Uniqueness of a Memoryless Online Server Strategy on the Real Line

Ganesh R. Baliga[*] Anil M. Shende[†] Sunil M. Shende[‡]

Abstract

Server problems can be used to model many useful applications in computer science. A k-server problem, with $k \geq 2$, requires the design of an online algorithm: the algorithm schedules the movement of k servers in a metric space with the goal of minimizing the total distance moved by the servers on a sequence of requests to points in that space. A server strategy is *c-competitive* if on any request sequence, the net distance moved is within c times the optimal distance, where c is called the competitive factor of the algorithm.

A k-server strategy is *memoryless* if it schedules servers online using *only* the current configuration of servers and the position of the next request, without maintaining any book-keeping information about past service. In this paper, we study a sub-class of memoryless server strategies, called *uniform* server algorithms, with 2 servers on the real (Euclidean) line. We show that there is a *unique* algorithm in this sub-class whose competitive factor is best possible on the real line.

Keywords: Analysis of algorithms; k-server problems; online algorithms.

[*]Dept. of Computer Science, Rowan College of New Jersey, Glassboro, NJ 08028. Email: baliga@saturn.rowan.edu

[†]Dept. of Computer Science, Bucknell University, Lewisburg, PA 17837. Email: shende@bucknell.edu

[‡]Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588. Email: sunil@calypso.unl.edu

Journal of Computing and Information
Special Issue: Proceedings of Seventh International Conference of Computing and Information (ICCI'95), Trent University, Peterborough, Ontario, Canada, July 5-8, 1995, pp. 81-91.

1 Introduction

Online problems arise quite naturally, and rather often, in computer science. A candidate algorithm for an online problem must make dynamic decisions without full knowledge of the impact of these decisions on the future performance of the algorithm. Many data structuring problems and scheduling problems are online problems. Server problems [7], a variation of the general formulation of online problems [1], can be used to model a variety of situations arising in computer science including, among others, paging and planning the motion of the several heads in a multi-headed disk (see [8] for some recent results on server problems).

Informally, the generic k-server problem requires the design of an online algorithm to schedule k identical servers residing in a metric space. At any stage, a point in that space is requested; in response, the algorithm must schedule at least one server to *service* the request, i.e. move to the requested point. The total metric distance moved by all the servers is the *cost* of the algorithm for serving the request. We seek to minimize the cumulative cost of the algorithm for servicing a sequence of requests, without any *a priori* knowledge of the sequence. Clearly, for every request sequence, there is an associated *optimal* cost of service. A k-server algorithm is called *c-competitive*, $c \geq 1$, on a metric space if on every request sequence in the space, its cost is at most c times the corresponding optimal cost in that space, where c is called the *competitive factor* of the algorithm in the space. An algorithm which achieves the best possible competitive factor for a given metric space is said to be *strongly* competitive in that space.

It has been shown [7] that for the k-server problem, no deterministic algorithm can have a competitive factor strictly less than k when the metric space is non-trivial, i.e. when the space contains at least $k+1$ points. For fixed k, there exist deterministic algorithms for the k-server problem which are competitive in every metric space (e.g., see [6]). Similarly, for some special kinds of metric spaces such as lines and trees, there exist deterministic, strongly competitive algorithms for the k-server problem (e.g. see [2, 4]). A well-known open problem in the area of k-server problems is the resolution of the *k-server conjecture* [7]; the conjecture asserts that for every integer $k \geq 2$, there is a deterministic k-server algorithm that is k-competitive - hence strongly competitive - in *every* metric space. To date, the conjecture has been proven only for the case $k = 2$ [3, 5, 7]. However, the witnessing 2-competitive algorithms track, implicitly, the *past history* of requests and servicings in order to schedule service for the next request. An *unbounded* amount of computational resources like space and time are thus needed for service. For most applications modeled by the k-server problem, this is infeasible since resources are limited and fast, real-time response is critical.

This has motivated the investigation of *memoryless* k-server algorithms, viz. algorithms that do not maintain any book-keeping information about past service, but *only* use the current configuration of servers and the position of the next request to schedule service. Naturally, a memoryless algorithm, if shown to be *strongly competitive*, is a resource-optimal, online strategy for the problem. For instance, the algorithm DOUBLE COVERAGE [4] is a memoryless, k-competitive algorithm on lines and trees.

In this paper, we investigate memoryless 2-server algorithms on a simple, infinite metric space: the Euclidean line, $\mathbf{R}^1$. We postulate a natural assumption of *uniformity* for memoryless algorithms in this space. Intuitively, a uniform algorithm is memoryless and moves its servers by proportional amounts; we defer the formal definition to Section 3 of the paper. We show that the algorithm DOUBLE COVERAGE (henceforth abbreviated as **DC**) when restricted to the case $k = 2$, is the *unique*, uniform 2-competitive algorithm on the metric space $\mathbf{R}^1$. This sharp characterization implies that if the k-server conjecture is true and is witnessed by a uniform algorithm, then the restriction of the algorithm, for $k = 2$ servers on the Euclidean line, *must coincide* with **DC**. Unfortunately, for metric spaces such as the Euclidean plane, most reasonable uniform extensions of **DC** with 2 servers, are *not* 2-competitive. It seems, therefore, that under the assumption of uniformity, the existence of a 2-competitive, 2-server algorithm that works in every Euclidean space, is unlikely.

The paper is organized as follows. In Section 2, we state some preliminary definitions and notation. We define uniform 2-server algorithms on $\mathbf{R}^1$ and states our main result (Theorem 1) in Section 3, subsequently proving it in Section 4. A summary of the paper appears in Section 5.

2 Preliminaries

We shall consider 2-server algorithms operating in a simple, infinite metric space: the Euclidean (real) line, $\mathbf{R}^1$. Define the *signed* Euclidean distance, $d(a, b)$, between any points $a, b \in \mathbf{R}^1$ to be $(b - a)$. The *cost* incurred by a 2-server algorithm moving a server from point a to b is the absolute value of $d(a, b)$. We now develop some minimal notation *specific to 2-server algorithms* in the metric space $\mathbf{R}^1$.

We let σ denote a finite sequence of requested points. $[\cdot, \cdot, \ldots, \cdot]$ will denote explicit request sequences. For a request sequence σ, and for each integer $i > 0$, σ^i denotes the request sequence σ repeated i times. A 2-server algorithm, $\mathcal{X}$, has its servers labeled by x_1 and x_2. At any stage, the servers respectively occupy points p and q with $p \leq q$: we denote this canonically by saying that $\mathcal{X}$ is in the *configuration* $\langle p, q\rangle$. For any configuration $f = \langle p, q\rangle$, we denote by $D(f)$ the quantity $d(p, q) \geq 0$. Without loss of generality, we assume that server x_1 never crosses server x_2 at any stage of the algorithm. Also, when there is no confusion, we shall sometimes identify the label of a server with the point in $\mathbf{R}^1$ that it currently covers.

Suppose that algorithm $\mathcal{X}$ starts at some configuration $f = \langle p, q \rangle$ and subsequently gets a request sequence σ. Then $\mathcal{X}(f, \sigma)$ is defined as the configuration after $\mathcal{X}$ has serviced σ; the associated cumulative cost of servicing σ is denoted by $C_{\mathcal{X}}(f, \sigma)$. We define $C_{\text{OPT}}(f, \sigma)$ to be the *optimal* cost of servicing the sequence σ starting from configuration f.

Definition 1 *Let $c \geq 1$. The online algorithm $\mathcal{X}$ is said to be c-competitive on $\mathbf{R}^1$ if and only if for any starting configuration f, there exists a real constant e such that for any request sequence σ,*

$$C_{\mathcal{X}}(f, \sigma) \quad \leq \quad c \cdot C_{\text{OPT}}(f, \sigma) \ + \ e. \tag{1}$$

Every metric space $\mathcal{M}$ has its corresponding metric distance function. By appropriately defining the notions of configuration and cost, Definition 2 can be extended to any k-server algorithm, $k \geq 2$, and to any metric space $\mathcal{M}$. A k-server algorithm is said to be c-competitive if and only if it is c-competitive on *every* metric space. In the remainder of the paper, whenever we say that an algorithm is c-competitive, we shall assume that the algorithm satisfies Equation (1).

3 Uniform 2-server Algorithms

Intuitively, a deterministic, online 2-server algorithm is *memoryless* if and only if it services the current request based *solely* on the position of the request and the current configuration of its two servers. We further restrict our memoryless algorithms on $\mathbf{R}^1$ as follows:

Definition 2 *A memoryless 2-server algorithm $\mathcal{X}$ on the space $\mathbf{R}^1$ is said to be* uniform *if and only if there exist real constants β_1 and β_2 such that the following holds. For $p < q$, let $\mathcal{X}$ be in the configuration $\langle p, q \rangle$, and let $r \in \mathbf{R}^1$ be the next requested point which is distinct from p and q. Also, let d_1 be the signed distance moved by the server providing service at r, let d_2 be the signed distance moved by the other server, and let $\beta_r = d_2/d_1$.*

1. *If r lies* outside *the closed interval $[p, q]$ on $\mathbf{R}^1$ (i.e. $r < p$ or $r > q$), then $\beta_r = \beta_1$.*
2. *If r lies* inside *the closed interval $[p, q]$ on $\mathbf{R}^1$ (i.e. $p < r < q$), then $\beta_r = \beta_2$.*

Observe that a uniform strategy does not depend on the location of the points in the current server configuration and the current request, but only on the constants β_1 and β_2. This assumption is quite reasonable, and perhaps desirable: memoryless online server algorithms should have *succinct descriptions* and should adopt strategies unencumbered by any *exceptional behavior.* The definition of uniformity can be suitably generalized to higher dimensional Euclidean metric spaces. To our knowledge, all the memoryless online server strategies studied in the literature satisfy Definition 2, when restricted to the space $\mathbf{R}^1$.

Indeed, our main result is motivated by the algorithm **DC**, which uses an elegant, deterministic and *uniform* strategy when restricted to the case of $k = 2$ servers operating in $\mathbf{R}^1$ (see [2] for details). In general, with $k \geq 2$ servers, the algorithm is k-competitive on $\mathbf{R}^1$. Since k is the best possible competitiveness factor for any deterministic k-server algorithm, it follows that **DC** is strongly competitive on the space $\mathbf{R}^1$. For the sake of completeness, we provide below a brief description of **DC** on $\mathbf{R}^1$, specialized to the case when $k = 2$.

At any stage, let $f = \langle p, q\rangle$ (with $p \leq q$) be a configuration of **DC** and let the next point requested be $r \notin \{p, q\}$. In response, **DC** performs exactly one of the following two actions depending on r:

(A): If r is not in the closed interval $[p, q]$ on the real line, then move the server nearest to r (breaking ties arbitrarily) to service the request *without* moving the other server.

(B): If r is in the closed interval $[p, q]$, then service the request with the server closer to r, while simultaneously moving the other server towards r by the same distance. In other words, the next configuration attained by the algorithm is given by $\langle r, p+q-r)\rangle$ if $(r-p) \leq (q-r)$ and is given by $p+q-r, r$ otherwise.

4 The uniqueness of DC

Observe that **DC** is a uniform algorithm according to Definition 2, with constants $\beta_1 = 0$ and $\beta_2 = -1$. In light of this, our main result provides a sharp characterization of uniform, 2-server strategies on $\mathbf{R}^1$.

Theorem 1 **DC** *is the* only *uniform, 2-competitive algorithm with 2 servers on* $\mathbf{R}^1$.

We defer the proof of Theorem 1 to the end of the section. We shall first establish some results that delineate the consequences of 2-competitiveness of a uniform 2-server algorithm. Combining these results will provide us with the characterization in Theorem 1.

Our competitiveness results for a 2-server algorithm $\mathcal{X}$ are established in the framework of a game between the player, $\mathcal{X}$, and an arbitrary *adversary*, $\mathcal{A}$. The adversary has its own two servers labeled as a_1 and a_2; if the adversary's current configuration is $\langle p', q'\rangle$ with $p' \leq q'$, then servers a_1 and a_2 respectively cover the points $p^{'}$ and $q^{'}$. In general, the game proceeds in rounds; a round begins with the adversary specifying the next request and servicing the request with one of *its* servers. The player completes the round by servicing the request with one of its servers. To establish a contradiction to the assumed 2-competitiveness of the player, it suffices to show that from any given starting configuration f, the adversary has a strategy such that on a suitably chosen request sequence σ, the quantity $(C_{\mathcal{X}}(f,\sigma) - c \cdot C_{\mathcal{A}}(f,\sigma))$ is greater than e, where e is the constant depending on f in Equation (1) (with $c = 2$) for the player.

Given a sufficiently small real constant $\epsilon > 0$, we define two configurations f_1 and f_2 to be ϵ-aligned if and only if f_1 and f_2 have one common point, and the remaining pair of corresponding points are no more than ϵ distance apart. Our first proposition shows that the adversary can always *force* a c-competitive player $\mathcal{X}$ to reach a configuration that is ϵ-aligned with the adversary's configuration. We remark that the claim holds for *any* metric space, not just $\mathbf{R}^1$.

Proposition 1 *Suppose that $\mathcal{X}$ is a c-competitive 2-server algorithm. Let f be any configuration, let p and q be any two distinct points. Then, for any chosen $\epsilon > 0$, there exists an integer $i \geq 1$ such that the configurations $\langle p, q \rangle$ and $\mathcal{X}(f, [p,q]^i)$ are ϵ-aligned.*

PROOF. Otherwise, we can construct an adversary $\mathcal{A}$ that stays in the configuration $\langle p, q \rangle$; each subsequent request will cause $\mathcal{X}$ to incur cost greater than ϵ while $\mathcal{A}$ incurs no cost. The ratio of $\mathcal{X}$'s cost to $\mathcal{A}$'s cost thus grows unboundedly, contradicting Definition 1. $\square$

Proposition 2 *Let $c > 0$ be a constant. Let $\mathcal{X}$ be a uniform 2-server algorithm over the metric space $\mathbf{R}^1$. Suppose that there is an adversary $\mathcal{A}$ such that for any starting configuration $f = \langle p, q \rangle$, there is a constant $\epsilon > 0$ depending on f and a request sequence σ such that the following holds.*

1. $D(\mathcal{X}(f,\sigma)) \geq D(f)$,
2. $\mathcal{X}(f,\sigma)$ *and* $\mathcal{A}(f,\sigma)$ *are ϵ-aligned, and*
3. $C_{\mathcal{X}}(f,\sigma) - c \cdot C_{\mathcal{A}}(f,\sigma)) > c\epsilon$.

Then, $\mathcal{X}$ is not *c-competitive on $\mathbf{R}^1$.*

PROOF. Assume that the three conditions in the hypothesis hold, and by way of contradiction, let $\mathcal{X}$ satisfy Equation (1) with the constant e. Let $f_0 = f$ and let $f_1 = \langle p_1, q_1 \rangle$ denote the configuration $\mathcal{X}(f_0, \sigma)$. From condition 1, if $t = d(p_1, q_1)/d(p,q)$

then $t \geq 1$. Now, define a *similarity* transformation T on $\mathbf{R}^1$ given by $T(x) = t \cdot (x - p) + p_1$: observe that T maps the points p and q to the points p_1 and q_1 respectively. Moreover, for any distinct points $r, s \in \mathbf{R}^1$, $d(T(r), T(s)) = t \cdot d(r, s)$. With a slight abuse of notation, for any request sequence σ', we let $T(\sigma')$ denote the ordered request sequence obtained by applying the transformation T to each requested point in σ'.

The adversary $\mathcal{A}$ adopts the following strategy. Starting at configuration f_0, it first provides the request sequence $\sigma_0 = [\sigma, p_1, q_1]$ to $\mathcal{X}$. Then, it provides sequences $\sigma_1, \sigma_2, \ldots$ such that $\sigma_i = T(\sigma_{i-1})$, $i \geq 1$. The servicing policy for $\mathcal{A}$ is straightforward: if it used a server[1] on any request r in σ_0, then for every $i \geq 1$, $\mathcal{A}$ uses that *same* server for the corresponding request, $T^i(r)$, in the sequence σ_i.

Now, since $\mathcal{X}(f_0, \sigma) = \langle p_1, q_1 \rangle$, it follows that the player incurs no cost on the last two requests in σ_0, while from condition 2, we conclude that the adversary can reach configuration $\langle p_1, q_1 \rangle$ after processing σ_0, such that

$$C_{\mathcal{A}}(f_0, \sigma_0) \leq C_{\mathcal{A}}(f_0, \sigma) + \epsilon.$$

Therefore, using condition 3, we have $C_{\mathcal{X}}(f_0, \sigma_0) - c \cdot C_{\mathcal{A}}(f_0, \sigma_0)) = \delta > 0$.

Since $\mathcal{X}$ is a uniform algorithm, it follows that the distances moved by servers on each request in $T(\sigma_0)$ are t times the distances moved by the servers on the corresponding request in σ_0. In general, for $i \geq 0$, let $\sigma^{[i]} = [\sigma_0, \sigma_1, \ldots, \sigma_i]$ denote the cumulative request sequence up to stage i, and let $f_{i+1} = \mathcal{X}(f_0, \sigma^{[i]})$, the configuration of $\mathcal{X}$ after processing sequence $\sigma^{[i]}$). From the above discussion, the following equation holds for $i \geq 0$:

$$C_{\mathcal{X}}(f_i, \sigma_i) - c \cdot C_{\mathcal{A}}(f_i, \sigma_i) \quad = \quad \delta t^i$$

From this, we can conclude that for $i \geq 0$:

$$C_{\mathcal{X}}(f_0, \sigma^{[i]}) - c \cdot C_{\mathcal{A}}(f_0, \sigma^{[i]}) \quad = \quad \delta \sum_{j=0}^{i} t^j$$

Since $\delta > 0$ and $t \geq 1$, the right hand side of the above equation grows unboundedly with increasing i and will eventually exceed the constant e. This contradicts the assumption that $\mathcal{X}$ is c-competitive. $\square$

For the remainder of the paper, we assume that the player, denoted $\mathcal{P}$, is a *uniform* 2-server algorithm. Without loss of generality, we may assume that the player starts from the configuration $\langle 0, L \rangle$, for some positive real number L.

[1]It is easy to see that the $\mathcal{A}$ need only move *one* server per request.

Lemma 1 *Let $\mathcal{P}$ be a 2-competitive algorithm with constants β_1 and β_2. From any starting configuration $\langle 0, L\rangle$, if $\mathcal{P}$ gets a request at some point $r \in \mathbf{R}^1$, then $\mathcal{P}$ schedules the server that is closer to r to provide service at r.*

PROOF. Let $\langle 0, L\rangle$ be the starting configuration of $\mathcal{P}$, for some $L > 0$. Since $\mathcal{P}$'s servers never cross each other, it follows that when $r < 0$ or when $r > L$, the servers closer to r, respectively, x_1 and x_2, service these requests.

Let r be such that $r < L - r$, i.e. r is in the open interval $(0, L)$ and is closer to x_1. Arguing by contradiction, suppose that $\mathcal{P}$ schedules the server x_2 to move to r. The adversary selects the strategy of servicing a request with *its* closer server, and provides the following set of requests:

1. Request r.
2. Choose a positive $\epsilon < 2(L - 2r)/3$, and request $L + r + \epsilon$.
3. For some finite $i \geq 1$, request the sequence $[r, L+r+\epsilon]^i$ until the player's servers are ϵ-aligned at the configuration $[r, L + r + \epsilon]$ by Proposition 1.

Elementary calculations reveal that the player incurs a cost of at least $2L + \epsilon$ while the the adversary's cost is exactly $2r+\epsilon$; hence $C_{\mathcal{P}} - 2C_{\mathcal{A}} > 2\epsilon$ on the above sequence. Since the adversary and the player end up ϵ-aligned, it follows that the player's servers must be at least a distance of L apart. This satisfies all three conditions of Proposition 2, and we conclude that $\mathcal{P}$ is not 2-competitive. Hence, the closer server x_1 must have serviced the initial request at r. □

Lemma 2 *If $\mathcal{P}$ is 2-competitive, then the constant $\beta_1 = 0$.*

PROOF. Let $f = \langle 0, L\rangle$ be the initial configuration of $\mathcal{P}$. Suppose that $\mathcal{P}$ gets a request at the point $r = 2L$. Arguing by contradiction, when $\beta_1 \neq 0$, $\mathcal{P}$ moves server x_1 to the point $q = \beta_1 L$, since x_2 must provide service at r by Lemma 1.

We construct an adversary $\mathcal{A}$ that is initially in the configuration $\langle 0, L\rangle$. On getting the request at r, $\mathcal{A}$ moves to the configuration $\langle 0, 2L\rangle$. At this stage, the player's configuration is $\langle q, 2L\rangle$, the absolute distance between x_1 and x_2 is $(2L - q)$, the player's cost is $(L+ \mid q \mid)$ and the adversary's cost is exactly L. Now, $\mathcal{A}$ chooses a positive constant, ϵ, such that $\epsilon < 3 \mid q \mid /10$, and selects one of the following strategies depending on the sign of q.

1. If $q > 0$ then $\mathcal{A}$ requests the point $p = (L + q/2 - \epsilon)$ and services the request with server a_2. Next, $\mathcal{A}$ requests the sequence of points $[0, p]^i$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\langle 0, p\rangle$ by Proposition 1.

2. If $q < 0$ then $\mathcal{A}$ requests the point $p = (L+q/2+\epsilon)$ and services the request with server a_1. Next, $\mathcal{A}$ requests the sequence of points $[p, 2L]^i$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\langle p, 2L\rangle$ by Proposition 1.

When $q > 0$ (respectively, when $q < 0$), it is easy to see that the point p is closer to the player's server x_1 (respectively, x_2). By Lemma 1, $\mathcal{P}$ is obliged to move x_1 (respectively, x_2) to service the request at p when $q > 0$ (respectively, when $q < 0$). However, the adversary services p with its server a_2 (respectively, a_1). In both cases, the final configuration reached by the player's servers is such that the distance between $\mathcal{P}$'s servers is greater than L. Hence, to apply Proposition 2 and contradict the 2-competitiveness of $\mathcal{P}$, it only remains to show that $C_{\mathcal{P}} - 2C_{\mathcal{A}} > 2\epsilon$ for the above sequence of requests.

If $q > 0$, careful counting establishes that the net cost to the player before ϵ-alignment at $\langle 0, p\rangle$ is at least $(4L+q/2-\epsilon)$ while the adversary's cost is $(2L-q/2+\epsilon)$. Hence, $C_{\mathcal{P}} - 2C_{\mathcal{A}} \geq 3q/2 - 3\epsilon > 2\epsilon$, since $\epsilon < 3q/10$.

Likewise, if $q < 0$, then $\mathcal{P}$'s net cost before ϵ-alignment with $\mathcal{A}$ at configuration $\langle p, 2L\rangle$ is at least $(4L - 5q/2 - \epsilon)$, while the adversary incurs a cost of $(2L + q/2 + \epsilon)$. Hence, $C_{\mathcal{P}} - 2C_{\mathcal{A}} \geq -7q/2 - 3\epsilon > 2\epsilon$, since $\epsilon < -3q/10 < -7q/10$.

Applying Proposition 2 yields the desired contradiction that $\mathcal{P}$ is not 2-competitive. Thus, $\beta_1 = 0$, i.e. the other server does not move at all. □

Lemma 3 *If $\mathcal{P}$ is 2-competitive, then the constant $\beta_2 = -1$.*

PROOF. At the outset, we note that Lemma 1 and the fact that servers never cross, together imply that $\beta_2 \geq -1$. Otherwise, in configuration $\langle 0, L\rangle$, if the request is at r such that $L/(1+ \mid \beta_2 \mid) < r < L/2$, then server x_1 services, but the servers will cross each other.

For the sake of contradiction, let β_2 be strictly greater than -1. Let $\langle 0, L\rangle$ be the initial configuration of $\mathcal{P}$ and $\mathcal{A}$. If $\mathcal{P}$ get a request at the point $r > L/2$, it services r using server x_2 (by Lemma 1) but simultaneously moves x_1 to a point $q < L - r$ (since $\beta_2 > -1$). Depending on β_2, $\mathcal{A}$ selects between the following two strategies; in both cases, $\mathcal{A}$ uses a suitably chosen positive constant $\epsilon < L/2$ whose value is made explicit later in the analysis.

1. If $\beta_2 > 0$, then $\mathcal{A}$ chooses a positive constant $\epsilon < L/2$, requests $r = (L/2 + \epsilon)$, and services r with its *farther* server a_1. The next request is now placed at the point $p = (3L/2 + 2\epsilon)$, and $\mathcal{A}$ services it with the server a_2. Finally, $\mathcal{A}$ requests the sequence $[r, p]^i$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $[r, p]$ by Proposition 1.

2. If $-1 < \beta_2 \leq 0$, then $\mathcal{A}$ chooses another positive constant $\epsilon < L/2$, requests $r = (L/2 + \epsilon)$, and services r with its *farther* server a_1, incurring a cost of $(L/2 + \epsilon)$. The next request is to the point $p = (2L + q)$, and $\mathcal{A}$ services it with the server a_2. The point, $t = (L + q + \epsilon)$, is requested next and serviced by $\mathcal{A}$ with its server a_1. Finally, $\mathcal{A}$ requests the sequence of points $[p + 2\epsilon, t]^i$ for some $i \geq 1$ until the player's servers are ϵ-aligned at the configuration $\langle t, p + 2\epsilon \rangle$ by Proposition 1. $\mathcal{A}$ services the first request to $p + 2\epsilon$ with its server a_2.

We now analyze each of the above cases and determine appropriate values for the constants chosen above. First, suppose that $\beta_2 > 0$. On the request sequence chosen by $\mathcal{A}$, Lemma 2 implies that $\mathcal{P}$ must service the second request at p with its server x_2, without moving the server x_1. Hence, the total cost of $\mathcal{P}$ is at least $(2L + \epsilon + 2 \mid q \mid)$ while $\mathcal{A}$ incurs a cost of $(L + 3\epsilon)$. Since $q = \beta_2(\epsilon - L/2)$ by uniformity, the adversary's choice of a constant $\epsilon < \beta_2 L/(7 + 2\beta_2) < L/2$ forces ϵ to be less than $2 \mid q \mid /7$. This ensures that $C_{\mathcal{P}} - 2C_{\mathcal{A}} > 2\epsilon$; applying Proposition 2 yields the desired contradiction to the 2-competitiveness of $\mathcal{P}$.

Otherwise, suppose that $-1 < \beta_2 \leq 0$. By Lemma 1 and Lemma 2, $\mathcal{P}$ uses the server x_2 for providing service for each of the first three requests to r, p and t respectively. Counting net cost, we can see that the cost to $\mathcal{P}$ will be at least $(5L + 2q - \epsilon)$ while the adversary incurs cost equal to $(2L + 2q + 3\epsilon)$. Recall that by uniformity, $q = \beta_2(\epsilon - L/2)$. Hence, by choosing a constant $\epsilon < (1 + \beta_2)L/(9 + 2\beta_2) < L/2$, the adversary can ensure that $\epsilon < (L - 2q)/9$. Consequently, $C_{\mathcal{P}} - 2C_{\mathcal{A}} > 2\epsilon$, and the contradiction follows from the conditions of Proposition 2. □

PROOF. (of **Theorem 1**) Lemma 2 and Lemma 3 together establish Theorem 1. □

5 Summary

We have provided a very sharp characterization of uniform, 2-competitive 2-server algorithms in the space $\mathbf{R}^1$. Assuming that the notion of uniformity is consistently extended to higher dimensional Euclidean spaces (e.g., in terms of the ratios of distances travelled along the basis vectors etc.), Theorem 1 implies that any uniform, 2-server algorithm must mimic **DC** when restricted to the subspace $\mathbf{R}^1$. This imposes a very tight constraint on uniform, 2-server algorithms in Euclidean spaces.

It is not clear how one could extend the uniformity assumption to arbitrary spaces, but it would be of interest to obtain useful generalizations of uniformity for higher dimensional Euclidean spaces. However, many reasonable uniform extensions of **DC** (with 2 servers) are not 2-competitive in the two dimensional Euclidean plane, $\mathbf{R}^2$. We believe, therefore, that it is very unlikely that the k-server conjecture will be resolved by a uniform algorithm, i.e. a general k-competitive witness algorithm will probably need to track at least *some* past history of service.

References

[1] Allan Borodin, Nathan Linial, and Michael Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 373–382, 1987.

[2] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1990.

[3] Marek Chrobak and Lawrence L. Larmore. A new approach to the server problem. *SIAM Journal on Discrete Mathematics*, 4(3):323–328, August 1991.

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

[5] Marek Chrobak and Lawrence L. Larmore. Generosity helps, or an 11-competitive algorithm for three servers. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 196–202, 1992.

[6] Sandy Irani and Ronitt Rubinfeld. A competitive 2-server algorithm. *Information Processing Letters*, 39:85–91, 1991.

[7] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988.

[8] Lyle A. McGeoch and Daniel D. Sleator, editors. *On-Line Algorithms: Proceedings of a DIMACS Workshop*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, February 1991.