Online Algorithms 16/11/11

Lecture Notes 3: Paging, K-Server and Metric Spaces

Professor: Yossi Azar *Scribe:Maor Dan*

1 Introduction

This lecture covers the Paging problem. We present a competitive online algorithms and a lower bound on competitive algorithms for solving the paging problem. We also discussed the K-Servers problem in Metric Spaces, presented some examples for K-Servers in different Metric Spaces, and presented a lower bound for competitive K-Servers algorithms.

2 Paging

In the Paging problem we have **K** slots for memory pages and an unlimited number of pages read requests, σ.

example($K = 3$):

$$\sigma = 1, 2, 3, 2, 4, 2, 5, 2, 1, 2, 1, 5, 2, 5, 3, ...$$

5	6	4

If a page isn't mapped to one of the slots then the Algorithm pays a price of **1** and also needs to decide which mapped page to replace with the new one. We want to find an algorithm that minimizes the number of page faults (the event of replacing a page).

2.1 LRU

A Common Paging Algorithm is LRU - Least Recently Used. LRU, upon a page fault, replaces the mapped page which was last used farthest in the past with the new required page.

Theorem 2.1. *LRU is K-Competitive.*

Proof. Let σ be a sequence of page requests. We split this sequence to "Blocks" according to the following: the longest sub-sequence with exactly K different page requests.

$\sigma = \sigma_1\sigma_2\sigma_3\sigma_4...$

where each σ_i is a "block".

so, in the previous example with K = 3:

$$\sigma = \underbrace{1,2,3,2}_{\sigma_1}, \underbrace{4,2,5,2}_{\sigma_2}, \underbrace{1,2,1,5,2,5}_{\sigma_3}, 3, ...$$

LRU might get a page fault at most once for each page in the "block", hence, at most K times over each σ_i. We want to claim that **OPT** gets a page fault at least once every block.

Consider the following example:
In block σ_2 **OPT** may finish with the following page slots mapping:

2	5	1

and our claim fails because **OPT** will not have a page fault in block σ_3.

If we try dividing into K+1 "blocks" we will make sure **OPT** will get at least one page fault in each block, but at the same time, we can no longer guarantee that **LRU** will only get K page faults.

Note that **OPT** fails at least once for each two consecutive blocks. That is due to the fact that two consecutive blocks contains at least K+1 different page requests and hence **OPT** fails at least once. This proves 2K-competitiveness and we are interested to prove K-competitiveness.

We'll show that **OPT** fails at least once for each "Shifted Block". We define a Shifted Block, $\sigma'_i = (\sigma_i - P_i) \bigcup P_{i+1}$
Where P_i is the first page request in σ_i.

At the beginning of the shifted block P_i must be mapped to one of the slots because it was last request. Also, in σ'_i there are at least K page requests that are different from P_i because P_{i+1} is different from all page requests in σ_i and σ_i contains exactly K different page requests. Therefore there must be at least one page fault for **OPT**. Note that the last block might not be full. Hence, assuming we have t blocks:

LRU $\leq k \cdot t$
OPT $\geq t - 1$

If we assume the legitimate assumption that **OPT** and **LRU** begin in the same state we get that they both fail on P_1 which is considered a shifted block by itself (σ'_0). In this case we get

OPT $\geq t$

and LRU is K competitive.

Note: This proof holds also for proving **FiFo** is K-competitive. **LRU** keeps a timestamp for each mapped page of the last access, and upon a fault, it replaces the page with the earliest timestamp. **FiFo** keeps a timestamp for each mapped page of the first access and upon a fault, it replaces the page with the minimal timestamp. □

2.2 Lower Bound

Theorem 2.2. *Every Deterministic Paging algorithm is at least K-Competitive. (assuming the number of different possible pages, $n \geq k+1$)*

Proof. WLOG $n = k+1$ (because we can choose our input sequence as we like and we choose one that contains $k+1$ different pages) Let A be a paging algorithm, we choose σ so each page request is the exact one page that A is missing from the memory at this moment. Algorithm A fails at each request.

The Optimal Algorithm, at each page fault, replaces the page that it will need in the most distant future. This future requests will be at least K request into the future.

This is since only one page is missing and hence there is a page which will not be requested in the next k-1 requests.

The optimal algorithm fails at most once every k requests and A fails at each request. Therefore A is at least K-Competitive (in the bad meaning of at least) □

3 K-Servers

3.1 Background

A Metric Space: A collection of points, X where between every pair of points a distance is defined by a function $d(x, y)$ that satisfy the following:

1. $d(x, y) \geq 0$ **And** $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$
3. The Triangle Inequality: $d(x, y) + d(y, z) \geq d(x, z)$

A Uniform Metric: A Metric space where $d(x, y) = 1$ for any $x \neq y$

An example of a Metric Space:

A Graph, $G = (V, E)$, $W : E \to R^+$

With the distance function **d(x,y)** defined as the shortest route (according to edges weights) between x and y.

3.2 The K-Servers Problem

K servers are positioned in (X, d), a metric space. A series of service requests, $\sigma = x_1, x_2, x_3, ...$ each indicates a location, P_i, needed to be serviced. At each request x_i a server is to be moved to P_i, at the price of the distance between its previous location, a_i, and P_i. The total cost is $\sum_{i=1}^{t} d(a_i, P_i)$ where t is the length of σ.

The decision the algorithm has to make at each step is which server to send to fulfill the service request. With $K = 1$, all algorithm acts the same and we have no interesting comparisons to make. Note that Paging is also a K-servers problem in a uniform metric space where each page is a point in the space and each memory slot is a server. We can

come to a conclusion (based on what we know about paging) that if we are looking for an algorithm that works for **Any** metric space, it is at least bad as K-competitive. (for deterministic algorithms).

Are there metric spaces where we can get a better competitive ratio?

3.3 Lower Bounds

Theorem 3.1. *Any algorithm for K-servers problem in a given metric space with $n \geq K+1$ points is at least bad as K-competitive.*

Proof. WLOG $n = k + 1$ (same explanation as before). Given an algorithm, A, we choose σ so it will always request a service in an un-covered point.

We claim that:

$A(\sigma) = \sum_j d(P_{j+1}, P_j)$

This since we choose a sequence that the next service point is where the server which serves the current point was. We'll show K algorithms $A_i, i = 1, 2, 3, 4, ..., K$ such that their total cost equals approximately $A(\sigma)$.

- At the beginning the servers are located at the points $1, 2, 3, ..., K$ and there is a "Hole" in $K + 1$.
- The first request would be "$K + 1$". Each A_i will send a server from i to $K + 1$.
- After the first request, they all service the sequence so that their servers positioning remains unique (each has a "Hole" in a different place. Note that only and exactly one server has a hole in the required point).

To illustrate the algorithm, assume that the second request was for point 1, only A_1 will move a server (because only A_1 has a hole in 1), and it will move server k+1 because it is the last server served and every other algorithm has a server there. So to remain unique, A_1 must move his server from point $K + 1$ to point 1. And so on...

Note that given a series σ, the above definition uniquely defines the algorithms.

We suggest that at each step, only one A_i moves a server and it moves the server exactly from P_{j-1} to P_j (as was explained above):

$$\sum_i A_i(\sigma) = \underbrace{\sum_{j=1} d(P_{j-1}, P_j)}_{A(\sigma)=\text{Rest of Steps}} + \underbrace{\sum_{i=1}^{K} d(i, K+1)}_{B=\text{First Step}} = A(\sigma) + B$$

since

$$A(\sigma) = \sum d(P_j, P_{j+1}) \underbrace{=}_{\text{Symmetry}} \sum d(P_{j+1}, P_j)$$

Therefore there is i such that:

$$A_i(\sigma) \leq \frac{1}{K} A(\sigma) + \frac{B}{K}$$

or

$$A(\sigma) \geq K A_i(\sigma) - B.$$

This implies that for $A(\sigma)$ which is large enough

$$A(\sigma) \geq (K - \epsilon) A_i(\sigma)$$

since otherwise

$$K A_i(\sigma) - B \leq A(\sigma) \leq (K - \epsilon) A_i(\sigma)$$

which implies that the cost $A_i(\sigma)$ and $A(\sigma)$ are bounded while we know that $A(\sigma) \to \infty$.

Hence, for large enough $A(\sigma)$

$$A(\sigma) \geq (K - \epsilon) A_i(\sigma) \geq (K - \epsilon) OPT(\sigma)$$

Finally, since the inequality holds for any $\epsilon > 0$ it implies that there is no better than K competitive algorithm.

Hence there is a K-Competitive lower bound. □

3.4 An Example: The Real Line

We'll show a K-competitive algorithm for K-server on the real line. Let's try some guesses for an algorithm for K=2:

1. Greedy - Send closest server to service the request.
2. Divide the space to areas of responsibility.

   Both aren't 2-competitive because we can give them both a series of requests for service in 2 constant points in space which of course OPT will service with the initial price of sending server1 to the first point and server2 to the second. But both our suggested algorithms has infinite cost if these two constant points are chosen to only be serviced by one server. This server will go back and forth between two points and pay infinite price.
3. Round Robin (the servers take turns)

   We can make this one fail with a series of requests as follows: two requests for the same location in an extremely far point x_1 and two requests for another far point (far from x_1), x_2. These four requests repeat themselves infinitely. OPT will keep one server at x_1 and one at x_2. Alg3 will have both servers go back and forth from x_1 to x_2 and hence has infinite cost.

The following algorithm works: Double Cover (DC).

3.5 Double Cover

DC is defined as follows: for a request for a service in some point x, move the closest server from each side of x towards x, both moves the same distance until one reaches x (meaning one server will probably not reach x, and the payment will be twice the distance travelled).

if x is positioned left or right from all servers only the closest server moves to x.

- A property of DC is that it is Memoryless (or History Independent).
- A Lazy algorithm is one that only moves the server that services the request.

We can make DC lazy without increasing its cost if instead of moving two servers at each request we will just move one and remember the "Virtual" location of the other server. All decisions will be done according to the virtual locations. We actually move a server only when according to the virtual locations it is the server that is closest to the service point.

The above actually implies that we can convert any online algorithm to this problem to a lazy algorithm without increasing its cost. We actually reduce the cost because we can prevent redundant movement.

We will prove DC is k competitive in its non-lazy form, which also proves it for the lazy version.

Theorem 3.2. *DC is K-competitive.*

Proof. Note that the order of the servers along the real line during the algorithm run is kept. (because if a server movement overlaps another server's location, we simple split the movement into two, the first server gets to the location of the second and the second continues from there to the target location)

We denote the server locations for DC algorithm as:

$$x_1 \leq x_2 \leq x_3 \leq ... \leq x_k$$

and the Opt algorithm's as:

$$y_1 \leq y_2 \leq y_3 \leq ... \leq y_k$$

We define a potential function, $\Phi = \underbrace{K \cdot \sum_i |x_i - y_i|}_{\Phi_1} + \underbrace{\sum_{i<j} |x_i - x_j|}_{\Phi_2}$

We'll show that $DC(\sigma) + \Phi \leq K \cdot OPT(\sigma)$

Actually We'll show that $\Delta DC(\sigma) + \Delta\Phi \leq K \cdot \Delta OPT(\sigma)$ for each step in the algorithms parallel run.

In the beginning of the process we assume the locations of the servers in both algorithms is the same. To avoid an additive constant we'll also assume all servers of both algorithm start at the same position. this assumption makes Φ equal zero along with DC and OPT.

Lets also assume OPT makes his move first and then DC makes his move:

First, OPT moves one server and pays Δy.

$\Delta OPT(\sigma) = \Delta y$
$\Delta DC(\sigma) = 0$
$\Delta\Phi_1 \leq K \cdot \Delta y$
$\Delta\Phi_2 = 0$ because DC didn't change yet.
$\Delta\Phi = \Delta\Phi_1 + \Delta\Phi_2 \leq K \cdot \Delta y$

Then we have two possibilities for DC:

1. Only one DC server moves Δx:

   $\Delta DC = \Delta x$

   $\Delta OPT = 0$

   since OPT must have a server in the requested point, the matching y_i of the moving x server must be in the direction in which x is moving and so: $\Delta\Phi_1 = -K\Delta x$

   and since $|x_i - x_j|$ is only changed for pairs containing the moving servers (and there are $k-1$ of them), and the moving server is getting away from them by Δx:

   $\Delta\Phi_2 = (k-1)\Delta x$

   $\Delta\Phi = -\Delta x$

   $\Delta DC + \Delta\Phi = \Delta x - \Delta x = 0 \leq K \cdot OPT(= 0)$

2. Two DC servers move, each moves Δx:

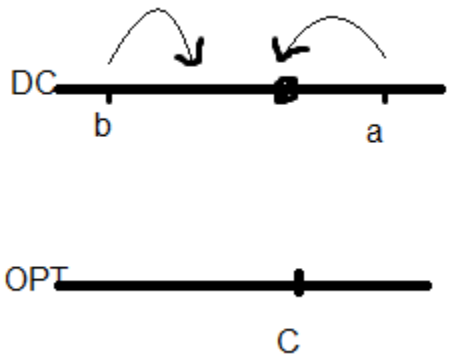

Figure 1:

   $\Delta DC = 2\Delta x$

   $\Delta OPT = 0$

   For pairs containing only one of the moving servers the two moves cancels themselves because one of the servers got closer by Δx and the other got away by Δx. For the

moving pair the distance between them got smaller by $2\Delta x$. $\Delta\Phi_2 = -2\Delta x$

If we prove that $\Delta\Phi_1 \leq 0$ then we get:

$\Delta DC + \Delta\Phi \leq 0 \leq K \cdot \Delta OPT$ and we're done.

In Figure 1, the points a and b are the origins of the two servers of DC. The point c is the point requiring service.

One possibility is that a is matched to an OPT server which is on c or left from there. In that case a-server got closer by Δx. The b-server in the worst case got further by Δx and we get that total change is ≤ 0

If the above does not hold then b is matched to an OPT server which is on c or right from there. (These two options covers all cases since in DC there are no server between a and b). In this case the y matching server for a-server is on the right. This case it is exactly the same. Server b got closer by Δx and a might have at worst got further by Δx. And we get again a change which is ≤ 0.

□