List Accessing and k-Server Problems

Hsiang-Hsuan (Alison) Liu

1 List Accessing Problem

List Accessing problem

We are given a list of ℓ items with a pointer pointing at the beginning of the list. The `Access`(x) request is to find the item x in the list by moving the pointer through the list. Once the item x is found, moving x to any position closer to the front of the list is free. Find a reorganization rule that minimizes the search time for n `Access` operations.

1.1 The `Move-to-Front` Algorithm (`MTF`)

`Move-to-Front` Algorithm

After accessing an item, move it to the front of the list without changing the relative order of the other items.

Theorem 1. *`MTF` for the List Accessing problem is $2 - \frac{1}{\ell}$-competitive.*

Proof. Let the request sequence $\sigma = \{r_1, r_2, \cdots, r_n\}$. For any $i \in \{1, 2, \cdots, n\}$, we denote t_i as the (actual) cost that `MTF` incurs for processing request `Access`(r_i). We say that item i and j form an *inversion* if i is before j in the `MTF` list but i is after j in the `OPT` list. A potential function Φ_i is defined by the number of inversions in the `MTF` list and the `OPT` list after `Access`(r_i). For completeness, we let Φ_0 be the number of inversions in the initial lists. Assume that the algorithms start from the same list, $\Phi_0 = 0$. Let $a_i = t_i + \Phi_i - \Phi_{i-1}$ be the amortized cost for `MTF` processing request `Access`(r_i). By summing up all the amortized costs, we get

$$\texttt{MTF}(\sigma) = \sum_{i=1}^{n} t_i = \Phi_0 - \Phi_n + \sum_{i=1}^{n} a_i \leq \sum_{i=1}^{n} a_i.$$

The last inequality is from the fact that $\Phi_i \geq 0$ for any i and $\Phi_0 = 0$.

Next, we claim that $a_i \leq 2 \cdot \texttt{OPT}_i - 1$ for all i, where $\texttt{OPT}_i$ is the cost incurred by `OPT` on the i-th request. If the claim is true, $\texttt{MTF}(\sigma) \leq \sum_{i=1}^{n} a_i \leq \sum_{i=1}^{n} (2 \cdot \texttt{OPT}_i - 1) = 2 \cdot \texttt{OPT}(\sigma) - n$. Since $\texttt{OPT}(\sigma) \leq \ell \cdot n$, $\texttt{MTF}(\sigma) \leq 2 \cdot \texttt{OPT}(\sigma) - n \leq 2 \cdot \texttt{OPT}(\sigma) - \frac{\texttt{OPT}(\sigma)}{\ell}$.

Now, the only task left is to show the correctness of the claim (that is, $a_i \leq 2 \cdot \texttt{OPT}_i - 1$ for all i). For any request `Access`(r_i), assume that the item r_i is at the k-th position in the `MTF` list and at the j-th position in the `OPT` list. Assume that right before `Access`(r_i), there are g items before r_i in the `MTF` list but after r_i in the `OPT` list. These g items contribute to Φ_{i-1} but not to Φ_i due to the fact that `MTF` moves r_i to the front of the list after `Access`(r_i). On the other hand, the rest of $k - 1 - g$ items in the `MTF` list that are in front of r_i will be new inversions that contribute to Φ_i. Since there are $k - 1$ items before r_i in the `MTF` list, $\texttt{OPT}_i = j \geq (k - 1 - g) + 1 = k - g$. Therefore, $a_i = k + \Phi_i - \Phi_{i-1} = k + ((k - 1 - g) - g) = 2(k - r) - 1 \leq 2 \cdot \texttt{OPT}_i - 1$ as we desired.

Note that `OPT` might also move r_i to somewhere closer to the front of the list or toward the end by actively accessing an item b later in the list and swapping with b. In the front case, the potential further decreased. In the latter case, the potential is increased by exactly the extra cost by `OPT`. □

1.2 List Accessing problem lower bound

Theorem 2. *The* List Accessing *problem is at least* $2 - \frac{2}{\ell+1}$*-competitive*

Proof. Given any online algorithm ALG, we consider the adversary σ that keeps accessing the last item in the list. The cost incurred by ALG is $\ell \cdot n$, where n is the number of requests.

Now, we consider $\ell!$ static algorithms, each starting with one of the $\ell!$ permutations of the items, that never change their list ordering when serving the requests. Note that each static algorithm can achieve its initial list configuration by at most ℓ^2 swappings. For each request, the accessed item is at the i-th position in $(\ell-1)!$ of the static algorithms for all $i \in \{1, 2, \cdots, n\}$. Therefore, the total cost of serving all n requests of all static algorithms is

$$n \cdot \sum_{i=1}^{\ell} i \cdot (\ell-1)!.$$

The average cost of serving the n requests of a static algorithm is

$$\frac{n \cdot \sum_{i=1}^{\ell} i \cdot (\ell-1)!}{\ell!} = \frac{n(\ell+1)}{2}.$$

Thus, there is at least one static algorithm with a total cost of at most $\frac{n(\ell+1)}{2}$. Since the optimal solution is at least as good as this static algorithm, the optimal cost is at most $\frac{n(\ell+1)}{2} + \ell^2$. In this case,

$$\frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} \geq \frac{\ell n}{\frac{n(\ell+1)}{2} + \ell^2} = 2 - \frac{2}{\ell+1} \text{ when } n \text{ is large enough.}$$

□

2 k-Server Problem

k-SERVER problem

Let $k > 1$ be an integer, and let $(\mathcal{M}, d)$ be a metric space where $\mathcal{M}$ is a set of points with $|\mathcal{M}| > k$ and d is a metric over $\mathcal{M}$. The algorithm is presented with a sequence $\sigma = r_1, r_2, \cdots, r_n$ of n requests where a request r_i is a point in the space. We say that a request r is served if one of the servers is located at r. An algorithm must serve all the requests sequentially by moving the servers to the requests. The goal is to find a strategy that minimizes the total moving distance of all servers in servicing σ.

2.1 Greedy is unbounded for k-Server

`Greedy` Algorithm

When there is a request r that is not served, move the closet server to r to serve it.

Theorem 3. *`Greedy` algorithm has an unbounded competitive ratio*

Proof. Consider the adversarial instance where there are 2 servers and $\mathcal{M}$ has three points a, b, and c. Furthermore, let $d(a, b) = 1$, $d(b, c) = 2$, and $d(a, c) = 3$. Let the servers initially sit at a and c. (This can be done by first requesting a and c.) Next, the requests sequence is $b, a, b, a, b, a, \cdots$, repeating n times. The optimal strategy is to move the server at c to b, and the optimal cost is at most $3 + 3 + 2 = 6$. However, in the `Greedy` algorithm, the server at c will remain at c, while the other server will move between a and b. The cost of `Greedy` algorithm is then at least n. When n is arbitrarily large, the ratio $\frac{\texttt{Greedy}(\sigma)}{\texttt{OPT}(\sigma)} \geq \frac{n}{6} \approx n$. □

2.2 `Double Coverage` Algorithm (`DC`) on a line metric

`Double Coverage` Algorithm

If the request falls outside the convex hull of the servers, serve it with the nearest server
Otherwise, move the two closest servers toward the request at equal speeds until at least one server reaches it

Theorem 4. *`Double Coverage` algorithm is k-competitive*

Proof. Let $M_{\min}$ denote the cost of the minimum cost matching between OPT's and DC's servers. Letting $s_1, s_2, \cdots, s_k$ be the position of DC's servers, we denote $\Sigma_{DC} = \sum_{i<j} d(s_i, s_j)$. We define the potential function $\Phi = k \cdot M_{\min} + \sum_{DC}$. Now, considering the special order where once a request is released, OPT first moves its server, and then DC moves, we claim that:

1. If OPT moves a distance d, the potential change $\Delta\Phi_i = \Phi_i - \Phi_{i-1} \leq kd$, and
2. If DC moves d, the potential change $\Delta\Phi_i = \Phi_i - \Phi_{i-1} \leq -d$.

If we have the following claim, $\texttt{DC}_i + \Delta\Phi_i \leq k \cdot \texttt{OPT}_i$, and $\texttt{DC} \leq k \cdot \texttt{OPT}$ as $\Phi_i \geq 0$ for any i.

Now, we prove the two claims.

1. Since the DC servers do not move, $\sum_{DC}$ remains the same. On the other hand, $M_{\min}$ cannot increased by more than d. Therefore, $\Delta M_{\min} \leq kd$.
2. There are two cases where DC moves its servers:

(a) If DC moves a single server for a distance d, then $\Delta \sum_{DC} = (k-1) \cdot d$ since this server must move away from other $k-1$ servers. Moreover, $\Delta M_{\min} = -d$ because DC just served this new request and has a server at this position. In total, $\Delta\Phi_i = k \cdot \Delta M_{\min} + \Delta \sum_{DC} = -kd + (k-1)d = -d$

(b) If DC moves two servers, s_1 and s_2, each for a distance of $d' = \frac{d}{2}$, they move toward each other. Without loss of generality, we assume that s_1 is on the left of s_2, and s_2 serves the request r_i. The only change in $\sum_{DC}$ is $\Delta d(s_1, s_2) = -2d'$ since for any other server s, $\Delta d(s, s_1) = -\Delta d(s, s_2)$. For the change of $M_{\min}$, there are two cases. If the optimal server matched with s_1 is at r_i or its right, the matching regarding s_1 is decreased by d', and the matching regarding s_2 is increased by d'. Thus, the $\Delta M_{\min} = 0$ in this case. On the other hand, if the optimal sever matched with s_2 is at r_i or its left, the matching regarding s_2 is decreased by d', and the matching regarding s_1 is increased by at most d'. Thus, $\Delta M_{\min} \leq 0$. Therefore, in the case where DC moves two servers, the change of potential $\Delta\Phi_i \leq k \cdot 0 + 2d' = d$.

□

2.3 k-Server problem lower bound

Theorem 5. *The k-Server problem is at least k-competitive*

Proof. Consider a metric space with $k+1$ points, $v_0, v_1, v_2, \cdots, v_k$. For any algorithm ALG, which has its k servers at points $v_1, v_2, \cdots, v_k$, we design the adversary $\sigma = r_1, r_2, \cdots, r_n$ that always requests the position that there is no ALG server. The request sequence σ starts with requesting v_0. The total cost of ALG on the n requests is $\sum_{i=1}^{n} d(r_i, r_{i+1})$.

Let $B_1, B_2, \cdots B_k$ be k algorithms where the servers of B_h are at any points but v_h initially. (Note that each B_h has a server at $r_1 = v_0$.) Each B_h serves an uncovered request r_i with the server occupying r_{i-1}. That is, if a server covers r_i, B_h does nothing.

We claim that after each request is processed, different algorithms will always be in different configurations. Then, the total cost incurred by all B_h serving σ is

$$\sum_{i=2}^{n} d(r_{i-1}, r_i) = \sum_{i=1}^{n-1} d(r_i, r_{i+1}).$$

It follows that the average cost of these algorithms is

$$\frac{\sum_{i=1}^{n-1} d(r_i, r_{i+1})}{k} = \frac{\mathrm{ALG}(\sigma)}{k}.$$

By the average bound argument, $\frac{\mathrm{ALG}(\sigma)}{\mathrm{OPT}(\sigma)} \geq k$.

Now, we prove the correctness of the claim by induction. By definition of B_h's, initially, all B_h algorithms have different configurations. Given any two algorithms B_h and B_ℓ, letting S_h^j be the configuration of B_h after request r_j, there are three cases concerning request r_i:

1. r_i **is in** S_h^{i-1} **and** S_ℓ^{i-1}**.** In this case, both B_h and B_ℓ do not move, and $S_n^i = S_h^{i-1} \neq S_\ell^{i-1} = S_\ell^i$.
2. r_i **is in** S_h^{i-1} **but not in** S_ℓ^{i-1}**.** In this case, B_h does not move, but B_ℓ moves the server located at r_{i-1} to r_i. Thus, $S_h^i = S_h^{i-1}$ contains r_{i-1} while S_ℓ^i does not.
3. r_i **is in neither** S_h^{i-1} **nor** S_ℓ^{i-1}**.** In this case, $S_h^{i-1} \setminus \{r_{i-1}\} \neq S_\ell^{i-1} \setminus \{r_{i-1}\}$. Thus, $S_h^i \neq S_\ell^i$.

□