Sommersemester 2016

Online and Approximation Algorithms

http://wwwalbers.in.tum.de/lehre/2016SS/oa/index.html.en

Susanne Albers

Fakultät für Informatik

TU München

0. Organizational matters

Lectures:	4 SWS	
	Wed	08:00–10:00, MI 00.13.009A
	Fri	08:00–10:00, MI 00.13.009A
Exercises:	2 SWS	
	Wed	12:00–14:00, 01.07.023

Teaching assistant: Dr. S. van der Ster, D. Frascaria
ster@in.tum.de
frascari@informatik.tu-muenchen.de

Bonus: If at least 50% of the maximum number of points of the homework assignments are attained and student presents the solutions of at least two problems in the exercise sessions, then the grade of the final exam can be improved by 0.3 (or 0.4).

0. Organizational matters

Valuation: 8 ECTS (4 + 2 SWS)

Office hours: by appointment (albers@informatik.tu-muenchen.de)

0. Organizational matters

Problem sets: Made available on Friday by 10:00 on the course webpage. Must be turned in one week later before the lecture.

Exam: Written exam; no auxiliary means are permitted, except for one hand-written sheet of paper

Prerequisites: Grundlagen: Algorithmen und Datenstrukturen (GAD)
Diskrete Wahrscheinlichkeitstheorie (DWT)

Effiziente Algorithmen und Datenstrukturen
(advantageous but not required)

0. Literature

- [BY] A. Borodin und R. El-Yaniv. Online Computation and Competitive Analysis. Cambridge University Press, Cambridge, 1998. ISBN 0-521-56392-5

- [V] V.V. Vazirani. Approximation Algorithms. Springer Verlag, Berlin, 2001. ISBN 3-540-65367-8

- Handouts

0. Content

Online and approximation algorithms

Optimization problems for which the computation of an optimal solution is hard or impossible.

Have to resort to approximations:
Design algorithms with a provably good performance.

0. Content

Online algorithms

- Scheduling
- Paging
- List update
- Randomization
- Data compression
- Robotics
- Matching

0. Content

Approximation algorithms

- Traveling Saleman Problem
- Knapsack Problem
- Scheduling (makespan minimization)
- SAT (Satisfiability)
- Set Cover
- Hitting Set
- Shortest Superstring

1. Introduction

Online and approximation algorithms

Optimization problems for which the computation of an optimal solution is hard or impossible.

Have to resort to approximations:
Design algorithms with a provably good performance.

1.1 Online problems

Relevant input arrives incrementally over time. Online algorithm has to make decisions without knowledge of any future input.

1. Ski rental problem: Student wishes to pick up the sport of skiing.

   Renting equipment: 10$ per season

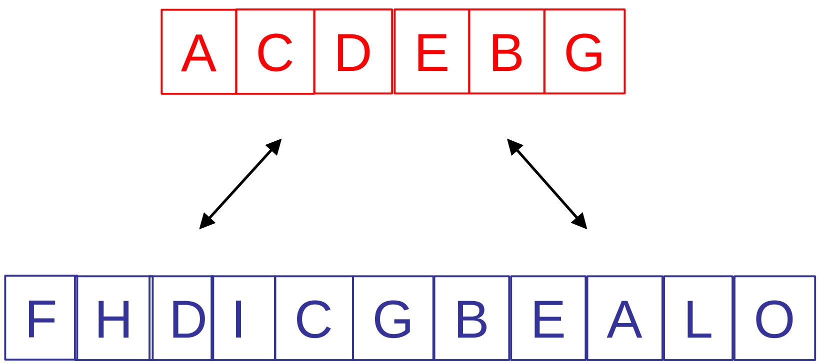
   Buying equipment: 100$

   Do not know how long (how many seasons) the student will enjoy skiing.

2. Currency conversion: Wish to convert 1000$ into Yen over a certain time horizon.

1.1 Online problems

3. Paging/caching: Two-level memory system



small fast memory

large slow memory

σ = A C B E D A F …

Request: Access to page in memory system

Page fault: requested page not in fast memory; must be loaded into fast memory

Goal: Minimize the number of page faults

1.1 Online problems

4. Data structures: List update problem

Unsorted linear list

L: B → C → E → A → D

σ = A A C B E D A …

Request: Access to item in the list

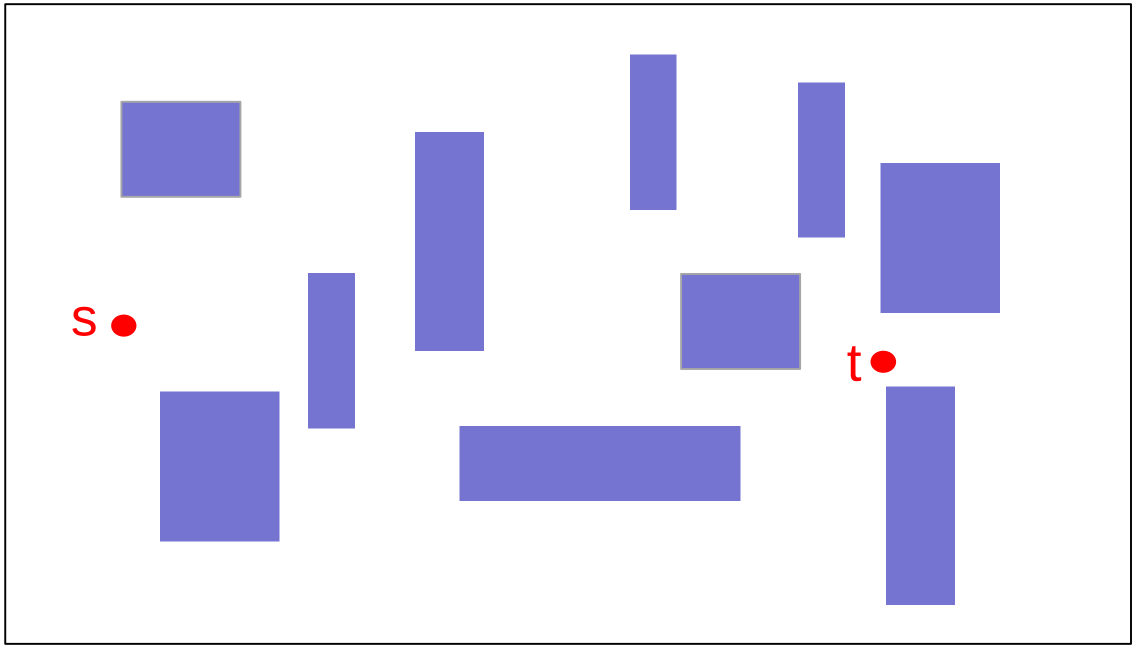
Cost: Accessing the i-th item in the list incurs a cost of i.

Rearrangements: After an access, requested item may be moved at no extra cost to any position closer to the front of the list (free exchanges). At any time two adjacent items may be exchanged at a cost of 1.

Goal: Minimize cost paid in serving σ.

1.1 Online problems

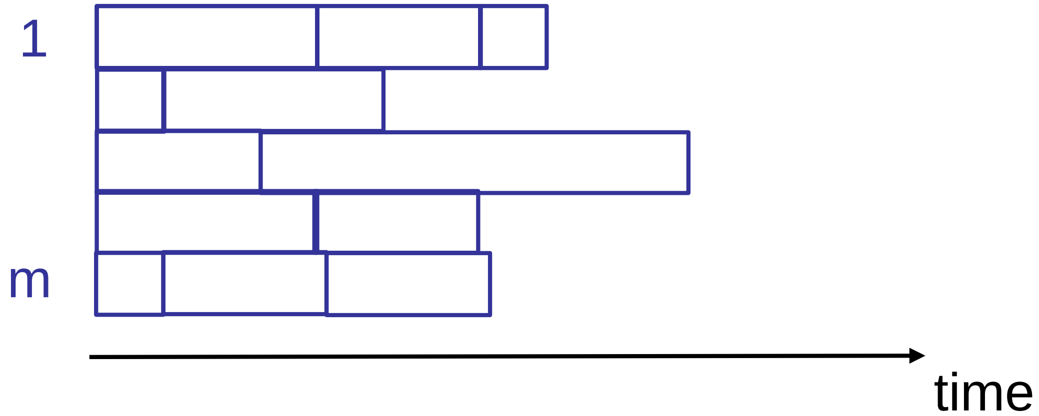
5. Robotics: Navigation



Unknown scene: Robot has to find a short path from s to t.

1.1 Online problems

6. Scheduling: Makespan minimization



m identical parallel machines

Input portion: Job J_i with individual processing time p_i.

Goal: Minimize the completion time of the last job in the schedule.

1.2 NP-hard optimization problems

Assuming P ≠ NP, NP-hard optimization problems cannot be solved optimally in polynomial time.

1. Scheduling: Makespan minimization (see above)

   Entire job sequence is known in advance. Famous optimization problem studied by Ronald Graham in 1966.

2. Traveling Salesman Problem: n cities, $c(i,j)$ = cost/distance to travel from city i to city j, $1 \leq i,j \leq n$.

   Goal: Find tour that visits each city exactly once and minimizes the total cost.

1.2 NP-hard optimization problems

3. Knapsack Problem: n items with individual weights $w_1, \ldots, w_n \in \mathbb{N}$ and values $a_1, \ldots, a_n \in \mathbb{N}$. Knapsack of total weight (capacity) W.

   Goal: Find a feasible packing, i.e. a subset of the items whose total weight does not exceed W, that maximizes the value obtained.

4. Max SAT: n Boolean variables $\{x_1, \ldots, x_n\}$ with associated literals $\{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ and clauses $C_1, \ldots, C_m$. Each clause is a disjunction of literals.

   Goal: Find an assignment of the variables that maximizes the number of satisfied clauses.

1.2 NP-hard optimization problems

- 5. Shortest Superstring: Finite alphabet Σ, n strings $\{s_1, \dots, s_n\} \subseteq \Sigma^+$.
  Goal: Find shortest string that contains all s_i as substring.

2. Online algorithms

Formal model:

A

Online algorithm has to serve σ.

$$\sigma = \sigma(1)\ \sigma(2)\ \sigma(3)\ \ldots.\ \sigma(t)\ \sigma(t+1)\ \ldots$$

Each request σ(t) has to be served without knowledge of any future requests.

Goal: Optimize a desired objective, typically the total cost incurred in serving σ.

2. Competitive analysis

Online algorithm A is compared to an optimal offline algorithm OPT that knows the entire input σ in advance and can serve is optimally, with minimum cost.

	A	⟷	OPT
Cost:	A(σ)		OPT(σ)

Online algorithm A is called c-competitive if there exists a constant a, which is independent of σ, such that

$$A(\sigma) \le c \cdot OPT(\sigma) + a$$

holds for all σ.

2.1 Scheduling

Makespan minimization: m identical parallel machines.

n jobs $J_{1, \dots, }J_n$. p_t = processing time of J_t, $1 \le t \le n$

Goal: Minimize the makespan

Algorithm Greedy: Schedule each job on the machine currently having the smallest load.

Algorithm is also referred to as *List Scheduling*.

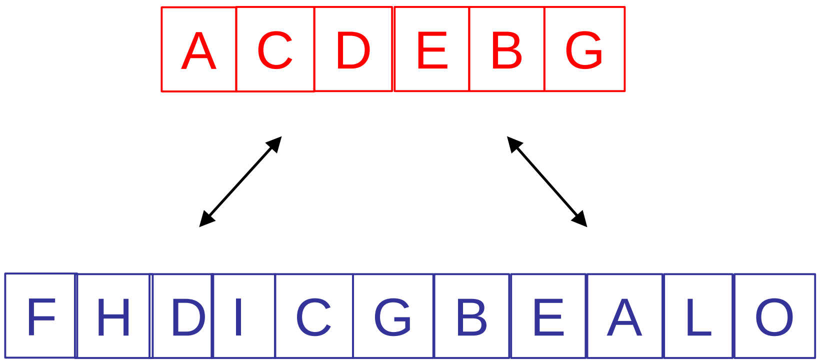
Theorem: Greedy is (2-1/m)-competitive.

Theorem: The competitive ratio of Greedy is not smaller than 2-1/m.

See e.g. [BY], page 205.

2.2 Paging

Two-level memory system



small fast memory

large slow memory

σ = A C B E D A F …

Request: Access to page in memory system

Page fault: requested page not in fast memory; must be loaded into fast memory

Goal: Minimize the number of page faults

2.2 Paging

Popular online algorithms

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

Let k be the number of pages that can simultaneously reside in fast memory.

Theorem: LRU and FIFO are k-competitive.

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

2.2 Paging

Marking algorithms: Serve a request sequence in phases. First phase starts with the first request. Any other phase starts with the first request following the end of the previous phase.

At the beginning of a phase all pages are unmarked. Whenever a page is requested, it is marked. On a fault evict an arbitrary unmarked page in fast memory. If no such page is available, the phase ends and all marks are erased.

Flush-When-Full: If there is a page fault and there is no empty slot in fast memory, evict all pages.

2.2 Paging

Offline algorithm

- MIN: On a page fault evict the page whose next request is farthest in the future.

Theorem: MIN is an optimal offline algorithm for the paging problem, i.e. it achieves the smallest number of page faults/page replacements.

See [BY], pages 33 – 38.

2.2 Paging

An algorithm is a *demand paging* algorithm if it only replaces a page in fast memory if there is a page fault.

Fact: Any paging algorithm can be turned into a demand paging algorithm such that, for any request sequence, the number of memory replacements does not increase.

2.3 Amortized analysis

General concept to analyze the cost of a sequence of operations executed, for instance, on a data structure.

Wish to show: An individual operation can be expensive, but the average cost of an operation is small.

Amortization: Distribute cost of a sequence of operations properly among the operations.

Example: Binary counter with increment operation. Cost of an operation is equal to the number of bit flips.

2.3 Amortized analysis, binary counter

Operation	Counter value	Cost
	00000	
1	00001	1
2	00010	2
3	00011	1
4	00100	3
5	00101	1
6	00110	2
7	00111	
8	01000	
9	01001	
10	01010	
11	01011	
12	01100	
13	01101	

2.3 Amortized analysis

- Potential function technique

$$\Phi : \text{Config } D \rightarrow \mathbb{R}$$

It will be convenient if $\Phi(t) \geq 0$ and $\Phi(0) = 0$

Actual cost of operation t: $a(t)$

Amortized cost of operation t: $a(t) + \Phi(t) - \Phi(t-1)$

The goal is to show that for all t:

$$a(t) + \Phi(t) - \Phi(t-1) \leq c$$

2.3 Amortized competitive analysis

Given σ, wish to show $A(\sigma) \leq c \cdot OPT(\sigma)$

Potential: Φ : (Config A, Config OPT) $\rightarrow \mathbb{R}$

Again, it will be convenient if $\Phi(t) \geq 0$ and $\Phi(0) = 0$

A's actual cost of operation t: $A(t)$

A's amortized cost of operation t: $A(t) + \Phi(t) - \Phi(t-1)$

OPT's actual cost of operation t: $OPT(t)$

The goal is to show that for all t:

$$A(t) + \Phi(t) - \Phi(t-1) \leq c \cdot OPT(t)$$

2.4 List update problem

Unsorted, linear linked list of items

L: B → C → E → A → D

σ = A A C B E D A …

Request: Access to item in the list

Cost: Accessing the i-th item in the list incurs a cost of i.

Rearrangements: After an access, requested item may be moved at no extra cost to any position closer to the front of the list (free exchanges). At any time two adjacent items may be exchanged at a cost of 1.

Goal: Minimize cost paid in serving σ.

2.4 List update problem

Online algorithms

- Move-To-Front (MTF): Move requested item to the front of the list.
- Transpose: Exchange requested item with immediate predecessor in the list.
- Frequency Count: Store a frequency counter for each item in the list. Whenever an item is requested, increase its counter by one. Always maintain the items of the list in order of non-increasing counter values.

Theorem: MTF is 2-competitive.

Theorem: Transpose and Frequency Count are not c-competitive, for any constant c.

Theorem: Let A be a deterministic list update algorithm. If A is c-competitive, for all list lengths, then $c \geq 2$.

See [BY], pages 7 – 13.

2.5 Randomized online algorithms

A = randomized online algorithm

A(σ) random variable, for any σ

Competitive ratio of A defined w.r.t. an adversary ADV who

- generates σ
- also serves σ

ADV knows the description of A

Critical question: Does ADV know the outcome of the random choices made by A?

2.5 Randomized online algorithms

Oblivious adversary:

$$\boxed{A}$$

$$\downarrow$$

$$\sigma = \underbrace{\sigma(1)\ \sigma(2)\ \sigma(3)\ \dots.\ \sigma(m)}_{\boxed{\text{ADV}}}$$

Does not know the outcome of the random choices made by A.

Generates the entire σ in advance.

Adaptive adversary:

A

$\sigma = \sigma(1)\ \sigma(2)\ \sigma(3)\ \ldots.\ \sigma(t-1)$

$\sigma(t)$

ADV

Does know the outcome of the random choices made by A on the first t-1 requests when generating $\sigma(t)$.

Adaptive online adversary: Serves σ online.

Adaptive offline adversary: Serves σ offline.

2.5 Three types of adversaries

Oblivious adversary: Does not know the outcome of A‘s random choices; serves σ offline. A is c-competitive against oblivious adversaries, if there exists a constant a such that

$$E[A(\sigma)] \leq c \cdot ADV(\sigma) + a$$

holds for all σ generated by oblivious adversaries. Constant a must be independent of input σ.

Adaptive online adversary: Knows the outcome of A‘s random choices on first t-1 requests when generating σ(t); serves σ online. A is c-competitive against adaptive online adversaries, if there exists a constant a such that

$$E[A(\sigma)] \leq c \cdot E[ADV(\sigma)] + a$$

holds for all σ generated by adaptive online adversaries. Constant a must be independent of input σ.

2.5 Three types of adversaries

Adaptive offline adversary: Knows the outcome of A‘s random choices on first t-1 requests when generating σ(t); serves σ offline. A is c-competitive against adaptive offline adversaries, if there exists a constant a such that

$$E[A(\sigma)] \leq c \cdot E[OPT(\sigma)] + a$$

holds for all σ generated by adaptive offline adversaries. Constant a must independent of input σ.

2.5 Relating the adversaries

Theorem: If there exists a randomized online algorithm that is c-competitive against adaptive offline adversaries, then there also exists a c-competitive deterministic online algorithm.

Theorem: If A is c-competitive against adaptive online adversaries and there exists a d-competitive algorithm against oblivious adversaries, then there exists a cd-competitive algorithm against adpative offline adversaries.

Corollary: If A is c-competitive against adaptive online adversaries, then there exists a c^2-competitive deterministic algorithm.

2.6 Randomized paging

- **Algorithm RMARK:** Serve σ in phases.
- At the beginning of a phase all pages are unmarked.
- Whenever a page is requested, it is marked.
- On a page fault, choose a page uniformly at random from among the unmarked pages in fast memory and evict it.

A phase ends when there is a page fault and the fast memory only contains marked pages. Then all marks are erased and a new phase is started (first request is the missing page that generated the fault).

Theorem: RMARK is $2H_k$-competitive against oblivious adversaries.

Here $H_k = \sum_{i=1}^{k} 1/i$ is the k-th Harmonic number.

2.6 Randomized paging

Theorem: Let A be a randomized online paging algorithm. If A is c-competitive against oblivious adversaries, then $c \geq H_k$.

See e.g. [BY], pages 49-53.

2.6 Randomized paging

- Will develop an alternative proof for the lower bound based on Yao's minimax principle. The latter is based on von Neumann's minimax theorem.

Informally: Performance of best randomized algorithm is equal to the performance of the best deterministic algorithm on a worst-case input distribution.

Let P be a probability distribution on possible inputs (request sequences).

Let A be any deterministic online algorithm. The competitive ratio c_A^P of A given P is the infimum of all c such that

$$E[A(\sigma)] \leq c \cdot E[OPT(\sigma)] + a$$

where σ is generated according to P.

2.6 Randomized paging

Theorem: Yao's Minimax Principle

$$\inf_R c_R = \sup_P \ \inf_A c_A^P$$

where c_R is the competitive ratio of randomized algorithm R.

Other performance measure is running time:

$$\inf_R T_R = \sup_P \ \inf_A T_A^P$$

T_R = expected worst-case running time of randomized algorithm R

T_A^P = expected running time of deterministic algorithm A if input is generated according to P.

2.6 Randomized paging

Theorem: Let A be a randomized online paging algorithm. If A is c-competitive against oblivious adversaries, then $c \geq H_k$.

See e.g. [BY], pages 120-122.

2.6 Randomized paging

Online algorithm

- Random: On a fault evict a page chosen uniformly at random from among the pages in fast memory.

Theorem: Random is k-competitive against adaptive online adversaries.

Theorem: Let A be a randomized online paging algorithm. If A is c-competitive against adaptive online adversaries, then $c \geq k$.

See e.g. [BY], page 47.

2.7 Refinements of competitive paging

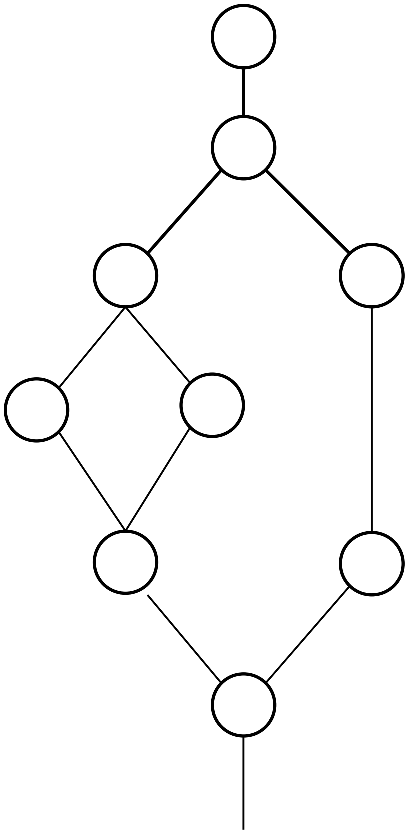
Deficiencies of competitive analysis:

- Competitive ratio of LRU/FIFO much higher than ratios observed in practice (typically in the range [1,2]).
- In practice LRU much better than FIFO

Reason: Request sequences in practice exhibit locality of reference, i.e. (short) subsequences reference few distinct pages.

2.7 Refined models

1. Access graph model: G(V,E) undirected graph. Each node represents a memory page. Page p can be referenced after q if p and q are adjacent in the access graph.



Competitive factors depend on G.

$\forall\, G: c_{LRU}(\mathrm{G}) \leq c_{FIFO}(\mathrm{G})$

$\forall\, T: c_{LRU}(\mathrm{T})$ smallest possible ratio

Problem: quantify $c_A(\mathrm{G})$ for arbitrary G

2.7 Refined models

2. Markov paging: n pages

q_{ij} = probability that request to page i is followed by request to page j

$$Q = \begin{pmatrix} q_{11} & \cdots & q_{1n} \\ \vdots & \ddots & \vdots \\ q_{n1} & \cdots & q_{nn} \end{pmatrix}$$

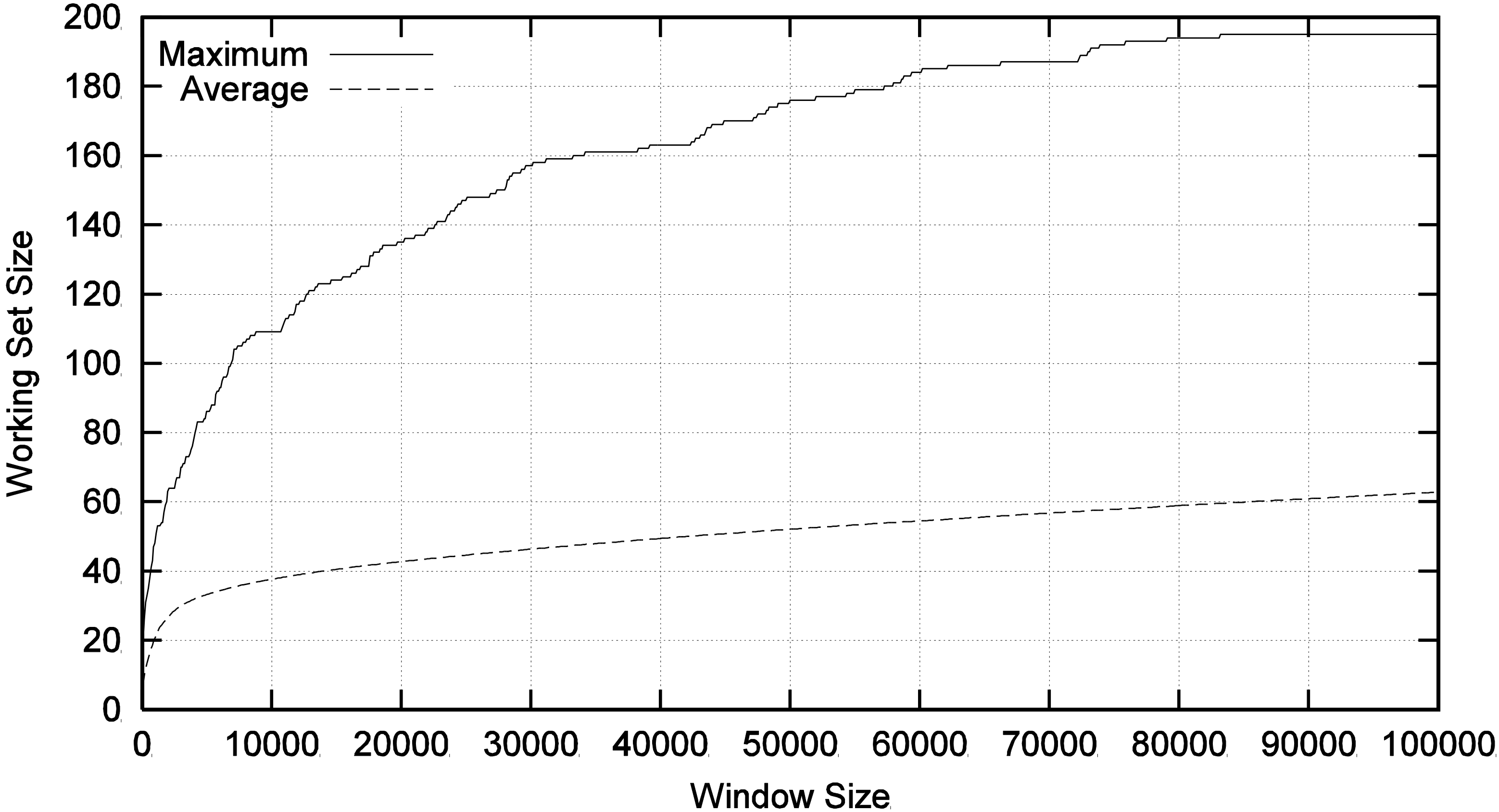
Page fault rate of A on σ = # page faults incurred by A on σ / |σ|

2.7 Refined models

2. Denning's working set model: n pages

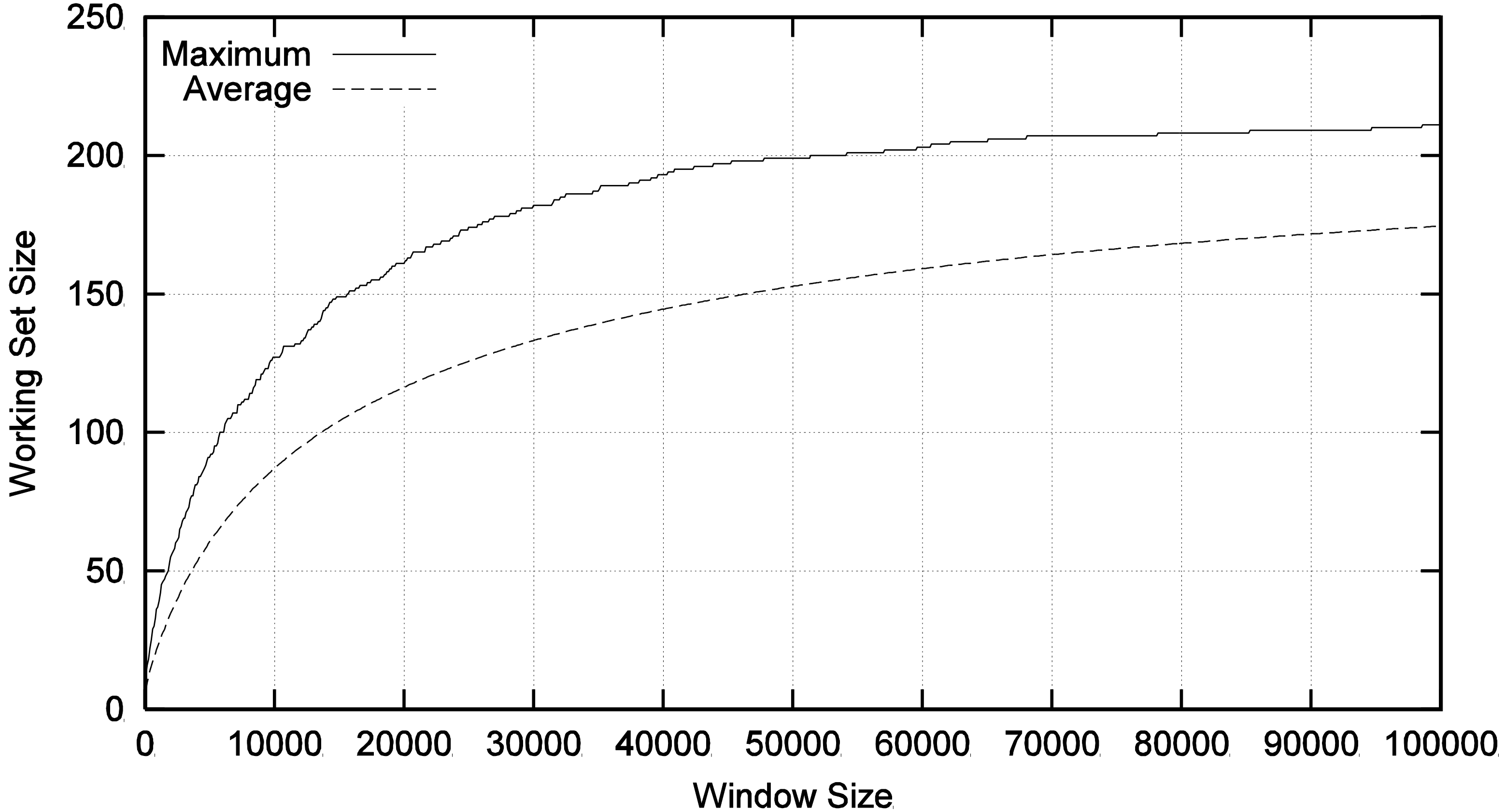
Concave function

2.7 Refined models



SPARC, GCC, 196 pages

2.7 Refined models



SPARC, COMPRESS, 229 pages

2.7 Refined models

Program executed on CPU characterized by concave function f.
It generates σ that are consistent with f.

Max-Model: σ consistent with f if, for all $n \in \mathbb{N}$, the number of distinct pages referenced in any window of length n is at most f(n).

Average-Model: σ consistent with f if, for all $n \in \mathbb{N}$, the average number of distinct pages referenced in windows of length n is at most f(n).

2.7 Refined models

- ∀ concave f: page fault rate of LRU ≤ page fault rate of any online alg. A

- ∃ concave f: page fault rate of LRU < page fault rate of FIFO

- page fault rate of LRU ≤ $\frac{k-1}{f^{-1}(k+1)-2}$

2.8 Randomized list update

Algorithm Random Move-To-Front (RMTF): With probability ½, move requested item to the front of the list.

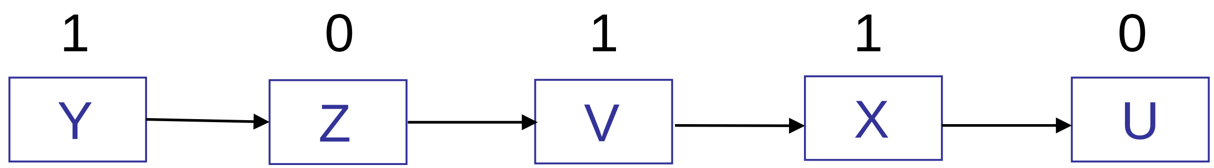
Theorem: The competitive ratio of RMTF is not smaller than 2, for a general list length n.

See [BY], page 27.

2.8 Randomized list update

Unsorted, linear linked list of items

L:



σ = X X Z Y V U X …

Algorithm BIT: Maintain bit b(x), for each item x in the list. Bits are initialized independently and uniformly at random to 0/1. Whenever an item is requested, its bit is complemented. If value changes to 1, item is moved to the front of the list.

Theorem: BIT is 1.75-competitive against oblivious adversaries.

See [BY], pages 24-26.

2.8 Randomized list update

L: W → V → U → Y → X → Z

σ = … X U Y V V W W X …

Algorithm TIMESTAMP(p): Let $0 \le p \le 1$. Serve a request to item x as follows.

With probability p move x to the front of the list.

With probability 1-p, insert x in front of the first item in the list that has been referenced at most once since the last request to x.

Theorem: TIMESTAMP(p), with $p = (3-\sqrt{5})/2$, achieves a competitive ratio of $(1+\sqrt{5})/2 \approx 1.62$ against oblivious adversaries.

2.8 Randomized list update

Algorithm Combination: With probability 4/5 serve a request sequence using BIT and with probability 1/5 serve is using TIMESTAMP(0).

Theorem: Combination is 1.6-competitive against oblivious adversaries.

Theorem: Let A be a randomized online algorithm for list update. If A is c-competitive against adaptive online adversaries, for a general list length, then $c \geq 2$.

2.9 Data compression

String S to be represented in a more compact way using fewer bits.
Symbols of S are elements of an alphabet Σ, e.g. $\Sigma = \{x_1, \ldots, x_n\}$.

Encoding: Convert string S of symbols into string I of integers.
Encoder maintains a linear list L of all the elements of Σ. It reads the symbols of S sequentially. Whenever symbol x_i has to be encoded, encoder looks up the current position of in L, outputs this position and updates the list using a given algorithm.

$S = \ldots \quad x_1 \; x_1 \; x_2 \; x_1 \; x_2 \; x_3 \; x_3 \; x_2 \quad \ldots$

L: $\boxed{x_3} \rightarrow \boxed{x_4} \rightarrow \boxed{x_2} \rightarrow \boxed{x_5} \rightarrow \boxed{x_1} \rightarrow \boxed{x_6}$

$I = \ldots \quad 5 \;\; 1 \;\; 4 \;\; 2 \quad \ldots$

Generates compression because frequenctly occuring symbols are stored near the front of the list and can be encoded using small integers/ few bits.

2.9 Data compression

Decoding: Decoder also maintains a linear list L of all the elements of Σ. It reads the integers of I sequentially. Whenever integer j has to be decoded, it looks up the symbol currently stored at position j in L, outputs this symbol and updates the list using the same algorithm as the encoder.

I = … 5 1 4 2 …

L: $x_3 \rightarrow x_4 \rightarrow x_2 \rightarrow x_5 \rightarrow x_1 \rightarrow x_6$

S = … x_1 x_1 x_2 x_1 x_2 x_3 x_3 x_2 …

2.9 Data compression

Integers of I have to encoded using a variable-length prefix code.

A prefix code satisfies the „prefix property“:

No code word is the prefix of another code word.

Possible encoding of j : $2\lfloor \log j \rfloor + 1$ bits suffice

- $\lfloor \log j \rfloor$ $0's$ followed by
- binary representation of j, which requires $\lfloor \log j \rfloor + 1$ bits

2.9 Data compression

Two schemes

- Byte-based compression: Each byte in the input string represents a symbol.
- Word-based compresion: Each „natural language“ word represents a symbol.

The following tables report on experiments done using the Calgary corpus (benchmark library for data compression).

2.9 Byte-based compression

File	TS		MTF		
	Bytes	% Orig.	Bytes	% Orig.	Size in Bytes
bib	99121	89.09	106478	95.70	111261
book1	581758	75.67	644423	83.83	768771
book2	473734	77.55	515257	84.35	610856
geo	92770	90.60	107437	104.92	102400
news	310003	82.21	333737	88.50	377109
obj1	18210	84.68	19366	90.06	21504
obj2	229284	92.90	250994	101.69	246814
paper1	42719	80.36	46143	86.80	53161
paper2	63654	77.44	69441	84.48	82199
pic	113001	22.02	119168	23.22	513216
progc	33123	83.62	35156	88.75	39611
progl	52490	73.26	55183	77.02	71646
progp	37266	75.47	40044	81.10	49379
trans	79258	84.59	82058	87.58	93695

2.9 Word-based compression

File	TS		MTF		
	Bytes	% Orig.	Bytes	% Orig.	Size in Bytes
bib	34117	30.66	35407	31.82	111261
book1	286691	37.29	296172	38.53	768771
book2	260602	42.66	267257	43.75	610856
news	116782	30.97	117876	31.26	377109
paper1	15195	28.58	15429	29.02	53161
paper2	24862	30.25	25577	31.12	82199
progc	10160	25.65	10338	26.10	39611
progl	14931	20.84	14754	20.59	71646
progp	7395	14.98	7409	15.00	49379

2.9 Burrows-Wheeler transformation

Transformation: Given S, compute all cyclic shifts and sort them lexicographically.

In the resulting matrix M, extract last column and encode it using MTF encoding. Add index I of row containing original string.

0	a a b r a c
1	a b r a c a
2	a c a a b r
3	b r a c a a
4	c a a b r a
5	r a c a a b

(c a r a a b, I=1)

2.9 Burrows-Wheeler transformation

Back-transformation: Sort characters lexicographically, gives first and last columns of M.

Fill remaining columns by repeatedly shifting last column in front of the first one and sorting lexicographically.

0	a	c
1	a	a
2	a	r
3	b	a
4	c	a
5	r	b

(c a r a a b, I=1)

2.9 Burrows-Wheeler transformation

Back-transformation using linear space:

- M‘= matrix M in which columns are cyclically rotated by one position to the right.
- Compute vector T that indicates how rows of M and M‘ correspond, i.e. row j of M‘ is row T[j] in M.   Example: T = [4 , 0 , 5 , 1 , 2 , 3]

	M	M‘
0	a a b r a c	c a a b r a
1	a b r a c a	a a b r a c
2	a c a a b r	r a c a a b
3	b r a c a a	a b r a c a
4	c a a b r a	a c a a b r
5	r a c a a b	b r a c a a

2.9 Burrows-Wheeler transformation

Back-transformation using linear space:

- L : vector, first column of M‘ = last column of M

- L[T[j]] is cyclic predecessor of L[j]

For i=0, , N-1, there holds S[N-1-i] = L[T^i [I]]

2.9 Burrows-Wheeler transformation

File	Bytes	% Orig.	bits/char	Size in Bytes
bib	28740	25.83	2.07	111261
book1	238989	31.08	2.49	768771
book2	162612	26.62	2.13	610856
geo	56974	55.63	4.45	102400
news	122175	32.39	2.59	377109
obj1	10694	49.73	3.89	21504
obj2	81337	32.95	2.64	246814
paper1	16965	31.91	2.55	53161
paper2	25832	31.24	2.51	82199
pic	53562	10.43	0.83	513216
progc	12786	32.27	2.58	39611
progl	16131	22.51	1.80	71646
progp	11043	22.36	1.79	49379
trans	18383	19.62	1.57	93695

2.9 Burrows-Wheeler transformation

Program	mean bits per character
compress	3.36
gzip	2.71
BW-Trans	2.43
comp-2	2.47

compress: version 4.32 of LZW-based tool

gzip: version 1.24 of Gaily‘s LZ77-based tool

comp-2: Nelson‘s comp-2 coder

2.9 Data compression

Assume that S is generated by a memoryless source $P = (p_1, \ldots, p_n)$

In a string generated according to P, each symbol is equal to x_i with probability p_i.

The entropy of P is defined as

$$H(P) = \sum_{i=1}^{n} p_i \log(1/pi\ )$$

It is a lower bound on the expected number of bits needed to encoded one symbol in a string generated according to P.

2.9 Huffman code

Constructs optimal prefix codes.

Code tree constructed using greedy approach.

Maintain forest of code trees.

- Initially, each symbol x_i represents a tree consisting of one node with accumulated probability p_i.
- While there exist at least two trees, choose T1, T2 having the smallest accumulated probabilies and merge them by adding a new root. New accumulated probability is the sum of those of T1, T2.

2.9 Data compression

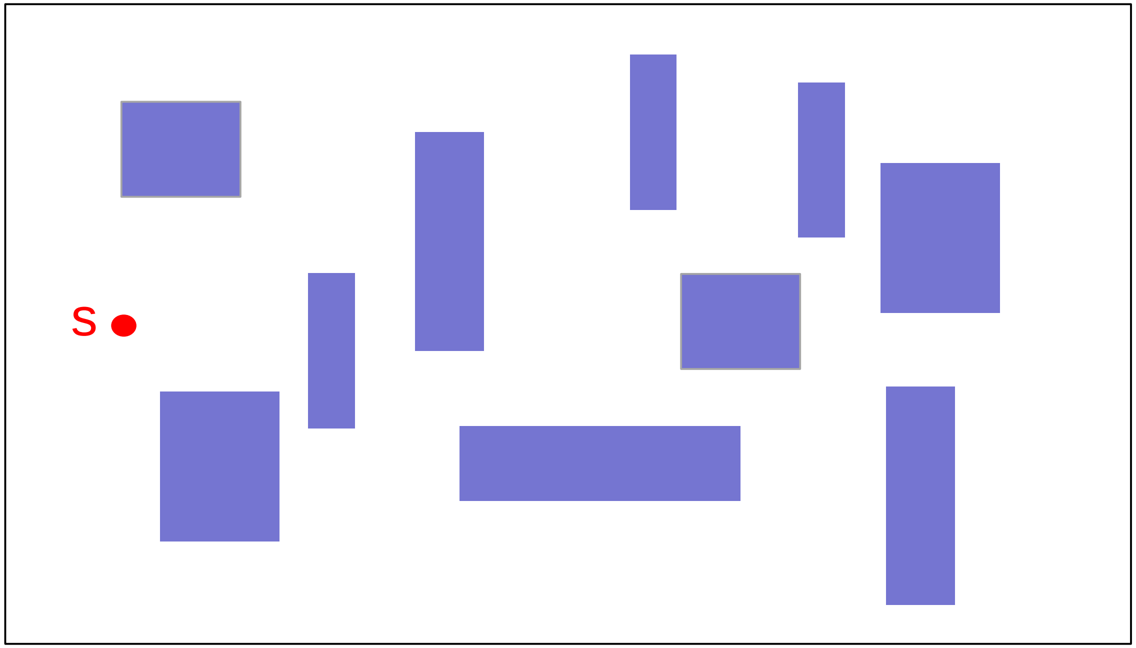
$E_{MTF}(P)$ = expected number of bits needed to encode one symbol using MTF encoding

Theorem: For each memoryless source P, there holds

$$E_{MTF}(P) \leq 1 + 2\,H(P).$$

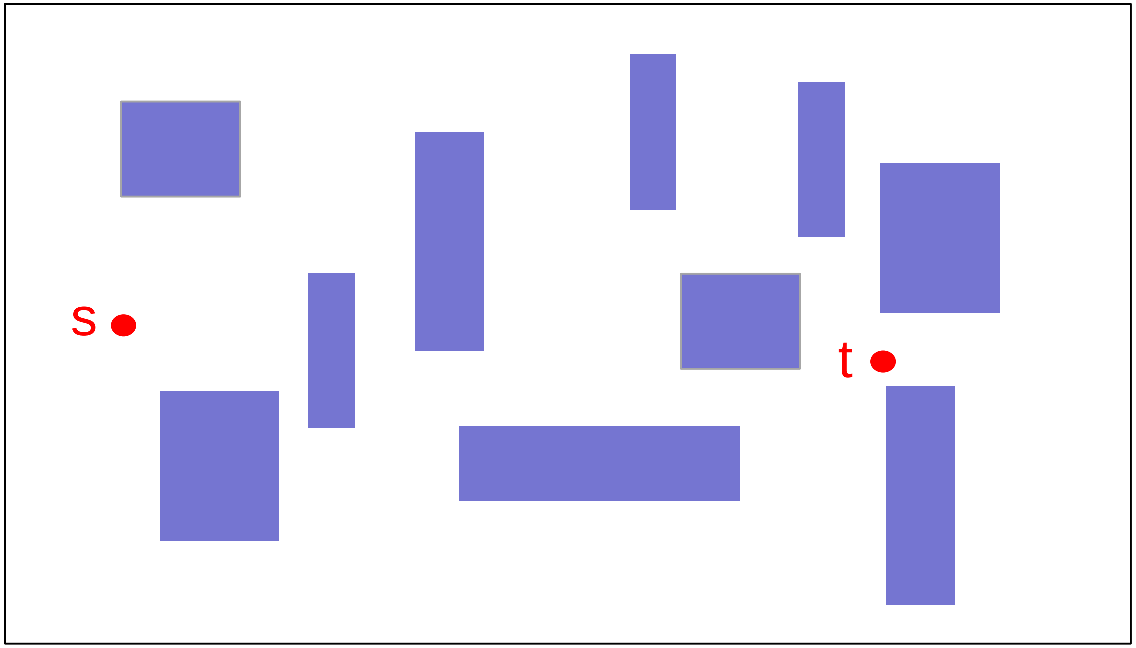
See: J.L. Bentley, D.D. Sleator, R.E. Tarjan, V.K. Wei. A locally adaptive data compression scheme. CACM 29(4), 320-330, 1986.

3 Problems: Navigation, Exploration, Localization



2.10 Robotics

Navigation: Find a short path from s to t.



Robot always knows its current position and the position of t.

Does not know in advance the position/extent of the obstacles.

Tactile robot: Can touch/sense the obstacles.

2.10 Robot navigation

The material on navigation is taken from the following two papers.

- A. Blum, P. Raghavan, B. Schieber. Navigating in unfamiliar geometric Terrain. SIAM J. Comput. 26(1):110-137, 1997.

- R.A. Baeza-Yates, J.C. Culberson, G.J.E. Rawlins. Searching in the plane. Inf. Comput. 106(2):234-252, 1993.

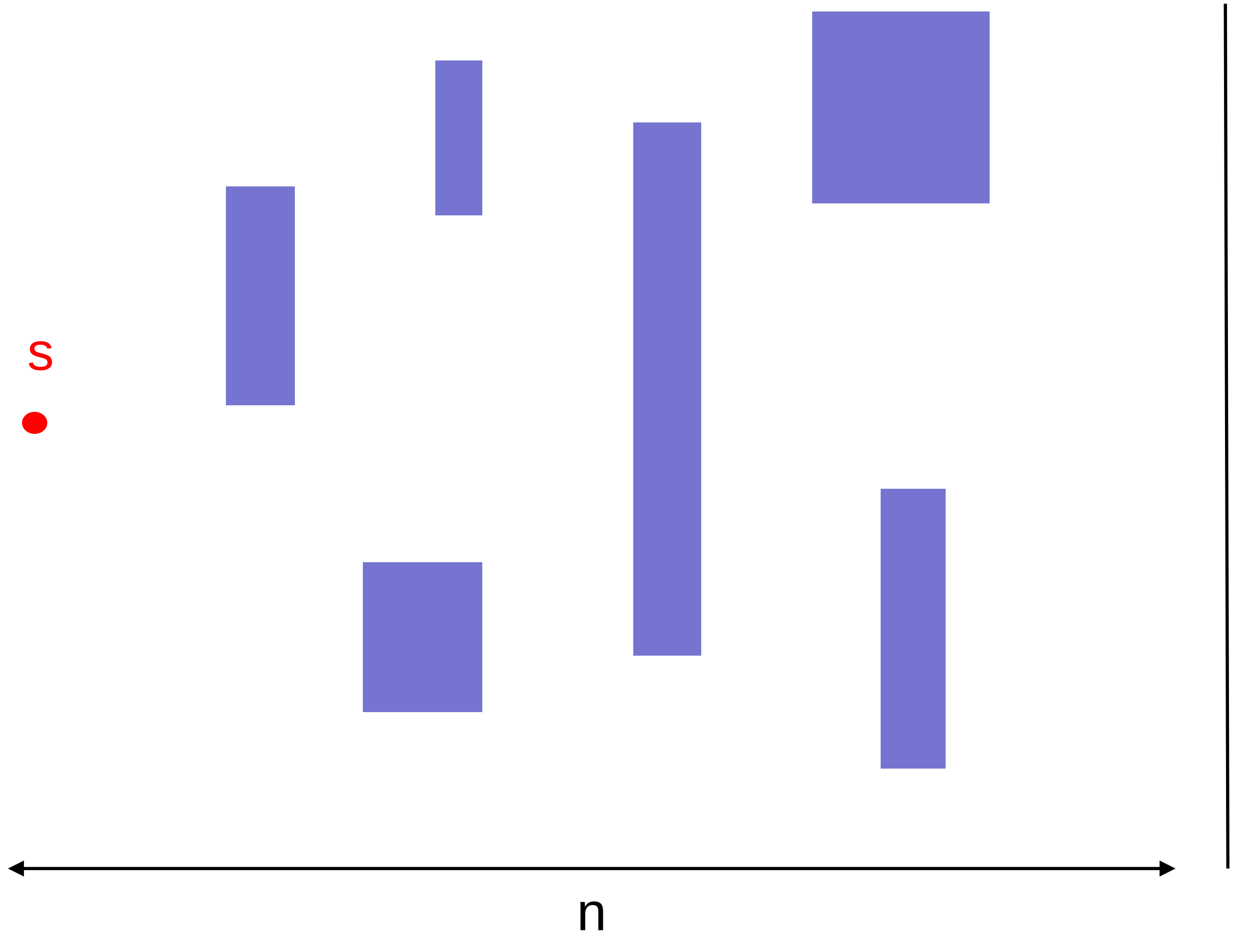
2.10 Navigation on the line

Tactile robot has to find a target t on a line. The position of t is not known in advance.

2.10 Wall problem

Reach some point on a vertical wall that is a distance of n away.

Assumption: Obstacles have width of at least 1 and are aligned with axes.



2.10 Wall problem

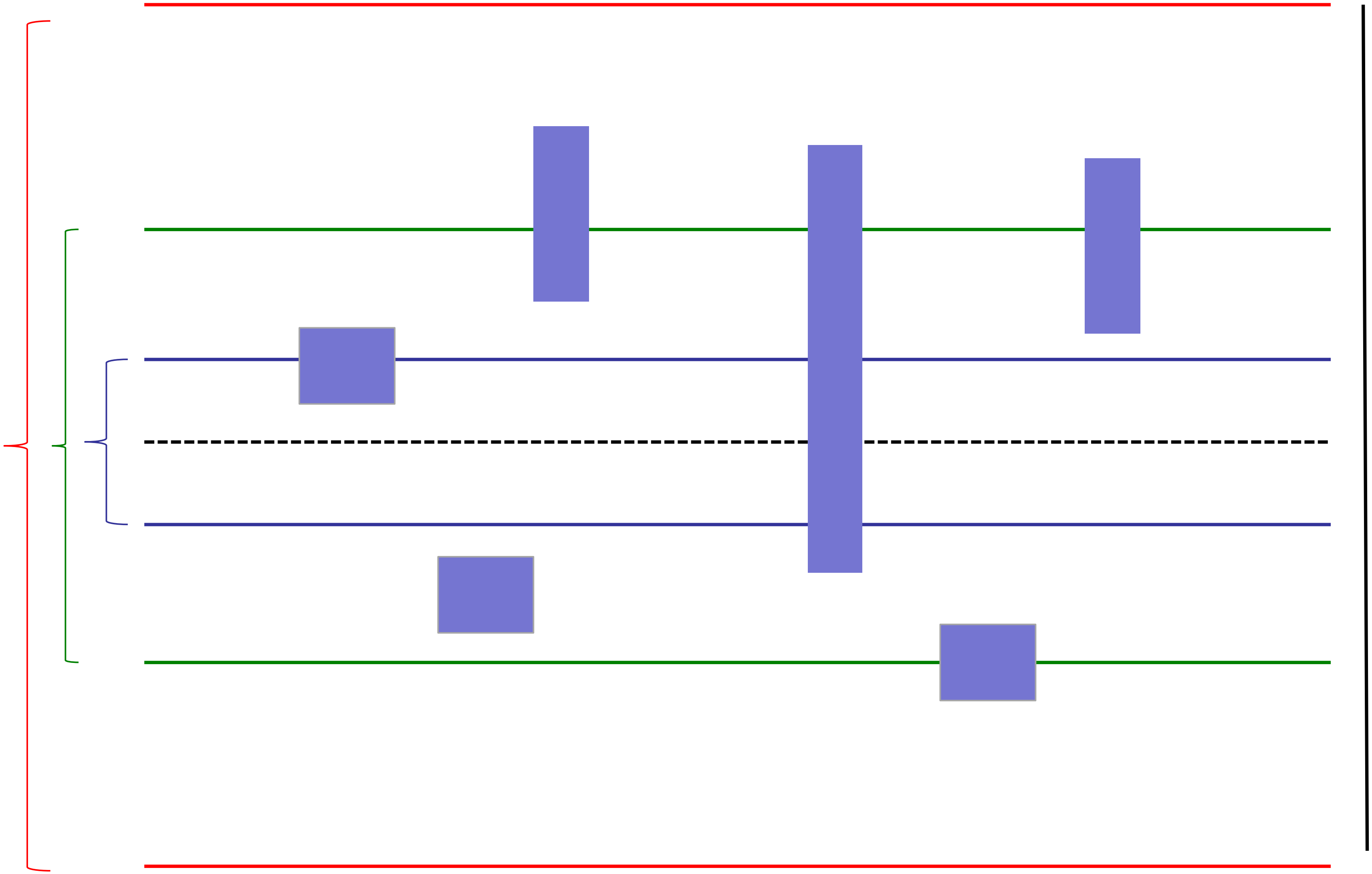
Theorem: Every deterministic online algorithm has a competitive ratio of $\Omega(\sqrt{n})$.

Upper bound: Design an algorithm with competitive ratio of $O(\sqrt{n})$.

Idea: Try to reach wall within a small window around the origin.

Double window size whenever the optimal offline algorithm OPT would also have a high cost within the window, i.e. if OPT's cost within the window of size W has cost W.

2.10 Wall problem



2.10 Wall problem

Window of size W: $W_0 = n$ (boundaries $y = +W/2$ $y = -W/2$)

$\tau := W/\sqrt{n}$

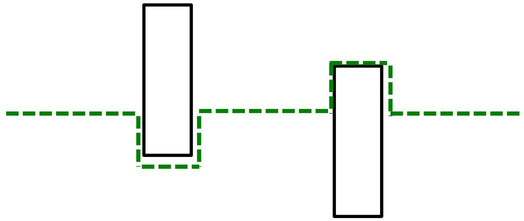
Sweep direction = north/south

Sweep counter (initially 0)

Always walk in +x direction until obstacle is reached.

Rule 1: Distance to next corner $\leq \tau$

Walk around obstacle and back to original y-coordinate.



2.10 Wall problem

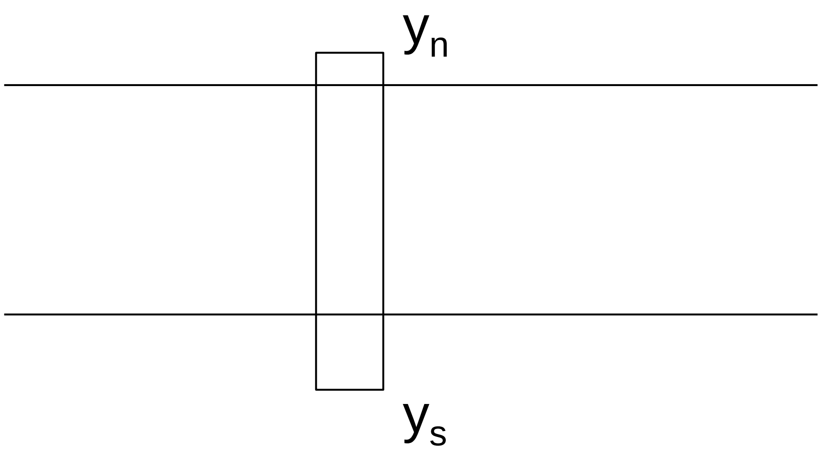
Rule 2: $y_n > W/2$ and $y_s < -W/2$ (y_n and y_s are y-coordinates of northern and southern corners of obstacle)

$W := 4 \min \{y_n , |y_s|\}$

Walk to next corner within the window.

Sweep counter := 0

Sweep direction := north if at y_s, and south y_n

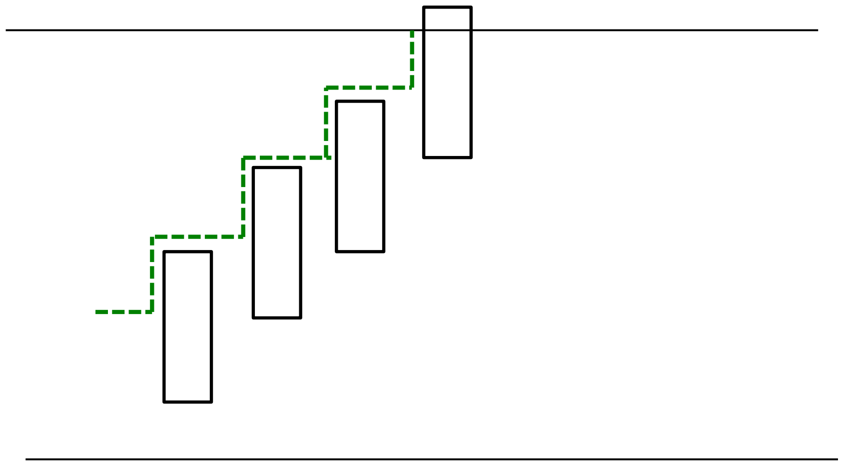


2.10 Wall problem

Rule 3: Distance to nearest corner > τ but $y_n \leq W/2$ or $y_s \geq -W/2$

Walk in sweep direction and then around obstacle.

If window boundary is reached, increase sweep counter by 1 and change sweep direction. If sweep counter reaches $\sqrt{n}$, double window size and set sweep counter to 0.



2.10 Wall problem

Analysis: W_f = last window size

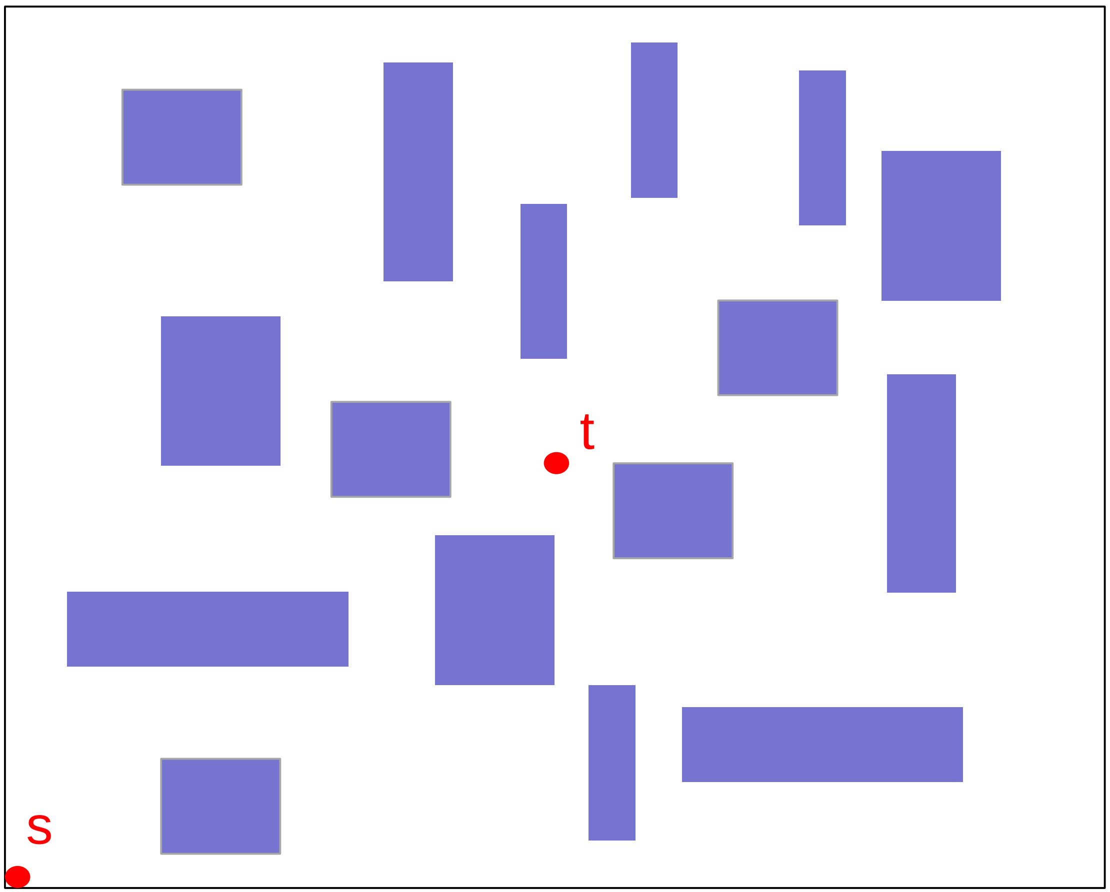
Lemma: Robot walks a total distance of $O(\sqrt{n}\,W_f)$.

Lemma: Length of shortest path is $\Omega(W_f)$.

2.11 Room problem

Square room s = lower left corner t = (n,n) center of room

Rectangular obstacles aligned with axes; unit circle can be inscribed into any of them. No obstacle touches a wall.



2.10 Paths

Greedy <+x,+y>: Walk due east, if possible, and due north otherwise.

Paths <+x,-y>, <-x,+y> and <-x,-y> are defined analogously.

Brute-force <+x>: Walk due east. When hitting an obstacle walk to nearest corner, then around obstacle. Return to original y-coordinate.

Monotone path from (x_1,y_1) to (x_2,y_2): x- and y-coordinates do not change their monotonicity along the path.

2.10 Algorithm for room problem

Invariant: Robot always knows a monotone path from (x_0,n) to (n,y_0) that touches no obstacle. Initially $x_0 = y_0 = 0$.

In each iteration x_0 or y_0 increases by at least $\sqrt{n}$.

1. Walk to $t' = (x_0+\sqrt{n}, y_0+\sqrt{n})$
   Specifically, walk along monotone path to y-coordinate $y_0+\sqrt{n}$, then brute-force <+x>. If t' is below the monotone path, then walk to point with y-coordinate $y_0+\sqrt{n}$ on the monotone path. If t' is in an obstacle, take its north-east corner.
2. Walk Greedy <+x,+y> until x- or y-coordinate is n. Assume that point $(\hat{x}, \mathrm{n})$ is reached.
3. Walk Greedy <+x,-y> until a point $(n,\hat{y})$ or old monotone path is reached. Gives new monotone path. Set $(x_0,y_0) := (\hat{x}, \hat{y})$

2.10 Algorithm for room problem

4. If $x_0 < n - \sqrt{n}$ and $y_0 < n - \sqrt{n}$, then goto Step 1.
   If $y_0 \geq n - \sqrt{n}$, walk to (x_0,n) and then brute-force <+x>.
   If $x_0 \geq n - \sqrt{n}$, walk to (n,y_0) and then brute-force <+y>.

Theorem: The above algorithm is $O(\sqrt{n})$-competitive.

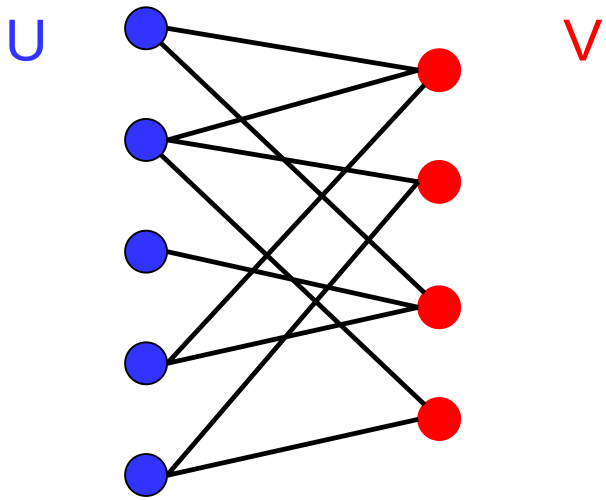
The algorithm can be generalized to rooms of dimension 2N x 2n, where $N \geq n$ and $t = (N,n)$.

In Step 1, set $t' = (x_0+\sqrt{n}\,r,\ y_0+\sqrt{n})$ where $r=N/n$. In Step 4 an x-threshold of $n - \sqrt{n}\,r$ and is considered.

2.11 Bipartite matching

Input: G = (U ∪ V, E) undirected bipartite graph.
There holds $U \cap V = \emptyset$ and $E \subseteq U \times V$.

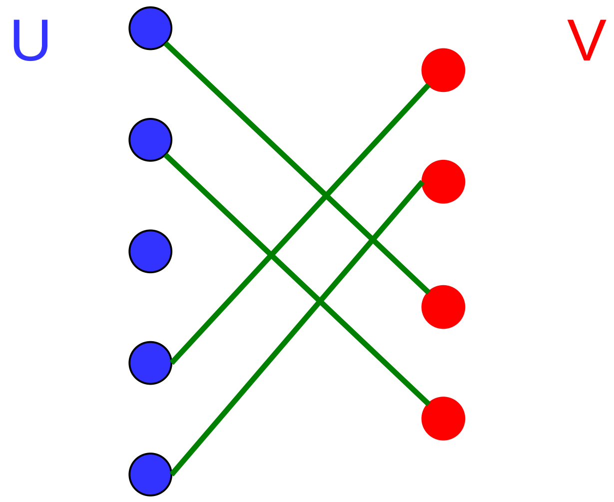
Output: Matching M of maximum cardinality
M ⊆ E is a matching if no vertex is adjacent to two edges of M.



2.11 Bipartite matching

Input: G = (U ∪ V, E)

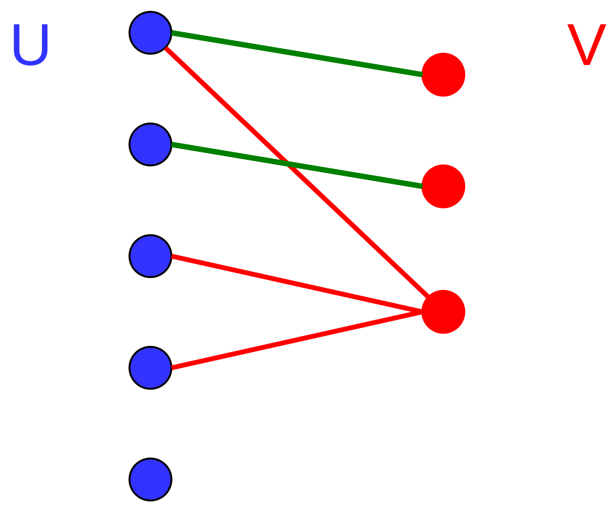
Output: Matching M of maximum cardinality



2.11 Online bipartite matching

U given initially $v \in V$ arrive one by one

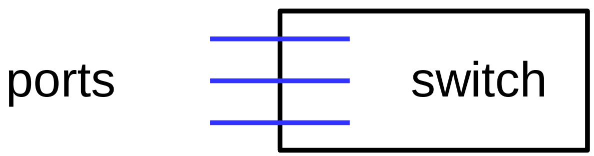
$v \in V$ arrives: neighbors in U are known;
has to be matched immediately



R.M. Karp, U.V. Vazirani, V.V. Vazirani: An optimal algorithm for on-line bipartite matching. STOC 1990: 352-358.

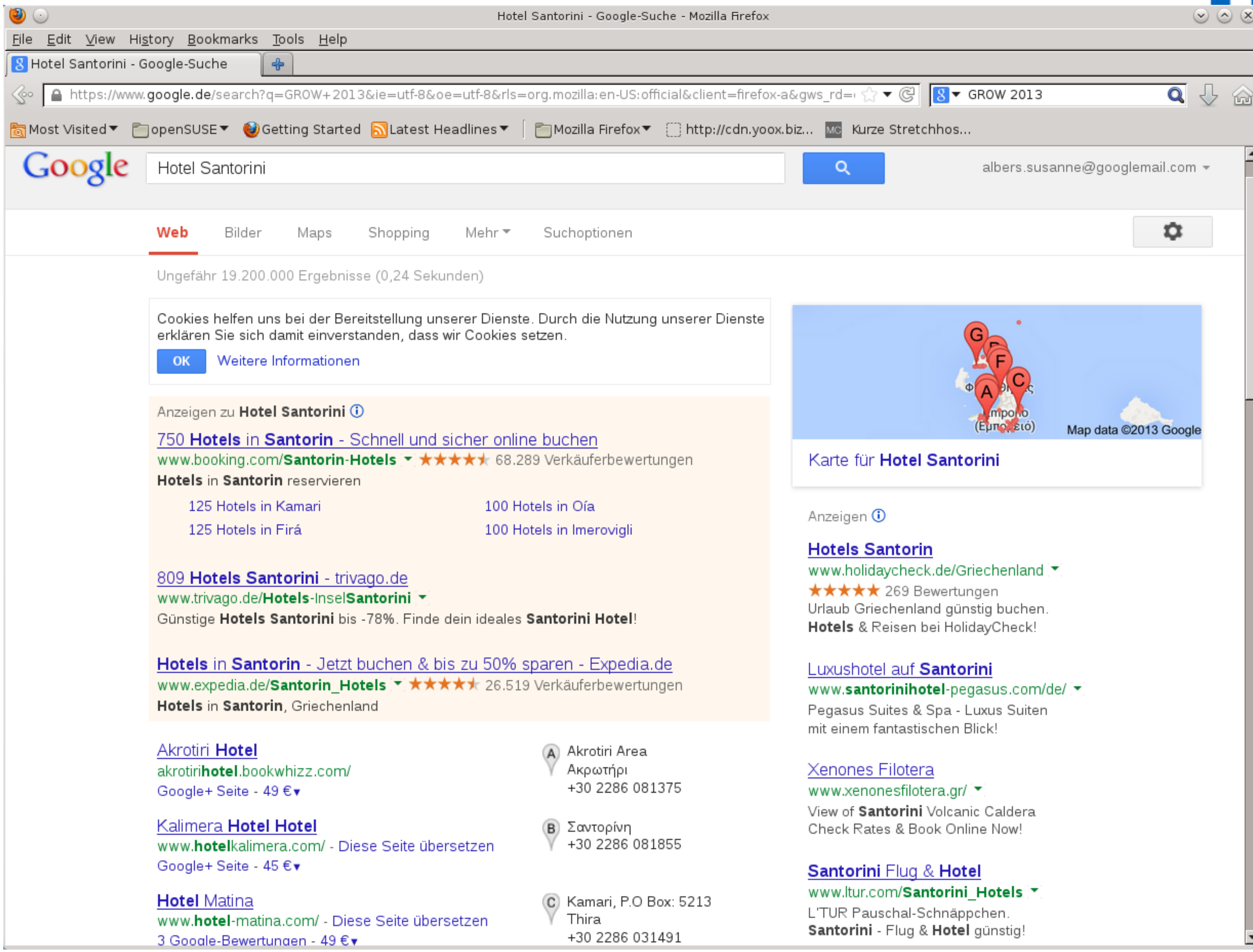
2.11 Applications

- Switch routing: U = set of ports   V = data packets



- Market clearing: U = set of sellers   V = set of buyers

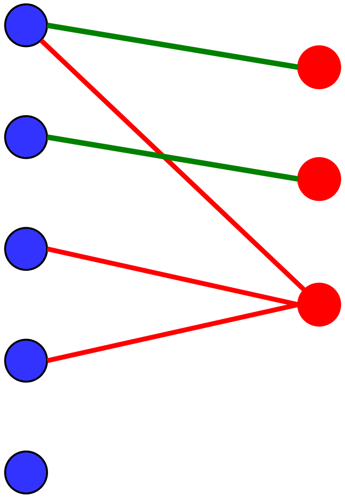
- Online advertising: U = advertiser   V= users



Hotel Santorini - Google-Suche - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Hotel Santorini - Google-Suche
https://www.google.de/search?q=GROW+2013&ie=utf-8&oe=utf-8&rls=org.mozilla:en-US:official&client=firefox-a&gws_rd=
GROW 2013
Most Visited
openSUSE
Getting Started
Latest Headlines
Mozilla Firefox
http://cdn.yoox.biz...
Kurze Stretchhos...
Google
Hotel Santorini
albers.susanne@googlemail.com
Web
Bilder
Maps
Shopping
Mehr
Suchoptionen
Ungefähr 19.200.000 Ergebnisse (0,24 Sekunden)
Cookies helfen uns bei der Bereitstellung unserer Dienste. Durch die Nutzung unserer Dienste erklären Sie sich damit einverstanden, dass wir Cookies setzen.
OK
Weitere Informationen
Anzeigen zu Hotel Santorini
750 Hotels in Santorin - Schnell und sicher online buchen
www.booking.com/Santorin-Hotels ★★★★★ 68.289 Verkäuferbewertungen
Hotels in Santorin reservieren
125 Hotels in Kamari
100 Hotels in Oía
125 Hotels in Firá
100 Hotels in Imerovigli
809 Hotels Santorini - trivago.de
www.trivago.de/Hotels-InselSantorini
Günstige Hotels Santorini bis -78%. Finde dein ideales Santorini Hotel!
Hotels in Santorin - Jetzt buchen & bis zu 50% sparen - Expedia.de
www.expedia.de/Santorin_Hotels ★★★★★ 26.519 Verkäuferbewertungen
Hotels in Santorin, Griechenland
Akrotiri Hotel
akrotirihotel.bookwhizz.com/
Google+ Seite - 49 €
A
Akrotiri Area
Ακρωτήρι
+30 2286 081375
Kalimera Hotel Hotel
www.hotelkalimera.com/ - Diese Seite übersetzen
Google+ Seite - 45 €
B
Σαντορίνη
+30 2286 081855
Hotel Matina
www.hotel-matina.com/ - Diese Seite übersetzen
3 Google-Bewertungen - 49 €
C
Kamari, P.O Box: 5213
Thira
+30 2286 031491
G
F
C
A
Map data ©2013 Google
Karte für Hotel Santorini
Anzeigen
Hotels Santorin
www.holidaycheck.de/Griechenland
★★★★★ 269 Bewertungen
Urlaub Griechenland günstig buchen.
Hotels & Reisen bei HolidayCheck!
Luxushotel auf Santorini
www.santorinihotel-pegasus.com/de/
Pegasus Suites & Spa - Luxus Suiten
mit einem fantastischen Blick!
Xenones Filotera
www.xenonesfilotera.gr/
View of Santorini Volcanic Caldera
Check Rates & Book Online Now!
Santorini Flug & Hotel
www.ltur.com/Santorini_Hotels
L'TUR Pauschal-Schnäppchen.
Santorini - Flug & Hotel günstig!

2.11 Adwords problem

Advertisers Users with queries



2.11 Adwords problem

- U = set of advertisers B_u = daily budget of advertiser u
- V = sequence of queries v
- c_{uv} = cost paid by u when ad shown to v (bid)

Goal: Maximize revenue, while respecting budgets.

Unit budgets, unit cost $\Rightarrow$ Online bipartite matching

2.11 Competitive analysis

Maximization problem

	A	OPT
Cost:	$A(\sigma)$	$OPT(\sigma)$

Online algorithm A is called c-competitive if there exists a constant a, which is independent of σ, such that

$$A(\sigma) \geq c \cdot OPT(\sigma) + a$$

holds for all σ.

2.11 Greedy algorithms

An algorithm has the greedy property if an arriving vertex $v \in V$ is matched if there is an unmatched adjacent vertex $u \in U$ available.

Theorem: Let A be a greedy algorithm. Then its competitive ratio is at least ½

Proof: $G = (U \cup V, E)$

M_{OPT} = optimum matching

$2|M_{OPT}|$ = number of matched vertices in M_{OPT}

$(u,v) \in M_{OPT}$ arbitrary

In A's matching at least one of the two vertices is matched

Number of vertices in A's matching at least $|M_{OPT}|$

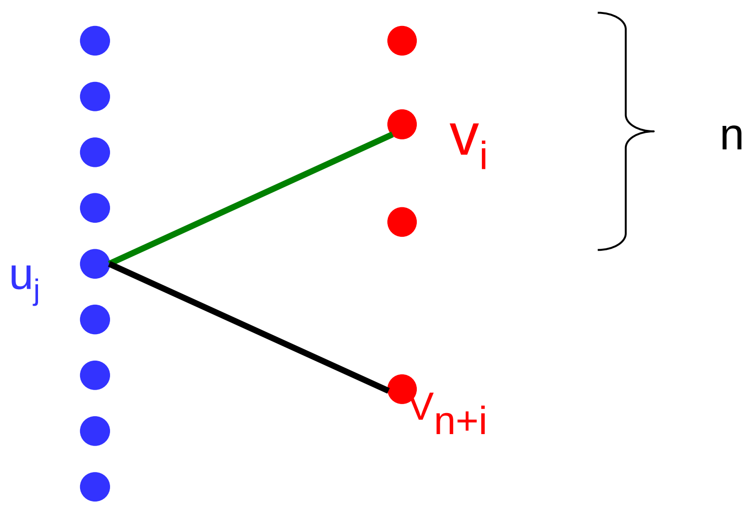
2.11 Deterministic online algorithms

Theorem: Let A be any deterministic algorithm. If A is c-competitive, then $c \leq \frac{1}{2}$

Proof: $G = (U \cup V, E)$ $\quad |U| = |V| = 2n$ even

$v_1, \ldots, v_n$ incident to all $u \in U$

v_{n+i}: If v_i matched by A to u_j, then v_{n+i} is incident to u_j only; otherwise to all $u \in U$

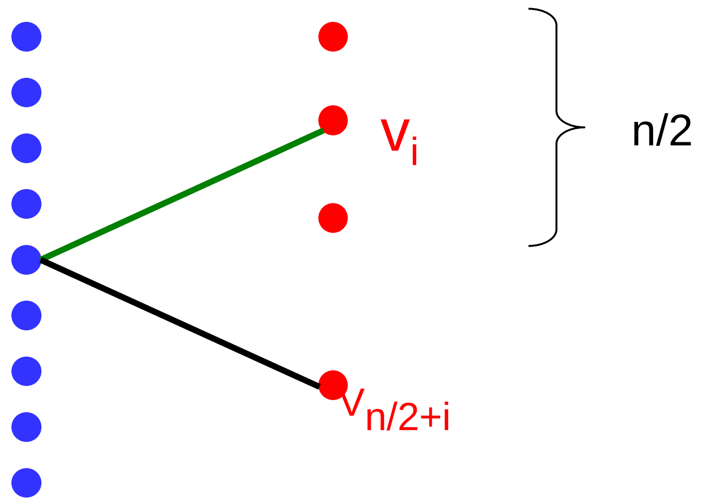


2.11 Deterministic online algorithms

Theorem: Let A be any deterministic algorithm. If A is c-competitive, then $c \leq ½$

Proof: A : $|M_A| \leq n$ Among v_i and v_{n+i} only one can be matched

OPT : $|M_{OPT}| = 2n$ $v_{n+1}, \ldots, v_{2n}$ with 1 neighbor are matched to them. All other v can be matched arbitrarily.

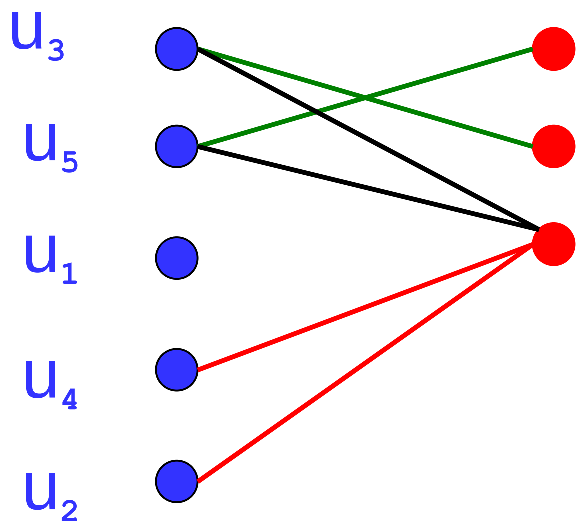


2.11 Ranking algorithm

Init: Choose permutation π of U uniformly at random.

Arrival of $v \in V$: $N(v)$ = set of unmatched neighbors.

If $N(v) \neq \emptyset$, match v with $u \in N(v)$ of smallest rank, i.e. $\pi(u)$-value



2.11 Analysis of Ranking

Theorem: Ranking achieves a competitive ratio of 1-1/e ≈ 0.632 against oblivious adversaries.

Outline of analysis:

1. It suffices to consider $G = (U \cup V, E)$ having a perfect matching (each vertex is matched).

2. Analyze Ranking on G with perfect matching.

2.11 Reduction to G with perfect matching

$G = (U \cup V, E)$ π = permutation of U $w \in U \cup V$

$H = G \setminus \{w\}$

$$\pi_H = \begin{cases} w \in U \rightarrow \text{permutation obtained from } \pi \text{ by deleting } w \\ w \in V \rightarrow \pi \end{cases}$$

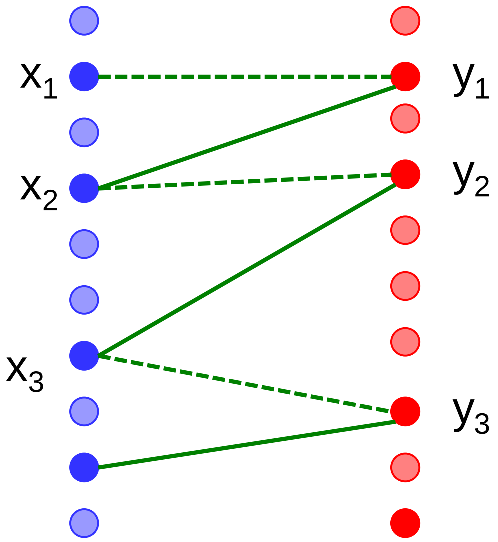
M= Ranking(G, π) M_H = Ranking(H, π_H)

Lemma: $|M| \geq |M_H|$

2.11 Lemma: $|M| \geq |M_H|$

Case 1: $w \in U$

$x = x_1$



y_i matched x_i in Ranking (G,π)

x_{i+1} matched y_i in Ranking (H, π_H)

Process stops with

x_k not matched in Ranking (G, π)

$\rightarrow |M_H| = |M|$

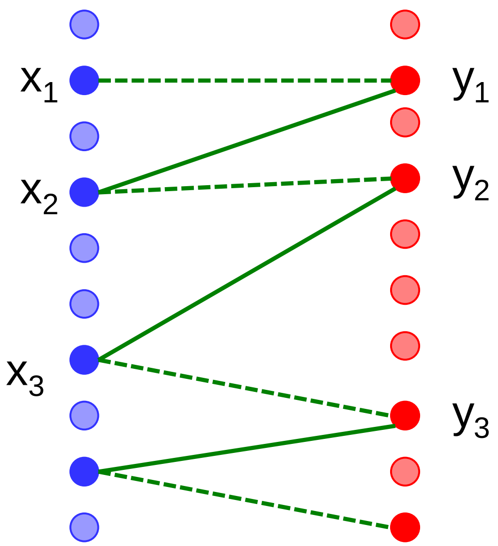
y_k not matched in Ranking (H, π_H)

$\rightarrow |M_H| = |M| - 1$

2.11 Lemma: $|M| \geq |M'|$

Case 1: $w \in U$

$w = x_1$



y_i matched x_i in Ranking (G,π)

x_{i+1} matched y_i in Ranking (H, π_H)

Process stops with

x_k not matched in Ranking (G, π)

$\rightarrow |M_H| = |M|$

y_k not matched in Ranking (H, π_H)

$\rightarrow |M_H| = |M| - 1$

2.11 Reduction to G with perfect matching

Corollary: Comp. ratio of Ranking assumed on G having a perfect matching.

Proof: $G = (U \cup V, E)$ arbitrary

M_{OPT} = optimum matching

H = obtained from G by deleting all vertices not in M_{OPT}

$\forall \pi \qquad |\text{Ranking}(G, \pi)| \geq |\text{Ranking}(H, \pi_H)|$

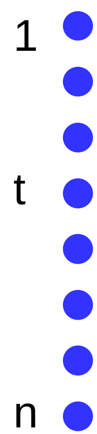
$E[|\text{Ranking}(G)|\} \geq E[|\text{Ranking}(H)|]$

M_{OPT} = optimum matching for G and H

2.11 Analysis on G with perfect matching

$|U| = |V| = n \qquad t \in \{1, \dots n\}$

p_t = probability (over all π) that vertex of rank t in U is matched



$E[|\text{Ranking}(G)|] = \sum_{1 \le t \le n} p_t$

Main Lemma: $1 - p_t \le 1/n \cdot \sum_{1 \le s \le t} p_s$

2.11 Main theorem

Thm: Ranking achieves competitive ratio of 1-1/e.

Proof: $E[|\text{Ranking}(G)|] / |OPT(G)| = 1/n \cdot \sum_{1 \le t \le n} p_t$

Determine infimum of $1/n \cdot \sum_{1 \le t \le n} p_t$

Main Lemma implies $1 + S_{t-1} \le S_t (1 + 1/n)$ $S_t = \sum_{1 \le s \le t} p_s$

$S_t = \sum_{1 \le s \le t} (1-1/(n+1))^s$ solves inequality with equality

Main Lemma: $1 - p_t \le 1/n \cdot \sum_{1 \le s \le t} p_s$

2.11 Main theorem

$$\frac{1}{n} S_n = \frac{1}{n} \sum_{1 \le s \le n} \left(1 - 1/(n+1)\right)^s$$

$$= \left(1 - (1 - 1/(n+1))^n\right)\left(1 + \frac{1}{n}\right) - \frac{1}{n} + \frac{1}{n}\left(1 - 1/(n+1)\right)^n$$

$$= 1 - \left(1 - 1/(n+1)\right)^n \xrightarrow{n \to \infty} 1 - 1/e$$

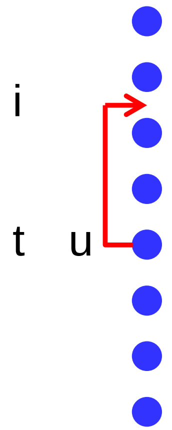
2.11 Establishing Main Lemma

$G = (U \cup V, E)$ $|U| = |V| = n$

M^* = perfect matching $u = m^*(v)$ vertex to which v is matched in M^*

Fix π and (u,v) such that u has rank t in π and $u = m^*(v)$

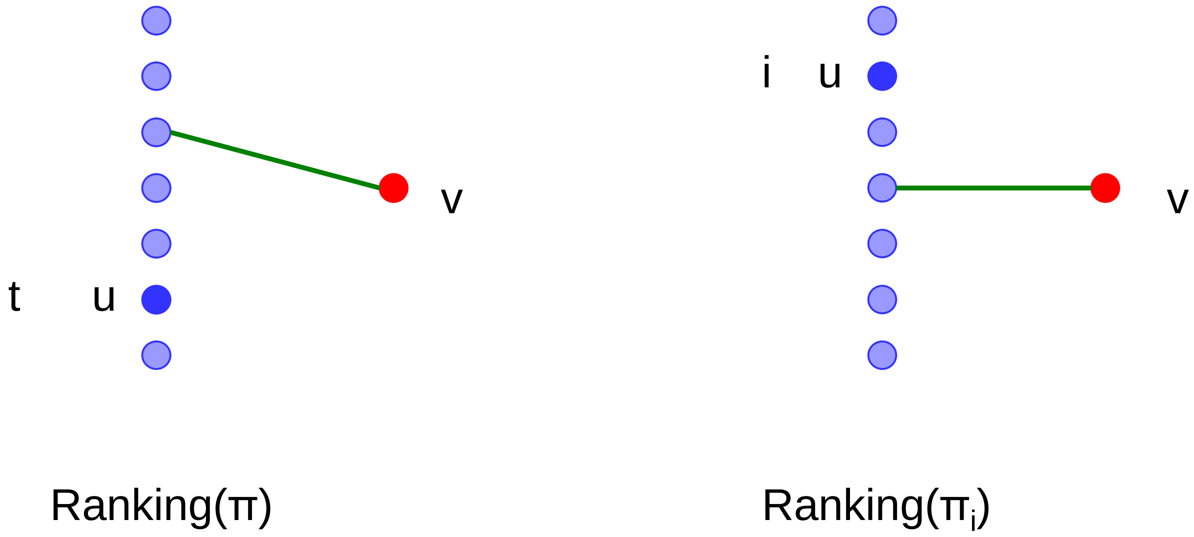
π_i = permutation in which u is reinserted so that its rank is i $1 \leq i \leq n$



2.11 Claim

Claim: If u not matched in Ranking (π), then for i = 1,…, n,

v is matched in Ranking (π_i) to u_i of rank at most t in π_i.

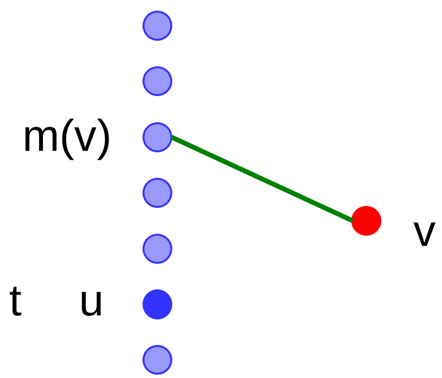


2.11 Proof Claim

X = { unmatched vertices with rank < t in π when Ranking executed with π }

X_i = { unmatched vertices with rank < t in π when Ranking executed with π_i }

Invariant: $X \subseteq X_i$ at any time before arrival of v



m(v) = partner of v in Ranking(π), rank < t in π

Invariant $\rightarrow$ when v arrives, $m(v) \in X_i$

m(v) has rank $\leq$ t in π_i

2.11 Proof of invariant

$X \subseteq X_i$ holds before arrival of a $y \in V$

x = partner of y in Ranking (π) x_i = partner of y in Ranking (π_i)

Suppose $x \neq x_i$ and x_i has rank $< t$ in π

x_i has smaller rank than x in π hence $x_i \notin X$

2.11 Establishing Main Lemma

Main Lemma: $1 - p_t \leq 1/n \cdot \sum_{1 \leq s \leq t} p_s$

Proof: For each π construct S_π

u = vertex of rank t in π v vertex such that $u = m^*(v)$

$S_\pi = \{ (v, \pi_i) \mid 1 \leq i \leq n \}$

S_π is marked if, for i = 1,…, n, v is matched in Ranking (π_i) to u_i of rank at most t in π_i.

Claim $\Rightarrow$ u not matched in Ranking(π), then S_π is marked

Claim: If u not matched in Ranking (π), then for i = 1,…, n, v is matched in Ranking (π_i) to u_i of rank at most t in π_i.

2.11 Establishing Main Lemma

Main Lemma: $1 - p_t \leq 1/n \cdot \sum_{1 \leq s \leq t} p_s$

Proof: For each π construct S_π

u = vertex of rank t in π v vertex such that $u = m^*(v)$

$S_\pi = \{ (v, \pi_i) \mid 1 \leq i \leq n \}$

S_π is marked if, for i = 1,…, n, v is matched in Ranking (π_i) to u_i of rank at most t in π_i.

Claim $\Rightarrow$ u not matched in Ranking(π), then S_π is marked

$$1 - p_t \leq \# \text{ marked sets } S_\pi / n! = \sum_{\pi \in P} |S_\pi| / (n \cdot n!)$$

$P = \{\pi \mid S_\pi \text{ is marked} \}$

2.11 Establishing Main Lemma

Proposition: Elements in S_π with $\pi \in P$ are distinct

$$1 - p_t \;\leq\; \sum_{\pi\in P} |S_\pi| / (n\cdot n!) = |\cup_{\pi\in P} S_\pi| / (n\cdot n!)$$

For any π', count occurrences of π' in $|\cup_{\pi\in P} S_\pi|$: (v_1,π') (v_2,π') (v_3,π') …

#occurrences of π' in $|\cup_{\pi\in P} S_\pi| \leq$ #v being matched to vertex of rank $\leq t$ in π'

$$= |R(\pi')|$$

$R(\pi')$ = { vertices of rank $\leq t$ in U being matched in Ranking(π') }

2.11 Establishing Main Lemma

$R(\pi)$ = { vertices of rank $\leq t$ in U being matched in Ranking(π) }

$$1 - p_t \leq |U_{\pi \in P} S_\pi| / (n \cdot n!) \leq \sum_{\pi'} |R(\pi')| / (n \cdot n!)$$

$$= 1/n \cdot \sum_{\pi} |R(\pi)| / n!$$

$$= 1/n \cdot \sum_{1 \leq s \leq t} p_s$$

2.11 Proof of claim

Claim: Elements in all sets S_π with $\pi \in P$ are distinct.

For a fixed π, elements of $S_\pi = \{ (v, \pi_i) \mid 1 \leq i \leq n \}$ are distinct

Suppose $(v, \pi_i) = (v, \pi'_j)$ where $(v, \pi_i) \in S_\pi$ $(v, \pi'_j) \in S_{\pi'}$

Let u vertex such that $u = m^*(v)$

Removing u in π_i and π'_j and reinserting it at position t, we obtain identical permutation, i.e. $\pi = \pi'$

2.12 Energy-efficient algorithms

- Power-down mechanisms: Transistion an idle system into low-power stand-by or sleep states
- Dynamic speed scaling: Modern microprocessors can run at variable speed/frequency. Required power at speed s is $P(s) = s^{\alpha}$, where $\alpha > 1$. More generally P(s) may be an arbitrary convex function.
- Networking: Optimize transmission energy in the network

2.12 Energy-efficient algorithms

Power-down mechanisms:

- System with an active state and several low-power states.
- Each state has an individual power consumption rate.
- Transitions between the various states also consume energy.
- Goal: Minimize energy consumption in an idle period.

Example: Advanced Configuration and Power Interface (ACPI)

Open standard for device configuration and power management by the operating systems. 1 active state; 4 sleep states; 1 soft-off state; 1 mechanical-off state

2.12 Energy-efficient algorithms

General system:

- $S = (s_0, \ldots, s_l)$ $l+1$ states; s_0 = active state
- $R = (r_0, \ldots, r_l)$ power consumption rates per time unit; $r_i > r_j$ for $0 \le i < j \le l$
- $D = (d_{ij})_{0 \le i,j \le l}$ d_{ij} = energy needed to transition from s_i to s_j

  Triangle inequality: $d_{ij} \le d_{ik} + d_{kj}$ for all i, j, k

2.12 Properties

Lemma: During any idle period, the following properties hold.

(a) System never powers up and then down again.

(b) If the system powers up, then it powers to s_0.

Lemma: We may assume w.l.o.g. that $d_{i0} = 0$. If $d_{i0} > 0$, for some i, then the following system of transitions energies is equivalent.

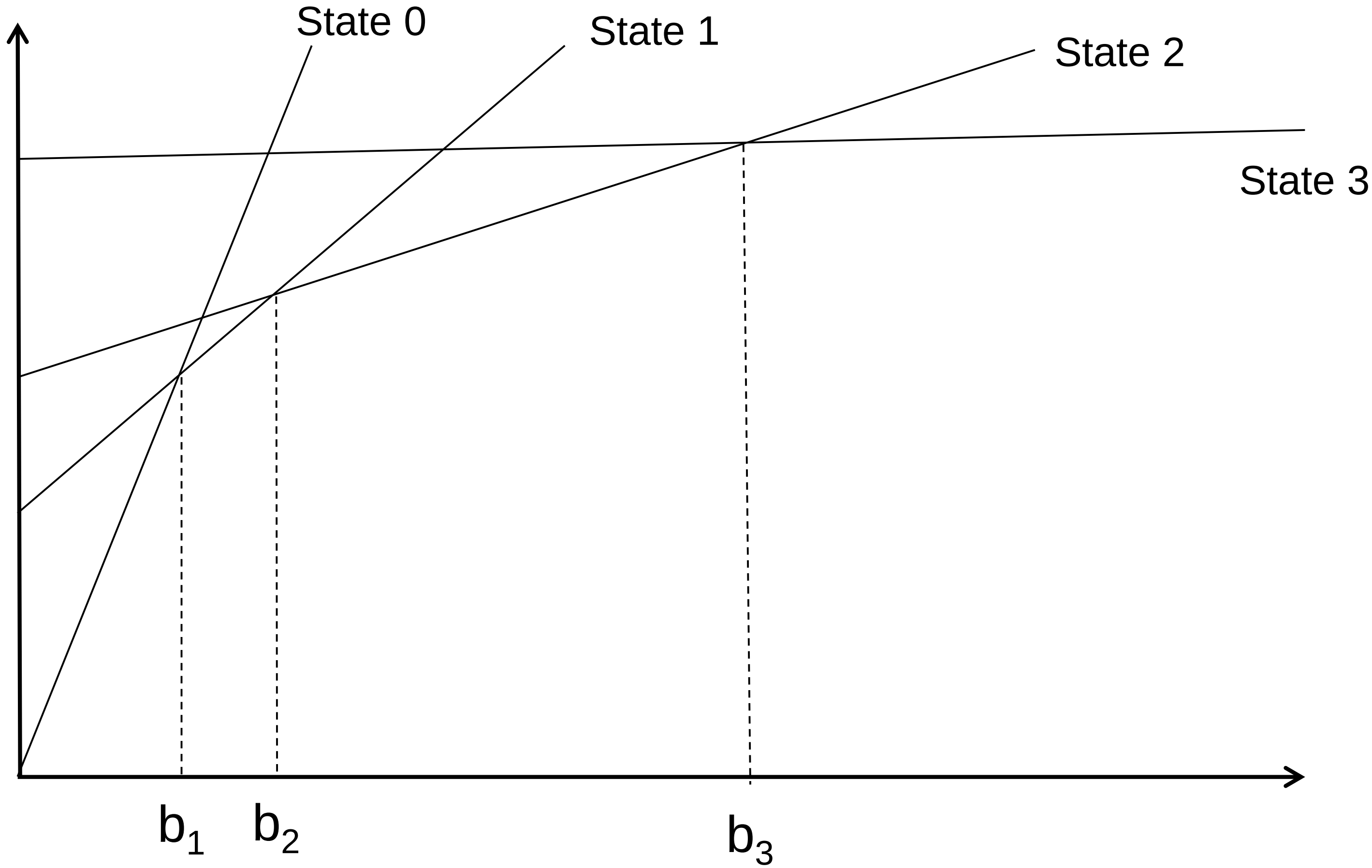
$$d'_{ij} = d_{ij} + d_{j0} - d_{i0} \qquad \text{for } i < j$$

$$d'_{ij} = 0 \qquad \text{for } i > j$$

Let $D(i) = d_{0i}$

2.12 Offline algorithm

$OPT(t) = \min_i \{D(i) + r_i t\}$ $S(t)$ = optimal state for time t



b_i = first time when s_i becomes optimal state

2.12 Properties

Algorithm LEA (Lower Envelope Algorithm): At any time t, use state S(t), i.e. the state used by the optimal offline algorithm if the idle period has total length t.

Theorem: LEA achieves a competitive ratio of $3 + 2\sqrt{2} \approx 5.82$, for general state systems.

Theorem: Given S,R and D**,** an online algorithm with a competitive ratio of $c^* + \varepsilon$ can be constructed. Here c^* is the best competitive ratio possible for the system.

Material taken from: J. Augustine, S. Irani, C. Swamy: Optimal power-down strategies. SIAM J. Comput. 37(5):1499-1516, 2008.

2.13 Financial Games

Online search: Find maximum/minimum in a sequence of prices that are revealed sequentially.

Period i: Price p_i is revealed. If p_i is accepted, then the reward is p_i; otherwise the game continues.

Application: job search, selling of a house.

One-way trading: An initial wealth of D_0, given in one currency has to be traded to some other asset or currency.

Period i: Price/exchange rate p_i is revealed. Trader must decide on the fraction of the remaining initial wealth to be exchanged.

2.13 Financial Games

Portfolio selection: s securities (assets) such as stocks, bonds, foreign currencies or commodities

Period i: price vector $\vec{p_i} = (p_{i1}, \ldots, p_{is})$

p_{ij} = # units of the j-th asset that can be bought for 1$

vector of price changes $\vec{x_i} = (x_{i1}, \ldots, x_{is})$

$x_{ij} = p_{ij} / p_{i+1,j}$

Portfolio: specifies a distribution of the wealth on the s assets just before period i

$\vec{b_i} = (b_{i1}, \ldots, b_{is})$ and $\Sigma b_{ij} = 1$

At the end of first period the wealth per initial 1$ is $\sum_{j=1}^{s} b_{1j} x_{1j}$

2.13 Relation between search and trading

Theorem: a) Let A_1 be a randomized algorithm for one-way trading. Then there exists a deterministic algorithm A_2 for one-way trading such that $A_2(\sigma) = E[A_1(\sigma)]$, for all price sequences σ.

b) Let A_2 be a deterministic algorithm for one-way trading. Then there exists a randomized search A_3 such that $E[A_3(\sigma)] = A_2(\sigma)$, for all σ.

2.13 Search problems

Will concentrate on search problems.

Prices in $[m,M]$ $0< m \leq M$ $\varphi := M/m$

Discrete time, finite time horizon, n periods; both m and M are known to player.

Online algorithm is c-competitive if there exists a constant a such that

$$c\,A(\sigma) + a \geq OPT(\sigma)$$

for all price sequences.

2.13 Algorithms

Algorithm Reservation Price Policy (RPP): Accept first price of value at least $p^* := \sqrt{Mm}$. Here p^* is called the reservation price.

Theorem: RPP is $\sqrt{\varphi}$-competitive.

Algorithm EXPO: Let $\varphi = 2^k$ for some positive integer k.

RPP_i = deterministic RPP with price $m\,2^i$.

With probability 1/k, choose RPP_i for i=1, …, k.

Theorem: EXPO is $c(\varphi)\log \varphi$-competitive, where $c(\varphi)$ tends to 1 as $\varphi \to \infty$.

Material taken from [BY], pages 265-268.

2.13 k-server problem

Metric space M; k mobile servers; request sequence σ.

Request: $x \in M$; one of the k servers must be moved to x, if the point is not already covered. Moving a server from y to x cost dist(y,x).

Goal: Minimize total distance traveled by all the servers in processing σ.

Special cases: Paging; caching fonts in printers; vehicle routing.

Results: General metric spaces:

Deterministic: $k \leq c \leq 2k-1$

Randomized: $\Omega(\log k) \leq c \leq \tilde{O}(\log^2 k \log^3 n)$, where n is size of M.

Special metric spaces:

Competitive ratio of k for lines, trees, spaces of size N=k+1 and resistive spaces.

2.13 k-server problem

Theorem: Let M be a metric space consisting of at least k+1 points and let A be a deterministic online algorithm. If A is c-competitive, then c ≥ k.

Trees: Will restrict ourselves on metric spaces that are trees.

Consider a request at point r. Server s_i is a neighbor if no other server is located between s_i and r.

Algorithm Coverage: In response to a request at r, move all neighboring servers with equal speed in the direction of r until one server reaches r.

Theorem: Coverage is k-competitive.

2.14 Metrical task systems

$(\mathcal{M}, \mathcal{R})$ $\mathcal{M}$ = (M,dist) metric space $\mathcal{R}$ = set of allowed tasks

M: set of states in which an algorithm can reside $|M| = N$

dist(i,j) = cost of moving from state i to state j

$r \in \mathcal{R}$: $r = (r(1), \ldots, r(N))$

$r(i) \in \mathbb{R}_0^+ \cup \{\infty\}$ cost of serving task in state i

Algorithm A: Initial state 0.

Sequence of requests/tasks: $\sigma = r_1, \ldots, r_n$.

Upon the arrival of r_i, A may first change state and then has to serve r_i.

A[i] : state in which r_i is served.

$$A(\sigma) = \sum_{i=1}^{n} dist(A[i-1], A[i]) + \sum_{i=1}^{n} r_i(A[i])$$

2.14 Example: paging

Pages $p_1, \ldots, p_n$ fast memory of size k

Sets $S_1, \ldots S_l$, where $l = \binom{n}{k}$ subsets of $\{p_1, \ldots, p_n\}$ having size k

For each set S_i, there is a state s_i, $i = 1, \ldots, \binom{n}{k}$

$dist(s_i,s_j) = |S_j \setminus S_i|$

Request r = p

$$r(s_i) = \begin{cases} 0 & \text{if } p \in S_i \\ \infty & \text{otherwise} \end{cases}$$

2.14 Example: list update

List consisting of n items.

n! states s_i, where $1 \leq i \leq n!$, for each possible permutation of the n items

dist(s_i,s_j) = number of paid exchanges needed to transform the two lists
(We may assume w.l.o.g. that algorithm only works with paid exchanges.)

Request r = x
$r(s_i)$ = position of item x in list s_i.

2.14 Results

Deterministic: $c = 2N - 1$

Randomized: $\Omega(\log N / \log\log N) \leq c \leq O(\log^2 N \log\log N)$

Approximation Algorithms

3.1 Basics

NP-hard optimization problems: Computation of approximate solutions

Example: Job scheduling. m identical parallel machines.

n jobs with processing times $p_1, \ldots, p_n$. Assign the jobs to machines so that the makespan is as small as possible.

List scheduling: Assign each job to a least loaded machine. (2-1/m)-approximation.

General setting: Optimization problem Π, P = set of problem instances

For $I \in P$ is $F(I)$ the set of feasible solutions

For $s \in F(I)$, $w(s)$ is the value of the solution (objective function value)

Goal: Find $s \in F(I)$ such that $w(s)$ is minimal if Π is a minimization problem (and maximal if Π is a maximization problem).

3.1 Basics

An approximation algorithm A for Π is an algorithm that, given an $I \in P$, outputs an $A(I) = s \in F(I)$ and has a running time which is polynomial in the encoding length of I.

Algorithm A achieves an approximation ratio of c if

$w(A(I)) \leq c \cdot OPT(I)$ (Π is a minimization problem)

$w(A(I)) \geq c \cdot OPT(I)$ (Π is a maximization problem)

for all $I \in P$. Here OPT(I) denotes the value of an optimal solution.

Sometimes an additive constant of b is allowed in the above inequalities. This constant b must be independent of the input. In this case c is referred to as an asymptotic approximation ratio.

Problem Max Cut: Undirected graph G=(V,E), where V is the set of vertices and E is the set of edges. Find a partition (S, V\S) of V such that the number of edges between S and V\S is maximal.

S is called a cut. Edges between S and V\S are called cut edges.

Symmetric difference: $S \Delta \{v\}$

$$S \Delta \{v\} = \begin{cases} S \cup \{v\} & \text{if } v \notin S \\ S \setminus \{v\} & \text{if } v \in S \end{cases}$$

Algorithm Local Improvement (LI):

$S := \emptyset$;

while $\exists\, v \in V$ such that $w(S \Delta \{v\}) > w(S)$ **do** $S := S \Delta \{v\}$ **endwhile;**

output S;

Theorem: LI achieves an approximation ratio of 1/2.

3.2 Traveling Salesman Problem

Euclidean Traveling Salesman Problems (ETSP): n cities $s_1, \ldots, s_n$ in $\mathbb{R}^2$.

dist(s_i,s_j) = Euclidean distance between s_i and s_j. Find a tour that visits each city exactly once and has minimum length.

Will design algorithms with approximation ratios of 2 and 1.5.

Formally, a tour is a Hamiltonian cycle. G=(V,E) V={$v_1,\ldots,v_n$}

A tour is a permutation π on {1,..., n} such that

$\{v_{\pi(i)}, v_{\pi(i+1)}\} \in E$ and $\{v_{\pi(n)}, v_{\pi(1)}\} \in E$.

Traveling Salesman Problems (TSP): Weighted graph G=(V,E) with V={$v_1,\ldots,v_n$} and a function w: E → $\mathbb{R}^+$ that assigns a length/weight to each edge. Find a tour of minimum length, i.e. a permutation π on {1,..., n} such that

$\sum_{i=1}^{n-1} w(\{v_{\pi(i)}, v_{\pi(i+1)}\}) + w(\{v_{\pi(n)}, v_{\pi(1)}\})$ is minimum.

TSP and ETSP are NP-hard

3.2 Traveling Salesman Problem

Minimum spanning tree: Weighted graph G=(V,E) with $w: E \rightarrow \mathbb{R}^+$. A minimum spanning tree T is a tree such that each $v \in V$ is vertex of T and $\sum_{e \in T} w(e)$ is minimum.

The following algorithm works with multigraph, i.e. several copies of an edge may be contained in E.

Algorithms MST:

1. Compute a minimum spanning tree for G=(V,E) with $V=\{s_1,\ldots,s_n\}$ and $w(s_i,s_j)$= Euclidian distance between s_i and s_j.
2. Construct graph H in which all edges of T are duplicated.
3. Compute an Eulerian cycle C in H (each edge is traversed exactly once).
4. Determine the order $s_{\pi(1)}, \ldots, s_{\pi(n)}$ of the first occurrences of $s_1,\ldots,s_n$ on C and output this sequence $s_{\pi(1)}, \ldots, s_{\pi(n)}$.

Theorem: Algorithm MST achieves an approximation ratio of 2.

3.2 Traveling Salesman Problem

Minimum spanning tree: Weighted graph G=(V,E) with w: E $\rightarrow \mathbb{R}^+$. A minimum spanning tree T is a tree such that each v ∈ V is vertex of T and $\sum_{e \in T} w(e)$ is minimum.

The following algorithm works with multigraph, i.e. several copies of an edge may be contained in E.

Algorithms MST:

1. Compute a minimum spanning tree for G=(V,E) with V={$s_1,\ldots,s_n$} and $w(s_i,s_j)$= Euclidian distance between s_i and s_j.
2. Construct graph H in which all edges of T are duplicated.
3. Compute an Eulerian cycle C in H (each edge is traversed exactly once).
4. Determine the order $s_{\pi(1)}, \ldots, s_{\pi(n)}$ of the first occurrences of $s_1,\ldots,s_n$ on C and output this sequence $s_{\pi(1)}, \ldots, s_{\pi(n)}$.

Theorem: Algorithm MST achieves an approximation ratio of 2.

3.2 Traveling Salesman Problem

The purpose of the edge duplication is to ensure that each vertex has even degree.

Proposition: In any tree T the number of vertices having odd degree is even.

Minimum perfect matching: Weighted graph G=(V,E) with w: $E \to \mathbb{R}^+$. A perfect matching is a subset $F \subseteq E$ such that each vertex $v \in V$ is incident to exactly one edge of F. Precondition: |V| is even. A perfect matching of minimum total weight is called a minimum perfect matching. There exist polynomial time algorithms for computing it.

3.2 Traveling Salesman Problem

Algorithm Christiofides:

1. Compute a minimum spanning tree T for $s_1,\ldots,s_n$.
2. In T determine the set V' of vertices having odd degree and compute a minimum perfect matching F for V'.
3. Add F to T and compute an Eulerian cycle C.
4. Determine the order $s_{\pi(1)}, \ldots, s_{\pi(n)}$ of the first occurrences of $s_1,\ldots,s_n$ on C and output this sequence $s_{\pi(1)}, \ldots, s_{\pi(n)}$.

Theorem: Algorithm Christofides achieves an approximation factor of 1.5

Theorem: The approximation ratio of the Christofides algorithm is not smaller than 1.5.

3.2 Traveling Salesman Problem

Problem Hamiltonian Cycle (HC): $G=(V,E)$ unweighted graph. Does G have a Hamiltonian cycle, i.e. a cycle that visits each vertex exactly once?

Theorem: Let $c>1$. If $P \neq NP$, then TSP does not have an approximation algorithm that achieves a performance factor of c.

3.3 Job scheduling

Makespan minimization: Schedule n jobs with processing times $p_1, \ldots, p_n$ to m identical parallel machines so as to minimize the makespan, i.e. the completion time of the last job that finishes in the schedule.

Algorithm Sorted List Scheduling (SLS):

1. Sort the n jobs in order of non-increasing processing times $p_1 \geq \ldots \geq p_n$.
2. Schedule the job sequence using List Scheduling (Greedy).

Theorem: SLS achieves an approximation factor of 4/3.

3.3 Approximation schemes

An approximation scheme for an optimization problem is a set $\{A(\varepsilon) \mid \varepsilon > 0\}$ of approximation algorithms for the problem such that $A(\varepsilon)$ achieves an approximation factor of $1+\varepsilon$, in case of a minimization problem, and $1-\varepsilon$ in case of a maximization problem.

PTAS = Polynomial Time Approximation Scheme

3.3 PTAS for Knapsack

Problem Knapsack: n objects with weights $w_1, \ldots, w_n \in \mathbb{N}$ and values $v_1, \ldots, v_n \in \mathbb{N}$. Knapsack with weight bound b. Find a subset $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} w_i \leq b$ such that $\sum_{i \in I} v_i$ is maximal.

Problem is NP-hard.

For j=1,…,n and any non-negative integer i let

$F_j(i)$ = minimum weight of a subset of $\{1,\ldots, j\}$ whose total value is at least i. If no such subset exists, set $F_j(i) := \propto$.

Observation: Let OPT be the value of an optimal solution.

Then $OPT = \max\{i \mid F_n(i) \leq b\}$

Lemma: a) $F_j(i) = 0$ for $i \leq 0$ and $j \in \{1,\ldots,n\}$

b) $F_0(i) = \propto$ for $i > 0$

c) $F_j(i) = \min\{F_{j-1}(i),\ w_j + F_{j-1}(i-v_j)\}$ for $i,j > 0$

3.3 PTAS for Knapsack

Algorithm Exact Knapsack

$F_j(i)$ for j=0 and $i \leq 0$ are known.

1. i:=0;
2. **repeat**
3. i:= i+1;
4. **for** j := 1 **to** n **do**
5. $F_j(i) = \min\{ F_{j-1}(i),\ w_j + F_{j-1}(i-v_j) \}$;
6. **endfor;**
7. **until** $F_n(i) > b$;
8. output i-1;

Theorem: Exact Knapsack has a running time of O(n OPT).

3.3 PTAS for Knapsack

Algorithm Scaled Knapsack(ε) $\varepsilon > 0$

1. $v_{max} := \max \{v_j \mid 1 \le j \le n\}$;
2. $k := \max \{1, \lfloor \varepsilon\, v_{max} / n \rfloor\}$
3. **for** $j := 1$ **to** n **do** $v_j(k) = \lfloor v_j / k \rfloor$ **endfor;**
4. Using algorithm Exact Knapsack, compute OPT(k) and S(k), i.e. the value and the subset of objects of an optimal solution for the Knapsack Problem with values $v_j(k)$ and unchanged weights w_k and b.
5. output $OPT^* = \sum_{j \in S(k)} v_j$.

Theorem: Scaled Knapsack(ε) achieves an approximation factor of $1-\varepsilon$.

Theorem: Scaled Knapsack(ε) has a running time of $O(n^3/\varepsilon)$.

3.3 PTAS for Makespan Minimization

m identical parallel machines, n jobs with processing times $p_1, \ldots, p_n$.

Algorithm SLS(k)

1. Sort $J_1, \ldots, J_n$ in order of non-increasing processing times such that $p_1 \geq \ldots \geq p_n$.
2. Compute an optimal schedule for the first k jobs.
3. Schedule the remaining jobs using List Scheduling (Greedy).

Theorem: For constant m and $k = \lceil (m-1)/\varepsilon \rceil$, algorithm SLS(k) is a PTAS.

3.3 PTAS for Makespan Minimization

Will construct PTAS for an arbitrary/variable number of machines.

Problem Bin Packing: n elements $a_1, \ldots, a_n \in [0,1]$. Bins of capacity 1. Pack the n elements into bins, without exceeding their capacity, so that the number of used bins is as small as possible.

Observation: There exists a schedule with makespan t if and only if $p_1, \ldots, p_n$ can be packed into m bins of capacity t.

Notation: $I = \{p_1, \ldots, p_n\}$

bins(I,t) = minimum number of bins of capacity t needed to pack I

OPT = min {t | bins(I,t) ≤ m}

$LB \leq OPT \leq 2\, LB$ $\qquad$ $LB = \max \{\frac{1}{m}\sum_{i=1}^{n} p_i, \max_{1 \leq i \leq n} p_i\}$

Execute binary search on [LB, 2LB] and solve a bin packing problem for each guess.

3.3 PTAS for Makespan Minimization

Bin packing for a constant number of element sizes.

k = number of element sizes

t = capacity of bins

Problem instance $(n_1, \ldots, n_k)$ with $\sum_{j=1}^{k} n_j = n$

Subproblem specified by $(i_1, \ldots, i_k)$ where i_j is the number of elements of element size j.

$\text{bins}(i_1, \ldots, i_k)$ = minimum number of bins to pack $(i_1, \ldots, i_k)$

3.3 PTAS for Makespan Minimization

Compute $Q = \{ (q_1,\dots,q_k) \mid bins(q_1, \dots, q_k) = 1, 0 \le q_i \le n_i \text{ for } i=1, \dots, k\}$

Q contains $O(n^k)$ elements

Compute k-dimensional table with entries $bins(i_1, \dots, i_k)$,

where $(i_1, \dots, i_k) \in \{0,\dots, n_1\} \times \dots \times \{0,\dots, n_k\}$

Initialize bins(q)=1 for all $q \in Q$ and

compute $bins(i_1, \dots, i_k) = 1 + \min_{q \in Q} bins(i_1-q_1,\dots, i_k-q_k)$

Takes $O(n^{2k})$ time.

Reduction from scheduling to bin packing: Two types of errors occur.

- Round the element sizes to a bounded number of sizes.

- Stop the binary search to ensure polynomial running time.

3.3 PTAS for Makespan Minimization

Basic algorithm: ε = error parameter $t \in [LB, 2LB]$

1. Ignore jobs of processing time smaller than εt.

2. Round down the remaining processing times.

   $p_i \in [t\varepsilon(1+\varepsilon)^i, t\varepsilon(1+\varepsilon)^{i+1})$ $i \geq 0$ is rounded to $t\varepsilon(1+\varepsilon)^i$

   $t\varepsilon(1+\varepsilon)^{i+1} < t$ implies $i+1 < \log_{1+\varepsilon} 1/\varepsilon$ and $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$ job classes suffice

3. Compute optimal solution to this problem with bin capacity t.
   Makespan for original job sizes is at most $t(1+\varepsilon)$.

4. Remaining jobs ignored so far are first assigned to the available capacity in the open bins. Then new bins of capacity $t(1+\varepsilon)$ are used.

Let $\alpha(I,t,\varepsilon)$ denote the number of used bins.

3.3 PTAS for Makespan Minimization

Lemma: $\alpha(I,t,\varepsilon) \le bins(I,t)$

Proof: Obvious if no new bins are opened to assign the small, initially ignored elements. Each time a new bin is opened, all the open ones are filled to an extent of at least t.

Corollary: $\min \{t \mid \alpha(I,t,\varepsilon) \le m\} \le OPT$.

Execute binary search on [LB,2LB] until the length of the search interval is at most εLB.

$(1/2)^i \, LB \le \varepsilon LB$ implies $i = \lceil \log_2 1/\varepsilon \rceil$

Let T be the right interval boundary when the search terminates.

3.3 PTAS for Makespan Minimization

Lemma: $T \leq (1+\varepsilon)\,OPT$

Proof: $\min\{t \mid \alpha(I,t,\varepsilon) \leq m\}$ in the interval $[T-\varepsilon LB, T]$.

Hence $T \leq \min\{t \mid \alpha(I,t,\varepsilon) \leq m\} + \varepsilon LB \leq (1+\varepsilon)\,OPT$.

Basic algorithm with $t = T$ produces a makespan of at most $(1+\varepsilon)T$

Theorem: The entire algorithm produces a solution with a makespan of at most $(1+\varepsilon)^2 T \leq (1+3\varepsilon)\,OPT$.

The running time is $O(n^{2k} \lceil \log_2 1/\varepsilon \rceil)$ where $k = \lceil \log_{1+\varepsilon} 1/\varepsilon \rceil$.

3.4 Max-SAT and randomization

Problem Max-≥kSAT: Clauses $C_1,\ldots,C_m$ over Boolean variables $x_1,\ldots,x_n$.

$C_i = l_{i,1} \vee \ldots \vee l_{i,k(i)}$ where $k(i) \geq k$ and

literals $l_{i,j} \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ for $j=1,\ldots,k(i)$

Find an assignment to the variables that maximizes the number of satisfied clauses.

Example: $C_1 = x_1 \vee \bar{x}_2 \vee x_3$ $\quad C_2 = x_1 \vee \bar{x}_3$ $\quad C_3 = x_2 \vee \bar{x}_3$

Max-≥kSAT is NP-hard

3.4 Max-SAT and randomization

Definition: A randomized approximation algorithm is an approximation algorithm that is allowed to make random choices. In polynomial time a random number in the range $\{1,\ldots,n\}$, $n \in \mathbb{N}$, is chosen, where the coding length of n is polynomial in the coding length of the input.

Algorithm A achieves an approximation factor of c if

$E[w(A(I))] \leq c \cdot OPT(I)$ (in case of a minimization problem)

$E[w(A(I))] \geq c \cdot OPT(I)$ (in case of a maximization problem)

for all $I \in P$.

3.4 Max-SAT and randomization

Algorithm RandomSAT:

for i:=1 to n do

Choose a bit $b \in \{0,1\}$ uniformly at random;

if b=0 then x_i := 0 else x_i := 1; endif;

endfor;

Output the assignment of the variables $x_1,\ldots,x_n$;

Theorem: The expected number of satisfied clauses achieved by RandomSAT is at least $(1-1/2^k)m$.

3.4 Max-SAT and randomization

Derandomization

$E[X|B]$ = expected value of X if event B holds

Algorithm DetSAT:

for i:=1 to n do

Compute $E_0 = E[\, X \mid x_j = b_j$ for $j=1,\dots, i-1$ and $x_i = \text{false}];$

Compute $E_1 = E[\, X \mid x_j = b_j$ for $j=1,\dots, i-1$ and $x_i = \text{true}];$

if $E_0 \geq E_1$ then $b_i := 0$ else $b_i := 1$; endif;

endfor;

Output $b_1,\dots,b_n$;

Theorem: DetSAT satisfies at least $E[X] = (1-1/2^k)m$ clauses.

Algorithm achieves the best possible performance. If P ≠ NP, no approximation factor greater than $1-1/2^k + \varepsilon$, for $\varepsilon > 0$, can be achieved.

3.4 Max-SAT and randomization

LP relaxations

Example: max $x+y$

s.t. $x + 2y \leq 10$

$3x - y \leq 9$

$x,y \geq 0$

Consider Max-SAT, which corresponds to Max-≥1SAT

Formula φ with clauses $C_1,\ldots,C_m$ over Boolean variables $x_1,\ldots,x_n$.

For each clause C_j define

$V_{j,+}$ = set of unnegated variables in C_j

$V_{j,-}$ = set of negated variables in C_j

3.4 Max-SAT and randomization

Formulation as integer linear program

For each x_i introduce variable y_i. For each clause C_j introduce variable z_j.

$$y_i = \begin{cases} 1 & \text{if } xi = \text{true} \\ 0 & \text{if } xi = \text{false} \end{cases} \qquad z_j = \begin{cases} 1 & \text{if } C_j \text{ satified} \\ 0 & \text{if } C_j \text{ not satisfied} \end{cases}$$

$$\begin{aligned} \max \quad & \textstyle\sum_{j=1}^{m} z_j \\ \text{s.t.} \quad & \textstyle\sum_{i:xi\in V_{j,+}} y_i + \sum_{i:xi\in V_{j,-}} (1 - yi) \geq z_j \quad && j=1,\ldots,m \\ & y_i, z_j \in \{0,1\} && i=1,\ldots,n \quad j=1,\ldots,m \end{aligned}$$

Integer linear programming (ILP) is NP-hard

Theorem: (Khachyian 1980) LP is in P.

3.4 Max-SAT and randomization

Relaxed linear program for MaxSAT

$$\max \quad \sum_{j=1}^{m} z_j$$

$$\text{s.t.} \quad \sum_{i:xi\in V_{j,+}} y_i + \sum_{i:xi\in V_{j,-}} (1 - yi) \geq z_j \quad j=1,\ldots,m$$

$$y_i,\ z_j \in [0,1] \qquad i=1,\ldots,n \quad j=1,\ldots,m$$

Algorithm RRMaxSAT (RandomizedRounding MaxSAT)

Find optimal solution $(\widehat{y}_1, \ldots, \hat{y}_n)$ $(\widehat{z}_1, \ldots, \hat{z}_m)$ to the relaxed LP for MaxSAT;
for i:=1 to n do

Choose a bit b ∈ {0,1} such that $b = \begin{cases} 1 & \text{with probability } \widehat{y}_i \\ 0 & \text{with probability } 1 - \widehat{y}_i \end{cases}$

if b=1 then x_i := 1 else x_i := 0; endif;

endfor;

Output the assignment of the variables $x_1,\ldots,x_n$;

3.4 Max-SAT and randomization

Theorem: RRMaxSAT achieves an approximation factor of $1-1/e \approx 0.632$.

Theorem: Given a formular φ, apply both RandomSAT and RRMaxSAT and select the better of the two solutions. Then the resulting algorithm achieves an approximation factor of ¾.

3.5 Probabilistic approximation algorithms

Definition: A probabilistic approximation algorithm for an optimization problem is an approximation algorithm that outputs a feasible solution with probability at least ½.

Problem Hitting Set: Ground set $V = \{v_1,\ldots,v_n\}$ and subsets $S_1,\ldots, S_m \subseteq V$. Find the smallest set $H \subseteq V$ with $H \cap S_i \neq \emptyset$ for $i=1,\ldots,m$.

H is called a hitting set.

Formulation as ILP: Variables $x_1, \ldots, x_n$

$$x_i = \begin{cases} 1 & \text{if } v_i \in H_{OPT} \\ 0 & \text{if } v_i \notin H_{OPT} \end{cases}$$

$$\min \quad \sum_{i=1}^{n} x_i$$

$$\text{s.t.} \quad \sum_{i:v_i \in S_j} x_i \geq 1 \qquad j=1,\ldots,m$$

$$x_i \in \{0,1\} \qquad i=1,\ldots,n \qquad \text{relaxed to} \quad x_i \in [0,1]$$

Algorithm RRHS (RandomizedRounding HittingSet)

Find optimal solution $(\hat{x}_1, \ldots, \hat{x}_n)$ to the relaxed LP for HittingSet;

H := ∅

for i:=1 to $\lceil \ln(2m) \rceil$ do
    for j:=1 to n do
        Choose a bit b ∈ {0,1} such that $b = \begin{cases} 1 & \text{with probability } \hat{x}_j \\ 0 & \text{with probability } 1 - \hat{x}_j \end{cases}$
        if b=1 then H := H ∪ $\{v_j\}$ endif;

endfor;

Output H;

Theorem: For each instance of HittingSet there holds:

(1) RRHS finds a feasible solution with probability at least ½.

(2) E[|RRHS(I)|] ≤ $\lceil \ln(2m) \rceil$ OPT(I).

3.5 Probabilistic approximation algorithms

Theorem: Let p be a fixed polynomial and A be a polynomial time algorithm that, for each instance I of an optimization problem, computes a feasible solution with probability $1/p(|I|)$. Then, for each $\varepsilon>0$, there exists a polynomial time algorithm A_ε, that outputs a feasible solution with probability $1-\varepsilon$.

Theorem: Let A be a randomized approximation algorithm with approximation factor c for a minimization problems. The, for any $\varepsilon>0$ and $p<1$ there exists an approximation algorithm $A_{\varepsilon,p}$ that, for each input instance I and probability at least p, computes a solution of value at most $(1+\varepsilon)\cdot c\cdot OPT(I)$.

3.6 Set Cover

Problem: Universe $U = \{u_1,\ldots,u_n\}$. Sets $S_1,\ldots,S_m \subseteq U$ with associated non-negative costs $c(S_1),\ldots,c(S_m)$. Find $J \subseteq \{1,\ldots,m\}$ such that $\bigcup_{j\in J} S_j = U$ and $\sum_{j\in J} c(S_j)$ minimal.

Greedy approach: Repeatedly choose the most cost-effective set. At any time let C be the set of covered elements. Cost-effectiveness of S is $c(S) / |S-C|$.

Algorithm Greedy:

1. $C := \emptyset$;
2. **while** $C \neq U$ **do**
3. Determine the current most cost-effective set S and $\alpha = c(S) / |S-C|$;
4. Choose S and set price(e) := α, for all $e \in S-C$;
5. $C := C \cup S$;
6. **endwhile;**
7. Output the selected sets;

Theorem: Greedy achieves an approximation factor of $H_n = \sum_{k=1}^{n} 1/k$.

Theorem: The approximation factor of Greedy is not smaller than H_n.

3.7 Shortest Superstring

Problem: Σ finite alphabet, n strings $S = \{s_1,\ldots, s_n\}$. Find shortest string s such that all s_i of S are substring of s. W.l.o.g. no s_i is substring of any s_j, where $i \neq j$.

Example: S = {ate, half, lethal, alpha, alfalfa} s = lethalalphalfalfate

3.7 Shortest Superstring

Reduction to Set Cover: Let s_i,s_j be strings such that the last k characters of s_i are equal to the first k characters of s_j.

σ_{ijk} = composition of s_i and s_j, with an overlap of k characters

M = set of all σ_{ijk}, for all feasible combinations of i,j and k

U = $\{s_1,\dots,s_n\}$

Sets: set(π) for all $\pi \in M \cup U$ where

set(π) = $\{s_i \in U \mid s_i$ is substring of $\pi\}$

cost of set(π) is equal to $|\pi|$

3.7 Shortest Superstring

Algorithm (Shortest Superstring via Set Cover):

1. Apply the Greedy algorithm for Set Cover to the above Set Cover instance. Let set(π_1), … , set(π_k) be the selected sets.

2. Concatenate $\pi_1, \dots, \pi_k$ in an arbitrary order and output the resulting string.

Lemma: $OPT \leq OPT_{SC} \leq 2\,OPT$, where OPT_{SC} is the optimum solution to the Set Cover instance.

3.8 Duality in linear programming

Optimize linear objective function subject to linear constraints.

Primal Program

min $\sum_{j=1}^{n} c_j x_j$

s.t. $\sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i=1,\ldots,m$

$x_j \geq 0 \qquad j=1,\ldots,n$

Dual Program

max $\sum_{i=1}^{m} b_i y_i$

s.t. $\sum_{i=1}^{m} a_{ij} y_i \leq c_j \qquad j=1,\ldots,n$

$y_i \geq 0 \qquad i=1,\ldots,m$

3.8 Duality in linear programming

Theorem: LP-Duality

The primal program has a finite optimum if and only if the dual program has a finite optimum.

Vectors x*= $(x_1^*, \dots, x_n^*)$ and y*= $(y_1^*, \dots, y_m^*)$ are optimal solutions if and only if

$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$.

Theorem: Complementary Slackness

Let x=$(x_1, \dots, x_n)$ and y= $(y_1, \dots, y_m)$ be feasible solutions to the primal and dual programs, respectively. The solutions x, y are optimal if and only if the following conditions hold.

Primal slackness conditions: For each j = 1,…,n there holds

$$x_j = 0 \quad \text{or} \quad \sum_{i=1}^{m} a_{ij} y_i = cj$$

Dual slackness conditions: For each i = 1,…,m there holds

$$y_i = 0 \quad \text{or} \quad \sum_{j=1}^{n} a_{ij} x_j = b_i$$

3.8 Dual fitting technique

Consider a minimization problem (analogously maximization problem)

1. (P) Primal program; val(x*) = value of an optimal solution x*.
   (D) Dual program

2. Compute solution x for (P) and vector y for (D), which may be infeasible, such that val(x) ≤ val‘(y), where val‘(y) is the objective function value of (D).

3. Divide y by α such that y‘= y / α is feasible for (D). Then val‘(y‘) ≤ val(x*).

4. Technique achieves and approximation factor of α because

$$\mathrm{val}(x) \leq \mathrm{val}'(y) \leq \mathrm{val}'(\alpha\, y') \leq \alpha\, \mathrm{val}'(y') \leq \alpha\, \mathrm{val}(x^*)$$

3.8 Set Cover and LP

Problem: Universe $U = \{u_1,\ldots,u_n\}$. Sets $S_1,\ldots,S_m \subseteq U$ with associated non-negative costs $c(S_1),\ldots,c(S_m)$. Find $J \subseteq \{1,\ldots,m\}$ such that $\bigcup_{j\in J} S_j = U$ and $\sum_{j\in J} c(S_j)$ minimal.

Formulation as LP: Set system $\Sigma = \{S_1,\ldots,S_m\}$

(P) $\min \ \sum_{S\in\Sigma} c(S)\, x_S$

s.t. $\sum_{S:\, e\in S} x_S \geq 1 \qquad e \in U$

$x_S \in \{0,1\} \qquad S \in \Sigma \qquad$ relaxed to $x_S \in [0,1]$

(D) $\max \sum_{e\in U} y_e$

s.t. . $\sum_{e\in S} y_e \leq c(S) \qquad S \in \Sigma$

$y_e \geq 0 \qquad e \in U$

Intuitively: Want to pack elements into sets s.t. cost of the sets is observed.

3.8 Set Cover and LP

Greedy approach: Repeatedly choose the most cost-effective set. At any time let C be the set of covered elements. Cost-effectiveness of S is $c(S) / |S-C|$.

Algorithm Greedy:

1. C:= Ø;
2. **while** $C \neq U$ **do**
3. Determine the current most cost-effective set S and $\alpha = c(S) / |S-C|$;
4. Choose S and set price(e) := α , for all $e \in S-C$;
5. $C := C \cup S$;
6. **endwhile;**
7. Output the selected sets;

Theorem: Greedy achieves an approximation factor of H_n.

3.8 Primal-dual algorithms

Repeatedly modify the primal and dual solutions until relaxed complementary slackness conditions hold.

(P) min $\sum_{j=1}^{n} c_j x_j$

s.t. $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ $\quad$ i=1,…,m

$x_j \geq 0$ $\quad$ j=1,…,n

(D) max $\sum_{i=1}^{m} b_i y_i$

s.t. $\sum_{i=1}^{m} a_{ij} y_i \leq c_j$ $\quad$ j=1,…,n

$y_i \geq 0$ $\quad$ i=1,…,m

Relaxed primal slackness conditions: Let α≥1. For each j = 1,…,n, there holds

$x_j = 0$ or $c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$

Relaxed dual slackness conditions: Let ß≥1. For each i = 1,…,m, there holds

$y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq ß\, b_i$

3.8 Primal-dual algorithms

Lemma: Let x,y be feasible primal and dual solutions satisfying the relaxed complementary slackness conditions. Then val(x) ≤ αß val'(y).

General scheme:

- Many algorithms work with α = 1 or ß =1.
- Algorithm starts with non-feasible primal „solution" and feasible dual solution, e.g. x=0 and y=0.
- In each iteration one improves the feasibility of the primal solution and the optimality of the dual solution until the primal solution is feasible and the relaxed complementary slackness conditions hold.
- Primal solution is always modified such that it remains integral. Modifications of the primal and dual solutions are done in a synchronized way.

3.8 Primal-dual algorithm for Set Cover

(P) min val(x) = $\Sigma_{S\in\Sigma}$ c(S) x_S

s.t. $\Sigma_{S:\, e\in S}\, x_S \geq 1$ $\quad$ $e \in U$

$x_S \in [0,1]$ $\quad$ $S \in \Sigma$

(D) max $\Sigma_{e\in U}\, y_e$

s.t. . $\Sigma_{e\in S}\, y_e \leq c(S)$ $\quad$ $S \in \Sigma$

$y_e \geq 0$ $\quad$ $e \in U$

Choose $\alpha = 1$ and $ß = f$ $\quad$ f=frequency of the element occurring most often in any set

Set is called dense if $\Sigma_{e\in S}\, y_e = c(S)$

Relaxed primal slackness conditions: For $S \in \Sigma$, x_S=0 or $\Sigma_{e\in S}\, y_e = c(S)$

Intuitively, cover contains only dense sets.

Relaxed dual slackness conditions: For $e \in U$, $y_e = 0$ or $1 \leq \Sigma_{S:\, e\in S}\, x_S \leq f$

Intuitively, each element is covered at most f times.

3.8 Primal-dual algorithm for Set Cover

Algorithm:

1. Set x=0 and y=0. No element is covered.
2. **while** there exists an uncovered element e **do**

   (a) Increase y_e until a set S is dense;

   (b) Add all dense sets S to the cover and set x_S=1;

   (c) Elements of all sets of (b) are covered;

   endwhile;
3. Output x;

Theorem: The above algorithm achieves an approximation factor of f.

Theorem: The approximation factor of the above algorithm is not smaller than f.