An Optimal On-line Algorithm for k Servers on Trees

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

October 4, 1996

Abstract

We investigate the k-server problem when the metric space is a tree. For this case we present an on-line k-competitive algorithm for k servers. The competitiveness ratio k is optimal. The algorithm is memoryless, in the sense that it does not use any information from the past.

1 Introduction

Let M be a metric space. That is, for any two points $x, y \in M$ we are given their distance $\|x,y\| \geq 0$ such that $\|x,y\| > 0$ for $x \neq y$, and the triangle inequality is also satisfied: $\|x,y\| + \|y,z\| \geq \|x,z\|$ for all $x, y, z \in M$.

We are also given k servers that can move among points of M. At each time slot, a request $x \in M$ appears, and we have to "serve" this request, that is, choose one of our servers and move it to x. Other servers are also allowed to move. Our measure of cost is the distance by which we move our servers. The problem is to design a strategy that minimizes the cost of servicing a sequence of requests given on-line. The server problem is an abstraction of several practical problems, including heuristics for linear search, paging, and motion planning for 2-headed disks. For example, in the paging (or, equivalently, caching) problem we have a two-level memory system with a total of n pages, k of which can reside in fast memory. Given a reference to a memory page x which is not currently in the fast memory, we remove some page y from the fast memory and replace it with x. Our objective is to minimize the number of page replacements. This can be modeled by a server problem on an n-point space where all distances are equal to 1. See [2, 7, 9] for more references.

Suppose that we are given a sequence R of requests. A *schedule* is a specification of which server serves which request of R. Any schedule that achieves a minimum cost is called an *optimal* schedule. It is known (see [2]) how to compute an optimal schedule for R in polynomial time, assuming that the whole sequence R is given off-line, in advance.

However, no *on-line* algorithm can achieve the optimal cost on each sequence of requests. The current research on this problem concentrates on finding *c-competitive* on-line algorithms, that is, algorithms which compute a schedule of cost at most $c \cdot opt + b$, where opt is the cost of the optimal schedule and b is a constant that is allowed to depend only on the initial configuration. Manasse, McGeoch, and Sleator [7] proved that there is no on-line c-competitive algorithm for $c < k$, for any metric space with at least $k+1$ points. The problem of whether there is a general on-line algorithm with competitiveness ratio k remains open.

For $k = 2$, the problem was solved by Manasse, McGeoch, and Sleator [7] who gave a 2-competitive algorithm for 2 servers. Irani and Rubinfeld [5] proved that a version of a balancing algorithm (dividing the work more or less equally among the servers) is 10-competitive for 2 servers. Raghavan and Snir [9] presented a randomized 2-competitive algorithm whose competitiveness ratio is between 3 and 6.

Some research has been done for some specific metric spaces. Chrobak et al. [2] presented k-competitive algorithms for the line and the weighted cache problem, where the distance to x from any other point is $w(x)$, the weight of x. The weighted cache problem was also considered in [9], where a randomized k-competitive algorithm for this problem was given. Using randomization for the paging problem, it is possible to beat the lower bound k for deterministic algorithms. Fiat et al. [4] presented a randomized algorithm for this problem with ratio $2H_k$, improved later to H_k by McGeoch and Sleator in [8] (where H_k is the kth harmonic number). Very recently, Coppersmith et al. [3] have discovered a randomized algorithm with competitiveness constant k that works for a class of metric spaces called *resistive*. This class includes many known metric spaces, for example, trees.

Raghavan and Snir [9] introduced a notion of a *memoryless* algorithm, an algorithm whose behaviour does not depend on the past. A memoryless algorithm makes the decision based only on the current configuration of the servers and the position of the request.

In this paper, we present a k-competitive algorithm for k servers on trees. By a *tree* we mean a metric space that is topologically equivalent to a free tree (in the graph theoretic sense), and the distance is measured along the branches of the tree. The lower bound of k for k servers presented in [7] also applies to the special case of a tree, and therefore this result is best possible for deterministic algorithms. The algorithm is also memoryless. This generalizes a similar result for the line in [2]. It should be pointed out here that, in the case of trees, some server algorithms with memory can be transformed into memoryless algorithms with the same asymptotic competitiveness ratio, by a careful encoding of the memory state in the binary expansion of the distances between the servers. This requires, however, complicated and expensive bit operations, unlike our algorithm, that is realized by a simple, piecewise linear function.

An interesting feature of this algorithm is that it can be applied to other problems, even though they may not have a tree structure at first glance. For example, consider the paging problem, where our metric space consists of n points where all distances are equal to 1. We "embed" this into a tree as follows: our tree is a star with n arms of length 0.5, and we place n points at the ends of those arms. By applying our algorithm to this star, we obtain a k-competitive algorithm for the paging problem. Quite surprisingly, this algorithm turns out to be equivalent to the Flush-When-Full cache strategy from [6]. In the same way, it also gives a k-competitive algorithm for a much more difficult problem: the *symmetric weighted cache*, where each point x is given some weight $w(x) \geq 0$, and the distance between x and y is $w(x) + w(y)$. It is easy to see that the symmetric and nonsymmetric version of weighted cache are almost equivalent: on any sequence of requests their optimal costs differ by at most half of the difference of weights of the initial and final configuration.

We also show that our result gives an algorithm for an arbitrary n-point metric space with competitiveness ratio $k(n-1)$. This improves the previously known bound of $2\binom{n}{k} - 1$ that can be derived from the work of Borodin, Linial, and Saks [1]. For some specific metric spaces this ratio can be made yet smaller.

2 The adversary and potential methods

The adversary method is used very often for proving lower bounds on the complexity of various computational problems. We employ some of the ideas of this method for establishing the *upper* bound on the competitiveness ratio. In the proofs, we view the computation as a game between our servers $s_1, \cdots, s_k$ and adversary's servers $a_1, \cdots, a_k$, and measure the ratio between S, the work done by our servers, and A, the work of the adversary's servers. The goal is to show that independently of the way the adversary moves, we have $S \leq cA + b$. This implies immediately that the algorithm is c-competitive, since one of the adversary's computations will correspond to the optimal schedule. The same approach has been used in other proofs of competitiveness.

The potential method has been used before in the analysis of the amortized complexity of various data structures. The technique can be viewed as follows. Given an on-line server algorithm, we define a potential function Φ that maps any possible configuration of the servers (ours and the adversary's) to a nonegative real number. Let Φ^t denote the value of the potential function after t steps of the algorithm.

The adversary starts from the same initial configuration as our servers. We assume that Φ^0, the initial value of the potential, depends only on this initial configuration. Additionally, we assume that at each step of the game, the adversary first moves a single server to the request point, and then we apply our algorithm to serve the request. It is not hard to prove that those restrictions do not lead to loss of generality (see [7]). Let $\widetilde{\Phi}^t$ denote the value of the potential function after $t-1$ steps and one additional adversary's move. A^t and S^t denote, respectively, the adversary's and our costs at step t. Our technique relies on the following lemma, whose simple proof is left to the reader.

Lemma 2.1 *Suppose that the following two conditions hold:*

(i) $\widetilde{\Phi}^t - \Phi^{t-1} \le \alpha A^t$, *and*

(ii) $\widetilde{\Phi}^t - \Phi^t \ge \beta S^t$.

Then $S \le (\alpha/\beta)A + \Phi^0$, *that is, our algorithm is c-competitive for* $c = \alpha/\beta$.

3 The competitive algorithm for trees

In this paper, by a *tree* we understand a planar embedding of a free tree (in the graph-theoretic sense). If T is a tree, then the distance $\|x, y\|$ is the arc-length of the unique simple path through T from x to y. This path, with x excluded, is denoted by $(x, y]$, and called an *interval*.

For simplicity, s_p and a_i will also denote the current positions of the servers s_p, a_i. If the request is on point x, then we call our server s_p *active*, if there are no other our servers in the interval $(s_p, x]$. If several servers occupy the same position as s_p, and all satisfy the condition above, then only *one* of them is chosen arbitrarily as the active one, the others are not. Our algorithm works as follows.

Algorithm 1: Move all our active servers continuously with the same speed towards x until one of them (obviously the closest one) reaches the request. Note that during this motion some active servers may become nonactive, and then they halt.

More precisely, the algorithm can be formulated as follows.

> **while** none of our servers is on x **do begin**
> let $d = \min_p \|s_p, y_p\|$, where $y_p \in (s_p, x]$ is either a vertex or x;
> move each active server s_p by d towards x
> **end**.

Our potential function is defined now by

$$\Phi = k\|M_{\min}\| + \sum_{p<q} \|s_p s_q\|,$$

where $M_{\min}$ is a minimum weight matching in a bipartite graph with components $\{s_1, \cdots, s_k\}$ and $\{a_1, \cdots, a_k\}$, where the weight of edge (s_p, a_i) is $\|s_p a_i\|$.

Lemma 3.1 $\widetilde{\Phi}^t - \Phi^{t-1} \leq kA^t$.

Proof. Since the adversary moves only one server, only one edge in $M_{\min}$ can change. If this server moves by d, then the potential increases by at most kd. This implies the lemma. $\square$

Lemma 3.2 $\widetilde{\Phi}^t - \Phi^t \geq S^t$.

Proof. Without loss of generality we assume that the request is on a_1. Assume that the servers $s_1, \cdots, s_q$ are active. It is easy to see that there is a minimum matching $M_{\min}$ in which a_1 is matched to one of $s_1, \cdots, s_q$. Suppose that all our servers moved by d. Then the weight of a minimum matching cannot increase by more than $(q-2)d$. For $p = 1, \cdots, q$ let ℓ_p be the number of our servers s_r such that s_p is between s_r and a_1. When s_p moves by d, its distance to $l_p - 1$ of our servers increases by d, but the distances to $k - \ell_p$ of our servers decreases by d. Therefore during this movement the change of the potential is at most

$$\begin{aligned} k(q-2)d + \sum_{p=1}^{q}(\ell_p - 1 - k + \ell_p)d &= \left[k(q-2) + 2\sum_{p=1}^{q}\ell_p - q - kq\right] d \\ &= \left[k(q-2) + 2k - q - kq\right] d \\ &= -qd, \end{aligned}$$

and our cost is qd. This proves the lemma. $\square$

From Lemmas 3.1 and 3.2, using Lemma 2.1 we obtain immediately the following result.

Theorem 3.1 *Algorithm* 1 *is k-competitive.*

4 Applications

As it was shown in the Introduction, our algorithm can be applied directly to other metric spaces that can be "embedded" in a tree. Another example of such spaces are so-called *ultrametric spaces.* A metric space M is called *ultrametric* if for any $x, y, z \in M$ we have $\|xy\| \leq \max\{\|xz\|, \|yz\|\}$. It is easy to show that each ultrametric space can be isometrically embedded into a tree, so our algorithm can be applied to ultrametric spaces as well.

In this section we show that this algorithm can also be used to obtain an algorithm that is $k(n-1)$-competitive on every n-point metric space.

Let M be a metric space with n points.

Algorithm 2. Fix a minimum spanning tree T of M, and apply Algorithm 1 pretending that T is the underlining metric space.

By using the subscript T we will distinguish the distances and costs in T from those in M.

Lemma 4.1 (i) $S \leq S_T$, *and*

(ii) $A_T \leq (n-1)A$.

Proof. Inequality (i) follows directly from the triangle inequality. To prove (ii) consider a single move of the adversary when he moves a server from x to y. Consider the path P from x to y in T. If (u, v) is any edge on P, then $\|u, v\| \leq \|x, y\|$ because otherwise T would not be minimum. Since P has at most $n-1$ edges, $\|x, y\|_T \leq (n-1)\|x, y\|$, and (ii) follows. □

Since by Theorem 3.1 $S_T \leq kA_T$, we obtain the following.

Theorem 4.1 *Algorithm* 2 *is* $k(n-1)$ *competitive in any* n*-point metric space.*

For some specific metric spaces this bound can be yet essentially improved. Denote by δ the *diameter* of T, that is, the maximum number of edges in a simple path of T. Then, by the same argument as above, we have that Algorithm 2 is in fact $k\delta$-competitive. For example, if M is an $\sqrt{n} \times \sqrt{n}$ grid (with all weights equal to one) then it has a spanning tree of diameter $2\sqrt{n}$, so the competitiveness ratio is $2k\sqrt{n}$. If M is a hypercube (with all weights equal to one), then it has a spanning tree of diameter $\log n$, and then the competitiveness ratio is only $k \log n$.

5 Final remarks

Note that our algorithm works even when we relax our definition of a tree, allowing infinite trees. In fact, the tree may be dynamic, in the sense that we start with a single point, and each request consists of a request point and a path leading to this point from some already existing branch of the tree.

Example. A robot is confined to the space in a room above a certain track on the floor. The robot is always free to move vertically, but can only move horizontally along its track when it is resting on the floor. Let T be the set of positions of the robot, and $\|x, y\|$ is defined to be the total movement needed for the robot to get from position x to position y. Then T is a tree in this more general sense if the track is one-dimensional, in fact, if the track is any tree.

There is another property of our algorithm we would like to emphasize. In our approach we can view the servers as identical robots that move along the tree. By "identical" we mean that they have the same speed and execute the same program. Their program is simple: given the request site x move towards x unless you see another robot on your path to x (in which case, obviously, the other robot will reach x earlier so following it would be rather silly). We find it interesting that this somewhat chaotic behaviour, when robots actually compete between themselves to serve the request, leads to an effective algorithm.

Acknowledgments. Allan Borodin pointed out to us that our potential function can be espressed by the formula we currently use in the paper. The same formula was used in [3].

References

[1] A. Borodin, N. Linial, and M. Saks, An optimal online algorithm for metrical task systems, in Proc. 19th Annual ACM Symposium on Theory of Computing, 1987, pp. 373-382.

[2] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan, New results on server problems, submitted for publication.

[3] D. Coppersmith, P.G. Doyle, P. Raghavan, and M. Snir, Random walks on weighted graphs and applications to online algorithms, manuscript.

[4] A. Fiat, R. Karp, M. Luby, L.A. McGeoch, D. Sleator, and N.E. Young, Competitive paging algorithms, Tech. Report CMU-CS-88-196, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1988.

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

[6] A. Karlin, M. Manasse, L. Rudolph and D. Sleator, Competitive snoopy caching, Algorithmica, 3 (1988), pp. 79-119.

[7] M. Manasse, L.A. McGeoch, and D. Sleator, Competitive algorithms for server problems, Tech. Report CMU-CS-88-197, Carnegie-Mellon University; in Proc. 20th Annual ACM Symposium on Theory of Computing, 1988, pp. 322-333.

[8] L. McGeoch and D. Sleator, A strongly competitive randomized paging algorithm, manuscript.

[9] 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, Springer-Verlag, vol. 372, 1989, pp. 687-703.