# A Note on the Server Problem and a Benevolent Adversary

Marek Chrobak and Lawrence L. Larmore
Department of Computer Science
University of California
Riverside, CA 92521

October 4, 1996

**Abstract**

We consider the *k-server problem*, with the additional assumptions that (1) the adversary makes only lazy moves (if it has a free server, not matched with our server, then it puts a request on such a server without moving it), and (2) whenever all our servers are matched with his servers, then he announces it to our algorithm. We show a $k$-competitive algorithm against such an adversary. As with the original server problem, no better than a $k$-competitive on-line algorithm is possible in this model.

**Keywords:** On-line algorithms, competitive analysis, the server problem.

## 1 The Server Problem

The *k-server problem* is defined as follows: Let $M$ be a metric space, in which we have $k$ servers that occupy points of $M$. At each time step we are given a request point $r \in M$, and we have to move one of our servers to $r$. Note that the decision of which server to move has to be made *on-line*, without the knowledge about the future requests. Our goal is to minimize the total distance traveled by our servers. Since it is known that no on-line algorithm can guarantee an optimal schedule, it is natural to look for algorithms that only approximate the optimal schedule as close as possible.

Let $\mathcal{A}$ be an on-line server algorithm. For a given initial configuration $K^0 \in \Lambda^k M$ of $k$ servers ($\Lambda^k X$ = the collection of all multisets of cardinality $k$ drawn from $X$) and a sequence of requests $\varrho \in M^*$ ($X^*$ = the set of finite sequences drawn from $X$), let $cost_{\mathcal{A}}(K^0, \varrho)$ be the total cost of $\mathcal{A}$ on $\varrho$ when starting from $K^0$, and $cost_{opt}(K^0, \varrho)$ the cost of the corresponding optimal schedule. (For simplicity, we will often omit the parameters $K^0$ and $\varrho$.) Then, for a given constant $c$, $\mathcal{A}$ is called *c-competitive* if for any $K^0$ there is a constant $\lambda(K^0)$ such that, for any $\varrho \in M^*$,

$$cost_{\mathcal{A}}(K^0, \varrho) \leq c \cdot cost_{opt}(K^0, \varrho) + \lambda(K^0)$$

Manasse, McGeoch and Sleator [5] proved that there are no $c$-competitive $k$-server algorithms for $c < k$ in any metric space with at least $k+1$ points. The problem of whether there is an on-line algorithm that achieves $c = k$ for each $k$ is open. The famous "$k$-server conjecture" from [5] states that this is indeed true. Up to now, it has been proven only for $k = 2$ in [5, 2]; and recently Fiat *et al* [4] announced the result that for each $k$ there is a $c$-competitive algorithm, where $c$ is a large constant. Some research has been done for some restricted metric spaces. It is known that there is a $k$-competitive algorithm for trees [1]. Coppersmith *et al* [3] have presented a randomized $k$-competitive algorithm for so-called *resistive* spaces. The reader is referred to [1, 2, 3, 4, 5] for more information about the problem and current progress.

## 2 Benevolent Adversary

Very often the server problem is stated in terms of a game between our $k$ servers and the adversary's $k$ servers. In this game the adversary makes the requests and serves them with his servers, trying to maximize the ratio $cost_{\mathcal{A}}/cost_{adv}$ (where $cost_{adv}$ denotes the adversary's cost). This setting is used both for proving lower and upper bounds on the competitiveness constant. In case of the lower bound proof we show an adversary strategy that for any possible algorithm $\mathcal{A}$ forces the ratio $cost_{\mathcal{A}}/cost_{adv}$ to be appropriately large. In upper bound proofs, one has to present a strategy for *our* servers such that, no matter what the adversary does, the inequality $cost_{\mathcal{A}} \leq c \cdot cost_{adv} + \lambda(K^0)$ remains true during the game.

Let us define now a slightly different game.

**Rule 0:** Initially all our servers are on different points and are matched with the adversary's servers.

**Rule 1:** If the adversary has any server that is not matched with any of our servers, then he puts the new request on such a server without moving it (a so-called *lazy* move).

**Rule 2:** When all our servers are matched with the adversary's servers, then the adversary announces it to us, and then he can choose a new request point $r$ and move one of his servers to $r$.

The idea of considering these restrictions of the adversary behavior is due to Tom Payne. An adversary that plays according to rules 0-2 above is called *benevolent*. It is not hard to see that the lower bound of $k$ on the competitiveness constant remains true even if the adversary is benevolent. This observation is the main motivation of this work, since it implies that if the $k$-server conjecture is false, the proof of this will have to involve more sophisticated methods than those used up to now.

We show in this section an algorithm whose competitiveness ratio against such a benevolent adversary is $k$.

For simplicity, denote the distance between points $x, y$ by $xy$.

**Algorithm $\mathcal{A}$:** The game proceeds in "phases," where, at the beginning and at the end of each phase, the adversary's servers and the algorithms servers lie on the same points. At the beginning of one phase, let $K \in \Lambda^k M$ be the configuration of our servers, which of course is also the configuration of the adversary's servers, and that the adversary requests a new point $r \notin K$.

In a precomputation step, we will order our servers, *i.e.*, we will let $K = \{x_1, \dots x_k\}$, and, during that phase, our servers will be moved in that order of their indices. There will be $p$ rounds in one phase, for $1 \le p \le k$, and servers $x_1, \dots x_p$ will be moved.

The indices of the servers are chosen according to the following rules:

- The first server $x_1$ is chosen to be the one closest to $r$.
- For $i > 1$, $x_i$ is chosen so as to minimize $x_{i-1}x_i + rx_i$.

We move as follows: to serve the request at $r$, move the server at $x_1$ to $r$. The second request, if a lazy call, must be at $x_1$, since that is now the only possible position where the adversary could have a server unmatched by one of ours. To meet this request, move the server from $x_2$, and the next lazy request can only be at $x_2$. Generally, the $i^{\text{th}}$ lazy request must be at $x_{i-1}$, and we service it by moving the server from $x_i$.

After $p$ requests, for some $1 \le p \le k$, we will have servers on $r, x_1, \dots, x_{p-1}, x_{p+1}, \dots, x_k$, and the benevolent adversary will announce that a match has been achieved, thus ending the phase.

**Theorem 1** *Algorithm $\mathcal{A}$ is $k$-competitive against the benevolent adversary.*

*Proof:* We define a *potential function* $\Phi$ to be the sum of all distances between the $k$ points occupied by our servers, that is, $\Phi = \sum_{1 \le i < j \le k} x_i x_j \ge 0$. All we need to show is that our cost in one phase, plus the potential at the end of that phase, does not exceed $k$ times the adversary cost of the phase plus the potential at the beginning of the phase, *i.e.*:

*Claim:* Over one phase, $\Delta cost_{\mathcal{A}} + \Delta\Phi \le k \cdot \Delta cost_{adv}$.

It is notationally convenient to write $x_0 = r$.

Let $p$ be the number of rounds in the phase. Thus, the adversary's cost is $x_0 x_p$, while our cost is $\sum_{i=1}^{p} x_{i-1} x_i$.

The potential before the phase is the total edge weight of a clique of size $k$, and the potential after the phase is the total edge weight of a slightly different clique obtained by deleting $x_p$ and inserting $x_0$. The difference of potentials is then the weights of all edges incident to $x_0$, minus the weight of all edges incident to $x_p$. We can write (note the cancellation of $x_0x_p$):

$$\Delta\Phi = \sum_{i\neq 0} x_0x_i - \sum_{i\neq p} x_ix_p$$

For each $i$, $(x_0x_i - x_ix_p - x_0x_p) \leq 0$ by the triangle inequality, while for each $i \leq p$, $(x_{i-1}x_i + x_0x_i - x_{i-1}x_p - x_0x_p) \leq 0$, since otherwise $x_p$ would be a better choice for $x_i$ in the preprocessing step. Thus:

$$\begin{aligned}
\Delta cost_{\mathcal{A}} + \Delta\Phi - k \cdot \Delta cost_{adv} &= \sum_{i=1}^{p} x_{i-1}x_i + \sum_{i\neq 0} x_0x_i - \sum_{i\neq p} x_ix_p - k \cdot x_0x_p \\
&= \sum_{i=1}^{p}(x_{i-1}x_i + x_0x_i - x_{i-1}x_p - x_0x_p) + \sum_{p+1}^{k}(x_0x_i - x_ix_p - x_0x_p) \\
&\leq 0
\end{aligned}$$

This verifies the claim. Finally, by summing over all phases, we obtain:

$$cost_{\mathcal{A}}(K^0, \varrho) \leq c \cdot cost_{opt}(K^0, \varrho) + \Phi(K^0)$$

The adversary can gain nothing by halting before completion of a phase, since the lazy requests are "free" to him. □

*Remark:* It is not hard to find adversary strategies to show, that if we modify the preprocessing step by either minimizing only $rx_i$ or only $x_{i-1}x_i$, then Theorem 1 is not true for $k \geq 3$ (although it remains true for $k = 2$).

**Added during revision**. During revision, we noticed that the algorithm presented here is an example of the general "*Work Function Algorithm*" presented at the *DIMACS Workshop on On-Line Algorithms* on February 11, 1991, and which was unknown to us at the time the original version of this paper was written. We will give more details in the written version of that presentation.

Also, a referee has kindly pointed out the similarity between the results in this manuscript and a result by Raghavan and Snir [6], in which they prove that there is a $k$-competitive randomized against an adversary that is very similar to the benevolent adversary described in this paper.

## References


[1] M. Chrobak and L. L. Larmore, An optimal algorithm for the server problem on trees, *SIAM Journal of Computing*, to appear.

[2] M. Chrobak and L. L. Larmore, A new approach to the server problem, *SIAM Journal on Discrete Mathematics*, to appear.

[3] D.Coppersmith, P.G.Doyle, P. Raghavan and M. Snir, Random walks on weighted graphs and applications to on-line algorithms, *Proceedings 22nd ACM STOC* (1990) pp. 369–378.

[4] A.Fiat, Y.Rabani, Y.Ravid, Competitive $k$-server algorithms, *Proc. 31st Symposium on Foundations of Computer Science*, (1990), pp. 454–463.

[5] M.Manasse, L.A.McGeoch, D.Sleator, Competitive algorithms for server problems, *Proc. 20th ACM Symposium on Theory of Computing*, (1988), pp. 322–333, Also, *J. of Algorithms* **11** (1990) pp. 208–230.

[6] P. Raghavan and M. Snir, Memory versus randomization in on-line algorithms, *Proc. of 16th International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science vol. 372, Springer-Verlag, (1989), pp. 687–703.