Chapter 9
Online Algorithms

Algorithm Theory
WS 2019/20

Fabian Kuhn

Online Computations

- Sometimes, an algorithm has to start processing the input before the complete input is known

- For example, when storing data in a data structure, the sequence of operations on the data structure is not known

Online Algorithm: An algorithm that has to produce the output step-by-step when new parts of the input become available.

Offline Algorithm: An algorithm that has access to the whole input before computing the output.

- Some problems are inherently online
  - Especially when real-time requests have to be processed over a significant period of time

Competitive Ratio

- Let's again consider optimization problems
  - For simplicity, assume, we have a minimization problem

Optimal offline solution $\mathbf{OPT}(\boldsymbol{I})$:

- Best objective value that an offline algorithm can achieve for a given input sequence I

Online solution $\mathbf{ALG}(\boldsymbol{I})$:

- Objective value achieved by an online algorithm ALG on I

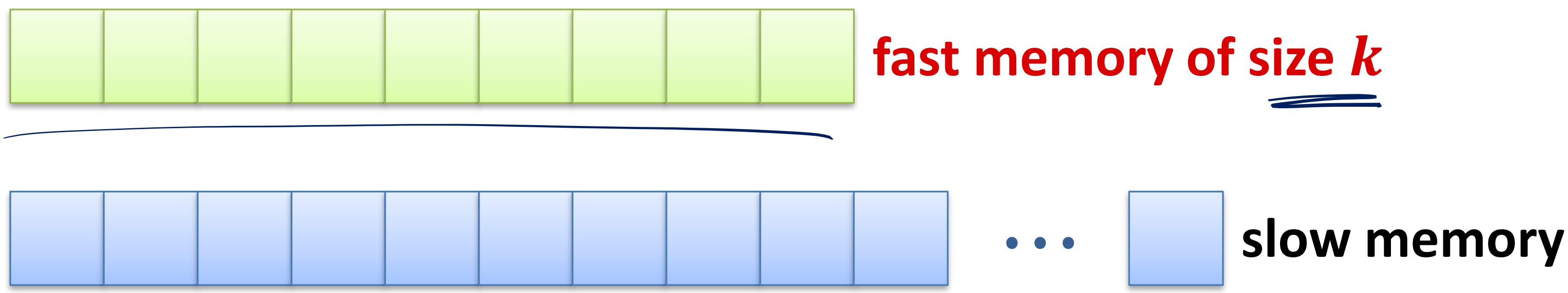
Competitive Ratio: An algorithm has competitive ratio $c \geq 1$ if

$$\mathbf{ALG}(\boldsymbol{I}) \leq \boldsymbol{c} \cdot \mathbf{OPT}(\boldsymbol{I}) + \boldsymbol{\alpha}.$$

- If $\alpha = 0$, we say that ALG is strictly c-competitive.

Paging Algorithm

Assume a simple memory hierarchy:



If a memory page has to be accessed:

- Page in fast memory (hit): take page from there
- Page not in fast memory (miss): leads to a page fault
- Page fault: the page is loaded into the fast memory and some page has to be evicted from the fast memory
- Paging algorithm: decides which page to evict
- Classical online problem: we don't know the future accesses

Paging Strategies

k-competitive

Least Recently Used (LRU):

- Replace the page that hasn't been used for the longest time

First In First Out (FIFO):

- Replace the page that has been in the fast memory longest

Last In First Out (LIFO):

- Replace the page most recently moved to fast memory

Least Frequently Used (LFU):

- Replace the page that has been used the least

Longest Forward Distance (LFD):

- Replace the page whose next request is latest (in the future)
- LFD is **not** an online strategy!

LRU and FIFO Algorithms

Lemma: Algorithm LFD has at least one page fault in each phase i interval (for $i = 1, \dots, p-1$, where p is the number of phases).

Corollary: The number of page faults of an optimal offline algorithm is at least $p-1$, where p is the number of phases

Theorem: The LRU and the FIFO algorithms both have a competitive ratio of at most k.

Proof:

- We will show that both have at most k page faults per phase
- We then have (for every input I):

$$\mathrm{LRU}(I), \mathrm{FIFO}(I) \le k \cdot p \le k \cdot \mathrm{OPT}(I) + k$$

Lower Bound

Theorem: Even if the slow memory contains only $k + 1$ pages, any deterministic algorithm has competitive ratio at least k.

Proof:

- Consider some given deterministic algorithm ALG
- Because ALG is deterministic, the content of the fast memory after the first i requests is determined by the first i requests.
- Construct a request sequence inductively as follows:
  - Assume some initial slow memory content
  - The $(i + 1)^{\text{st}}$ request is for the page which is not in fast memory after the first i requests (throughout we only use $k + 1$ different pages)
- There is a page fault for every request
- OPT has a page fault at most every k requests
  - There is always a page that is not required for the next $k - 1$ requests

Randomized Algorithms

- We have seen that deterministic paging algorithms cannot be better than k-competitive

- Does it help to use randomization?

Competitive Ratio: A randomized online algorithm has competitive ratio $c \geq 1$ if for all inputs I,

$$\mathbb{E}[\mathbf{ALG}(\boldsymbol{I})] \leq \boldsymbol{c} \cdot \mathbf{OPT}(\boldsymbol{I}) + \boldsymbol{\alpha}.$$

- If $\alpha \leq 0$, we say that ALG is strictly c-competitive.

Adversaries

- For randomized algorithm, we need to distinguish between different kinds of adversaries (providing the input)

Oblivious Adversary:

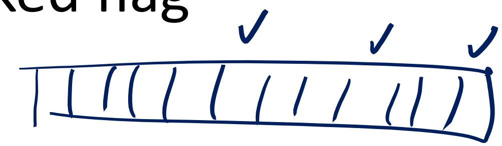
- Has to determine the complete input sequence before the algorithm starts
  - The adversary cannot adapt to random decisions of the algorithm

Adaptive Adversary:

- The input sequence is constructed during the execution
- When determining the next input, the adversary knows how the algorithm reacted to the previous inputs
- Input sequence depends on the random behavior of the alg.
- Sometimes, two adaptive adversaries are distinguished
  - offline, online : different way of measuring the adversary cost

The Randomized Marking Algorithm

- Every entry in fast memory has a marked flag



k

- Initially, all entries are unmarked.
- If a page in fast memory is accessed, it gets marked
- When a page fault occurs:
  - If all k pages in fast memory are marked, all marked bits are set to 0
  - The page to be evicted is chosen uniformly at random among the unmarked pages
  - The marked bit of the new page in fast memory is set to 1

Phase Partition

We partition a given request sequence σ into phases as follows:

- **Phase $\mathbf{0}$**: empty sequence
- **Phase $\boldsymbol{i}$** : maximal sequence that immediately follows phase $i-1$ and contains at most k distinct page requests

Example sequence ($\boldsymbol{k=4}$):

$$|2, 5, 12, 5, 4, 2, |10, 8, 3, 6, |2, 2, 6, 6, 8, 3, 2, 6, |9, 10, 6, 3, 10, |2, 1, 3, 5|$$

Example

Input Sequence (k=6):

2, 5, 3, 3, 6, 8, 2, 9, 5 | 7, 1, 2, 5, 2, 3, 7, 4 | 8, 1, 2, 7, 5,3,6,9,6,10,4,1,2 ...

phase 1 phase 2 phase 3 phase 4

Fast Memory:

10	6	1	9	4	2

Observations:

- At the end of a phase, the fast memory entries are exactly the k pages of that phase
- At the beginning of a phase, all entries get unmarked
- #page faults depends on #new pages in a phase

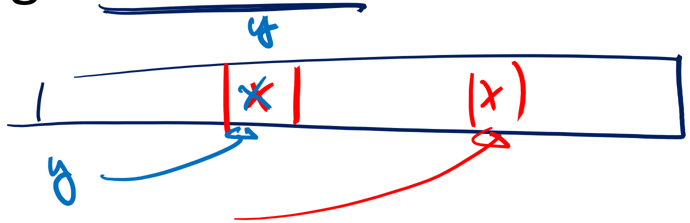
Page Faults per Phase

Consider a fixed phase i:

- Assume that of the k pages of phase i, m_i are new and $k - m_i$ are old (i.e., they already appear in phase $i - 1$)

- All m_i new pages lead to page faults (when they are requested for the first time)

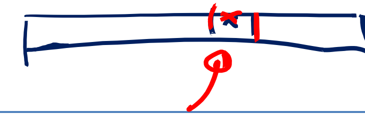
- When requested for the first time, an old page leads to a page fault, if the page was evicted in one of the previous page faults



- We need to count the number of page faults for old pages

Page Faults per Phase

m_i new pages
$k - m_i$ old pages



Phase i, j^{th} old page that is requested (for the first time):

- There is a page fault if the page has been evicted
- There have been at most $m_i + j - 1$ distinct requests before
- The old places of the $j - 1$ first old pages are occupied (marked)
- The other $\leq m_i$ pages are at uniformly random places among the remaining $k - (j - 1)$ places (oblivious adv.)
- Probability that the old place of the j^{th} old page is taken:

$$\leq \frac{m_i}{k - (j - 1)}$$

Page Faults per Phase

$F_{i,j} := \begin{cases} 1 & \text{if } j^{\text{th}} \text{ old page} \to \text{page fault} \\ 0 & \text{otherwise} \end{cases}$

Phase $i > 1$, j^{th} old page that is requested (for the first time):

- Probability that there is a page fault:

$$\leq \frac{m_i}{k-(j-1)}$$

$F_i = \sum_{j=1}^{k-m_i} F_{i,j}$

$\mathbb{E}[F_i] = \sum_{j=1}^{k-m_i} \mathbb{E}[F_{i,j}]$

Number of page faults for old pages in phase i: F_i

$$\mathbb{E}[F_i] = \sum_{j=1}^{k-m_i} \mathbb{P}\big(j^{\text{th}} \text{ old page incurs page fault}\big)$$

$$\leq \sum_{j=1}^{k-m_i} \frac{m_i}{k-(j-1)} = m_i \cdot \sum_{\ell=m_i+1}^{k} \frac{1}{\ell}$$

$$= m_i \cdot \big(H(k) - H(m_i)\big) \leq m_i \cdot (H(k) - 1)$$

$\sum_{\ell=1}^{k} \frac{1}{\ell} = H(k)$

$m_i \geq 1$

exp. page faults in phase i:

$\leq m_i \cdot H(k)$

Competitive Ratio

Theorem: Against an oblivious adversary, the randomized marking algorithm has a competitive ratio of at most $2H(k) \leq 2\ln(k) + 2$.

Proof:

- Assume that there are p phases

old pages new pages

- #page faults of rand. marking algorithm in phase i: $F_i + m_i$
- We have seen that

$$\mathbb{E}[F_i] \leq m_i \cdot (H(k) - 1) \leq m_i \cdot \ln(k)$$

- Let F be the total number of page faults of the algorithm:

$$\mathbb{E}[F] \leq \sum_{i=1}^{p} (\underbrace{\mathbb{E}[F_i] + m_i}_{\leq m_i \cdot H(k)}) \leq H(k) \cdot \sum_{i=1}^{p} m_i$$

Competitive Ratio

Theorem: Against an oblivious adversary, the randomized marking algorithm has a competitive ratio of at most $2H(k) \leq 2\ln(k) + 2$.

Proof:

- Let F_i^* be the number of page faults in phase i in an opt. exec.
- Phase 1: m_1 pages have to be replaced → $F_1^* \geq m_1$
- Phase $i > 1$:

  $i-1$ i

  k m_i

  - Number of distinct page requests in phases $i-1$ and i: $\boldsymbol{k + m_i}$
  - Therefore, $\boldsymbol{F_{i-1}^* + F_i^* \geq m_i}$
- Total number of page requests F^*:

$$F^* = \sum_{i=1}^{p} F_i^* \geq \frac{1}{2} \cdot \left(\underbrace{F_1^*}_{\geq m_1} + \sum_{i=2}^{p} \underbrace{\left(F_{i-1}^* + F_i^*\right)}_{\geq m_i} \right) \geq \frac{1}{2} \cdot \sum_{i=1}^{p} m_i$$

Competitive Ratio

Theorem: Against an oblivious adversary, the randomized marking algorithm has a competitive ratio of at most $2H(k) \leq 2\ln(k) + 2$.

Proof:

- Randomized marking algorithm:

$$\mathbb{E}[F] \leq H(k) \cdot \sum_{i=1}^{p} m_i$$

- Optimal algorithm:

$$F^* \geq \frac{1}{2} \cdot \sum_{i=1}^{p} m_i$$

Remark: It can be shown that no randomized algorithm has a competitive ratio better than $H(k)$ (against an obl. adversary)

Randomized Lower Bound

Yao's Principle (more precisely Yao's Minimax Principle):

exp. cost of best randomized alg. for worst-case input

~~=~~ $\geq$

exp. cost of best deterministic alg. for a given random input distr.

Proving a lower bound using Yao's principle:

- Design a random input distribution
- Show that every deterministic algorithm has a bad expected competitive ratio if the input is chosen at random according to this distribution
- Yao's principle then implies that every randomized algorithm is at least equally bad for worst-case input
  - worst-case fixed input: holds even for oblivious adversary

Randomized Paging Lower Bound

Input Distribution

- There are $k + 1$ different pages in the slow memory
- In each step, a uniformly random page is requested

Deterministic Online Algorithms

- Consider some request i
  - Current state of the fast memory depends on requests $i - 1$ and on the algorithm, assume that page p is **not** in fast memory
  - $\mathbb{P}(\text{page fault}) = \mathbb{P}(\text{request for page } p) = \frac{1}{k+1}$
- Expected #page faults after n requests:

$$\frac{n}{k+1}$$

Randomized Paging Lower Bound

Best Offline Algorithm: Longest Forward Distance

- After each page fault, optimal offline algorithm loads the page that will not be used for the longest possible time
- After a page fault, all $k+1$ pages are requested at least once before the next page fault

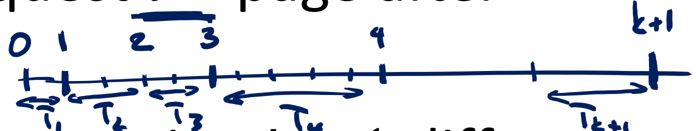
time between two page faults = time to request each page at least once $\cancel{-1}$

Claim: If T = time to request each page once, then

coupon collector process

$$\mathbb{E}[T] = (k+1)\cdot H(k+1)$$

- For $i \in \{0,\dots,k+1\}$: T_i time to request i^{th} page after requesting $i-1$ different pages



- Probability for req. i^{th} page after requesting $i-1$ diff. pages:

$$p_i = \frac{k+1-(i-1)}{k+1}$$

$$T = \sum_{i=1}^{k+1} T_i$$

Randomized Paging Lower Bound $\frac{n}{k+1}$

Claim: If T = time to request each page once, then

$$\mathbb{E}[T] = (k+1) \cdot H(k+1)$$

- For $i \in \{0, \dots, k+1\}$: T_i time to request i^{th} page after requesting $i-1$ different pages $T_i \sim Geom(p_i)$
- Prob. for req. i^{th} page after req. $i-1$ diff. pages: $p_i = \frac{k+1-(i-1)}{k+1}$

$$T = T_1 + T_2 + \dots + T_{k+1}$$

$$T_i \sim Geom(p_i) \qquad E[T_i] = \frac{1}{p_i} = \frac{k+1}{k+1-(i-1)} = (k+1) \cdot \frac{1}{k+1-(i-1)}$$

$$E[T] = \sum_{i=1}^{k+1} E[T_i] = (k+1) \cdot \sum_{i=1}^{k+1} \frac{1}{k+1-(i-1)} = (k+1) \cdot \sum_{j=1}^{k+1} \frac{1}{j}$$

lin. of exp.

$$= \underline{(k+1) H(k+1)}$$

Randomized Paging Lower Bound

Claim: For $k+1$ pages and n uniformly random requests, the optimal expected number of page faults is at ~~most~~ least

$$\frac{n}{(k+1)\cdot H(k+1)} - 1$$

- Average time $\bar{T}$ between page faults

$$\mathbb{E}[\bar{T}] = \mathbb{E}[T] \cancel{/k} = (k+1)H(k+1) \cancel{/k}$$

- Number of page faults $X = \lfloor n/\bar{T} \rfloor$:

$$\mathbb{E}[X] = \mathbb{E}\left[\left\lfloor \frac{n}{\bar{T}} \right\rfloor\right] \geq \mathbb{E}\left[\frac{n}{\bar{T}}\right] - 1 \geq \frac{n}{\mathbb{E}[\bar{T}]} - 1$$

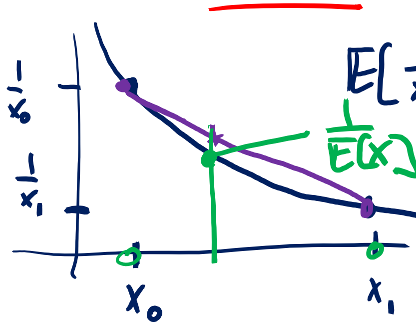
$$\mathbb{E}\left[\frac{1}{\bar{T}}\right] \geq \frac{1}{\mathbb{E}[\bar{T}]}$$

pos. rand. var. X

$X > 0$

$$\mathbb{E}\left[\frac{1}{X}\right] \geq \frac{1}{\mathbb{E}(X)}$$

Jensen's ineq.



$$\mathbb{E}\left[\frac{1}{X}\right] = P(X=x_0)\cdot\frac{1}{x_0} + P(X=x_1)\,\frac{1}{x_1}$$

Randomized Paging Lower Bound

Theorem: Every randomized paging algorithm has competitive ratio at least $H(k)$ even for an oblivious adversary.

1. Assume we $k + 1$ pages and uniformly random page requests
2. Expected number of page faults of best deterministic algorithm
$$= \frac{n}{k+1}$$
3. Expected number of page faults of optimal algorithm
$$\geq \frac{n}{(k+1) \cdot H(k+1)} - 1$$
4. Yao's principle now proves the theorem
   – not really necessary here, step 2 also works directly for randomized alg.

Self-Adjusting Lists

- Linked lists are often inefficient
  - Cost of accessing an item at position i is linear in i
- But, linked lists are extremely simple
  - And therefore nevertheless interesting
- Can we at least improve the behavior of linked lists?
- In practical applications, not all items are accessed equally often and not equally distributed over time
  - The same items might be used several times over a short period of time
- **Idea:** rearrange list after accesses to optimize the structure for future accesses
- **Problem:** We don't know the future accesses
  - The list rearrangement problems is an online problem!

Model

- Only find operations (i.e., access some item)
  - Let's ignore insert and delete operations
  - Results can be generalized to cover insertions and deletions

Cost Model:

- Accessing item at position i costs i
- The only operation allowed for rearranging the list is swapping two adjacent list items
- Swapping any two adjacent items costs 1

Rearranging The List

Frequency Count (FC):

- For each item keep a count of how many times it was accessed
- Keep items in non-increasing order of these counts
- After accessing an item, increase its count and move it forward past items with smaller count

Move-To-Front (MTF):

- Whenever an item is accessed, move it all the way to the front

Transpose (TR):

- After accessing an item, swap it with its predecessor

Cost

Cost when accessing item at position i:

- Frequency Count (FC): between i and $2i - 1$
- Move-To-Front (MTF): $2i - 1$
- Transpose (TR): $i + 1$

Random Accesses:

- If each item x has an access probability p_x and the items are accessed independently at random using these probabilities, FC and TR are asymptotically optimal

Real access patterns are not random, TR usually behaves badly and the much simpler MTF often beats FC

Move-To-Front

- We will see that MTF is competitive
- To analyze MTF we need competitive analysis and amortized analysis

Operation k:

- Assume, the operation accesses item x at position i
- c_k: actual cost of the MTF algorithm

$$c_k = 2i - 1$$

- a_k: amortized cost of the MTF algorithm
- c_k^*: actual cost of an optimal offline strategy
  - Let's call the optimal offline strategy OPT