Advanced Algorithms

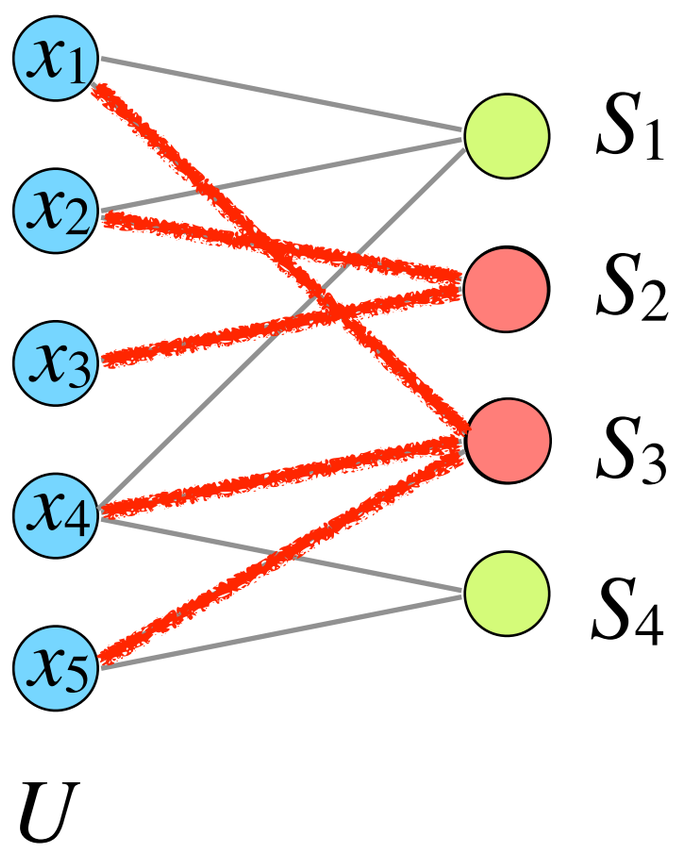
南京大学

尹一通

Set Cover

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

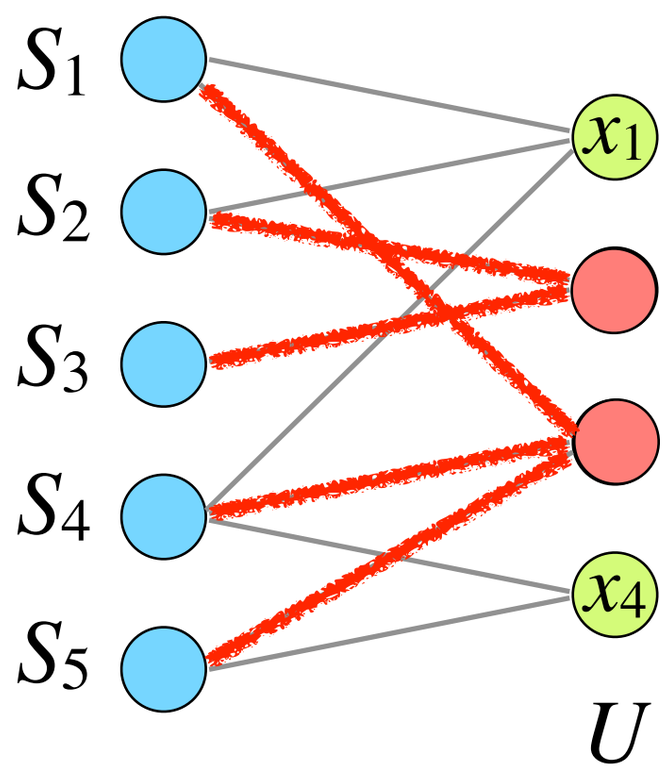
Find the smallest $C \subseteq \{1, 2, \ldots, m\}$ that $\cup_{i \in C} S_i = U$.



Hitting Set

Instance: A number of sets $S_1, S_2, \ldots, S_n \subseteq U$.

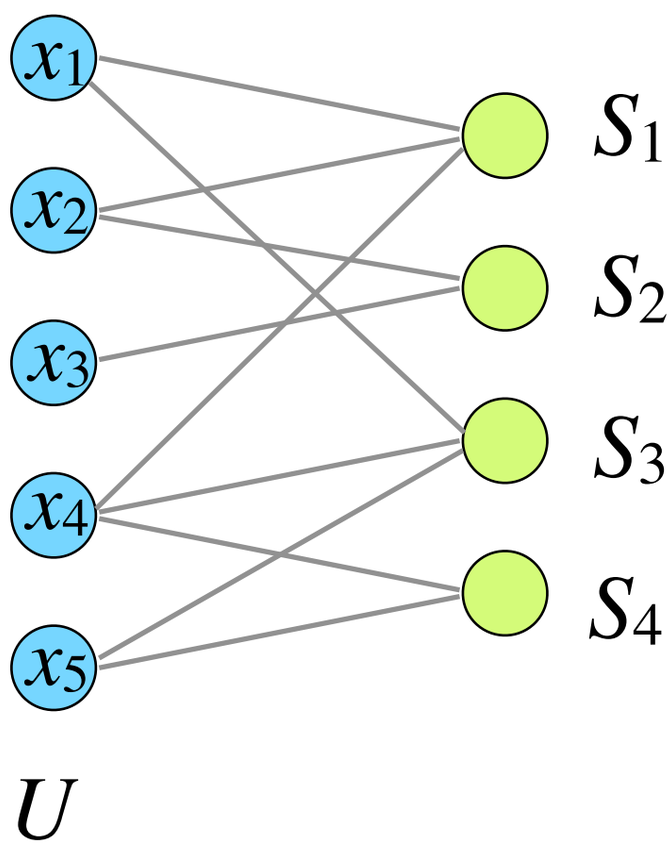
Find the smallest $H \subseteq U$ that $\forall i, S_i \cap H \neq \emptyset$.



Set Cover

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

Find the smallest $C \subseteq \{1, 2, \ldots, m\}$ that $\cup_{i \in C} S_i = U$.



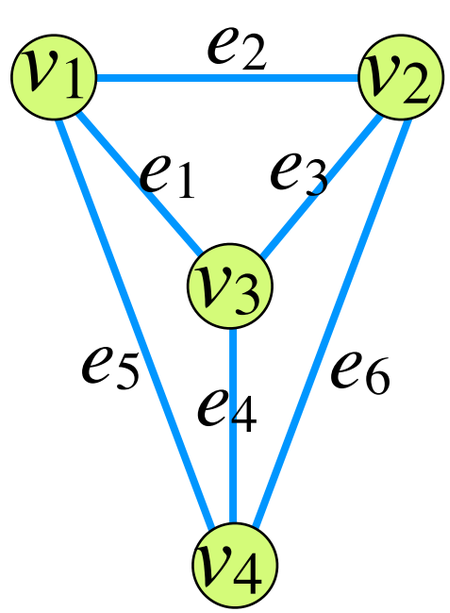
- **NP**-hard
- one of Karp's 21 **NP**-complete problems
- *frequency*: # of sets an element is in

$$frequency(x) = |\{S_i : x \in S_i\}|$$

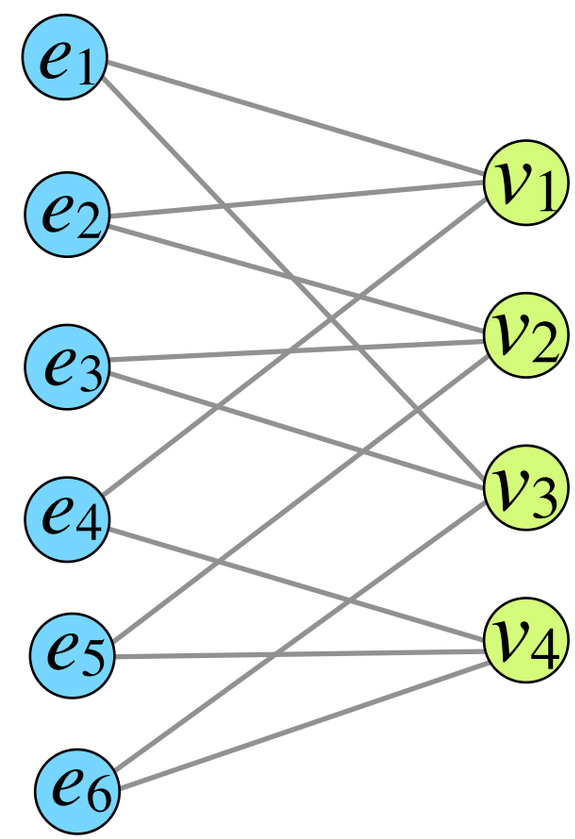
Vertex Cover

Instance: An undirected graph $G(V,E)$

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in C.



incidence graph



instance of set cover with *frequency* =2

Vertex Cover

Instance: An undirected graph $G(V,E)$

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in C.

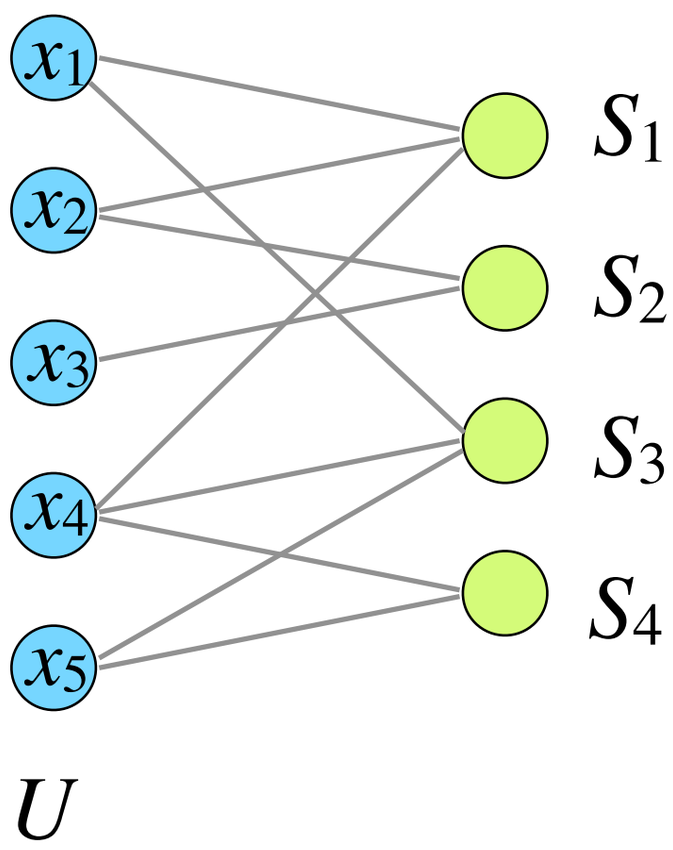
- **NP**-hard
- one of Karp's 21 **NP**-complete problems

VC is **NP**-hard $\Rightarrow$ SC is **NP**-hard

Set Cover

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

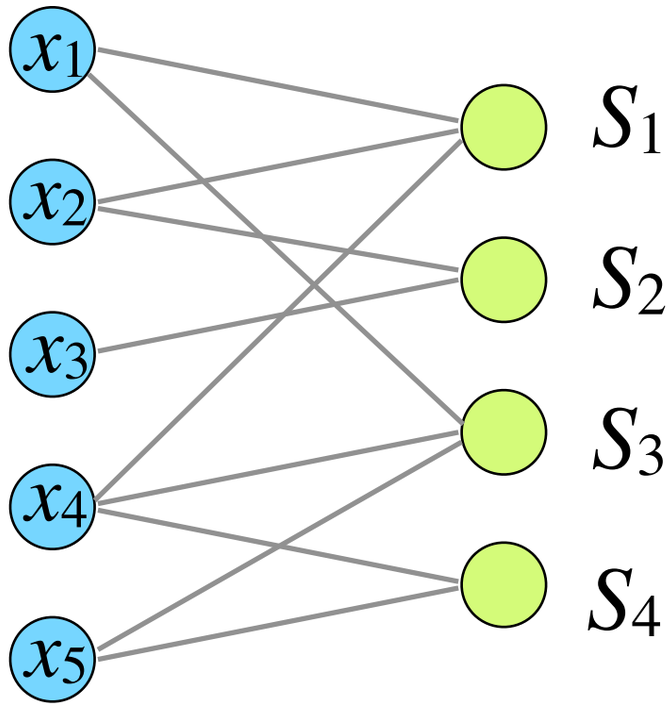
Find the smallest $C \subseteq \{1, 2, \ldots, m\}$ that $\cup_{i \in C} S_i = U$.



GreedyCover

Initially $C = \emptyset$;
while $U \neq \emptyset$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \setminus S_i$;

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.



GreedyCover

Initially $C=\emptyset$;
while $U\neq\emptyset$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \setminus S_i$;

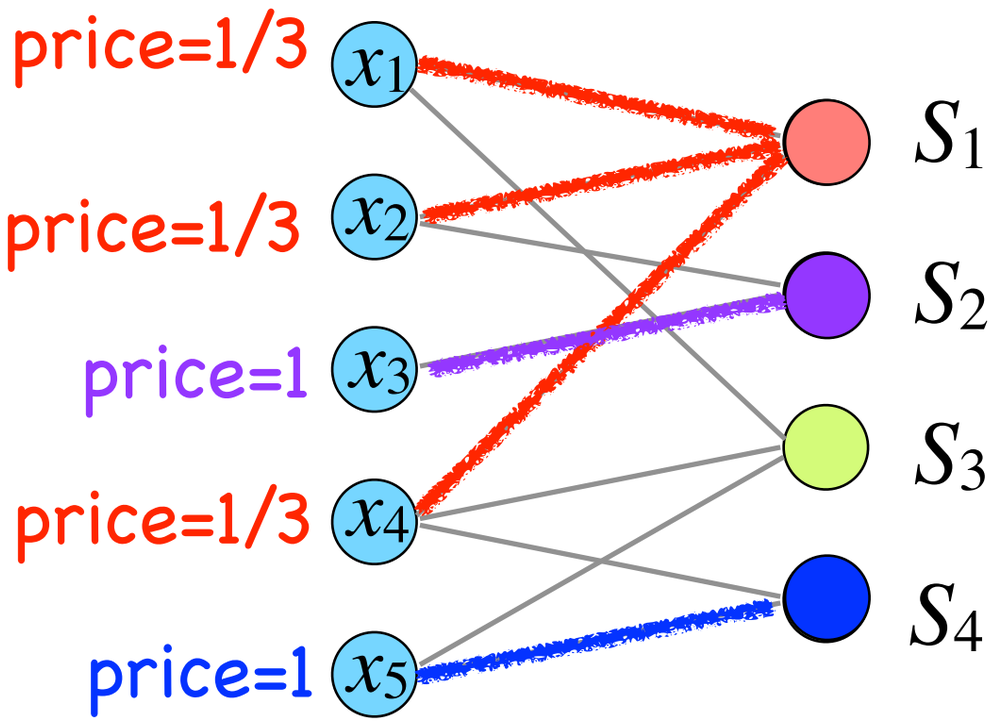
OPT(I): value of minimum set cover of instance I

SOL(I): value of the set cover returned by the **GreedyCover** algorithm on instance I

GreedyCover has *approximation ratio* α if

$$\forall \text{ instance } I, \qquad \frac{\mathrm{SOL}(I)}{\mathrm{OPT}(I)} \leq \alpha$$

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.



GreedyCover

Initially $C=\emptyset$;
while $U\neq\emptyset$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \setminus S_i$; $\forall x \in S_i \cap U$, price(x)=1/$|S_i \cap U|$

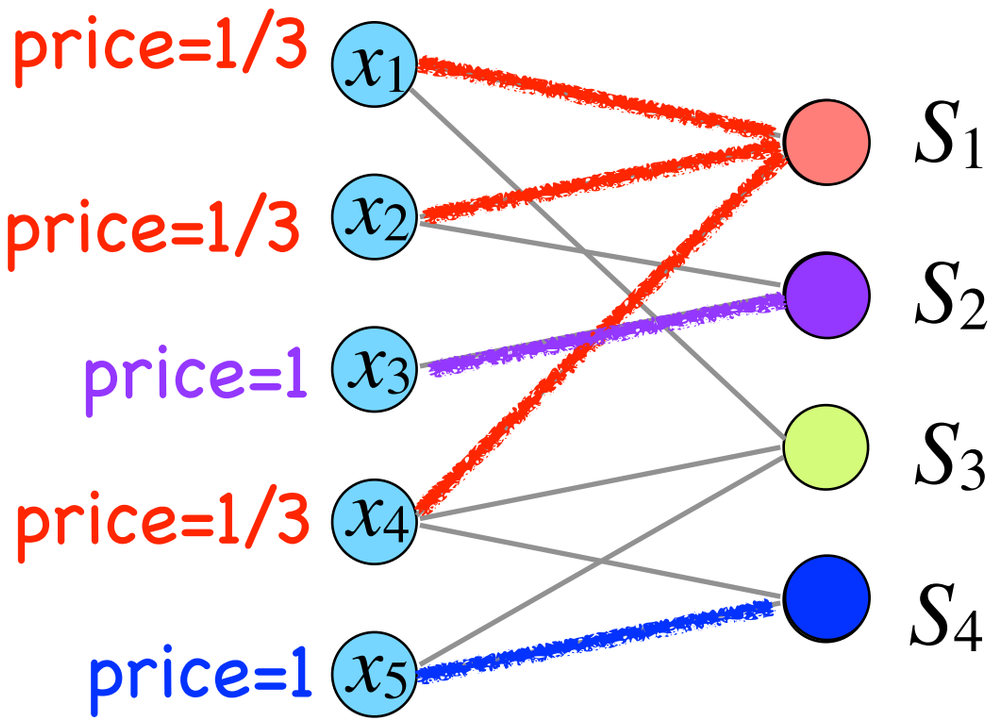
$$|C| = \sum_{x \in U} price(x)$$

enumerate $x_1, x_2, \ldots x_n$ in the order in which they are covered

elements can be *matched* to the sets in OPT cover

$$\exists S_i,\ |S_i| \geq \frac{|U|}{OPT} \quad \Rightarrow \quad \boxed{price(x_1) \leq \frac{OPT}{|U|}}$$

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.



GreedyCover

Initially $C=\emptyset$;
while $U\neq\emptyset$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \setminus S_i$; $\forall x \in S_i$, price(x)=1/$|S_i \cap U|$

$$|C| = \sum_{x \in U} price(x)$$

enumerate $x_1, x_2, \ldots x_n$ in the order in which they are covered

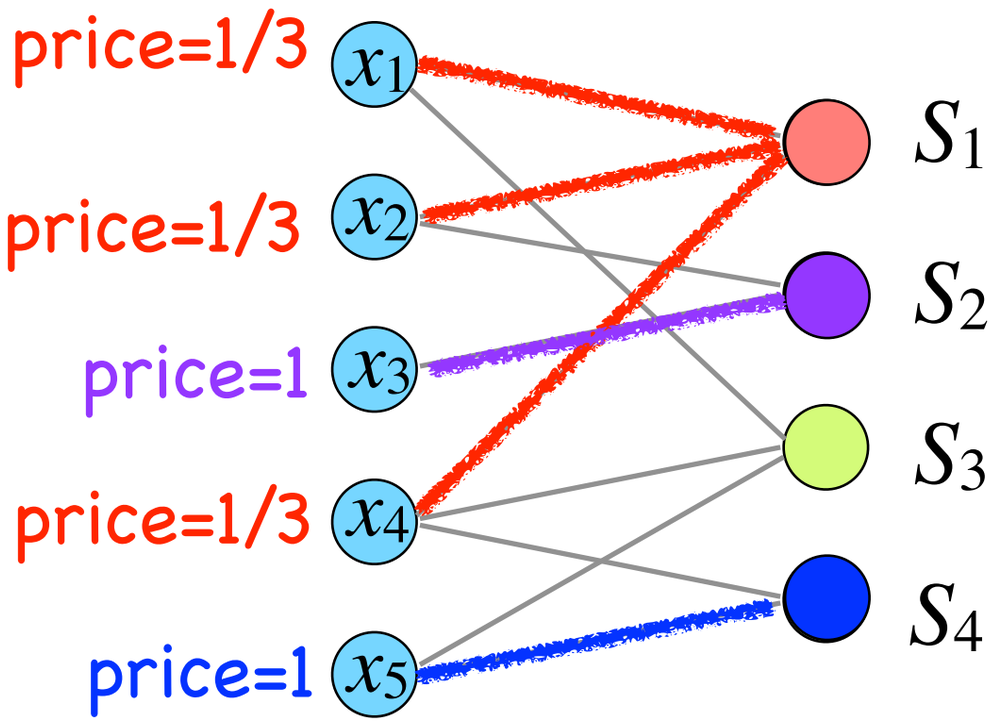
$$price(x_k) \leq \frac{OPT}{n-k+1}$$

consider U_t in iteration t where x_k is covered:

$$|U_t| \geq n\text{-}k\text{+}1$$

all $S_i \cap U_t$ form a set cover instance: $\leq$OPT

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.



GreedyCover

Initially $C=\emptyset$;
while $U\neq\emptyset$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \setminus S_i$; $\forall x \in S_i$, price(x)=1/$|S_i \cap U|$

$$|C| = \sum_{x \in U} price(x) \leq \sum_{k=1}^{n} \frac{OPT}{n-k+1} = H_n \cdot OPT$$

enumerate $x_1, x_2, \ldots x_n$ in the order in which they are covered

$$\boxed{price(x_k) \leq \frac{OPT}{n-k+1}}$$

GreedyCover

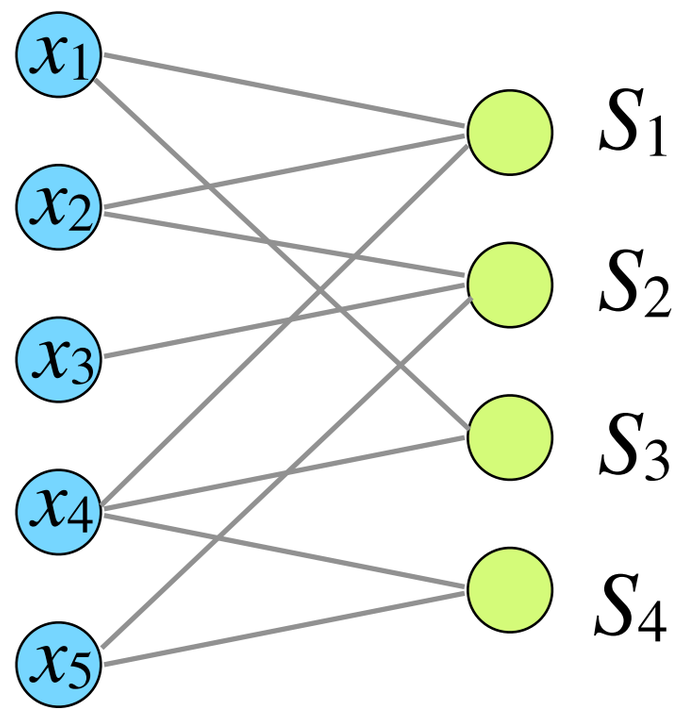
Initially $C=\emptyset$;
while $U\neq\emptyset$ do:
  add i with largest $|S_i\cap U|$ to C;
  $U = U \setminus S_i$;

- *GreedyCover* has approximation ratio $H_n \approx \ln n + O(1)$.
- [Lund, Yannakakis 1994; Feige 1998] There is no poly-time (1-o(1))ln n-approx. algorithm unless **NP** = quasi-poly-time.
- [Ras, Safra 1997] For some c there is no poly-time $c \ln n$-approximation algorithm unless **NP** = **P**.
- [Dinur, Steuer 2014] There is no poly-time (1-o(1))ln n-approximation algorithm unless **NP** = **P**.

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

Primal: $C \subseteq \{1, 2, \ldots, m\}$ that $\cup_{i \in C} S_i = U$.

$\text{OPT}_{\text{primal}} = \min |C|$



Dual: $M \subseteq U$ that $\forall i, |S_i \cap M| \leq 1$.

$\forall C, \forall M$: $|M| \leq |C|$

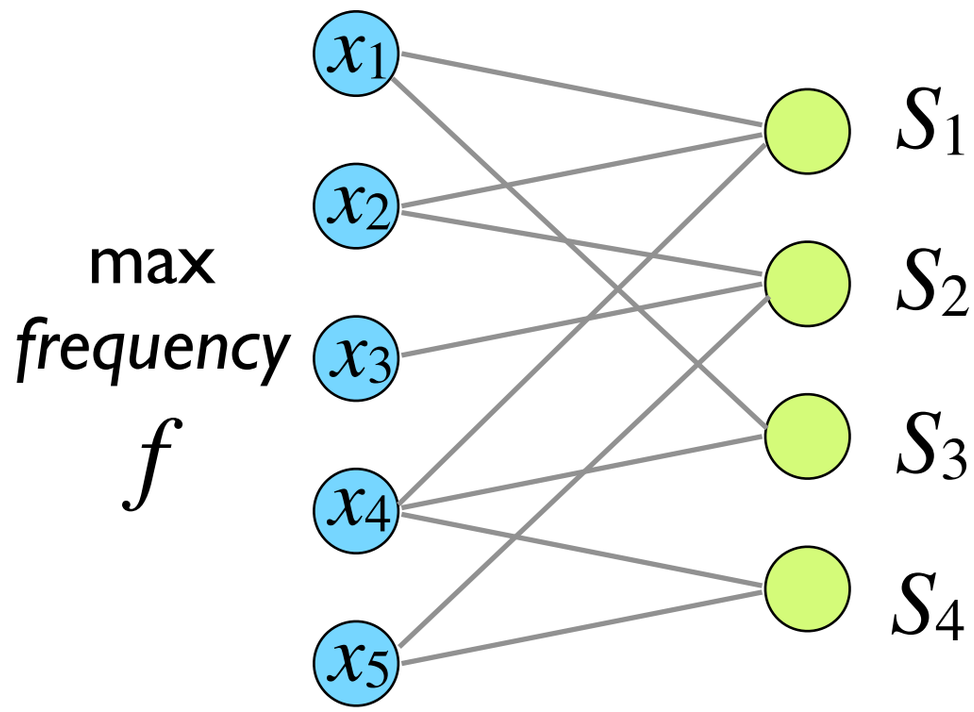
every $x \in M$ must consume a set to cover

$\forall M$: $|M| \leq \text{OPT}_{\text{primal}}$

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

Primal: $C \subseteq \{1, 2, \ldots, m\}$ that $\cup_{i \in C} S_i = U$.

$$\text{OPT}_{\text{primal}} = \min |C|$$



Find a *maximal* M;
return $C = \{i : S_i \cap M \neq \emptyset\}$;

M is *maximal* $\Rightarrow$ C must be a cover

$$|C| \leq f \cdot |M| \leq f \cdot \text{OPT}_{\text{primal}}$$

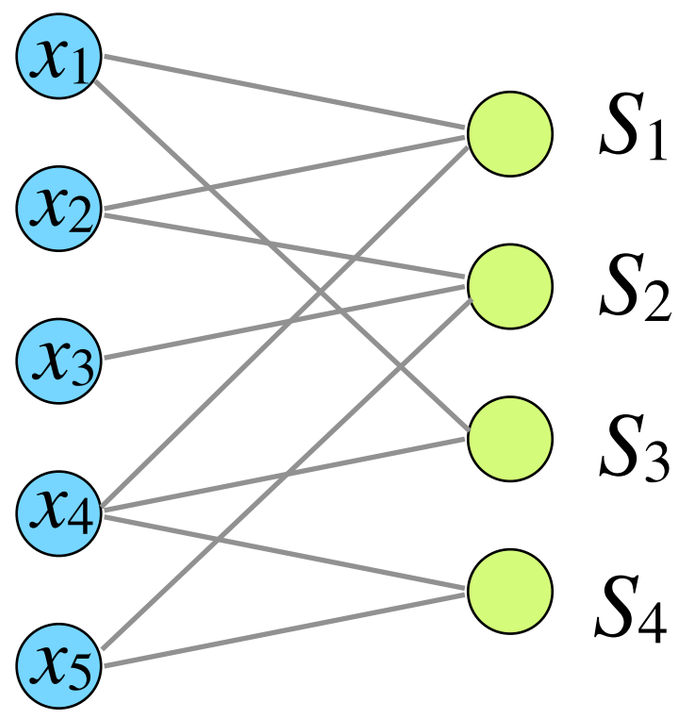
Dual: $M \subseteq U$ that $\forall i, |S_i \cap M| \leq 1$.

$$\forall M: \ |M| \leq \text{OPT}_{\text{primal}}$$

Instance: A number of sets $S_1, S_2, \ldots, S_m \subseteq U$.

Find a *maximal* $M \subseteq U$ that $\forall i, |S_i \cap M| \leq 1$;
return $C = \{i : S_i \cap M \neq \emptyset\}$;

max *frequency* f



Frequency assumption:

$$\forall x \in U, \ |\{i : x \in S_i\}| \leq f$$

$$|C| \leq f \cdot \mathrm{OPT}$$

For vertex cover: This gives a 2-approximation algorithm.

Vertex Cover

Instance: An undirected graph $G(V,E)$

Find the smallest $C \subseteq V$ that every edge has at least one endpoint in C.

a 2-approximation algorithm:

Find a *maximal matching*;
return the *matched* vertices;

- [Dinur, Safra 2005] There is no poly-time <1.36-approximation algorithm unless **NP = P**.
- [Khot, Regev 2008] Assuming the unique games conjecture, there is no poly-time (2-ε)-approximation algorithm.

Computational Complexity

- decision problem f: $\{0,1\}^* \to \{0,1\}$
- formal language $L \subseteq \{0,1\}^*$ $\quad L = \{x: f(x)=1\}$
- poly-time Turing machine (Algorithm) M:

  length of input

  $\forall$ input $x \in \{0,1\}^*$, $M(x)$ terminates in time < $\mathrm{poly}(|x|)$
- $\mathbf{P}, \mathbf{NP}$: classes of formal languages (decision problems)
- $L \in \mathbf{P}$: $\exists$ poly-time TM M *decides* L
  - $x \in L \Rightarrow M(x)$ accepts;
  - $x \notin L \Rightarrow M(x)$ rejects
- $L \in \mathbf{NP}$: $\exists$ poly-time TM M *verifies* L
  - $x \in L \Rightarrow \exists$ *certificate* $y \in \{0,1\}^*$, $M(x,y)$ accepts;
  - $x \notin L \Rightarrow \forall\ y \in \{0,1\}^*$, $M(x,y)$ rejects;

  nondeterministic poly-time TM accepts L

Computational Complexity

- decision problem $f: \{0,1\}^* \to \{0,1\}$
- formal language $L \subseteq \{0,1\}^*$ $\quad L = \{x: f(x)=1\}$
- poly-time Turing machine (Algorithm) M:

  $\forall$ input $x \in \{0,1\}^*$, $M(x)$ terminates in time < $\mathrm{poly}(|x|)$
- $\mathbf{P}, \mathbf{NP}$: classes of formal languages (decision problems)
- $L \in \mathbf{P}$: $\exists$ poly-time TM M *decides* L
  - $x \in L \Rightarrow M(x)$ accepts; • $x \notin L \Rightarrow M(x)$ rejects
- $L \in \mathbf{NP}$: $\exists$ poly-time TM M *verifies* L

$L \in \mathrm{co}\mathbf{NP}$: $\overline{L} \in \mathbf{NP}$ "no"-instances are easy to verify

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathrm{co}\mathbf{NP}$$

Computational Complexity

- decision problem $f: \{0,1\}^* \rightarrow \{0,1\}$
- formal language $L \subseteq \{0,1\}^*$ $\quad L = \{x: f(x)=1\}$
- poly-time (Turing) *reduction* from L to L':

a poly-time TM M that decides L
given accesses to an *oracle* that decides L'

L' is poly-time decidable $\Rightarrow$ L is poly-time decidable
L is hard $\Rightarrow$ L' is hard

" L' is at least as hard as L "

- a problem is **NP**-hard if every $L \in$ **NP** is poly-time reducible to it
- L is **NP**-complete if $L \in$ **NP** and L is **NP**-hard

Optimization

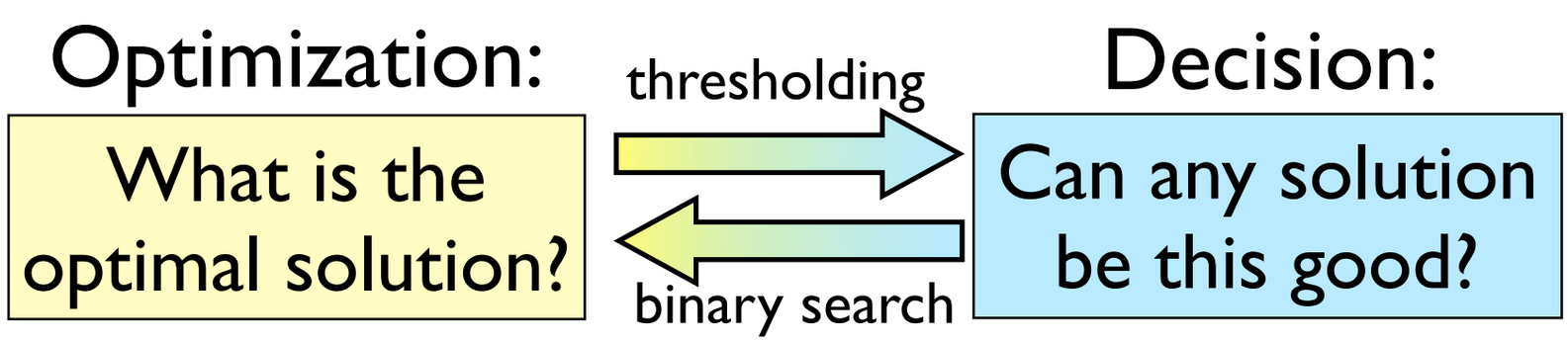
Optimization problem Π: minimization/maximization

- a set D of valid instances (inputs);
- each instance $I \in D$ defines a set of feasible solutions $S(I)$;
- an objective function obj that assigns each instance $I \in D$ and solution $s \in S(I)$ a value.

NP-optimization problem Π:

- feasibility of a solution is poly-time checkable;
- objective function is poly-time computable.

optimal solution is certificate



Approximation

Optimization problem Π: minimization/maximization

- a set D of valid instances (inputs);
- each instance $I \in D$ defines a set of feasible solutions $S(I)$;
- an objective function obj that assigns each instance $I \in D$ and solution $s \in S(I)$ a value.

$\mathrm{OPT}(I)$ = objective value of optimal solution $s^* \in S(I)$ of instance I

- algorithm A: returns a solution $s \in S(I)$ on every instance I

$\mathrm{SOL}_A(I)$ = objective value of the solution $s \in S(I)$ returned by A on instance I

minimization: approximation ratio of algorithm A is α

if $\forall$ instance I : $\dfrac{\mathrm{SOL}_A(I)}{\mathrm{OPT}(I)} \leq \alpha$

Approximation

Optimization problem Π: minimization/maximization

- a set D of valid instances (inputs);
- each instance $I \in D$ defines a set of feasible solutions $S(I)$;
- an objective function obj that assigns each instance $I \in D$ and solution $s \in S(I)$ a value.

$\mathrm{OPT}(I)$ = objective value of optimal solution $s^* \in S(I)$ of instance I

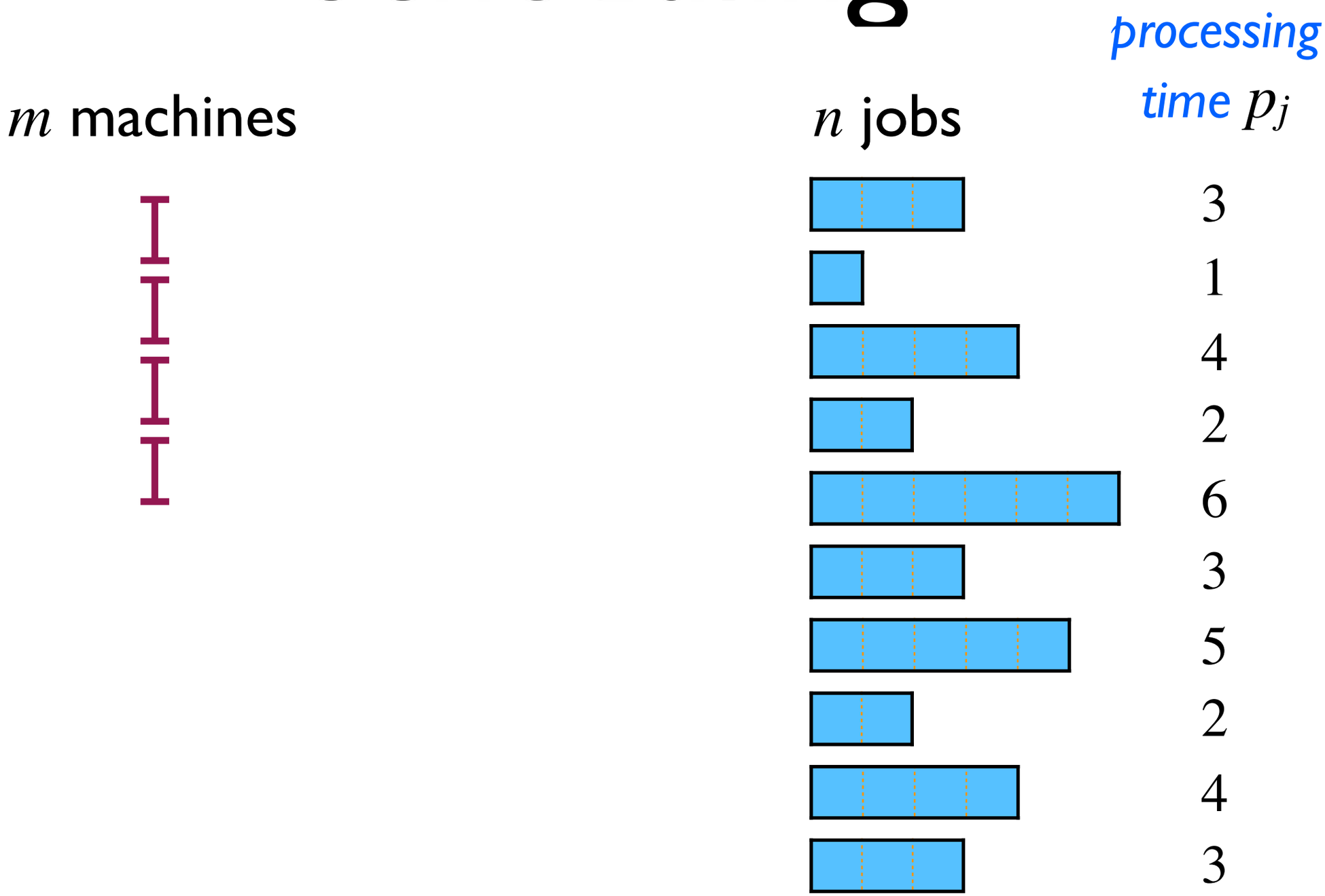
- algorithm A: returns a solution $s \in S(I)$ on every instance I

$\mathrm{SOL}_A(I)$ = objective value of the solution $s \in S(I)$ returned by A on instance I

maximization: approximation ratio of algorithm A is α

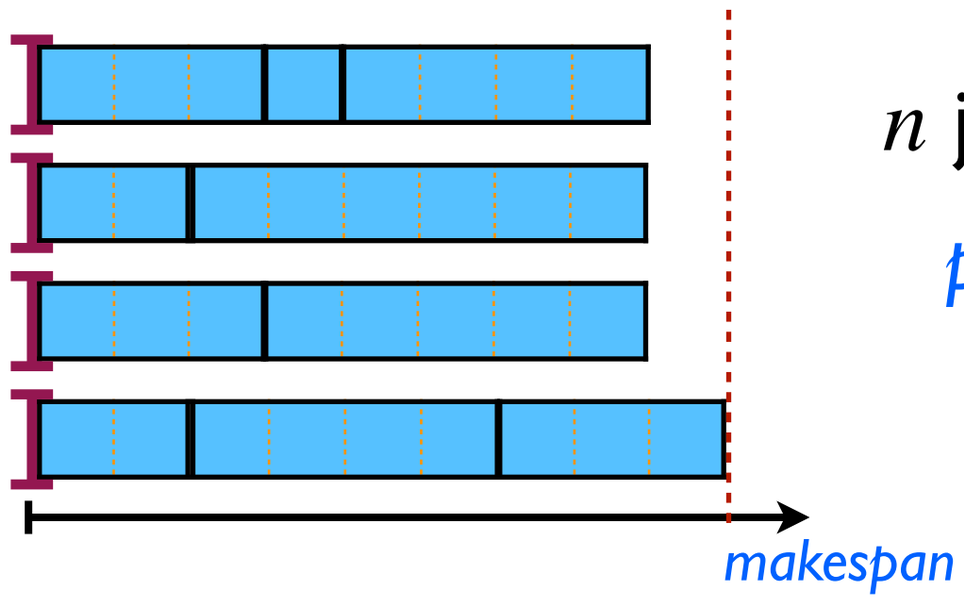
if $\forall$ instance I : $\dfrac{\mathrm{SOL}_A(I)}{\mathrm{OPT}(I)} \geq \alpha$

Scheduling



Scheduling

m machines



n jobs with

processing time p_j

completion time: $$C_i = \sum_{\substack{j\text{: jobs assigned} \\ \text{to machine } i}} p_j$$

makespan: $$C_{\max} = \max_i C_i$$

Instance: n jobs $j=1, 2, \ldots, n$
each with processing time $p_j \in \mathbb{Z}^+$.

Solution: A schedule of n jobs to m machines
that minimizes the *makespan* $C_{\max}$.

"minimum *makespan* on *identical* machines": $\mathrm{P}| \, |C_{\max}$

Graham's "$\alpha|\beta|\gamma$" notation for scheduling

α: machine environment

- 1: a single machine;
- P: m identical machines;
- Q: m machines with different speed s_i, the length of job j on machine i is p_j/s_i;
- R: m unrelated machines, the length of job j on machine i is p_{ij};

β: job characteristics

- r_j: each job has a release time r_j;
- d_j: each job has a deadline d_j;
- pmtn: preemption is allowed;

γ: objective

- $C_{\max}$: *makespan*; $\sum_j C_j$: total completion time; $L_{\max}$: maximum lateness;

Instance: n jobs $j=1, 2, \ldots, n$
each with processing time $p_j \in \mathbb{Z}^+$.

Solution: A schedule of n jobs to m machines
that minimizes the *makespan* $C_{\max}$.

"minimum *makespan* on *identical* machines": $P| \,|C_{\max}$

when $m=2$, the problem can solve the **partition** problem:

Input: n numbers $x_1, x_2, \ldots, x_n \in \mathbb{Z}^+$.

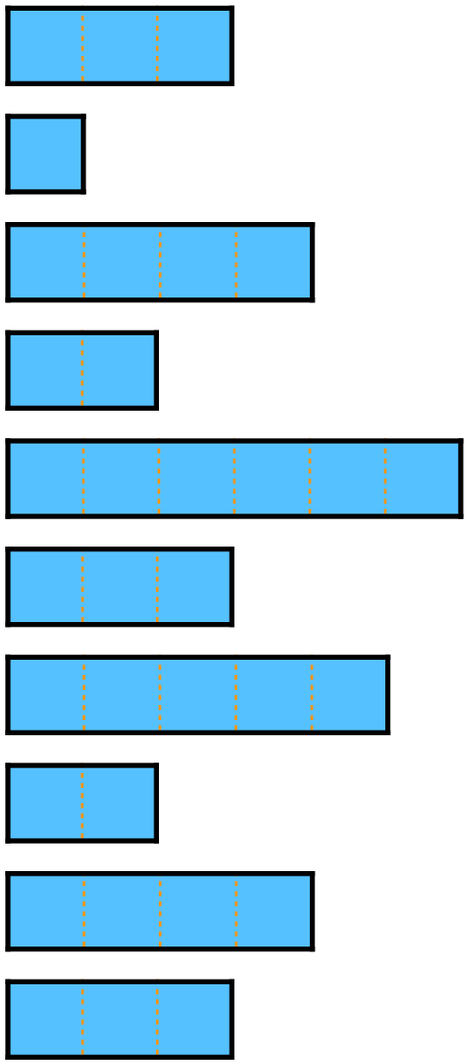
Determine whether $\exists$ a partition of $\{1, 2, \ldots, n\}$
into A and B such that $\sum_{i \in A} x_i = \sum_{i \in B} x_i$.

the **partition** problem is among Karp's 21 **NPC** problems

Graham's *List* Algorithm

m machines

n jobs



***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

n jobs: $p_1, p_2, ..., p_n$; m machines

$$OPT \geq \max_j p_j \qquad\qquad OPT \geq \frac{1}{m}\sum_j p_j$$

for the schedule returned by the list algorithm:

makespan $C_{\max} = C_i \quad \leq 2 \cdot OPT$

the last job assigned to machine i is job l

before job l was assigned, machine i is the least heavily loaded

$$C_i - p_\ell \leq \frac{1}{m}\sum_j p_j \quad \leq OPT$$

$$p_\ell \leq \max_j p_j \quad \leq OPT$$

***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
assign job j to the *current least heavily loaded* machine;

returns a schedule with makespan $C_{\max} \leq (2 - \frac{1}{m}) \cdot OPT$

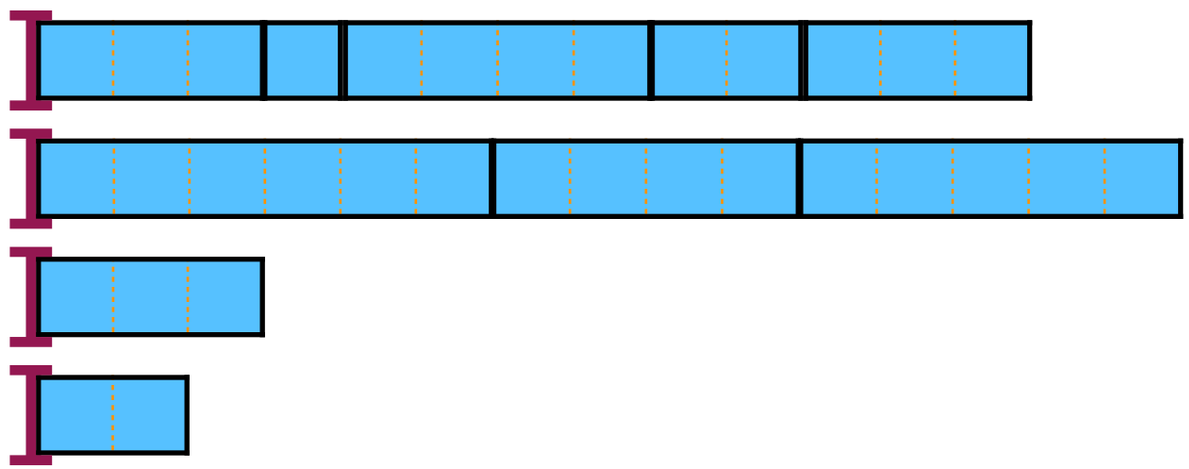
Tight in the worst-case!

$$C_i - p_\ell \leq \frac{1}{m} \sum_{j \neq \ell} p_j$$

$$C_i \leq \frac{1}{m} \sum_j p_j + \left(1 - \frac{1}{m}\right) p_\ell \ \leq (2 - \frac{1}{m}) \cdot OPT$$

Local Search

start with a solution:



locally modify the solution to make improvement until no improvement can be made (local optimum)

Start with an arbitrary schedule;
repeat until no job is reassigned (a local optimum is encountered):
  let l be a job that fished last;
  if $\exists$ machine i s.t. *job l will finish earlier after reassigned* to machine i
    transfer job l to machine i;

Start with an arbitrary schedule;
repeat until no job is reassigned (a local optimum is encountered):
  let l be a job that fished last;
  if ∃ machine i s.t. *job l will finish earlier after reassigned* to machine i
    transfer job l to machine i;

$$OPT \geq \max_j p_j \qquad\qquad OPT \geq \frac{1}{m}\sum_j p_j$$

in a local optimum: suppose makespan $C_{\max} = C_i$

for the job l that finished last

local optimum ⇒ $C_i - p_l$ must be the least heavily loaded

$$C_i - p_\ell \leq \frac{1}{m}\sum_{j\neq\ell} p_j$$

$$C_i \leq \frac{1}{m}\sum_j p_j + \left(1 - \frac{1}{m}\right) p_\ell \leq (2 - \tfrac{1}{m}) \cdot OPT$$

Start with an arbitrary schedule;
repeat until no job is reassigned (a local optimum is encountered):
  let l be a job that fished last;
  if ∃ machine i s.t. *job l will finish earlier after reassigned* to machine i
    transfer job l to machine i;

finds a schedule with makespan $C_{\max} \leq (2 - \frac{1}{m}) \cdot OPT$

***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

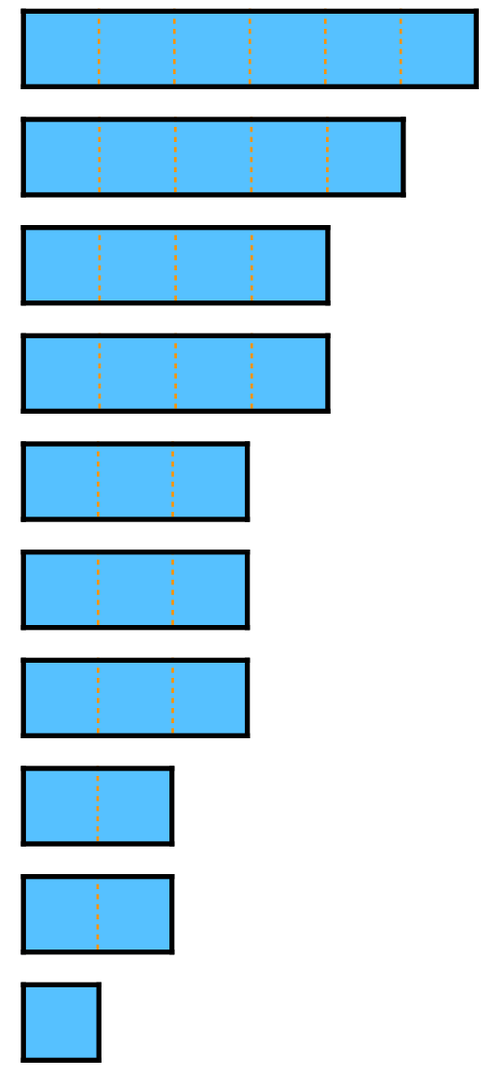
the schedule returned by the *List* algorithm must be a local optimum

⇨ the schedule returned by the *List* algorithm has makespan $C_{\max} \leq (2 - \frac{1}{m}) \cdot OPT$

Longest Processing Time (LPT)

m machines

n jobs



***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

***Longest Processing Time* (LPT)**

$p_1 \geq p_2 \geq \cdots \geq p_n$;
for j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

$$OPT \geq \frac{1}{m} \sum_j p_j$$

for the schedule returned by the LPT algorithm:

makespan $C_{\max} = C_i \quad \leq \frac{3}{2} \cdot OPT$

the last job assigned to machine i is job l

$$C_i - p_\ell \leq \frac{1}{m} \sum_j p_j \leq OPT$$

WLOG:

$$C_i > \max_j p_j \Rightarrow \left.\begin{array}{l} p_\ell \leq p_{m+1} \\ OPT \geq p_m + p_{m+1} \geq 2p_{m+1} \end{array}\right\} \Rightarrow p_\ell \leq \frac{1}{2} OPT$$

***Longest Processing Time* (LPT)**

$p_1 \geq p_2 \geq \cdots \geq p_n$;
for j=1, 2, ... , n
  assign job j to the *current*
  least heavily loaded machine;

for the schedule returned by the LPT algorithm:

makespan $C_{\max} \leq \frac{3}{2} \cdot OPT$

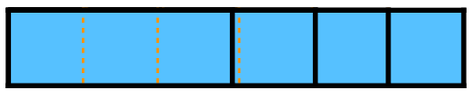
- With a more careful analysis, the *LPT* is a $4/3$-approximation algorithm.
- The problem of minimum makespan on identical machines has a **PTAS** (**P**olynomial **T**ime **A**pproximation **S**cheme).

$\forall \varepsilon > 0$, $\exists$ poly-time $(1+\varepsilon)$-algorithm for the problem

Online Scheduling

m machines

n jobs arrive one-by-one



schedule decision must be made when a job arrives
without seeing jobs in the future

***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

Competitive Analysis

***List* Algorithm** (Graham 1966)

For j=1, 2, ... , n
  assign job j to the *current least heavily loaded* machine;

the **competitive ratio** of the online algorithm is α if:

$\forall$ input sequence I:

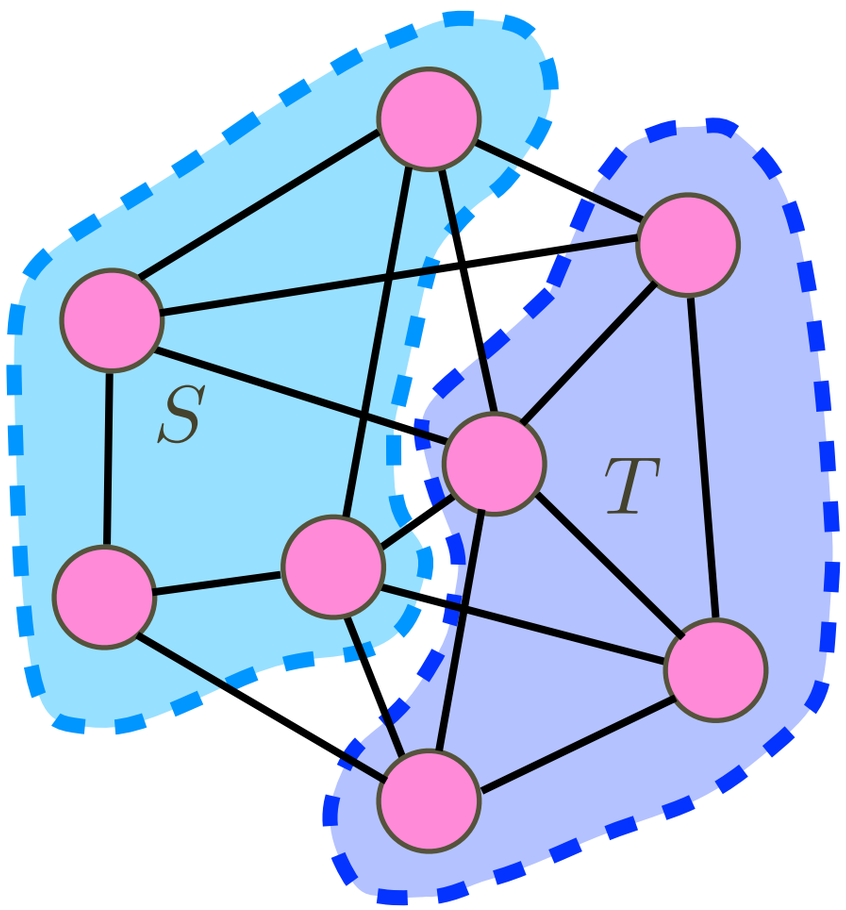
$$\frac{\text{solution returned by the } \textit{online} \text{ algorithm on } I}{\text{solution returned by the } \textit{optimal offline} \text{ algorithm on } I} \leq \alpha$$

the list algorithm is 2-competitive

Max-Cut

Instance: An undirected graph $G(V, E)$.

Solution: A bipartition of V into S and T that maximizes the cut $E(S,T) = \{uv \in E: u \in S, v \in T\}$.

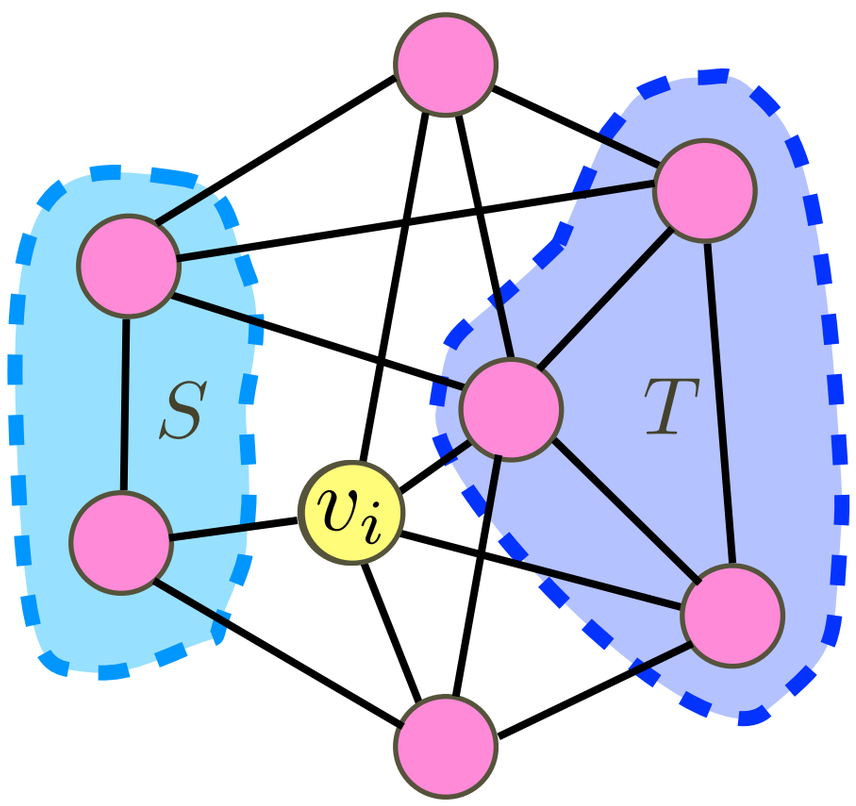


- **NP**-hard.
- One of Karp's 21 **NP**-complete problems (reduction from the *Partition* problem).
- a typical **Max-CSP** (Constraint Satisfaction Problem).
- *Greedy* is 1/2-approximate.

Max-Cut

Instance: An undirected graph $G(V, E)$.

Solution: A bipartition of V into S and T that maximizes the cut $E(S,T) = \{uv \in E: u \in S, v \in T\}$.



GreedyMaxCut

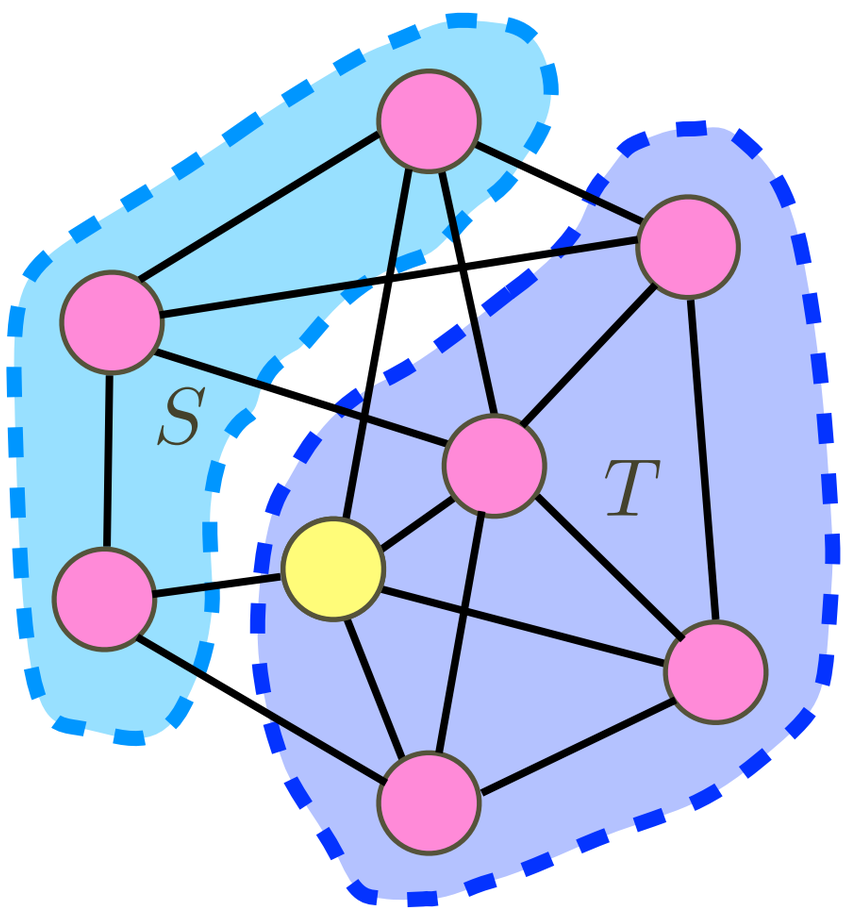
$V = \{v_1, v_2, \ldots, v_n\}$;
initially, $S=T=\varnothing$;
for $i = 1,2, \ldots, n$
  v_i joins one of S, T
  to maximize *current* $E(S,T)$

GreedyMaxCut is $1/2$-approximate

Max-Cut

Instance: An undirected graph $G(V, E)$.

Solution: A bipartition of V into S and T that maximizes the cut $E(S,T) = \{uv \in E: u \in S, v \in T\}$.

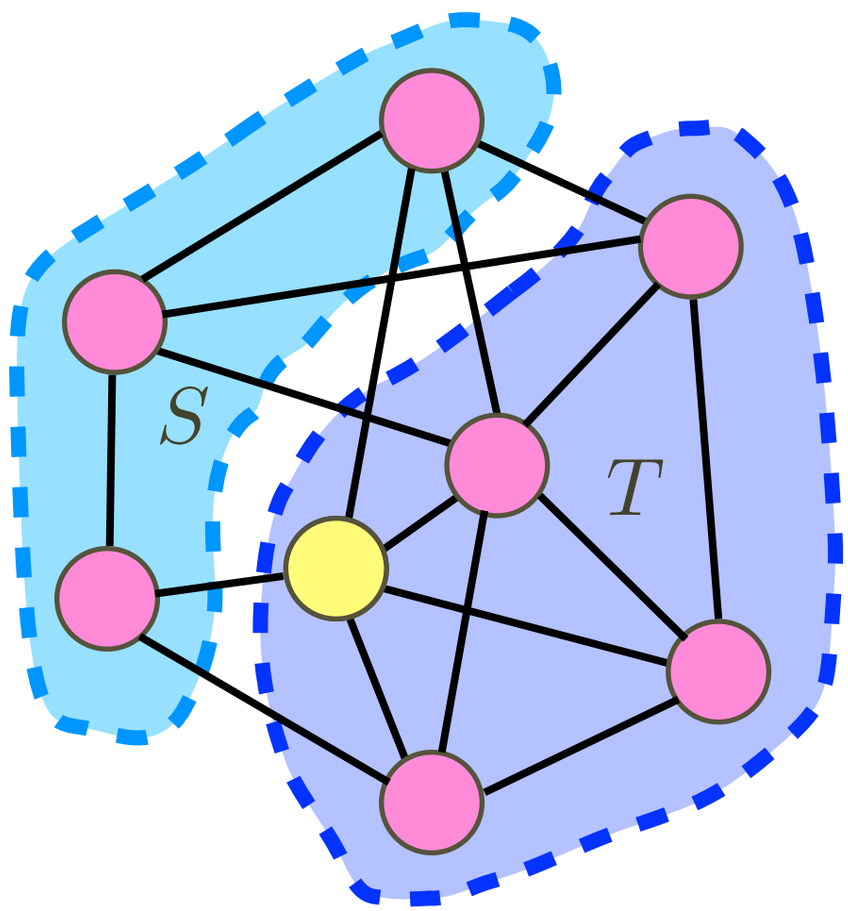


Local Search

Start with an *arbitrary* bipartition;
repeat until nothing changed:
  if $\exists v$ flipping side will increase cut
    v moves to the other side;

Local Search

Start with an *arbitrary* bipartition;
repeat until nothing changed:
  if $\exists v$ flipping side will increase cut
    v moves to the other side;



in a local optimum:

$$\forall\, v \in S,\ |E(v, S)| \le |E(v,T)|$$

$$\Rightarrow\ 2|E(S, S)| \le |E(S,T)|$$

$$\forall\, v \in T,\ |E(v, T)| \le |E(v,S)|$$

$$\Rightarrow\ 2|E(T, T)| \le |E(S,T)|$$

$$|E(S, S)| + |E(T, T)| \le |E(S,T)|$$

$$\text{OPT} \le |E| = |E(S, S)| + |E(T, T)| + |E(S,T)| \le 2|E(S,T)|$$

$$\Rightarrow\ |E(S,T)| \ge \tfrac{1}{2} \cdot OPT$$

Local Search

Start with an *arbitrary* bipartition;
repeat until nothing changed:
  if $\exists v$ flipping side will increase cut
    v moves to the other side;

in a local optimum: $|E(S,T)| \geq \frac{1}{2} \cdot OPT$

GreedyMaxCut

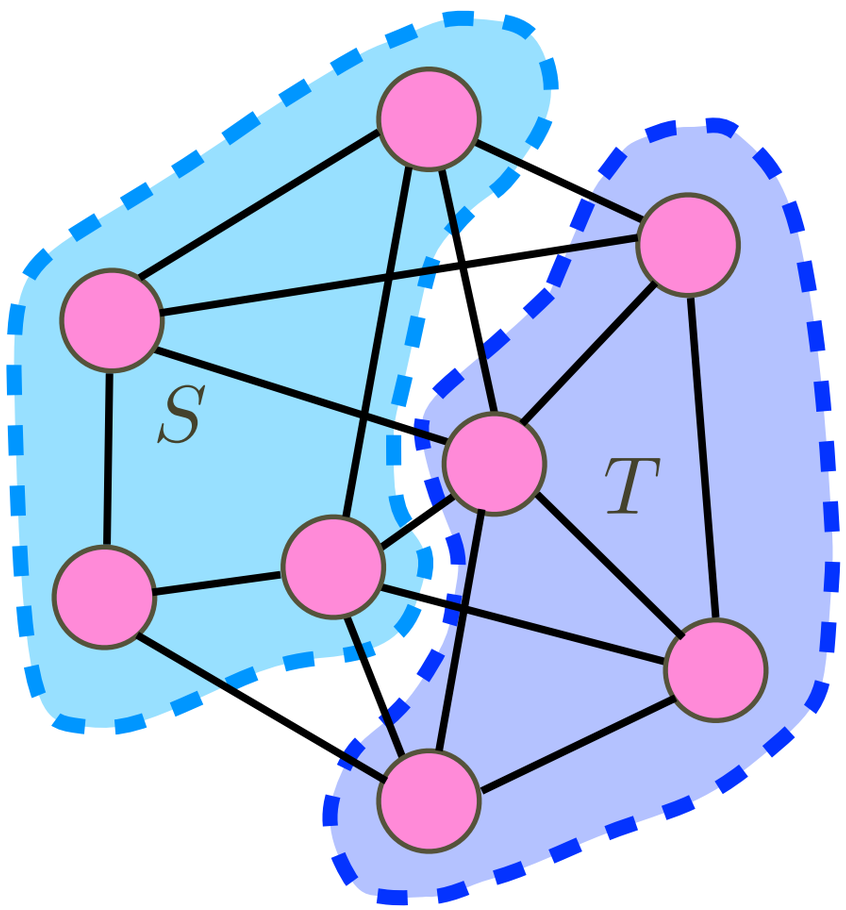
$V = \{v_1, v_2, \ldots, v_n\}$;
initially, $S=T=\emptyset$;
for $i = 1,2, \ldots, n$
  v_i joins one of S, T
  to maximize *current* $E(S,T)$

Is the cut returned by *GreedyMaxCut* locally optimal?

Max-Cut

Instance: An undirected graph $G(V, E)$.

Solution: A bipartition of V into S and T that maximizes the cut $E(S,T) = \{uv \in E: u \in S, v \in T\}$.



- **NP**-hard.
- *Greedy* and *local search* are 1/2-approximate.
- Rounding *semidefinite programing* has approximation ratio 0.878~.
- Assuming the *unique game conjecture*, no poly-time <0.878-approximate algorithm.