Approximation schemes

Def: An approximation problem has a **PTAS** (poly-time approx. scheme) if
$\forall \varepsilon > 0$ there exists a $(1+\varepsilon)$-approx alg. (for min) or $(1-\varepsilon)$-approx (for max)

? the runtime of the alg is poly in the size of the input of the problem, but it might be exponential (or worse) in $\frac{1}{\varepsilon}$ ... $\varepsilon = 0.1 \rightsquigarrow n^{10}$, $\varepsilon = 0.001 \rightsquigarrow n^{2100}$

Typical: $n^{f(\varepsilon)}$, $f(\varepsilon) \cdot n^{\text{constant}}$, $f(\varepsilon) \cdot \ln(n)$

Def: An approximation problem has a **FPTAS** (fully poly-time approx scheme) if there is an algorithm $A(\varepsilon, \text{input})$ polynomial in both the size of the input and $1/\varepsilon$, such that
$\forall \varepsilon > 0$ is $A(\varepsilon, -)$ a $(1+\varepsilon)$-approx of the problem (for min) or $(1-\varepsilon)$-approx (for max)

We can make a problem easier by writing the input in unary ($6 = 111111$)

Binary: length of n is $\log_2(n)$
Unary: length of n is n
} size of the input in unary is longer than in binary → the alg has more time

→ $n = 2^{\log_2(n)}$, so the length of n in unary is exponential in the length in binary

Def: An algorithm is **pseudopolynomial** if it is polynomial in the size of the input, provided the input is written in unary.

Def: A problem is **strongly NP-hard** if it is NP-hard even for input in unary.

→ hamiltonian cycle is strongly NP-hard, but knapsack is not

if we find a pseudopolynomial alg. for a strongly NP-hard problem, then $P = NP$
↳ the unary version is still NP-hard ⇒ we can solve everything in NP polynomially

Partition problem

Given $a_1, \dots, a_n \in \mathbb{N}$ s.t. $\sum a_i = 2B$, can we split them into two groups s.t. both sum up to B?

→ NP-hard, but not strongly NP-hard

3-Partition problem

Given $a_1, \dots, a_{3n} \in \mathbb{N}$ s.t. $\sum a_i = nB$, can we partition them into n triples (groups of size 3) s.t. each triples sums up to B?

→ strongly NP-hard, even under the restriction $\frac{B}{4} < a_1, \dots, a_{3n} < \frac{B}{2}$

Knapsack FPTAS

→ WLOG $w_i \le B$

Input: weights $w_1, \dots, w_n \in \mathbb{N}$, values $c_1, \dots, c_n \in \mathbb{N}$, capacity $B \in \mathbb{N}$

Out: $J \subseteq \{1, \dots, n\}$ s.t. $\sum_{j \in J} w_j \le B$

Objective: max $\sum_{j \in J} c_j$

→ the decision version "can a value of at least V be achieved" is NP-hard

- Pseudopolynomial dynamic programming alg:

→ we first allow only item 1, then also item 2, ..., item j, ..., item n

A_j = set of pairs (W, C) where W is the weight and C is the price of a "potentially optimal" subset of the first n items

$A_1 := \{(0,0), (w_1, c_1)\}$ ... (w_1, c_1) if we chose item 1, $(0,0)$ if we do not

$A'_j := A_{j-1} \cup \{(W + w_j, C + c_j) \mid (W, C) \in A_{j-1}\}$

→ to get A_j, we remove all dominated options, that is options $(W, C) \in A'_j$ s.t. $\exists (W', C') \in A'_j$ s.t. $W' \le W$ & $C' \ge C$ ... (W, C) is not better in any way

👁 $|A_j| \le \bar{C}_j := \sum_{i \le j} c_i$ ... for $\forall C \le \bar{C}_j$ there is at most one option (W, C) in A_j ↳ the one with the lowest W

→ at the end, we pick the most valuable option from A_n

👁 this is pseudopolynomial but not polynomial

- n iterations, at each we spend time $O(\sum_{i \le n} c_i)$ } total time $O(n \cdot \sum c_i)$

→ if the input is written in unary, this is at most quadratic in input size

→ but if it is in base 2 or higher, this might be exponential in input size

- FPTAS for knapsack

→ idea: the total value $\sum c_i$ can be too big, so we will try to make it smaller

→ c_{max} ... max of c_i, clearly $OPT \ge c_{max}$

→ map the interval $[0, \dots, c_{max}]$ to $\{0, 1, \dots, \ell\}$ ... $\hat{c}_i := \left\lfloor \frac{c_i}{c_{max}} \cdot \ell \right\rfloor$ ... $\hat{c}_{max} = \ell$

→ use the previous alg. to solve knapsack with the same weights and values $\hat{c}_i$ and take the same items as it chooses

- making $(1+\varepsilon)$-approx: We want $ALG \ge (1-\varepsilon) OPT$ ... error $\le \varepsilon \cdot OPT$

- notice that the output is the same as if we got $\hat{c}_i$ by rounding c_i down to the nearest integer multiple of $\frac{c_{max}}{\ell}$ ... since $\forall x \in \left[i \frac{c_{max}}{\ell}, (i+1) \frac{c_{max}}{\ell}\right)$ is $\left\lfloor \frac{x}{c_{max}} \cdot \ell \right\rfloor = i$

→ in this version, each c_i changes at most by $\frac{c_{max}}{\ell}$, so the total error is at most $n \cdot \frac{c_{max}}{\ell} \le \frac{n}{\ell} \cdot OPT$ ... for error $\le \varepsilon \cdot OPT$ choose $\frac{n}{\ell} \le \varepsilon \Rightarrow \ell \ge \frac{n}{\varepsilon}$ ... $\ell := \lceil \frac{n}{\varepsilon} \rceil$

Time complexity: $O(n \cdot \sum \hat{c}_i) \le O(n \cdot n\ell) = O(n^3/\varepsilon)$ ... FPTAS

SCHEDULING

- m identical machines, n jobs with processing times $p_1, p_2, \dots, p_n$

Input: $m, n \in \mathbb{N}$, $p_1, \dots, p_n \in \mathbb{Q}^+$

Output: $I_1, \dots, I_m$ – a partition of $\{1, \dots, n\}$

Objective: minimize $\max_i \sum_{j \in I_i} p_j$ ← makespan

Recall: The greedy alg LIST is a 2-approx.

↳ put next job to the least loaded machine

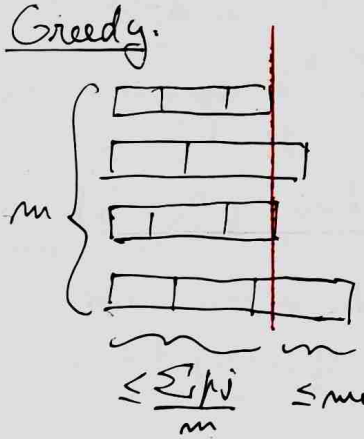
Exercise: This works even when jobs have prerequisities

↳ we first find a topological order of the DAG

Recall: If we first order the jobs based on their processing time (longest jobs first) and then run LIST, it is a 4/3-approx.

→ $\max_j p_j \le OPT$ & $\frac{\sum p_j}{m} \le OPT$ ... $LB := \max\left(\max p_j, \frac{\sum p_j}{m}\right)$

Greedy:



the difference between the last and first machine to finish is $\le LB$

$\Rightarrow ALG \le 2 \cdot LB \le 2 \cdot OPT$

Constructing a PTAS for scheduling

↳ FPTAS impossible ∵ Scheduling is strongly NP-complete

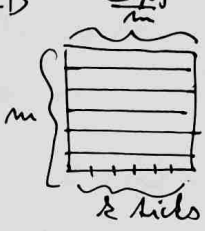
I ... input instance – we will modify it

→ fix ε, we will define a family $A_1, A_2, \dots, A_\varepsilon, \dots$ of algorithms

I' ... remove jobs of length $< \frac{1}{\varepsilon} LB$ ← forget short jobs (short if $< \frac{1}{\varepsilon} LB$, long if $\ge \frac{1}{\varepsilon} LB$)

I'' ... take I' and round down processing times to the closest integer multiple of $\frac{1}{\varepsilon^2} LB$ ... gap $= \frac{1}{\varepsilon^2} LB$

there are at most εm long jobs



Algorithm A_ε:

1. compute LB, I', I'' from I
2. $ALG(I'')$ ... solve it optimally using dynamic programming
3. $ALG(I')$ ... use the partition given by $ALG(I'')$
4. $ALG(I)$ ... add short jobs by LIST greedy alg.

$OPT(I) \geq OPT(I') \geq OPT(I'') = ALG(I'')$

Lemma: $ALG(I) \leq (1+\frac{1}{\varepsilon}) \cdot OPT(I)$ ... if polynomial, we have PTAS !

Proof: first we claim that $ALG(I') \leq (1+\frac{1}{\varepsilon}) \cdot ALG(I'')$ ← we do not lose much by rounding

- we round $p_j \rightsquigarrow p''_j$

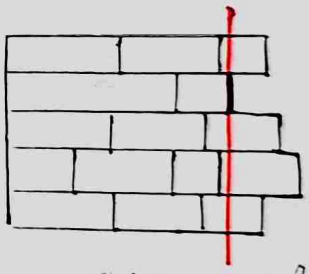
claim: $p_j \leq (1+\frac{1}{\varepsilon}) \cdot p''_j$

proof: $p_j \leq p''_j + \underbrace{\frac{1}{\varepsilon^2} LB}_{\text{gap}} \leq p''_j + \frac{1}{\varepsilon}\left(\frac{1}{\varepsilon} LB\right) \leq p''_j + \frac{1}{\varepsilon} p''_j = (1+\frac{1}{\varepsilon}) p''_j$ ← p_j is long

Since $ALG(I'') \leq OPT(I)$ we have that $ALG(I') \leq (1+\frac{1}{\varepsilon}) \cdot OPT(I)$

- case A) $ALG(I) = ALG(I')$ ... then we are done
- case B) $ALG(I) > ALG(I')$
  - ↳ then the last job to finish is short and was scheduled by greedy

$\Rightarrow ALG(I) \leq LB + \frac{1}{\varepsilon} LB = (1+\frac{1}{\varepsilon}) LB \leq (1+\frac{1}{\varepsilon}) OPT$



$\leq LB$ $\leq$ length of last job to finish

Solving $ALG(I'')$ optimally

there are at most ε^2 different sizes of jobs after rounding

↳ $\left\lfloor i \cdot \frac{1}{\varepsilon^2} LB \right\rfloor$ for $i = 1, \ldots, \varepsilon^2$

In fact we know that for $i < \varepsilon$ there are no such jobs because $\varepsilon \cdot \frac{1}{\varepsilon^2} LB = \frac{1}{\varepsilon} LB$ is the limit on short/long jobs

$\Rightarrow$ let $K \leq \varepsilon^2$ be the number of job sizes in I''

Remark: If m were not a part of the input (it was constant) then we could just try all possible schedules and take the best one

↳ # possible assignments $\leq m^{\varepsilon \cdot m}$ = constant

↳ because # long jobs $\leq \varepsilon \cdot m$

But what if m is a part of the input?

Solving ALG(I") using dynamic programming

Def: A **configuration** of a machine is a vector $(c_1, c_2, \dots, c_K)$ where

c_i = # jobs of size s_i to be scheduled on this machine

↳ recall that there are only K different sizes $s_1, \dots, s_K$

jobs in I" are long ↴

Since $c_i \in \{0, 1, \dots, k\}$, there are at most $(k+1)^K$ valid configurations

→ this is a constant ∵ k is fixed

The input I" can be written as $(n_1, n_2, \dots, n_k)$ where n_i = # jobs of size s_i

→ **want:** min T s.t. I" is solvable with make-span T

→ let T be given: we will try to solve I" with makespan $\leq T$

↳ we will find m_T = min # machines s.t. I" is solvable in time T

→ if $m_T > m$ then $T < OPT(I")$

→ because all job sizes are rounded, T and OPT are integral and we can use a binary search to find $T = OPT(I")$ and with it an optimal solution

→ if solving the decision problem for T is polynomial, then the entire process is polynomial

Let $\mathcal{C}$ be the set of valid configurations $(c_1, \dots, c_K)$ s.t. $\sum_{i=1}^{K} c_i s_i \leq T$

↳ the machine has to finish in time $\leq T$

Denote by $OPT(n_1, \dots, n_K)$ the min # machines required to schedule $(n_1, \dots, n_K)$ with makespan $\leq T$

Dynamic program:

$$OPT(n_1, \dots, n_K) = 1 + \min_{(c_1, \dots, c_K) \in \mathcal{C}} OPT(n_1 - c_1, \dots, n_K - c_K)$$

Since $n_i \leq n$, there are $\leq n^K$ input types, so we are filling out a table with $\leq n^K$ entries and each entry requires checking $\leq (k+1)^K$ previous entries

$$\Rightarrow \text{time} \leq n^K \cdot (k+1)^K \in O\left((n \cdot k)^{k^2}\right) = O\left(\left(\frac{n}{\varepsilon}\right)^{1/\varepsilon^2}\right)$$

where $\varepsilon = \frac{1}{k}$, so this is polynomial in n, but exponential in $1/\varepsilon^2$

Theorem: There $\exists$ PTAS for scheduling on identical parallel machines.

Proof: We only need to verify that the binary search is polynomial

Clearly $LB = \max\left(\max_j p_j, \frac{\sum p_j}{m}\right) \leq OPT \leq \frac{\sum p_j}{m} + \max p_j$!! $\max p_j$

$\Rightarrow$ hence the difference between the initial lower and upper bound is $\leq p_M$

$\Rightarrow$ binary search $\in \log_2(p_M)$ = length of input "p_M" when written in binary

$\Rightarrow$ bin search $\in O(\text{linear})$

BIN PACKING PROBLEM

Input: n items $a_1, a_2, \dots, a_n \in (0,1]$

Output: partition of $\{1,\dots,n\}$ into $I_1, \dots, I_m$ s.t. $\forall i: \sum_{j \in I_i} a_j \le 1$

Objective: min $m :=$ # bins

Scheduling point of view: We have a lot of machines and need to finish all jobs before a deadline

Proposition: Unless $P \neq NP$, there is no $(\frac{3}{2}-\varepsilon)$-approx alg for bin packing.

Proof: We reduce it to the NP-complete partition problem.

↳ given $a_1, \dots, a_n \in \mathbb{N}$ s.t. $\sum a_i = 2B$, is there $I \subseteq [n]$ s.t. $\sum_{j \in I} a_j = B$?

→ Make bin packing items of sizes $s_i = \frac{a_i}{B}$. Note that if some $s_i > 1$ then partition is impossible. Clearly this can be packed into 2 bins $\Leftrightarrow$ $a_1, \dots, a_n$ have a perfect split.

→ an alg. with approx ratio $\rho < \frac{3}{2}$ would always output 2 when OPT = 2

$\Rightarrow$ we would decide partition in poly time ▨

→ However, this uses only 2 bins ... what if we allow a constant difference from the optimum?

Algorithms:

- **First-Fit:** process $a_1, \dots, a_n$ in order, placing a_j into the first bin it fits in, and if it fits nowhere, create a new bin
- **First-Fit-Decreasing:** first sort the items and process them in order of non-increasing size $a_{i_1} \ge a_{i_2} \ge \dots \ge a_{i_n}$ as in First-Fit

Theorem: First-Fit is a 2-approx.

Proof: Suppose the alg. constructed $I_1, I_2, \dots, I_m$ and let $B_i := \sum_{j \in B_i} a_j$

$B_1 + B_2 > 1$ ← otherwise the alg would not create B_2

$B_i + B_{i+1} > 1 \quad \forall i$

$B_m + B_1 > 1$ ← sum the inequalities

$2 \sum B_i > m$

$\Rightarrow m < 2 \sum a_j \le 2\,OPT \Rightarrow$ 2-approx ▨

$= 2 \sum a_j$

Fact: $FFD(I) \le \frac{11}{9} OPT(I) + 4$ for any input I.

↳ first-fit-decreasing

Bin packing does not have the Rescaling Property

→ for many problems, we can convert an algorithm of the form

$$ALG \leq \rho \cdot OPT + C$$

to an approx. algorithm

$$ALG' \leq \rho \cdot OPT$$

But for bin-packing this is impossible because there is $\frac{11}{9} OPT + 4$, but there is no $(\frac{3}{2} - \varepsilon) \cdot OPT$

Example: Scheduling has the Rescaling property

↳ given processing times $p_1, p_2, \dots, p_n$
we can rescale them to $\alpha \cdot p_1, \alpha \cdot p_2, \dots, \alpha \cdot p_n$ for any $\alpha \in \mathbb{Q}^+$
and the optimal assignment will not change
→ the $+C$ is negligible if we rescale it using enormous α

- **Vertex cover:** consider an input with k disjoint copies of the original graph

Def: An asymptotic PTAS (APTAS) is a family of algorithms A_ε, $\varepsilon > 0$ and a constant c s.t. $\forall \varepsilon > 0$:

$$A_\varepsilon(I) \leq (1+\varepsilon) \cdot OPT(I) + c \quad \text{for } \forall \text{ input } I$$

for minimization problems

Theorem: Bin packing has an APTAS with $c = 1$.

Modifying the input for given $\varepsilon > 0$

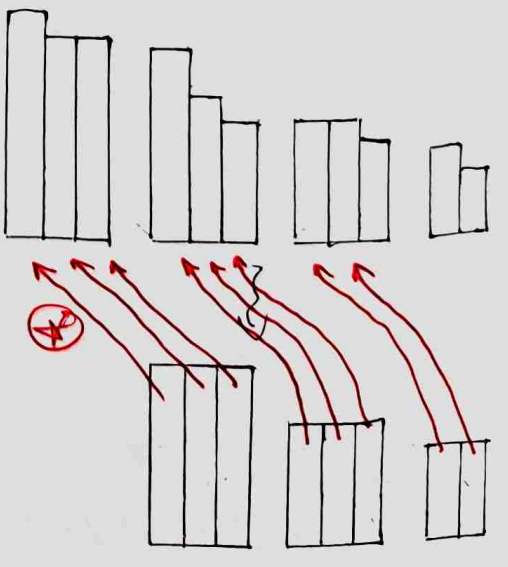
$I = a_1, a_2, \dots, a_n \in [0,1]$

I' ... remove small pieces: a_i is $\begin{cases} \text{small} & \text{if } a_i \leq \frac{\varepsilon}{2} \\ \text{large} & \text{if } a_i > \frac{\varepsilon}{2} \end{cases}$

I'' ... apply k-linear grouping to I' ... k will be set later

- first group: k largest pieces of I'
- second group: the next k largest pieces
- ⋮
- final group: $\leq k$ smallest pieces

To get I'', discard the first group, and for the other groups, round each piece to the largest piece in that group



Algorithm A_ε (ℓ to be specified later)

1. Compute I', I'', ℓ
2. $ALG(I'')$ ← solve I'' optimally using dynamic programming
3. $ALG(I')$ ← convert the packing of I'' to a packing of I' by simply shrinking $\forall a \in I''$ to its original size, and adding ℓ new bins to accomodate the first group of I
4. $ALG(I)$ ← use First-Fit to extend the packing of I' by the small pieces

$ALG(I'') \le OPT(I') \le ALG(I') = ALG(I'') + \ell$ ← $ALG(I'') = OPT(I'')$

(*) → given any packing of I', we can make a packing of I'' by moving every $a \in I''$ from group j into the spot previously occupied by some $a' \in I'$ from group $j-1$

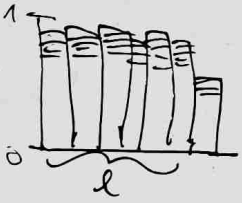
$ALG(I') \le OPT(I) + \ell$

↳ $ALG(I') \le ALG(I'') + \ell \le OPT(I') + \ell \le OPT(I) + \ell$

claim: $ALG(I) \le \max\{ALG(I'), (1+\varepsilon)OPT(I)+1\}$ for $0 < \varepsilon < 1$

Proof: If $ALG(I)$ does not use any extra bin, we are happy

⇒ Suppose that $ALG(I)$ opens a new bin to accomodate the small pieces



if $ALG(I) = \ell + 1$, then the first ℓ bins cannot fit any small piece ... they must have size $\ge 1 - \frac{\varepsilon}{2}$

⇒ $\sum a_j \ge \ell \cdot (1 - \frac{\varepsilon}{2})$ ⊠ $OPT \ge \sum_j a_j$

$$ALG(I) = \ell + 1 \le \frac{1}{1-\frac{\varepsilon}{2}} \cdot OPT(I) + 1 \le (1+\varepsilon) \cdot OPT(I) + 1$$

Note: $\frac{1}{1-\frac{\varepsilon}{2}} \le 1+\varepsilon$ for $\varepsilon \in (0,1)$ ... $2 \le (1+\varepsilon)(2-\varepsilon) \iff \varepsilon^2 - \varepsilon \le 0 \iff \varepsilon(\varepsilon-1) \le 0 \iff \varepsilon \in [0,1]$ ▨

claim: If $\ell := \lfloor \varepsilon \cdot \sum a_j \rfloor$ (all $a_j \in I$) then $ALG(I) \le (1+\varepsilon) \cdot OPT(I) + 1$ ⇒ APTAS

Proof: We need to show that $ALG(I') \le (1+\varepsilon) OPT(I) + 1$

$$ALG(I') \le OPT(I) + \ell \le OPT(I) + \varepsilon \cdot \sum a_j \le OPT(I) + \varepsilon \cdot OPT(I) \le (1+\varepsilon) OPT + 1$$ ▨

Solving $ALG(I'')$ using dynamic programming

① if $\ell = \lfloor \varepsilon \cdot \sum a_j \rfloor = 0$, then we can solve I' directly without going to I''

↳ Let $n := |I'|$ = # large pieces. Then $1 > \varepsilon \cdot \sum a_j \ge \varepsilon \cdot n \cdot \frac{\varepsilon}{2}$ (each a_j is large) ⇒ $n < \frac{2}{\varepsilon^2}$

⇒ There are only constantly many pieces ... goto ② but without linear grouping

② $k = \lfloor \varepsilon \sum a_j \rfloor \geq 1$ $\quad n = \#\text{large pieces}$

→ after linear grouping, there are at most $\frac{n}{k}$ piece sizes

$$\frac{n}{k} \leq \frac{1}{k} \cdot \underbrace{\frac{2}{\varepsilon} \sum a_j}_{\text{there are } n \text{ pieces of size} > \frac{\varepsilon}{2}} = \frac{2}{\varepsilon} \sum a_j \cdot \frac{1}{\lfloor \varepsilon \sum a_j \rfloor} \leq \frac{2}{\varepsilon} \sum a_j \cdot \frac{1}{\frac{1}{2} \varepsilon \sum a_j} = \frac{4}{\varepsilon^2}$$

↖ constantly many piece sizes

⇒ we can use the dynamic alg. for scheduling that computes the min # machines needed to finish all tasks with makespan $\leq T := 1$

↳ $T = 1$ corresponds to bin packing ▨

LINEAR PROGRAMMING RECAP

For $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ solve

(A) $\min c^T x$ s.t. $Ax \geq b$, $x \in \mathbb{R}^n$

(B) $\min c^T x$ s.t. $Ax = b$, $x \geq 0$

(C) $\min c^T x$ s.t. $Ax \geq b$, $x \geq 0$

maximization by $\max c^T x = \min -c^T x$

$Ax \geq b \iff -Ax \leq -b$

Conversions:

- equations → inequalities: $Ax = b \rightsquigarrow Ax \leq b$, $Ax \geq b$
- inequalities → equations: $Ax \geq b \rightsquigarrow Ax - I_m s = b$, $s \geq 0$ ↳ slack var.
- $x \in \mathbb{R}^n \to x \geq 0$: $x \in \mathbb{R}^n \rightsquigarrow x = x^+ - x^-$, $x^+, x^- \geq 0$

Geometry:

- solution set of a linear equation = hyperplane
- solution set of a linear inequality = halfspace

⇒ solution set of LP is intersection of these = convex polyhedron P

→ set of optimal solutions = some face of P

→ we search for it by going in the direction of the vector c

Complexity: Solvable in poly time using Ellipsoid method / Interior point method

Integer programming: $x \in \mathbb{Z}^n$, for example $x \in \{0,1\}^n$

↳ linear relaxation = corresponding LP

👁 $OPT_{INT} \geq OPT_{LP}$ because relaxation has more feasible solutions

! integer programming is NP-complete

Duality: Consider the following LP:

$$\min\ 6x_1 + 4x_2 + 2x_3 \quad \text{s.t.} \quad x_1, x_2, x_3 \geq 0$$

$$\begin{aligned} y_1 \quad & 4x_1 + 2x_2 + x_3 \geq 5 \\ y_2 \quad & x_1 + x_2 \geq 3 \\ y_3 \quad & x_2 + x_3 \geq 4 \end{aligned}$$

To get a lower bound on objective function, we can try to combine inequalities

$$6x_1 + 4x_2 + 2x_3 \geq 1\cdot(4x_1 + 2x_2 + x_3) + 2\cdot(x_1 + x_2) \geq 1\cdot 5 + 2\cdot 3 = 11$$

To maximize the lower bound, we want to solve:

$$\max\ 5y_1 + 3y_2 + 4y_3 \quad \text{s.t.} \quad y_1, y_2, y_3 \geq 0$$

$$\begin{aligned} 4y_1 + y_2 &\leq 6 \\ 2y_1 + y_2 &\leq 4 \\ y_1 + y_3 &\leq 2 \end{aligned}$$

← do not overshoot the constant in the objective function

In general: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$

Primal (P)

$$\begin{aligned} &\min c^T x \\ &Ax \geq b \\ &x \geq 0 \\ &x \in \mathbb{R}^n \end{aligned}$$

Dual (D)

$$\begin{aligned} &\max b^T y \\ &A^T y \leq c \\ &y \geq 0 \\ &y \in \mathbb{R}^m \end{aligned}$$

	PRIMAL	DUAL
variables	$x \in \mathbb{R}^n$	$y \in \mathbb{R}^m$
matrix	$A \in \mathbb{R}^{m\times n}$	$A^T \in \mathbb{R}^{n\times m}$
constraint vector	$b \in \mathbb{R}^m$	$c \in \mathbb{R}^n$
objective	$\min c^T x$	$\max b^T y$
constraints	$x_j \geq 0$	$(A^T y)_j \leq c_j$
	$x_j \leq 0$	$(A^T y)_j \geq c_j$
	$x_j \in \mathbb{R}$	$(A^T y)_j = c_j$
	$(Ax)_i \leq b_i$	$y_i \leq 0$
	$(Ax)_i \geq b_i$	$y_i \geq 0$
	$(Ax)_i = 0$	$y_i \in \mathbb{R}$

Weak duality theorem: If x, y are feasible solutions of P, D, then $c^T x \geq b^T y$

Strong duality theorem: P has optimum $\Leftrightarrow$ D has optimum and they are equal: $\min c^T x = \max b^T y$

Corollary: **Complementarity slackness conditions** ... something has to be tight

If $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ are feasible solutions of P and D, then they are optimal $\Leftrightarrow$

① $\forall j$: $x_j = 0 \ \vee\ (A^T y)_j = c_j$, and

② $\forall i$: $y_i = 0 \ \vee\ (Ax)_i = b_i$

Proof: $\Rightarrow$ From strong duality $c^T x = (A^T y)^T x = y A x = (Ax)^T y = b^T y$

Equivalently: $(b - Ax)y = 0$ & $(A^T y - c)x = 0$

Each summand of the scalar product has to sum to zero since $Ax \leq b$, $x \geq 0$, $A^T y \leq c$, $y \leq 0$

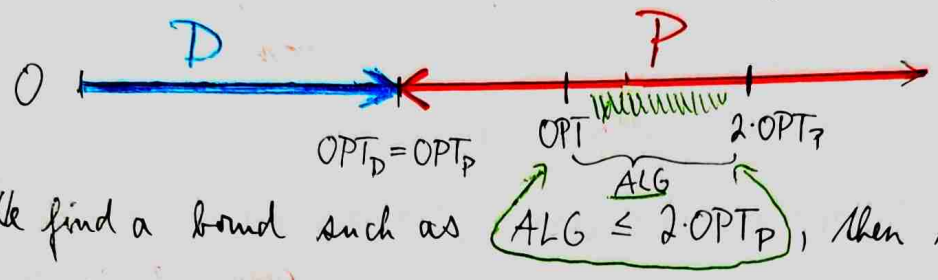
$\Leftarrow$ We just do the steps in reverse order and equality $\Rightarrow$ optimal ∎

PRIMAL - DUAL METHOD

minimization problem ... we solve it using integer programming $\rightsquigarrow OPT$

→ we relax it $\rightsquigarrow$ Primal, Dual. $\rightsquigarrow$ $OPT_P = OPT_D$

→ solve the relaxation and round it to get an integral solution $\rightsquigarrow ALG$



We find a bound such as $ALG \leq 2 \cdot OPT_P$, then $ALG \leq 2 \cdot OPT$

Idea: Instead of solving the relaxation, try to satisfy the complementarity conditions to get very close to the optimum.

Primal-Dual Algorithm:

1. $y \leftarrow$ trivial feasible solution of the Dual
2. $S \leftarrow$ empty solution of the problem
3. While S is not feasible:
4. Increase some variables of the dual until some constraint becomes tight
5. Modify S based on the tight constraint.

MINIMAL SET COVER

Input: $A_1, A_2, \dots, A_m \subseteq [n]$ s.t. $\bigcup A_i = [n]$

$c_1, c_2, \dots, c_m \geq 0$ costs

Output: $I \subseteq [m]$ s.t. $\bigcup_{i \in I} A_i = [n]$

Objective: $\min \sum_{i \in I} c_i$

Integer program:

$\min \sum_{i=1}^{m} c_i x_i$, variables $x_i \in \{0,1\}$ $\rightsquigarrow$ $x_i \geq 0$ relaxation $i \in [m]$

s.t. $\forall t \in [n]: \sum_{i : t \in A_i} x_i \geq 1$ ... we buy every $t \in [n]$

Primal LP:

$\min \sum_{i=1}^{m} c_i x_i$

buying sets to get $\forall t \in [n]$

s.t. $\forall t \in [n]: \sum_{i : t \in A_i} x_i \geq 1$

$x_i \geq 0$, $i \in [m]$

Dual LP:

$\max \sum_{t=1}^{n} y_t$

selling $t \in [n]$ to get max profit but there are constraints

s.t. $\forall i \in [m]: \sum_{t \in A_i} y_t \leq c_i$

$y_t \geq 0$, $t \in [n]$

Algorithm:

1. $\forall e \in [n] : y_e \leftarrow 0$ ← start by selling nothing
2. $I \leftarrow \emptyset$, $E \leftarrow \emptyset$ ... I = purchased sets, E = covered elements
3. While $E \neq [n]$, choose $e \notin E$:
4. $\quad \delta \leftarrow \min_{i : e \in A_i} \left(c_i - \sum_{e' \in A_i} y_{e'} \right)$ ← see for how much we can sell y_e so that we do not break any constraints
5. $\quad y_e \leftarrow y_e + \delta$
6. $\quad$ Choose $i \in [m]$ s.t. $e \in A_i$ & $\sum_{e' \in A_i} y_{e'} = c_i$: ← cover e by buying some A_i with tight constraint
4. $\quad I \leftarrow I \cup \{i\}$, $E \leftarrow E \cup A_i$

This produces a feasible solution of Minimal Set Cover ← it is an integral solution of the Primal

Theorem: This is an f-approx where $f := \max_{e \in [n]} (\#i : e \in A_i)$

Proof: Denote by $\bar{y}_e$ the value of y_e at the end of the algorithm

$\forall i \in I : \sum_{e \in A_i} \bar{y}_e = c_i$ ... we satisfy a complementarity condition

we could just write $y_e \leftarrow \delta$ since each y_e is increased at most once

$$ALG = \sum_{i \in I} c_i = \sum_{i \in I} \sum_{e \in A_i} \bar{y}_e = \sum_{e \in [n]} \bar{y}_e \cdot |\{i \in I \mid e \in A_i\}| \leq f \sum_{e \in [n]} \bar{y}_e \leq f \cdot OPT_D \leq f \cdot OPT$$

■

Standard primal-dual analysis:

- → each object in the solution was given by a tight dual inequality
- ⇒ we can rewrite cost of primal solution in terms of dual variables
- → we then compare this with the dual objective function and bound it by some multiple of dual optimum

Hitting Set Problem

Input: $A_1, A_2, \dots, A_m \subseteq [n]$

$c_1, c_2, \dots, c_n \geq 0$ costs

Output: $S \subseteq [n]$ s.t. $\forall i \in [m]: A_i \cap S \neq \emptyset$

Goal: $\min \sum_{t \in S} c_t$

Integer program / relaxation:

$$\min \sum_{t \in [n]} c_t x_t \;, \quad x_t \in \{0,1\} \quad \rightsquigarrow x_t \geq 0 \text{ relaxation}$$

$$\text{s.t. } \forall i \in [m]: \sum_{t \in A_i} x_t \geq 1$$

Want to buy some $t \in [n]$ to hit $\forall$ set & minimize cost

Dual:

$$\max \sum_{i \in [m]} y_i \;, \quad y_i \geq 0 \;, \quad i \in [m]$$

$$\text{s.t. } \forall t \in [n]: \sum_{i: t \in A_i} y_i \leq c_t$$

Want to sell some sets for max profit, but there is a constraint how much we can make off a single $t \in [n]$

Algorithm:

1. $\forall i \in [m]: y_i \leftarrow 0$ ← start by selling nothing
2. $S \leftarrow \emptyset$, $I \leftarrow \emptyset$ … S = selected elements, I = hit sets
3. While $I \neq [m]$ choose $i \notin I$:
4. $\quad \delta \leftarrow \min_{t: t \in A_i} \left(c_t - \sum_{j: t \in A_j} y_j \right)$ ← try to sell A_i for as much as possible
5. $\quad y_i \leftarrow y_i + \delta$
6. $\quad$ Choose $t \in A_i$ s.t. $\sum_{i: t \in A_i} y_i = c_t$
7. $\quad S \leftarrow S \cup \{t\}$, $I \leftarrow I \cup \{j \mid t \in A_j\}$

Theorem: This is a k-approx for $k = \max_{i \in [m]} |A_i|$.

Proof: At the end of the algorithm:

$$\forall t \in S: \sum_{i: t \in A_i} y_i = c_t$$

$$ALG = \sum_{t \in S} c_t = \sum_{t \in S} \sum_{i: t \in A_i} y_i = \sum_{i \in [m]} y_i \left| \{ t \in S \mid t \in A_i \} \right| = \sum_{i \in [m]} y_i |S \cap A_i|$$

$$\leq k \cdot \sum_{i \in [m]} y_i \leq k \cdot OPT_D \leq OPT$$

Feedback Vertex Set

Input: $G=(V,E)$, weights w_t for $t \in V = [n]$

Output: $S \subseteq V$ s.t. $\forall$ cycle has a vertex in S ← S is a feedback vertex set

Goal: $\min \sum_{t \in S} w_t$

👁 $G[V-S]$ is acyclic

👁 This is a special case of the Hitting Set Problem
⇒ we could get a (max cycle-length)-approx

$\mathcal{C}$:= set of all cycles $C \subseteq V$

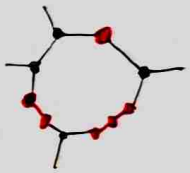
❓ Can we do better if we use the structure of a graph?

Def: A topological path in G is a path whose every vertex has degree = 2

👁 If the hitting set algorithm adds into S a vertex $t \in V$ that lies on a topological path P, then no other $t' \in P$ will be added later

⇒ we want to bound the term $|S \cap C|$ where C is a cycle (see ⊛)

👁 If $C \subseteq V$ is a cycle containing k vertices of deg ≥ 3, then $|S \cap C| \leq 2k$ at the end of the algorithm



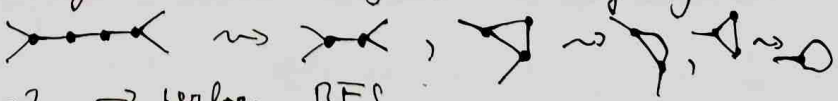
Algorithm:

1. $y \leftarrow 0$, $S \leftarrow \emptyset$, perform ⊛⊛
2. While $\exists$ cycle in G:
3. Find cycle C with $\leq 2\lceil \log_2 n \rceil$ vertices of deg ≥ 3
4. Increase y_C until there $\exists t \in C$ s.t. $w_t = \sum_{C' : t \in C'} y_{C'}$
5. $S \leftarrow S \cup \{t\}$
6. Remove t from G and repeatedly delete vertices of degree 1 from G until every vertex left has degree ≥ 2 → cannot be in a cycle ⊛⊛

Theorem: This is a $4\lceil \log_2 n \rceil$-approximation. Fact: there $\exists$ 2-approx based on primal-Dual

Proof: $y_C > 0$ only if C contains $\leq 2\lceil \log_2 n \rceil$ vertices of deg ≥ 3 in the graph at the time we increased y_C. Note that at this time $S \cap C = \emptyset$
⇒ $|S \cap C| \leq 4\lceil \log_2 n \rceil$ at the end of the alg. (see ⊛) ▨

Lemma: If min deg $\delta(G) \geq 2$ then $\exists$ cycle with $\leq 2\lceil \log_2 n \rceil$ vertices of deg ≥ 3.

Proof: Contract all topological paths 
Now there is no vertex of deg = 2 ⇒ perform BFS

If $\exists$ edge between 2 vertices at the same DFS level, we found a cycle
If no such edge exists, we will find a cycle via a cross edge 
If until we find the first cycle the tree is at least binary ⇒ at most $\lceil \log_2 n \rceil$ levels ▨

Dijkstra's Algorithm via Primal-Dual

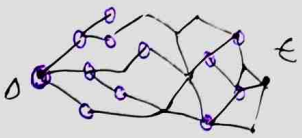
Input: $G=(V,E)$, $s,t \in V$, costs c_e for $e \in E$

Output: s-t path P

Objective: $\min \sum_{e \in P} c_e$

$$\mathcal{S} := \{ S \subseteq V \mid s \in S \ \& \ t \notin S \}$$

↳ S is an s-t cut



$$S \in \mathcal{S}: \ \partial(S) := \{ e \in E \mid e \cap S = 1 \}$$

Integer program:

$$\min \sum_{e \in E} c_e x_e, \quad x_e \in \{0,1\}$$

$$\text{s.t. } \forall S \in \mathcal{S}: \ \sum_{e \in \partial(S)} x_e \geq 1$$

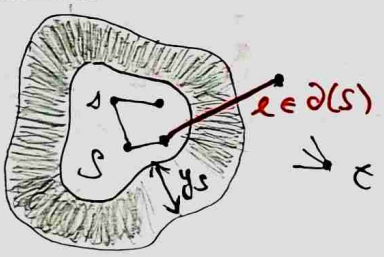
- ∀ s-t path has to contain $e \in \partial(S)$ for ∀ S
- ∀ feasible solution induces a subgraph of G in which s and t are connected by a path
  ↳ one can argue this using min cut = max flow (≥ 1 / ≥ 1)
- infeasible ⇒ does not induce s-t path because some $\partial(S)$ is missed

Dual (we relax $x_e \geq 0$)

$$\max \sum_{S \in \mathcal{S}} y_S, \quad y_S \geq 0$$

$$\text{s.t. } \forall e \in E: \ \sum_{S: e \in \partial(S)} y_S \leq c_e$$

Intuition:



Variables y_S can be interpreted as the width of the boundary of S ... let us call it a moat (příkop)

→ for ∀ $e \in E$, the total width of all moats that cross e must be at most c_e

⇒ the moats cannot overlap too much

Goal: make the moats as wide as possible!

↳ every s-t path must cross all moats

Algorithm:

1. $y \leftarrow 0$, $F \leftarrow \emptyset$
2. While there ∄ s-t path in (V,F):
3. $\quad W \leftarrow$ connected component of (V,F) containing s
4. $\quad$ Increase y_W until for some $e \in \partial(W)$ we have $\sum_{S: e \in \partial(S)} y_S = c_e$
5. $\quad F \leftarrow F \cup \{e\}$
6. Return some s-t path in (V,F)

Lemma: The set F always forms a tree containing s.

⇒ thus the s-t path we return is unique

Proof: Adding $e \in \partial(W)$ an edge from the boundary cannot create any loop because $e \cap W = 1$ ∎

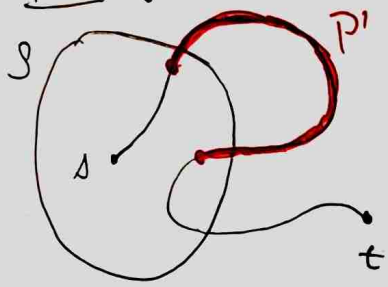
This algorithm really just simulates Dijkstra ⇒ it finds a shortest path
→ but we will argue for this using the standard primal-dual analysis

Theorem: The algorithm finds a shortest s-t path.

Proof: For $\forall$ edge $e \in P$ of the returned path we have $c_e = \sum_{S: e \in \partial(S)} y_S$

$$\Rightarrow ALG = \sum_{e \in P} c_e = \sum_{e \in P} \sum_{S: e \in \partial(S)} y_S = \sum_{S \in \mathcal{S}} y_S \cdot |P \cap \partial(S)| \overset{?}{\leq} \sum_{S \in \mathcal{S}} y_S \leq OPT_D \leq OPT$$

Need: $y_S > 0 \Rightarrow |P \cap \partial(S)| = 1$



Suppose $|P \cap \partial(S)| > 1$ and take a subpath $P' \subseteq P$ joining two vertices of S
→ since $y_S > 0$, when we increased y_S, F was a tree spanning the vertices of $S = W$
$\Rightarrow F \cup P'$ contains a cycle, contradicting the lemma ▨

Steiner Forrest Problem

Input: $G = (V, E)$, costs $c_e \geq 0$ for $e \in E$, $(s_1, t_1), \dots, (s_k, t_k) \in V \times V$
Output: $F \subseteq E$ s.t. $\forall i$: $\exists s_i$-t_i path in (V, F) ↳ not necessarily distinct! $s_1 = s_2 = t_3$ possible
Objective: $\min \sum_{e \in F} c_e$

$$\mathcal{S}_i := \{S \subseteq V \mid |S \cap \{s_i, t_i\}| = 1\}$$

Integer program:

$$\min \sum_{e \in E} c_e x_e, \quad x_e \in \{0,1\}$$
$$\text{s.t.} \sum_{e \in \partial(S)} x_e \geq 1 \quad \text{for } \forall S \in \mathcal{S}_1 \cup \dots \cup \mathcal{S}_k$$

Dual (we relax $x_e \geq 0$)

$$\max \sum_{i=1}^{k} \sum_{S \in \mathcal{S}_i} y_S, \quad y_S \geq 0$$
$$\text{s.t. } \forall e \in E: \sum_{S: e \in \partial(S)} y_S \leq c_e$$

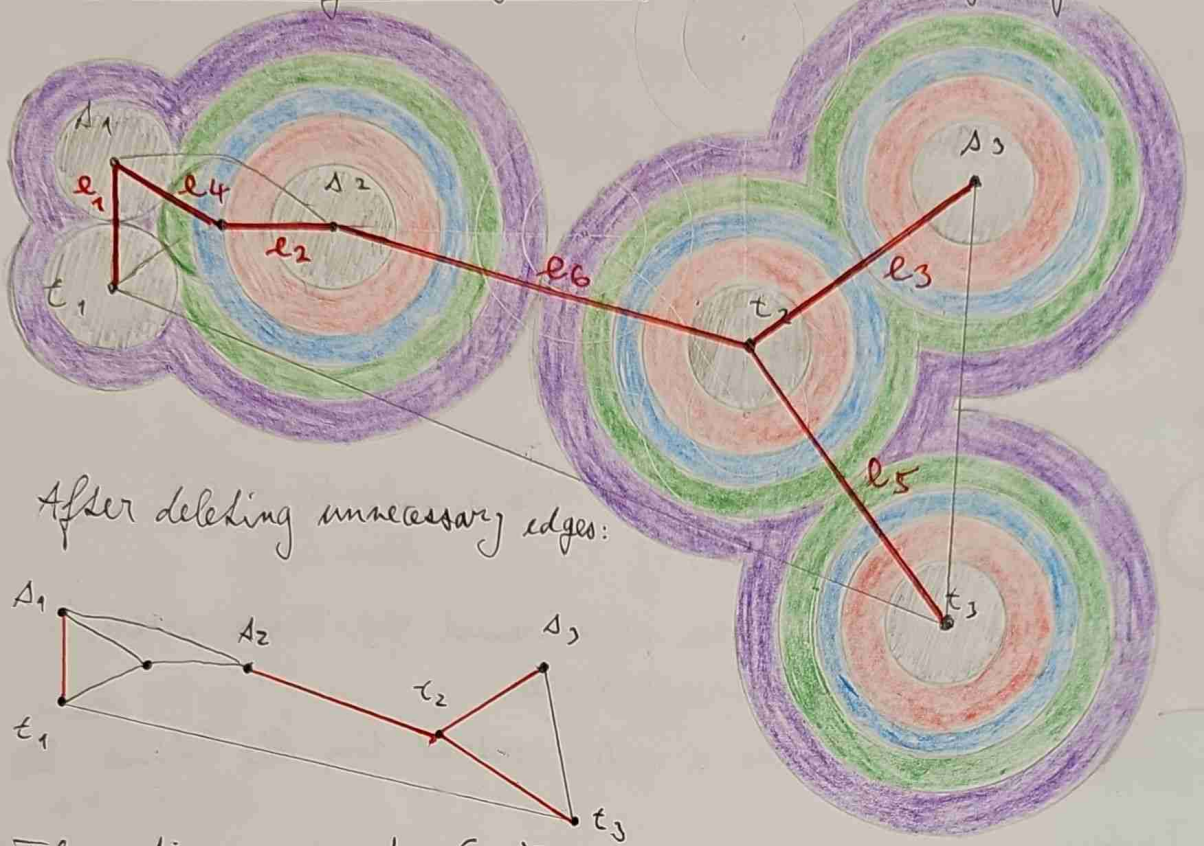
Algorithm:

1. $\vec{y} \leftarrow 0$, $F \leftarrow \emptyset$
2. While not all s_i-t_i are connected in (V, F):
   C is an **active moat** if $C \in \mathcal{C}$ ... it is a component that will have to change for feasibility
3. $\quad \mathcal{C} \leftarrow \{C \subseteq V \mid C$ is a component of (V, F) s.t. $C \in \mathcal{S}_i$ for some $i\}$
4. $\quad$ Increase y_C for all $C \in \mathcal{C}$ uniformly until $\exists e \in \partial(W)$ s.t. $\sum_{S: e \in \partial(S)} y_S = c_e$
   $W \in \mathcal{C}$
5. $\quad F \leftarrow F \cup \{e\}$
6. $F' \leftarrow$ the necessary edges of F

↑ we grow all active moats simultaneously

Step 6 is well-defined because F is always a forrest since we only add edgs from its boundary
$\Rightarrow$ there are unique s_i-t_i paths in F
$\Rightarrow$ we just keep these paths

Visualization of the algorithm: cost c_e = length of e in the picture

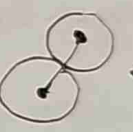


After deleting unnecessary edges:

The active components $C \in \mathcal{C}$ of (V, F) at any given point in the algorithm are the ones that are currently growing ... $\exists i: |S \cap \{s_i, t_i\}| = 1$

Lemma: For $\mathcal{C}$ in each iteration of the algorithm:

$$\sum_{C \in \mathcal{C}} |F' \cap \partial(C)| \leq 2|\mathcal{C}|$$



↳ # times an edge from the final solution (after reduction) crosses a moat

Theorem: The algorithm is a 2-approx.

Proof: It is polynomial and outputs a feasible solution

$$ALG = \sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S: e \in \partial(S)} y_S = \sum_S y_S |F' \cap \partial(S)| \overset{?}{\leq} 2 \sum_S y_S \leq 2 \cdot OPT_D \leq 2 \cdot OPT$$

Why "$\leq$"? We show this inequality by induction on # iterations of the algorithm

→ initially all $y_S = 0$, so it holds

Suppose it holds at the beginning of the i-th iteration of the main loop

- we increase all y_C for $C \in \mathcal{C}$ by the same $\varepsilon \geq 0$

$\Rightarrow \sum_S y_S |F' \cap \partial(S)|$ is increased by $\varepsilon \cdot \sum_{C \in \mathcal{C}} |F' \cap \partial(C)|$

$\Rightarrow 2 \sum_S y_S$ is increased by $2\varepsilon \cdot |\mathcal{C}|$

$\Rightarrow$ by Lemma: $\varepsilon \cdot \sum_{C \in \mathcal{C}} |F' \cap \partial(C)| \leq \varepsilon \cdot 2|\mathcal{C}|$

∎

Proof of Lemma: Fix an iteration i for which we are proving this.

→ at the beginning of iter. i we have $F_i = \{e_1, e_2, \dots, e_{i-1}\}$

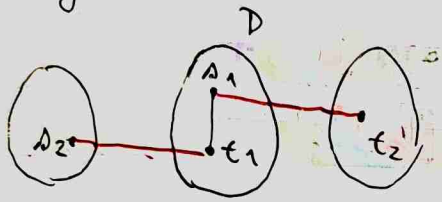
→ let $\mathcal{C}$ be the active components of (V, F_i)

Recall: F is always a forest & F' = final solution after deleting redundant edges
⇒ we keep only the unique $s_i - t_i$ paths in F

→ let $\mathcal{D}$ be the components D of (V, F_i) that are not active ($D \notin \mathcal{C}$) and that satisfy $|F' \cap \delta(D)| \geq 1$

⇒ at least 1 edge from the boundary of D is necessary in the final solution

actually $\forall D \in \mathcal{D}$ we have $|F' \cap \delta(D)| \geq 2$

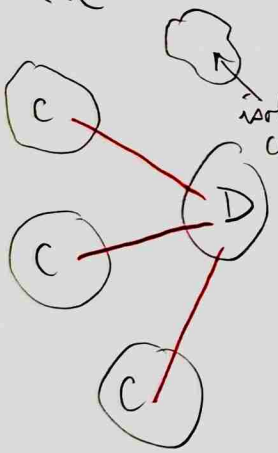


If there were only 1 edge in the final solution, this edge would have to lie on the unique $s_\ell - t_j$ path in F for some j, and $|D \cap \{s_j, t_j\}| = 1$
⇒ but then $D \in \mathcal{C}$ ↯

$$\sum_{C \in \mathcal{C}} |F' \cap \delta(C)| = \sum_{S \in \mathcal{C} \cup \mathcal{D}} |F' \cap \delta(S)| - \sum_{D \in \mathcal{D}} |F' \cap \delta(D)| \leq 2(|\mathcal{C}| + |\mathcal{D}|) - 2|\mathcal{D}| = 2|\mathcal{C}|$$

↳ $S \in \mathcal{C} \cup \mathcal{D}$

$\leq 2(|\mathcal{C}| + |\mathcal{D}|)$ since the graph of components of (V, F_i) connected by the edges of the final solution $F' \setminus F_i$ is a forest in which $|F' \cap \delta(S)|$ = degree of vertex corresponding to S



→ and in a forest $\sum \deg v = 2|E| \leq 2(|V|-1) \leq 2|V|$ ∎

SEMIDEFINITE PROGRAMMING

Def: A symmetric matrix $X \in \mathbb{R}^{n \times n}$ is positive semidefinite, $X \succeq 0$ if

1. $\forall y \in \mathbb{R}^n : y^T X y \geq 0$
2. all eigenvalues of X are non-negative
   ↳ since X is symmetric, eigenvalues are real
3. $X = V^T V$ for some $V \in \mathbb{R}^{n \times n}$
4. $X = \sum_{i=1}^{n} \alpha_i v_i v_i^T$ for some $\alpha_i \geq 0$ and $v_i \in \mathbb{R}^n$

} equivalent definitions

Def: A semidefinite program looks like this: $C, A^{(1)}, \dots, A^{(m)} \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^m$

$$\min / \max \sum_{i,j} c_{ij} x_{ij}, \quad \text{where } X \in \mathbb{R}^{n \times n} \text{ is a PSD variable matrix}$$

$$\text{s.t. } \forall k: \sum_{i,j} a^{(k)}_{ij} x_{ij} = b_k$$

↳ $\forall i,j: x_{ij} = x_{ji}$ & $X \succeq 0$

hidden constraints $y^T X y \geq 0$ $\forall y$

How are SDPs solved?

→ there $\exists$ alg that checks whether $X \in \mathbb{R}^{n \times n}$ is PSD, and if not, outputs y s.t. $y^T X y < 0$

→ SDPs can be (under some reasonable assumptions) solved within an additive error of ε using the Ellipsoid method in polynomial time in # variables # constraints and $\log\left(\frac{1}{\varepsilon}\right)$

⇒ we will treat them like they could be solved exactly since a small enough ε will always be enough

Def: A vector program looks like this: $C, A^{(1)}, \dots, A^{(m)} \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^m$

$$\min / \max \sum_{i,j} c_{ij} \langle v_i | v_j \rangle, \quad \text{where } v_i \in \mathbb{R}^n \text{ for } i = 1, \dots, n$$

$$\text{s.t. } \forall k: \sum_{i,j} a^{(k)}_{ij} \langle v_i | v_j \rangle = b_k \qquad \langle v_i | v_j \rangle = v_i^T v_j$$

Observation: Semidefinite and vector programs are equivalent.

SDP ⇒ VP: given a solution $X \in \mathbb{R}^{n \times n}$ of the SDP, compute $V \in \mathbb{R}^{n \times n}$ s.t. $X = V^T V$

→ let $v_i \in \mathbb{R}^n$ be the ith column of V ⇒ $\langle v_i | v_j \rangle = x_{ij}$

VP ⇒ SDP: given a solution $v_1, \dots, v_n \in \mathbb{R}^n$ of the VP, let $V \in \mathbb{R}^{n \times n} = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{pmatrix}$

⇒ then $X := V^T V$ is PSD and $x_{ij} = \langle v_i | v_j \rangle$

∎

MAX CUT using SDP

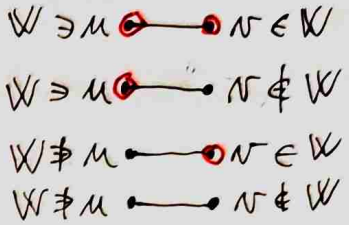
→ $i<j$, take $V=[n]$

Input: $G=(V,E)$, weights $w_{ij}\geq 0$ for $ij\in E$

Output: $W\subseteq V$ ... cut = edges between W and $U:=V\setminus W$

Objective: $\max \sum_{ij\in\partial(W)} w_{ij}$

↳ $\partial(W) := \{ij\in E \mid |\{i,j\}\cap W|=1\}$



Recall: Simply putting each $v\in V$ into W with prob $=50\%$ gives a $\frac{1}{2}$-approximation.

$$\mathbb{E}[ALG] = \sum_{uv\in E} w_{uv}\, P[u\in W, v\notin W] = \frac{1}{2}\sum_{uv\in E} w_{uv} \geq \frac{1}{2}OPT$$

Quadratic integer program:

variables $y_i\in\{-1,+1\}$, $i=1,\dots,n$ (↙ $\notin W$, ↘ $\in W$), obj: $\max \frac{1}{2}\sum_{ij\in E} w_{ij}(1-y_iy_j)$

This solves MaxCut ... $1-y_uy_v = \begin{cases}0 & \Leftrightarrow u,v \text{ are in the same partite}\\ 2 & \Leftrightarrow u,v \text{ are on the opposite sides of the cut}\end{cases}$

Vector programming relaxation:

$$\max \frac{1}{2}\sum_{\substack{ij\in E\\ i<j}} w_{ij}(1-\langle v_i|v_j\rangle) \quad \text{s.t. } \forall i\in[n]: v_i\in\mathbb{R}^n,\ \langle v_i|v_i\rangle = 1$$

unit vectors

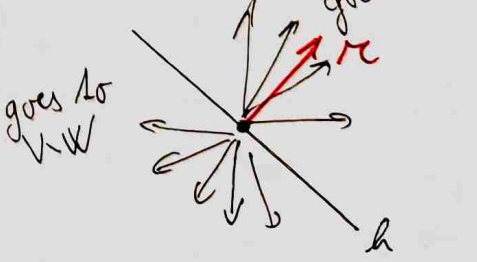
↳ $\|v_i\|=1$

This is a relaxation in the sense that given a solution $y_1,\dots,y_n$ of the QIP, we can define $v_i := (y_i,0,\dots,0)$. Then $\langle v_i|v_j\rangle = y_iy_j$ and $\langle v_i|v_i\rangle = 1$

⇒ If OPT_{VP} is the optimum of the VP then $OPT_{VP}\geq OPT$

Algorithm:

1. $v_1,\dots,v_n \leftarrow$ solve (approximately) the VP
2. pick $r=(r_1,\dots,r_n)$ at random by sampling $\forall r_i\in N(0,1)$ ↙ Normal distrib.
3. $W\leftarrow \{i\in V \mid \langle v_i|r\rangle\geq 0\}$



Intuition: r determines a hyperplane

$h := \{x\in\mathbb{R}^n \mid \langle x|r\rangle = 0\}$

whose normal vector is r

⇒ we categorize $i\in V$ based on which side of h the vector v_i is

- v_i in a similar direction of r ⇝ $i\in W$
- v_i points in the opposite direction than r ⇝ $i\notin W$

Observation: If $r=(r_1,\dots,r_n)$ where $r_i \sim N(0,1)$, then the normalization $r/\|r\|$ is uniformly distributed on the unit sphere in $\mathbb{R}^n$

$\Rightarrow$ the direction of r is completely random

Proof: The probability density function of $r_i \sim N(0,1)$ is $f_i(x)=\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

Since the variables are sampled independently, their joint probability density function is

$$f(\vec{x}) = f_1(x_1)\cdot \dots \cdot f_n(x_n) = (\sqrt{2\pi})^{-n} e^{-\frac{1}{2}(x_1^2+\dots+x_n^2)} = (\sqrt{2\pi})^{-n} \cdot e^{-\frac{1}{2}\|x\|^2}$$

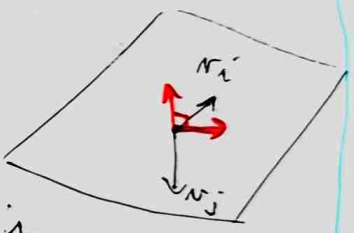
$\Rightarrow f(\vec{x})$ depends only the distance of $\vec{x}$ from the origin ▨

Lemma: $P[ij \in \partial(W)] = \frac{\varphi}{\pi}$ where φ is the angle between v_i and v_j

↳ prob. that $ij \in E$ is in the cut (that is: $|\{i,j\} \cap W| = 1$)

Proof: Fix $ij \in E$. If $v_i = v_j$ then ij is not in the cut so $P=0$ ✓

→ Suppose that $v_i \neq v_j$ and choose two orthonormal vectors $\vec{a}, \vec{b}$ in the 2-dimensional hyperplane determined by v_i and v_j



→ when we generate r, we do it with respect to the standard basis $(e^1, e^2, \dots, e^n)$ of $\mathbb{R}^n$

$\Rightarrow$ but we could equivalently use any other orthonormal basis, since we can just map e^i to the i-th vector of this basis

↳ we rotate the space and the prob. distribution does not change thanks to ↑ IMPORTANT!

$\Rightarrow$ extend $\vec{a}, \vec{b}$ to an orthonormal basis B of $\mathbb{R}^n$

WLOG: assume that r was generated with respect to B

↳ the distribution $r/\|r\|$ is the same (uniform on unit sphere) regardless of the basis

$\Rightarrow$ we now work in this basis

$[v_i]_B = (i_1, i_2, 0, 0, \dots, 0)$ } since they lie in the plane determined by $\vec{a}, \vec{b}$

$[v_j]_B = (j_1, j_2, 0, 0, \dots, 0)$ }

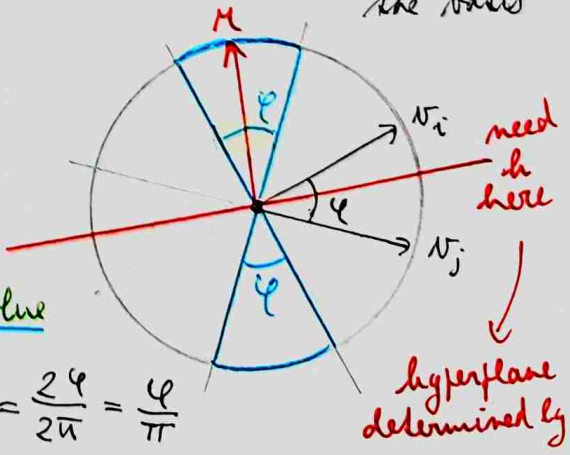
$r = (r_1, r_2, \dots, r_n)$

$\langle v_i | r \rangle = i_1 \cdot r_1 + i_2 \cdot r_2$

$\langle v_j | r \rangle = j_1 \cdot r_1 + j_2 \cdot r_2$

ij is in the cut $\Leftrightarrow$ r is in the blue region

$\rightarrow P[ij \in \text{cut}] = \frac{2\varphi}{2\pi} = \frac{\varphi}{\pi}$



$\Rightarrow$ the assignment of i and j to W depends only on how v_i, v_j look in the 2-dim plane determined by $\vec{a}, \vec{b}$ resp. v_i, v_j ▨

Theorem: The algorithm is a 0.878-approx.

Proof: Individually for $\forall$ edge $ij \in E$ we have:

- $\mathbb{E}[\text{contribution of } ij \text{ to } ALG] = w_{ij} \frac{\varphi}{\pi}$
- Contribution of ij to OPT_{VP}: $\frac{1}{2} w_{ij} (1 - \langle v_i | v_j \rangle) = w_{ij} \frac{1}{2}(1 - \cos\varphi)$

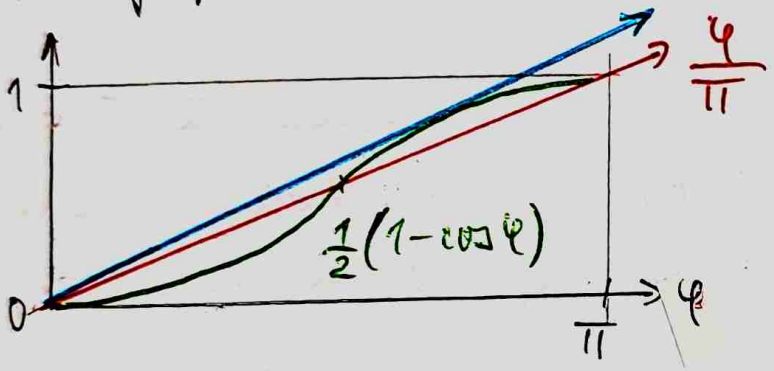
WE NEED: $ALG \geq 0.878 \cdot OPT_{VP}$

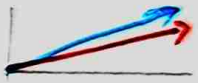
$\hookrightarrow \cos\varphi = \frac{\langle v_i | v_j \rangle}{\|v_i\| \cdot \|v_j\|}$

$\nwarrow \|v_i\| = \|v_j\| = 1$

Claim: $\frac{\varphi}{\pi} \geq 0.878 \cdot \frac{1}{2}(1 - \cos\varphi)$

The graph looks like this:



$\rightarrow$ find a tangent line and take the ratio of the slopes of 

$\blacksquare$

Fact: If $P \neq NP$, then there is no α-approx for $\alpha > \frac{16}{17} \approx 0.941$

Fact: If $P \neq NP$ and we assume the unique games conjecture, then there is no α-approx for $\alpha > 0.878$.

$\hookrightarrow$ we will talk about it at the last lecture

Coloring 3-Colorable Graphs

$|V| = n$

Input: $G = (V, E)$ s.t. $\chi(G) \leq 3$

Output: A coloring of the vertices of G

Objective: min # colors

- **Wigderson's algorithm:**

  👁 given $v \in V$, its neighborhood must be 2-colorable (bipartite) $\Rightarrow$ easy

  1. $n \leftarrow |V|$
  2. While $\exists v \in V$ s.t. $\deg(v) \geq \sqrt{n}$:
  3. take 3 new colors and 3-color $v \cup N(v)$
  4. remove the newly colored vertices from the graph
  5. Color the rest greedily using $\sqrt{n}$ colors

  ↳ if $\forall v$ has deg $< \sqrt{n}$ then $\sqrt{n}$ colors are enough

  } since $\sqrt{n} \cdot \sqrt{n} = n$ there are at most $\sqrt{n}$ iterations

  ↳ always remove at least $\sqrt{n}+1$ vertices

Theorem: This algorithm uses at most $4\sqrt{n}$ colors

↳ $3\sqrt{n} + \sqrt{n}$

Note: $4\sqrt{n} \in O(n^{0.5})$

Fact: Best known is $O(n^{0.211})$

Fact: If $P \neq NP$ then constant approx ratio impossible.

Open problem: Is $O(\log^{\alpha} n)$ possible?

- **Semidefinite programming algorithm**

  Def: $g(n) \in \tilde{O}(f(n))$ if $g(n) \in O(f(n) \log^{c} n)$ for some constant c

  ↳ $n^2 \log^{10} n \in \tilde{O}(n^2)$ ... we forget logarithmic factors

  Goal: Make an algorithm that uses $\tilde{O}(n^{1/4})$ colors ↗ $\chi \leq 3 \Rightarrow \alpha \geq \frac{n}{3}$

  Idea: ① Make an algorithm that for G with $\chi(G) \leq 3$ & max deg Δ, which is at least some constant δ, outputs an independent set of expected size at least $\gamma \cdot n$ for some $\gamma > 0$.

  ⇒ keep finding ind. sets, always using 1 new color to color its vertices

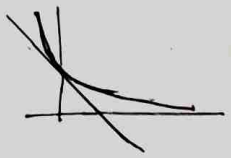
  $n_t :=$ # vertices left after t iterations, $n_0 := n$

  $\mathbb{E}[n_t] = \mathbb{E}[n_{t-1}] - \gamma \cdot \mathbb{E}[n_{t-1}] = (1-\gamma)\mathbb{E}[n_{t-1}] = (1-\gamma)^t n$

  induction ↗

Lemma: After $k = \frac{1}{\gamma}\ln(n)$ iterations: $\mathbb{E}[\#\text{vertices left}] \le 1$

Proof: Using $1-x \le e^{-x}$ we have



$$(1-\gamma)^k n \le e^{-\gamma k} n = e^{-\ln n} n = 1$$ ∎

Corollary: We can set a constant δ — in our case $\delta = 5$ will work nicely and after $O(\frac{1}{\gamma}\log n)$ iterations we will have $\Delta < \delta$
$\Rightarrow$ we will be able to greedily color the rest using $< \delta$ colors

→ our algorithm will produce an independent set with

$$\mathbb{E}[\text{ind. set size}] \ge \Omega\left(n \cdot \Delta^{-\frac{1}{3}} (\ln\Delta)^{-\frac{1}{2}}\right),$$

provided that $\Delta \ge 5$.

→ our γ is the constant from Ω times $\frac{1}{\sqrt[3]{\Delta}\cdot\sqrt[2]{\ln\Delta}}$

↳ so γ increases as Δ goes down

Corollary: We will have a polynomial algorithm that uses $O(\Delta^{\frac{1}{3}}\sqrt{\ln\Delta}\log n)$ colors.

Theorem: We can modify it to use $\tilde{O}(n^{1/4})$ colors.

Proof: Using the idea from Wigderson's algorithm, we color vertices with $\deg(v) > n^{3/4}$, until $\Delta \le n^{3/4}$

$\Rightarrow$ at most $n^{1/4}$ iterations $\Rightarrow$ we use $3\cdot n^{1/4} \in O(n^{1/4})$ colors

After:

$$\Delta^{1/3} \le (n^{3/4})^{1/3} = n^{1/4}, \quad (\ln\Delta)^{\frac{1}{2}} \le (\ln n^{3/4})^{\frac{1}{2}} = \left(\tfrac{3}{4}\ln n\right)^{\frac{1}{2}}$$ ∎

Algorithm for large ind. set in 3-colorable graph

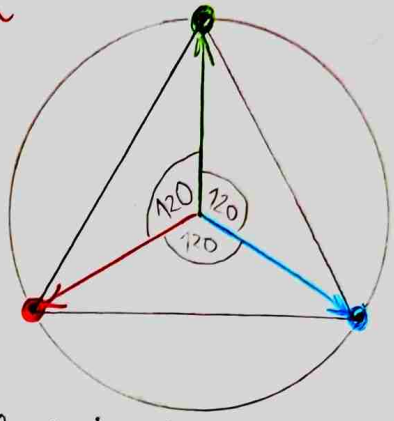
Consider the following vector program:

$\min 0$ ... we just want a feasible solution

variables: $v_i \in \mathbb{R}^n$ for $i \in V = [n]$

s.t. $\forall i \in V$: $\langle v_i | v_i \rangle = \|v_i\| = 1$

$\forall ij \in E$: $\langle v_i | v_j \rangle = -\frac{1}{2}$



Observation: If $\chi(G) = 3$ then $\exists$ feasible solution.

Proof: Map color classes to vertices of an equilateral triangle

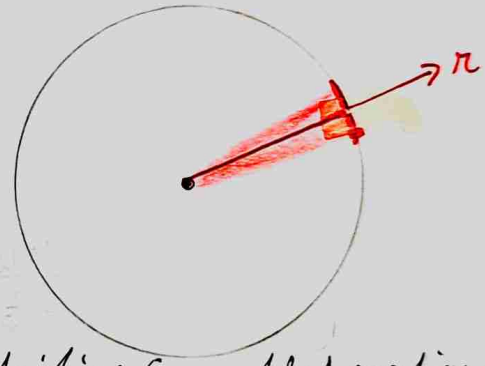
→ if $ij \in E$, v_i and v_j are different vertices

$\Rightarrow$ degree between v_i and v_j $= 120° = \frac{2\pi}{3}$

$\cos\varphi = \frac{\langle v_i | v_j \rangle}{\|v_i\| \cdot \|v_j\|} \Rightarrow \langle v_i | v_j \rangle = 1 \cdot 1 \cdot \cos\left(\frac{2\pi}{3}\right) = -\frac{1}{2}$ ▨

Algorithm:

1. $v_1, \dots, v_n \leftarrow$ solve the VP
2. $r = (r_1, \dots, r_n)$ where $r_i \sim N(0,1)$
3. $\varepsilon \leftarrow \sqrt{\frac{2}{3} \ln \Delta}$ ← relatively big number
4. $S(\varepsilon) \leftarrow \{ i \in V \mid \langle v_i | r \rangle \geq \varepsilon \}$
5. $T(\varepsilon) \leftarrow \{ i \in S(\varepsilon) \mid \forall ij \in E : j \notin S(\varepsilon) \}$
6. Return $T(\varepsilon)$ ← clearly an ind. set



Intuition: Guess that vertices whose vectors are clustering near r are mostly independent

Notation: Probability density function and cumulative dist. function of $N(0,1)$

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad \Phi(x) = \int_{-\infty}^{x} f(t)\,dt, \quad F(x) := \int_{x}^{\infty} f(t)\,dt = 1 - \Phi(x)$$

since $\int_{-\infty}^{\infty} f(t)\,dt = 1$

Lemma 1: For $\forall i \in V$: $P[i \in S(\varepsilon)] = F(\varepsilon)$.

$\Rightarrow$ thus $\mathbb{E}[|S(\varepsilon)|] = n \cdot F(\varepsilon)$ → see alg. for MAX-CUT

Proof: Recall that we can use any orthonormal basis to generate r

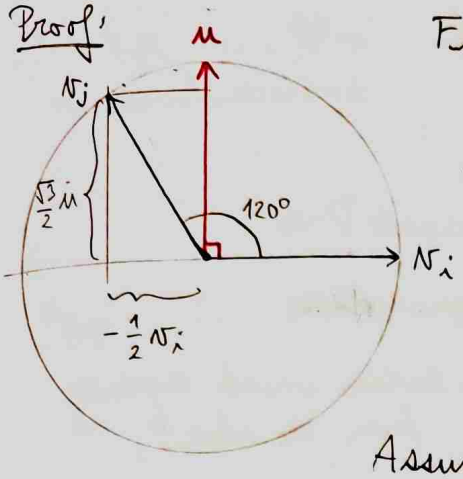
$\Rightarrow$ fix $i \in V$ and let v_i be the first vector of an orthonormal basis B

$\Rightarrow$ written in B, we have $[v_i]_B = (1, 0, \dots, 0)$, $r = (r_1, \dots, r_n)$

$\langle v_i | r \rangle = r_1 \Rightarrow P[i \in S(\varepsilon)] = P[r_1 \geq \varepsilon] = F(\varepsilon)$ since $r_1 \sim N(0,1)$ ▨

Lemma 2: For $\forall ij \in E$: $\underline{P[j \in S(\varepsilon) \mid i \in S(\varepsilon)] \le F(\sqrt{3}\varepsilon)}$

Proof:



Fix $ij \in E$. Note that the angle between v_i, v_j is 120° since they satisfy the VP

→ WLOG assume that r was generated w.r.t. an orthonormal basis with first vector v_i and second u

$\Rightarrow [v_i]_B = (1,0,\dots,0)$, $[u]_B = (0,1,\dots,0)$

$\Rightarrow \langle v_i | r \rangle = r_1 \sim N(0,1)$, $\langle u | r \rangle = r_2 \sim N(0,1)$

Assume $i \in S(\varepsilon)$, so $\langle v_i | r \rangle = r_1 \ge \varepsilon$

↳ $P[j \in S(\varepsilon)] = P[\langle v_j | r \rangle \ge \varepsilon] = ?$

$v_j = -\frac{1}{2}v_i + \frac{\sqrt{3}}{2}u$, so $[v_j]_B = (-\frac{1}{2}, \frac{\sqrt{3}}{2}, 0, \dots, 0)$

$\Rightarrow \langle v_j | r \rangle = -\frac{1}{2}r_1 + \frac{\sqrt{3}}{2}r_2 \le -\frac{\varepsilon}{2} + \frac{\sqrt{3}}{2}r_2$

$\Rightarrow P[\langle v_j | r \rangle \ge \varepsilon] \le P[-\frac{\varepsilon}{2} + \frac{\sqrt{3}}{2}r_2 \ge \varepsilon] = P[r_2 \ge \sqrt{3}\varepsilon] = F(\sqrt{3}\varepsilon)$ ▨

Corollary: By setting ε so that $F(\sqrt{3}\varepsilon) \le \frac{1}{2\Delta}$ we get $\mathbb{E}[|T(\varepsilon)|] \ge \frac{1}{2}\mathbb{E}[|S(\varepsilon)|]$.

↳ if i has d neighbors and $P[j \in S \mid i \in S] \le \frac{1}{2d}$ for $\forall j \in N(i)$, then $P[i \in T \mid i \in S] \ge \frac{1}{2}$ since $P[\exists j \in N(i) \mid i \in S] \le \frac{1}{2d} \cdot |N(i)| = \frac{1}{2}$

Lemma 3: For $\forall x > 0$: $\frac{x}{1+x^2} f(x) \le F(x) \le \frac{1}{x} f(x)$

Proof: Note $f'(t) = -t f(t)$, so $\left[-\frac{1}{t}f(t)\right]' = \frac{1}{t^2}f(t) + f(t) = \left(1 + \frac{1}{t^2}\right) f(t)$

Lower bound: $\left(1+\frac{1}{x^2}\right)F(x) = \int_x^\infty \left(1+\frac{1}{x^2}\right) f(t)\,dt \ge \int_x^\infty \left(1+\frac{1}{t^2}\right) f(t)\,dt = \left[-\frac{1}{t}f(t)\right]_x^\infty = \frac{1}{x}f(x)$

↳ we now divide: $F(x) \ge \frac{1}{x} \cdot \frac{1}{1+\frac{1}{x^2}} f(x) = \frac{x}{x^2+1} f(x)$

Upper bound: $F(x) = \int_x^\infty f(t)\,dt \le \int_x^\infty \left(1+\frac{1}{t^2}\right) f(t)\,dt = \frac{1}{x} f(x)$ ▨

Theorem: The algorithm produces an ind. set of expected size $\mathbb{E}[|T(\varepsilon)|] \ge \Omega\left(n \cdot \Delta^{-\frac{1}{3}} (\ln \Delta)^{-\frac{1}{2}}\right)$.

Proof: We ⊛ want $F(\sqrt{3}\varepsilon) \le \frac{1}{2\Delta}$. By Lemma 3:

$F(\sqrt{3}\varepsilon) \le \frac{1}{\sqrt{3}\varepsilon} f(\sqrt{3}\varepsilon) = \frac{1}{\sqrt{3}\varepsilon} \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{3}{2}\varepsilon^2} \le \frac{1}{2\varepsilon} e^{-\frac{3}{2}\varepsilon^2} \le \frac{1}{2} \cdot \frac{1}{\Delta}$

need:
- $\varepsilon \ge 1$
- $e^{-\frac{3}{2}\varepsilon^2} = \frac{1}{\Delta} \Leftrightarrow \frac{3}{2}\varepsilon^2 = \ln(\Delta)$

$\Rightarrow$ set $\varepsilon := \sqrt{\frac{2}{3}\ln\Delta}$ (and end the ind. set. stage of the coloring alg. once $\varepsilon < 1 \dots \Delta \le 5$)

By ⊛ and Lemma 1 and Lemma 3 we have:

$\mathbb{E}[|T(\varepsilon)|] \ge \frac{1}{2}\mathbb{E}[|S(\varepsilon)|] = \frac{1}{2} n F(\varepsilon) \ge \frac{n}{2} \cdot \underbrace{\frac{\varepsilon}{1+\varepsilon^2}}_{\ge \frac{1}{2\varepsilon} \approx (\ln\Delta)^{-\frac{1}{2}}} \frac{1}{\sqrt{2\pi}} e^{-\frac{\varepsilon^2}{2}} \ge \Omega\left(n \cdot \Delta^{-\frac{1}{3}} (\ln\Delta)^{-\frac{1}{2}}\right)$

$e^{-\frac{3}{2}\varepsilon^2} = \left(e^{-\frac{\varepsilon^2}{2}}\right)^3 = \left(\frac{1}{\Delta}\right)^3 = \Delta^{-\frac{1}{3}}$

▨

ONLINE ALGORITHMS

- input is revealed one by one – very common in real-world scenarios, datastructures, ...

Def: An online algorithm is R-competitive if $\exists C \geq 0$ s.t.

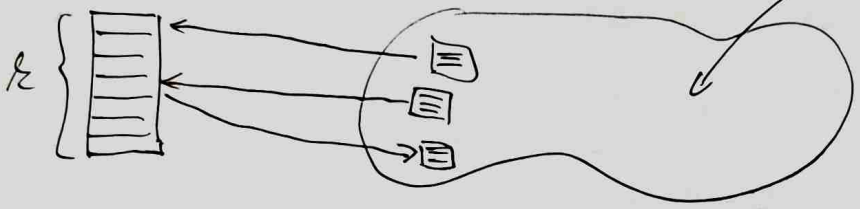
$$ALG(I) \leq R \cdot OPT(I) + C$$ ← additive constant?

for all finite input sequences I. ← for minimization

↳ OPT is offline – knows the entire input sequence from the start

Paging / Caching

k ... size of fast memory



large disk space / cloud storage

↳ slow memory

→ programs need to read pages from the slow memory

Input: k = fast memory size,

N = universe size ... we have pages $N = \{0, 1, 2, \dots, N-1\}$

$C_0 \in [N]^k$ = initial configuration of the fast memory ($[N]^k = \{A \subseteq N \mid |A| = k\}$ — can also be empty)

$r_1, r_2, \dots, r_n \in N$ = input sequence of page requests

Output: Sequence of configurations of the fast memory

$C_1, C_2, \dots, C_n \in [N]^k$ s.t. $r_i \in C_i$

Objective: $\min \sum_{i=1}^{n} |C_i \setminus C_{i-1}|$ ← min # accesses to slow memory (page faults)

Paging strategies

- LRU: least recently used
- FIFO: first-in, first-out

} only evict a page when $r_i \notin C_{i-1}$

- FWF: flush-when-full: whenever there is a page fault and the cache is full, evict all pages ... this is called a flush

- RAND: replace a random page

→ offline strategy:

- LFD (longest forward distance): replace the page whose next request is latest

Fact: LFD is an optimal (offline) algorithm for paging

(h,k)-paging

→ OPT can see the entire input sequence so he has an unfair advantage

⇒ in (h,k)-paging we give online algorithm fast memory of size k but optimum gets fast memory of size only $h \le k$

↳ we still compare # page faults

Fact: LRU, FIFO, FWF are all $\frac{k}{k-h+1}$-competitive

↳ for $k = 2h$ we get $\frac{2h}{h+1} < 2$-competitive

↳ for $k = h$ (normal paging) we get k-competitive

they are actually the best

Theorem: No deterministic paging algorithm is better than k-competitive.

Proof: first **Lemma**: For any sequence $r_1, \dots, r_n$ of requests chosen from $N = k+1$ pages, $LFD = OPT \le \frac{n}{k}$ page-faults.

proof of lemma: suppose that at i-th request r_i, LFD evicted some page p

↳ there are $k+1$ pages in total, r_i ——— p (≥ $k-1$)

↳ from the k pages in the cash, p was the one requested the latest in the future

→ all $k-1$ other pages appear between r_i and the next request of p, which is now the only page not in cache

⇒ LFD faults at most once every k requests

We construct a "cruel sequence" for our deterministic algorithm

→ let $N = k+1$ and generate an input sequence $r_1, \dots, r_n$ s.t. $r_{i+1} \notin C_i$

↳ it is online, so we can use ALG's previous configuration

→ by design ALG faults always so $ALG = n$

⇒ $\frac{ALG}{OPT} \ge k$, and no additive constant will help since we can make n arbitrarily long

Note: if $N > k+1$ then just use the first $k+1$ pages

∎

Marking algorithms

- fix an input sequence and divide it into phases:
  - the first phase starts at the first request
  - at the start of a phase, all pages are unmarked
  - whenever a page is requested, it becomes marked
  - if there are already k distinct marked pages and a $(k+1)$-th page should be marked, the phase ends and this page is marked in the new phase

Example: $k=3$, input = $\overline{1\,2}\,1\,1\,2\,\overline{3}\,1\,3,\overline{4}\,2\,\overline{4}\,5\,\overline{2},\overline{1}\,2\,\overline{3},\overline{4}\,5\,\overline{}$

Def: An algorithm is marking if it never evicts a marked page

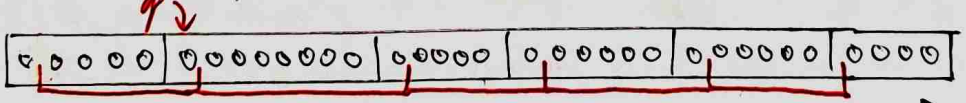
FWF is the most trivial marking alg. provided we start with empty cache

LRU is a marking algorithm
↳ when a page is marked it becomes the most recently used
↳ LRU evicts the least recently used

FIFO is not marking – it evicts based on how long has a page been in the cache, not based on how recently it was requested

Theorem: Any marking algorithm is k-competitive.

Proof: Any marking alg. has $\leq k$ page faults per phase since there are k distinct page references in the phase, and once a page has been requested, it cannot be evicted. Thus we cannot fault twice on the same page



last phase can be short

OPT faults at least once in each red segment
↳ it has in its memory the first page q from phase i
→ in the rest of phase i + first request of phase $i+1$, it will receive other $(k-1)+1$ distinct page requests, none of them being q
→ it must fault at least once ($k-1$ spaces in memory besides q)

↳ thus $OPT \geq \#\text{phases} - 1 \Rightarrow ALG \leq k \cdot OPT + k$
$ALG \leq k \cdot \#\text{phases}$

Corollary: LRU, FWF and FIFO are all k-competitive.

Proof: LRU is marking
FWF is marking (after we initially delete the entire cache)
→ FIFO is not marking but it does not fault on the same page twice in a single phase ... once it faults on p, p is the last-in ▨

Randomized algorithms

→ rand. algs. have an advantage over deterministic ones

? Can we do better than k-competitive ?

Proposition: The basic randomized algorithm RAND is not better than k-competitive.

Proof: Suppose $N = k+1$ ... we have pages $0, 1, \dots, k$

requests: $(0, 2, \dots, k)^* (1, 2, \dots, k)^* (0, 2, \dots, k)^* (1, 2, \dots, k)^* \dots$ ← repeated many times

- OPT pays 1 for each block – simply replace 0/1 by 1/0
- RAND removes the "correct" element for the block with prob $= \frac{1}{k}$

⇒ if we make $*$ = num repetitions large enough, this will be almost like a Bernoulli distribution

⇒ $\mathbb{E}[\text{RAND}]$ per block $\geq k$

Finally, we can dominate any additive constant by getting sufficiently many blocks ▨

Def: The k-th harmonic number is $H_k := 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \sim \ln(k)$

• The MARK algorithm

Algorithm (MARK): Erase cache at the beginning. Then maintain a list of marked pages (as in the definition) and on page fault: evict a random unmarked page
↳ when all pages in cache are marked and there is a page fault,
this means that a new phase just started
⇒ so all pages get unmarked and we can evict anything
⇒ once the new page is brought in, it is marked (as it is the first page in a new phase)

Exercise: In general, MARK is not H_k-competitive.

↳ consider $k = 2$, $N = 4$

Theorem: MARK is $2H_k$-comp. Moreover, if $N = k+1$, then it is H_k-comp.

Proof: WLOG assume that we start with an empty cache
↳ otherwise hide this into the additive constant

- Fix an input sequence $r_1, r_2, \dots, r_n$ and consider its phases

Def: For a given phase i, we call the pages that were in the cache at the very end of phase $i-1$ (when the first request of phase i arrived) the old pages. If during phase i we request a page that is not old, it is new.

👁 the first request of each phase is to a new page

Analyzing phase i: → in total k distinct pages requested in phase i

$m_i :=$ # distinct new pages requested in phase i

👁 all requests within a phase are to either new or old pages, and
→ the worst case for the online player is that first requests to new pages precede all requests to old pages
↳ WLOG assume that this happens → we only look at first requests since we are marking ↳ subsequent requests do not fault

- the m_i (first) requests to new pages cause m_i faults
- what about the following $k-m_i$ (first) requests to old pages?

$$P[\text{fault on (first) request to } 1^{st} \text{ old page}] = 1 - \frac{k-m_i}{k} = \frac{m_i}{k}$$

$$P[\text{fault on (first) request to } 2^{nd} \text{ old page}] = 1 - \frac{k-m_i-1}{k-1} = \frac{m_i}{k-1}$$

$$P[\text{fault on (first) request to } j^{th} \text{ old page}] = 1 - \frac{k-m_i-(j-1)}{k-(j-1)} = \frac{m_i}{k-j+1}$$

$$P[\text{old page in cache}] = \frac{\#\text{ old unmarked pages in cache}}{\#\text{ old unmarked pages}}$$

$$P[\dots (k-m_i)^{th}] = \frac{m_i}{k-(k-m_i)+1} = \frac{m_i}{m_i+1}$$

$$\mathbb{E}[\text{faults in phase } i] \le m_i + \frac{m_i}{m_i+1} + \frac{m_i}{m_i+2} + \dots + \frac{m_i}{k}$$

$$\le m_i + \left(\frac{m_i}{2} + \dots + \frac{m_i}{m_i}\right) + \frac{m_i}{m_i+1} + \dots + \frac{m_i}{k} = m_i H_k$$

Bounding OPT:

👁 in any two consecutive phases $i-1$ and i, there are $k+m_i$ distinct page requests and OPT must fault on the m_i new pages

$$\left.\begin{aligned} OPT &\ge m_2 + m_4 + m_6 + \dots \\ OPT &\ge m_3 + m_5 + m_7 + \dots \end{aligned}\right\} OPT \ge \frac{1}{2}\sum m_i$$

↑ $+m_1$ since we start with empty cache

$$\Rightarrow \quad \frac{\mathbb{E}[\text{MARK}]}{OPT} \leq \frac{H_k \sum m_i}{\frac{1}{2}\sum m_i} = 2H_k$$

see proof that marking algs are k-comp.

- if $N = k+1$: Then OPT faults at least once per phase and $m_i = 1$ so $OPT \geq \#\text{phases}$ and $\mathbb{E}[\text{MARK}] \leq H_k \#\text{phases}$ ▨

Theorem: No randomized alg. can be better than H_k-competitive.

↳ not even when $N = k+1$

Proof: Assume $N = k+1$. Pages $= 0, 1, \ldots, k$

→ we know the prob. distribution used by ALG, so $\forall$ page i we can calculate

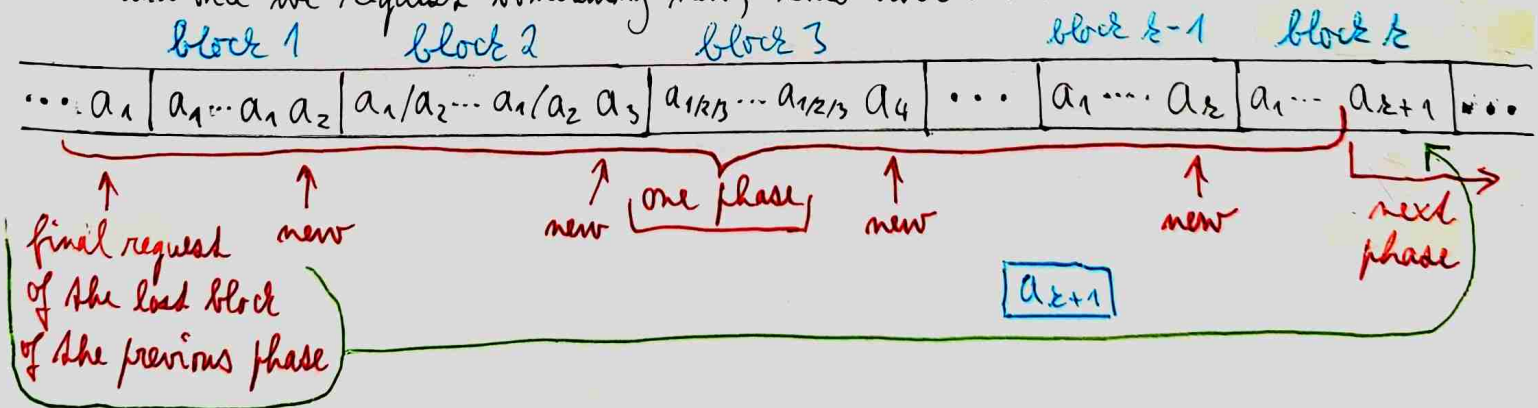
$p_i := P[\text{page } i \text{ is currently not in ALG's cache}]$ — currently ⇒ p_i is always being updated

$\mathbb{E}[\text{fault when requesting page } i] = p_i$ $\quad \sum_{i=0}^{k} p_i = 1$ ← since $N = k+1$

↳ there is only 1 true page that would vanish

⇒ we will construct an evil input sequence for ALG

- it will be divided into phases
  ↳ we will maintain marks (even though ALG does not need to be marking)
- each phase will be divided into k blocks → at the end of j-th block we request for the first time the $(j+1)$-th marked page of this phase
  ↳ at the start of the first block... 1 marked page
  ↳ start of the j-th block... j marked pages
  ↳ start of the k-th block... k marked pages
  ⇒ the next phas begins with the last request of the k-th block
  → in block j, we can use all the pages requested in the earlier blocks and once we request something new, this block ends



Goal:

block 1	block 2	block 3	...	block k-1	block k
$\geq \frac{1}{k}$	$\geq \frac{1}{k-1}$	$\geq \frac{1}{k-2}$		$\geq \frac{1}{2}$	≥ 1

$\mathbb{E}[\#\text{faults in block } j] \geq \frac{1}{k-j+1}$

$\mathbb{E}[\#\text{faults}]$:

$$\Rightarrow \mathbb{E}[\#\text{faults per phase}] \geq \frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{2} + 1 = H_k$$

Moreover, we can destroy additive constant by having many phases

Note: OPT can service each phase with exactly 1 page fault

Constructing block j so that $\mathbb{E}[\#\text{faults on block } j] \geq \frac{1}{\ell - j + 1}$

Recall: $p_i = P[\text{page } i \text{ is not in the cache}]$

$\mathbb{E}[\text{fault when requesting page } i] = p_i$

$\sum p_i = 1$

want: $\mathbb{E} \geq u$

! we have marks but ALG does not have to be marking
⇒ it might evict a marked page
⇒ marked page can have $p_i > 0$

M := set of marked pages ... $|M| = j$

$u := \ell + 1 - j$ ... # unmarked pages

p_i change fluidly based on requests and so does P_{OUT} ... is updates

$P_{OUT} := \sum_{i \in M} p_i$ ... prob that a marked page is not in the cache

Case A: $P_{OUT} = 0$ ... then all marked pages are 100% in cache

⇒ simply request the unmarked page $a \notin M$ that has the highest prob $p_a \geq \frac{1}{u}$

Case B: $P_{OUT} > 0$ ... ALG maybe evicted some marked pages

⇒ let $\varepsilon := p_m > 0$ for some $m \in M$ with $p_m > 0$

⇒ we can keep requesting already marked pages and they will increase the desired $\mathbb{E}$

↳ always request the marked page $b \in M$ with the highest $p_b > 0$

→ STOP when the total $\mathbb{E}$ of this block exceeds $\frac{1}{u}$ ... we are happy

↳ then simply request any unmarked page to end the block

↳ or when $P_{OUT} \leq \varepsilon$ ← the $\mathbb{E}$ gains would be negligible

↳ then request the unmarked page $a \notin M$ with the highest p_a

$$p_a \geq \frac{1 - P_{OUT}}{u} \geq \frac{1 - \varepsilon}{u}$$

$$\Rightarrow \mathbb{E}[\#\text{faults}] \geq \varepsilon + p_a \geq \varepsilon + \frac{1-\varepsilon}{u} = \frac{1 + (u-1)\varepsilon}{\varepsilon} \geq \frac{1}{\varepsilon}$$

($u \geq 1$)

first request gives at least $p_m = \varepsilon$

last request

∎

Fact: There exists a H_ℓ-competitive randomized alg.

Generalizations of Paging

→ each page has a size = how much it takes up in the cache
cost = how expensive is it to load it from the slow memory

- weighted paging: when size = 1 but cost can vary
  ↳ the offline variant can be modeled using flow networks
- file caching: cost = 1 but size can vary, or cost = size
  ↳ the offline variant is NP-hard

$H_k \sim \ln(k)$

Fact: There exist $O(\log k)$-competitive randomized algorithms.

SCHEDULING ON UNIFORMLY RELATED MACHINES

k-Server Problem

k servers

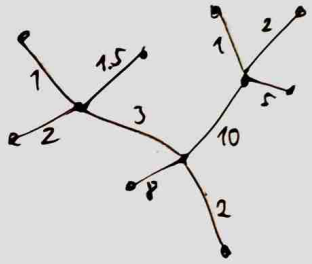
Input: Metric space M, initial position of servers $s_1, s_2, \dots, s_k \in M$
request sequence $r_1, r_2, \dots, r_n \in M$

Output: We need to serve every request
↳ when a request $r_i \in M$ arrives to a point without a server, we need to move a server to that point

Objective: min. total distance traveled by the servers

Paging with a fast memory of size k is the k-server problem for a metric space of size $|M| = N$ = slow memory size, where the distance between any two points is 1.

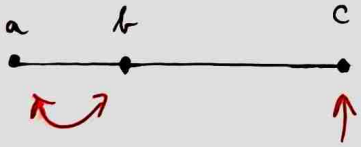
Def: (M, d) is a tree metric if there is a tree T whose vertices are the points of M s.t.,
$d(x,y)$ = distance between x and y in T



Fact: The offline version of the k-server problem can be modeled using flow networks in time $O(k n^2)$ where n = # requests.

Observation: The greedy algorithm is not competitive.
↳ competitive ratio = ∞

Proof: $k = 2$, M = 3 points on a line:



requests: c, a, b, a, b, a, b, ...

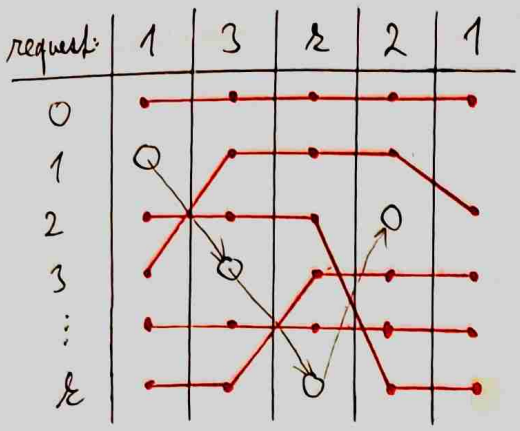
Greedy = serve each request with the nearest server

The k-Server Conjecture

Theorem: If $|M| \geq k+1$, then no deterministic online alg. can be better than k-competitive.

Proof: Select $k+1$ points from M ... the k initial positions of the servers + 1 extra
→ make a cruel sequence: always request the point without a server
↳ hole := the unique point without a server

points = $0, 1, 2, \dots, k$



red lines = movement of servers

→ the hole also "moves"

$ALG = d(r_1, r_2) + d(r_2, r_3) + \dots$ (because of how we generate the requests)

↖ we use that d is symmetric

The movement of the hole uniquely determines the movement of the servers

→ and cost ALG = total movement of the hole

⇒ for each red line (server movement of ALG), consider an algorithm whose hole-movement is this red line ⇝ algorithms $A_1, A_2, \dots, A_k$

$ALG = A_1 + \dots + A_k$ & each A_i serves the sequence ⇒ $\exists i: A_i \leq \frac{1}{k} ALG$

- we can kill any additive constant of ALG by making the sequence long enough
- formally, the initial position of each A_i is the same as ALG
  → we need to pay for moving the servers of A_i to their "correct" initial positions

Conjecture: For any metric space, there is a k-comp. deterministic alg.

Theorem: There exists a determ. alg that is $(2k-1)$-comp in any metric space.

↳ we describe the algorithm below

Def: A configuration of k servers is a multiset of size k

↳ more servers can be at the same point

Def: Given a sequence of requests σ, an initial config. C_0 and a final configuration C, the work function

$W_\sigma(C)$:= optimal offline cost of serving σ, starting from C_0 and finishing in C

Algorithm:

Suppose we have already served $\sigma = r_1, \dots, r_i$ and we get a new request $r \notin C$

1. Determine which server $s \in C$ of the current configuration minimizes

$$\min_{s \in C} W_{\sigma r}(C - \{s\} \cup \{r\}) + d(s, r)$$

↑ concatenation of σ and r is $\sigma r = r_1, \dots, r_i, r$

2. Move the best s to r

Note: We can calculate the work function using dynamic programming

k-Server on Trees

Configuration = multiset

→ metric space (M, d)

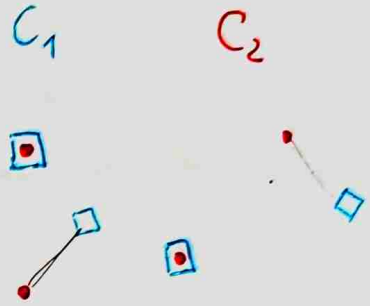
Notation: C_0 = initial configuration of servers

$\sigma = r_1, r_2, \dots, r_n$ = input sequence

C_i = configuration after serving r_i ← $r_i \in C_i$

Def: We denote the distance between two configurations by $d(C_1, C_2)$

$d(C_1, C_2)$ = min. cost of a perfect matching



→ we can state the objective of k-server problem as:

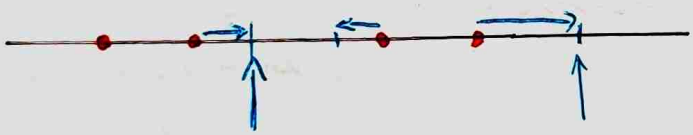
$$\min \sum_{i=1}^{n} d(C_0, C_i)$$

- Double-Coverage (DC) algorithm for $M = \mathbb{R}$

↳ line with Euclidean metric

DC: If r_i falls outside the convex hull of the servers (it is way to the left/right), serve it with the nearest server

↳ Otherwise, if it falls between two adjacent servers, move both by the same speed towards r_i, until one of them reaches it



● = servers

↑ = requests

Theorem: DC is k-competitive for $\mathbb{R}$.

Proof: We will use potential functions

→ we will keep track of the configurations of ALG: $C_0, C_1, C_2, \dots, C_n$

and of the adversary algorithm: $C_0 = A_0, A_1, A_2, \dots, A_n$

- C, A ... current configurations
- C', A' ... configurations after processing the next request

$\Phi(C_i, A_j) \geq 0$: compares any two configurations of ALG and the adversary

↳ we get money when adversary moves, and we pay when ALG moves

→ we will chose Φ so that it has the following two properties

① $\Phi(C,A') - \Phi(C,A) \leq k \cdot d(A,A')$ ⟵ we get at most k times how much adversary moved

how much money we get — movement cost of the adversary

② $\Phi(C',A') - \Phi(C,A') \leq -d(C,C')$ ← we pay for the movement + we can throw money away

how much we "get" ↪ we need to pay — movement cost

Sum it up: ① + ② $= \Phi(C',A') - \Phi(C,A)$

$\Rightarrow \Phi(C_m,A_m) - \Phi(C_0,A_0) \leq k \cdot OPT - ALG$

$\Rightarrow ALG \leq k \cdot OPT + \Phi(C_0,A_0) - \Phi(C_m,A_m)$ → k-competitive

addditive constant

What is the potential? $C = \{s_1, s_2, \dots, s_k\}$

$\Phi(C,A) := k \cdot d(C,A) + \Sigma_{DC}$, where $\Sigma_{DC} := \sum_{i<j} d(s_i,s_j)$

↪ distance between all $\binom{k}{2}$ pairs of servers in C

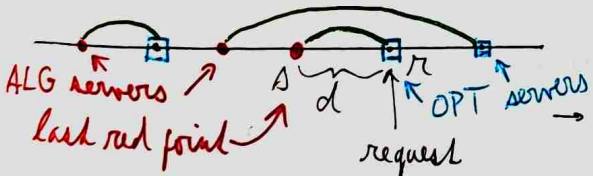
Proving ①: Σ_{DC} does not change

⇒ want: $k \cdot d(C,A') - k\,d(C,A) \leq k\,d(A,A')$

$\Leftrightarrow d(C,A') \leq d(C,A) + d(A,A')$ ✓ ... Δ inequality for matchings

Proving ②: How does Φ change when DC moves?

• case A: DC moves 1 server ... so the request was on the edge



Adversary already moved, so $r \in A'$

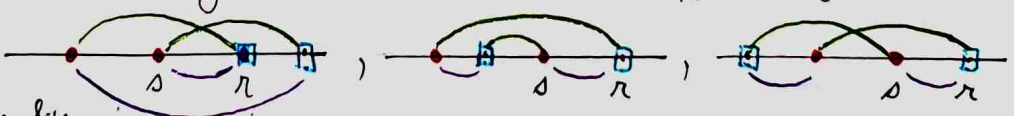
→ the server $s \in C$ that moves is on the edge

⇒ Σ_{DC} increases by $(k-1) \cdot d$

How does $d(C,A') \to d(C',A')$ change? ↪ distance $|r-s|$

there is a min matching between C and A' in which $s \to r$

↳ consider:

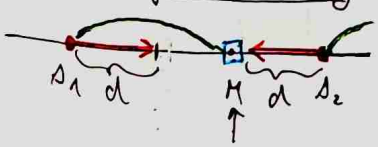


⇒ $d(C,A')$ decreases by at least d

⇒ $\Delta\Phi \leq -k \cdot d + (k-1)d = -d = -d(C,C')$

• case B: DC moves 2 servers s_1, s_2 towards r a distance d

- changes to matching: similar to before, there ∃ min C-A' matching in which one of the servers is matched to r ... suppose s_1 is matched to r



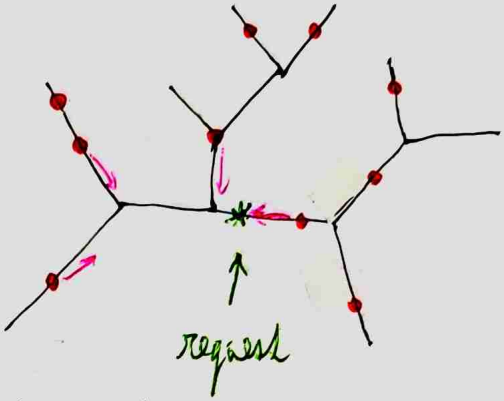
- s_1 moving to r decreases match cost by at least d
- s_2 moving to r increases match cost by at most d

} match cost doesn't increase

- changes to distances Σ_{DC}
  - → because s_1 and s_2 move in opposite directions, their effect on all servers $s \notin \{s_1, s_2\}$ cancels out
  - → but they move closer to each other, so Σ_{DC} decreases by $2d$

$$\Rightarrow \Delta\Phi \leq 0 - 2d = -d(C, C')$$ ▨

- **DC Algorithm for Tree metrics**
  - → move all servers that "see" the request by the same speed until one reaches it



Theorem: DC-Tree is k-competitive.

Proof: By the same argument, but we need to analyze the second inequality (when ALG moves) a bit more carefully

- → suppose that ℓ servers move a distance d towards the request r
- 🧐 there is again a min. C-A' matching in which one of the moving servers is matched to r
  - → this server decreases the match cost by at least d
  - → the other servers increase it by at most $(\ell-1)\cdot d$

} total increase by at most $(\ell-2)d$

- changes to Σ_{DC} let s be a stationary server
  - exactly 1 moving server is moving away from s
    - ↪ the other $(\ell-1)$ are moving towards s

} Σ_{DC} decreases by $(\ell-2)d$

  - each pair of moving servers is getting closer to one another by $2d$

$$\Phi = k \cdot d(C, A) + \Sigma_{DC}$$

$$\Delta\Phi \leq k\cdot(\ell-2)d - \binom{\ell}{2}\cdot 2d - (k-\ell)(\ell-2)d = \cdots = -d\cdot\ell = -d(C, C')$$ ▨

Randomized k-Server Algorithms

🧐 In uniform metric spaces (all distances are the same) no randomized algorithm can be better than H_k-competitive

↪ because this is just paging

$H_k \sim \ln(k)$

Fact: The lower bound for the general case is $\Omega(\log^2(k))$.

APPROXIMATING METRIC SPACES BY A TREE METRIC

Given a finite metric space (V,d), we want a tree metric (V',T) s.t.

$V \subseteq V'$ and $\forall u,v \in V'$: T_{uv} = distance between u and v in T

tree with edges of non-negative lengths

Def: If $\forall u,v \in V$: $d_{uv} \le T_{uv} \le \alpha \cdot d_{uv}$,
we say that the embedding of d into (V',T) has distortion α

Fact: There are metric spaces for which distortion $\in \Omega(n)$ ($|V| = n$)

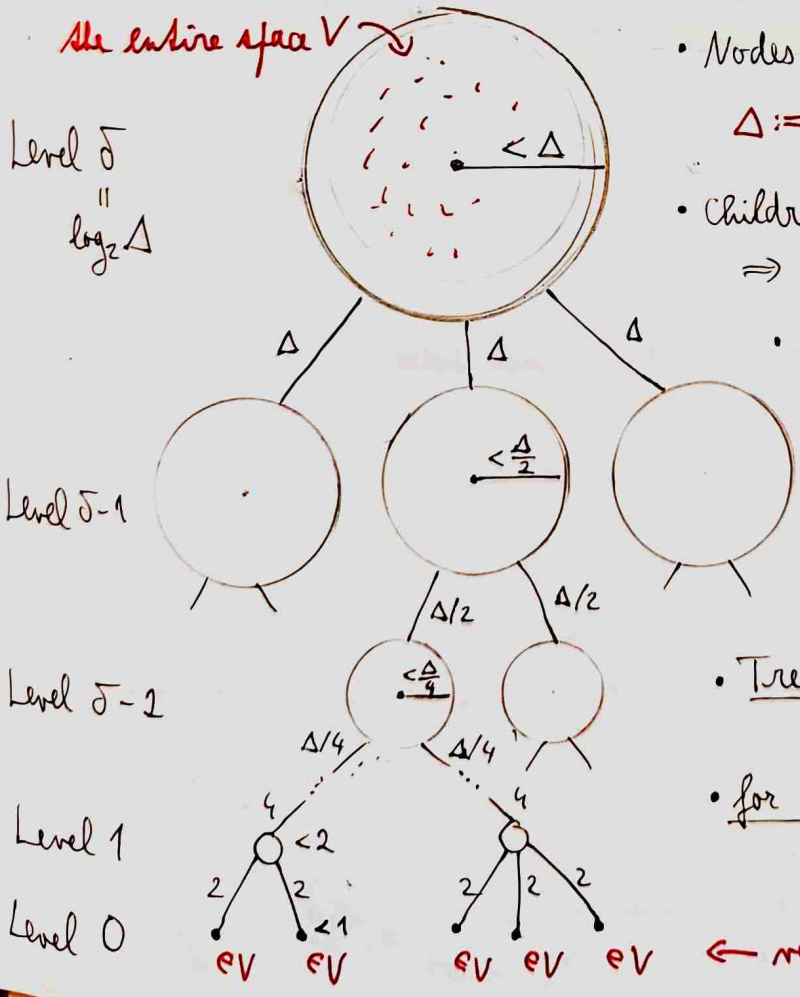
→ if we have a low-distortion embedding, we can convert many problems for the variant of tree metrics, which is often much simpler

Theorem: There is a randomized algorithm that for a given (V,d) s.t. $d_{uv} \ge 1$ $\forall u \neq v \in V$ (rescale) produces (V',T) with $\mathbb{E}[\text{distortion}] \in O(\log n)$

$$d_{uv} \le T_{uv} \quad \& \quad \mathbb{E}[T_{uv}] \le O(\log n) \cdot d_{uv}$$

Fact: There $\exists$ spaces for which this is the best possible.

- Tree ~ Hierarchical cut decomposition of (V,d)



- Nodes of the tree = subsets of V

$\Delta := 2^{\delta} > 2 \cdot \max_{u,v} d_{uv}$ ← smallest power of 2 larger than this

- Children of a node S partition S
  ⇒ each level is a partition of the entire space
  - the vertices of V in a node S at level i are all contained in a ball with center $\in V$ and radius $\in [2^{i-1}, 2^i)$

⇒ for level 0 = leaves we have radius < 1
⇒ so leaf nodes contain a single $v \in V$

- Tree metric: length of edge from level i to level $i-1$ is 2^i
- for $u,v \in V$: T_{uv} = distance along the tree between the two leaves corresponding to u and v
  ↪ get to closest common ancestor

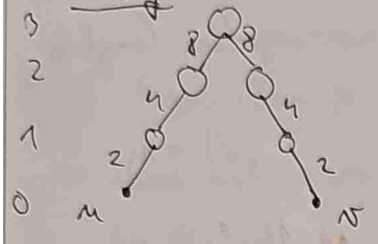
Lemma: If T is a hierarchical decomposition of (V, d) then

$$\forall u, v \in V: \quad \underline{d_{uv} \le T_{uv} \le 2^{i+2}}$$

where i is the level of the LCA of u and v

↳ least (closest) common ancestor

Proof:



$$T_{uv} = 2 \cdot \sum_{j=1}^{i} 2^j = 2(2^{i+1} - 2) = 2^{i+2} - 4 \le 2^{i+2}$$

- let S be the LCA of u and v (at level i)
  - ↳ $d_{uv} < 2^{i+1}$ since $S \sim$ ball of radius $< 2^i$

$$\Rightarrow T_{uv} = 2 \cdot 2^{i+1} - 4 \ge 2^{i+1} > d_{uv}$$

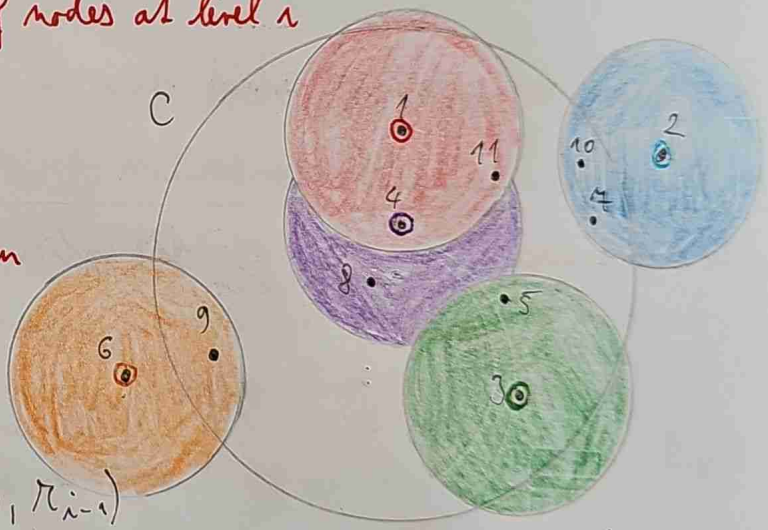
↑ $i \ge 1 \Rightarrow 2^{i+1} \ge 4$

Algorithm:

1. $\Pi \leftarrow$ random permutation of $V = [n]$
2. $\delta \leftarrow$ smallest δ s.t. $2^\delta > 2 \cdot \max_{u,v} d_{uv}$
3. pick $r_0 \in [1/2, 1)$ unif. randomly and set $\underline{r_i \leftarrow 2^i r_0}$ for $i = 1, 2, \dots, \delta$

   r_i = radius at level i
4. $\mathcal{C}(\delta) \leftarrow \{V\}$ ... $\mathcal{C}(i)$ = set of nodes at level i

   ↳ created the root
5. For $i = \delta$ down to 1:
6. $\quad \mathcal{C}(i-1) \leftarrow \emptyset$ — make children
7. $\quad$ For $\forall C \in \mathcal{C}(i)$:
8. $\quad\quad S \leftarrow C$
9. $\quad\quad$ For $j = 1, \dots, n$:
10. $\quad\quad\quad W \leftarrow S \cap B(\Pi(j), r_{i-1})$
11. $\quad\quad\quad$ If $W \neq \emptyset$: create a child of C and add W into $\mathcal{C}(i-1)$ and put $S \leftarrow S \setminus W$



Theorem: $\underline{\mathbb{E}[T_{uv}] \le O(\log n) \cdot d_{uv}}$

Proof: From Lemma: if LCA of u and v has level $i+1$ then $T_{uv} \le 2^{i+3}$

→ u and v are in different sets on level i ← this w. also settles (u,v)

$\Rightarrow \exists w \in V$ s.t. $|B(w, r_i) \cap \{u, v\}| = 1$ ← w cuts (u,v) on level i

Furthermore, we say that w settles (u,v) on level i if

w is the first vertex in the random permutation s.t. $B(w, r_i) \cap \{u, v\} \neq \emptyset$

↳ $\Pi(1), \Pi(2), \dots, \Pi(j) = w$

$X_{i,w}$:= event that w cuts (u,v) on level i

$S_{i,w}$:= event that w settles (u,v) on level i

→ we will find an upper bound $b_w \geq P[S_{iw} | X_{iw}]$ that depends only on w

→ let $\mathbb{1}(X)$ where X is a random variable be 1 if X occurs and 0 if not

↳ indicator for X

$$T_{uv} \leq \max_{i=0,\dots,\delta-1} \mathbb{1}[\exists w \in V : S_{iw} \,\&\, X_{iw}] \cdot 2^{i+3} \quad \leftarrow (*)$$

↳ w settles (u,v) as our alg would pick it

↳ if it cuts (u,v) it cannot be at the level of LCA or higher since (u,v) are in the same ball there

$$\leq \sum_{w \in V} \sum_{i=0}^{\delta-1} \mathbb{1}[S_{iw} \,\&\, X_{iw}] \cdot 2^{i+3}$$

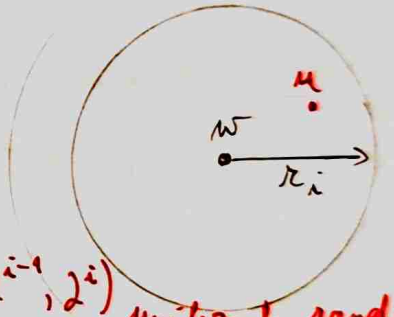
$$\Rightarrow \mathbb{E}[T_{uv}] \leq \sum_{w \in V} \sum_{i=0}^{\delta-1} P[S_{iw} \,\&\, X_{iw}] \cdot 2^{i+3}$$

$$= 8 \cdot \sum_{w \in V} \sum_{i=0}^{\delta-1} P[X_{iw}] \cdot P[S_{iw} | X_{iw}] \cdot 2^{i}$$

$$\leq 8 \cdot \underbrace{\sum_{w \in V} b_w}_{H_n} \cdot \underbrace{\sum_{i=0}^{\delta-1} P[X_{iw}] \cdot 2^{i}}_{\leq 2 \cdot d_{uw}} \qquad 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

$$\leq 16 \cdot H_n \cdot d_{uw} \leq O(\log n) \cdot d_{uw}$$

• Proving $\sum_{i=0}^{\delta-1} P[X_{iw}] \cdot 2^{i} \leq 2 \cdot d_{uw}$ for fixed $w \in V$



→ WLOG $d_{uw} \leq d_{vw}$

$$P[X_{iw}] = P[d_{uw} \leq r_i < d_{vw}] \qquad \text{use that } r_i \in [2^{i-1}, 2^i) \text{ uniformly randomly}$$

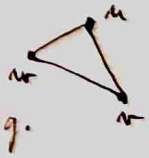
$$= \frac{\lambda([2^{i-1}, 2^i) \cap [d_{uw}, d_{vw}))}{\lambda([2^{i-1}, 2^i))} \qquad \lambda(I) = \text{length of the interval } I$$

$$= 2^{i-1}$$

$$\sum_{i=0}^{\delta-1} P[X_{iw}] 2^{i} = \sum_{i=0}^{\delta-1} 2 \cdot \lambda([2^{i-1}, 2^i) \cap [d_{uw}, d_{vw})) = 2 \cdot \lambda([d_{uw}, d_{vw})) = 2(d_{vw} - d_{uw})$$

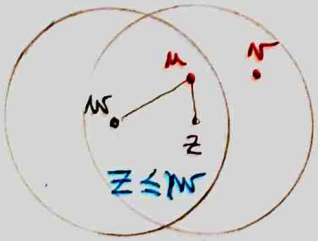
$$\leq 2 d_{uv}$$

↳ intervals $[2^{i-1}, 2^i)$ for $i = 0, \dots, \delta-1$ partition the entire interval $[1/2, \Delta/2)$



↳ Δ-ineq.

• Bound on $P[S_{iw} | X_{iw}]$



→ define an ordering $\leq$ on V by $z \leq w$ if

$$\min(d_{zu}, d_{zv}) \leq \min(d_{wu}, d_{wv})$$

if X_{iw} happens and $z \leq v$ then $B(z, r_i) \cap \{u, v\} \neq \emptyset$

⇒ if we want S_{iw} to also happen then w must come in the random permutation before all $z \leq w$

$V = \{w_1 \leq w_2 \leq \dots \leq w_n\}$ w_j = j-th closest to (u,v)

↳ if $w = w_j$ then this happens with prob $\leq \frac{1}{j}$

$$\Rightarrow P[S_{iw} | X_{iw}] \leq \frac{1}{j} =: b_w$$

$$\Rightarrow \sum_{w \in V} b_w = \sum_{j=1}^{n} b_{w_j} = \sum_{j=1}^{n} \frac{1}{j} = H_n$$

APPLICATION OF TREE METRIC – Buy-at-bulk network design

Multicommodity flow problem:

Input: $G=(V,E)$, length l_e for each $e \in E$

(s_1, t_1, D_1)
(s_2, t_2, D_2)
$\vdots$
(s_k, t_k, D_k)

→ demand

we want to send D_i units of commodity i from s_i to t_i

$\Rightarrow$ we need to buy capacities c_e for $e \in E$

we can send C units in total through an edge with capacity C

$f: \mathbb{R}_0^+ \to \mathbb{R}_0^+$ cost function for capacities

↳ $f(0)=0$ & $f(a+b) \le f(a)+f(b)$ ← f is subadditive

Output: Assignment of capacities c_e for each $e \in E$
and paths $P_1, \dots, P_k$ that will be used for routing

$\Rightarrow$ want: $\forall e \in E: \sum_{i: e \in P_i} D_i \le c_e$

Objective: $\min \sum_{e \in E} f(c_e) \cdot l_e$ ← we pay $f(C)$ for capacity C on 1 unit of length

Observation: This is easy for trees

1. $P_i \leftarrow$ the unique s_i–t_i path in T
2. $c_e \leftarrow \sum_{i: e \in P_i} D_i$

} clearly the optimum

Idea: Given G, let d_{uv} be the distance from u to v using edge lengths l_e

→ compute (V', T) approximating (V, d)

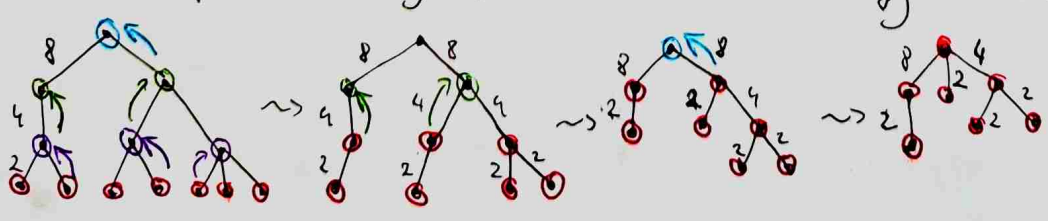
→ run the algorithm on T and then somehow translate the solution back to G

Problem: $V' \supseteq V$ ... how do we convert?

Theorem: For any tree metric (V', T) with $V \subseteq V'$ defined by a hierarchical cut decomposition (with the vertices of V as the leaves of T), we can find in poly-time another tree metric (V, T') s.t. $T_{uv} \le T'_{uv} \le 4 T_{uv}$.

Proof: Start with only the leaves of T being nodes of V

→ we now start contracting from the bottom-up until we are left with only $|V|$ nodes that we identify with V based on the contractions



To get T' we multiply all edge lengths by 4

- $T'_{uv} \le 4 T_{uv}$ since distances could have only decreased during contractions + final $\cdot 4$
- $T_{uv} \le T'_{uv}$ ... recall that if LCA of u and v is on level i then $T_{uv} = 2^{i+2} - 4$
  → since the contraction moves u and v upwards but it never identifies them, we have $T'_{uv} \ge 4 \cdot$ (length of an edge from LCA to its children) $= 4 \cdot 2^i = 2^{i+2}$ ▨

Corollary: Our randomized tree metric approx. alg can output these nice trees → $d_{uv} \le T'_{uv}$ & $E[T'_{uv}] \le O(\log n) d_{uv}$

Algorithm:
1. Given $G=(V,E)$, find a tree metric (V,T') that approximates (V,d)
2. $P'_i \leftarrow$ the unique s_i-t_i path in T'
3. for $\forall$ edge xy of T' in a path P'_i let $P_{xy} \leftarrow$ shortest x-y path in G
4. $P_i \leftarrow$ concatenation of paths P_{xy} for edges $xy \in P'_i$
5. $c_e \leftarrow \sum_{i : e \in P_i} D_i$

Theorem: This is an $O(\log n)$-approximation.

Notation:

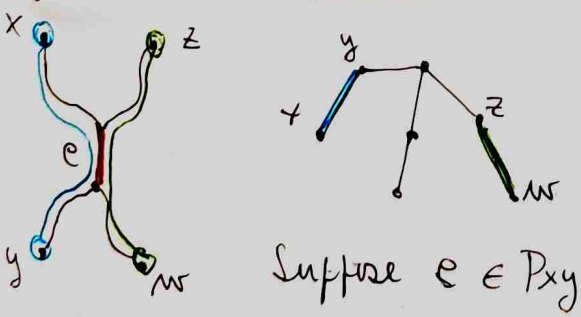
$xy \in T'$ ... edges of T'

P_{xy} .. edges of a fixed shortest x-y path in G
P'_{uv} ... edges of the unique u-v path in T'
$c'_{xy} := \sum_{i : xy \in P'_{s_i t_i}} D_i$ ... total capacity we demand from an edge $xy \in T'$

Lemma: $ALG = \sum_{e \in E} f(c_e) \cdot l_e \le \sum_{xy \in T'} f(c'_{xy}) \cdot T'_{xy}$

Proof:

graph G / tree T



Given $e \in E$, look at the edges $xy \in T$ such that $e \in P_{xy}$
→ we demand c'_{xy} from each such edge
⇒ in G we pay $d_{xy} \cdot f(c'_{xy}) \le T'_{xy} f(c'_{xy})$

$d_{xy} = \sum_{e \in P_{xy}} l_e$ = length of P_{xy}

Suppose $e \in P_{xy}, P_{zw}$

⇒ for e we pay $l_e \cdot f(c_e) = l_e f(c'_{xy} + c'_{zw}) \le l_e \left(f(c'_{xy}) + f(c'_{zw}) \right)$ ↓ sub-additivity

$$\sum_{xy \in T'} T'_{xy} f(c'_{xy}) \ge \sum_{xy \in T'} d_{xy} f(c'_{xy}) = \sum_{xy \in T'} f(c'_{xy}) \cdot \sum_{e \in P_{xy}} l_e = \sum_{e \in E} l_e \cdot \sum_{xy \in T' : e \in P_{xy}} f(c'_{xy})$$

$$\ge \sum_{e \in E} l_e f\left(\sum_{xy \in T' : e \in P_{xy}} c'_{xy} \right) = \sum_{e \in E} l_e f(c_e)$$

▨

→ we want to bound OPT. Suppose it uses

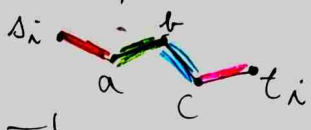
- P_i^* ... paths s_i–t_i in G
- $C_e^* = \sum_{i : e \in P_i^*} D_i$ ... capacities of edges

$$OPT = \sum_{e \in E} f(C_e^*) \cdot l_e$$

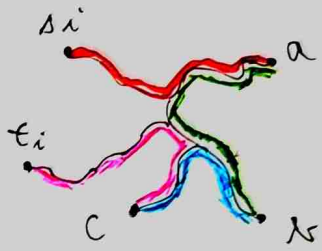
Translate the optimal solution for T' by defining capacities $\bar{C}_{xy}$ for $xy \in T'$.
For $\forall e = uv \in E$ install C_e^* units of capacity on $\forall xy \in P'_{uv}$

$$\bar{C}_{xy} = \sum_{uv \in E : xy \in P'_{uv}} C^*_{uv}$$

Graph G



Tree T'



this clearly is a feasible solution for the s_i–t_i problem on T'

$$\Rightarrow \text{cost} = \sum_{xy \in T'} f(\bar{C}_{xy}) \cdot T'_{xy} \geq \sum_{xy \in T'} f(C'_{xy})\, T'_{xy}$$

← since we solved T' optimally

By Lemma: $ALG \leq$ cost of optimal solution translated to T' ← $OPT_{T'}$

need $\in O(\log n) \cdot OPT$

claim: $OPT_{T'} \leq \sum_{uv \in E} f(C^*_{uv}) \cdot T'_{uv}$

$$\hookrightarrow \mathbb{E}[ALG] \leq \mathbb{E}[OPT_{T'}] \leq \sum_{uv \in E} f(C^*_{uv}) \cdot \mathbb{E}[T'_{uv}] \leq \sum_{uv \in E} f(C^*_{uv}) \cdot O(\log n) \cdot d_{uv}$$

$$= O(\log n) \cdot \sum_{uv \in E} f(C^*_{uv}) \cdot d_{uv} = O(\log n) \cdot OPT$$

Proof:

$$OPT_{T'} = \sum_{xy \in T'} f(\bar{C}_{xy}) \cdot T'_{xy} \leq \sum_{xy \in T'} T'_{xy} \sum_{uv \in E : xy \in P'_{uv}} f(C^*_{uv})$$

← subadditivity

($\sum C^*_e$ = $\sum$ over $uv \in E : xy \in P'_{uv}$)

$$= \sum_{uv \in E} f(C^*_{uv}) \cdot \underbrace{\sum_{xy \in P'_{uv}} T'_{xy}}_{T'_{uv}}$$

ONLINE MATCHING

Input: Bipartite graph $G=(L,R;E)$, where the vertices of R are known from the start and the vertices of L (together with their incident edges) are revealed in an online fashion

For simplicity: assume that $|L|=|R|=n$ & that G has a perfect matching

Output: Matching M of G

Objective: max $|M|$

↳ alg has to decide immediately and irrevocably if the new vertex l should be matched with some neighbour or not

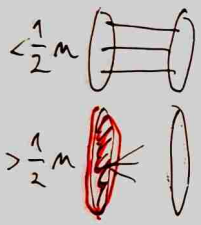
- Deterministic online algorithm

Alg: when l_i comes in, match it with its first (or arbitrary) free neighbour (if there is one)

Theorem: This is 2-competitive & no deterministic alg can be better.

Proof:  This finds an inclusion maximal matching

claim: any such matching must have size $\geq \frac{1}{2}n$



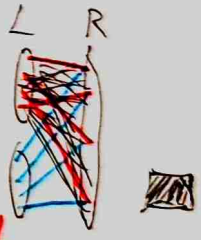
Suppose not. Since G has PM, by Hall's theorem the portside has $>\frac{1}{2}n$ neighbours

→ since #blocked vertices $<\frac{1}{2}n$, some neighbor must be free

$\Rightarrow$ we can add one more edge $\Rightarrow$ not $\subseteq$-maximal

Why cannot one do better?

- consider: first half of L is connected to every vertex of R
  second half to the matched vertices of ALG



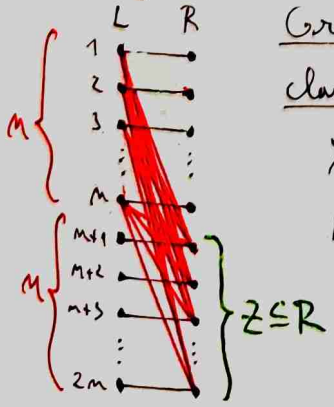
↳ make n large to kill additive constant

- Simple randomized alg.

Alg: match an incoming vertex with a uniformly randomly selected free neighbour

Theorem: This is also only 2-competitive.

Proof: The above argument that it is 2-competitive, we need a lower bound



Graph: $\forall i: (l_i, r_i) \in G$ & $\{l_1,\dots,l_n\}$ and $\{r_{n+1},\dots,r_{2n}\}$ are $K_{n,n}$

claim $\mathbb{E}[ALG \text{ on } G] \leq \frac{n}{2} + O(\log n) \Rightarrow \frac{ALG}{OPT} \longrightarrow \frac{1}{2}$ as $n \to \infty$

$X_i :=$ # free vertices in Z after i steps $\rightsquigarrow \mathbb{E}[ALG] = n + \mathbb{E}[X_n]$

$p_i :=$ probability that l_i was matched to Z

$p_1 = \frac{n}{n+1}$ $\quad \mathbb{E}[X_1] = n - \frac{n}{n+1}$, $\mathbb{E}[X_2] \leq n - \frac{n}{n+1} - \frac{n-1}{n}$

$p_2 \geq \frac{n-1}{n}$ $\quad \mathbb{E}[X_n] \leq n - \frac{n}{n+1} - \frac{n-1}{n} - \dots - \frac{2}{3} - \frac{1}{2}$

$p_j \geq \frac{n-(j-1)}{n+1-(j-1)}$ $\quad = \frac{1}{n+1} + \frac{1}{n} + \dots + \frac{1}{3} + \frac{1}{2} = H_{n+1} - 1 \in O(\log n)$

• The RANKING Algorithm

(for simplicity, but the same argument works for $|L| \neq |R|$)

Algorithm: In: (L, R, E) , $|L| = |R| = n$

1. rank the vertices of R as $r_{\pi(1)} \prec r_{\pi(2)} \prec \dots \prec r_{\pi(n)}$ using a random permutation π
2. Upon the arrival of a vertex $l_i \in L$, match l_i to the $\prec$-minimal unmatched neighbour $r \in N(l_i)$ of l_i

→ we no longer assume that there is a perfect matching !

Theorem: RANKING is $(1 - \frac{1}{e}) \approx 0.632$-competitive.

Analysis:

- imagine that $r_1, \dots, r_n$ are items in a store, and that $l_1, \dots, l_n$ are buyers coming in 1-by-1 to buy them
  ↳ each buyer can buy at most one from a select set of items
- for buyer l_i and item r_j define the value of r_j to l_i by

$$v_i(r_j) := \begin{cases} 1, & \text{if } r_j \in N(l_i) \\ 0, & \text{otherwise} \end{cases} \quad , \quad v_i(\emptyset) := 0$$

- for a matching $M \subseteq L \times (R \cup \{\emptyset\})$, notice that

$$|M| = \sum_{i \in [n]} v_i(M(i)) \quad , \text{ where } M(i) = \text{item chosen by } l_i \dots \text{possibly } \emptyset$$

- we will choose some costs c_j for items r_j. The utility of r_j to l_i is

$$u_i(r_j) := v_i(r_j) - c_j$$

Observation: RANKING is equivalent to the following market process:

- we choose a probability distribution D over $[0,1]$ ... if $X \sim D$ then $X \in [0,1]$ whose cumulative density function $F : x \mapsto P[X \leq x]$ is continuous
  ↳ so $\forall x : P[X = x] = 0$
- for each item r_j choose a random cost $c_j \sim D$
- when buyer l_i arrives, he buys an item maximizing his utility
  ↳ or nothing if there is nothing left

Why is this equivalent ?

→ since $c_j \in [0,1]$, the utility of $r_j \in N(l_i)$ is also random and

$$u_i(r_j) = 1 - c_j \in [0,1]$$

→ we can order the costs using a permutation $\pi : [n] \to [n]$ s.t.

$$c_{\pi(1)} \leq c_{\pi(2)} \leq \dots \leq c_{\pi(n)}$$

→ choosing max. utility item $\equiv$ choosing min. cost item

→ since $\forall x: P[X=x]=0$, all costs are different, so in fact

$$c_{\pi(1)} < c_{\pi(2)} < \cdots < c_{\pi(n)}$$

→ this clearly defines a (random) ranking of R ◼

We now define the distribution $\hat{D}$ we will be using

- to sample from $\hat{D}$, first uniformly randomly choose $w \in [0,1]$ and then return $e^{w-1} \in [0,1]$

Theorem: The market process where costs are sampled from $\hat{D}$ gives expected "social welfare" $= \sum_{i \in [n]} v_i(M(i)) = |M|$ at least $(1-\frac{1}{e})n$.

Proof: Put $UTIL_i := \begin{cases} 0, & \text{if buyer } l_i \text{ purchased nothing} \\ 1-c_j, & \text{if he purchased } r_j \end{cases}$ for buyer $l_i \in L$

Revenue → $REV_j := \begin{cases} 0, & \text{if nobody bought item } r_j \\ c_j, & \text{otherwise} \end{cases}$ for item $r_j \in R$

$$\sum_{l_i \in L} UTIL_i + \sum_{r_j \in R} REV_j = \sum_{(l_i, r_j) \in M} ((1-c_j) + c_j) = |M|$$

Lemma: For $\forall (l_i, r_j) \in \bar{E}: \mathbb{E}[UTIL_i + REV_j] \geq 1 - \frac{1}{e}$

→ if we have this, then given an optimal matching M^* and our matching M:

$$\mathbb{E}[|M|] = \mathbb{E}\left[\sum_{l_i \in L} UTIL_i + \sum_{r_j \in R} REV_j\right] \geq \mathbb{E}\left[\sum_{(l_i, r_j) \in M^*} (UTIL_i + REV_j)\right]$$

$$= \sum_{(l_i, r_j) \in M^*} \mathbb{E}[UTIL_i + REV_j] \geq \underline{\left(1-\frac{1}{e}\right)|M^*|} \quad \text{... what we want}$$

Proof of Lemma: Fix an edge $(l_i, r_j) \in E$

→ remove r_j from the input and let M' be the matching our market process finds

→ let $\bar{c} = e^{y-1}$ be the cost of the item matched to l_i in M'

↳ or, if l_i is not matched to anything, put $\bar{c} := 1 = e^{1-1}$

if $c_j < \bar{c}$, then r_j is sold ⇒ $UTIL_i \geq 1 - \bar{c}$

$$\mathbb{E}[REV_j] = \mathbb{E}[c_j \mathbb{1}[r_j \text{ is sold}]] \geq \mathbb{E}[c_j \cdot \mathbb{1}[c_j < \bar{c}]] = \int_0^y e^{x-1} dx$$

$$= e^{y-1} - e^{1} = \bar{c} - \frac{1}{e}$$

$$\Rightarrow \mathbb{E}[UTIL_i + REV_j] \geq \underline{1-\bar{c}} + \underline{\bar{c} - \frac{1}{e}} = 1 - \frac{1}{e}$$

◼

PROBABILISTICALLY CHECKABLE PROOFS

Recall: A decision problem is in NP if $\exists$ verifier V s.t. for every

- YES instance: $\exists$ short (polynomial) proof Π s.t. $V(\Pi) = 1$
- NO instance: $\forall$ short proof Π : $V(\Pi) = 0$ → evaluates in poly-time

Def: A decision problem is in $PCP_{c,s}[r(n), q(n)]$, where $0 \le s < c \le 1$ and $r(n)$ and $q(n)$ are functions of the (bit encoding) length of the input, if there exists a verifier V that given a (short) proof Π

- uses $r(n)$ random bits (flips n coins) to decide which $q(n)$ bits of Π to examine ⇝ he gets a shorter proof Σ
- he then selects a poly-time computable function $f: \{0,1\}^{q(n)} \to \{0,1\}$ and
  - accepts if $f(\Sigma) = 1$
  - rejects if $f(\Sigma) = 0$

Finally we require that for every

- YES instance: $\exists$ proof Π s.t. $P[V \text{ accepts } \Pi] \ge c$ ← completeness
- NO instance: $\forall$ proof Π : $P[V \text{ accepts}] < s$ ← soundness

$NP = PCP_{1,0}[0, poly(n)]$

Theorem (PCP Theorem): $NP = PCP_{1, 1/2}[O(\log n), O(1)]$

→ we use $O(\log n)$ random bits, so there are in total $2^{O(\log n)} = poly(n)$ combinations we can get

⇒ this is just enough to be able to access every bit of the proof

→ we then read constantly many bits of the proof and select one of constantly many functions to evaluate this short segment

Theorem: For $\forall \varepsilon, \delta > 0$: $NP = PCP_{1-\varepsilon, 1/2+\delta}[O(\log n), 3]$, where the verifier can only use the two following functions to evaluate the 3 bits:

- $even(x_1, x_2, x_3) = 1$ if $x_1 + x_2 + x_3$ is even
- $odd(x_1, x_2, x_3) = 1$ if $x_1 + x_2 + x_3$ is odd

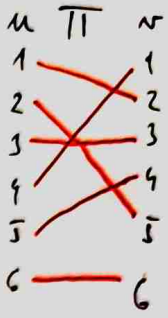
} and 0 otherwise

UNIQUE GAMES PROBLEM

↳ edge = ordered pair

Input: $G=(V,E)$, number of colors k, $\forall (u,v)\in E$ a permutation $\Pi_{uv}:[k]\to[k]$

↳ we cannot have both $(u,v)\in E$ and $(v,u)\in E$



Output: Coloring $C: V\to[k]$

Goal: max. # conditions (edges) satisfied

↳ $(u,v)\in E$ is satisfied if $\Pi_{uv}(C_u)=C_v$

Proposition: We can decide in poly-time whether or not can all conditions be satisfied.

Proof: Once we set C_u, it uniquely determines the color of all its neighbours, and this propagates through the entire connected component containing u

⇒ from each component select some vertex and try every color on it

↳ all conditions can be satisfied ⇔ $\forall$ component $\exists$ color that works ■

Fact: Given an instance in which $\geq (1-\varepsilon)$-fraction of conditions can be satisfied, there $\exists$ poly-time alg. that satisfies $\geq \left(1-O(\sqrt{\varepsilon\log k})\right)$-fraction of the conditions

UNIQUE GAMES CONJECTURE

UGC: $(\forall\varepsilon)(\exists k)$ s.t. it is NP-hard to distinguish between instances

① where $\geq (1-\varepsilon)$-fraction of the conditions can be satisfied

② where $\leq \varepsilon$-fraction —"—

Theorem: Assuming UGC, a $(2-\varepsilon)$-approx for Vertex Cover is NP-hard

↳ recall that a 2-approx is easy using LP

Theorem: Assuming UGC, the SDP algorithm for Max Cut is the best possible

↳ $(0.878+\varepsilon)$-approx is NP-hard

Def: The Linear UGC makes the same claim as UGC but only for instances, where all the permutations are powers of the cyclic permutation on $[k]$

↳ so $(\forall uv\in E)(\exists a<k)$ s.t. $\forall i\in[k]: \Pi_{uv}(i)=i+a \pmod k$

Def: The Bipartite UGC makes the claim only for instances where G is bipartite and all vertices in the same partite have the same degree.

Clearly BUGC ⇒ UGC and LUGC ⇒ UGC

Fact: BUGC ⇔ UGC ⇔ LUGC