A Competitive 2-Server Algorithm

Sandy Irani *
Ronitt Rubinfeld †
Computer Science Division
U.C. Berkeley
Berkeley, CA 94720

June 23, 1995

Keywords: Design of Algorithms, Analysis of Algorithms, On-line Algorithms, Amortized Analysis.

Abstract

The K-server problem is the problem of planning the motion of K mobile servers in a metric space. We give an on-line algorithm for the 2-server problem in any metric space. The total cost of this algorithm on any sequence of requests is bounded by 10 times the cost of the optimal off-line algorithm on that sequence. The rule is a modified version of the balance algorithm; it sends the server that minimizes the quantity: (total distance traversed so far by that server + twice the distance of that that server to the next request). This is the first provably competitive rule that can be evaluated in a constant number of arithmetic operations per request with only one variable. This contrasts with the 2-competitive 2-server algorithm in [MMS] which requires maintaining $O(t)$ memory locations and $O(t)$ time to decide which server to send, where t is the minimum of the number of points in the metric space of the metric space and the number of

*Research supported by a Tandem Fellowship
†Research supported by an IBM Graduate Fellowship and NSF Grant CCR 88-13632

requests. Our rule naturally generalizes to more than two servers, and we conjecture that it is also competitive in this case.

1 Introduction

The K-server problem, first introduced in [MMS], is the problem of managing the motion of K mobile servers on a metric space and is defined as follows. There are K servers that are placed initially on K points in a metric space. They are free to move around from point to point: however, the cost of moving a server from one point to another is the distance between the two points. Denote the cost of moving a server from point i to j as c_{ij}. The costs satisfy the triangle inequality, $c_{ij} \leq c_{ik} + c_{kj}$ for all nodes i,k,j. In this paper, we assume that the distance between two points is symmetric. (i.e. for all i, j, $c_{ij} = c_{ji}$.) A request sequence for the K-server problem is a sequence of node names. The algorithm must send one of the servers to every node in the sequence in order. The object of the algorithm is to minimize the total cost incurred by the servers.

An algorithm is *on-line* if it determines which server to send to satisfy a particular request without any knowledge of the future requests. An algorithm is *off-line* if it bases its choices on the entire sequence of requests. Chrobak *et al.*have an $O(kn^2)$ method for finding the optimal way to service a sequence when the entire sequence is available ahead of time. We call this best strategy the optimal algorithm. The performance of an on-line algorithm is evaluated in comparison to the optimal strategy. This idea was first introduced in [ST] in order to evaluate on-line algorithms for updating lists. Let $C_A(\sigma)$ denote the cost that algorithm A incurs in servicing the sequence σ of requests. Let $C_{OPT}(\sigma)$ denote the cost of the optimal algorithm on σ. We assume both algorithms start from the same configuration. Algorithm A is called *α-competitive* on a class of graphs $\mathcal{C}$, if for any graph in $\mathcal{C}$ there is some constant β such that for every finite sequence of requests, σ,

$$C_A(\sigma) \leq \alpha \cdot C_{OPT}(\sigma) + \beta.$$

Note that β may depend on the initial configuration of the servers but not on σ. The *competitive ratio* for an algorithm A is the smallest α such that A

is α-competitive. An algorithm is *strongly competitive* if it achieves the best possible competitive ratio.

[MMS] exhibit a lower bound of K for the competitive ratio of any on-line algorithm for the K-server problem on any metric space with at least $K+1$ points. They conjecture that there is an on-line algorithm that is K-competitive for the K-server problem in any metric space. In the case where $K = 2$, [MMS] present an on-line strategy which is 2-competitive. However, their strategy requires $O(t)$ arithmetic operations to decide which server to send, and it requires that $O(t)$ variables be saved in memory where $t = \min \{|\sigma|, \text{number of points in the metric space}\}$. If the K-server problem is viewed strictly as an information theoretic problem, then their algorithm solves the problem for $K = 2$. However, the performance of on-line algorithms with restricted computational resources is also of fundamental importance for the K-server problem and for on-line algorithms in general. One paper where this issue has been addressed is [RS], where they examine on-line algorithms with restricted space.

In this paper, we present an algorithm for the general two server problem that requires a constant number of arithmetic operations per request, and one variable. This is the first provably competitive server rule with this property. The algorithm is shown to be 10-competitive. Recently, Chrobak and Larmore have exhibited that our algorithm is no better than 6-competitive [CL1]. They also have exhibited a 4-competitive algorithm that requires constant space and a constant decision time per request. It is not obvious, however, how to generalize their rule for more than 2 servers. Our rule has a natural formulation for more than two servers and we conjecture that this algorithm is $O(K)$-competitive when there are K servers.

The rule is a modified version of the balance algorithm. The balance algorithm sends the server that minimizes the quantity: (total cost incurred so far by that server + the cost of moving that server to the next request). The balance algorithm has been shown to have an unbounded competitive ratio [MMS]. The algorithm that sends the closest server to service a request, and the algorithm that sends the server that has incurred the smallest total cost so far both have unbounded competitive ratios as well. The new rule, which we call $BALANCE2$, sends the server that minimizes the quantity: (total cost incurred so far by that server + twice the cost of moving that server to the next request). Surprisingly this rule, which is very similar to the other rules, is 10-competitive.

Until very recently, it was unknown whether there was an on-line algorithm for the K-server problem that is $f(K)$-competitive, where the function f is only a function of K in any metric space. This question was answered by Fiat, Rabani, and Ravid who give an algorithm for the K-server problem whose competitive ratio is a function only of K [FRR]. It is conjectured in [MMS] that there is an algorithm that is K-competitive in any metric space. Raghavan and Snir have investigated memoryless algorithms, which are algorithms for the K-server problem that maintain no state information [RS]. They give tight bounds for the competitive ratio for a randomized memoryless on-line algorithm for both the paging problem and the related weighted cache problem. They also exhibit a randomized memoryless algorithm for the K-server problem that has a competitive ratio of 2 and $(n-1)^2$ for the cases where the number of servers are 2 and $n-1$ respectively. (n is the number of points in the metric space.) Chrobak and Larmore have a 2-competitive 2-server algorithm that requires $O(t)$ arithmetic operations and $O(t)$ variables, where $t = \min\ \{n, |\sigma|\}$ [CL2]. Although the algorithm has the same competitive ratio and time and space requirements of the original [MMS] 2-server algorithm, their result demonstrates a new approach to the server problem.

2 Notation and the 2-Server Rule

We discuss the performance of the $BALANCE2$ and the optimal algorithm (also denoted OPT) with respect to an arbitrary fixed sequence of requests in a fixed metric space. The optimal algorithm knows the entire sequence in advance. On the other hand, $BALANCE2$ is an on-line algorithm, and must decide which server to move to the current request without any information about future requests, including the number of future requests.

The cost that $BALANCE2$ incurs in servicing requests i through j (inclusive) is denoted $\mathcal{C}_{BAL2}[i,j]$. $\mathcal{C}_{OPT}[i,j]$ denotes the cost of the optimal algorithm on requests i through j. $BALANCE2$'s 2 servers are named a and b and OPT's servers are named x and y. We use superscripts to distinguish the distance travelled by a particular server; for example, $\mathcal{C}^a[i,j]$ denotes the total cost incurred by $BALANCE2$'s server a in servicing requests i through j. Thus, $\mathcal{C}_{BAL2}[i,j] = \mathcal{C}^a[i,j] + \mathcal{C}^b[i,j]$ and $\mathcal{C}_{OPT}[i,j] = \mathcal{C}^x[i,j] + \mathcal{C}^y[i,j]$.

$BALANCE2$ decides which server to send to service the $i+1^{st}$ request in the sequence as follows. Let d_a denote the distance after the i^{th} request between a and request $i+1$, and d_b denote the distance after the i^{th} request between b and request $i+1$. The algorithm compares the two quantities:

$$\mathcal{C}^a_{BAL2}[1,i] + 2d_a \qquad\qquad \mathcal{C}^b_{BAL2}[1,i] + 2d_b$$

If the former is smaller, $BALANCE2$ sends server a to service request $i+1$. If the latter is smaller, $BALANCE2$ sends server b. If they are equal, $BALANCE2$ makes an arbitrary choice. $BALANCE2$ just has to maintain one variable with the value $\mathcal{C}^a_{BAL2}[1,i] - \mathcal{C}^b_{BAL2}[1,i]$.

3 The Main Theorem

The following theorem bounds the cost of the algorithm $BALANCE2$ with respect to the optimal algorithm.

Theorem 1 *$\mathcal{C}_{BAL2}[1,m] \leq 10 \cdot \mathcal{C}_{OPT}[1,m]$, for any sequence of m requests.*

Proof:

To prove the theorem we introduce the following definitions:

After the i^{th} request is serviced, both $BALANCE2$ and the optimal algorithm have a server at the location of the i^{th} request. The distance between the other two servers, (the servers of $BALANCE2$ and OPT that did not service the i^{th} request) is called the *configuration distance* and is denoted $\Phi(i)$. $\Phi(i)$ is always nonnegative. By our assumption that the initial configurations of $BALANCE2$ and OPT are the same, $\Phi(0) = 0$. The figure below shows a sample snapshot of both algorithms after time i and the corresponding configuration distance. The point labeled r_i is the point of the i^{th} request. Because of the triangle inequality, the configuration distance is the minimum distance the servers of algorithm $BALANCE2$ have to move in order to be in the same position as the optimal algorithm.

We say that server a is *ahead* of server b at time t if $\mathcal{C}^a[1,t] \geq \mathcal{C}^b[1,t]$, otherwise it is *behind*. a moves ahead of b at the instant that the total cost incurred by a is more than that for b; this can occur when a is in the middle of moving to service a request. Let $\overline{\mathcal{C}}^a[i,j]$ denote the cost incurred by a from request i through request j while it is ahead. Similarly for $\overline{\mathcal{C}}^b[i,j]$. Let $\overline{\mathcal{C}}_{BAL2}[i,j]$ be defined to be $\overline{\mathcal{C}}^a[i,j] + \overline{\mathcal{C}}^b[i,j]$.

Lemma 2 $\overline{\mathcal{C}}_{BAL2}[1,j] = \max\{\mathcal{C}^a[1,j], \mathcal{C}^b[1,j]\}$

Proof: By induction on the number of times a server moves ahead of another. Suppose that after request j, server a is ahead. Let i be the last request where b is ahead. By the inductive hypothesis, $\overline{\mathcal{C}}_{BAL2}[1,i] = \max\{\mathcal{C}^a[1,i], \mathcal{C}^b[1,i]\} = \mathcal{C}^b[1,i]$. a moves to service request $i+1$ and after the request is ahead of b. $\overline{\mathcal{C}}_{BAL2}[i+1,i+1] = \overline{\mathcal{C}}^a[i+1,i+1]$ since server a is the only server moving to request $i+1$. And $\overline{\mathcal{C}}^a[i+1,i+1] = \mathcal{C}^a[1,i+1] - \mathcal{C}^b[1,i]$. Putting it all together, we get that $\overline{\mathcal{C}}_{BAL2}[1,i+1] = \overline{\mathcal{C}}_{BAL2}[1,i] + \overline{\mathcal{C}}_{BAL2}[i+1,i+1] = \mathcal{C}^a[1,i+1] = \max\{\mathcal{C}^a[1,i], \mathcal{C}^b[1,i]\}$. Since server a is ahead from time $i+1$ to time j, $\overline{\mathcal{C}}_{BAL2}[i+1,j] = \mathcal{C}^a[i+1,j]$. So $\overline{\mathcal{C}}_{BAL2}[1,j] = \mathcal{C}^a[1,j] = \max\{\mathcal{C}^a[1,j], \mathcal{C}^b[1,j]\}$ $\square$

Corollary 3 $\mathcal{C}_{BAL2}[1,j] \leq 2 \cdot \overline{\mathcal{C}}_{BAL2}[1,j]$

We analyze the performance of $BALANCE2$ by examining sequences of consecutive requests, called *epochs*, that $BALANCE2$ services with the same server. Every time $BALANCE2$ changes servers, a new epoch begins. Formally, $[i,j]$ is an *epoch* if $BALANCE2$ uses the same server to service requests i through j and uses a different server to service requests $i-1$ and $j+1$. The servers that services requests i through j is the *active* server in epoch $[i,j]$. We prove the following lemma:

Lemma 4 *Let* $[i,j]$ *be an epoch. Then*

$$\overline{\mathcal{C}}_{BAL2}[i,j] + \Phi(j) - \Phi(i-1) \leq 5 \cdot \mathcal{C}_{OPT}[i,j].$$

Theorem 1 follows from lemma 4 because by letting $t_1, \ldots t_s$ be the first request in each epoch, $(t_{s+1} = m+1)$, we have:

$$\begin{aligned}
\mathcal{C}_{BAL2}[1,m] &\leq 2 \cdot \overline{\mathcal{C}}_{BAL2}[1,m] \\
&= 2\,[\Phi(0) - \Phi(m)] \\
&+ 2 \sum_{k=1}^{s} \left[(\overline{\mathcal{C}}_{BAL2}[t_k, t_{k+1}-1] - \Phi(t_k - 1) + \Phi(t_{k+1} - 1)) \right] \\
&\leq 2 \sum_{k=1}^{s} 5 \cdot \mathcal{C}_{OPT}[t_k, t_{k+1}-1] \qquad \text{(by Lemma 4)} \\
&\leq 10 \cdot \mathcal{C}_{OPT}[1,m]
\end{aligned}$$

Proof of Lemma 4: We call the active server a, and the other server b. Recall that OPT's servers are named x and y. A snapshot of the servers after request $i-1$ is shown below. Without loss of generality, x services the $i-1^{st}$ request for the optimal algorithm.

The proof of the lemma breaks down into cases.

Case I: The active server (server a) is ahead at time j.

Since a is ahead at the end of the epoch, by lemma 2,

$$\overline{\mathcal{C}}_{BAL2}[i,j] = \overline{\mathcal{C}}_{BAL2}[1,j] - \overline{\mathcal{C}}_{BAL2}[1,i-1]$$

$$= \mathcal{C}^a[1,j] - max\{\mathcal{C}^a[1,i-1], \mathcal{C}^b[1,i-1]\}.$$

We prove the slightly stronger result that

$$\mathcal{C}^a[1,j] - max\{\mathcal{C}^b[1,i-1], \mathcal{C}^a[1,i-1]\} - \Phi(i-1) + \Phi(j) \leq 3 \cdot \mathcal{C}_{OPT}[i,j]$$

There are three subcases for **Case I**.

Case Ia: OPT uses server y to services all the requests in epoch $[i,j]$.

Servers b and x service request $i-1$, and a and y service all the i^{th} through j^{th} requests. Then by the triangle inequality,

$$\mathcal{C}^a[i,i] \leq \mathcal{C}^y[i,i] + \Phi(i-1).$$

Because y and a move together for the rest of the epoch,

$$\Phi(j) = 0 \text{ and } \mathcal{C}^a[i+1,j] = \mathcal{C}^y[i+1,j].$$

$$\begin{aligned}
\mathcal{C}^a[i,j] &= \mathcal{C}^a[i,i] + \mathcal{C}^a[i+1,j] \\
&\leq \Phi(i-1) - \Phi(j) + \mathcal{C}^y[i,i] + \mathcal{C}^y[i+1,j] \\
&= \Phi(i-1) - \Phi(j) + \mathcal{C}^y[i,j] \\
&\leq \Phi(i-1) - \Phi(j) + \mathcal{C}_{OPT}[i,j]
\end{aligned}$$

So we have

$$\begin{aligned}
\mathcal{C}_{OPT}[i,j] &\geq \mathcal{C}^a[i,j] - \Phi(i-1) + \Phi(j) \\
&= (\mathcal{C}^a[1,j] - \mathcal{C}^a[1,i-1]) - \Phi(i-1) + \Phi(j) \\
&\geq \mathcal{C}^a[1,j] - max\{\mathcal{C}^a[1,i-1], \mathcal{C}^b[1,i-1]\} - \Phi(i-1) + \Phi(j)
\end{aligned}$$

Case Ib: OPT uses server x to service request j, the last request in the epoch.

Let l be the last time before j that OPT uses y to service a request. If the optimal algorithm never uses y in the epoch, then we set $l = i-1$. For now we assume that $l > i-1$. The case where $l = i-1$ is slightly different than the case where $l > i-1$. We will not examine the case where $l = i-1$, but the analysis is similar.

We divide the epoch into two parts: first we analyze moves i through $l+1$ and then moves $l+2$ through the end of the sequence.

Moves i through $l+1$: The position of the servers after request l and after $l+1$ are shown below.

Since $BALANCE2$ used server a instead of b to service the $l+1^{st}$ request, $\mathcal{C}^b[1,l]+2\beta \geq \mathcal{C}^a[1,l]+2\alpha$. Since b has not moved, $\mathcal{C}^b[1,l]=\mathcal{C}^b[1,i-1]$, and $2\beta \geq \mathcal{C}^a[1,l]-\mathcal{C}^b[1,i-1]+2\cdot\alpha$. Because a moves a distance α to service the $l+1^{st}$ request, $\mathcal{C}^a[1,l]+\alpha=\mathcal{C}^a[1,l+1]$, and $2\beta \geq \mathcal{C}^a[1,l+1]-\mathcal{C}^b[1,i-1]+\alpha$. Adding a β to both sides gives, $3\beta \geq \mathcal{C}^a[1,l+1]-\mathcal{C}^b[1,i-1]+\beta+\alpha$. Since x and b started out at time i together and b has not moved, we know that $\beta \leq \mathcal{C}^x[i,l+1]$ and thus $3\mathcal{C}^x[i,l+1] \geq \mathcal{C}^a[1,l+1]-\mathcal{C}^b[1,i]+\alpha+\beta$. Finally, since $\Phi(l+1) \leq \beta+\alpha$,

$$3\cdot\mathcal{C}^x[i,l+1] \geq \mathcal{C}^a[1,l+1]-\mathcal{C}^b[1,i]+\Phi(l+1) \tag{1}$$

Moves $l+2$ through j : For the rest of the sequence x and a are moving together, so $\mathcal{C}^x[l+2,j]=\mathcal{C}^a[l+2,j]$ and $\Phi(l+1)=\Phi(j)$. So

$$\mathcal{C}^x[l+2,j]+3\mathcal{C}^x[i,l+1] \geq$$

$$\mathcal{C}^a[l+2,j]+\mathcal{C}^a[1,l+1]-\mathcal{C}^b[1,i-1]+\Phi(l+1)$$

And finally, putting the whole sequence together,

$$3\cdot\mathcal{C}^x[i,j] \geq \mathcal{C}^a[1,j]-\mathcal{C}^b[1,i-1]+\Phi(j)$$

$$\geq \mathcal{C}^a[1,j]-max\{\mathcal{C}^a[1,i-1],\mathcal{C}^b[1,i-1]\}-\Phi(i-1)+\Phi(j)$$

Case Ic: OPT uses server y to service request j and $\exists\ h$ such that $i \leq h < j$ and OPT uses x to service request h.

Let h be the the last time in the epoch that OPT uses server x to service a request. Then, OPT uses server y to service requests $h+1$ through j. The following picture shows the configuration just after time h and after time $h+1$.

The analysis for requests i through h of case Ib is the same as the analysis for requests i through $l+1$ of case Ic, so from Equation 1,

$$3 \cdot \mathcal{C}^x[i,h] \geq \mathcal{C}^a[1,h] - \mathcal{C}^b[1,i-1] + \alpha + \beta. \tag{2}$$

Since y and a are moving together for requests $h+1$ through j, $\Phi(j) = \beta$. By the triangle inequality, $\mathcal{C}^a[h+1,h+1] \leq \alpha + \mathcal{C}^y[h+1,h+1]$ and $\mathcal{C}^a[h+2,j] = \mathcal{C}^y[h+2,j]$. This yields that $\mathcal{C}^a[h+1,j] - \alpha \leq \mathcal{C}^y[h+1,j]$. Combining with equation 2,

$$\begin{aligned} 3 \cdot \mathcal{C}_{OPT}[i,j] &\geq \mathcal{C}^a[1,j] - \mathcal{C}^b[1,i-1] + \Phi(j) \\ &\geq \mathcal{C}^a[1,j] - max\{\mathcal{C}^a[1,i-1], \mathcal{C}^b[1,i-1]\} - \Phi(i-1) + \Phi(j) \end{aligned}$$

Case II The active server is behind at time j.

The active server in epoch $[i,j]$ is behind during the entire epoch. Therefore, $\overline{\mathcal{C}}_{BAL2}[i,j] = 0$. We prove the following:

$$-\Phi(i-1) + \Phi(j) \leq 5 \cdot \mathcal{C}_{OPT}[i,j]$$

There are three subcases for **Case II**.

Case IIa: OPT uses server y to service request j, the last request in the epoch. Below are pictured the position of the servers before i and after j.

$\Phi(j) \leq \mathcal{C}^x[i,j]$ since b is stationary throughout the epoch.

Case IIb: OPT uses server x to services all the requests in epoch $[i, j]$. Below are pictured the position of the servers before i and after i.

$\Phi(i)$ is the distance between b and y. Thus by the triangle inequality,

$$\Phi(i) \leq \mathcal{C}_{OPT}[i, i] + \mathcal{C}_{BAL2}[i, i] + \Phi(i-1) \tag{3}$$

Because algorithm $BALANCE2$ chooses to move a instead of b to service the i^{th} request, we know that

$$\mathcal{C}^a[1, i-1] + 2 \cdot \mathcal{C}_{BAL2}[i, i] \leq \mathcal{C}^b[1, i-1] + 2 \cdot \mathcal{C}^x[i, i]$$

$$\mathcal{C}_{BAL2}[i, i] \leq \mathcal{C}_{OPT}[i, i] + (\mathcal{C}^b[1, i-1] - \mathcal{C}^a[1, i-1])/2 \tag{4}$$

Since x moved from b to a in the epoch $[i, j]$, for any point, the difference between (the distance between that point and a) and (the distance between that point and b) is bounded by $\mathcal{C}^x[i, j]$. Algorithm $BALANCE2$ decides to send b to service the $j + 1^{st}$ request. Let d_a be the distance from a to the $j + 1^{st}$ request and d_b the distance from b to the $j + 1^{st}$ request. We have $\mathcal{C}^b[1, j] + 2 \cdot d_b \leq \mathcal{C}^a[1, j] + 2 \cdot d_a$ and

$$(\mathcal{C}^b[1, j] - \mathcal{C}^a[1, j])/2 \leq d_a - d_b \leq \mathcal{C}^x[i, j]. \tag{5}$$

Since x and a move together on requests $i + 1$ through j, $\mathcal{C}_{OPT}[i + 1, j] = \mathcal{C}^x[i + 1, j] = \mathcal{C}^a[i + 1, j]$.

$$\mathcal{C}^a[1, j] - \mathcal{C}^a[1, i-1] = \mathcal{C}_{BAL2}[i, j] = \mathcal{C}_{OPT}[i + 1, j] + \mathcal{C}_{BAL2}[i, i] \tag{6}$$

Since b does not move in the epoch, $\mathcal{C}^b[1, i-1] = \mathcal{C}^b[1, j]$, and combining this fact with Equation 6,

$$\mathcal{C}^b[1, i-1] - \mathcal{C}^a[1, i-1] = \mathcal{C}^b[1, j] - \mathcal{C}^a[1, j] + \mathcal{C}_{OPT}[i + 1, j] + \mathcal{C}_{BAL2}[i, i] \tag{7}$$

Combining Equations 4, 5, and 7,

$$\mathcal{C}_{BAL2}[i,i]/2 \leq 2 \cdot \mathcal{C}_{OPT}[i,j] \tag{8}$$

and from Equations 3 and 8, and the fact that $\Phi(i) = \Phi(j)$,

$$\Phi(j) - \Phi(i-1) \leq 5 \cdot \mathcal{C}_{OPT}[i,j]$$

Case IIc: OPT uses server x to service request j and $\exists\, h$ such that $i \leq h < j$ OPT uses y to service request h.

Let l be the the last time in the epoch that OPT uses server y. The position of the servers before and after time $l+1$ is pictured below.

Since OPT uses server x to service all the requests $l+1$ through j, we can use the analysis for requests i through j in case IIb. The difference, however, is that before request $l+1$, y and a are in the same place (in case IIb, the distance between the two is bounded by $\Phi(i-1)$). Also, x and b do not start out in the same place, but the distance between the two is bounded by $\mathcal{C}^x[i,l]$. Thus Equation 3 is replaced by

$$\Phi(j) \leq \mathcal{C}_{OPT}[i,l+1] + \mathcal{C}_{BAL2}[l+1,l+1] \tag{9}$$

The rest of the analysis through Equation 8 holds, except the sequence starts at request $l+1$ instead of i. Thus Equation 8 is replaced by

$$\mathcal{C}_{BAL2}[l+1,l+1]/2 \leq 2 \cdot \mathcal{C}_{OPT}[l+1,j] \tag{10}$$

Putting Equations 9 and 10 together, and the fact that $\Phi(l+1) = \Phi(j)$,

$$\Phi(j) \leq 5 \cdot \mathcal{C}_{OPT}[i,j]$$

$\square$

References

[BKT] Berman, P., Karloff, H., Tardos, G., 'A Competitive Three Server Algorithm', to appear in *Symposium on Discrete Algorithms* 1990.

[CL1] Chrobak, M., Larmore, L., 'On Fast Algorithms for Two Servers', to appear in *Journal of Algorithms*.

[CL2] Chrobak, M., Larmore, L., 'A New Approach to the Server Problem', to appear in *SIAM Jounal on Discrete Algorithms*.

[FRR] Fiat, A., Rabani, Y., Ravid, R., 'Competitive k-server Algorithms.' *31st Annual Symposium on the Foundations of Computer Science*, 1990.

[MMS] Manasse, M. S., McGeoch, L. A. and Sleator, D. D., 'Competitive algorithms for on-line problems'. In *20th Symposium on the Theory of Computing*, pages 322-333, Chicago, 1988.

[RS] Raghavan, P., Snir, M., 'Memory Versus Randomness in On-line Algorithms'. IBM Research Report, revision of the paper in *ICALP*, 1989.

[ST] Sleator, D. D., Tarjan, R. E., 'Amortized Efficiency of List Update and Paging Rules'. *CACM*, 28(2), 202-208, 1985.