

## Notes for Lecture 21

### 1 Randomized Paging Algorithms

The *paging problem* is the following. We have a slow memory with  $N$  distinct pages and a fast memory, also referred to as *cache*, that can contain at most  $k$  pages, where  $k < N$ . An unknown sequence  $\sigma_1, \sigma_2, \dots$  of page requests is received. If a requested page  $\sigma$  is in the fast memory at the time of request, no cost is incurred. If the requested page is not in the cache, a page must be evicted from the cache in order to bring in page  $\sigma$ . In the last lecture, we saw several deterministic schemes for paging. We, also, introduced a family of deterministic algorithms, the *marking algorithms*, which achieve competitive ratio of  $k$ . Finally, we showed a lower bound of  $k$  for the competitive ratio of any deterministic algorithm, thus establishing that the marking algorithms are optimal deterministic algorithms.

In this lecture we consider the performance of *randomized paging algorithms*, in which the mechanics of the page evictions are not deterministic. Having such an algorithm in hands, there are two types of adversaries against which we could test its performance:

- *Oblivious Adversaries* which know the code of the algorithm and, based on this information only, must commit to a request sequence.
- *Adaptive Adversaries* which not only know the code of the algorithm but, moreover, can see its responses to requests  $\sigma_1, \sigma_2, \dots, \sigma_i$  before deciding what page  $\sigma_{i+1}$  to request next.

Note that for deterministic algorithms the above separation of adversaries into two categories is meaningless: given the code of a deterministic algorithm one can predict its response to a sequence of requests. For randomized algorithms, on the other hand, where the separation is meaningful, comparison against different types of adversaries possibly gives different competitive ratios. Which type of adversary is the judicious choice to test an online algorithm against depends on the nature of the problem.

Coming back to the paging problem, we claim that the natural choice of adversary is the oblivious one. There is no formal justification for this, of course, just the supposition that the page requests to follow should not depend on the choices that the algorithm has done so far; there is no reason to believe that the page requests change depending on the choices of the algorithm as to which pages to evict from the fast memory or not. Therefore, in the discussion to follow, *we will focus on oblivious adversaries*. Here is the definition of the *competitive ratio* against oblivious adversaries:

**Definition 1** *An online algorithm ALG is  $c$ -competitive against oblivious adversaries if there exists a constant  $\alpha$  such that for all inputs  $\sigma$*

$$\mathbb{E}[\text{ALG}(\sigma)] \leq c \text{OPT}(\sigma) + \alpha$$

## 1.1 A Randomized Online Paging Algorithm

Here is a simple randomized marking algorithm, whose performance we will analyze next.

**Algorithm** RANDMARK

- Initially all pages in the fast memory are unmarked.
- On request  $x$ :
  - if** fast memory is not full **then**
    - bring page  $x$  in the fast memory and mark it; /\*cost=1\*/
  - else**
    - if** all pages in fast memory are marked **then**
      - unmark all pages in fast memory;
  - endif**
  - if**  $x$  is in fast memory **then**
    - mark  $x$ ; /\* cost=0 \*/
  - else**
    - choose  $y$  uniformly at random from the set of unmarked pages in fast memory;
    - replace  $y$  by  $x$ ;
    - mark  $x$ ; /\* cost=1 \*/
  - endif**
- endif**

Let  $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{k}$  be the  $k$ -th harmonic number ( $H_k \sim \ln k$ ). The following theorem specifies the performance of RANDMARK.

**Theorem 2** [2] *Algorithm RANDMARK is  $(2H_k)$ -competitive against oblivious adversaries, where  $k$  is the size of the fast memory.*

PROOF: Let us divide the request sequence into phases recursively as follows:

- *phase 0* is the null sequence
- *phase  $i$*  is the longest sequence following phase  $i - 1$  in which at most  $k$  distinct pages are requested

We will analyze phase  $i$  of the request sequence for some fixed  $i \geq 0$ . Call a page requested in phase  $i$  **new**, if it was not requested in phase  $i - 1$ , and **old**, if it was requested in phase  $i - 1$ <sup>1</sup>. Let  $M_i$  be the number of new pages requested in phase  $i$ ,  $x_j$  the  $j$ -th old page requested and  $m_j$ ,

---

<sup>1</sup>For phase 1, call new all requested pages that were not present in the fast memory when the algorithm started and old those that were present in the fast memory.

$m_j \leq M_i$ , the number of new pages requested before  $x_j$  is requested. Clearly, each request for a new page causes an eviction. Moreover, at the time of the request for  $x_j$  fast memory will contain  $m_j + j - 1$  marked pages, none of which is  $x_j$ , and  $k - (m_j + j - 1)$  old pages, comprising a random subset of the  $k - (j - 1)$  pages present in the fast memory at the beginning of phase  $i$  and not yet requested during phase  $i$ . Thus, the probability that  $x_j$  is not in the fast memory at the time it is requested is equal to

$$\frac{\binom{k-(j-1)-1}{k-(m_j+j-1)}}{\binom{k-(j-1)}{k-(m_j+j-1)}} = \frac{m_j}{k-j+1} \leq \frac{M_i}{k-j+1}$$

(It can be checked that the above is an upper bound on the probability that  $x_j$  is not in the fast memory even for  $i = 1$  and even if the fast memory was not full at the beginning of the algorithm.) Therefore, the expected number of evictions in phase  $i$  is at most

$$M_i + \sum_{j=1}^{k-M_i} \frac{M_i}{k-j+1} = M_i(1 + H_k - H_{M_i}) \leq M_i H_k$$

Now, let  $s_i$  be the cost incurred by the optimal algorithm OPT during phase  $i$  and  $t_i$  the expected cost incurred by RANDMARK. We have shown that  $t_i \leq M_i H_k$ . Also,  $s_1 = M_1$ , since every new page in phase 1 causes an eviction, and  $s_{i-1} + s_i \geq M_i$ , since  $k + M_i$  distinct pages are requested during phase  $i - 1$  and  $i$  so at least  $M_i$  of these must cause a page fault.

If the request sequence  $\sigma$  has  $n$  phases we have that

$$\mathbb{E}[\text{RANDMARK}(\sigma)] = \sum_{i=1}^n t_i \leq H_k \sum_{i=1}^n M_i$$

Also summing the inequalities involving the  $s_i$ 's we get

$$2(s_1 + s_2 + \dots + s_n) - s_n \geq \sum_{i=1}^n M_i$$

which implies

$$\text{OPT}(\sigma) = \sum_{i=1}^n s_i \geq \frac{1}{2} \sum_{i=1}^n M_i$$

Combining the above we get

$$\mathbb{E}[\text{RANDMARK}(\sigma)] \leq (2H_k) \text{OPT}(\sigma)$$

which implies that RANDMARK is  $2H_k$ -competitive.  $\square$

## 1.2 A Lower Bound

We will derive a lower bound on the competitive ratio of any randomized online paging algorithm against oblivious adversaries.

**Theorem 3** [2] *The competitive ratio of any randomized online paging algorithm ALG against oblivious adversaries is at least  $H_k$ , where  $k$  is the size of the fast memory.*

PROOF: Let us limit ourselves to the case where the total number of pages is  $k + 1$ . In this case, at any step exactly one page is missing from ALG's fast memory. However, because ALG is randomized, the oblivious adversary does not know which page is missing and can only compute the probability  $p_x$  that any given page  $x$  is not in the fast memory. Clearly,  $\sum_x p_x = 1$ . Moreover, if a page  $x$  is requested, the expected cost incurred by ALG is  $p_x$ , where  $p_x$  was the probability that  $x$  was not in the cache before the request. As a result of responding to the request,  $p_x$  changes to 0, and the probabilities of some other pages may change. We will describe, next, how, by keeping track of these probabilities, a request sequence can be constructed that incurs to ALG expected cost of at least  $H_k$  times the optimum cost.

The adversary will maintain a set of marked pages for the sequence it has generated so far like a marking algorithm would do (see previous lecture). Also, the input sequence will be divided in phases as described below. In the beginning of each phase the number of marked pages will be  $k$  and the number will vary during the phase. Specifically, a phase will be divided into subphases as follows:

- the *kick-start subphase*: it starts with a sequence of requests for pages that are marked and ends with a request of the unmarked page; after this request all pages that were marked become unmarked and the last page that was requested becomes marked; by keeping track of the probabilities  $\{p_x\}_x$  we will make sure that the expected cost of this subphase will be 1.
- the *marking subphases*, labelled 1 through  $k - 1$ : in the beginning of subphase  $i$ , exactly  $i$  pages are marked; the subphase begins with a sequence of requests for pages that are marked and ends with a request of one of the unmarked pages, which subsequently becomes marked as well; we will make sure that the expected cost of this subphase will be  $\frac{1}{k-i+1}$ .

If every phase of the request sequence has the above structure it is clear that every phase will incur expected cost of  $H_k$  to ALG. On the other hand, it is not hard to check that a marking algorithm (see previous lecture) would incur cost of 1 per phase for a sequence of requests divided into phases with the above properties. Therefore, to finish the proof, it remains to describe the structure of the subphases so that they satisfy the above properties.

Consider a situation in which the number of unmarked pages is  $q$ ,  $q \in \{1, \dots, k\}$ . The goal of the adversary is to generate some requests that cause ALG to incur an expected cost of at least  $1/q$  and decrease the number of unmarked pages to  $q - 1$  (except if  $q = 1$ , in which case the number of unmarked pages changes to  $k$ ), as described above. Let  $S$  be the set of marked pages, and let  $P = \sum_{x \in S} p_x$ ; then  $u = n - |S|$ . If  $P = 0$  then there must be an unmarked page  $x$  with  $p_x > 1/u$ . In this case, the subphase consists of a single request to  $x$ . The expected cost of this request is at least  $1/u$ . If  $P > 0$ , then there must be  $x \in S$  such that  $p_x > 0$ . Let  $\epsilon = p_x$ , and let the first request of the subphase be  $x$ . Next, a set of requests is generated by the following loop, where  $P$  denotes the current total probability of the marked pages:

- **while**  $P > \epsilon$  and the total expected cost of all the requests in this subphase so far does not exceed  $1/u$

request page  $x \in S$ , where  $p_x = \max_{j \in S}(p_j)$

Each iteration of this loop adds at least  $\epsilon/|S| > 0$  to the total expected cost of this subphase. Thus the loop must terminate. If the total expected cost ends up exceeding  $1/u$ , then an arbitrary request is made to an unmarked page, and the subphase is over. If the loop terminates with  $P \leq \epsilon$ , then a request is generated to the unmarked page  $j$  with the highest probability value. Note that  $p_j \geq (1 - P)/u$ . The following inequalities finish the proof:

$$\text{expected cost of the subphase} \geq \epsilon + p_j \geq \epsilon + \frac{1 - P}{u} \geq \epsilon + \frac{1 - \epsilon}{u} \geq \frac{1}{u}.$$

□

We have demonstrated a lower bound of  $H_k$  on the best possible competitive ratio of a randomized paging algorithm, and have shown that the randomized marking algorithm achieves  $2H_k$ . A much more complicated algorithm has been given that achieves the competitive ratio of  $H_k$  [1].

## 2 Introduction to the $k$ -Server Problem

**Definition 4** A metric space is a set of points  $M$  and a distance function  $d$  from  $M \times M$  into the nonnegative reals satisfying:

- $\forall x. d(x, x) = 0$
- $\forall x, y. d(x, y) = d(y, x)$
- $\forall x, y, z. d(x, y) \leq d(x, z) + d(z, y)$

**The  $k$ -Server Problem:** The  $k$ -server problem is defined as follows. There are  $k$  servers that respond to requests at points in the space. At a general step each of the  $k$  servers is located at a point in the space. Their *configuration* is described by a  $k$ -tuple  $(x_1, x_2, \dots, x_k)$  of points in the space. When a request  $r$  is received the servers move to a new configuration  $(y_1, y_2, \dots, y_k)$  such that  $r \in \{y_1, y_2, \dots, y_k\}$ , and the cost of the move is  $\sum_{i=1}^k d(x_i, y_i)$ . The off-line optimization problem is to serve a sequence  $\sigma = r_1 r_2 \dots r_n$  of requests at minimum cost, starting from a fixed initial configuration. An on-line algorithm must base its response to each request only on the requests already received, and its performance is measured by its competitive ratio.

**The Lazy Algorithm:** Any algorithm for the  $k$ -server problem can be transformed, without increasing its cost, into a *lazy algorithm* which, at each step, moves only one server to the request point, if that point is not already covered, while keeping the other servers stationary. The lazy algorithm simulates in some internal memory the moves that the original algorithm makes. Then, for each request, it finds —by looking at the simulation— which server the original algorithm would move to serve the request and moves precisely that server to the request-point —from whatever point that server is located. Thus, for every server, a sequence of moves that take place from the time the server serves a request to the time it serves a new request is replaced by the lazy algorithm by a single move from the point of the first request to the point of the second request. By the triangle inequality the total cost will not increase by doing so.

**The Greedy Algorithm:** In the *greedy algorithm*, at each step, the nearest server to the request moves to it, and the others are stationary. The following 3-point, 2-server example illustrates this. Here the metric is the Euclidean distance in one dimension. Suppose that the initial configuration



is  $(a, b)$  and the request sequence is  $cbababab\dots$ . The greedy algorithm would move the server from  $b$  to  $c$  at the first step and, to serve the subsequent requests, the other server would shuttle between  $a$  and  $b$ . On the other hand, after the first request is served, the optimal algorithm would place one server permanently at  $a$  and the other at  $b$ , incurring no further cost. The ratio of costs between the two algorithms tends to infinity.

Finally, here is a *lower bound* on the competitive ratio of any deterministic algorithm.

**Theorem 5** *In every metric space  $(M, d)$  such that  $|M| \geq k + 1$  and  $d(x, y) > 0$  whenever  $x \neq y$ , no deterministic  $k$ -server algorithm can achieve a competitive ratio less than  $k$ .*

PROOF: The adversary restricts itself to  $k + 1$  points. In each server configuration, exactly one of these points is uncovered. The adversary simulates the deterministic algorithm ALG and, at each step, requests the uncovered point. Each request is served by the point that is uncovered at the next step. Thus, if the request sequence is  $r_1, r_2, \dots, r_n$ , the cost to ALG is  $\sum_{i=1}^{n-1} d(r_i, r_{i+1})$ .

To finish the proof, we will show that the cost to OPT is at most  $\frac{1}{k} \sum_{i=1}^{n-1} d(r_i, r_{i+1}) + c$ , where  $c$  depends only on the metric space and not on the request sequence. Let us consider the parallel execution of  $k$  lazy algorithms, among which at least one will turn out to have cost at most  $\frac{1}{k} \sum_{i=1}^{n-1} d(r_i, r_{i+1}) + c$ :

1. The algorithms have different (unordered) initial configurations, which are the  $k$  possible configurations (sets) in which point  $r_1$  is covered (contained). Constant  $c$  will be the cost of assuming a specific initial configuration.
2. At each step  $i$ ,  $i \geq 2$ , of the request sequence, if the request point  $r_i$  is left uncovered in some of the executions, then in these executions that point will be served from the server at point  $r_{i-1}$  (there must be a server there) at cost  $d(r_{i-1}, r_i)$ .

The following property can be established by an easy induction:

**Lemma 6** *For all  $i \geq 2$ , at the beginning of step  $i$ , all algorithms cover point  $r_{i-1}$  and have different (unordered) configurations.*

From lemma 6, it follows that, in the beginning of each step, exactly one of the executions is found with the request point uncovered and, therefore, has to pay  $d(r_{i-1}, r_i)$  to get it served. Therefore, the total cost incurred by the algorithms we run in parallel is  $\sum_{i=1}^{n-1} d(r_i, r_{i+1})$ , which implies that there exists at least one algorithm with cost  $\leq \frac{1}{k} \sum_{i=1}^{n-1} d(r_i, r_{i+1}) + c$ .  $\square$

The following conjecture states that the above lower bound can be matched. In the following lecture we will see how the Koutsoupias-Papadimitriou algorithm [3] comes very close to resolving this conjecture.

**Conjecture 1** *In every metric space, there exists a deterministic  $k$ -server algorithm that is  $k$ -competitive.*

## References

- [1] Dimitris Achlioptas, Marek Chrobak and John Noga. “Competitive Analysis of Randomized Paging Algorithms,” *Theor. Comput. Sci.* 234 (1–2), pp. 203–218, 2000.
- [2] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator and Neal E. Young. “Competitive Paging Algorithms,” *J. Algorithms* 12 (4), pp. 685–699, 1991.
- [3] Elias Koutsoupias and Christos H. Papadimitriou. “On the  $k$ -Server Conjecture,” *J. ACM* 42 (5), pp. 971–983, 1995.