Advanced Algorithms 11/14, 2017

Lecture 9

Lecturer: Mohsen Ghaffari *Scribe: Roko Zaja*

This draft has not been checked by the lecturer yet.

1 Online Algorithms and Competitive Analysis

Competitive analysis is a method invented for analyzing online algorithms, in which the performance of an online algorithm (which must satisfy an unpredictable sequence of requests, completing each request without being able to see the future) is compared to the performance of an optimal offline algorithm that can view the sequence of requests in advance. An algorithm is competitive if its competitive ratio (the ratio between its performance and the offline algorithm's performance) is bounded.

1.1 Ski Rental Problem

The ski rental problem is a name given to a class of problems in which there is a choice between continuing to pay a repeating cost or paying a one-time cost which eliminates or reduces the repeating cost.

Assume that renting skis costs 1 dollar per day and buying skis costs B dollars. There is a number of days d (unknown to you) that you will ski. We are looking for an algorithm that minimizes the ratio between what you pay using the algorithm (that does not know d) and what you would pay optimally if you knew d in advance.

Algorithm: *Rent for the first $B-1$ days of skiing. On day B, buy skis.*

We claim that our algorithm is $(2-1/B)$ competitive. Let σ be any request sequence specifying n skiing days. Then our algorithm has cost n if $n \leq B-1$ and cost $2B-1$ if $n \geq B$. Since the optimum offline cost is given by $minn, B$, it follows that our algorithm is $(2-1/B)$ competitive.

We can't do any better. Any competitive algorithm (ALG) must buy skis at some point in time, say, day j.The adversary simply presents skiing requests until the algorithm buys and then ends the sequence. Thus, the online cost is $j-1+B$, whereas the optimal offline cost is $minj, B$.

If $j \leq B$, then

$$\frac{cost_{ALG}}{cost_{OPT}} = \frac{j-1+B}{minj, B} = \frac{j-1+B}{j} \geq 2 - \frac{1}{B}.$$

If $j \geq B$, then

$$\frac{cost_{ALG}}{cost_{OPT}} = \frac{j-1+B}{minj, B} = \frac{j-1+B}{B} \geq 2 - \frac{1}{B}.$$

1.2 Linear Search

Our problem will be the *list accessing problem.* In this problem, we have a finite pile/list of papers on the desk. Requests for elements in the list arrive online. At each moment we choose some of the following steps:

- linear/sequential search to depth i that costs i
- free swap - move item i arbitrarily upwards at cost 0
- paid swap - pick consecutive items and swap them at cost 1

Move-To-Front (MTF): *Upon finding the requested paper, move it to the foremost of the pile.*

Let m denote the number of papers in the list. We define potential function Φ for the analysis of MTF.

$$\Phi = \text{number of pairs (i,j) ordered differently between } ALG \text{ and } OPT.$$

Let c_i be the cost incurred by ALG on request r_i. The *amortized cost* a_i for serving request r_i is defined as

$$a_i = c_i + \Phi_i - \Phi_{i-1}.$$

The intuition behind a potential function and the amortized cost is to measure "how good" the current configuration of the online algorithm is compared to an optimal offline algorithm.

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} a_i + \Phi_0 - \Phi_n \leq \sum_{i=1}^{n} a_i + \Phi_0.$$

If we can show that $a_i \leq c \cdot OPT(r_i)$, then it follows that the online algorithm ALG is c-competitive. Let x be the element requested in r_i, which is at position k in MTF's list and at position l in the list organized by OPT. We denote the number of free and paid exchanges used by OPT on this request by f and p, rspectively. Let v be the number of elements that are in front of x in MTF's list but behind x in OPT's list before the request. Bt moving x to the front of the list, v inversions are removed and at most $j-1$ new inversions are created. Thus, the increase in potential due to actions of MTF is at most $-v+j-1$. By reorganizing the list, OPT can increase the potential by at most $p-f$. Thus, the amortized cost satisfies:

$$a_i = c_i + \Phi_i - \Phi_{i-1} = k + \Phi_i - \Phi_{i-1} \leq k - v + j - 1 + p - f \leq j + p + k - v - 1.$$

Recall that v is the number of elements in MTF's list that were in front of x but came after x in OPT's list. Thus, $k-1-v$ elements are before x in both lists. Since OPT contained x at position j, we conclude that $k-v-1 \leq j-1$. We get

$$a_i \leq j + p + (j-1) \leq 2(j+p) - 1 = 2OPT(r_i) - 1.$$

Summing over all requests $r_1, ..., r_n$ results in $\sum_{i=1}^{n} a_i \leq 2OPT(\sigma) - n$. Since $OPT(\sigma) \leq nm$ we can conclude that

$$\sum_{i=1}^{n} a_i \leq (2 - \frac{1}{m})OPT(\sigma).$$

We finally use that MTF and OPT start with identical list configurations such that $\Phi_0 = 0$. We get the following theorem:

Theorem 1. *MTF achieves a competitive ratio of* $(2 - 1/m)$ *for the list accessing problem.*

1.3 Paging/Caching

We have a space of N pages, a cache and a hard drive. Cache can hold k pages. Accessing a page that is in the cache costs 0 units. Accessing a page that is on the hard drive costs 1 unit(and we can move it to the cache).
Sleator and Tarjan give tight bounds on the best competitive ratio which can be achieved by any deterministic online paging algorithm.They show that two commonly used paging algorithms achieve a competitive ratio of k. These algorithms are FIFO which on a fault evicts the page that was placed in the fast memory least recently and LRU which on a fault evicts the page that was used least recently.

Definition 2 (Conservative strategy)**.** *A strategy is conservative, if on any consecutive request sequence containing k or fewer distinct page references, the algorithm will incur k or fewer units of cost.*

Theorem 3. *If A is any deterministic online paging algorithm, then its competitive ratio is at least k.*

Proof. We assume that A and OPT both start with the same set of pages in the cache. The adversary restricts its request sequence to a set of $k+1$ pages: the k pages initially residing in the cache and one other page. The adversary always requests the page that is outside of A's cache. This process can be continued for an arbitrary number of requests, resulting in an arbitrarily long sequence σ on which A faults on every request.

We must now show that cost of OPT is $\lceil \sigma/k \rceil$. At each fault, the adversary adopts the following strategy: evict the page whose first request occurs farthest in the future. Suppose a page x is evicted by OPT. The next fault occurs the next time x is requested. The adversary is guaranteed that all the other pages in the adversary's fast memory will be requested before x is requested again. There will be at least $k-1$ pages requested between any two faults, so the adversary faults at most on every kth request. $\square$

Definition 4. *The oblivious adversary(weak adversary) knows the algorithm's code, but does not get to know the randomized results of the algorithm.*

1-Bit LRU (RMA): *At first all pages in cache are unmarked.*

- *when accessing a page in cache, mark it*
- *when accessing a page not in cache, bring it to the cache and mark it, evict an arbitrary unmarked page, if there is one*
- *otherwise evict an arbitrary page, replace it with the new page and unmark all the pages*

Theorem 5. *The RMA has competitive ratio $O(log(k))$ against any oblivious adversary.*

Proof. Assume OPT and RMA start with the same cache contents. As before we divide the sequence σ into phases. The ith phase ends immediately before the $k+1$st distinct page is requested in the phase. We will analyze the cost of both OPT and RMA phase by phase. Note that once a page is marked, it is not evicted from the cache for the remainder of the phase. Therefore, if we denote the set of pages requested in phase i by P_i, then at the end of a phase i, the contents of RMA's cache is exactly P_i. Furthermore, RMA will not fault twice on the same page within a phase. Thus, we need only account for faults incurred on the first request to any given page in a phase.
Let m_i be the number of new requests in phases i. A page requested in the first phase is also new if it is not one of the pages initially in the cache. Since any new page is not in RMA's cache at the beginning of a phase, RMA must fault once on every new page requested. Now we must analyze the expected number of faults on requests to old pages. What is the probability that RMA faults on the jth old page requested? Let's suppose that just before the jth old page is requested, there have been l new pages requested so far in the phase. It is easy to show by induction that at this time there are exactly l pages in P_{i-1}. Since the adversary has fixed the request in advance, the probability that it is not in the cache is exactly $l/k-(j+l-1)$. Since l is always at most m_i, the probability of a fault on the request to the jth old page is at most

$$\frac{m_i}{k-(j+m_i-1)}.$$

Therefore, the expected cost of the marking algorithm in the ith phase is

$$E[cost_{RMA}(\sigma)] \le m_i + \sum_{j=1}^{k-m} \frac{m_i}{k-(j+m_i-1)} \le m_i H_k.$$

Summing up over all phases, we get that the expected cost for the algorithm over the entire sequence is

$$E[cost_{RMA}(\sigma)] \le H_k \sum_i m_i.$$

Now we must prove a lower bound for the optimal cost. We claim that

$$cost_{OPT}(\sigma) \ge \sum_i \frac{m_i}{2}.$$

Consider the $(i-1)$st and ith phases. The number of distinct pages requested in both phases is $k+m_i$. Since OPT has only k pages in the cache at the beginning of the $(i-1)$st phase, it must incur at least m_i faults during the two phases. Applying this argument to every pair of adjacent phases, we have that $cost_{OPT}(\sigma) \geq \sum_i m_{2i}$ and $cost_{OPT}(\sigma) \geq \sum_i m_{2i+1}$. Therefore OPT has cost at least the average of these, i.e. $cost_{OPT}(\sigma) \geq \sum_i \frac{m_i}{2}$. Thus, $E[cost_{RMA}(\sigma)] \leq 2H_k cost_{OPT}(\sigma)$. □

Theorem 6. *Let R be any randomized paging algorithm . If the number of pages is greater than or equal to $k+1$ where k is the size of the cache, the competitive ratio of R against any oblivious adversary is $\Omega(log(k))$.*

Proof. We will find our lower bound on c_R by exhibiting a probability distribution P for which $c_A^P \geq H_k$ for all deterministic algorithms A. Let S be a set of $k+1$ pages which include the k pages initially in the cache. Take P to be the uniform distribution on the $k+1$ pages in S. That is, a sequence σ of m requests is generated by independently selecting each request at random from S. Clearly, the expected performance of any deterministic algorithm on inputs generated from the distribution is

$$E_P[cost_A(\sigma)] = \frac{m}{k+1}$$

for $\sigma = m$. To see this, note that the probability that a given requested page is not in the cache is $\frac{1}{k+1}$. we now need an upper bound on the expected performance of OPT.

Once Again, we divide the sequence of page requests into non-overlapping phases such that each phase contains maximal runs of request to at most k distinct pages. As we have seen, if there are r distinct phases then the optimal algorithm can service the sequence with at most $r+1$ faults. At the beginning of a phase, OPT replaces the one page currently in the cache that will not be requested in the phase. Therefore, if $N(m)$ is the random variable which is the number of phases in a sequence σ of length m (generated from P), then the expected offline cost satisfies

$$E_P[cost_{OPT}] \leq E[N(m)+1].$$

Since the durations of successive phases, are independent, identically distributed random variables, we have by the elementary renewal theorem that

$$\lim_{m\to\infty} \frac{m}{E[N(m)]} = E[X_i],$$

where $E[X_i]$ is the expected length of the ith phase.

The expected length of a phase, $E[X_i]$ is easily seen to be $(k+1)H_k$ ("coupon collectors problem"). Therefore, we have that

$$\lim_{m\to\infty} \frac{E_P[cost_A]}{E_P[cost_{OPT}] \geq \frac{m/k+1}{m/E[X_i]}} \geq \frac{E[X_i]}{k+1} = \frac{(k+1)\cdot H_k}{k+1} = H_k.$$

□

1.4 Lost Cow Problem

A cow comes to an infinitely long straight fence. The cow knows that there is a gate in the fence, and she wants to get to the other side. Unfortunately, she doesn't know where the gate is located. Assume that the gate is positioned an integer number of steps away from the cow and that the cow can only recognize the gate when directly upon it. How can she optimally find the gate? We suppose the cow knows that there is a gate somewhere but she does not know how far. What is the minimum number of steps she must make to find the gate as a function of the actual (unknown) distance to the gate?

Theorem 7. *The doubling strategy is 9-competitive for the lost cow problem.*

References