Randomization in Online algorithms

A Project Report

submitted by

Saran Neti Maruti Ramanarayana (100841098)

Advanced Data Structures, COMP 5408, Winter 2011

Carleton University

Abstract

Algorithms which should be able to produce an output without looking at the entire input sequence are called online algorithms. Using competitive analysis, their performance is compared to offline algorithms, and it has been discovered that randomized online algorithms appear to outperform deterministic ones. We look at randomized online algorithms for two problems, list update and paging, against three different adversary models with different powers - oblivious, adaptive online and adaptive offline. We then formally describe a framework called the request answer games and prove the limitations of randomization against adaptive adversaries and bound their cost with respect to deterministic algorithms against oblivious adversaries.

Contents

1 Introduction

Online algorithms, in some sense, are the best possible ways by which we are able to programatically predict the future inputs and compute our current answers in our attempt to optimize a final cost. This is in contrast to offline algorithms which get to see the whole input sequence to compute their optimal answer. Some problems are intrinsically offline, where as some others like paging, telephone call routing need online algorithmic solutions. Clearly an online algorithm can perform at most as good as optimal offline algorithm for the same input sequence, because at a certain point it might output an answer that may be optimum if the input stopped, but its contribution to the final cost may not be optimum considering future inputs. The following definitions are taken from [5].

1.1 Optimal algorithm

An optimization problem P takes a set $\{I\}$ of inputs, and a cost function C that associates each input I to a set of feasible outputs $F(I)$ at a cost. An optimal algorithm OPT is that which minimizes $C(I,O)$ for each $O \in F(I)$ for all legal inputs in $\{I\}$.

$$OPT(I) = \min_{O \in F(I)} C(I,O)$$

1.2 Adversary models

We give adversaries different powers in order to come up with the best "worst-case" sequence of inputs for the online algorithm.

In case of a deterministic online algorithm, the adversary knows the exact behavior of the algorithm and can run his copy of the algorithm to make sure the next input in the sequence will be disastrous and result in a high competitive ratio. This does not help us in analyzing the performance of one deterministic algorithm vs another. So, comparison is made to an optimal offline algorithm that can see the sequence beforehand, i.e the adversary is allowed to construct a sequence, and then we are allowed to construct the optimal offline algorithm OPT. Note that different input sequences will have different optimal offline algorithms.

A deterministic c-approximation algorithm ALG is that which is at most c times worse than OPT for all inputs. i.e if there exists an $\alpha \geq 0$ such that $ALG(\sigma) - c \,.\, OPT(\sigma) \leq \alpha$ (If P is a minimization problem. Similar inequality for a maximization problem). c is called the competitive ratio and is also

denoted as $R(ALG)$.

In the case of Randomized algorithms the optimal algorithm isn't so clear. In literature, there are three kinds of adversaries discussed.

1.2.1 Oblivious adversary

This is the same as the case for deterministic models, where the request sequence is first constructed and the randomized online algorithm is then chosen for that sequence, and its cost compared against the optimal offline cost.

In this case, the inequality for a c-competitive algorithm is $E[ALG(\sigma)] - c \,.\, OPT(\sigma) \leq \alpha$. We need to take the expectation of $ALG(\sigma)$ because the answer might change for each run of the algorithm for that σ, and hence $ALG(\sigma)$ is a random variable.

1.2.2 Adaptive online adversary

The next element of the request sequence σ is constructed based on full knowledge of the current internal state of *ALG*, including the values of the random variables. The cost of the output of *ALG* is compared against the lowest online cost the adversary has to pay. In this case c-competitive maybe defined as $E[ALG(\sigma) - c \,.\, ADON(\sigma)] \leq \alpha$.

1.2.3 Adaptive offline adversary

In this case too, the adversary can choose the next element of σ based on complete knowledge of the internal state of *ALG*, but the his output will be the one that is offline optimizing, i.e better than the adaptive online case. This is impractical because there is no clear way for the adversary to decide what the offline optimal output is when even he doesn't know the full input - because he'll change the input depending on *ALG*. But after the entire input sequence is decided and *ALG* terminates, the adversary computes the offline optimal and cost comparisons are made against that.

2 The List Access Problem

The competitive analysis of the static list accessing problem was first investigated by Sleator and Tarjan [8]. Given a list, it takes $O(i)$ time to access the element at position i. The problem is to modify the order of elements during an access,

at a cost in a given cost model, so that future access times are lower. The cost model includes two kinds of transposition : paid and free. In free transpositions, the accessed element at position i can be placed any location before i at no cost. In paid transpositions elements at different positions can be exchanged at a cost of 1. Before we proceed to look at online algorithms, it might be worth it to look at some results for offline algorithms.

2.1 Offline algorithms

Given an input sequence σ and a list of length l containing all the elements from σ, an optimum offline algorithm devises a strategy combining paid and free transpositions that minimizes the total access cost. The original Sleator-Tarjan paper claimed that only free transpositions will suffice for the optimum algorithm, but a counter example is provided by Reingold and Westbrook in [6]. The simplest most laborious way is to enumerate all possible strategies, i.e after every access, enumerate the list in $l!$ different configurations, while calculating the access time, and proceed. Finally take the minimum. The time required for this is exponentially large $O\left((l!)^{|\sigma|}\right)$. By restricting these sets using Subset transfers [6] Reingold and Westbrook give an optimum offline algorithm that runs in $O\left(2^l(l-1)!|\sigma|\right)$. They also prove that for lists of size 3 or greater, no deterministic algorithm that is allowed to look ahead upto a finite length in σ can compute the optimum. In 2000, Ambühl proved that offline list access problem is NP-Hard [2].

2.2 Deterministic online algorithms

Online algorithms produce outputs as they process inputs. So, technically they do not need σ to be of bounded length. But in order to bound their cost for arbitrary inputs when compared to their offline counterparts we will assume $\sigma's$ of finite length. So, given a sequence σ is it possible to construct an online algorithm that is as good as $OPT(\sigma)$? The answer is no, and in fact there's a lower bound on what the best deterministic online algorithms can do - for a list of size l, the competitive ration of the best online deterministic algorithm is at least $2-\frac{2}{l+1}$ [5].

Here are three simple strategies that use only free transpositions :

- MTF (Move to Front) is a strategy where an element that is being accessed is moved to the front of the list.

- *TRANS* (Transpose) is a strategy where the accessed element is moved one position forward.
- *FC* (Frequency Count) counts the frequency of accesses of elements and reorganizes the elements of the list monotonically according to their frequency after each access.

Using potential function analysis MTF can be shown to be $2-\frac{2}{l+1}-competitive$, which is the optimal for any deterministic algorithm where as FC and $TRANS$ are shown to be non-optimal [5].

2.3 Randomized online algorithms

We will look in detail at a simple randomized online algorithm called BIT. The algorithm is simple, as follows :

- For a list of size l have a bit $b(x)$ corresponding to each element x in the list.
- Initialize the bit randomly (to either 0 or 1) while hiding it from the adversary.
- Upon request for an element x, set $b(x) = not(b(x))$ and it is 1 move it to the front.

As opposed to MTF, this algorithm maintains state in a space of $O(l)$.

Theorem 1. *Given a finite length sequence σ of m accesses, $E[BIT(\sigma)] \leq 1.75 . OPT(\sigma) - 3m/4$*

Proof. The proof uses a potential function and looks at the difference in potentials between $OPT(\sigma)$ and $BIT(\sigma)$ after each event - partition of operations performed by the algorithms when serving requests for element accesses [7] .

There are two kinds of events : Servicing of access by both BIT and OPT, possibly including free exchanges, and a paid exchange made by OPT. Note that given a σ, OPT uses some strategy that fixes the order of events. Let bit_i, opt_i be the costs of event i to BIT and OPT. Also observe that since the initial $b(x)$ were chosen randomly, for any item x after j^{th} event, $b(x)$ is equally likely to be 0 or 1.

We will now define the potential function Φ by counting the number of inversions between BIT and OPT. Suppose BIT's list looks like $< .....y.....x..... >$ and $OPT's$ list looks like $< .....x.....y..... >$, then x and y count as an inversion. Now, suppose you want to access x, then $cost_{BIT}(x) = cost_{OPT}(x) +$

$\#\,of\ inversions\,(y,x) - \#\,of\ inversions\,(x,y)$. Also the number of accesses to x before it moves to the front $= 1 + b(x)$, which is either 1 (a type 1 inversion $= \phi_1$) or 2 (a type 2 inversion ϕ_2), depending on $b(x)$. We define the potential function $\Phi = 2\phi_2 + \phi_1$. Φ can be calculated by looking at just the internal states of both the algorithms, i.e it depends only on the current state of lists. The amortized cost of event i

$$a_i = bit_i + \Phi_i - \Phi_{i-1} \tag{2.1}$$

Case 1. Event i is an access to item x.

Let $OPT's$ list look like $< .....x..... >$where x occurs at the k^{th} position. Clearly $opt_i = k$ and $bit_i \leq k + R$ where $R = \#\,of\ inversions\ like\ (y,x)$. Let $\Delta\Phi = \Phi_i - \Phi_{i-1} = A + B + C$ where

- A = change in potential due to new inversions created during access.
- B = change in potential due to old inversions removed during access
- C = change in potential due to old inversions changing their type
  Suppose $R = r$ (the number of inversions at the time of access). Then

  Case 1. If $b(x) = 1$. Then it will be made 0 and the position of x will not change. But its type will change for the better. i.e $B = 0$, $C = -r$ and $B + C = -r$

  Case 2. If $b(x) = 0$. Then it will be made 1 and the position of x will change, and it will no longer be an inversion. i.e $C = 0$, $B = -r$ and $B + C = -r$.

  Taking the expectation of 2.1 we get

  $$E\,[a_i] = E\,[bit_i + \Delta\Phi] \leq E\,[(k+R) + (A-R)] = k + E\,[A]$$

  So now, we need to figure out what A is, i.e when are new inversions created?
  Let $OPT's$ list be like $< z_1 z_2 z_3 z_4 .....z_{k-1} x..... >$ where x occurring at the k^{th} position is moved by OPT to k'^{th} position. A new inversion is created when z_i also precedes x in $BIT's$ list but only one of OPT or BIT moves it x ahead of z_i. Let Z_i be the random variable that measures the change in potential due to each (x, z_i).

  Case 1. If $b(x) = 0$. Then it will be made 1 and BIT will move it to the front of the list. Then $Z_i \leq 1 + b(z_i)\ for\ 1 \leq i \leq k' - 1$ and $Z_i \leq 0\ for\ k' \leq i \leq k - 1$

Case 2. If $b(x) = 1$. Then it will be made 0 and *BIT* will not change $x's$ position i.e $Z_i = 0 \; for \; 1 \leq i \leq k' - 1$. But a new inversion of type 1 maybe created i.e $Z_i \leq 1 \; for \; k' \leq i \leq k-1$.

Hence

$$E[A] = \sum_{i=1}^{k-1} E[Z_i] \leq \frac{1}{2}\left(\sum_{i=1}^{k'-1}(1+b(z_i)+0)\right) + \frac{1}{2}\left(\sum_{i=k'}^{k-1} 1\right)$$
$$= \frac{1}{2}\left(\sum_{i=1}^{k'-1}(\frac{1}{2}(2)+\frac{1}{2}(1))\right) + \frac{1}{2}\left(\sum_{i=k'}^{k-1} 1\right) \leq \frac{3}{4}(k-1)$$

Therefore $E[a_i] \leq 1.75 \; . \; opt_i - 3/4$, i.e the expected amortized cost of accessing x in $BIT's$ list is less than 1.75 times that of the cost of accessing x in *OPT*.

Case 3. Event i is a paid exchange by *OPT* of items x and y.

In this case, *OPT* pays 1 for the exchange, and in the worst case the exchange creates an inversion that increases Φ by 2 with probability 0.5 and by 1 with probability 0.5. Hence $E[a_i] \leq 1.5 \, . \, opt_i$

□

The above theorem shows that randomized algorithms can perform better than deterministic ones against oblivious adversaries, i.e in cases when σ is already chosen. How is the amortized cost of *BIT* better than *MTF*? It is clearly based on the initial sequence of random bits $b(x)$. It is interesting to note that the deterministic "move to front every alternate access" is also strictly 2-competitive. Despite the fact that *BIT* lies in between this and *MTF*, it works better than both. But only against oblivious adversaries. Reingold et at. show that for the list update problem no randomized algorithm can perform better against adaptive (online or offline) adversaries than a deterministic algorithm against an oblivious one [7].

Against oblivious adversaries, the best known online algorithm, *COMB*, a combination of *BIT* and *TIMESTAMP* due to Albers, von Stengel, and Werchner, is 1.6-competitive [1] and Teia proved that no online algorithm can be better than 1.5-competitive [9].

3 The Paging Problem

Consider two levels of memory, a large memory that contains a fixed set of N pages, a small fast one that can store a subset $p < N$ pages. When a page is

accessed, it is looked up in the fast memory, cache, and retrieved if found, i.e a hit. If the page is missing in cache, called a miss, an existing page is evicted and replaced by the queried page from the slow memory. The problem is to come up with an algorithm that will minimize the number of future misses by evicting the right pages.

3.1 The optimal algorithm

Unlike the list access problem, paging has a simple algorithm that computes the optimum strategy. Longest Forward Distance, LFD, which was proved optimal by Belady [3], replaces the page that will be requested next the latest. This is an offline algorithm that needs to know the entire sequence of page accesses in advance.

3.2 Deterministic online algorithms

Deterministic algorithms for list access have analogs for deterministic paging algorithms. If an item in the list gets accessed then its usual update rule is applied, and if the element is not in the list, then the last element of the list is deleted and the new one is inserted. The items in the list can be viewed as the items in the cache.

Here are a few deterministic paging algorithms.

- *LRU* (Least Recently Used) - Replace the least recently used page, when a page needs to be evicted. This is an analog of MTF.
- $FIFO$ (First in First Out) - Replace the page that has been the fast memory longest. Corresponds to a list accessing algorithm where a new element is inserted at the front, but no rearrangement happens elsewhere.
- *LFU* (Least frequently Used) - Replace the least frequently used item in the list, analogs to FC.

But the cost models are different for list access and paging.

The (h,k)-paging problem

In the (h,k)-paging model, we measure the performance of an online paging algorithm with a cache size of k relative to an optimal offline paging algorithm with a cache size $h \leq k$.

Marking algorithm

A lot of algorithms show common properties abstracted out into the concept of a marking algorithm.

- Partition the page requests into phases - phase $(i+1)$ begins on the request that constitutes the $(k+1)^{th}$ distinct page request since the start of the i^{th} phase. Such a partition is called the $k-phase$ partition.
- Associate with the slow memory, a bit called the mark. At the beginning of a $k-phase$ unmark all pages.
- During a $k-phase$ we mark a page when it is first requested during the $k-phase$.
- A marking algorithm never evicts a marked page from fast memory.

Theorem. *Any marking algorithm ALG is* $\frac{k}{k-h+1} - competitive$

Proof. For any phase $i \geq 1$, the number of page faults in $ALG \leq k$. OPT has $h-1$ pages. It must incur at least $k-(h-1)$ faults (more if some of those $h-1$ are not in the k request pages)

Therefore, for every request sequence, $ALG(\sigma) \leq \frac{k}{k-h+1} OPT(\sigma) + \alpha$ where $\alpha \leq k$ is the maximum number of page faults by ALG during the last phase. □

Theorem. *LRU is a marking algorithm, hence it is* $\frac{k}{k-h+1} - competitive.$

Proof. Suppose it was not and evicted a marked item x during a $k-phase$. It must have marked x at some point during the $k-phase$, when it was the most recently used. But for x to become the least recently used, there have to be at least $k+1$ different pages during the $k-phase$, which is a contradiction. □

Theorem. *Let ALG be any deterministic paging algorithm, then* $R(ALG) \geq k$.

Proof. First observe that for any finite sequence σ of requests chosen from a set of $k+1$ pages, $LFD(\sigma) \leq \frac{|\sigma|}{k}$. Assume that there are $k+1$ pages $p_1, p_2, p_3 ... p_{k+1}$ the first k of which are in $ALG's$ cache. Let $\sigma = \{r_i\}$ where $r_1 = p_{k+!}$ and r_{i+1} is the unique page not in $ALG's$ cache after serving $r_1, r_2, ..., r_i$. Now, σ can be made arbitrarily long and ALG faults on each request. i.e $ALG(\sigma) = |\sigma|$ and $OPT(\sigma) \leq \frac{|\sigma|}{k}$. □

The above theorem shows that the competitiveness depends on the size of the cache, unlike the list access problem, and hence more interesting results are obtained by comparisons with an optimum algorithm of smaller cache sizes.

3.3 Randomized online algorithms

We shall look at a simple randomized algorithm for paging called *RANDOM* - when a page fault occurs, evict a page chosen randomly and uniformly among all fast memory pages. Raghavan and Snir prove that *RANDOM* is $\frac{k}{k-h+1}-competitive$ in the $(h,k)-paging$ model against adaptive online adversaries. Although LRU, $FIFO$ are also $\frac{k}{k-h+1}-competitive$ *RANDOM* doesn't use any additional memory, where as the former use $\Omega(klogk)$ and $\Omega(logk)$ bits respectively.

Theorem 2. $R_{ADON}(RAND) = \frac{k}{k-h+1}$

Proof. Let $c = \frac{k}{k-h+1}$. $\Phi = c\,(h-\phi_i)$ where ϕ_i is the number of pages *RAND* and *ADON* have in common in their caches just after serving the i^{th} request. Let $a_i = RAND_i + \Phi_i - \Phi_{i-1}$. Consider what happens after $(i-1)^{th}$ request has been serviced. Let p be the page for the i^{th} request (assume p is not in $RAND's$ cache).

Case 1. p is in adversary's cache

If *RAND* evicts a common page, which happens with a probability $\frac{\phi_{i-1}}{k}$, there is no change in Φ. Else, Φ changes by $c(\phi+1)-c(\phi)=c$. Therefore the expected decrease in potential $= c.(1-\frac{\phi_{i-1}}{k}) \geq c.(1-\frac{h-1}{k}) = \frac{k}{k-h+1}.\frac{k-h+1}{k} = 1$. *RAND* pays 1 and *ADON* pays 0.

Case 2. p is not in adversary's cache and evicts a page not common to *RAND*

Then in the worst case, *RAND* also does that and the change in potential is 0, else potential decreases. $\Phi_i - \Phi_{i-1} \leq 0$. $c.ADON_i - RAND_i = c.1 - 1 \geq 0$

Case 3. p is not in adversary's cache but evicts a common page q

Suppose *RAND* evicts a page q, then $\Delta\Phi = 0-0 = 0$. Suppose *RAND* evicts a a page not in common, then $\Delta\Phi = 1-1 = 0$. Suppose *RAND* evicts a common page but not q, then increase in potential = $c.\frac{\phi_{i-1}-1}{k} \leq \frac{k}{k-h+1}.\frac{h-1}{k} = \frac{h-1}{k-h+1}$ and $c.ADON_i - RAND_i = c.1 - 1 = \frac{h-1}{k-h+1}$

□

MARK is a randomized algorithm that is $2H_k - competitive$ against an oblivious adversary, where H_k is the k^{th} harmonic number. It is proven that for any randomized paging algorithm ALG with a cache size k, $R(ALG) \geq H_k$. The $PARTITION$ algorithm is a randomized algorithm that achieves this optimum bound [5].

4 On the power of randomization in online algorithms

We've looked at two simple randomized algorithms for two different problems and in each case they outperform their deterministic counterparts against oblivious adversaries. In this section we'll see that, under a very general framework of request-answer games, deterministic algorithms have at most quadratically worse performance compared to randomized ones. The following definitions are taken from [4].

4.1 Definitions in Request-Answer games

A request-answer game consists of a request set R, a finite answer set A, and the cost functions $f_n : R^n \times A^n \to \mathbb{R} \cup \{\infty\}$ for $n = 0, 1, ...$ Let f denote the union, over all nonnegative integers n of the functions f_n.

A deterministic online algorithm G is a sequence of functions $g_i : R^i \to A$ for $i = 1, 2, ...$ For any sequence of requests $r = (r_1, r_2, ...r_n)$ we define $G(r) = (a_1, a_2, a_3..., a_n) \in A^n$ with $a_i = g_i(r_1, r_2, ..., r_i)$ for $i = 1, ..., n$, i.e the answers given by G until some i depend on the requests r_1 to r_i. The cost of G for the request sequence r is $c_G(r) = f_n(r, G(r))$. Optimal cost for the same request sequence is the minimum over all cost answers a, i.e $c(r) = \min\{f_n(r,a) | a \in A^n\}$. G is $\alpha - competitive$ if for every request sequence r, $c_G(r) \leq \alpha(c(r))$ where α is a linear function over reals. For simplicity assume its a constant.

A randomized algorithm G is a probability distribution over deterministic online algorithms G_x, the request sequence r, the answer sequence $G(r)$, and the cost $c_G(r)$ are random variables, and we say G is $\alpha - competitive$ by taking its expectation i.e $E_x(c_{G_x}(r))) \leq \alpha(c(r))$. Since everything is predictable in a deterministic algorithm, an oblivious adversary is as powerful as adaptive ones, but that is not the case for randomized algorithms.

An adaptive offline adversary Q is a sequence of functions $q_n : A^n \to R \cup \{stop\}$ where $n = 0, 1, ..., d_Q$ and q_{d_Q} only takes the value "stop". For a deterministic algorithm G and an adaptive adversary Q we define the actual request answer sequences $r(G,Q) = (r_1, r_2, ...r_n)$, $a(G,Q) = (a_1, a_2, .., a_n)$, $n = n(G,Q)$ recursively with $r_{i+1} = q_i(a_1, a_2, ...a_i)$ for $i = 0, 1, ..., n-1$,while $a(G,Q) = G(r(G,Q))$ and $q_n(a(G,Q)) = stop$. These objects are uniquely defined in the order $r_1, a_1, r_2, a_2, ..., r_n, a_n, n$. The cost of an algorithm G against an adversary Q is $c_G(Q) = f_n(r(G,Q), a(G,Q)$, and the cost of the adaptive offline adversary Q against the algorithm G is $c_Q(G) = c(r(G,Q))$.

An adaptive online adversary $S = (Q,P)$ is an offline adversary Q supplemented with a sequence P of functions $p_n : A^n \to A$ for $n = 0,1,...,d_Q$. Since $r(G,S)$ is independent of P, we have $r(G,S) = r(G,Q)$, $a(G,S) = a(G,Q)$ and $c_G(S) = c_G(Q)$. The answer sequence of the adversary S is defined as $b(G,S) = (b_1,...,b_n)$ where $n = n(G,Q)$ and $b_{i+1} = p_i(a_1,a_2,...a_i)$ for $i = 0,1,...,n-1$. The cost of S against the algorithm G is $c_S(G) = f_n(r(G,S),b(G,S))$

4.2 Some theorems

Theorem 3. *If there is a randomized strategy that is $\alpha-competitive$ against any offline adaptive adversary then there also exists an $\alpha-competitive$ deterministic algorithm.*

Proof. The proof is by contradiction.

- Consider a two person game between R and A where R requests A and A answers.
- Position (r,a) is immediately winning for R if $f_n(r,a) > \alpha(c(r))$
- Position (r,a) is winning for R if there is an adaptive rule that will, regardless of how A plays, and in a fixed number of moves bring R to an immediately winning position
- When the game starts R wins iff there exists an offline adversary Q such that for any deterministic algorithm G $c_G(Q) > \alpha(c_Q(G))$.

Suppose G_{rand} be a randomized algorithm that is $\alpha-competitive$ against any adaptive offline adversary Q and R wins using Q. We will see that these two things cannot happen. Let G_{rand} be distributed over deterministic algorithms G_x, then since R wins against each of these deterministic algorithms, $E_x(c_{G_x}(Q)) > E_x(\alpha(c_Q(G_x)))$ which implies $E\left(c_{G_{rand}}(Q)\right) > E(\alpha(c_Q(G_{rand})))$ which means G_{rand} is not $\alpha-competitive$. Therefore, if there is an $\alpha-competitive$ randomized algorithm, then R does not have a winning strategy.

Now, if R does not have a winning strategy, then we'll see that A must have a winning strategy using a deterministic algorithm G for which $c_G(Q) \leq \alpha(c_Q(G))$. We assume finiteness of the game - request and answer sequence, as usual. A position (r,a) is a winning position for R iff there is a request r_{n+1}such that for every answer a_{n+1}, (rr_{n+1},aa_{n+1}) is winning. Therefore, A uses the strategy that starts of with a position that is not winning for R and continues to answer such

that the next position is also not winning. This is guaranteed to exist because we know R does not have a winning strategy.

□

Theorem 4. *Suppose G is $\alpha-$competitive against any online adaptive adversary and H is a $\beta-$competitive randomized algorithm against any oblivious adversary, then G is a $\alpha.\beta-$competitive against any adaptive offline adversary.*

Proof. The point is to show that $E_x\left(c_{G_x}(Q)\right) \leq \alpha\left(\beta\left(E_x\left(c_Q\left(G_x\right)\right)\right)\right)$.

- Let the randomized algorithm $H = \left\{H_y\right\}$, then for every request sequence r, $E_y\left(c_{H_y}(r)\right) \leq \beta\left(c(r)\right)$.
- For each y define an adaptive online adversary $S_y = (Q, P_y)$ such that for any deterministic algorithm F, the output of S_y, $b(F, S_y) = H_y(r(F, Q))$. i.e these adaptive online algorithms S_y produce the same answer as the deterministic H_y.
- Since G is $\alpha-$*competitive* against any adaptive online adversaries, $E_x\left(c_{G_x}(S_y)\right) \leq E_x\left(\alpha\left(c_{S_y}(G_x)\right)\right)$ for any fixed y, and taking expectations over y, $E_y E_x\left(c_{G_x}(S_y)\right) \leq E_y E_x\left(\alpha\left(c_{S_y}(G_x)\right)\right)$.
- Therefore for every y,

$$\begin{aligned} E_x\left(c_{G_x}(Q)\right) = E_y E_x\left(c_{G_x}(S_y)\right) &\leq E_y\left(\alpha\left(E_x\left(c_{S_y}(r_x)\right)\right)\right) \\ = \alpha\left(E_x E_y\left(c_{H_y}(r_x)\right)\right) = \alpha\left(E_x\left(c_H(r_x)\right)\right) &\leq \alpha\left(E_x\left(\beta\left(c(r_x)\right)\right)\right) \\ &= \alpha\left(\beta\left(E_x(c_Q(G_x))\right)\right) \end{aligned}$$

□

4.3 Observations

The proofs for the above two theorems rely on the observation that a randomized algorithm can be thought of as a probability distribution over a set of deterministic algorithms. Using this decomposition, intuitively it makes sense that a randomized algorithm against an adaptive offline adversary, who knows the current internal state of the algorithm and gets to make optimum choices effectively has the same power as an oblivious adversary against deterministic algorithms, who again can simulate the deterministic algorithm and make optimum choices.

Adaptive offline adversary is so powerful that randomization doesn't help against it.

There is another interesting adversarial case that comes to mind, the power of which is between adaptive online and adaptive offline. An adaptive online adversary gets to choose the next request based on the previous answers, but has to pay his online price. What if instead, the adversary tries to figure out, in part of full, the random state within *ALG* based on the online output it gives, and then change subsequent elements of σ to the detriment of *ALG*. Let us look at *BIT*. Consider an adversary that is allowed to change σ, i.e be adaptive, but doesn't get to see either the list maintained by *BIT* or its random bits. But before serving the original sequence, the adversary tries to figure out the list and its random bits by measuring the list access time for that element. Suppose the list contains $\{a,b,c,d\}$ in some order, the adversary could issue an element, say a, twice to find that the access times are different - and by that he could figure out both the current position and the random bit associated with that element. If the access times for a are same, then he could try to use different elements until access times are different. If the length of the list is l, then clearly after $O(l)$ element accesses, the adversary can completely figure out the internal state of *BIT*, which renders it completely deterministic. At this point, the adaptive online adversary can pay the offline price because the algorithm becomes deterministic (assuming σ is large enough to offset the initial state "discovery" costs).

Effects of randomization in algorithms clearly affect the cost function. It might be interesting to investigate the properties of the relation between the internal random state of an algorithm and the cost benefit the randomization provides, including whether the mapping can be made one-way, to prevent discovery attacks. The question is, given a randomized algorithm G does there exist an algorithm G_D that produces a sequence of requests that discovers the internal state of G, rendering it deterministic? This may be possible against barely random algorithms, but against behavioral algorithms the discovery sequences must be intermittently generated.

References

[1] Susanne Albers, Bernhard Von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *INFORMATION PROCESSING LETTERS*, 56:135–139, 1995.

[2] Christoph AmbÃŒhl. Offline list update is np-hard. In *In Proceedings of the 8th Annual European Symposium (ESA 2000), volume 1879 of LNCS*, pages 42–51. Springer, 2000.

[3] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.*, 5:78–101, June 1966.

[4] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *Algorithmica*, pages 379–386, 1990.

[5] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

[6] Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. Technical report, 1996.

[7] Nick Reingold, Jeffery Westbrook, and Daniel D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1992.

[8] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of The ACM*, 28:202–208, 1985.

[9] Boris Teia. A lower bound for randomized list update algorithms. *Inf. Process. Lett.*, 47:5–9, August 1993.