CS787: Advanced Algorithms	**Scribe:** Jennifer Cao,Sihan Liu
Lecture 15: Online Algorithm III	**Date:** October 24, 2019

Last time we check investigate the online algorithms for page caching.

- For deterministic algorithm, we show that **FIFO(First In First Out)** and **LRU(Least Recently Used** are **k-competitive** and the bound is tight for any deterministic algorithm.
- For the randomized setting, we show that **Marking Algorithm** has achieved competitive ratio of $\log k$.

15.1 $\log k$ is tight for any randomized paging algorithm

It is hard to construct a deterministic adversarial setting directly due to the randomness. Instead, we formulate the problem in a game theoretical manner where the problem essentially becomes a min-max game between algorithm designer and the sequence constructor.

15.1.1 Yao's Min-max Principle

Let α be one specific decision instance made by the algorithm and A be the set of all possible instances.

Let σ be the one specific sequence instance and Σ be the set of all possible sequence instances.

Denote $\Delta(\cdot)$ as the mixture operator.

Denote $cost(\alpha, \sigma)$ as the cost by applying decisions α to the input σ.

The game is essentially:

$$\min_{\alpha \sim \Delta(A)} \max_{\sigma \sim \Delta(\Sigma)} \mathbb{E}\left[\frac{cost(\alpha, \sigma)}{cost(OPT, \sigma)}\right]$$

Applying the min-max theorem, we can reduce it to:

$$\begin{aligned}
&\min_{\alpha \sim \Delta(A)} \max_{\sigma \sim \Delta(\Sigma)} \mathbb{E}\left[\frac{cost(\alpha, \sigma)}{cost(OPT, \sigma)}\right] \\
&= \max_{\sigma \sim \Delta(\Sigma)} \min_{\alpha \sim \Delta(A)} \mathbb{E}\left[\frac{cost(\alpha, \sigma)}{cost(OPT, \sigma)}\right] \\
&\geq \min_{\alpha \sim A} \mathbb{E}\left[\frac{cost(\alpha, \sigma)}{cost(OPT, \sigma)}\right], \forall \sigma \sim \Delta(\Sigma)
\end{aligned}$$

(notice that we have got rid of the randomness in our decisions)

Theorem 15.1.1 *Any randomized online algorithm cannot do better than* $\mathbf{\Omega(\log k)}$

Our sequence only picks from page $1, \ldots, k, k+1$, where k is the cache capacity. At every time, it picks a random page uniformly from the options. Then, it can easily be shown that regardless of what decisions the algorithm has made, its expected cost over T steps will be: $\mathbb{E}\left[cost(\alpha, \sigma)\right] = \frac{T}{k+1}$ as a cache miss occurs w.p $P = \frac{1}{k+1}$.

Next, define **phase** as the shortest sub-sequence where all $(k+1)$ pages has appeared for at least once. The O.P.T will make exactly 1 cache miss during one phase. Therefore, for a phase σ_p of length T, it can be shown that: $\mathbb{E}\left[cost(OPT, \sigma_p)\right] = \frac{T}{\mathbb{E}[\text{length of the phase}]}$

Calculating the length of a phase is exactly the standard coupon collector problem.

With probability 1, we will see the first new page at the start of the sequence.

Assume that we have already seen l pages, then the probability that the next page is a unseen new page will be $P = \frac{k+1-l}{k+1}$.

Thus, we have:

$$\begin{aligned}\mathbb{E}\left[\text{length of a phase}\right] &= 1 + \frac{k+1}{k} + \frac{k+1}{k-1} + \cdots + k + 1 \\ &= (k+1) \cdot H_{k+1}, \text{where } H_{k+1} \text{ is the } (k+1)_{th} \text{ harmonic series}\end{aligned}$$

In summary, we have:

$$\begin{aligned}cost(OPT) &= \frac{T}{(k+1) \cdot H_{k+1}} \\ cost(Algo) &= \frac{T}{(k+1)}\end{aligned}$$

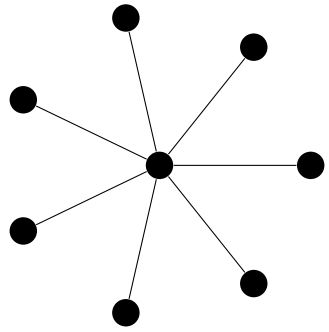
Thus, any randomized cannot do better than $\Omega H_{k+1} \approx \Omega(\log k)$ competitive ratio under the given random sequence.

15.2 K-Server Problem

Initially, we have k-servers scattered around a metric space. A sequence of requests happen and we need to send a server to handle each response. The objective is to minimize the total distance traveled by the servers.

15.2.1 K-Server Problem captures the paging problem

For a cache with capacity k, we define a star graph with $k + 2$ vertices:



$K_{1,7}$

Our servers scatter on the corner vertices. Every time, a response happen at one of the corner vertices. If a request happens at one of the vertex without a server, another server has to move to the vertex with a cost of 2 and the process thereby corresponds to a cache miss.

15.2.2 Competitive Online Algorithm for k-server

We focus on the strategy when the metric space is a 1-d straight line though it can be generalized to any arbitrary metric space.

Algorithm 1: Move servers in response to request

---

if Requests happen outside the interval covered by the servers **then**
  Move the closest server to handle the event
else
  Move both the server on the left side and right side of the request until the first one arrives
end if

To help us conduct the competitive analysis, we define the following auxiliary potential function:

$\varphi = k\cdot$(min cost match between server positions by Algo and OPT)+(pair-wise distances of servers in Algo)

Make the following notation

- $M =$ (min cost match between server positions by Algo and OPT).
- $S =$ (pair-wise distances of servers in Algo).

Next, we consider how moves from the Optimal algorithm and our algorithm will influence the potential function. For convenience of analysis, we assume the optimal algorithm moves first, and our algorithm moves after in response to a request.

Lemma 15.2.1 *For two different server arrangements X_{OPT} and X_{Algo}, we can always first sort X_{OPT} and X_{Algo} on the line and then match them by order to obtain a min cost matching.*

The lemma is useful when arguing for the change ΔM.

Suppose O.P.T moves by d:

- $\Delta M \leq d$ since our algorithm has not moved yet and thus the original match immediately gives $\Delta M = d$. The optimal match will only have smaller cost. Thus, we must have $\Delta M \leq d$.
- $\Delta S = 0$ since our algorithm has never moved.

Suppose our algorithm moves by d:
If the request is outside the range, we have:

- $\Delta M = -d$, as the server moved by our algorithm gets closer to the rightmost server of the O.P.T for exactly d.
- $\Delta S = (k-1)d$, as rightmost server of our algorithm is getting further away from all the other servers.

If the request is within the interval of servers L and R,

- $\Delta M <= 0$. By our lemma, there could be at two types of min-cost matching before our algorithm moves.
  - Both L and R get matched to some servers located on the left of the request (inclusive). Then R gets closer to its original matching partner for exactly d units and L gets further from its partner for at most d units.
  - Both L and R get matched to some servers located on the right of the request (inclusive). Then L gets closer to its original matching partner for exactly d units and R gets further from its partner for at most d units.
- $\Delta S = -2d$ as the two servers are moving towards each other and their distance to other servers cancel out exactly.

In summary, we must have $\Delta\varphi_i \leq k\Delta Cost(OPT)_i - \Delta Cost(Algo)_i$, where $\Delta Cost(OPT)_i$ and $\Delta Cost(Algo)_i$ is the distance moved by O.P.T and by our algorithm accordingly in response to request i.

Summing it over all the steps/requests from $1, \cdots, i$, we then have:
$\varphi_i \leq k \cdot Cost(OPT)_i + Cost(Algo)_i$, where $Cost(OPT)_i$ and $Cost(Algo)_i$ are the cumulative distance/cost moved by O.P.T and our algorithm accordingly up to request i.

Since $_i \geq 0$, we thus have $k \cdot Cost(OPT)_i \leq Cost(Algo)_i$.

Therefore, our algorithm is k-competitive.