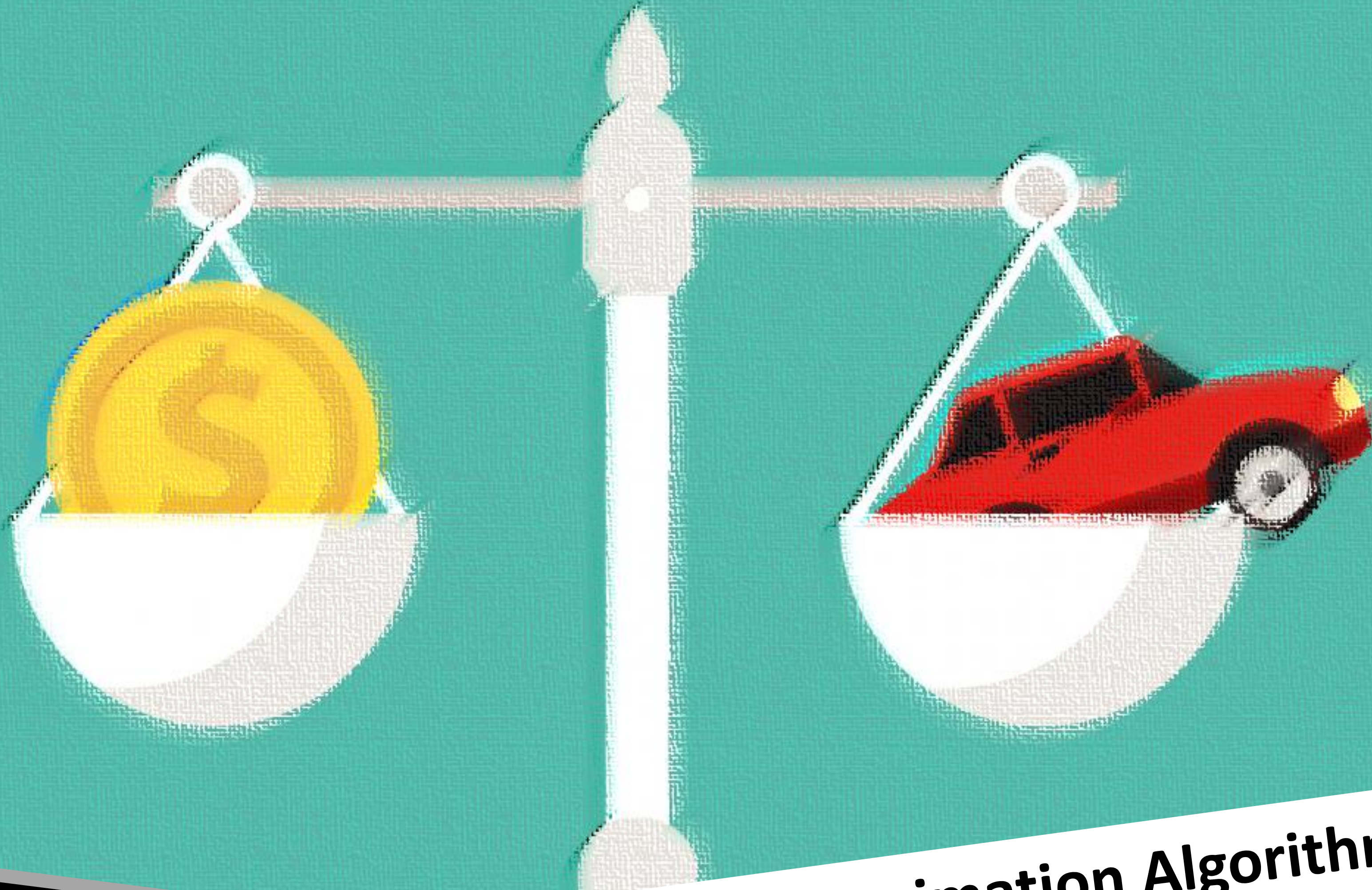


Approximation Algorithms

Dec 27th, 2018

Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學 National Taiwan University

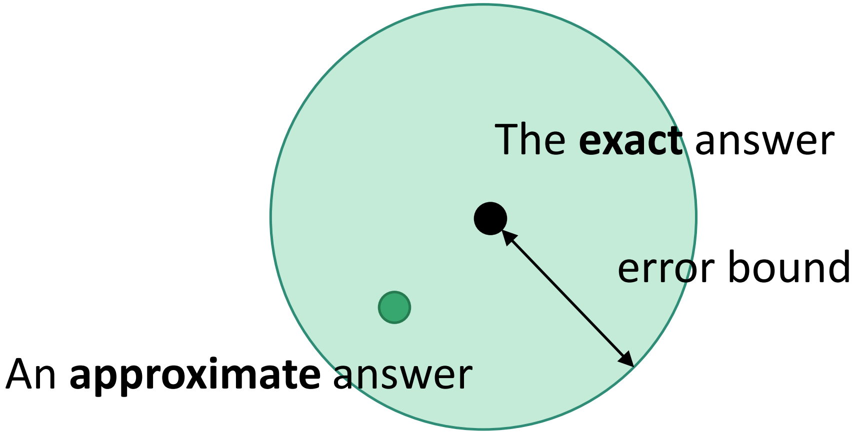
Slides credited from Hsueh-I Lu & Hsu-Chun Hsiao

Outline

- Approximation Algorithms
- Examples
  - Vertex Cover
  - Traveling Salesman Problem
  - Set Cover
  - 3-CNF-SAT

Approximation

- "A value or quantity that is nearly but not exactly correct"
- **Approximation algorithms for optimization problems**: the approximate solution is guaranteed to be close to the exact solution (i.e., the optimal value)
  - Cf. heuristics search: no guarantee
  - Note: we cannot approximate decision problems



Why Approximation?

- Most practical optimization problems are NP-hard
  - It is widely believed that P ≠ NP
  - Thus, polynomial-time algorithms are unlikely, and we must sacrifice either **optimality**, **efficiency**, or **generality**
- Approximation algorithms sacrifice **optimality,** return **near-optimal** answers
  - How “near” is near-optimal?

Approximation Algorithms

- $\rho(n)$-approximation algorithm
- Approximation ratio $\rho(n)$
  - *n*: input size
  - C^*: cost of an optimal solution
  - *C*: cost of the solution produced by the approximation algorithm

$$\max\left(\frac{C}{C*}, \frac{C^*}{C}\right) \leq \rho(n)$$

Maximization problem: $C^*/C \leq \rho(n)$

Minimization problem: $C/C^* \leq \rho(n)$

Approximation Ratio $\rho(n)$

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

n: input size
C^*: cost of an optimal solution
C: cost of an approximate solution

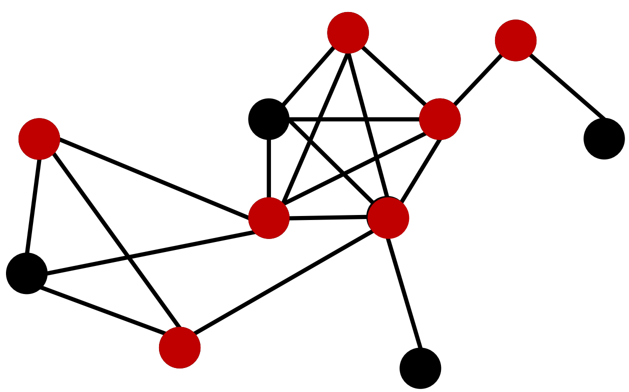
- $\rho(n) \geq 1$
- Smaller is better ($\rho(n) = 1$ indicates an exact algorithm)
- Challenge: prove that *C* is close to C^* without knowing C^*

7 Vertex Cover

Textbook 35.1 – The vertex-cover problem

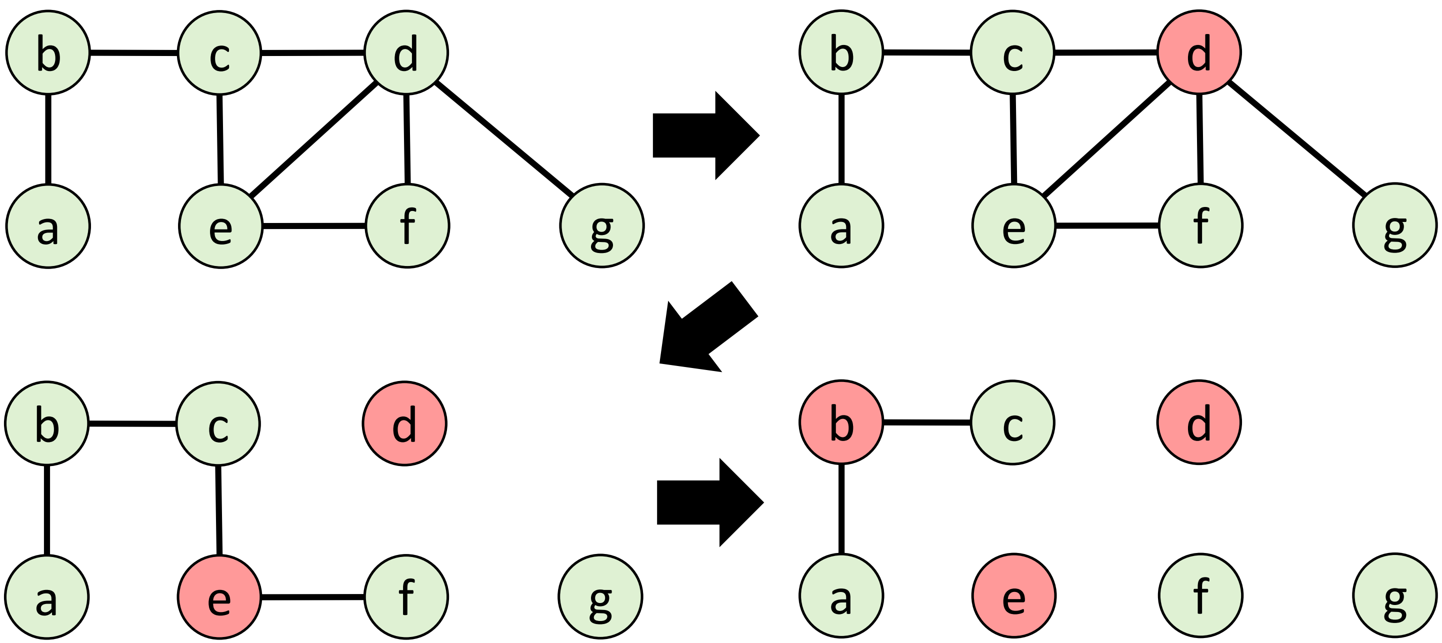
Vertex Cover Problem

- A vertex cover of G = (V, E) is a subset V' ⊆ V s.t. if (w, v) ∈ E, then w ∈ V' or v ∈ V'
  - A vertex cover "covers" every edge in G
- Optimization problem: find a minimum size vertex cover in G ⇨ NP-complete
- Decision problem: is there a vertex cover with size smaller than *k*



Greedy Heuristic Algorithm

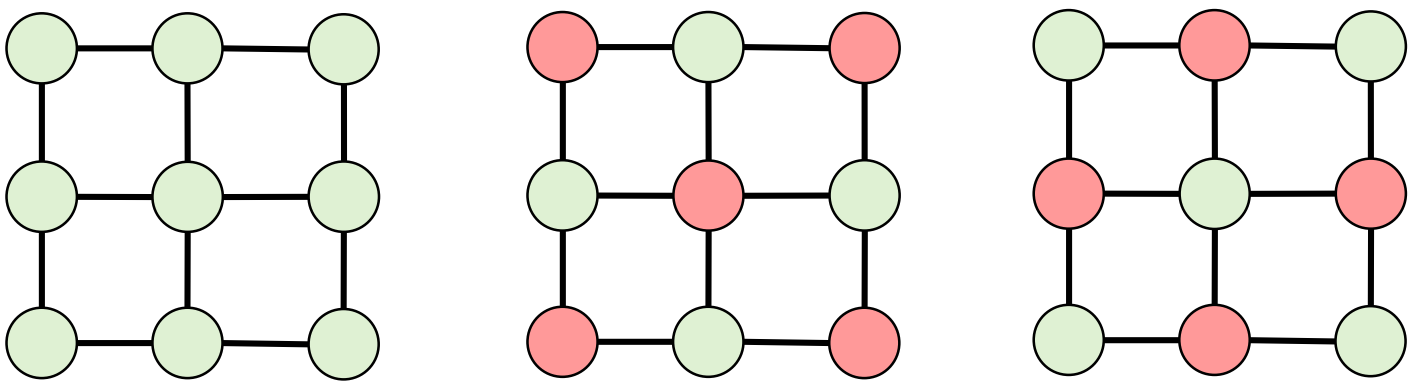
- Idea: cover as many edges as possible (vertex with the maximum degree) at each stage and then delete the covered edges



{b, d, e} is the optimal solution!

Greedy Heuristic Algorithm

- Idea: cover as many edges as possible (vertex with the maximum degree) at each stage and then delete the covered edges
- The greedy heuristic cannot always find optimal solution (otherwise P=NP is proven)



- There is no guarantee that C is always close to C^* either

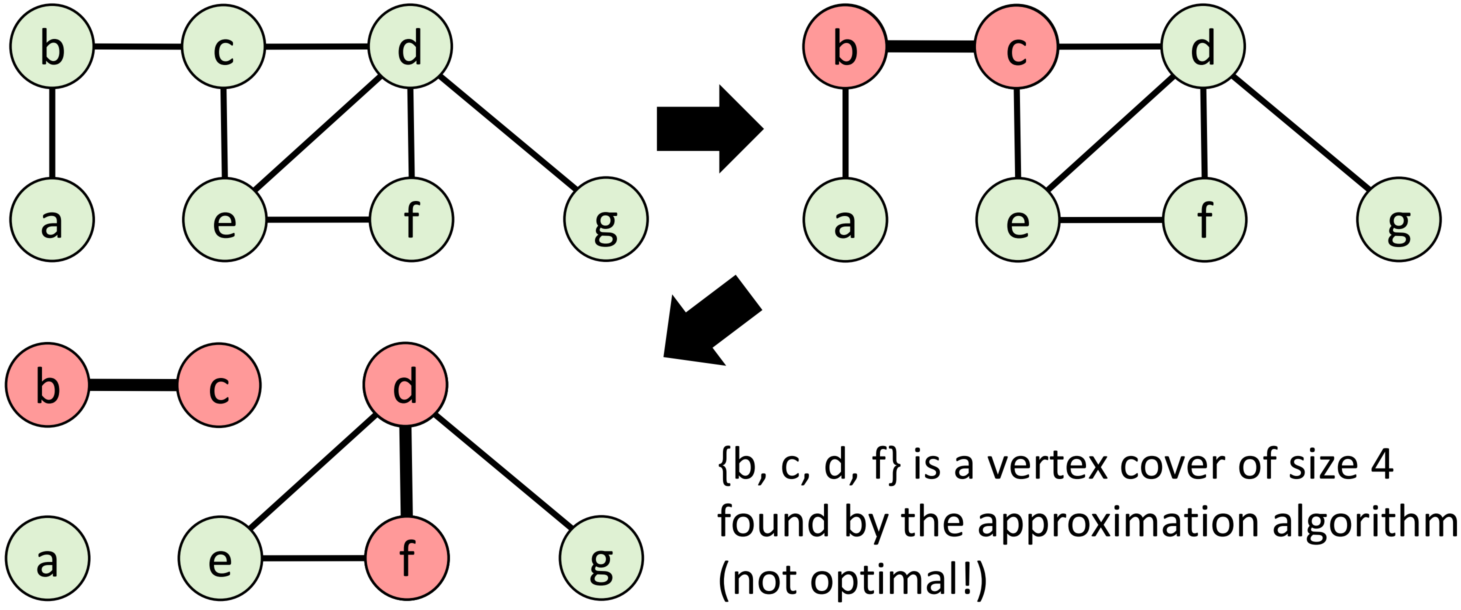
Approximate Algorithm

```
APPROX-VERTEX-COVER(G)
  C = ∅
  E' = G.E
  while E' ≠ ∅
    let (u, v) be an arbitrary edge of E'
    C = C ∪ {u, v}
    remove from E' every edge incident on either u or v
  return C
```

- APPROX-VERTEX-COVER
  - Randomly select one **edge** at a time
  - Remove all incident edges
- Running time = $O(|V| + |E|)$

Approximate Algorithm

- APPROX-VERTEX-COVER
  - Randomly select one **edge** at a time
  - Remove all incident edges



{b, c, d, f} is a vertex cover of size 4 found by the approximation algorithm (not optimal!)

Approximate Algorithm

Theorem. APPROX-VERTEX-COVER is a 2-approximation for the vertex cover problem.

- 3 things to check
- Q1: Does it give a feasible solution?
  - A feasible solution for vertex cover is a node set that covers all the edges
  - Finding an optimal solution is hard, but finding a feasible one could be easy
- Q2: Does it run in polynomial time?
  - An exponential-time algorithm is not qualified to be an approximation algorithm
- Q3: Does it give an approximate solution with approximation ratio ≤ 2?
  - Other names: 2-approximate solution, factor-2 approximation

2-Approximation Solution

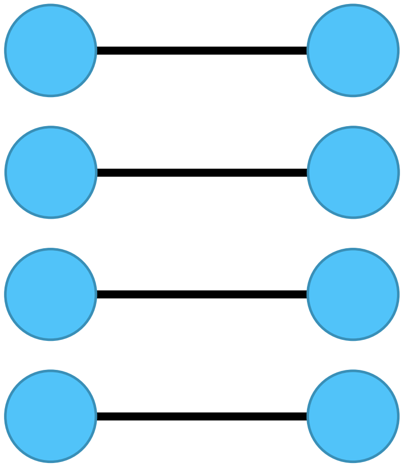
Prove that $\rho(n) = 2$. That is $|C| \leq 2|C^*|$.

- Suppose that the algorithm runs for *k* iterations. Let *C* be the output of APPROX-VERTEX-COVER. Let OPT be any optimal vertex cover of *G*.
- If *k* = 0, then $|C| = |C^*| = 0$
- If *k* > 0, then $|C| = 2k$. It suffices to ensure that $|C^*| \geq k$
  - Observe that all those *k* edges (*u*, *v*) chosen by APPROX-VERTEX-COVER in those *k* iterations form a matching of *G*. Just for OPT (or any feasible solution) to cover this matching requires at least *k* nodes.

The proof doesn't require knowing the actual value of C*!

Approximation Analysis

- Tight analysis: check whether we underestimate the quality of the approximate solution obtained by APPROX-VERTEX-COVER

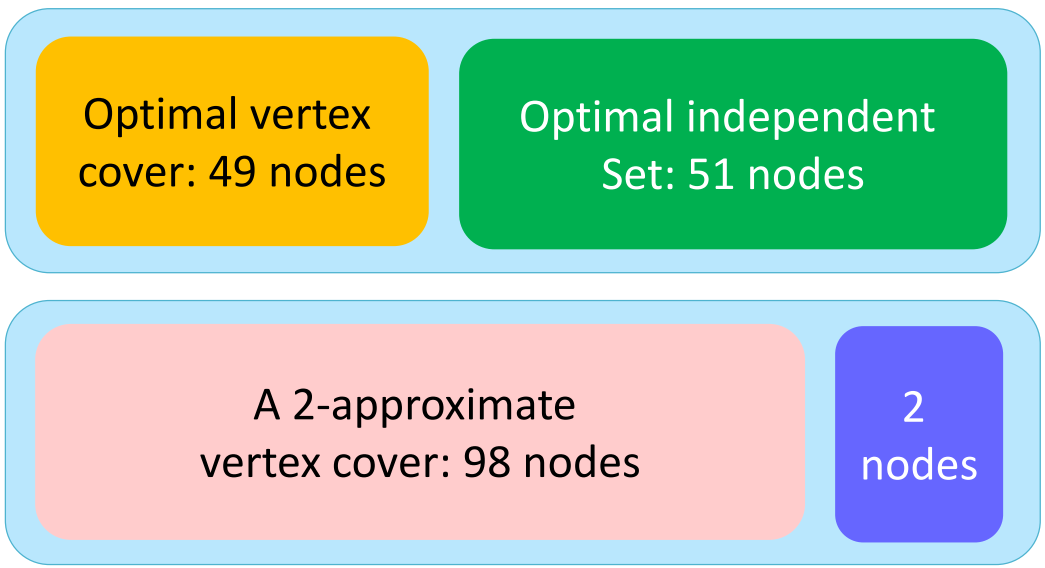


Yes, it is tight!

- This factor-2 approximation is still the best known approximation algorithm
  - Reducing to 1.99 is a significant result

Vertex Cover v.s. Independent Set

- C is a vertex cover of graph $G=(V, E)$ iff $V - C$ is an independent set of G
- Q: Does a 2-approximation algorithm for vertex cover imply a 2-approximation for maximum independent set?



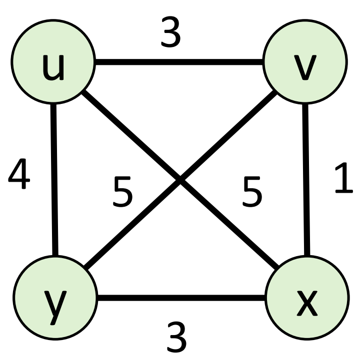
17 Traveling Salesman Problem

Textbook 35.2 – The traveling-salesman problem

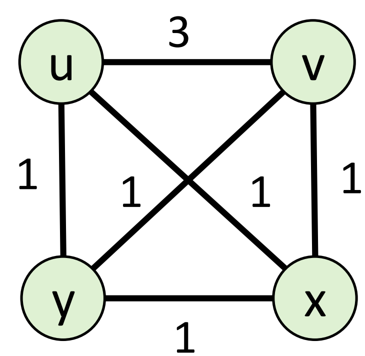
Traveling Salesman Problem (TSP)

- Optimization problem: Given a set of cities and their pairwise distances, find a tour of lowest cost that visits each city exactly once.
- Inter-city distances satisfy **triangle inequality** if for all vertices

$$d(u,w) \leq d(u,v) + d(v,w), \forall u,v,w \in V$$



w/ triangle inequality



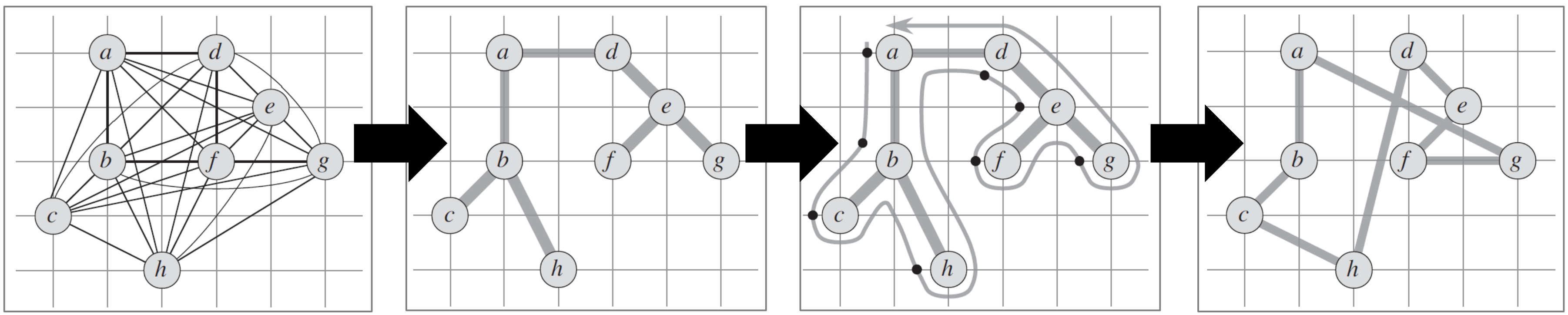
w/o triangle inequality

Approximate Algorithm

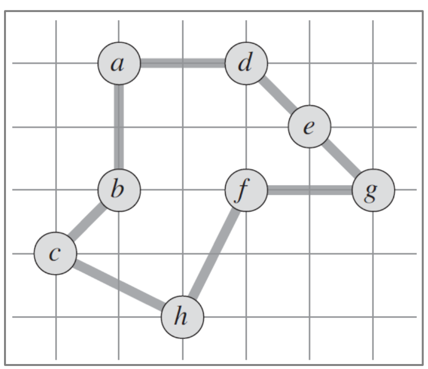
```
APPROX-TSP-TOUR(G)
  select a vertex r from G.V as a "root" vertex
  grow a minimum spanning tree T for G from root r using
  MST-PRIM(G, d, r)
  H = the list of vertices visited in a preorder tree walk of T
  return C
```

- APPROX-TSP-TOUR
  - Grow an MST from a random root
- MST-PRIM
  - For (*n* - *1*) iterations, add the least-weighted edge incident to the current subtree that does not incur a cycle
- Running time = $O(|E| + |V| \log |V|) = O(|V|^2)$

Approximate Algorithm



H = a, b, c, h, d, e, f, g, a



H* = a, b, c, h, f, g, e, d, a

Approximate Algorithm

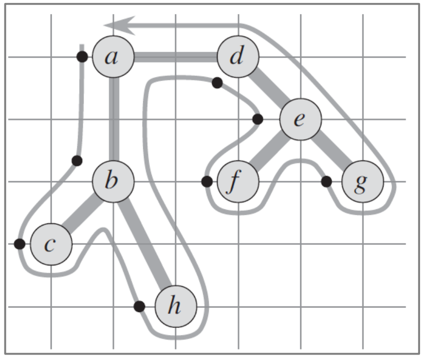
Theorem. APPROX-TSP-TOUR is a 2-approximation for the TSP problem.

- 3 things to check
- Q1: Does it give a feasible solution?
  - A feasible solution is a path of *G* visiting each cities exactly once
  - The property of a complete graph is needed
- Q2: Does it run in polynomial time?
- Q3: Does it give an approximate solution with approximation ratio ≤ 2?

2-Approximation Solution

Prove that $\rho(n) = 2$. That is $\text{cost}(H) \leq 2 \times \text{cost}(H^*)$.

- With triangle inequality: $\text{cost}(H) \leq 2 \times \text{cost}(\text{MST})$



- Let *H** denote an optimal tour formed by some tree plus an edge: $\text{cost}(\text{MST}) \leq \text{cost}(H^*)$
- Hence, $\text{cost}(H) \leq 2 \times \text{cost}(\text{MST}) \leq 2 \times \text{cost}(H^*)$

General TSP

Theorem 35.3. If P ≠ NP, there is **no** polynomial-time approximation algorithm with **a constant ratio bound *ρ*** for the general TSP

- Proof by contradiction
- Suppose there is such an algorithm A with a constant ratio ρ. We will use A to solve HAM-CYCLE in polynomial time.
- Algorithm for HAM-CYCLE
  - Convert $G = (V, E)$ into an instance I of TSP with cities V (resulting in a complete graph $G' = (V, E')$):

$$c(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ \rho|V| + 1 & \text{otherwise.} \end{cases}$$

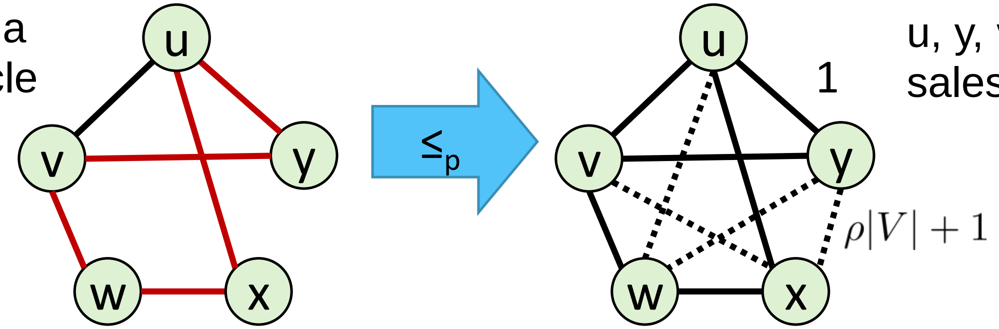
  - Run A on I
  - If the reported cost $\leq \rho|V|$, then return "Yes" (i.e., G contains a tour that is an HC), else return "No."

General TSP

Theorem 35.3. If P ≠ NP, there is **no** polynomial-time approximation algorithm with **a constant ratio bound *ρ*** for the general TSP

- Analysis
  - If *G* has an HC: *G'* contains a tour of cost |*V*| by picking edges in *E*, each has 1 cost
  - If *G* does not have an HC: any tour of *G'* must use some edge not in *E*, which has a total cost $\geq (\rho|V|+1)+(|V|-1) > \rho|V|$
  - Algorithm *A* guarantees to return a tour of cost $\leq \rho \times \mathrm{cost}(H^*)$
- HAM-CYCLE can be solved in polynomial time, contradiction
  - *A* returns a cost $\leq \rho|V|$ if *G* contains an HC; *A* returns a cost $> \rho|V|$, otherwise

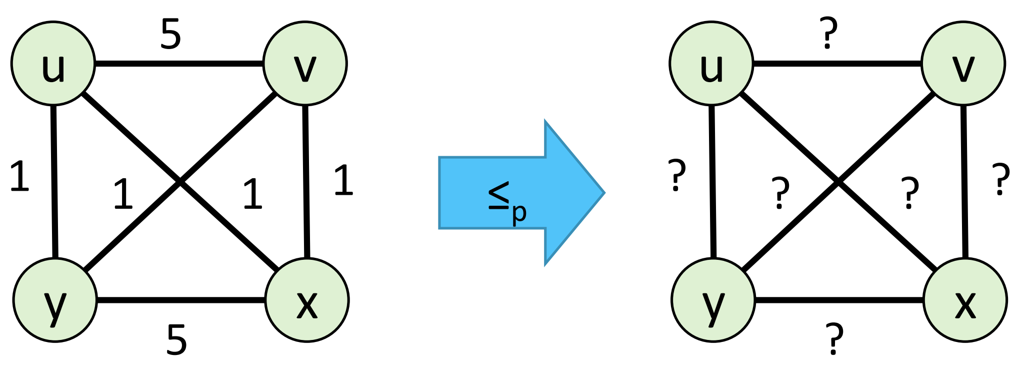
u, y, v, w, x, u is a Hamiltonian Cycle



u, y, v, w, x, u is a traveling-salesman tour with cost |V|

Exercise 35.2-2

Show how in polynomial time we can transform one instance of the traveling-salesman problem into another instance whose cost function satisfies the triangle inequality. The two instances must have the same set of optimal tours. Explain why such a polynomial-time transformation does not contradict Theorem 35.3, assuming that P ≠ NP.



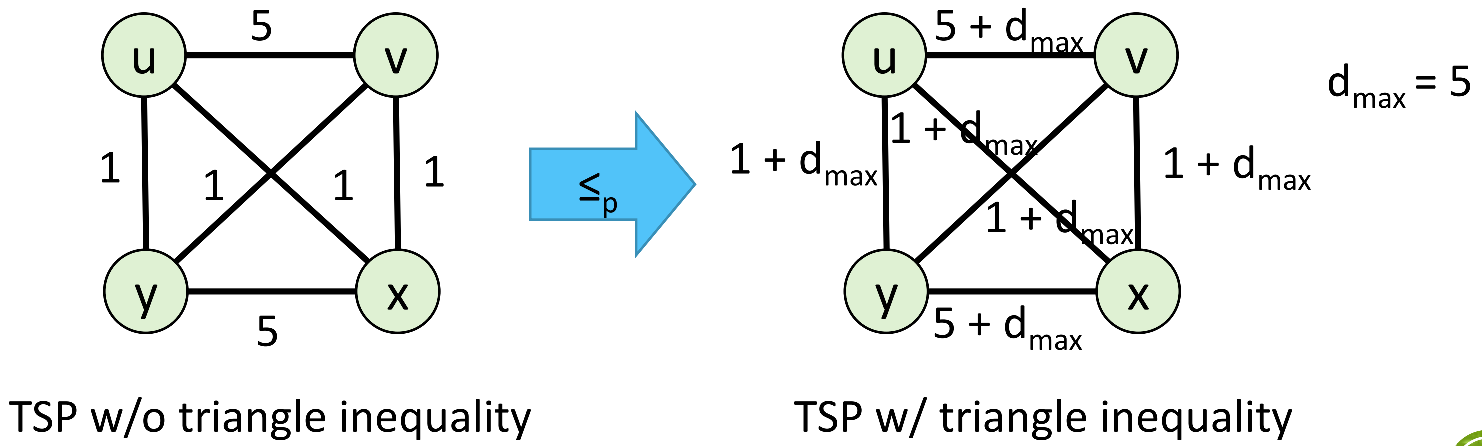
TSP w/o triangle inequality

TSP w/ triangle inequality

Exercise 35.2-2

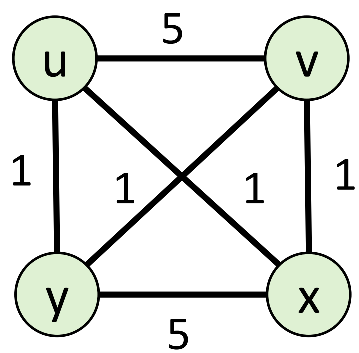
- For example, we can add d_{max} (the largest cost) to each edge
- G contains a tour of minimum cost k ⇔ G' contains a tour of minimum cost $k + d_{\max} \times |V|$
- G's satisfies triangle inequality because for all vertices $u, v, w \in V$

$$d'(u,w) = d(u,w) + d_{\max} \leq 2 \times d_{\max} \leq d'(u,v) + d'(v,w)$$

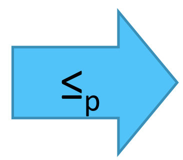


Exercise 35.2-2

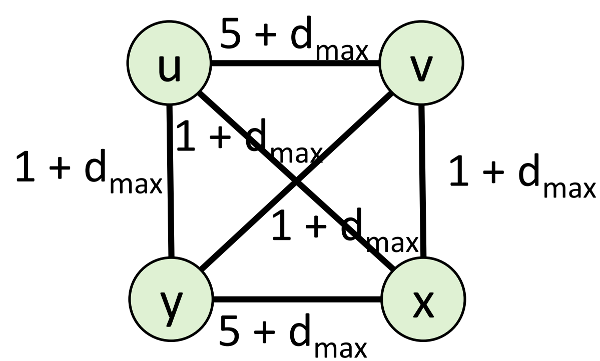
TSP w/o triangle inequality



$$\text{cost}(H) = 12$$
$$\text{cost}(H^*) = 4$$
$$\frac{\text{cost}(H)}{\text{cost}(H^*)} > 2$$

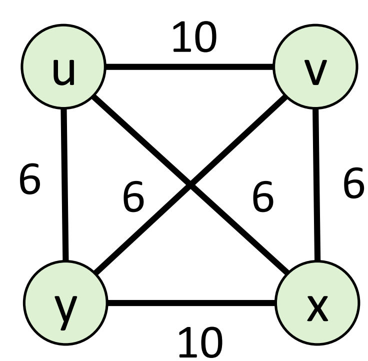


TSP w/ triangle inequality



$d_{max} = 5$

approximate



$$\text{cost}(H) = 32$$
$$\text{cost}(H^*) = 24$$
$$\frac{\text{cost}(H)}{\text{cost}(H^*)} \leq 2$$

28

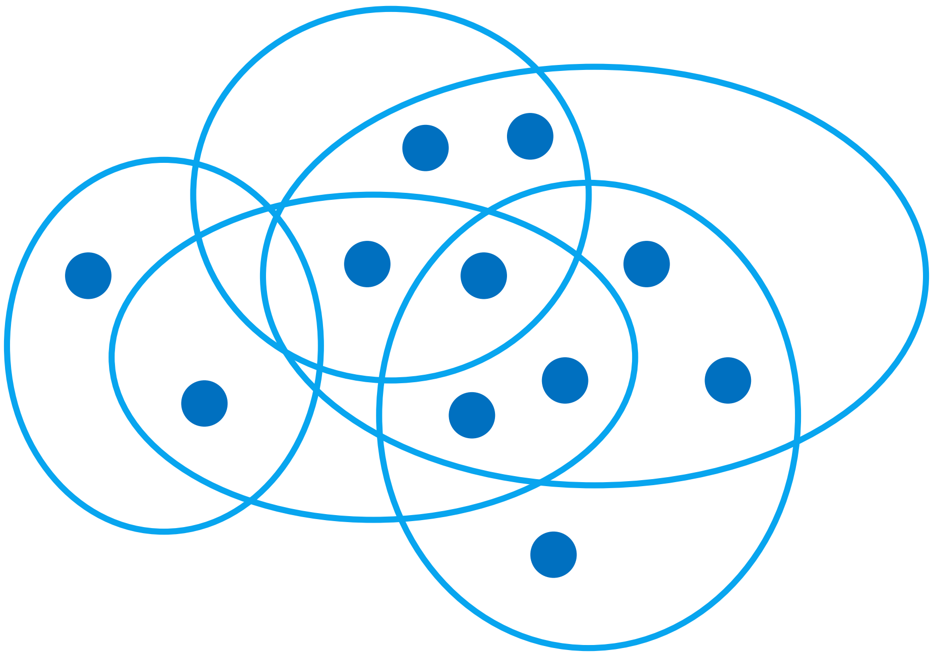
Set Cover

Textbook 35.3 – The set-covering problem

Set Cover

- Optimization problem: Given k subsets $\{S_1, S_2, \ldots, S_k\}$ of $1, 2, \ldots, n$, find an index subset C of $\{1, 2, \ldots, k\}$ with minimum $|C|$ s.t.

$$\cup_{i \in I} S_i = \{1, 2, \cdots, n\}$$



Set cover is NP-complete.

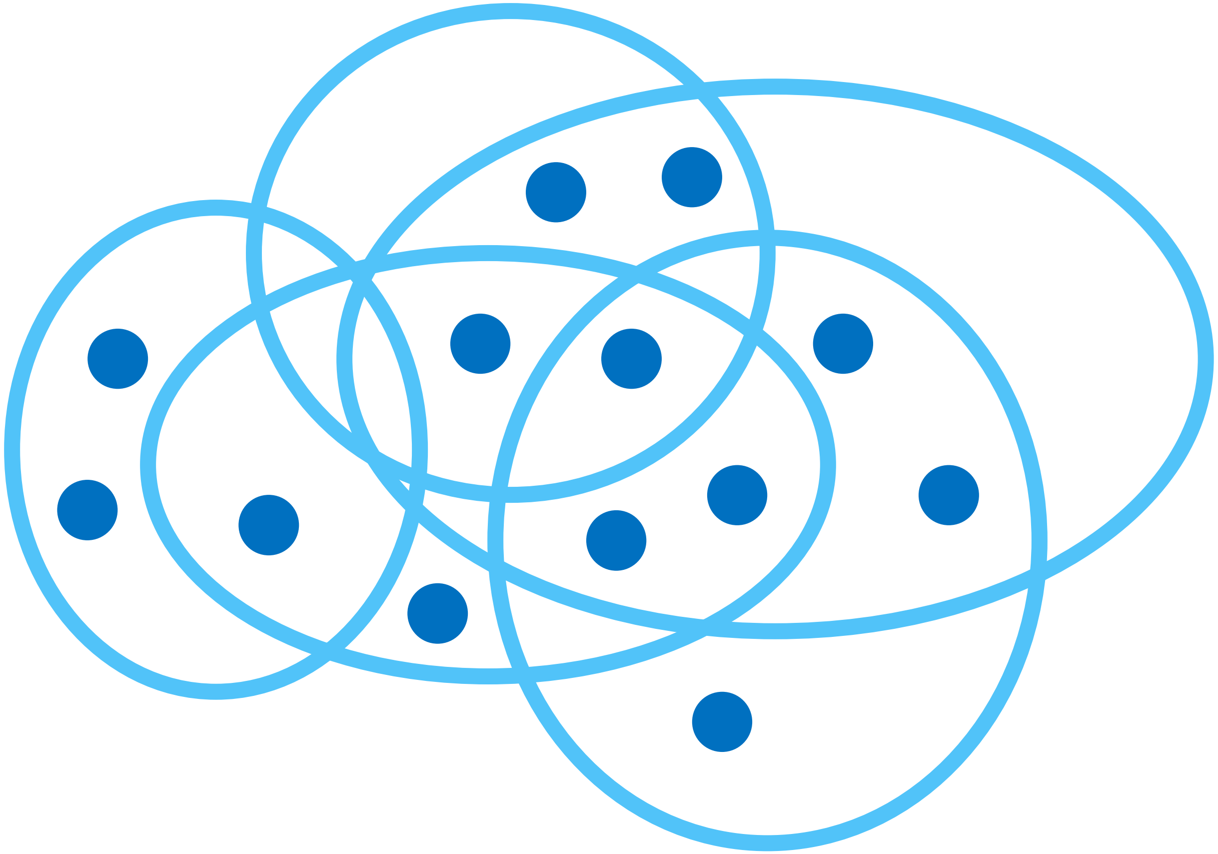
1) It is in NP
2) It is NP-hard

Approximate Algorithm

```
GREEDY-SET-COVER(S)
  I = ∅
  C = ∅
  while C ≠ {1, 2, …, n}
    select i be an index maximizing |Si - C|
    I = I ∪ {i}
    C = C ∪ Si
  return I
```

- GREEDY-SET-COVER
  - At each stage, picking the set S that covers the greatest number of remaining elements that are uncovered
- Running time = ?

Algorithm Illustration



Approximate Algorithm

Theorem. GREEDY-SET-COVER is a $O(\log n)$-approximation for the set cover problem.

- 3 things to check
- Q1: Does it give a feasible solution?
  - A feasible solution output is a collection of subsets whose union is the ground set {1, 2, …, *n*}.
- Q2: Does it run in polynomial time?
- Q3: Does it give an approximate solution with $\rho(n) = O(\log n)$?

$O(\log n)$-Approximation Solution

Prove that $\rho(n) = O(\log n)$. That is, $|I| \leq O(\log n) \times |I^*|$.

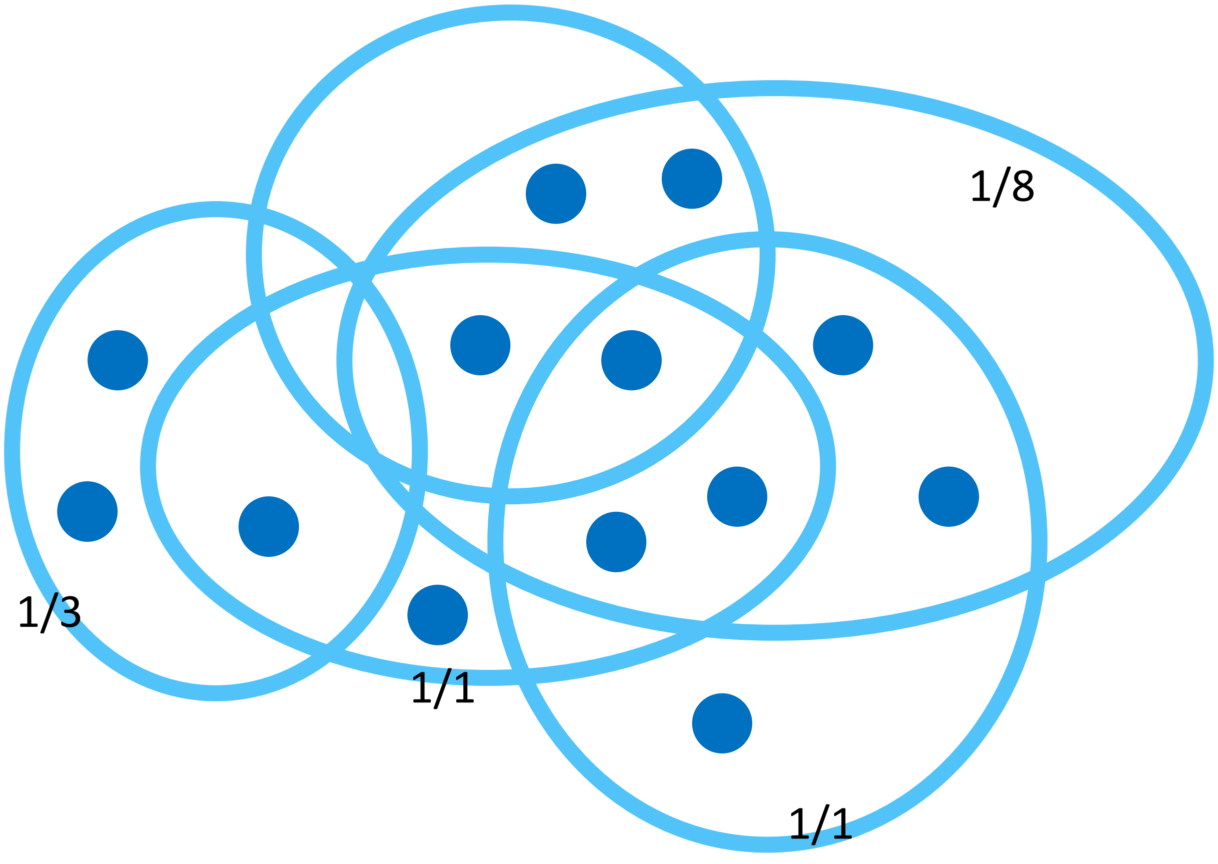
- Let *I** denote an optimal set cover. We plan to prove that

$$|I| \leq |I^*| \left(\frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + 1 \right)$$

Total Price

- For brevity, we re-index those subsets s.t. for each *i*, S_i is the *i*-th set selected by GREEDY-SET-COVER
- Let C_i be the *C* right before the elements of S_i is inserted into *C*
- If an element *j* is inserted into *C* in the *i*-th iteration, the price of *j* is $\frac{1}{|S_i - C_i|}$
- The sum of price of all *n* integers is exactly $|I|$

Algorithm Illustration



Bound

- For brevity, we re-index the integers s.t. they are inserted into *C* according to the increasing order of these integers
- When *j* is about to be put into *C*, there are at least *n-j+1* uncovered numbers. *I** is a collection of sets that can cover these *n-j+1* numbers. There is an index *t* ϵ *I** s.t. S_t can cover at least $\frac{n-j+1}{|I^*|}$ uncovered numbers
- We have $|S_i - C_i| \geq \frac{n-j+1}{|I^*|}$, where *j* is inserted into *C* in the *i*-th iteration.
- The price of *j* is $\frac{1}{|S_i - C_i|} \leq \frac{|I^*|}{n-j+1}$

$O(\log n)$-Approximation Solution

- The sum of price of all *n* integers is exactly $|I|$
- The price of *j* is at most $\frac{|I^*|}{n-j+1}$
- Therefore, we can prove that

$$|I| \leq \sum_{j=1}^{n} \frac{1}{n-j+1} |I^*| = H_n \cdot |I^*| = O(\log n) \cdot |I^*|$$

38

3-CNF-SAT

Textbook 35.4 – Randomization and linear programming

Randomized Approximate Algo

- **Randomized algorithm**'s behavior is determined not only by its input but also by values produced by a random-number generator

	Exact	Approximate
Deterministic	MST	APPROX-TSP-TOUR
Randomized	Quick Sort	MAX-3-CNF-SAT

3-CNF-SAT Problem

- Decision problem: Satisfiability of Boolean formulas in 3-*conjunctive normal form* (3-CNF)

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

  - 3-CNF = AND of clauses, each of which is the OR of exactly 3 distinct literals
  - A literal is an occurrence of a variable or its negation, e.g., x_1 or $\neg x_1$

$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$ → satisfiable

What is the optimization version of 3-CNF-SAT?

MAX-3-CNF-SAT

- Optimization problem: find an assignment of the variables that satisfies **as many clauses as possible**
  - Closeness to optimum is measured by the fraction of satisfied clauses

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$ satisfies 3 clauses

$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$ satisfies 2 clauses

This clause is always satisfied.
For simplicity, we assume no clause containing both literal and its negation.

Randomized Approximation Algo

- Randomly set each literal to be 0 or 1 (丟硬幣)
- Then…
- End

Theorem 35.6. Given an instance of MAX-3-CNF-SAT with n variables $x_1, x_2, \ldots, x_n$ and m clauses, the randomized algorithm that independently sets each variable to 1 with probability 1/2 and to 0 with probability 1/2 is a **randomized 8/7-approximation algorithm**

Randomized Approximation Algo

Theorem 35.6. Given an instance of MAX-3-CNF-SAT with *n* variables $x_1, x_2, \ldots, x_n$ and *m* clauses, the randomized algorithm that independently sets each variable to 1 with probability 1/2 and to 0 with probability 1/2 is a **randomized 8/7-approximation algorithm** (satisfying 8/7 of clauses *in expectation*)

- Proof
  - Each clause is the OR of exactly 3 distinct literals

$$Pr[x_i = 0] = Pr[x_i = 1] = 1/2$$

$$\rightarrow \forall x_1 \neq x_2 \neq x_3, Pr[(x_1 \vee x_2 \vee x_3) = 0] = 1/8$$

$$\rightarrow \mathbb{E}[\text{\# of satisfied clauses}] = m \times \mathbb{E}[\text{clause } j \text{ is satisfied}]$$

$$\geq m \times (1 - 1/8) = 7m/8$$

$$\rightarrow \rho(n) = \frac{\text{max \# of satisfied clauses}}{\mathbb{E}[\text{\# of satisfied clauses}]} = 8/7$$

Concluding Remarks

- Most practical optimization problems are NP-hard
  - It is widely believed that P ≠ NP
  - Thus, polynomial-time algorithms are unlikely, and we must sacrifice either **optimality**, **efficiency**, or **generality**
- Approximation algorithms sacrifice **optimality,** return **near-optimal** answers

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

Maximization problem: $C^*/C \leq \rho(n)$

Minimization problem: $C/C^* \leq \rho(n)$

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada.miulab.tw

Email: ada-ta@csie.ntu.edu.tw