Online Algorithms 30/11/2011

Lecture Notes 5: Random On-Line Algorithms

Professor: Yossi Azar *Scribe:Elizabeth Firman*

1 Introduction

In the previous lesson we saw that the ski - rental problem is $\frac{e}{e-1}$ - competitive. We saw the survival game and showed that it is $H_n \approx logn$ - competitive. Then we used it to show that the paging algorithm - RMA, is $2H_k \approx logk$ - competitive. The question is, can we get a better competitive ratio? We'll see that H_k is the lower bound to the paging problem. But first we'll intreduce a mathematical problem that will help us with the analysis.

2 Coupon Collector

In the coupon collector problem we have n slots, and we randomly throw balls into them (to a random slot). We throw independently and with uniform distribution. How long does it take for all the slots to contain at least one ball? (time unit = a single throw)

Answer: $nH_n = nlogn$

Proof. Let X_i be a random variable. X_i = the time that passed between the i full slots and i+1 full slots.
Note that $X_i \sim\ G(\frac{n-i}{n})$, hence $E(x_i) = \frac{n}{n-i}$.
$E(\text{time in total}) = E(\sum_{i=0}^{n-1} \frac{n}{n-i}) = n(\frac{1}{n} + \frac{1}{n-1} + ... + 1) = nH_n$

2.1 Theorem

Let R be a random paging algorithm, then R is at least H_k competitive (for k slots in the memory and number of pages $> k$).
Proof. First, lets reduce the number of pages to $k+1$. Now, for an algorithm R, we will construct a random sequence σ as follows: On every step, choose randomly, independently and with uniform distribution, a page between 1 and $k+1$. On every step there will be a page fault with propability $\frac{1}{k+1}$ (because there are $k+1$ pages, and only one page is in the memory). Hence, for a sequence of length m, we'll get: $E(R(\sigma)) = \frac{m}{k+1}$, when the expectation is both on R and on σ.
To calculate the expectation of page faults for OPT, we will divide it into phases. In each phase we will get one page fault, because the difference between two phases is a single page. Now, the question is, what is the length of the phase?
Note that this is exactly the coupon collector problem.
$E(\text{length of a phase}) = \underbrace{(k+1)H_{k+1}}_{\text{a}}\underbrace{-1}_{\text{b}} = (k+1)(1 + \frac{1}{2} + ... + \frac{1}{k+1}) - 1 = (k+1)H_k$

a The coupon collector problem for $k+1$ pages.

b The phase ends before the last page fault.

$E(\text{number of page faults}) = \frac{m}{(k+1)H_k}$
$(\text{number of faults} = \frac{m}{\text{length of phase}})$, hence we get a lower bound of H_k.
Note that the last equation is true because of the renewal theory, and generally $E(\frac{1}{x}) \neq \frac{1}{E(x)}$
Note: RMA is as bad as $2H_k - 1$.

3 Randomized Algorithms

We can describe an on-line problem as a game. For a sequence of n requests, we'll describe the game as a tree (see figure 1). The depth of a tree is $2n$, the odd layers are the requests and the even layers are the responses of the algorithm. Each leaf contains the the total payment of the on-line and the OPT algorithms, meaning that we have the competitive ratio for every sequence of length n.

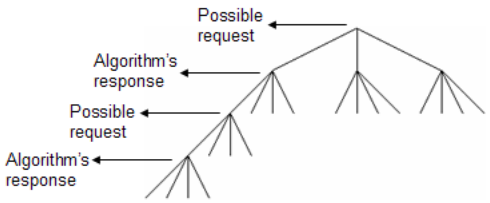

Figure 1: The tree describing the game

The game is between the algorithm and an opponent. The opponent gives a sequence of requests and serves them. The sequence as well as the response, may depend on the algorithm's response. The goal of the opponent is to get the maximal competitive ratio, and of course the goal of the algorithm is to get the minimal competitive ratio.

We have two types of opponents:
Oblivious opponent: Knows the algorithm, but cannot see the coin tosses.
Adaptive opponent: Knows the algorithm and can see the coin tosses (until the current point in the game).

If the adaptive opponent could see the future coin tosses, then this would make the algorithm deterministic. For a deterministic algorithm there is no difference between the two opponents, because in this case the opponents can check all the options and choose a sequence that will give the maximal competitive ratio.
The adaptive opponent cannot find the worst possible sequence, because it depends on the coin tosses, making it much harder to find the competitive ratio.
Note: This is a zero-sum game (what one loses the other gains)

Let Q denote the opponent and A denote the algorithm.
$A(Q)$ - The cost of algorithm A when playing versus adaptive opponent Q.
$Q(A)$ - The cost of adaptive opponent Q when playing versus algorithm A.
Note: The above values are the expectation of the cost on the coin tosses of the algorithm.

3.1 Definition

Algorithm A is called c - competitive if $A(Q) \leq cQ(A)$, for every adaptive opponent Q.

3.2 The adaptive opponent:

The adaptive opponent is divided into two types:
Adaptive offline - serves all the requests at the end of the sequence (a strong opponent).
Adaptive online - serves the requests as they come, like the algorithm (weak opponent).

$Q^{obliv.} \propto Q^{adapt.online} \propto Q^{adapt.offline}$

$C^{obliv.}_{rand.} \leq C^{adpt.online}_{rand.} \leq C^{adapt.offline}_{rand} \leq C^{adapt./obliv.}_{det.}$

when C is the best competitive ratio.

3.3 Theorem 1

If R is a randomized algorithm which is α - competitive versus an adaptive - offline opponent, then there is a deterministic algorithm A which is α - competitive.

3.4 Theorem 2

If G is a randomized algorithm which is α - competitive versus an adaptive - online opponent, and there exists a randomized algorithm H which is β - competitive versus an oblivious opponent, then G is a randomized algorithm which is $\alpha\beta$ - competitive versus an adaptive - offline opponent.

3.5 Conclusion

Using the first theorem with the second one, we get the existence of a deterministic algorithm $\alpha\beta$ - competitive.
Also note that $\beta \leq \alpha$, therefore there is a deterministic algorithm α^2 - competitive.

Example: Paging

4 Scheduling

4.1 The Problem

Assume we have m identical machines. Tasks are coming on-line. We assign each task a machine, that stays there (doesn't move).
Let w_i denote the weight of task i.
Let A denote an algorithm, and $A(i) = j$ denote that A assigns task i to machine j.
Let $l_j = \sum_{i|A(i)=j} w_i$, the total load on a machine j.
The maximal load is: $max_j\ l_j$
The goal is to minimize the maximal load.
At the first lecture we saw the greedy algorithm, which assigns to a task the machine with the minimal load. We showed that this algorithm is 2 - competitive.

4.2 Theorem

The greedy algorithm is $2 - \frac{1}{m}$ - competitive.

proof. Using the law of conservation of volume, we get : $OPT \geq h + \frac{w}{m}$.
For a single task we get $OPT \geq w$, hence:
$ON = h + w = h + \frac{w}{m} + (1 - \frac{1}{m})w \leq OPT + (1 - \frac{1}{m})OPT = (2 - \frac{1}{m})OPT$

Does this analysis gives us a tight bound? Yes. We'll show a sequence which achieves an equality: Assume we have $m(m-1)$ unit tasks, and one task of size m.
$G(\sigma) = 2m - 1$, $OPT(\sigma) = m$, hence we get the following competitive ratio: $\frac{2m-1}{m} = 2 - \frac{1}{m}$

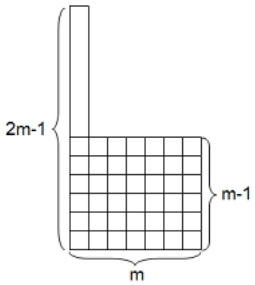

Figure 2: The greedy algorithm

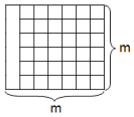

Figure 3: The optimal algorithm

Is there an algorithm with a better competitive ratio?

- *m = 2*
  The greedy algorithm is $\frac{3}{2}$ - competitive.
  Lets take the sequence: (1, 1, 2)
  There are two possible ways for any algorithm to assign the incoming sequence of tasks to the two machines:

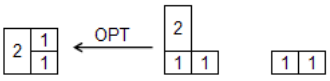

Figure 4: The competitive ratio is at least $\frac{3}{2}$

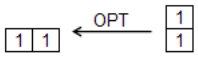

Figure 5: The competitive ratio is at least 2

  We get a deterministic lower bound of $\frac{3}{2}$.

- *m = 3*
  The greedy algorithm is $\frac{5}{3}$ - competitive.
  Lets take the sequence: (1, 1, 1, 3, 3, 3, 6) and any on-line algorithm A.
  We can divide to three groups, all the ways that any algorithm can assign the incoming

sequence of tasks to the three given machines:

Figure 6: A puts two unit tasks one on top of the other, then the competitive ratio is at least 2.

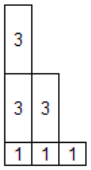

Figure 7: A puts two tasks of size three on the same machine: $A = 7$ and $OPT = 4$, then the competitive ratio is $\frac{7}{4}$ (greater than the greedy algorithm).

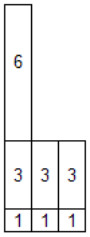

Figure 8: $A = 10$ and the competitive ratio is $\frac{10}{6} = \frac{5}{3}$ (as the greedy algorithm).

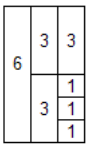

Figure 9: $OPT = 6$

- $m = 4$
  The greedy algorithm is $\frac{7}{4} = 1.75$ - competitive.

Claim 1: There is an algorithm, which is $2 - \frac{1}{m} - \frac{1}{m+1}$ - competitive, for $m \geq 10$.
Claim 2: There is an algorithm, which is $2 - \frac{1}{70} = 1.986$ - competitive, for $m \geq 70$.
The greedy algorithm "over - balances" the load on the machines, and that's why a single task can ruin everything. If suddenly comes a "heavy" task, it ruins the balance. The new algorithm tries too keep the machines unbalanced so the competitive ratio is not greater then 1.986. It sorts the machines from the minimal load to the maximal load, and puts the task on the machine with the largest load it can, so the competitive ratio stays under 1.986.

Question: what if we allow additive constant