7 The Server Problem

The *k-server problem* in a metric space M is a forcing metrical task system where the set of states is $\Lambda^k M$ and the tasks are in a one-to-one correspondence with M itself. The cost of satisfying request $x \in M$ while in state $S \in \Lambda^k M$ is defined by

$$cost(x, S) = \begin{cases} 0 & \text{if } x \in S \\ \infty & \text{otherwise} \end{cases}$$

Exercise 7.1 *There is a definition of the k-server problem in our textbook. Prove that is equivalent to the definition given above.*

Exercise 7.2 *Prove that the k-paging problem is equivalent to the k-server problem in a uniform space.*

Theorem 7.3 *If a metric space M has at least $k+1$ points, and $C < k$, then there is no C-competitive online algorithm for the k-server problem in M.*

The proof of Theorem 7.3 is in the textbook.

Lazy Algorithms. An algorithm for the k-server problem is *lazy* if, given a request r, it does not move if there is a server at x, and if there is no server at r, it moves only one server.

Memoryless Algorithms. An algorithm for the k-server problem is *memoryless* if, given a request r, its decision is solely a function of r and the current configuration.

7.1 Move Closest

The strategy *move closest* is simply defined to be the strategy of moving the closest server to the request. This strategy is both lazy and memoryless.

Exercise 7.4 *Move closest is not C-competitive for any C.*

7.2 Lower Bound and the k-Server Conjecture

Lemma 7.5 *If M is a metric space with at least $k+1$ points, and if $C < k$, then there is no C-competitive online algorithm for the k-server problem in M.*

Conjecture 7.6 *Given any metric space M, there is a k-competitive online algorithm for the k-server problem in M.*

Proof: Assume that the initial configuration is $\{x_1, \ldots, x_k\}$. Pick one more point, x_0, in M. Let $\mathcal{A}$ be an online algorithm for the k-server problem in M.

Suppose that $\mathcal{A}$ is C-competitive for some $C < k$. Then there exists a constant K such that $cost_{\mathcal{A}} \leq C\, cost_{opt} + K$ for any request sequence. Let $L = \sum_{j=1}^{k} d(x_j, x_{_i})$, which does not depend on the request sequence.

Let $\delta > 0$ be the smallest distance between two points, *i.e.,*$\delta = \min d(x_i, x_j : i \neq j$. Let ϱ be the cruel request sequence of length n, where $n\delta(k - C) > k(K + L)$. Such an n must exist, since $k > C$.

We define k different offline algorithms which each serve the request sequence. $\mathcal{B}_1, \ldots, \mathcal{B}_k$, as follows.

1. Initially, $\mathcal{B}_i$ moves a server from x_i to x_0.

2. Henceforth, the following invariant will hold after every step: for every point x_i, either $\mathcal{A}$ has no server at x_i, or there is exactly one j for which $\mathcal{B}_j$ has no server at x_i.

3. If the request is x_i, then $\mathcal{A}$ does not have a server at x_i, since the request sequence is cruel. By condition 2, each $\mathcal{B}_j$ has a server at x_j, and serves the request at no cost.

4. In order to serve the request, $\mathcal{A}$ must move a server from some point x_l to x_i. For exactly one j, $\mathcal{B}_j$ does not have a server at x_l. Then, $\mathcal{B}_j$ moves its server from x_i to x_l.

Let $cost_{\mathcal{A}}$ be the total cost incurred by $\mathcal{A}$ to serve the entire request sequence, and let $cost_{\mathcal{B}_j}$ be the cost incurred by $\mathcal{B}_j$. We claim that

$$cost_{\mathcal{A}} = \sum_{j=1}^{k} cost_{\mathcal{B}_j} + \sum_{j=1}^{k} d(x_j, x_{-i})$$

The second term on the right hand side is kL, which is constant, while the first term grows without bound as the length of the request sequence increases.

Pick j such that $cost_{\mathcal{B}_j}$ is minimum. Then, $cost_{opt} \leq cost_{\mathcal{B}_j} \leq \frac{1}{k}(cost_{\mathcal{A}} + L)$. Combining all these inequalities, we have

$$\begin{aligned}
n\delta &< cost_{\mathcal{A}} && (1)\\
cost_{\mathcal{A}} &\leq C\,cost_{opt} + K && (2)\\
k\,cost_{\mathcal{A}} &\leq kC\,cost_{opt} + kK && (3)\\
k\,cost_{opt} &\leq cost_{\mathcal{A}} + L && (4)\\
kC\,cost_{opt} &\leq C\,cost_{\mathcal{A}} + CL \leq C\,cost_{\mathcal{A}} + kL && (5)\\
k\,cost_{\mathcal{A}} &\leq C\,cost_{\mathcal{A}} + kL + kK && (6)\\
(k-C)\,cost_{\mathcal{A}} &\leq k(L+K) && (7)\\
(k-C)n\delta &< (k-C)\,cost_{\mathcal{A}} && (8)\\
(k-C)n\delta &< k(L+K) && (9)
\end{aligned}$$

which contradicts our choice of n. □

The k-server conjecture is still open, but has been proved in a number of special cases.

7.3 Double Cover

In the early days of the k-server problem, most results were rather *ad hoc*, meaning that they didn't appear to depend on any deep analysis. Results were mostly for special cases, such as for metric spaces of special interest. One important special case is the line, $\mathbb{R}$.

The *double cover* algorithm (DC) for the k-server problem on the line is memoryless, but not lazy. Instead of moving just one server to the request point, it moves as many as two servers. The server closest to the request point r serves the request, but if there is a server on the other side of r, it moves the same distance toward r.



Theorem 7.7 *DC is k-competitive.*

Before we prove Theorem 7.7, we need to settle on notation. Let $\{s_1, s_2, \ldots, s_k\}$ be the set of positions of our servers, where the indices indicate order in the line, *i.e.,*$s_1 \leq s_2 \leq \ldots \leq s_k$. Similarly, let $\{a_1, a_2, \ldots, a_k\}$, where $a_1 \leq a_2 \leq \ldots \leq a_k$, be the positions of the optimal servers.

In our analysis, a "black" node is an ordered $2k$-tuple $(s_1, s_2, \ldots, s_k, a_1, a_2, \ldots, a_k)$, while a "red" node is an ordered $(2k+1)$-tuple $(s_1, s_2, \ldots, s_k, a_1, a_2, \ldots, a_k, r)$, where r is the request point. Without loss of generality, the adversary is lazy, *i.e.,*moves only one server at a time, and then only to the request point; thus, we can assume that, at any red node, $r = a_i$ for some i.

Proof: We use the Coppersmith-Doyle-Raghavan-Snir potential, defined by

$$\Phi = \sum_{i<j}(s_j - s_i) + k \sum_i |s_i - a_i|$$

We need to prove the following two inequalities:

1. If x is a black node and the adversary moves from x to a red node y, then

$$\Phi(x) + k\, cost(x, y) \geq \Phi(y)$$

2. If y is a red node, then DC will move to a black node x such that

$$\Phi(y) \geq cost(y, x) + \Phi(x)$$

We first prove 1. Let $x = (s_1, s_2, \ldots, s_k, a_1, a_2, \ldots, a_k)$, and let $y = (s_1, s_2, \ldots, s_k, a'_1, a'_2, \ldots, a'_k, r)$. We can assume that the optimal servers do not pass each other, *i.e.,*the server at a_i moves to a'_i. Thus, $cost(x, y) = \sum_i |a_i - a'_i|$. Finally

$$\begin{aligned}
\Phi(x) + k\, cost(x, y) &= \sum_{i<j}(s_j - s_i) + k \sum_i |s_i - a_i| + k \sum_i \left|a_i - a'_i\right| \\
&= \sum_{i<j}(s_j - s_i) + k \sum_i \left(|s_i - a_i| + \left|a_i - a'_i\right| \right) \\
&\geq \sum_{i<j}(s_j - s_i) + k \sum_i \left|s_i - a'_i\right| \quad \text{(by the triangle inequality)} \\
&= \Phi(y)
\end{aligned}$$

We now prove 2. Let $y = (s_1, s_2, \ldots, s_k, a_1, a_2, \ldots, a_k, r)$, and let $x = (s'_1, s'_2, \ldots, s'_k, a_1, a_2, \ldots, a_k)$. In the trivial case that $r = s_i$ for some i, then $cost(y, x) = 0$ and $\Phi(x) = \Phi(y)$, and we are done. Thus, we can assume that $r \neq s_i$ for any i.

Case I: DC moves one server. Then $r < s_1$ or $r > s_k$. Without loss of generality, $r < s_1$, since the other case is similar. Only s_1 moves, a distance of $\delta = s_1 - r$. Thus, $\left(s'_j - s'_1\right) = (s_j - s_1) + \delta$ for all $1 < j \leq k$, and $\left(s'_j - s'_i\right) = (s_j - s_i)$ for all $2 \leq i < j \leq k$, and thus $\sum_{i<j}\left(s'_j - s'_i\right) = \sum_{i<j}(s_j - s_i) + (k-1)\delta$.

Since $r = a_i$ for some i, we know that $a_1 \le r = s'_1 \le s_1$. Thus, $|s'_1 - a_1| = |s_1 - a_1| - \delta$. Since only the server at s_1 moves, $|s'_1 - a_1| = |s_1 - a_1|$ for all $i > 1$. Thus, $\sum_i |s'_i - a_i| = \sum_i |s'_i - a_i| - \delta$.

$$\begin{aligned}
\Phi(y) &= \sum_{i<j}(s_j - s_i) + k \sum_i |s_i - a_i| \\
&= \sum_{i<j}(s'_j - s'_i) - (k-1)\delta + k \sum_i |s'_i - a_i| + k\,\delta \\
&= \Phi(x) + \delta \\
&= \Phi(x) + cost(y, x)
\end{aligned}$$

and we are done.

Case II: DC moves two servers. Then $s_p < r < s_{p+1}$ for some p. Let $\delta = \min\{r - s_p, s_{p+1} - r\}$. The server from s_p moves δ to the right, while the server from s_{p+1} moves δ to the left, and thus $cost(y, x) = 2\,\delta$.

We analyze the change in the first term of the CDRS potential by breaking the summation into four parts.

1. If $\{i, j\} \cap \{p, p+1\} = \emptyset$, then $s'_j - s'_i = s_j - s_i$.
2. If $i < p$, then $s'_p - s'_i = s_p - s_i + \delta$ and $s'_{p+1} - s'_i = s_{p+1} - s_i - \delta$
3. If $j > p+1$, then $s'_j - s'_p = s_j - s_p - \delta$ and $s'_j - s'_{p+1} = s_j - s_{p+1} + \delta$.
4. $s'_{p+1} - s'_p = s_{p+1} - s_p - 2\,\delta$

Combining the terms and canceling, we have $\sum_{i<j}(s'_j - s'_i) = \sum_{i<j}(s'_j - s'_i) - 2\,\delta$.

We now show that the second term of the CDRS potential does not increase. We know that $r = a_i$ for some i. If $i \le p$, the server from s_p moves closer to a_p. Thus $|s'_p - a_p| = |s_p - a_p| - \delta$, while, by the triangle inequality, $|s'_{p+1} - a_{p+1}| \le |s'_{p+1} - a_{p+1}| + |s_{p+1} - s'_{p+1}| = |s'_{p+1} - a_{p+1}| + \delta$. If $i \ge p+1$, the server from s_{p+1} moves closer to a_{p+1}. By an argument similar to the case $i \le p$, we know that $|s'_{p+1} - a_{p+1}| = |s_{p+1} - a_{p+1}| - \delta$ and $|s'_p - a_i| \le |s_p - a_i| + \delta$. If $i \notin \{p, p+1\}$, then $|s'_i - a_i| = |s_i - a_i|$. Thus $\sum_i |s'_i - a_i| \le \sum_i |s_i - a_i|$.

Combining those results, we have

$$\begin{aligned}
\Phi(y) &= \sum_{i<j}(s_j - s_i) + k \sum_i |s_i - a_i| \\
&\le \sum_{i<j}(s'_j - s'_i) + 2\,\delta + k \sum_i |s'_i - a_i| \\
&= \Phi(x) + 2\,\delta \\
&= \Phi(x) + cost(y, x)
\end{aligned}$$

and we are done. □

Exercise 7.8 *Let M be any metric space which is a subspace of $\mathbb{R}$. That is, M is any set of real numbers, and $d(x, y) = |x - y|$ for any $x, y \in M$. Using Theorem 7.7, prove that there exists a lazy k-competitive algorithm for the k-server problem in M. (This algorithm will not be memoryless.)*