UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Faculty of Computer Science, Electrical Engineering and Mathematics

Algorithms and Complexity research group

Jun.-Prof. Dr. Alexander Skopalik

Online Algorithms

Notes of the lecture SS13

by Vanessa Petrausch (vape@mail.upb.de)

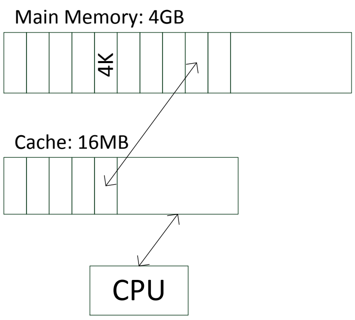
Contents

1 Introduction

Definition 1.1. "classical" optimization problem
given input instance $\rightarrow$ compute solution that max-/minimizes object function, e.g. shortest path

Definition 1.2. Online problem

- instance is not shown in advance
- revealed step by step
- decision (part of solution) have to be made each step, e.g. paging/caching



Definition 1.3. Optimisation problem II

- I_π set of instances
- For each $\sigma \in I_\pi$ there is
  - set of solutions S_σ
  - objective functions $f_\sigma : S_\sigma \rightarrow \mathbb{R}_{\geq 0}$
  - min/max
- OPT(a) value of optimal solution
- $A(\sigma)$ solution computed by algorithm A
- $w_A(\sigma) = f_\sigma(A(\sigma))$ value of $A's$ solution

Online Optimization Problem

- Input is of the form $\sigma = (\sigma_1, \cdots, \sigma_p)$, p is not fixed
- Online algorithm reacts on every σ_i
  - does not know $\sigma_{i+1}, \sigma_{i+2}, \cdots$
  - does not know their number (p)
- These decisions form the solution A(σ) $\leftarrow S_\sigma$
- Offline algorithms: know the future

Definition 1.4. Competitive ratio

- An online algorithm A for minimization problem π has a competitive ratio $r > 1$ if there is some constant $\tau \in \mathbb{R}$ s.t.

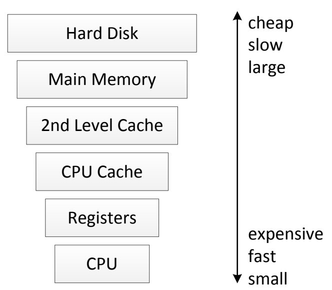
$$w_A(\sigma) \leq r \cdot OPT(\sigma) + \tau \quad \forall \sigma \in I_\pi$$

- A is strict r-competitive

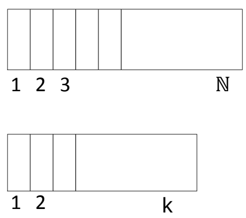
$$w_A(\sigma) \leq r \cdot OPT(\sigma) \quad \forall \sigma \in I_\pi$$

2 Paging

2.1 Deterministic Algorithms



here: only two levels



input: $\sigma = (\sigma_1, \cdots \sigma_n)$ sequence of page requests
$\sigma_i \in \mathbb{N}$ denotes the number of requested page

- if σ_i is in the cache, no additional cost
- if σ_i is not in the cache, cost of 1 (the algorithm has to load the page into the cache: page fault)
- if cache is full, the algorithm has to choose a page in the cache that has to be removed

Deterministic Algorithms

- LRU (least-recently used) removes the page requested least recently
- LFU (last-frequently used) removes the page that was requested least of them
- FIFO (first-in-first-out) removes the oldest page in cache
- LIFO (last-in-first-out) removes newest page in cache
- FWF (flush-when-full) completely empties the cache when the cache is full and there is a page fault
- LFD (longest-forwarded-distance) remove the page that will be requested the latest

2.1.1 Marking Algorithms

Decompose input $\sigma = (\sigma_1 \cdots \sigma_n)$ into phases as follows

- Phase 1: maximal prefix with k different pages
- Phase $i \geq 2$: maximal sequence following phase i-1 with at most k different pages
- Example: $k = 3$: $\sigma = \underbrace{1, 2, 4, 2, 1,}_{Phase 1} \underbrace{3, 5, 2, 3, 5,}_{Phase 2} \underbrace{1, 2, 3, 4}_{Phase 3}$

A marking algorithm is an algorithm that never removes a marked page from the cache. At the beginning of a phase no page is marked. A page that is accessed during a phase becomes marked.

Theorem 2.1. *LRU is a marking algorithm*

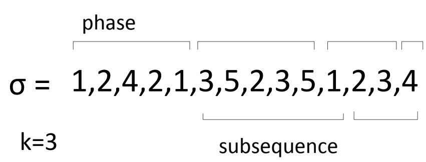
Proof. Assume LRU is not a marking algorithm.
$\Rightarrow$ There is an input sequence σ on which LRU removes a marked page x in phase i. Let σ_t be the corresponding event

- since x is marked, it was used in phase i before, let $\sigma_{t'}$ with $t' < t$ the first access of page x in phase i.
- of all pages requested after $\sigma_{t'}$, x is the most least recently used
- since x is removed at time σ_t there must be k different pages different from x accessed between $\sigma_{t'}$ and σ_t
  $\Rightarrow$ together with the requests of x this would be $k+1$ different pages requested in one phase. (contradiction definition phase) $\square$

Theorem 2.2. *Every marking algorithm is strict k-competitive*
(at most k time worse than optimal offline algorithm)

Proof. Let σ be an arbitrary input instance and l is the number of phases of this input instance. w.l.o.g (without loss of generality) $l \geq 2$

1. Cost of marking algorithm is at most $l \cdot k$
   - l phases, each phase at most k different request
   - every page is marked at the first request and never removed. At most one page fault per page.
2. Cost of an optimal offline algorithm is at least $k + l - 2$
   - k page faults in the first phase
   - one page fault in each of the following phases, except the last one ($l - 2$ phases).
   - Define subsequence i as follows:
     - starts with the second request of phase $i + 1$
     - ends with first request of phase $i + 2$
     - Example:



   - Beginning of phase $i + 1$, there is some request x
   - Beginning of subsequence i, x and $k + 1$ pages different from x in the cache

- in subsequence i there are k different (different from x) requests
  $\Rightarrow$ at least one page fault

$$OPT(\sigma) \geq k + l - 2$$
$$w_A(\sigma) \leq l \cdot k \leq (k + l - 2) \cdot k \leq k \cdot OPT$$

□

Corollary 2.1. *LRU is k-competitive*

2.1.2 Lower Bounds

Theorem 2.3. *LFU & LIFO are not competitive*

Proof.

- Given any τ, r construct sequence σ s.t. (such that)

$$w_{LFU}(\sigma) > r \cdot OPT(\sigma) + \tau$$

- Consider for any constant $l \geq 2 : \sigma(\underbrace{1^l}_{1,\cdots,1}, 2^l, \cdots, (k-1)^l, (k, k+1)^{l-1})$

- optimal solution, only $k + 1$ page faults

- LFU/LIFO:
  - until first request of $k + 1 : k$ page faults and $\{1 \cdots k\}$ in cache
  - Both remove k (last-in/least frequently)
  - following request of k : Both remove page $k + 1$
  - this repeats $\Rightarrow$ at least $2 \cdot (l - 1)$ page faults

- Choice of $l : 2(l - 1) > r \cdot (k + 1) + \tau = r \cdot OPT(\sigma) + \tau$ □

2.1.3 Optimal Offline Algorithm

Lemma 2.1. *Let A be an optimal offline algorithm different from LFD and σ an arbitrary input sequence where LFD and A behave differently. Let σ_t be the first request where they differ. Then there is an algorithm B that*

- *behaves like A on $\sigma_1, \cdots \sigma_{t-1}$*

- *at σ_t it removes the page from the cache that will be requested the latest*

- *incurs no higher cost than A*

Proof. We construct algorithm B as follows:

- on $\sigma_1, \cdots \sigma_{t-1}$ behaves like A

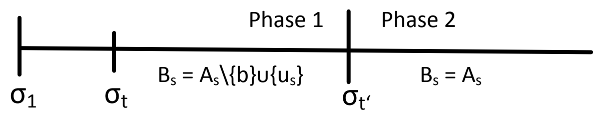
- at σ_t B removes the LFD-page

- (Idea: from now on, A and B have at least one page different in the cache)

- Let b be the LFD-page and a be the page that A chooses.
- Cache content of A after $\sigma_t : X \cup \{b\}$; of B is $X \cup \{a\}$ with $|X| = k - 1$
- Denote content of A (or B) cache before σ_s with A_s (or B_s, respectively)

- Divide $\sigma_{t+1}, \sigma_{t+2}, \cdots$ into two phases
  - Phase 1 includes all $s \geq t + 1$ with $B_s = (A_s \setminus \{b\}) \cup \{u_s\}$
  - Phase 2 includes all $s \geq t + 1$ with $B_s = A_s$

Construct algorithm B such that there is an event t' and all events between $\sigma_{t+1} \cdots \sigma_{t'}$ are in phase 1 and all events between $\sigma_{t'+1}, \sigma_{t'+2} \cdots$ are in phase 2.



- Phase 1: At request σ_s algorithm B works as follows
(reminder: $B_s = (A_s \setminus \{b\}) \cup \{u_s\})$
  1. request $\sigma_s \in A_S \cap B_s$: no page faults
  2. request $\sigma_s \notin A_S \cup B_s$: A and B cause page faults
     (a) A replaces b: B replaces $u_s \Rightarrow A_{s+1} = B_{s+1}$ (in phase 2)
     (b) A replaces $v \neq b$: B replaces $v \Rightarrow B_{s+1} = (A_{s+1} \setminus \{b\}) \cup \{u_s\}$
     (still in phase 1)
  3. request u_s : Only A causes page fault
     (a) A replaces $b \Rightarrow A_{s+1} = B_{s+1}$ (phase 2)
     (b) A replaces $v \neq b \Rightarrow B_{s+1} = A_{s+1} \setminus \{b\} \cup \{v\}$ (phase 1)
  4. request of b : Only B causes page faults and B removes page u_s from cache. Then $A_{s+1} = B_{s+1}$ (phase 2)

- Phase 2: B behaves like A and never leaves phase 2.
Observe that 1) - 4) ensure that we only reach configurations in phase 1 and 2. It remains to show that B causes not more page faults than A:
  - Obvious in case 1, 2 and 3
  - case 4:
    * can only happen <u>once</u>
    * b was the latest requested page at time t
      $\Rightarrow$ there must have been a request of page a
    * until first request of $a : u_s = a$
      $\Rightarrow$ first request of a : case 3
      $\Rightarrow$ also one page fault of A $\square$

Theorem 2.4. *LFD (longest-forwarded-distance) is an optimal offline algorithm for paging*

Proof. Let A_{OPT} be an optimal offline algorithm different from LFD. We modify A_{OPT} without increasing its cost, s.t. the resulting algorithm is LFD. Repeatedly apply Lemma 1.1.: For any sequence σ, let $A_0 = A_{OPT}$

1. Let σ_t be the first request where A_0 and LFD differ.
2. Apply Lemma 1.1. and let A_1 be algorithm B from Lemma 1.1.
3. repeat step 1 and 2 to obtain algorithm A_i until A_i behaves like LFD ($\Rightarrow$ same costs of A and LFD) □

Theorem 2.5. *There is no deterministic r-competitive online algorithm for paging with $r < k$.*

Proof. Let A be an arbitrary deterministic online algorithm for paging. We show that for any $\tau \in \mathbb{R}$ and every $r < k$ there exists a sequence σ with

$$w_A(\sigma) > r \cdot OPT(\sigma) + \tau$$

- We construct sequence σ with $k + l$ different page request
- $k + 1$ different pages
- $\sigma_1, \cdots \sigma_k$: k different pages, i.e. $1, 2, \cdots, k$
  $\sigma_{k+1}, \cdots \sigma_{k+l}$: request the page that is not in the cache of A
  $\Rightarrow A$ causes $k + l$ page faults.
- Show that LFD will have first k and then at most $k + \frac{\lceil l \rceil}{k} \leq k + 1 + \frac{l}{k}$ page faults.
- For every choice of k, τ and $r < k$ we can choose a l, such that

$$w_A(\sigma) = k + l > r(k + 1 + \frac{l}{k}) + \tau \text{ by}$$
$$l > \frac{k}{k - r} \cdot (r(k+1) - k + \tau)$$

□

2.2 Randomised Algorithms

Idea: algorithms use randomness for some of their decisions. Hope, that by using these algorithms, at the end you have better competitive factor than k.
Two simple algorithms:

1. RANDOM: Upon a page fault, select a page from the cache uniformly at random and replace it.
2. MARK: If we have a page request, we mark the requested page. If we have a page fault, we choose unmarked page uniformly at random. If all pages are marked, remove all markings and choose the page to remove uniformly at random.

Redefined Measures:

- Costs are random variables that depend on the random decisions of the algorithm.
- We study expected cost:

$$E(w_A(\sigma)) = \sum_{i=-\infty}^{\infty} i \cdot Pr(w_A(\sigma) = i)$$

where $Pr(w_A(\sigma) = i)$ is the probability that cost of A on input σ is exactly i.

2.2.1 Worst-Case Analysis as a Game

1. algorithm A tries to achieve a certain competitive ratio
2. adversary (Adv) chooses an input sequence such that algorithm A violates that competitive ratio. Adv knows A including the probability distribution of $A's$ random bits.

When does the adversary chooses σ and what does he know?

1. Oblivious (Obl): adversary choose σ at the beginning (no knowledge about realization of random experiments)
   Comparison: $OPT(\sigma)$
2. adaptive adversary: creates σ online after observing the realization of $A's$ random experiments.
   σ is now a random variable
   (a) adaptive online: constructs a solution for comparison online.
   (b) adaptive offline: takes the expected value of the optimal solution of σ : $E(OPT(\sigma))$

Notation:
Online Algorithm A, adversary Adv. Input created by $Adv : \sigma_{Adv}$, cost of Adv on $\sigma_{Adv} : w_{Adv}$

Definition 2.1. Let A be a randomized online Algorithm. A has a competitive factor of $r \geq 1$ against a class $C \in \{Obl, AdOn, AdOf\}$ of adversaries if there is a constant $\tau \in \mathbb{R}$ s.t. for every $Adv \in C$:

$$E(w_A(\sigma_{Adv}) \leq r \cdot E(w_{Adv}) + \tau$$

holds. If $\tau = 0$ then A is strict r-competitive.

2.2.2 Potential Function

- For online algorithms let S_A be the set of configurations of A and S_{Adv} the set of configurations of Adv.
- Paging: $S_A = S_{Adv}$ = set of possible contents of the cache.
- A potential function $\Phi : S_A \times S_{Adv} \to \mathbb{R}$ creates for a sequence $\sigma_1 \cdots \sigma_n$ a sequence of potential $\Phi_0, \Phi_1, \cdots, \Phi_n$ where Φ_0 is the potential value before σ_1 and $\Phi_i (i \geq 1)$ the value of the event σ_i.
- Cost of algorithm A at event $\sigma_i : A_i$
- amortised cost of A at event $\sigma_i = A_i + \Phi_i - \Phi_{i-1}$
- Cost of adversary: Adv_i

Theorem 2.6. *Let A be an online algorithm and $C \in \{Obl, AdOn, AdOf\}$. If there is a constant $b \geq 0$ s.t. for every $Adv \in C$ there is a potential function Φ which satisfies following two conditions then A is r-competitive against C.*

1. $\forall i \geq 1 : E(a_i) \leq r \cdot E(Adv_i)$
2. $\forall i \geq 1 : E(\Phi_i) \in [-b, b]$

Proof. Let $Adv \in C$ and $\sigma = (\sigma_1, \cdots, \sigma_n)$ input created by Adv.
(Note: $E(X + Y) = E(X) + E(Y)$ holds, even if X, Y are correlated.)

$$\begin{aligned}
E(w_A(\sigma)) &= \sum_{i=1}^{n} E(A_i) \\
&= \sum_{i=1}^{n} E(a_i - \Phi_i + \Phi_{i-1}) \\
&= \sum_{i=1}^{n} (E(a_i) - E(\Phi_i) + E(\Phi_{i-1})) \\
&= \sum_{i=1}^{n} E(a_i) + E(\Phi_o) - E(\Phi_n) \\
&\leq r \cdot \sum_{i=1}^{n} E(Adv_i) + 2b \\
&= r \cdot w_{Adv} + 2b
\end{aligned}$$

□

2.2.3 Analysis of RANDOM

Theorem 2.7. *RANDOM is k-competitive against an adaptive online adversary.*

Proof. Let $Adv \in AdOn$

- Denote by z_i the number of pages in the caches of RANDOM and Adv that both have in common after σ_i.

- Let $\Phi_i = k(k - z_i)$ for $i \geq 1$ and $\Phi_0 = k^2$. Observe $\Phi_i \in [0, k^2]$
- Let $Rand_i$ and Adv_i be the cost of RANDOM and Adv respectively after σ_i. To use Theorem 2.6. we need to show:

$$\begin{aligned} E(a_i) \leq k \cdot E(Adv_i) \text{ which is equivalent to} \\ E(\Phi_i - \Phi_{i-1}) \leq k \cdot E(Adv_i) - E(Rand_i) \end{aligned} \tag{1}$$

- Case distinction: (cache is already filled with k pages)
  Let P with $|P| = z_{i-1}$ pages in common before σ_i. Let $p = \sigma_i$ be the next page. Note: P and p are random variables.
- We show that equation 1 holds for every choice of P and p.

  1. p is in cache of RANDOM $\Rightarrow Rand_i = 0$
     - If p is in the cache of Adv then number of pages in common stays the same: $\Phi_i - \Phi_{i-1} = 0$ $\surd$
     - If p is not in the cache of Adv then $\Phi_i - \Phi_{i-1} \in \{0, k\}$ and $Adv_i = 1$ $\surd$
  2. p is not in cache of RANDOM, but in the cache of $Adv_i \Rightarrow Rand_i = 1$ and $Adv_i = 0$
     (a) RANDOM removes a page $\in P : \Phi_i - \Phi_{i-1} = 0$
     (b) RANDOM removes a page $\notin P : \Phi_i - \Phi_{i-1} = -k$
     Probability for choosing a page $\notin P$: $\frac{k-z_{i-1}}{k}$
     (a)+(b) $\Rightarrow$

$$E(\Phi_i - \Phi_{i-1}) = \frac{k - z_{i-1}}{k} \cdot (-k) = z_{i-1} - k \leq -1 \quad \surd$$

  3. p is not in cache of RANDOM and not in the cache of Adv
     $k \cdot E(Adv_i) - E(Rand_i) = k - 1$
     (a) Adv removes page $\notin P$ then
     $\Phi_i - \Phi_{i-1} \in \{0, \cdots, k\}$ $\surd$
     (b) Adv removes page $\in P$ then
     Potential only changes if RANDOM removes a different page $\in P$
     Probability for this is: $\frac{z_{i-1}-1}{k}$ which gives

$$E(\Phi_i - \Phi_{i-1}) = (\frac{z_{i-1} - 1}{k}) \cdot k \leq k - 1$$

  $\Rightarrow$ This shows Equation 1 for all choices of P and p. □

Lower Bound for RANDOM

geometric random variables:

- X : number of repetitions of experiments with probability p until first success. $Pr(X = i) = (1 - p)^{i-1} \cdot p$; $E(X) = \frac{1}{p}$
- Cut-off: $Y = min\{X, n\}$

Lemma 2.2. *Let X be a geometric random variable with parameter p and $n \in \mathbb{N}$. For $Y = min\{X, n\}$ $E(Y) = \frac{1-(1-p)^n}{p}$*

Proof. Let $q = 1 - p$

$$\begin{aligned}
E(Y) &= \sum_{i=1}^{n} i \cdot Pr(min\{X, n\} = i) \\
&= \sum_{i=1}^{n-1} i \cdot Pr(X = i) + \sum_{i=n}^{\infty} n \cdot Pr(X = i) \\
&= \sum_{i=1}^{\infty} min\{i, n\} \cdot p \cdot q^{i-1} \\
&= \sum_{i=1}^{\infty} i \cdot p \cdot q^{i-1} - \sum_{i=n+1}^{\infty} (i - n) \cdot p \cdot q^{i-1} \\
&= E(X) - q^n \cdot \sum_{i=1}^{\infty} i \cdot p \cdot q^{i-1} \\
&= (1 - q^n) \cdot E(X) \\
&= \frac{1 - q^n}{p}
\end{aligned}$$

□

Theorem 2.8. *The competitive factor of RANDOM against an oblivious adversary is at least k.*

Proof. Consider an oblivious adversary that chooses
$\sigma = ((a_1, \cdots, a_k), (b_1, a_2, \cdots a_k)^l, (b_2, a_2, \cdots a_k)^l, \cdots, (b_m, a_2, \cdots a_k)^l)$
$OPT(\sigma) = k + m$ page faults.
RANDOM:

- consider a block $(b_i, a_2, \cdots a_k)^l$
- At beginning at most $k - 1$ of these pages are in the cache
- page fault is successful if cache content is $\{b_i, a_2, \cdots a_k\}$ afterwards
- otherwise removed a page $\in \{b_i, a_2, \cdots a_k\}$ from the cache
- Probability of successful page fault is at most $\frac{1}{k}$
- Using Lemma 2.2. the expected number of page faults per block is $k \cdot (1 - (1 - \frac{1}{k})^l)$
- $E(w_{RANDOM}(\sigma)) \geq k + m \cdot k \cdot (1 - (1 - \frac{1}{k})^l) \geq m \cdot k \cdot (1 - (1 - \frac{1}{k})^l)$
- For any $r < k$ and $\tau \in \mathbb{R}$ choose m and l such that
  - $E(w_{RANDOM}(\sigma)) > r \cdot OPT(\sigma) + \tau$
  - $m \cdot k \cdot (1 - (1 - \frac{1}{k})^l) > r \cdot (k + m) + \tau$

- since $\lim_{l\to\infty}(1-(1-\frac{1}{k})^l) = 0$ and $r < k$, there is a l such that $r' = k((1-(1-\frac{1}{k})^l) > r$
- For this l: $m \cdot r' > r(k+m) + \tau$ holds with $m = 1 + \frac{r \cdot k + \tau}{r' - r}$

□

2.2.4 Analysis of MARK

Theorem 2.9. *MARK is* $2 \cdot H_k$*-competitive against* *oblivious* *adversary.*
($H_k = \sum_{i=1}^{k} \frac{1}{i} = \Theta(log\ k)$*)*

Proof. Let σ be input chosen by adversary. Consider phases as in the proof of the deterministic case.

- phase 1: MARK and adversary each have k page faults
- phase $i \geq 2$:
  - old page: page accessed in phase $i-1$
  - new page: no access in phase $i-1$
  - Let m_i be the number of these new pages in phase i
  - new pages cause exactly one page fault
  - old pages: probability that page is still in cache when first accessed decreases with the number of new pages accessed before
  - worst case: each of the m_i new pages is accessed (at least once) before the $k - m_i$ old pages are accessed
  - sort old pages $j \in \{1, \cdots, k - m_i\}$ by their first access in phase i
  - P_j probability of j still in cache at first access
  - $P_1 = \frac{k-m_i}{k}$, $P_j = \frac{k-m_i-(j-1)}{k-(j-1)}$
    $k - m_i - (j-1) \leftarrow$ number of marked old pages in the cache
    $k - (j-1) \leftarrow$ total number of unmarked old pages (including) those not in cache.
  - Expected number of page faults caused by page j:
    $P_j \cdot 0 + (1 - P_j) \cdot 1 = 1 - P_j$
  - Total number of page faults in phase i:

$$\begin{aligned} m_i + \sum_{j=1}^{k-m_i}(1-P_j) &= \sum_{j=1}^{k-m_i} \frac{m_i}{k-(j-1)} + m_i \\ &\leq m_i \cdot \sum_{j=1}^{k} \frac{1}{k-(j-1)} \\ &= m_i \cdot H_k \end{aligned}$$

- Let n be the number of phases and $m_1 = k$ then

$$E(w_{MARK}(\sigma)) \leq H_k \cdot \sum_{i=1}^{n} m_i$$

  optimal offline solution

- Consider 2 phases $i-1$ and i. There are $k - m_i$ different pages accessed in the sequence consisting of both phases.
- at most k of these pages in the cache at beginning $\Rightarrow$ at least m_i page faults
- Consider 1st phase and every sequence of two consecutive phases and add page faults: $\sum\limits_{i=1}^{n} m_i$
- $OPT(\sigma) \geq \frac{1}{2} \sum\limits_{i=1}^{n} m_i$ thus $E(w_{MARK}(\sigma)) \leq 2 \cdot H_k \cdot OPT(\sigma)$

□

2.2.5 Lower Bounds for Randomized Online Algorithms

Theorem 2.10. *There is no randomized online algorithm against oblivious adversaries with competitive factor smaller than H_k.*

Proof. Let A be an arbitrary randomized online algorithm for paging.

- The oblivious adversary constructs an input sequence σ consisting of $k+1$ different pages.
- The adversary can compute for a given sequence $(\sigma_1, \cdots, \sigma_q)$ a probability distribution $(p_1, \cdots, p_{k+1})$ with $p_i \in [0,1]$ and $\sum\limits_{i=1}^{k+1} p_i = 1$.
- p_i : probability that page i is <u>not</u> in the cache after step σ_q
- The adversary constructs σ in phases (like marking algorithm)
- m phases and each phase consists of k different pages. Pages are marked after first access + last page of previous phase
- each phase σ' is divided into k subphases $\sigma'_1, \cdots, \sigma'_k$

$$\sigma = (\overbrace{\sigma_1 \cdots}\overbrace{\underbrace{\cdots}_{\sigma'_1}\underbrace{\cdots}_{\sigma'_2}\cdots\underbrace{\cdots}_{\sigma'_4}}^{\sigma'}\cdots\cdots)$$

  Each subphase

  - exactly one page becomes marked
    $\rightarrow$ after σ'_j exactly $j+1$ marked pages
  - consists of first zero or more requests of already marked pages, followed by exactly one request of an unmarked page

- Aim: Expected costs for A for $\sigma'_j : \frac{1}{k-j+1}$

- construct σ'_j:
  - Let M set of marked pages at start σ'_j
  - $|M| = j$ and number of unmarked pages $U = k + 1 - j$
  - Let $\gamma = \sum\limits_{i \in M} p_i$
  - If $\gamma = 0$ then there is an unmarked page a with $p_a \geq \frac{1}{U}$, request a and subphase ends
  - otherwise $\gamma > 0$ then there is a marked page m with $p_m > 0$
  - Let $\epsilon = p_m$ and request m. Request more marked pages as follows:
    * while the total expected cost of A for this subphase is less than $\frac{1}{U}$ and while $\gamma > \epsilon$ request page $l \in M$ with $l = \underset{i \in M}{argmax}\ p_i$
    * Finally pick unmarked page b with $b = \underset{i \notin M}{argmax}\ p_i$
- Remarks:
  - Expected cost of A = sum of p_i of requested pages.
  - $p_1, \cdots, p_k + 1$ and γ have to be recomputed each iteration
  - while loop terminates if $\gamma > \epsilon$ then $p_l \geq \frac{\gamma}{|M|} \geq \frac{\epsilon}{|M|}$
- Expected cost of A in σ'_j
  - case $\gamma = 0 : p_a \geq \frac{1}{U}$. Expected cost $\geq \frac{1}{U}$ $\surd$
  - while loop terminates with expected cost $\geq \frac{1}{U}$ $\surd$
  - while loop terminates with $\gamma \leq \epsilon$:
    $b = \underset{i \notin M}{argmax}\ p_i$; $p_b \geq \frac{1-\gamma}{U}$
  - Cost of A in $\sigma'_j : \epsilon + p_b \geq \epsilon + \frac{1-\gamma}{U} \geq \epsilon + \frac{1-\epsilon}{U} \geq \frac{1}{U}$ $\surd$
  - Expected cost of A in phase σ' is
    $\sum\limits_{j=1}^{k} \frac{1}{k+1-j} = H_k$. Thus

$$E(w_A(\sigma)) \geq k + (m-1) \cdot H_k$$
$$and$$
$$OPT = k + m - 1$$

- By choosing m large enough the Theorem follows □

3 The k-Server-Problem

3.1 Introduction

Let $k \geq 2$ and $\mathcal{M} = (M, d)$ a metric space where $|M| > k$ and M is a set of points (arbitrary set) and $d : M \times M \to \mathbb{R}_{\geq 0}$ is a metric distance function with

1. $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$ Symmetry
3. $d(x, z) \leq d(x, y) + d(y, z)$ triangle inequality

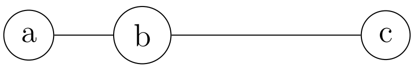
Example $(\mathbb{R}^2, d)$ with d euclidean distance function.
If M is finite, representation by complete weighted graph.

k-Server-Problem

- Algorithm controls k mobile servers which are located on points of M.
- Input $\sigma = (\sigma_1, \cdots \sigma_n)$ is a sequence of points $\sigma_i \in M$ (request).
- A request σ_i is served if a server is on position σ_i.
- Algorithm may move servers at cost of distance.

3.1.1 Greedy Algorithm

on request σ_i move the server that is closest to σ_i.
Example: $k = 2, |M| = 3., \sigma = (c, (a, b)^l)$



$d(a, b) < d(b, c)$

- after request c: one server at c
- after request a: one server at c and a each
- following request: greedy moves server between a and b
- OPT: one server at a and b each

3.1.2 The k-Server Conjecture

Any metric space allows for a deterministic k-competitive k-server algorithm

- lower bound of k (later in lecture)
- upper bound: $(2k - 1)$-competitive algorithm (Koutsoupias and Papadimitriou)

Lazy algorithms

- Only moves servers if no server on requested point
- Only moves one server and only to requested point
- Paging as k-server problem
  - M = set of pages, distance =1
  - position of k- servers $\approx k$ pages in cache
- k-headed disk-problem
  - $M = [0,1]$
  - $d(x,y) = |x-y|$ line metric

3.1.3 Optimal Offline Algorithm

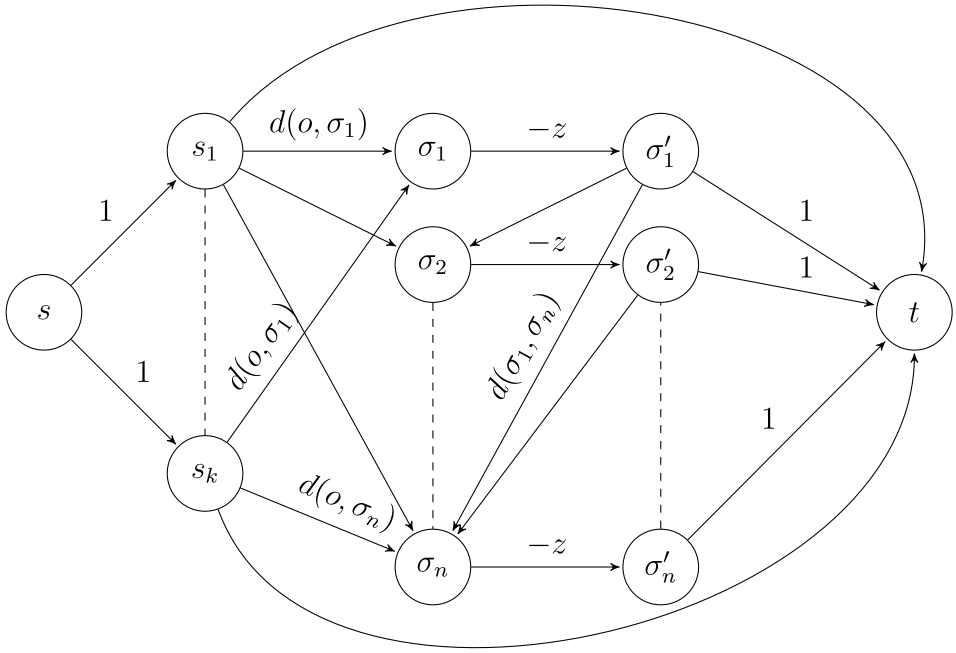
- Dynamic programming: $\mathcal{O}(|\sigma|\ |M|^k)$
- Reduction to Min-Cost-Flow-Problem
  - input: directed graph $G=(V,E)$ with
    - source $s \in V$
    - target $t \in V$
    - capacity function $u : E \to \mathbb{R}_{\geq 0}$
    - cost function $c : E \to \mathbb{R}$
    - no negative cycles
  - output: maximal flow $f : E \to \mathbb{R}_{\geq 0}$ with minimal costs $c(f) = \sum\limits_{l \in E} f(l) \cdot c(l)$
- flow conservation $\sum\limits_{l=(u,v)\in E} f(l) = \sum\limits_{l=(v,u)\in E} f(l) \ \ \forall v \in V \setminus \{s,t\}$
- capacities:
  - $\forall e \in E: \ 0 \leq f(e) \leq u(e)$
  - value of flow: $|f| = \sum\limits_{l=(s,v)} f(l) = \sum\limits_{l=(v,t)} f(l)$

Successive-Shortest-Path-Algorithm

- integer capacities $u : E \to \mathbb{N}$
  $\Rightarrow \exists$ min-cost-flow with integers that is computed by this algorithm
- $\mathcal{O}(n^3 F)$ running time, (only pseudo polynomial, F is value of maximal flow)

Given a k-server problem by a metric $\mathcal{M} = (M, d)$ and input sequence $\sigma = (\sigma_1 \cdots \sigma_n)$. w.l.o.g (without loss of generality)are all servers at the same point $\sigma \in M$ at beginning and $n \geq k$.

Construct instance of min-cost-flow as follows:



- $G = (V, E)$ with
  - $V = \{s, t\} \cup \{s_1, \cdots s_k\} \cup \{\sigma_1, \cdots, \sigma_n\} \cup \{\sigma'_1, \cdots, \sigma'_n\}$
  - $E = \{(s, s_i) \mid i \in \{1 \cdots k\}\} \cup$
    $\{(s_i, t) \mid i \in \{1 \cdots k\}\} \cup$
    $\{(s_i, \sigma_j) \mid i \in \{1 \cdots k\}, j \in \{1 \cdots n\}\} \cup$
    $\{(\sigma_j, \sigma'_j) \mid j \in \{1 \cdots n\}\} \cup$
    $\{(\sigma'_j, \sigma_l) \mid j \in \{1 \cdots n\}, l \in \{1 \cdots n\}, l > k\} \cup$
    $\{(\sigma'_j, t) | j \in \{1 \cdots n\}\}$
  - $u(l) = 1 \ \forall l \in E$
  - Cost function:
    * $c(s, s_i) = 0$
    * $c(s_i, \sigma_j) = d(o, \sigma_j)$
    * $c(s_i, t) = 0$
    * $c(\sigma_j, \sigma'_j) = -z$ with $z > 2 \cdot \underset{x,y \in M, x \neq y}{max}(d(x, y))$
    * $c(\sigma'_j, \sigma_l) = d(\sigma_j, \sigma_l)$
    * $c(\sigma'_j, t) = 0$
  - Observe: no negative cycles

- capacities of 1, integer flow $\Rightarrow f(l) = 0$ or $f(l) = 1 \;\; \forall l \in E$
- max flow has value k
- flow corresponds to edge disjoint paths
- let p_i be the path that contains s_i, then there is $l \geq 0$ and $j_1 \cdots j_l$ such that $p_i = (s, s_i, \sigma_{j_1}, \sigma'_{j_1}, \cdots, \sigma_{j_l}, \sigma'_{j_l}, t)$
  with cost: $d(\sigma, \sigma_{j_1}) + d(\sigma_{j_1}, \sigma_{j_2}) + \cdots + d(\sigma_{j_{l-1}}, \sigma_{j_l}) - lz$
  which corresponds to cost of a server answering this sequence plus additional lz term
- Every edge $e = (r_j, \sigma'_j)$ is contained in exactly one path p_i
- obtain a solution L for k-server: Let server i answer requests σ_j if $e = (\sigma_j, \sigma'_j)$ is contained in p_i
- cost of L = cost of flow $f + nz$

Correctness: If there was a solution L' with cost less than L (L is obtained form f) we could construct a flow with less cost than f. ↯
Running time: $\mathcal{O}(n^3 k)$

3.2 Lower Bound for Deterministic Online Algorithm

Theorem 3.1. *Let $\mathcal{M} = (M, d)$ be an arbitrary metric space with $|M| \geq k+1$. There is no r-competitive online algorithm for the k-server-problem on $\mathcal{M}$ for average $r < k$.*

Proof. Let A be an arbitrary lazy online algorithm for k-server-problem. Let $B = \{b_1, \cdots, b_{k+1}\} \subseteq M$ an arbitrary subset of M with $k+1$ elements. We assume that A starts with k different points of B.

$\Rightarrow A$ always has at most one server on each point. Input σ : always request the point in B on which A has no server.

Lemma 3.1. $w_A(\sigma) \geq \sum_{i=1}^{n-1} d(\sigma_i, \sigma_{i+1})$

Proof. (Lemma 3.1.)

- After request σ_i we request σ_{i+1} the point that was covered by the server that answered request σ_i
- cost for answering $\sigma_i \geq d(\sigma_i, \sigma_{i+1})$ for all $i \leq n-1$

□

Lemma 3.2. $OPT(\sigma) \leq \frac{1}{k} \sum_{i=1}^{n-1} d(\sigma_i, \sigma_{i+1})$

Proof. (Lemma 3.2.) Indirect proof: Define a class C of algorithms.

- For each $S \subseteq B$ with $\sigma_1 \in S$ and $|S| = k$ there is an algorithm C_S. C_S works as follows:

- Initially C_S places servers on S
- for request σ_1: nothing to do
- for σ_i ($i \geq 2$) and no server on σ_i it moves server on σ_{i-1} to σ_i

- There are k different sets S. Thus $|C| = k$.
- Let S^i be the set of points on which servers of C_S are located after σ_i
- We show that for all different sets $S_1 \neq S_2$ and all $i \geq 0$: $S_1^i \neq S_2^i$ holds:
  $i = 0$: obvious
  I.S.: Case distinction by σ_{i+1}
  - $\sigma_{i+1} \in S_1^i$ and $\sigma_{i+1} \in S_2^i$: no movement of either algorithm
    $S_1^{i+1} = S_1^i \neq S_2^i = S_2^{i+1}$
  - $\sigma_{i+1} \in S_1^i$ and $\sigma_{i+1} \notin S_2^i$: observe $\sigma_i \in S_1^i$ and $\sigma_i \in S_2^i$
    After σ_{i+1} : $\sigma_i \in S_1^{i+1}$ but C_{S_2} moves server from σ_i to σ_{i+1}
    Thus: $\sigma_i \notin S_2^{i+1}$
  - $\sigma_{i+1} \notin S_1^i$ and $\sigma_{n+1} \in S_2^i$: symmetric to case above
  - $\sigma_{i+1} \notin S_1^i$ and $\sigma_{i+1} \notin s_2^i$: Cannot happen, would imply $S_1^i = S_2^i$. Thus two algorithms never have their servers on exactly the same positions.
- there are k algorithms C_S
- Each has a server on σ_i after request σ_i

  $\Rightarrow$ for every $b \in B \setminus \{\sigma_i\}$ there is exactly one algorithm C_S with $b \notin S^i$
- For $b = \sigma_{i+1}$ only one algorithm has cost of $d(\sigma_i, \sigma_{i+1})$
- sum of costs of all algorithms:

  $\sum\limits_S w_{C_S}(\sigma) = \sum\limits_{i=1}^{n-1} d(\sigma_i, \sigma_{i+1})$ Average cost: $\frac{1}{k} \sum\limits_{i=1}^{n-1} d(\sigma_i, \sigma_{i+1})$

  There has to be an algorithm with cost no higher than average cost □

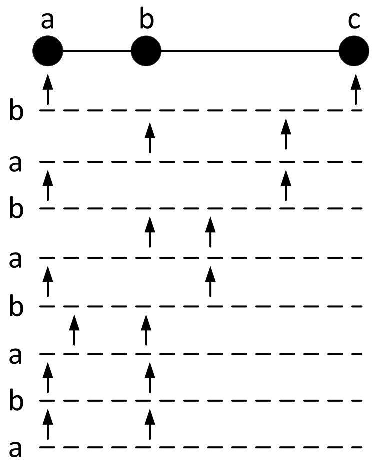
Combination of Lemma 3.1. and Lemma 3.2. proofs the Theorem. □

3.3 k-Server Problem on a Line

Is motivated by k-headed disk problem. $\mathcal{M} = ([0,1], d)$ with $d(x,y) = |x-y|$
Algorithm is called Double Coverage (DC)

- If request σ_i is left (or right) of all servers DC-algorithm move leftmost (rightmost) server to σ_i

- otherwise the DC-algorithm moves the two servers left and right of σ_i with the same velocity towards σ_i. It stops both servers as one arrives at σ_i



Theorem 3.2. *The DC-algorithm is k-competitive for the k-server-problem on the line*

Proof. Potential function Φ

- configuration of DC: $s_1, \cdots, s_k \in [0,1]$
- configuration of OPT: $o_1, \cdots, o_k \in [0,1]$
- $\Phi = k \cdot M_{min} + \Sigma_{DC}$ with $M_{min} = \min_{\pi \in \mathcal{S}_k} \{\sum_{i=1}^{k} d(s_i, o_{\pi(i)})\}$
- minimum cost matching between OPT's and DC's servers.
  - $\mathcal{S}_k$: Set of permutations of $\{1 \cdots k\}$ and
  - $\Sigma_{DC} = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(s_i, s_j)$ sum of pairwise distances of DC's servers
- DC_i and OPT_i the cost of DC and OPT serving request σ_i
- Φ_0 potential before σ_1 and Φ_i potential after step σ_i $(i \geq 1)$
- amortized cost after step i: $a_i = DC_i + \Phi_i - \Phi_{i-1}$ need to show (see lecture 3)
  1. For every $i \geq 1 : a_i \leq k \cdot OPT_i(\sigma)$ and
  2. for every $i \geq 1 : \Phi_i \in [-b, b]$
- Note that (2) holds for $b = 2k^2$ since $d()$ is bounded by 1.

  $0 \leq \Phi_i \leq k^2 + \binom{k}{2} \leq 2k^2$
- for property (1) we show that $\Phi_i - \Phi_{i-1} \leq k \cdot OPT_i(\sigma) - DC_i(\sigma)$
- Note: In step i DC and OPT may move and change the potential. Therefore let Φ'_{i-1} be the potential after OPT answered request σ_i but before DC's movement.

Lemma 3.3. $\Phi'_{i-1} \leq \Phi_{i-1} + k \cdot OPT_i(\sigma)$

Proof. (Lemma 3.3)

- OPT moves one server and the distance is $OPT_i(\sigma)$
- $k \cdot M_{min}$ changes by at most $k \cdot OPT(\sigma)$
  (Consider the same assignment or permutation, distance of one pair increases by at most $OPT_i(\sigma)$)
- Σ_{DC} does not chance □

Lemma 3.4. $\Phi_i \leq \Phi'_{i-1} - DC_i(\sigma)$

Proof. (Lemma 3.4.)
Two cases: DC moves one or two servers

1. one server
   - σ_i is left of all servers (right case is analogue). Let S_{left} be the leftmost server of DC
   - Let $o'_1, \cdots, o'_k \in [0,1]$ be the positions of the servers of OPT after request σ_i
   - $M'_{min} = \min_{\pi \in S_k} \sum_{i=1}^{k} d(s_i, o_{\pi(i)})$
   - there is a server $o'_j = \sigma_i$ (j answered the request σ_i) and o'_j is left of S_{left}
     (a) There is an optimal assignment π which assigns S_{left} to o'_j
     DC moves S_{left} by distance DC_i towards o'_j
     First term of potential decreases by $k \cdot DC_i(\sigma)$
     (b) Pairwise distance between DC's server change:
     S_{left} moves away from all $k-1$ remaining servers by distance $DC_i(\sigma)$
     second term increases by $(k-1)DC_i(\sigma)$
   - combining (a) and (b) we get the new potential

$$\begin{aligned}\Phi_i &\leq \Phi'_{i-1} - k \cdot DC_i(\sigma) + (k-1)DC_i \\ &= \Phi'_{i-1} - DC_i(\sigma)\end{aligned}$$

2. two servers
   - Let s_1, s_2 be two servers
   - each moves by distance $\frac{DC_i(\sigma)}{2}$
     (a) OPT has a server j on σ_i and there is an optimal assignment π which assigns s_1 or s_2 to j. That server moves by distance $\frac{DC_i(\sigma)}{2}$ towards j.
     The other server moves at most $\frac{DC_i(\sigma)}{2}$ away from its assigned server.
     $\rightarrow M_{min}$-term of Φ does not increase
     (b) Second term$\sum_{DC} i$
     For every server $s' \neq s_1, s_2$: exactly one of s_1, s_2 moves towards S', the other moves away by the dame distance
     The distance between s_1 and s_2 decreases by $DC_i(\sigma)$

- combining (a) and (b) we get

$$\Phi_i \leq \Phi'_{i-1} - DC_i(\sigma)$$

□

Combining both lemmas we get

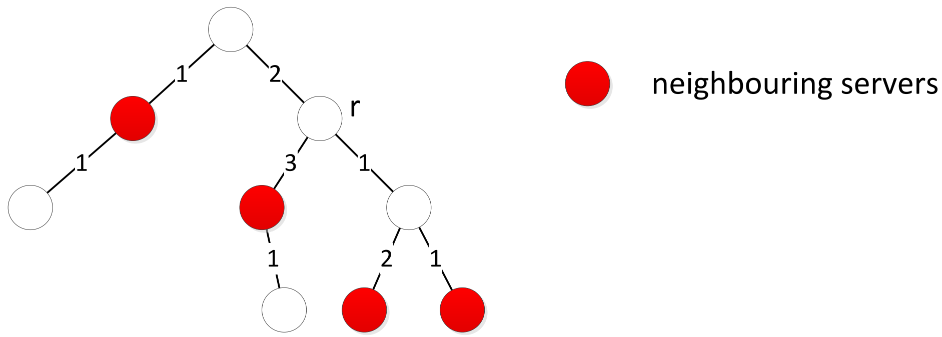
$$\Phi_i \leq \Phi'_{i-1} \leq \Phi_{i-1} + k \cdot OPT - DC_i(\sigma)$$

which proofs that the DC-algorithm is k-competitive on the line □

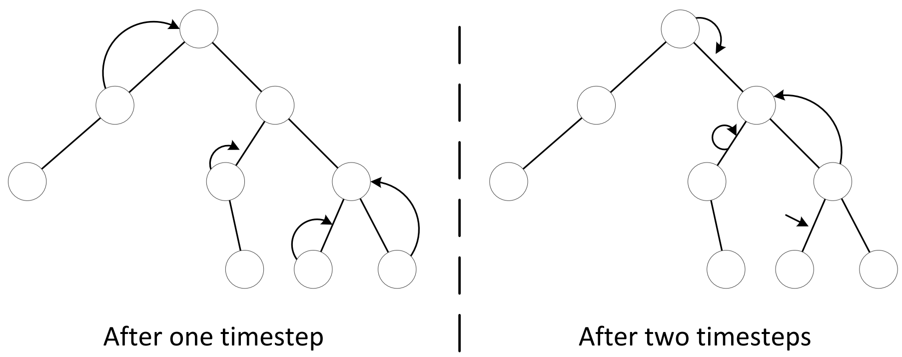
3.4 The DC-Algorithm on Trees

$\mathcal{M} = (M, d)$ is a tree-metric if there exists a tree $G = (V, E)$ with $V = M$ and edge weights $w : E \to \mathbb{R}_{\geq 0}$ s.t. that distance $d(x, y)$ is exactly the weight of the path between x and y in G. (Because of the tree-structure, paths are always unique)

- same algorithm. We redefine "neighbour" and movement
- neighbour:
  - Consider any configuration of k servers and a request r
  - We say a server s is neighbour of r if there is no other server on the path from s to r



  - if two servers are on the same point, only one of them is a neighbour
- movement
  - edge weight are distances
  - all neighbouring servers move with the same speed towards the request



- servers might stop being neighbours, stop movement
- servers that stop on edges between two points: Simulate DC by a lazy algorithm. Then servers always on points of the metric

Theorem 3.3. *DC-algorithm is k-competitive on arbitrary tree-metrics*

Proof. Same potential function as for the line.

$$\Phi = k \cdot \min_{\pi \in \mathcal{S}_k} \{\sum_{i=1}^{k} d(s_i, o_{\pi(i)})\} + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} d(s_i, s_j)$$

Lemma 3.5. $\Phi'_{i+1} \leq \Phi_i + k \cdot OPT_i(\sigma)$

Lemma 3.6. $\Phi_i \leq \Phi'_{i-1} - DC_i(\sigma)$

Proof. (Lemma 3.6)
We divide the movement of servers into phases. A phase ends when a server reaches request σ_i or when the number of moving servers decreases. Consider a phase in which m servers move, each by distance d.

1. Term M_{min} : There is an optimal assignment π which assigns a neighbouring server of DC to the server of OPT that moved to σ_i. That server moves by distance d towards the assigned server. The remaining $m-1$ active servers increase their distance by at most d
   $k \cdot M_{min}$ increases by at most $k(m-2)d$

2. Term Σ_{DC}:

   - Consider the $(k-m)$ servers that are not neighbours of σ_i. For each there is exactly one server moving away from it and $m-1$ active servers are moving towards it. For these pairs Σ_{DC} decreases in total by
     $(k-m)(m-2)d$
   - Every pair of active servers move towards each other and reduces the distance by $2d$. Σ_{DC} decreases by $\binom{m}{2}2d = dm(m-1)$.

Combining all three values shows that the potential decreases by at least md. This corresponds to the cost of moving servers, summing over all phases implies the lemma. □

□

3.5 Applying DC-Algorithm

- For a general finite metric $\mathcal{M} = (M, d)$ with $|M| = N$, let $G = (V, E)$ be a weighted graph representing $\mathcal{M}$.
- Compute a MST (Minimal Spanning Tree) $T = (V, E_T)$ and solve the k-server-problem on the tree-metric given by T.
- Note: Distance might increase in $\mathcal{M}_T$ compared to $\mathcal{M}$.

- Using DC-algorithm we get $w_{DC}(\sigma) = k \cdot OPT_T(\sigma) + \tau$ where OPT_T is optimal offline solution for $\mathcal{M}_T$
- For MST we know, that for each edge $e = \{x, y\} \in E$ the cost of the path from x to y in T is at most $(N-1)w_e$.
  $\Rightarrow$ Thus $OPT_T(\sigma) \leq (N-1)OPT(\sigma)$

Corollary 3.1. *The DC-algorithm is $(N-1)k$-competitive for arbitrary metrics with N points.*

3.6 The 2-Server-Problem in Euclidean Spaces

Here only consider unit square $M = [0,1]^2$ in two dimension.

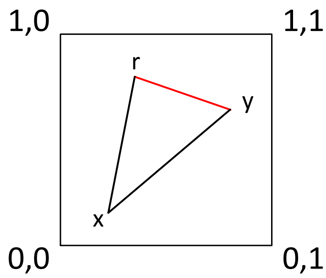
$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Definition 3.1. (Slack)

For three points $x, y, r \in M$ we define

$$slack(x, y, r) = d(x, y) + d(x, r) - d(y, r)$$

Note: Slack is non-negative due to the triangle inequality.



For each $\gamma \in [0,1]$ we consider the following algorithm:
$SlackCover_\gamma(SC_\gamma)$:

- Let x, y be the current positions of servers of SC_γ
- Let r be the position of the current request
- w.l.o.g. assume $d(x, r) \leq d(y, r)$
- SC_γ moves y by $y \cdot slack(x, y, r)$ towards x
- SC_γ moves x to r

<u>Note:</u>

- $SC_{\frac{1}{2}}$ on the line corresponds to the DC-algorithm
- Since $d(x, y) \leq d(y, r)$ we do not move y beyond x
- After movement of y, the server y is not further away from r than before

Theorem 3.4. *The algorithm $SC_{\frac{1}{2}}$ is 3-competitive for the 2-server-problem on the euclidean unit square.*

Proof.
Notes:

- x, y positions of SC_γ's servers
- o_1, o_2 positions of OPT's servers

Potential function

$$\Phi = aM_{min} + b \cdot d(x, y)$$

where M_{min} is defined as in the proof for DC and $a, b \in \mathbb{R}$ are parameters to be chosen later. As usual:

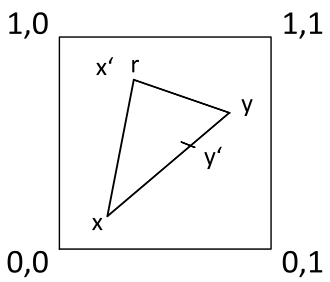
- input sequence $\sigma = (\sigma_1, \cdots, \sigma_n)$
- potential values $\Phi_0, \Phi_1, \cdots, \Phi_n$
- Φ is bounded by 0 and $\sqrt{2}(2a + b)$.

It remains to show that

$$\Phi_i - \Phi_{i-1} \leq \underbrace{a \cdot OPT_i(\sigma)}_{1} - \underbrace{SC_i(\sigma)}_{2}$$

Let o'_1, o'_2 be OPT's server positions after request $r := \sigma_i$ and Φ'_{i-1} the potential value before the step of SC_γ

1. $a \cdot OPT_i(\sigma)$
   - w.l.o.g. OPT is lazy, thus it moves one server by distance $OPT_i(\sigma)$
   - $d(x, y)$ does not change
   - $\Phi'_{i-1} - \Phi_{i-1} \leq a \cdot OPT_i(\sigma)$
2. $SC_i(\sigma)$
   Influence of SC_γ's movement: cost of $SC_i(\sigma) = d(x, r) + \gamma \cdot slack(x, y, r)$. We have to show that potential decreases by at least this amount. Let x', y' be the positions after serving $r := \sigma_i$

   We first consider the change of the second term of Φ

   $$\Delta d(x, y) := d(x', y') - d(x, y) = d(r, y') - d(x, y) \leq d(r, y) - d(x, y)$$

   Change of first term:

- depends on optimal assignment π before movement
- w.l.o.g. o'_1 is on request r
- Case 1:
  - x is assigned to o'_1. M_{min} decreases due to movement of the server on x towards r by $d(x,r)$ and increases by movement of server on y is at most $\gamma \cdot slack(x,y,r)$. Thus in total

$$\Phi_i - \Phi_{i-1} \leq a \cdot [\gamma \cdot slack(x,y,r) - d(x,r)] + b \cdot [d(r,y) - d(x,y)]$$

- Case 2:
  - y is assigned to o'_1
  - After moving x to r there is an optimal matching which assigns x' to o'_1 (on r) and y' to o'_2

$$\begin{aligned}\Delta M_{min} &= [\underbrace{d(x',o'_1)}_{=0} + d(y',o'_2)] - [d(x,o'_2) + d(y,o'_1)] \\ &= \underbrace{d(y',o'_2)] - d(x,o'_2)}_{triangle\ inequality} - d(y,r) \\ &\leq d(y',x) - d(y,r) \\ &= d(y,x) - \gamma \cdot slack(x,y,r)\end{aligned}$$

  - Thus using first term we get:

$$\Phi_i - \Phi'_{i-1} \leq a \cdot [d(y,x) - \gamma \cdot slack(x,y,r) + b \cdot [d(r,y) - d(x,y)]$$

- For both cases we have to show that

$$\Phi_i - \Phi'_{i-1} \leq -SC_i(\sigma) = -[d(x,r) + \gamma \cdot slack(x,y,r)]$$

- for case 1 we get:

  $a \cdot [\gamma \cdot slack(x,y,r) - d(x,r)] + b \cdot [d(r,y) - d(x,y)] \leq -[d(x,r) + \gamma \cdot slack(x,y,r)]$

  equivalent to:

  $d(x,y)[\gamma(a+1) - b)] + d(x,r)[\gamma(a+1) + 1 - a] + d(y,r)[b - \gamma(a+1)] \leq 0$

- for case 2:

  $d(x,y)[\gamma(1-a) + a - b] + d(x,r)[\gamma(1-a) + 1] + d(y,r)[ba\gamma(1-a)] \leq 0$

If we find parameters a, b, γ that satisfy both inequalities, we have shown that SC_γ is a-competitive. For $a = 3, b = 2, \gamma = \frac{1}{2}$ this is the case.

□

For an arbitrary metric space with N points, we can find N corresponding points in a high dimensional euclidean space. The distance between two points in the euclidean space is not smaller and at most $\mathcal{O}(log(n))$ larger than the distance in $\mathcal{M}$.

Corollary 3.2. *We can solve the 2-server problem in arbitrary metrics with N points with a competitive factor of $\mathcal{O}(log(N))$*

4 Approximation of Metric Spaces

Example: arbitrary metric by tree metric, distances stretched by almost $(N-1)$. With DC-algorithm $k(N-1)$ comp. algorithm.

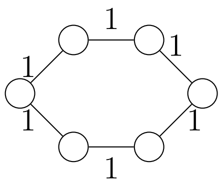
Definition 4.1. Let $\mathcal{M} = (M, d)$ be an arbitrary metric. We say a metric $\mathcal{M}' = (M', d')$ with $M \leq M'$ **dominates** M if $d(x,y) \leq d'(x,y)\ \forall x, y \in M$. Let S be a set of metrics, that dominate M and D a probability distribution over S. We say that (S, D) is an α-**approximation** of M if $\forall x, y \in M$

$$\underset{(M',d')\sim D}{E}[d'(x,y)] \leq \alpha \cdot d(x,y)$$

We also say that M is **embedded** in S and call α the stretch.

4.1 Approximations with Tree Metrics

In the deterministic way there can not be an embedding better than $\Omega(N)$. An embedding with the MST is asymptotically optimal. An example, where no asymptotic embedding is possible is a circle with edge-costs of 1.



Removing one edge gives a stretch of $N-1$. However in general one may add additional point.

Theorem 4.1. *For a metric $\mathcal{M}$ with N points, there is a set S of tree metrics that dominate $\mathcal{M}$ and probability distribution D over S, s.t. (S, D) is a $\mathcal{O}(log(N))$ approximation of $\mathcal{M}$. (S, D) can be computed efficiently.*

Proof. Let $\mathcal{M} = (V, d)$ an arbitrary metric with $N = |V|$ points. We assume that the minimal distance between two different points is greater than 1. Furthermore with Δ we denote the maximal distance between two points of V. Let δ such that $2^{\delta-1} < \Delta \leq 2^{\delta}$

Proof in two parts:

1. Recursive partitioning of V to generate tree metric
2. How to do it randomized to achieve stretch of $\mathcal{O}(log(N))$

1. Recursive Partitioning

Definition 4.2. A **partition** of a metric $\mathcal{M} = (M, d)$ with **radius** $r \geq 1$ is a partition of V in classes $V_1, \cdots, V_l$ such that for all sets V_i there exists a center $c_i \in V$ with $d(c_i, v) \leq r,\ \ \forall v \in V_i$. Note:

1. c_i does not need to be in V_i
2. diameter $\underset{x,y \in V_i}{max}\ d(x,y) \leq 2r$

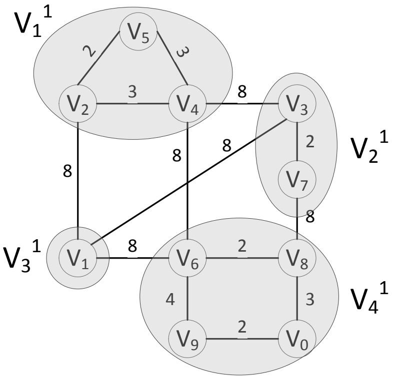
Definition 4.3. A hierarchical partitioning of $\mathcal{M} = (V, d)$ is a sequence $D_0, D_1, \cdots D_\delta$ of $\delta + 1$ partitions of V with the following properties:

1. $D_\delta = \{V\}$: trivial partition with radius of 2^δ
2. for all $i < \delta$, D_i is a partition of V with radius 2^i that refines D_{i+1}. That is, each class of D_i is a subset of a class of D_{i+1}

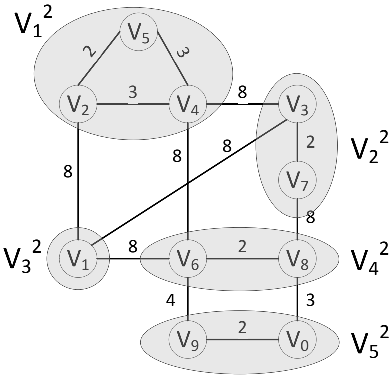
For such a partitioning $D_0, \cdots D_\delta$ we construct a tree metric:

- tree T, set of nodes are the classes of the partitions D_i
- root of T is class V (class of D_δ)
- nodes of level 1 are partitions of $D_{\delta-1}$
- nodes of level 2 are partitions of $D_{\delta-2}$
  $\cdots$
- leaves of T are partitions of D_0 which consists of N classes
  (Note: minimal distance > 1)
- edges of T: for every $i < \delta$ and every class X of D_i there is a class Y of D_{i+1} with $X \leq Y$. There is an edge between the two nodes representing X and Y with weight 2^{i+1}

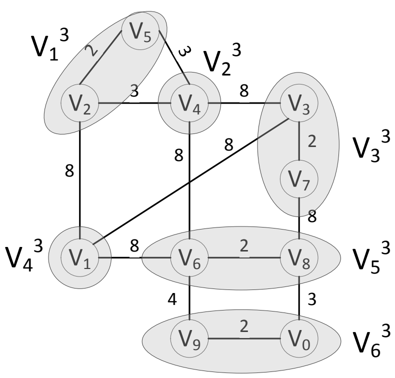
Example: $\Delta = 16$, $\delta = 4$
$D_4 = \{V\} = \{V_0, V_1, \cdots, V_9\}$



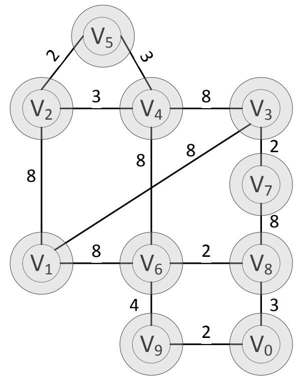
Level 1 of Partition
$D_3 = \{\{V_1^1\}, \{V_2^1\}, \{V_3^1\}, \{V_4^1\}\}$



Level 2 of Partition
$D_2 = \{\{V_1^2\}, \{V_2^2\}, \{V_3^2\}, \{V_4^2\}, \{V_5^2\}\}$

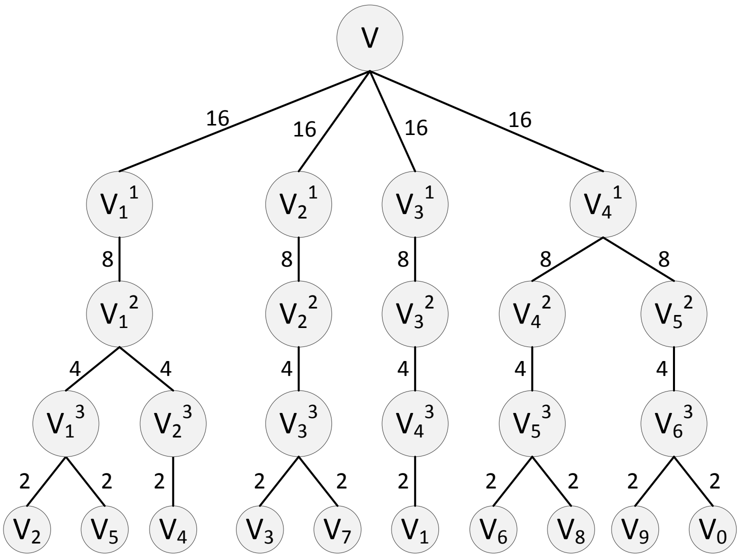


Level 3 of Partition
$D_1 = \{\{V_1^3\}, \{V_2^3\}, \{V_3^3\}, \{V_4^3\}, \{V_5^3\}, \{V_6^3\}\}$



Level 4 of Partition
$D_0 = \{\{V_0\}, \{V_1\}, \cdots, \{V_9\}\}$

There is a bisection between the leaves of T and V. We use T for a tree metric (V_T, d_T) over the set $V_T \geq V$ where $d_T(x, y)$ is defined as the path length in the tree T.



Lemma 4.1. *For every hierarchical partitioning of a metric $\mathcal{M}$ the resulting tree metric dominates $\mathcal{M}$.*

Proof. Let $x, y \in V$ be arbitrary points. The diameter of the classes of a partition D_i is at most 2^{i+1}. In all partitions D_i with $2^{i+1} < d(x, y)$ the points x, y are in different classes. In particular in partition D_j with

$$j = \lceil log_2 \, d(x, y) \rceil - 2 \ \text{ since } \ 2^{j+1} = 2^{\lceil log_2 \, d(x,y) \rceil - 1} < 2^{log_2 \, d(x,y)} = d(x, y)$$

On the path from the two leaves of T there must be two edges between classes of partitions D_j and D_{j+1}. Thus

$$d_T(x, y) \geq 2 \cdot 2^{j+1} = 2^{j+2} = 2^{\lceil log_2 \, d(x,y) \rceil} \geq d(x, y)$$

□

2. Randomised Partitioning

A randomized algorithm to compute a hierarchical partitioning. For a set $X \leq V$ and a point $v \in V$ and a radius $r \geq 1$, we denote by $B = (X, v, r)$ the **sphere** in X with radius r and center v. That is $B(X, v, r) = \{x \in X \mid d(x, v) \leq r\}$

Algorithm 1 HierPart ($\mathcal{M} = (V, d)$)

choose β uniformly at random from $[1, 2]$
choose a permutation π of the set $\{1, \cdots N\}$ uniformly at random
$D_\delta = \{V\}$
for $i = \delta - 1, i \geq 0, i - -$ **do**
  if D_{i+1} has a class with more than one element **then**
    $\beta_i = 2^{i-1} \cdot \beta$
    D_i =PARTITION($\mathcal{M}, D_{i+1}, \beta_i, \pi$)
  else
    $D_i = D_{i+1}$
end for
return $(D_0, D_1, \cdots, d_\delta)$

Algorithm 2 PARTITION ($\mathcal{M}, D, \alpha, \pi$)

$D' = \{\}$
for each class X in partition D **do**
  for $i = 1, 1 \leq N, i + +$ **do**
    $B_{\pi(i)} := B(X, V_{\pi(i)}, \alpha)$
    $X := x \setminus B_{\pi(i)}$
    if $B_{\pi(i)} \neq \emptyset$ **then**
      add $B_{\pi(i)}$ to D'
  end for
end for
return D'

- `PARTITION` considers class one after the other and partitions each class further
- For this it considers spheres with radius α around points of V chosen by the random ordery π
- Points of current class within such a sphere are new classes

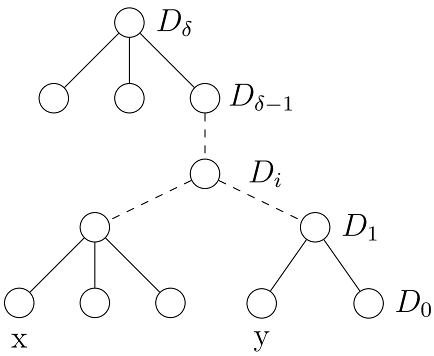
Lemma 4.2. *Let d_T be the tree metric constructed by algorithm `HierPart`($\mathcal{M} = (V, d)$). For every $x, y \in V$ it holds that*

$$E[d_T(x, y)] \leq 64 \cdot H_N \cdot d(x, y)$$

Proof. Let $x, y \in V$ be arbitrary points. Consider the tree T generated by hier. part., $D_0, D_1, \ldots, D_\delta$ of the algorithm `HierPart`.
Consider the path from x to y in T up to which level? If this level corresponds to D_i

- x and y are in different classes in $D_0, \ldots, D_{i-1}$
- x and y are in the same class in $D_i, \ldots, D_\delta$

- Let z_x and z_y be the centres around which `PARTITION` constructed the classes of D_{i-1} which contain x and y respectively



If z_x is before z_y in permutation π, we say that that point z_x separates $\{x, y\}$ on level $i-1$, otherwise we say that that point z_y separates $\{x, y\}$. For point $z \in V$ and every $j \in \{0, 1, \ldots, \delta - 1\}$ we denote by $A(z, j)$ the event that point z separates the pair $\{x, y\}$ on level j. There is exactly one point $z \in V$ and one level $j \in \{0, 1, \ldots, \delta - 1\}$ for which event $A(z, j)$ occurs. If event $A(z, j)$ occurs than

$$d_T(x, y) = 2 \cdot \sum_{i=1}^{j+1} 2^i \leq 2^{j+3}$$

Thus

$$E[d_T(x, y)] \leq \sum_{z \in V} \sum_{j=0}^{\delta-1} 2^{j+3} \cdot Pr[A(z, j)]$$

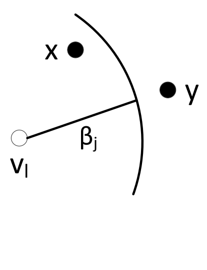
- Sort the points of V.
- For any $z \in V$ define $d(z, \{x, y\}) := min\{d(z, x), d(z, y)\}$
- Let $V = \{v_1, \ldots, v_N\}$ with $d(v_1, \{z, x\}) \leq d(v_2, \{z, x\}) \leq \cdots \leq d(v_N, \{z, x\})$

Lemma 4.3. *For every point $v_l \in V$ and every level $j \in \{0, 1, \cdots, \delta - 1\}$ it holds*

$$Pr[A(v_l, j)] \leq \frac{d(x, y)}{l \cdot 2^{j-1}}$$

Proof. w.l.o.g. $d(v_l, x) \leq d(v_l, y)$. If v_l separates $\{x, y\}$ on level j, the following two conditions must be true:

1. when constructing partition D_j, the sphere around v_l (line 4, `PARTITION`) is the first sphere containing x or y
2. The radius $\beta_j = 2^{j-1}\beta$ is in the interval $[d(v_l, x), d(v_l, y)]$ otherwise the sphere would contain neither or both points



Probability for 2:

$$
\begin{aligned}
&Pr[\beta_j \in [d(v_l, x), d(v_l, y)]] \\
&= Pr[\beta_j \in \left[\frac{(v_l, x)}{2^{j-1}}, \frac{d(v_l, y)}{2^{j-1}}\right]] \\
&\leq \frac{d(v_l, y)}{2^{j-1}} - \frac{(v_l, x)}{2^{j-1}} \\
&\leq \frac{d(x, y)}{2^{j-1}}
\end{aligned}
$$

Note:
$\beta \in [1, 2]$ probability:
$\beta \in I \leq |I \cap [1, 2]| \leq |I|$

If β_j is in the interval such that condition 2. is fulfilled, then 1. can only occur if v_l is before $v_1, \cdots, v_{l-1}$ in permutation π, otherwise a sphere around on of those points with radius β_j would contain at least one of $\{x, y\}$. Probability for v_l of being in front in π is $\frac{1}{l}$.
Combining both probabilities we can bound the probability for the event $A(v_l, j)$ by $\frac{1}{l} \cdot \frac{d(x,y)}{2^{j-1}}$ □

Using Lemma 4.3. we can bound

$$
\begin{aligned}
E[d_T(x, y)] &\leq \sum_{l=1}^{N} \sum_{j=0}^{\delta-1} 2^{j+3} \cdot Pr[A(v_l, j)] \\
&\leq \sum_{l=1}^{N} \sum_{j=0}^{\delta-1} 2^{j+3} \cdot \frac{d(x, y)}{l \cdot 2^{j-1}} \\
&\leq 16 \cdot \delta \cdot H_N \cdot d(x, y)
\end{aligned} \tag{2}
$$

Lemma 4.4. *For every vertex v_l there are at most four levels $j \in \{0, 1, \ldots \delta - 1\}$ for which event $A(v_l, j)$ can occur.*

Proof. w.l.o.g. let $d(v_l, x) \leq d(v_l, y)$

1. Case: $d(x, y) \leq d(v_l, x)$
   - Then $d(v_l, x) \geq d(v_l, y) - d(x, y) \geq d(v_l, y) - d(v_l, x)$
   - Thus $d(v_l, x) \geq \frac{d(v_l,y)}{2}$
   - now let j be the largest value from $\{0, 1, \ldots \delta - 1\}$ such that the interval $[2^{j-1}, 2^j]$ (from which β_j is chosen) has a non-empty intersection with the interval $[d(v_l, x), d(v_l, y)]$ (in which β_j has to lie if $A(v_l, j)$ occurs)
   - Therefore $d(v_l, y) > 2^{j-1}$ and $d(v_l, x) \geq \frac{d(v_l,y)}{2} > 2^{j-2}$
   - Thus in partition D_{j-2} vertex v_l cannot separate $\{x, y\}$ and since j was chosen to be the largest value, event $A(v_l, i)$ can only occur for $i \in \{j - 1, j\}$
2. Case: $d(x, y) > d(v_l, x)$
   - Then $d(x, y) \geq d(v_l, y) - d(v_l, x) > d(v_l, y) - d(x, y)$
   - This implies $d(x, y) > \frac{d(v_l,y)}{2}$
   - Let j be chosen as in 1. Case.

- Then $d(v_l, y) > 2^{j-1}$ and thus $d(x, y) > \frac{d(v_l,y)}{2} > 2^{j-2}$ which means that in partition D_{j-3} x and y have to belong to different classes, since each class has diameter at most 2^{j-2}
- Thus v_l cannot separate $\{x, y\}$ on a level $i \leq j - 4$
- Since we chose j to be the largest value, event $A(v_l, i)$ can only occur for $i \in \{j - 3, j - 2, j - 1, j\}$ □

Using Lemma 4.4. we can bound equation (2) since there are at most four values of j for which $Pr[A(v_l, j)] > 0$ for every l.

$$
\begin{aligned}
E[d_T(x, y)] &\leq \sum_{l=1}^{N} \sum_{j=0}^{\delta-1} 2^{j+3} \cdot Pr[A(v_l, j)] \\
&\leq \sum_{l=1}^{N} 4 \cdot \frac{16 \cdot d(x, y)}{l} \\
&\leq 64 \cdot H_N \cdot d(x, y)
\end{aligned}
$$

□

We have shown: Every metric can be embedded into a tree metric with stretch of $\mathcal{O}(log(N))$ □

Observation: For every tree metric $\mathcal{M}_T = (V_T, d_T)$ generated by above algorithm the following hold

$$\max_{x,y \in V_T} d_T(x, y) \leq 8 \cdot \max_{x,y \in V} d(x, y)$$

Proof. We define

$$\Delta = \max_{x,y \in V} d(x, y) \text{ and } \delta \in \mathbb{N}$$

such that

$$2^{\delta-1} < \Delta \leq 2^{\delta}$$

The longest path in T:

$$2 \cdot \sum_{j=1}^{\delta} 2^j \leq 2^{\delta+2} < 8\Delta$$

□

Theorem 4.2. *There is a randomised online algorithm for the k-server-problem which is $\mathcal{O}(k \cdot log(N))$-competitive for every metric with N points.*

Proof. Input σ, Metric $\mathcal{M} = (M, d)$.

- Construct a $\mathcal{O}(log(N))$-approximation (S, D) with the algorithm above and choose a tree metric $\mathcal{M}_T$ from S according to D.
- Interpret σ as input for $\mathcal{M}_T$ (Note: $\mathcal{M} \leq \mathcal{M}_T$) and use DC-algorithm.
- Let $OPT(\sigma)$ and $OPT_T(\sigma)$ be optimal offline solution for metric $\mathcal{M}$ and $\mathcal{M}_T$ respectively.
- $DC_T(\sigma)$ is the solution of the DC-algorithm

- $d(L)$ and $d_T(L)$ cost of a solution using metric d and d_T respectively.

$$
\begin{aligned}
E[d(DC_T(\sigma)] &\leq E[d_T(DC_T(\sigma))] \\
&\leq E[k \cdot d_T(OPT_T(\sigma)) + \tau] \\
&\leq k \cdot E[d_T(OPT_T(\sigma))] + \tau \\
&\leq k \cdot E[d_T(OPT(\sigma))] + \tau \\
&\leq k \cdot \mathcal{O}(log(N)) \cdot d(OPT(\sigma)) + \tau
\end{aligned}
$$

□

5 Scheduling

- Set of jobs $J = \{1, \dots n\}$
- Set of machines $M = \{1, \dots m\}$
- Each job $j \in J$ has a size $p_j \in \mathbb{R}_{>0}$
- Each machine $i \in M$ has a speed $s_i \in \mathbb{R}_{>0}$
- if a job $j \in J$ is processed by machine $i \in M$ it takes time $\frac{p_j}{s_i}$
- A schedule $\pi : J \to M$ assigns each job to a machine
- $L_i(\pi)$ is the load of machine $i \in M$ in schedule π

$$L_i(\pi) = \frac{\sum\limits_{j \in M, \pi(j)=i} p_j}{s_i}$$

- **Makespan** $C(\pi)$ is the maximal load i.e.

$$C(\pi) = \max_{i \in M} L_i(\pi)$$

- In the following we seek to minimize the makespan.

Online Scheduling

- Set of machines and speed are unknown
- jobs arrive one after another
- job have to be assigned immediately to a machine
- number and size of future jobs are unknown

5.1 Identical Machines

- All machines have speed 1
- Greedy-strategy aka Least-Loaded-algorithm
  → assigns each job to the machine that has currently the smallest load

Theorem 5.1. *The Least-Loaded-algorithm is strict $2 - \frac{1}{m}$-competitive*

Proof. Lower bound for optimal schedule π^* :

$$C(\pi^*) \geq \frac{1}{m} \sum_{j \in J} p_j \text{ and } C(\pi^*) \geq \max_{j \in J} p_j$$

Schedule π of least-loaded: Let $i \in M$ be the machine with maximal load $C(\pi) = L_i(\pi)$. Let $j \in J$ be the last job that was added to i: At that time i was the least-loaded machine: The load is at most $\frac{1}{m} \sum\limits_{k=1}^{j-1} p_k$

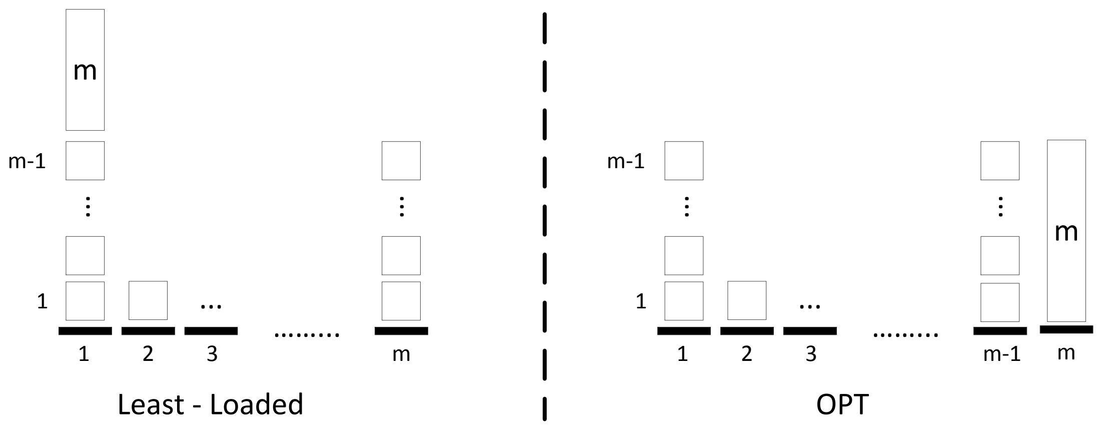
$$
\begin{aligned}
C(\pi) = L_i(\pi) &\leq \frac{1}{m}(\sum_{k=1}^{j-1} p_k) + p_j \\
&\leq \frac{1}{m}(\sum_{k \in J \setminus \{j\}} p_k) + p_i \\
&= \frac{1}{m}\sum_{k \in J} p_k + (1 - \frac{1}{m})p_j \\
&\leq C(\pi^*) + (1 - \frac{1}{m}) \cdot \max_{k \in J} p_k \\
&\leq (2 - \frac{1}{m}) \cdot C(\pi^*)
\end{aligned}
$$

□

Lower bound for Least-Loaded

Let m be the number of machines and an input instance with $n = m(m-1)+1$ jobs. The first $m(m-1)$ jobs have size 1 and the last job has size m. The Least-Loaded schedules the smallest jobs equally on all machines, i.e. $(m-1)$ jobs on each machine and the last job on an arbitrary machine. The load on this machine is $(m-1)+m = 2m-1$. OPT would schedule m jobs of size 1 on each of the machines $1 \ldots m-1$ and then the job of size m on machine m. The makespan is m.

$$
\frac{\text{Least-Loaded}}{\text{OPT}} = \frac{2m-1}{m} = 2 - \frac{1}{m}
$$



5.2 Machines with Speed

What about greedy? 2 variants

1. choose the machine that has smallest load before scheduling current job
2. choose machine that has smallest load after assigning the job

Example:

- current loads: $M_1 = 1, M_2 = 0$
- new job $p_2 = 3$
  1. assigns job to $M_2 \Rightarrow$ Loads: $M_1 = 1, M_2 = 3 \succ 4$
  2. assigns job to $M_1 \Rightarrow$ Loads: $M_1 = 2, M_0 = 3 \succ 2$

If we make s_1 arbitrary large then variant (1) creates an arbitrary bad solution. For variant (2) it can be shown that the competitive factor is $\Theta(log(m))$

Slow Fit

Algorithm with constant competitive factor.
Assume we know the makespan of the optimal solution. Let $\alpha = OPT(\sigma)$
SlowFit(α) computes a schedule π with $C(\pi) \leq 2\alpha$

- sort machines according to their speeds in increasing order, i.e. $s_1 \leq s_2 \leq \ldots \leq s_m$
- Let π_j be the partial schedule computed by SlowFit(α) for the jobs $1 \ldots j$

Algorithm 3 SlowFit (α)

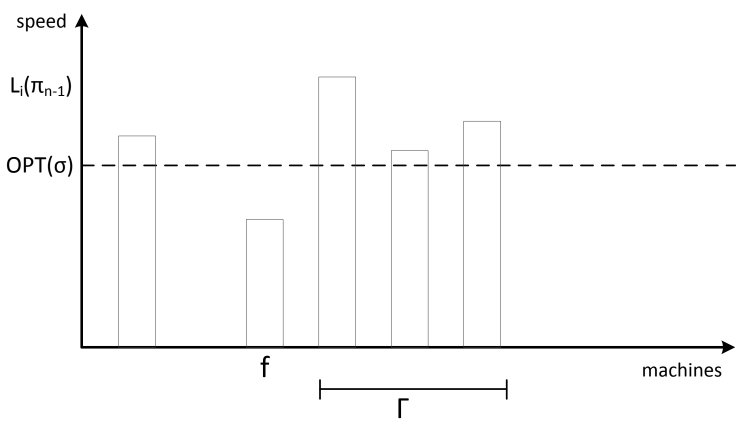
---

schedule a new job $j \in J$ with size p_j to the slowest machine $i \in M$ which has load of less than 2α after this assignment, i.e.
$min\{i \in M \mid L_i(\pi_{j-1} + \frac{p_j}{s_i} \leq 2\alpha\}$
if no such machine exists output an error-message

Lemma 5.1. *Let $\alpha \in \mathbb{R}_{\geq 0}$ be arbitrary and σ be an arbitrary input with $OPT(\sigma) \leq \alpha$ then SlowFit(α) produces no error and computes a schedule π with $C(\pi) \leq 2\alpha$*

Proof. It suffices that SlowFit(α) does not output an error-message. Assume there is an input $\sigma = (p_1, \ldots, p_n)$ and SlowFit(α) outputs error at job p_n
First observe that not for all $i \in M$ $L_i(\pi_{n-1}) > OPT(\sigma)$ since otherwise

$$\sum_{j=1}^{n-1} p_j = \sum_{i \in M} s_i L_i(\pi_{n-1}) > \sum_{i \in M} s_i \cdot OPT(\sigma) \geq \sum_{i \in M} s_i \cdot L_i(\pi^*) = \sum_{j=1}^{n} p_j \;\lightning$$



Consider the fastest machine $f \in M$ with $L_f(\pi_{n-1}) \leq OPT(\sigma)$. Observe that $f < m$

because otherwise the following would hold:

$$L_m(\pi_{n-1}) + \frac{p_m}{s_m} \leq 2 \cdot OPT(\sigma) \leq 2\alpha$$

and there would be no error. Let $\Gamma = \{i \in M \mid i > f\}$. All machines in Γ have load $\geq OPT$ and $\Gamma \neq \emptyset$. The total size of jobs on machines m in Γ

$$\sum_{i \in \Gamma} s_i \cdot L_i(\pi_{n-1}) > \sum_{i \in \Gamma} s_i \cdot OPT(\sigma)$$

There must exist a job $j \in J \setminus \{n\}$ with $\pi_{n-1}(j) \in \Gamma$ and $\pi^*(j) = i$ and $i \notin \Gamma$

$$\frac{p_j}{s_i} \leq OPT(\sigma) \text{ and } i \leq f$$

Due to sorting of speeds also

$$\frac{p_j}{s_f} \leq OPT(\sigma)$$

Consider the event when j was scheduled by SlowFit(α). It could have been scheduled to machine f since:

$$L_f(\pi_{j-1}) + \frac{p_j}{s_f} \leq L_f(\pi_{n-1}) + \frac{p_j}{s_f} \leq OPT(\sigma) + OPT(\sigma) \leq 2\alpha$$

But it was scheduled to a faster machine in Γ which is a contradiction to the definition of the algorithm. □

But we do not know $OPT(\sigma)$:

Algorithm 4 SlowFit

1: Set $\alpha_0 = \frac{p_1}{s_m}$
2: Start with phase $k = 0$
3: **for** job j **do**
4: Try to schedule j with `SlowFit`(α_k) while ignoring all jobs of previous phases
5: **if** `SlowFit`(α_k) produces an error **then**
6: increase k by 1
7: Set $\alpha_k = 2^k \cdot \alpha_0$ and go to step 4
8:
9: **end for**

Theorem 5.2. *SlowFit is strict 8-competitive for online scheduling.*

Proof. Let $0, 1, \ldots h$ be the phases of SlowFit for an arbitrary input σ. By σ_k we denote the subsequence of jobs of phase k. Using Lemma 5.1. we obtain a lower bound for OPT:

- if $h = 0$: $OPT \geq \alpha_0$ and SlowFit is 2-competitive

- if $h > 0$: consider the phase $h-1$ and the first job j of phase h. Since we ignored all jobs of phases before $h-1$ SlowFit(α_{h-1}) produces an error when processing job j only if for subsequence

$$\sigma_{h-1}: \; OPT(\sigma_{h-1}, j) > \alpha_{h-1} = 2^{h-1}\alpha_0$$

  Upper bound of schedule π of SlowFit: Summing up over the makespan of the phases

$$C(\pi) \leq \sum_{k=0}^{h} 2\alpha_k = 2 \cdot \sum_{k=0}^{h} 2^k \alpha_0 \leq 2^{h+2}\alpha_0$$

  Combining both equations:

$$C(\pi) \leq 2^{h+2}\alpha_0 = 8 \cdot 2^{h-1}\alpha_0 \leq 8 \cdot OPT(\sigma_{n-1}j) \leq 8 \cdot OPT(\sigma)$$

□

Remarks:

- best known online algorithm is 5,828-competitive
- lower bound is 2,438

6 Summary

1. Introduction
   - competitive ratio; strict competitive ratio
2. Paging
   - Deterministic
     - marking algorithms: LRU is one (Proof this)
     - marking algorithm is k-competitive
     - LFD is optimal
     - lower bound of k for deterministic algorithms
   - Random
     - 3 types of adversaries
     - redefinition of competitive ratio
     - `RANDOM` k-competitive (Proof with potential function, amortized costs)
     - lower bound of k for `RANDOM`
     - `MARK`: randomised version of marking algorithm, $2H_k$-competitive ratio (Proof)
     - lower bound of H_k for `MARK`
3. k-Server-Problem
   - greedy-algorithm bad idea
   - computing optimal offline solution with reduction to Min-Cost-Flow in polynomial time (be able to do this reduction in exam)
   - lower bound for deterministic online algorithm, OPT via indirect proof, classes of algorithms
   - DC on the line algorithm, k-competitive (know potential function and general steps of proof)
   - DC on trees, same potential function f, proof only differs for movement of DC
   - 2-servers in arbitrary spaces
     - Slack Cover
     - $SC_{\frac{1}{2}}$
     - potential function method
     - case distinction (what do we have to show, which cases and outcome)
4. Approximation of Metric Spaces
   - dominate, embedding, deterministic is not a good idea
   - probabilistic embeddings
   - tree embedding
     1. hierarchical partitioning → tree metric, dominates (be able to proof)

2. generating `HierPart` algorithm, subroutine `PARTITION`
   Proof: Exp. dist(x,y), probability that they get separated
   depends on level and permutation
   last step: $\delta \to 4$ levels

5. Scheduling

   - identical machines
   - $2 - \frac{1}{2}$-competitive Least-Loaded (be able to write down complete proof)
   - lower bound
   - `SlowFit`
     - we assume OPT
     - "guess" OPT