CS294-1 On-line Computation & Network Algorithms **Spring 1997**

Lecture 5: February 4

Lecturer: Yair Bartal *Scribe: Anupam Gupta*

In this lecture, we define the k-server problem and prove a lower bound of k for the the competitive ratio for any metric space. We then describe a k-competitive algorithm for the special case when the metric spaces under consideration are either line or trees.

5.1 The k-server problem

Definition 5.1 *A metric space $M = (V, d)$ consists of a set of points V with a distance function $d : V \mapsto \Re$ satisfying the following properties:*

- $d(u, v) \geq 0$, *for all* $u, v \in V$.
- $d(u, v) = 0$ *iff* u=v.
- $d(u, v) + d(v, w) \geq d(u, w)$ *for all* $u, v, w \in V$.

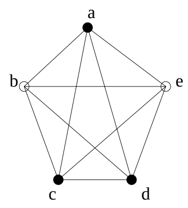
NOTE: Sometimes it is convenient to think of a finite metric space over n points as the complete weighted graph over n vertices with weights corresponding to distance between the corresponding points. Similarly, given a weighted (not necessarily complete) graph, we can associate a metric space with it by letting the distance between any pair of points to be the (weighted) length of the shortest path between them in the graph.

The *k-server problem* is due to Manasse, Mcgeoch and Sleator [MMS88].

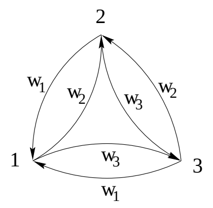
Problem 5.2 *Consider a metric space $\mathcal{M}$ with a distance function d (which is symmetric in most of the cases we consider). There are k servers in this space. The request sequence is $\sigma = \sigma_1\sigma_2\ldots$, where σ_i is a point in $\mathcal{M}$. The online algorithm's response to σ_i should be to move a server j to point σ_i to serve the request.*

The cost measure is the total distance moved by all the servers.

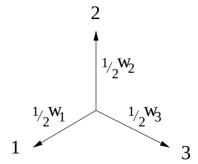
EXAMPLE: Let $\mathcal{M}$ be the *uniform* metric space over n points. That is the metric space where the distance between every pair of points is 1. It is easy to see that the k-server problem on the uniform metric space is isomorphic to the paging problem with n pages in slow memory and a cache of size k. Each point in the metric space corresponds to a particular page and the set of points covered by a server correspond to the k pages in the cache.



EXAMPLE: If we consider the weighted paging problem, where the cost of bringing page p_i into the cache is w_i, the we get the following directed graph.



This, unfortunately, gives us an asymmetric distance function. To set this right, we could instead use a star on $n+1$ vertices, with the weights as shown.



This would give us a cost which would differ from the one in the asymmetric case by a fixed constant, and as the length of the request sequence increases, its cost would asymptotically approach the previous cost.

EXAMPLE: We can model a k-head disk scheduling by considering $\mathcal{M}$ to be the real line along which the k servers (which are the heads) can move.

5.1.1 Lower bounds

Theorem 5.3 *For any metric space, the competitive ratio of the k-server problem is at least k. Moreover, this lower bound holds for any randomized algorithm against an adaptive online adversary.*

Proof: We can assume that $\mathcal{M}$ has at least $n = k+1$ points (else the problem is trivial). We fix our attention to any n of the points in $\mathcal{M}$ and call them $S = \{1, 2, \ldots, k+1\}$. Let the k servers be at the points $\{1, 2, \ldots, k\}$ and other point is "free".

Now we need to show an adversary ADV and a request sequence σ such that for any (potentially randomized) online adversary A,

$$A(\sigma) \geq k\ ADV(\sigma). \tag{5.1}$$

Instead, we will give a set of k adversaries $ADV_1, \ldots, ADV_k$ such that $A(\sigma) \geq \sum_{i=1}^{k} ADV_i(\sigma)$. This implies that at least one of the ADV_i's would satisfy the condition (5.1), which in turn would imply the theorem.

The request sequence σ (for both the deterministic and randomized cases) simply causes the next request at the point in S not covered by A's servers. Thus A has to move one of its servers at every request.

We make one further assumption: the adversary's servers can have any initial position. This is purely for notational convenience, as this would give us at most an additional constant additive cost (to move the adversary' servers at the beginning). This can be swallowed up in the definition of the competitive ratio. Thus the adversary ADV_i starts with its servers on the points $S - \{i\}$. This placement of the servers gives us two properties, which we shall try to maintain invariant over time.

1. All the adversaries ADV_j have a server at the point of the next request (i.e. at the point not covered by A).
2. For every point covered by the servers of A, there is a unique ADV_j which does not have a server at that point.

At the first request σ_1, let A move server from j to $k+1$ to serve it. In reply, ADV_j moves its server from $k+1$ to j. The cost incurred by both is the same (as we consider the metric space to be symmetric), and the properties listed above are maintained. The new state of the algorithm and the adversaries is isomorphic to the initial state and hence the process can be repeated. So, for any σ, we have $A(\sigma) = \sum_{i=1}^{k} ADV_i(\sigma)$. ■

5.1.2 The k-server Algorithm for the Line

We will now study a k-server algorithm for the line called the Double Coverage algorithm. This was given by Chrobak, Karloff, Payne and Vishwanathan [CKPV90]. This is an optimal on-line algorithm, i.e. it is k-competitive. We refer to the algorithm as DC in the subsequent discussion.

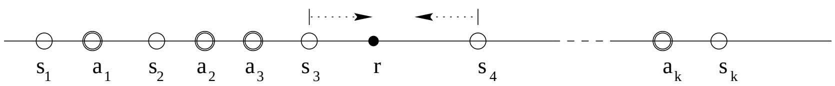
Let $s_1, s_2, \ldots, s_k$ denote the servers on the line from left to right. Note that we can always maintain this order, because if s_i has to be moved to the right of s_{i+1}, then we can move it to the current location of s_{i+1}, and then move s_{i+1} instead to its right. This change would incur the same cost.

The strategy is simple. If the next request r is on one side of all the servers, the server nearest to r is moved to serve the request. It can be verified that this is the optimal move, that is any algorithm that moves another server can be replace by an algorithm that moves the nearest server without increasing its cost.

Note: Unfortunately, this greedy approach does not generalize. That is, the greedy algorithm that *always* moves the nearest server is not competitive. Consider the case in the figure above. Let σ

be $BABABABA\ldots$. The greedy approach would keep oscillating the server s_1 between A and B, giving us an unbounded cost. However, just moving s_2 to B and keeping σ_1 at A would give us a constant cost. Hence the greedy algorithm is not competitive.

In the other case, the request r is between the servers s_i and s_{i+1}. W.l.o.g. it is closer to s_i and $d(r, s_i) = \delta$. We now move the server s_i to r, but we also move the other server s_{i+1} a distance δ closer to r. In the figure below, $i = 3$.



NOTE: The way the algorithm is described it is *memoryless*, i.e., its decision is a function of its current configuration alone. In the actual implementation of the algorithm we would run a simulation of the algorithm, while moving server is a *lazy* manner that is we will move a server only when it moves to the location of the current request (and only one server). It follows from the triangle inequality that the cost the lazy version pays is upper bounded by the cost of the memoryless algorithm.

Theorem 5.4 *The* DOUBLE COVERAGE *algorithm is k-competitive.*

Proof: We assume that the actions take place in the following order: when the request is made, the adversary makes his move and then DC makes its move. We want a potential function Φ which has the following properties:

1. $\Phi \geq 0$
2. When the adversary incurs a cost x, the change in potential, $\Delta\Phi \leq k \cdot x$.
3. When the online algorithm incurs a cost x, the change in potential, $\Delta\Phi \leq -x$.

Lemma 5.5 *The existence of such a potential function Φ implies that the* DOUBLE COVERAGE *algorithm is k-competitive.*

Proof: If the potential function Φ satisfies the above properties, then after the moves of the adversary and DC to satisfy a request, $\Delta\Phi \leq k$ (cost of adversary) − (cost of DC). Let Φ_0 and Φ_f be the initial and final values of the potential function after request sequence σ. Thus $\Phi_f - \Phi_0 \leq k\ ADV(\sigma) - ON(\sigma)$. But $\Phi_f \geq 0$, and thus $ON(\sigma) \leq k\ ADV(\sigma) - \Phi_0$. ■

We define $\Phi = k\Psi + \Theta$. Here Ψ is the weight of the minimum weight matching in the bipartite graph where the servers of the adversary and the online algorithm form the two parts. For the line, it is trivial to see that it is simply $\sum_{i=1}^{k} d(s_i, a_i)$. Further, $\Theta = \sum_{i<j} d(s_i, s_j)$, which is simply the sum of the distances between every pair of on-line servers.

1. With this definition, Φ is clearly non-negative.

2. Assume the adversary moves a_i to position a_i'. This does not change Θ. As for the Ψ term, the change is simply $d(a_i', s_i) - d(a_i, s_i) \leq d(a_i', a_i)$, which is the cost of the move. Hence $\Delta\Phi \leq k\ d(a_i', a_i)$.

3. The adversary must always have a server at the position r. If the request is to one side of all the servers, then moving a server a distance x to r only decreases Ψ by x. However, it also increases Θ by $(k-1)x$, and hence $\Delta\Phi = -kx + (k-1)x = -x$.

   In the other case, we have two possibilities: either a_i is at r or to its right, or a_{i+1} is at r or to its left. Thus, when we move both servers towards r, at least one of the servers reduces its distance from its match, the other potentially may increase its distance from its match by at most the same amount. Hence $\Delta\Psi \leq 0$. For the other term, for any $j \notin \{i, i+1\}$, the sum of the distances from s_j to s_i and s_{i+1} remains the same, as the movements cancel out. So $\Delta\Theta$ is $-x$, and hence $\Delta\Phi \leq -x$. ■

5.1.3 The k-server Algorithm for Trees

The algorithm for the line can be enhanced to work for trees. For this, we say that a server *sees* a request at point r if there is no other server on the path between between it and the point r. If there are more than once server at the same point we break ties arbitrarily to determine one that sees the request.

When a request is made, all servers who see the request start moving towards r at identical speeds. Note that after some time, a server that was previously seeing the request may cease to see it. As soon as this happens, the server stops. Further, as soon as a server gets to r, all of them stop. It is easy to see that this is a generalization of the algorithm for the line. This algorithm is due to Chrobak and Larmore [CL91].

Theorem 5.6 *The above algorithm is k-competitive.*

The above theorem can be proved using the same potential function used for the case of the line. The details are left for a future exercise.

References

[CKPV90] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. Ne w Results on Server Problems. In *Proc. 1st ACM-SIAM Symp. on Discrete Algorithms*, pages 291-300, January 1990.

[CL91] M. Chrobak and L. Larmore. An Optimal On-Line k-Server Algorithm for Trees. *SIAM Journal of Computing*, 20:144-148, 1991.

[MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive Algorithms for On-Line Problems. In *Proc. of the 20th Ann. ACM Symp. on Theory of Computing*, pages 322-333, May 1988.