PROTOCOL FOR AVG. SALARY

Friends: $F_1, F_2, \dots F_n$

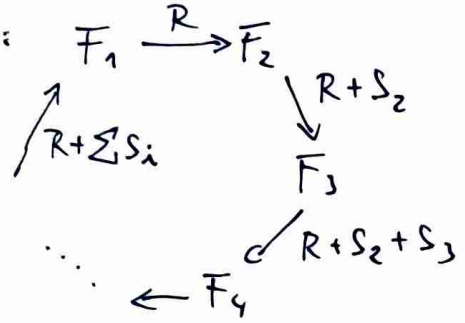
Salaries: $S_1, S_2, \dots S_n$ ← secret

Goal: $\sum S_i$ ↳ to abuse the protocol

Conditions: salaries are secret, if k friends cooperate, they can only find out the sum of salaries of the others

↳ there is no external authority

idea:



1. F_1 generates a random R
2. friends add S_i, one at a time
3. F_1 subtracts R from the total

problem: F_1 and F_3 can discover S_2 !

Solution: Assume $\forall S_i < B$, $N := n \cdot B \Rightarrow \sum S_i < N$

↳ $\forall F_i$ picks random numbers $R_{i1}, R_{i2}, \dots R_{in} \in [N-1]$ s.t.

$$\sum_{j=1}^{n} R_{ij} = S_i \mod N$$

F_1 makes $R_{11} + R_{12} + \dots + R_{1n} = S_1 \mod N$

F_2 makes $R_{21} + R_{22} + \dots + R_{2n} = S_2 \mod N$

F_n makes $R_{n1} + R_{n2} + \dots + R_{nn} = S_n \mod N$

$\oplus = \sum S_i \mod N$

F_1 F_2 F_n

→ $\forall F_i$ distributes the numbers ... gives R_{ij} to F_j ⇒ F_j has j-th column

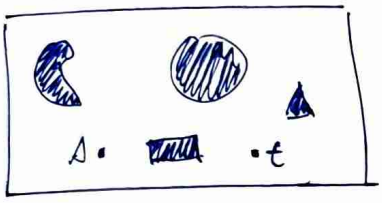
→ using: sum of columns = sum of rows

Optimization problems

Def: Optimization problem is $(\mathcal{I}, \mathcal{F}, f, g)$ where

- $\mathcal{I}$ - set of all valid instances of the problem
- $\mathcal{F}(I)$ = set of feasible solutions of $I \in \mathcal{I}$
- $f(I, S)$ = cost of solution $S \in \mathcal{F}(I)$
- g = min / max flag

Example:



rectangle with shapes = holes

goal: find shortest s-t path avoiding all holes

→ this is hard to encode

$\boxed{\text{E.g.}}$ is not NP-opt p.

Def: NP-optimization problem is an opt. problem s.t.

① instances and solutions are finite strings over a finite alphabet Σ

② the solution length is poly-bounded by the instance length

$$(\forall I \in \mathcal{I})(\forall S \in \mathcal{F}(I)): \; |S| \leq \text{poly}(|I|)$$

③ there is a poly-time decision procedure that checks for $\forall$ string $s \in \Sigma^*$ whether $s \in \mathcal{F}(I)$

④ $f(I, S)$ is poly-time computable

Ex: Shortest path in graph → start & target vertices

$\mathcal{I}$ = pairs (graph G, $(s, t) \in V_G \times V_G$)

$\mathcal{F}$ = for each graph G and (s, t), $\mathcal{F}(G, s, t)$ is the set of all s-t paths in G

f = total length of the given path

g = min

Def: For $I \in \mathcal{I}$ let $OPT(I) :=$ optimal value $= \min/\max_{S \in I} f(I,S)$

For an algorithm A let $A(I) =$ value of the solution constructed by A

Note: For randomized alg. we will be interested in $\mathbb{E}[A(I)]$

Def: Let $P = (\mathcal{I}, \mathcal{F}, f, g)$ be an opt. problem and A a poly-time (in $|I|$) algorithm which solves it. We say that

- approx. ratio of A is R
- A is an R-approx of P

$\equiv$ { min: $\forall I: A(I) \leq R \cdot OPT(I)$; max: $\forall I: A(I) \geq \frac{OPT(I)}{R}$ }

↑ $\mathbb{E}(A(I))$ for rand. A

TRAVELING SALESMAN PROBLEM

→ complete graph on vertices $1, 2, \dots n$
↳ if not, add edges of "infinite" lenght

→ non-negative, symmetric distance function $d: V \times V \to \mathbb{Z}^+$

- $\mathbb{Q}^+$ ... we can transform to $\mathbb{Z}^+$
- $\mathbb{R}^+$ ... problem: we want this to be a NP-opt p. → how to represent π as a finite string?

→ feasible solutions : permutations of vertices

→ goal : shortest tour $= \min \left(d(v_1, v_n) + \sum_{i=1}^{n-1} d(v_i, v_{i+1}) \right)$

Theorem: Let $\alpha(n)$ be a poly-time computable function (e.g. $n, n^2, 2^n, \dots$)
Unless $P = NP$, there is no $\alpha(n)$-approx alg for general TSP.

Proof: We want to reduce the Hamiltonian cycle decision problem to TSP
↳ a reduction needs to be polynomial, that is why $\alpha(n)$ has to be poly

G ... general graph ⇒ add weights to $\{u,v\} \in \binom{V}{2}$

- $uv \in E \Rightarrow w(uv) = 1$
- $uv \notin E \Rightarrow w(uv) = n \cdot \alpha(n)$

} instance of TSP → I

G has a H.C. $\Leftrightarrow$ $OPT = n$
G has no H.C. $\Leftrightarrow$ $OPT > n \cdot \alpha(n)$ ... need a non-edge

→ after running the algorithm A on I

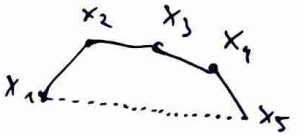
G has a H.C. $\Leftrightarrow$ $A(I) \leq n \cdot \alpha(n)$
G has no H.C. $\Leftrightarrow$ $A(I) > OPT > n \cdot \alpha(n)$

} we just solved H.C. in poly-time

TSP approximations

Def: $f: V \times V \to \mathbb{R}^+_0$ is a **metric** $\equiv$

1) $f(u,v) = f(v,u)$ ... symmetry
2) $f(u,w) \le f(u,v) + f(v,w)$ ... Δ-ineq
3) $f(u,v) = 0 \iff u = v$ ... positivity



general Δ-ineq: $f(x_1, x_k) \le f(x_1,x_2) + f(x_2,x_3) + \dots + f(x_{k-1}, x_k)$

Notation: The **shortcutting** of a sequence $u_1, \dots u_k \subset V$, where possibly $u_i = u_j$ is the sequence where we keep only the first occurence of each u_i.

TSP – MST algorithm : 2-approx

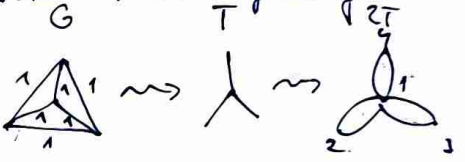
Assume — distance d is a metric
G is complete, otherwise add edges with $d(e) = +\infty$

Def: A tour of G is **eulerian** $\equiv$ it visits each edge once & is closed.

↳ **Fact:** E. tour exists $\iff$ all degrees of G are even.

Algorithm

1. $T \leftarrow$ **minimal spanning tree** of G
2. find an eulerian tour in "2T"
3. return the shortcutting of the E. tour



$1,2,1,3,1,4,1 \rightsquigarrow 1,2,3,4$

Correctness:

- "2T" has all deg. even $\Rightarrow$ E. tour exists
- E. tour visits all edges $\Rightarrow$ all vertices $\Rightarrow$ shortcutting is a H. cycle

Complexity: all easy poly-time

Cost: $\text{cost}(T) \le OPT$ ... because we can get a spanning tree by taking the optimal H.C. and removing 1 edge
↳ and T is the **min.** spanning tree

$\text{cost}(\text{shortcutting}) \le \text{cost}(\text{E. tour})$ ... from the general Δ-ineq

$\| $

$2 \cdot \text{cost}(T) \le 2 \cdot OPT \quad \Rightarrow$ **2-approx.** ∎

How to improve?

step 2.: we are making T even by adding an entire copy of T
$\Rightarrow$ we want to add less edges so that we still remove all odd edges
#vertices with odd deg is even
$\Rightarrow$ we will find a perfect matching

Christofides algorithm: 1.5 - approx

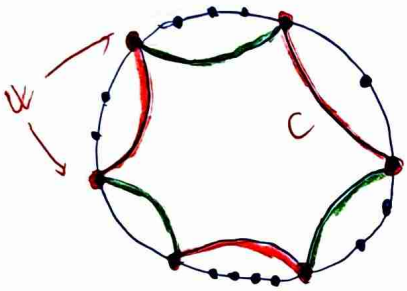
1. $T \leftarrow$ min. spanning tree of G
2. $W \leftarrow$ vertices with odd deg in T
3. $M \leftarrow$ min cost perfect matching M on W ... recall: G is complete $\Rightarrow$ exists
4. $T' \leftarrow T + M$ has all degrees even
5. find an Eulerian tour in T'
6. return a shortcutting of the tour

Correctness: ✓ same as last time

Complexity: polytime

Cost: Let H be the optimal H.C., denote the cost of $G' \subseteq G$ by $\ell(G')$

- $\ell(T) \leq \ell(H)$ ∵ T is a min. spanning tree and $H - e$ is also a spanning tree ($\subseteq H$)
- Now order the vertices of G in the order they are visited by H



→ create C = cycle on W following the order of H

$\ell(C) \leq \ell(H)$ using Δ-ineq

$|W|$ is even $\Rightarrow$ $|C|$ is even

→ C defines 2 perfect matchings of W: M_1, M_2

→ since $\ell(M_1) + \ell(M_2) = \ell(C) \leq \ell(H)$, at least one M_i has $\ell(M_i) \leq \frac{1}{2}\ell(H)$

$\Rightarrow \ell(M) \leq \frac{1}{2}\ell(H)$ ∵ M is min. PM of W

$\Rightarrow \ell(T') = \ell(T) + \ell(M) \leq \ell(H) + \frac{1}{2}\ell(H) = \frac{3}{2}\ell(H)$

→ and shortcutting cannot make it longer $\Rightarrow$ $\frac{3}{2}$-approx ∎

Remark:

This algorithm is 50 years old

→ an improvement is from 2020 and it is a $\left(\frac{3}{2} - 10^{-36}\right)$-approx !

$\Rightarrow$ 1.5 is very good

Discrete probability recap

Def: **Discrete probability space** (Ω, P)

- Ω ... finite or countable set of elementary events
- $P : \Omega \to [0,1]$ s.t. $\sum_{w \in \Omega} P[w] = 1$

Examples:

1. coin toss : $\Omega = \{H, T\}$, $P[H] = P[T] = \frac{1}{2}$
2. fair die : $\Omega = \{1,2,3,4,5,6\}$, $P[*] = \frac{1}{6}$
3. n tosses : $\Omega = \{H,T\}^n$, $P[*] = \frac{1}{2^n}$
4. random $k \in [n]$: $\Omega = \{1,2,\dots n\}$, $P[*] = \frac{1}{n}$

} **uniform prob. space** for a finite p.s. is $P[*] = \frac{1}{|\Omega|}$

5. random $k \in \mathbb{N}$: $\Omega = \mathbb{N}$, $P[k] = \frac{1}{2^k}$ } **countable p.s.**

Def: **Random variable** on (Ω, P) is a function $X : \Omega \to \mathbb{R}$.

Ex:

1. n tosses : X = # of heads
2. die toss : # of dots on the face

Def: The **expectation** of a r.v. X is $\mathbb{E}[X] = \sum_{w \in \Omega} X(w) \cdot P[w]$

Def: An **event** is a subset $A \subseteq \Omega$. It has probability $P[A] = \sum_{w \in A} P[w]$

Def: **Indicator variable** of an event $A \subseteq \Omega$ is $I_A(w) = \begin{cases} 0, & w \notin A \\ 1, & w \in A \end{cases}$, $\mathbb{E}[I_A] = P[A]$

Theorem: $\mathbb{E}[\alpha X + Y] = \alpha \mathbb{E}[X] + \mathbb{E}[Y]$

Def: The **conditional prob.** of A assuming B is $P[A|B] := \frac{P[A \cap B]}{P[B]}$

Note: The function $P_B : \Omega \to [0,1]$, $A \mapsto P[A|B]$ is also a prob. measure on Ω

- $w \notin B$: $P[w|B] = 0$
- $w \in B$: $P[w|B] = \frac{P[w]}{P[B]}$

} $\sum_{w \in \Omega} P[w|B] = 1$

Theorem: $B_1 \dot\cup B_2 \dot\cup \dots \dot\cup B_n = \Omega \Rightarrow$ $P[A] = \sum_i P[B_i] \cdot P[A|B_i]$ ← TOTAL PROBABILITY

Def: Events A, B are **independent** $A \perp B \equiv P[A \cap B] = P[A] \cdot P[B]$

Ex: Suppose we have a biased coin with $P[\text{Heads}] = p$. Simulate a fair coin

1. toss the coin twice
2. if HT ⇝ simulate H
   TH ⇝ simulate T
   ... $P[HT] = P[TH] = p(1-p)$
3. if HH or TT reroll

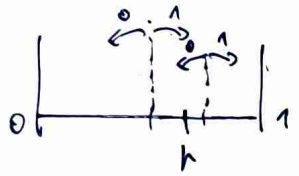
$$\mathbb{E}[\text{tries until success}] = \frac{1}{P[\text{success}]} = \frac{1}{2p(1-p)}$$

Ex: Suppose we want to simulate a biased coin with $P[H] = p$ using a fair coin.

→ need: event with probability p

⇒ generating a random number $r \in [0,1]$ ⇒ event := $r \le p$

→ H = 0, T = 1, we stop when we are sure if $r \le p$ or $r > p$



→ we can simply compare bits

$p = 0.0110\ldots$
$r = 0.0110\ldots$ generate bits of r until they don't match

→ if $p = 0.01101$, $r = 0.01100$ ⇒ $r < p$

$$\mathbb{E}[\text{flips until success}] = \frac{1}{P[\text{success}]} = \frac{1}{P[\text{the bit we generate is different}]} = \frac{1}{1/2} = 2$$

RANDOMIZED QUICKSORT

IN: set S of distinct comp. elements

1. if $|S| = 1$: return S
2. choose uniformly randomly $p \in S$
3. split S: ← pivot
   $S^- \leftarrow \{a \in S \mid a < p\}$
   $S^+ \leftarrow \{a \in S \mid a > p\}$
4. return Quicksort(S^-), p, Quicksort(S^+)

worst case: $\Omega(n^2)$

claim: average case : $n \log n$

Notes:

if S includes duplicates, the algorithm works too, we just need to modify steps 3 and 4

$S^= \leftarrow \{a \in S \mid a = p\}$ ← multiset

return $Q(S^-)$, $S^=$, $Q(S^+)$

→ fix an input $x_1, \ldots x_n$ and let $y_1 < y_2 < \cdots < y_n$ be the sorted sequence

for $i \neq j$: y_i and y_j are compared at most once (one has to be the pivot)

⇒ define random var. X = # total comparisons, and indicators

$$X_{ij} := \begin{cases} 1, & \text{if } y_i \text{ and } y_j \text{ were compared} \\ 0, & \text{else} \end{cases}$$

$\mathbb{E}X = \sum_{i<j} \mathbb{E}X_{ij}$ = expected run-time

↳ $\mathbb{E}X_{ij} = P[\underbrace{y_i \text{ and } y_j \text{ have been compared}}_{C_{ij}}] = ?$

→ let B_ℓ = the event that y_i and y_j were separated at recursion level ℓ

$B_\ell \cap B_{\ell'} = \emptyset$ for $\ell \neq \ell'$ ... they can't be separated twice

$\bigcup_\ell B_\ell = \Omega$ ... they have to be separated eventually

⇒ law of total probability: $P[C_{ij}] = \sum_\ell P[B_\ell] \cdot P[C_{ij} | B_\ell]$

↳ assume that B_ℓ occured, we want $P[C_{ij} | B_\ell]$

$P[C_{ij} | B_\ell] = \frac{2}{j-i+1}$ $\quad y_1 < \dots < \underbrace{y_i < \dots < y_j}_{j-i+1} < \dots y_n$

↳ for separation of y_i, y_j, one of $y_i, y_{i+1}, \dots y_j$ has to be the pivot

→ for the comparison to occur, the pivot has to be y_i or y_j

$P[C_{ij} | B_\ell]$ doesn't depend on ℓ

$$\Rightarrow P[C_{ij}] = \sum_\ell P[B_\ell] \cdot \frac{2}{j-i+1} = \frac{2}{j-i+1} \cdot \underbrace{\sum_\ell P[B_\ell]}_{1} = \frac{2}{j-i+1}$$

$$\Rightarrow \mathbb{E}[X] = \sum_{i<j} P[C_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \left(\frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n-i+1} \right)$$

$$\leq 2 \sum_{i} \left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right) = \underline{2n H_n}$$

HARMONIC NUMBER

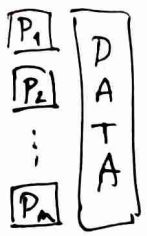
⇓

Theorem: The expected number of comparisons is $\leq 2n H_n \sim 2n \cdot \ln(n)$.

• Algorithm Types

- **Las Vegas** – always produces correct solution
  – runtime varies based on random choices
- **Monte Carlo** – runtime is fixed or bounded
  – might return an incorrect solution (with some small probability)

CONTENTION RESOLUTION IN A DISTRIBUTED SYSTEM



- n processes $P_1, \dots P_n$ operate with a single shared database
- processes operate synchronously in discrete rounds
- no communication between $P_1, \dots P_n$

→ they need to access the database, but if multiple processes attempt to do so in the same round, they all fail

GOAL: Protocol ensuring that all processes access the database "often enough"

Fact: for $n \geq 2$: $\frac{1}{4} \leq \left(1-\frac{1}{n}\right)^n \leq \frac{1}{e} \leq \left(1-\frac{1}{n}\right)^{n-1} \leq \frac{1}{2}$ (when $n \to \infty$, both $\to \frac{1}{e}$)

Protocol: Every round $\forall P_i$ will with prob. $S = \frac{1}{n}$ attempt to access the database.

→ events $A_{i,t}$:= process P_i succeeds in round t.

$$P[A_{it}] = S \cdot (1-S)^{n-1} \quad \dots P_i \text{ succes} \ \& \ n-1 \text{ fails}$$
$$= \frac{1}{n}\left(1-\frac{1}{n}\right)^{n-1} \geq \frac{1}{en}$$

$0.4 \geq 0.3 \rightarrow 1-0.4 \leq 1-0.3$

→ events $F_{i,t}$:= P_i doesn't succeed in any of the first t rounds

$$P[F_{it}] = (1-P[A_{it}])^t \leq \left(1-\frac{1}{en}\right)^t = \left[\left(1-\frac{1}{en}\right)^{en}\right]^{\frac{1}{en}t} \leq \left(\frac{1}{e}\right)^{\frac{1}{en}t} = e^{-\frac{1}{en}t}$$

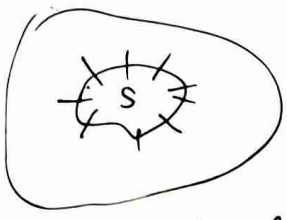
↳ for $t = k \cdot en \ln(n)$ we have

$$P[F_{it}] \leq e^{-k \ln(n)} = \frac{1}{n^k}$$

$$\Rightarrow P[\exists \text{process that has not succeeded in } t = k \cdot en \ln(n) \text{ rounds}] \leq n \cdot \frac{1}{n^k} \leq \frac{1}{n^{k-1}}$$

Theorem: After $t = k \cdot en \ln(n)$ rounds, all processes succeed with probability $\geq 1 - \frac{1}{n^{k-1}}$. $k=2$: $1-\frac{1}{n}$, $k=3$: $1-\frac{1}{n^2}$

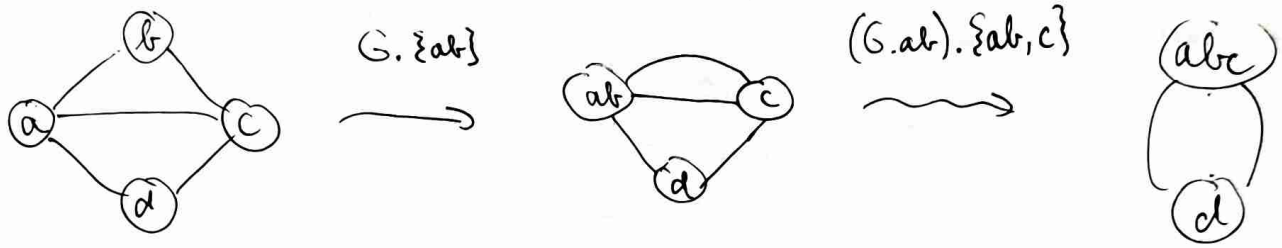
GLOBAL MINIMAL CUT



IN: $G=(V,E)$ connected
OUT: $S \subseteq V$ s.t. $\emptyset \neq S \neq V$
GOAL: min cut size = min # edges between S and $V \setminus S$

- **Explicit solution** using flows – recall: min cut ~ max flow
  - ⇒ fix $t \in V$, and for $\forall s \in V \setminus \{t\}$ find max s-t flow ⇝ min cut
  - the best min-cut is the global min cut
  - ⇒ complexity $n \cdot \mathcal{O}(n^2 m)$ ← Goldbach/Goldberg/Dinitz
- **Monte Carlo solution**
  - ⇒ our algorithm will run in $\log(n) \cdot n^2 \cdot m$ and find min cut with % $\geq 1 - \frac{1}{n}$.

Notation: edge contraction is denoted as $G.e$, we work with MULTIGRAPHS



1. $\forall$ cut in $G.e$ corresponds to a cut in G of the same size
   ⇒ $|\text{mincut}(G)| \leq |\text{mincut}(G.e)|$ for $\forall e$
2. if C is a cut in G and $e \notin C$, then there is a cut of size $|C|$ in $G.e$
   ⇒ $|\text{mincut}(G)| = |\text{mincut}(G.e)|$

Alg: randomMincut

1. while $|V(G)| > 2$:
2. $\quad e \leftarrow$ uniformly randomly choose $e \in E(G)$
3. $\quad G \leftarrow G.e$
4. return the cut corresponding to the two vertices

Analysis:

- ⇒ fix a mincut C, let $k = |C|$ ... implies $\text{mindeg} \geq k$
- if the alg never chooses any $e \in C$, then it outputs C
- $|E| \geq \frac{k \cdot n}{2}$ ... as $\text{mindeg} \geq k$
  - ⇒ $P[e \in C \text{ is chosen in iteration } 1] = \frac{|C|}{|E|} \leq \frac{k}{\frac{kn}{2}} = \frac{2}{n}$ ($n_i = n - i + 1$)
- ⇒ there are $n-2$ iterations ... let n_i denote $|V(G)|$ at the start of iter i
- assuming no $e \in C$ contracted in iter $1, 2, \ldots i-1$: $P[e \in C \text{ chosen in iter } i] \leq \frac{2}{n_i}$

→ let E_i be the event that no $e \in C$ was chosen in iter i

$$P[E_1] \geq 1 - \frac{2}{n} = \frac{n-2}{n}$$
$$P[E_i \mid E_1 \cap E_2 \cap \dots \cap E_{i-1}] \geq 1 - \frac{2}{n_i} = \frac{n_i - 2}{n_i} = \frac{n-i-1}{n-i+1}$$

if all E_i occur, the alg returns a global mincut (C) → what is the %?

$$P[E_1 \cap E_2 \cap \dots \cap E_{n-2}] = P[E_{n-2} \mid E_1 \cap \dots \cap E_{n-3}] \cdot P[E_1 \cap \dots \cap E_{n-3}]$$
$$\vdots \qquad \text{↳ now decompose this}$$
$$= P[E_{n-2} \mid E_1, \dots, E_{n-3}] \cdot P[E_{n-3} \mid E_1, \dots, E_{n-4}] \cdot \dots \cdot P[E_3 \mid E_1, E_2] \cdot P[E_2 \mid E_1] \cdot P[E_1]$$
$$\geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{n-5}{n-3} \cdot \dots \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2 \cdot 1}{n(n-1)} = \frac{2}{n(n-1)}$$

⇒ the alg finds the global mincut with probability $\geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}}$

⇒ failure % $\leq 1 - \frac{1}{\binom{n}{2}}$

⇒ failure % after k reruns $\leq \left(1 - \frac{1}{\binom{n}{2}}\right)^k$

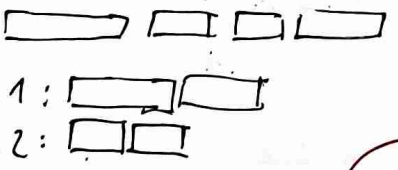
- $k = \binom{n}{2}$: $\leq \left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2}} \leq \frac{1}{e}$
- $k = \binom{n}{2} \ln(n)$: $\leq \left(\frac{1}{e}\right)^{\ln(n)} = \frac{1}{n}$

Theorem: If we repeat the alg. $\binom{n}{2} \ln(n)$ times, we get global mincut with probability $\geq 1 - \frac{1}{n}$. Since 1 iter $\in \Theta(m)$, we have $T(n) \in \Theta(n^2 m \ln(n))$

SCHEDULING WITH GREEDY ALGORITHMS

→ scheduling is the following problem: n jobs with processing times $P_1, \dots P_n$ and m identically fast machines to run them on

Example.

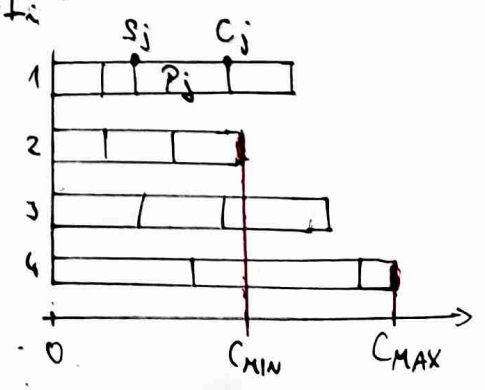


SCHEDULE

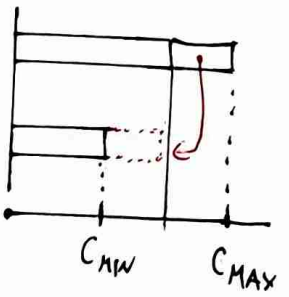
Goal: find a partition $I_1, \dots I_m$ of $\{1, \dots n\}$ s.t. $\max_i \sum_{j \in I_i} P_j$ is minimal

Notation: for a given schedule and a job $j \in I_i$

- startup time of P_j: $S_j = \sum_{k \in I_i, k < j} P_k$
- completion time of P_j: $C_j = S_j + P_j$
- $C_{MAX} = \max_j C_j$
- $C_{MIN} = \min_i \max_{j \in I_i} C_j$



• Local search algorithm



1. start with any schedule
2. while there $\exists$ job j s.t. $C_j = C_{MAX}$ & $S_j > C_{MIN}$
3. reassign it to a machine with min. completin time
4. return the shortened schedule

👁 runs fast: each job is reassigned at most once ∵ C_{MIN} never decreases

→ bound the optimum:

$OPT \geq \frac{\sum P_j}{m}$ ... average ideal job distribution

$OPT \geq P_j$ for $\forall j$

→ consider the last job to finish: job k ... $C_{MAX} = C_k = S_k + P_k$

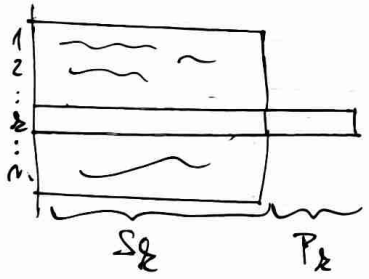
$S_k \leq C_{MIN}$ ... if not, we could improve the schedule

↳ all machines are still busy when P_k starts

$C_{MIN} \leq \frac{\sum P_j}{m}$ ... the first machine to end will be faster than the average

$\Rightarrow C_{MAX} = S_k + P_k \leq \frac{\sum P_j}{m} + P_k \leq OPT + OPT = 2OPT \quad \Rightarrow$ 2-approx

→ we can find a better bound



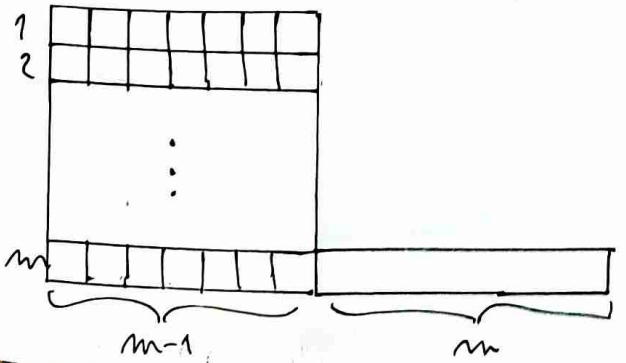
$$S_k \leq \frac{\sum P_j - P_k}{m}$$

$$\Rightarrow C_{MAX} \leq \frac{\sum P_j}{m} - \frac{P_k}{m} + P_k \leq OPT + \left(1 - \frac{1}{m}\right) OPT = \left(2 - \frac{1}{m}\right) OPT$$

Theorem: The local search alg. is a $(2 - \frac{1}{m})$-approximation.

? is $(2 - \frac{1}{m})$ tight for this alg ? YES !

↳ consider: $m \cdot (m-1)$ jobs of size 1 and one job of size m



→ the alg will not improve this

$$\left.\begin{array}{l} \Rightarrow OPT = m \\ ALG = 2m - 1 \end{array}\right\} \quad \frac{2m-1}{m} = 2 - \frac{1}{m}$$

- **List scheduling alg.** (greedy)
  1. order the jobs arbitrarily
  2. process the jobs according to the order and assign the new job to the least heavily loaded machine

Theorem: This algorithm is a $(2-\frac{1}{m})$-approximation

Pf: It is at least as good as the previous one (that one would find no improvement)
→ and $(2-\frac{1}{m})$ is tight - the same counterexample as last time works

ONLINE × OFFLINE ALGS

Def: Online algorithms are algorithms, where the input is revealed in a step-by-step manner, and the next decision has to be made without knowing the future (= the rest of the input)

Online algorithms have a big disadvantage to offline algorithms, but are often needed in real-world scenarios

! The online List scheduling alg is as good as the offline Local search alg

Def: Competitive ratio is how we call the approx. ratio in the context of online algs.

Theorem: The greedy list scheduling alg has competitive ratio $(2-\frac{1}{m})$.

- **Largest processing time first** (LPT)
  1. order the jobs by their processing times $P_1 \geq P_2 \geq \dots \geq P_n$
  2. greedily assign the next job to the least heavily loaded machine

Lemma: WLOG we can assume that the job P_n is completed last.

Pf: Let P_i be the last finished job.
→ instance $I := P_1, P_2, \dots P_n$, instance $I' := P_1, P_2, \dots P_i$

$OPT(I') \leq OPT(I)$
$ALG(I') = ALG(I)$
$\Rightarrow \frac{ALG(I)}{OPT(I)} \leq \frac{ALG(I')}{OPT(I')}$

⇒ the approx ratio of the modified problem is a bound of the original one ▨

→ distinguish 2 cases: remember P_n is the shortest job & completes last

1) $P_n \leq \frac{OPT}{3}$: $C_{MAX} = C_n = S_n + p_n \leq \frac{4}{3} \cdot OPT$

↳ $S_n \leq OPT$ ... see local search alg. analysis

2) $P_n > \frac{OPT}{3}$: ... all jobs $> \frac{OPT}{3}$

👁 in the OPT schedule, there are at most 2 jobs on any machine

→ if $n \leq m$, there is only 1 job on each machine ⇒ ALG = OPT

⇒ WLOG $m < n \leq 2m$ ↳ $P_1, \dots P_m$

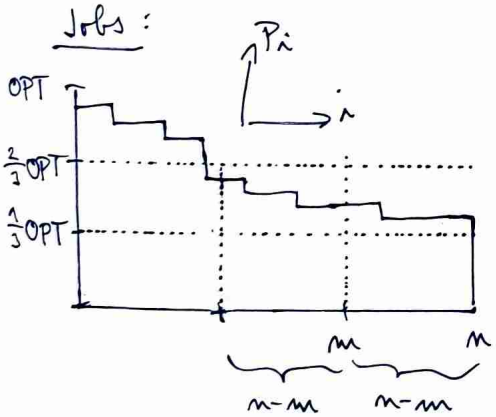
👁 OPT puts the m longest jobs to m distinct machines

- if $n \geq m+1$: OPT has to put P_{m+1} to a busy machine $P_1 \geq P_2 \geq \dots \geq P_m$
  → OPT puts it to P_2, so $OPT \geq P_2 + P_{m+1} \geq P_m + P_{m+1}$
- if $n \geq m+2$: OPT can again put P_{m+2} locally to P_m, but then the longer $P_{m+1} \geq P_{m+2}$ would be paired with the longer $P_{m-1} \geq P_m$
  since $OPT \geq P_{m-1} + P_{m+1}$ and $P_{m+1} \geq P_{m+2}$ we have $OPT \geq P_{m-1} + P_{m+2}$

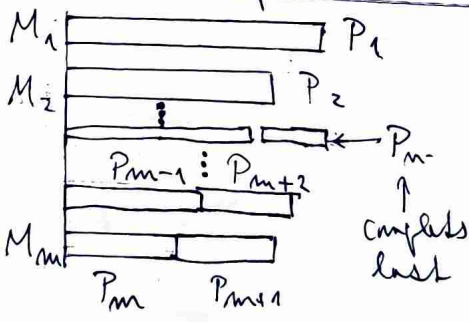
→ in general, its not hard to see that $OPT \geq P_{m-j+1} + P_{m+j}$

⇒ since $P_{m+j} \geq \frac{OPT}{3}$, we have $P_{m-j+1} \leq \frac{2}{3} OPT$

Jobs:



How LPT-first schedules them



Notice that the alg matches the jobs in the same way as we bounded OPT

$P_{m-j+1} + P_{m+j} \leq OPT$

⇒ the alg finds an optimal solution

Theorem: The approx ratio of LPT alg is $\frac{4}{3}$. → it can be made a bit tighter

BIN PACKING PROBLEM

IN: n items $a_1, \dots a_n \in [0,1]$

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

Goal: minimize m = # bins

Scheduling intuition: We have a lot of machines and we need to finish all jobs before a deadline – how many machines do we need?

Greedy approaches:

→ in every case, if a_j doesn't fit into any bin, create a new one if it fits somewhere – how do we pick where to place it?

- first fit – place a_j to the first bin it fits into
- best fit – place it into the fullest fit possible
- any fit – place it into an arbitrary bin it fits to

👁 if we upper bound any-fit, we also upper bound the other two

Theorem: Every any-fit algorithm has approx ratio ≤ 2.

Proof: Suppose the alg constructed $I_1, I_2, \dots I_m$, let $B_\ell := \sum_{i \in I_\ell} a_i$

👁 $B_1 + B_2 > 1$ ... otherwise the alg wouldn't create B_2

$B_\ell + B_{\ell+1} > 1 \quad \forall \ell = 1, \dots m-1$

$B_1 + B_m > 1$ ← sum the inequalities

$2 \sum B_\ell > m$

$2 \sum a_i$

$\Rightarrow m < 2 \sum a_i$

👁 $OPT \geq \sum a_i$

$m < 2OPT \Rightarrow$ 2-approx ▨

Proposition: A $(\frac{1}{2} - \varepsilon)$-approx of bin-packing is impossible unless $P = NP$.

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. Notice that the instance can be packed into 2 bins $\Leftrightarrow$ the Partition instance has a perfect split... otherwise we need ≥ 3 bins.

$\Rightarrow$ So an alg. with approx. ratio $\rho < \frac{3}{2}$ would always output 2 when $OPT = 2$ $\left(\frac{ALG}{OPT} \leq \rho < \frac{3}{2}\right)$

$\Rightarrow$ we would decide Partition in poly-time ▨

Fact: The approx. ratio of both first-fit and best-fit is 1.7 (tight).

↳ but best-fit behaves better on real instances.

EDGE DISJOINT PATHS PROBLEM (EDP)

IN: $G=(V,E)$ and k pairs of vertices $(s_1,t_1),\dots(s_k,t_k)$

OUT: $I \subseteq \{1,\dots,k\}$ together with a path P_i for $\forall i \in I$ s.t. P_i connects $s_i - t_i$ and all P_j are edge disjoint

Goal: max $|I|$.

Alg (greedy): ... → we always remove the shortest possible path from the graph

1. $I \leftarrow \emptyset$
2. for each $i \in [k] \setminus I$, let P_i denote the shortest $s_i - t_i$ path in G
3. select the shortest P_j, if there is none: goto 5)
4. $I \leftarrow I \cup \{j\}$, $G \leftarrow G - P_j$, goto 2) ← remove used edges & repeat
5. return I and the paths

Theorem: The greedy alg. has approx. ratio $(2\sqrt{m}+1)$, $m = |E(G)|$.

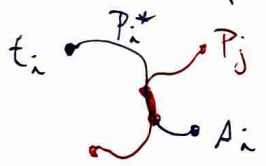
Proof: Consider an optimal solution OPT and denote by P_i^* the $s_i - t_i$ path in OPT.

$OPT_\ell := \{ i \in OPT \mid |P_i^*| > \sqrt{m} \}$ ← long paths

$OPT_s := OPT \setminus OPT_\ell$ ← short paths

👁 $|OPT_\ell| \le \sqrt{m}$ ... otherwise $|E| > \sqrt{m}\cdot\sqrt{m}$ ↯

Consider P_i^* for $i \in OPT_s \setminus I$ ... short paths used by OPT, but not by ALG.



ALG didn't use P_i^*, because an edge was already used by some P_j s.t. $|P_j| \le |P_i^*|$ ... because ALG is greedy

↳ we say that P_j blocks the short path P_i^*

👁 every greedy path P_j, $j \in I$, blocks at most $\sqrt{m}$ short paths from OPT

↳ P_j blocks at most $|P_j|$ paths, and $|P_j| \le |P_i^*| \le \sqrt{m}$

Since every path in $OPT_s \setminus I$ is blocked by a greedy path we have

$|OPT_s \setminus I| \le \sqrt{m} \cdot |I|$ ← max. num paths which could be blocked

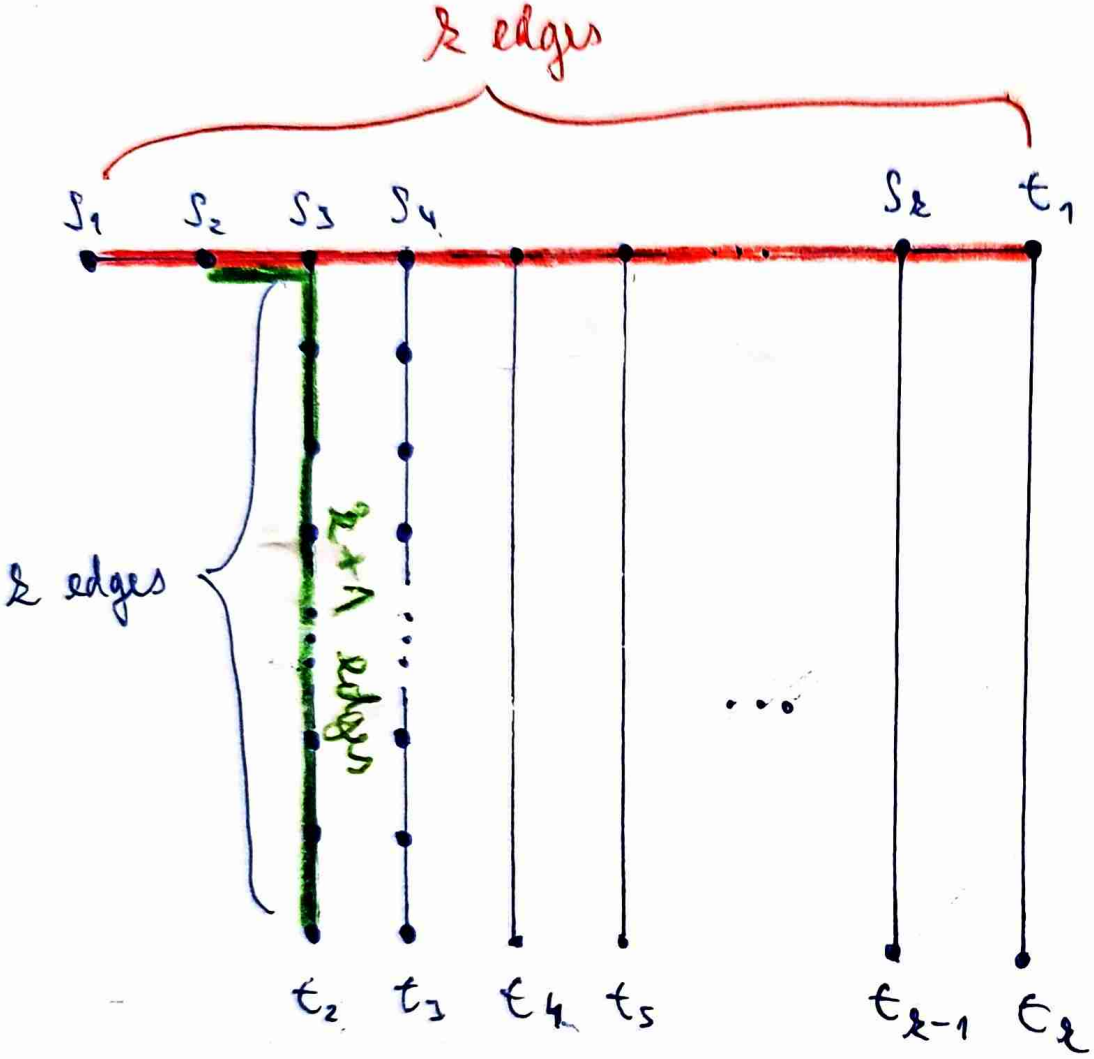
$\Rightarrow |OPT| = |OPT_\ell| + |OPT_s| \le \sqrt{m} + |OPT_s \setminus I| + |I| \le \sqrt{m} + \sqrt{m}\cdot|I| + |I|$ ($1 \le |I|$)

$\le (2\sqrt{m}+1)\cdot|I|$ ∎

! The $\sqrt{m}$ estimate is tight - see picture

Fact: For directed graphs: $\forall \varepsilon > 0$: $m^{\frac{1}{2}-\varepsilon}$ approximation is NP hard. ($k \ge 2$).

↳ we know better approx. algs for undirected, but they are still polynomial



GREEDY: 1 path of len = k

OPT: $k-1$ paths of len = $k+1$

$$n = k + (k-1) \cdot k = k^2$$

$$\text{ratio} = \frac{k-1}{1} = \sqrt{n} - 1$$

ONLINE ALGORITHM FOR EDP

→ based on our analysis of the offline greedy alg. we propose the following

Alg Short: if $d(s_i, t_i) \leq \sqrt{m}$, try to connect $s_i - t_i$

Alg Any: if there is a $s_i - t_i$ path, connect them

Online alg: with prob. 50% run Alg Short(s_i, t_i), otherwise run Alg Any(s_i, t_i).

Fact: This has competitive ratio $2 \cdot (2\sqrt{m} + 1)$

Intuition: We are making the "right" decision 50% of the time, so we lose 50% of the expected effectiveness. But we also need Alg Any, because using only Alg Short could result in not choosing any path at all

PATHS IN GRAPHS WITH CAPACITIES

→ every edge $e \in E(G)$ has capacity $c \in \mathbb{N}$ ... up to c paths can use every edge

↳ constant for G

Alg (greedy with capacities)

1. $I \leftarrow \emptyset$, $\beta \leftarrow \sqrt[c+1]{m}$. $\forall e \in E: d(e) \leftarrow 1$ ↳ lenghts of edge e
2. for each $i \in [k] \setminus I$, let P_i be a d-shortest $s_i - t_i$ path
3. find $j \in [k] \setminus I$ s.t. $d(P_j) \leq d(P_i)$ for $\forall i$
   ↳ if there is none P_j or if P_j violates the capacities, go to 5)
4. $I \leftarrow I \cup \{j\}$, $\forall e \in P_j: d(e) \leftarrow d(e) \cdot \beta$, go to 2) ← make used edges longer
5. return I and the paths

The structure of this alg is very similar to greedy without capacities

Theorem: The approximation ratio of the alg is $(3 \cdot c \cdot \sqrt[c+1]{m} + 1)$.

Note: $c = 1$ gives $3\sqrt{m} + 1$, so the same poly-level as previous greedy

Intuition: capacities make the problem easier because it harder to make a mistake.

Proof: Consider an optimal sol. OPT and let P_i^* denote the $s_i - t_i$ path used by OPT.

Def: A path P used by ALG is short $\equiv d(P) = \sum_{e \in P} d(P) \leq m^{\frac{c}{c+1}} = (\sqrt[c+1]{m})^c$
↳ $c = 1$, we have $\sqrt{m}$

Def: Denote by d_i the edge-lenght function at the end of iter i of ALG.

Def: Denote by $\bar{d}$ the edge-lenght function at the end of the last iteration, where ALG selected a short path.

⇒ we have: $d_0, d_1, d_2, \ldots d_\ell = \bar{d}, d_{\ell+1}, \ldots$

👁1: $\bar{d}(E) = \sum_{e \in E} \bar{d}(e) \le m(1+2|I|)$ ... even though $d(e)$ grows exponentially

↳ think about a partition of $E = E_0 \dot\cup E_1 \dot\cup \dots \dot\cup E_\ell$ ← ℓ = iter where $d_\ell = \bar{d}$

where E_i := edges used by ALG for the last time in iter i (last from $1,\dots \ell$)

↳ E_0 if not used at all

$$\sum_{e\in E} \bar{d}(e) = \sum_{i=0}^{\ell} \sum_{e \in E_i} \bar{d}(e) \le$$

→ we scale all these edges at the end of iter

$$\le m + \sum_{i=1}^{\ell} m^{\frac{c}{c+1}} \cdot \beta$$

↳ $d(P) \le m^{\frac{c}{c+1}}$

E_0 - the beginning

↳ $E_1, \dots E_\ell$ correspond to short paths used by ALG

$$\le m + 2 \cdot \ell \cdot m \le m + 2 \cdot m \cdot |I| \quad \leftarrow \beta = \lceil m^{\frac{1}{c+1}} \rceil$$ 👁1 ✓

👁2: For each $P \in OPT \setminus I$, $\bar{d}(P) \ge m^{\frac{c}{c+1}}$ ← paths used by OPT but not ALG are all long

↳ assume that $\bar{d}(P) < m^{\frac{c}{c+1}}$ for some $P \in OPT \setminus I$.

? How many greedy paths can use e before the end of iter ℓ?

! $< C$, because $\beta = \lceil \sqrt[c+1]{m} \rceil$, so $\beta^c \ge m^{\frac{c}{c+1}}$, but $\bar{d}(P) < m^{\frac{c}{c+1}}$

⇒ we haven't exhausted the capacity of any $e \in P$, so P can be still used by greedy in iter $\ell+1$. But P is short and ℓ is the last iter with short P_i ↯

We can finish the proof by combining 👁1 and 👁2.

Note that $|OPT \setminus I| \cdot m^{\frac{c}{c+1}} \le \sum_{P \in |OPT \setminus I|} \bar{d}(P) \le c \cdot \bar{d}(E)$ if we used every $e \in E$ c-times

$$\Rightarrow |OPT \setminus I| \le \frac{c \cdot \bar{d}(E)}{m^{\frac{c}{c+1}}} \le \frac{c \cdot m(1+2|I|)}{m^{\frac{c}{c+1}}} = c \cdot \sqrt[c+1]{m}\,(1+2|I|) \overset{1 \le |I|}{\le} 3c \cdot \sqrt[c+1]{m} \cdot |I|$$

$$\Rightarrow |OPT| \le |OPT \setminus I| + |I| \le 3c \cdot \sqrt[c+1]{m}\,|I| + |I| = (3c \cdot \sqrt[c+1]{m} + 1)|I|.$$ ▨

OPT_{long} $\quad$ $OPT_{short} \le I$

↳ we want to say that there can't be too many long paths

$$\#\text{long paths} \le \frac{\text{total length used by all long paths } (*)}{\text{min length of a long path}}$$

(*) worst case: they use every edge C times → (*) $\le C \cdot \bar{d}(E)$

k-CENTER PROBLEM

→ 1-center = given a finite metric space, find a center = a point that minimizes the max. distance to any other point

→ k-center = we want k "centers", such that every point is close to at least one of them

Def: Given a metric space (M,d) define for $v \in M$, $S \subseteq M$

$$d(v,S) := \min_{s \in S} d(v,s), \qquad R(S) = \max_{v \in M} d(v,S)$$

k-center problem:

IN: finite metric space (M,d) (finite), and $k \in \mathbb{N}^+$

OUT: $S \subseteq M$ s.t. $|S| \leq k$

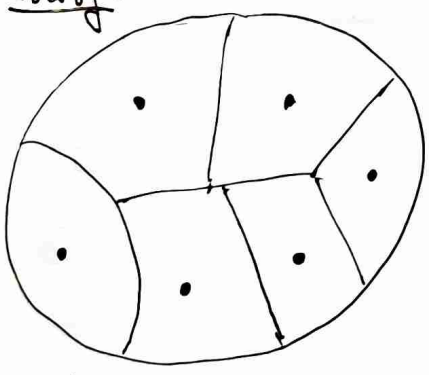
Objective: min $R(S)$

Algorithm - max greedy

1. $S \leftarrow \{s\}$ for a random $s \in M$
2. While $|S| < k$:
3. $\quad S \leftarrow S \cup \{s\}$ for $s = \underset{s \in M \setminus S}{\operatorname{argmax}}\, d(s,S)$ → add the point which is the most distant from S → break ties randomly

Theorem: This algorithm is a 2-approximation.

Proof:



The optimum divides the space M into certain regions (each point is assigned to its closest center)

① if ALG chose each center from a different region, then we have won, since for $\forall$ region R:

$$\max_{u,v \in R} d(u,v) \leq \max_{u,v \in R} \left(d(u,C_R) + d(C_R,v) \right) \leq 2 \cdot OPT$$

(C_R → center of R in OPT)

② suppose that 2 distinct points chosen by ALG fall into the same region, and suppose that $\exists v$ s.t. $d(v,S) > 2 \cdot OPT$

→ this cannot happen since the algorithm added 2 centers (from the same region) which have distance $\leq 2 \cdot OPT$, but it always selects max. greedily, so it should have chosen v ↯ ∎

LINEAR PROGRAMMING ALGORITHMS

→ linear programming recap:

$\max c^T x$, $x \in \mathbb{R}^n$ s.t. $Ax \leq b$, $A \in \mathbb{R}^{m \times n}$	$\max c^T x$, $x \geq 0$ s.t. $Ax = b$	max ↔ min

→ ! LP is polynomial - using ellipsoid alg.

→ integer programming

IP: $\max c^T x$ s.t. $Ax \leq b$, $x \in \mathbb{Z}^n$ ← NP-complete (can solve SAT)

↳ binary if $x \in \{0,1\}^n$ (binary)

→ linear relaxation of this IP would be $Ax \leq b$, $x \in \mathbb{R}^n$, or $x \in [0,1]$

👁 the optimum of the relaxation is always at least as good as of the integer program, as we are considering more posibilities

⇒ General strategy: formulate a problem as an integer program, solve the linear relaxation (poly) and show that the optimum of the relaxation gives a "good" solution of the original problem

GRAPH BALLANCING

IN: undirected $G=(V,E)$, weights $w: E \to \mathbb{Q}^+$

→ for a given orientation of G, define the weight of a vertex $v \in G$ as

$$w(v) := \sum_{u:\, u \to v} w(uv) \quad \dots \text{ sum of weights of incoming edges}$$

OUT: an orientation of G

Objective: min. the weight of the heaviest vertex ... min. $\max_{v \in V} w(v)$

IP: denote $w_{uv} = w_{vu} = w(\{u,v\})$, variables $x_{uv}, x_{vu} \in \{0,1\}$ for $\forall uv \in E$

$\forall u,v: x_{uv} + x_{vu} = 1$ ⇝ $x_{uv} = 1$: $u \to v$, $x_{vu} = 1$: $v \to u$

$\forall v: \sum_{u:\, uv \in E} x_{uv} \cdot w_{uv} \leq W$

$\min W$ → W is a variable modeling the max vertex weight

Relaxation: $x_{uv}, x_{vu} \in [0,1]$

→ this does not solve the original problem, it will potatly produce some solution with real numbers for x_{uv}

→ but we can use the OPT_{LP} to define a solution of the problem

Algorithm:

1. solve the linear relaxation $\rightsquigarrow X^*$
2. output the solution:

orient the edge $uv \in E$ as $\begin{cases} u \to v & \text{if } x^*_{uv} \geq \frac{1}{2} \\ v \to u & \text{if } x^*_{uv} < \frac{1}{2} \end{cases}$ $\left(\Leftrightarrow x_{vu} \geq \frac{1}{2}\right)$

Theorem: This is a 2-approximation.

Proof: It is poly $\because$ LP are poly

👁 it is a feasible solution (it produces an orientation) since $x_{uv} + x_{vu} = 1$

→ IP (OPT) produced an integral solution $\tilde{x}$, LP (OPT_{LP}) produced x^*

👁 $\forall v: \sum_{u : uv \in E} x^*_{uv} w_{uv} \leq OPT_{LP} \leq OPT$

↳ since the relaxation is more general

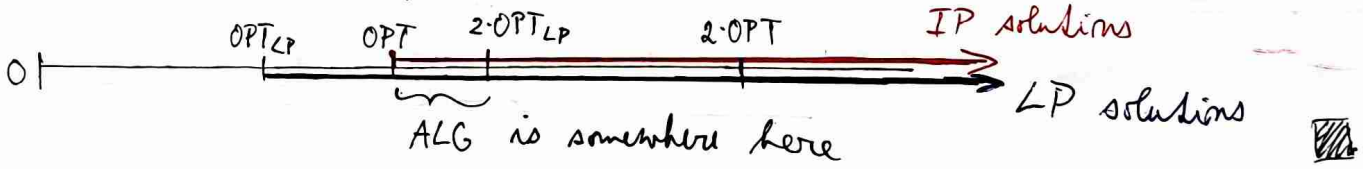
👁 ALG created an integral solution $x'_{uv} = \begin{cases} 1, & x^*_{uv} \geq \frac{1}{2} \\ 0, & \text{else} \end{cases}$

→ we want to make the sum as small as possible

→ if $x' = 0$, that's good, but what if a lot of $x'_{uv} = 1$ for some v?

$\forall v: \sum_{u : uv \in E} x'_{uv} w_{uv} \leq 2 \cdot \sum_{u : uv \in E} x^*_{uv} w_{uv}$, as $\forall x^*_{uv}$ can double at most

$\Rightarrow ALG \leq 2 \cdot OPT_{LP} \leq 2 \cdot OPT$



MAX SAT

→ some SAT instances are not satisfiable, but we can still try to satisfy as many clauses as we possibly can

IN: Conjugation of clauses $C_1 \wedge \dots \wedge C_m$, where each C_j is a disjunction of k_j literals: $C_j = l_1 \vee \dots \vee l_{k_j}$, variables $= x_1, \dots x_n$

OUT: assignment of variables $(a_1, \dots a_n) \in \{0,1\}^n$ ↳ literal $\in \{x_i, \neg x_i\}$

Objective: max # of satisfied clauses $= \max \#_j : C_j(\vec{a}) = 1$

👁 this is still NP-complete: we can solve SAT ... satisfiable $\Leftrightarrow OPT = m$

WLOG assume that:

① $C_j \neq \emptyset$ ↳ not satisfiable ② literals don't repeat in C_j ③ there is no $l \vee \neg l$ in C_j ↳ always satisfiable

Note: We have always written approx. ratios as a number ≥ 1, but it is customary to write $R \leq 1$ for some problems – e.g. Max-SAT

Max SAT variations:

- Max-kSAT ... $k_j \leq k$ for $\forall j$
- Max-EkSAT ... $k_j = k$ for $\forall j$

Max-3SAT is NP complete since 3-SAT is NP complete

Fact: Max-2SAT is NP-hard even though 2SAT is poly-solvable.

- Random SAT alg

Alg: Choose $(a_1, \dots a_n) \in \{0,1\}^n$ uniformly randomly.

→ we want the $\mathbb{E}$ number of satisfied clauses

⇒ create an indicator variable for each clause

↳ but wait! $C_j(\vec{a})$ is already an indicator for C_j being satisfied

$$\Rightarrow \mathbb{E}[ALG] = \mathbb{E}\left[\sum_{j=1}^{m} C_j(\vec{a})\right] = \sum_{j=1}^{m} \mathbb{E}[C_j(\vec{a})] = \sum_{j=1}^{m} P[C_j(\vec{a}) = 1]$$

if C_j has k_j literals, then $P[C_j(\vec{a}) = 1] = 1 - \frac{1}{2^{k_j}}$.

↳ there is only 1 assignment that fails (thanks to our WLOG assumption)

Theorem: Rand SAT is a $\frac{1}{2}$-approx for Max SAT and $\frac{7}{8}$-approx for Max-E3SAT.

Proof: For Max SAT: $\forall j: k_j \geq 1$, so $P[C_j(\vec{a}) = 1] \geq 1 - \frac{1}{2} = \frac{1}{2}$

$$\Rightarrow \mathbb{E}[ALG] \geq \sum_{j=1}^{m} \frac{1}{2} = \frac{1}{2} m \geq \frac{1}{2} OPT \quad \dots m = ALL \geq OPT$$

For Max-E3SAT: $\forall j: k_j = 3$, so $P = 1 - \frac{1}{2^3} = \frac{7}{8}$ $\Rightarrow \mathbb{E}[ALG] = \frac{7}{8} m \geq \frac{7}{8} OPT$ ∎

Fact: Rand SAT is the best possible approx. alg for Max-E3SAT

↳ $(\frac{7}{8} + \varepsilon)$-approx for any $\varepsilon > 0$ implies $P = NP$

• Biased SAT alg

→ the biggest disadvantage of solving Max SAT with Rand SAT are unit clauses ... if (x_i) appears more often than $(\neg x_i)$, then we should probably set $P[x_i = 1] > P[x_i = 0]$

↳ but not too much, as x_i may also appear in some larger clauses

→ Golden ratio recap: $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$, $\varphi - 1 = \frac{1}{\varphi} \approx 0.618$.

$$\varphi^2 - \varphi - 1 = 0 \implies 1 - \frac{1}{\varphi} - \frac{1}{\varphi^2} = 0 \implies 1 - \frac{1}{\varphi^2} = \frac{1}{\varphi} \quad (*)$$

Alg: Biased SAT

1. choose $\forall a_i \in \{0,1\}$ independently randomly as:
2. if $\#(x_i)$ unit clauses $>$ $\#(\neg x_i)$ unit clauses:

$$a_i \leftarrow \begin{cases} 1, & \text{prob} = \frac{1}{\varphi} \approx 0.618 \\ 0, & \text{prob} = 1 - \frac{1}{\varphi} \approx 0.382 \end{cases}$$

→ Rand SAT has $\frac{1}{2}$ both

3. otherwise: $(\#(x_i) \leq \#(\neg x_i))$

$$a_i \leftarrow \begin{cases} 0, & \text{prob} = \frac{1}{\varphi} \approx 0.618 \\ 1, & \text{prob} = 1 - \frac{1}{\varphi} \approx 0.382 \end{cases}$$

→ Rand SAT has $\frac{1}{2}$ both

Theorem: Biased-SAT is a $\frac{1}{\varphi}$-approx of Max SAT. ($\frac{1}{\varphi} \hookrightarrow 0.618$)

Rand: $\frac{1}{2} = 0.5$

Biased: $\frac{1}{1.618} \approx 0.618$

Proof:

If (x_i) and $(\neg x_i)$ both appear as unit clauses:

OPT can't do better

→ exactly 1 clause from the pair $(x_i) \wedge (\neg x_i)$ can be satisfied (by any assignment)

⇒ we can forget all these extra pairs, and analyse a simpler version

$$(x_1) \wedge (x_1 \vee x_2) \wedge (x_1) \wedge (\neg x_1) \wedge (\neg x_1) \wedge (x_1) \rightsquigarrow (x_1) \wedge (x_1 \vee x_2)$$

After this reduction, for any unit clause $C_j = \ell$, there is no clause $C_i = \bar{\ell}$

⇒ so there are more occurrences of ℓ than $\bar{\ell}$, hence $P[C_j(\vec{a}) = 1] = \frac{1}{\varphi}$

For clauses of length $\ell_j \geq 2$, we have

$$P[C_j(\vec{a}) = 1] \geq 1 - \frac{1}{\varphi^{\ell_j}} \geq 1 - \frac{1}{\varphi^2} \overset{(*)}{=} \frac{1}{\varphi} \approx 0.618$$

↓

worst case: chance to fail each individual literal $= \frac{1}{\varphi}$

↳ to fail entire clause = fail $\forall$ literal

↳ Hence every clause which can be handeled worse than OPT, we satisfy with prob ≥ 0.618

• Max SAT using Linear Programming

Integer program: $y_1, \dots y_n \in \{0,1\}$ … $y_i = 1 \equiv x_i = 1$

$\max \sum_{j=1}^{m} z_j$ $\quad z_1, \dots z_m \in \{0,1\}$ … $z_j = 1 \equiv C_j = 1$

$$\text{s.t.} \sum_{i: x_i \in C_j} y_i + \sum_{i: \neg x_i \in C_j} (1 - y_i) \geq z_j \quad \text{for } \forall j = 1, \dots m \qquad (*)$$

Linear relaxation: $y_i \in [0,1]$, $z_j \in [0,1]$

Alg: LP-SAT

1. solve the relaxation $\rightsquigarrow y_i^*, z_j^*$
2. choose for $\forall i$ randomly $a_i \leftarrow \begin{cases} 1, & \text{prob} = y_i^* \\ 0, & \text{prob} = 1 - y_i^* \end{cases}$

Lemma: If C_j has k_j literals, then $P[C_j(\vec{a}) = 1] \geq \left(1 - \frac{1}{e}\right) \cdot z_j^*$

Theorem: LP-SAT is a $\left(1 - \frac{1}{e}\right) \approx 0.632$-approx for Max SAT. → better than Biased SAT

Pf: It is poly since LP is poly. Thanks to the lemma we have

$$E[ALG] = \sum_{j=1}^{m} P[C_j(\vec{a}) = 1] \geq \left(1 - \frac{1}{e}\right) \sum_{j=1}^{m} z_j^* = \left(1 - \frac{1}{e}\right) \cdot OPT_{LP} \geq \left(1 - \frac{1}{e}\right) \cdot OPT$$

Proof of Lemma: Fix a clause C_j with k literals. Notice that the optimum of the relaxation and $P[C_j = 1]$ will not change if we

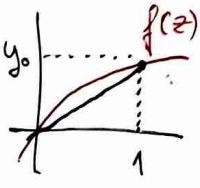
- rename the variables … obviously
- change all literals x_i in the instance I to $\neg x_i$, forming I'
  ↳ from a feasible solution $\vec{y}, \vec{z}$ of I, we get $y_i' = 1 - y_i$ and $z_j' = z_j$.

Thus WLOG $C_j = x_1 \vee x_2 \vee \dots \vee x_k$. We have

$$P[C_j(\vec{a}) = 0] = \prod_{i=1}^{k} (1 - y_i^*) \leq \left(\frac{1}{k} \sum_{i=1}^{k} (1 - y_i^*)\right)^k \quad \dots \text{AG-ineq: } \sqrt[n]{\prod a_i} \leq \frac{1}{n} \sum a_i$$

$$= \left(1 - \frac{1}{k} \sum_{i=1}^{k} y_i^*\right)^k \leq \left(1 - \frac{z_j^*}{k}\right)^k \quad \dots \text{since } \sum_{i: x_i \in C_j} y_i^* = \sum_{i=1}^{k} y_i^* \geq z_j^* \qquad (*)$$

The function $f(z) := 1 - \left(1 - \frac{z}{k}\right)^k$ is concave on $[0,1]$, as $f''(z) \leq 0$ on $[0,1]$.



→ hence on $[0,1]$: $y_0 = f(1) = 1 - \left(1 - \frac{1}{k}\right)^k$

$f(0) = 0 \rightarrow$ $f(z) \geq a \cdot z$, $f(1) = a \cdot 1 = a$ $\Big\}$ $f(z) \geq f(1) \cdot z$

$$\rightarrow P[C_j(\vec{a}) = 1] \geq 1 - \left(1 - \frac{z_j^*}{k}\right)^k = f(z_j^*) \geq f(1) \cdot z_j^* = \left(1 - \underbrace{\left(1 - \frac{1}{k}\right)^k}_{\leq 1/e \dots \text{standard bound}}\right) \cdot z_j^* \geq \left(1 - \frac{1}{e}\right) \cdot z_j^*$$

- **Best-SAT alg.** (↳ $P[\text{fail}] = \frac{1}{2}$; ↳ $P[\text{fail}] = \frac{1}{2^{\ell_j}}$)

→ Rand SAT struggled with unit clauses, but handled large clauses well

→ LP-SAT handles short clauses well, as it is easy points for the LP, but the relaxation struggles with long clauses, as it finds a wildly non-integral solution

Alg: Best-SAT : combine the two

1. for the given input, run with { prob 50% → Rand SAT ; prob 50% → LP-SAT

Theorem: Best-SAT is a $\frac{3}{4}$-approx for MaxSAT. (↳ alternatively, run both and pick the better one, also $\frac{3}{4}$-appr.)

Proof: For any clause C_j with ℓ_j literals:

$$P[C_j=1] = P[\text{Rand}]\cdot P[C_j=1 \mid \text{Rand}] + P[\text{LP}]\cdot P[C_j=1 \mid \text{LP}]$$

$$\geq \frac{1}{2}\cdot\underbrace{\left(1-\frac{1}{2^{\ell_j}}\right)}_{R(\ell_j)} + \frac{1}{2}\cdot\underbrace{\left(1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right)}_{L(\ell_j)}\cdot z_j^* \geq \frac{1}{2}z_j^*\left(R(\ell_j)+L(\ell_j)\right) \geq \frac{3}{4}z_j^* \quad (\text{↳ } 1 \geq z_j^*)$$

- $\ell_j = 1$: $R = 1-\frac{1}{2} = \frac{1}{2}$, $L = 1-0 = 1$ $\Rightarrow \frac{R+L}{2} = 0.75$
- $\ell_j = 2$: $R = 1-\frac{1}{4} = 0.75$, $L = 1-\left(\frac{1}{2}\right)^2 = 0.75$ $\Rightarrow \frac{R+L}{2} = 0.75$
- $\ell_j \geq 3$: $R \geq 1-\frac{1}{8} = 0.875$, $L \geq 1-\frac{1}{e} \approx 0.632$ $\Rightarrow \frac{R+L}{2} \approx \frac{1.507}{2} > 0.75$

Thus $\mathbb{E}[ALG] = \sum_{j=1}^{m} P[C_j(\vec{a})=1] \geq \frac{3}{4}\sum_j z_j^* = \frac{3}{4}OPT_{LP} \geq \frac{3}{4}OPT$ ∎

Visually:

Def: The integrality gap of an integer program with optimum OPT, and its linear relaxation with optimum OPT_{LP} is

- min problem: int. gap $= \frac{OPT}{OPT_{LP}} \geq 1$
- max problem: int. gap $= \frac{OPT_{LP}}{OPT} \geq 1$

(controls how large can this interval be)

Def: The integrality gap of a family of integer programs, that solve a certain approximation problem (e.g. MaxSAT) is the max int. gap. over all instances.

Best-SAT has integrality gap $\alpha = \frac{4}{3}$

- $\leq \frac{4}{3}$ since $OPT \geq \mathbb{E}[ALG] \geq \frac{3}{4}OPT_{LP} \geq \frac{3}{4}\cdot\alpha\cdot OPT$ $\Rightarrow \alpha \leq \frac{4}{3}\cdot\frac{OPT}{OPT}$
- $\geq \frac{4}{3}$: $(x_1 \vee x_2)\wedge(x_1 \vee \neg x_2)\wedge(\neg x_1 \vee x_2)\wedge(\neg x_1 \vee \neg x_2)$ $\rightsquigarrow OPT = 3$
  $y_1 = y_2 = \frac{1}{2}$, $z_1 = z_2 = z_3 = z_4 = 1$ $\Rightarrow OPT_{LP} = 4$ } $\alpha \geq \frac{4}{3}$

Corollary: We cannot devise a better approx. ratio for Max SAT, unless we employ some stronger bounds than $OPT \le OPT_{LP}$

Fact: There exist better approx. algs for Max SAT, based on semidefinite programing.

- **Derandomizing Max SAT algs** ⇝ **method of conditional expectations** → general method

→ we deterministically set x_1 so that the $\mathbb{E}$ value of the algorithm does not decrease

→ denote by W the objective ... $W = \sum C_j(\vec{a})$ ... # satisfied clauses

set $x_1 \leftarrow b_1$ where if $\mathbb{E}[W \mid x_1 = 1] \ge \mathbb{E}[W \mid x_1 = 0]$, then $b_1 = 1$, otherwise $b_1 = 0$

👁 $\mathbb{E}[W \mid x_1 = b_1] \ge \mathbb{E}[W]$ ← $\mathbb{E}$ is w.r.t. the randomized alg.

↳ $\mathbb{E}[W] = P_{ALG}[x_1 = b_1] \cdot \mathbb{E}[W \mid x_1 = b_1] + \underbrace{P_{ALG}[x_1 \neq b_1]}_{1 - P[x_1 = b_1]} \cdot \underbrace{\mathbb{E}[W \mid x_1 \neq b_1]}_{\le \mathbb{E}[W \mid x_1 = b_1]} \le p \cdot E + (1-p) \cdot E = \mathbb{E}[W \mid x_1 = b_1]$

→ suppose that we have already set $x_1 \leftarrow b_1$, $x_2 \leftarrow b_2$, ... $x_i \leftarrow b_i$. What about x_{i+1}?

set $x_{i+1} \leftarrow b_{i+1}$ where if $\mathbb{E}[W \mid (x_1 \dots x_i, x_{i+1}) = (b_1, \dots b_i, 1)] \ge$
$\mathbb{E}[W \mid (x_1 \dots x_i, x_{i+1}) = (b_1, \dots b_i, 0)]$, then $b_{i+1} = 1$, otherwise $b_{i+1} = 0$

👁 $\mathbb{E}[W \mid (x_1, \dots x_{i+1}) = (b_1, \dots b_{i+1})] \ge \mathbb{E}[W \mid (x_1, \dots x_i) = (b_1, \dots b_i)] \ge \dots \ge \mathbb{E}[W]$ (*)

→ how to calculate the conditional expectations for a given probabilistic Max SAT alg?

$$\mathbb{E}[W \mid (x_1, \dots x_i) = (b_1, \dots b_i)] = \sum_{j=1}^{m} \underbrace{P[C_j = 1 \mid (x_1, \dots x_i) = (b_1, \dots b_i)]}_{p_j} \quad \dots p_j = ?$$

- if setting $(x_1, \dots x_i)$ to $(b_1, \dots b_i)$ already satisfies C_j, then $p_j = 1$
- otherwise, p_j can be calculated by looking at the literals of C_j that are not set

example: $C_j = x_3 \vee \neg x_5 \vee \neg x_7$

- $P[C_j = 1 \mid x_1 = 1, x_2 = 0, x_3 = 1] = 1$ ... $x_3 = 1$ satisfies C_j
- $P[C_j = 1 \mid x_1 = 1, x_2 = 0, x_3 = 0] = 1 - P_{ALG}[x_5 = 1 \ \& \ x_7 = 1]$ → there is only 1 way to fail the clause
  - Rand SAT: $1 - \frac{1}{2} \cdot \frac{1}{2} = 0.75$
  - LP-SAT: $1 - y_5^* \cdot y_7^*$ where y_i^* are from the alg
  - Best-SAT: $P[x_i = 1] = \underbrace{P[\text{Rand}]}_{50\%} \cdot \underbrace{P[x_i = 1 \mid \text{Rand}]}_{50\%} + \underbrace{P[LP]}_{50\%} \cdot \underbrace{P[x_i = 1 \mid LP]}_{y_i^*} = \frac{1}{4} + \frac{1}{2} y_i^*$

Theorem: Suppose that ALG_R is a randomized Max SAT alg. with approx. ratio α, and define the deterministic alg ALG as setting all variables $x_1, \dots x_n$ to $b_1, \dots b_n$ as described above. Then ALG is a deterministic α-approx of Max SAT.
⇒ Thus there is a det. $\frac{1}{4}$-approx, by derandomizing Best SAT.

Pf: $ALG = \mathbb{E}[W \mid (x_1, \dots x_n) = (b_1, \dots b_n)] \ge$ (*) $\mathbb{E}[W] = \mathbb{E}[ALG_R]$ ∎

Weighted Max SAT

→ each clause C_j now has a <u>weight</u> $w_j \geq 0$ and we want to set the variables $x_1, \dots x_n$ to $a_1, \dots a_n$ to maximize

$$W = \sum_{j=1}^{m} w_j \cdot C_j(\vec{a})$$ → 1 if satisfied, 0 if fail

→ we previously considered only the case when all weights are equal

<u>Fact</u>: all previous algorithms (and the derandomization) also work for weighted Max SAT, using the same proofs (but more letters)

Maximal Cut

<u>IN</u>: undirected $G=(V,E)$, weights $w: E \to \mathbb{Q}^+$, $w_{uv} := w(\{u,v\})$

<u>OUT</u>: Partition of V into sets $S, V \setminus S$ ... the cut = edges between S and $V \setminus S$

<u>Objective</u>: maximize $\sum_{e \in E(S, V\setminus S)} w_e$ → $E(S, V\setminus S)$ = edges between S and $V \setminus S$

<u>Algorithm</u>: $\forall v \in V$: put v into S with probability 50%

<u>Theorem</u>: This "random cut" is a <u>$\frac{1}{2}$-approx.</u>

<u>Proof</u>: Define indicators $X_{uv} \in \{0,1\}$ s.t. $X_{uv} = 1 \equiv uv$ is in the cut

$\Rightarrow ALG = \sum_{uv \in E} w_{uv} X_{uv}$ ... random variable for the weight of the cut

$$\mathbb{E}[ALG] = \sum_{uv \in E} w_{uv} \mathbb{E}[X_{uv}] = \sum_{uv \in E} w_{uv} P[uv \in \text{cut}]$$

→ since the end points u, v were placed into S or $V \setminus S$ independently randomly:

- $u \in S$, $v \in S$ ... 25% → not cut
- $u \in S$, $v \in V \setminus S$ ... 25% → in cut
- $u \in V \setminus S$, $v \in S$ ... 25% → in cut
- $u \in V \setminus S$, $v \in V \setminus S$ ... 25% → not cut

$\Rightarrow P[uv \in \text{cut}] = 50\%$

$$\Rightarrow \mathbb{E}[ALG] = \frac{1}{2} \sum_{uv \in E} w_{uv} \geq \frac{1}{2} OPT$$ → cannot do better than all edges ∎

<u>Note</u>: This can be <u>derandomized</u> using the conditional expectations method.

<u>Fact</u>: There is a 0.878-approximation via relaxing a semidefinite program.

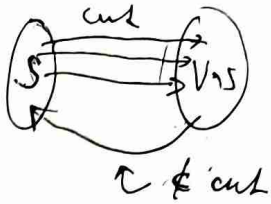
<u>Fact</u>: If there exists a α-approx for $\alpha > \frac{16}{17} \approx 0.941$, then $P = NP$.

Maximal Oriented Cut.

IN: directed $G=(V,E)$, weights $w: E \to \mathbb{Q}^+$, $w_{uv} := w((u,v))$

OUT: partition of V into S, $V \setminus S$ ... cut $= E(S, V\setminus S)$ = edges from S to $V\setminus S$

Obj: maximize $\sum_{e \in E(S,V\setminus S)} w_e$ ... weights $S \to V\setminus S$



Alg: $\forall v \in V$: put v into S with prob. 50%

Theorem: This "random cut" is a $\frac{1}{4}$ approx.

Proof: Identical proof as for unoriented, but $P[uv \in cut] = 25\%$ ∎

- **LP-based alg.** – model using an integer program

variables: $\forall v \in V$: $x_v \in \{0,1\}$ ... $x_v = 1 \equiv v \in S$

$\forall uv \in E$: $z_{uv} \in \{0,1\}$ ... $z_{uv} = 1 \equiv u \in S \ \&\ v \in V\setminus S$ ($x_u = 1$, $x_v = 0$)

$$\max w^T z = \sum_{uv \in E} w_{uv} \cdot z_{uv}$$

conditions: $\forall uv \in E$:

$z_{uv} \le x_u$: $u \notin S \Rightarrow z_{uv} = 0$; $u \in S \Rightarrow z_{uv} = 0$ or $z_{uv} = 1$

$z_{uv} \le 1 - x_v$: $v \in S \Rightarrow z_{uv} = 0$; $v \notin S \Rightarrow z_{uv} = 0$ or $z_{uv} = 1$

→ and we want to max. $\sum z_{uv}$

- **linear relaxation**: $x_v \in [0,1]$, $z_{uv} \in [0,1]$

Algorithm:

1. Solve the linear relaxation $\rightsquigarrow x^* \in [0,1]^n$, $z^* \in [0,1]^m$
2. $\forall v \in V$: put v into S with probability $\frac{1}{4} + \frac{x_v^*}{2}$

Theorem: The above alg. is a $\frac{1}{2}$-approximation.

Proof: Define indicators $X_{uv} \in \{0,1\}$ s.t. $X_{uv} = 1 \equiv$ edge $u \in S \to v \in V\setminus S$ is in the cut.

$$\mathbb{E}[ALG] = \sum_{uv \in E} w_{uv} \mathbb{E}[X_{uv}] = \sum_{uv \in E} w_{uv} P[uv \in cut] = \sum_{uv \in E} w_{uv} P[u \in S] \cdot P[v \notin S]$$

$$= \sum_{uv \in E} w_{uv} \left(\frac{1}{4} + \frac{x_u^*}{2}\right)\left(1 - \frac{1}{4} - \frac{x_v^*}{2}\right) = \sum_{uv \in E} w_{uv} \underbrace{\left(\frac{1}{4} + \frac{x_u^*}{2}\right)\left(\frac{1}{4} + \frac{1-x_v^*}{2}\right)}_{(*)}$$

We want to show $\mathbb{E}[ALG] \ge \frac{1}{2} OPT_{LP} = \frac{1}{2} \sum w_{uv} z_{uv}^* \ge \frac{1}{2} OPT$ ⇒ 2-approx

↳ $OPT_{LP} \ge OPT$

⇒ we need $(*) \ge \frac{1}{2} z_{uv}^*$

We do this by $(*) \ge \min\left(\frac{x_u^*}{2}, \frac{1-x_v^*}{2}\right) = \frac{1}{2}\min(x_u^*, 1-x_v^*) \ge \frac{1}{2} z_{uv}^*$

The first inequality holds, as if $x_u^* \le 1 - x_v^*$, then

$(*) \ge \left(\frac{1}{4} + \frac{x_u^*}{2}\right)^2 = \frac{1}{16} + \frac{x_u^*}{4} + \frac{(x_u^*)^2}{4} \ge \frac{x_u^*}{2}$ → solve the quadratic equation

and similarly for the case $x_u^* > 1 - x_v^*$. ∎

Minimal Vertex Cover

IN: $G=(V,E)$

OUT: $W \subseteq V$ s.t. $\forall e \in E: e \cap W \neq \emptyset$

Objective: min $|W|$

👁 W is a vertex cover $\Leftrightarrow$ $V \setminus W$ is an independent set

$\Rightarrow$ min vertex cover is NP complete

- Combinatorial approx:

Algorithm: $W \leftarrow \emptyset$

1. while $\exists$ uncovered edge e:
2. add both endpoints of e into W

Theorem: This is a 2-approx, and the bound is tight.

Pf: 👁 always finds a vertex cover

👁 the edges $e_1, e_2, e_3, \dots e_k$ chosen by the alg. are a matching M

Since $W = \cup M$ is a vertex cover, we have

$$k = |M| \leq OPT \leq |W| = ALG = 2k$$

↳ if M is a matching, and C a v.c., then $|M| \leq |C|$

Thus $k = \frac{ALG}{2} \leq OPT \Rightarrow$ 2 approx

The bound is tight, consider

$OPT = 2$, $ALG = 4$ no matter how we chose edges ▨

- LP-approx:

Integer program: variables $x_v \in \{0,1\}$, $x_v = 1 \equiv v \in W$

$$\min \sum_{v \in V} x_v$$

$$\text{s.t. } \forall uv \in E: x_u + x_v \geq 1$$

$\rightsquigarrow$ linear relaxation: $x_v \in [0,1]$ (or even $x_v \geq 0$ works)

Alg:

1. solve the linear relaxation $\rightsquigarrow x^*$
2. $W \leftarrow \{v \in V \mid x_v^* \geq \frac{1}{2}\}$

Theorem: This is a 2-approx.

Proof: it gives a feasible solution

→ if $uv \in E$, then $X_u^* + X_v^* \geq 1 \Rightarrow$ one of them is $\geq \frac{1}{2} \Rightarrow \in W$

it is a 2-approx:

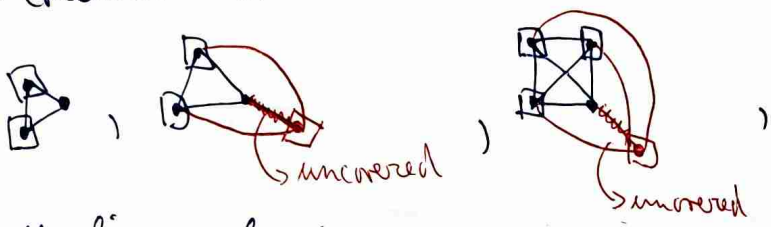
$$ALG = \sum_{v \in W} 1 \leq \sum_{v \in W} 2 \cdot X_v^* \leq 2 \sum_{v \in V} X_v^* = 2 \cdot OPT_{LP} \leq 2 \cdot OPT$$

⇓ ∎

Proposition: It has integrality gap $\geq 2\left(1 - \frac{1}{n}\right)$. ⟷ int. gap ≤ 2

Proof: We want to find a graph s.t. $\frac{OPT}{OPT_{LP}} \geq 2\left(1 - \frac{1}{n}\right)$

→ consider K_n



$\Rightarrow OPT = n-1$

→ the linear relaxation can do: $\forall v: X_v = \frac{1}{2}$, then $\sum_v X_v = \frac{n}{2}$ is a feasible sol.

$$\Rightarrow \frac{OPT}{OPT_{LP}} \geq \frac{n-1}{n/2} = 2\left(1 - \frac{1}{n}\right)$$

∎

Max Independent Set

→ we could try to use our min. vertex cover 2-approx and the fact that

→ Ⓚ $n = \alpha + \tau$, where α = size max ind set, τ = size min vertex cover

→ if we have a c-approx of vertex cover, find cover W, and look at the independend set $S := V \setminus W$, then since $|W| \leq c \cdot \tau$ (c-approx) we have

$$|S| = n - |W| \geq n - c \cdot \tau = n - c \cdot (n - \alpha) = c \cdot \alpha - (c-1)n$$

→ for 2-approx: $|S| \geq 2\alpha - n$

$\Rightarrow$ if $\alpha \leq \frac{n}{2}$, we have no guarantee that S will even contain anything!

→ for example K_n: $\alpha = 1$, and the 2-approx finds cover $W = V$, so our alg. would output $|S| = 0$ .... $\frac{OPT}{ALG} = \frac{1}{0} = \infty$-approx!

• **LP-approach:**

$\max \sum_v X_v$ s.t. $\forall uv \in E: X_u + X_v \leq 1$, $X_v \in \{0,1\} \rightsquigarrow [0,1]$

this has integrality gap $\geq \frac{n}{2}$ ... K_n: $OPT = 1$, $OPT_{LP} \geq \frac{n}{2}$ ↳ $\forall v: X_v = \frac{1}{2}$

Corollary: No approx alg. based on this LP can have approx ratio better than $\frac{n}{2}$. (unless we use something smarter than $OPT_{LP} \geq OPT$)

- **Greedy approach**

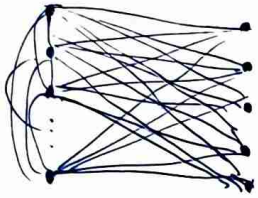
Algorithm: $\Delta := \max \deg(G)$

1. $S \leftarrow \emptyset$ ... ind set
2. while $G \neq \emptyset$:
3. pick any $v \in G$, and add it to S
4. remove v and all its neighbours from G

Theorem: This is not a c-approx. for any $c < \infty$.

Proof: Consider the graph ...

K_m I_k ... complete bipartite graph with parts K_m and I_k



$OPT = k$, $n = m + k$

$$\mathbb{E}[ALG] = \frac{m}{m+k} \cdot 1 + \frac{k}{m+k} \cdot k = \frac{m+k^2}{m+k}$$

If Greedy picks $v \in K_m$ first, it deletes everything else and $|S| = 1$

If it picks $v \in I_k$, it deletes the entire K_m and then $|S| = |I_k| = k$

$$\Rightarrow \text{approx ratio} = \frac{OPT}{\mathbb{E}[ALG]} = \frac{k(m+k)}{m+k^2} \quad ; \quad n = m+k$$

By choosing $k = \sqrt{n}$ and letting $n \to \infty$:

$$\frac{OPT}{\mathbb{E}[ALG]} = \frac{\sqrt{n} \cdot n}{(n-\sqrt{n})+n} = \frac{n\sqrt{n}}{2n-\sqrt{n}} \to \frac{\sqrt{n}}{2-0} = \frac{1}{2}\sqrt{n}$$

Thus we can make the approx ratio arbitrarily large

If Greedy outputs S, then $|S| \geq \frac{n}{\Delta+1}$. $\Delta = \max \deg$

↳ for $\forall v \in S$ we remove $\leq \Delta + 1$ vertices

$\Rightarrow$ Greedy has no constant approximation factor, it is a "n-approximation"

Fact: If for any $\varepsilon > 0$ exists a $n^{1-\varepsilon}$-approx, then $P = NP$.

independent set on k vertices

Minimal Set Cover

IN: $S_1, S_2, \dots, S_m \subseteq [n]$ s.t. $\bigcup S_i = [n]$ ← UNIVERSE

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

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

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

} want: buy some S_i's to cover the entire universe for the lowest cost

LP for min set cover: ↦ $x_i = 1$ when we buy S_i

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

(*) s.t. $\forall e \in [n]: \sum_{i : e \in S_i} x_i \geq 1$ ... make sure we buy $\forall e \in [n]$

Min Set Cover is a generalization of Min Vertex Cover

↳ vertex cover ~ Universe = edges, S_i = edges incident with vertex v_i, $c_i = 1$

Def: Denote

$f := \max_{e \in [n]} |\{j \mid e \in S_j\}|$ ... f = max # variables that appear in a (*) constraint

↳ in how many sets is e present

$g := \max_{j \in [m]} |S_j|$ ... size of the biggest set

For vertex cover: $f = 2$ ... edges have 2 ends, g = max deg

- Linear relaxation algorithm

Alg:

1. solve the linear relaxation ⇝ x^*
2. $I \leftarrow \{i \in [m] \mid x_i^* \geq \frac{1}{f}\}$

this gives feasible solution

→ $e \in [n]$, then $\sum_{i : e \in S_i} x_i \geq 1$ so $\exists i : x_i \geq \frac{1}{f}$

Theorem: This is a f-approx. alg

Proof:

$$ALG = \sum_{i \in I} c_i \leq \sum_{i \in I} c_i \, f \, x_i^* \leq f \sum_{i=1}^{m} c_i x_i^* = f \cdot OPT_{LP} \leq f \cdot OPT$$ ∎

↳ this is very similar to what we did with vertex cover

- **Primal - Dual Algorithm** (min set cover)

Primal LP:

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

$\forall e \in [n]: \sum_{i: e \in S_i} x_i \geq 1$

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

Dual LP

$y_e \geq 0, \quad e \in [n]$

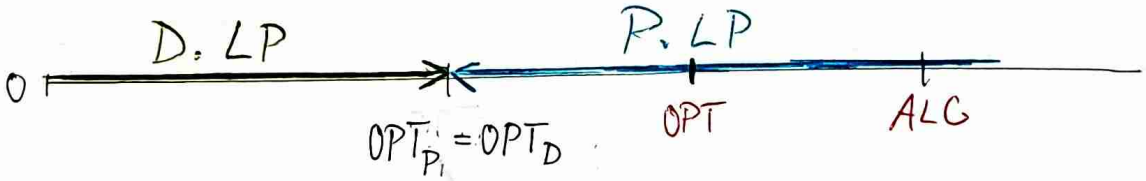
$\forall i \in [m]: \sum_{e \in S_i} y_e \leq c_i$

obj: $\max \sum_{e=1}^{n} y_e$

Intuition:

- **Primal:** want to buy $\forall e \in [n]$, by buying "percentages" of sets e belongs to. Goal: buy as cheap as possible
- **Dual:** want to sell as many $e \in [n]$ as possible so that we make as much money as we can. But there are certain limits on the total profit we can make from certain groups of items

Duality theorem: $(\forall \bar{x} \in P)(\forall \bar{y} \in D)$ $\bar{x}, \bar{y}$ feasible, $cost(\bar{x}) \geq cost(\bar{y})$ and (important!) $OPT_P = OPT_D$ (primal, dual)



Complementarity conditions: Something has to be tight

If $\bar{x}$ and $\bar{y}$ are feasible solutions of P and D, then they are optimal ($\Leftrightarrow$)

(1) $\forall i: \quad \bar{x}_i = 0 \quad \vee \quad \sum_{e \in S_i} \bar{y}_e = c_i$, and

(2) $\forall e: \quad \bar{y}_e = 0 \quad \vee \quad \sum_{i: e \in S_i} \bar{x}_i = 1$

Intuition: Optimum ~ if we buy/sell something, we can then sell/buy it back for the same amount

(1): if $\bar{x}_i > 0$, bought S_i $\Rightarrow$ selling elements from S_i with max profit
if $\sum \bar{y}_e < c_i$, not selling elements of S_i optimaly $\Rightarrow$ dont buy S_i

(2) if $\bar{y}_e > 0$, sold e $\Rightarrow$ buying element e as efectively as possible
if $\sum \bar{x}_i > 1$, not buying e optimaly $\Rightarrow$ dont sell $\bar{y}_e$

→ we will try to approximately fulfill the complementary conditions to get a solution very close to $OPT_P = OPT_D$.

Algorithm:

I = purchased sets
E = covered elements
y_e = market price of e

1. $I \leftarrow \emptyset$, $E \leftarrow \emptyset$, $\forall e \in [n]: y_e \leftarrow 0$
2. choose $e \notin E$: (uncovered e)
3. $\delta \leftarrow \min_{i: e \in S_i} \left(C_i - \sum_{e' \in S_i} y_{e'} \right)$
4. $y_e \leftarrow y_e + \delta$ ☒
5. for $\forall i$ s.t. $e \in S_i$ & $\sum_{e' \in S_i} y_{e'} = C_i$:
6. $I \leftarrow I \cup \{i\}$, $E = E \cup S_i$

- we try to increase the price of y_e as much as possible
- if the condition for $S_i \ni e$ is tight, buy S_i

👁 The solution produced by this algorithm is feasible, is an integral solution of P, and fulfills condition (1) (¥)

Theorem: This is an f-approximation.

Remark: The simple LP alg was also f-approx, but this algorithm is nicer, as it does not actually require solving the LP ⇒ easy to implement.

Proof: Denote by $\overline{y_e}$ the value of y_e at the end of the algorithm

👁 $\forall i \in I: \sum_{e \in S_i} \overline{y_e} = C_i$ (¥)

👁 ☒ we could just write $y_e \leftarrow \delta$ since each y_e is increased only once — we cover e in step 5, so it cannot be chosen again

$$ALG = \sum_{i \in I} C_i = \sum_{i \in I} \sum_{e \in S_i} \overline{y_e} = \sum_{e \in [n]} \overline{y_e} \cdot \underbrace{|\{i \in I \mid e \in S_i\}|}_{\leq f} \leq f \cdot OPT_D \leq f \cdot OPT$$

■

- **Greedy algorithm for min. set cover**

1. $I \leftarrow \emptyset$, $E \leftarrow \emptyset$, $\forall e \in [n]: q_e \leftarrow 0$
2. while $E \neq [n]$:
3. for $\forall i$: $S_i \subseteq E$ denote by $A_i := S_i \setminus E$ the uncovered elements in S_i
4. $\rho_i \leftarrow \frac{c_i}{|A_i|}$ ... price per uncovered element in S_i
5. $j \leftarrow$ index i with minimal ρ_i ... most efficient S_j
6. set $q_e \leftarrow \rho_j$ for $\forall e \in A_j$ ... q_e = how efficiently we bought e
7. $I \leftarrow I \cup \{j\}$, $E \leftarrow E \cup A_j$

👁 this is polynomial and produces a feasible solution

Theorem: This is a H_g-approx, where $H_g = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{g} \approx \ln(g)$

Proof: Recall that $g = \max_i |S_i|$

👁 q_e has a similar role as y_e in the Primal-Dual algorithm, namely

$\forall q_e \geq 0$ and $\forall q_e$ changed only once

👁 $\sum_{e \in [m]} q_e = \sum_{i \in I} c_i = ALG$

↳ by the definition of ρ_i ... $\sum_{e \in A_i} \rho_i = |A_i| \rho_i = c_i$

claim: $\overline{y} := \frac{1}{H_g} \vec{q}$ is a feasible solution of the Dual LP

↳ the vector $\vec{q} = (q_1, \ldots, q_m)$

corollary: $\sum_{e \in [m]} \overline{y}_e = \frac{1}{H_g} \sum q_e = \frac{1}{H_g} ALG \leq OPT_D \leq OPT \Rightarrow H_g$ approx

proof: look at $S_i = \{e_1, e_2, \ldots, e_\ell\}$, $\ell \leq g$

↳ we label the elements of S_i so that the algorithm first covers e_ℓ, then $e_{\ell-1}$, ... and finally e_1

$\Rightarrow q_\ell \leq q_{\ell-1} \leq \cdots \leq q_1$ ..... we purchase the most efficient first

👁 $q_{e_\ell} \leq \frac{c_i}{\ell}$, $q_{e_{\ell-1}} \leq \frac{c_i}{\ell-1}$, ..., $q_{e_j} \leq \frac{c_i}{j}$

$\Rightarrow \sum_{e \in S_i} q_e \leq \frac{c_i}{\ell} + \frac{c_i}{\ell-1} + \cdots + \frac{c_i}{j} + \cdots + \frac{c_i}{1} = H_\ell c_i \leq H_g c_i$

The dual condition is $\sum_{e \in S_i} y_e \leq c_i$ ⇒ need to divide q by H_g ∎

Proposition: The H_g-approx. bound is tight.

Proof: We construct a counter example

elements $= [n]$, sets $= S_1, S_2, \dots, S_n, T$ where $S_i = \{i\}$, $T = [n]$

costs $= c_1, c_2, \dots, c_n, c_T$ where $c_i = \frac{1}{i}$, $c_T = 1+\varepsilon > 1$

T has effectivity $\mu_T = \frac{1+\varepsilon}{n} = \frac{1}{n} + \delta$

$\Rightarrow$ greedy will buy c_n with $\mu_n = \frac{1}{n}$

After this, T will have effectivity $\mu_T = \frac{1+\varepsilon}{n-1} = \frac{1}{n-1} + \delta$ $\Rightarrow$ buy c_{n-1}

$\Rightarrow$ greedy will buy all c_i, as T will never be the most effective

$\Rightarrow ALG = \frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2} + 1 = H_n$, $OPT = 1+\varepsilon \to 1$, $\varepsilon \to 0$

$\to$ since $g = n$ in this case, we have $\frac{ALG}{OPT} \to H_g$ ▨

$\subseteq$-Maximal Independent Set = MIS

$\to$ we have seen that the largest independent set problem cannot be approximated

$\to$ we will look for an independent set that cannot be extended, but we do not care about its size

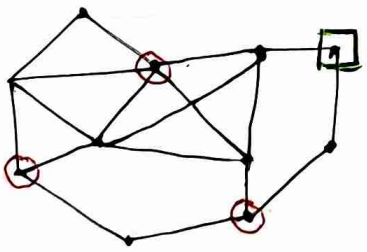
IN: $G = (V, E)$

OUT: $I \subseteq E$ s.t. I is independent ... $E \cap \binom{I}{2} = \emptyset$

Objective: I is $\subseteq$-maximal ... there is no independent I' s.t. $I \subset I'$

Equivalently: $I \cup N(I) = V$ ... $N(I) = \{v \in V \mid \exists u \in I : uv \in E\}$

Example:



○ alone are not $\subseteq$-max ind., but ○ + □ yes

Parallel algorithms: We have many processors, maybe one per each vertex/edge, but we want log or poly-log (e.g. $\log^2$) time

$\to$ we always want to cover a constant-fraction of the solution in each iteration to guarantee fast enough convergence – log many iterations

Parallel Randomized Algorithm for MIS

- K_n ... we need only 1 vertex
- C_n, P_n ... we need linearly many vertices in each iteration to ensure convergence

→ idea: choosing vertices will depend on their degree $= d_v$

Algorithm: PARA-MIS

Goal: $I \cup N(I) = V$

1. $I \leftarrow \emptyset$
2. while $V \neq \emptyset$ → add vertices with no neighbours
3. $(\forall v \in V)$: if $d_v = 0$, then $I \leftarrow I \cup \{v\}$, $V \leftarrow V \setminus \{v\}$
4. $(\forall v \in V)$: "mark" v with probability $p_v = \frac{1}{2 \cdot d_v}$
5. $(\forall uv \in E)$: if both u and v have been marked, unmark the one with the smaller degree (*)
6. $S \leftarrow \{v \mid v \text{ marked}\}$, $I \leftarrow I \cup \{S\}$, $V \leftarrow V \setminus (I \cup N(I))$, $E \leftarrow E \cap \binom{V}{2}$ — update graph
7. output I

④ selects new vertices to be added to the ind. set
⑤ fixes them so that they in fact are independent
(*) which one should we keep (add to I)?
- keep smaller degree ⇒ larger ind set ... but we don't care about size
- keep bigger degree ⇒ converges faster

Parallelism: We have one processor for each vertex and each edge
- steps 3, 4 are performed in parallel for $\forall$ vertex, and 5 for each edge

▽ There can be collisions in step 5, which would not be a problem if processing sequentially, but this is in parallel

Fact: This can be handled in constant time ⇒ it is OK

Theorem: The expected number of iterations of PARA-MIS is $O(\log n)$.

↳ proof follows

↳ # processors $= n + m$

PARA-MIS Analysis

Def: $v \in V$ is **good** $\equiv$ it has at least $\frac{d_v}{3}$ neighbours with deg $\leq d_v$

otherwise bad $\hookrightarrow \geq \frac{1}{3}$ of its neighbours have lower degree

→ we want to add vertices with high degree to I, as they will eliminate many other vertices (their neighbours)

→ good vertices eliminate vertices with small degree, so it is OK that we will not be able to add them to I later

Lemma [1]: In each iteration: v good $\Rightarrow$ $P[v \text{ removed in } ⑥] \geq \alpha$ $(\approx 8\%)$.

Corollary: On average, a constant fraction of good vertices is removed each round

Def: $uv \in E$ is **good** $\equiv$ u is good or v is good
bad $\equiv$ both u and v are bad

Lemma [2]: # good edges $\geq \frac{1}{2}|E|$

Corollary: Denote $E_i :=$ edges after i-th iteration. Then $\mathbb{E}[|E_{i+1}|] \leq (1-\frac{\alpha}{2})\mathbb{E}[|E_i|]$

Proof: At least half of E_i are good, each good edge has a good vertex which is removed with probability $\geq \alpha$, so the edge is removed with prob. $\geq \alpha$. Now: $\mathbb{E}[|E_{i+1}|] \leq \mathbb{E}[|E_i|] - \alpha \cdot \#\text{good edges} \leq \mathbb{E}[|E_i|] - \alpha \cdot \frac{1}{2}\mathbb{E}[|E_i|]$ ■

Theorem: The expected number of iterations is $O(\log n)$ ($|E| \leq n^2$)

Proof: After $t = \text{constant} \cdot \log n$ phases: $\mathbb{E}[|E_t|] \leq (1-\frac{\alpha}{2})^t n^2 \leq \frac{1}{2}$ ■

- **Proof of Lemma [2]:** Orient the edges towards the bigger-deg endpoint

Suppose that $uv \in E$ is bad (so u & v are bad) and oriented $u \to v$

v is bad $\Leftrightarrow$ $d_v^{IN} < \frac{1}{3}d_v$ ... if $u \to v$ then u has smaller deg

$\Rightarrow$ For a bad edge $u \to v$, the end-vertex of this edge is bad, and $d_v^{IN} < \frac{1}{3}d_v$, $d_v^{OUT} \geq \frac{2}{3}d_v$, so we have

Hence we can map the bad in-edge $u \to v$ onto 2 out-edges, since $d_v^{OUT} \geq 2 d_v^{IN}$, we can map each bad in-edge of v onto two unique out edges

$\Rightarrow$ in total, each bad edge can be mapped onto 2 unique edges $\Rightarrow |E| \geq 2 \cdot |E_B|$ ■ (bad edges)

$\Rightarrow |E_B| \leq \frac{1}{2}|E|$

Proof of Lemma 1: Suppose v is good, we want $P[v \text{ removed}] \geq \alpha$

→ v is removed if one of its neighbours w was marked in ④ and remained marked in ⑤

$$P[\exists w \in N(v) \text{ that was marked in ④}] = 1 - P[\text{none of } w \in N(v) \text{ were marked}]$$

$$= 1 - \prod_{w \in N(v)} P[w \text{ not marked}] = 1 - \prod_{w \in N(v)} \left(1 - \frac{1}{2d_w}\right) \geq 1 - \prod_{\substack{w \in N(v) \\ d_w \leq d_v}} \left(1 - \frac{1}{2d_w}\right) \quad \ulcorner d_v$$

↳ we mark vertices independently

$$\geq 1 - \prod_{\substack{w \in N(v) \\ d_w \leq d_v}} \left(1 - \frac{1}{2d_v}\right) = 1 - \left(1 - \frac{1}{2d_v}\right)^{\#\text{neighbors of } v \text{ with lower deg } (*)} \geq 1 - \left(1 - \frac{1}{2d_v}\right)^{\frac{1}{3}d_v}$$

v good $\Rightarrow (*) \geq \frac{1}{3} d_v$

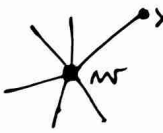
$$\geq 1 - \underbrace{\left[\left(1 - \frac{1}{2d_v}\right)^{2d_v}\right]^{\frac{1}{6}}}_{\leq \frac{1}{e}} \geq \underline{\underline{1 - e^{-\frac{1}{6}}}} \approx 0.153$$

→ some $w \in N(v)$ will be marked with prob $\geq \bullet$. Will it stay marked?

Claim: $P[w \text{ stays marked after ⑤}] \geq \frac{1}{2}$.

Corollary: $P[v \text{ is removed}] \geq \frac{1}{2} \cdot (1 - e^{-\frac{1}{6}}) \approx 0.077$

Proof: w can be unmarked only if it has a marked neighbour x with $d_x \geq d_w$



$d_x \geq d_w$ $\quad P[x \text{ is marked} \mid w \text{ marked}] = P[x \text{ marked}] = \frac{1}{2d_x} \leq \frac{1}{2d_w}$ (independence)

$$\Rightarrow P[\exists \text{ marked } x \in N(w) \text{ with } d_x \geq d_w] \leq d_w \cdot \frac{1}{2d_w} = \frac{1}{2}$$ ∎

PAIRWISE INDEPENDENCE

Def: Random events $A_1, A_2, \ldots A_n$ are

- **fully independent** $\equiv \forall \{i_1, \ldots i_k\} \subseteq [n]: P(A_{i_1} \cap \cdots \cap A_{i_k}) = \prod_{l=1}^{k} P(A_{i_l})$
- **pairwise independent** $\equiv \forall i \neq j \in [n]: P(A_i \cap A_j) = P(A_i) \cdot P(A_j)$

Def: Random variables $X_1, \ldots, X_n$ are

- **fully independent** $\equiv \forall \{i_1, \ldots i_k\} \subseteq [n] \quad \forall x_{i_1} \in \mathrm{Im}(X_{i_1}), \ldots, \forall x_{i_k} \in \mathrm{Im}(X_{i_k})$:

$$P[X_{i_1} = x_{i_1} \wedge \ldots \wedge X_{i_k} = x_{i_k}] = \prod_{l} P[X_{i_l} = x_{i_l}]$$

- **pairwise independent** $\equiv \forall i \neq j \in [n] \quad \forall x_i \in \mathrm{Im}(X_i), \forall x_j \in \mathrm{Im}(X_j)$

$$P[X_i = x_i \,\&\, X_j = x_j] = P[X_i = x_i] \cdot P[X_j = x_j]$$

Example: n coin-flips: events A_{ij} = the results of flips i and j are the same

A_{ij} are pair-wise independent $\quad \forall ij: P[A_{ij}] = \frac{1}{2}$

- $i,j \neq k,l$: $P[A_{ij} \wedge A_{kl}] = P[A_{ij}] \cdot P[A_{kl}]$
- what if they are not distinct? $P[A_{1,2} \wedge A_{2,3}] = P[A_{1,2}] \cdot P[A_{2,3} \mid A_{1,2}]$

A_{ij} are not fully ind – fails for 3 events

$1=2$	3	
0	0	$\frac{1}{2}$
0	1	
1	0	
1	1	

$P[A_{12} \mid A_{23} \wedge A_{13}] = 1$, but we want $= \frac{1}{8}$

(A_{23}: $2=3$, A_{13}: $1=3$)

DERANDOMIZING PARA-MIS

→ we will derandomize it using pairwise independence – general method !

Let A_v = event that vertex v was marked in step ④ of the alg. ($P[A_v] = \frac{1}{2d_v}$)

$X_v \in \{0,1\}$ indicator for A_v ... $X_v = 1$ iff v was marked

→ since $X_v \in \{0,1\}$, there are 2^n, $n = |V|$ configurations of X_v $\forall v \in V$

→ assume that we are considering only poly-many configurations, and

- run steps ⑤ and ⑥ in parallel for each conf ⇒ #conf · $(n+m)$ processors
- select the best result – the one that gets rid of the most edges in ⑥

→ our earlier analysis showed that $\mathbb{E}[|E_{i+1}|] \leq (1 - \frac{\alpha}{2})\mathbb{E}[|E_i|]$ (*)

↳ a constant fraction of edges is removed each iteration

→ suppose this would still hold for our poly-many configurations (maybe different α)

→ then the configuration we choose (fewest edges) satisfies $|E_{i+1}| \leq \mathbb{E}[|E_{i+1}|]$

⇒ the deterministic parallel alg still converges in $O(\log n)$ iterations

↳ but with poly-times more processors

- How did we prove (*)? Using Lemma [1] and lemma [2]

→ but the only place where we assumed something about the probability space of events A_v was in the proof of lemma [1] where we did:

① $P[\exists w \in N(v): A_w] = 1 - P[\forall w \in N(v): \neg A_w] = 1 - \prod_{w \in N(v)} P[\neg A_w] \geq 2\alpha \approx 0.153$ (assumes full ind)

② $\forall x \in N(v): P[A_x \mid A_w] = P[A_x]$ ← assumes pairwise independence ... A_x, A_w

Plan: Show that if A_v are pairwise ind. then $P[\exists w \in N(v): A_w] \geq \frac{5}{36} \approx 0.139$

$\Rightarrow$ if we manage to select a poly-size subset of the configurations of X_v such that $P[A_v]$ don't change and A_v remain pairwise ind., then deterministic derandomization will still converge logarithmically, only this time with $\alpha \approx 0.069$ instead of $\alpha \approx 0.077$

Problems:

(1) Prove that pair-wise independence is enough

(2) Describe how to select poly-size subset of the configurations with pw ind.

1) Proving that pair-wise independence is enough

Lemma: If A_v are pair-wise ind. and v is good, then $P[\exists w \in N(v) : A_w] \geq \frac{5}{36}$

Proof: Denote $D := \{w \in N(v) \mid d_w \leq d_v\}$, since v is good: $|D| \geq \frac{1}{3} d_v$

$\rightarrow$ denote $p_v := P[A_v] = \frac{1}{2d_v}$

a) if $\exists w \in N(v)$: $p_w \geq \frac{1}{6} = \frac{6}{36} > \frac{5}{36}$, we win

b) otherwise, we show that the vertices from D together have $p \geq \frac{1}{6}$:

$$\sum_{w \in D} p_w = \sum_{w \in D} \frac{1}{2d_w} \geq \sum_{w \in D} \frac{1}{2d_v} = |D| \cdot \frac{1}{2d_v} \geq \frac{1}{3} d_v \cdot \frac{1}{2d_v} = \underline{\underline{\frac{1}{6}}}$$

since $\forall w \in D$ has $p_w < \frac{1}{6}$ and $\sum_{w \in D} p_w \geq \frac{1}{6}$, there $\exists D' \subseteq D$ s.t.

(*) $\frac{1}{6} \leq \sum_{w \in D'} p_w \leq \frac{2}{6} = \frac{1}{3}$ ... just remove vertices until $\leq \frac{2}{6}$

claim: $P[\exists w \in D' : A_w] = P\left[\bigcup_{w \in D'} A_w\right] \geq \frac{5}{36}$ $\Rightarrow$ we win

$\hookrightarrow$ we can estimate $P[U]$ using the inclusion-exclusion principle

- simple case: $|D'| = 2$, $D' = \{w, \overline{w}\}$, then

$$P[w \cup \overline{w}] = \underbrace{P[w]}_{p_w} + \underbrace{P[\overline{w}]}_{p_{\overline{w}}} - \underbrace{P[w \cap \overline{w}]}_{p_w \cdot p_{\overline{w}}} = \sum_{D'} p_w - p_w p_{\overline{w}} \geq \frac{1}{6} - \frac{1}{36} = \underline{\underline{\frac{5}{36}}}$$

($\geq \frac{1}{6}$ (*), $< \frac{1}{6}$; pairwise independence)

- general case: $|D'| > 2$, but the same argumentation: / inclusion-exclusion

$$P\left[\bigcup_{w \in D'} p_w\right] \geq \sum_{D'} P[A_w] - \frac{1}{2} \sum_{w \neq \overline{w} \in D'} P[A_w \cap A_{\overline{w}}] = \sum_{D'} p_w - \frac{1}{2} \sum_{w \neq \overline{w} \in D'} p_w \cdot p_{\overline{w}}$$

$\hookrightarrow$ we count each intersection twice $\Rightarrow$ need $\frac{1}{2}$

$$\geq \sum_{w \in D'} p_w - \frac{1}{2} \sum_{w, \overline{w} \in D'} p_w \cdot p_{\overline{w}} = \underbrace{\sum_{D'} p_w}_{\geq \frac{1}{6}} \left(1 - \frac{1}{2} \underbrace{\sum_{D'} p_{\overline{w}}}_{\leq \frac{1}{3}}\right) \geq \frac{1}{6}\left(1 - \frac{1}{2} \cdot \frac{1}{3}\right) = \frac{1}{6} \cdot \frac{5}{6} = \underline{\underline{\frac{5}{36}}}$$

$\hookrightarrow$ can be $=$ (*)

2) Constructing a pair-wise independent subset ↗ 0/1

Simple example: Events $A_1, \dots, A_n$ where A_i = coin flip, $P[A_i] = \frac{1}{2}$

↳ our newly created events A_r have probabilities $P[A_r] = \frac{1}{2^{d_r}}$

→ consider 4 coinflips A_1, A_2, A_3, A_4 ⇝ $2^4 = 16$ configurations in total

①	②	③	④
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

subset of 8 out of 16 configurations, each with $P = \frac{1}{8}$

notice that still $\forall i : P[A_i] = \frac{1}{2}$

and A_i are pair wise independent

$$P[A_i \cap A_j] = \frac{1}{4} \quad , \quad P[A_i] \cdot P[A_j] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

→ we will devise a general approach using systems of hash functions

HASH FUNCTIONS

Def: Let U be a finite universe of keys (some finite field) and let V be a finite set of hash values (some finite field) with $|V| = n$.
A system $\mathcal{H}$ of hash functions $h \in \mathcal{H}$, $h : U \to V$ is called

① **2-universal** $= \forall x_1 \neq x_2 \in U : P_{h \in \mathcal{H}}[h(x_1) = h(x_2)] \leq \frac{1}{n}$

↳ the chance of x_1 and x_2 colliding is not worse than if hash values were completely random

② **strongly 2-universal** $:\equiv (\forall x_1 \neq x_2 \in U)(\forall y_1, y_2 \in V) : P_{h \in \mathcal{H}}\left[\begin{matrix} h(x_1) = y_1 \\ h(x_2) = y_2 \end{matrix}\right] = \frac{1}{n^2}$

Lemma: strongly 2-universal ⇒ 2-universal

Proof: $P_{h \in \mathcal{H}}[h(x_1) = h(x_2)] = \sum_{y \in V} P[h(x_1) = y \wedge h(x_2) = y] = \sum_{y \in V} \frac{1}{n^2} = n \cdot \frac{1}{n^2} = \frac{1}{n}$ ∎

Lemma: strongly 2-universal ⇒ **uniform** … $(\forall x \in U)(\forall y \in V) : P[h(x) = y] = \frac{1}{n}$

Proof: Choose any $\bar{x} \neq x$: $P[h(x) = y] = \sum_{z \in V} P[h(x) = y \wedge h(\bar{x}) = z] = \sum_{z \in V} \frac{1}{n^2} = \frac{1}{n}$ ∎

Theorem: strongly 2-universal ⇒ $P_h[h(x_1) = y_1 \wedge h(x_2) = y_2] = P_h[h(x_1) = y_1] \cdot P_h[h(x_2) = y_2]$

Proof: $P_h[h(x_1) = y_1 \wedge h(x_2) = y_2] = \frac{1}{n^2}$

$P_h[h(x_1) = y_1] \cdot P[h(x_2) = y_2] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$ ∎

Since $h \in \mathcal{H}$ uniformly randomly, $\forall x \in U$ $h(x) \in V$ is a random variable

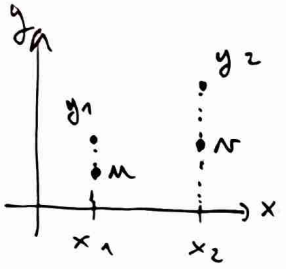
Theorem: $\forall x_1 \neq x_2 :$ $h(x_1)$, $h(x_2)$ PAIRWISE INDEPENDENT

Theorem: Strongly 2-universal $\Rightarrow \forall x_1 \neq x_2 \in U \quad \forall y_1, y_2 \in V$:

$$P_{h \in H}\left[h(x_1) \leq y_1 \wedge h(x_2) \leq y_2\right] = P_h\left[h(x_1) \leq y_1\right] \cdot P_h\left[h(x_2) \leq y_2\right]$$

Proof: Since the hash functions are uniform, we have

$$P_h[h(x_1) \leq y_1] = \frac{y_1}{n} \quad \text{and} \quad P_h[h(x_2) \leq y_2] = \frac{y_2}{n} \Rightarrow RHS = \frac{y_1 \cdot y_2}{n^2}$$



$$LHS = \sum_{m \leq y_1} \sum_{n \leq y_2} P_h[h(x_1) = m \wedge h(x_2) = n]$$

$$= \sum_{m \leq y_1} \sum_{n \leq y_2} \frac{1}{n^2} = \sum_{m \leq y_1} \frac{y_2}{n^2} = \frac{y_1 y_2}{n^2}$$

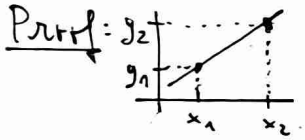
Constructing a strongly 2-universal system of hash functions

Let K be a finite field of size $|K| = n$ and consider

$$\mathcal{H} = \left\{ h : K \to K,\ h(x) = a \cdot x + b \mid a, b \in K \right\} \quad \ldots \text{all linear functions}$$

Theorem: $\mathcal{H}$ is strongly 2-universal and $|\mathcal{H}| = |K|^2 = n^2$ ← obviously

Proof:



$$P_h[h(x_1) = y_1 \ \&\ h(x_2) = y_2] = \frac{1}{|K|^2} = \frac{1}{n^2}$$

$\rightarrow \exists!$ linear function going through these points

Constructing a pairwise independent subset

We have: Events $A_0, A_1, \ldots, A_{n-1}$ with probabilities $p_0, p_2, \ldots, p_{n-1}$

↳ A_i are fully independent, but size of prob. space is exponential in n

We want: Events $H_0, H_1, \ldots, H_{n-1}$ with probabilities $p_0, p_2, \ldots, p_{n-1}$ (close enough)

↳ H_i pair-wise ind. and size of prob space only polynomial in n

→ take system $\mathcal{H}$ described above, $h : K \to K$, where $N := |K| \geq n$

- uniformly randomly choose $h \in \mathcal{H}$ ↳ e.g. $K = \mathbb{Z}_p$ where p is the first prime $\geq n$
- define random variables $\bar{h}_0, \ldots, \bar{h}_{n-1} : \Omega \to K$ as $\bar{h}_i = h(i)$

→ since $\mathcal{H}$ is strongly 2-universal, h_i are pairwise independent

- define events $H_0, \ldots, H_{n-1}$ as H_i = event that $\bar{h}_i \leq p_i N$

→ since $\bar{h}_i$ are uniform, $P[H_i] = \frac{1}{N} \cdot \lfloor p_i \cdot N \rfloor \approx p_i$

Example: $N = 10$, $p = 0.78$
$H_i : \bar{h}_i \leq 7.8$
$P[\leq 7.8] = P[\leq 7] = \frac{7}{10}$

H_i are independent thanks to Theorem ✳

→ size of prof space:

	$\overline{h_0}$	$\overline{h_1}$	...	$\overline{h_{N-1}}$
h_1	0	1	...	5
h_2	4	2	...	1
h_3	8	10	...	11
⋮	⋮	⋮		⋮
h_{N^2}	2	1	...	4

There are only $|\mathcal{H}| = N^2$ functions $h \in \mathcal{H}$, so only N^2 many configurations of variables $\overline{h_0}, \dots, \overline{h_{N-1}}$

Derandomized PARA-MIS

1. before running the algorithm construct $\mathcal{H}$ over K with $|K| = N \geq |V|$ (as described above; field ↖, # vertices ↑)
2. instead of running step ④ (marking $v \in V$ with prob $= \frac{1}{2d_v}$), do:
3. label the remaining vertices as $v_0, v_1, \dots, v_k$
4. for $\forall h \in \mathcal{H}$ calculate $h(0), h(1), \dots, h(k-1)$ and
5. for $\forall i < k$: mark v_i if $h(i) \leq N \cdot \frac{1}{2d_{v_i}}$ and
6. perform steps ⑤ (resolving collisions) and ⑥ (updating graph)
7. pick the best result = removed the most edges

Theorem: MIS can be solved deterministically using $O(n^4)$ processors in time $O(\log n)$.

Proof: As analyzed before, the above alg. terminates after $\log$ many iterations

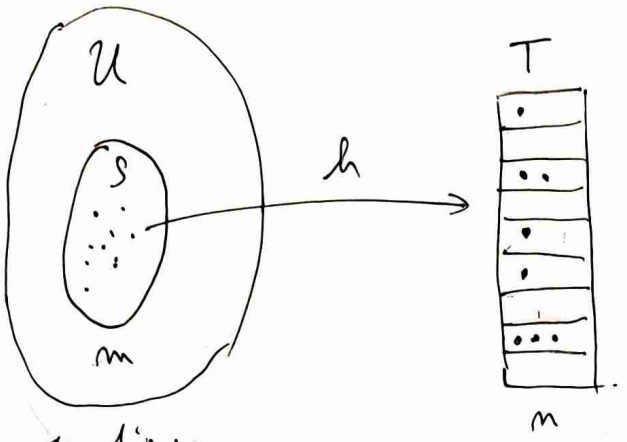
↳ processors: one for each edge and vertex for each $h \in \mathcal{H}$

$\Rightarrow (n+m) \cdot N^2 \in O(n^2) \cdot N^2$

→ fact: if p is a prime, then the next prime $q > p$ satisfies $p < q < 2p$

⇒ we can choose the field as $\mathbb{Z}_p$ where $p < 2n \Rightarrow N^2 = p^2 \in O(n^2)$ ∎

DYNAMIC DICTIONARY



- big universe U
- elements we want to store $S \subseteq U$
  - ↳ $|S| = m$
- table T, in each index a linked list with elements $x \in S$
  - ↳ $|T| = n$

→ operations:

$\mathbb{E}[\text{insert}] \in O(1)$
$\mathbb{E}[\text{delete}] \in O(1)$
$\mathbb{E}[\text{find}] \in O(1)$

Suppose we have a system of hash functions
$\mathcal{H}$ ... $h \in \mathcal{H}$: $h: S \to T$

Goal: minimize # collisions

Theorem: Let $\mathcal{H}$ be 2-universal and $n \in O(m)$ ... for example $n \leq 10m$
Suppose we choose $h \in \mathcal{H}$ uniformly randomly when the dictionary is created and then hash only using h. Then $\mathbb{E}[\text{operation}] \in O(1)$.

Proof: We use values $h(x)$ as indices into T.

Denote $n_i := \#x \in S : h(x) = i$ ... #collisions at index i

Want: $\forall x \in S : \mathbb{E}_h[n_{h(x)}] \in O(1)$

→ we will count # collisions of x. Let $C_y = \begin{cases} 1, & h(y) = h(x) \\ 0, & \text{otherwise} \end{cases}$

Since $\mathcal{H}$ is 2 universal : $\mathbb{E}[C_y] = P_h[h(y) = h(x)] \leq \frac{1}{n}$ ... $y \neq x$

$$\Rightarrow \mathbb{E}[n_{h(x)}] = \underbrace{1}_{\text{element } x} + \sum_{y \neq x} \mathbb{E}[C_y] \leq 1 + \sum_{y \neq x} \frac{1}{n} = 1 + \frac{m-1}{n} \in O(1)$$ ∎

Example: Let M and N be finite fields, we want a system of hash functions
$\mathcal{H} \subseteq \{f \mid f: M \to N\}$ ... what to do if $|M| \gg |N|$?

⇒ Take $\overline{\mathcal{H}} \subseteq \{f \mid f: M \to M\}$ and calculate values modulo N

Fact: If $\overline{\mathcal{H}}$ is strongly 2-universal ⇒ $\mathcal{H}$ is 2-universal
↳ if also $N \mid M$, then $\mathcal{H}$ is strongly 2-universal

Corollary: Fields of sizes 2^n are nice ⇒ Galois Field (2^n)

STATIC DICTIONARY

→ operations:
- construct $\in O(\text{poly}(|S|))$
- find $\in O(1)$

! space $\in O(|S|)$

→ S = everything to store
→ we want a static look-up DS
→ no modifying
→ find: worst case $O(1)$, not average-case

Notation: $m := |S|$, table T of size $n := |T| \in O(m)$ and $n \geq m$

Strategy: Similar to dynamic dictionary
- $\mathcal{H}$ ... 2-universal system of hash functions $h: S \to T$
- pick random $h \in \mathcal{H}$ and map the elements from S to T

? How to handle collisions ? ... linked list could be worse than $O(1)$

$C := \left|\left\{\{x_1, x_2\} \in \binom{S}{2} \mid x_1 \neq x_2 \;\&\; h(x_1) = h(x_2)\right\}\right|$ ... # pairwise collisions

Lemma [1]: The average number of samples of $h \in \mathcal{H}$ until $C \leq n$ is ≤ 2.

Proof: $\mathbb{E}_{h \in \mathcal{H}}[C] \leq \binom{m}{2} \cdot \frac{1}{n}$ ... 2-universal $\Rightarrow P_h[h(x_1) = h(x_2)] \leq \frac{1}{n}$

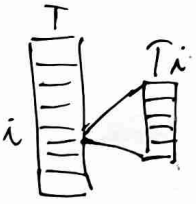
$\leq \binom{n}{2} \cdot \frac{1}{n} = \frac{n-1}{2} < \frac{n}{2}$ ... we used $m \leq n$

→ now we randomly select $h \in \mathcal{H}$:

$P[C > n] < \frac{\mathbb{E}[C]}{n} \leq \frac{n/2}{n} = \frac{1}{2}$ ... Markov inequality

→ so $P[C \leq n] \geq \frac{1}{2}$ $\Rightarrow$ lemma o dělbánci: $\mathbb{E}[\text{trials}] \leq 2$ ∎

→ Assume we have selected $h \in \mathcal{H}$ with $C \leq n$



collisions at index $i \in T$

→ for $\forall i \in T$ calculate $n_i = \#x \in S : h(x) = n_i$

→ to handle collisions, create for $\forall i \in T$ a smaller hash table T_i and system $\mathcal{H}_i \subseteq \{f \mid f: S \to T_i\}$ (2-universal). Handeling collisions in T_i:

define: $C_i := \left|\left\{\{x_1, x_2\} \in \binom{S_i}{2} \mid \begin{matrix} x_1 \neq x_2 \\ h(x_1) = h(x_2) = i \end{matrix} \;\&\; h_i(x_1) = h_i(x_2)\right\}\right|$ ... collisions inside T_i

Lemma [2]: If $|T_i| \geq n_i^2$, then $\mathbb{E}[\#\text{trials of } h_i \in \mathcal{H} \text{ until } C_i = 0] \leq 2$.

Proof: $\mathbb{E}_{h_i}[C_i] \leq \binom{n_i}{2} \cdot \frac{1}{|T_i|} \leq \binom{n_i}{2} \cdot \frac{1}{n_i^2} = \frac{n_i - 1}{2 n_i} < \frac{1}{2}$

$\Rightarrow P_h[C_i \geq 1] \leq \frac{\mathbb{E}[C_i]}{1} < \frac{1}{2}$ $\Rightarrow P[C < 1] = P[C = 0] \geq \frac{1}{2} \Rightarrow \mathbb{E}[\text{trials}] \leq 2$ ∎

Algorithm: constructing static dict. for S. WLOG $|S|$ = power of 2

→ it is nice if tables have the sizes of 2^n

→ $P_2(n) := \min\{2^k \mid 2^k \geq n\}$ ... nearest power of 2

1. $\mathcal{H} \leftarrow$ 2-universal system of $h: S \to T$, where $n := |T| = P_2(|S|)$
2. sample random $h \in \mathcal{H}$ until $C \leq n$ → on average 2 attempts
3. for $\forall i \in T$:
4. calculate $n_i := \#x \in S: h(x) = i$
5. $\mathcal{H}_i \leftarrow$ 2-universal system of $h: S \to T_i$ where $T_i = P_2(n_i^2)$
6. sample random $h_i \in \mathcal{H}_i$ until $C_i = 0$ → on average 2 attempts

find $\in O(1)$ ... every element is behind 2 hashes: $h(x)$, $h_i(x)$
!!
i

$\#$[dict. construction] $\in$ poly

Theorem: This has space complexity $\in O(|S|)$

Proof: We need to remember the hash functions h and h_i for $\forall i \in T$

→ assume that each can be represented by a constant number of elements from the field we represent S with

→ for the sizes of tables

$$\text{space} = |T| + \sum_{i=1}^{n} |T_i| \in O\left(|S| + \underbrace{\sum_{i=1}^{n} n_i^2}_{\hookrightarrow \text{ is this } \in O(|S|)\ ?}\right)$$

$$\#\text{total collisions} = \sum_{i=1}^{n} \underbrace{\#\text{collisions on index } i}_{\leq \binom{n_i}{2}}$$

$$\| \quad C$$

$$\Rightarrow \sum_{i=1}^{n} \binom{n_i}{2} = C \leq n \in O(|S|) \quad \Rightarrow \sum n_i^2 \in O(|S|)$$

∎

TESTING MATRIX MULTIPLICATION

• matrix multiplication

$A, B \in K^{n \times n} \rightsquigarrow A \cdot B = ?$

👁 we need at least $O(n^2)$ operations to read the input

→ trivial alg : $O(n^3)$, Strassen alg : $O(n^{2.81})$

→ best known : $O(n^\omega)$, $\omega < 2.373$

• problem: given $A, B, C \in K^{n \times n}$, test if $A \cdot B = C$

→ idea: pick random $\vec{x} \in K^n$ and test if $A \cdot (B\vec{x}) = C\vec{x}$ ... time $\in O(n^2)$

→ how to pick $\vec{x}$ at random? ... pick $\vec{x} \in \{0,1\}^n$ random 0-1 vector

Algorithm: Freivalds algorithm

1. pick $\vec{x} \in \{0,1\}^n$ uniformly at random
2. $\vec{b} \leftarrow B\vec{x}$, $\vec{a} \leftarrow A\vec{b}$, $\vec{c} \leftarrow C\vec{x}$
3. if $\vec{a} = \vec{c}$ output YES (probably $AB = C$)
   if $\vec{a} \neq \vec{c}$ output NO (definitely $AB \neq C$)

Time complexity

$O(n^2)$ = linear in input size

Theorem: Error probability $\leq \frac{1}{2}$. Hence if run k times, error prob $\leq \frac{1}{2^k}$.

Pf: If $AB = C$, the algorithm always answers correctly

⇒ suppose $AB \neq C \iff AB - C \neq 0$ and denote $D := AB - C$

claim: $P_{\vec{x}}[ABx \neq Cx \mid AB \neq C] = P_{\vec{x}}[D\vec{x} \neq \vec{0}_n \mid D \neq \mathcal{O}_{n,n}] \geq \frac{1}{2}$ ... and we win

→ if D is non-zero, then it has a non-zero row $\vec{r} \in K^n$

→ if we show that $P_{\vec{x}}[r^T x \neq 0 \mid r^T \neq \vec{0}_n] \geq \frac{1}{2}$, we win

$r^T x = \sum_i r_i x_i$, WLOG let the non-zero coordinate be the last one: $r_n \neq 0$

→ consider two vectors
$\vec{x}_0 = (x_1, x_2, \ldots, x_{n-1}, 0)$
$\vec{x}_1 = (x_1, x_2, \ldots, x_{n-1}, 1)$

👁 $r^T x_0$ and $r^T x_1$ differ exactly by r_n
⇒ at most one of $r^T x_0$, $r^T x_1$ can be $= 0$

⇒ if we try all 2^n combinations of $\vec{x} \in \{0,1\}^n$, then at most half of them we get $r^T x = 0$ ⇒ $P_{\vec{x}}[r^T x \neq 0] \geq \frac{1}{2}$

▨

TESTING POLYNOMIAL IDENTITY

$p(x_1, \dots, x_m)$ = polynomial in variables $x_1, \dots x_m$ e.g. $x_1 x_2^2 - 3x_3^3 \cdot x_5$ (monomials)

Problem: given $p(x_1, \dots, x_m)$ test if it is identically equal to 0, that is if all monomials in p have coefficient $= 0$

Fact: If $p(x) \not\equiv 0$ then #roots of $p \le$ degree of p ... over any field K

Idea: If we evaluate $p(x)$ at points $S \subseteq K$, $|S| \gg$ degree, then very small chance that $\forall x \in S: p(x) = 0$

$\Rightarrow$ the field K should be large enough

Notation: $d :=$ total degree of $p(x_1, \dots, x_m)$. E.g.: $d(x_1 x_2^2 + x_3^2) = 3$

Lemma: Let $p(x_1, \dots, x_m)$ be a polynomial over K with total degree d, $p \not\equiv 0$.

$$\forall S \subseteq K: \quad P_{\vec{x} \in S^m}[p(\vec{x}) = 0] \le \frac{d}{|S|}$$

Alg: Take $S \subseteq K$, $|S| > d$
→ pick $x \in S$: if $p(x) = 0$ output ZERO otherwise NONZERO

Thm: Error prob $\le \frac{d}{|S|}$

Corollary: If $|S| \ge 2d$, after running k times: Error prob $\le \left(\frac{d}{|S|}\right)^k \le 2^{-k}$

Proof: In dimension one ($n = 1$) we have

$$P_{x \in S}[p(x) = 0] = \frac{\#x \in S : p(x) = 0}{|S|} \le \frac{d}{|S|}$$

For $n \ge 2$ prove by induction. Let k be the max. power of x_1 in p and write

$$p(\vec{x}) = x_1^k \cdot A(x_2, \dots, x_m) + B(\vec{x})$$

↳ degree of $x_1 < k$

↳ we collected all terms of $p(\vec{x})$ with x_1^k

Define probabilities:

$$P[p(\vec{x}) = 0] = \underbrace{P[A=0]}_{\alpha} \cdot \underbrace{P[p=0 \mid A=0]}_{\le 1} + \underbrace{P[A \neq 0]}_{\le 1} \cdot \underbrace{P[p=0 \mid A \neq 0]}_{\beta} \le \alpha + \beta \overset{?}{\le} \frac{d}{|S|}$$

- $\alpha = P_{x_2, \dots, x_m \in S}[A(x_2, \dots, x_m) = 0] \le \frac{\deg(A)}{|S|} = \frac{d-k}{|S|}$ ... induction hypothesis for $n = n-1$
- $\beta = P_{x_1, \dots, x_m \in S}[p(\vec{x}) = 0 \mid A(x_2, \dots, x_m) \neq 0]$

→ if $A \neq 0$ and we substitute in the values for $x_2, \dots, x_m$ then $A(x_2, \dots, x_m)$ is just a number and we can treat $p(\vec{x})$ as a single variable polynom $q(x_1)$ with degree $= k$

$\Rightarrow$ we use induction hypothesis for $n = 1$, so $\beta \le \frac{k}{|S|}$

$$\Rightarrow P[p(\vec{x}) = 0] \le \alpha + \beta \le \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}$$

∎

TESTING EXISTENCE OF PERFECT MATCHING

recall: perfect/max matching can be solved in $O(n^3)$ ... Edmond's algorithm

→ suppose we have a bipartite graph $G=(U,V,E)$ with $|U|=|V|=n$

Def: "Adjacency matrix" of this bipartite graph is $A \in \{0,1\}^{n \times n}$

$K_{3,3}$ 

$$A = \begin{array}{c} \\ u_1 \\ u_2 \\ \vdots \\ u_n \\ \in U \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & \dots & v_n \in V \end{array} \\ \begin{pmatrix} 0 & 1 & & \\ 1 & 0 & 1 & \\ & 0 & & \\ & & 1 & \end{pmatrix} \end{array} \qquad A_{u,v} := \begin{cases} 1, & uv \in E \\ 0, & uv \notin E \end{cases}$$

$\begin{pmatrix} 1&1&1 \\ 1&1&1 \\ 1&1&1 \end{pmatrix}$

Perfect matchings of G correspond to sets of 1s in A, where there is exactly one 1 in each row and column

such sets correspond to permutations $\Rightarrow$ PM $\sim$ permutations

Recall: $\det(A) = \sum_{\pi \in S_n} (-1)^{\text{sign}(\pi)} \prod_{i=1}^{n} a_{i,\pi(i)}$, S_n = group of permutations on $[n]$

$\Rightarrow$ each PM contributes +1 or -1 to $\det(A)$ based on the sign of the perm.

$\Rightarrow$ if we removed the sign, we would get # perfect matchings

Def: Edmond's matrix of this bipartite G is an $n \times n$ matrix B

$$b_{uv} = \begin{cases} x_{uv}, & uv \in E \\ 0, & uv \in \bar{E} \end{cases}$$

→ x_{uv} variables instead of 1s

$\Rightarrow \det(B)$ = polynomial

Theorem: The polynomial $\det(B) \equiv 0$ $\Leftrightarrow$ $G=(U,V,E)$ has no PM

Proof: $\det(B) = \sum_{\pi: U \to V} (\pm 1) \prod_{u \in U} b_{u,\pi(u)}$ ← good permutations

$\Leftarrow$: PMs of G $\sim$ permutations π where $\forall b_{u\pi(u)} \neq 0$

↳ if # PMs $= 0$ $\Rightarrow$ $\det \equiv 0$

$\Rightarrow$: could it happen that $\det(B) \equiv 0$ because some ± 1 canceled?

not because each good permutation $\prod_{u \in U} x_{u\pi(u)}$ is a unique monomial $\Rightarrow$ they cannot cancel ∎

Fact: rank(B) = size of the max matching in G

Algorithm: for testing if a bipartite graph has PM

IN: bipartite $G=(U,V,E)$ with $|U|=|V|=n$

1. create Edmond's matrix B and calculate $\det(B)$
2. test if $\det(B)\equiv 0$ using the previous prob. alg
3. if $\det(B)\equiv 0$, output NO, otherwise YES

- select field K with $|K|\gg n$,
- $S\subseteq K$, $|S|\gg n$
- $(x_{ij})_{n\times n} \leftarrow$ random $x\in S$
- substitute into B
- evaluate $\det(B)$

Correctness: After running step ② k-times, Error rate $\leq \frac{1}{2^k}$

↳ see the alg. for polynomial identity testing

Time complexity:

- we can do this deterministically in $O(n^3)$ ... Edmond's alg

👁 this alg is as fast as we can calculate $\det(B)$

Fact: We can calculate det as fast as we can multiply matrices

$\Rightarrow$ exist $O(n^\omega)$, $\omega<2.373$ algorithm

Fact: There exist fast parallel algs for $\det(A)$, $A\in K^{n\times n}$ where the elements of K can be encoded in k bits

$\rightarrow$ time $\in O(\log^2 n)$, # processors $\in O(n^{3.5}k)$

So $O(n^{3.5})$ processors per bit and time is independent of #bits

Theorem: Using this result and the polynomial identity testing alg:

- fast parallel probabilistic alg. for testing PM in bipartite graphs

❓ What about general graphs? $\rightarrow |V|=n$, $V=\{v_1,\dots,v_n\}$

Def: The Tutte matrix of a graph $G=(V,E)$ is an $n\times n$ matrix B.

$$b_{ij}=\begin{cases} x_{\{i,j\}}, & v_iv_j\in E \ \&\ i<j \\ -x_{\{i,j\}}, & v_iv_j\in E \ \&\ i>j \\ 0, & \text{otherwise}\end{cases}$$

$x_{\{i,j\}}$ are variables

Fact: The polynomial $\det(B)\equiv 0 \iff G$ has no perfect matchings.

Corollary: For general graphs, we can just use Tutte matrix instead of Edmond's matrix and the alg. still works.

FINDING A PERFECT MATCHING

→ focus on the Edmond's matrix of a bipartite graph $G=(U,V,E)$, $|U|=|V|=n$
and on the Tutte matrix of a general graph $G=(V,E)$, $|V|=n$

recall: $\det(\text{matrix}) \equiv 0 \iff G$ has no PM

👁 if $uv \in E$ is contained in all perfect matchings, then the Edmond's / Tutte matrix of the graph $G' = G - uv$ (same set of vertices, remove edge) will have $\det \equiv 0$

Corollary: If G has a single perfect matching, then we can find it by checking if $\det(\text{matrix for } G-uv) \equiv 0$ for $\forall uv \in E$.

Fact: We don't know any fast deterministic parallel algorithms for finding a PM in a general graph. We can do it if the graph has only poly-many PMs, but general graphs may have exp. many (K_n). The hard part is to get the processors synchronized to focus on a single PM.

⇒ we will describe a randomized procedure that will allow us to isolate a single PM with high probability

Def: Given bipartite $G=(U,V,E)$, $|U|=|V|=n$ and a weight function $w: E \to \mathbb{N}$, we let $C \in \mathbb{N}^{n\times n}$ be the matrix with elements:

$$C_{uv} := \begin{cases} 2^{w(uv)}, & uv \in E \\ 0, & uv \notin E \end{cases} \rightsquigarrow \text{Edmonds matrix with } x_{uv} = 2^{w(uv)}$$

Notation: for a set of edges $M \subseteq E$ let $w(M) := \sum_{e\in M} w(e)$

Lemma: If G has a unique minimum-weight PM $M \subseteq E$, then

(1) $W := w(M)$ is the largest W s.t. 2^W divides $\det(C)$

(2) Also $\forall uv \in E$ we have $uv \in M \iff \frac{\det(C^{uv})}{2^{W-w(uv)}}$ is an odd integer,

where C^{uv} is C after removing the row for $u \in U$ and column for $v \in V$.

Proof: $\det(C) = \sum_{\pi: U\to V} (\pm 1) \prod_{u\in U} C_{u\pi(u)}$

→ recall that the nonzero $\prod C_{u\pi(u)} = \prod 2^{w(u\pi(u))} = 2^{\sum w(u\pi(u))} = 2^{w(M_\pi)}$
correspond to PMs $M_\pi := \{u\pi(u) \in E \mid u \in U\}$ with weight $w(M_\pi)$

→ $\det(C)$ is a sum of these powers of 2, some of them might cancel due to ± 1, but if M is the unique min. weight PM, then $2^{w(M)}$ will not cancel:
$\det(C) = 2^{w(M)} + \sum 2^{w(M')}$, but all $w(M') > w(M)$ ⇒ claim (1) holds.

To show (2) let $uv \in E$, we want to show that

$uv \in M \iff \frac{\det(C^{uv})}{2^{W-w(uv)}}$ is an odd integer

$\iff 2^{W-w(uv)}$ is the largest power of 2 dividing $\det(C^{uv})$

$\Rightarrow$: after removing u, v from G, then $M' := M \setminus \{uv\}$ is clearly a unique min-weight PM in $G-u-v$ with $w(M') = W - w(uv)$, use (1)

$\Leftarrow$: we don't know whether $uv \in M$, so $G-u-v$ might not have a unique min. weight PM. If not, all of those min. weight PMs M' with weight $w(M')$ will contribute $\pm 2^{w(M')}$ to $\det(C^{uv})$. They might cancel, or they might not. Whatever the case, notice that if $2^{W'}$ is the largest power of 2 dividing $\det(C^{uv})$, then $G-u-v$ has a PM M' with weight $w(M') \leq W'$.

→ so, since $2^{W-w(uv)}$ is the largest power of 2 dividing $\det(C^{uv})$, there $\exists$ PM M' of $G-u-v$ with $w(M') \leq W - w(uv)$.

$\Rightarrow M'' := M' \cup \{uv\}$ is a PM of G with $w(M'') \leq W$

→ since M is unique min-weight PM, it must be $M'' = M \Rightarrow uv \in M$ ▨

Idea: If we can construct a weight $w: E \to \mathbb{N}$ for any graph G s.t. there is a unique min-weight PM, the algorithm for finding it is clear

→ since there can be exp. many PMs, it would be intuitive to take an exp. long number N, and assign each $e \in E$ a random weight $\leq N$.

→ however, the # processors of the parallel alg for determinant is linear in the #bits of the entries of the matrix $\Rightarrow$ we can't have exp. long numbers

Theorem (Isolating lemma): Let $\mathcal{S} = \{S_1, \dots, S_k\}$ where $S_i \subseteq \{e_1, e_2, \dots, e_m\}$ be an arbitrary set system. Let $R \subseteq \mathbb{N}$ be a set of weights with $|R| = r$, and let $w(e_1), \dots, w(e_2) \in R$ be assigned uniformly ind. randomly. Then

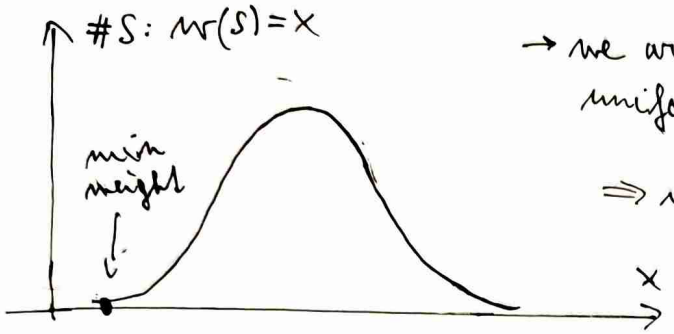
$$P[\text{there is not a unique min-weight } S \in \mathcal{S}] \leq \frac{m}{r}$$

Corollary: Take $\mathcal{S}$ = perfect matchings of G and $\{e_1, \dots, e_m\} = E(G)$.

When $R = \{1, 2, \dots, 2m\}$, we have $P[\text{fail}] \leq \frac{m}{2m} = \frac{1}{2}$

→ this should be counter-intuitive: a linear number of weights is enough, and the probability does not depend on $|\mathcal{S}| = \#$PMs at all

Intuition: The weights $w(S) = \sum_{e \in S} w(e)$ are not uniformly distributed



→ we are essentially summing independent uniform distributions

⇒ it should not be surprising if the result looks something like the normal distribution (central limit theorem)

↳ we are interested in the very tail of this distribution

Proof: for $\forall i \in [m]$ define events

A_i = event that $\exists S, \bar{S} \in \mathcal{S}$ such that $e_i \in S$, $e_i \notin \bar{S}$ and both $S, \bar{S}$ have minimal weight among all $S_j \in \mathcal{S}$

👁 if $S \neq S'$ both have min. weight, then they satisfy at least one A_i ↳ since $S \neq S'$

$\Rightarrow P[\exists S, S' \text{ both with min. weight}] \leq \sum P[A_i]$

claim: $P[A_i] \leq \frac{1}{r}$ $\Rightarrow$ so $P[\dots] \leq m \cdot \frac{1}{r}$ and we are done

$\mathcal{S}_0 := \{S \mid S \in \mathcal{S}, e_i \notin S\}$ ⇝ let $S_0 \in \mathcal{S}_0$ have min weight in $\mathcal{S}_0$

$\mathcal{S}_1 := \{S \setminus \{e_i\} \mid S \in \mathcal{S}, e_i \in S\}$ ⇝ let $S_1 \in \mathcal{S}_1$ have min weight in $\mathcal{S}_1$

→ let $\alpha := w(S_0) - w(S_1)$, note that α could be outside R

👁 if $\alpha = w(e_i)$, then $S := S_0$ and $\bar{S} := S_1 \cup \{e_i\}$ satisfy

$w(S) = w(\bar{S}) = \min_{S' \in \mathcal{S}} w(S')$, and $e_i \in S$, $e_i \in \bar{S}$ $\Rightarrow$ A_i is satisfied

$\Rightarrow P[A_i] \leq P[w(e_i) = \alpha] \leq \frac{1}{r}$ as $w(e_i)$ is chosen randomly from R, and the random variable α depends only on $w(e_j)$, $j \neq i$, which are independent from $w(e_i)$. (Also if $\alpha \notin R$, then $P[w(e_i) = \alpha] = 0$). ∎

Algorithm: PM in bipartite $G=(U,V,E)$

1. $M \leftarrow \emptyset$, independently choose $w(uv) \in \{1,\dots,2|E|\}$ for $\forall uv \in E$ randomly
2. calculate C and $\det(C)$ → C = Edmond's matrix with $x_{uv} = 2^{w(uv)}$
3. find the largest W s.t. 2^W divides $\det(C)$
4. for $\forall uv \in E$ calculate in parallel $d := \det(C^{uv})$, where C^{uv} is C after deleting the row corresponding to $u \in U$ and column to $v \in V$.
5. if $d/2^{W-w(uv)}$ is an odd integer, add uv into M
6. test if M is a matching (in parallel check for each vertex if $\deg_M = 1$)
   ↳ if yes, output M, if no, FAIL.

Theorem: The algorithm finds a PM with probability $\geq \frac{1}{2}$.
⇒ If we run it independently k times, then error rate $\leq \frac{1}{2^k}$

Proof: The correctness of the algorithm follows from the earlier lemma.
The low error rate is thanks to the isolating lemma. ■

Theorem: time $\in O(\log^2 n)$, # processors = poly(n)

Proof: Because the parallel determinat alg has these parameters,
and our choice of weights ensures that the elements of C aren't too big. ■

Finding PM in general graph

→ in a general graph, we replace the Edmond's matrix with the **Tutte matrix**, where $x_{\{i,j\}}$ is replaced by $2^{w(v_i,v_j)}$

Fact: If we use Tutte matrix instead of Edmond's matrix and modify step ⑤

5. if $d/2^{W-2w(uv)}$ is an odd integer, add uv into M

then the alg. works for general graphs.

Note: For Tutte matrix C we mean by C^{uv} the matrix we get from C after deleting the 2 rows and 2 columns corresponding to vertices u, v

↳ if $C \sim G$, then $C^{uv} \sim G-u-v$