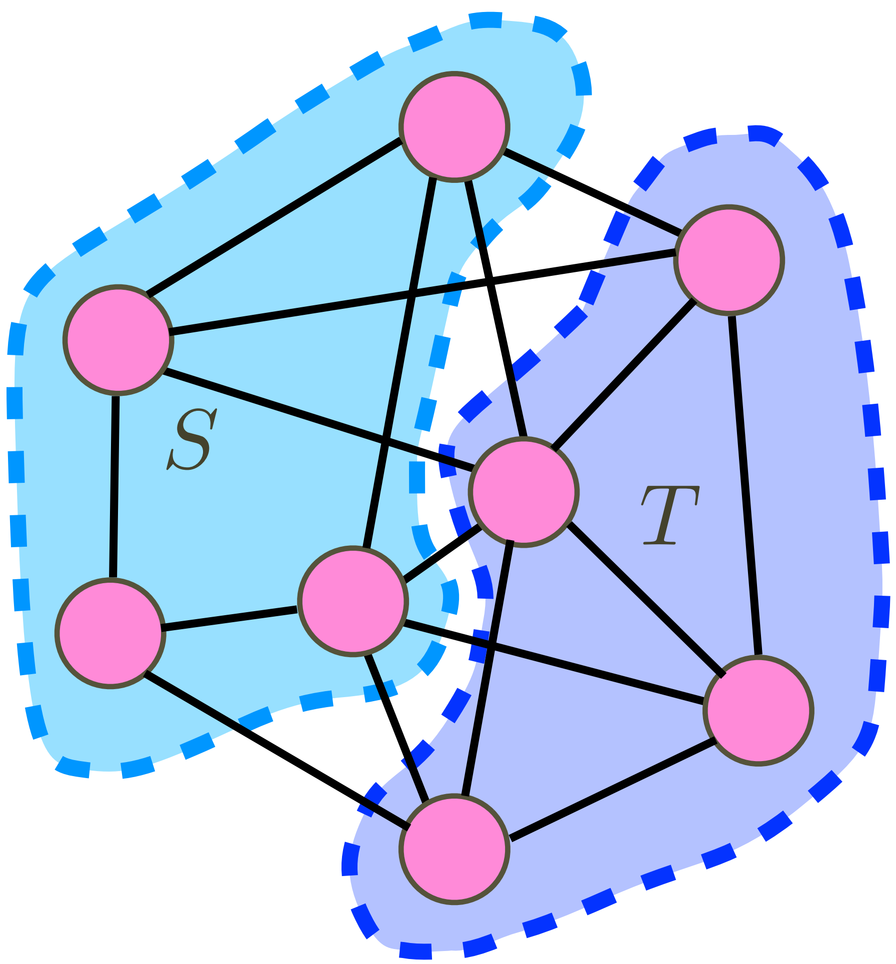
Advanced Algorithms

Greedy and Local Search

尹一通 Nanjing University, 2022 Fall

Max-Cut

Instance: An undirected graph $G(V, E)$.
Solution: A bipartition of V into S and T that maximizes the cut $E(S,T) = \big\{\{u,v\} \in E \mid u \in S \wedge v \in T\big\}$.

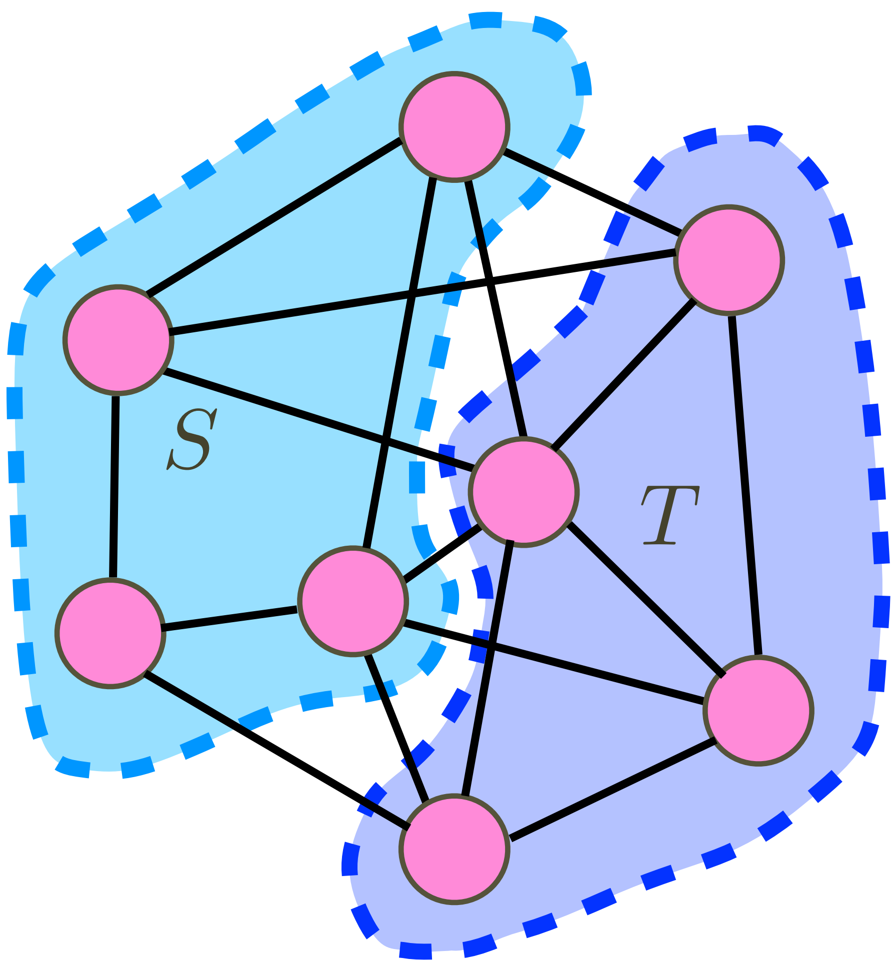


- **NP**-hard.
- One of Karp's 21 **NP**-complete problems (reduction from the *Partition* problem).
- a typical **Max-CSP** (Constraint Satisfaction Problem).
- *Greedy* is $1/2$-approximate.

Greedy Algorithm

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

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



Greedy Cut:

initially, $S = T = \emptyset$;

for $i = 1,2,\ldots,n$:

v_i joins one of S, T

to maximize current $E(S, T)$;

Approximation Ratio

Algorithm $\mathscr{A}$:

Greedy Cut:

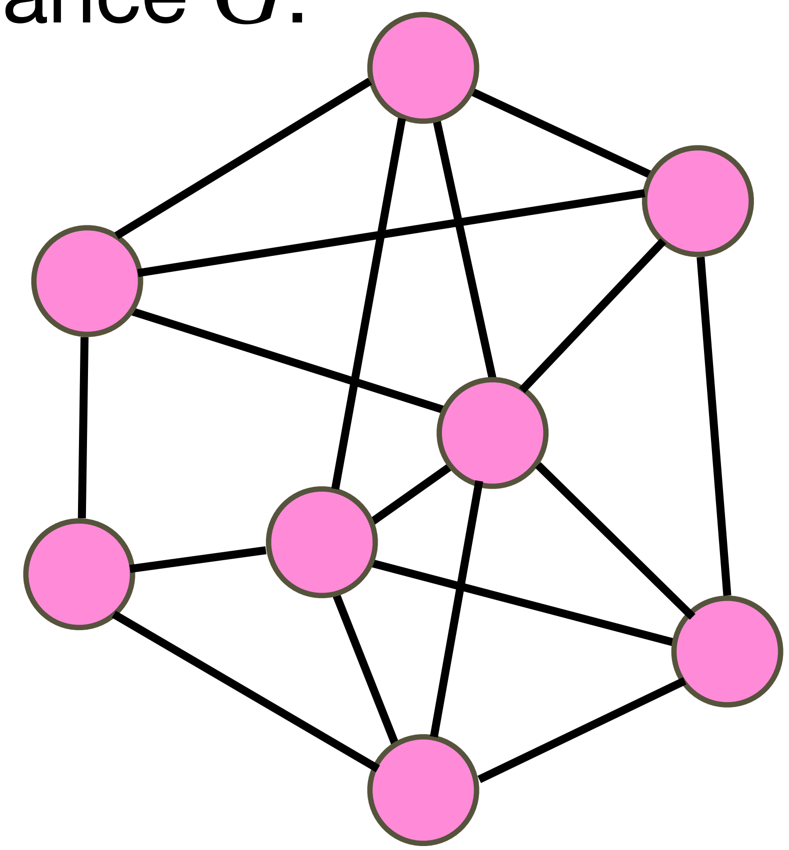
initially, $S = T = \varnothing$;

for $i = 1,2,\ldots,n$:

v_i joins one of S, T

to maximize current $E(S, T)$;

Instance G:



OPT_G: value of max-cut in G

SOL_G: value of the cut returned by $\mathscr{A}$ on G

Algorithm $\mathscr{A}$ has approximation ratio α if

$$\forall \text{ instance } G, \quad \frac{SOL_G}{OPT_G} \geq \alpha$$

Approximation Algorithm

Greedy Cut:

initially, $S = T = \varnothing$;

for $i = 1,2,\ldots,n$:

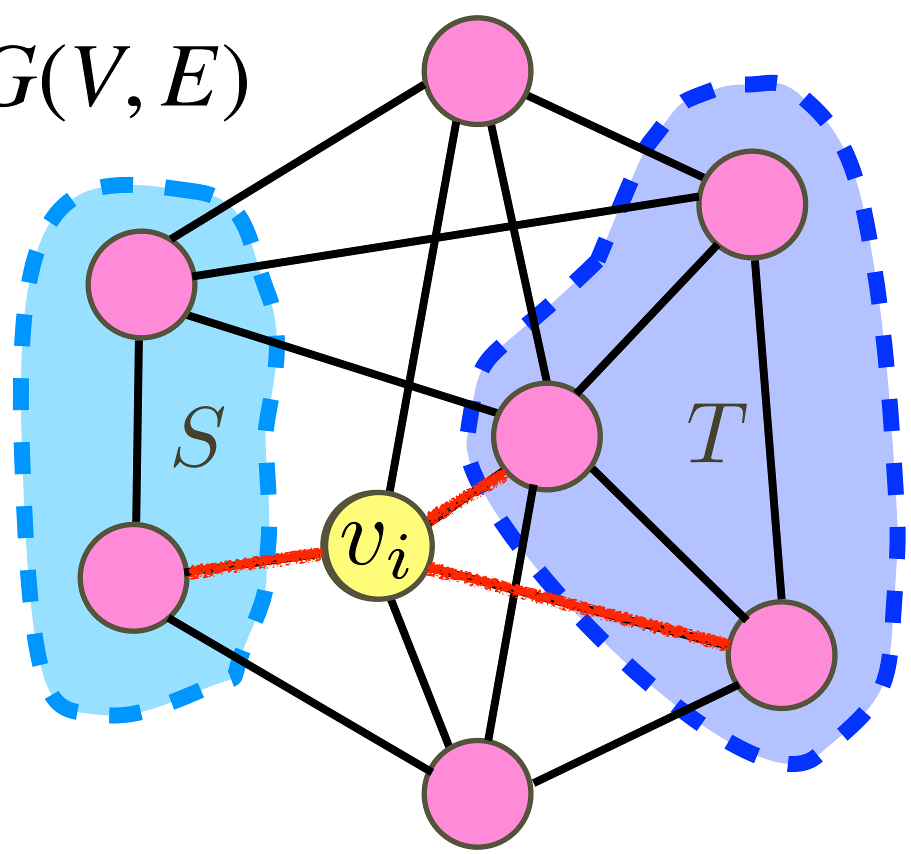
    v_i joins one of S, T

    to maximize current $E(S, T)$;

(S_i, T_i):

current (S, T) in the beginning of i-th iteration

$G(V, E)$



$E(S, T) = \{uv \in E \mid u \in S, v \in T\}$

$$\frac{SOL_G}{OPT_G} \geq \frac{SOL_G}{|E|} \geq \frac{1}{2}$$

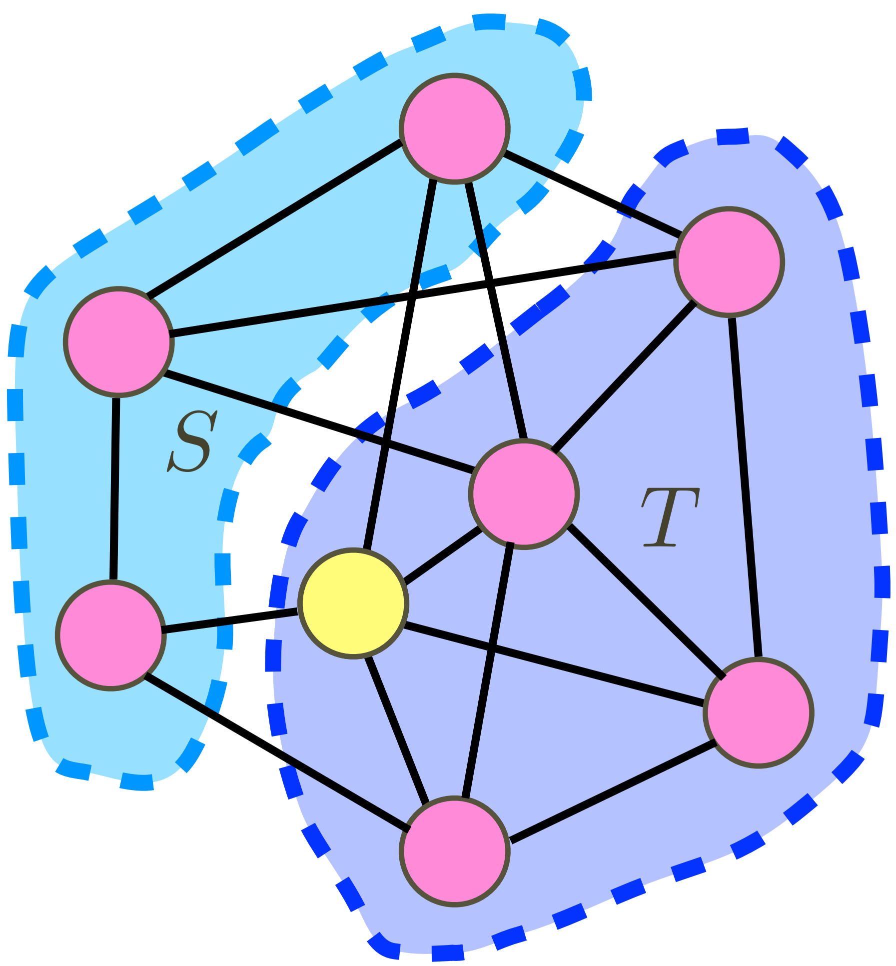
$\forall v_i,\ \geq 1/2$ of $|E(S_i, v_i)| + |E(T_i, v_i)|$ contributes to SOL_G

$$|E| = \sum_{i=1}^{n} \left(|E(S_i, v_i)| + |E(T_i, v_i)| \right)$$

Local Search

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

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



Local Search:

initially, (S, T) is an arbitrary cut;

repeat until nothing changed:

if $\exists v$ switching side increases cut

v switches to the other side;

locally improve the solution until no improvement can be made (*local optima*, *fixpoint*)

Local Search

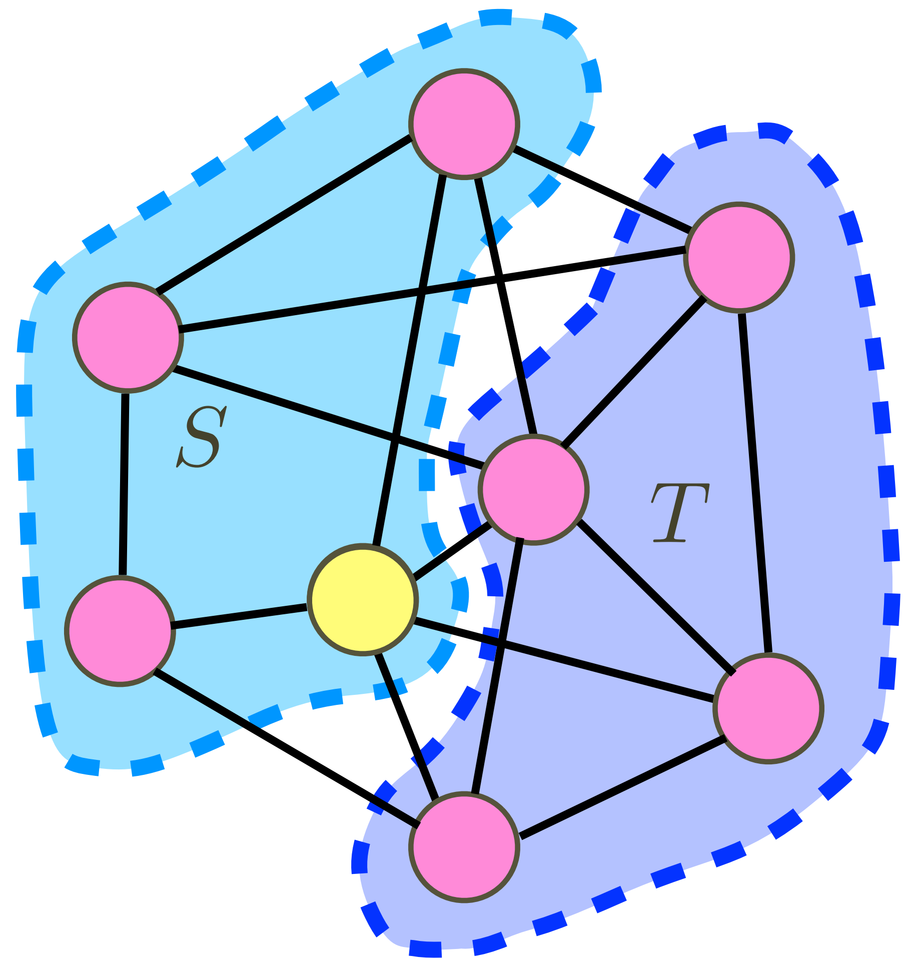
Local Search:

initially, (S, T) is an arbitrary cut;

repeat until nothing changed:

  if $\exists v$ switching side increases cut

    v switches to the other side;



in a local optima:

$$\forall v \in S : |E(v,S)| \le |E(v,T)| \implies 2|E(S,S)| \le |E(S,T)|$$

$$\forall v \in T : |E(v,T)| \le |E(v,S)| \implies 2|E(T,T)| \le |E(S,T)|$$

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

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

$$\implies |E(S,T)| \ge \frac{1}{2} OPT$$

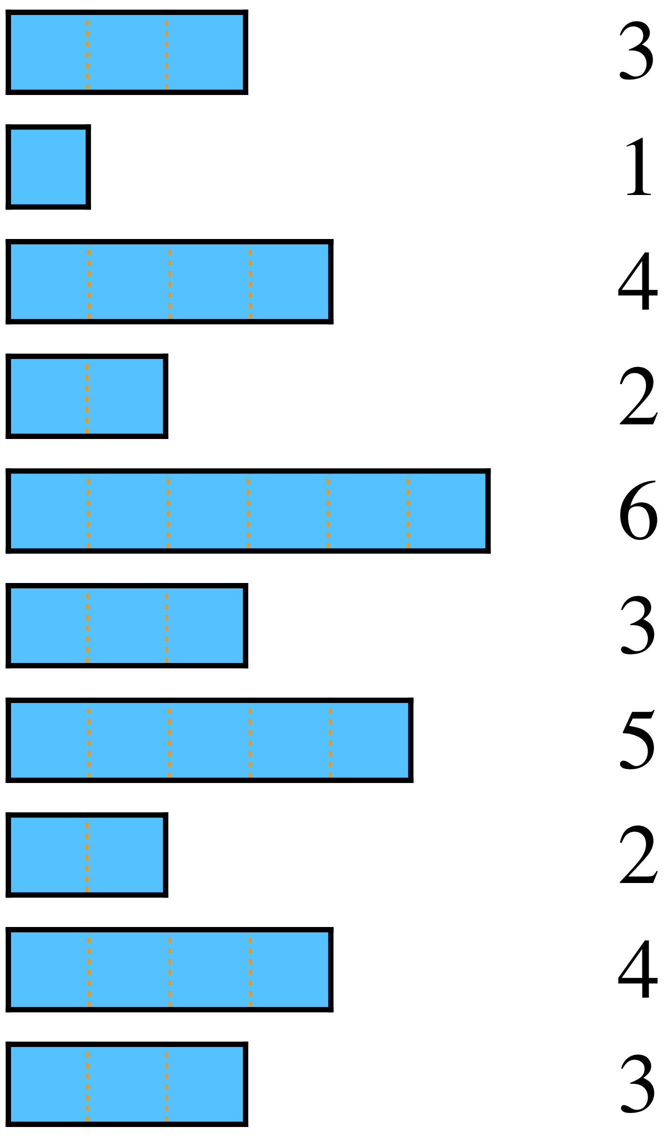
Scheduling

Scheduling

m machines

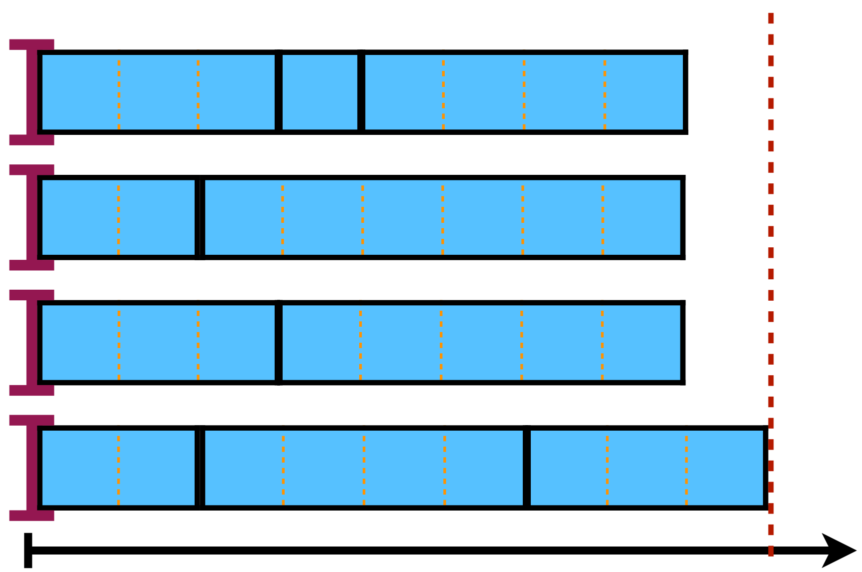
n jobs

processing time p_j



Scheduling

m machines



n jobs with

processing time p_j

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

Makespan: $C_{\max} = \max_{1 \le i \le} C_i$

Instance: n jobs $j = 1,\ldots,n$ with processing times $p_j \in \mathbb{R}^+$

Solution: An assignment of n jobs to m *identical* machines that minimizes the ***makespan*** $C_{\max}$

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

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: release times; d_j: deadlines; pmtn: preemption;
- γ: objective
  - $C_{\max}$: makespan; $\sum_i C_i$: total completion time; $L_{\max}$: maximum lateness;

Instance: n jobs $j = 1,\ldots,n$ with processing times $p_j \in \mathbb{R}^+$

Solution: An assignment of n jobs to m *identical* machines that minimizes the ***makespan*** $C_{\max}$

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

- Reducible from the **partition problem:**

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

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

- One of Karp's 21 **NPC** problems

Approximation Ratio

Instance: n jobs $j = 1, \ldots, n$ with processing times $p_j \in \mathbb{R}^+$

Solution: An assignment of n jobs to m *identical* machines that minimizes the ***makespan*** $C_{\max}$

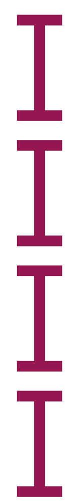
An algorithm $\mathscr{A}$ for a minimization problem has **approximation ratio α** if

$$\forall \text{ instance } I, \quad \frac{SOL_I}{OPT_I} \leq \alpha$$

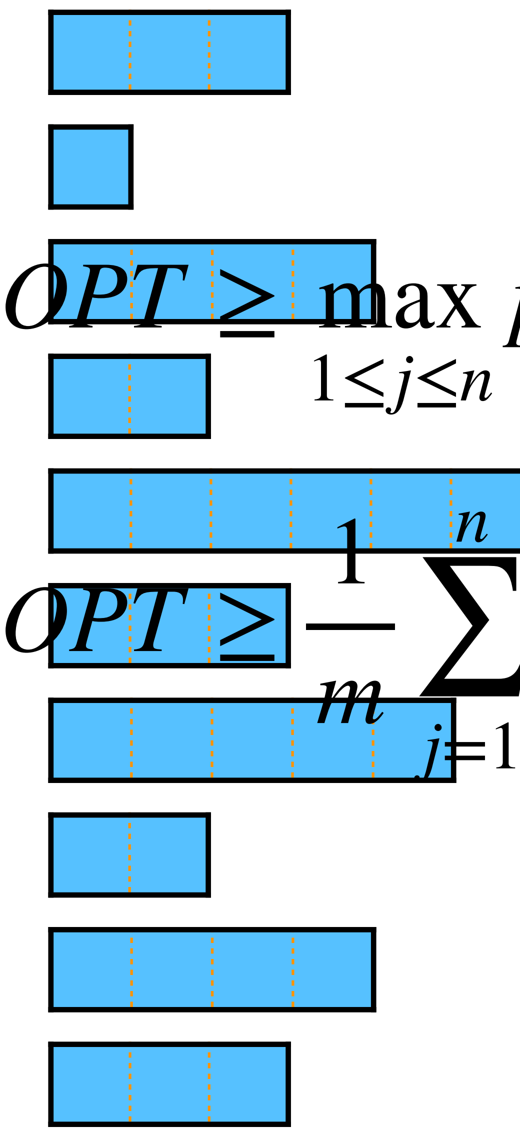
- SOL_I : solution returned by the algorithm on instance I
- OPT_I : optimal solution of instance I

Graham's *List* Algorithm

m machines



n jobs



$$OPT \geq \max_{1 \leq j \leq n} p_j$$

$$OPT \geq \frac{1}{m} \sum_{j=1}^{n} p_j$$

***List* algorithm (Graham 1966):**

For $j = 1,2,\ldots,n$:

assign job j to the current

least heavily loaded machine;

***List* algorithm (Graham 1966):**

For $j = 1,2,\ldots,n$:

assign job j to the current

least heavily loaded machine;

- n jobs with processing times $p_1, \ldots, p_n$ assigned to m machines:
- Optimal **makespan**: $OPT \geq \max_{1 \leq j \leq n} p_j$ $\quad OPT \geq \frac{1}{m} \sum_{j=1}^{n} p_j$
- Solution returned by the *List* algorithm:
  - suppose $C_{\max} = C_{i*} \quad \leq 2 \cdot OPT$
  - and the last job assigned to machine $i*$ is ℓ
- Before job ℓ is assigned, machine $i*$ is the *least heavily loaded*

$$\Longrightarrow \quad \left.\begin{aligned} C_{i*} - p_\ell &\leq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq OPT \\ p_\ell &\leq \max_{1 \leq j \leq n} p_j \quad \leq OPT \end{aligned}\right\}$$

List algorithm (Graham 1966):

For $j = 1,2,\ldots,n$:

assign job j to the current

least heavily loaded machine;

- n jobs with processing times $p_1, \ldots, p_n$ assigned to m machines:
- Optimal **makespan**: $OPT \geq \max_{1\leq j\leq n} p_j$ $\quad OPT \geq \frac{1}{m}\sum_{j=1}^{n} p_j$
- Solution returned by the *List* algorithm:
  - suppose $C_{\max} = C_{i*} \leq \left(1-\frac{1}{m}\right)p_\ell + \frac{1}{m}\sum_{1\leq j\leq n} p_j \leq \left(2-\frac{1}{m}\right)OPT$
  - and the last job assigned to machine i^* is ℓ
- Before job ℓ is assigned, machine i^* is the *least heavily loaded*

$$\Longrightarrow \quad \left.\begin{array}{l} C_{i*} - p_\ell \leq \frac{1}{m}\sum_{j\neq\ell} p_j \\ p_\ell \leq \max_{1\leq j\leq n} p_j \end{array}\right\}$$

Graham's *List* Algorithm

***List* algorithm** (Graham 1966)**:**

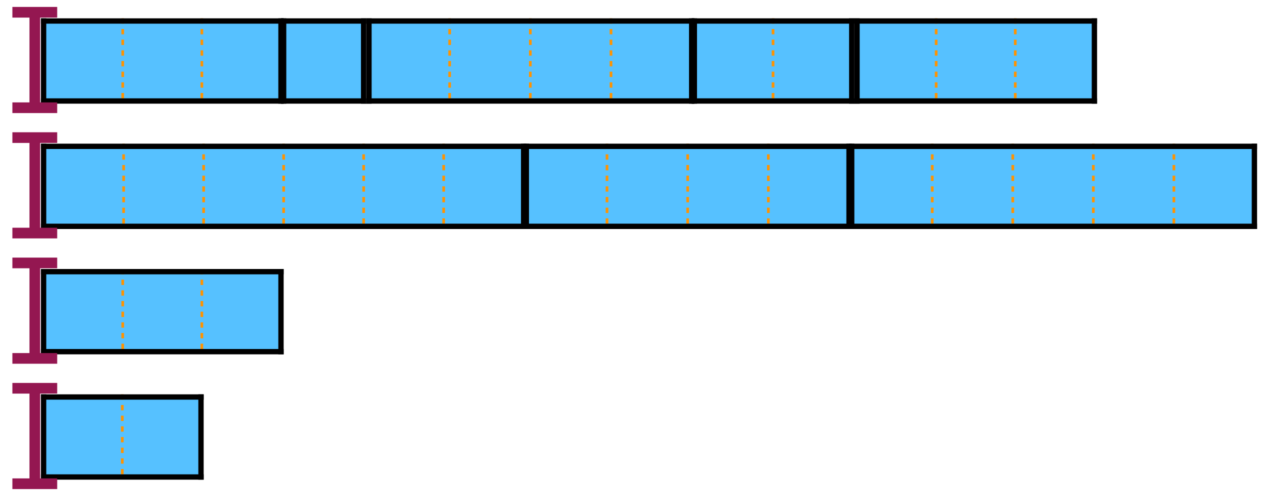
For $j = 1,2,\ldots,n$:

assign job j to the current

least heavily loaded machine;

- n jobs are assigned to m machines
- The *List* algorithm returns a schedule with makespan:
  - $C_{\max} \leq \left(2 - \frac{1}{m}\right) OPT$
  - This is tight in the worst case.

Local Search



locally improve the solution until no improvement can be made
(*local optima*, *fixpoint*)

Local search:

Start from an arbitrary schedule;
repeat until no job is reassigned (a local optima):
    if the last finished job ℓ can finish earlier by moving to machine i
        transfer job ℓ to machine i;

Local search:

Start from an arbitrary schedule;

repeat until no job is reassigned (a local optima):

    if the last finished job ℓ can finish earlier by moving to machine i

        transfer job ℓ to machine i;

- Optimal **makespan**: $OPT \geq \max_{1 \leq j \leq n} p_j$ $\quad OPT \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j$
- In a local optima:
  - suppose $C_{\max} = C_{i^*} \leq \left(1 - \frac{1}{m}\right) p_\ell + \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq \left(2 - \frac{1}{m}\right) OPT$
  - and job ℓ finishes the last
- local optima $\Longrightarrow C_{i^*} - p_\ell$ is the least heavy load

$$C_{i^*} - p_\ell \leq \frac{1}{m} \sum_{j \neq \ell} p_j$$

$$p_\ell \leq \max_{1 \leq j \leq n} p_j$$

Local search:

Start from an arbitrary schedule;

repeat until no job is reassigned (a local optima):

if the last finished job ℓ can finish earlier by moving to machine i

transfer job ℓ to machine i;

For a local optima: $$C_{\max} \le \left(2 - \frac{1}{m}\right) OPT$$

List algorithm (Graham 1966):

For $j = 1, 2, \ldots, n$:

assign job j to the current

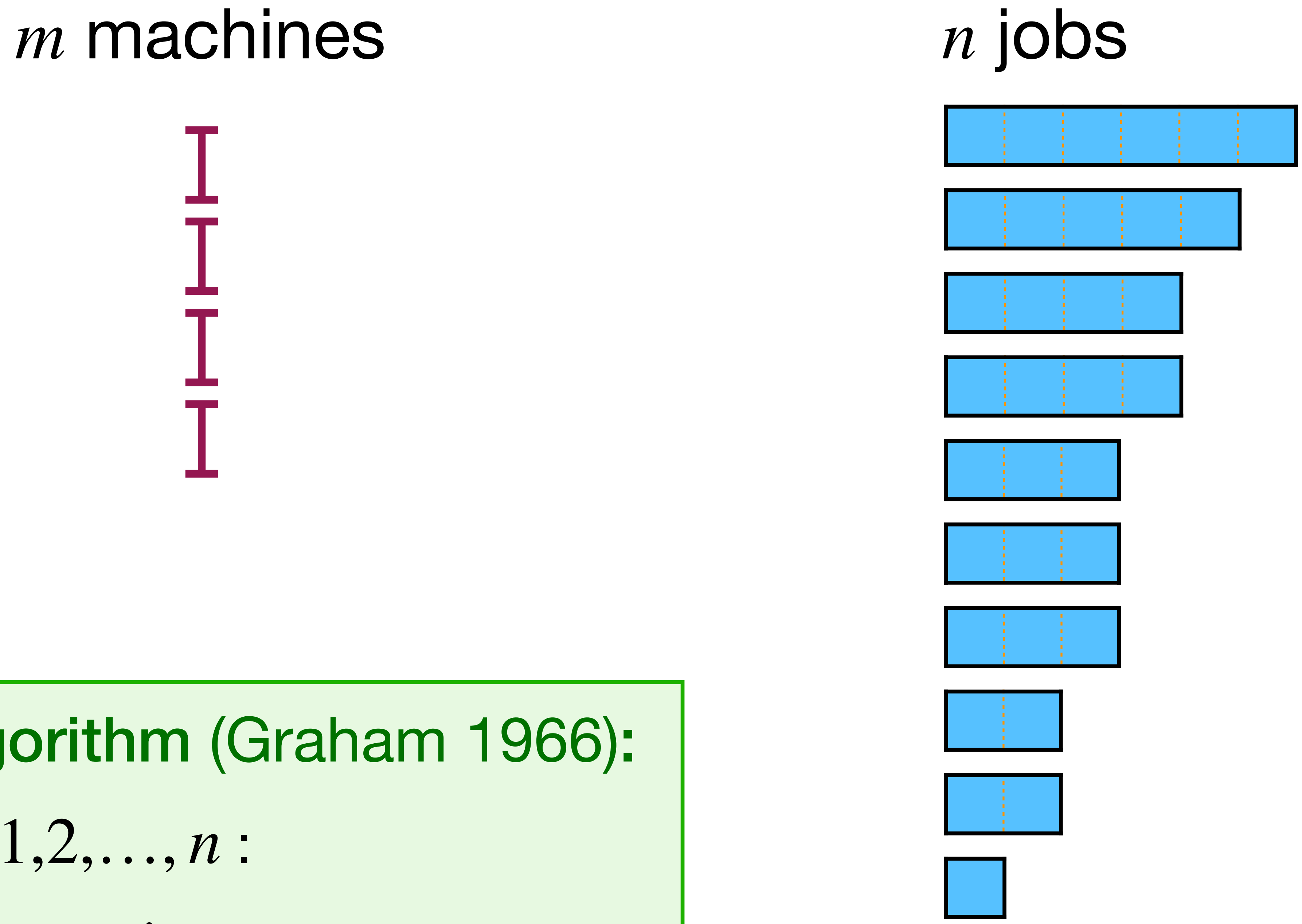
least heavily loaded machine;

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

- $\Longrightarrow$ the schedule returned by the *List* algorithm:

$$C_{\max} \le \left(2 - \frac{1}{m}\right) OPT$$

Longest Processing Time (LPT)



***List* algorithm** (Graham 1966)**:**

For $j = 1,2,\ldots,n$:

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 $j = 1,2,\ldots,n$:

assign job j to the current least heavily loaded machine;

- Optimal **makespan:** $OPT \geq \frac{1}{m} \sum_{1 \leq j \leq n} p_j$
- Solution returned by the *LPT* algorithm:
  - suppose $C_{\max} = C_{i*} \quad \leq \frac{3}{2} \cdot OPT$
  - and the last job assigned to machine $i*$ is ℓ
- Before job ℓ is assigned, machine $i*$ is the *least heavily loaded*

$$\implies \quad C_{i*} - p_\ell \leq \frac{1}{m} \sum_{1 \leq j \leq n} p_j \leq OPT$$

WLOG: $\ell > m \implies p_\ell \leq p_{m+1}$

Pigeonhole : $OPT \geq p_m + p_{m+1} \geq 2p_{m+1}$

$$\implies p_\ell \leq \frac{1}{2} OPT$$

Longest Processing Time (LPT)

$p_1 \geq p_2 \geq \cdots \geq p_n;$

For $j = 1,2,\ldots,n$:

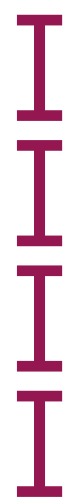
assign job j to the current least heavily loaded machine;

- Solution returned by the *LPT* algorithm:
  - makespan $C_{\max} \leq \frac{3}{2} \cdot OPT$
- Can be improved to 4/3-**approx**. with a more careful analysis.
- The problem of minimum makespan on identical machines has a **PTAS** (**P**olynomial-**T**ime **A**pproximation **S**cheme):

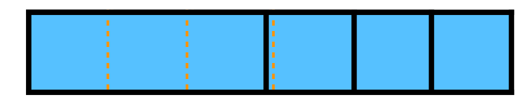
$\forall \epsilon > 0$, a $(1+\epsilon)$-approx. solution can be returned in time $f(\epsilon) \cdot \mathrm{poly}(n)$

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)**:**

Upon receiving a job:

assign the job to the current least heavily loaded machine;

Competitive Analysis

***List* algorithm** (Graham 1966)**:**

Upon receiving a job:

assign the job to the current

least heavily loaded machine;

the list algorithm is $(2 - 1/m)$-competitive

An online algorithm $\mathscr{A}$ for a minimization problem has competitive ratio α if

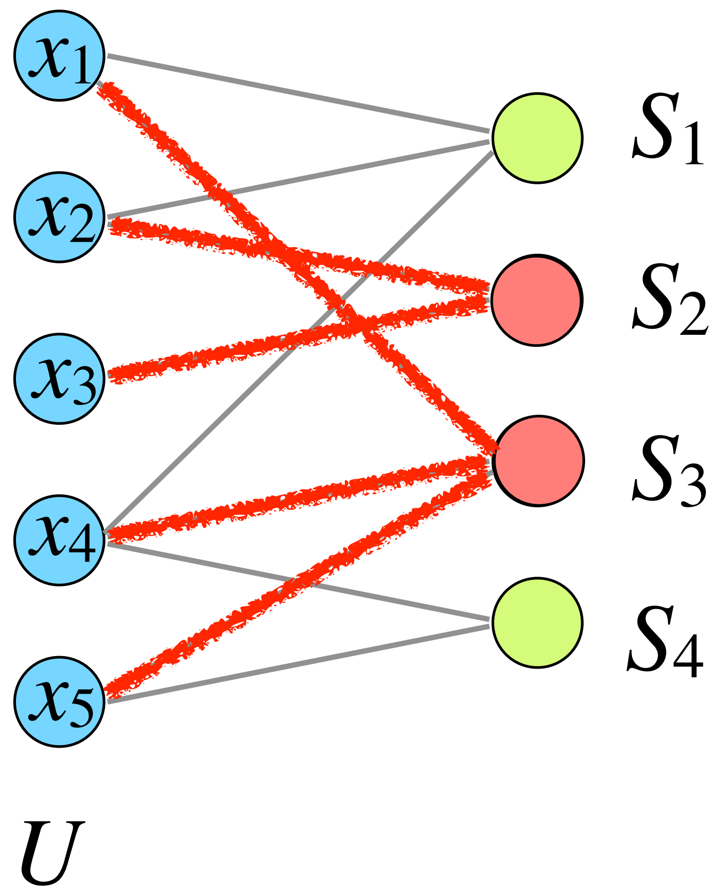
$$\forall \text{ instance } I, \quad \frac{SOL_I}{OPT_I} \leq \alpha$$

- SOL_I : solution returned by the online algorithm on instance I
- OPT_I : solution returned by an optimal offline algorithm on I

Set Cover

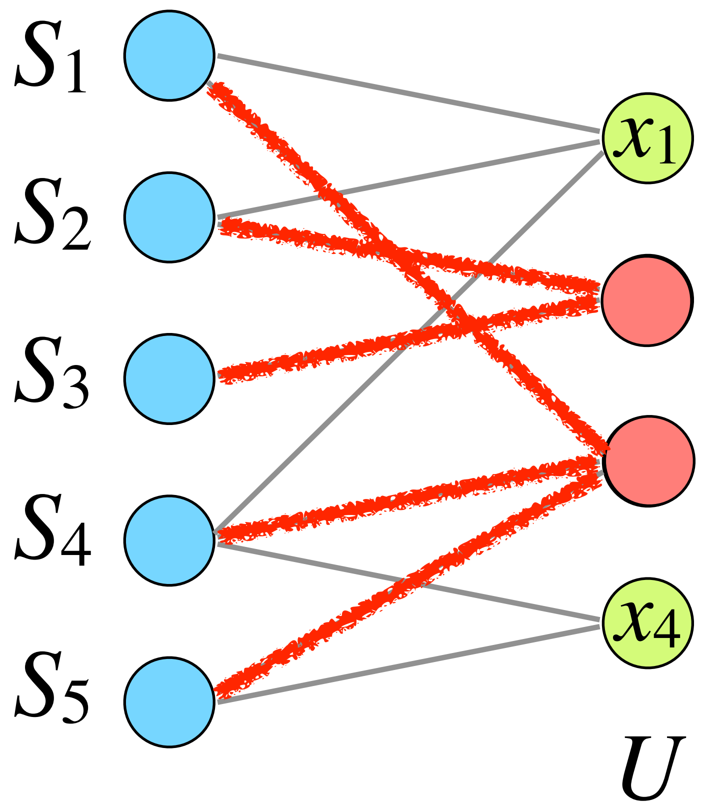
Set Cover

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.
Find the smallest $C \subseteq \{1,\ldots,m\}$ s.t. $\bigcup_{i \in C} S_i = U$.



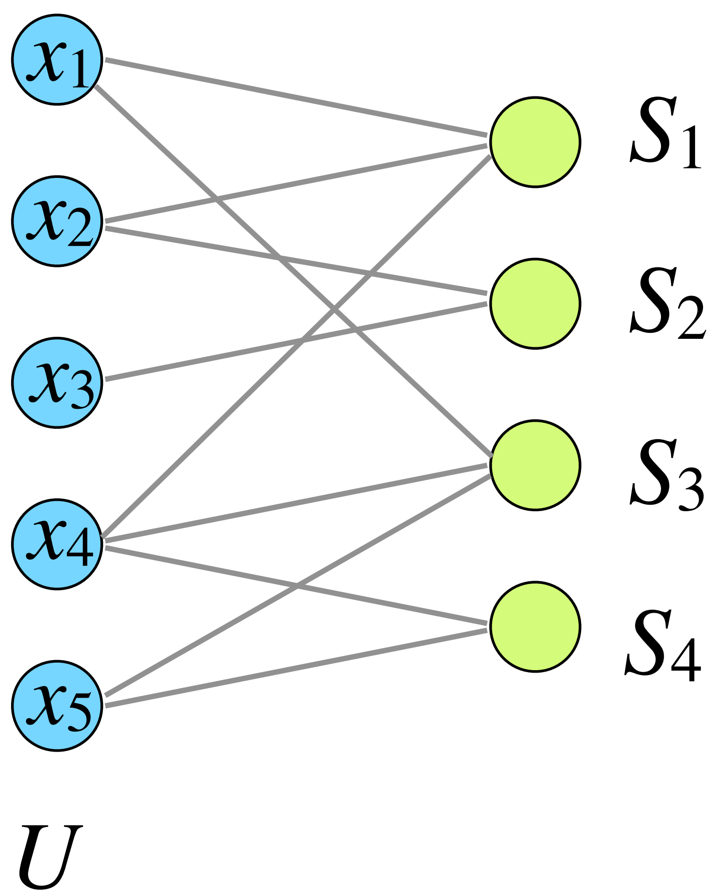
Hitting Set

Instance: A sequence of subsets $S_1, \ldots, S_n \subseteq U$.
Find the smallest $H \subseteq U$ s.t. $\forall i : S_i \cap H \neq \emptyset$.



Set Cover

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.
Find the smallest $C \subseteq \{1, \ldots, m\}$ s.t. $\bigcup_{i \in C} S_i = U$.

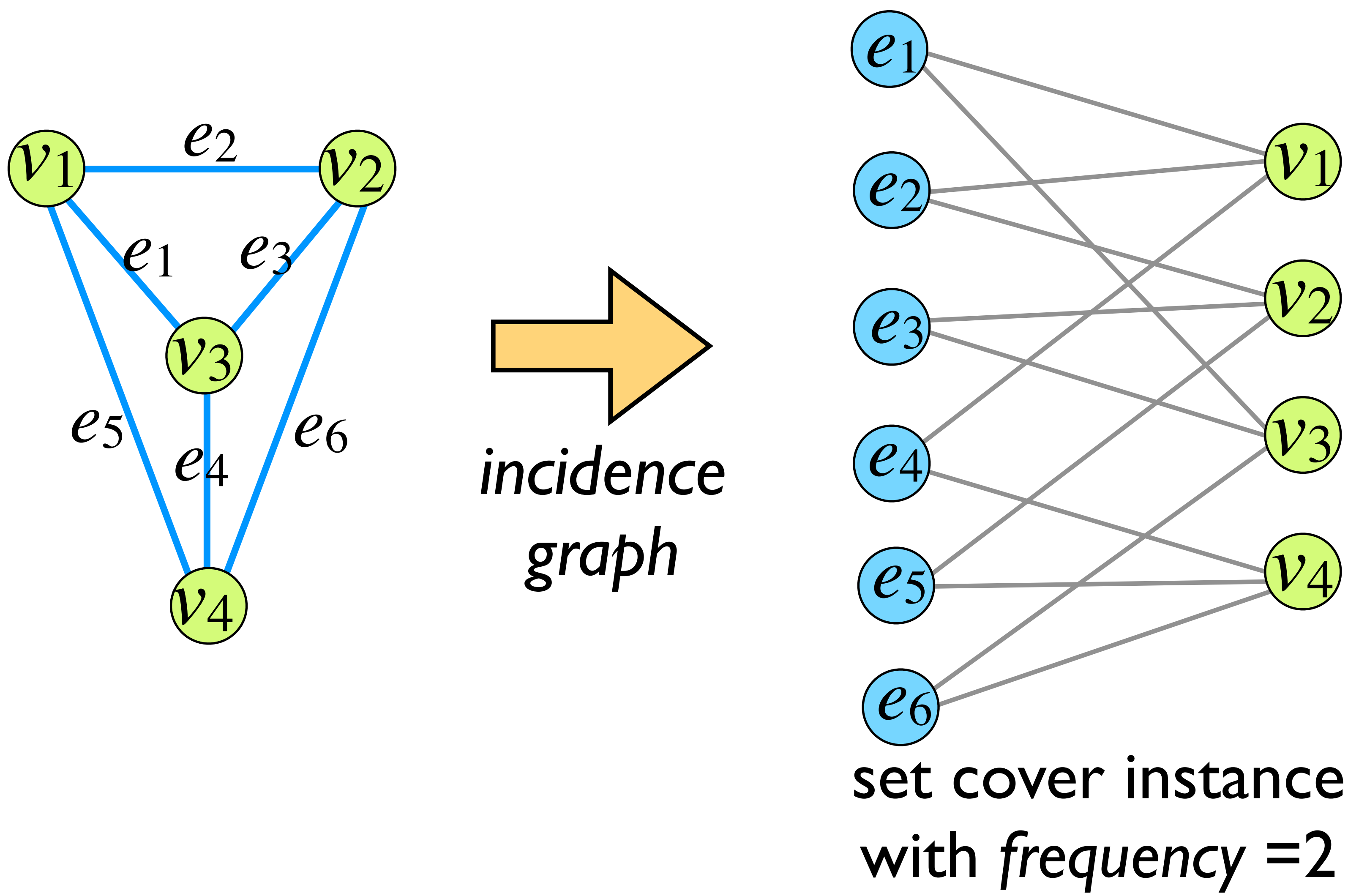


- **NP**-hard
- one of Karp's 21 **NP**-complete problems
- frequency of an element

$$\text{frquency}(x) = \left| \{ i \mid x \in S_i \} \right|$$

Vertex Cover

Instance: An undirected graph $G(V, E)$.
Find the smallest $C \subseteq V$ that intersects all edges.



Vertex Cover

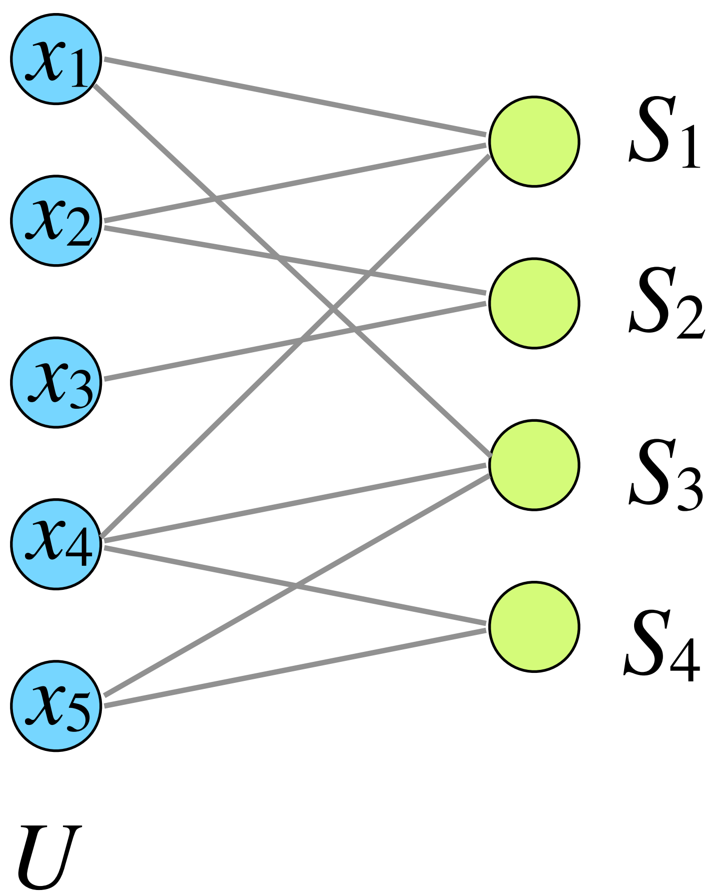
Instance: An undirected graph $G(V, E)$.
Find the smallest $C \subseteq V$ that intersects all edges.

- $\mathbf{NP}$-hard
- one of Karp's 21 $\mathbf{NP}$-complete problems

VC is $\mathbf{NP}$-hard $\Longrightarrow$ SC is $\mathbf{NP}$-hard

Greedy Set Cover

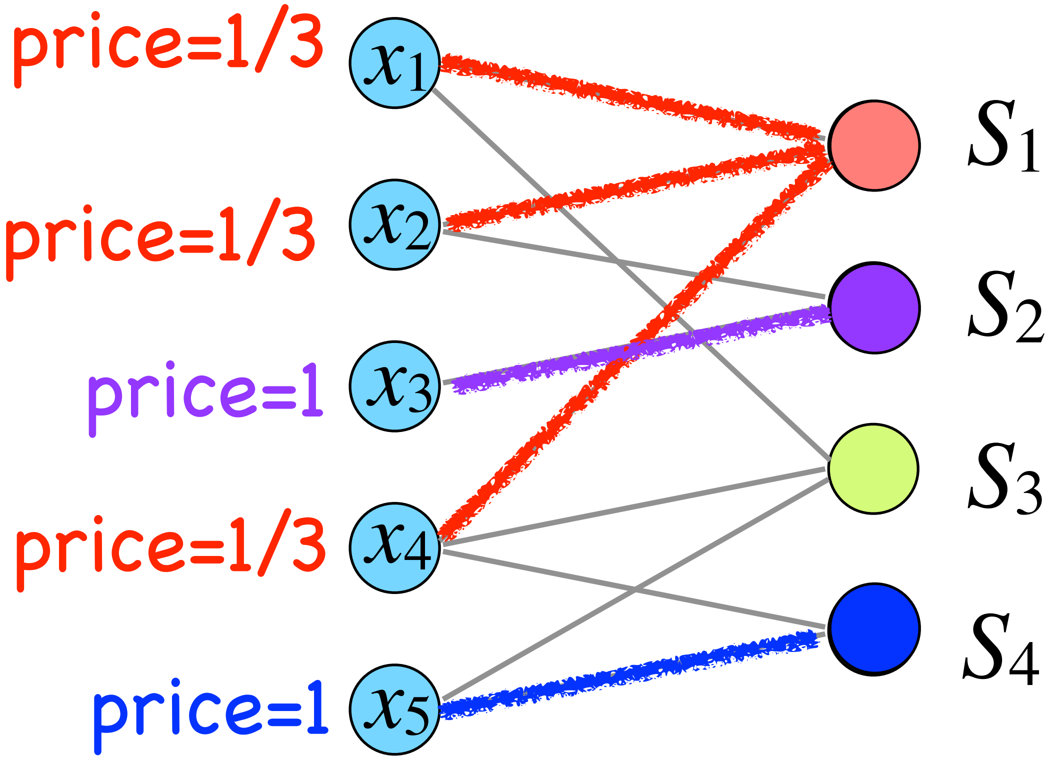
Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.
Find the smallest $C \subseteq \{1, \ldots, m\}$ s.t. $\bigcup_{i \in C} S_i = U$.



Greedy Cover:

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 sequence of subsets $S_1, \ldots, S_m \subseteq U$.



Greedy Cover:

initially $C = \varnothing$;

while $U \neq \varnothing$ do:

  add i with largest $|S_i \cap U|$ to C;

  $U = U \setminus S_i$; ∀x∈S_i∩U, price(x)=1/|S_i∩U|

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

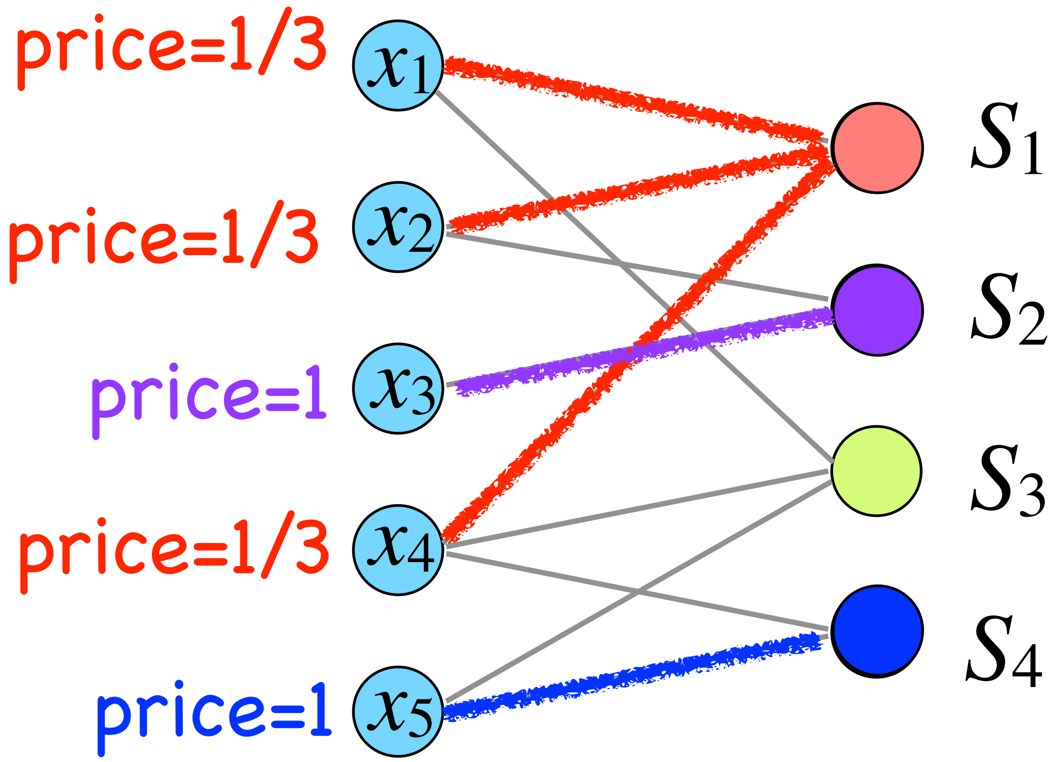
- **Averaging principle:** require $\geq \dfrac{|U|}{\max_i |S_i|}$ sets to cover U

$$OPT \geq \frac{|U|}{\max_i |S_i|}$$

- x_1 : first element covered by the ***GreedyCover*** algorithm

$$\text{price}(x_1) = \frac{1}{\max_i |S_i|} \implies \text{price}(x_1) \leq \frac{OPT}{|U|}$$

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.



Greedy Cover:

initially $C = \varnothing$;

while $U \neq \varnothing$ 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/$|S_i \cap U|$

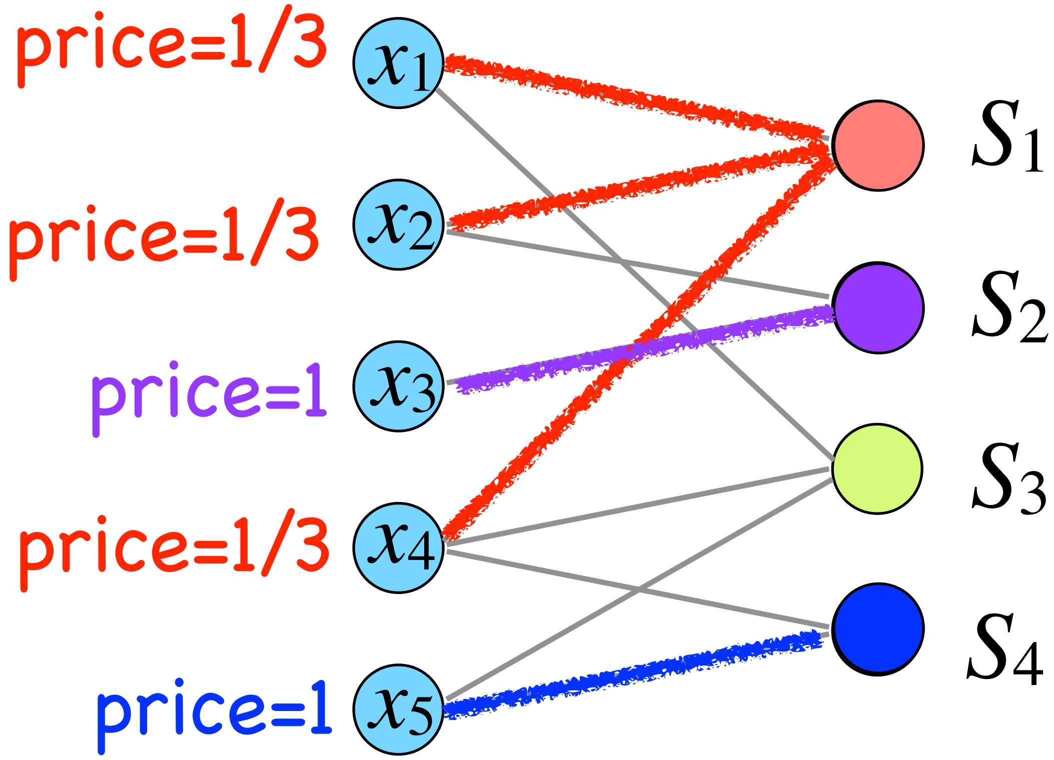
$$|C| = \sum_{x \in U} \text{price}(x)$$

- $x_1, \ldots, x_\ell$: covered in the 1st iteration in ***GreedyCover***

$$\forall 1 \leq k \leq \ell : \qquad \text{price}(x_k) = \text{price}(x_1) = \frac{1}{\max_i |S_i|}$$

$$\forall 1 \leq k \leq \ell : \qquad \text{price}(x_k) \leq \frac{OPT}{|U|} \leq \frac{OPT}{|U| - k + 1}$$

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.



Greedy Cover:

initially $C = \varnothing$;

while $U \neq \varnothing$ 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/$|S_i \cap U|$

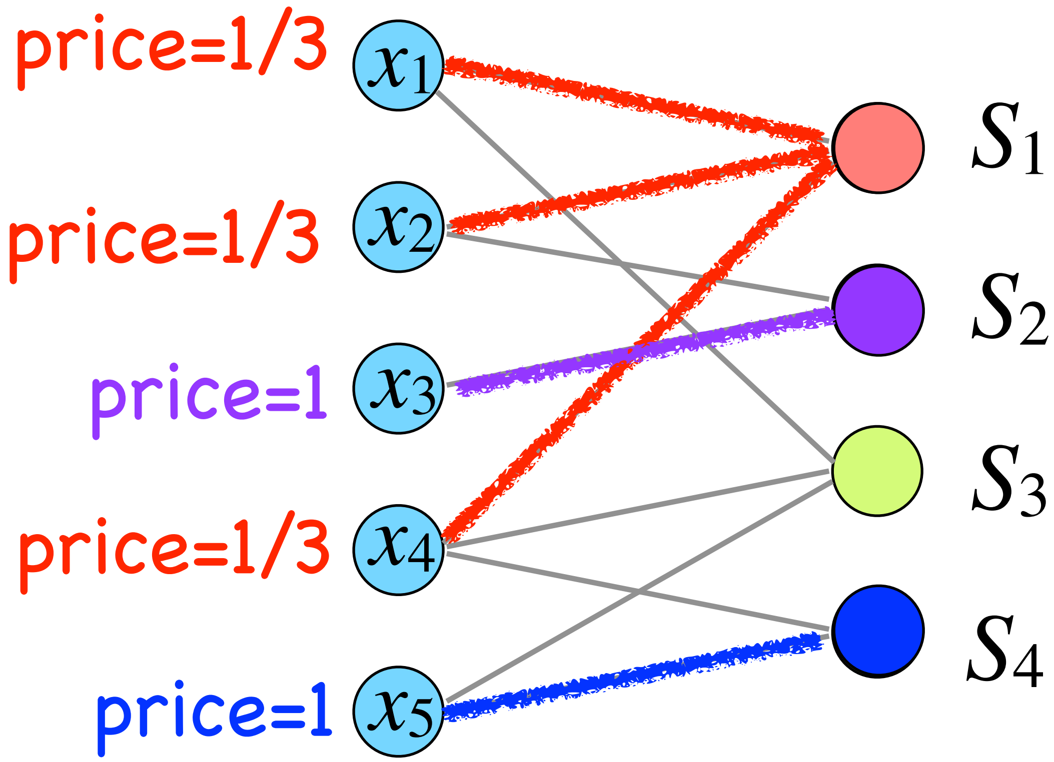
$$|C| = \sum_{x \in U} \mathrm{price}(x)$$

- $x_1, \ldots, x_\ell$: covered in the 1st iteration in *GreedyCover*
- $x_{\ell+1}$: 1st element covered by *GreedyCover* on a new instance I' with $|U'| = |U| - \ell$ and $OPT' \leq OPT$

for $k = \ell + 1$:

$$\mathrm{price}(x_k) \leq \frac{OPT'}{|U'|} \leq \frac{OPT}{|U| - k + 1}$$

Instance: A sequence of subsets $S_1, \ldots, S_m \subseteq U$.



Greedy Cover:

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/$|S_i \cap U|$

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

- x_k : kth element covered by the ***GreedyCover*** algorithm

$$\text{price}(x_k) \leq \frac{OPT}{|U| - k + 1}$$

$$SOL = \sum_{k=1}^{n=|U|} \text{price}(x_k) \leq \sum_{k=1}^{n} \frac{OPT}{n - k + 1} = H_n \cdot OPT$$

Harmonic number

Approximation of Set Cover

Greedy Cover:

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

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

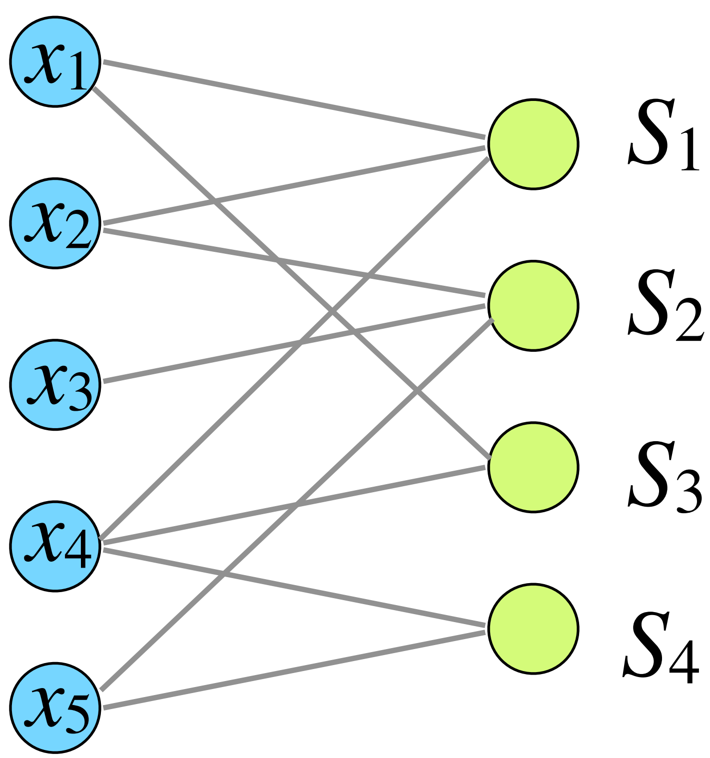
Submodular Optimization

Set Cover with Budget

Instance: A sequence of subsets $S_1, \ldots, S_n \subseteq U$.

(Minimum set cover)
Find the smallest $C \subseteq \{1, \ldots, n\}$ s.t. $\bigcup_{i \in C} S_i = U$.

(Maximum k-cover)
Find $C \subseteq \{1, \ldots, n\}$ with $|C| \leq k$ to maximize $\bigcup_{i \in C} S_i$.



- Objective and constraint are switched.
- Max-k-cover can solve minimum set cover
- Max-k-cover is **NP**-hard

Instance: A sequence of subsets $S_1, \ldots, S_n \subseteq U$.
Find $C \subseteq \{1, \ldots, n\}$ with $|C| \le k$ to maximize $\bigcup_{i \in C} S_i$.

Greedy Cover:

initially $C = \emptyset$;
while $|C| < k$ do:
  add i with largest $|S_i \cap U|$ to C;
  $U = U \backslash S_i$;

- Δ_ℓ : # of elements covered additionally in the ℓth iteration
- Σ_ℓ : # of elements covered within the first ℓ iterations

$$\Sigma_\ell = \Sigma_{\ell-1} + \Delta_\ell \qquad \Sigma_\ell = \sum_{j=1}^{\ell} \Delta_j \qquad SOL = \Sigma_k$$

Greedy Cover:

initially $C = \varnothing$;

while $|C| < k$ do:

  add i with largest $|S_i \cap U|$ to C;

  $U = U \backslash S_i$;

- Δ_ℓ: # of elements covered additionally in the ℓth iteration
- Σ_ℓ: # of elements covered within the first ℓ iterations

$$\Delta_\ell \geq \frac{1}{k}(OPT - \Sigma_{\ell-1})$$

- # of elements covered in OPT but not in the first $\ell - 1$ iterations are $\geq OPT - \Sigma_{\ell-1}$
- There are at most k sets in OPT.
- There is a set in OPT that can cover (in addition to the $\Sigma_{\ell-1}$ elements covered in the first $\ell - 1$ iterations) $\geq \frac{1}{k}(OPT - \Sigma_{\ell-1})$ elements.
- ***GreedyCover*** will select that set (or a better set) in the ℓth iteration.

Greedy Cover:

initially $C = \varnothing$;

while $|C| < k$ do:

  add i with largest $|S_i \cap U|$ to C;

  $U = U \setminus S_i$;

- Δ_ℓ: # of elements covered additionally in the ℓth iteration
- Σ_ℓ: # of elements covered within the first ℓ iterations

$$\Delta_\ell \geq \frac{1}{k}(OPT - \Sigma_{\ell-1}) \Longrightarrow OPT - \Sigma_\ell \leq \left(1 - \frac{1}{k}\right)(OPT - \Sigma_{\ell-1})$$

$$\Downarrow$$

$$\Sigma_\ell - \Sigma_{\ell-1} \geq \frac{1}{k}(OPT - \Sigma_{\ell-1})$$

$$\Uparrow$$

Greedy Cover:

initially $C = \varnothing$;

while $|C| < k$ do:

add i with largest $|S_i \cap U|$ to C;

$U = U \backslash S_i$;

- Δ_ℓ: # of elements covered additionally in the ℓth iteration
- Σ_ℓ: # of elements covered within the first ℓ iterations

$$\Delta_\ell \geq \frac{1}{k}(OPT - \Sigma_{\ell-1}) \implies OPT - \Sigma_\ell \leq \left(1 - \frac{1}{k}\right)(OPT - \Sigma_{\ell-1})$$

$$\implies OPT - \Sigma_k \leq \left(1 - \frac{1}{k}\right)^k OPT \leq \frac{1}{\mathrm{e}} OPT$$

$$\implies SOL = \Sigma_k \geq \left(1 - \frac{1}{\mathrm{e}}\right) OPT$$

$(1 - 1/\mathrm{e})$-approx

- [Feige 1998] There is no poly-time $(1 - 1/\mathrm{e} + \epsilon)$-approximation algorithm unless **NP**=**P**

Submodular Function

Submodular function:

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular if

$$\forall S, T \subseteq [n] : \ f(S \cup T) \leq f(S) + f(T) - f(S \cap T)$$

Proposition: For set function $f : 2^{[n]} \to \mathbb{R}$, define:

$$\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular iff:

$$\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)$$

- Submodular function captures the **law of diminishing marginal productivity (diminishing returns)** in many natural applications

Examples of Submodular Functions

- **Coverage:** given sets $S_1, \ldots, S_n \subseteq \Omega$

$$\forall C \subseteq [n]: \quad f(C) = \left| \bigcup_{i \in C} S_i \right|$$

- **Cut:** graph $G([n], E)$, $\forall S \subseteq [n]: \; f(S) = \left| E(S, V \backslash S) \right|$
- **Linear function:** $\forall S \subseteq [n]: \; f(S) = \sum_{i \in S} w_i$
- **Entropy:** $f(S) = H(X_i : i \in S)$ for random variables $X_1, \ldots, X_n$
- **Matroid rank:** $f(S) = \text{rank}(A_{[m] \times S})$ for $m \times n$ matrix A
- **Facility location, social welfare, influence in a social network, …**

Submodular Function

Submodular function:

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular if

$$\forall S, T \subseteq [n] : \ f(S \cup T) \leq f(S) + f(T) - f(S \cap T)$$

Proposition: For set function $f : 2^{[n]} \to \mathbb{R}$, define:

$$\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular iff:

$$\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)$$

- Submodular function captures the **law of diminishing marginal productivity (diminishing returns)** in many natural applications

Submodularity of Coverage

Proposition: For set function $f : 2^{[n]} \to \mathbb{R}$, define:

$$\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is **submodular** iff:

$$\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is **monotone** if

$$\forall S \subseteq T : f(S) \leq f(T)$$

Instance: A sequence of subsets $S_1, \ldots, S_n \subseteq U$.
Find $C \subseteq \{1,\ldots,n\}$ with $|C| \leq k$ to maximize $\bigcup_{i \in C} S_i$.

$$\forall C \subseteq \{1,\ldots,n\} : \; f(C) = \left| \bigcup_{i \in C} S_i \right|$$

Submodular Maximization

Instance: A *monotone submodular* set function $f : 2^{[n]} \to \mathbb{R}$.

Maximize $f(S)$ subject to $|S| \leq k$. (cardinality constraint)

Greedy Submodular Maximization:

initially $S = \emptyset$;

while $|S| < k$ do:

    add $i \notin S$ with largest $f_S(i)$ into S;

Proposition: For set function $f : 2^{[n]} \to \mathbb{R}$, define:

$$\forall S \subseteq [n], \forall i \in [n] : \quad f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

A set function $f : 2^{[n]} \to \mathbb{R}$ is submodular iff:

$$\forall S \subseteq T, \forall i \notin T : \quad f_S(i) \geq f_T(i)$$

Submodular Maximization

Instance: A *monotone submodular* set function $f : 2^{[n]} \to \mathbb{R}$.

Maximize $f(S)$ subject to $|S| \le k$. (cardinality constraint)

Greedy Submodular Maximization:

initially $S = \varnothing$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into S;

Theorem (Nemhauser, Wolsey, Fisher 1978):

For **monotone submodular** set function $f : 2^{[n]} \to \mathbb{R}_{\ge 0}$, the greedy algorithm gives a $(1 - 1/\mathrm{e})$-approximation of

$$OPT = \max \left\{ f(S) \mid |S| \le k \right\}$$

Greedy Submodular Maximization:

initially $S = \emptyset$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into S;

$f : 2^{[n]} \to \mathbb{R}$

$f_S(i) \triangleq f(S \cup \{i\}) - f(S)$

Submodular:

$\forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i)$

- S : current S in an iteration
- i : the i added into S in that iteration

$$f_S(i) \geq \frac{1}{k}\left(OPT - f(S)\right)$$

- Let S^* be the optimal solution that achieves $OPT = f(S^*)$.

$$OPT - f(S) \leq f_S(S^*) \triangleq f(S^* \cup S) - f(S) \leq \sum_{j \in S^*} f_S(j) \leq k \cdot f_S(i)$$

(monotone) (submodular) (greedy)

Greedy Submodular Maximization:

initially $S = \varnothing$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into S;

$f : 2^{[n]} \to \mathbb{R}$

$$f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

Submodular:

$$\forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i)$$

- S : current S in an iteration
- i : the i added into S in that iteration

$$f_S(i) \geq \frac{1}{k}\left(OPT - f(S)\right)$$

- $S^{(\ell)}$: the S constructed after ℓ iterations

$$f\left(S^{(\ell)}\right) - f\left(S^{(\ell-1)}\right) \geq \frac{1}{k}\left(OPT - f\left(S^{(\ell-1)}\right)\right)$$

$$\implies \quad OPT - f\left(S^{(\ell)}\right) \leq \left(1 - \frac{1}{k}\right)\left(OPT - f\left(S^{(\ell-1)}\right)\right)$$

Greedy Submodular Maximization:

initially $S = \emptyset$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into S;

$f : 2^{[n]} \to \mathbb{R}$

$$f_S(i) \triangleq f(S \cup \{i\}) - f(S)$$

Submodular:

$\forall S \subseteq T, \forall i \notin T : f_S(i) \geq f_T(i)$

- $S^{(\ell)}$: the S constructed after ℓ iterations

$$OPT - f\left(S^{(\ell)}\right) \leq \left(1 - \frac{1}{k}\right)\left(OPT - f\left(S^{(\ell-1)}\right)\right)$$

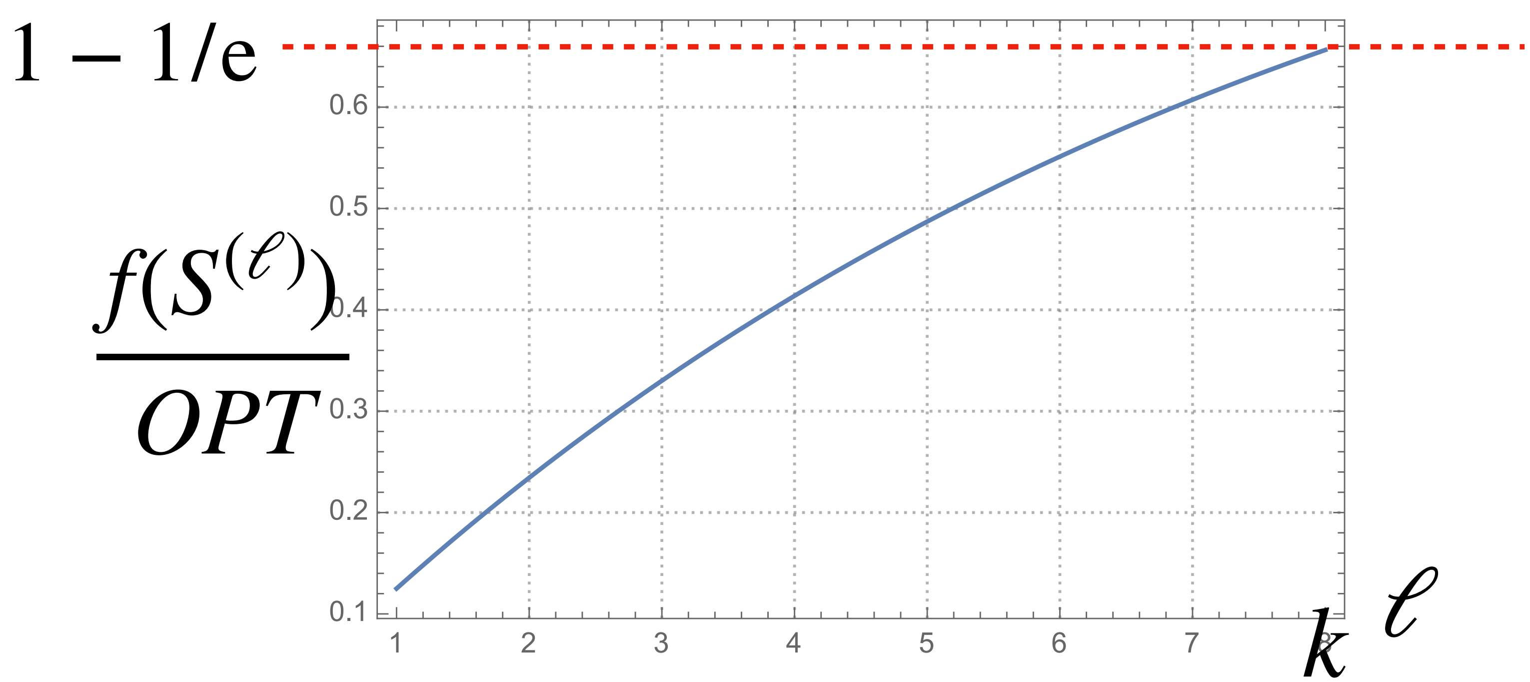
$$\implies OPT - f\left(S^{(k)}\right) \leq \left(1 - \frac{1}{k}\right)^k \left(OPT - f(\emptyset)\right) \leq \frac{1}{\mathrm{e}} OPT$$

$$\implies SOL = f\left(S^{(k)}\right) \geq \left(1 - \frac{1}{\mathrm{e}}\right) OPT$$

Greedy Submodular Maximization:
$S^{(\ell)} \leftarrow S^{(\ell-1)} \cup \{i_\ell\}$ with i_ℓ maximizing $f(S^{(\ell-1)} \cup \{i_\ell\}) - f(S^{(\ell-1)})$

- **Submodularity + monotonicity:**

$$f(S^{(\ell-1)} \cup \{i_\ell\}) - f(S^{(\ell-1)}) \geq \frac{1}{k}\left(OPT - f(S^{(\ell-1)})\right)$$



$$OPT - f(S^{(\ell)}) \leq \left(1 - \frac{1}{k}\right)\left(OPT - f(S^{(\ell-1)})\right)$$

$$\implies OPT - f(S^{(k)}) \leq \left(1 - \frac{1}{k}\right)^k OPT \leq \frac{1}{\mathrm{e}} OPT$$

Submodular Maximization

Instance: A *monotone submodular* set function $f : 2^{[n]} \to \mathbb{R}$.

Maximize $f(S)$ subject to $|S| \leq k$. (cardinality constraint)

Greedy Submodular Maximization:

initially $S = \emptyset$;

while $|S| < k$ do:

add $i \notin S$ with largest $f_S(i)$ into S;

Theorem (Nemhauser, Wolsey, Fisher 1978):

For **monotone submodular** set function $f : 2^{[n]} \to \mathbb{R}_{\geq 0}$, the greedy algorithm gives a $(1 - 1/\mathrm{e})$-approximation of

$$OPT = \max \left\{ f(S) \mid |S| \leq k \right\}$$

Submodular Maximization

MONOTONE MAXIMIZATION

Constraint	Approximation	Hardness	technique
$\|S\| \leq k$	$1 - 1/e$	$1 - 1/e$	greedy
matroid	$1 - 1/e$	$1 - 1/e$	multilinear ext.
$O(1)$ knapsacks	$1 - 1/e$	$1 - 1/e$	multilinear ext.
k matroids	$k + \epsilon$	$k / \log k$	local search
k matroids & $O(1)$ knapsacks	$O(k)$	$k / \log k$	multilinear ext.

NON-MONOTONE MAXIMIZATION

Constraint	Approximation	Hardness	technique
Unconstrained	$1/2$	$1/2$	combinatorial
matroid	$1/e$	0.48	multilinear ext.
$O(1)$ knapsacks	$1/e$	0.49	multilinear ext.
k matroids	$k + O(1)$	$k / \log k$	local search
k matroids & $O(1)$ knapsacks	$O(k)$	$k / \log k$	multilinear ext.

From Prof. Jan Vondrák’s slides “Optimization of Submodular Functions”

Submodular Minimization

Constraint	Approximation	Hardness	alg. technique
Unconstrained	1	1	combinatorial
Parity families	1	1	combinatorial
Vertex cover	2	2	Lovász ext.
k-unif. hitting set	k	k	Lovász ext.
Multiway k-partition	$2 - 2/k$	$2 - 2/k$	Lovász ext.
Facility location	$\log n$	$\log n$	combinatorial
Set cover	n	$n/\log^2 n$	trivial
$\lvert S\rvert \geq k$	$\tilde{O}(\sqrt{n})$	$\tilde{\Omega}(\sqrt{n})$	combinatorial
Shortest path	$O(n^{2/3})$	$\Omega(n^{2/3})$	combinatorial
Spanning tree	$O(n)$	$\Omega(n)$	combinatorial

From Prof. Jan Vondrák's slides "Optimization of Submodular Functions"