Competitive k-Server Algorithms

Amos Fiat * Yuval Rabani * Yiftach Ravid *

Abstract

In this paper we give deterministic competitive k-server algorithms for all k and all metric spaces. This settles the k-server conjecture [MMS] up to the competitive ratio. The best previous result for general metric spaces was a 3-server randomized competitive algorithm [BKT] and a non-constructive proof that a deterministic 3-server competitive algorithm exists [BBKTW]. The competitive ratio we can prove is exponential in the number of servers. Thus, the question of the minimal competitive ratio for arbitrary metric spaces is still open.

1 Introduction

Competitive algorithms were introduced by Sleator and Tarjan [ST] in the context of searching a linked list of elements and the paging problem. [ST] sought a worst case complexity measure for on-line algorithms that have to make decisions based upon current events without knowing what the future holds. The immediate problem is that on-line algorithms are incomparable, on-line algorithm A may be better than on-line algorithm B for one sequence of events and algorithm B may be better than A for another sequence of events. The conceptual breakthrough in [ST] was to compare the algorithms, not to each other, but to a globally optimal algorithm that knows the request sequence in advance. The *competitive ratio* of an on-line algorithm A is defined as the supremum, over all sequences of events and all possible algorithms ADV, of the ratio between the cost of A and the cost of ADV. An algorithm that achieves a competitive ratio of at most c is called *c-competitive*. The competitive ratio may depend on the size and parameters of the problem. Algorithms are called competitive if the competitive ratio is independent of the parameters of the problem or if the dependency is provably unavoidable.

Sleator and Tarjan gave competitive algorithms for managing a linked list of elements and for paging. Karlin et al. [KMRS] later gave competitive algorithms for Snoopy Caching.

Borodin, Linial and Saks [BLS] generalize the concept to arbitrary *task systems*. Task systems capture a very large set of on-line problems but the generality of task systems implies

*Department of Computer Science, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, ISRAEL.

that task systems cannot perform very well relative to an optimal off-line (prescient) algorithm. [BLS] give an upper bound on the competitive ratio of any task system and show that some task systems have a matching lower bound. The competitive ratio upper bound for task systems depends on the number of states of the system. For a limited set of task systems, Manasse, McGeoch and Sleator [MMS] later gave a decision procedure to determine if a given on-line algorithm is c-competitive.

[MMS] generalize the paging problem to the *k-server problem*. The on-line k-server problem may be defined as follows: We are given a metric space $\mathcal{M}$ and k servers which move among the points of $\mathcal{M}$. Repeatedly, a request, a point x in the metric space, is given. In response to this request, we must choose one of the k servers and move it from its current location to x, incurring a cost equal to the distance from its current location to x. Note that the paging problem with k page slots in memory and n pages overall is isomorphic to the k-server problem on a metric space with n points and a uniform distance matrix (except with 0s on the diagonals). Let $\mathcal{A} = \{A(k, \mathcal{M})\}$ be a family of on-line k-server algorithms for the metric space $\mathcal{M}$, where k ranges over all positive integers, and $\mathcal{M}$ ranges over all possible metric spaces. $\mathcal{A}$ is called *competitive* if there exists a sequence $c_1, c_2, c_3, \ldots$, such that for each metric space $\mathcal{M}$, and for each k, $A(k, \mathcal{M})$ is c_k-competitive. The competitive ratio for the algorithm of [BLS] depends on the number of points in the metric space.

Another version of the k-server problem is to charge for "time" rather than "transport". If we assume that all servers move at some common speed and allow all servers to move simultaneously then the off-line problem becomes one of minimizing the total time spent to serve the requests, subject to the limitation that requests are served in order of arrival. The on-line algorithm may position its servers to deal with future events but gets the next request only when the current event is dealt with. We call this version of the problem the *min-time server problem*.

[MMS] give a lower bound for the competitive ratio of any on-line k-server algorithm: for any deterministic k-server algorithm and any metric space with more than k points there exists a sequence of requests such that the cost of the on-line algorithm is no less than k times the cost of an optimal off-line algorithm, minus an additive term.

[MMS] also conjectured that this lower bound is tight, up to an additive term. This conjecture is known as the *k-server conjecture*. They constructed k-competitive algorithms for all metric spaces if $k = 2$ and for all $(k + 1)$-point metric spaces. (Other competitive 2-server algorithms were later given by Irani and Rubinfeld [IR], by Chrobak and Larmore [CL2], by Turpin [Tur], and by the authors).

Prior to this paper, only the additional case $k = 3$ was solved for general metric spaces using the randomized HARMONIC algorithm suggested by Raghavan and Snir [RS]. This is due to Berman, Karloff and Tardos [BKT]. The competitive ratio is bounded by 3^{17000} [Rag]. Recently, Grove [Gro] showed that HARMONIC is $O(k2^k)$-competitive. The competitive ratio for randomized on-line algorithms is described as an expectation. It is important to make

precise the definition of the worst case competitive ratio for randomized algorithms. This can be done in terms of an adversial game with various assumptions on the strength of the adversary. HARMONIC uses randomization rather weakly; the randomization is used to select the next move but is not used to "hide" the on-line configuration from the adversary designing the sequence. The lower bound of k from [MMS] holds for such randomized algorithms.

A general result of Ben-David et al. [BBKTW] gives a non-constructive proof that the existence of any randomized on-line algorithm, which uses randomization of the form used by HARMONIC, implies the existence of a deterministic on-line algorithm, at the cost of squaring the competitive ratio. Randomized algorithms that use randomization to hide the on-line configuration from the adversary are dealt with by Fiat et al. [FKLMSY], McGeoch and Sleator [MS], and Karlin et al. [KMMO].

Competitive k-server algorithms were discovered for specific metric spaces. Specifically, k-competitive deterministic on-line algorithms for points on a line (Chrobak et al. — [CKPV]) and for points on a tree (Chrobak and Larmore — [CL1]). Randomized on-line algorithms were discovered for resistive graphs (Coppersmith et al. — [CDRS]) and points on a circle ([CDRS] and Karp — [Kar]). A deterministic competitive k-server algorithm for the circle was recently discovered (Fiat et al. [FRRS]). [CKPV] also prove that the optimal off-line k-server problem is equivalent to network flow problems and thus has a polynomial-time solution.

The [MMS] definition of the competitive ratio allows an additive term in addition to the ratio; *i.e.*, the on-line algorithm is allowed to perform some (constant) amount of work for free. The analysis of the line and tree algorithms above ([CKPV], [CL1]) require this additive term. The analysis gives an additive term if the initial configuration does not have all servers starting at one common point. This term depends on the initial distances between the servers. While the analysis is clearly overly pessimistic, neither of these algorithms is k-competitive if one discards the additive term.

[FKLMSY] introduce the concept of an on-line algorithm competitive against a set of on-line algorithms. The idea is to combine two on-line algorithms to obtain a third algorithm which has the advantages of both, at least to within some ratio. The new algorithm can be viewed as some kind of MIN operator on the two input algorithms. For the paging problem, [FKLMSY] show that the MIN operation is possible and give tight bounds on what is realizable and what is not. Performing a MIN operation for other metric spaces was left as an open problem.

Our main result is a competitive k-server algorithm for any metric space, called the Expand-Contract algorithm (denoted EC). We give a recursive construction for the k-server algorithm using ℓ-server algorithms, $\ell < k$. The base case is the optimal greedy algorithm for one server. Our algorithm is deterministic and runs in polynomial time, with respect to the length of the input sequence. It requires no additive term in the definition of the competitive ratio, irrespective of the initial configuration. However, the bound we can prove on its competitive ratio is exponential in $k \log k$.

Our construction also gives competitive algorithms for the min-time server problem.

Extending the results in [FKLMSY], we describe a MIN operator for on-line k-server algorithms on any metric space. Viewed properly, the existence of a MIN operator follows immediately from a result of Baeza-Yates, Culberson and Rawlins on the m-lane cow problem [BCR] or from a result of Papadimitriou and Yannakakis on traversing a dynamically revealed layered graph [PY]. Generalizations of the results of [BCR] and [PY] appear in [FFKRRV].

Our competitive k-server algorithm is derived from the MIN operator applied to a large set of (non-competitive) k-server algorithms. The basic observation we make is that one of our non-competitive k-server algorithms *is* competitive if the optimal off-line algorithm does little work. If the MIN operator is applied to a set of algorithms, one of which is competitive, then the MIN algorithm must also be competitive. Thus, we relate the work done by the optimal off-line algorithm to the competitiveness of our MIN algorithm. If the optimal algorithm did perform a great deal of work then we perform a reorganization step, the cost of which is charged against the work that the optimal algorithm already did.

2 Basic Definitions

We define and describe k-server algorithms that work on any specific metric space. The underlying metric space $\mathcal{M} = (X, \text{dist})$ will usually be omitted from the definitions. Definitions 1, 2, and 6 of k-server algorithms below are equivalent to the definitions in [MMS]. The definition of competitiveness against other algorithms follows [FKLMSY].

Definition 1 *A* k-server algorithm *gets an initial k-server configuration C_0 and a sequence of requests $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$. A* configuration *is the set of points occupied by the k servers in the metric space. The request sequence σ is a sequence of points in the metric space. The k-server algorithm selects a sequence of configurations $C_1, C_2, \ldots, C_{|\sigma|}$ such that $\sigma_i \in C_i$. We say that such an algorithm* serves *σ.*

Definition 2 *An* on-line k-server algorithm *gets an initial k-server configuration C_0 and a sequence of requests $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$. The request sequence is presented element by element. Following the presentation of request σ_i, the on-line k-server algorithm selects a configuration C_i such that $\sigma_i \in C_i$. The configuration C_i does not depend on requests $\sigma_{i+1}, \ldots, \sigma_{|\sigma|}$.*

Definition 3 *A* minimal match *between two configurations C and C' is the weight of a minimum weight perfect matching in a complete bipartite graph with the points of C on the right and the points of C' on the left. The weight associated with the edge (p,q) is* $\text{dist}(p, q)$. *We denote the minimal match between the two configurations by* $\text{MM}(C, C')$.

Definition 4 *Suppose a k-server algorithm A is given a request sequence $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$ and an initial configuration C_0. Let C_i, $i = 1, \ldots, |\sigma|$, be the configurations selected by A. A* fixed

numbering *of A's servers is a labelling of the points in each* C_i, $i = 0, \ldots, |\sigma|$, *with distinct labels chosen from* $\{1, \ldots, k\}$ *such that for* $1 \leq i \leq |\sigma|$, *there exists a minimal match of* C_i *and* C_{i-1} *such that all matched pairs of points have the same label. We say that the request* σ_i *is* served by *server* s *iff* σ_i *is labelled* s *in* C_i. *We say that a server* s moves *from a point* p *to a point* q *iff* p *is labelled* s *in configuration* C_i *and* q *is labelled* s *in configuration* $C_{i'}$ *and* $i < i'$.

Definition 5 *A* k*-server algorithm* A *is* lazy, *iff for every request sequence* $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$ *and for every initial configuration* C_0 *the following holds. Let* C_i, $i = 1, \ldots, |\sigma|$, *be the configurations selected by* A. *Then, for every* i, $1 \leq i \leq |\sigma|$, *if* $\sigma_i \in C_{i-1}$, *then* $C_i = C_{i-1}$, *and if* $\sigma_i \notin C_{i-1}$, *then there exists* $p \in C_{i-1}$ *such that* $C_i = C_{i-1} \setminus \{p\} \cup \{\sigma_i\}$.

Definition 6 *The* cost *associated with a* k*-server algorithm* A *given an initial configuration* C_0 *and a request sequence* σ *is denoted by*

$$\mathrm{cost}_A(C_0, \sigma) = \sum_{i=1}^{|\sigma|} \mathrm{MM}(C_i, C_{i-1}).$$

Definition 7 *Where* $c \in \mathbb{R}$, *an on-line* k*-server algorithm* A *is said to be* c-competitive against an algorithm A' *iff for every request sequence* σ *and for every initial configuration* C_0,

$$\mathrm{cost}_A(C_0, \sigma) \leq c \cdot \mathrm{cost}_{A'}(C_0, \sigma)$$

The infimum of all such c *is called the* competitive ratio *of* A *against* A'.

Definition 8 *An on-line* k*-server algorithm* A *serving requests in a metric space* $\mathcal{M}$ *is said to be* c-competitive *iff for every* k*-server algorithm* A' *serving requests in* $\mathcal{M}$, A *is* c*-competitive against* A'. *For a metric space* $\mathcal{M}$, *an infinite sequence of algorithms* $A_1, A_2, \ldots, A_k, \ldots$, *where for every* k, $k \geq 1$, A_k *is an on-line* k*-server algorithm serving requests in* $\mathcal{M}$, *is said to be* competitive *iff there exists a function* $c(k)$ *of* k *alone such that for every* k, $k \geq 1$, A_k *is* $c(k)$*-competitive.*

Note that this definition is similar to the definition in [MMS] excluding the additive term that we disallow. Any algorithm that is competitive by this definition is also competitive according to [MMS].

Observation 1 *Let* A *be a* c*-competitive* k*-server algorithm serving a request sequence* σ *and starting at a server configuration* C_A. *Suppose a* k*-server algorithm* ADV *starts serving* σ *at a server configuration* C_{ADV}. *Then,*

$$\mathrm{cost}_A(C_A, \sigma) \leq c \cdot (\mathrm{cost}_{ADV}(C_{ADV}, \sigma) + \mathrm{MM}(C_A, C_{ADV})).$$

```
procedure initializeMin ( C , m , (A_0, ..., A_{m-1}))
begin
    current ← 0
    for j := 0 to m − 1 do begin
                advance_j ← 0
                last_j ← C
    end
    scale ← 0
end

function Min ( request ) : configuration
begin
    for j := 0 to m − 1 do begin
                next_j ← A_j's next configuration after serving request
                advance_j ← advance_j + MM(next_j, last_j)
                last_j ← next_j
    end
    if scale = 0 then scale ← min_{j=0,...,m−1} advance_j
    while advance_current > scale · m/(m−1) do begin
                current ← current + 1(mod m)
                scale ← scale · m/(m−1)
    end
    return ( next_current )
end
```

Figure 1: The Min Algorithm

Proof: Consider a new k-server algorithm ADV$'$ that given σ and C_A selects the same configurations as ADV given σ and C_{ADV} (except for the initial configuration, of course). As

$$\mathrm{cost}_A(C_A, \sigma) \leq c \cdot \mathrm{cost}_{ADV'}(C_A, \sigma)$$

and

$$\mathrm{cost}_{ADV'}(C_A, \sigma) \leq \mathrm{cost}_{ADV}(C_{ADV}, \sigma) + \mathrm{MM}(C_A, C_{ADV})$$

the observation follows. □

3 The MIN Operator

In this section, we exhibit an on-line k-server algorithm which is competitive against several other on-line k-server algorithms. We define an operator MIN over a fixed number of on-line al-

gorithms. Given $c \in I\!R$ and m on-line k-server algorithms, $A_0, \ldots, A_{m-1}$, $\mathrm{MIN}(A_0, \ldots, A_{m-1})$ is an on-line k-server algorithm which is c-competitive against A_i for every $0 \leq i \leq m-1$. [FKLMSY] show a MIN operator with $c = m$ for the special case of the uniform k-server problem. Here we show that it is always possible to construct $\mathrm{MIN}(A_0, \ldots, A_{m-1})$ with $c \leq 2em$, where e is the base of the natural logarithm.

Informally, the method used to construct MIN is as follows: Simulate $A_0, \ldots, A_{m-1}$ on the sequence of requests received $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$ and the initial configuration C_0 (wlog assume that $\sigma_1 \notin C_0$). Follow one of the algorithms, say A_i. Scale all costs so that $\min_{i=0,\ldots,m-1}\{\mathrm{cost}_{A_i}(C_0, \sigma_1)\}$ is 1. Before serving σ_j check $\mathrm{cost}_{A_i}(C_0, \sigma_1 \ldots \sigma_j)$ against a bound L, which is initially $m/(m-1)$. If $\mathrm{cost}_{A_i}(C_0, \sigma_1 \ldots \sigma_j)$ is greater than L, then multiply L by $m/(m-1)$, switch to $A_{i+1(\mathrm{mod}\ m)}$, and check the cost of this algorithm against the new value of L. Repeat until an algorithm $A_{i+\ell(\mathrm{mod}\ m)}$'s cost does not exceed the current value of the bound L. Continue by following $A_{i+\ell(\mathrm{mod}\ m)}$.

MIN is equivalent to the strategy given in [BCR] for the m-lane cow problem. The m-lane cow problem is to traverse m paths of unknown lengths and reach the end of one of them, and while doing so, traveling at most some constant times the length of the shortest path. [BCR] give a strategy for which the distance traveled during the traversal is at most $2em + 1$ times the length of the shortest path. For any A_i, define a path that traverses A_i's configurations. The length of the path between two adjacent configurations is the minimal match between them, so the total length of a path is equal to A_i's total work. A minor modification of [BCR] gives the following result:

Theorem 1 *Given m on-line k-server algorithms, $A_0, \ldots, A_{m-1}$, for serving a sequence of requests, let* $\mathrm{MIN} = \mathrm{MIN}(A_0, \ldots, A_{m-1})$. *For any request sequence σ and for any initial configuration C_0,*

$$\mathrm{cost}_{MIN}(C_0, \sigma) \leq 2em \cdot \min_{i=0,\ldots,m-1}\{\mathrm{cost}_{A_i}(C_0, \sigma)\}$$

Proof: The MIN operator uses the algorithms $A_0, \ldots, A_{m-1}$ cyclically. It follows an algorithm for some time, then switches to a configuration of a different algorithm and follows it. This is repeated until all algorithms are exhausted. When all algorithms are exhausted, we say that a *cycle* is complete. When a cycle is complete, and the end of the request sequence σ is not reached, a new cycle begins. Observe the jth cycle. The amount of work performed in the jth cycle (assuming MIN does not reach the end of σ during that cycle) is at most

$$2 \cdot \left(\frac{m}{m-1}\right)^{m(j-1)} + 2 \cdot \left(\frac{m}{m-1}\right)^{m(j-1)+1} + \cdots + 2 \cdot \left(\frac{m}{m-1}\right)^{mj-1}.$$

(The cost of switching from A_i to $A_{i+1(\mathrm{mod}m)}$ can be bounded by the cost so far of A_i plus the cost so far of $A_{i+1(\mathrm{mod}m)}$). Now, suppose that MIN reaches the end of σ during the jth

cycle. We may assume that MIN pays for the entire jth cycle. This only makes a possible increase in the cost charged for MIN. So, the total cost of MIN to serve σ is at most

$$2 \cdot \sum_{i=0}^{mj-1} \left(\frac{m}{m-1}\right)^i = 2m \left(\frac{m}{m-1}\right)^{m(j-1)} - 2m + 2.$$

If $j > 1$, since the $(j-1)$th cycle ended without reaching the end of σ, all algorithms input to MIN pay at least $(\frac{m}{m-1})^{m(j-2)}$, so the ratio between the cost of MIN and the cost of the best input algorithm is at most $2em$. If $j = 1$, the best input algorithm pays at least 1, while MIN pays at most $2 + 2 \cdot \frac{m}{m-1} + \cdots + 2 \cdot (\frac{m}{m-1})^{m-1} = 2m(\frac{m}{m-1})^{m-1} - 2m + 2$. The ratio is again at most $2em$. □

We note that a similar result can be reached using a layered graph traversal algorithm given by [PY].

4 The Expand-Contract Algorithm

In this section we show how to construct a competitive on-line k-server algorithm using competitive on-line algorithms for fewer than k servers. Our algorithm, EC (expand-contract algorithm), is defined by induction over the number of servers k. Let EC_k denote the algorithm for k servers. The basis for the induction is the greedy one-server algorithm EC_1, that for each request moves its single server to cover the request. Obviously, this algorithm is 1-competitive. Suppose that all algorithms EC_1, EC_2, ..., EC_{k-1} are defined. Following is the definition of EC_k. EC_k maintains two variables, a point x in the metric space and a non-negative real r, which define a sphere of radius r centered at x. Initially, the values of x and r are determined according to the initial configuration of the servers as follows. Choose among the server positions two points which are the farthest apart. One of the points is arbitrarily chosen as x. Twice the distance to the other point is r. EC_k uses $2k(k-1)$ algorithms, which are defined by $2k$ partitions of the sphere centered at x with radius r. For i, $1 \leq 2k$, the ith partitioning sphere has its center at x and a radius of $r \cdot \frac{i}{4k+2}$. With respect to each partitioning sphere, $k-1$ different k-server algorithms are defined. For a given sphere, for ℓ, $1 \leq \ell < k$, the ℓth algorithm partitions its servers into two sets, one of size ℓ and the other of size $k - \ell$. The size-ℓ set is used to serve requests inside the partitioning sphere. For this, EC_ℓ is used. Similarly, the size-$(k-\ell)$ set of servers and $\mathrm{EC}_{k-\ell}$ are used to serve requests outside the partitioning sphere. The initial configurations for EC_ℓ and $\mathrm{EC}_{k-\ell}$ are determined by the initial configuration of EC_k. The ℓ servers closest to x are used by EC_ℓ. The other servers are used by $\mathrm{EC}_{k-\ell}$. (ties are broken arbitrarily). EC_k runs the MIN of those $2k(k-1)$ algorithms on the request sequence. In parallel, it computes the optimal cost to serve the sequence of requests given so far starting at any initial configuration. EC_k continues to serve

requests with this MIN, until one of two special events happens: If the distance d between x and the next request is more than r, the algorithm modifies r. Its new value is set to $2d$. New $2k(k-1)$ algorithms are defined with respect to the new $2k$ partitioning spheres. EC_k restarts the MIN computation, with the initial configurations of the new $2k(k-1)$ algorithms derived from EC_k's current configuration. EC_k also restarts the computation of the optimal cost. This restart operation is call an *expand* step. The request that caused the expand step is served after the expand step is performed. The other special event happens if the optimal cost computed by EC_k exceeds $r/6$ after the optimal algorithm serves the next request. In this case, EC_k serves this request by moving all its servers to the location of the request. This is called a *contract* step. As in the expand step, EC_k restarts everything. The variable x is assigned to be the location of the request that caused the contract step. The next request to a point not covered by the algorithm's servers (i.e., any request different than the one that caused the contract step) determines the new radius r, which is taken to be twice the distance between the new x and this request. New partitioning spheres are defined. New $2k(k-1)$ algorithms are defined. MIN is restarted with EC_k's current configuration (which is all servers at the point that caused the contract step). The computation of the optimal cost is restarted.

We formally define the above-mentioned concepts:

Definition 9 *Given a metric space $\mathcal{M} = (X, \mathrm{dist})$, for $x \in X$ and $r \in \mathbb{R}$ we define the* sphere $B(x, r)$ *as the set of points $\{y \in X | \mathrm{dist}(x, y) \leq r\}$.*

Definition 10 *A* partition set $\mathcal{B}(x, r, m)$ *is the set of spheres $\{B(x, ir/(2m+2))\}_{i=1}^{m}$. In the context of some $\mathcal{B}(x, r, m)$ we use B_i as a shorthand for $B(x, ir/(2m+2))$.*

Definition 11 *We define the* optimal cost *of a request sequence σ, $\mathrm{cost}_{OPT}(\sigma)$, as*

$$\mathrm{cost}_{OPT}(\sigma) = \min_{A,C}\{\mathrm{cost}_A(C, \sigma)\},$$

where A runs over all k-server algorithms and C runs over all k-server initial configurations. (The existence of the minimum follows from the existence of lazy optimal algorithms for all C, σ; see [MMS]).

We examine optimal algorithms on request sequences $\sigma = \sigma_1 \ldots \sigma_{|\sigma|}$ of at least two requests, where $\{\sigma_i\}_{i=1}^{|\sigma|} \subset B(\sigma_1, r)$ for $r = 2 \cdot \mathrm{dist}(\sigma_1, \sigma_2)$. Let A be a k-server algorithm and C_0 be a k-server initial configuration such that $\mathrm{cost}_A(C_0, \sigma) = \mathrm{cost}_{OPT}(\sigma)$. In Lemma 1 we show that if the optimal cost of σ is less than $r/6$ then A maintains a non-trivial partitioning of its servers with respect to some $B_i \in \mathcal{B} = \mathcal{B}(\sigma_1, r, 2k)$. Some of A's servers remain in B_i while serving σ and others remain in $\overline{B_i} = X \setminus B_i$. We use the following definition in the proof of the lemma:

```
procedure ECinitial ( C_0 )
begin
1    (x, y) ← x, y ∈ C_0 that maximize dist(x, y);
     r ← 2 · dist(x, y); ϱ ← xy
2    Let {B_ℓ^i | 1 ≤ i ≤ 2k, 1 ≤ ℓ ≤ k − 1}
                be k-server algorithms defined by B(x, r, 2k)
3    initializeMin ( C_0 ,2k(k − 1), (B_ℓ^i))
end

function EC ( p ) : configuration
begin
4    ϱ ← ϱp
     {case 1}   if p ∈ EC server configuration then
5                    return ( EC current configuration )
     {case 2}   if dist(p, x) > r then                {an expand step}
6                    r ← 2 · dist(p, x)
7                    Let {B_ℓ^i | 1 ≤ i ≤ 2k, 1 ≤ ℓ ≤ k − 1} be
                          k-server algorithms defined by B(x, r, 2k)
8                    initializeMin ( EC server configuration ,2k(k − 1),(B_ℓ^i))
9                    ϱ ← xp
10                   return ( Min( p ) )
11   {case 3}   if cost_OPT(ϱ) > r/6 then             {a contract step}
12                   x ← p; r ← 0; ϱ ← x
13                   return ( All servers are at x )
14   {case 4}   return ( Min( p ) )
end
```

Figure 2: The Expand-Contract Algorithm

Definition 12 *Let $C_1, \ldots, C_{|\sigma|}$ be the configurations selected by A given σ and C_0. Given a fixed numbering of A's servers, we say that A moves server s* across *the boundary of B_i (denoted by ∂B_i) iff there exist $0 \leq i, j \leq |\sigma|$ and points $p \in B_i$, $q \notin B_i$ such that p is labelled s in C_i and q is labelled s in C_j.*

Lemma 1 *If $\mathrm{cost}_{OPT}(\sigma) < r/6$ then there exist: i, $1 \leq i \leq 2k$, ℓ, $1 \leq \ell \leq k-1$ and a fixed numbering of A's servers such that while A serves σ, all of A's configurations have points labelled $1, \ldots, \ell$ in B_i and points labelled $\ell+1, \ldots, k$ in $\overline{B_i}$.*

Proof: Let ν be the number of B_i's, $1 \leq i \leq 2k$, where A moved some server across ∂B_i. There are only k servers, so if $\nu = 2k$ then there are at least k pairs (B_i, B_{i+1}), $1 \leq i < 2k$, such that A moved the same server across both ∂B_i and ∂B_{i+1}. Therefore, $\mathrm{cost}_A(C_0, \sigma) \geq k \cdot r/(4k+2) \geq r/6$, a contradiction. So, $\nu < 2k$. Now, let $B_i \in \mathcal{B}$, $1 \leq i \leq 2k$, be a sphere such that A does not move any server across ∂B_i. The first request $\sigma_1 \in B_i$ and the second $\sigma_2 \notin B_i$. So, there exists $1 \leq \ell \leq k-1$ such that while A serves σ, ℓ servers remain in B_i and $k-\ell$ servers remain in $\overline{B_i}$. Label points occupied by servers in B_i with $1, \ldots, \ell$ and points occupied by servers in $\overline{B_i}$ by $\ell+1, \ldots, k$. □

Corollary 1 *Let C_0 be a k-server configuration such that $\sigma_1 \in C_0$. Let A be any k-server algorithm. Define a fixed numbering on A's servers given σ and C_0 by numbering points in C_0 in order of increasing distance from σ_1. If $\mathrm{cost}_A(C_0, \sigma) < r/6$ then there exist i, $1 \leq i \leq 2k$, and ℓ, $1 \leq \ell \leq k-1$, such that while A serves σ, all of A's configurations have points in B_i labelled $1, \ldots, \ell$ and points in $\overline{B_i}$ labelled $\ell+1, \ldots, k$.*

Definition 13 *Let $\mathcal{B} = \mathcal{B}(x, r, 2k)$ be a partition set. Let $B_i \in \mathcal{B}$, $1 \leq i \leq 2k$. For $\ell = 1, \ldots, k-1$, let EC_ℓ be a c_ℓ-competitive on-line ℓ-server algorithm. Define $\mathcal{B}_\ell^i$ as follows, $\mathcal{B}_\ell^i$ is an on-line k-server algorithm that numbers its servers $1, \ldots, k$, server i being the ith farthest server from the center x before $\mathcal{B}_\ell^i$ begins operation. $\mathcal{B}_\ell^i$ serves requests in B_i with servers $1, \ldots, \ell$ and requests in $\overline{B_i}$ with servers $\ell+1, \ldots, k$. It does that by simulating EC_ℓ with the servers $1, \ldots, \ell$ on the subsequence of requests in B_i, and by simulating $\mathrm{EC}_{k-\ell}$ with the servers $\ell+1, \ldots, k$ on the subsequence of requests in $\overline{B_i}$.*

We have the following lemma concerning $\mathcal{B}_\ell^i$:

Lemma 2 *Suppose a k-server algorithm A is given a request sequence σ and an initial configuration C_0. Suppose that while serving σ, A maintains a fixed numbering of its servers and serves requests in B_i with the first ℓ servers and requests in $\overline{B_i}$ with the other $k-\ell$ servers. Then, for any initial configuration C,*

$$\mathrm{cost}_{\mathcal{B}_\ell^i}(C, \sigma) \leq \max\{c_\ell, c_{k-\ell}\}(\mathrm{cost}_A(C_0, \sigma) + \mathrm{MM}_\ell(C_0, C)),$$

where $\mathrm{MM}_\ell(C_0, C)$ *is the minimal match between the points in* C_0 *labelled* $1, \ldots, \ell$ *and the points in* C *labelled* $1, \ldots, \ell$ *plus the minimal match between the points in* C_0 *labelled* $\ell+1, \ldots, k$ *and the points in* C *labelled* $\ell+1, \ldots, k$.

Proof: Follows from the competitiveness of EC_ℓ and $\mathrm{EC}_{k-\ell}$ and observation 1. □

5 Competitive Analysis

Given a request sequence σ and an initial configuration C_0, EC divides σ into subsequences $\tau^1, \ldots, \tau^d$ called *phases*. Each contract step performed by EC ends a phase, the request that caused the contract included in the phase. The end of σ also ends a phase. We use the term *complete* to refer to a phase that ends with a contract step. The last phase might be *incomplete*. We use a slightly different analysis for request sequences that end with a complete phase and for request sequences that end with an incomplete phase.

A phase τ^j is further divided by EC into subsequences $\tau^j_1, \ldots, \tau^j_{p_j}$ called *sub-phases* of τ^j. An expand step performed by EC ends a sub-phase, the request that caused the expand *not* included in the sub-phase. The end of a phase also ends a sub-phase. We use the term *complete* to refer to a sub-phase that ends with a contract step. All sub-phases in a complete phase, except the last sub-phase, are *incomplete*.

Our proof that EC is competitive follows these steps:

(a) We show that the cost of EC to serve an incomplete sub-phase is bounded from above by some value proportional to the optimal cost incurred during that sub-phase plus the radius of the sphere maintained by EC during that sub-phase. By Lemma 1, an algorithm achieving the optimal cost satisfies the conditions of Lemma 2 with respect to at least one $\mathcal{B}^i_\ell$.

(b) We prove that the optimal cost incurred during the complete sub-phase in a complete phase is an upper bound on the sum of the optimal costs on all the incomplete sub-phases in that phase. Every expand step doubles, at least, the radius of the sphere maintained by EC. Therefore, the last radius bounds the sum of all previous radii in a phase. These radii bound the optimal costs incurred during the respective sub-phases.

(c) We show that the cost of EC to serve a complete phase is bounded from above by some value proportional to the optimal cost incurred during the complete sub-phase of that phase. The cost of EC during all incomplete sub-phases of a complete phase is bounded by some value proportional to the sum of optimal costs incurred during these sub-phases plus the sum of radii of the spheres maintained by EC during these sub-phases. This sum of radii is bounded by the last radius. The complete sub-phase can be treated as an incomplete sub-phase, except for the last request, which causes a contract step. The cost

of the contract step is proportional to the last radius. A lower bound for the optimal cost incurred during the last sub-phase is proportional to the last radius.

(d) We prove that the optimal cost incurred during the complete sub-phase in a complete phase is a lower bound on the cost of any k-server algorithm to serve the entire phase. The sequence of requests for which the optimal cost is incurred during the complete sub-phase begins with one or two request locations that *must* be occupied by adversary servers at the start of the phase; the remainder is a subsequence of the requests served during the entire phase. This proves EC's competitiveness for request sequences that end with a complete phase.

(e) We conclude the proof by showing that EC is competitive for request sequences that end with an incomplete phase. We show that EC's cost during the last phase can be amortized against the adversary's cost during the entire sequence.

We use C^j to denote EC's configuration just before serving requests from phase τ^j. We use x^j to denote the center of all spheres maintained by EC during τ^j. We use C_t^j to denote EC's configuration just before serving requests from τ_t^j. We use r_t^j to denote the radius of the sphere maintained by EC during τ_t^j.

During a sub-phase τ_t^j, EC computes the minimal cost of an accumulated sequence ϱ (see Fig. 2, line 11). We use ϱ_t^j to denote the value of ϱ at the end of τ_t^j. At the end of τ_t^j, unless $t = j = 1$, $\varrho_t^j = x^j \tau_t^j$ (see Fig. 2, lines 4,9). At the end of τ_1^1, $\varrho_1^1 = x^1 y \tau_1^1$, where x^1, y define the radius of the initial sphere (see Fig. 2, line 1,4)

Let $\Delta = 4ek(k-1) \cdot \max\{c_\ell, c_{k-\ell}\}_{\ell=1}^{k-1}$.

Lemma 3 *If τ_t^j is an incomplete sub-phase, then*

$$\mathrm{cost}_{EC}(C_t^j, \tau_t^j) \leq \Delta \cdot (\mathrm{cost}_{OPT}(\varrho_t^j) + 2kr_t^j).$$

Proof: We assume at first that τ_t^j is not the first sub-phase of σ. Let $\mathcal{B} = \mathcal{B}(x^j, r_t^j, 2k)$ be the partition set during τ_t^j. Let A be a k-server algorithm and C an initial configuration such that $\mathrm{cost}_A(C, \varrho_t^j) = \mathrm{cost}_{OPT}(\varrho_t^j)$. From EC, r_t^j is twice the distance between the first two requests in ϱ_t^j (see Fig. 2, line 6). Note that τ_t^j is an incomplete sub-phase, so $\mathrm{cost}_{OPT}(\varrho_t^j) < r/6$. Therefore, by Lemma 1, there exist i, $1 \leq i \leq 2k$, a fixed numbering of A's servers and ℓ, $1 \leq \ell \leq k-1$, such that while A serves ϱ_t^j, points labelled $1, \ldots, \ell$ are in B_i and points labelled $\ell+1, \ldots, k$ are not in B_i.

A's cost is minimal, so we may assume that A is lazy and that $x^j \in C$. Therefore, without loss of generality, we may assume that A, given τ_t^j and C, constructs the same configurations as A given ϱ_t^j and C. Therefore, the conditions required by Lemma 2 hold for A given τ_t^j and C, so,

$$\mathrm{cost}_{\mathcal{B}_\ell^i}(C_t^j, \tau_t^j) \leq \max\{c_\ell, c_{k-\ell}\} \cdot (\mathrm{cost}_A(C, \tau_t^j) + \mathrm{MM}_\ell(C, C_t^j)). \quad (1)$$

If τ_t^j started after a contract step then EC has all servers at $x^j \in B(x^j, r_t^j)$ (see Fig. 2, line 13). If τ_t^j started after an expand step, then, by induction over t, EC has all servers in $B(x_{t-1}^j, r_{t-1}^j)$ initially since the request causing the expand at the end of τ_{t-1}^j is not included in τ_{t-1}^j. In both cases, $C_t^j \subset B(x^j, r_t^j)$. A's cost is minimal, so we may assume that $C \subset B(x^j, r_t^j)$. Therefore,

$$\mathrm{MM}_\ell(C, C_t^j) \leq k \cdot 2r_t^j. \tag{2}$$

EC given τ_t^j and C_t^j, follows MIN of $2k(k-1)$ algorithms including $\mathcal{B}_\ell^i$ (see Fig. 2, line 14). Therefore, by Theorem 1,

$$\mathrm{cost}_{EC}(C_t^j, \tau_t^j) \leq 2e[2k(k-1)] \cdot \mathrm{cost}_{\mathcal{B}_\ell^i}(C_t^j, \tau_t^j). \tag{3}$$

By Equations 1, 2 and 3 the lemma holds, except for τ_1^1.

To prove the correctness of the lemma for τ_1^1, we can use the same proof with the following modifications: The first two requests in ϱ_1^1 are in C_1^1 (see Fig. 2, line 1), so $\mathrm{cost}_{\mathcal{B}_\ell^i}(C_1^1, \varrho_1^1) = \mathrm{cost}_{\mathcal{B}_\ell^i}(C_1^1, \tau_1^1)$. In Equation 1 replace τ_t^j with ϱ_1^1 and C_t^j with C_1^1. The rest of the proof remains the same. □

Lemma 4 *If $\tau^j = \tau_1^j \ldots \tau_{p_j}^j$ is a complete phase, then*

$$\sum_{t=1}^{p_j-1} \mathrm{cost}_{OPT}(\varrho_t^j) \leq \mathrm{cost}_{OPT}(\varrho_{p_j}^j).$$

Proof: For $1 \leq t < p_j$, $\mathrm{cost}_{OPT}(\varrho_t^j) < r_t^j/6$; otherwise τ^j would have ended during τ_t^j. Because a contract step was performed at the end of $\tau_{p_j}^j$, $\mathrm{cost}_{OPT}(\varrho_{p_j}^j) \geq r_{p_j}^j/6$. For $1 \leq t < p_j$, $r_t^j \leq r_{t+1}^j/2$, because the expand step ending τ_t^j was performed only when the distance to the new request was at least twice the distance to the request that started τ_t^j. So,

$$\sum_{t=1}^{p_j-1} \mathrm{cost}_{OPT}(\varrho_t^j) < \sum_{t=1}^{p_j-1} r_t^j/6 \leq r_{p_j}^j/6 \leq \mathrm{cost}_{OPT}(\varrho_{p_j}^j).$$

□

Let $\Gamma = 2\Delta + 12k(2\Delta + 1)$.

Lemma 5 *If $\tau^j = \tau_1^j \ldots \tau_{p_j}^j$ is a complete phase, then*

$$\mathrm{cost}_{EC}(C^j, \tau^j) \leq \Gamma \cdot \mathrm{cost}_{OPT}(\varrho_{p_j}^j).$$

Proof: Observe the first $p_j - 1$ sub-phases. They are all incomplete. We use Lemma 3 and Lemma 4 to bound EC's cost on these sub-phases. By Lemma 3,

$$\sum_{t=1}^{p_j-1} \text{cost}_{EC}(C_t^j, \tau_t^j) \leq \Delta \sum_{t=1}^{p_j-1} (\text{cost}_{OPT}(\varrho_t^j) + 2kr_t^j) \tag{4}$$

By Lemma 4,

$$\sum_{t=1}^{p_j-1} \text{cost}_{OPT}(\varrho_t^j) \leq \text{cost}_{OPT}(\varrho_{p_j}^j). \tag{5}$$

Note that for every $1 \leq t < p_j$, $r_t^j \leq r_{t+1}^j/2$ (see Fig. 2, line 6). So,

$$\sum_{t=1}^{p_j-1} 2kr_t^j \leq 2kr_{p_j}^j, \tag{6}$$

Equations 4, 5 and 6 give:

$$\sum_{t=1}^{p_j-1} \text{cost}_{EC}(C_t^j, \tau_t^j) \leq \Delta\text{cost}_{OPT}(\varrho_{p_j}^j) + \Delta 2kr_{p_j}^j. \tag{7}$$

Now we handle the last sub-phase, $\tau_{p_j}^j$, which is a complete sub-phase. Observe that without the last request, this sub-phase is incomplete. We use τ to denote the sequence of requests derived from $\tau_{p_j}^j$ by removing the last request. The cost of EC to serve τ can be bounded by the optimal cost incurred during τ (incurred for the sequence ϱ, equals $\varrho_{p_j}^j$ without the last request) using Lemma 3. The additional cost of EC to serve the last request of $\tau_{p_j}^j$ is the cost of a contract step. This cost is bounded by the maximal distance of k servers from the request causing the contract. More formally, $\text{cost}_{OPT}(\varrho) < r_{p_j}^j/6$ (see Fig. 2, line 11), so by Lemma 3,

$$\text{cost}_{EC}(C_{p_j}^j, \tau) \leq \Delta \cdot (\text{cost}_{OPT}(\varrho) + 2kr_{p_j}^j). \tag{8}$$

The cost of the contract step performed by EC to serve the last request in τ^j is at most $2kr_{p_j}^j$, because all of EC's servers are in $B(x^j, r_{p_j}^j)$ while performing the contract step. The request sequence ϱ is a prefix of the request sequence $\varrho_{p_j}^j$, so $\text{cost}_{OPT}(\varrho) \leq \text{cost}_{OPT}(\varrho_{p_j}^j)$. Therefore,

$$\text{cost}_{EC}(C_{p_j}^j, \tau_t^j) \leq \Delta\text{cost}_{OPT}(\varrho_{p_j}^j) + (\Delta + 1) \cdot 2kr_{p_j}^j. \tag{9}$$

From Equations 7 and 9, EC's cost over all sub-phases in τ^j is

$$\begin{aligned}
\sum_{t=1}^{p_j} \text{cost}_{EC}(C_t^j, \tau_t^j) &= \sum_{t=1}^{p_j-1} \text{cost}_{EC}(C_t^j, \tau_t^j) + \text{cost}_{EC}(C_{p_j}^j, \tau_{p_j}^j) \\
&\leq \Delta\text{cost}_{OPT}(\varrho_{p_j}^j) + \Delta 2kr_{p_j}^j + \Delta\text{cost}_{OPT}(\varrho_{p_j}^j) + (\Delta + 1) \cdot 2kr_{p_j}^j \\
&= 2\Delta\text{cost}_{OPT}(\varrho_{p_j}^j) + (2\Delta + 1) \cdot 2kr_{p_j}^j.
\end{aligned}$$

To complete the proof we must get rid of the additive term $(2\Delta+1)\cdot 2kr^j_{p_j}$. The last sub-phase ends with a contract step, so $r^j_{p_j} \le 6\cdot \mathrm{cost}_{OPT}(\varrho^j_{p_j})$. Thus,

$$(2\Delta+1)\cdot 2kr^j_{p_j} \le 12k(2\Delta+1)\cdot \mathrm{cost}_{OPT}(\varrho^j_{p_j}).$$

It follows that

$$\begin{aligned}\mathrm{cost}_{EC}(C^j,\tau^j) &= \sum_{t=1}^{p_j}\mathrm{cost}_{EC}(C^j_t,\tau^j_t)\\ &\le 2\Delta\mathrm{cost}_{OPT}(\varrho^j_{p_j}) + (2\Delta+1)\cdot 2kr^j_{p_j}\\ &\le 2\Delta\mathrm{cost}_{OPT}(\varrho^j_{p_j}) + 12k(2\Delta+1)\cdot \mathrm{cost}_{OPT}(\varrho^j_{p_j})\\ &\le (2\Delta+12k(2\Delta+1))\cdot \mathrm{cost}_{OPT}(\varrho^j_{p_j}).\end{aligned}$$

□

Lemma 6 *If $\tau^j = \tau^j_1\ldots\tau^j_{p_j}$ is a complete phase, then for any k-server algorithm ADV that serves all of σ and starts in C_O,*

$$\mathrm{cost}_{OPT}(\varrho^j_{p_j}) \le \mathrm{cost}_{ADV}(C_{ADV},\tau^j),$$

where C_{ADV} is ADV's configuration just before serving requests from τ^j.

Proof: We assume at first that if $j=1$ then $p_j \ne 1$. If $j=1$ then x^1 is a point in the initial configuration $C_0 = C_{ADV}$. If not, x^j is the request that ended τ^{j-1}. In either case, C_{ADV} must have a server located at x^j. Let A be a k-server algorithm that given the sequence $x^j\tau^j$ and the initial configuration C_{ADV}, moves to configuration C_{ADV} in order to serve x^j and then chooses the configurations selected by ADV for τ^j. Since $x^j \in C_{ADV}$, $\mathrm{cost}_A(C_{ADV}, x^j\tau^j) = \mathrm{cost}_{ADV}(C_{ADV},\tau^j)$. Let $\tau = \tau^j_1\ldots\tau^j_{p_j-1}$ be the concatenation of τ^j's incomplete sub-phases. As $x^j\tau^j = x^j\tau\tau^j_{p_j}$, it follows that $\mathrm{cost}_{OPT}(\varrho^j_{p_j}) \le \mathrm{cost}_{OPT}(x^j\tau^j)$ (note that $\varrho^j_{p_j} = x^j\tau^j_{p_j}$). Therefore,

$$\mathrm{cost}_{OPT}(\varrho^j_{p_j}) \le \mathrm{cost}_{OPT}(x^j\tau^j) \le \mathrm{cost}_A(C_{ADV},x^j\tau^j) = \mathrm{cost}_{ADV}(C_{ADV},\tau^j).$$

To complete the proof for the case where $j=p_j=1$, note that $\varrho^1_1 = x^1y\tau^1_1$, where y is one of the two points defining the radius. during τ^1_1 (see Fig. 2, line 1). C_{ADV} contains both x^1 and y, so we may define A that serves ϱ^1_1 with the same cost as ADV incurs in serving $\tau^1 = \tau^1_1$. The rest of the proof remains the same. □

Lemma 7 *If all phases defined by running EC on a request sequence σ and an initial configuration C_0 are complete, then for any k-server algorithm ADV,*

$$\text{cost}_{EC}(C_0, \sigma) \leq \Gamma \cdot \text{cost}_{ADV}(C_0, \sigma).$$

Proof: Let D^j be ADV's configuration just before serving τ^j. By lemma 6,

$$\text{cost}_{OPT}(\varrho^j_{p_j}) \leq \text{cost}_{ADV}(D^j, \tau^j).$$

By lemma 5,

$$\text{cost}_{EC}(C^j, \tau^j) \leq \Gamma \cdot \text{cost}_{OPT}(\varrho^j_{p_j}),$$

so

$$\text{cost}_{EC}(C^j, \tau^j) \leq \Gamma \cdot \text{cost}_{ADV}(D^j, \tau^j).$$

Therefore,

$$\begin{aligned} \text{cost}_{EC}(C_0, \sigma) &= \sum_{j=1}^{d} \text{cost}_{EC}(C^j, \tau^j) \\ &\leq \sum_{j=1}^{d} \Gamma \cdot \text{cost}_{ADV}(D^j, \tau^j) \\ &\leq \Gamma \cdot \text{cost}_{ADV}(C_0, \sigma). \end{aligned}$$

□

Lemma 7 proves that EC is competitive for all request sequences that end with a complete phase. We conclude the proof of EC's competitiveness by handling the case where a request sequence ends with an incomplete phase.

Lemma 8 *If the last phase defined by running EC on a request sequence σ and an initial configuration C_0 is incomplete, then for any k-server algorithm ADV,*

$$\text{cost}_{EC}(C_0, \sigma) \leq 4\Gamma \cdot \text{cost}_{ADV}(C_0, \sigma).$$

Proof: Let $\tau^d = \tau^d_1 \ldots \tau^d_{p_d}$ be the last phase. For every t, $1 \leq t \leq p_d$, τ^d_t is an incomplete sub-phase. By Lemma 3,

$$\begin{aligned} \text{cost}_{EC}(C^d, \tau^d) &= \sum_{t=1}^{p_d} \text{cost}_{EC}(C^d_t, \tau^d_t) \\ &\leq \Delta \cdot \sum_{t=1}^{p_d} (\text{cost}_{OPT}(\varrho^d_t) + 2kr^d_t). \end{aligned} \tag{10}$$

All sub-phases of τ^d are incomplete, so for every t, $1 \leq t \leq p_d$, $\text{cost}_{OPT}(\varrho_t^d) < r_t^d/6$. Note that for every t, $1 \leq t < p_d$, $r_t^d \leq r_{t+1}^d/2$, so

$$\sum_{t=1}^{p_d} \text{cost}_{OPT}(\varrho_t^d) \leq r_{p_d}^d/3, \tag{11}$$

and

$$\sum_{t=1}^{p_d} r_t^d \leq 2r_{p_d}^d. \tag{12}$$

By Equations 10, 11 and 12 we have

$$\text{cost}_{EC}(C^d, \tau^d) \leq (4k + \frac{1}{3}) \cdot \Delta \cdot r_{p_d}^d.$$

We consider four cases:

1. $d > 1$ and $r_{p_d}^d \leq 2r_{p_1}^1$.

2. $d > 1$ and $r_{p_d}^d > 2r_{p_1}^1$.

3. $d = 1$ and $p_1 \neq 1$.

4. $d = p_d = 1$.

Case 1: If $r_{p_d}^d \leq 2r_{p_1}^1$ then consider the following: τ^1 is a complete phase, so $\text{cost}_{OPT}(\varrho_{p_1}^1) \geq r_{p_1}^1/6$. We charge EC's cost during τ^d against ADV's cost during $\tau_{p_1}^1$, which must be at least $r_{p_1}^1/6$. At most $12(4k + \frac{1}{3}) \cdot \Delta$ is added to the competitive ratio. Now, $12(4k + \frac{1}{3}) \cdot \Delta < 3\Gamma$, so the competitive ratio is bounded by 4Γ in this case.

Case 2: If $r_{p_d}^d > 2r_{p_1}^1$ then consider the following: Before serving σ, the maximal distance between two servers of ADV was $r_1^1/2 \leq r_{p_1}^1/2$. While ADV served σ, it had to place a server at x^d and to place a server at a distance of $r_{p_d}^d/2$ from x^d (not necessarily both in the same configuration). Therefore, ADV's cost to serve σ is at least $r_{p_d}^d/2 - r_{p_1}^1/2 > r_{p_d}^d/4$. We charge EC's cost during τ^d against ADV's cost during the entire sequence. This increases the competitive ratio by at most $4(4k + \frac{1}{3}) \cdot \Delta < \Gamma$, so 4Γ is a bound for the competitive ratio in both cases.

Case 3: If $\tau_{p_1}^1$ started with an expand step then ADV moved a distance of at least $r_{p_1}^1/4$ during τ^1, because the farthest server from x^1 in C_0 is at a distance of $r_1^1/2 \leq r_{p_1}^1/4$ from x^1. We charge EC's cost during τ^1 against ADV's cost during τ^1. The competitive ratio remains bounded by 4Γ.

Case 4: If τ^1 consists of a single incomplete sub-phase τ_1^1 then there are two cases: If $\text{cost}_{ADV}(C_0, \tau_1^1) \geq r_1^1/6$ then we charge EC's cost during τ^1 against ADV's cost during τ^1.

The competitive ratio remains bounded by 4Γ. If $\text{cost}_{ADV}(C_0, \tau_1^1) < r_1^1/6$ then ADV satisfies the conditions of Lemma 2, so there exists some $\mathcal{B}_\ell^i$ such that

$$\text{cost}_{\mathcal{B}_\ell^i}(C_0, \tau_1^1) \leq \max\{c_\ell, c_{k-\ell}\} \cdot (\text{cost}_{ADV}(C_0, \tau_1^1) + \text{MM}_\ell(C_0, C_0)).$$

$\text{MM}_\ell(C_0, C_0) = 0$, because EC's servers are numbered according to their distance from x^1 in C_0. The competitive ratio in this case is

$$4ek(k-1) \cdot \max\{c_\ell, c_{k-\ell}\} \leq \Delta < 4\Gamma.$$

□

The claims above have the following consequence:

Theorem 2 *EC is a $2^{O(k \log k)}$-competitive k-server algorithm.*

Proof: Let c_k denote the competitive ratio for the k-server EC algorithm. From Lemmas 7 and 8,

$$\text{cost}_{EC}(C_0, \sigma) \leq 4\Gamma \cdot \text{cost}_{ADV}(C_0, \sigma).$$

Therefore,

$$\begin{aligned} c_k &\leq 4\Gamma \\ &= 4(2\Delta + 12k(2\Delta + 1)) \leq 152k\Delta \\ &\leq 1824k^3 \max\{c_\ell, c_{k-\ell}\}_{\ell=1}^{k-1} = 1824k^3 c_{k-1} \end{aligned}$$

Solving this recurrence, with the initial condition $c_1 = 1$, gives $c_k = 2^{O(k \log k)}$. □

6 Acknowledgments

We are very grateful to Howard Karloff and Sundar Vishwanathan for their comments. We thank the two referees for the excellent and very useful reports they submitted.

References

[BBKTW] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the Power of Randomization in On-line Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 379–386, May 1990.

[BKT] P. Berman, H.J. Karloff, and G. Tardos. A Competitive Three-Server Algorithm. In *Proc. of the 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 280–290, January 1990.

[BLS] A. Borodin, N. Linial, and M. Saks. An Optimal On-line Algorithm for Metrical Task Systems. In *Proc. of the 19th Ann. ACM Symp. on Theory of Computing*, pages 373–382, May 1987.

[BCR] R.A. Baeza-Yates, J.C. Culberson, and G.J.E. Rawlins. Searching with Uncertaincy. Technical report, University of Waterloo, October 1987.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random Walks on Weighted Graphs and Applications to On-line Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 369–378, May 1990.

[CKPV] M. Chrobak, H.J. Karloff, T. Payne, and S. Vishwanathan. New Results on Server Problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[CL1] M. Chrobak and L. Larmore. An Optimal On-line Algorithm for the Server Problem on Trees. *SIAM Journal of Computing*, 20:144–148, 1991.

[CL2] M. Chrobak and L. Larmore. On Fast Algorithms for Two Servers. In *Proc. Mathematical Foundations of Computer Science*, Banská Bystrica, 1990. Also, to appear in *Journal of Algorithms*.

[FFKRRV] A. Fiat, D.P. Foster, H.J. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive Algorithms for Layered Graph Traversal. In *Proc. of the 32nd Ann. Symp. on Foundations of Comp. Sci.*, pages 288–297, October 1991.

[FKLMSY] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive Paging Algorithms. To appear in *Journal of Algorithms.*

[FRRS] A. Fiat, Y. Rabani, Y. Ravid, and B. Schieber. A Deterministic $O(k^3)$-Competitive k-Server Algorithm for the Circle. Manuscript.

[Gro] E. Grove. The Harmonic k-Server Algorithm is Competitive. In *Proc. of the 23nd Ann. ACM Symp. on Theory of Computing*, pages 260–266, May 1991.

[IR] S. Irani and R. Rubinfeld. A Competitive 2-Server Algorithm. *Information Processing Letters*, 39:85–91, 1991.

[Kar] R.M. Karp. A Randomized $(n+1)k$ Competitive Algorithm on the Graph. Personal communication, 1989.

[KMMO] A.R. Karlin, M.S. Manasse, L.A. McGeoch, and S. Owicki. Competitive Randomized Algorithms for Non-uniform Problems. In *Proc. of the 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 301–309, January 1990.

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

[MMS] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive Algorithms for On-line Problems. *Journal of Algorithms*, 11:208–230, 1990. Earlier version in *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 322–333, May 1988.

[MS] L.A. McGeoch and D.D. Sleator. A Strongly Competitive Randomized Paging Algorithm. Submitted to *Algorithmica*.

[PY] C.H. Papadimitriou and M. Yannakakis. Shortest Paths Without a Map. In *Proc. 16th ICALP*, pages 610–620, July 1989.

[Rag] P. Raghavan. Lecture Notes on Randomized Algorithms. Technical report, IBM Yorktown, September 1990.

[RS] P. Raghavan and M. Snir. Memory versus Randomization in On-Line Algorithms. In *Proc. 16th ICALP*, July 1989. To appear in Springer-Verlag Lecture Notes in Computer Science 372.

[ST] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communication of the ACM*, 28:202–208, 1985.

[Tur] G. Turpin. Recent Work on the Server Problem. Master's thesis, University of Toronto, 1989.