Geometric Two-Server Algorithms

Ran El–Yaniv* Jon Kleinberg†

Abstract

For two servers in d-dimensional space under the L_1 metric, we give an optimal 2-competitive algorithm which uses constant time and space per request. This considerably extends the class of metric spaces for which an optimal, fast two-server algorithm is known; previously, the only such spaces were those that could be embedded in a tree. The algorithm itself can be seen as a higher-dimensional generalization of the "double-coverage" strategy of Chrobak et al.

Keywords: analysis of algorithms, on-line algorithms.

1. Introduction

In recent years, the k-server problem has been the focus of a great deal of attention. The problem itself has a simple statement: we are given a metric space M and k mobile servers that can move around in this space. At each time step, a point $x \in M$ is *requested*, and we must immediately move one of the servers to this point. The goal is to minimize the total distance traveled by all the servers over the course of a finite request sequence σ.

An *on-line* server algorithm S must make its decisions without knowledge of the location of future requests. As such, the *cost* $S(\sigma)$ incurred by the algorithm S on a sequence of requests σ will typically be larger than $OPT(\sigma)$, which is the cost incurred by an *optimal off-line* server algorithm to serve the same sequence (assuming that the off-line algorithm knows σ in its entirety ahead of time). The algorithm S is said to be c-competitive if for some constant β depending only on the initial configuration, the quantity $S(\sigma) - c \cdot OPT(\sigma)$ is bounded above by β over all finite sequences σ. The infimum of the set of c for which the algorithm is c-competitive is its *competitive ratio.* The k-server problem revolves around the question of finding *competitive,* on-line k-server algorithms for general and special metric spaces.

In this paper, we consider the case in which the metric space M is d-dimensional Euclidean space. We obtain a simple 2-competitive algorithm for two servers in $\Re^d$, for any dimension d, under the L_1 metric. Recall that in the L_1 ("Manhattan") metric, the distance between points $x = (x^1, \ldots, x^d)$ and $y = (y^1, \ldots, y^d)$ is given by

$$L_1(x, y) = \sum_{i=1}^{d} |x^i - y^i|,$$

*Department of Computer Science, Univ. of Toronto. This research was performed while visiting the Laboratory for Computer Science, MIT, Cambridge MA 02139 USA.

†Laboratory for Computer Science, MIT, Cambridge MA 02139 USA. Author is supported by an ONR Graduate Fellowship.

while the standard Euclidean distance, L_2, is defined by

$$L_2(x,y) = \left(\sum_{i=1}^{d}(x^i - y^i)^2\right)^{1/2}.$$

Our motivation in looking at this problem is two-fold. First, the trade-off between the computational resources of an on-line algorithm and the competitive ratio it can achieve is an issue that is still not well understood. In our case, it has been known since the introduction of the k-server problem that there exists a deterministic 2-competitive two-server algorithm, and that no deterministic two-server algorithm can achieve a ratio better than 2 in any metric space [MMS]. However, all known two-server algorithms attaining the optimum ratio of 2 must perform a considerable amount of computation on each request [MMS, CL1]; for example, in an infinite metric space such as $\Re^d$, these algorithms slow down with each additional request.

Thus it is natural to search for more efficient algorithms. In the setting of the server problem, it is standard to treat arithmetic operations and comparisons on real numbers as unit-cost operations; following Irani and Rubinfeld, and Chrobak and Larmore [IR, CL2], we refer to a server algorithm as *fast* if it maintains a constant number of real variables, and performs a constant number of operations on each request. Within this framework, a fundamental question that is still open is the following: does there exist a two-server algorithm that is both fast and 2-competitive?

Indeed, the best fast, deterministic algorithm known for two servers in a general metric space is only 4-competitive[CL2] (improving on a 10-competitive algorithm [IR]); and, until now, the only known metric spaces in which a fast algorithm could achieve the optimal ratio of 2 were those that could be embedded in a tree [CKPV, CL3].[1] Thus, our result considerably extends the class of metric spaces for which there is known to be a fast, optimal algorithm. Moreover, since distances in L_1 and L_2 differ by a factor of at most $\sqrt{d}$ in $\Re^d$, our algorithm gives the best competitive ratio for a fast two-server algorithm in two and three dimensions under the Euclidean metric ($2\sqrt{2} \sim 2.83$ and $2\sqrt{3} \sim 3.46$ respectively).

As an additional motivation, we note that the fast 2-competitive algorithm for the tree is based on a very elegant technique, "Double-Coverage," that was first discovered for servers on a line [CKPV]. However, attempts to extend this technique to produce optimal deterministic algorithms in spaces more general than trees have so far not been successful. Thus, an interesting feature of our main result is that it follows the style of Double-Coverage, and can therefore be viewed as an extension of that algorithm to the case of two servers in high dimensions.

2. A Geometric Lemma

In what follows, the term "distance" will mean L_1 distance unless explicitly stated otherwise. We will use x^i to denote the i^{th} coordinate of $x \in \Re^d$. For two points $x, y \in \Re^d$, define $\tau_d(x,y)$ to be the closed d-dimensional box (possibly degenerate) with x and y as opposite corners. That is, $w \in \Re^d$ lies in $\tau_d(x,y)$ if and only if w^i is between $\min(x^i, y^i)$ and $\max(x^i, y^i)$ for

[1] Note that all three-point spaces can be embedded in a tree, so the balancing algorithm of [MMS] is included in this category as well.

$i = 1, \ldots, d$. A point $z \in \Re^d$ induces the following partial order $\preceq_z$ on $\Re^d$: $x \preceq_z y$ if and only if x is contained in the box $\tau_d(y, z)$. It is easily verified that $\preceq_z$ satisfies the properties of a partial order.

Now consider the following problem. We have two sets P and Q, each consisting of n points in $(\Re^d, L_1)$, and we wish to find a minimum-cost perfect matching between them. The following lemma is central to the analysis of our algorithm.

Lemma 1 *For each $q \in Q$, there is a minimum-cost matching M in which q is matched to one of the points of P which is minimal with respect to $\preceq_q$.*

Proof. Consider a matching M' in which $p' \in P$ is matched to q, and p' is not minimal with respect to $\preceq_q$. Let p be a minimal element of $(P, \preceq_q)$ such that $p \preceq_q p'$, and suppose that p is matched to some $q' \in Q$. We can draw a shortest path from p' to q that passes through p; thus the matching M in which p is matched to q and p' is matched to q' has cost at most that of M'. ∎

3. The Algorithm in the Plane

We first give the algorithm in the plane, and then show how to extend it to higher dimensions. We number the quadrants of the plane in counterclockwise order, beginning with the positive quadrant. As is standard in analyzing server algorithms, we view the request sequence as a game between the algorithm and an omniscient adversary. In each round of the game, the adversary generates a request and serves it optimally (using knowledge of the future requests). The algorithm then must also serve the request, without knowing anything about the behavior of the adversary servers or the remainder of the request sequence.

Assume that the algorithm's servers are at points s_1 and s_2, the adversary servers are at points a_1 and a_2, and a request comes to some point in the plane. Since the L_1 metric is invariant under translations and 90-degree rotations, we can assume that the requested point r is the origin and that the server closer to the request, s_1, lies in the first quadrant.

First, the algorithm computes the partial order $\preceq_r$ on the two servers. Since s_1 is closer to r, it will be a minimal element. If s_2 is not also minimal, we simply move s_1 along any L_1 shortest path to the origin. Otherwise, both are minimal and we have the following cases:

1. s_2 lies in the third quadrant. Here we move both servers at the same speed along shortest paths to the origin until s_1 reaches it.

2. s_2 lies in the second quadrant. Here, we move both servers at the same speed to the positive y-axis — if s_2 reaches it first, we also move it down the y-axis — until s_2 is no longer minimal. (This happens when s_1 reaches the positive y-axis.) We then revert to the case in which only s_1 is minimal.

3. s_2 lies in the fourth quadrant. This is strictly analogous to the previous case.

4. s_2 also lies in the first quadrant. Then consider the rectangle $\tau_2(s_1, s_2)$, and let y be the corner of this rectangle closest to r. We move both servers at the same speed along L_1 shortest paths to y until s_1 reaches it; now s_2 is no longer minimal, and we proceed as above.

Theorem 1 *The above algorithm is 2-competitive in the L_1 plane.*

Proof. We analyze the algorithm's performance using the CDRS potential function [CDRS]

$$\Phi = L_1(s_1, s_2) + 2M_{\min}(S, A),$$

where $M_{\min}(S, A)$ denotes the value of the minimum-cost matching between the adversary servers and the on-line servers. By the standard potential-function argument, the following two facts will imply that our algorithm is 2-competitive (see e.g. [CKPV]):

1. When the adversary moves, paying c, it can raise Φ by at most $2c$.

2. When the algorithm moves, paying c', it lowers Φ by at least c'.

The first of these facts holds in any metric space, and has been proved in earlier papers [CKPV, CL3, CDRS]; the argument is simply that the value of the minimum-cost matching goes up by at most c' while the $L_1(s_1, s_2)$ term of Φ is untouched.

Now consider the second fact. We break the behavior of the algorithm into two *phases*. In the first (possibly empty) phase, both servers are moving; in the second phase, only one server is moving. In the second phase, in which one server moves a distance d, the value of $L_1(s_1, s_2)$ increases by at most d. However, there is a minimum-cost matching in which the moving server is matched to the request point; this is seen by applying Lemma 1 with P equal to the set of on-line servers, Q equal to the set of adversary servers, and q equal to the origin r, which is currently occupied by one of the adversary servers. Hence the value of the minimum-cost matching goes down by d, so Φ goes down by at least d, as desired.

When the algorithm is moving both servers (each a distance d) the matching component cannot go up (whichever server is matched to the origin is decreasing its contribution to the cost optimally). Meanwhile, in each of the four cases above, the servers move closer to one another by an amount equal to the sum of the distances they travel; thus, the distance component $L_1(s_1, s_2)$ goes down by $2d$. Hence Φ decreases by at least $2d$, and the proof is complete. ∎

Under the Euclidean (L_2) metric, the on-line servers can follow the above algorithm and pay at most the L_1 cost. Meanwhile, the adversary only has to pay the L_2 cost, which can be smaller by at most a factor of $\sqrt{2}$. Thus we have

Corollary 1 *The algorithm is $2\sqrt{2}$-competitive (~ 2.83) in the Euclidean plane.*

4. The Algorithm in $\Re^d$

Consider now the case of two servers in $\Re^d$, under the L_1 metric. As before, by a translation of $\Re^d$, we may assume that the request r is at the origin; we also relabel the servers if necessary so that s_1 is closer to the request.

The algorithm in d dimensions works as follows. If only one server is minimal, it moves on a shortest path to the origin while the other remains fixed. Otherwise, both servers are minimal. Let $s_1 = (s_1^1, \ldots, s_1^d)$ and $s_2 = (s_2^1, \ldots, s_2^d)$ in coordinates; we proceed in the following two stages:

1. As long there is a j for which s_1^j and s_2^j have opposite sign (i.e. $s_1^j s_2^j < 0$), we choose such a j and move the servers at the same speed towards the hyperplane $\{x : x^j = 0\}$ so as to bring one of these values to 0.

2. The servers begin the second stage with all j satisfying $s_1^j s_2^j \geq 0$; that is, they lie in the same closed orthant. By relabeling the axes, we may assume this is the positive orthant. Let y be the corner of $\tau_d(s_1, s_2)$ closest to the origin; that is,

$$y = (\min(s_1^1, s_2^1), \ldots, \min(s_1^d, s_2^d)).$$

As before, denote the j^{th} coordinate of y by y^j. The servers repeatedly execute the following step: each picks a j for which $s_i^j - y^j$ is strictly positive, and moves at a fixed speed towards the hyperplane $\{x : x^j = y^j\}$ so as to decrease s_i^j to y^j, stopping when it reaches this hyperplane. Each such move reduces the number of coordinates j for which $s_i^j - y^j > 0$ for one of the servers s_i. Thus, after at most d such steps, server s_1 will be sitting on y. At this point, only s_1 is minimal, and we proceed as above.

Theorem 2 *The algorithm is 2-competitive under the L_1 metric in $\Re^d$.*

Proof. We use the same potential function Φ. The analysis of the adversary's move and the case in which only one on-line server is minimal are handled as in the previous section. When both on-line servers move a distance d, we must show that the potential Φ drops by at least $2d$. This will be implied by the following property:

($*$) Both servers are moved a distance d closer to the origin and $2d$ closer to each other simultaneously.

The motion of the servers in the first stage of the algorithm clearly satisfies this property. Observe that throughout the second stage, there is a shortest path from s_1 to s_2 that passes through the corner y of $\tau_d(s_1, s_2)$. But each step in this stage brings a server simultaneously closer to the origin and to y; thus Property ($*$) is maintained in the second stage as well. ∎

Corollary 2 *The algorithm is $2\sqrt{d}$-competitive under the Euclidean metric in $\Re^d$.*

5. Conclusion and Open Problems

The existence of a fast 2-competitive algorithm for two servers in $\Re^d$ appears to raise the possibility that the following question could have an affirmative answer: does there exist a fast, 2-competitive two-server algorithm in any metric space? We find this issue interesting since it addresses the question of how the competitive ratio of an algorithm varies with its computational resources, in the setting of a canonical on-line problem.

It would also be interesting to determine whether the natural generalization of this algorithm achieves a constant competitive ratio for more than two servers. There is of course some freedom in choosing exactly how to define this generalization; however, we note in closing that fairly simple examples show that most of the obvious extensions to three servers are not 3-competitive.

References

[CKPV] M. Chrobak, H. Karloff, T. Payne, S. Vishwanathan, "New results on server problems," *SIAM J. Discrete Math.*, 4(1991), pp. 172–181.

[CL1] M. Chrobak, L. Larmore, "A new approach to the server problem," *SIAM J. Discrete Math.*, 4(1991), pp. 323–328.

[CL2] M. Chrobak, L. Larmore, "On fast algorithms for two servers," *J. Algorithms*, 12(1991), pp. 607–614.

[CL3] M. Chrobak, L. Larmore, "An optimal on-line algorithm for k-servers on trees," *SIAM J. Computing*, 20(1991), pp. 144–148.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, M. Snir, "Random walks on weighted graphs, with applications to on-line algorithms," *Journal of the ACM*, 40(1993), pp. 421–453.

[IR] S. Irani, R. Rubinfeld, "A competitive 2-server algorithm," *Information Processing Letters*, 39(1991), pp. 85–91.

[MMS] M. Manasse, L. McGeoch, D. Sleator, "Competitive algorithms for server problems," *J. Algorithms*, 11(1990), pp. 208–230.