The K-Server Problem with Distinguishable Servers

Lee Newberg[1]

May 23, 1991

[1]Supported by a National Science Foundation Graduate Fellowship.

Abstract

This report gives a survey of existing results in the field of on-line k-server algorithms and presents some new findings. The survey includes optimal on-line algorithms for k servers on a line or a tree, an optimal on-line algorithm for 2 servers in any metric space, and an optimal on-line algorithm for $n-1$ servers in a metric space with n points.

The first part of the newer material pertains to the traditional k-server problem. The equivalence of the Tree and Line Potential Functions is discussed. An algorithm is given that works in any finite metric space with a competitiveness that is independent of the distances between the points.

The second part of the newer material discusses a variation of the k-server problem in which the servers have different costs. In this model each server pays a cost proportional to the distance it moves but the constant of proportionality may vary from server to server. Upper and lower bounds on the competitiveness are given for the k-server problem on a uniform metric space and for the general 2-server problem.

Contents

1 Introduction

This report discusses results in the field of on-line k-server algorithms.[1] An *on-line* problem is one in which an algorithm must handle a sequence of requests, satisfying each request without knowledge of the future requests. The goal is to minimize the cost associated with serving the request sequence. An algorithm for deciding where to place files on a disk is an example of an on-line algorithm. A memory manager that must decide which pages of information to put in memory is another example.

Another on-line problem involves the use of k mobile servers in a *metric space*.[2] In this case, each request is a location. To serve a request, the algorithm must move one of the k servers from its current location to the requested site. The cost associated with the request is the distance the server moves. This is known as the k-server problem.

The memory manager problem is an example of the k-server problem. In this case the servers are blocks of physical memory and the requests are for pages of virtual memory. Moving a server from one page to another corresponds to removing the first page from memory and replacing it with the requested page. The distance between two pages corresponds to the cost of such an operation. In this case the cost is independent of the two pages involved so the distance between any two points in the metric space is a constant.

In the traditional k-server problem the servers are indistinguishable. The cost of satisfying a request is a function of where the server is moving from and where it is moving to, but not which server it is. When this report discusses the *k-server problem* that is what it will mean.

However, there is no reason the k servers must be indistinguishable. It is possible, for instance, that the memory manager may control more than one kind of physical memory. Even in a non-hierarchical memory it may be responsible for managing both fast SRAM memory and slower DRAM memory. In this case the cost of replacing a page depends on the server (i.e. block of physical memory). A problem of this type will

[1] The word algorithm as used in this report means deterministic algorithm.

[2] Italicized words appear in the glossary

be an instance of the *k-distinguishable-server problem*. This paper discusses both the traditional k-server problem and the newer k-distinguishable-server problem.

Research in the field of on-line k-server algorithms involves finding good algorithms and proving the existence or non-existence of these good algorithms. Two basic concepts that allow us to talk sensibly about the performance of an on-line algorithm are *optimal* and *competitive*.

For any given sequence of requests there will be algorithms which achieve the least cost when serving that sequence. An algorithm OPT is called *optimal* if for every other algorithm ALG and every sequence σ the cost for OPT on σ is less than or equal to the cost of ALG on σ. An algorithm is called *competitive* if its performance on any sequence of requests is within a constant factor of the performance of an optimal algorithm on that sequence of requests. An algorithm is *c-competitive* if that constant is c. The lower the value of the *competitive ratio*, c, the more competitive an algorithm is. An optimal algorithm is itself 1-competitive.

An *optimal on-line* algorithm is one which achieves the best competitive ratio among all on-line algorithms. For most interesting problems, the optimal algorithms are not on-line algorithms. It is usually necessary to know the entire sequence of future requests in order to compute a best way to satisfy the current request. In such cases the optimal on-line algorithms are not optimal among all algorithms. For instance, it is shown in [MMS88] that the best on-line solutions to the k-server problem cannot be better than k-competitive. This proof is given in Section 2. An adaptation of the proof to the k-distinguishable-server problem is given in Section 6.

There are two main parts to this paper. Part I covers k-server problems where the cost to move a server is independent of the server. Part I is predominantly a survey of existing results for the on-line k-server problem and serves as an introduction to Part II.

Section 3 discusses the k-server problem on uniform metric spaces. A uniform metric space is one in which the distance between any two points is unity.

Section 4.2 covers the algorithm presented in [CKPV90] for k servers on a line. We give the algorithm for k servers on a tree from [CL89a] in Section 4.3. To analyze their algorithms both papers use *potential functions* which we describe in Section 4.1. Although the potential functions used in the two papers appear different they are in fact equivalent. This is explained in Section 4.4.

Next in the survey of previously solved problems is the general algorithm from [CL89b] for 2 servers in any space. Our slightly simplified version of their proof is presented in Section 4.5. The potential function used in the proofs is the same as that used for the proofs for the line space and tree space. The rough equivalence of these three problems is exploited in Part II when the spaces are revisited in the context of the k-distinguishable-server problem.

The survey ends with a presentation of an optimal on-line algorithm for $n-1$ servers in a space with n points. The algorithm is from [MMS88] though the proof is new. This is in Section 5.1.

Part I is not all introduction, however. Section 5.2 presents a new competitive algorithm for the k-server problem on a finite space. The algorithm is a generalization of the $(n-1)$-server algorithm given in Section 5.1. The competitive ratio for the algorithm is shown to be $\binom{n}{k} - 1$ where n is the number of points in the space and k is the number of servers.

The algorithm and supporting proofs show a strong similarity to those for solving metrical task systems given in [BLS87] although the results were discovered independently.

Part II covers k servers with different costs. Each of the results from Part I is revisited but the possibility of servers with distinguishable costs is considered.

The algorithms that perform well when the servers are indistinguishable also perform well when the servers are distinguishable. With almost no modification to the original proofs we can prove a respectable competitive ratio for these algorithms. A striking result is that we can prove better competitive ratios for these algorithms by modifying the potential functions employed. These better competitive ratios are achieved without modifying the behavior of the algorithms.

Part I
Equal Costs

2 Minimum Competitiveness

There is an excellent proof presented in [MMS88] that shows that an on-line algorithm for the k-server algorithm on a space with at least $k+1$ points cannot be better than k-competitive. However, before proving this lower bound on competitiveness we'll need some definitions and a few observations.

Let ALG be an algorithm and suppose there is a request to a location x. We call ALG's response to the request *reasonable* if either

1. ALG already has a server at x and does not move any of its servers, or
2. ALG does not have a server at x and moves one server to that location, but does not move any of its other servers.

We call ALG *reasonable* if every response it makes is reasonable. In general, an algorithm must always move a server to a requested location if it does not have one there already, but it might also move other servers to other locations if it so desired.

Notice that any algorithm can be easily modified to be a reasonable algorithm without raising its competitive ratio. In response to a request, the modified algorithm would still move the server that is moving reasonably. However, rather than moving the servers that aren't moving to the requested location, the modified algorithm can simply remember that it wanted to move them. The current request will still be served so no harm is done in the short run.

When it becomes reasonable to move a server then the algorithm can trace out all the remembered movements for that server. This modified algorithm does exactly as much work as the original algorithm except that it delays it until the work is reasonable. After any given request the modified algorithm has done no more work than the original algorithm so its competitive ratio must be at least as good. In fact, there is no reason for a server to trace out the entire path of saved movements. It can go straight to the requested location. This will be no more expensive than following the saved path because of the *triangle inequality.*

Thus, every algorithm can be converted to a reasonable algorithm without increasing its cost. Therefore, when minimizing a cost over all algorithms it is sufficient to minimize over only those algorithms that are reasonable.

In addition to assumptions about an algorithm's reasonableness, we may want to modify what initial conditions it assumes. When comparing two algorithms it is natural to assume that both have to start with their servers in the same places so that neither has an unfair advantage over the other. A close examination of the definition of the competitive ratio shows that this assumption is not necessary.

If an algorithm wishes to start with its servers in a configuration other than that dictated in the statement of the problem, it can simply move its servers into the desired configuration before satisfying the first request. This extra moving will incur a cost but the cost is a constant independent of the request sequence. Therefore the cost can be hidden in the additive constant, a, present in the definition of c-competitiveness. Thus an algorithm that gets to start in a configuration of its choosing has a competitive ratio no better than the equivalent algorithm that must start in the specified starting configuration.

We need some notation for handling sequences of requests. This report will use σ to indicate a sequence of requests. σ_i will mean the first i requests of the sequence σ. Contrastingly, the symbol $\sigma(i)$ will be used to indicate the ith request of a sequence σ. $C_{\mathrm{ALG}}(\sigma_i)$ will be used to mean the total cost incurred by an algorithm ALG for servicing the first i requests of a sequence σ. Similarly, $C_{\mathrm{OPT}}()$ will be used to mean the cost for an optimal algorithm.

We need one more idea before we can embark on the proof that on-line algorithms for the k-server problem cannot be better than k-competitive. We need to introduce the important concept of an *adversary.* An algorithm, ALG, will have a competitive ratio of c if there exists a constant, a, such that for every sequence of requests, σ, we

have that $C_{\mathrm{ALG}}(\sigma) \leq cC_{\mathrm{OPT}}(\sigma) + a$. Note that although there may be some sequences for which ALG performs cheaply, the competitive ratio will be bounded from below by the sequences which cause ALG to perform the worst.

We can imagine that there is an adversary that chooses a sequence of requests to demonstrate the worst performance of an on-line algorithm. The adversary will have its own copy of the servers and will show how to serve the sequence of requests more efficiently than the on-line algorithm does. Thus, proving that the competitive ratio of an on-line algorithm is c is equivalent to proving that the algorithm achieves a competitive ratio of c even against its worst adversary. This viewpoint will prove extremely useful in many proofs.

Note that the on-line algorithm cannot see where the adversary's servers are or how they move. They are imaginary and exist only to those reading the proofs.

We are now prepared to prove the theorem from [MMS88].

Theorem 1 *Let ALG be an on-line k-server algorithm for a metric space with at least $k+1$ points. Then, the competitive ratio of ALG is at least k.*

Proof: From observations made previously we know that we can assume that ALG is reasonable. Furthermore we can assume that its k servers start in different places. Choose a subspace of our metric space with those k positions plus one more, chosen arbitrarily. These $k+1$ positions will be a sufficiently big space to prove the theorem. Label the locations by the numbers $1, ..., k+1$ and let d_{ij} stand for the distance between location i and location j.

We will prove the theorem by showing that there is an adversary who can choose a sequence of requests that will cost ALG at least k times as much as an optimal algorithm. The adversary's choice is quite easy. At each step, the adversary requests a move to the location not covered by one of ALG's servers.

The cost for ALG on such a sequence is given by:

$$C_{\mathrm{ALG}}(\sigma_n) = \sum_{i=1}^{n} d_{\sigma(i+1)\sigma(i)}$$

We know that the adversary will choose the $(i+1)$st request to be the location vacated by ALG as it served the ith request. Therefore we know that ALG's ith move was from $\sigma(i+1)$ to $\sigma(i)$ and that ALG paid $d_{\sigma(i+1)\sigma(i)}$ for this move. Hence, we get the above formula.

To show that ALG cannot be better than k-competitive we must show that there is an algorithm that can serve the sequence σ_n for $1/k$th the cost. To do this we shall demonstrate k algorithms and show that at least one of them will incur a cost of no more than

$$\frac{1}{k} C_{\mathrm{ALG}}(\sigma_n)$$

when serving σ_n.

The k algorithms will behave the same and will only differ in where they start their servers. There are $\binom{k+1}{k} = k+1$ ways to choose k starting positions for servers from the $k+1$ available locations. Of these, k will have a server on $\sigma(1)$. These k possibilities will be the starting positions for the k algorithms we are constructing.

If an adversary already has a server at the requested location then the adversary will do nothing. If the request, $\sigma(n)$, is not occupied by one of the adversary's servers, the adversary will move the server currently located at $\sigma(n-1)$.

Note that after a request, $\sigma(n)$, each of the k algorithms will have a different configuration of servers. That is, the set of locations occupied by servers will be different for each of the k algorithms. To see this, notice that it is true before the first request.

For the purpose of induction, assume that is true up through the request $\sigma(n-1)$. Let's compare two of the algorithms. Before the request to $\sigma(n)$ they have different server configurations. If they both have servers at location $\sigma(n)$ then neither will move any servers to satisfy the nth request. Therefore they will have distinct configurations after the nth request. Similarly, if neither algorithm has a server at $\sigma(n)$ then both will move the server currently at $\sigma(n-1)$. Hence their server configurations will remain distinct.

Suppose one of the algorithms has a server at $\sigma(n)$ but the other does not. Since both algorithms

just served a request at $\sigma(n-1)$ they both have a server there. To satisfy the request to $\sigma(n)$ exactly one of the algorithms will move a server from $\sigma(n-1)$. The other will leave a server at that location. Therefore the server configurations will be different.

We're trying to show that one of these k algorithms will run cheaply. To demonstrate this let's add up the total cost we'd have to pay if we ran all k algorithms simultaneously. There is no cost to serve the first request since all k algorithms already have a server at $\sigma(1)$.

After $n-1$ requests each of the k algorithms will have a server at $\sigma(n-1)$. Since each algorithm has its servers in a different configuration and since no algorithm will ever have two servers at the same location, we know exactly what configurations of servers the algorithms will have. All but one of the algorithms will already have a server at $\sigma(n)$. The algorithm that must move will incur a cost of $d_{\sigma(n-1)\sigma(n)}$. Thus the total cost to run all k algorithms on the sequence σ_n is

$$\sum_{i=2}^{n} d_{\sigma(i-1)\sigma(i)}$$

Since $d_{ij} = d_{ji}$ this is the same cost incurred by ALG except for the absence of one term. Since the sum of the costs of the k algorithms is not more than $C_{\mathrm{ALG}}(\sigma_n)$ it must be true that at least one of the algorithms costs no more than $C_{\mathrm{ALG}}(\sigma_n)/k$.

We have just shown that there exists a sequence of requests, σ_n, such that the cost for ALG on that sequence is at least k times the cost of another algorithm. Therefore ALG must be at least k times more expensive than an optimal algorithm and hence cannot have a competitive ratio less than k. ■

Notice that this proof does not use the triangle inequality. Hence the theorem holds for any "space" M even if the distance function does not satisfy the triangle inequality.

At this time we are not aware of any results proving a better lower bound for the competitive ratio that is applicable to all metric spaces. To the best of our knowledge, even if attention is restricted to algorithms for servers in a specific metric space no better lower bound is known.

3 Uniform Metric Spaces

There are some metric spaces on which the k-server problem can be solved by an on-line algorithm which is k-competitive. These include uniform metric spaces, lines, and trees. There is a k-competitive algorithm for the k-server problem in a space with $k+1$ points. Additionally, there is an algorithm which is 2-competitive for the 2-server problem that works in any metric space.

However, for many instances of the k-server problem there is no known k-competitive on-line algorithm. At this time there is no k-competitive algorithm for the k-server problem in a general metric space when $k > 2$. Until 1990 there weren't any known competitive algorithms for general metric spaces when $k > 3$. Competitive algorithms are now known (see [FRR90] and [G91]) though the competitive ratios are exponential in k.

One class of spaces in which optimal on-line algorithms are known is the set of uniform metric spaces. A *uniform metric space* is a space in which the distance between any two distinct points is 1. There are several optimal on-line algorithms for these spaces. For instance, see [CKPV90] and [CL89a].

One algorithm which is particularly simple is called the LRU (for *least recently used*) algorithm. If a request is made to a point already covered by a server then LRU does nothing. Otherwise LRU moves the least recently moved server.

Theorem 2 *LRU is k-competitive.*

Proof: Without loss of generality we will assume that the adversary never requests a point in which one of LRU's servers is located. We will also assume that the adversary is reasonable.

The cost to run LRU is given by

$$C_{\mathrm{LRU}}(\sigma_n) = n$$

since each move costs exactly 1. In particular, LRU will pay k in any run of k consecutive requests. We will show that an adversary must pay at least 1 in any run of k consecutive requests except possibly the very first run. This implies that LRU is k-competitive.

Consider the k requests $\sigma(n+1), \ldots, \sigma(n+k)$ when $n \geq 1$. For the purposes of contradiction

suppose that the adversary incurs no cost during the run. Since LRU has k servers and always moves the least recently moved server we know that LRU will move each of its servers exactly once during this run. Furthermore, because the adversary always requests a location in which we have no server, we know that every request will be distinct from the the k requests preceding it. We can conclude that no two of the requests $\sigma(n), \ldots, \sigma(n+k)$ will be to the same location because each must be different from those that precede it.

Since both the adversary and LRU are reasonable we know that after the request to $\sigma(n)$ they will each have a server at that location. Since the adversary serves the entire run of requests for free, it must also have servers at the k requested sites $\sigma(n+1), \ldots, \sigma(n+k)$. Therefore the adversary must have servers at $k+1$ distinct locations.

Since the adversary has only k servers we have established a contradiction. Thus we have shown that the adversary must pay more than nothing and therefore at least 1 during any run of k requests. Hence LRU is k-competitive. ■

4 Lines and Trees

Optimal on-line algorithms are known for the k-server problem when the metric space is a line or a tree. The line and tree problems are quite similar and are similar to the general 2-server problem as well. In the following sections we demonstrate the similarities and present optimal on-line algorithms.

4.1 The Potential Function

For the proofs of the competitiveness of the algorithms for the line and the tree problems we will use a new tool. We will use a *potential function* to prove how competitive our algorithm is. A function Φ is a *potential function* demonstrating a competitive ratio of c for an algorithm ALG if it satisfies the following conditions:

- Φ is nonnegative.
- Every response to a request by the adversary increases Φ by no more than c times the cost charged to the adversary for that response.
- Every response to a request by ALG decreases Φ by at least the cost charged to ALG for that response.

Generally Φ will be a function of the current position of all of the adversary's and all of ALG's servers as well as any internal state ALG may wish to store. When working with potential functions it is often convenient to assume that the adversary will move first in response to its own request followed by ALG's response to the request.

Suppose we are trying to prove that some algorithm ALG is c-competitive. We can reduce the task to that of finding a potential function Φ satisfying the above conditions.

Theorem 3 *Let ALG be an algorithm. If there exists a potential function Φ satisfying the above constraints for a number $c > 0$ then ALG is c-competitive.*

Proof: We will show that ALG is c-competitive by showing that

$$C_{\mathrm{ALG}}(\sigma) + \Phi(\sigma) \leq cC_{\mathrm{ADV}}(\sigma) + \Phi_0$$

for any sequence σ and adversary ADV. We use Φ_0 to indicate the value of the potential function before any moves are made by ALG or ADV. The value of Φ after ALG and ADV have satisfied the sequence σ is denoted by $\Phi(\sigma)$. Since $\Phi(\sigma) \geq 0$ for all σ a proof of this inequality proves that ALG is c-competitive.

The proof that the above inequality holds will be by induction on the length of the sequence σ. Note that the inequality holds trivially when σ is of zero length. Suppose that we know that the inequality holds for all sequences of length less than n. Let σ_n be a sequence of n requests and let σ_{n-1} be the first $n-1$ requests of σ_n.

The induction hypothesis gives us that for any adversary ADV

$$C_{\mathrm{ALG}}(\sigma_{n-1}) + \Phi(\sigma_{n-1}) \leq cC_{\mathrm{ADV}}(\sigma_{n-1}) + \Phi_0.$$

By the definition of a potential function we are guaranteed that

$$\Delta\Phi \leq c\Delta C_{\mathrm{ADV}} - \Delta C_{\mathrm{ALG}}$$

where

$$\begin{aligned} \Delta C_{\mathrm{ALG}} &= C_{\mathrm{ALG}}(\sigma_n) - C_{\mathrm{ALG}}(\sigma_{n-1}) \\ \Delta C_{\mathrm{ADV}} &= C_{\mathrm{ADV}}(\sigma_n) - C_{\mathrm{ADV}}(\sigma_{n-1}) \\ \Delta\Phi &= \Phi(\sigma_n) - \Phi(\sigma_{n-1}) \end{aligned}$$

Thus, it follows that

$$C_{\mathrm{ALG}}(\sigma_n) + \Phi(\sigma_n) \leq cC_{\mathrm{ADV}}(\sigma_n) + \Phi_0.$$

■

4.2 k Servers on a Line

The DOUBLE_COVERAGE algorithm described in [CKPV90] is an optimal on-line algorithm for the k-server problem on the line. In response to a request at a location not covered by one of the algorithm's servers it will move one or two servers. If all DOUBLE_COVERAGE's servers are on one side of the request then it will move a nearest server to the requested location. If the request falls between two servers then a closest neighbor on each side of the request is moved. These two servers are moved towards the request by the distance of the closer server.

The DOUBLE_COVERAGE algorithm for k servers is k-competitive. We will present the proof given in [CKPV90]. The proof is, essentially, giving a function Φ and showing that it satisfies the definition of a potential function for the algorithm and a competitive ratio of k.

Theorem 4 *DOUBLE_COVERAGE with k servers is k-competitive.*

Proof: Let $a_1, \ldots, a_k$ be the adversary's servers and let $s_1, \ldots, s_k$ be our (i.e DOUBLE_COVERAGE's) servers. We will also use a_i or s_j to represent the location of a server on the real line. By relabeling the servers when necessary we can ensure that $s_1 \leq \ldots \leq s_k$ and $a_1 \leq \ldots \leq a_k$.

Define a potential function Φ by

$$\Phi = \Psi + \Theta$$

where

$$\Psi = k \sum_i |a_i - s_i|$$

and

$$\Theta = \sum_{i<j} |s_i - s_j|.$$

Since each term of Φ is nonnegative, Φ can never be negative.

We must check that any moves by the adversary or by us in response to a request change the potential function correctly. We will assume that the adversary is reasonable. For the purposes of the analysis we will break up each move into phases. A phase of our movement ends when we move a server past another of our servers or a server belonging to the adversary. A phase of the adversary's movement ends when it moves a server past one of its own servers or one of our servers. At the start of any of these phases we may need to relabel the servers to ensure the preservation of the orderings $s_1 \leq \ldots \leq s_k$ and $a_1 \leq \ldots \leq a_k$ during the phase but this won't change the value of Φ.

During a phase of the adversary's movement, the adversary may move the server a_i by a distance d incurring a cost d. If the move is towards s_i then Ψ will decrease by kd. If the move is away from s_i then Ψ will increase by kd. In either case $\Delta\Psi \leq kd$ and hence $\Delta\Phi \leq kd$.

During a phase of a move by DOUBLE_COVERAGE, Φ will decrease by at least the cost incurred. Consider first the situation where DOUBLE_COVERAGE is moving just one server. It will be either s_1 or s_k. Without loss of generality we will assume that s_1 is moving to the left (i.e. in the negative direction) along the real line. Let d be the distance it moves. Since the adversary has already served the request it has some server to the left of s_1. Therefore a_1 is to the left of s_1. Φ decreases by exactly d, the cost incurred by DOUBLE_COVERAGE, because Ψ's value will decrease by kd and Θ's value will increase by $(k-1)d$.

Lastly, we must consider a phase of a move by DOUBLE_COVERAGE in which it is moving two servers towards the request and towards each other. Suppose the two servers moving are s_i and s_{i+1} and that a_j is a server belonging to the adversary at the requested location. If $j \leq i$ then s_i which is moving towards a_j is also moving towards a_i. In this case Ψ will not increase because any increase caused by the movement of s_{i+1} will

be canceled by the movement of s_i. If $j \geq i+1$ then the roles of s_i and s_{i+1} are reversed and we still have that Ψ does not increase.

We are left to show that Θ will decrease by enough to pay for the movement of the two servers. The terms that do not involve s_i or s_{i+1} will not change. Since every server s_j when $j \neq i, i+1$ is either to the left of both moving servers or to the right of both moving servers we have that one of the moving servers is moving towards s_j and the other away. Hence the sum $|s_j - s_i| + |s_j - s_{i+1}|$ remains constant during the phase. The only term in Θ we have not yet accounted for is $|s_i - s_{i+1}|$. This term will decrease by exactly the cost of the movement during the phase. ■

Roughly speaking, Ψ is used to increase the potential function when the adversary moves and Θ is used to decrease the potential function when DOUBLE_COVERAGE moves. Ψ is affected when DOUBLE_COVERAGE moves, though. It is only the fact that one of our servers is moving towards the adversary's server to which it is matched that prevents Ψ from growing too much. This will be exploited in other algorithms that we shall see.

4.3 k Servers on a Tree

An algorithm similar to DOUBLE_COVERAGE works on a tree. A *tree* metric space is a tree in the usual graph theoretic sense. Each edge is a line segment with a length. The points of the metric space are the points of the line segments including their endpoints (i.e. the vertices of the graph).

For any two points x and y in the tree we will use $[x, y]$ to mean the unique simple path from x to y. In the obvious way we will use $[x, y)$, $(x, y]$, and (x, y) to indicate the same path but not including one or both of the endpoints. The distance between x and y is the length of the path $[x, y]$ and will be written $|x - y|$.

The algorithm we will describe comes from [CL89a]. They call it Algorithm 1. To avoid confusion we will call it TREE. Suppose there is a request at the location r. The TREE server s_i is called active if $[s_i, r]$ contains no servers belonging to TREE other than the server s_i itself. If several of TREE's servers are sitting at a point s and the interval $(s, r]$ contains no servers belonging to TREE then exactly one of the servers at s is chosen arbitrarily to be active.

Notice that we can describe DOUBLE_COVERAGE using the active-server terminology. It moves all of its active servers (i.e. one or two) towards the request at the same speed until a server reaches the request. TREE behaves quite similarly. All active servers are moved at the same speed towards the request until one of them reaches the requested location.

There is a somewhat subtle difference between the line and tree algorithms, though. In DOUBLE_COVERAGE once a server is active for a request it remains active until the request is served. This is not true for TREE. The movement of the servers may cause one server to move between another server and the request. This second server now has a server on its path to the request and does not remain active. It halts its movement. Only those servers that are still active will continue to move.

Notice that any active server is not only moving towards the request but is also moving towards every other moving server. This follows from the fact that the metric space is a tree. Suppose there is a moving server at s_i. Removing the point s_i from the tree separates the tree into at least two connected components. The server s_i is moving towards all servers on the component containing the request r. The only servers that s_i is moving away from are those on the components not containing r. However, none of these servers will be active because the path from any of them to r necessarily contains the server s_i. This observation will be useful in the proof that TREE is k-competitive.

As with the line algorithm it will be convenient to break up the motion of the servers into phases. A phase ends whenever a server reaches the request, reaches a vertex of the tree, or reaches any server belonging to TREE or the adversary. Notice that any server that is active during a phase will be active for the whole phase.

Theorem 5 *The TREE algorithm for k servers is k-competitive.*

Proof: As with the proof of the on-line line algorithm for the line we will use a potential function.

Let π be a permutation of the integers $1, \ldots, k$. Define the nonnegative function Φ by

$$\Phi = \Psi + \Theta$$

where

$$\Psi = k \min_{\pi} \left\{ \sum_{i} |s_i - a_{\pi(i)}| \right\}$$

and

$$\Theta = \sum_{i<j} |s_i - s_j|.$$

As in previous arguments $s_1, \ldots, s_k$ are TREE's servers and $a_1, \ldots a_k$ are an adversary's servers.

Notice that any π that minimizes Ψ can be thought of as a bipartite minimum-weight perfect matching between the adversary's and TREE's servers. Thus, this potential function is the same as the one used in the proof of Theorem 4 except in that proof we implicitly used $\pi(i) = i$ because that yields a minimum-weight perfect matching.[3]

We must show that $\Delta\Phi \leq kd$ when an adversary moves with cost d and that $\Delta\Phi \leq -d$ when TREE moves with cost d.

First, consider a phase of the adversary's move. Assume, for the moment, that we fix the matching π. As the adversary moves server a_j a distance d incurring a cost d we are guaranteed that Φ does not increase by more than kd. When we allow π to vary to minimize Ψ we may lower the value of Φ but we will not increase it. Therefore $\Delta\Phi \leq kd$ for any move by the adversary costing d.

Now let us consider the movement of TREE's servers during a phase. Let n be the number of servers that are active. Suppose each of the active servers moves a distance d during this phase so that TREE incurs a total cost of nd. We need to show that $\Delta\Phi \leq -nd$.

Consider Θ first. Those terms involving only inactive servers will not change. There will be $\binom{n}{2}$ terms involving only active servers. Each of these terms will decrease by $2d$ during the phase since all the active servers are moving towards each other.

The remaining terms involve one active server and one inactive server. Each inactive server s_i has a server between it and the request r. The closest server s_a (which was chosen arbitrarily in the case of ties) to r along this path will be active. During the phase, s_a will be moving towards r and hence away from s_i.

However, all the other $n-1$ active servers will be moving towards the inactive server s_i. Let s_b be an active server different from s_a. Since s_b is active, the active server s_a will not be on the path $[s_b, r]$. Thus if we remove s_a from the tree, s_b will be in the same connected component as r. Since s_i is not in the connected component containing r, the unique path $[s_b, s_i]$ necessarily passes through s_a. Since s_b is moving towards s_a it is also moving towards s_i.

Thus, since the choice of s_b was arbitrary, we have shown that all but one of the active servers is moving towards any given inactive server. Since there are $k-n$ inactive servers the change to Θ due to the terms involving an active server and an inactive server is $(k-n)(1-(n-1))d$. The total change in Θ due to all its terms is therefore

$$\Delta\Theta = -\binom{n}{2} 2d + (k-n)(1-(n-1))d,$$

or more simply,

$$\Delta\Theta = -[(n-2)k + n]d.$$

We are trying to prove that $\Delta\Phi \leq -nd$ for this phase. Given the bound we have on $\Delta\Theta$ we must show that Ψ does not increase by more than $(n-2)kd$. We will assume the π chosen at the beginning of the phase minimizes Φ during the entire phase. (As before we have that even if we had to choose a different π to minimize Φ later in the phase this action could only lower Φ further.) In such a situation all the terms of Ψ involving TREE's inactive servers do not change.

If at the beginning of the phase a π that minimizes Ψ matched an active server with the adversary's server at the request site then the term for that active server would decrease by kd. The other $n-1$ terms due to active servers would increase Ψ by no more than kd each. Thus, in such a case we would be able to show that $\Delta\Psi \leq (n-1)kd - kd = (n-2)kd$. This is the bound we are seeking.

Therefore, to conclude the proof it is sufficient to show that there is a π that minimizes Ψ and

[3] Proof left to the reader.

matches the adversary's server at the request site to one of TREE's active servers. Choose a π that minimizes Ψ and suppose the server s_i is not active and is matched to the server $a_{\pi(i)}$ which is at the request site. Since s_i is inactive and since $a_{\pi(i)}$ is at the request we know that the path $[s_i, a_{\pi(i)}]$ contains an active server. Suppose this active server is s_k. It is matched to the server $a_{\pi(k)}$. The fact that s_k is between s_i and $a_{\pi(i)}$ gives

$$|s_k - a_{\pi(i)}| = |s_i - a_{\pi(i)}| - |s_i - s_k|.$$

The triangle inequality gives

$$|s_i - a_{\pi(k)}| \leq |s_i - s_k| + |s_k - a_{\pi(k)}|.$$

Together they imply that

$$\begin{aligned} & |s_k - a_{\pi(i)}| + |s_i - a_{\pi(k)}| \\ \leq \quad & |s_i - a_{\pi(i)}| + |s_k - a_{\pi(k)}| \end{aligned}$$

Thus switching the matching so that the adversary's server at the request $a_{\pi(i)}$ is matched to the active server s_k and the adversary's server at $a_{\pi(k)}$ is matched to the inactive server s_i does not increase Ψ. Therefore this new matching also minimizes Ψ.

The existence of this matching proves that Ψ does not increase by more than $(n-2)kd$ during a phase. Together with the known decrease in Θ this shows that the potential function decreases by at least nd which is the cost that TREE is paying for the phase.

We have shown that during every phase, and hence every move by either TREE or the adversary Φ changes correctly. Therefore TREE is k-competitive. ■

4.4 The Equivalence of the Tree and Line Potential Functions

It is readily apparent that the potential function for TREE given in this report is equivalent to the potential function for DOUBLE_COVERAGE as given here and in [CKPV90]. This is not true of the potential function for TREE given in [CL89a]. They give a different potential function that looks as if it is unrelated to the potential function for DOUBLE_COVERAGE. In this section we will show that the two potential functions for TREE are equivalent. It will follow that the potential for the line in [CKPV90] and the potential for the tree in [CL89a] are equivalent.

The potential function used in [CL89a] is an integral over the points of the tree. That is,

$$\Phi' = \int_T \lambda(x)\ dx$$

where $\lambda(x)$ is a function that will be defined on all points of T except for a set of measure zero. It will be piecewise constant as well and hence the integral will be well defined.

The function $\lambda(x)$ will be defined on the *ordinary points* of the tree that do not contain a server. Call a point x in a tree T an *ordinary point* if the set $T - \{x\}$ has two connected components. (Otherwise call it a *vertex*.) For any ordinary point x not containing a server let T_1 and T_2 be the two components of $T - \{x\}$. Let l_i be the number of adversary's servers in T_i and let m_i be the number of TREE's servers in T_i. Without loss of generality assume that $l_2 \geq m_2$. Define

$$\lambda(x) = l_2^2 - m_2^2 + l_1 l_2.$$

Although this function Φ' looks quite different, it is equivalent to Φ as defined in Theorem 5. Notice that $l_1 + l_2 = m_1 + m_2 = k$. Therefore,

$$\begin{aligned} \lambda &= l_2^2 - m_2^2 + l_1 l_2 \\ &= (l_1 + l_2) l_2 - (m_1 + m_2) m_2 + m_1 m_2 \\ &= k(l_2 - m_2) + m_1 m_2 \end{aligned}$$

The integral Φ' can be broken into two terms in a suggestive way. Let

$$\Phi' = \Psi' + \Theta'$$

where

$$\Psi' = k \int_T (l_2 - m_2)\ dx$$

and

$$\Theta' = \int_T m_1 m_2\ dx$$

This parallels the definition of Φ in Theorem 5 and it is not hard to see that $\Psi' = \Psi$ and $\Theta' = \Theta$.

We will show why $\Theta' = \Theta$. Consider an ordinary point x that does not contain a server. The product $m_1 m_2$ for that point is the number of TREE's servers that are in T_1 times the number in T_2. That is, it is the number of pairs of servers where one server is in T_1 and the other is in T_2. This is equal to the number of pairs of servers $\{s_i, s_j\}$ where $x \in [s_i, s_j]$. The value of Θ', the integral of $m_1 m_2$ over T, is thus the sum of the lengths of the paths $[s_i, s_j]$. This is the definition of Θ!

The case of $\Psi' = \Psi$ is a little more complicated but quite similar. It is left as an exercise to the reader.

4.5 Two Servers in Any Metric Space

Although the same algorithm for the k-server problem works on the line and the tree it does not work on a general metric space when $k > 2$. To describe the algorithm in a more general setting we would need a more general definition of active server. To make the proof work using the same potential function we would need three properties that we discovered were true in the case of the tree. First we would need that every active server is moving towards every other active server. As before, moving towards each other means that if two servers each move a distance d then the distance between them decreases by $2d$.

Secondly, we need that at most one active server is moving away from any given inactive server.

Thirdly, we would need that the adversary's server at the request site was matched by a minimum-weight perfect matching to one of the active servers which is moving towards it.

When there are at least 3 servers there is no obvious way to make them satisfy these conditions simultaneously if the metric space is not a tree. However the situation is not so bad when $k = 2$. In this section we will present a solution for the 2-server problem that works in any metric space. The algorithm is from [CL89b] though the proof is a little different. Both the proof from [CL89b] and the proof about to be presented embed the metric space in a larger space. However, the space we use is simpler and allows the omission of parts of their proof.

Intuitively the 2-server algorithm, TWO, is quite similar to TREE. A server is active if the other server is not *between* it and the request. We will describe *between* in a moment. In the event that both servers s_1 and s_2 are in the same location, one of them is chosen arbitrarily to be the active one.

For three points x, y, and z in a metric space M we say that y is *between* x and z if

$$|x - y| + |y - z| = |x - z|.$$

That is, y is between x and z if there is a shortest path from x to z passing through y. When M is a tree this definition of active is the same as the one given in Section 4.3.

For the purpose of describing the algorithm we will assume that the metric space M has a finite number of points. This requirement eases the description of the algorithm but is not necessary. We will remove it later.

It is convenient to imagine that the servers are moving around in a metric space which is a superset of the metric space M. Suppose M has n points $\{1, \ldots, n\}$. We will embed M into the space $\mathbb{R}^n$ under the ℓ^∞ metric.

Let d_{ij} be the distance between the points $i, j \in M$. We can prove that the mapping $f : M \to \mathbb{R}^n$ with $f(x) = (d_{1x}, \ldots, d_{nx})$ is an isometric embedding. We must show that for all $x, y \in M$

$$|f(x) - f(y)| = d_{xy}.$$

This is not too difficult. We have that

$$\begin{aligned} |f(x) - f(y)| &= \max_{i=1,\ldots,n} \{|d_{ix} - d_{iy}|\} \\ &\leq \max_{i=1,\ldots,n} \{d_{xy}\} \\ &= d_{xy}. \end{aligned}$$

We also have that

$$\begin{aligned} |f(x) - f(y)| &\geq |d_{xx} - d_{xy}| \\ &= d_{xy}. \end{aligned}$$

Thus, equality is proven. Henceforth we will abuse notation and identify M with $f(M)$.

In $\mathbb{R}^n$, any server that wishes to move "just a little" towards some point will have no trouble doing so. We may not be able to do this if we are

restricted to points in M. Furthermore, unless a server is between two other points in $\mathbb{R}^n$ it can move a small distance towards both of them simultaneously. That is, by moving a distance ϵ it can get closer to each of the two points by ϵ. It is these properties that make the TWO algorithm work in $\mathbb{R}^n$.

We will describe TWO as an algorithm that runs on $\mathbb{R}^n$ but it is not hard to convert it to an algorithm that runs on M. Converting TWO to a reasonable algorithm in the usual way does the trick. Since all requests are to points in M all movement for the reasonable algorithm will be to points in M. Therefore when we prove that TWO is 2-competitive on $\mathbb{R}^n$ we will have shown that it's reasonable version is 2-competitive on M.

The movement for TWO is broken up into two phases. In the first phase both servers move towards each other and, simultaneously, towards the request. The phase ends when one server comes between the other server and the request. In the second phase the server which is still active moves to the request. Note that for either phase there are degenerate cases in which no movement occurs.

A slight modification of Theorem 5 will prove that TWO is 2-competitive so long as we can show:

1. There exists a minimum-weight perfect bipartite matching that matches the adversary's server at the request site to one of TWO's active servers.

2. At most one active server is moving away from any given inactive server.

3. All active servers are moving towards each other and towards the request.

The first is easy. During the first phase both servers are active so the adversary's server at the request site must be matched with an active server under any matching. An argument similar to that presented at the end of Theorem 5 shows that during the second phase any minimum-weight perfect matching that does not satisfy the requirement can be converted to one which does. As was proven before, the swapping of matched partners will not change the weight of the matching.

The second item is also easy. During the first phase there are no inactive servers so the statement is true vacuously. During the second phase there is only one active server so no inactive server can have more than one active server moving away from it.

We must consider the third item in each of the two phases. In the second phase the only active server is moving towards the request. Since there are no other active servers it is true, vacuously, that it is moving towards all of them.

The first phase is harder. We have servers at $s_1, s_2 \in \mathbb{R}^n$ and there is a request at $r \in \mathbb{R}^n$. We must show that it is possible to move TWO's two servers towards each other and towards the request simultaneously.

Define

$$\begin{aligned} d_1 &= \frac{1}{2}\left(|s_1 - r| + |s_1 - s_2| - |s_2 - r|\right) \\ d_2 &= \frac{1}{2}\left(|s_2 - r| + |s_1 - s_2| - |s_1 - r|\right) \\ d &= \min\{d_1, d_2\}. \end{aligned}$$

Intuitively, d_1 is the distance that the server at s_1 must move so that it comes between s_2 and r. Similarly for d_2. Since the first phase terminates when one of the servers comes between the other server and the request, d is the distance that both servers will move during the first phase.

For the server that starts at s_1 at the beginning of the first phase use s_1' to indicate where it has moved to by the end of the phase. Similarly, use s_2' for the location at the end of the first phase of the server starting at s_2. To show the truth of the third item for the first phase of the movement we demonstrate that it is possible to find s_1' and s_2' such that

$$\begin{aligned} |s_1' - s_1| &= d \\ |s_2' - s_2| &= d \\ |s_1' - r| &= |s_1 - r| - d \\ |s_2' - r| &= |s_2 - r| - d \\ |s_1' - s_2'| &= |s_1 - s_2| - 2d. \end{aligned}$$

Furthermore, we show that either s_1' is between s_2' and r or vice-versa.

We compute each coordinate of s_1' and s_2' separately. For the ith coordinate of s_1', written $s_1'(i)$, we choose any real number that satisfies

$$|s_1'(i) - s_1(i)| \leq d$$

$$\begin{aligned} |s_1'(i) - s_2(i)| &\leq |s_1 - s_2| - d \\ |s_1'(i) - r(i)| &\leq |s_1 - r| - d. \end{aligned}$$

These three inequalities define intervals. We must choose $s_1'(i)$ from their intersection. The first two inequalities can be satisfied simultaneously because $|s_1(i) - s_2(i)|$, the distance between the centers of the intervals, is not more than $|s_1 - s_2|$, the sum of the radii. Similarly, the first and third inequalities can be satisfied simultaneously.

The second and third inequalities can also be satisfied simultaneously. From the definition of d we see that

$$\begin{aligned} |s_2(i) - r(i)| &\leq |s_2 - r| \\ &= |s_1 - r| + |s_1 - s_2| - 2d_1 \\ &\leq |s_1 - r| + |s_1 - s_2| - 2d \end{aligned}$$

Since the first expression is the distance between the centers of the intervals and the last expression is the sum of the radii this implies that the two intervals overlap. Whenever three intervals on the real line intersect pairwise it follows that the intersection of all three is nonempty.[4] Thus, we will be able to choose a value for $s_1'(i)$.

Just to be definite[5] we'll let $s_1'(i)$ have the legal value that is nearest to $s_1(i)$. Note that any other value within all three intervals would do just as well.

In an analogous manner, we choose $s_2'(i)$ so that it satisfies

$$\begin{aligned} |s_2'(i) - s_2(i)| &\leq d \\ |s_2'(i) - s_1'(i)| &\leq |s_1 - s_2| - 2d \\ |s_2'(i) - r(i)| &\leq |s_2 - r| - d. \end{aligned}$$

and is closest to $s_2(i)$. Notice that we have s_2 moving towards s_1' rather than s_1. The second and third inequalities can be satisfied simultaneously because

$$\begin{aligned} |s_1'(i) - r(i)| &\leq |s_1' - r| \\ &\leq |s_1 - r| - d \\ &= |s_2 - r| + |s_1 - s_2| - 2d_2 - d \\ &\leq |s_2 - r| + |s_1 - s_2| - 3d \end{aligned}$$

[4]The proof is left for the reader.

[5]TWO is a deterministic algorithm.

The other pairs of inequalities are solvable. Thus, all inequalities can be satisfied simultaneously.

We must show that s_1' and s_2' satisfy the five equalities given previously. We have immediately that

$$\begin{aligned} |s_1' - s_1| &\leq d \\ |s_2' - s_2| &\leq d \\ |s_1' - r| &\leq |s_1 - r| - d \\ |s_2' - r| &\leq |s_2 - r| - d \\ |s_1' - s_2'| &\leq |s_1 - s_2| - 2d. \end{aligned}$$

and hence for each equality, we need only show that the left-hand side is greater than or equal to the right-hand side. Choose a, b and c so that

$$\begin{aligned} |s_1(a) - s_2(a)| &= |s_1 - s_2| \\ |s_1(b) - r(b)| &= |s_1 - r| \\ |s_2(c) - r(c)| &= |s_2 - r|. \end{aligned}$$

Using the triangle inequality, we show that s_1 and s_2 have moved a distance d.

$$\begin{aligned} &|s_1' - s_1| \\ &\quad \geq |s_1'(a) - s_1(a)| \\ &\quad \geq |s_2(a) - s_1(a)| - |s_2(a) - s_1'(a)| \\ &\quad \geq |s_2 - s_1| - (|s_2 - s_1| - d) \\ &\quad = d. \end{aligned}$$

Similarly,

$$\begin{aligned} &|s_2' - s_2| \\ &\quad \geq |s_2'(a) - s_2(a)| \\ &\quad \geq |s_1(a) - s_2(a)| - |s_1(a) - s_1'(a)| \\ &\qquad\quad - |s_1'(a) - s_2'(a)| \\ &\quad \geq |s_1 - s_2| - d - (|s_1 - s_2| - 2d) \\ &\quad = d. \end{aligned}$$

Hence $|s_1' - s_1| = d$ and $|s_2' - s_2| = d$.

We also get that s_1 and s_2 are now closer to r by d.

$$\begin{aligned} &|s_1' - r| \\ &\quad \geq |s_1'(b) - r(b)| \\ &\quad \geq |s_1(b) - r(b)| - |s_1(b) - s_1'(b)| \\ &\quad \geq |s_1 - r| - d. \end{aligned}$$

Hence $|s_1' - r| = |s_1 - r| - d$ and similarly $|s_2' - r| = |s_2 - r| - d$.

Next we must prove that the two servers are moving towards each other. We get immediately that

$$\begin{aligned} |s_1' - s_2'| &\geq |s_1 - s_2| - |s_1' - s_1| - |s_2' - s_2| \\ &= |s_1 - s_2| - 2d. \end{aligned}$$

Hence, the equality $|s_1' - s_2'| = |s_1 - s_2| - 2d$ is proved.

The last thing we must check for finite metric spaces is that one of the two servers comes between the other and the request. That is, we need either of:

$$\begin{aligned} |s_2' - s_1'| + |s_1' - r| - |s_2' - r| &= 0 \\ |s_1' - s_2'| + |s_2' - r| - |s_1' - r| &= 0 \end{aligned}$$

We compute both:

$$\begin{aligned} &|s_2' - s_1'| + |s_1' - r| - |s_2' - r| \\ &\quad = |s_2 - s_1| - 2d + |s_1 - r| - d - |s_2 - r| + d \\ &\quad = 2(d_1 - d) \\ &|s_1' - s_2'| + |s_2' - r| - |s_1' - r| \\ &\quad = |s_1 - s_2| - 2d + |s_2 - r| - d - |s_1 - r| + d \\ &\quad = 2(d_2 - d) \end{aligned}$$

Since $d = \min\{d_1, d_2\}$, one of the two expressions will be zero.

When M has infinitely many points the algorithm needs to be modified slightly. At a point where only n distinct requests have been made it is sufficient to embed M into $\mathbb{R}^n$. As long as requests are made to only these n locations it is as if M had only n points and hence we can run TWO in $\mathbb{R}^n$ as before. That is, although the servers move around in $\mathbb{R}^{|M|}$, we only keep track of the n coordinates representing the requested locations.

When a request is made to a new location $r_{n+1} \in M$ we run the algorithm in $\mathbb{R}^{n+1}$. To do this we must locate r_{n+1} in $\mathbb{R}^{n+1}$. We must also provide an isometric embedding of $\mathbb{R}^n$ into $\mathbb{R}^{n+1}$ that preserves the first n coordinates.

For $i = 1, \ldots, n$ we set

$$r_{n+1}(i) = |r_{n+1} - r_i|.$$

For the isometric embedding we define a function $f : \mathbb{R}^n \to \mathbb{R}^{n+1}$. Write $f(x)(i)$ for the ith coordinate of $f(x)$. Define f by

$$\begin{aligned} f(x)(i) &= x(i) \\ f(x)(n+1) &= \max_i \{ |\, |x - r_i| - |r_{n+1} - r_i| \,| \}. \end{aligned}$$

where i ranges from 1 to n in both expressions.

We must check that f is an isometry and that r_{n+1} is the correct distance from the previously requested locations. First we will prove the isometry. Let $x, y \in \mathbb{R}^n$ be any two points. We will prove that $|f(x) - f(y)| = |x - y|$. Now,

$$\begin{aligned} &|f(x) - f(y)| \\ &\quad = \max\{|x - y|, |f(x)(n+1) - f(y)(n+1)|\} \end{aligned}$$

Thus we need to prove that

$$f(x)(n+1) - f(y)(n+1) \leq |x - y|.$$

Let $m \leq n$ be a coordinate such that r_m is a request that achieves the maximum in the definition of $f(x)(n+1)$. That is,

$$f(x)(n+1) = |\, |x - r_m| - |r_{n+1} - r_m| \,|.$$

We get

$$\begin{aligned} &f(x)(n+1) - f(y)(n+1) \\ &\quad \leq |\, |x - r_m| - |r_{n+1} - r_m| \,| - \\ &\qquad\quad |\, |y - r_m| - |r_{n+1} - r_m| \,| \\ &\quad \leq |x - r_m| - |y - r_m| \\ &\quad \leq |x - y|. \end{aligned}$$

The first inequality is from the definition of f. The second inequality is the triangle inequality in $\mathbb{R}$. The last inequality is the triangle inequality in $\mathbb{R}^n$. Hence f is an isometric embedding.

As soon as we verify that $|r_{n+1} - r_i|$ is the same in M and $\mathbb{R}^{n+1}$ for all $i \leq n$ we are done. It is sufficient to prove that $r_i(n+1) = |r_{n+1} - r_i|$. With this information we will know that for all $1 \leq i, j \leq n+1$ it is true that $r_i(j) = |r_i - r_j|$. We have already proven that under such circumstances the distances between the requests is the same in M and $\mathbb{R}^{n+1}$.

We compute:

$$\begin{aligned} r_i(n+1) &= \max_{j=1,\ldots,n}\{|\,|r_i - r_j| - |r_j - r_{n+1}|\,|\} \\ &\geq |\,|r_i - r_i| - |r_i - r_{n+1}|\,| \\ &= |r_i - r_{n+1}| \end{aligned}$$

To get the other inequality we choose an $m \leq n$ such that r_m is a request that maximizes the expression in the definition of $r_i(n+1)$. It then follows that

$$\begin{aligned} r_i(n+1) &= |\,|r_i - r_m| - |r_m - r_{n+1}|\,| \\ &\leq |r_i - r_{n+1}|. \end{aligned}$$

Thus we have proven

Theorem 6 *The TWO algorithm for two servers in any metric space is 2-competitive.*

5 Finite Metric Spaces

So far we have seen optimal on-line algorithms for uniform metric spaces, lines, and trees. We have also seen an optimal on-line algorithm for 2 servers in any metric space. These algorithms are all quite similar. We have already shown how DOUBLE_COVERAGE and TWO are like TREE.

LRU, the algorithm for uniform metric spaces, is also like TREE. A uniform metric space can be embedded into a tree that looks like an asterisk with many spokes. Each spoke has length 1/2 and the points of the uniform metric space are identified one-to-one with the tips of the spokes. As measured by paths through the tree the tips of the spokes are unit distance apart just as in the uniform metric space. To run TREE on the uniform metric space we pretend to run it on the whole tree we have constructed — not just the tips of the spokes. We keep track of the location of each server as if it were in the tree instead of the uniform space. Only when a server pretends to move to the tip of a spoke do we actually move it from the previous tip that it occupied. If the arbitrarily-decided ties that arise under the TREE algorithm are decided in favor of the server that has moved least recently then TREE and LRU will behave identically.

Thus, all the algorithms we have seen so far are quite similar.[6] However, there are other ways to optimally solve the on-line k-server problem. The optimal on-line solution to the k-server problem on a space of $k+1$ points presented in [MMS88] is an example of another approach.

5.1 *n-1* Servers in a Space of Size n

Before we present the algorithm of [MMS88] and prove its competitive ratio, we must introduce some more terminology and notation. Let M be a metric space. A *configuration* of k servers is a function S from the integers $\{1,\ldots,k\}$ to the metric space M that gives the locations of the k servers.

There is a natural equivalence relation that arises when the k servers are indistinguishable. We say that two configurations S and T are equivalent if they are the same except for a relabeling of the servers. Precisely, $S \simeq T$ if there exists a permutation π of the integers $\{1,\ldots,k\}$ such that $S(\pi(i)) = T(i)$ for $i = 1,\ldots,k$.

The distance between two configurations is the minimum amount of movement needed to move the k servers from one configuration to the other. As with the equivalence relation, we must allow for relabeling so we define the distance between configurations S and T by

$$|S - T| = \min_{\pi}\left\{\sum_{i=1}^{k}|S(\pi(i)) - T(i)|\right\}.$$

This distance function satisfies the triangle inequality. Let S_1, S_2, and S_3 be configurations and let π_{12} and π_{23} be permutations such that

$$\begin{aligned} |S_1 - S_2| &= \sum_{i=1}^{k}|S_1(\pi_{12}(i)) - S_2(i)| \\ |S_2 - S_3| &= \sum_{i=1}^{k}|S_2(\pi_{23}(i)) - S_3(i)| \end{aligned}$$

We check that $|S_1 - S_2| + |S_2 - S_3| \geq |S_1 - S_3|$ as follows:

$$|S_1 - S_2| + |S_2 - S_3|$$

[6] If we had the desire to do so we could probably design a single abstract algorithm that performed optimally in all the cases we have examined so far.

$$
\begin{aligned}
&= \sum_{i=1}^{k} (|S_1(\pi_{12}(i)) - S_2(i)| \\
&\qquad + |S_2(\pi_{23}(i)) - S_3(i)|) \\
&= \sum_{i=1}^{k} (|S_1(\pi_{12}(\pi_{23}(i))) - S_2(\pi_{23}(i))| \\
&\qquad + |S_2(\pi_{23}(i)) - S_3(i)|) \\
&\geq \sum_{i=1}^{k} |S_1(\pi_{12}(\pi_{23}(i))) - S_3(i)| \\
&\geq \min_{\pi} \left\{ \sum_{i=1}^{k} |S_1(\pi(i)) - S_3(i)| \right\} \\
&= |S_1 - S_3|
\end{aligned}
$$

Thus the set of configurations modulo the equivalence relation forms a metric space.

We write $C_{\mathrm{OPT}}(\sigma_m, S)$ for the total cost of a cheapest (off-line) way to serve a request sequence σ_m of length m, starting from any start configuration and ending in a configuration S. It is important to note that $C_{\mathrm{OPT}}(\sigma_m, S)$ is still well defined in the case that S does not contain a server at the last request site. In such a case some movement must be made after the last request is served to get to the configuration S.

We can define C_{OPT} recursively on the length of the sequence of requests. For any S, let $C_{\mathrm{OPT}}(\sigma_0, S) = 0$. Let $\mathcal{F}_m$ to be the set of all configurations that have a server at the location $\sigma(m)$. We define

$$C_{\mathrm{OPT}}(\sigma_m, S) = \min_{S' \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_{m-1}, S') + |S' - S|\}.$$

This is the same as our previous definition because a cheapest way to serve m requests and end up in S is the same as the cheapest way to satisfy $m-1$ requests, end up in a configuration that satisfies the mth request, and then move to S.

Notice that

$$C_{\mathrm{OPT}}(\sigma_m, S) \geq C_{\mathrm{OPT}}(\sigma_{m-1}, S)$$

for all S because the left-hand side represents the cost of sequence which is a superset of the sequence of the right-hand side. Equality holds when $S \in \mathcal{F}_m$.

Now let's begin the description of the BAL (which is short for *balancing*) algorithm for k servers in a metric space with n points where $n = k+1$. Let the n points be $\{1, \ldots, n\}$ and let S_i be the configuration of k servers that has exactly one server at every location except i.

Assume that initially BAL starts with its servers in configuration S_n but that the adversary may start with its servers in any of the n configurations $\{S_1, \ldots, S_n\}$. We will assume that the adversary is reasonable and we will define BAL to be reasonable so we need only ever consider the n configurations just described.

Let's define d_{ij} to be the distance between points i and j. In terms of configurations of servers we get $|S_i - S_j| = d_{ij}$.

BAL is quite simple. Each of BAL's servers keeps track of how much work it has done. Let $D_i(\sigma_m)$ be the sum of all work done up to and including the mth request by the server whose location is i after the mth request. If BAL gets a request to a location in which it already has a server it does nothing. If BAL is in configuration S_i and the $(m+1)$st request is to location i BAL chooses a j that minimizes $D_j(\sigma_m) + d_{ji}$ and moves the server at j to the requested location i. That is, BAL moves a server whose total work after the move would be minimum. Equivalently, BAL moves a server that minimizes $D_i(\sigma_{m+1})$.

Since BAL does no work and does not change any of its state when a location it occupies is chosen as a request we will assume without loss of generality that the adversary always requests the unique location not occupied by one of BAL's servers.

It is convenient to define $D_i(\sigma_m)$ even if i is unoccupied by BAL after serving the request $\sigma(m)$. In this situation we define $D_i(\sigma_m) = D_i(\sigma_{m-1})$. When $m = 0$ we define $D_i(\sigma_m)$ to be zero for $i = 1, \ldots, n$.

Lemma 7 *$D_i(\sigma_m) = C_{\mathrm{OPT}}(\sigma_m, S_i)$ for all $i = 1, \ldots, n$ and all nonnegative integers m.*

Proof: We will prove this by induction. For the case $m = 0$ we note that no server has moved yet so $D_i(\sigma_0) = 0$ for all i. Furthermore, since we are allowing the adversary to start in any configura-

tion it is also true that $C_{\rm OPT}(\sigma_0, S_i) = 0$ for all i.

Let's suppose that for all i

$$D_i(\sigma_{m-1}) = C_{\rm OPT}(\sigma_{m-1}, S_i)$$

and let's compute $D_i(\sigma_m)$. Note that if $i \neq \sigma(m)$ then after the mth request either there is a server at i and it hasn't moved or a server has just vacated the location. In either case D_i does not change. That is,

$$\begin{aligned} D_i(\sigma_m) &= D_i(\sigma_{m-1}) \\ &= C_{\rm OPT}(\sigma_{m-1}, S_i). \end{aligned}$$

The last term, $D_{\sigma(m)}(\sigma_m)$, can be computed by examining how BAL decides which server to move. It chooses a server that can move with lowest total cost thus we get that

$$\begin{aligned} & D_{\sigma(m)}(\sigma_m) \\ &= \min_{j \neq \sigma(m)} \{D_j(\sigma_{m-1}) + d_{j,\sigma(m)}\} \\ &= \min_{j \neq \sigma(m)} \{C_{\rm OPT}(\sigma_{m-1}, S_j) + |S_j - S_{\sigma(m)}|\} \end{aligned}$$

The minimum does not include the point $\sigma(m)$ because BAL has no server at that location before the request is made.

Now let's compute $C_{\rm OPT}(\sigma_m, S_i)$ for all i. If $S_i \in \mathcal{F}_m$ then we know that $i \neq \sigma(m)$. Thus we can conclude that

$$\begin{aligned} C_{\rm OPT}(\sigma_m, S_i) &= C_{\rm OPT}(\sigma_{m-1}, S_i) \\ &= D_i(\sigma_m) \end{aligned}$$

When it is not true that $S_i \in \mathcal{F}_m$ then it must be that $i = \sigma(m)$. Thus the recurrence relation gives

$$\begin{aligned} & C_{\rm OPT}(\sigma_m, S_i) \\ &= \min_{S' \neq S_{\sigma(m)}} \{C_{\rm OPT}(\sigma_{m-1}, S') + |S' - S_{\sigma(m)}|\} \\ &= D_{\sigma(m)}(\sigma_m) \end{aligned}$$

We conclude $D_i(\sigma_m) = C_{\rm OPT}(\sigma_m, S_i)$ for all i and all m. ■

Theorem 8 *The BAL algorithm for k servers in a metric space with $n = k+1$ points is k-competitive.*

Proof: The total cost for BAL in serving the sequence σ_m is the sum of the costs attributable to each of its servers. That is,

$$\begin{aligned} C_{\rm BAL}(\sigma_m) &= \sum_{i \neq \sigma(m+1)} D_i(\sigma_m) \\ &= \sum_{S_i \neq S_{\sigma(m+1)}} C_{\rm OPT}(\sigma_m, S_i) \end{aligned}$$

On the other hand the cost to the adversary is

$$C_{\rm ADV}(\sigma_m) = \min_{S_i} \{C_{\rm OPT}(\sigma_m, S_i)\}$$

because the adversary must satisfy all the requests but can end up in any configuration.

To show that BAL is k-competitive we must show that $C_{\rm BAL}(\sigma_m) - kC_{\rm ADV}(\sigma_m)$ is bounded from above by a constant that is independent of the sequence σ_m. Comparing the formulae for $C_{\rm BAL}$ and $C_{\rm ADV}$ we see that it is sufficient to show that $|C_{\rm OPT}(\sigma_m, S_i) - C_{\rm OPT}(\sigma_m, S_j)|$ is bounded by a constant that is independent of σ_m.

Let

$$d_{\max} = \max_{i,j=1,\ldots,n} \{d_{ij}\}.$$

If we can prove that

$$C_{\rm OPT}(\sigma_m, S_i) \leq C_{\rm OPT}(\sigma_m, S_j) + d_{\max}$$

for all i and j then we are done.

The inequality is little more than a statement of the triangle inequality.

$$\begin{aligned} & C_{\rm OPT}(\sigma_m, S_i) \\ &= \min_{S \in \mathcal{F}_m} \{C_{\rm OPT}(\sigma_{m-1}, S) + |S - S_i|\} \\ &\leq \min_{S \in \mathcal{F}_m} \{C_{\rm OPT}(\sigma_{m-1}, S) + |S - S_j| \\ &\qquad + |S_j - S_i|\} \\ &= \min_{S \in \mathcal{F}_m} \{C_{\rm OPT}(\sigma_{m-1}, S) + |S - S_j|\} \\ &\qquad + |S_j - S_i| \\ &= C_{\rm OPT}(\sigma_m, S_j) + d_{ji} \\ &= C_{\rm OPT}(\sigma_m, S_j) + d_{\max} \end{aligned}$$

■

5.2 k Servers in a Space of Size n

The k-competitive algorithm BAL that runs in a metric space of $k+1$ points can generalized to an algorithm FINITE that runs in any finite metric space M. Unfortunately, FINITE is not k-competitive. We will see shortly that its competitive ratio is at most $\binom{n}{k}-1$ where n is the number of points in the metric space. Notice that when $n=k+1$ the competitive ratio is $\binom{k+1}{k}-1=k$ as expected.

As we did in Section 5.1 we assume that the adversary only chooses requests at locations not occupied by a server belonging to FINITE. We assume that the adversary is reasonable and hence the adversary will always have its servers at distinct locations. We will define FINITE so that it always has its servers at distinct locations. Under these assumptions the only configurations that will be important are those with k servers at distinct locations. There are $\binom{n}{k}$ of these.

FINITE is a generalization of BAL except that we dispense with the $D_i(\sigma_m)$ values and deal directly with $C_{\mathrm{OPT}}(\sigma_m, S)$. Initially, we set

$$C_{\mathrm{OPT}}(\sigma_0, S) = 0$$

for every configuration S. Let $S_1, \ldots, S_{m-1}$ be the configurations that FINITE occupied after the first $m-1$ requests and suppose the mth request is to a location $\sigma(m)$. Recall that $\mathcal{F}_m$ is the set of configurations that have a server at $\sigma(m)$. FINITE moves to a configuration $S_m \in \mathcal{F}_m$ that minimizes

$$C_{\mathrm{OPT}}(\sigma_m, S_m) + |S_m - S_{m-1}|$$

among all configurations in $\mathcal{F}_m$.

Intuitively, FINITE is moving to a configuration that is cheap for the adversary but isn't too far from S_{m-1}. Moving to configurations which are cheap for the adversary helps make FINITE competitive since the adversary is always choosing requests at locations where FINITE has no server. An adversary in the same configuration as FINITE is thus forced to move to satisfy the next request. The net result is that an adversary that was performing cheaply is forced to do more movement.

BAL made sure that each server did an equal share of the work. In some sense it is correct to say that FINITE makes sure that each configuration does an equal share of the work. We will see that FINITE can serve the sequence for no more than the sum of the costs over all but one of the configurations. Using the fact that the adversary can serve a sequence incurring the cost of the cheapest configuration will lead easily into a proof that the competitive ratio of FINITE is not more than $\binom{n}{k}-1$.

Lemma 9 is best motivated by the fact that it is useful in Theorem 10. In that theorem we bound the cost incurred by FINITE while serving a sequence σ. We will be considering sums of the form

$$\sum_{S \neq S_i} C_{\mathrm{OPT}}(\sigma_i, S)$$

for $i = m, m-1$. The difference of these two sums will include terms of the form found in the lemma.

Lemma 9

$C_{\mathrm{OPT}}(\sigma_m, S_{m-1}) - C_{\mathrm{OPT}}(\sigma_{m-1}, S_m) = |S_m - S_{m-1}|$

for $m > 0$.

Proof:

$$\begin{aligned} & C_{\mathrm{OPT}}(\sigma_m, S_{m-1}) \\ &= \min_{S \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_{m-1}, S) + |S - S_{m-1}|\} \\ &= \min_{S \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_m, S) + |S - S_{m-1}|\} \\ &= C_{\mathrm{OPT}}(\sigma_m, S_m) + |S_m - S_{m-1}| \end{aligned}$$

The first equality is the recurrence relation defining C_{OPT}. The second equality is true because $S \in \mathcal{F}_m$. The third equality follows because FINITE chooses S_m to minimize the expression. ■

We can plug this result directly into Theorem 10.

Theorem 10 *Define $C_{\mathrm{FIN}}(\sigma_m)$ to be the total cost incurred by FINITE serving the sequence σ_m. Then*

$$C_{\mathrm{FIN}}(\sigma_m) \leq \sum_{S \neq S_m} C_{\mathrm{OPT}}(\sigma_m, S)$$

Proof: The proof is by induction on m. When $m = 0$, both sides are identically zero.

Now, for $m > 0$,

$$\begin{aligned}
&\sum_{S \neq S_m} C_{\mathrm{OPT}}(\sigma_m, S) - \sum_{S \neq S_{m-1}} C_{\mathrm{OPT}}(\sigma_{m-1}, S) \\
&\quad \geq \; C_{\mathrm{OPT}}(\sigma_m, S_{m-1}) - C_{\mathrm{OPT}}(\sigma_{m-1}, S_n) \\
&\quad = \; |S_m - S_{m-1}| \\
&\quad = \; C_{\mathrm{FIN}}(\sigma_m) - C_{\mathrm{FIN}}(\sigma_{m-1})
\end{aligned}$$

The first inequality comes from comparing terms in the two sums. All terms involving configurations other than S_{m-1} and S_m can be thrown away because $C_{\mathrm{OPT}}(\sigma_m, S) \geq C_{\mathrm{OPT}}(\sigma_{m-1}, S)$. The first equality comes from Lemma 9.

Thus, the induction hypothesis,

$$C_{\mathrm{FIN}}(\sigma_{m-1}) \leq \sum_{S \neq S_{m-1}} C_{\mathrm{OPT}}(\sigma_{m-1}, S),$$

implies

$$C_{\mathrm{FIN}}(\sigma_m) \leq \sum_{S \neq S_m} C_{\mathrm{OPT}}(\sigma_m, S).$$

■

Since we allow the adversary to start in any configuration and since it does not care in which configuration it ends up, we know

$$C_{\mathrm{ADV}}(\sigma_m) = \min_S \{C_{\mathrm{OPT}}(\sigma_m, S)\}.$$

The proof of the competitiveness of FINITE follows easily.

Theorem 11 *FINITE for k servers in a metric space M with n points has a competitive ratio of $\binom{n}{k} - 1$.*

Proof: We must show that

$$C_{\mathrm{FIN}}(\sigma_m) - \left[\binom{n}{k} - 1\right] C_{\mathrm{ADV}}(\sigma_m)$$

is bounded above by a constant independent of σ_m. Given the formulae we have for C_{FIN} and C_{ADV} the problem reduces to showing that

$$|C_{\mathrm{OPT}}(\sigma_m, S) - C_{\mathrm{OPT}}(\sigma_m, S')|$$

is bounded above by a constant independent of σ_m.

We proceed much as we did in Theorem 8. Define $d_{\max}$ to be the maximum distance between two points of M. The maximum distance between two configurations is bounded by $kd_{\max}$, a constant. The recurrence relation for C_{OPT} and the triangle inequality give

$$\begin{aligned}
&C_{\mathrm{OPT}}(\sigma_m, S) \\
&\quad = \; \min_{S'' \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_{m-1}, S'') + |S'' - S|\} \\
&\quad \leq \; \min_{S'' \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_{m-1}, S'') \\
&\qquad\qquad + |S'' - S'| + |S' - S|\} \\
&\quad = \; \min_{S'' \in \mathcal{F}_m} \{C_{\mathrm{OPT}}(\sigma_{m-1}, S'') + |S'' - S'|\} \\
&\qquad\qquad + |S' - S| \\
&\quad \leq \; C_{\mathrm{OPT}}(\sigma_m, S') + kd_{\max}
\end{aligned}$$

■

Part II
Distinguishable Costs

Traditionally, the k-server problem has been formulated so that the k servers are indistinguishable. We can imagine situations where this is not the case. An example is the SRAM versus DRAM memory management problem mentioned in the introduction. Each block of memory is a server. The pages of data are the points of the metric space. Moving a server from one page to another corresponds to flushing the first page from the block of memory and loading the second page in. The cost of the move is its duration.

If the pages reside on different media (e.g. cache, disks, tape drives) then different moves may have different costs. Traditionally, the k-server model does assume that the cost can vary based on the points in the metric space. However the cost must be independent of the server serving the request. In this case the servers are blocks of memory so the traditional model insists that every piece of memory behaves the same as every other. If we want to break that assumption and let the cost be

a function of the server we must look into a more general model.

The model we consider in this report is one where the cost of any move is proportional to the distance in the metric space but the constant of proportionality may vary from server to server. We call this the k-distinguishable-server problem.

6 Obvious Bounds

It would be nice to have bounds on the competitiveness of the k-distinguishable-server problem in various metric spaces. For instance, a theorem similar to the one by [MMS88] that states that no on-line k-server algorithm can have a competitive ratio better than k would be good.[7] Fortunately, a few bounds on the competitive ratio of on-line algorithms for distinguishable servers are immediate.

Define $\mu_i > 0$ to be the *cost coefficient* for the ith server. By *cost coefficient* we mean that the the cost incurred by moving the ith server from x to y is $\mu_i|x - y|$. The k-tuple $(\mu_1, \ldots, \mu_k)$ is the *tuple of cost coefficients*. Generally, an on-line algorithm ALG and the adversary will have the same tuple of cost coefficients for their servers.

Let's relax this last constraint for the moment and assume that the adversary and ALG may have independent cost coefficients. If the cost coefficient of one of ALG's servers is increased but ALG still moves exactly as before then ALG's cost will be at least as much as before. The adversary's cost will remain unchanged. Hence, ALG's competitive ratio will be at least what it was before. If a cost coefficient of ALG's is lowered then the competitive ratio will be no worse than it was before. Similarly, if the cost coefficient of one of the adversary's servers is increased the competitive ratio cannot worsen and if that coefficient is decreased then the ratio cannot improve. It is also true that if the cost coefficient of each of ALG's servers is multiplied by some positive constant or if the cost coefficient of each of the adversary's servers is divided by some positive constant then ALG's competitive ratio will be multiplied by that constant.

[7] See Section 2 for a statement and proof of the theorem by [MMS88].

From these simple observations we can prove Lemma 12.

Lemma 12 *Suppose an on-line algorithm ALG and an adversary each have k servers with cost coefficients $(\mu_1, \ldots, \mu_k)$. Let*

$$\begin{aligned} \mu_{\min} &= \min_i\{\mu_i\} \\ \mu_{\max} &= \max_i\{\mu_i\}. \end{aligned}$$

If the metric space M has at least $k+1$ points then the competitive ratio of ALG is at least $k\frac{\mu_{\min}}{\mu_{\max}}$

Proof: Fix a sequence of requests. Let $C_{\mathrm{ALG}}(\sigma_n)$ be ALG's cost for serving the first n requests of the sequence. Let $C'_{\mathrm{ALG}}(\sigma_n)$ be the cost for ALG if the same moves are made but each server has a cost coefficient of unity. Similarly, define $C_{\mathrm{ADV}}(\sigma_n)$ and $C'_{\mathrm{ADV}}(\sigma_n)$ to be the costs incurred by the adversary with the given cost coefficients and unit cost coefficients, respectively.

Theorem 1 tells us that the competitive ratio when the servers of the adversary and ALG have unit cost coefficients is at least k. Since, for all n,

$$\begin{aligned} C_{\mathrm{ALG}}(\sigma_n) &\geq \mu_{\min} C'_{\mathrm{ALG}}(\sigma_n) \\ C_{\mathrm{ADV}}(\sigma_n) &\leq \mu_{\max} C'_{\mathrm{ADV}}(\sigma_n) \end{aligned}$$

it follows that the competitive ratio for ALG is at least $k\frac{\mu_{\min}}{\mu_{\max}}$. ■

Likewise we can prove:

Lemma 13 *Suppose an instance of the k-server problem can be solved by an on-line algorithm with a competitive ratio of c. If cost coefficients $(\mu_1, \ldots, \mu_k)$ are introduced then this instance of the k-distinguishable-server problem can be solved by an algorithm with a competitive ratio of no more than $c\frac{\mu_{\max}}{\mu_{\min}}$ where*

$$\begin{aligned} \mu_{\min} &= \min_i\{\mu_i\} \\ \mu_{\max} &= \max_i\{\mu_i\}. \end{aligned}$$

Proof: Run the algorithm as if the cost coefficients were unity. ■

The lower bound given in Lemma 12 can be tightened. Rather than bounding the adversary's

cost by $\mu_{\max}$ times the distance moved we can bound it by μ_{avg} times the distance moved where

$$\mu_{\text{avg}} = \frac{1}{k}\sum_{i=1}^{k}\mu_i.$$

Furthermore, the adversary has the option of not using its more expensive servers. By assuming that the adversary only uses its cheapest servers we may be able to prove an even tighter bound for the competitive ratio.

Theorem 14 *Suppose an on-line algorithm ALG and an adversary each have k servers with cost coefficients $\mu_1 \leq \ldots \leq \mu_k$. If the metric space M has at least $k+1$ points then the competitive ratio of ALG is at least*

$$\max_j \left\{ \frac{k\mu_{\min}}{(k+1-j)M_j} \right\}$$

where

$$M_j = \frac{1}{j}\sum_{i=1}^{j}\mu_i$$

the average of the j lowest cost coefficients. In particular, the competitive ratio is at least

$$\frac{k\mu_{\min}}{\mu_{\text{avg}}}.$$

Proof: The proof is very similar to that of Theorem 1. Choose a subset of M with $k+1$ points and assume that ALG starts with its servers in distinct locations. Assume that the adversary always requests the unique location not occupied by one of ALG's servers. Even if ALG always moves its cheapest server the cost to satisfy a request sequence will be

$$C_{\text{ALG}}(\sigma_n) \geq \sum_{i=1}^{n}\mu_{\min}|\sigma(i+1)-\sigma(i)|.$$

Suppose the adversary decides to use only its cheapest j servers and suppose one starts at the location of the first request and the other $j-1$ start spread through the remaining k locations, at most one to a location. There are $t = j!\binom{k}{j-1}$ ways to place the servers since they are distinguishable.

We run t adversaries each with the same algorithm but differing in which of the t configurations they begin. To satisfy a request $\sigma(n)$ an adversary either does nothing if it already has a server at the request or, otherwise, moves the server from $\sigma(n-1)$.

In Theorem 1 we had that no two distinct adversaries would end up in the same configuration after a given request sequence. The same observations and arguments prove this fact for our current situation. Thus, after each request we know that the adversaries occupy the t distinct configurations that have a server at the last request.

We add up the costs charged to the adversaries for a request $\sigma(n)$. Among the adversaries, $j!\binom{k-1}{j-2}$ will already have a server at $\sigma(n)$ and will incur no cost to serve the request. The remaining $j!\binom{k-1}{j-1}$ adversaries will not have a server at $\sigma(n)$ and will have to move the one from $\sigma(n-1)$. The cost for each of these adversaries will be, on average, $M_j|\sigma(n)-\sigma(n-1)|$. The average of the cost coefficients is M_j because we have adversaries in all configurations and hence it is equally likely that we are moving any of the servers.

The total cost to move all t adversaries is

$$\sum_{i=2}^{n} j!\binom{k-1}{j-1} M_j|\sigma(n)-\sigma(n-1)|$$

therefore there must be one that is at least as cheap as the average

$$\frac{k+1-j}{k}M_j\sum_{i=2}^{n}|\sigma(n)-\sigma(n-1)|.$$

It follows that the competitive ratio cannot exceed

$$\max_j \left\{ \frac{k\mu_{\min}}{(k+1-j)M_j} \right\}.$$

In particular, the $j=k$ term gives that

$$\frac{k\mu_{\min}}{\mu_{\text{avg}}}$$

is a lower bound on the competitive ratio of any on-line algorithm. ■

The proofs in this section do not account for the fact that an on-line algorithm may move a server with a cost coefficient different from $\mu_{\min}$. We make the conjecture that an adversary can force a better competitive ratio than the one given in Theorem 14.

Conjecture 1 *No deterministic on-line algorithm for the k-distinguishable-server problem on a metric space with at least $k+1$ points can achieve a competitive ratio better than k.*

7 Uniform Metric Spaces

In specific cases of the k-distinguishable-server problem we can get better bounds on the competitive ratio. For instance, the bound of Lemma 13 can be improved if M is the uniform metric space. In this case we have a competitive ratio of $k\frac{\mu_{\text{avg}}}{\mu_{\min}}$ for LRU.

Assume that the adversary always makes requests to locations which do not contain a server belonging to LRU. And assume that, initially, LRU uses its servers from cheapest to most expensive. From then on it uses the least recently used server. In effect, it is using the sequence cheapest to most expensive over and over again. Thus the average cost to serve a sequence for LRU is bounded by μ_{avg} times what it would have been with unit cost coefficients.

At worst (for LRU) the adversary can manage to use $\mu_{\min}$ times the cost it would have had with unit cost coefficients. Thus LRU has a competitive ratio of $k\frac{\mu_{\text{avg}}}{\mu_{\min}}$. We do not know if there is an on-line algorithm that achieves a better competitive ratio.

8 Two Servers

In this section we look at the k-distinguishable-server problem for $k = 2$ servers in various metric spaces. First we consider the line. Then we generalize the results to the Tree and to arbitrary metric spaces.

8.1 Two Servers on a Line

We will demonstrate an upper bound to the competitive ratio of an optimal on-line algorithm for 2 servers on a line by giving an algorithm. Afterwards we will prove a lower bound.

The algorithm we will use to find an upper bound is the DOUBLE_COVERAGE algorithm from Section 4.2. Although we tried tweaking the parameters of DOUBLE_COVERAGE to make it more suitable for the k-distinguishable-server model we did not find any algorithms that were better. We discuss some of these attempts in Section 8.2.

Theorem 15 *The DOUBLE_COVERAGE algorithm for 2 distinguishable servers on a line has a competitive ratio not exceeding*

$$\frac{3\left(\frac{\mu_{\max}}{\mu_{\min}}\right)+1}{2}$$

where $\mu_{\min}$ is the lower of the two cost coefficients and $\mu_{\max}$ is the higher.

Proof: The proof will involve a potential function Φ. Without loss of generality assume that the smaller cost coefficient is unity and the other is $\mu > 1$. Let s_1 be DOUBLE_COVERAGE's unit-cost server and let s_2 be the more expensive server. Let a_1 and a_2 be the cheaper and more expensive of the adversary's servers. As usual, we will also use s_1, s_2, a_1, and a_2 to represent the location of these servers.

Let π be a permutation of $\{1, 2\}$ and define the potential function to be

$$\Phi = \Psi + \Theta$$

where

$$\Psi = \frac{3\mu+1}{2}\min_{\pi}\left\{\sum_{i=1}^{2}|s_i - a_{\pi(i)}|\right\}$$

and

$$\Theta = \frac{\mu+1}{2}|s_1 - s_2|$$

Except for the coefficients at the front of the Ψ and Θ terms this potential function is identical to the one given in Theorem 4. The introduction of

the permutations π is really nothing new. In Theorem 4 it was unnecessary to consider the minimum over all permutations because the servers were indistinguishable and we could relabel them at will. We kept relabeling them so that the permutation $\pi(i) = i$ minimized the Ψ term.

The potential function is nonnegative so we are left to check that

1. When the adversary moves Φ increases by no more than $(3\mu + 1)/2$ times the cost of the move.

2. When DOUBLE_COVERAGE moves Φ decreases by at least the cost of the move.

The first item is easy. A server move by the adversary cannot increase Ψ by more than $(3\mu + 1)/2$ times the distance moved by the server. Thus, Ψ increases by no more than $(3\mu + 1)/2$ times the cost of the move.

The second item is not much harder. Assume the adversary is reasonable. Recall from Theorem 4 that we are guaranteed that at least one of DOUBLE_COVERAGE's moving servers is moving towards the server it is matched to under a minimum-distance perfect matching. When a request falls between s_1 and s_2 we know that Ψ will not increase because the server moving towards its match will lower one term of the sum enough to cover any possible increase in the other term. The movement of the servers towards each other will cause a drop in Θ exactly equal to the cost of the movement.

When a request is not between DOUBLE_COVERAGE's servers we have that the decrease in Ψ is sufficient to cover the cost of the move and the increase in Θ. Ψ decreases by $(3\mu + 1)/2$ times the distance the server moves while Θ increases by $(\mu + 1)/2$ times the distance moved. Thus Φ decreases by μ times the distance moved. Regardless of which of DOUBLE_COVERAGE's servers is moving, this will be a decrease of at least the cost of the move. ■

We will prove a lower bound on the competitiveness of any on-line algorithm ALG that runs on a line by demonstrating a strategy that can be employed by an adversary to keep ALG's costs high. The strategy is a loop. The servers start at certain points on the line and the adversary makes requests until the servers return to their initial positions. For each iteration of the loop, the cost charged to ALG will be at least twice the cost charged to the adversary. Thus the competitive ratio of ALG cannot be better than $k(= 2)$. This is a better bound then the obvious bounds derived earlier.

Although it is not known if this bound is tight, it is interesting to note that when $\mu_{\max}$ is strictly greater than $\mu_{\min}$ the competitive ratio must be strictly greater than k. For the traditional k-server problem no one has ever proven that an optimal on-line algorithm must have a competitive ratio exceeding k.

Theorem 16 *No on-line algorithm ALG for 2 distinguishable servers on a line can have a competitive ratio better than*

$$1 + \left(\frac{2\mu}{\mu + 1}\right)^2$$

where $\mu = \mu_{\max}/\mu_{\min}$.

Proof: Without loss of generality assume that the cheaper server has a cost coefficient of 1 and the other server has a coefficient of $\mu > 1$. Let s_1 and s_2 be respectively the cheaper and more expensive servers belonging to ALG. Likewise define a_1 and a_2 for the adversary. Without loss of generality because we are minimizing over all on-line algorithms we will assume that ALG is reasonable.

When we want to prove that ALG is c-competitive for some value c then we must show that there exists a constant a such that for every request sequence σ

$$cC_{\mathrm{OPT}}(\sigma) - C_{\mathrm{ALG}}(\sigma) + a \geq 0.$$

To prove that ALG is not c-competitive it sufficient to find an adversary and an infinite sequence σ such that

$$\inf_n \{cC_{\mathrm{ADV}}(\sigma_n) - C_{\mathrm{ALG}}(\sigma_n)\} = -\infty.$$

As usual we are using σ_n to mean the first n requests of σ. We will describe an adversary and a σ such that ALG is not c-competitive unless

$$c \geq 1 + \left(\frac{2\mu}{\mu + 1}\right)^2.$$

The sequence σ will be a repeating loop. The initial position for the loop involves two locations separated by a unit distance. Both ALG and the adversary have a server at each of the locations.

Assuming that ALG is competitive, the adversary can force the servers into this initial configuration. The adversary makes the first two requests to points that are unit distance apart. It then requests them alternately until ALG moves its servers into a configuration that covers both locations. The adversary serves the sequence by moving its servers to the two locations.

This always works unless the reasonable ALG tries to serve every request using just one server. However such an algorithm is not competitive. This initialization may have a cost but the cost to the adversary can be absorbed into the additive constant present in the definition of competitive ratio.

The requests made during an iteration of the loop follow a simple pattern. Choose ϵ such that $0 < \epsilon \ll 1$. The adversary alternates making requests at the location initially occupied by s_2 and the location between s_1 and s_2 that is a distance ϵ from s_2. This continues as long as ALG uses s_2 to serve the request. As soon as s_1 is used, the adversary makes a request to the location just vacated by s_1.

Let d be the product of the distance ϵ and the number of requests for which ALG uses s_2. That is, d is the total distance moved by s_2 before s_1 is moved. ALG has only two options during an iteration of the loop. It gets to choose d and it gets to choose which server to move to the location vacated by s_1.

If ALG moves s_1 to satisfy the request at the vacated location then

$$\Delta C_{\mathrm{ALG}} \geq \mu d + 2(1-\epsilon)$$

for an iteration. The first term is the cost of the wiggling and the second term is a lower bound on the cost to move s_1 in and then back out. If ALG uses s_2 instead of s_1 to satisfy the last request then it will incur an additional cost of at least $(\mu-1)(1-\epsilon)$.

We will define the adversary so that at the beginning of an iteration it will be in one of two configurations. Either $|a_1 - s_1| = |a_2 - s_2| = 0$ or $|a_1 - s_2| = |a_2 - s_1| = 0$. The first configuration is called *matched* and the second configuration is called *switched*. We will write $C_{\mathrm{ADV}}(\sigma_n, M)$ (resp. $C_{\mathrm{ADV}}(\sigma_n, S)$) for the cost to serve the sequence σ_n and end in the matched (resp. switched) configuration.

By switching the locations of its two servers, the adversary can change from configuration M to S or vice-versa for a total cost of $1 + \mu$. Hence we always have that

$$|C_{\mathrm{ADV}}(\sigma_n, M) - C_{\mathrm{ADV}}(\sigma_n, S)| \leq 1 + \mu.$$

The adversary's cost for a sequence will be the smaller of these two costs. Since the difference between the two costs is bounded by a constant, the cost to the adversary will be within a constant of any weighted average of $C_{\mathrm{ADV}}(\sigma_n, M)$ and $C_{\mathrm{ADV}}(\sigma_n, S)$. In particular, the cost to the adversary on a sequence σ_n will be within a constant of

$$\left(\frac{1}{2} + \frac{\mu-1}{2c(\mu+1)}\right) C_{\mathrm{ADV}}(\sigma_n, S) + \left(\frac{1}{2} - \frac{\mu-1}{2c(\mu+1)}\right) C_{\mathrm{ADV}}(\sigma_n, M)$$

where c is a prospective competitive ratio for ALG.

We want to show that $cC_{\mathrm{ADV}} - C_{\mathrm{ALG}}$ is unbounded below unless c is large enough. With this in mind we define

$$\Omega(\sigma_n) = c\left[\left(\frac{1}{2} + \frac{(\mu-1)(1-\epsilon)}{2c(\mu+1)}\right) C_{\mathrm{ADV}}(\sigma_n, S) + \left(\frac{1}{2} - \frac{(\mu-1)(1-\epsilon)}{2c(\mu+1)}\right) C_{\mathrm{ADV}}(\sigma_n, M)\right] - C_{\mathrm{ALG}}(\sigma_n).$$

If we can show that $\inf_n\{\Omega(\sigma_n)\} = -\infty$ then we know that ALG is not c-competitive. Therefore, it is sufficient to show that there is a $\delta > 0$ such that for all iterations

$$\Delta\Omega \leq -\delta.$$

Let's calculate $\Delta C_{\mathrm{ADV}}(M)$ and $\Delta C_{\mathrm{ADV}}(S)$ for an iteration assuming that ALG does not switch its servers (i.e. does not pay the extra $(\mu-1)(1-\epsilon)$

at the end of the iteration.) The adversary has the option of serving an entire iteration with its server at the same location as s_2. Alternatively, the adversary can move the other server in at the start and then move it back at the end of the iteration.

When ALG does not switch its servers, both options for the adversary leave the adversary in the same configuration in which it began the iteration. That is, it finishes switched if and only if it starts switched. It finishes matched if and only if it starts matched.

The adversary has two options for serving an iteration. Thus, we compute:

$$\begin{aligned} \Delta C_{\mathrm{ADV}}(M) &\leq \min\{\mu d, 2\} \\ \Delta C_{\mathrm{ADV}}(S) &\leq \min\{d, 2\mu\} \end{aligned}$$

The first term in each minimization represents the wiggling option. The second term is the other option. It is the cost the adversary must pay to move the outside server in at the beginning of the iteration and back out at the end.

We've already established that

$$\Delta C_{\mathrm{ALG}} \geq \mu d + 2(1-\epsilon).$$

Thus,

$$\begin{aligned} \Delta\Omega \leq \\ & c\left[\left(\frac{1}{2} + \frac{(\mu-1)(1-\epsilon)}{2c(\mu+1)}\right)\min\{d, 2\mu\}\right. \\ & \left. + \left(\frac{1}{2} - \frac{(\mu-1)(1-\epsilon)}{2c(\mu+1)}\right)\min\{\mu d, 2\}\right] \\ & - \mu d - 2(1-\epsilon). \end{aligned}$$

The right-hand side of this expression is a piecewise linear function of d. When $d = 0$ it is negative. Also as $d \to \infty$ it goes to $-\infty$.

We wish to show that the right-hand side is negative no matter what the value of $d \geq 0$ unless c is big enough. The right-hand side is nonnegative for some value of d if and only if it is nonnegative at its maximum. Since as a function of d it is piecewise linear its maximum must occur at one of the points of nondifferentiability. That is, a nonnegative maximum will occur at $d = 2/\mu$ or $d = 2\mu$.

Evaluating, we see that $\Delta\Omega(d = 2/\mu)$ can be nonnegative for all values of ϵ only if

$$c \geq \frac{5\mu^2 + 2\mu + 1}{(\mu+1)^2} = 1 + \left(\frac{2\mu}{\mu+1}\right)^2.$$

We also have that $\Delta\Omega(d = 2\mu)$ can be nonnegative for all values of ϵ only if

$$c \geq \frac{2\mu^3 + \mu^2 + 4\mu + 1}{(\mu+1)^2}.$$

This second bound is more restrictive than the first so we can ignore it.

Thus, no matter what value ALG chooses for d, $\inf_n\{\Omega(\sigma_n)\}$ will be $-\infty$ unless c is as big as stated in the theorem. If ALG never switches its two servers the theorem is proved.

Suppose ALG does switch its servers at the end of an iteration. The additional cost to ALG for a switch must be at least $(\mu-1)(1-\epsilon)$. We can think of this an as additional step in which the adversary incurs no cost. We must check that Ω does not increase as a result of this extra step.

The C_{ALG} term of Ω causes Ω to drop by at least $(\mu-1)(1-\epsilon)$. However, the other two terms of Ω may increase due to the fact that ALG's switch causes a renaming of the adversary's configurations S and M. That is, S and M as well as their costs, $C_{\mathrm{ADV}}(\sigma_n, M)$ and $C_{\mathrm{ADV}}(\sigma_n, S)$, swap. We know that the difference between these costs is not more than $1+\mu$. Thus, switching S and M cannot increase Ω by more than $(\mu-1)(1-\epsilon)$. Hence, combining to two changes to Ω, we see that $\Delta\Omega \leq 0$ for ALG's extra switch step. ■

As already remarked before, this bound is $> k$ when $\mu > 1$. As $\mu \to \infty$ the bound goes to 5. The value 5 seems unnatural and occurs as the result of much messy algebra. We don't know if it can be improved.

8.2 Varying the Speeds

The way DOUBLE_COVERAGE is designed, when both servers are moving they move the same distance. We can imagine that the servers are moving continuously through time along the line, at the same speed, and stop when one reaches the request. An obvious question is "What happens if we make them move at different speeds?"

Before exploring this topic we must take another look at potential functions. Multiplying a potential function Φ by a positive constant will change some of its properties. It will remain nonnegative. However, it may no longer be true that a move by

an adversary causes Φ to increase by no more than the competitive ratio times the cost of the move. Furthermore, a move by an on-line algorithm ALG may not cause Φ to decrease by at least the cost of the move.

The scaled potential function is still useful in that it can be converted into a canonical potential function by multiplication. Thus the existence of a scaled potential function, like the existence of a canonical potential function, allows us to prove a competitive ratio for an algorithm.

Lemma 17 *Let ALG be an on-line algorithm. Suppose Φ is a nonnegative function. Consider a move by an adversary. Let $\Delta\Phi(\text{ADV})$ be the resulting change in Φ and let $\Delta C(\text{ADV})$ be the cost of the move. Define*

$$A = \sup_{\text{all moves}} \left\{ \frac{\Delta\Phi(\text{ADV})}{\Delta C(\text{ADV})} \right\}$$

Similarly, for a move by ALG define

$$B = - \sup_{\text{all moves}} \left\{ \frac{\Delta\Phi(\text{ALG})}{\Delta C(\text{ALG})} \right\}.$$

If A and B are positive and $c = A/B$ then ALG is c-competitive.

Proof: The function Φ/B is a canonical potential function for ALG proving a competitive ratio of c. ■

We wish to consider variations to DOUBLE_COVERAGE. If we wish to prove that they have a better competitiveness then that proven in Theorem 15 we must allow the potential function to change as well. We can generalize by considering a potential function of the form

$$C \min_{\pi} \left\{ \sum_{i=1}^{2} |s_i - a_{\pi(i)}| \right\} + D|s_1 - s_2|$$

for some constants $C, D > 0$. Since we may scale our potential function it is as general to consider

$$\Phi = \min_{\pi} \left\{ \sum_{i=1}^{2} |s_i - a_{\pi(i)}| \right\} + D|s_1 - s_2|$$

for $D > 0$.

As usual we will assume that the cost coefficients are $\{1, \mu\}$ with $\mu > 1$. Also, s_1, s_2, a_1, and a_2 are as before. We will consider two variations.

1. When a request falls between s_1 and s_2, s_1 will move with speed $E \geq 0$ towards the request. s_2 will move at unit speed.

2. When a request is not between s_1 and s_2 but is closer to s_2 then s_1 will move with speed $F \geq 0$ towards the request. s_2 will move at unit speed. If $F > 1$ then it is possible that s_1 will overtake s_2. At that point s_2 halts and only s_1 continues.

To see if we've gained anything by allowing these variations we will compute A and B as defined in the statement of Lemma 17 and calculate their ratio. As we have before, we assume that in response to a request, first the adversary moves and then DOUBLE_COVERAGE moves. We also assume that the adversary is reasonable.

To get A we note that for every unit of distance that a server of the adversary moves, Φ can increase by at most 1. The cost of the movement will be at least one. It is not hard to imagine a situation where both these inequalities are equalities. Therefore

$$A = 1.$$

We can separate the computation of B into four cases based on the location of the request and which server is matched to the adversary's server at the request.

1. The request falls between the servers and the adversary's server at the request is matched to s_1.

2. The request falls between the servers and the adversary's server at the request is matched to s_2.

3. The request falls on the s_1 side of DOUBLE_COVERAGE's servers and hence the adversary's server at the request is matched to s_1 under a minimum-weight matching.

4. The request falls on the s_2 side of DOUBLE_COVERAGE's servers and hence the adversary's server at the request is matched to s_2 under a minimum-weight matching.

For each case, we calculate $\Delta\Phi$ and ΔC for each unit of distance that s_2 moves and take their ratio.

1. s_1 moves a distance E towards the s_2 and towards its match. s_2 moves unit distance towards the s_1 but may be moving away from its match. Thus

$$\Delta\Phi \leq -ED - E - D + 1$$

and

$$\Delta C = E + \mu$$

so

$$B \leq \frac{ED + E + D - 1}{E + \mu}.$$

2. In a similar manner this case gives

$$B \leq \frac{ED - E + D + 1}{E + \mu}.$$

3. Only s_1 moves. It moves away from s_2 but towards its match hence,

$$B \leq \frac{-D + 1}{1}.$$

4. s_1 moves a distance F towards s_2 but may be moving away from its match. s_2 moves a unit distance away from s_1 but is moving towards its match. Thus

$$B \leq \frac{FD - F - D + 1}{F + \mu}.$$

Since we must have $B > 0$ the third item gives us that $D < 1$. It then follows that the fourth constraint is always tighter than the third. Thus we can ignore the third constraint. The first two items can be combined into

$$B \leq \frac{D(E+1) - |E-1|}{E + \mu}.$$

We now have just two constraints on B. The first is a function of E and D and the second is a function of F and D. Maximizing the first with respect to E gives $E = 1$. Maximizing the second with respect to F gives $F = 0$. However $E = 1, F = 0$ corresponds to the unmodified DOUBLE_COVERAGE algorithm! Varying the speeds did not help us.

As expected, if we solve for D getting

$$D = \frac{\mu + 1}{3\mu + 1}$$

and plug into our constraint equations we get that

$$B = \frac{2}{3\mu + 1}$$

and a competitive ratio of

$$\frac{A}{B} = \frac{3\mu + 1}{2}.$$

Needless to say, other generalizations of DOUBLE_COVERAGE and other potential functions may be tried. We did not find any combinations that we could use to prove a better competitive ratio than that achieved by the unmodified DOUBLE_COVERAGE algorithm.

8.3 Two Servers in Any Metric Space

Just as DOUBLE_COVERAGE gives a competitive ratio of $(3\mu+1)/2$ for servers with cost coefficients $\{1, \mu\}$, the unmodified TREE and TWO algorithms give the same competitive ratio under the same conditions. The potential function defined in Section 8.1 works for both TREE and TWO. The algorithms themselves are exactly as they would be for two equal-cost servers.

The $1 + [2\mu/(1+\mu)]^2$ lower bound, from Section 8.1, applicable to lines also applies to trees. Every tree contains a line segment as a subset though there may not be one of length at least unity. However, the lower bound for lines is described with points unit distance apart for convenience only. Any two distinct points on a line segment will do.

The lower bound applies to any general metric space M that contains a line segment. Note that if M contains at least two distinct points then when it is embedded in $\mathbb{R}^n$ it will contain a line segment connecting them. Thus if an adversary is allowed to make requests in $\mathbb{R}^n$ the lower bound applies.

9 Finite Metric Spaces

A modification to FINITE gives us an on-line algorithm that works for distinguishable servers in any finite metric space M. Number the servers

from 1 to k and let $\mu_1, \ldots, \mu_k$ be their cost coefficients. As in Section 5.2 a configuration is a function $S : \{1, \ldots, k\} \to M$ that gives the location of each of the k servers.

We define an equivalence relation different from the one given in Section 5.2. We say that a permutation π of the integers $\{1, \ldots, k\}$ preserves cost coefficients if for every i, we have $\mu_i = \mu_{\pi(i)}$. Two configuration S_1 and S_2 of distinguishable servers are said to be equivalent if there exists a permutation π preserving cost coefficients such that for all i we have that $S_1(i) = S_2(\pi(i))$

Similarly we define the distance between two permutations by

$$|S_1 - S_2| = \min_{\pi} \left\{ \sum_{i=1}^{k} |S_1(i) - S_2(\pi(i))| \right\}$$

where the minimum is taken over all permutations π that preserve cost coefficients.

With these redefinitions we run FINITE as we did before. Lemma 9 and Theorem 10 remain true with the new definitions for equivalence and distance. In a metric space with n points there are $(n)_k = k!\binom{n}{k}$ distinct configurations modulo the equivalence relation[8]. Theorem 11 remains true if all occurrences of $\binom{n}{k}$ are replaced with $(n)_k$.

Putting it all together we get:

Theorem 18 *FINITE for k distinguishable servers in a metric space with n points has a competitive ratio not exceeding $(n)_k - 1$.*

Part III
Summary

10 Future Work

For indistinguishable servers there is room for improvement. Although the FINITE algorithm is a generalization of the BAL algorithm it is not also a generalization of the TWO algorithm. For instance, FINITE does not perform as well as TWO for the case of 2 servers in a metric space with 4 points. Can the two algorithms BAL and TWO be merged into a more general solution that has the advantages of FINITE and TWO?

There are several open questions for the k-distinguishable-server problem. Can a better general lower bound for the competitive ratio of an on-line algorithm be found? How can we narrow the gap between the lower bound and the competitive ratios of the various algorithms presented in this paper?

[8]This is true if the cost coefficients are distinct. In any case this is an upper bound on the number of distinct configurations

Acknowledgements

For their help, I owe thanks to many of the "Theory" students in the Graduate Computer Science Program at UC Berkeley. I also owe thanks to my advisor Dick Karp for input and advice on the substance and presentation of this report.

I owe thanks to my parents, Herbert Newberg and Babette Josephs, and my sweetheart, Heidi Marvin, for being there when I needed them.

The bulk of the original research presented in this paper was done during the summer of 1990 in the Café Strada in Berkeley. I thank them for the coffee and for the place to sit in the sun.

11 Glossary

The following definitions should prove useful.

a_i The adversary's ith server. Also used to represent the location of this server.

adversary An imagined opponent that chooses a sequence of requests to demonstrate the worst performance of an on-line algorithm. The adversary will have its own set of k servers which it uses to show how to efficiently serve a sequence of requests.

Note that the on-line algorithm cannot see where the adversary's servers are or how they move. They are imaginary and exist only to those reading the proofs.

between A point y is between the points x and z if

$$|x-y|+|y-z|=|x-z|.$$

That is, if there exists a shortest path from x to z that passes through y.

$C_{\mathrm{ALG}}(\sigma)$ The cost incurred by an algorithm ALG on a request sequence σ.

c-competitive An algorithm ALG is c-competitive if there exists a constant a, such that for every sequence of requests σ, we have that

$$C_{\mathrm{ALG}}(\sigma) \leq cC_{\mathrm{OPT}}(\sigma)+a$$

competitive An algorithm is competitive if it is c-competitive for some constant c.

competitive ratio The competitive ratio of an algorithm is the smallest c for which the algorithm is c-competitive.

configuration A configuration is a function S from the integers $\{1,\ldots,k\}$ to a metric space M that gives the locations of k servers. Let π be a permutation of the integers $\{1,\ldots,k\}$. Two configurations S and T are equivalent, $S \simeq T$, if there exists a π such that $S(\pi(i)) = T(i)$. The distance between two configurations S and T is given by

$$|S-T| = \min_{\pi}\left\{\sum_{i=1}^{k}|S(\pi(i))-T(i)|\right\}.$$

If some servers are distinguishable then the above definitions for equivalence and distance still hold except that the permutations π are limited to that subset of permutations that map each server to a server having the same cost coefficient.

$C_{\mathrm{OPT}}(\sigma,S)$ The cost of a cheapest (off-line) way to serve a request sequence σ starting from any configuration and ending in the configuration S.

cost coefficient The cost coefficient for the ith server is μ_i if the cost to move it from x to y is $\mu_i|x-y|$.

distance function See *metric space*.

k-distinguishable-server problem This is the same as the k-server problem except that the cost associated with a move by a server is proportional to the distance moved. The constant of proportionality may vary from server to server.

k-server problem This is the task of finding an efficient on-line algorithm for k servers. The k-server problem involves a metric space with a distance function. k servers are located at points in the metric space. A request is a point in the metric space. One of k servers must be moved from its current point to the requested point. The cost charged for such a move is the distance between the two points. If more than one server moves then the cost is the sum of all the distances moved.

line A metric space whose points are the points of a line. The distance between points is the normal Euclidean distance along the line.

matched configuration The configuration in the 2-distinguishable-server problem where $|a_1-s_1| = |a_2-s_2| = 0$.

metric space A metric space is a set, M, along with a distance function $d : M \times M \to \mathbb{R}$. The elements of M are called points and the distance function gives the distance between any two points. The distance function must satisfy certain constraints. Let x, y, and z be points in M.

- For every x, $d(x,x) = 0$. That is, the distance from a point to itself must be zero.
- For every x and y with $x \neq y$, we must have that $d(x,y) = d(y,x) > 0$. In other words, the distance from x to y is equal to the distance from y to x and it must be positive.
- d must satisfy the triangle inequality which states that for every x, y, and z, we have that $d(x,z) \leq d(x,y)+d(y,z)$. So, if x, y, and z are the vertices of a triangle then the length of any side must

be less than or equal to the sum of the lengths of the other two sides.

Often we will write $|x - y|$ or d_{xy} instead of $d(x, y)$.

mu μ The ratio of the largest cost coefficient to the smallest cost coefficient.

μ_i The cost coefficient of the ith server.

$\mu_{\max}$ The largest cost coefficient.

$\mu_{\min}$ The smallest cost coefficient.

minimum-weight perfect matching A permutation π such that

$$\sum_{i=1}^{k} |s_i - a_{\pi(i)}|$$

is minimum over all permutations.

optimal An algorithm is optimal on a sequence of requests if its cost is the smallest achievable on that sequence. An *optimal on-line* algorithm is an on-line algorithm which achieves the smallest competitive ratio among all on-line algorithms.

ordinary point A point x in a tree T is an *ordinary point* if the set $T - \{x\}$ has exactly two connected components. Compare to *vertex.*

potential function A function Φ is a potential function demonstrating a competitive ratio of $c > 0$ for an algorithm ALG if it satisfies the following conditions:

- Φ is nonnegative.
- Every response to a request by the adversary increases Φ by no more than c times the cost charged to the adversary for that response.
- Every response to a request by ALG decreases Φ by at least the cost charged to ALG for that response.

The existence of a potential function satisfying these criteria is sufficient proof that ALG is c-competitive.

reasonable We call an algorithm ALG's response to a request at x *reasonable* if either

1. ALG already has a server at x and does not move any of its servers, or
2. ALG does not have a server at x and moves one server to that location, but does not move any of its other servers.

We cal ALG *reasonable* if every response it makes is reasonable.

s_i An algorithm's ith server. Also used to represent the location of this server.

sigma σ A (usually finite) sequence of requests.

σ_i This is a sequence of i requests or the first i requests in a longer sequence.

$\sigma(i)$ The ith request in a sequence σ. It is also used to describe the location of the ith request in a sequence σ.

switched configuration The configuration in the 2-distinguishable-server problem where $|a_1 - s_2| = |a_2 - s_1| = 0$.

tree A metric space like a tree in the graph theoretic sense. Each edge is a line segment with a length. The points of the metric space are the points of the line segments including their endpoints (i.e. the vertices of the graph). The distance between two points is the length of the unique simple path between them.

triangle inequality See *metric space.*

uniform metric space A metric space in which the distance between any two distinct points is 1.

vertex A point x in a tree T is a *vertex* if the set $T - \{x\}$ has more than two connected components. Compare to *ordinary point.*

References

[BLS87] Allan Borodin, Nathan Linial, and Michael Saks. "An Optimal Online Algorithm for Metrical Task Systems."

Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp 373-382, 1987.

[CKPV90] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. "New Results on Server Problems." *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, Chapter 32, pp 291-300, January 1990.

[CL89a] Marek Chrobak and Lawrence L. Larmore. "An Optimal On-line Algorithm for k Servers on Trees." Department of Mathematics and Computer Science, University of California, Riverside, CA 92521. July 24, 1989.

[CL89b] Marek Chrobak and Lawrence L. Larmore. "A New Approach to the Server Problem." Department of Mathematics and Computer Science, University of California, Riverside, CA 92521. September 19, 1989.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. "Competitive Algorithms for Online Problems." *Proceedings 31st Annual Symposium on Foundations of Computer Science Volume II*, pp 454-469, October 1990.

[G91] Eddie Grove. "The Harmonic Online K-Server Algorithm is Competitive." *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing* 1991.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. "Competitive Algorithms for On-line Problems." *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pp 322-333, May 1988.