Lecture 28.
Approximation Algorithms

Introduction to Algorithms
Sungkyunkwan University

Hyunseung Choo
choo@skku.edu

Lecture Contents

- Approximation Algorithms
  - Vertex Cover Problem
  - Traveling Salesman Problem
  - Job Scheduling Problem

Approximation Algorithms

- Possible ways to deal with NP-completeness
  - If the input is small, an algorithm with exponential running time may be satisfactory
  - Isolate special cases that can be solved in polynomial time
  - **Find near-optimal solutions in polynomial time**
- An approximation algorithm is an algorithm which can provide a **near-optimal** solution for a **NP-complete problem** in **polynomial time**

Approximation Ratio

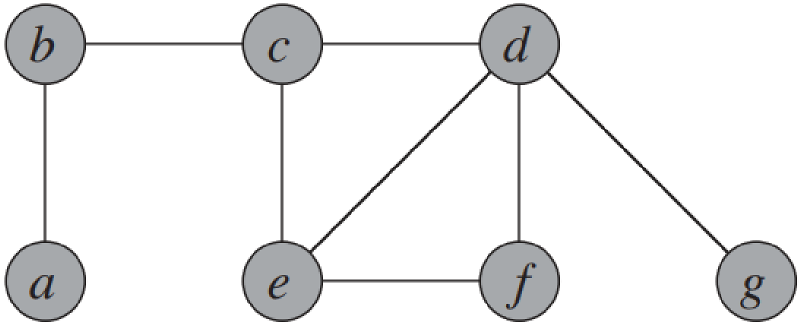
- An approximation algorithm for a problem has an **approximation ratio** of $\rho(n)$ if for any input of size n:

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

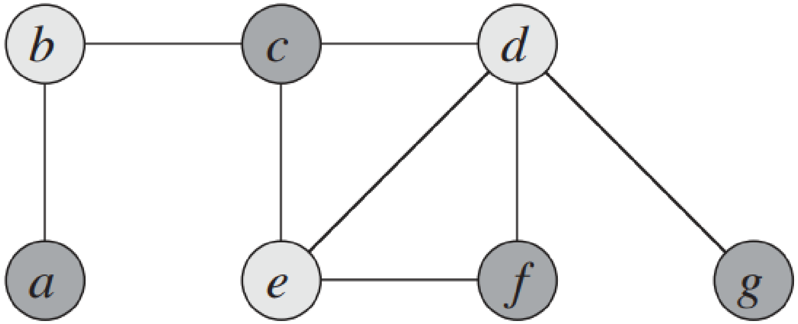
  - C is the cost of the approximation algorithm
  - $C^*$is the optimal cost of an optimal solution
  - $\rho(n) \geq 1$ ($\frac{C^*}{C} \geq 1$ for maximization problems, $\frac{C}{C^*} \geq 1$ for minimization problems)
  - The algorithm is called an $\rho(n)$-$approximation\ algorithm$
  - An optimal algorithm has $\rho(n) = 1$

Vertex Cover Problem

- A **vertex cover** of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both); the **size of a vertex cover** is the number of vertices in it
- The **vertex cover problem** is to find a vertex cover of **minimum size** in a given undirected graph



Input graph



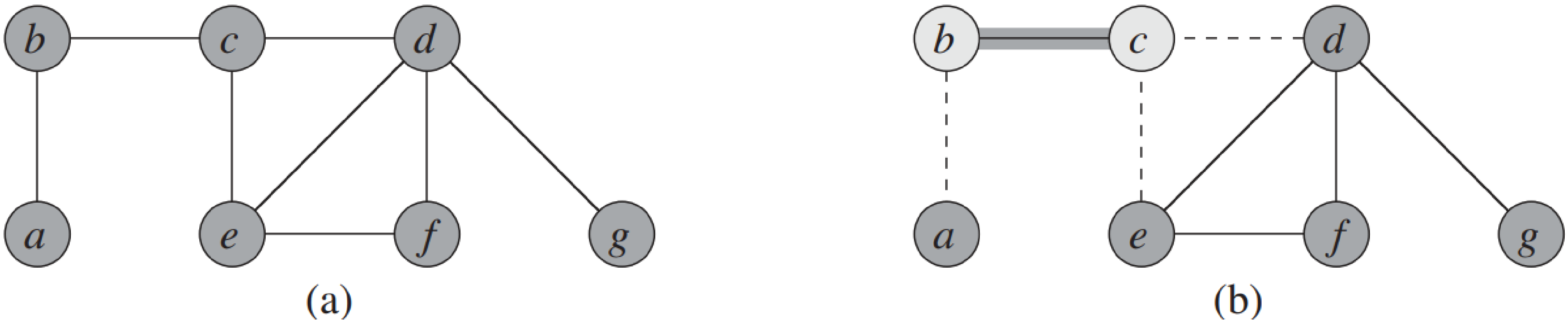
Optimal vertex cover includes vertices b,d,e

Vertex Cover Problem: Algorithm

APPROX-VERTEX-COVER(G)

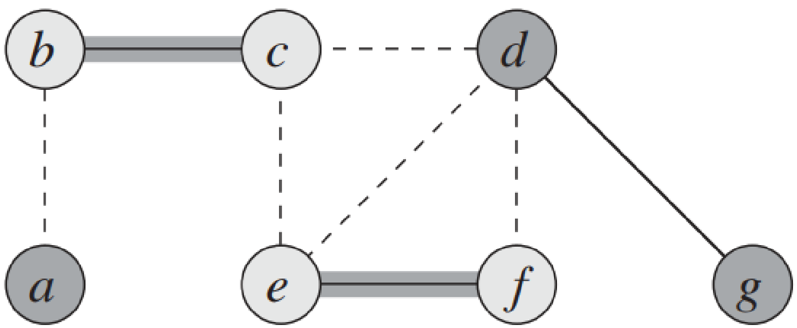
$C = \emptyset$
$E' = G.E$
while $E' \neq \emptyset$
    let (u, v) be an arbitrary edge of E'
    $C = C \cup \{u, v\}$
    remove from E' every edge incident on either u or v
return C

Vertex Cover Problem: Example (1/3)

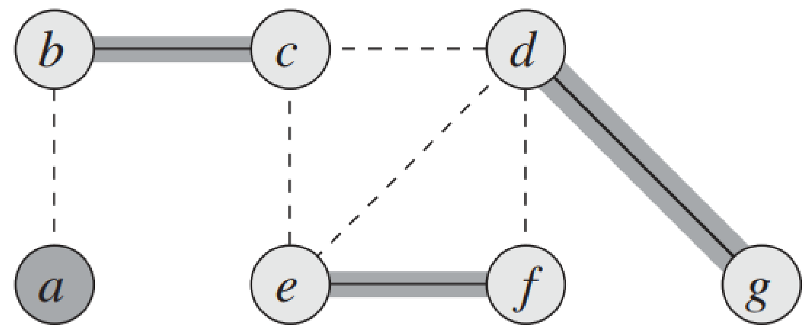


(a) Input graph G. **(b)** The edge (b, c), shown heavy, is the first edge chosen by APPROX-VERTEX-COVER. Vertices b and c, shown lightly shaded, are added to the set C containing the vertex cover being created. Edges (a, b), (c, e), and (c, d), shown dashed, are removed since they are now covered by some vertex in C.

Vertex Cover Problem: Example (2/3)



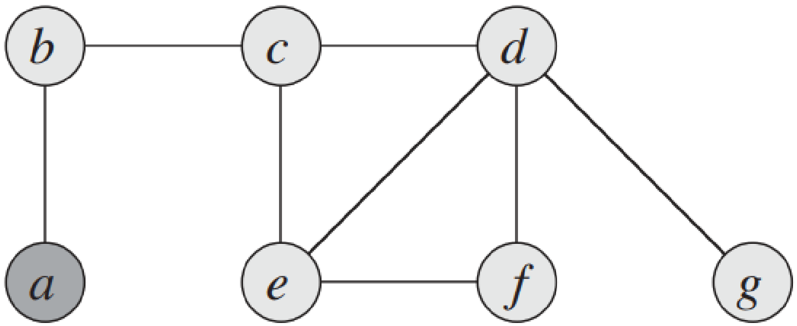
(c)



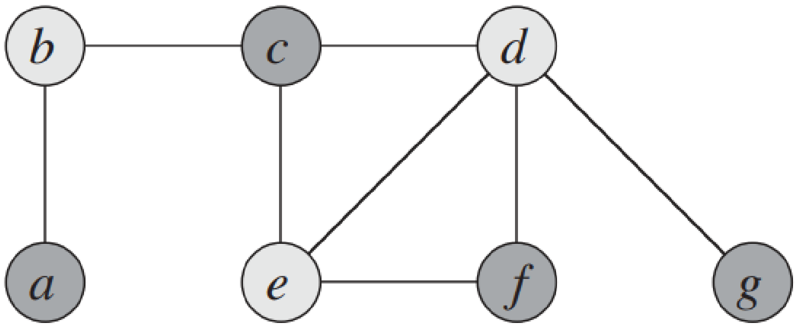
(d)

(c) Edge (e, f) is chosen; vertices e and f are added to C; edges (d, e) and (d, f) are removed. **(d)** Edge (d, g) is chosen; vertices d and g are added to C

Vertex Cover Problem: Example (3/3)



(e)



(f)

(e) The set C which is the vertex cover produced by APPROX-VERTEX-COVER, contains six vertices: b, c, d, e, f, g. **(f)** The optimal vertex cover for this problem contains only three vertices: $b, d,$ and e.

Cover Vertex Problem: Analysis

- APPROX-VERTEX-COVER is a 2-approximation algorithm

Proof:

- Let A denote the set of edges that line 4 of APPROX-VERTEX-COVER picked

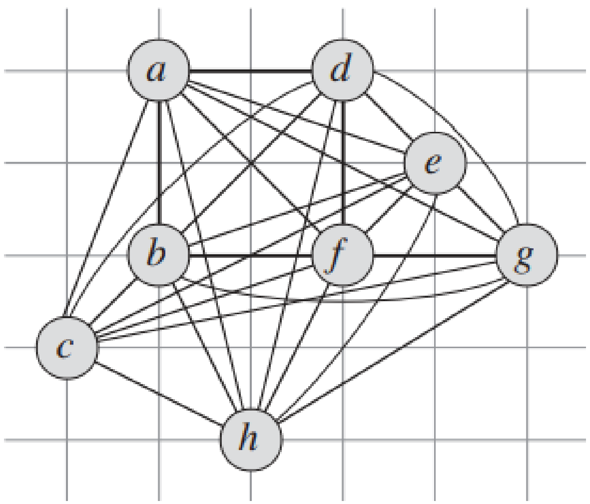
- An optimal solution must include at least one vertex of each edge in A, so $|C^*| \geq |A|$

- In line 5 of APPROX-VERTEX-COVER, for each edge of A, two vertices are added to C, so, number of vertices in C is $|C| = 2|A| \leq 2|C^*|$ that means

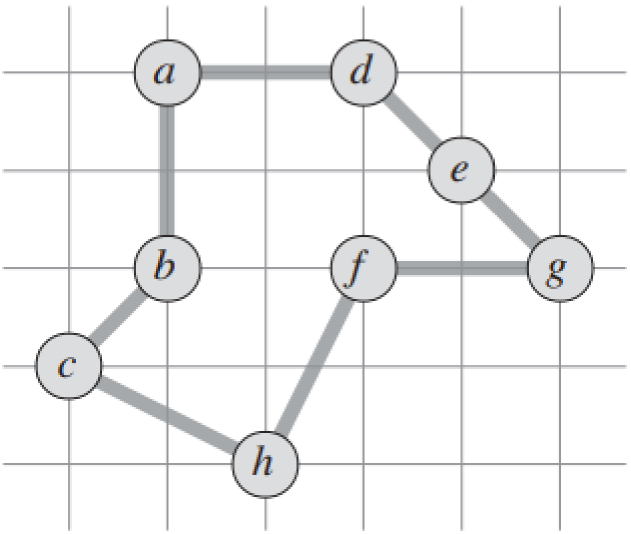
$$\frac{|C|}{|C^*|} \leq 2$$

Traveling Salesman Problem

- In traveling salesman problem, we are given a complete undirected graph $G = (V, E)$ that has a non-negative integer cost $c(u, v)$ associated with each edge $(u, v) \in E$, and we must find a Hamiltonian cycle of G with minimum cost



Input graph



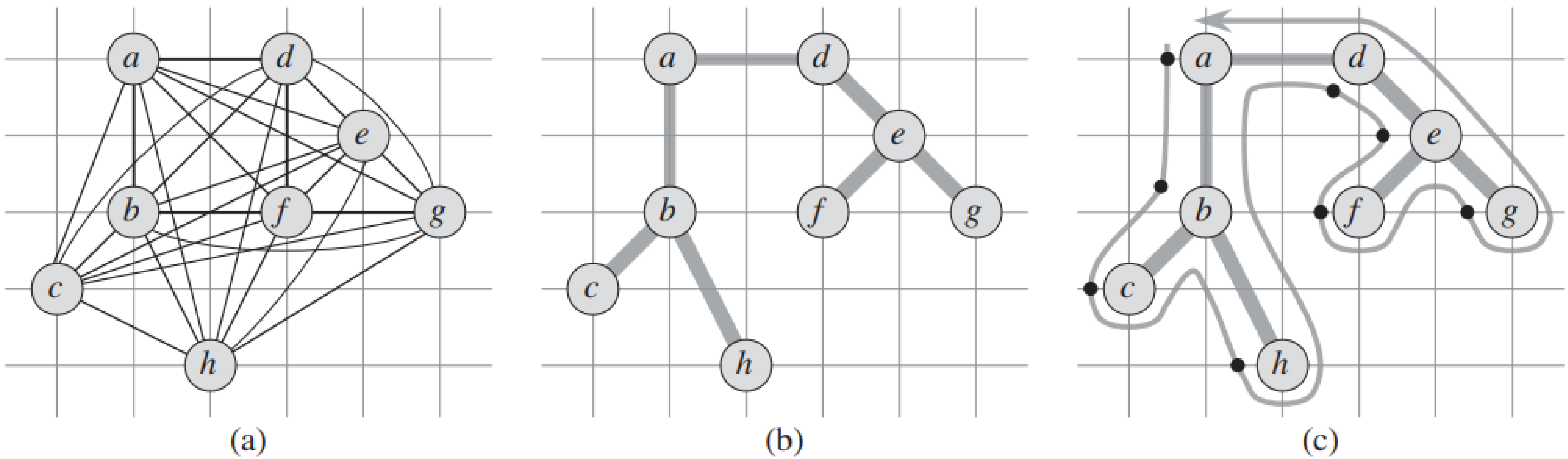
A Hamiltonian cycle with minimum cost

Traveling Salesman Problem: Algorithm

- The following algorithm uses minimum spanning tree to create a tour whose cost is no more than twice that of the minimum spanning tree's weight. MST-PRIM algorithm is used to compute the minimum spanning tree

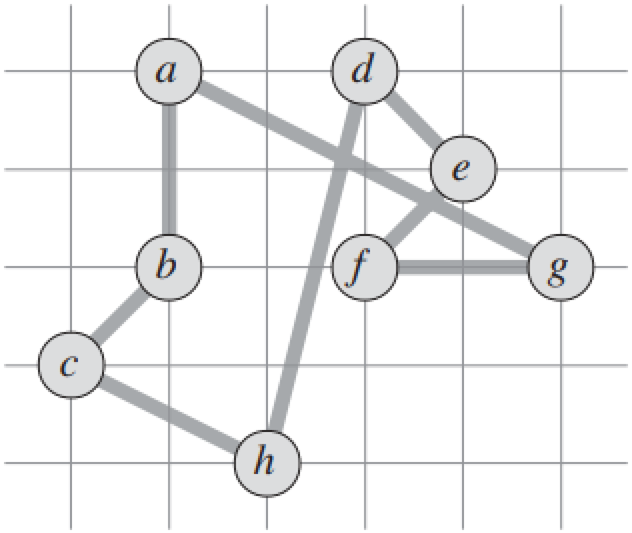
```
APPROX-TSP-TOUR(G, c)
select a vertex r ∈ G.V to be a "root" vertex
compute a minimum spanning tree T for G from root r
    using MST-PRIM(G, c, r)
let H be a list of vertices, ordered according to when they are first visited
    in a preorder tree walk of T
return the hamiltonian cycle H
```

Traveling Salesman Problem: Example

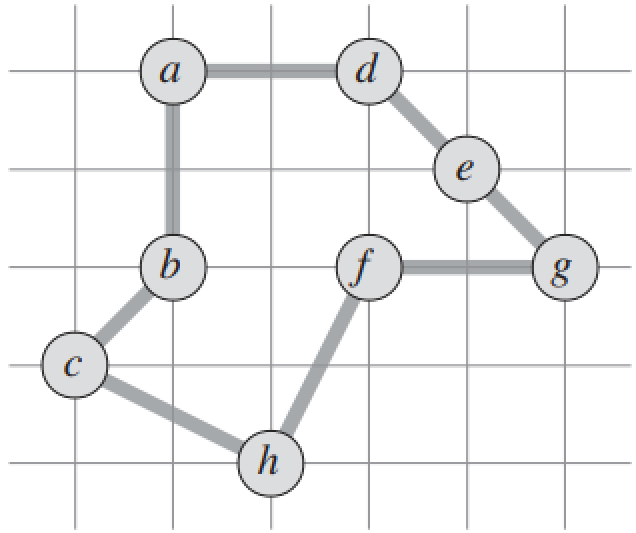


(a) A complete undirected graph. **(b)** A minimum spanning tree T of the graph computed by MST-PRIM, a is the root vertex. **(c)** A walk of T, starting at a. A **full walk** of the tree visits the vertices in the order $a;\ b;\ c;\ b;\ h;\ b;\ a;\ d;\ e;\ f;\ e;\ g;\ e;\ d;\ a$. A **preorder walk** of T lists a vertex just when it is first encountered, as indicated by the dot next to each vertex, yielding the ordering $a;\ b;\ c;\ h;\ d;\ e;\ f;\ g$.

Traveling Salesman Problem: Example



(d)



(e)

(d) A tour obtained by visiting the vertices in the order given by the preorder walk, which is the tour H returned by APPROX-TSP-TOUR. **(e)** An optimal tour H^* for the original complete graph

Traveling Salesman Problem: Analysis

- APPROX-TSP-TOUR is a 2-approximation algorithm

Proof:

\- Let H^* denote an optimal tour. A spanning tree is obtained if one edge is removed from the tour. Therefore, the weight of the minimum spanning tree in line 2 of the algorithm is a lower bound for the cost of H^*

$$c(T) \leq c(H^*)$$

\- Let W denote a full walk of the minimum spanning tree T. Since the full walk traverses every edge of T exactly twice

$$c(W) = 2c(T) \leq 2c(H^*)$$

Traveling Salesman Problem: Analysis

- APPROX-TSP-TOUR is a 2-approximation algorithm

Proof (continue):

- The tour H is obtained by removing vertices from the full walk W, so

$$c(H) \leq c(W) \leq 2c(H^*)$$
$$\frac{c(H)}{c(H^*)} \leq 2$$

Job Scheduling Problem

- Suppose we have n jobs, each of which take time t_i to process, and m identical machines on which we schedule the jobs. Jobs cannot be split between machines. For a given scheduling, let A_j be the set of jobs assigned to machine j. Let $T_j = \sum_{i \in A_j} t_i$ be the load of machine j. The job scheduling problem asks for an assignment of jobs to machines that minimizes the makespan, where the makespan is defined as the maximum load over all machines.
- Example: 5 jobs with processing time {3,3,2,2,2}, and 2 machines
  - The optimal schedule: {3,3} and {2,2,2}, makespan = 6

Job Scheduling Problem: Algorithm

- Consider the following greedy algorithm which iteratively allocates the next job to the machine with the least load.

GREEDY-JOB-SCHEDULING (A, T, t, m, n)

for $j = 1\ to\ m$
  $A_j = \emptyset$
  $T_j = 0$
for $i = 1\ to\ n$
  $j = \underset{k}{\operatorname{argmin}}\, T_k$
  $A_j = A_j \cup \{i\}$
  $T_j = T_j + t_i$
return A

A: sets of jobs assigned to machines
T: loads of machines
t: processing times of jobs
m: number of machines
n: number of jobs

Job Scheduling Problem: Example

- We are given 5 jobs with processing time {3,3,2,2,2}, and 2 machines
  - The optimal schedule: {3,3} and {2,2,2}, makespan = 6
  - The schedule given by GREEDY-JOB-SCHEDULING: {3,2,2} and {3,2}, makespan = 7

Job Scheduling Problem: Analysis

- GREEDY-JOB-SCHEDULING is a 2-approximation algorithm

Proof:

- Let T^* denote the optimal makespan, then:

$$T^* \geq \max_i t_i$$

$$T^* \geq \frac{1}{m} \sum_i^n t_i$$

Job Scheduling Problem: Analysis

- GREEDY-JOB-SCHEDULING is a 2-approximation algorithm

Proof (continue):

- Consider machine j with maximum load T_j. Let i be the last job scheduled on machine j. Before i was scheduled, j had the smallest load, so j must have had load smaller than the average load. Then,

$$T_j = (T_j - t_i) + t_i \leq \frac{1}{m}\sum_{i}^{n} t_i + \max_i t_i \leq T^* + T^* = 2T^*$$

Thanks to contributors

Mr. Pham Van Nguyen (2022)

Prof. Hyunseung Choo (2001 – 2022)