Trackless On-Line Algorithms for the Server Problem

Wolfgang W. Bein * Lawrence L. Larmore †

Abstract

A class of "simple" online algorithms for the k server problem is identified. This class, for which the term *trackless* is introduced, includes many known server algotithms. The k-server conjecture fails for trackless algorithms. A lower bound of $\frac{23}{11}$ on the competitiveness of any deterministic trackless 2-server algorithm and a lower bound of $1+\sqrt{2}/2$ on the competitiveness of any randomized trackless 2-server problem are given.

1 Introduction

Irani and Rubinfeld [9] give a simple algorithm for the 2-server problem in an arbitrary metric space, which is between 6 and 10-competitive. This algorithm is remarkable in at least two ways. First, it uses only constant memory and, for each request, serves the request in constant time. Secondly, the algorithm can be oblivious to the actual metric space, as long as the distances from the servers to the request point are known. In effect, the algorithm in its implementation never requires a data type "point." By contrast, the work function algorithm (WFA) [6], which is known to be 2-competitive for the 2-server problem (and $(2k-1)$-competitive for the k-server problem [10]) may need to retain information about all previously requested points.

Borodin and El-Yaniv [1] posed as an open question to show a lower bound greater than 2 on the competitive ratio of any deterministic 2-server algorithm that uses constant time and space per request. They also mention that for such a lower bound the model of computation needed to be formalized. Such a model is difficult to formalize because unlimited information can be hidden even in a single real number.

In this paper we introduce the concept of a *trackless* online algorithm. *Tracklessness* is a property of an online algorithm for the server problem which is a restriction on what input data the algorithm uses in its computation, and on what outputs the algorithm gives. Specifically, for each request point, the algorithm is only given as input the distances of the current server positions to the request point. A trackless algorithm may memorize such distance values, but is restricted from storing explicitly any points of the metric space. In the algorithm's response to a request, the algorithm is limited to producing output that identifies which server moves, and does not mention points in the metric space explicitly. Many known algorithms are trackless. These include BALANCE, BALANCE_SLACK, RANDOM_SLACK, and HARMONIC, whereas, for example, WFA is not trackless. The concept of tracklessness can be generalized to other online problems. For example, there is an obvious extension to the paging problem.

*Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: bein@cs.unlv.edu.
Research supported by NSF grant CCR-9821009.

†Department of Computer Science, University of Nevada, Las Vegas, NV 89154. Email: larmore@cs.unlv.edu.
Research supported by NSF grant CCR-9821009.

It would be desirable to settle the k-server conjecture by finding a k-competitive algorithm for the k-server problem, and it would be desirable if that algorithm were simple, in the sense that it uses few inputs as well as little time and space.

The results in this paper give an indication that this may not be possible. We suspect that if there is a k-competitive online algorithm for the k-server problem, it cannot be simple. In particular, we show that, if only trackless algorithms are considered, the k-server conjecture fails. (In fact, it fails for $k = 2$.) This is true despite the fact that the trackless model does not require memoryless computation.

Our main result is a lower bound of $\frac{23}{11} \approx 2.09$ on the competiveness of any deterministic trackless online algorithm for the 2-server problem. For randomized trackless online algorithms for the 2-server problem against an oblivious adversary, we show a lower bound of $1+\sqrt{2}/2 \approx 1.7071$ on the competitiveness. This is higher than the best known lower bound for the same problem without the trackless restriction.

2 The Trackless k-Server Problem

In the *k-server problem* we are given $k \geq 2$ mobile servers $1, \ldots, k$ that reside in a metric space M. A sequence of requests $\varrho = r^1 \ldots r^n$ is issued, where each request is specified by a point $r^t \in M$. To "satisfy" this request, one of the servers must be moved to r^t, $t = 1, \ldots n$, at a cost equal to the distance from its current location to r^t. An algorithm $\mathcal{A}$ for the k-server problem decides which server should be moved at each step t. $\mathcal{A}$ is said to be *online* if its decisions are made without the knowledge of future requests. Our goal is to minimize the total service cost. $\mathcal{A}$ is *C-competitive* if the cost incurred by $\mathcal{A}$ to service each request sequence ϱ is at most C times the optimal (offline) service cost for ϱ, plus possibly an additive constant independent of ϱ. (See Chapter 1 of [1] for a comprehensive discussion of competitiveness.) The *competitive ratio of $\mathcal{A}$* is the smallest C for which $\mathcal{A}$ is C-competitive.

Assume now that the servers are initially located at points $s_1^0 \ldots s_k^0 \in M$. Let $s_i^t \in M$ denote the location of server i at time t, meaning after the request r^t has been serviced. Note that $s_i^{t-1} r^t$ is the distance between the location of s_i and the request r^t at the time that request is made, and before any server has moved. We say that a point $p \in M$ is *active at step t* if there is either a server or a request at p at the time of the t^{th} request. That is, p is active at step t if and only if $p \in \left\{ r^t, s_1^{t-1}, \ldots, s_k^{t-1} \right\}$.

We define a $\mathcal{A}$ to be a *trackless* algorithm for the k-server problem if:

1. $\mathcal{A}$ can only move a server to an active point. That is, every output of $\mathcal{A}$ at step t is of the form "Move s_i to r" or "Move s_i to s_j."

2. All calculations by $\mathcal{A}$ at step t may use as inputs only $\left\{ s_i^{t-1} r^t \right\}$ for $1 \leq i \leq k$.

We do not restrict the computational power of a trackless algorithm, although all trackless algorithms currently in the literature use only $O(k)$ time at each step to make their calculations. We now review some of these.

Irani Rubinfeld Algorithm. The Irani Rubinfeld Algorithm, also known as BALANCE [9], is a deterministic trackless algorithm for the k-server problem in an arbitrary metric space. It uses $O(k)$ memory and $O(k)$ time at each step. For $k = 2$, the competitiveness of BALANCE is at most 10, but a 6 point counter-example shows that the competitiveness cannot be less than 6 for general spaces [3].

BALANCE_SLACK. BALANCE_SLACK is a deterministic trackless algorithm for the 2-server problem in an arbitrary metric space [3]. It uses $O(1)$ memory and $O(1)$ time at each step. It is 4-competitive for an arbitrary metric space.

RANDOM_SLACK. RANDOM_SLACK is a randomized trackless algorithm for the 2-server problem in an arbitrary metric space [3]. It is memoryless and uses $O(1)$ time at each step. It is 2-competitive for an arbitrary metric space.

HARMONIC. HARMONIC is a randomized trackless algorithm for the k-server problem in an arbitrary metric space. It uses $O(k)$ time at each step, and is memoryless. Its competitiveness is conjectured to be $\binom{k+1}{2}$, see [12]. For $k = 2$, its competitiveness is known to be 3, see [5].

LRU. *Least Recently Used* and other marking algorithms are trackless k-competitive algorithms for the paging problem with cache size k, which is equivalent to the k-server problem in a uniform space [2, 13]. They use $O(k)$ memory. (See [1] for a survey.)

MARK. MARK is a trackless randomized algorithm for the paging problem with cache size k [8]. It uses $O(k)$ memory and is $2H_k$-competitive, where H_k is the k^{th} Harmonic number.

The Tree Algorithm. The tree algorithm is a k-competitive algorithm for the k-server problem in trees [4]. A metric space M is a *tree* if M is path-connected and acyclic. The tree algorithm, although not explictly trackless, can be rewritten as a trackless algorithm. This algorithm automatically extends to all spaces which can be embedded in trees; we call such spaces *dendral* ("tree-like").

Ultrametric Spaces. An ultrametric space is a space such that, for any three points a, b, c, $ab \leq \max\{ac, bc\}$. There is a k-competitive deterministic trackless algorithm for the k-server problem in an arbitrary ultrametric space.[1] The proof is that every ultrametric space is a dendral space, so the trackless tree algorithm applies.

3 The Deterministic Lower Bound

In this section, we prove that no trackless algorithm for the 2-server problem which works for all metric spaces can be less than $\frac{23}{11}$-competitive. First we state, without proof, three easy technical lemmas.

We say that $\mathcal{A}$'s servers *match* the servers of an optimal offline algorithm (referred to as "the optimal servers", for short) if they are at the same points.

Lemma 1 *If the optimal servers are at two distinct points x and y, there is a request sequence whose optimal cost is zero, and which forces $\mathcal{A}$ to move its two servers to x and y.*

Lemma 2 *If the optimal servers match $\mathcal{A}$'s servers at distinct points x and y, where $xy = d_1$, and if there is a point z such that $xz = yz = d_2$, then there is a request sequence such that, if $\mathcal{A}$ serves this sequence, it pays $d_1 + d_2$, the optimal cost is d_2, and at the end of the service $\mathcal{A}$'s servers match the optimal servers, either at x, z or at y, z.*

We say that $\mathcal{A}$ is *lazy* if it moves at most one server at each step, and then only to the current request point. Without loss of generality, every trackless online algorithm is lazy, as follows from Lemma 3 below.

[1] Rainer Burkhardt suggested looking at these spaces.

Lemma 3 *Given any trackless online algorithm for the k-server problem, there is a lazy trackless online algorithm for the k-server problem which does not have greater cost.*

We are now ready to state the lower bound result.

Theorem 1 *There is no deterministic trackless online algorithm for the 2-server problem which is C-competitive for any* $C < \frac{23}{11} \approx 2.09$.

Proof: Let M be the set of vertices of the tiles of a tiling of the plane into congruent equilateral triangles (see Figure 1). We think of M as an infinite graph, where the edges are the edges of the triangles. Distance is defined to be the length (number of edges) of the shortest path between two points.

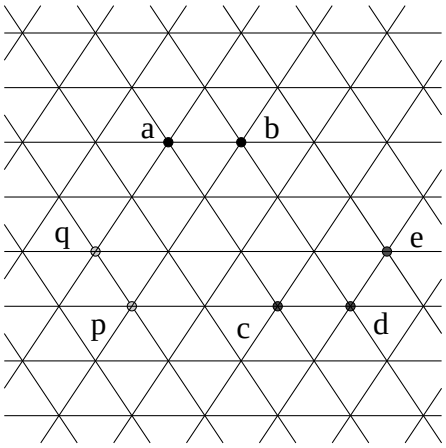


Figure 1: The metric space M

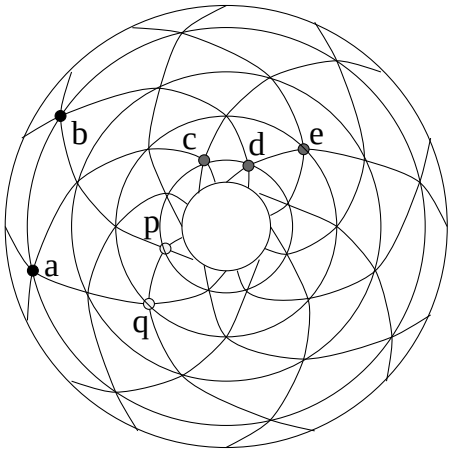


Figure 2: A mapping of M onto a torus

Now, let $\mathcal{A}$ be any trackless algorithm for the 2-server problem in M. We assume that the two servers begin at points which are distance 1 apart in M. We show that there exists a request sequence, α, in M such that, after $\mathcal{A}$ services α, its two servers are again distance 1 apart, and the total cost of moving the servers is positive and is at least $\frac{23}{11}$ times the optimal cost. We call such a sequence a *phase*. To show that $\mathcal{A}$ cannot be C-competitive for any $C < \frac{23}{11}$, we simply request many phases.

We now describe the request sequence α. We let a, b be the locations of the servers 1 and 2, respectively, at the beginning of a phase. Pick points $c, d, e, p, q \in M$ such that:

1. $cd = de = 1$, $ad = ae = 4$, and $ac = bc = bd = be = 3$.
2. $pq = 1$, $aq = 2$, and $ap = bp = bq = 3$.

Because of the geometry of M such points exist, see Figure 1. For now, assume that, if c is the first request point, $\mathcal{A}$ serves that request with server 1. We have six cases.

Case 1: If cd is requested, $\mathcal{A}$ serves that sequence with servers $1, 2$. In this case, define α to be the concatenation of four parts. The first part is cd. The second part is a sequence given by Lemma 1 which forces the servers to a and d, which are 4 apart. The third part is given by Lemma 2, and forces the servers to points which are 2 apart. The fourth part is given by Lemma 2 again, and forces the servers to points which are 1 apart. The algorithm pays 6 to service cd, pays at least 3 to service the second part, at least 6 to service the third part, and at least 3 to service the fourth part. Thus, the algorithm pays at least 18 to service α. The optimal service serves cd with

servers $2, 2$, at a cost of 4. The cost of servicing the second part is 0, as the optimal servers are already at a, d. The optimal cost to service the third part is 2 and to service the fourth part is 1, by Lemma 2. Thus, the optimal cost of servicing α is 7.

Case 2: If *cde* is requested, $\mathcal{A}$ serves that sequence with servers $1, 1, 2$. In this case, define α to be *cde*, followed by a sequence, as given by Lemma 1, that forces the servers to a and e, followed by two sequences that force the servers to points which are 1 apart, due to Lemma 2 as described in Case 1. The total cost for $\mathcal{A}$ for the phase is at least 20, while the optimal cost is 8.

Case 3: If *cded* is requested, $\mathcal{A}$ serves that sequence with servers $1, 1, 1, 2$. In this case, define α to be *cded*, followed by a sequence, as given by Lemma 1, that forces the servers to a and d, followed by two sequences that force the servers to points which are 1 apart, as given by Lemma 2. The total cost for $\mathcal{A}$ for the phase is at least 21, while the optimal cost is 9.

Case 4: If *cdede* is requested, $\mathcal{A}$ serves that sequence with servers $1, 1, 1, 1, 2$. In this case, define α to be *cdede*, followed by a sequence, as given by Lemma 1, that forces the servers to a and e, followed by two sequences that force the servers to points which are 1 apart, as given by Lemma 2. The total cost for $\mathcal{A}$ for the phase is at least 22, while the optimal cost is 10.

Case 5: If *cdeded* is requested, $\mathcal{A}$ serves that sequence with servers $1, 1, 1, 1, 1, 2$. In this case, define α to be *cdeded*, followed by a sequence, as given by Lemma 1, that forces the servers to a and d, followed by two sequences that force the servers to points which are 1 apart, as given by Lemma 2. The total cost for $\mathcal{A}$ for the phase is at least 23, while the optimal cost is 11.

Case 6: If *cdeded* is requested, $\mathcal{A}$ serves that sequence with servers $1, 1, 1, 1, 1, 1$. In this case, define α to be *pqpqpq*, followed by a sequence, as given by Lemma 1, that forces the servers to p and q, which are 1 apart. It is here that the trackless property of $\mathcal{A}$ plays a role. Since $\mathcal{A}$ is trackless, it must respond to p the same way it would respond to c, namely by moving server 1, because the only input $\mathcal{A}$ uses is the pair of distances $(s_1 r, s_2 r)$, which is $(3, 3)$ whether r is c or p. Similarly, once server 1 is at p, $\mathcal{A}$ must respond to the request at q in the same manner as it would to the request d if server 1 were at c, because in each case, the input is the pair $(1, 3)$. Extending this argument for the entire sequence, we see that $\mathcal{A}$ must serve $\alpha = pqpqpq$ in the same way as it would serve *cdeded*, namely with $1, 1, 1, 1, 1, 1$. Thus, after six steps, server 1 is at q and server 2 is at b. The total cost for $\mathcal{A}$ for the phase is at least 11, while the optimal cost is 5.

Since $\max\left\{\frac{18}{7}, \frac{20}{8}, \frac{21}{9}, \frac{22}{10}, \frac{23}{11}, \frac{11}{5}\right\} = \frac{23}{11}$, we have $cost_{\mathcal{A}} \geq \frac{23}{11} cost_{opt}$ for the phase.

We return to the situation where if c is the first request, $\mathcal{A}$ serves that request with server 2. By the property of tracklessness, $\mathcal{A}$ must also serve with 2 if the first request is p. M is symmetric in such a way that a and b are interchanged and c and p are interchanged (see Figure 1), and so, by symmetry, a request sequence can always be chosen so that $cost_{\mathcal{A}} \geq \frac{23}{11} cost_{opt}$ for the phase.

This completes the proof of the lower bound. □

Finally we note that the metric space M can be mapped onto a torus as in Figure 2. A moment's thought shows that with such a mapping the counterexample can be produced using this finite space of only 64 points.

4 The Randomized Lower Bound

In this section, we show that no trackless randomized online algorithm for the 2-server problem can have competitiveness less than $1 + 2^{-\frac{1}{2}} \approx 1.7071$.

Let n be a large integer, and let $\epsilon = 2\sqrt{2}/n$. Let M be a metric space consisting of a point a and an unlimited number of points $b_1, b_1, b_2, \ldots$ where the distance from a to each b_i is 1, and the

distance from b_i to b_j is ϵ for $i \neq j$. We refer to the set $b_1, b_2, b_3, \ldots$ as the "B-cluster." Let the initial positions of the servers be a and b_1. We define a randomized oblivious adversary against which every deterministic algorithm is not better than $(1 + \sqrt{2}/2)$-competitive. By [**?**], this proves the lower bound.

The adversary request sequence consists of a sufficiently large number of *phases*. At the beginning of each phase, the optimal servers are located at a and at some b_k, and the algorithm's two servers are located at a and b_ℓ for some ℓ. We now describe one phase.

Points in the B-cluster will not be re-used from phase to phase; b_1 will be considered to be used before the first phase, hence will never be requested. Let m be the number of points in the B-cluster that have been used up to the beginning of this phase. (Those points will necessarily be $b_1 \ldots b_m$.)

For each phase, independently, the adversary chooses one of two strategies, which we call "buzz" and "wander." With probability $\sqrt{2} - 1$, the adversary chooses to "buzz" during that phase, and with probability $2 - \sqrt{2}$ the adversary chooses to "wander" during that phase.

If the adversary chooses to buzz during the current phase, it requests $(b_{m+1}b_{m+2})^n a$. In this case, the adversary moves its servers to b_{m+1} and b_{m+2} and then at the end of the phase moves one server to a. The optimal cost for this phase is thus $2 + \epsilon$.

With probability $2 - \sqrt{2}$, the adversary chooses to *wander* during the current phase. In this case, the adversary randomly chooses an integer i in the range $1 \ldots n$, with equal probability for each i. The adversary then requests the sequence $b_{m+1}b_{m+2}b_{m+3} \ldots b_{m+i}a$. The adversary serves by moving one server around inside the B-cluster and leaving the other server at a. Thus, the optimal cost for the phase is $i\epsilon$.

In either case, the phase ends with the request a. The expected optimal cost for one phase is

$$Ecost_{opt} \;=\; \frac{n+1}{2}\Big(2-\sqrt{2}\Big)\epsilon + (2+\epsilon)\Big(\sqrt{2}-1\Big) = 4\sqrt{2} - 4 + O\Big(\frac{1}{n}\Big)$$

We now compute the expected cost $Ecost_{\mathcal{A}}$ during one phase. For any integer $t \geq 0$, let $\mathcal{A}_t$ be the deterministic algorithm which serves the first t requests in the B-cluster with the same one server that starts in the B-cluster, and then, at step $t+1$, if the request point is in the B-cluster, moves its server from a. After that, $\mathcal{A}_t$ uses its two servers to serve all further requests in the B-cluster, finally moving one of them back to a when a is requested at the end of the phase. Let $\mathcal{A}_\infty$ be the *stubborn* algorithm, which serves all requests in the B-cluster with one server, never moving the server from a during that phase. It is clear that any lazy randomized algorithm $\mathcal{A}$ must behave as either $\mathcal{A}_\infty$, or $\mathcal{A}_t$ for some t, during each phase.

Lemma 4 *During each phase,* $Ecost\mathcal{A} \geq 2\sqrt{2} - O\big(\frac{1}{n}\big)$.

Proof: Case I: $\mathcal{A} = \mathcal{A}_\infty$

In this case, $\mathcal{A}$ expends $2n\epsilon$ with probability $\sqrt{2} - 1$, and with probability $2 - \sqrt{2}$, expends ϵi where the average value of i is $\frac{n+1}{2}$. Thus

$$Ecost_{\mathcal{A}} = \tfrac{n+1}{2}\big(2-\sqrt{2}\big)\epsilon + 2n\epsilon\big(\sqrt{2}-1\big) \geq 6 - 2\sqrt{2} > 2\sqrt{2}.$$

Case II: $\mathcal{A} = \mathcal{A}_t$ for $t < n$

If the adversary buzzes, then $\mathcal{A}$ expends $t\epsilon$ before it moves a over to the B-cluster, after which further service is free, other than the final move back to a. It must also pay 2 to jump from a to the B-cluster and back.

If the adversary wanders, then either $i \leq t$, in which case $\mathcal{A}$ never moves its server from a, but pays $i\epsilon$ to move its server around inside the B-cluster, or $i > t$, in which case $\mathcal{A}$ pays $t\epsilon$ to

move its server around inside the B-cluster before it moves a server from a, but then must pay $(i-t-1)\epsilon$ more to move servers around inside the B cluster after that. It must also pay 2 to jump from a to the B-cluster and back. Summing up the terms, we have

$$\begin{aligned} Ecost_{\mathcal{A}_t} &= \sum_{i=1}^{t} \frac{1}{n}\Big(2-\sqrt{2}\Big) i\epsilon + \sum_{i=t+1}^{n} \frac{1}{n}\Big(2-\sqrt{2}\Big)(2+(i-1)\epsilon) + \Big(\sqrt{2}-1\Big)(2+t\epsilon) \\ &= 2\sqrt{2} \pm O\Big(\frac{1}{n}\Big) \end{aligned}$$

Case III: $\mathcal{A} = \mathcal{A}_t$ for $t \geq n$

If $t \geq 2n$, then $\mathcal{A}_t$ behaves exactly as $\mathcal{A}_\infty$, and we are done. Thus, we can assume that $n \leq t < 2n$. If the adversary buzzes, $\mathcal{A}$ expends $t\epsilon + 2$, while if the adversary wanders, $\mathcal{A}$ expends $i\epsilon$. Summing up the terms, we have

$$\begin{aligned} Ecost_{\mathcal{A}} &= \sum_{i=1}^{n} \frac{1}{n}\Big(2-\sqrt{2}\Big) i\epsilon + \Big(\sqrt{2}-1\Big)(2+t\epsilon) \\ &\geq 2\sqrt{2} + (t-n)\epsilon \geq 2\sqrt{2} \end{aligned}$$

□

Theorem 2 *There is no randomized trackless online algorithm for the 2-server problem which is C-competitive for any $C < 1 + 2^{-\frac{1}{2}} \approx 1.7071$ against the oblivious adversary.*

Proof: Suppose that $C < 1 + 2^{-\frac{1}{2}}$. Let ϱ be the request sequence in M consisting of the concatenation of a large number of independent randomly chosen phases. Then, by Lemma 4, for any deterministic trackless online algorithm $\mathcal{A}$, the ratio $cost_{\mathcal{A}}/cost_{opt}$ converges to

$$\frac{2\sqrt{2}}{4\sqrt{2}-4} \pm O\Big(\frac{1}{n}\Big) = 1 + 2^{-\frac{1}{2}} \pm O\Big(\frac{1}{n}\Big)$$

which exceeds C if n is sufficiently large. □

5 Final Comments

The randomized competitiveness of the 2-server problem is not known. The best known upper bound is 2, and the best known lower bound is $1 + e^{-\frac{1}{2}} \approx 1.6065$ [7], while we can show a lower bound of $1 + 2^{-\frac{1}{2}} \approx 1.707$ for the competitiveness of any randomized trackless algorithm for the 2-server problem in general metric spaces. This is an indicator, but not a proof, that the randomized competitiveness and the randomized trackless competitiveness of the k-server problem in general are different.

For uniform spaces, the deterministic competitiveness of the k-server problem is k, and since a uniform space can be embedded in a tree, there is also a k-competitive trackless algorithm. But the situation for randomized algorithms is different. In particular, there is a $\frac{3}{2}$-competitive randomized online algorithm for the 2-server problem in a uniform space and this is known to be optimal against an oblivious adversary [11]. Preliminary calculations indicate that there is no randomized trackless algorithm for that problem with competitiveness less than $\frac{20}{13} \approx 1.538$.

References

[1] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[2] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *J. Comput. Systems Sci.*, 50:244–258, 1995.

[3] Marek Chrobak and Lawrence L. Larmore. On fast algorithms for two servers. *J. Algorithms*, 12:607–614, 1991.

[4] Marek Chrobak and Lawrence L. Larmore. An optimal online algorithm for k servers on trees. *SIAM J. Comput.*, 20:144–148, 1991.

[5] Marek Chrobak and Lawrence L. Larmore. HARMONIC is three-competitive for two servers. *Theoret. Comput. Sci.*, 98:339–346, 1992.

[6] Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In Lyle A. McGeoch and Daniel D. Sleator, editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. AMS/ACM, 1992.

[7] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inform. Process. Lett.*, 63:79–83, 1997.

[8] Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12:685–699, 1991.

[9] Sandy Irani and R. Rubinfeld. A competitive 2-server algorithm. *Inform. Process. Lett.*, 39:85–91, 1991.

[10] Elias Koutsoupias and Christos Papadimitriou. On the k-server conjecture. *J. ACM*, 42:971–983, 1995.

[11] Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[12] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM J. Res. Dev.*, 38, 1994.

[13] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.