A Randomized Algorithm for Two Servers on the Line

Yair Bartal [*] Marek Chrobak [†] Lawrence L. Larmore [‡]

April 2, 1998

Abstract

In the k-server problem we wish to minimize, in an online fashion, the movement cost of k servers in response to a sequence of requests. For 2 servers, it is known that the optimal deterministic algorithm has competitive ratio 2, and it has been a long-standing open problem whether it is possible to improve this ratio using randomization. We give a positive answer to this problem when the underlying metric space is a real line, by providing a randomized online algorithm for this case with competitive ratio at most $\frac{155}{78} \approx 1.987$. This is the first algorithm for 2 servers that achieves a competitive ratio smaller than 2 in a non-uniform metric space with more than three points.

We consider a more general problem called the (k, l)-*server problem*, in which a request is served using l out of k available servers. We show that the randomized 2-server problem can be reduced to the *deterministic* $(2l, l)$-server problem. We prove a lower bound of 2 on the competitive ratio of the $(4, 2)$-server problem. This implies that one unbiased random bit is not sufficient to improve the ratio of 2 for the 2-server problem. Then we give a $\frac{155}{78}$-competitive algorithm for the $(6, 3)$-server problem on the real line. Our algorithm is simple and memoryless. The solution has been obtained using linear programming techniques that may have applications for other online problems.

[*]Bell-Labs, Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974. Email: `yairb@research.bell-labs.com`. This research was partially conducted while the author was at International Computer Science Institute, Berkeley.

[†]Department of Computer Science, University of California, Riverside, CA 92521. Email: `marek@cs.ucr.edu`. Research supported by NSF grant CCR-9503498. This research was partially conducted when the author was visiting International Computer Science Institute, Berkeley.

[‡]Department of Computer Science, University of Nevada, Las Vegas, NV 89154-4019. Email: `larmore@cs.unlv.edu`. Research supported by NSF grant CCR-9503441.

1 Introduction

In the *k-server problem* we are given k mobile servers that occupy a metric space M. A sequence of requests is issued, where each request is specified by a point $r \in M$. The request is satisfied by moving one of the servers to r at a cost equal to the distance from its current location to r. Our goal is to design an algorithm $\mathcal{A}$ that serves the requests at a small cost and satisfies the following *online property:* the decision of which server to choose is made without knowledge of future requests.

An online algorithm $\mathcal{A}$ is said to be *C-competitive* if the cost incurred by $\mathcal{A}$ to service each request sequence ϱ is at most C times the optimal service cost for ϱ, plus possibly an additive constant independent of ϱ. The *competitive ratio of* $\mathcal{A}$ is the smallest C for which $\mathcal{A}$ is C-competitive. The competitive ratio is commonly used as a performance measure of online algorithms for the k-server problem, as well as for other online problems.

Manasse, McGeoch and Sleator [16] introduce the k-server problem and prove that k is a lower bound on the competitive ratio of any deterministic online algorithm in any metric space with at least $k+1$ points. They also present an algorithm for the 2-server problem which is 2-competitive, and thus optimal, for any metric space. In the general case, for $k \geq 3$, the best currently published upper bound on the competitive ratio is $2k-1$, given by Koutsoupias and Papadimitriou [12, 13]. The *k-Server Conjecture* is that there exists a k-competitive algorithm that works in all metric spaces, but so far this conjecture has been proven only in a number of special cases, including trees and spaces with at most $k+2$ points [6, 7, 14].

For many online problems it is possible to improve the competitive ratios using randomization. Very little is known, however, about randomized algorithms for k servers. When the underlying space is uniform (all distances are 1) the competitive ratio is $H_k \approx \ln k$, the k-th harmonic number [1, 17]. Metric spaces with three points have been investigated by [11] and [15]. The results of Bartal *et al.* [3] imply that in a metric space with $k + o(\sqrt[6]{k}/\log k)$ points there is a randomized k-server algorithm whose competitive ratio is smaller than k. Except for these cases, no randomized online algorithm with competitive ratio better than k is known. This is true, in fact, even for $k=2$ and 4-point spaces.

Some lower bounds on the competitive ratios of randomized k-server algorithms are known. Blum *et al.* [4] give an asymptotic $\Omega(\sqrt{\log k/\log\log k})$ lower bound for an arbitrary space with at least $k+1$ points, and a $\Omega(\log k/\log\log k)$ lower bound for $k+1$ equally spaced points on the line. For two servers, the best known lower bound is $1 + e^{-1/2} \approx 1.6065$ [9].

The main contribution of this paper is a randomized online algorithm for two servers on the line whose competitive ratio is at most $\frac{155}{78} \approx 1.987$. This is the first algorithm for 2 servers that achieves a competitive ratio smaller than 2 in a metric space with unbounded distances, or, in fact, in a non-uniform metric space with more than three points. In addition to being a nontrivial version of the 2-server problem, the real-line case is of its own interest, since it can be seen as the problem of motion planning for 2-headed disks (see [5]).

Our method is to consider a more general problem called the *(k,l)-server problem*, in which a request must be served using l out of k available servers. We show that the randomized 2-server problem can be reduced to the *deterministic* $(2l,l)$-server problem. We examine first the case $l=2$. Quite unexpectedly, it turns out that no deterministic online algorithm can be better than 2-competitive for the $(4,2)$-server problem, even in metric spaces with only four points. This implies that each randomized algorithm for 2 servers defined by uniformly choosing between two deterministic algorithms has competitive ratio at least 2, and thus is no better than a deterministic algorithm. In other words, one unbiased random bit does not help. This result relates to recent work on *barely random* algorithms, that is, algorithms that use a constant number of random bits (see, for example, [2, 10]). The proof of this lower bound is given in Section 3.

In Section 4 we consider the next value of l, that is the $(6,3)$-server problem. We concentrate on the metric space consisting of points on the real line. For this case we present an algorithm called DC2 (Double Coverage, and "2" to distinguish it from the algorithm in [6]), whose competitive ratio is at most $\frac{155}{78} \approx 1.987$. Algorithm DC2 is very simple and memoryless. The competitive analysis has been achieved using linear programming techniques that may have applications for other online problems. Our analysis is nearly tight, since we can also show that the competitive ratio of DC2 is no better than $\frac{107}{54} \approx 1.981$.

2 Preliminaries

Let $k \geq l \geq 1$. In the *(k,l)-server problem* we are given k servers $s_1, \ldots, s_k$ that reside and move in a metric space M. (For simplicity, if no ambiguity arises, we also use s_i to denote the current location of the i^{th} server.) Initially all servers are on the same point. At each time step a request $r \in M$ is issued. In response to this request we must move l servers to r in order to "satisfy" the request. The k-server problem is a special case of the (k,l)-server problem where $l = 1$.

Formally, we define an *online (k,l)-server algorithm* to be a function $\mathcal{A}(\varrho)$ that, to a given request sequence $\varrho \in M^*$, assigns the k server locations after serving ϱ, with at least l of these locations being on the last request of ϱ. The sequence of configurations determined by $\mathcal{A}$ on a given request sequence ϱ is referred to as a *(k,l)-service* of ϱ, or simply a *service*, if k, l are understood from context. The cost of $\mathcal{A}$ on ϱ, $cost_{\mathcal{A}}(\varrho)$, is defined as the total distance traveled by its servers while serving ϱ.

Denote by $opt(\varrho)$ the optimal cost of serving the request sequence ϱ. Algorithm $\mathcal{A}$ is *C-competitive* if there is a constant b such that $cost_{\mathcal{A}}(\varrho) \leq C \cdot opt(\varrho) + b$ for every request sequence ϱ . The *competitive ratio* of $\mathcal{A}$ is defined to be the minimum C for which $\mathcal{A}$ is C-competitive.

A *randomized algorithm* can be defined as a probability distribution $\mathcal{B}$ on the set of all deterministic algorithms for a given problem. The definition of competitiveness extends naturally to randomized algorithms, by replacing $cost_{\mathcal{A}}(\varrho)$ by $cost_{\mathcal{B}}(\varrho)$, the *expected* cost of $\mathcal{B}$ on ϱ. We will say that $\mathcal{B}$ is *l-uniform* if it is defined by a uniform probability distribution on l deterministic algorithms.

Lemma 1 *Let $\mathcal{B}$ be a randomized l-uniform online algorithm for 2 servers. Then there is a deterministic online algorithm $\mathcal{A}$ for the $(2l,l)$-server problem such that $cost_{\mathcal{A}}(\varrho) = l \cdot cost_{\mathcal{B}}(\varrho)$ for each request sequence ϱ.*

Proof: Suppose that $\mathcal{B}$ is a uniform distribution on deterministic algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_l$. Then $\mathcal{A}$ runs all algorithms $\mathcal{A}_i$ simultaneously, using all $2l$ servers. Note that $\mathcal{A}$ is well-defined since each request will be covered by l servers, and $cost_{\mathcal{A}}(\varrho) = \sum_{i=1}^{l} cost_{\mathcal{A}_i}(\varrho) = l \cdot cost_{\mathcal{B}}(\varrho)$ for each ϱ. □

Lemma 2 *Let $\mathcal{A}$ be a deterministic online algorithm for the $(2l,l)$-server problem. Then there is an l-uniform randomized online algorithm $\mathcal{B}$ for 2 servers such that $cost_{\mathcal{B}}(\varrho) = (1/l) cost_{\mathcal{A}}(\varrho)$ for each request sequence ϱ.*

Proof: First, we claim that there is a $(2l,l)$-server algorithm $\mathcal{A}'$, whose cost on each request sequence is the same as $\mathcal{A}$'s, which also satisfies the following *l-way property*: for each request r and each $i = 1, \ldots, l$, either s_i or s_{l+i} is on r. In order to construct $\mathcal{A}'$, we simulate $\mathcal{A}$ as follows: Let p be the last request, r be the new request, and suppose that the l-way property holds for p. Without loss of generality, we can assume that p was served by $s_1, \ldots, s_l$. On request r, $\mathcal{A}$ chooses $m \leq l$ servers $s_{l+i_1}, \ldots, s_{l+i_m}$ to serve r. $\mathcal{A}$ also chooses $l - m$ servers among $s_1, \ldots, s_l$ to serve r. Since it does not matter which $l - m$ servers are moved from p, we can move the $l - m$ servers that are not in the set $\{s_{i_1}, \ldots, s_{i_m}\}$. By repeating this process at each request, we obtain the desired algorithm $\mathcal{A}'$.

In the second step of the proof we transform $\mathcal{A}'$ into $\mathcal{B}$. Let $\mathcal{A}'_i$ be a deterministic 2-server algorithm that uses $\mathcal{A}'$ to determine the movement of its two servers s_i and s_{l+i}. Each $\mathcal{A}'_i$ is a well-defined 2-server algorithm, because of the l-way property. Then we define $\mathcal{B}$ to be the uniform distribution on $\mathcal{A}'_1, \ldots, \mathcal{A}'_l$. The cost of $\mathcal{B}$ on each request sequence ϱ is $cost_{\mathcal{B}}(\varrho) = (1/l)\sum_{i=1}^{l} cost_{\mathcal{A}'_i}(\varrho) = (1/l)cost_{\mathcal{A}'}(\varrho) = (1/l)cost_{\mathcal{A}}(\varrho)$. □

Corollary 1 *For each ϱ there is an optimal $(2l, l)$-service in which each request is served either by servers $s_1, \ldots, s_l$ or by $s_{l+1}, \ldots, s_{2l}$.*

The above corollary follows directly from Lemmas 1 and 2, and the corresponding fact about randomized algorithms for 2 servers. It can be easily explained without reference to randomized algorithms: as in the proof of Lemma 2, ϱ has an optimal service $\mathcal{A}'$ with the l-way property. Since $cost_{\mathcal{A}'}(\varrho) = \sum_{i=1}^{l} cost_{\mathcal{A}'_i}(\varrho)$, there is an i for which $cost_{\mathcal{A}'}(\varrho) \geq l \cdot cost_{\mathcal{A}'_i}(\varrho)$. Instead of $\mathcal{A}'$, we can then use a service $\mathcal{A}''$ that simulates each server of $\mathcal{A}'_i$ with l servers moving together. Then $cost_{\mathcal{A}''}(\varrho) = l \cdot cost_{\mathcal{A}'_i}(\varrho) \leq cost_{\mathcal{A}'}(\varrho)$.

We summarize the results from this section in the following theorem.

Theorem 1 *The following two statements are equivalent:*

(a) There is a deterministic C-competitive online algorithm for the $(2l, l)$-server problem.

(b) There is a C-competitive randomized l-uniform online algorithm for the 2-server problem.

Throughout the paper, for the purpose of competitive analysis, we will view the problem as a game between our algorithm and the adversary who serves the requests with his own servers. Without loss of generality we can assume that the adversary serves each request optimally. Because of Corollary 1, we can also assume that the adversary has just two servers a_1, a_2, that he serves each request with one server, but we charge him the cost equal l times the distance traveled by his servers. At each step the adversary chooses a server a_i, moves a_i to a point r, announces that a request has been made on r, and then our algorithm serves the request with its servers. When proving an upper bound on the competitive ratio, our goal is to show that our algorithm's cost is no more than C times the adversary's cost, independent of the adversary strategy. To prove a lower bound, we must present an adversary strategy of arbitrarily large cost that forces our algorithm to pay no less than C times the adversary cost minus a constant.

In order to prove that a given algorithm $\mathcal{A}$ is C-competitive we will use a potential argument, which is to define a *potential function* Φ that maps server configurations (ours and the adversary's) into nonnegative real numbers, and satisfies the following inequality at each move:

$$\Delta cost + \Delta\Phi \quad \leq \quad C \cdot \Delta opt, \tag{1}$$

where $\Delta cost$, $\Delta\Phi$, and Δopt are, respectively, $\mathcal{A}$'s cost, the potential change, and the adversary cost in a given move. Inequality (1) implies the C-competitiveness of $\mathcal{A}$ by simple summation over the whole request sequence.

3 The $(4, 2)$-Server Problem

Theorem 2 *There is no online deterministic algorithm for the $(4, 2)$-server problem that is better than 2-competitive.*

Proof: Let N be a large integer and $\epsilon = 1/2N$. We use a 4-point metric space with points x, y, z, z', where the distances are: $xz = yz = 1$, $xy = 2$ and $zz' = \epsilon$. The distances xz' and yz' are in the range $1 - \epsilon \leq xz', yz' \leq 1 + \epsilon$, but their exact values are not relevant.

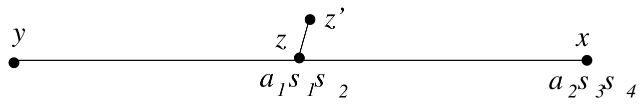

Figure 1: The space used in the lower bound proof and the initial configuration.

Let $\mathcal{A}$ be an online algorithm. The adversary strategy is divided into phases. We start with $s_1 = s_2 = a_1 = z$ and $s_3 = s_4 = a_2 = x$. (see Fig. 1). The adversary now alternates requests on z' and z, stopping when one of the following two events occurs, whichever happens first:

(A) $\mathcal{A}$ moves one server from x to z (or z'). Then the adversary requests x, completing the phase.

(B) The distance traveled by s_1 and s_2 to serve the requests on z and z' is 2, *i.e.*, the pair $z'z$ has been requested exactly N times. We now proceed as follows:

(1) The adversary requests y. Without loss of generality, $\mathcal{A}$ moves s_1 and s_2 to y, and s_3, s_4 stay on x.

(2) Now the adversary requests z. We consider three subcases:

(2.a) If $\mathcal{A}$ moves two servers from y then the adversary requests y, completing the phase.

(2.b) If $\mathcal{A}$ moves one server from x and one server from y then the adversary requests y, completing the phase.

(2.c) If $\mathcal{A}$ moves two servers from x then the adversary requests x. Now the adversary repeats Step (2) with x and y interchanged.

Note that the request sequence generated by the adversary up to this point has the form:

$$z'z\,z'z\,z'z \ldots z'z\,y\,z\,x\,z\,y\,z\,x\,z\,y\,z\,x\,z \ldots$$

If the last request was on x (respectively, y), the adversary alternates the request on x (resp. y) and z to make sure that $\mathcal{A}$ has two servers on x (resp. y) and two servers on z, before starting the next phase.

We now analyze the costs of $\mathcal{A}$ and the adversary, to show that, asymptotically, $\mathcal{A}$'s cost is at least twice the adversary cost. For clarity, we ignore low-order terms $O(\epsilon)$ in our analysis, and when we say that "the cost of ... is a" we mean that this cost is actually $a + O(\epsilon)$.

In Case (A), suppose the cost of serving the sequence $zz'zz' \ldots$ is d. Since Case (A) occurred, we have $d \leq 2$. The adversary cost is d, and $\mathcal{A}$'s cost is $2 + d \geq 2d$, so the ratio in this phase is at least 2.

To analyze Case (B), let $j \geq 1$ be the number of repetitions of Step (2). $\mathcal{A}$ pays 2 to serve $zz'zz' \ldots$, then it pays 4 for each repetition of Step (2), and then it pays either 4 or 2 (or more), depending on whether case (2.a) or (2.b) occurred. So the total cost of $\mathcal{A}$ is at least $4 + 4j$.

We claim that the adversary cost is $2j + 2$. Initially, the adversary has a_1 on z and a_2 on x. If j is odd, move a_2 to z to serve $zz'zz' \ldots$ at no cost, and then to y when it is requested. In the remaining $j - 1$ iterations use a_1 to serve the requests on z and x. Then the cost is $4 + 4(j - 1)/2 = 2j + 2$. If j is even, a_2 stays on x and all requests on z and y are served by a_1 at the cost of $2 + 4j/2 = 2j + 2$. In either case, the adversary returns to the initial configuration.

It is possible that a phase never ends. In this case $\mathcal{A}$ pays 4 during each iteration, and the adversary pays 4 every second iteration, thus the ratio approaches 2. □

Note that the lower bound proof uses a metric space with only four points, and that these points could be located on a line, or they can form a star (a metric space corresponding to the weighted-cache problem). Theorems 1 and 2 imply the following result.

Corollary 2 *There is no randomized* 2*-uniform online algorithm for* 2 *servers with competitive ratio better than* 2.

4 The $(6,3)$-Server Problem for the Line

4.1 Algorithm DC2.

We name our servers $s_1, ..., s_6$, ordered from left to right. Let the request point be r. We serve r in a sequence of as many as three "transitions," each of which moves one or more servers toward r. DC2 repeats this process until there are at least three servers at r.

Outward Transition: If $s_i > r$, for any $i = 1, 2, 3$, then move s_i to r. Similarly, if $s_i < r$, for any $i = 4, 5, 6$, then move s_i to r. We will sometimes refer to these transitions as *leftward* or *rightward*, depending on which of the two cases above occurred. (Clearly, only one of these cases can occur.)

Inward Transition: Suppose $s_3 \leq r \leq s_4$ and there are fewer than three servers on r. Each server has a *speed* assigned to it: s_1 and s_6 have speed 2, and the other four servers have speed 1. Pick largest i and smallest j such that $s_i < r < s_j$, and move s_i and s_j continuously towards r, at their assigned speeds, until one of them reaches r.

By definition, outward transitions will be executed first, if they apply. The algorithm is illustrated in Figure 2. The request on r is served in a sequence of three transitions: (i) first s_3 moves to r, then (ii) s_2 and s_4 move towards r with the same speed, and s_2 reaches r first, and then (iii) s_1 and s_4 move towards r, with s_2 moving twice as fast as s_4. Servers s_1 and s_4 arrive at r simultaneously.

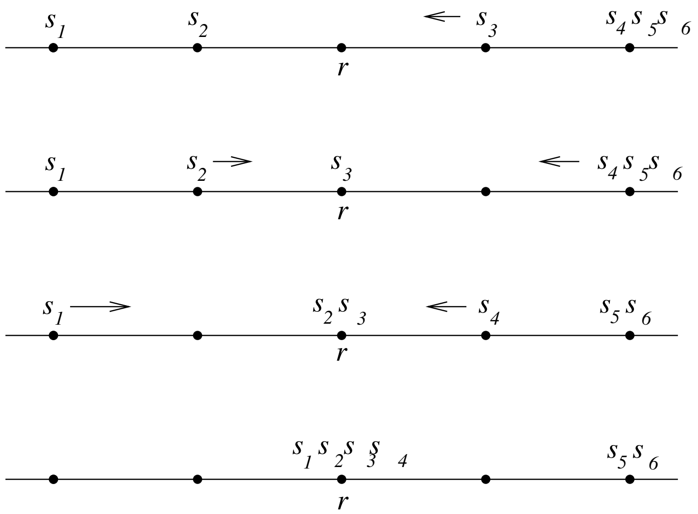

Figure 2: An example of DC2 serving a request.

4.2 The Linear Programming Approach

In this section we outline the method we used to estimate the competitive ratio of algorithm DC2 and to compute the potential function. The general idea is to assume that the potential function Φ can be expressed as a linear combination of the distances between the servers, with some unknown coefficients α_i. With each move we can associate an inequality (1), which now becomes a linear inequality involving the α_i and the competitive ratio C of DC2. This gives us a linear program whose objective function is to minimize C.

For better illustration, we restrict the movement of the adversary servers by assuming that he only moves his servers a_1 and a_2 leftward. We call this adversary *leftist*. We also assume that initially all servers are on the same point.

Without loss of generality we can assume that the adversary moves each server a_i only when he is going to request it. We will divide each step into *basic moves*. Each basic move involves either an adversary moving a server, or DC2 moving one or more servers, or possibly a combined move of the adversary and DC2. It has the important property that during a basic move, the set of servers that are in motion does not change.

Each move of a_1 (to the left) is a basic move. Any inward transition of DC2 is also a basic move. Given a leftward adversary, DC2 will not execute any rightward transitions. When the request is on a_1, using DC2, we execute the leftward transitions in up to three basic moves: (i) first move s_3, (ii) if s_3 reaches s_2 then we move s_2 and s_3 together, and (iii) if they both reach s_1, then all s_1, s_2 and s_3 move together. When the adversary intends to make a request with a_2 to the left of s_3, this adversary move and the resulting leftward transitions are replaced by the following basic "drag" moves: (i) when a_2 passes s_3 moving left, then s_3 moves to the left together with a_2, (ii) when a_2 passes s_2 moving left, then then s_2 and s_3 move to the left together with a_2, and (iii) when a_2 passes s_1 moving left, then s_1, s_2 and s_3 move to the left together with a_2. Further, we can also assume that a_2 never moves to the left of a_1, for whenever this happens, we can simply switch the labels of a_1 and a_2.

With these assumptions, all servers are always ordered $a_1, s_1, s_2, s_3, a_2, s_4, s_5, s_6$ from left to right. This gives us a partition of the interval $[a_1, s_6]$ into seven intervals between consecutive servers. Let x_i be the length of the ith interval. We assume now that the potential function is a linear combination of the x_i:

$$\Phi = \sum_{i=1}^{7} \alpha_i x_i \tag{2}$$

where the α_i are unknown non-negative coefficients.

Now we can set up our linear program. Let $\Delta cost$, Δopt and $\Delta\Phi$ denote our cost, the adversary cost, and the potential change during a basic move. Inequalities corresponding to the basic moves have the form $\Delta cost + \Delta\Phi \leq C \cdot \Delta opt$. For example, when s_3 and s_6 move, the corresponding inequality is $3 + \alpha_3 - \alpha_4 - 2\alpha_7 \leq 0$. When a_2 moves left, the corresponding inequality is $-\alpha_4 + \alpha_5 \leq 3C$, etc.

The objective is to minimize C. Unfortunately, the solution for this linear program is $C = 2$. Thus we cannot analyze DC2 accurately using a potential function in the form of (2).

The main idea leading to more accurate analysis is to notice that it is not necessary to use the same weight on a given interval at all times. We can allow weights to change according to the following rule: a weight of x_i can be decreased at any time, but it can increase only when $x_i = 0$. (The intuition is that the potential Φ represents the "savings" of our algorithm, so we can decrease its value whenever convenient, without invalidating the analysis.)

Divide the computation into *phases*, each phase starting when a_1 makes a request. In each given phase we distinguish two stages. Stage 1 starts when the phase starts. When s_3 and s_4 meet on a request for the first time in this phase, Stage 1 ends and Stage 2 begins, and it lasts till the end of the phase. If s_3 and s_4 do not meet, Stage 1 lasts throughout the phase and Stage 2 is empty. In Stage 1, x_2 and x_5 will have weights α_2 and α_5. In Stage 2, they will have weights α_2' and α_5', respectively. Since x_2 may be non-zero when Stage 1 ends, we need the inequality $\alpha_2 \geq \alpha_2'$. Similarly, x_5 will be generally non-zero when Stage 2 ends (and the next phase begins), so we also need $\alpha_5' \geq \alpha_5$.

	Basic Move	Linear Program		
		minimize		C
	$\leftarrow s_3$	$1 - \alpha_3 + \alpha_4$	$\leq$	0
(Stage 1)	$\leftarrow s_2 s_3$	$2 - \alpha_2 + \alpha_4$	$\leq$	0
(Stage 2)	$\leftarrow s_2 s_3$	$2 - \alpha_2' + \alpha_4$	$\leq$	0
	$\leftarrow s_1 s_2 s_3$	$3 - \alpha_1 + \alpha_4$	$\leq$	0
	$s_3 \rightarrow\leftarrow s_4$	$2 + \alpha_3 - \alpha_4 - \alpha_5 + \alpha_6$	$\leq$	0
	$s_3 \rightarrow\leftarrow s_5$	$2 + \alpha_3 - \alpha_4 - \alpha_6 + \alpha_7$	$\leq$	0
	$s_3 \rightarrow\leftarrow s_6$	$3 + \alpha_3 - \alpha_4 - 2\alpha_7$	$\leq$	0
(Stage 1)	$s_2 \rightarrow\leftarrow s_4$	$2 + \alpha_2 - \alpha_3 - \alpha_5 + \alpha_6$	$\leq$	0
(Stage 2)	$s_2 \rightarrow\leftarrow s_4$	$2 + \alpha_2' - \alpha_3 - \alpha_5' + \alpha_6$	$\leq$	0
	$s_2 \rightarrow\leftarrow s_5$	$2 + \alpha_2' - \alpha_3 - \alpha_6 + \alpha_7$	$\leq$	0
(Stage 1)	$s_1 \rightarrow\leftarrow s_4$	$3 + 2\alpha_1 - 2\alpha_2 - \alpha_5 + \alpha_6$	$\leq$	0
(Stage 2)	$s_1 \rightarrow\leftarrow s_4$	$3 + 2\alpha_1 - 2\alpha_2' - \alpha_5' + \alpha_6$	$\leq$	0
	$\leftarrow a_1$	α_1	$\leq$	$3C$
	$\leftarrow a_2$	$-\alpha_4 + \alpha_5$	$\leq$	$3C$
(Stage 1)	$\leftarrow s_3 a_2$	$1 - \alpha_3 + \alpha_5$	$\leq$	$3C$
(Stage 2)	$\leftarrow s_3 a_2$	$1 - \alpha_3 + \alpha_5'$	$\leq$	$3C$
(Stage 1)	$\leftarrow s_2 s_3 a_2$	$2 - \alpha_2 + \alpha_5$	$\leq$	$3C$
(Stage 2)	$\leftarrow s_2 s_3 a_2$	$2 - \alpha_2' + \alpha_5'$	$\leq$	$3C$
	$\leftarrow s_1 s_2 s_3 a_2$	$3 - \alpha_1 + \alpha_5$	$\leq$	$3C$
	weight change	α_2'	$\leq$	α_2
	weight change	α_5	$\leq$	α_5'
		$\alpha_1, \ldots, \alpha_7, \alpha_2', \alpha_5'$	$\geq$	0

Table 1: The linear program for the leftist adversary.

The right side of Table 1 shows the complete linear program. The explanation of each inequality is given in the left column. The notation for moves in Table 1 is self-explanatory. For example, "$s_3 \rightarrow\leftarrow s_5$" denotes the inward transition involving s_3 and s_5, and "$\leftarrow s_1 s_2$" denotes an outward basic move of s_1 and s_2.

The solution of the linear program in Table 1 is $C = \frac{155}{78}$, and the weights are given in the table below:

α_1	α_2	α_2'	α_3	α_4	α_5	α_5'	α_6	α_7
$\frac{155}{26}$	$\frac{149}{26}$	$\frac{145}{26}$	$\frac{111}{26}$	$\frac{77}{26}$	$\frac{232}{26}$	$\frac{240}{26}$	$\frac{142}{26}$	$\frac{56}{26}$

This solution constitutes a proof that DC2 is $\frac{155}{78}$-competitive against the leftist adversary. In the next section we will show how to extend this proof to the general case.

4.3 Competitive Analysis of DC2

We define now a potential function used in the proof of competitiveness. Recall that our servers are ordered $s_1, \ldots, s_6$, from left to right. Also, without loss of generality, we assume that a_1 is always to the left of a_2. It is convenient to express the potential as the sum of three quantities, Ψ, Γ and Λ. We define Ψ and Γ first:

$$\begin{aligned}\Psi &= \frac{1}{26}[\, 33|a_1 - s_1| + 60|a_1 - s_2| + 62|a_1 - s_3| + 33|a_2 - s_6| + 60|a_2 - s_5| + 62|a_2 - s_4|\,] \\ \Gamma &= \frac{1}{26}[\, 23(s_6 - s_1) + 26(s_5 - s_2) + 28(s_4 - s_3)\,]\end{aligned}$$

The formulas for Ψ and Γ are represented in Figure 3, which is a weighted graph whose vertices are the eight points a_i and s_j. Ψ is represented by dashed edges below the line, and Γ is represented by dashed edges above the line.

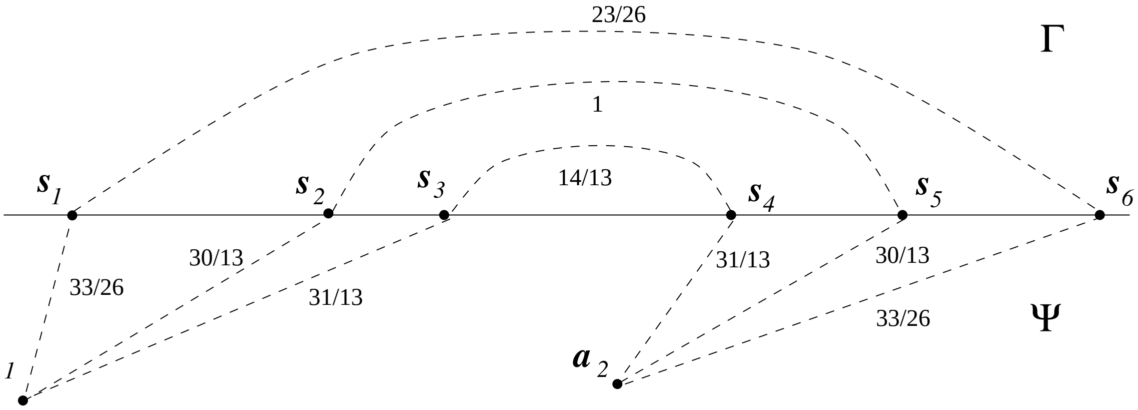

Figure 3: The representation of Ψ and Γ.

The relationship between $\Psi + \Gamma$ and the potential function in the previous section can be seen by comparing the weights of the intervals between the servers. For example, if a_2 is to the left of s_4 then the weight in $\Psi + \Gamma$ of the interval between s_4 and s_5 is $23/26 + 1 + 30/13 + 33/26 = 142/26$, the same as α_6 in the previous section. However, not all the weights are the same as before. We need to incorporate into the potential the idea of weights that can vary in different stages of the computation. Furthermore, since now we are not placing any restrictions on the adversary behavior, the servers may not necessarily be ordered as in the previous section. We solve these difficulties by introducing another, appropriately defined, term Λ. Let $[x]^+ = \max\{x, 0\}$. Then Λ is defined as follows:

$$\begin{aligned}\theta_1 &= \min\left\{ \begin{array}{c} [s_2 - a_1]^+ \\ s_2 - s_1 \end{array} \right\} \\ \theta_2 &= \min\left\{ \begin{array}{c} [a_2 - s_5]^+ \\ s_6 - s_5 \end{array} \right\} \\ \Lambda_1 &= \frac{2}{13}\min\left\{ \begin{array}{c} \theta_1 \\ 2(s_4 - s_3) + [\theta_1 - s_6 + s_5]^+ \end{array} \right\}\end{aligned}$$

$$\begin{aligned}
\Lambda_2 &= \frac{2}{13}\min\left\{\begin{array}{c}\theta_2 \\ 2(s_4-s_3)+[\theta_2-s_2+s_1]^+\end{array}\right\} \\
\Lambda &= \max\{\Lambda_1,\Lambda_2\}
\end{aligned}$$

Now we are ready to prove our main theorem.

Theorem 3 *Algorithm* DC2 *is* $\frac{155}{78}$*-competitive.*

Proof: We use the potential function

$$\Phi = \Psi + \Gamma + \Lambda,$$

where Ψ, Γ, Λ are defined above. Note that Φ is preserved under all symmetries of the line, *i.e.*, translation and inversion (flipping).

At each step the adversary moves one of its servers to the request point, announces the request, and then our servers serve the request. For the purpose of the analysis, we divide each service into at most three *basic moves*. Each adversary move is a basic move. Also, each inward transition is a basic move. The leftward transitions are replaced by a sequence of three basic moves (each of which could be empty): (i) moving s_3 first till it reaches s_2, (ii) moving s_2 and s_3 together till they reach s_1, and (iii) moving s_1, s_2 and s_3 together. The rightward transitions are replaced by rightward basic moves in a similar fashion. Note that in each basic move the server ordering and the set of servers that move do not change.

We view each basic move as a continuous process during which the servers move at their assigned speeds (all speeds are 1, except for s_1 and s_6, whose speeds are 2 during the inward transitions). By δ we denote the length of the time interval needed to execute a given basic move. Our goal is to show that $\Delta cost + \Delta\Phi \le \frac{155}{78}\Delta opt$.

Adversary Moves. By symmetry, we may assume that the adversary moves a_1. The adversary cost is $\Delta opt = 3\delta$. This move only affects Ψ and Λ_1. If $a_1 < s_1$ or $a_1 > s_2$ then $\Delta\Psi \le \frac{155}{26}\delta$ and $\Delta\Lambda_1 = 0$. If $s_1 < a_1 < s_2$ then $\Delta\Psi \le \frac{89}{26}\delta$ and $\Delta\Lambda_1 \le \frac{2}{13}\delta$. So $\Delta\Phi \le \frac{155}{78}\Delta opt$ in each case.

Outward Basic Moves. By symmetry, we only need to analyze leftward moves. We summarize the values of $\Delta cost$, $\Delta\Psi$, $\Delta\Gamma$ and $\Delta\Lambda$ in the table below. The numbers represent the worst-case values (that is, the upper bounds) of the corresponding quantities.

Move	$\Delta cost$	$\Delta\Psi$	$\Delta\Gamma$	$\Delta\Lambda$
$\leftarrow s_3$	δ	$-\frac{31}{13}\delta$	$\frac{14}{13}\delta$	$\frac{4}{13}\delta$
$\leftarrow s_2 s_3$	2δ	$-\frac{61}{13}\delta$	$\frac{27}{13}\delta$	$\frac{6}{13}\delta$
$\leftarrow s_1 s_2 s_3$	3δ	$-\frac{155}{26}\delta$	$\frac{77}{26}\delta$	0

The verification of the numbers in this table is quite straightforward. To illustrate, we explain the numbers in the last row, corresponding to the move when s_1, s_2 and s_3 move to the left. Since all three servers move, $\Delta cost = 3\delta$. Since a_1 is to the left of s_1, s_2, s_3, the value of each absolute value $|a_1 - s_1|$, $|a_1 - s_2|$ and $|a_1 - s_3|$ in Ψ decreases by δ, and thus $\Delta\Psi = -\frac{155}{26}\delta$. In Γ, all differences $s_6 - s_1$, $s_5 - s_2$ and $s_4 - s_3$ increase by δ, and thus $\Delta\Gamma = \frac{77}{26}\delta$. The argument for $\Delta\Lambda$ is more subtle: During this move, $s_1 = s_2$. Thus we have $\theta_1 = 0$, implying that $\Lambda_1 = 0$. $\Delta\theta_2 = 0$, and $2(s_4 - s_3) + [\theta_2 - s_2 + s_1]^+ = 2(s_4 - s_3) + \theta_2$, therefore $\Lambda_2 = \frac{2}{13}\theta_2$. We conclude that $\Delta\Lambda_2 = 0$.

Inward Basic Moves. The inward basic moves are simply the inward transitions. By symmetry we may assume a_2 is between s_3 and s_4. Therefore $\Lambda_2 = 0$, and $\Delta\Phi = \Delta\Psi + \Delta\Gamma + \Delta\Lambda_1$. We summarize the costs and potential changes in the table below. As before, the numbers in the table represent the upper bounds on $\Delta cost$, $\Delta\Psi$, $\Delta\Gamma$ and $\Delta\Lambda_1$.

Note that the transition "$s_1 \rightarrow\leftarrow s_4$" has been divided into two cases, depending on whether $a_1 \leq s_1$ or $a_1 > s_1$.

Move	$\Delta cost$	$\Delta\Psi$	$\Delta\Gamma$	$\Delta\Lambda_1$
$s_3 \rightarrow\leftarrow s_4$	2δ	0	$-\frac{28}{13}\delta$	0
$s_3 \rightarrow\leftarrow s_5$	2δ	$\frac{1}{13}\delta$	$-\frac{27}{13}\delta$	0
$s_3 \rightarrow\leftarrow s_6$	3δ	$-\frac{2}{13}\delta$	$-\frac{37}{13}\delta$	0
$s_2 \rightarrow\leftarrow s_4$	2δ	$-\frac{1}{13}\delta$	$-\frac{27}{13}\delta$	$\frac{2}{13}\delta$
$s_2 \rightarrow\leftarrow s_5$	2δ	0	-2δ	0
$a_1 \leq s_1$ and $s_1 \rightarrow\leftarrow s_4$	3δ	$\frac{2}{13}\delta$	$-\frac{37}{13}\delta$	$-\frac{4}{13}\delta$
$a_1 > s_1$ and $s_1 \rightarrow\leftarrow s_4$	3δ	$-\frac{64}{13}\delta$	$-\frac{37}{13}\delta$	0

To illustrate, we analyze the two last moves in this table: "$s_2 \rightarrow\leftarrow s_5$" and "$s_1 \rightarrow\leftarrow s_4$". The numbers for the other moves can be verified by straightforward calculations.

First, we consider the move "$s_2 \rightarrow\leftarrow s_5$", the inward transition involving s_2 and s_5. Since both servers move at speed 1, $\Delta cost = 2\delta$. Since s_5 moves towards a_2, the expression $|a_2 - s_5|$ in Ψ decreases by δ, while $|a_1 - s_2|$ cannot increase by more than δ, and the other terms in Ψ do not change. Therefore $\Delta\Psi \leq 0$. In Γ, only the term $(s_5 - s_2)$ changes, decreasing by 2δ, and thus $\Delta\Gamma = -2\delta$. To determine the upper bound on $\Delta\Lambda_1$, note that in this move $s_3 = s_4$, implying that $\Lambda_1 = \frac{2}{13}[\theta_1 - s_6 + s_5]^+$. Since s_5 decreases by δ, and θ_1 increases by at most δ, we obtain $\Delta\Lambda_1 \leq 0$.

Now we consider the move "$s_1 \rightarrow\leftarrow s_4$". This case occurs when s_2 and s_3 are already on a_2. Server s_1 moves at speed 2, and s_4 at speed 1, so $\Delta cost = 3\delta$. Since $(s_6 - s_1)$ decreases by 2δ and $(s_4 - s_3)$ decreases by δ, we have $\Delta\Gamma = -\frac{37}{13}\delta$.

To estimate the change of Ψ and Λ_1, we distinguish two subcases. Suppose first that $a_1 \leq s_1$. Then $|a_1 - s_1|$ increases by 2δ and $|a_2 - s_4|$ decreases by δ, so $\Delta\Psi = \frac{2}{13}\delta$. Also, in this case $\theta_1 = s_2 - s_1$, and thus θ_1 decreases by 2δ, while $s_4 - s_3$ decreases by δ. Therefore $\Delta\Lambda_1 = -\frac{4}{13}\delta$.

The second subcase is when $a_1 > s_1$. Then $|a_1 - s_1|$ decreases by 2δ and $|a_2 - s_4|$ decreases by δ, thus $\Delta\Psi = -\frac{64}{13}\delta$. In Λ_1 we have $\theta_1 = [s_2 - a_1]^+$, and thus $\Delta\theta_1 = 0$, while $s_4 - s_3$ decreases by δ. Thus $\Delta\Lambda_1 \leq 0$.

Summarizing, we have $\Delta cost + \Delta\Phi \leq \frac{155}{78}\Delta opt$ for each move. Therefore, by summation over the given request sequence, DC2 is $\frac{155}{78}$-competitive. □

4.4 Lower Bound on the Competitive Ratio of DC2

In this section we show that the competitive ratio of DC2 is no better than $\frac{107}{54} \approx 1.981$. Thus our analysis in the previous section is nearly tight.

Suppose that C is the competitive ratio of DC2. Given a configuration K, we define

$$\phi(K) \quad = \quad \sup_{\varrho} \{ cost(K, \varrho) - C \cdot opt(K, \varrho) \}$$

where $cost(K, \varrho)$ and $opt(K, \varrho)$ denote the cost of servicing the request sequence ϱ, by DC2 and the adversary, respectively, starting from K.

We first make a few simple observations about the function ϕ. C-competitiveness of DC2 implies that $0 \leq \phi(K) < \infty$. If two configurations K and K' are isometric (one can be obtained from the other by a translation or a flip) then, $\phi(K) = \phi(K')$. Furthermore, if K' is a configuration obtained from K by multiplying each distance between the servers by a number $\beta > 0$ then $\phi(K') = \beta\phi(K)$.

We also need the following fact that follows from the definition of ϕ and C-competitiveness of DC2.

Fact 1 *Let $\Delta cost$, Δopt be the costs of* DC2 *and the adversary, respectively, during a sequence of moves that starts at configuration K and ends at configuration K'. Then $\phi(K) \geq \Delta cost - C\Delta opt + \phi(K')$.*

We now introduce a notation for configurations. Let a' denote the adversary server that is on the request point and let a be the other server. Since ϕ is invariant under translation, we can assume that $s_1 = 0$. (As before, we use the same notation for the server and for its location on the line.) Similarly, by symmetry, we may assume that we only have two types of configurations: *type* A, in which the request is served by s_1, s_2, s_3, and *type* B, in which the request is served by s_2, s_3, s_4. In both cases we have $a' = s_2$. We will denote these configurations by $A(s_4, s_5, s_6, a)$ and $B(s_4, s_5, s_6, a)$, respectively.

	K	K'	$\Delta cost$	Δopt	Adversary Action
1.	$A(3, 12, 12, 12)$	$B(3, 6, 12, 12)$	12	9	$r = 3$, serve with a'.
2.	$B(3, 6, 12, 12)$	$A(0, 3, 6, -6)$	12	9	$r = 6$, serve with a', invert.
3.	$A(0, 3, 6, -6)$	$A(8, 11, 14, 8)$	24	6	$r = -8$, serve with a.
4.	$A(8, 11, 14, 8)$	$A(2, 8, 8, 8)$	15	0	$r = 8$, serve with a, invert.
5.	$A(0, 24, 24, -16)$	$A(16, 40, 40, 16)$	48	0	$r = -16$, serve with a.
6.	$A(16, 40, 40, 16)$	$B(24, 32, 40, 40)$	48	0	$r = 16$, serve with a, invert.
7.	$B(24, 32, 40, 40)$	$B(25, 30, 40, 40)$	5	3	$r = 25$, serve with a'.
8.	$B(25, 30, 40, 40)$	$A(0, 5, 30, -10)$	20	15	$r = 30$, serve with a', invert.
9.	$A(0, 5, 30, -10)$	$A(50, 55, 80, 50)$	150	120	$r = -50$, serve with a.
10.	$A(50, 55, 80, 50)$	$A(30, 50, 50, 50)$	55	0	$r = 50$, serve with a, invert.
11.	$A(30, 50, 50, 50)$	$A(0, 30, 30, -20)$	70	90	$r = 30$, serve with a', invert.
12.	$A(0, 6, 6, -4)$	$A(12, 18, 18, 12)$	36	24	$r = -12$, serve with a.
13.	$A(12, 18, 18, 12)$	$A(3, 12, 12, 12)$	21	0	$r = 12$, serve with a, invert.

Table 2: Description of the moves.

Moves in Table 2 are described as follows. The request point r is a number on the line (recall that s_1 is always at 0). After moving one adversary server to the request point, and moving our other servers as required by the algorithm, it may be necessary to invert the line or translate it to return to the condition that $s_1 = 0$ and s_2 is at the request point.

Theorem 4 *The competitive ratio of* DC2 *is no less than* $\frac{107}{54}$.

Proof: Let $\phi_A(s_4, s_5, s_6, a) = \phi(A(s_4, s_5, s_6, a))$. Applying Fact 1 to the moves 1-4 in Table 2, we get

$$\begin{aligned} 3\phi_A(1,4,4,4) &= \phi_A(3,12,12,12) \\ &\geq 63 - 24C + \phi_A(2,8,8,8) \\ &= 63 - 24C + 2\phi_A(1,4,4,4) \end{aligned}$$

and therefore $\phi_A(1,4,4,4) \geq 63 - 24C$. Applying Fact 1 to the moves 5-11 in Table 2, we get

$$\begin{aligned} 4\phi_A(0,6,6,-4) &= \phi_A(0,24,24,-16) \\ &\geq 396 - 228C + \phi_A(0,30,30,-20) \\ &= 396 - 228C + 5\phi_A(0,6,6,-4) \end{aligned}$$

and therefore $\phi_A(0,6,6,-4) \leq 228C - 396$. Finally, using the two inequalities derived above, and applying Fact 1 to the moves 12-13 in Table 2, we get

$$\begin{aligned} 228C - 396 &\geq \phi_A(0,6,6,-4) \\ &\geq 57 - 24C + \phi_A(3,12,12,12) \\ &\geq 57 - 24C + 189 - 72C \\ &= 246 - 96C, \end{aligned}$$

and thus $C \geq \frac{107}{54}$. □

5 Final Comments

We have presented a $\frac{155}{78}$-competitive randomized online algorithm for 2 servers on the real line. This is the first online algorithm for 2 servers that achieves a competitive ratio smaller than 2 in a non-uniform metric space with more than 3 points.

One natural question is whether we can improve the competitive ratio by using other server speeds. We experimented with different speeds for s_1 and s_6, but the calculations analogous to the ones in the paper yield only $C \approx 1.986$ for speeds approximately 1.75. Thus we doubt that this idea could lead to a substantial improvement of the competitive ratio.

It still remains an open problem whether there is a less-than-2-competitive randomized algorithm for 2 servers that works in an arbitrary metric space. The general problem appears very hard, so it is reasonable to concentrate on other natural special cases: star-shaped spaces (or the weighted-cache problem), and trees. We believe that our technique can be extended to continuous trees.

Any result that considerably improves the lower bound of $1+e^{-1/2} \approx 1.6065$ from [9] would also be of interest. That lower bound was recently slightly improved by Chrobak and Larmore (unpublished manuscript) to ≈ 1.608. (The precise value is $3/(3-2x)$, where x is the unique solution to the equation $x = e^{-x}$.)

In addition to being useful for studying randomized 2-server algorithms, the (k,l)-server problem is of its own interest. What is the best competitive ratio of online (k,l)-server algorithms? The problem is apparently very difficult, since even the case $l = 1$, that is, the k-server problem, is still open. As given in this paper, the competitive ratio of the $(4, 2)$-server problem is 2. It is possible that the work function techniques [8, 12, 13] can be extended to at least some cases, for example $k = 2l$ and $k = l - 1$. A lower bound of ≈ 1.831104 (more precisely, the solution to the equation $9C^3 - 21C^2 + 5C + 6 = 0$) for the $(6, 3)$ server problem is known but unpublished. It may also be possible to extend the k-competitive algorithm for k servers on trees [7] to this more general problem.

References

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. In *Proc. 4th European Symp. on Algorithms*, LNCS 1136, pages 419–430. Springer, 1996.

[2] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

[3] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proc. 29th Symp. on Theory of Computing*, pages 711–719, 1997.

[4] A. Blum, H. Karloff, Y. Rabani, and M. Saks. A decomposition theorem and lower bounds for randomized server problems. In *Proc. 33rd Symp. on Foundations of Computer Science*, pages 197–207, 1992.

[5] A. R. Calderbank, E. G. Coffman, and L. Flatto. Sequencing problems in two-server systems. *Mathematics of Operations Research*, 10:585–598, 1985.

[6] Marek Chrobak, Howard Karloff, T. H. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. Also in Proc. 1st Symp. on Discrete Algorithms, San Francisco, 1990, pp. 291-300.

[7] Marek Chrobak and L. L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[8] Marek Chrobak and L. L. Larmore. The server problem and on-line games. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 7, pages 11–64, 1992.

[9] Marek Chrobak, L. L. Larmore, C. Lund, and N. Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 1997. To appear.

[10] Sandy Irani and S. Seiden. Randomized algorithms for metrical task systems. In *Proc. 4th Workshop on Algorithms and Data Structures*, pages 159–170, 1995.

[11] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized competitive algorithms for non-uniform problems. In *Proc. 1st Symp. on Discrete Algorithms*, pages 301–309, 1990.

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

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

[14] E. Koutsoupias and C. Papadimitriou. The 2-evader problem. *Information Processing Letters*, 57:249–252, 1996.

[15] C. Lund and N. Reingold. Linear programs for randomized on-line algorithms. In *Proc. 5th Symp. on Discrete Algorithms*, 1994. 382-391.

[16] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[17] L. McGeoch and D. Sleator. A strongly competitive randomized paging algorithm. *Journal of Algorithms*, 6(6):816–825, 1991.