6.854 Advanced Algorithms

Lecture 20: 19/11/2003 Lecturer: Erik Demaine, David Karger
Scribes: Anastasios Sidiropoulos

More Online Algorithms

20.1 Randomization

A randomized online algorithm can be viewed as a probability distribution over deterministic online algorithms; at each execution, the randomized algorithm picks a deterministic algorithm according to a probability distribution. In practice, a randomized algorithm makes random choices during the execution. This way, the deterministic algorithm is defined at the end of the execution.

Clearly, the cost of an execution is a random variable, since it depends on the random choises made by the algorithm. Thus, we need to introduce the following definition of the competitive ratio of a randomized online algorithm.

Definition 1 *A randomized online algorithm $\mathcal{A}$ has competitive ratio α if, for every input sequence σ, $E[C_{\mathcal{A}}(\sigma)] \leq \alpha C_{OPT}(\sigma)$.*

There are different ways to define an adversary for a randomized algorithm. This is because we can make different assumptions for the knowledge of the adversary conserning the random choises of the algorithm. More specifically, we consider the following different kinds of adversaries:

Oblivious: An oblivious adversary knows the probability distribution of the randomized algorithm, but not the specific random choises. That is, the adversary picks the worst-case sequence in advance.

Fully adaptive: A fully adaptive adversary knows the probability distribution and the random choises made by the algorithm in advance. Under this assumption, a randomized algorithm is equivalent to a deterministic algorithm.

Adaptive: An adaptive adversary knows the random choises of the randomized algorithm so far. This implies that at each step of the execution, the next part of the input will be a function of the current state of the algorithm. Intuitively, an adaptive adversary is more powerful than an oblivious adversary, but less powerful than a fully adaptive adversary.

20.2 Paging revisited

We consider the paging problem, in a cache of size k. We will show that using randomization, an online algorithm can achieve a better competitive ratio than any deterministic algorithm.

20.2.1 The RANDOM paging algorithm

The RANDOM paging algorithm is the simplest randomized algorithm for the paging problem. At each step, if the requested page is not in the cache, the algorithm evicts a page which is chosen uniformly at random.

For any $c < k$, it is known that no deterministic paging algorithm can be c-competitive (Theorem 3 of scribe notes for Lecture 19). We will show that RANDOM achieves the same competitive ratio with the best possible deterministic algorithm. We note that RANDOM has constant space requirements, while FIFO, and LRU require additional $O(k)$ space.

Theorem 1 RANDOM *is k-competitive, versus an adaptive adversary.*

Proof: We will use a potential function argument. Let d_t be the number of pages in RANDOM's cache, that are not in OPT's cache, at time t. Define $\Phi(t) = kd_t$, to be the potential function. Let $c_{RAND}(i)$ be the amortized cost of the i-th request for RANDOM, and let $c_{OPT}(i)$ be the cost of the same request for OPT. We will show that for each request i, it is $c_{RAND}(i) \leq k \cdot c_{OPT}(i)$. We have to distinguish between the following cases:

Case 1, The next page is in RANDOM*'s cache:* In this case, it is $c_{RAND}(i) = 0$.

Case 2, The next page is not in RANDOM*'s cache, and it is in OPT's cache:* If RANDOM evicts a shared page, then $\Phi(i) - \Phi(i-1) = 0$. Otherwise, $\Phi(i) - \Phi(i-1) = -k$. Since the evicted page is chosen uniformly at random, we obtain

$$E[\Phi(i) - \Phi(i-1)] = \frac{d_i}{k}(-k) = -d \leq -1.$$

The last inequality follows from the fact that the requested page was in OPT's cache, but not in RANDOM's cache.

Case 3, The next page is neither in RANDOM*'s, nor in OPT's cache, and OPT evicts an unshared page:* In this case, it is $\Phi(i) - \Phi(i-1) \leq 0$, so the amortized cost is at most 1.

Case 4, The next page is neither in RANDOM*'s, nor in OPT's cache, and OPT evicts a shared page:* If RANDOM evicts the same, or an unshared page, then $\Phi(i) - \Phi(i-1) = 0$. Otherwise, $\Phi(i) - \Phi(i-1) = k$. Since there are $k - d_i - 1$ unshared pages (excluding the one that OPT evicted), the probability of the last event is $\frac{k-d_i-1}{k}$. Thus, we obtain

$$E[\Phi(i) - \Phi(i-1)] = k - d_i - 1 \leq k = k \cdot c_{OPT}(i)$$

Thus, the total cost of the RANDOM algorithm is at most k times the cost of the optimal offline algorithm. ■

20.2.2 The MARKING paging algorithm

In this section we will present the MARKING randomized paging algorithm, which is due to Fiat et. al. [1]. The algorithm keeps one bit per cache position, determining whether the corresponding page is marked.

Initially, all the pages are marked. For every request, if the requested page is already in the cache, the algorithm marks the requested page. On a miss, if all the pages are marked, the algorithm unmarks all of them. Next, it evicts one of the unmarked pages, which is chosen uniformly at random, and it marks the requested page. The algorithm is summarized in the following table.

The MARKING paging algorithm	
Initially:	All the pages are marked.
For every request:	*If* the requested page is not in cache, *then*
	If all pages are marked, *then*
	unmark all the pages.
	Chose uniformly at random an unmarked page p.
	Evict p.
	Mark requested page.

As the following Theorem states, this simple algorithm is $O(\log k)$-competitive. Note that this competitive ratio is better than the competitive ratio of any deterministic algorithm.

Theorem 2 *The* MARKING *paging algorithm is* $O(\log k)$*-competitive.*

Proof: In order to analyse the performance of the MARKING paging algorithm, we will partition the input sequence into phases of consecutive requests. The first phase starts on the first fault, and in general, every phase is a maximal sequence with k distinct requests (every phase starts exactly after the end of the previous phase). Observe that the MARKING algorithm implicitly keeps track of the phases by unmarking all the pages at the start of a phase.

Moreover, note that all the pages that are requested during a phase, are marked and they stay in the cache until the phase ends. Let S_i be the set of pages in the cache when phase i starts. During a phase i, we define a request to be *clean* if it is not in S_i, and *stale* otherwise. It is easy to see that the MARKING algorithm pays for every clean request, but OPT has to pay too. However, it might be the case that MARKING has to pay for some of the stale requests.

Suppose that at some step we have already seen s stale, and c clean requests. There are s pages from S_i already in the cache, but the remaining $k - s$ requests might not be in the cache, due to clean requests. Thus, the probability that a new stale request is not in the cache is at most $c/(k-s)$. If follows that if there are c_i clean requests in the whole phase, then the total expected cost of the stale requests is at most:

$$c_{stale}(i) \leq \frac{c_i}{k} + \frac{c_i}{k-1} + \ldots + \frac{c_i}{k-s_i} \leq c_i H_k = O(c_i \log k)$$

Since the cost for the clean requests is c_i, the total cost of phase i is $O(c_i \log k)$. Thus, the total cost of the MARKING algorithm is $O(\sum_i c_i \log k)$.

It remains to derive a lower bound for the cost of the optimal offline algorithm OPT. We define a potential function $\Phi(i)$ to be the number of pages that are in MARKING's, but not in OPT's cache, at the start of phase i. Since there exist c_i clean requests that are not in MARKING's memory at the start of phase i, it follows that there exist at least $c_i - \Phi(i)$ requests that are not in OPT's memory at the start of the phase. At the end of the phase, MARKING has the k most recent requests, while

OPT has evicted $\Phi(i+1)$ of them. Thus, the cost of the optimal offline algorithm during phase i is at least:

$$c_{OPT}(i) \geq \max\{c_i - \Phi(i), \Phi(i+1)\} \geq \frac{1}{2}(c_i - \Phi(i) + \Phi(i+1))$$

By summing over all phases, we obtain that the cost of the total cost of the optimal offline algorithm is at least $\frac{1}{2}(\Phi(l) - \Phi(1) + \sum_i c_i)$, where l is the total number of phases. Since $\Phi(1) = 0$, and $\Phi(l) \leq k$, the Theorem follows. ■

20.3 Lower bounds for randomized online algorithms

A (Las Vegas) randomized algorithm can be viewed as a probability distribution over a set of (correct) deterministic algorithms. Under this perspective, the design of a randomized algorithm can be phrased in terms of a game played between the randomized algorithm and an adversary. The randomized algorithm has to chose a deterministic algorithm to minimize a desired value (e.g. running time, or competitive ratio), while the adversary has to chose an input such that the same value is maximized.

If the algorithm and the adversary have n and m distinct choices respectively, then we can define an $n \times m$ *payoff matrix* M for this game. The value of the item in the ith row and the jth column of M is the value of the objective if the algorithm picks the ith strategy, and the adversary picks the jth input. If the first player (the algorithm) picks a row according to a probability distribution p, and the second player (the adversary) picks a column according to a probability distribution q, then the expected cost is $p^T M q$.

Von Neumann's Minimax Theorem states that for any *two-person zero-sum* game, specified by a matrix M, it is

$$\min_p \max_q p^T M q = \max_q \min_p p^T M q,$$

where the maximum and the minimum are taken over all possible distributions p and q. If one player picks a probability distribution p, then the optimal strategy for the other player is deterministic (i.e. q is a 0-1 vector).

Theorem 3 (Yao's Minimax Principle) *If there exists an input distribution against which no deterministic algorithm is c-competitive in expectation, then there is no c-competitive randomized online algorithm.*

Proof: Consider the payoff $E[C_A(\sigma) - c \cdot C_{OPT}(\sigma)]$. Suppose that we have a c-competitive randomized algorithm. It follows that the payoff is negative in expectation, even against the best σ-distribution. Thus, against this best σ-distribution, there exists a deterministic algorithm achieving negative payoff, a contradiction. ■

20.3.1 A lower bound for paging

In this section we will apply Yao's Minimax principle to obtain a lower bound for the competitive ratio of any randomized online paging algorithm, versus an oblivious adversary. This bound matches asymptotically the competitive ratio achieved by the MARKING algorithm, which implies that the MARKING algorithm is optimal within a constant factor.

Theorem 4 *Let R be a randomized paging algorithm. If R is c-competitive versus an oblivious adversary, then $c = \Omega(\log k)$.*

Proof: By Yao's Minimax Principle, the competitive ratio of an optimal randomized algorithm is equal to the expected competitive ratio of an optimal deterministic algorithm, under a worst case input distribution. Thus, in order to obtain a lower bound for the competitive ratio of any randomized algorithm, it suffices to bound the competitive ratio of any deterministic algorithm, under a specific probability distribution of the input request sequence.

Consider the input distribution, where each request in chosen uniformly at random from a set of $k+1$ distinct pages. We partition the input sequence into phases as follows: Phase 1 starts with the 1st request, and ends after all the pages have been requested at lest once. In general, phase i starts after the end of phase $i-1$, and ends just before the $(k+1)$th distinct page request within that phase.

Recall that in the case of a miss, the optimal offline algorithm evicts from the cache the page which will be requested furthest in the future. Clearly, the optimal offline algorithm misses at least once at each phase. Thus, it suffices to obtain an upper bound for the expected length of a phase.

Consider the following random experiment: We are given $k+1$ empty bins, and we repeatedly pick a bin uniformly at random, and we throw a ball in it. Clearly, the expected number of balls required to cover all bins is equal to the expected length of a phase. Thus, using a Coupon Collector's argument, we obtain that the expected length of a phase is $(k+1)H_{k+1}$.

Let now A be any deterministic algorithm. Since each request is chosen uniformly at random from a set of $k+1$ pages, A misses at each request with probability $1/(k+1)$. Thus, the expected number of misses of A per phase, is at least H_{k+1}. It follows that the expected number of misses of A is at least H_{k+1} times the number of misses of the optimal offline algorithm. ■

References

[1] A. Fiat, R. M. Karp, M. Luby, L.A. McGeoch, D. D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.