Online Algorithms	23/11/2011

Lecture Notes 4: Random Algorithms

Professor: Yossi Azar *Scribe: Oren Ish-Am*

1 Introduction

1.1 Notations

The following notation will be used throughout the class

- On-Line algorithms - ON
- Optimal Off-Line algorithm - OPT
- The sequence of requests - σ

A random algorithm A, can be seen as an algorithm capable of flipping coins.
Traditionally, the running time T is a random variable (r.v), and most times we wish to minimize $\mathbb{E}(T)$.
In online and approximation algorithms, the quality of the solution is a r.v. and we wish to minimize (or maximize) its value.

Definition 1. *A random Algorithm A (for a minimization problem) is C-competitive if* $\forall\sigma : \mathbb{E}\left(A(\sigma)\right) \leq C \cdot OPT(\sigma)$

Notice that:

- The expectation is *not* over the all possible input series σ, but over A's random coins
- First σ is set and only then does the coin tossing begin
- σ does not change as a function of the coin tossing output
- OPT has no need for randomness, it is optimal for this specific σ

Random algorithms often outperform their deterministic counterparts. Among the advantages of randomness, is that no specific input repeatedly causes worst-case performance.

2 Examples

2.1 QuickSort

1. Randomly pick an element called a *pivot* from the list
2. Partition the list to elements greater than pivot and smaller than pivot, move them to the right and left of pivot

3. Pivot is in final position - recursively sort sub-list of lesser elements and sub-list of greater elements

QuickSort has an expected $\Theta(n\log(n))$ running time on a list of length n, but can have a worst-case $\Theta(n^2)$ running time.

2.2 Find '1' in array

We are given a large array with $\frac{n}{2}$ 1's and $\frac{n}{2}$ 0's and wish to find an index which contains '1'.

- Any deterministic algorithm has an input array which will result in $T = \frac{n}{2}$
- A random algorithm will pick an index at each step, $\mathbb{E}(T) = 2$

2.3 Primality Testing

Given a number p as input, we wish to know if p is prime. The number is given in binary format so that $|input| \propto \log(p)$

- Naïve algorithms, looking for all possible factors up to $\sqrt{p}$, result in exponential running time
- Random algorithms exist which prove primality, w.h.p. for all numbers in polytime

2.4 MAX Cut

Given a Graph $G = (V, E)$, we wish to find a partition of the vertex set $S, V-S$, such that $E(S, V-S)$ is maximal.

Theorem 2.1. *If we place each vertex in S or $V-S$ w.p. 1/2, the algorithm achieves an approximation ratio of 1/2.*

Proof. For some edge $e = (u, v)$, one of 4 options exist:

- $u, v \in S$
- $u, v \in V-S$
- $u \in S, v \in V-S$
- $v \in S, u \in V-S$

Therefore, the probability that $e = (u, v)$ is in the cut is 1/2. For each $e \in E$ we create an indicator X_e s.t. $X_e = 1$ iff e is in the cut:

$$\mathbb{E}\left(\text{Size of Cut}\right) = \mathbb{E}\left(\sum_{e\in E} X_e\right) = \sum_{e\in E} \mathbb{E}(X_e) = \frac{1}{2}|E|$$

Since the size of the cut is at most $|E|$ the approximation ratio is at least 1/2. $\square$

3 Random On-Line Algorithms

3.1 Ski Rental

Pay 1 to rent a unit time, pay M to buy. We have seen a deterministic 2-competitive algorithm (buys after time M).

3.1.1 Random Algorithm 1

Buy w.p. 1/2 at time $M/2$ and w.p. 1/2 at time M
Analysis: The worst-case for ON is to have the series terminate right after it has decided to buy. It is therefore enough to consider two sequences, of length M and $M/2$:

Sequence of length M

$$OPT(\sigma) = M$$

$$\mathbb{E}(ON(\sigma)) = \frac{1}{2} \cdot 2M + \frac{1}{2} \cdot \frac{3}{2}M = \frac{7}{4}M$$

$$\text{Competitive Ratio} = \frac{7}{4}$$

Sequence of length $M/2$

$$OPT(\sigma) = \frac{M}{2}$$

$$\mathbb{E}(ON(\sigma)) = \frac{1}{2} \cdot \frac{M}{2} + \frac{1}{2} \cdot \frac{3}{2}M = M$$

$$\text{Competitive Ratio} = 2$$

Hence, there is no improvement.

3.1.2 Random Algorithm 2

Buy w.p. 1/2 at time $t = \alpha M, \alpha \leq 1$, and w.p. 1/2 at time M.
Analysis: Again, we will consider two possible sequences: of length M and αM.

Sequence of length M

$$OPT(\sigma) = M$$

$$\mathbb{E}(ON(\sigma)) = \frac{1}{2} \cdot (\alpha + 1)M + \frac{1}{2} \cdot 2M = \frac{\alpha + 3}{2}M$$

$$\text{Competitive Ratio} = \frac{\alpha + 3}{2}$$

Sequence of length αM

$$OPT(\sigma) = \alpha M$$

$$\mathbb{E}(ON(\sigma)) = \frac{1}{2} \cdot (\alpha + 1)M + \frac{1}{2} \cdot \alpha M = \frac{2\alpha + 1}{2}M$$

$$\text{Competitive Ratio} = 1 + \frac{1}{2\alpha}$$

To achieve the best possible competitive-ratio, we will ask for

$$\min\left(\max\left(\frac{3+\alpha}{2}, 1 + \frac{1}{2\alpha}\right)\right)$$

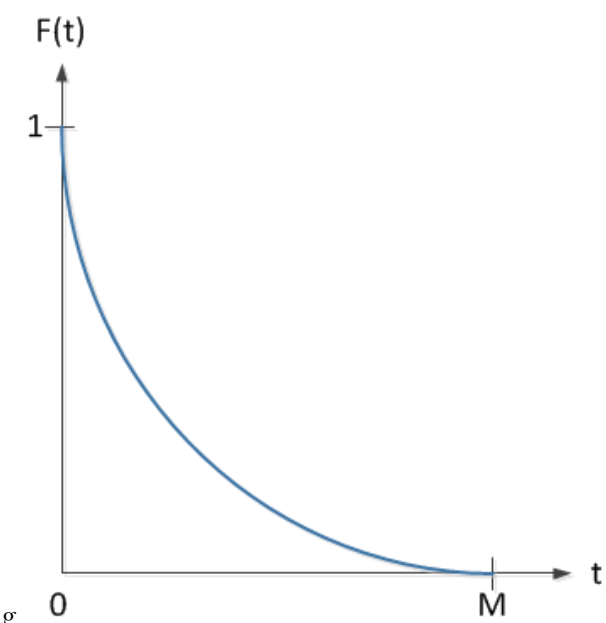
This will be achieved when both competitive-ratios are equal:

$$1 + \frac{1}{2\alpha} = \frac{3+\alpha}{2} \quad \Rightarrow \quad \alpha^2 + \alpha - 1 = 0$$

$$\alpha = \frac{\sqrt{5}-1}{2} \quad \Rightarrow \quad competitive - ratio \approx 1.809 < 2$$

3.1.3 General Random Algorithm

Buy with some continuous probability over $[0, M]$.
Analysis: Obviously, we can gain nothing by buying after time M, and so we wish to inspect the general case, where some probability for buying is given to each moment of time in $[0, M]$.
Let us denote by $F(t)$ the probability that we have *not* bought until time t.



Ski Rent.png

Figure 1: Buying with continuous probability

Note that $-F'(t)$ is the probability density function for buying at time t (the probability that we did not buy at $t + \Delta t$, minus the probability that we did not buy at t).

Given a series of requests of length $t, 0 \leq t \leq M$, we can compute the cost of this approach - Cost(t) as

$$\text{Cost}(t) = \overbrace{\int_0^t F(x)dx}^{\text{renting}} + \overbrace{[1 - F(t)]\, M}^{buying}$$

For optimal performance with this approach, we wish that for every series of length t, we can hold a certain competitive-ratio β against OPT. For $0 \leq t \leq M$, OPT pays t, and so we can define the target function:

$$\text{Cost}(t) = \beta t \quad \Rightarrow \quad g(t) = \int_0^t F(x)dx + [1 - F(t)]\, M - \beta t = 0$$

To solve this equation we can differentiate g w.r.t. t:

$$g'(t) = F(t) - MF'(t) - \beta = 0$$

This yields a simple differential equation:

$$\frac{F'(t)}{F(t) - \beta} = \frac{1}{M} \quad \Rightarrow \quad [\ln(F(t) - \beta)]' = \left(\frac{1}{M}t + \text{const}\right)'$$

And this leads to:

$$F(t) = C \cdot \mathrm{e}^{\frac{t}{M}} + \beta$$

We can find C and β according to the conditions at the extremes. At time $t = 0$ the probability that we *have* bought must be zero. At the end when $t = M$ the probability that we *have not* bought until now must be zero:

$$F(0) = 1, F(M) = 0 \quad \Rightarrow \quad C + \beta = 1, C \cdot \mathrm{e}^1 + \beta = 0$$

We find the final continuous probability function and a surprisingly good competitive-ratio, much better than the deterministic case:

$$F(t) = (1 - \beta)\mathrm{e}^{\frac{t}{M}} + \beta \quad , \quad \beta = \frac{\mathrm{e}}{\mathrm{e} - 1} \approx 1.58$$

3.2 Survivor Game

Start with n people. At every stage someone is eliminated and we don't know who it will be.
Let's assume that we are promoting TAU CS and we want to have a contestant present wearing the cool TAU-CS T-shirt at all times.
We choose a person, pay him \$1 (fixed price) until he leaves. Then we choose someone else etc.
We want to minimize the money spent on the campaign.
Any deterministic algorithm may lead to a rigged game so that we have to pay $\$n$, while OPT would pay \$1 (choose the winner first time around).

3.2.1 Random Algorithm

Every time someone leaves, choose a new one at random from those still left.
Analysis: If i people are left, the probability that we will pay \$1 at this stage (meaning that the one we chose will be the next to leave) is $\frac{1}{i}$.
This is assuming a fair game, which is doubtful.
The expected cost is:

$$\mathbb{E}\left(ON(\sigma)\right) = \sum_{i=n}^{1} \frac{1}{i} = H_n \approx \ln(n)$$

$$\text{Competitive Ratio} = \frac{ON}{OPT} = \frac{\ln(n)}{1} = \ln(n)$$

3.3 Paging

Recall that we have seen LRU - a deterministic k-competitive algorithm.

3.3.1 Random Algorithm 1

When encountering a Page Fault (PF), discard a random page.
Analysis: This does not seem to be a good approach. Consider the following scenario, n is the number of possible pages and k is the number of pages in the memory:

- $n = k + 1$
- Currently the memory holds $1, 2, ...k$
- $\sigma = \underbrace{(k+1, 1, 2, 3, ..., k-1)^l}_{A}\underbrace{(k, 1, 2, 3, ..., k-1)^l}_{B}(A)(B)\cdots$
- $l >> k$

OPT will replace $k+1$ with k (k with $k+1$) when moving from A to B (B to A) and pay only one PF per $k \cdot l$ page requests.
ON will take k PFs on average for each block (A or B), until it brings the correct page. This yields a competitive-ratio of k.
Using randomness, we can do much better...

3.3.2 (Random) Marking Algorithm - (R)MA

Algorithm works in phases:

1. At the start of each phase, all pages are unmarked
2. Every page requested becomes marked (whether in memory of brought in by a PF)

3. Upon PF, discard an unmarked page (chosen uniformly over all unmarked pages)
4. Phase ends directly before the first PF when all pages are marked. At this time Turn all pages to unmarked

RMA Analysis:

Theorem 3.1. *Phases are not affected by the randomness (which page to throw)*

Proof. Every phase (like in the LRU analysis) is a maximal requests series of k different pages. □

Theorem 3.2. *Every algorithm in the (R)MA family is at most k-Competitive*

Proof. Every phase RMA has at most k PFs and OPT has at least 1 PF for each 1-shifted phase (like in the LRU analysis). □

Theorem 3.3. *RMA is $2H_k$-Competitive, where* $H_k = \left(1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{k}\right) \approx \log(k)$

Proof. To prove this we will first need some definitions:

Definition 2. *Let m_i be the number of pages in phase i which are different than the pages in phase $i-1$*

For example $1\,2\,3\,2|\underbrace{4\,5\,1\,4\,5}_{m=2}|\underbrace{3\,4\,5\,4\,5}_{m=1}|$

Notice that $m_i \geq 1$ or the two blocks would become one.

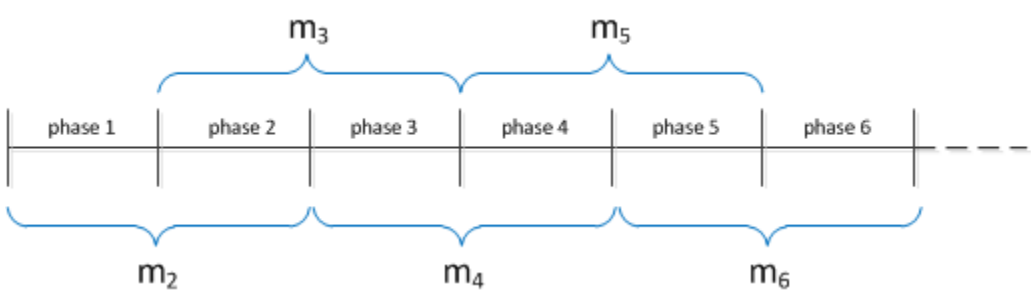


Figure 2: request phases

If we look at phases $i-1$ and i, we have requests for $k + m_i$ pages. OPT pays at least m_i for these two phases and so we can write:

$$OPT \geq \sum m_{2i} \quad \text{and also} \quad OPT \geq \sum m_{2i+1}$$

If we average we see that:

$$OPT \geq \frac{1}{2} \sum m_i$$

How much does RMA pay in phase i?

RMA pays at least m_i since these pages are new in phase i. While making room for the new pages, RMA may drop pages which will be requested later on in this phase but have not been requested yet (recall that pages requested are marked and are drop-resistant in current phase).

Analysis:

Consider what happens when we receive the $j+1$ request for an old unmarked page ($0 \leq j \leq k - m_i - 1$), when so far there have been $l \leq m_i$ new pages due to PFs:

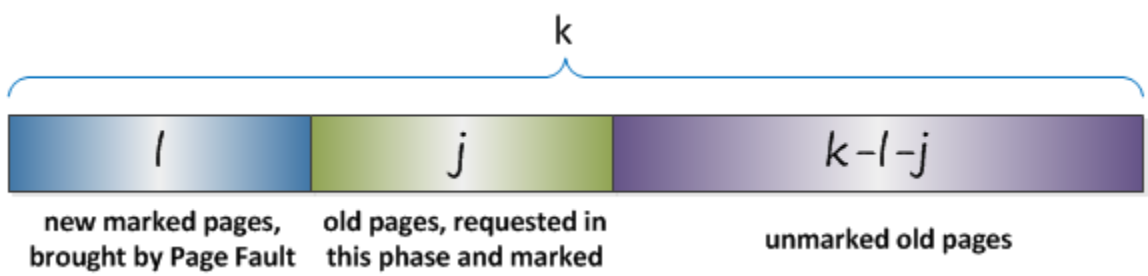


Figure 3: RMA Phase after m_i PFs and j old requests

The number of old unmarked pages (including those accidentally discarded during the phase) is $k - j$.

The probability that the requested page is in the memory is $\dfrac{k-l-j}{k-j}$ and the probability of a PF:

$$1 - \frac{k-l-j}{k-j} = \frac{l}{k-j} \leq \frac{m_i}{k-j}$$

And so the expected number of PFs for RMA in phase i:

$$\mathbb{E}\left[RMA(\sigma_i)\right] \leq m_i + \sum_{j=0}^{k-m_i-1} \frac{m_i}{k-j} = m_i + m_i \sum_{j=0}^{k-m_i-1} \frac{1}{k-j} \quad \Rightarrow$$

$$\mathbb{E}\left[RMA(\sigma_i)\right] \leq m_i + m_i \left(H_k - \overbrace{H_{m_i}}^{\geq 1} \right) \leq m_i H_k$$

So now $\mathbb{E}\left[RMA(\sigma)\right] \leq H_k \sum m_i$ and $OPT \geq \dfrac{1}{2} \sum m_i$.

We get:

$$\text{Competitive Ratio} = \frac{RMA}{OPT} = 2H_k$$

$\square$