Analysis and Design of Algorithms

CS3230

Week 11(Part 2)

+

Week 12 (Part 1)

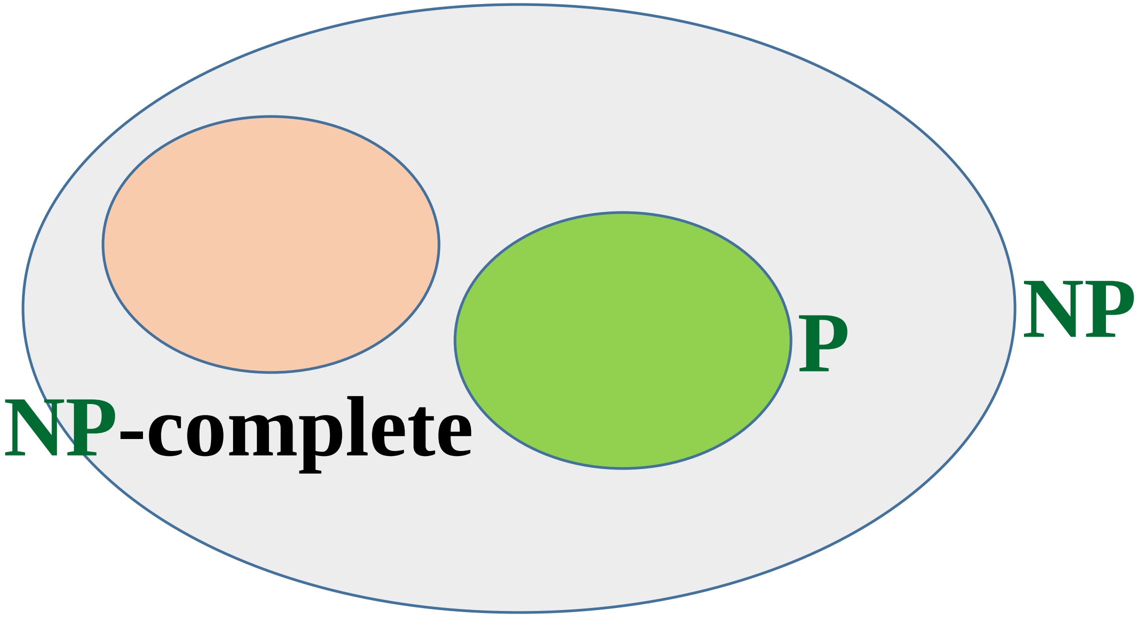
Approximation Algorithms

Diptarka Chakraborty

Ken Sung

NP versus P (Recap)

Is **P** = **NP** ?



If any **NP**-complete problem is solved in polynomial time

➔ P = NP

Problems in NP but believed not to be NP-complete

- **Graph isomorphism** : Given two graphs G and G', is G **isomorphic** to G' ?
- **Integer factoring**: Given an integer, compute all its prime factors.

Decision version: Given an integer n and two integer $2 \leq d_1 < d_2 < n$,

is there any prime factor of n in the range $[d_1, d_2]$?

It belongs to **NP** : Given any integer $x \leq d$, it is possible in polynomial time to determine if x is prime and to determine if x divides n.

Integer factoring is **believed** to be more difficult than problems in **P**, and easier than problems in **NP-complete**.

But no proof exists till now. Research is going on…

What to do when a problem is NP-Complete?

- Unless **P = NP**, NP-Complete problems have no poly time algorithms.

What to do when a problem is NP-Complete?

- Unless **P = NP**, NP-Complete problems have no poly time algorithms.
- But they come up frequently in real life. What can we do?
  - ❑ Can try to solve smaller instances optimally using exponential time algorithms (brute force or cleverer methods such as branch and bound).
  - ❑ Check if the problem instance has special features that make it more efficiently solvable
    - ❑ E.g., 1. If it's a knapsack problem with a small capacity T, then we can use the pseudo-polynomial DP algorithm; 2. SAT solvers.

Approximate Solutions

- A third option for optimization problems is to find a solution that is *nearly optimal* in its cost.

- Recall that an optimization problem looks like:

$$\max/\min C(x)$$

$$\text{such that } x \text{ satisfies some constraints}$$

Approximation Ratio

Let C^* be the optimal cost and C be the cost of the solution given by an approximation algorithm A.

An approximation algorithm A has an **approximation ratio** of $\rho(n)$ if:

$$\frac{C}{C^*} \leq \rho(n) \qquad \text{(for minimization)}$$

$$\frac{C}{C^*} \geq \rho(n) \qquad \text{(for maximization)}$$

Approximation Ratio

Let C^* be the optimal cost and C be the cost of the solution given by an approximation algorithm A.

An approximation algorithm A has an **approximation ratio** of $\rho(n)$ if:

$$\frac{C}{C^*} \leq \rho(n) \qquad \text{(for minimization)} \qquad \boxed{\geq 1}$$

$$\frac{C}{C^*} \geq \rho(n) \qquad \text{(for maximization)} \qquad \boxed{\leq 1}$$

Plan for Today

- ❑ $O(\log n)$-approximation for Set-Cover
- ❑ ½-approximation for Knapsack
- ❑ ½-approximation for Max-Cut
- ❑ $O\left(1/\sqrt{m}\right)$-approximation for Independent-Set

Two general approaches

- **Analyze a heuristic**: A “heuristic” is a procedure that does not always produce the optimal answer. Sometimes, we can show that it’s not too bad though.

- **Solve an LP relaxation**: Many problems can be reduced to Integer Programming. Solve the Linear Programming version instead.

Plan for Today

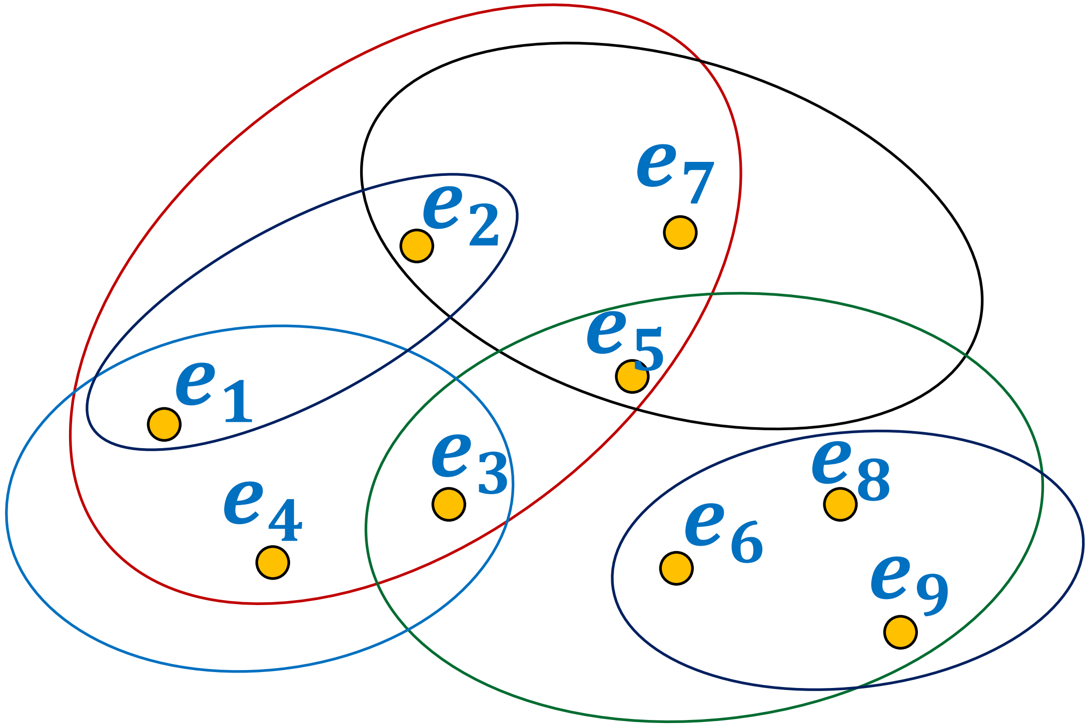
- ❑ **$\boldsymbol{O(\log n)}$-approximation for Set-Cover**
- ❑ ½-approximation for Knapsack
- ❑ ½-approximation for Max-Cut
- ❑ $O\left(\sqrt{m}\right)$-approximation for Independent-Set

Set Cover Problem

- A set $A = \{e_1, e_2, \ldots, e_n\}$
- $S_1, S_2, \ldots, S_k$, with $S_i \subseteq A$

Optimization version: Compute least number of sets that **cover** A.

Decision version: Does there exist j subsets that **cover** all the elements ?

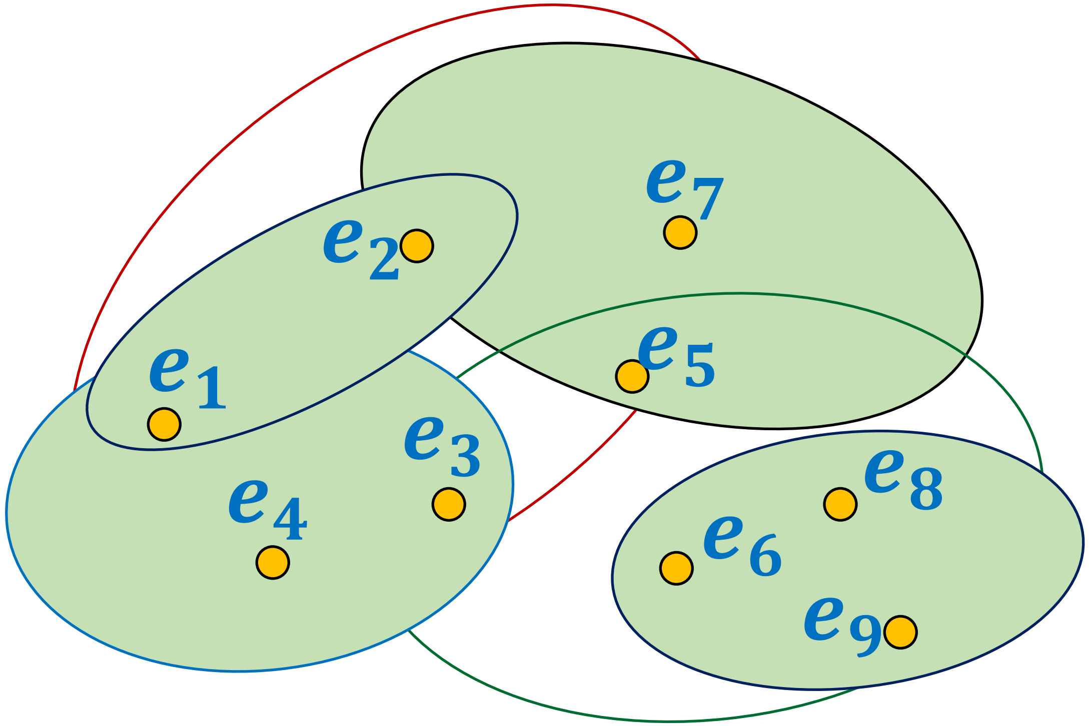


Set Cover Problem

- a set $\boldsymbol{A} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_n}\}$
- $\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k}$, with $\boldsymbol{S_i} \subseteq \boldsymbol{A}$

Optimization version: Compute least number of sets that **cover** $\boldsymbol{A}$.

Decision version: Does there exist $\boldsymbol{j}$ subsets that **cover** all the elements ?

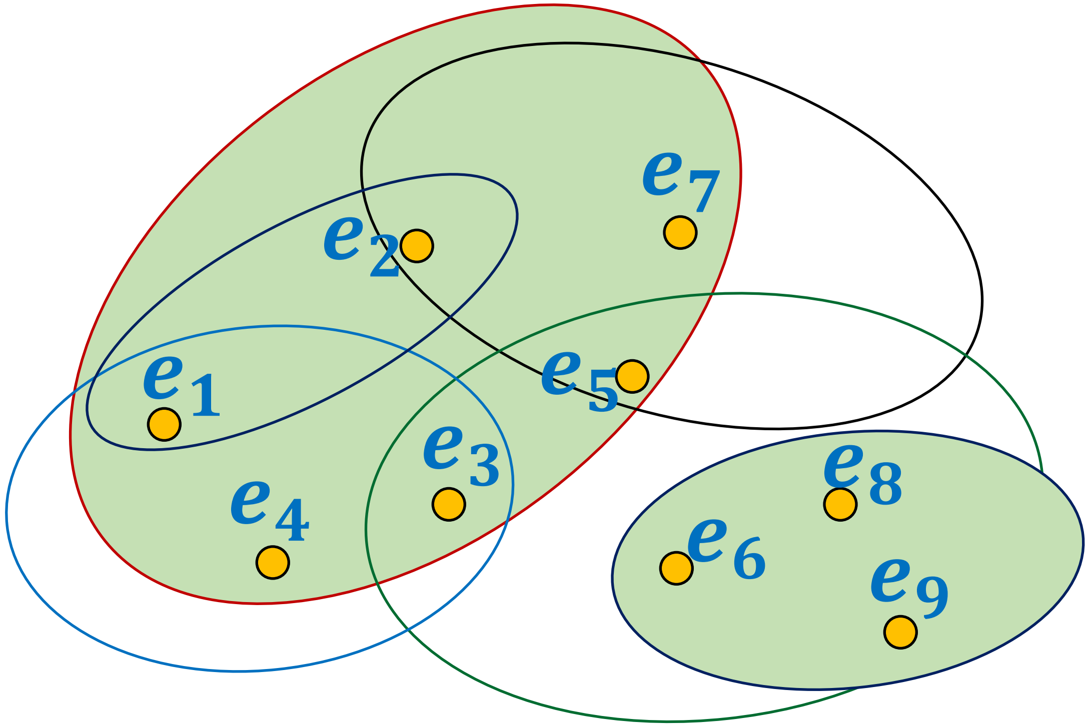


Set Cover Problem

- A set $\boldsymbol{A}$ = $\{\boldsymbol{e_1}, \boldsymbol{e_2},\ldots,\boldsymbol{e_n}\}$
- $\boldsymbol{S_1}, \boldsymbol{S_2},\ldots, \boldsymbol{S_k}$, with $\boldsymbol{S_i} \subseteq \boldsymbol{A}$

Optimization version: Compute least number of sets that **cover** $\boldsymbol{A}$.

Decision version: Does there exist $\boldsymbol{j}$ subsets that **cover** all the elements ?



Set Cover Problem

- A set $\boldsymbol{A} = \{\boldsymbol{e_1}, \boldsymbol{e_2}, \ldots, \boldsymbol{e_n}\}$
- $\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_k},$ with $\boldsymbol{S_i} \subseteq \boldsymbol{A}$

Optimization version: Compute <u>least number of sets</u> that **cover** $\boldsymbol{A}$.

Decision version: **NP**-complete

Approximation algorithm for the <u>optimization version</u>:

$\boldsymbol{R} \leftarrow \boldsymbol{A}$;

While $\boldsymbol{R}$<> empty **do**

{ Pick a set $\boldsymbol{S_i}$ such that $|\boldsymbol{R} \cap \boldsymbol{S_i}|$ is maximum;

Remove all elements $\boldsymbol{R} \cap \boldsymbol{S_i}$ from $\boldsymbol{R}$;

}

Return all subsets <u>picked</u> in the while loop.

Set Cover Problem

- A set A = $\{e_1, e_2, \ldots, e_n\}$
- $S_1, S_2, \ldots, S_k$, with $S_i \subseteq A$
- Set S_i has cost C_i

Optimization version: Compute sets of least cost that **cover** A.

Decision version: **NP**-complete

Approximation algorithm for the optimization version:

$R \leftarrow A$;

While R<> empty **do**

{ Pick a set S_i such that $\frac{C_i}{|R \cap S_i|}$ is **minimum**;

Remove all elements $R \cap S_i$ from R;

}

Return all subsets picked in the while loop.

How to analyze the greedy algorithm

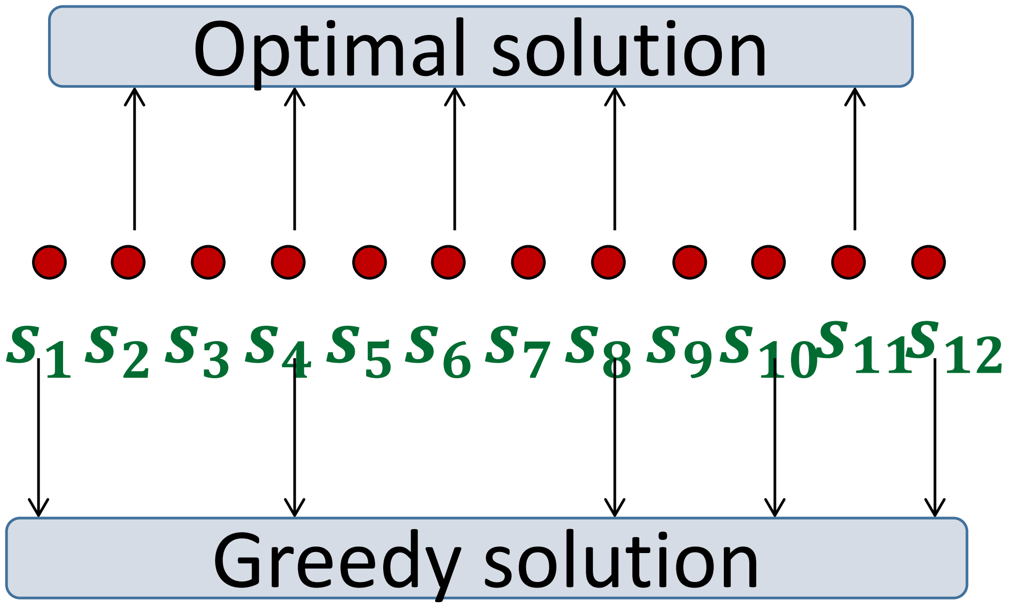
The challenge:

- No knowledge of the **optimal solution**.
- Aim to get a **worst case guarantee** for all possible instances.

Conquering the challenge:

- Pick any **arbitrary** instance.
- “**Compare**” greedy solution with its optimal solution.

How to analyze the greedy algorithm

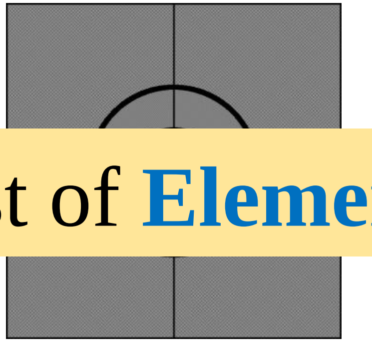


Question: How to account for the sets s_2, s_6, s_{11} in the greedy algorithm ?

(the sets belonging to "**Optimal**" but absent in "**Greedy**")

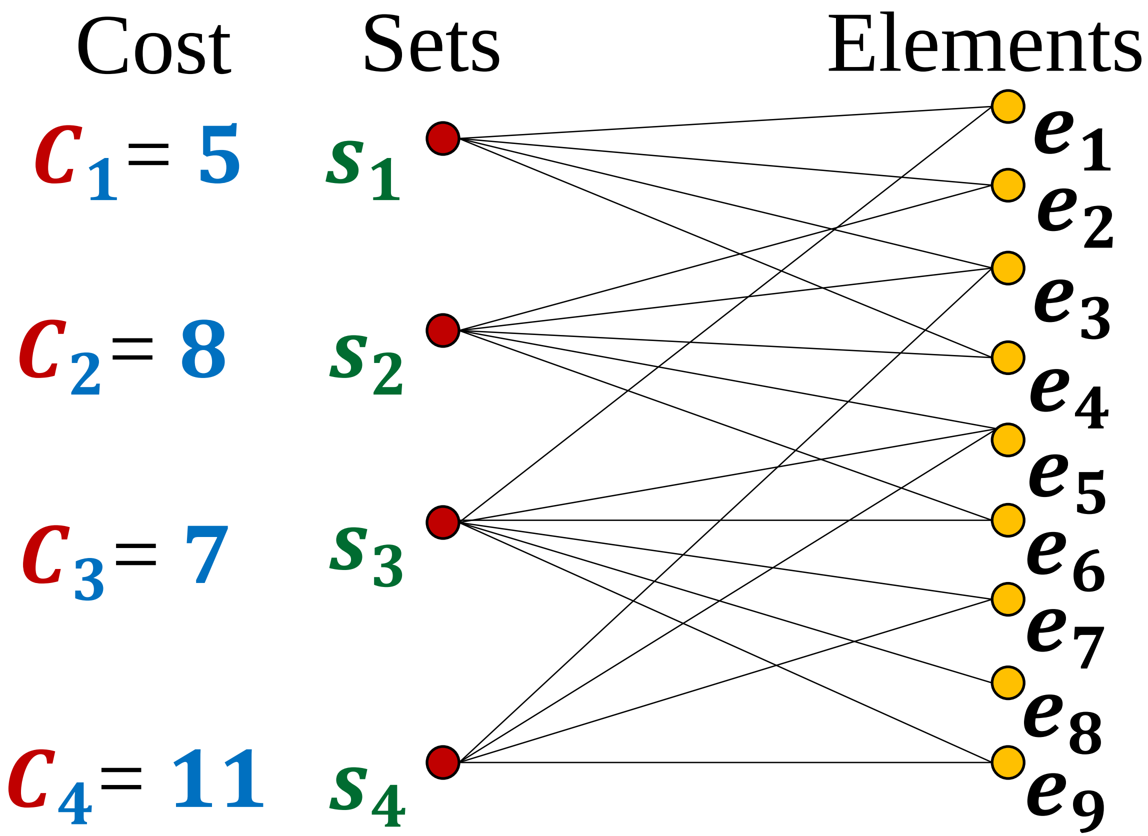
How to analyze the greedy algorithm

The key to analysis:



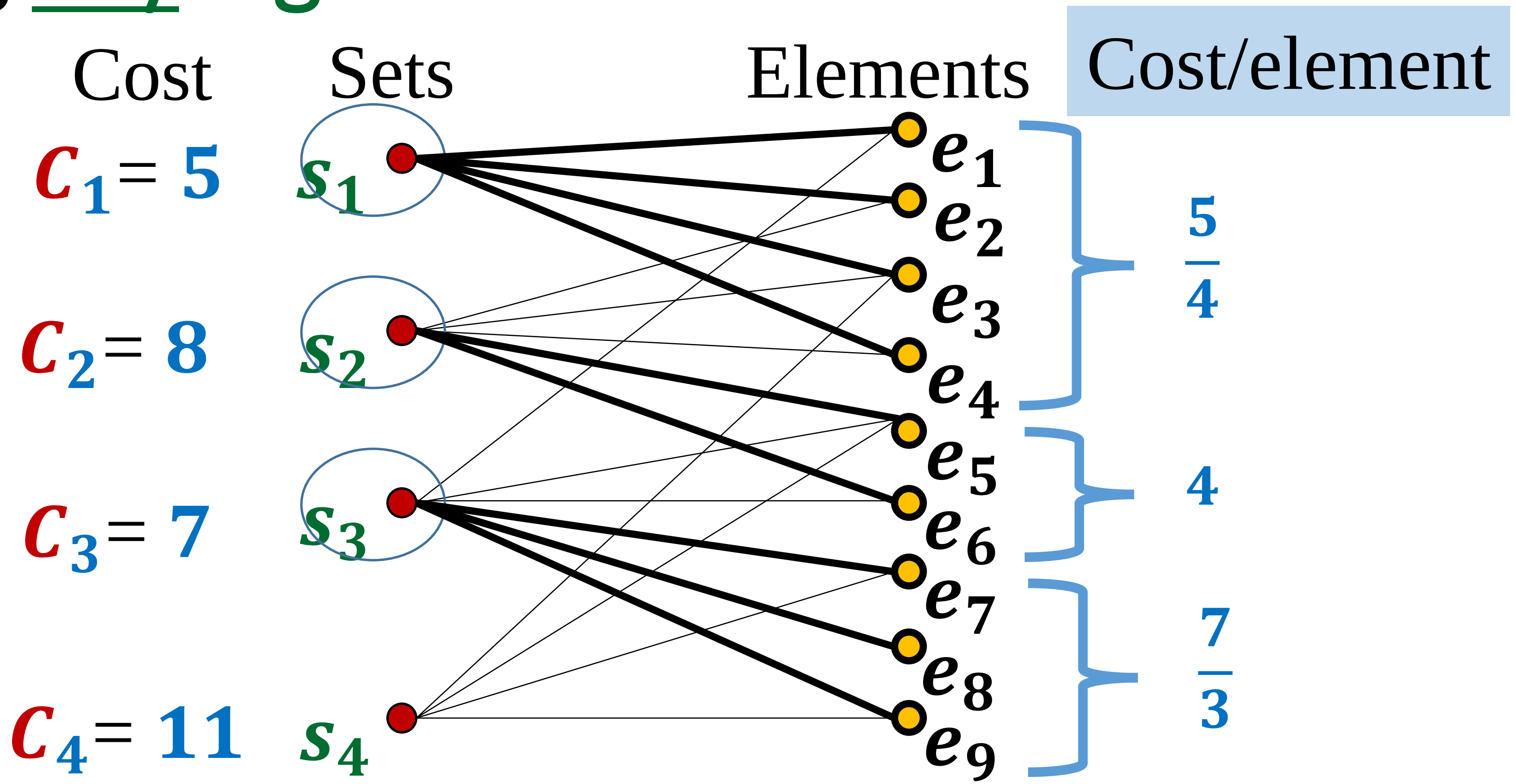
Cost of **Elements**

Cost of each set ➔ cost of each element



Cost of each set ➔ cost of each element

Viewing any algorithm



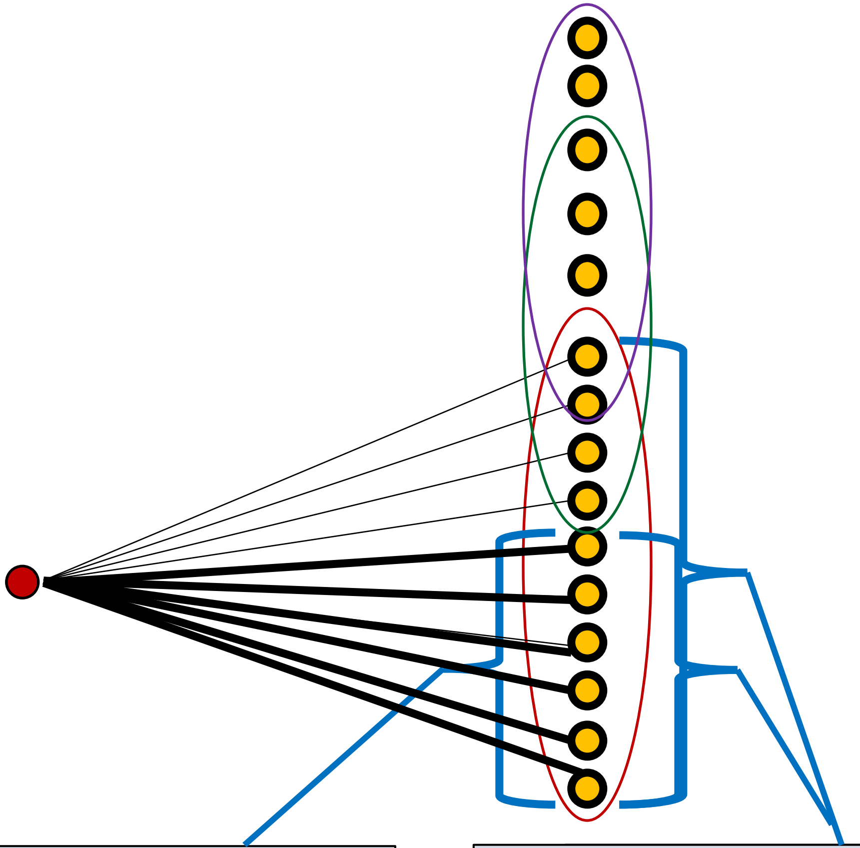
Sum of cost of **sets selected** = sum of cost paid for each **element**.

Greedy algorithm: Select the set with **least cost per element**

Any set s_i : present in Optimal but not in Greedy

Cost = c_i s_i

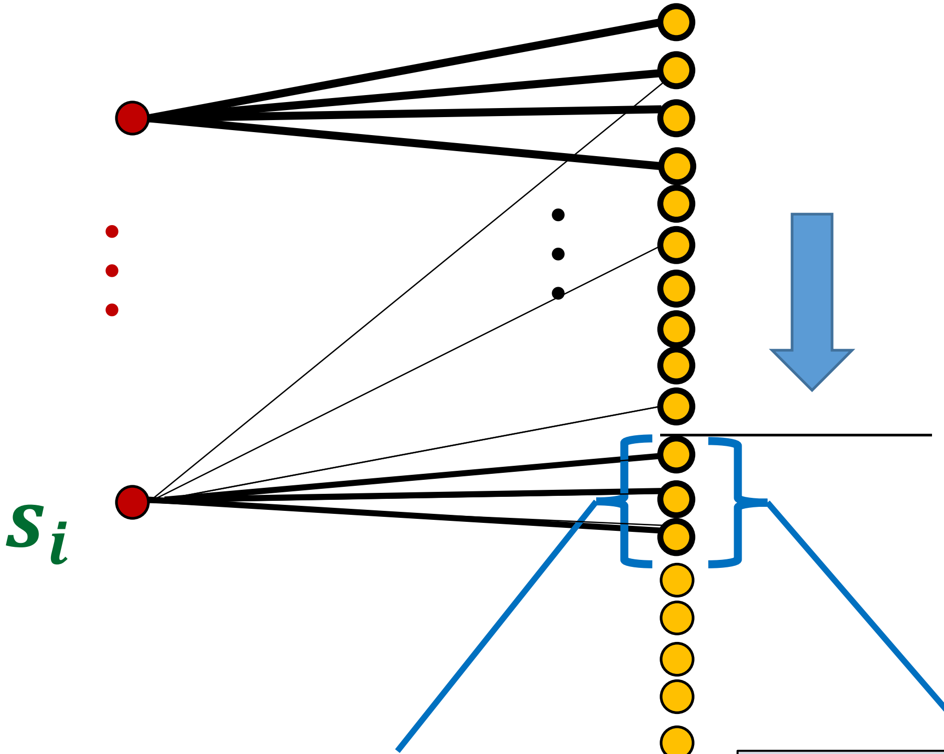
$|s_i| = m_i$



Cost of these elements in optimal algorithm = c_i

Cost of all the elements of s_i in greedy algorithm $\leq c_i \log m_i$

Any sequence of selecting the Optimal set cover



Cost of these elements in optimal algorithm = c_i

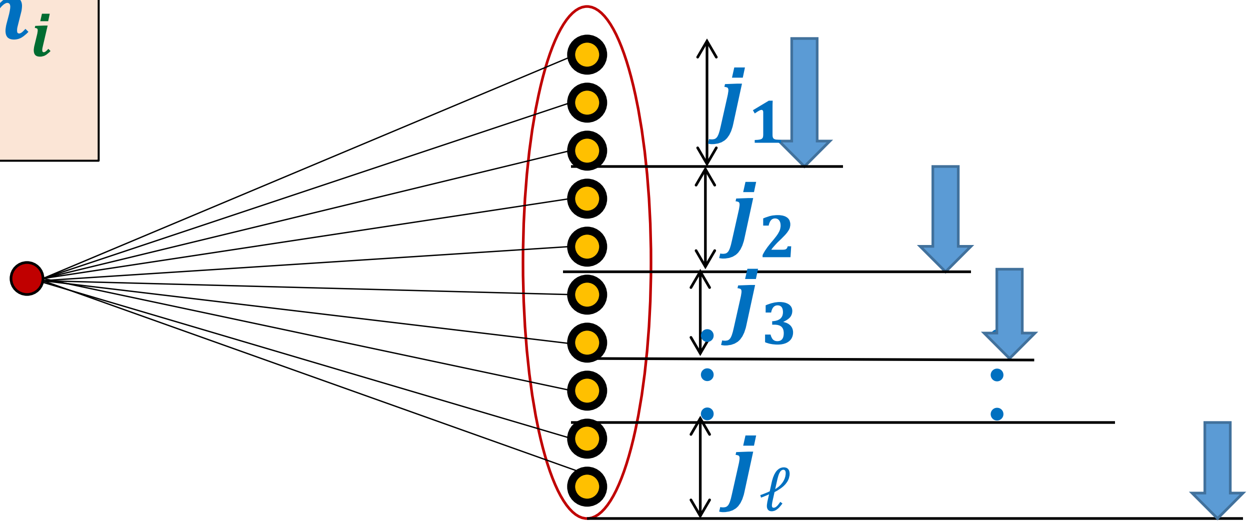
Cost of these elements in greedy algorithm $\leq c_i\ \log\ m_i$

The core of the analysis

$$\sum_{k=1}^{k=\ell} j_k = m_i$$

Cost = $C_i \; s_i$

$|s_i| = m_i$

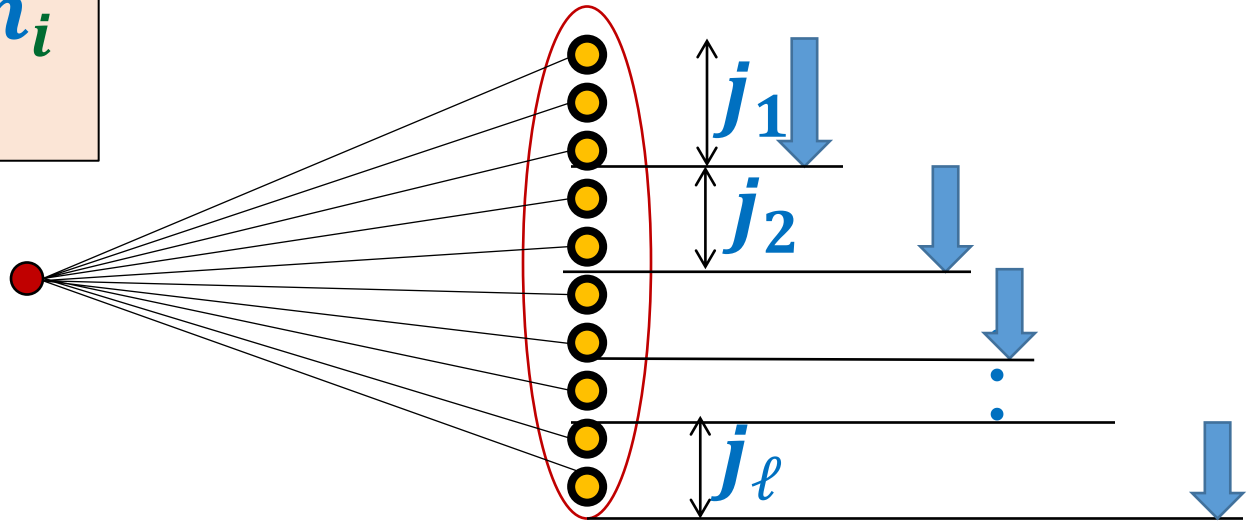


But what forced our greedy algorithm to not include s_i?

$$\sum_{k=1}^{k=\ell} j_k = m_i$$

Cost $= C_i \;\; s_i$

$|s_i| = m_i$



Cost of first j_1 elements covered by greedy $\leq \frac{C_i}{m_i} j_1$

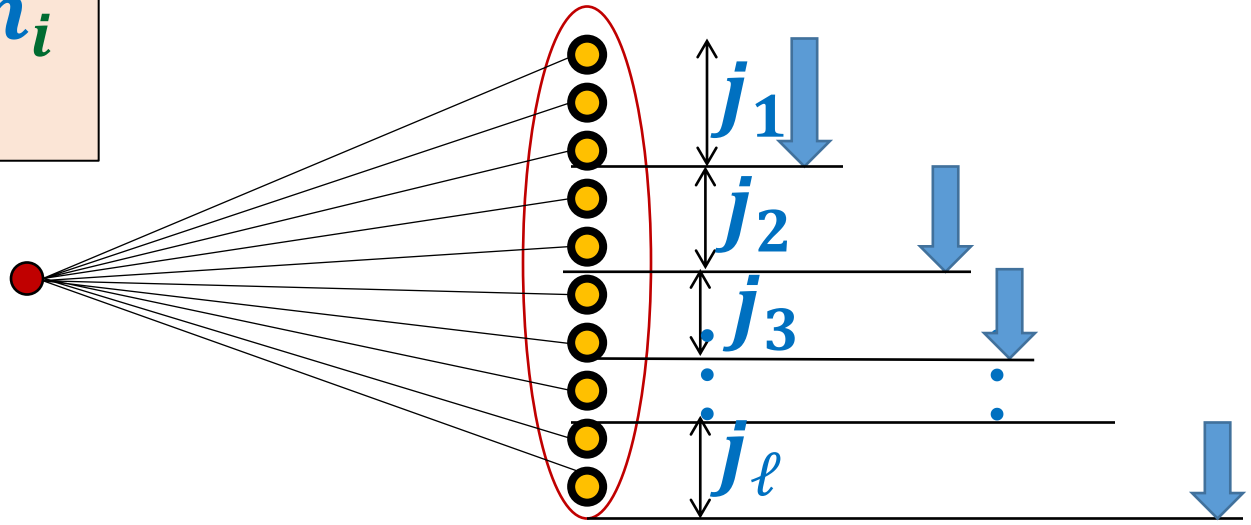
Cost of next j_2 elements covered by greedy $\leq \frac{C_i}{m_i - j_1} j_2$

Cost of last j_ℓ elements covered by greedy $\leq \frac{C_i}{m_i - j_1 - j_2 - \cdots - j_{\ell-1}} j_\ell$

$$\boxed{\sum_{k=1}^{k=\ell} j_k = m_i}$$

Cost = C_i $\quad s_i$

$|s_i| = m_i$



Cost of covering all elements of s_i by greedy =

$$\le \frac{C_i}{m_i} j_1 + \frac{C_i}{m_i - j_1} j_2 + \frac{C_i}{m_i - j_1 - j_2} j_3 + \cdots + \frac{C_i}{m_i - j_1 - j_2 - \cdots - j_{\ell-1}} j_\ell$$

$$\le C_i \left(\frac{j_1}{m_i} + \frac{j_2}{m_i - j_1} + \frac{j_3}{m_i - j_1 - j_2} + \cdots + \frac{j_\ell}{m_i - j_1 - j_2 - \cdots - j_{\ell-1}} \right)$$

$$\le C_i \log m_i$$

Recap

Theorem: The greedy algorithm achieves an **approximation ratio** of **O**($\log n$) for <u>every</u> instance of set cover problem.

Running time of greedy algorithm: **O**(nk)

Question: Any polynomial algorithm for set cover with approx. ratio < $\log n$?

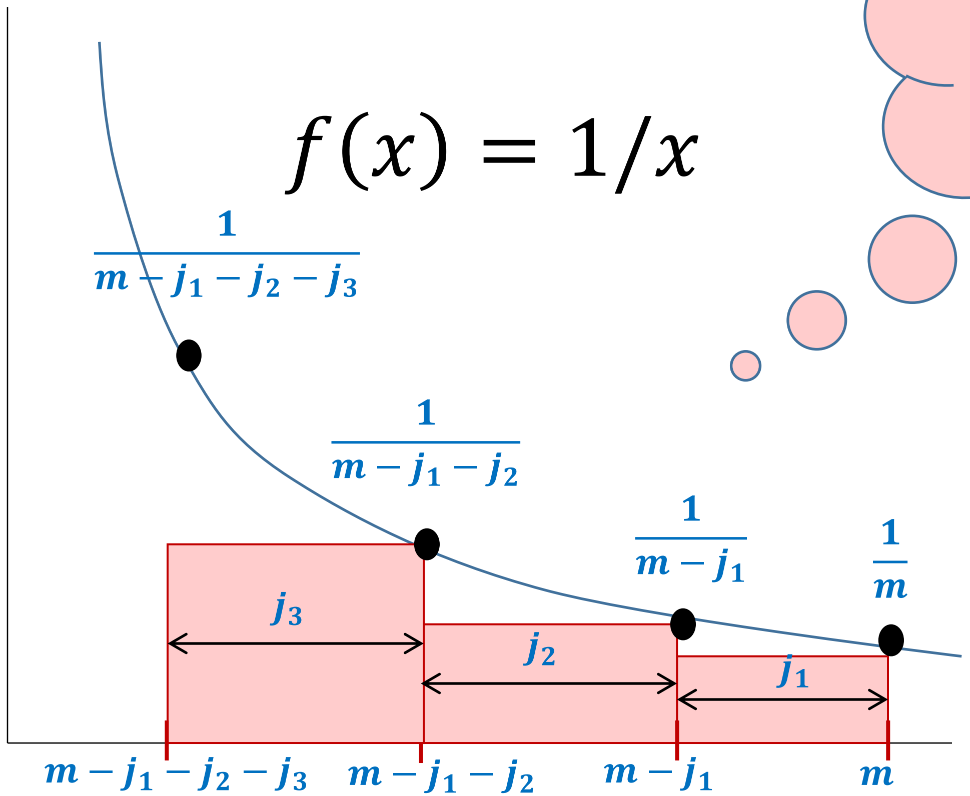
Answer: It is impossible unless "**P** = **NP**".

The next slide is for the **visual proof** for the discrete math problem of last slide

Let $j_1, j_2, \ldots, j_\ell$ be any arbitrary ℓ positive integers such that $\sum_{t=1}^{t=\ell} j_t < m$

Show that $\frac{j_1}{m} + \frac{j_2}{m-j_1} + \frac{j_3}{m-j_1-j_2} + \cdots + \frac{j_\ell}{m-j_1-j_2-\cdots-j_{\ell-1}} \le \log m$

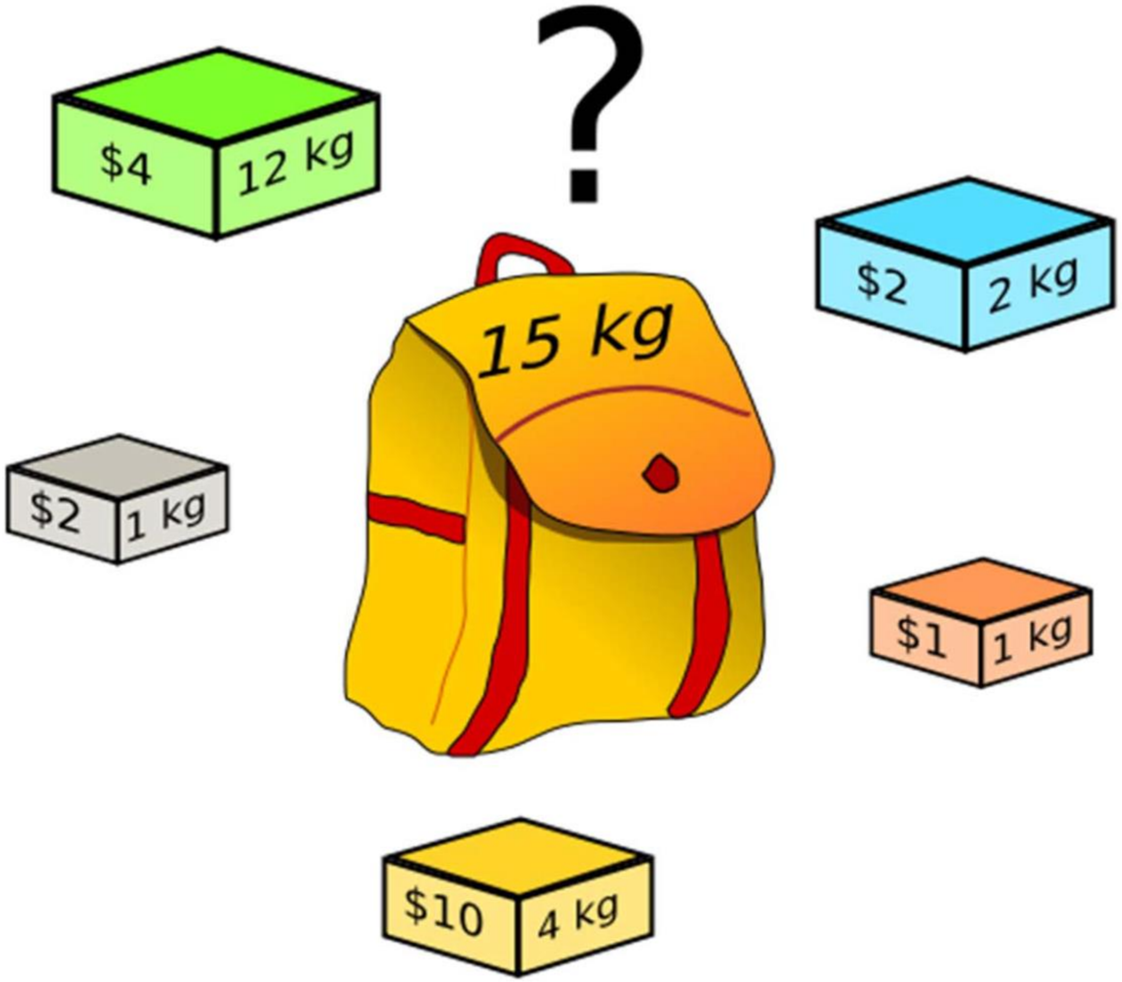
Relate the pink rectangles with the terms in the expression



Plan for Today

- ❑ **$O(\log n)$-approximation for Set-Cover**
- ❑ **½-approximation for Knapsack**
- ❑ ½-approximation for Max-Cut
- ❑ $O\left(1/\sqrt{m}\right)$-approximation for Independent-Set

Knapsack Problem



What is the maximum value you can get?

$15

Formal Definition

KNAPSACK

Input:

$(w_1, v_1), (w_2, v_2), \ldots, (w_n, v_n)$, and W

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes

$\Sigma_{i\in S} v_i$ such that $\Sigma_{i\in S} w_i \leq W$

Fractional Knapsack?

- Recall the greedy algorithm for FRACTIONAL KNAPSACK. Can be described as:
  - ❑ Assume $\frac{v_1}{w_1}, \ldots, \frac{v_n}{w_n}$ are in decreasing order.
  - ❑ Add items in this order while the total weight is at most W.
  - ❑ For the first item that cannot fit fully, put in the largest fraction possible so that the sum of weights is exactly W.

- Can we modify this algorithm for the integral knapsack problem?

Question 1

Show that the greedy algorithm which puts in items in order of decreasing value/kg until the weight is at most W has a very small approximation ratio.

Question 1: Solution

Suppose the input has two items: one of value 2 and weight 1, and the other of value T $(= W)$ and weight W.

Greedy algorithm would only put the first item in. Optimal solution puts the second item in. The approximation ratio is: $2/W$. This goes to zero as W becomes large.

Our Heuristic

Consider another step of comparing

- ❑ The greedy solution
- ❑ The item with the largest value and weight at most W and selecting the better of the two choices.

Call this the **modified greedy knapsack algorithm**. Surprisingly, approximation ratio for this algorithm is 0.5!

When is greedy much smaller than fractional knapsack?

- Order the items in decreasing value/kg.

- Suppose the fractional knapsack solution picks items 1 through k and some fraction of item $k+1$. Then, the greedy solution also picks items 1 through k but **not** item $k+1$.

- So, if the fractional knapsack solution is much larger than greedy, then it must contain a lot of item $k+1$'s weight.

Analysis of modified greedy

Assume that there's no item of weight $> W$. Let i_m be the item of maximum value. Let $V_g, V_g', V_{\text{f-opt}}, V_{\text{opt}}$ be the values of greedy, modified greedy, optimal fractional knapsack, and optimal knapsack solutions respectively.

Then:

$$V_{\text{f-opt}} \geq V_{\text{opt}}$$

So: $V_g' \geq \frac{1}{2} \cdot V_{\text{opt}}$, and so approx ratio is ½.

Analysis of modified greedy

Assume that there's no item of weight $> W$. Let i_m be the item of maximum value. Let $V_g, V_g', V_{\mathrm{f-opt}}, V_{\mathrm{opt}}$ be the values of greedy, modified greedy, optimal fractional knapsack, and optimal knapsack solutions respectively.

Then:

$$2V_g' = V_g' + V_g' \geq V_g + v_{i_m} \geq V_g + v_{k+1} \geq V_{\mathrm{f-opt}} \geq V_{\mathrm{opt}}$$

We are actually showing something stronger here!

So: $V_g' \geq \frac{1}{2} \cdot V_{\mathrm{opt}}$, and so approx ratio is ½.

Recap

- We showed a simple algorithm that returns a solution with value at least half of the fractional knapsack optimum, hence at least half of the knapsack optimum.

- One can even get a **fully polynomial time approximation scheme (FPTAS)** for knapsack!
  - For any $0 < \epsilon < 1$, there is an algorithm that has approx ratio $1 - \epsilon$ and running time $\mathrm{poly}\left(\frac{n}{\epsilon}\right)$.

Question 2

Consider an optimization problem where the objective is to maximize $C_1(x) + C_2(x)$, where C_1 and C_2 are non-negative cost functions.

Suppose $C_1(x)$ and $C_2(x)$ can individually be maximized in polynomial time. Let x_1 maximize $C_1(x)$ and let x_2 maximize $C_2(x)$.

What is the approx. ratio of the algorithm that outputs x_1 if $C_1(x_1) > C_2(x_2)$ and x_2 otherwise?

A) 0

B) 1/3

C) 1/2

D) 1

Question 2: Solution

C) ½

Let x^* be an optimal solution that maximizes $C_1(x) + C_2(x)$.

Suppose $C_1(x^*) \geq C_2(x^*)$. Then:

$$C_1(x_1) \geq C_1(x^*) \geq (C_1(x^*) + C_2(x^*))/2$$

Similarly, if $C_2(x^*) > C_1(x^*)$, then:

$$C_2(x_2) \geq C_2(x^*) \geq (C_1(x^*) + C_2(x^*))/2$$

Since we return the maximum of $C_1(x_1)$ and $C_2(x_2)$, the approx. ratio is ½.

Plan for Today

- ❑ **$O(\log n)$-approximation for Set-Cover**
- ❑ **½-approximation for Knapsack**
- ❑ **½-approximation for Max-Cut**
- ❑ $O(1/\sqrt{m})$-approximation for Independent-Set

Max-Cut

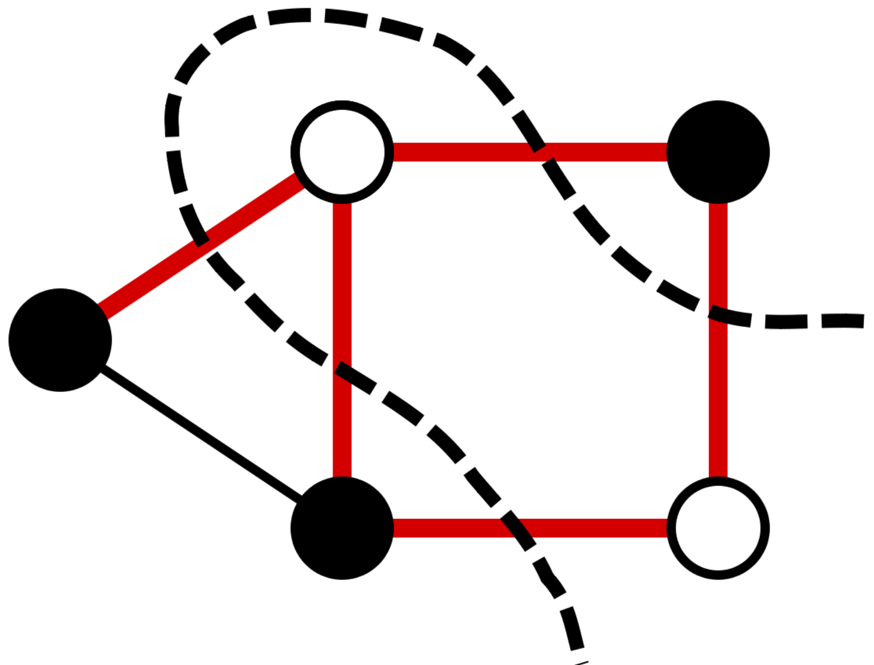
Input: A weighted graph $G = (V, E)$ with each edge e having a non-negative weight $w(e)$

Output: Find a subset $S \subseteq V$ that maximizes

$$\sum_{\substack{u \in S, v \notin S \\ (u,v) \in E}} w((u, v))$$

Max-Cut

- Arises commonly in graph partitioning problems
- One of Karp's original 21 NP-complete problems



Randomized Approximation

Let C^* be the optimal cost and C be the **expected** cost of the solution given by a **randomized** approximation algorithm A.

A randomized approximation algorithm A has an **approximation ratio** of $\rho(n)$ if:

$$\frac{C}{C^*} \leq \rho(n) \qquad \text{(for minimization)}$$

$$\frac{C}{C^*} \geq \rho(n) \qquad \text{(for maximization)}$$

Approximating Max-Cut

Here is a simple algorithm for Max-Cut:

$S \leftarrow \emptyset$

foreach $v \in V$

  $S \leftarrow S \cup \{v\}$ with probability ½

return S

Analysis

Claim:

$$\mathbb{E}\left[\sum_{u\in S, v\notin S} w\big((u,v)\big)\right] = \frac{1}{2}\sum_{e\in E} w(e)$$

Analysis

Claim:

$$\mathbb{E}\left[\sum_{u\in S, v\notin S} w\big((u,v)\big)\right] = \frac{1}{2}\sum_{e\in E} w(e)$$

Proof: For an edge $e \in E$, let X_e be the indicator variable that one endpoint of e is in S and the other isn't. Note:

$$\Pr[X_e = 1] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}.$$

The LHS is $\mathbb{E}[\sum_e X_e \cdot w(e)] = \sum_e w(e)\mathbb{E}[X_e] = \frac{1}{2} \cdot \sum_e w(e)$.

Finishing up

- We found a cut whose expected weight is at least half of the total weight. But clearly, the max cut weight *opt* is at most the total weight.

$$\mathbb{E}\left[\sum_{u \in S, v \notin S} w\big((u,v)\big)\right] = \frac{1}{2} \cdot \sum_{e} w(e) \geq \frac{1}{2} \cdot opt$$

- Getting a better than ½ approximation seems to require significantly new ideas. Best known approx. factor is ≈0.87 and conjectured that this is tight!

Question 3

Consider the MAX-2-SAT problem. Here we are given a list of clauses $C_1, \ldots, C_m$ on n Boolean variables $x_1, \ldots, x_n$ where each clause is the OR of two literals. The goal is to find an assignment that satisfies the maximum number of clauses. The problem is NP-complete.

Example: $C_1 = x_1 \lor x_2, C_2 = \overline{x_1} \lor x_2$. Here, $x_1 = 1, x_2 = 1$ satisfies both of the 2 clauses.

What is the approximation factor of the algorithm that just outputs a random assignment?

A) ¼
B) ½
C) ¾
D) 1

Question 4: Solution

C) ¾

A random assignment satisfied each clause with probability ¾ (only with probability ¼, both literals in the clause are false). So, the expected number of clauses satisfied is $\frac{3}{4}m \geq \frac{3}{4} \cdot opt$.

Plan for Today

- ❑ $O(\log n)$-approximation for Set-Cover
- ❑ ½-approximation for Knapsack
- ❑ ½-approximation for Max-Cut
- ❑ $O\left(1/\sqrt{m}\right)$-approximation for Independent-Set

LP Relaxation

- Extremely powerful technique!

- 4 stages:
  1. **Reduce** problem to integer programming
  2. **Relax** integral constraints, so that we get a linear program
  3. **Solve** linear program in polynomial time
  4. **Round** fractional solution to integral solution.

Some Comment

- We know an $O\left(\frac{1}{\sqrt{m}}\right)$-approximation for Independent-Set. If $m = \Omega(n^2)$, then this becomes an $O\left(\frac{1}{n}\right)$-approximation which is trivial. **(Why?)**

- It is known that it's NP-hard to get an $1/n^{1-\alpha}$-approximation for any constant $\alpha > 0$.

Acknowledgement

- The slides are modified from
  - The slides from Prof. Kevin Wayne
  - The slides from Prof. Surender Baswana
  - The slides from Prof. Erik D. Demaine and Prof. Charles E. Leiserson
  - The slides from Prof. Arnab Bhattacharya and Prof. Wing-Kin Sung