Algorithms

Instructor –
Dimitrios Katsaros

Basic definitions

- 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
    - 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 s *request-answer* game: an *adversary* generates requests, and an online algorithm has to serve them one at a time

The paging problem

Paging problem description

- Consider a two-level memory system: a small fast memory and a large slow memory
  - Each request specifies a page in the memory system
- A request is served, iff the corresponding page is in fast memory
  - If it is not, then a *page fault* occurs
  - Then, the page is elevated from slow memory
  - A paging algorithm (i.e., replacement algorithm) specifies which page to evict in order to accommodate the requested page
- If the algorithm is online, then the replacement decision 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

Competitive analysis

- An online algorithm A is compared to an *optimal offline algorithm*
- An optimal offline algorithm knows the entire request in advance and can server 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 *α* such that

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

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

Deterministic paging algorithms

- **LRU**: on a fault, evict the page in fast memory that was requested least recently
- **FIFO**: on a fault, evict the page that has been inn fast memory longest
- **LFU**: on a fault, evict the page that has been requested least frequently
- **FF/Belady/MIN/OPT**: on a fault, evict the page whose next request occurs furthest in the future

- In this lecture, we assume *k* is the number of pages that can simultaneously reside in fast memory

LRU competitiveness

Theorem 1. *The algorithm LRU (and FIFO) is c-competitive*.

Proof. Consider an arbitrary request sequence $\sigma=\sigma(1), \sigma(2), ..., \sigma(m)$. We will prove that $C_{LRU}(\sigma) \leq k \cdot C_{OPT}(\sigma)$. WLOG we assume that LRU and OPT initially start with the same fast memory.
We partition σ into phases P(0), P(1), P(2),… such that LRU has at most k faults 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 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.

LRU competitiveness

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 can not 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 the LRU has a fault on $\sigma(s_1) = q$ and $\sigma(s_2) = q$ with $t_i \leq s_1 < s_2 \leq t_{i+1} - 1$.

LRU competitiveness

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 $\sigma(s_1), \ldots, \sigma(s_t)$ contains requests to k+1 distinct pages, at least k of which must be difference from p.

Finally, suppose that within P(i), LRU does not fault twice on page but on one of the faults, page p is requested.

Let $t \geq t_i$ be the first time when p is evicted. Using the same arguments as above, we obtain that the subsequence $\sigma(t_i\text{-}1), \sigma(t_i), \ldots, \sigma(t)$ must contain k+1 distinct pages. ↵

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

Proof. Let $S=\{p_1,p_2,\ldots,p_{k+1}\}$ be a set of k+1 arbitrary pages. We assume WLOG that A and OPT initially have $p_1,p_2,\ldots,p_k$ 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 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. ↵

This theorem implies that LRU (and FIFO) achieve the best possible competitive ratio.

Comments

- The competitive ratios shown are not very meaningful from a practical point of view
- Note that the competitive ratio of LRU (and FIFO) become worse as the size of the fast memory increases!
- In practice, these algorithms perform better the bigger the fast memory is
- The competitive ratio of LRU (and FIFO) are the same, whereas in practice LRU performs much better

Proof techniques: Potential functions

Potential functions

- Given a request sequence $\sigma=\sigma(1),\sigma(2),\ldots\sigma(m)$ and a potential function Φ, the amortized online cost on request $\sigma(t)$, $1 \le t \le m$, is defined as $C_A(t) + \Phi(t) - \Phi(t-1)$
- $\Phi(t)$ is the value of the potential function after request $\sigma(t)$, i.e., $\Phi(t) - \Phi(t-1)$ is the change in potential that occurs during the processing of $\sigma(t)$
- In an amortized analysis using a potential function we usually show that for any request $\sigma(t)$

$$C_A(t) + \Phi(t) - \Phi(t-1) \le c \cdot C_{OPT}(t)$$

- If we can prove this inequality for all t, then it is easy to see that A is c-competitive. Summing up the previous inequality for all t:

$$\sum_{t=1}^{m} C_A(t) + \Phi(\mathrm{m}) - \Phi(0) \le c \sum_{t=1}^{m} C_{OPT}(t)$$

where $\Phi(0)$ is the initial potential

Potential functions

- Typically a potential function is chosen such that Φ is always non-negative and such that the initial potential is 0. Using these two properties, we obtain from the previous inequality the desired property: $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma)$
- The difficult part in a competitive analysis using a potential function is to construct Φ and show the inequality for all requests

LRU k-competitiveness using potential functions

- Let $\sigma=\sigma(1), \ldots, \sigma(m)$ be an arbitrary request sequence
- At any time let S_{LRU} be the set of pages contained in LRU's fast memory
- Let S_{OPT} be the set of pages contained in OPT's fast memory
- Set $S= S_{LRU} \setminus S_{OPT}$
- Assign integer weights from the range [1…k] to the pages in S_{LRU} such that for any two pages $p,q \in S_{LRU}$, $w(p)<w(q)$ iff the last request to p occurs earlier than the last request to q
- Let

$$\Phi = \sum_{p \in S} w(p)$$

- Consider an arbitrary request $\sigma(t)=p$ and assume WLOG that OPT serves the request first and that LRU serves second

LRU k-competitiveness using potential functions

- If OPT does not have a page fault on σ(t), then its cost is 0 and the potential does not change
- If OPT does have a page fault on σ(t), then its cost is 1
- OPT might evict a page that is in LRU's fast memory, in which case the potential increases
  - However, the potential can increase by at most k
- Next suppose that LRU does not have a fault on σ(t). Then, its cost is 0, and the potential can not change
- If LRU has a page fault, its cost on the request is 1
- We show that the potential decreases by at least 1
- Immediately before LRU serves σ(t), page p is only in OPT's fast memory
- By symmetry, there must be a page that is only in LRU's fast memory, i.e., there must exists a page q ϵ S

LRU k-competitiveness using potential functions

- If q is evicted by LRU during the operation, then the potential decreases by $w(q) \geq 1$
- Otherwise, since p is loaded into fast memory, the weight of q must decrease by 1, and thus the potential must decrease by 1

- In symmetry we have shown:
- ✓ Every time OPT has a fault, the potential increases by at most k
- ✓ Every time LRU has a fault, the potential decreases by at least 1
- ✓ Therefore, we conclude that the following must hold:

$$C_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k \cdot C_{OPT}(t)$$