This is to certify that the
thesis entitled

RESTRICTED *K*-SERVER PROBLEM

presented by

Jignesh Patel

has been accepted towards fulfillment
of the requirements for the

M.S. degree in Computer Science

Major Professor's Signature

6/11/2004

Date

MSU is an Affirmative Action/Equal Opportunity Institution

RESTRICTED K-SERVER PROBLEM

By

Jignesh Patel

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science and Engineering

2004

ABSTRACT

RESTRICTED K-SERVER PROBLEM

By

Jignesh Patel

The k-server problem is the problem of deciding how to use k servers in a metric space to serve requests arriving in the metric space in an online manner. Many practical online problems can be modeled using the k-server problem. Although the k-server problem has been studied extensively, little work has been done about its generalizations, and the work done focuses on caching, a special case of the k-server problem. Many practical problems which cannot be modeled using the simple k-server problem can be modeled by some generalization of it.

In this thesis, we first discuss the current work that has been done for the simple k-server problem. Then we consider a particular generalization in which the servers are not identical. We have types of requests and each server can serve only certain types of requests. We consider the simple problem with two server types and two request types. One server type can serve both request types and the other server type can serve only one request type.

We consider the problem with three metric spaces. First the general metric space, then the line metric space, and last the uniform metric space. For the general metric space we present a partial result for the BALANCE2 algorithm. For the line metric space we show that a modified version of the DOUBLE-COVERAGE algorithm is competitive. We give competitive algorithms for the uniform metric space.

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all the people who have helped me make this thesis possible.

First of all, I would like to thank my advisor, Dr. Eric Torng, for his constant guidance, support and encouragement throughout the course of the thesis. I would like to thank Dr. Richard Enbody and Dr. Abdol H Esfahanian for being on my thesis committee. I would like to thank Stephen Wagner and Martin Porcelli for their ideas and co-operation. I would also like to thank the staff of the CSE department for their support.

Finally I would like to thank my friends and family for all their help and encouragement.

Contents

List of Figures

Chapter 1

Introduction

A problem consists of input and output, and is defined by the relation between the two. An algorithm which solves the problem produces the desired output for any given input. Generally we assume that the entire input data is available before any output is produced. For example, for the problem of finding the shortest path in a graph, we have the entire graph as the input.

For many practical problems this is not true. That is, we may not have the complete input, and output has to be produced with the incomplete input. An important class of such problems is online problems.

1.1 Online Problems

An *online* problem consists of a sequence of inputs and a sequence of outputs, one output for each input. The important thing is that there is temporal (or some other) separation between the inputs in the input sequence. Each input is available at a particular time and the corresponding output has to be produced before the next input becomes available. That is, an online algorithm runs in real time or 'on line'.

A very common example of an online problem in computer science is the paging problem. The input to the problem is a sequence of page requests. For each page request, if the page is already in the fast memory, or there is room for the page in the fast memory, then there is no output; otherwise the output is a page to be evicted from the fast memory to make room for the current page. The algorithm for deciding which page to evict, if any, is an online algorithm since it has to produce the current output before it gets the next input. In paging the goal is to minimize the number of page faults. Note that the best output for any input depends on future inputs.

This is what makes online problems difficult to solve. Since the optimal output depends on future inputs, which are not available, typically we cannot generate an optimal output. If we know the whole input sequence at the start, then we can generate output for each input such that the output sequence is optimal. This is referred to as the *offline* version of the online problem. With the online version the goal then is to get as close to the optimal as possible. Since we can never get the optimal output, the natural question we can ask is how close are we to the optimal. This leads to the metric, called *competitive ratio*, which is used to compare the performance of online algorithms.

1.2 Competitive Ratio

The notion of competitive ratio was first defined by Manasse, McGeoch, and Sleator [10, 11]. The same idea was introduced earlier by Sleator and Tarjan [12]. Unlike traditional algorithms where the performance metric is usually the running time, for online algorithms the running time is usually ignored. Though algorithms with any running times are theoretically important, most online algorithms need

to be fast since they run in a real time setting. As mentioned in the last section, what we look for is how close is the output generated to the optimal output. Like in the case of running time analysis, we perform worst case analysis. Following is the definition of competitive ratio. $\text{COST}_X(\sigma)$ is the cost (or value, the quantity we want to maximize/minimize) of the output sequence generated by online algorithm X on input sequence σ. An online algorithm is generally referred as ON and the corresponding offline optimal algorithm is referred as OPT.

Definition 1. *An online algorithm* ON *has a competitive ratio of* α *if for all input sequences* σ, $\text{COST}_{\text{ON}}(\sigma) \leq \alpha\text{COST}_{\text{OPT}}(\sigma) + \beta$, *where* β *is a constant independent of* σ.

An algorithm with competitive ratio α is said to be α-competitive. An algorithm for which no such α exists is said to be non-competitive. If an algorithm is α-competitive, then no matter what the input sequence is, its solution will not be worse than α times the optimal solution (plus a constant). This intuitively seems like a good performance metric, though competitive analysis does have some drawbacks. An algorithm with a bad competitive ratio might perform well on average. Nevertheless, competitive analysis is a very useful theoretical tool.

1.3 Examples

We have already seen one example of an online problem, paging. Online problems occur in many practical situations in computer science and other areas. We give some examples of online problems.

Job Scheduling:

The general problem is as follows: we have one or more processors and jobs to be processed by them. The jobs arrive over time and the problem is to determine when

and where to schedule the jobs. The quantity to be optimized could be something like the total execution time or the total waiting time. It is an online problem because the jobs arrive over time and the algorithm has to produce output (that is, schedule the current jobs) without knowing the future inputs (jobs).

k-server problem:

The k-server problem is a general model for online problems. In this problem we have a metric space and k servers located at k points in the metric space. The input is a sequence of requests which are points in the metric space. If a request comes at a point, we need to serve it by moving one of our servers to that point, if one is not already present. The quantity to be optimized is the total distance moved by the servers to serve all the requests.

Chapter 2

The k-Server Problem

In the last chapter we mentioned the k-server problem as an example of online problems. It is a very common example of online problems and has been studied extensively. In this chapter we present the current results on the k-server problem and discuss various modifications of the problem

2.1 Definition

The k-server problem is a general model for online problems. Many practical online problems can be modeled using the k-server problem, and hence all the results for the k-server problems apply to the practical problems. We define the k-server problem as follows.

Definition 2. *The input is a metric space* M, *a set of* k *servers located at points in* M, *and a sequence of requests* $\sigma_1, \sigma_2, \ldots$, *each of which is also a point in* M. *After each request, the problem is to move each server some distance, possibly zero, with the condition that after moving the servers, there is a server at the request point. The cost to serve the request is the distance moved by all the*

servers. The goal is to minimize the total cost to serve all the requests in the request sequence.

At any time, the location of the k servers is called the current *configuration.* The set of configurations is denoted by Q. So for each request, the problem is to decide the next configuration such that the request point is in the configuration. The server moved to the request point is the server that *serves* the request. The metric space is generally assumed to have a non-negative symmetric distance function satisfying the triangle inequality. The optimal solution to the offline version of the k-server problem can be computed in polynomial time using dynamic programming.

2.2 Current results

In this section we discuss some of the current results on the k-server problem. The k-server problem was introduced by Manasse, McGeoch, and Sleator [10, 11].

They proved that no algorithm for the k-server problem is better than k-competitive for a metric space with at least $k + 1$ points, and conjectured that there is an algorithm which is k-competitive for any metric space. The problem of whether such a k-competitive algorithm exists is still open.

The first competitive algorithm for the k-server problem for all metric spaces was given by Fiat, Rabani, and Ravid [7]. But their competitive ratio was exponential in k. The ratio was later improved by others. In Chrobak and Larmore [4] the authors introduced the Work Function Algorithm (WFA). But they could only show that the WFA was 2-competitive for the 2-server problem, and they conjectured that it was actually k-competitive for the k-server problem.

In Koutsoupias and Papadimitriou [9], the authors showed that the WFA was $(2k - 1)$-competitive. It is believed that the WFA is in fact k-competitive.

2.2.1 Results for specific k

For $k = 2$, a 2-competitive algorithm was given by Manasse, McGeoch, and Sleator [10]. After them many others have given 2-competitive algorithms. In general these algorithms have time and space complexity in the order of the size of the metric space. A 10-competitive constant time and space algorithm, BALANCE2, was given by Irani and Rubinfeld [8]. In Chrobak and Larmore [5] the authors gave a 4-competitive algorithm with constant time and space complexity. For $k = |M| - 1$, Manasse, McGeoch, and Sleator [11] gave a k-competitive algorithm. For $k = |M| - 2$, Bartal and Koutsoupias [1] showed that the WFA is k-competitive.

2.2.2 Results for specific metric spaces

For a Line, a memoryless k-competitive algorithm, DOUBLE-COVERAGE, was given by Chrobak *et. al.* [3]. Bartal and Koutsoupias [1] also showed that the WFA is k-competitive for the line. For a Tree metric space, Chrobak and Larmore [6] gave a k-competitive memoryless algorithm.

If the metric space is a uniform metric space (the distance between any two points is 1), we get the problem of Paging. The points in the metric space are virtual pages, and the k servers are main memory pages. If a server is at a point, this means that the corresponding virtual page is in the corresponding main memory page. We can also consider the metric space as a complete graph. We know there are k-competitive algorithms for Paging.

2.3 Variations of the k-Server Problem

In the simple k-server problem, all the k servers are equal and the metric space is symmetric. We can consider some generalizations of the simple problem. These

generalized problems can model many practical online problems which the simple problem cannot.

We can have different variations by changing a property of the metric space or by changing the equality of the servers or both. Very little work has been done in this direction, and the work done is mostly with respect to the paging (caching) problem (k-server problem with uniform metric space).

In Chrobak *et. al.* [3] the authors give a solution for the weighted-cache problem, where the cost of moving a server to a point depends only on that point. This leads to an asymmetric metric space. The restricted caching problem in Brehob *et. al.* [2] leads to the problem where specific servers can only move to specific points in the uniform metric space. Other variations for the cache problem where each server has a different serving cost have been considered, like in Epstein, Imreh, and Stee [13].

For this thesis we consider the following generalization: All servers can move to every point in the metric space, but each server can serve only certain types of requests. That is we have a fixed number of request types, and each request type can only be served by some subset of the k servers. The metric space has symmetric non-negative weights satisfying the triangle inequality. We call it the *restricted* k-*server problem* (the servers are restricted as to which requests they can serve).

The most general problem would be one where we have k servers and 2^k types of requests. Each type of request can be served by all servers in one of the 2^k subsets of the k servers.

2.3.1 Our problem

In this thesis we consider the problem where we have two types of servers and two types of requests. The type of servers and requests are as follows:

- There are two types of servers, type A and type B. We call the type A server the A server and the type B server the B server.
- There are two request types, type 0 and type 1.
- A servers can serve requests of both types, B servers can serve requests of only type 1.

In most cases we consider the problem with just two servers defined below on different metric spaces.

Definition 3. *The restricted 2-server problem on metric space* M *is the server problem on* M *with one* A *server and one* B *server.*

In some sections, we generalize the results to more than two servers.

Some general conventions are given below:

- We use uppercase letter words as names of algorithms. OPT is the optimal algorithm, it solves the offline version of the problem optimally. ADV denotes the adversary against the online algorithm. We assume an adaptive offline adversary, meaning that it generates its next input after seeing the online algorithm's current output, and also it produces its output optimally offline. ON is a general name for any online algorithm.
- We use uppercase A and B for an online algorithm's A servers and B servers, and lowercase a and b for optimal or adversary's A servers and B servers.
- We assume that all servers start at the same location.

Chapter 3

Results

In this chapter we present our results for the restricted k-server problem. First we consider a general metric space, and then we consider two specific metric spaces.

3.1 Any Metric Space

In this section we consider the restricted 2-server problem on a general metric space. We look at the BALANCE2 (BAL2) and Work Function (WFA) algorithms.

3.1.1 BAL2

Here we first prove a partial result for any algorithm, and then we prove a stronger partial result for BAL2.

BAL2 was introduced by Irani and Rubinfeld [8]. It is 10-competitive for the simple 2-server problem and uses constant time and space. The way it works is, if X and Y are the two servers, when a request comes in, it compares $C_X + 2x$ with $C_Y + 2y$. Here C_X is the total cost incurred by server X before serving the current request and x is the distance of server X from the current request. Similarly for server Y. If the first quantity is smaller, then it uses X to serve the request,

otherwise it uses Y. For our restricted problem we only use the cost incurred by the A server to serve requests of type 1 in making this comparison.

Our basic approach is to split the costs to serve the requests of type 0 and type 1. C is the total cost. C0 is the cost to serve type 0 requests and C1 is the cost to serve type 1 requests. $C = C0 + C1$.

Notations:

- Cx_z^y denotes the cost to serve requests of type x (the total cost if x is absent) using the type y server for algorithm z, where x can be 0, 1 or empty, y can be A or B, and z can be OPT, ON or the name of some online algorithm.
- $C[i, j]$ denotes the cost from the i^{th} request to the j^{th} request. $C[j] = C[1, j]$. If there is no index, then it is the total cost. (x, y or z can be added as described in last point).
- A_z denotes the A server of algorithm z. Similarly B_z.
- A phase is a maximal contiguous sequence of requests of one type. We can have a 1-phase or a 0-phase. A phase can be denoted as $[i, j]$, which means requests i to j are of one type and requests $i - 1$ and $j + 1$ are of the other type.
- A cycle is a 1-phase followed by a 0-phase. Cycle $[i, j, k]$ means $[i, j - 1]$ is a 1-phase and $[j, k]$ is a 0-phase.
- The difference between the costs incurred by BAL2's two servers to service type 1 requests after the i^{th} request is denoted by Γ_i, i.e. $\Gamma_i = |C1^A_{BAL2}[i] - C1^B_{BAL2}[i]|$. Γ is the value after the last request.

Then we have the following.

Lemma 1. *For any online algorithm,* ON, *for any cycle* $[i, j, k]$

$$C0_{ON}[i, k] \leq C_{OPT}[i, k] + C1^{A}_{ON}[i, k] \tag{3.1}$$

Proof. Let ΔA be the distance between the two A servers before request j. Since the A servers are aligned before request i we have

$$\Delta A \leq C_{OPT}[i, j-1] + C1^{A}_{ON}[i, j-1]$$

And we have

$$\begin{aligned} C0_{ON}[i, k] &= C0_{ON}[j, k] \\ &\leq C_{OPT}[j, k] + \Delta A \\ &\leq C_{OPT}[j, k] + C_{OPT}[i, j-1] + C1^{A}_{ON}[i, j-1] \\ &\leq C_{OPT}[i, k] + C1^{A}_{ON}[i, k] \end{aligned}$$

□

Theorem 1. *For any online algorithm,* ON

$$C0_{ON} \leq C_{OPT} + C1_{ON}$$

Proof. Summing equation 3.1 over all cycles we get

$$C0_{ON} \leq C_{OPT} + C1^{A}_{ON}$$

Hence

$$C0_{ON} \leq C_{OPT} + C1_{ON} \tag{3.2}$$

□

Corollary 1. *Any online algorithm is* $50/\epsilon$*-competitive for request sequences for which the cost incurred to serve the type* 1 *requests is less than* $(50-\epsilon)\%$ *of the total cost, where* ϵ *is any positive real number.*

Proof. Given the condition of the request sequence we have

$$C1_{ON} \leq \frac{50-\epsilon}{100} C_{ON} \quad \text{and} \quad C0_{ON} \geq \frac{50+\epsilon}{100} C_{ON}$$

Substituting these values in equation 3.2 we get

$$\begin{aligned} \frac{50+\epsilon}{100} C_{ON} &\leq C_{OPT} + \frac{50-\epsilon}{100} C_{ON} \\ \frac{2\epsilon}{100} C_{ON} &\leq C_{OPT} \\ C_{ON} &\leq \frac{50}{\epsilon} C_{OPT} \end{aligned}$$

□

Next we consider BAL2.

Lemma 2. *Let* δ_i *be the distance between* A_{BAL2} *and* B_{BAL2} *after serving request* i. *Let* $\Delta_i = \max_{j \leq i} \delta_j$. *Then, after serving the* i^{th} *request* $(i \geq 1)$

$$\Gamma_i < 2\Delta_i$$

Proof. We prove this by induction on i. Initially both LHS and RHS are 0. After the first request $\Gamma_1 \leq \Delta_1$, hence $\Gamma_1 < 2\Delta_1$ (assuming the first request is not at the starting position of the two servers).

Now assume it is true for all values less than i. Consider the i^{th} request. If it is a type 0 request, then the LHS does not change and the RHS can only increase. Since $\Gamma_{i-1} < 2\Delta_{i-1}$ we have $\Gamma_i < 2\Delta_i$.

If it is a type 1 request then we look at the server which served the request. Let x be the distance (before servicing the request) between the request and the server

which served the request and let y be the distance between the request and the other server. Then we have two cases

Case 1 The leading server served the request. Then we must have

$$\Gamma_{i-1} + 2x \leq 2y$$

$$\Gamma_i + x \leq 2\Delta_i$$

$$\Gamma_i < 2\Delta_i$$

Case 2 The lagging server served the request. Then we must have

$$\Gamma_{i-1} + 2y \geq 2x \tag{3.3}$$

The lagging server moves by x. If $x \leq \Gamma_{i-1}$ then $\Gamma_i < \Gamma_{i-1} < 2\Delta_{i-1} \leq 2\Delta_i$. If $x > \Gamma_{i-1}$ then

$$\Gamma_i \leq x - \Gamma_{i-1}$$

$$\Gamma_i < 2x - \Gamma_{i-1}$$

Using equation 3.3 we get

$$\Gamma_i < 2y$$

$$\Gamma_i < 2\Delta_i$$

□

Since Δ_i is bounded above by $C_{OPT}[i]$ we have the following corollary.

Corollary 2. *After any request* i

$$\Gamma_i \leq 2C_{OPT}[i]$$

Theorem 2.

$$C0_{BAL2} \leq 2C_{OPT} + \frac{1}{2}\, C1_{BAL2}$$

Proof. Summing equation 3.1 (substituting ON with BAL2) over all cycles, we get

$$C0_{BAL2} \leq C_{OPT} + C1^{A}_{BAL2}$$

We have

$$C1^{A}_{BAL2} \leq \frac{1}{2}C1_{BAL2} + \frac{1}{2}\Gamma$$

Using Corollary 2 we have

$$C1^{A}_{BAL2} \leq \frac{1}{2}C1_{BAL2} + C_{OPT}$$

So we get

$$C0_{BAL2} \leq 2C_{OPT} + \frac{1}{2}C1_{BAL2} \tag{3.4}$$

□

Corollary 3. BAL2 *is* $400/3\epsilon$*-competitive for request sequences for which the cost incurred to serve the type* 1 *requests is less than or equal to* $(\frac{200}{3} - \epsilon)\%$ *of the total cost, where* ϵ *is any positive real number.*

Proof. Given the condition of the request sequence we have

$$C1_{BAL2} \leq \frac{\frac{200}{3} - \epsilon}{100}C_{BAL2} \quad \text{and} \quad C0_{BAL2} \geq \frac{\frac{100}{3} + \epsilon}{100}C_{BAL2}$$

Substituting these values in equation 3.4 we get

$$\begin{aligned} \frac{\frac{100}{3} + \epsilon}{100}C_{BAL2} &\leq 2C_{OPT} + \frac{1}{2}\,\frac{\frac{200}{3} - \epsilon}{100}C_{BAL2} \\ \frac{3\epsilon}{200}C_{BAL2} &\leq 2C_{OPT} \\ C_{BAL2} &\leq \frac{400}{3\epsilon}C_{OPT} \end{aligned}$$

□

We conjecture that the cost of serving type 1 requests, $C1_{BAL2}$, can be bounded as follows

$$C1_{BAL2} \leq mC_{OPT} + (2 - \delta)C0_{BAL2}$$

where m is a positive integer and δ is a positive real number > 0. This bound can then be combined with Theorem 2 to get a bound for C_{BAL2}.

3.1.2 Work Function Algorithm

Here we consider the *WFA* for the restricted problem. The *WFA* was shown to be $(2k - 1)$-competitive in Koutsoupias and Papadimitriou [9]. We show that an important property of the simple k-server problem, *quasiconvexity* does not hold for the restricted case.

The central idea in the *WFA* algorithm is the work function, $w_t(X)$, which is the optimal cost of servicing the first t requests and ending up with the servers in configuration X. The quasiconvexity property is as follows:

$$\forall t, \forall X, Y, \forall x \in X, \exists y \in Y : w_t(X) + w_t(Y) \geq w_t(X - x + y) + w_t(Y - y + x).$$

What the property says is that given any two configurations, for any point in the first configuration there is some point in the second configuration such that if we swap the two points, the total work function does not increase.

Quasiconvexity does not hold for the restricted problem as seen by the example in Figure 3.1. For the counterexample we consider the line metric space. Here all servers start at the same location, p, and the request sequence consists of a single request of type 1 at s.

The two figures show the configurations X and Y before and after the swap of the A servers. As we can see the sum of work function increases from $3x + 2y + 3z$ to $3x + 4y + 3z$ because of the swap.

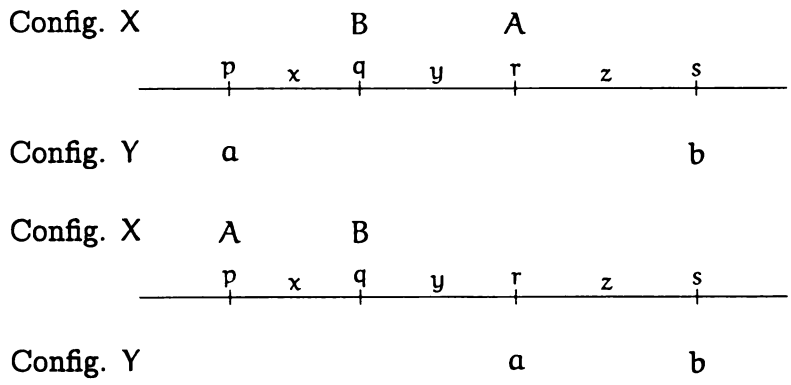

Figure 3.1: Quasiconvexity Counterexample. Here p, q, r and s are locations. x, y and z are distances between pq, qr and rs respectively. All servers start at p and there is a single request of type 1 at s.

Quasiconvexity is critical for the proof in Koutsoupias and Papadimitriou [9] to work. Although this proof won't work, we conjecture that the *WFA* is competitive for the restricted case, but not k-competitive.

3.2 Line Metric Space

In this section we consider the restricted 2-server problem on a line metric space. We consider the double coverage approach for it. In Chrobak *et. al.* [3], the authors have given the memoryless algorithm DOUBLE-COVERAGE, which is k-competitive for the k-server problem. A memoryless algorithm is one whose move depends only on the current configuration and the current request; it does not look at the past requests. A memoryless algorithm can be described by a function $f : Q \times \mathbb{R} \rightarrow Q$, where Q is the set of configurations. Memoryless algorithms are appealing because they are easy to implement.

DOUBLE-COVERAGE behaves as follows: it moves the servers immediately on the left and right of the request towards the request by an amount equal to the distance between the closer server and the request. The closer server then serves

the request. If there is no server on one of the sides, then DOUBLE-COVERAGE simply moves the closest server on the other side to the request.

The algorithm tries to 'cover' the request from both sides and hence the name. Since we have two types of requests in the restricted case we need to decide the double covering strategy for each type of request. The simplest approach is to do usual double coverage for type 1 requests and only move the A server for type 0 requests. We show below that this strategy is not competitive.

Lemma 3. *The simple double coverage strategy described above is not* c*-competitive for the restricted* k*-server problem for any constant* c.

Proof. We show this by example.

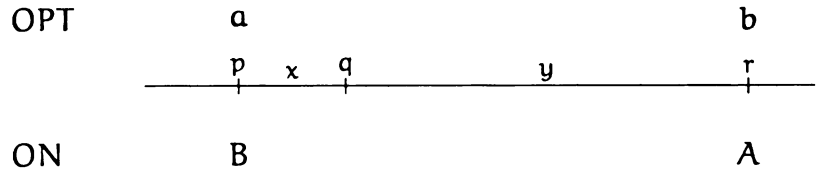

Figure 3.2: DOUBLE-COVERAGE Counterexample

Here p, q and r are locations on the line, x is the distance between p and q and y is distance between q and r. The initial positions of the servers for OPT and ON are given. Since all servers start at the same location (say at p), we can achieve this configuration by requesting a type 1 request at r. OPT moves the opposite type of the server moved by ON.

We first request a 0 at q. Both OPT and ON move their A server to q. Then we request a 1 at r. OPT doesn't have to move, ON moves the A server to r. Last we request a 1 at p, OPT moves the A server back to p, ON doesn't have to move. We end up in the original configuration.

By repeating the above requests a large number of times, the costs for the initial requests needed to reach the above configuration can be neglected. The competitive ratio will be the ratio for one cycle, which is y/x. By properly choosing x and y we can asymptotically approach any ratio we want, and hence simple double coverage is not competitive for any constant c. □

Here we consider the problem with just 2 servers, but the above proof easily extends to more than 2 servers with at least one A server. We believe a strategy which never moves the B server for requests of type 0 will not be constant competitive.

We give an algorithm, called MDC (for Modified Double Coverage), for the restricted 2-server problem and show that it is competitive. The algorithm works as follows: For an outside (not between A and B) request of type 0 on the B server side, MDC does partial double coverage— it moves the A server to the request point, but moves the B server only till the original position of the A server. For all other outside requests it just moves the nearer server to serve it. For an inside request MDC does complete double coverage.

MDC does simple double coverage for type 1 requests. It crosses servers for type 0 requests that are nearer to the B server than the A server. The intuition behind crossing servers is as follows: If MDC has to move the A server from position p to position q to serve a type 0 request, it needs to cover position p with another server, because the adversary might might have a server at p and use it to call MDC's A server back. And since the A server can serve type 1 requests, the adversary could use a type 1 request to bring MDC's A server back; thus moving the B server nearer to p may be helpful. We show that MDC is 6-competitive for the restricted 2-server problem on the line.

Theorem 3. MDC *is 6-competitive for the restricted 2-server problem on the line.*

Proof. The proof technique is similar to the proof technique used to show DOUBLE-COVERAGE is competitive in [3]. We show this by finding a nonnegative potential function, ϕ, such that for any request:

1. when ADV serves it, ϕ does not increase by more than 6 times the distance moved by ADV, and
2. when MDC serves it, ϕ decreases by at least the distance moved by MDC.

If the COST_{MDC} and COST_{ADV} are the total costs of MDC and ADV, then after serving all the requests, the maximum increase in ϕ is $6\text{COST}_{\text{ADV}}$, and the minimum decrease in ϕ is COST_{MDC}. Let ϕ_0 be the initial value and ϕ_n be the final value of ϕ. Since ϕ is nonnegative, we have

$$0 \leq \phi_n = \phi_0 + \text{increase in } \phi - \text{decrease in } \phi$$

$$0 \leq \phi_0 + 6\text{COST}_{\text{ADV}} - \text{COST}_{\text{MDC}}$$

$$\text{COST}_{\text{MDC}} \leq 6\text{COST}_{\text{ADV}} + \phi_0$$

This shows that MDC is 6-competitive. We give the potential function ϕ below and show that it has the above properties.

We define the following distances: LL is the distance between the left (server on the left side on the line) servers of MDC and ADV. Similarly RR is the distance between the right servers of MDC and ADV. Aa is the distance between server A (MDC's A server) and server a (ADV's A server). Similarly we define Ab, Bb and AB. Let $\phi_1 = \text{LL} + \text{RR}$, $\phi_2 = \text{Aa} + \min\{\text{Ab}, \text{Bb}\}$ and $\phi_3 = \text{AB}$. Then we use the

following potential function

$$\phi = 4\phi_1 + 2\phi_2 + \phi_3$$

Let d be the distance ADV moves its server to serve the request (assuming a lazy adversary). Clearly ϕ_1 increases by at most d. Depending on which server ADV moves, only one of the two terms of ϕ_2 can change and increase by at most d. ϕ_3 does not change. So ϕ can increase by at most $4d + 2d = 6d$. Hence ϕ has property (1).

Then when MDC moves to serve, we have the following cases. Without loss of generality we assume that A is the left server and B is the right server.

Case 1 Simple outside request. This includes any request to the left of A and type 1 request to the right of B. The algorithm moves the closer server to serve the request.

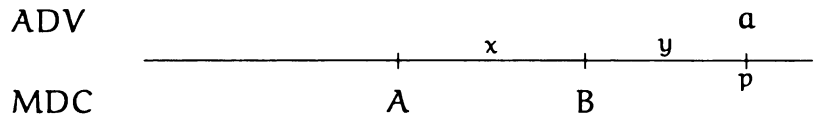

Figure 3.3: One simple outside request. There is a type 1 request at position p and ADV uses a to serve it. MDC moves B to p to serve the request. x is the distance between the original positions of A and B. y is the distance B moves.

Figure 3.3 shows one example, but we consider all simple outside requests. No matter where the other ADV server is, one term in ϕ_1 does not change and the other decreases by y. So ϕ_1 decreases by y.

If the request is to the left of A, then ϕ_2 cannot increase, because, depending on which server ADV uses, one of the terms in ϕ_2 decreases by y, and the other cannot increase by more than y.

If the request is to the right of B then ϕ_2 increases by at most y, since the first term does not change and the second can increase by at most y.

ϕ_3 always increases by y. So we have

$$\text{decrease in } \phi \geq 4y - 2y - y \geq y$$

Case 2 Non-crossing inside request. This includes any inside request nearer to A and type 1 request nearer to B. The algorithm does the double coverage in all cases and there is no crossing of servers.

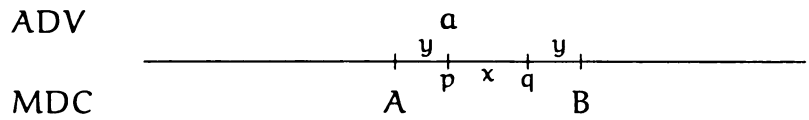

Figure 3.4: One non-crossing inside request. There is a type 1 request at position p and ADV uses a to serve it. MDC moves A to p to serve the request, and moves B to position q to double cover. x is the distance between the original positions of A and B. y is the distance A and B move.

Again, figure 3.4 shows one example but we consider all non-crossing inside requests. While one of the terms in ϕ_1 may increase by y, one of the terms in ϕ_1 decreases by y, so ϕ_1 cannot increase. Similarly, while one of the terms in ϕ_2 might increase by y, one of the terms in ϕ_2 decreases by y, so ϕ_2 cannot increase. ϕ_3 decreases by $2y$. So we have

$$\text{decrease in } \phi \geq 0 + 0 + 2y \geq 2y$$

Case 3 Crossing inside request. This is an inside request of type 0 nearer to B.

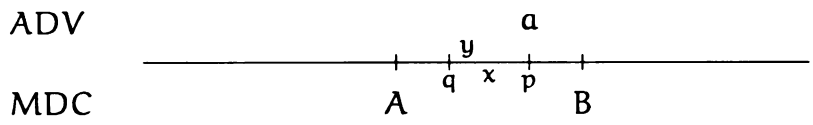

Figure 3.5: Crossing inside request. There is a type 0 request at position p and ADV uses a to serve it. MDC moves A to p to serve the request, and moves B to position q to double cover. x is the distance between the original positions of A and B. y is the distance A and B move.

If b is on the left of a, then LL cannot increase by more than $(x-y)$, and RR decreases by $(x-y)$. If b is on the right of a, then RR cannot increase by more than $(x-y)$ and LL decreases by $(x-y)$. So ϕ_1 cannot increase.

Aa decreases by y. Since the final position of B is $(x-y)$ away from original position of A, and vice versa, the min term cannot increase by more than $(x-y)$. So ϕ_2 decreases by at least $y-(x-y)=2y-x$.

ϕ_3 decreases by $2(x-y)$. So we have

$$\text{decrease in } \phi \geq 0+2(2y-x)+2(x-y) \geq 2y$$

Case 4 Outside request to 0 on B's side. In this case MDC does double coverage, but stops B at the original position of A.

Since B's final position is the same as A's original position, LL never changes. RR always decreases by y, no matter where server b is. So ϕ_1 decreases by y.

Aa always decreases by $(x+y)$. The min term cannot increase by more than y, since B takes A's place, and A's final position is distance y from B's original position. So ϕ_2 decreases by at least $(x+y-y)=x$.

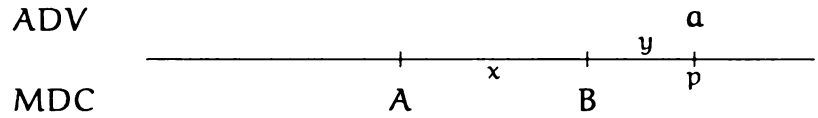

Figure 3.6: Outside request to 0 on B's side. There is a type 0 request at position p and ADV uses a to serve it. MDC moves A to p to serve the request, and moves B to the original position of A to double cover. x is the distance between the original positions of A and B. A moves a distance of $x + y$ and B moves a distance of x.

ϕ_3 increases by y. So we have

$$\text{decrease in } \phi \geq 4(y) + 2(x) - (y) \geq 2x + y$$

In each case the decrease in ϕ is at least equal to the distance moved by MDC. Hence ϕ also has property (2) □

3.2.1 More Than 2 servers

In this section we consider the restricted problem with more than 2 servers. The types of servers and requests are the same. We consider the problem with one A server and $k-1$ B servers for a total of k servers. We call this the *restricted* AB^{k-1}*-server problem.* We consider the restricted AB^{k-1}-server problem on the line.

This is a simple generalization of the restricted 2-server problem, and MDC works for this problem. The only difference in our analysis is for the case where we have B servers both between the A server and a type 0 request and also on the other side of the type 0 request. MDC works as follows:

It does the regular double cover for type 1 requests. For a type 0 request, let p be the position of the type 0 request and q be the position of the A server. We define *region1* as the region between p and q, and *region2* as the region

on the other side of p. MDC always moves the A server from q to p to serve the request. It moves other servers as follows:

- If there is no B server in both region1 and region2, MDC moves no other server.
- If there is at least one B server in region2 and no B server in region1, then MDC does the usual double cover, with possibly crossing servers.
- If there is at least one B server in region1 and no B server in region2, then MDC moves the B server in region1 that is closest to p, to q.
- If there is at least one server in both regions, then let S_i be the B server in region1 closest to p. Then S_{i+1} is the B server in region2 that is closest to p. Let x be the distance of S_i from q and y be the distance of S_i from p. Then MDC moves S_i to q, and then it moves S_{i+1} by a distance y towards p. Note here that the sum of the distances moved by the two B servers is $x+y$ which is equal (and in opposite direction) to the distance moved by the A server.

We show that MDC is $(2k+3)$-competitive.

Theorem 4. MDC *is* $(2k+3)$*-competitive for the restricted* AB^{k-1}*-server problem on the line.*

Proof. The proof is similar to the last proof; the main change is a slightly different potential function. Here the potential function must have the following properties:

1. when ADV serves a request, ϕ does not increase by more than $(2k+3)$ times the distance moved by ADV, and

2. when MDC serves a request, ϕ decreases by at least the distance moved by MDC.

As in the last proof, if such a nonnegative ϕ exists then MDC is $(2k+3)$-competitive. We give the ϕ below and show that it has the above properties.

Let S_i be MDC's servers and s_i be ADV's servers, both numbered from left to right. Let $\phi_1 = \sum_{i=1}^{k} |s_i - S_i|$, $\phi_2 = Aa$ and $\phi_3 = \sum_{i<j}(S_j - S_i)$. We use the following potential function

$$\phi = (2k+1)\phi_1 + 2\phi_2 + 2\phi_3$$

Let d be the distance ADV moves its server to serve the request (assuming a lazy adversary). Clearly ϕ_1 and ϕ_2 can both increase by at most d. ϕ_3 does not change. So ϕ can increase by at most $(2k+1)d + 2d = (2k+3)d$. Hence ϕ has property (1).

To prove property (2), we first consider the case when the request is of type 1. There are two possible cases. In the figures, we only show the MDC's servers which move.

Case 1.1 Outside request (all servers on one side of the request point). MDC moves the closest server to serve the request.

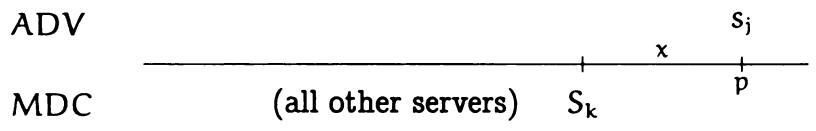

Figure 3.7: Outside request. There is a type 1 request at position p and ADV uses s_j to serve it. S_k is the rightmost server of MDC and S_k moves to p to serve the request. x is the distance S_k moves.

Since $j \leq k$, ϕ_1 decreases by x. ϕ_2 cannot increase by more than x. ϕ_3 always increases by $(k-1)x$. So we have

$$\text{decrease in } \phi \geq (2k+1)x - 2x - 2(k-1)x = x$$

Case 1.2 Inside request (request between two servers). MDC does double cover in all cases and there is no crossing of servers.

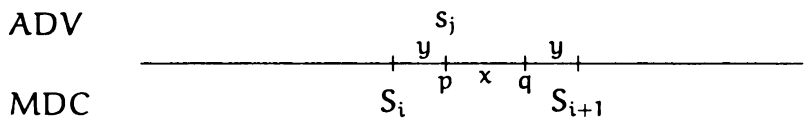

Figure 3.8: Inside request. There is a type 1 request at position p and ADV uses s_j to serve it. S_i is the MDC's server nearest to p and S_{i+1} is the nearest server on the other side of p. MDC moves S_i to p to serve the request, and moves S_{i+1} to position q to double cover. x is the distance between the original positions of S_i and S_{i+1}. y is the distance S_i and S_{i+1} move.

Depending on whether $j \leq i$ or $j > i$, one of the terms (either i^{th} or $(i+1)^{st}$) in ϕ_1 decreases by y. The other increases by at most y. So ϕ_1 cannot increase. ϕ_2 can increase at most by y. ϕ_3 decreases by 2y. So we have

$$\text{decrease in } \phi \geq 0 - 2y + 2(2y) \geq 2y$$

Now we consider the case when the request is of type 0. Depending on whether or not there are B servers in region1 and region2, we get the following four cases.

Case 2.1 No B servers in both regions. MDC just moves the A server to the request point.

This case is equivalent to Case 1.1.

Case 2.2 No B server in region1 and at least one B server in region2. Here MDC does double cover using the A server and the B server in region2 closest to the request point. Depending on whether or not the servers cross we get two cases.

Case 2.2.1 The servers do not cross.

This case is equivalent to Case 1.2.

Case 2.2.2 The servers cross.

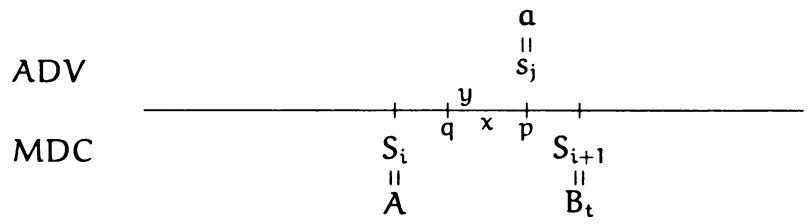

Figure 3.9: Only region1 empty, with servers crossing. There is a type 0 request at position p and ADV uses s_j to serve it. S_i is MDC's A server and S_{i+1} is the B server in region2 nearest to p. MDC moves S_i to p to serve the request, and moves S_{i+1} to position q to double cover. x is the distance between the original positions of S_i and S_{i+1}. y is the distance S_i and S_{i+1} move.

While one term, out of $|s_i - S_i|$ or $|s_{i+1} - S_{i+1}|$, in ϕ_1 might increase by $x - y$, the other term decreases by $(x - y)$. Thus ϕ_1 cannot increase. ϕ_2 decreases by y. ϕ_3 decreases by $2(x - y)$. So we have

$$\text{decrease in } \phi \geq 0 + 2y + 2(2(x - y)) \geq 2x \geq 2y$$

Case 2.3 No B server in region2 and at least one B server in region1. In this case, MDC moves the A server to the request point and the B server in region1 closest to the request point to the original position of the A server.

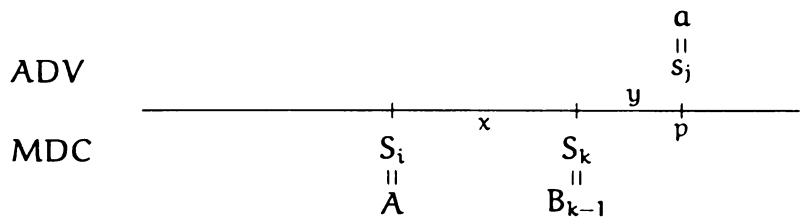

Figure 3.10: Only region2 empty. There is a type 0 request at position p and ADV uses s_j to serve it. S_i is MDC's A server and S_k is MDC's rightmost server. MDC moves S_i to p to serve the request, and moves S_k to the original position of S_i to double cover. x is the distance between the original positions of S_i and S_k. S_i moves by distance $x + y$ and S_k moves by distance x.

Here only the k^{th} term in ϕ_1 changes, and it decreases by y, so ϕ_1 decreases by y. ϕ_2 decreases by $(x + y)$. ϕ_3 increases by $(k - 1)y$. So we have

$$\text{decrease in } \phi \geq (2k + 1)(y) + 2(x + y) - 2(k - 1)y \geq 2x + y$$

Case 2.4 At least one B server in both regions. This case is shown in figure 3.11 below.

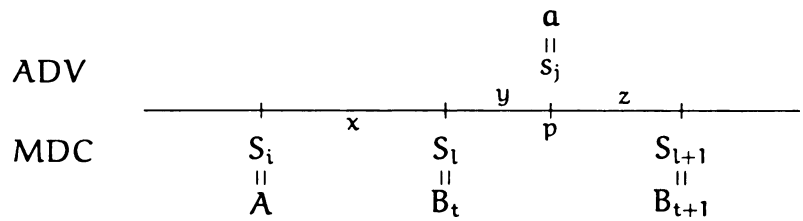

Figure 3.11: Neither region empty. There is a type 0 request at position p and ADV uses s_j to serve it. S_i is MDC's A server. S_l and S_{l+1} are MDC's B servers in region1 and region2, respectively, nearest to p. x is the distance between the original positions of S_i and S_l, y is the distance between S_l and p, and z is the distance between p and S_{l+1}. MDC moves S_i to p to serve the request. It moves S_l to the original position of S_i and moves S_{l+1} by a distance y towards p. S_i moves by distance $x + y$, S_l moves by distance x and S_{l+1} moves by a distance y.

If $z \geq y$, then S_i and S_{l+1} will not cross (S_i would be renamed as S_l). In this case, depending on whether $j \leq l$ or $j > l$, either the l^{th} term or the $(l + 1)^{st}$

term in ϕ_1 will decrease by y, and the other can increase by at most y. So ϕ_1 does not increase.

If $z < y$, then S_i and S_{l+1} will cross (S_i will be renamed as S_{l+1} and S_{l+1} will be renamed as S_l). In this case, depending on whether $j \leq l$ or $j > l$, either the l^{th} term or the $(l+1)^{st}$ term in ϕ_1 will decrease by z, and the other can only increase by at most z. So ϕ_1 does not increase.

ϕ_2 decreases by $(x+y)$.

If $z \geq y$, we can view the movement of MDC's servers as S_l and S_{l+1} moving towards each other by a distance y, so ϕ_3 decreases by $2y$. And if $z < y$, we can view the movement of MDC's servers as S_l and S_{l+1} moving towards each other by a distance z. So ϕ_3 decreases by $2z$. In either case, we can say that ϕ_3 does not increase.

So we have

$$\text{decrease in } \phi \geq 0 + 2(x+y) + 0 = 2(x+y)$$

In each case, the decrease in ϕ is at least equal to the distance moved by MDC. Hence ϕ also has property (2) □

The problem with more than one A server is much more complex than the problem with just one A server. MDC does not easily extend to the case with more than one A server. We conjecture that no memoryless algorithm is constant competitive for this case.

3.3 Uniform Metric Space

In this section we consider the restricted problem with the uniform metric space (or the complete graph). The cost to move a server from any point to any other

point is 1. We consider the problem with more than two servers. Let n be the number of points in the metric space, k be the number of A servers and l be the number of B servers. We assume that $k < l$. We call this problem the *restricted* A^kB^l*-server problem on* K_n, where K_n is the complete graph on n vertices.

For this problem we assume that requests are always generated at points where the online algorithm does not have a server which can serve the request. That is the online algorithm faults on all requests. We can easily trim (since it can only reduce the optimal cost) any request sequence to get this property. We consider the following two cases.

3.3.1 $k + l = n$

For the simple k-server problem, this case is trivial. However, for the restricted problem it is not trivial.

First we consider the case with just two servers, one A server and other B server, and two points, that is the restricted AB-server problem on K_2. We give an algorithm, USEB (for USE the B server whenever possible), which works as follows: it always uses the B server to serve a request of type 1. We prove that USEB is 3-competitive.

Theorem 5. USEB *is 3-competitive for the restricted* AB*-server problem on* K_2.

Proof. We cut the input sequence into phases as follows: A *new* location is a location where a request of a particular type has not been requested in the current phase. A request of type 0 at the 2^{nd} new location ends the current phase. The request ending the current phase is part of the next phase.

Because of the way the phases are defined, OPT must have one fault corresponding to each phase. A simple case analysis shows that the maximum length of a phase can be 3. This gives a competitive ratio of 3. □

Next we consider more than 2 servers, that is the restricted A^kB^l-server problem on K_{k+l}. We extend the phase definition as follows. Since there are k (instead of 1) A servers, a phase ends when a type 0 is requested at the $(k+1)^{st}$ new location. Again the ending request is not part of the current phase.

USEB still always uses B servers to serve requests of type 1. It uses a marking scheme to select which B server to use as follows:

USEB uses two types of marks, type 1-mark and type 0-mark. If a server serves a type x request it gets a type x-mark. When a request of type 0 comes in, USEB uses any one of the type 0-unmarked A servers. When a type 1 request comes in, it uses any one of the type 1-unmarked B servers. When USEB moves an A server to a location where there is already a B server, it unmarks that B server. At the end of a phase, USEB converts all 0 marks into 1 marks.

Theorem 6. USEB *is* 3k*-competitive for the restricted* A^kB^l*-server problem on* K_{k+l}.

Proof. As before, because of the way a phase is defined, OPT must have at least one fault corresponding to each phase. USEB faults on every request, so we find a limit on the number of requests in a phase.

From the way the algorithm works, we can never have a type 1 request when there is no unmarked B server. The 0 request, when there is no type 0 unmarked A server, ends the phase. Thus there are at most k type 0 requests in a phase. At the start of a phase there can be a maximum of k unmarked B servers (except for

the first phase) corresponding to the k A servers being co-located with k B servers. During a phase USEB can unmark a maximum of k B servers. So we can have maximum of k type 0 requests and 2k type 1 requests, which gives a competitive ratio of 3k. □

3.3.2 $k + l < n$

In this case there are two ways for a phase to end. A phase ends when either a 0 is requested to the $(k+1)^{st}$ new location or any request (0 or 1) is requested to the $(k+l+1)^{st}$ new location. Also, at the end of a phase, everything is unmarked.

For this case we give the algorithm, BTHENA (for first B THEN A), which is a natural extension of USEB. It works as follows: When a type 1 request comes in, the algorithm first uses any unmarked B servers, and if there are none, then it uses any unmarked A servers.

Theorem 7. BTHENA *is* $3k+l$*-competitive for the restricted* A^kB^l*-server problem on* K_n *(*$k+l<n$*).*

Proof. The proof is similar. A type 0 request when there is no 0-unmarked A server ends the phase, and a type 1 request when there is no unmarked server ends the phase. In the worst case there can be $k+l$ type 1 requests to get all servers 1-marked, k type 0 requests to get the A servers 0 marked (and potentially k B servers unmarked), and k more type 1 requests to get the B servers remarked. So there can be a maximum of $3k+l$ requests giving a competitive ratio of $3k+l$. □

Chapter 4

Conclusion

In this thesis we consider the k-server problem. We first described the k-server problem and then described the current work done in the field. We then discussed various generalizations of the normal k-server problem and we studied one generalization which we called the restricted k-server problem. In this generalization, we have different types of servers and types of requests. Certain types of servers can only serve certain types of requests. The particular problem we look at in the thesis has two types of of servers, A and B, and two types of requests 0 and 1. An A server can serve both types of requests, and a B server can only serve type 1 requests.

We looked at the restricted k-server problem with three metric spaces: the general metric space, the line metric space, and the uniform metric space. For the general metric space, we proved a partial result for the BALANCE2 algorithm. For the line metric space, we give a competitive algorithm which is a modified version of the DOUBLE-COVERAGE algorithm. We give competitive algorithms for the uniform metric space.

In the future we wish to extend the results of this thesis and prove the conjectures that we make. In particular we wish to complete the partial result for the BALANCE2 algorithm. We also think the work function algorithm is competitive and wish to prove so.

The generalization that we consider in this thesis is a very simple one. The problem with more complex relations between types of servers and types of requests is much more difficult, and a lot of work can be done in this area.

Bibliography

[1] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. In *Proceedings of 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 605-613, 2000.

[2] M. Brehob, R. Enbody, E. Torng, and S. Wagner. On-line restricted caching. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 374-383, 2001.

[3] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on the server problem. *SIAM Journal on Discrete Mathematics*, 4:172-181, 1991.

[4] M. Chrobak and L. Larmore. The server problem and on-line games. D. Sleator and L. McGeoch, editors, *On-line algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11-64. AMS, 1992.

[5] M. Chrobak and L. Larmore. On fast algorithms for two servers. *Journal of Algorithms*, 12:607-614, 1991.

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

[7] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proc. 31st Annual Symposium on Foundations of Computer Science*, pages 454-63 vol.2, 1990.

[8] S. Irani and R. Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39:85-91, 1991.

[9] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. In *Proc. of the 26th Annual ACM Symposium on Theory of Computing*, pages 507-511, 1994.

[10] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322-333, 1988.

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

[12] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202-208, 1985.

[13] L. Epstein, C. Imreh, and R. Stee. More on weighted servers or FIFO is better than LRU. *Theoretical Computer Science*, 306(1-3):305-317, 2003. Also in *Proc. of 27th MFCS (2002), LNCS 2420*, pages 257-268.