A New Approach to the Server Problem

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

October 4, 1996

Abstract

A new method for dealing with the server problem is proposed. The technique consists of embedding the given metric space M into a bigger metric space $cl(M)$ called the *closure* of M, and allowing our servers to move in $cl(M)$. We show how this technique can be applied to give a new optimal algorithm for two servers.

1 Introduction

The k server problem can be formulated as follows: Let M be a metric space, in which we have k mobile servers that can occupy points of M. Initially all servers are on some k specified points of M (called the *initial configuration*). At each time step we are given a request, specified by a location $r \in M$, and we have to choose which server to move to r to "serve" the request. Our measure of cost is the distance traveled by our servers, and the task is to design algorithms that minimize that cost.

The problem is that the requests have to be served on-line, that is, the choice of the server at the current step cannot depend on the future requests. It is known (see [2]) that if we were given the whole sequence of requests off-line, in advance, then an optimal schedule can be constructed efficiently in polynomial time. However, as it was shown in [6], no on-line algorithm can guarantee to yield a schedule that is better than k times the optimal one. Therefore the on-line restriction is essential.

Recently, the research on on-line algorithms concentrates on so-called *competitive* algorithms. Let $cost_{opt}(K, \sigma)$ be the optimal cost of servicing the sequence of requests σ when the servers start from configuration K. By $cost_{\mathcal{A}}(K, \sigma)$ we denote the respective cost of an on-line algorithm $\mathcal{A}$. An algorithm $\mathcal{A}$ is called *c-competitive* if for every initial configuration K there is a constant $b(K)$

such that for arbitrary sequence of requests σ we have

$$cost_{\mathcal{A}}(K,\sigma) \leq c \cdot cost_{opt}(K,\sigma) + b(K).$$

(We will often omit the parameters K and σ when they are understood.) In other words, for each initial configuration, the ratio $cost_{\mathcal{A}}/cost_{opt}$ approaches c if $cost_{opt}$ is large.

It is not known whether there is an on-line algorithm that achieves $c = k$ for each k. The famous "k-server conjecture" of Manasse, McGeoch and Sleator [6] states that this is indeed true. Up to now, it has been proven only for $k = 2$ in [6]. Irani and Rubinfeld [5] proved that a version of a balancing algorithm is 10-competitive for two servers. Some work has also been done on randomized algorithms. Raghavan and Snir [7] presented a randomized memoryless algorithm for two servers whose competitiveness constant is between 3 and 6. Berman *et al* [1] proved that a similar algorithm is competitive for three servers but the competitiveness constant is unknown.

In the general case, the solution is known only for some specific metric spaces. In [3] a k-competitive algorithm is given for trees, and it can be applied to all metric spaces that can be isometrically embedded in a tree (for example the weighted cache problem, see [2, 7]). Coppersmith *et al* [4] gave a randomized k-competitive algorithm for a broad class of metric spaces, that also includes trees.

In this note we give a new on-line algorithm for 2 servers. Our algorithm is 2-competitive, and thus optimal. We employ some new techniques that we believe can be extended to three or more servers. Our method is to allow our servers to move in a bigger metric space $cl(M)$, called the *closure* of M. All information about the past that the algorithm uses is recorded by the current positions of our servers in $cl(M)$. This allows us to use a simple and intuitive potential argument in the proof of competitiveness.

2 The 2-Server Algorithm

Let M be the given metric space. For simplicity we assume that M is finite. At the end of this section we describe how to implement the algorithm on infinite metric spaces. By $\|xy\|$ we denote the distance between points $x, y \in M$.

In the proof we look at the computation as a game between our servers s_1, s_2, and adversary's servers a_1 and a_2, and compare our cost $cost_{\mathcal{A}}$ to the adversary's cost $cost_{adv}$. Our goal is to show that, independently of the adversary's strategy, the inequality $cost_{\mathcal{A}} \leq c \cdot cost_{adv} + b$ holds. Each round is thought of as consisting of two steps: first the adversary moves a server and puts a request on its new position, and then our algorithm satisfies the request. Note that it is sufficient to show that $cost_{\mathcal{A}} \leq c \cdot cost_{adv} + b$ holds for arbitrary sequence of adversary's moves, since one of the adversary schedules will correspond to the optimal schedule.

We give first an intuitive description of our method. In the algorithm, we visualize the computation as taking place in a bigger metric space $cl(M)$ called the *closure* of M. Each point $u \in cl(M)$ is defined by a set of distances between u and the points in M, in such a way that the triangle inequality is preserved. We allow our servers to move through $cl(M)$, while the requests and adversary's servers are always in M. For simplicity we also use notation $\|uv\|$ to denote the distance between $u, v \in cl(M)$.

Our algorithm can be informally described as follows. Given a request on a point $r \in M$, we look at the points s_1, s_2 and r (for simplicity, s_i and a_i are also used to denote the current position of the corresponding server). If $\|rs_i\| + \|s_i s_j\| = \|rs_j\|$, for $i \neq j$, then we move s_i to r. Otherwise, we do the following: both servers move at the same speed towards the request, and simultaneously each of them moves at the same speed towards the other. Thus, if d is the speed of our servers (distance traveled in a unit of time), then after a unit of time both servers will get closer to the request by d, and closer one to another by $2d$. Eventually one of our servers will be ahead of the other one, in the sense that the first case considered above will apply.

Formal description of the algorithm. By R^+ we denote the set of non-negative reals. For $a, b, c \in R^+$, we define the predicate $\triangle(a, b, c)$ to mean that the numbers a, b, c satisfy the triangle inequalities: $a+b \geq c$, $a+c \geq b$ and $b+c \geq a$. The *closure* of M is the set of all functions $u : M \to R^+$ such that for any $x, y \in M$, $\triangle(\|xy\|, u(x), u(y))$ holds. We set $\|uv\| = \max_{x \in M} |u(x) - v(x)|$, making $cl(M)$ a metric space. M can be isometrically embedded into $cl(M)$ by mapping each $x \in M$ to u_x, where $u_x(y) = \|xy\|$ for each $y \in M$. We will abuse notation by identifying x with u_x whenever convenient, allowing us to think of M as a subspace of $cl(M)$.

We have to formalize now the notion of moving a server, or both servers through $cl(M)$. First some intuition. Suppose that the current location of s_i is u, and we want to move it towards v by some distance d. If $\|uv\| = e$, then for each $x \in M$ we have $|u(x) - v(x)| \leq e$, and there is some $y \in M$ such that $|u(y) - v(y)| = e$. Suppose that d is very small. Then, in order to move s_i as needed, for each such y we can update $s_i(y)$ as follows. If $u(y) - v(y) = e$ then $s_i(y) \leftarrow s_i(y) - d$, and if $u(y) - v(y) = -e$ then $s_i(y) \leftarrow s_i(y) + d$. Since M is assumed to be finite, and since d is sufficiently small, all triangle inequalities will be preserved, that is the new position of s_i is a valid point in $cl(M)$. And now the distance of s_i to v decreased by d, and its distance to u is equal to d. If d is larger, then this movement can be divided into a sequence of steps as described above. At each consecutive step the set of y's for which $s_i(y)$ has to be updated may vary. In a similar way it is possible to define the movement of two servers in the first phase of the algorithm. In both cases, the whole process may be combined into one step. Since when one server moves, it always moves to a request point in M, this case is easy to formalize as one step. However, the first phase of our algorithm, when two servers move, ends when our servers may be in points that are not in M. Below we describe how those points can be determined.

We define now the function $step_d(u, v, w) \in cl(M)$, that determines the new position of a server whose current position is u and which is to be moved by d simultaneously towards v and w. For any $x \in M$, let $p(x)$ be the closest real number to $u(x)$ such that $|p(x) - v(x)| \leq \|uv\| - d$ and

$|p(x) - w(x)| \leq \|uw\| - d$. Then we set $step_d(u,v,w) = p$. In Lemma 1 below we show that the function *step* is well-defined and satisfies the needed properties.

Lemma 1 *Let $u, v, w \in cl(M)$, and $0 \leq d \leq \frac{1}{2}(\|uv\| + \|uw\| - \|vw\|)$. Let also $p = step_d(u,v,w)$ be defined as above. Then*

(a) *p is a well-defined element of $cl(M)$,*

(b) *$\|pu\| = d$,*

(c) *$\|pv\| = \|uv\| - d$ and $\|pw\| = \|uw\| - d$,*

(d) *Suppose that $d = \frac{1}{2}(\|uv\| + \|uw\| - \|vw\|)$. Then $\|wp\| + \|pv\| = \|wv\|$.*

Proof: (a) We first show that $p(x)$ is defined for each $x \in M$. Let I_x be the intersection of the closed interval of radius $\|uv\| - d$ centered at $v(x)$ and the closed interval of radius $\|uw\| - d$ centered at $w(x)$. I_x cannot be empty since the distance between those centers does not exceed the sum of those radii, *i.e.*, $|v(x) - w(x)| \leq \|vw\| \leq \|uv\| + \|uw\| - 2d$, and in fact, I_x must contain a non-negative real. Thus $p(x)$ exists. Write $I_x = [a_x, b_x]$. We next show that $p \in cl(M)$, *i.e.*, $|p(x) - p(y)| \leq \|xy\| \leq p(x) + p(y)$ for all $x, y \in M$. Fix some $x, y \in M$.

Claim A: $|a_x - a_y| \leq \|xy\|$ and $|b_x - b_y| \leq \|xy\|$.

Without loss of generality, $a_x \leq a_y$ and $a_x = v(x) - \|uv\| + d \geq w(x) - \|uw\| + d$. If $a_y = v(y) - \|uv\| + d$, then $|a_x - a_y| = a_y - a_x = v(y) - v(x) \leq \|xy\|$. Otherwise $a_y = w(y) - \|uw\| + d$, whence $|a_x - a_y| = a_y - a_x \leq a_y - (w(x) + \|uw\| - d) \leq w(y) - w(x) \leq \|xy\|$. The second inequality can be proven in a similar way.

Claim B: $b_x + a_y \geq \|xy\|$.

Without loss of generality, $a_y = v(y) - \|uv\| + d$. If $b_x = v(x) + \|uv\| - d$, then $b_x + a_y = v(x) + v(y) \geq \|xy\|$. Otherwise, $b_x = w(x) + \|uw\| - d$, and then $b_x + a_y \geq w(x) + \|uw\| + v(y) \geq w(x) + w(y) \geq \|xy\|$, using the definition of $\|uw\|$.

We now return to the proof of (a). We have established that p is well defined, but need to show that $p \in cl(M)$. Let $x, y \in M$. To show that $|p(x) - p(y)| \leq \|xy\|$, it suffices (by symmetry) to show that $p(y) - p(x) \leq \|xy\|$. If $p(x) \geq u(x)$ and $p(y) \leq u(y)$ we are done since $p(y) - p(x) \leq u(y) - u(x) \leq \|xy\|$. If $p(x) < u(x)$, then $p(x) = b_x$ and $p(y) \leq b_y$. By Claim A, $p(y) - p(x) \leq b_y - b_x \leq \|xy\|$. If $p(y) > u(y)$, then $p(y) = a_y$ and $p(x) \geq a_x$. By Claim A, $p(y) - p(x) \leq a_y - a_x \leq \|xy\|$.

We now show that $p(x) + p(y) \geq \|xy\|$. If $p(x) \geq u(x)$ and $p(y) \geq u(y)$ then we are done since $p(x) + p(y) \geq u(x) + u(y) \geq \|xy\|$. If $p(x) < u(x)$, then $p(x) = b_x$ and $p(y) \geq a_y$, so by Claim B, $p(x) + p(y) \geq b_x + a_y \geq \|xy\|$.

(b) We show first that $\|pu\| \leq d$. Fix some $x \in M$. Without loss of generality assume that $p(x) \leq u(x)$. Then $p(x) = b_x$. We can also assume that $p(x) = v(x) + \|uv\| - d$. Then $|p(x) - u(x)| = u(x) - p(x) = u(x) - v(x) - \|uv\| + d \leq d$.

Now we show that $\|pu\| \geq d$. Pick $x \in M$ such that $|v(x) - u(x)| = \|uv\|$. Suppose $u(x) \leq v(x)$. Then $u(x) = v(x) - \|uv\| \leq a_x - d$, so $p(x) = a_x$ whence $p(x) - u(x) \geq d$, hence $\|pu\| \geq d$. The case $u(x) \geq p(x)$ is proved similarly. Thus $\|pu\| = d$.

(c) By symmetry it is sufficient to prove only the first inequality. From the triangle inequality, $\|pv\| \geq \|uv\| - \|pv\| = \|uv\| - d$, so we need only show that $\|pv\| \leq \|uv\| - d$, by showing that $|p(x) - v(x)| \leq \|uv\| - d\|$ for all $x \in M$. This holds since $p(x) \in I_x$, which is centered at $v(x)$ and has radius $\|uv\| - d$.

(d) The proof is by simple calculation, using (c) and the definition of d: $\|wp\| + \|pv\| = \|uw\| + \|uv\| - 2d = \|vw\|$. □

Algorithm $\mathcal{A}$: Let the request be on r. Let $d = \frac{1}{2}\min\{\|s_i s_j\| + \|s_i r\| - \|s_j r\|\}$ where the minimum is over the two choices of assignment of $\{i, j\}$ to $\{1, 2\}$. For the sake of exposition it is convenient to present the move as if consisting of two phases, each possibly empty.

Phase 1:

(a) Move the first of our servers from s_1 to $s'_1 = \mathit{step}_d(s_1, s_2, r)$.

(b) Move the second of our servers from s_2 to $s'_2 = \mathit{step}_d(s_2, s'_1, r)$.

Phase 2:

If $\|s'_1 r\| \leq \|s'_2 r\|$, move the first of our servers from s'_1 to r. Otherwise, move the other server from s'_2 to r.

Important remark: Recall that we visualize the algorithm in such a way that our servers move through $cl(M)$, and we can charge our servers their cost in $cl(M)$. In reality, however, our servers remain in M but only remember their virtual positions in $cl(M)$, and move to the request point only when they actually serve the request. Thus the cost we charge to our servers in $cl(M)$ may be different than the real cost. The adversary's servers are always on points of M (although we do not really need it for the proof, we could as well allow him to request points of $cl(M)$).

Let $\mathit{cost}'_{\mathcal{A}}$ be the cost of our servers in $cl(M)$. By the triangle inequality in $cl(M)$, and because $M \subseteq cl(M)$, we have the following fact.

Fact 1 $\mathit{cost}_{\mathcal{A}} \leq \mathit{cost}'_{\mathcal{A}}$.

In order to prove that Algorithm $\mathcal{A}$ is 2-competitive, we define a potential function:

$$\Phi = 2\|M_{\min}\| + \|s_1 s_2\|,$$

where $M_{\min}$ is a minimum weight matching in the bipartite graph whose components are $\{s_1, s_2\}$, $\{a_1, a_2\}$ where the weights are equal to the distances between the servers. (Notice that Φ is defined in $cl(M)$). The same potential function was used in [4]. Recall now that we view the computation as a game, where each round consists of two steps: the adversary moves one of his servers to the request point, and then our algorithm serves the request. Consider a single round. Let $\Delta cost_{adv}$, $\Delta cost'_{\mathcal{A}}$ denote the change of $cost_{adv}$ and $cost'_{\mathcal{A}}$ in this round. Let also $\Delta_{adv}\Phi$ and $\Delta_{\mathcal{A}}\Phi$ denote the changes of the potential, respectively, during the adversary's and our move.

Lemma 2 $\Delta_{adv}\Phi \leq 2 \cdot \Delta cost_{adv}$.

Proof: When the adversary moves, only one edge in $M_{\min}$ may change. If a_1 moves by d, then the distance from a_1 to its mate in $M_{\min}$ changes by at most d, so the change of the potential is at most $2d$. Therefore $\|M_{\min}\|$ cannot increase by more than $2d$. Note that the other matching may become minimum during the move, but if so, its weight cannot be greater than the one of $M_{\min}$. This implies the lemma. $\Box$

Lemma 3 $\Delta_{\mathcal{A}}\Phi + cost'_{\mathcal{A}} \leq 0$.

Proof: Consider the first phase, when both of our servers move by d. Suppose that the request is on a_1. By Lemma 1(c) one edge in the minimum matching gets shorter by d. But, by Lemma 1(b) and the triangle inequality, the length of the other cannot increase by more than d. Thus $\|M_{\min}\|$ cannot increase. But $\|s_1 s_2\|$ drops by $2d$ by applying twice Lemma 1(c), and our cost is exactly $2d$.

In the second phase, assume that s_1 moves, that is $\|a_1 s_1\| + \|s_1 s_2\| = \|a_1 s_2\|$. Then $\|a_1 s_1\| + \|a_2 s_2\| \leq \|a_1 s_1\| + \|a_2 s_1\| + \|s_1 s_2\| = \|a_1 s_2\| + \|a_2 s_1\|$. This means that a_1 must be matched to s_1 in some minimum matching $M_{\min}$. Now, if s_1 moves by e then $\|a_1 s_1\|$ drops by e and $\|s_1 s_2\|$ increases by e. Therefore the potential must decrease by e, equal to our cost. $\Box$

From Lemmas 2 and 3, by a simple summation, and by application of Fact 1 we obtain the following.

Theorem 1 *$cost_{\mathcal{A}} \leq 2 \cdot cost_{opt} + \Phi_0$, where Φ_0 is the initial potential. Therefore Algorithm $\mathcal{A}$ is 2-competitive.*

Implementation. If M is finite and small, then the implementation is easy: after each round the server that is outside M remembers its position in $cl(M)$. If $|M| = m$, this method uses $O(m)$ space and $O(m)$ time per request.

In case when M is large, or even infinite, a more practical approach is to work only on the portion of M consisting of all the previous request points. If we are given n requests, this leads to a method that takes $O(n)$ space and $O(n)$ time per request.

We describe now the details of this implementation. Let x_0 be the initial point where the servers are located, and let x_t be the request point during the t^{th} round, $t = 1, \ldots, n$. Denote by M_t the subspace of M induced by $x_0, \ldots, x_t$. Without loss of generality we can assume that $M = M_n$.

We modify Algorithm 1 in such a way that at round t it works in $cl(M_t)$ instead of $cl(M)$, and when the new request x_{t+1} appears, $cl(M_t)$ is embedded isometrically into $cl(M_{t+1})$. Therefore the whole computation can be in fact "embedded" into $cl(M)$. This will assure that the cost we charge to our servers is at least the real cost of the algorithm.

The embedding $\tau : cl(M_t) \to cl(M_{t+1})$ is defined as follows:

$$\begin{aligned} (\tau u)(x_i) &= u(x_i), \quad i = 0, \ldots, t, \\ (\tau u)(x_{t+1}) &= \max_{i=0,\ldots,t} |u(x_i) - \|x_i x_{t+1}\|\,| . \end{aligned}$$

By a slight abuse of notation, we also refer to τu as u.

We need to show that τu is actually a member of $cl(M_{t+1})$, *i.e.*, that $\triangle(||xy||, u(x), u(y))$ holds for all $x, y \in M_{t+1}$. The only case we need to check is that $y = x_{t+1}$ itself and $x \in M_t$, that is, $|u(x) - u(x_{t+1})| \leq \|xx_{t+1}\| \leq u(x) + u(x_{t+1})$ for any $x \in M_t$. Now $u(x_{t+1}) \geq |u(x) - \|xx_{t+1}\|\,|$, which implies $\|xx_{t+1}\| \leq u(x) + u(x_{t+1})$ and $u(x) - u(x_{t+1}) \leq \|xx_{t+1}\|$. It remains only to show that $u(x_{t+1}) - u(x) \leq \|xx_{t+1}\|$. Choose x_g such that $u(x_{t+1}) = |u(x_g) - \|x_g x_{t+1}\|\,|$. If $u(x_{t+1}) = u(x_g) - \|x_g x_{t+1}\|$, then $u(x_{t+1}) = u(x_g) - \|x_g x_{t+1}\| \leq u(x) + \|xx_g\| - \|x_g x_{t+1}\| \leq u(x) + \|xx_{t+1}\|$, by two applications of the triangle inequality. On the other hand if $u(x_{t+1}) = \|x_g x_{t+1}\| - u(x_g)$, then $u(x_{t+1}) = \|x_g x_{t+1}\| - u(x_g) \leq \|xx_{t+1}\| + \|xx_g\| - u(x_g) \leq \|xx_{t+1}\| + u(x)$, again by two applications of the triangle inequality.

In order to prove that τ is indeed an isometry, it is sufficient to show that for any $u, v \in cl(M_t)$ we have $|u(x_{t+1}) - v(x_{t+1})| \leq \|uv\|$. Choose x_g such that $u(x_{t+1}) = |u(x_g) - \|x_g x_{t+1}\|\,|$. Then $|u(x_{t+1}) - v(x_{t+1})| \leq |u(x_g) - \|x_g x_{t+1}\|\,| - |v(y) - \|x_g x_{t+1}\|\,| \leq \|u(x_g) - v(x_g)| \leq \|uv\|$ (because $|a - c| - |b - c| \leq |a - b|$ for $a, b, c \in R$).

Acknowledgements: We would like to thank Allan Borodin, who noticed that our potential function can be expressed by the formula that was also used in [4]. This observation led to a simplification of the proof of Theorem 1. We would also like to thank anonymous referees for insightful suggestions that helped us to improve the presentation of the paper.

References

[1] P. Berman, H. Karloff, and G. Tardos. A competitive algorithm for three servers. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 280–290, 1990.

[2] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991. Also in Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1990, pp. 291-300.

[3] M. Chrobak and L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[4] D. Coppersmith, P. G. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to online algorithms. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 369–378, 1990.

[5] S. Irani and R. Rubinfeld. A competitive 2-server algorithm. manuscript.

[6] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990. Also in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 322-333.

[7] P. Raghavan and M. Snir. Memory versus randomization in online algorithms. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 687–703, Springer-Verlag, 1989.