1 Online algorithms and competitive analysis

1.1 Basic definitions

Formally, many online problems can be described as follows. An online algorithm A is presented with a *request sequence* $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$. The algorithm A has to serve each request *online*, i.e., without knowledge of future requests. More precisely, when serving request $\sigma(t)$, $1 \leq t \leq m$, the algorithm does not know any request $\sigma(t')$ with $t' > t$. Serving requests incurs cost, and the goal is to serve the entire request sequence so that the total cost is as small as possible. This setting can also be regarded as a *request-answer* game: An adversary generates requests, and an online algorithm has to serve them one at a time.

In order to illustrate this formal model, we mention a concrete online problem.

The paging problem: Consider a two-level memory system that consists of a small fast memory and a large slow memory. Here, each request specifies a page in the memory system. A request is served if the corresponding page is in fast memory. If a requested page is not in fast memory, a *page fault* occurs. Then a page must be moved from fast memory to slow memory so that the requested page can be loaded into the vacated location. A paging algorithms specifies which page to evict on a fault. If the algorithm is online, then the decision which page to evict must be made without knowledge of any future requests. The cost to be minimized is the total number of page faults incurred on the request sequence.

Sleator and Tarjan [48] suggested to evaluate the performance on an online algorithm using *competitive analysis*. In a competitive analysis, an online algorithm A is compared to an *optimal offline algorithm*. An optimal offline algorithm knows the entire request

sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $C_A(\sigma)$ denote the cost incurred by A and let $C_{OPT}(\sigma)$ denote the cost paid by an optimal offline algorithm OPT. The algorithm A is called c-competitive if there exists a constant a such that

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$$

for all request sequences σ. Here we assume that A is a deterministic online algorithm. The factor c is also called the *competitive ratio* of A.

1.2 Results on deterministic paging algorithms

We list three well-known deterministic online paging algorithms.

- **LRU** (Least Recently Used): On a fault, evict the page in fast memory that was requested least recently.
- **FIFO** (First-In First-Out): Evict the page that has been in fast memory longest.
- **LFU** (Least Frequently Used): Evict the page that has been requested least frequently.

Before analyzing these algorithms, we remark that Belady [12] exhibited an optimal offline algorithm for the paging problem. The algorithm is called MIN and works as follows.

- **MIN**: On a fault, evict the page whose next request occurs furthest in the future.

Belady showed that on any sequence of requests, MIN achieves the minimum number of page faults.

Throughout these notes, when analyzing paging algorithms, we denote by k the number of pages that can simultaneously reside in

fast memory. It is not hard to see that the algorithm LFU is not competitive. Sleator and Tarjan [48] analyzed the algorithms LRU and FIFO and proved the following theorem.

Theorem 1 *The algorithms LRU and FIFO are k-competitive.*

Proof: We will show that LRU is k-competitive. The analysis for FIFO is very similar. Consider an arbitrary request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$. We will prove that $C_{LRU}(\sigma) \leq k \cdot C_{OPT}(\sigma)$. Without loss of generality we assume that LRU and OPT initially start with the same fast memory.

We partition σ into phases $P(0), P(1), P(2), \ldots$ such that LRU has at most k fault on $P(0)$ and exactly k faults on $P(i)$, for every $i \geq 1$. Such a partitioning can be obtained easily. We start at the end of σ and scan the request sequence. Whenever we have seen k faults made by LRU, we cut off a new phase. In the remainder of this proof we will show that OPT has at least one page fault during each phase. This establishes the desired bound.

For phase $P(0)$ there is nothing to show. Since LRU and OPT start with the same fast memory, OPT has a page fault on the first request on which LRU has a fault.

Consider an arbitrary phase $P(i)$, $i \geq 1$. Let $\sigma(t_i)$ be the first request in $P(i)$ and let $\sigma(t_{i+1} - 1)$ be the last request in $P(i)$. Furthermore, let p be the page that is requested last in $P(i-1)$.

Lemma 1 *$P(i)$ contains requests to k distinct pages that are different from p.*

If the lemma holds, then OPT must have a page fault in $P(i)$. OPT has page p in its fast memory at the end of $P(i-1)$ and thus cannot have all the other k pages request in $P(i)$ in its fast memory.

It remains to prove the lemma. The lemma clearly holds if the k requests on which LRU has a fault are to k distinct pages and if

these pages are also different from p. So suppose that LRU faults twice on a page q in $P(i)$. Assume that LRU has a fault on $\sigma(s_1) = q$ and $\sigma(s_2) = q$, with $t_i \leq s_1 < s_2 \leq t_{i+1} - 1$. Page q is in LRU's fast memory immediately after $\sigma(s_1)$ is served and is evicted at some time t with $s_1 < t < s_2$. When q is evicted, it is the least recently requested page in fast memory. Thus the subsequence $\sigma(s_1), \ldots, \sigma(t)$ contains requests to $k+1$ distinct pages, at least k of which must be different from p.

Finally suppose that within $P(i)$, LRU does not fault twice on page but on one of the faults, page p is request. Let $t \geq t_i$ be the first time when p is evicted. Using the same arguments as above, we obtain that the subsequence $\sigma(t_i - 1), \sigma(t_i), \ldots, \sigma(t)$ must contain $k+1$ distinct pages. □

The next theorem is also due to Sleator and Tarjan [48]. It implies that LRU and FIFO achieve the best possible competitive ratio.

Theorem 2 *Let A be a deterministic online paging algorithm. If A is c-competitive, then $c \geq k$.*

Proof: Let $S = \{p_1, p_2, \ldots, p_{k+1}\}$ be a set of $k+1$ arbitrary pages. We assume without loss of generality that A and OPT initially have $p_1, \ldots, p_k$ in their fast memories.

Consider the following request sequence. Each request is made to the page that is not in A's fast memory.

Online algorithm A has a page fault on every request. Suppose that OPT has a fault on some request $\sigma(t)$. When serving $\sigma(t)$, OPT can evict a page is not requested during the next $k-1$ requests $\sigma(t+1), \ldots, \sigma(t+k-1)$. Thus, on any k consecutive requests, OPT has at most one fault. □

The competitive ratios shown for deterministic paging algorithms are not very meaningful from a practical point of view. Note that the performance ratios of LRU and FIFO become worse as the size of

the fast memory increases. However, in practice, these algorithms perform better the bigger the fast memory is. Furthermore, the competitive ratios of LRU and FIFO are the same, whereas in practice LRU performs much better. For these reasons, there has been a study of competitive paging algorithms with *access graphs* [23, 36]. In an access graph, each node represents a page in the memory system. Whenever a page p is requested, the next request can only be to a page that is adjacent to p in the access graph. Access graphs can model more realistic request sequences that exhibit locality of reference. It was shown [23, 36] that using access graphs, one can overcome some negative aspects of conventional competitive paging results.

2 Randomization in online algorithms

2.1 General concepts

The competitive ratio of a randomized online algorithm A is defined with respect to an adversary. The adversary generates a request sequence σ and it also has to serve σ. When constructing σ, the adversary always knows the description of A. The crucial question is: When generating requests, is the adversary allowed to see the outcome of the random choices made by A on previous requests?

Ben-David *et al.* [17] introduced three kinds of adversaries.

- **Oblivious Adversary**: The oblivious adversary has to generate a complete request sequence in advance, before any requests are served by the online algorithm. The adversary is charged the cost of the optimum offline algorithm for that sequence.
- **Adaptive Online Adversary**: This adversary may observe the online algorithm and generate the next request based on

the algorithm's (randomized) answers to all previous requests. The adversary must serve each request online, i.e., without knowing the random choices made by the online algorithm on the present or any future request.

- **Adaptive Offline Adversary**: This adversary also generates a request sequence adaptively. However, it is charged the optimum offline cost for that sequence.

A randomized online algorithm A is called c-competitive against any oblivious adversary if there is a constant a such for all request sequences σ generated by an oblivious adversary, $E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma) + a$. The expectation is taken over the random choices made by A.

Given a randomized online algorithm A and an adaptive online (adaptive offline) adversary ADV, let $E[C_A]$ and $E[C_{ADV}]$ denote the expected costs incurred by A and ADV in serving a request sequence generated by ADV. A randomized online algorithm A is called c-competitive against any adaptive online (adaptive offline) adversary if there is a constant a such that for all adaptive online (adaptive offline) adversaries ADV, $E[C_A] \leq c \cdot E[C_{ADV}] + a$, where the expectation is taken over the random choices made by A.

Ben-David *et al.* [17] investigated the relative strength of the adversaries with respect to an arbitrary online problem and showed the following statements.

Theorem 3 *If there is a randomized online algorithm that is c-competitive against any adaptive offline adversary, then there also exists a c-competitive deterministic online algorithm.*

This theorem implies that randomization does not help against the adaptive offline adversary.

Theorem 4 *If A is a c-competitive randomized algorithm against any adaptive online adversary, and if there is a d-competitive algorithm against any oblivious adversary, then A is $(c \cdot d)$-competitive against any adaptive offline adversary.*

An immediate consequence of the above two theorems in the following corollary.

Corollary 1 *If there exists a c-competitive randomized algorithm against any adaptive online adversary, then there is a c^2-competitive deterministic algorithm.*

2.2 Randomized paging algorithms against oblivious adversaries

We will prove that, against oblivious adversaries, randomized online paging algorithms can considerably beat the ratio of k shown for deterministic paging. The following algorithm was proposed by Fiat *et al.* [27].

Algorithm MARKING: The algorithm processes a request sequence in phases. At the beginning of each phase, all pages in the memory system are unmarked. Whenever a page is requested, it is *marked*. On a fault, a page is chosen uniformly at random from among the unmarked pages in fast memory, and this pages is evicted. A phase ends when all pages in fast memory are marked and a page fault occurs. Then, all marks are erased and a new phase is started.

Fiat *et al.* [27] analyzed the performance of the MARKING algorithm.

Theorem 5 *The MARKING algorithm is $2H_k$-competitive against any oblivious adversary, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th Harmonic number.*

Note that H_k is roughly $\ln k$. Later, in Section 3.2, we will see that no randomized online paging algorithm against any oblivious adversary can be better than H_k-competitive. Thus the MARKING algorithm is optimal, up to a constant factor. More complicated paging algorithms achieving an optimal competitive ratio of H_k were given in [42, 1].

Proof: Given a request sequence $\sigma = \sigma(1), \ldots, \sigma(m)$, we assume without of generality that MARKING already has a fault on the first request $\sigma(1)$.

MARKING divides the request sequence into phases. A phase starting with $\sigma(i)$ ends with $\sigma(j)$, where j, $j > i$, is the smallest integer such that the set

$$\{\sigma(i), \sigma(i+1), \ldots, \sigma(j+1)\}$$

contains $k+1$ distinct pages. Note that at the end of a phase all pages in fast memory are marked.

Consider an arbitrary phase. Call a page *stale* if it is unmarked but was marked in the previous phase. Call a page *clean* if it is neither stale nor marked.

Let c be the number of clean pages requested in the phase. We will show that

1. the amortized number of faults made by OPT during the phase it at least $\frac{c}{2}$.
2. the expected number of faults made by MARKING is at most cH_k.

These two statements imply the theorem.

We first analyze OPT's cost. Let S_{OPT} be the set of pages contained in OPT's fast memory, and let S_M be the set of pages stored in MARKING's fast memory. Furthermore, let d_I be the value of

$|S_{OPT} \setminus S_M|$ at the beginning of the phase and let d_F be the value of $|S_{OPT} \setminus S_M|$ at the end of the phase. OPT has at least $c - d_I$ faults during the phase because at least $c - d_I$ of the c clean pages are not in OPT's fast memory. Also, OPT has at least d_F faults during the phase because d_F pages requested during the phase are not in OPT's fast memory at the end of the phase. We conclude that OPT incurs at least

$$\max\{c - d_I, d_F\} \geq \frac{1}{2}(c - d_I + d_F) = \frac{c}{2} - \frac{d_I}{2} + \frac{d_F}{2}$$

faults during the phase. Summing over all phases, the terms $\frac{d_I}{2}$ and $\frac{d_F}{2}$ telescope, except for the first and last terms. Thus the amortized number of page faults made by OPT during the phase is at least $\frac{c}{2}$.

Next we analyze MARKING's expected cost. Serving c requests to clean pages cost c. There are $s = k - c \leq k - 1$ requests to stale pages. For $i = 1, \ldots, s$, we compute the expected cost of the i-th request to a stale page. Let $c(i)$ be the number of clean pages that were requested in the phase immediately before the i-th request to a stale page and let $s(i)$ denote the number of stale pages that remain before the i-th request to a stale page.

When MARKING serves the i-th request to a stale page, exactly $s(i) - c(i)$ of the $s(i)$ stale pages are in fast memory, each of them with equal probability. Thus the expected cost of the request is

$$\frac{s(i) - c(i)}{s(i)} \cdot 0 + \frac{c(i)}{s(i)} \cdot 1 \leq \frac{c}{s(i)} = \frac{c}{k - i + 1}.$$

The last equation follows because $s(i) = k - (i - 1)$. The total expected cost for serving requests to stale pages is

$$\sum_{i=1}^{s} \frac{c}{k + 1 - i} \leq \sum_{i=2}^{k} \frac{c}{i} = c(H_k - 1).$$

We conclude that MARKING's total expected cost in the phase is bounded by cH_k. □