Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 241–250, January 1991. San Francisco, CA.

Index