CSL 758: Advanced Algorithms

Online Paging Problem (Continued)

Professor : Naveen Garg
: Kavitha Telikepalli

Scribe By : Amit Ruhela March 13, 2008

Introduction:

In the previous lecture, we defined Online Paging problem and the term competitive ratio for algorithms. We also studied Marker's algorithm for Online Paging problem.

Background:

For analysis purpose, the Marker's algorithm is again repeated here.

Algorithm Marker

Initially all pages in the fast memory are unmarked.
Repeat

- On a page miss, evict a randomly chosen unmarked page and mark the incoming page.
- On a page hit, mark the page requested if it is unmarked.

Until all pages are marked
Start the next round.

Competitive Ratio

For a deterministic paging algorithm 'A', let $f_A(r_1, r_2,\ldots,r_N)$ denote the number of times that the algorithm A misses on the page request sequence.
Let $f_{OPT}(r_1, r_2,\ldots,r_N)$ denote the number of times that an optimal offline algorithm OPT misses on the page request sequence.

Algorithm "A" is defined as C-Competitive if for every request sequence $(r_1, r_2, r_3, \ldots\ldots r_N)$,there exist a constant 'b' such that .

$$f_A(r_1, r_2, r_3, \ldots\ldots r_N) \leq C \cdot f_{OPT}(r_1, r_2, r_3, \ldots\ldots r_N) + b$$

where b is a constant independent of N but may depend on K(Cache size).

Here

$$C \approx \text{Max}_{(r1, r2, r3, \ldots\ldots rN)} \frac{f_A(r1, r2, r3, \ldots\ldots rN)}{f_{OPT}(r1, r2, r3, \ldots\ldots rN)}$$

For Randomized Algorithm

For a randomized paging algorithm 'A', let $E(f_A(r1, r2, r3, \ldots\ldots rN))$ denote the number of times that the algorithm A misses on the page request sequence.

$$C \approx \text{Max}_{(r1, r2, r3, \ldots\ldots rN)} \frac{E(f_A(r1, r2, r3, \ldots\ldots rN))}{f_{OPT}(r1, r2, r3, \ldots\ldots rN)}$$

Theorem 1 :
Algorithm Marker is (2Hk)-competitive against oblivious adversaries, where k is the size of the fast memory (Cache).

The marker algorithm proceeds in a series of **rounds**. At the beginning of each round, all the marker bits are reset. Suppose a round begins with r_i and ends with r_j, that is, r_j is the start of the next round. During the round, every location in the cache was marked. Each request for a new page causes it to be in the cache and for its marker bit to be set, whether it started in cache or not. So all k marker bits are set after there are k distinct page requests in $r_i, \ldots, r_{j-1}$, and r_j must be a request for a $(k+1)^{st}$ page.

Consider the requests in any round. Call an item **Old** if it is unmarked, but was marked in the last round. Old items were requested in the last round, but not yet in this round. Call an item **new** if it is neither old nor marked. A New item was not requested in the last round.

Let $\mathbf{l_r}$ be the number of requests to new items in a round.

Let S_O denote the set of items in the cache of the offline algorithm and S_M denote the set of items in the Marker algorithm's cache. Define

$d_{init,r}$ = $|S_O - S_M|$ at the beginning of the round r,

$d_{final,r}$= $|S_O - S_M|$ at the end of the round r.

Let M_O be the number of misses of the offline algorithm during the round. As we noted above, new items are not in the Marker algorithm's cache at the beginning of a round. The contents of the offline algorithm's cache differ from the Marker algorithm's in $d_{init,r}$ items at the start of the round. So at least $l_r - d_{init,r}$ of the new items will not be in the offline algorithm's cache either. Each of these items will be requested during the round and will generate a cache miss for the offline algorithm. So the offline algorithm has at least $l_r - d_{init,r}$ cache misses.

Another type of cache miss for the offline algorithm occurs because it looks ahead. The contents of the Marker cache S_M at the end of the round are all and only the k pages requested during the round. But the offline algorithm's cache contents S_O can be different, which means it must have thrown out some of those pages. The variable $d_{final,r}$ counts pages that were requested in this round and which have already been thrown out by the offline algorithm. Each of those must have been a cache miss (in order to cause a page to be thrown out). So the number of misses of the offline algorithm is at least $d_{final,r}$. Putting both arguments together, we see that the number of misses for the offline algorithm is at least:

F_{opt}(round r) = max(l_r - $d_{init,r}$, $d_{final,r}$)

The max function is a little tricky to use in recurrences, so instead we notice that the max of two numbers is at least their average. Therefore the number of cache misses is at least:

F_{opt}(round r) = max(l_r - $d_{init,r}$, $d_{final,r}$) ≥ $\frac{(l_r - d_{init,r} + d_{final,r})}{2}$

This is a bound for the number of misses in a round. We would like to get the average number of misses for many rounds. If we add up this sum for many rounds, we notice that the $d_{final,r}$ for one round is equal to the $d_{init,r}$ for the next. So all the$d_{init,r}$'s and$d_{final,r}$'s cancel except the first and the last, and their contribution is negligible if we sum over enough rounds.

i.e F_{opt}(All Rounds) =$\frac{\sum_r l_r}{2}$ + $\frac{d_{final} - d_{init}}{2}$

$\frac{d_{final} - d_{init}}{2}$ is a small no and we can ignore it.

It follows that the number of cache misses for the optimal offline algorithm ≥ $\frac{\sum_r l_r}{2}$

Upper Bound of Marker algorithm

Every request to a new page causes a cache miss. There are l_r of these. There is another type of cache miss which is a request for an old page that has been evicted in the current round r_i. There are k – l_r requests for old pages in the round. Let's say these request are O_1 , O_2 , O_3 …… O_{k-lr} .The probability of old page requests causing a cache miss is maximized when all the requests for new pages (which cause certain cache misses) come before the requests for old pages (which may or may not). So we assume that happens.

There are k – l_r requests for old pages. Since the l_r new page requests go first, when the first old page O_1 request happens, l_r out of k of the pages have been evicted (at random), so the probability that the first old page request causes a miss is

$\frac{l_r}{\mathrm{k}}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N1	N2	N3	N4	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11

N is request for a new page and O for the Old page.

Let's assume that when first request for old page O_1 comes, it can be a miss or a hit. If it was a miss then this would have been evicted by some new request N in this round from set l_r. Assume that it was N2. Now on the request for second old page O2, this N2 can't cause eviction for O2 (N2 has evicted O1 and can't cause eviction for O2 simultaneously). Therefore no of new pages which can cause eviction for O2 are l_r -1. Further when request for O1 comes, it can also evict old page O2. Therefore total no of pages which can evict O2 are l_r-1 +1 = l_r.

If it was a hit when request for O1 comes, then O1 can't contribute for eviction of O2. Therefore the no of previous requests that can cause eviction for O2 are same as l_r .

Let's say that E_2 is the event when a miss occurs on the first request of O2 and P is the probability that M (Marker algorithm) doesn't Suffers a miss on the first request for O1.

Therefore P_r (E_2=Marker Suffers a miss on the first request for O_2) =

$P_r(E_2 \mid$ M Suffers a miss on the first request for O1) P_r(M Suffers a miss on the first request for O1) +

$P_r(E_2 \mid$ M doesn't Suffers a miss on the first request for O1) P_r(M doesn't Suffers a miss on the first request for O1)

= $\frac{l_r-1+1}{k-1}$ x P + $\frac{l_r}{k-1}$ x (1-P) = $\frac{l_r}{k-1}$

Similarly Probability that second old page causes a miss = $\frac{l_r}{k-1}$

Generalizing above, Probability that third old page causes a miss = $\frac{l_r}{k-2}$

Probability that i^{th} old page causes a miss = $\frac{l_r}{k-i+1}$

And for last request k-$\boldsymbol{l_r}$, in this round r_i , the probability of miss will be $\frac{l_r}{k-(k-l_r)+1}$ = $\frac{l_r}{l_r+1}$

Now $E(f_A(round\ r_i))$ = $l_r + \frac{l_r}{k} + \frac{l_r}{k-1} + \ldots\ldots\ldots + \frac{l_r}{k-i+1} + \ldots\ldots\ldots + \frac{l_r}{l_r+1}$

= $l_r(1 + \frac{1}{k} + \frac{1}{k-1} + \cdots\ldots\ldots\ldots + \frac{1}{k-i+1} + \cdots\ldots\ldots\ldots + \frac{1}{l_r+1})$

$Let\ H_k = \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots\ldots\ldots\ldots + \frac{1}{k}\right)$ be the k − th Harmonic number ($Hk \approx \ln k$).

$E(f_A(round\ r_i))$ = $l_r(1 + H_k - H_{l_r})$

Since H_{l_r} is at least 1, therefore $E(f_A(round\ r_i)) \leq l_r.H_k$

For all rounds, the expected value is

$E(f_A(all\ rounds)) \leq H_K \sum_1^n l_r$

As C ≈ Max $_{(r1, r2,r3,.....rN)}$ $\frac{E(\boldsymbol{f_a}\ (r1,r2,r3,.....rN)}{\boldsymbol{f_{OPT}}\ (r1,r2,r3,.....rN)}$

Therefore C = $\frac{H_K\ \sum_1^n l_r}{\sum \frac{lr}{2}}$ = 2. H_K

Which implies that Marker algorithm is 2H_k-competitive.

Claim: The competitive ratio of any randomized online paging algorithm is at least H_K where k is size of the fast memory(cache).

Lower Bound on Running Time of Randomized Algorithms

The complexity of a problem is specified by lower and upper bounds on the complexity of algorithms that solve the problem:

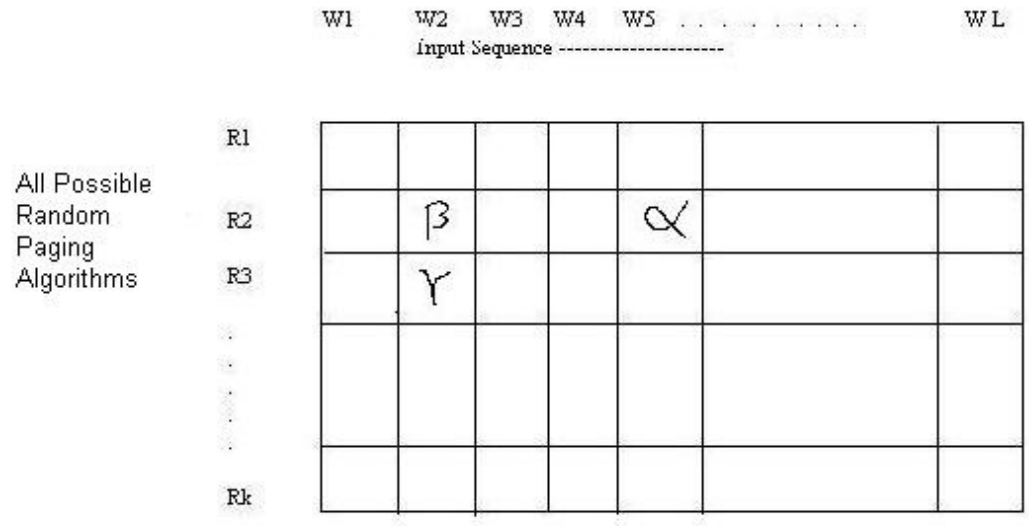
- Lower bounds are obtained by proving (e.g., using combinatorial arguments) that no algorithm can have a complexity smaller than the lower bound under consideration.
- Upper bounds are usually obtained by constructing an algorithm for the given problem that is correct and has complexity of at most the upper bound under consideration.

Las Vegas algorithm

In computing, a **Las Vegas algorithm** is a randomized algorithm that never gives incorrect results; that is, it always produces the correct result or it informs about the failure. A typical example of a Las Vegas algorithm is randomized quicksort (i.e., the pivot elements are chosen uniformly at random).

Matrix games

A matrix game can be represented by a matrix with R rows and W columns that has entries of the form $T(R_i,W_j)$. A matrix game (R; W; T) is played by two players. The first player selects a strategy R_i from a finite set R, the second player selects a input W_j from a finite set W, and the first and second player try to minimize and to maximize, respectively, the payoff $T(R_i,W_j)$. Here T_{ij} is the running time of algorithm A_i on on input W_j .



Game A
Let's say that in Matrix Game A,

- You choose a random algorithm(Min Player)
- I choose an input sequence(Max Player)

Your goal is to present a randomized algorithm that works well over all possible input Sequences and to minimize the payoff T. That is you are MIN player. My goal is to present such input on which your algorithm behaves badly and thus try to maximize the T. That is I am MAX Player.

Value of Game A is expected running time of your algorithm on my input (Lets say it is α). Clearly I will try to maximize T

Game A′
Let's say that in Matrix Game A′,

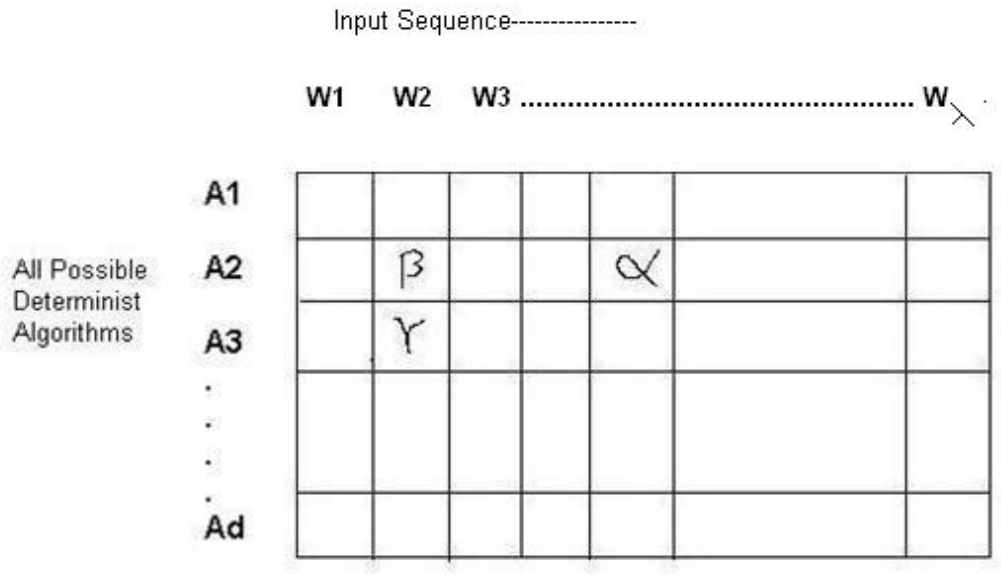
- I choose an input sequence(Max Player)
- You choose a random algorithm(Min Player)(In fact you will choose a deterministic Algorithm as you will try to minimize the payoff)

My goal is to choose an input which mostly behaves badly on all algorithms and thus try to maximize the T. That is I am MAX Player. Your goal is to present a randomized algorithm that works very well over my input and to minimize the payoff T. That is you are MIN player.

Value of Game A′ is expected running time of my input on your algorithm (Lets say it is β). Clearly you will try to minimize T.
It is very clear from the above games that Value of game A (α) ≥ Value of game A′ (β)

Mixed Strategy

A mixed strategy for a player in a matrix game is a probability distribution on the set of strategies for the player.



Game B

Let's say that in Matrix Game B,

- I choose a probability distribution q over columns or input sequence
- You choose a row or deterministic algorithm A_i

My goal is to choose a distribution over inputs with probability q so that any of your deterministic algorithms behaves badly on it. Your goal is to present a deterministic algorithm that works very well over my probabilistic input and to minimize the payoff T.

Value of Game B is averaged running time of your algorithm over my probabilistic inputs.
Value = $min_{A_i \in A} \sum_j q_j t_{ij}$

$S_j = \sum_i P_i \,.\, T_{ij}$ Represents the average value of a column J
$X_i = \sum_j Q_j \,.\, T_{ij}$ Represents the average value of a row i.

Now consider $\sum_j \sum_i P_i \,.\, Q_j T_{ij}$
$\sum_j \sum_i P_i \,.\, Q_j T_{ij}$ can also be written as $\sum_j Q_j \sum_i P_i \,.\, T_{ij}$ which Is averaged value of $S_j = \sum_i P_i \,.\, T_{ij}$ over Q.

Therefore $\sum_j Q_j \sum_i P_i \,.\, T_{ij}$ will always be less then $\max_j \sum_i P_i . T_{ij}$ **(1)**

Similarly

$\sum_j \sum_i P_i \,.\, Q_j \mathrm{T_{ij}}$ can also be written as $\sum_i P_i \sum_j Q_j \,.\, \mathrm{T_{ij}}$ which Is averaged value of $\mathrm{X_i} = \sum_j Q_j \,.\, \mathrm{T_{ij}}$ over P.

Therefore $\sum_i P_i \sum_j Q_j \,.\, \mathrm{T_{ij}}$ will always be greater then $\min_i \sum_j Q_j.\mathrm{T_{ij}}$ **(2)**

From (1) and (2) above we have

$$\min_i \sum_j Q_j.\mathrm{T_{ij}} \leq \sum_j \sum_i P_i \,.\, Q_j \mathrm{T_{ij}} \leq \max_j \sum_i P_i.\mathrm{T_{ij}} \quad \textbf{(3)}$$

Let's say that $\mathbb{P}$ is the set of all possible probability distributions over the rows and $\mathbb{Q}$ is the set of all possible probability distributions over the columns and no of columns is λ and no of rows is D.

Since the above statement is valid for any probability distribution $P = \{P1, P2 \dots Pd\} \in \mathbb{P}$ and $Q = \{Q1, Q2 \dots Q\lambda\}\} \in \mathbb{Q}$, Let's choose that P which will minimize $\max_j \sum_i P_i.\mathrm{T_{ij}}$ i.e. $\min_{P=\{P1,P2\dots Pd\} \in \mathbb{P}} \max_j \sum_i P_i.\mathrm{T_{ij}}$ which is minimum value of $\sum_i P_i.\mathrm{T_{ij}}$ for a given probability distribution $P = \{P1, P2 \dots Pd\} \in \mathbb{P}$.

Also let's choose that $Q = \{Q1, Q2 \dots Q\lambda\}\} \in \mathbb{Q}$ that will maximize $\min_i \sum_j Q_j.\mathrm{T_{ij}}$

i.e $\max_{Q=\{Q1,Q2\dots Q\lambda\}\} \in \mathbb{Q}} \min_i \sum_j Q_j.\mathrm{T_{ij}}$ which is maximum value of $\sum_j Q_j.\mathrm{T_{ij}}$ for a given probability distribution $Q = \{Q1, Q2 \dots Q\lambda\}\} \in \mathbb{Q}$.

Therefore equation 3 is also true for

$$\max_{Q=\{Q1,Q2\dots Q\lambda\}\} \in \mathbb{Q}} \min_i \sum_j Q_j.\mathrm{T_{ij}} \quad \min_{P=\{P1,P2\dots Pd\} \in \mathbb{P}} \max_j \sum_i P_i.\mathrm{T_{ij}} \leq \sum_j \sum_i P_i \,.\, Q_j \mathrm{T_{ij}}$$

Or

$$\max_{Q=\{Q1,Q2\dots Q\lambda\}\} \in \mathbb{Q}} \min_i \sum_j Q_j.\mathrm{T_{ij}} \leq \min_{P=\{P1,P2\dots Pd\} \in \mathbb{P}} \max_j \sum_i P_i.\mathrm{T_{ij}} \quad \textbf{(4)}$$

From (4) we can say that the expected running time of the optimal deterministic algorithm for an arbitrarily chosen input distribution is a lower bound on the expected running time of the every randomized Las Vegas algorithm against all inputs.

---x-x-x-x---