Advanced Algorithms 11 December 2018

Lecture 13: Online Algorithms and Competitive Analysis II

Lecturer: Mohsen Ghaffari *Scribe: Davin Choo*

We begin this lecture by describing and analyzing a randomized algorithm for the paging problem, then discuss a general online problem called the k-server problem.

1 Paging (Continued)

Definition 1 (Paging problem [ST85]). *Suppose we have a fast memory (cache) that can fit k pages and an unbounded sized slow memory. Accessing items in the cache costs 0 units of time while accessing items in the slow memory costs 1 unit of time. After accessing an item in the slow memory, we can bring it into the cache by evicting an incumbent item if the cache was full. What is the best online strategy for maintaining items in the cache to minimize the total access cost on a sequence of queries?*

1.1 Random Marking Algorithm (RMA) — A $\mathcal{O}(\log k)$-competitive algorithm for paging against oblivious adversaries

Consider the Random Marking Algorithm (RMA):

- Initialize all pages as marked
- Upon request of a page p
  - If p is not in cache,
    - If all pages in cache are marked, unmark all
    - Evict a random unmarked page
  - Mark page p

Example Suppose $k = 3$, $\sigma = (2, 5, 2, 1, 3)$.

Suppose the cache is initially:

Cache	1	3	4
Marked?	✓	✓	✓

When $\sigma(1) = 2$ arrives, all pages were unmarked. Suppose the random eviction chose page '3'. The newly added page '2' is then marked.

Cache	1	2	4
Marked?	✗	✓	✗

When $\sigma(2) = 5$ arrives, suppose random eviction chose page '4' (between pages '1' and '4'). The newly added page '5' is then marked.

Cache	1	2	5
Marked?	✗	✓	✓

When $\sigma(3) = 2$ arrives, page '2' in the cache is marked (no change).

Cache	1	2	5
Marked?	✗	✓	✓

When $\sigma(4) = 1$ arrives, page '1' in the cache is marked. At this point, any page request that is not from $\{1, 2, 5\}$ will cause a full unmarking of all pages in the cache.

Cache	1	2	5
Marked?	✓	✓	✓

When $\sigma(5) = 3$ arrives, all pages were unmarked. Suppose the random eviction chose page '5'. The newly added page '3' is then marked.

Cache	1	2	3
Marked?	✗	✗	✓

We denote a phase as the time period between 2 consecutive full unmarking steps. That is, each phase is a maximal run where we access k distinct pages. In the above example, $\{2, 5, 2, 1\}$ is such a phase for $k = 3$.

Observation As pages are only unmarked at the beginning of a new phase, the number of unmarked pages is monotonically decreasing within a phase.

Theorem 2. *RMA is $\mathcal{O}(\log k)$-competitive against any oblivious adversary.*

Proof. Let P_i be the set of pages at the *start* of phase i. Since requesting a marked page does not incur any cost, it suffices to analyze the *first time* any request occurs within the phase.

- Denote $\mathcal{N}$ as the set of *new requests* (page that are not in P_i).
- Denote $\mathcal{O}$ as the set of *old requests* (pages that are in P_i).
- By definition, $|\mathcal{O}| \leq k$ and $|\mathcal{N}| + |\mathcal{O}| = k$. Define $m_i = |\mathcal{N}|$.

Since every new request incurs a unit cost, the cost due to $\mathcal{N}$ is m_i.

$$\sigma = (\quad \cdots \quad \overbrace{\text{new} \quad \text{new} \quad \text{new} \quad \underset{\substack{1^{st} \\ l_1 = 3}}{\text{old}} \quad \text{new} \quad \underset{\substack{2^{nd} \\ l_2 = 4}}{\text{old}} \quad \cdots}^{\text{Phase } i} \quad \cdots \quad)$$

For $j \in \{1, \ldots, |\mathcal{O}|\}$, consider the first time the j^{th} old request x_j occurs. There are $k - (j - 1)$ possibilities for x_j. Say, there are l_j *distinct new* requests before it. Assuming *all* previous requests kicked out an old page from the cache, there are $\geq k - l_j - (j - 1)$ old pages remaining in the cache. So, $\Pr[x_j \text{ is } \textit{not} \text{ cache when requested}] \leq \frac{(k-(j-1))-(k-l_j-(j-1))}{k-(j-1)} = \frac{l_j}{k-(j-1)}$. Then,

$$\begin{aligned}
\text{Cost due to } \mathcal{O} &\leq \textstyle\sum_{j=1}^{|\mathcal{O}|} \frac{l_j}{k-(j-1)} && \text{Summing over all distinct } |\mathcal{O}| \text{ old requests} \\
&\leq \textstyle\sum_{j=1}^{|\mathcal{O}|} \frac{m_i}{k-(j-1)} && \text{Since } l_j \leq m_i = |\mathcal{N}| \\
&\leq m_i \cdot \textstyle\sum_{j=1}^{k} \frac{1}{k-(j-1)} && \text{Since } |\mathcal{O}| \leq k \\
&= m_i \cdot \textstyle\sum_{j=1}^{k} \frac{1}{j} && \text{Rewriting} \\
&= m_i \cdot H_k && \text{Since } \textstyle\sum_{i=1}^{n} \frac{1}{i} = H_n
\end{aligned}$$

Hence, $c_{RMA}(\text{Phase } i) = (\text{Cost due to } \mathcal{N}) + (\text{Cost due to } \mathcal{O}) \leq m_i + m_i \cdot H_k$.

We now analyze OPT's performance. By definition of phases, among all requests between two consecutive phases (say, $i - 1$ and i), a total of $k + m_i$ distinct pages are requested. So, OPT has to incur at least $\geq m_i$ to bring in these new pages. To avoid double counting, we lower bound $c_{OPT}(\sigma)$ by both odd and even i: $c_{OPT}(\sigma) \geq \sum_{\text{odd } i} m_i$ and $c_{OPT}(\sigma) \geq \sum_{\text{even } i} m_i$. Together,

$$2 \cdot c_{OPT}(\sigma) \geq \sum_{\text{odd } i} m_i + \sum_{\text{even } i} m_i \geq \sum_i m_i$$

Therefore, we have:

$$c_{RMA}(\sigma) \leq \sum_i (m_i + m_i \cdot H_k) = \mathcal{O}(\log k) \sum_i m_i \leq \mathcal{O}(\log k) \cdot c_{OPT}(\sigma)$$

□

Remark In the above example, $k = 3$, phase $1 = (2, 5, 2, 1)$, $P_1 = \{1, 3, 4\}$, $\mathcal{N} = \{5\}$, $\mathcal{O} = \{2, 1\}$. Although '2' appeared twice, we only care about analyzing the first time it appeared.

2 Yao's Minimax Principle

Given the sequence of random bits used, a randomized algorithm behaves deterministically. Hence, one may view a randomized algorithm as a random choice from a distribution of deterministic algorithms.

Let X be the space of problem inputs and A be the space of all possible deterministic algorithms. Denote probability distributions over A and X by $p_a = \Pr[A = a]$ and $q_x = \Pr[X = x]$ respectively. Define $c(a, x)$ as the cost of algorithm $a \in A$ on input $x \in X$.

Theorem 3 ([Yao77]).

$$C = \max_{x \in X} \mathbb{E}_p[c(A, x)] \geq \min_{a \in A} \mathbb{E}_q[c(a, X)] = D$$

Proof.

$$\begin{array}{rcll} C & = & \sum_q q_x \cdot C & \text{Sum over all possible inputs } x \\ & \geq & \sum_q q_x \mathbb{E}_p[c(A, x)] & \text{Since } C = \max_{x \in X} \mathbb{E}_p[c(A, x)] \\ & = & \sum_q q_x \sum_p p_a c(a, x) & \text{Definition of } \mathbb{E}_p[c(A, x)] \\ & = & \sum_p p_a \sum_q q_x c(a, x) & \text{Swap summations} \\ & = & \sum_p p_a \mathbb{E}_q[c(a, X)] & \text{Definition of } \mathbb{E}_q[c(a, X)] \\ & \geq & \sum_p p_a \cdot D & \text{Since } D = \min_{a \in A} \mathbb{E}_q[c(a, X)] \\ & = & D & \text{Sum over all possible algorithms } a \end{array}$$

$\square$

Implication If one can argue that *any* deterministic algorithm cannot do well on a given distribution of random inputs, then no randomized algorithm can do well on *all* inputs.

2.1 Application to the paging problem

Theorem 4. *Any (randomized) algorithm has competitive ratio $\Omega(\log k)$ against an oblivious adversary.*

Proof. Fix an arbitrary deterministic algorithm $\mathcal{A}$. Let $n = k + 1$ and $|\sigma| = m$. Consider the following random input sequence σ where the i-th page is drawn from $\{1, \ldots, k + 1\}$ uniformly at random.

By construction of σ, the probability of having a cache miss is $\frac{1}{k+1}$ for $\mathcal{A}$, regardless of what $\mathcal{A}$ does. Hence, $\mathbb{E}[c_{\mathcal{A}}(\sigma)] = \frac{m}{k+1}$.

On the other hand, an optimal offline algorithm may choose to evict the page that is requested furthest in the future. As before, we denote a phase as a maximal run where there are k distinct page requests. This means that $\mathbb{E}[c_{OPT}(\sigma)] = \text{Expected number of phases} = \frac{m}{\text{Expected phase length}}$.

To analyze the expected length of a phase, suppose there are i distinct pages so far, for $0 \leq i \leq k$. The probability of the next request being new is $\frac{k+1-i}{k+1}$, and one expects to get $\frac{k+1}{k+1-i}$ requests before having $i + 1$ distinct pages. Thus, the expected length of a phase is $\sum_{i=0}^{k} \frac{k+1}{k+1-i} = (k + 1) \cdot H_{k+1}$. Therefore, $\mathbb{E}[c_{OPT}(\sigma)] = \frac{m}{(k+1) \cdot H_{k+1}}$.

Putting together, we have $\frac{\mathbb{E}[c_{\mathcal{A}}(\sigma)]}{\mathbb{E}[c_{OPT}(\sigma)]} = H_{k+1} = \Theta(\log k)$. $\square$

Remark The length of a phase is essentially the coupon collector problem with $n = k + 1$ coupons.

3 The k-server problem

Definition 5 (k-server problem [MMS90]). *Consider a metric space (V, d) where V is a set of n points and $d : V \times V \to \mathbb{R}$ is a distance metric between any two points. Suppose there are k servers placed on V and we are given an input sequence $\sigma = (v_1, v_2, \ldots)$. Upon request of $v_i \in V$, we have to move one server to point v_i to satisfy that request. What is the best online strategy to minimize the total distance travelled by servers to satisfy the sequence of requests?*

Remark We do not fix the starting positions of the k servers, but we compare the performance of OPT on σ with same initial starting positions.

The paging problem is a special case of the k-server problem where the points are all possible pages, the distance metric is unit cost between any two different points, and the servers represent the pages in cache of size k.

3.1 Progress

It is conjectured that a deterministic k-competitive algorithm exists and a randomized $(\log k)$-competitive algorithm exists. The table below shows the current progress on this problem.

	Competitive ratio	Type
[MMS90]	k-competitive, for $k = \{2, n-1\}$	Deterministic
[FRR90]	$2^{\mathcal{O}(k \log k)}$-competitive	Deterministic
[Gro91]	$2^{\mathcal{O}(k)}$-competitive	Deterministic
[KP95]	$(2k-1)$-competitive	Deterministic
[BBMN11]	$\text{poly}(\log n, \log k)$-competitive	Randomized
[Lee18]	$\mathcal{O}(\log^6 k)$-competitive	Randomized

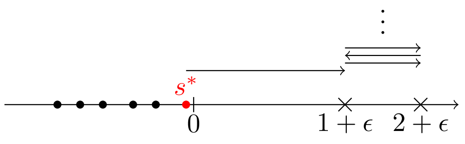
Remark [BBMN11] uses a probabilistic tree embedding, a concept we have seen in earlier lectures.

3.2 Special case: Points on a line

Consider the metric space where V are points on a line and $d(u,v)$ is the distance between points $u, v \in V$. One can think of all points lying on the 1-dimensional number line $\mathbb{R}$.

3.2.1 Greedy is a bad idea

A natural greedy idea would be to pick the closest server to serve any given request. However, this can be arbitrarily bad. Consider the following:

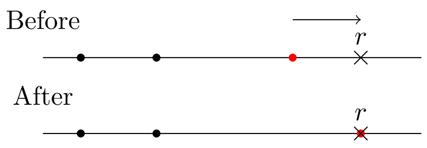


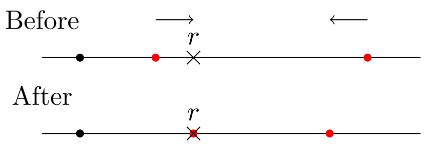
Without loss of generality, suppose all servers currently lie on the left of "0". For $\epsilon > 0$, consider the sequence $\sigma = (1+\epsilon, 2+\epsilon, 1+\epsilon, 2+\epsilon, \dots)$. The first request will move a single server s^* to "$1+\epsilon$". By the greedy algorithm, subsequent requests then repeatedly use s^* to satisfy requests from both "$1+\epsilon$" and "$2+\epsilon$" since s^* is the closest server. This incurs a total cost of $\geq |\sigma|$ while OPT could station 2 servers on "$1+\epsilon$" and "$2+\epsilon$" and incur a constant total cost on input sequence σ.

3.2.2 Double coverage — A k-competitive algorithm for points on a line

The *double coverage* algorithm does the following:

- If request r is on one side of all servers, move the closest server to cover it
- If request r lies between two servers, move both towards it at *constant speed* until r is covered





Theorem 6. *Double coverage (DC) is k-competitive on a line.*

Proof. Without loss of generality,

- Suppose location of DC's servers on the line are: $x_1 \leq x_2 \leq \cdots \leq x_k$
- Suppose location of OPT's servers on the line are: $y_1 \leq y_2 \leq \cdots \leq y_k$

Define potential function $\Phi = \Phi_1 + \Phi_2 = k \cdot \sum_{i=1}^{k} |x_i - y_i| + \sum_{i<j}(x_j - x_i)$, where Φ_1 is k times the "paired distances" between x_i and y_i and Φ_2 is the pairwise distance between any two servers in DC. For a given request r at time step t, we will first analyze OPT's action then DC's action. We analyze the change in potential $\Delta(\Phi)$ by looking at $\Delta(\Phi_1)$ and $\Delta(\Phi_2)$ separately, and further distinguish the effects of DC and OPT on $\Delta(\Phi)$ via $\Delta_{DC}(\Phi)$ and $\Delta_{OPT}(\Phi)$ respectively.

Suppose OPT moves server s^* by a distance of $x = d(s^*, r)$ to reach the point r, $c_{OPT}(t) \geq x$. Since s^* moved by x, $\Delta(\Phi_1) \leq kx$. Since OPT does not move DC's servers, $\Delta(\Phi_2) = 0$. Hence, $\Delta_{OPT}(\Phi_t) \leq kx$.

There are three cases for DC, depending on where r appears.

1. r appears exactly on a current server position
DC does nothing. So, $c_{DC}(t) = 0$ and $\Delta_{DC}(\Phi_t) = 0$. Hence,

$$c_{DC}(t) + \Delta(\Phi_t) = c_{DC}(t) + \Delta_{DC}(\Phi_t) + \Delta_{OPT}(\Phi_t) \leq 0 + kx + 0 = kx \leq k \cdot c_{OPT}(t)$$

2. r appears on one side of all servers $x_1, \ldots, x_k$ (say $r > x_k$ without loss of generality)
DC will move server x_k by a distance $y = d(x_k, r)$ to reach point r. That is, $c_{DC}(t) = y$. Since OPT has a server at r, $y_k \geq r$. So, $\Delta_{DC}(\Phi_1) = -ky$. Since only x_k moved, $\Delta_{DC}(\Phi_2) = (k-1)y$. Hence,

$$c_{DC}(t) + \Delta(\Phi_t) = c_{DC}(t) + \Delta_{DC}(\Phi_t) + \Delta_{OPT}(\Phi_t) \leq y - ky + (k-1)y + kx = kx \leq k \cdot c_{OPT}(t)$$

3. r appears between two servers $x_i < r < x_{i+1}$
Without loss of generality, say r is closer to x_i and denote $z = d(x_i, r)$. DC will move server x_i by a distance of z to reach point r, and server x_{i+1} by a distance of z to reach $x_{i+1} - z$. That is, $c_{DC}(t) = 2z$.

Claim 7. *At least one of x_i or x_{i+1} is moving closer to its partner (y_i or y_{i+1} respectively).*

Proof. Suppose, for a contradiction, that both x_i and x_{i+1} are moving *away* from their partners. That means $y_i \leq x_i < r < x_{i+1} \leq y_{i+1}$ at the end of OPT's action (before DC moved x_i and x_{i+1}). This is a contradiction since OPT must have a server at r but there is no server between y_i and y_{i+1} by definition. □

Since at least one of x_i or x_{i+1} is moving closer to its partner, $\Delta_{DC}(\Phi_1) \leq z - z = 0$.

Meanwhile, since x_i and x_{i+1} are moved a distance of z towards each other, $(x_{i+1} - x_i) = -2z$ while the total change against other pairwise distances cancel out, so $\Delta_{DC}(\Phi_2) = -2z$.

Hence,

$$c_{DC}(t) + \Delta(\Phi_t) = c_{DC}(t) + \Delta_{DC}(\Phi_t) + \Delta_{OPT}(\Phi_t) \leq 2z - 2z + kx = kx \leq k \cdot c_{OPT}(t)$$

In all cases, we see that $c_{DC}(t) + \Delta(\Phi_t) \leq k \cdot c_{OPT}(t)$. Hence,

$$\begin{array}{rrcll}
 & \sum_{t=1}^{|\sigma|} c_{DC}(t) + \Delta(\Phi_t) & \leq & \sum_{t=1}^{|\sigma|} k \cdot c_{OPT}(t) & \text{Summing over all inputs in the sequence } \sigma \\
\Rightarrow & \sum_{t=1}^{|\sigma|} c_{DC}(t) + (\Phi_{|\sigma|} - \Phi_0) & \leq & k \cdot c_{OPT}(\sigma) & \text{Telescoping} \\
\Rightarrow & \sum_{t=1}^{|\sigma|} c_{DC}(t) & \leq & k \cdot c_{OPT}(\sigma) & \text{Since } \Phi_t \geq 0 = \Phi_0 \\
\Rightarrow & c_{DC}(\sigma) & \leq & k \cdot c_{OPT}(\sigma) & \text{Since } c_{DC}(\sigma) = \sum_{t=1}^{|\sigma|} c_{DC}(t)
\end{array}$$

□

Remark One can generalize the approach of double coverage to points on a tree. The idea is as follows: For a given request point r, consider the set of servers S such that for $s \in S$, there is no other server s' between s and r. Move all servers in S towards r "at the same speed" until one of them reaches r.

References

[BBMN11] Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 267–276. IEEE, 2011.

[FRR90] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 454–463. IEEE, 1990.

[Gro91] Edward F Grove. The harmonic online k-server algorithm is competitive. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 260–266. ACM, 1991.

[KP95] Elias Koutsoupias and Christos H Papadimitriou. On the k-server conjecture. *Journal of the ACM (JACM)*, 42(5):971–983, 1995.

[Lee18] James R. Lee. Fusible hsts and the randomized k-server conjecture. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 438–449, 2018.

[MMS90] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[ST85] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Yao77] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 222–227. IEEE, 1977.