Design and Analysis of Algorithms

Lecture-44

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

Approximation Algorithms

- An algorithm that returns near-optimal solutions is said to be approximation algorithm.
- This technique does not guarantee the best solution.
- The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at the most polynomial time.
- Approximation algorithms are designed to get the solution of NP-complete problems in polynomial time.
- If we work on an optimization problem where every solution carries a cost, then an approximation algorithm returns a legal solution, but the cost of that legal solution may not be optimal.

Approximation Ratio

- Let C be the cost of the solution returned by an approximate algorithm, and C* is the cost of the optimal solution.
- An algorithm for a problem has an **approximation ratio** of P(n) for any input of size n, if the cost C of the solution produced by the algorithm is within a factor of P(n) of the cost C* of an optimal solution i.e.

$$\max(C/C^*, C^*/C) \le P(n)$$

- If an algorithm achieves an approximation ratio of P(n), we call it a **P(n)-approximation algorithm**.

Approximation Ratio(cont.)

- The approximation ratio measures how bad the approximate solution is distinguished with the optimal solution. A large (small) approximation ratio measures the solution is much worse than (more or less the same as) an optimal solution.
- Observe that P(n) is always ≥ 1, if the ratio does not depend on n, we may write P. Therefore, a 1-approximation algorithm gives an optimal solution.
- For a minimization problem, $0 < C \le C^*$, and the ratio C^*/C gives the factor by which the cost of the optimal solution is larger than the cost of approximate solution.
- Similarly, for a minimization problem, $0 < C^* \le C$, and the ratio C/C^* gives the factor by which the cost of the approximate solution is larger than the cost of optimal solution.

Vertex Cover Problem

- A **vertex cover** of an undirected graph G =(V, E) is a subset V’ ⊆ V such that if (u,v) is an edge of G, then either u ∈ V’ or v ∈ 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. We call such a vertex cover an optimal vertex cover.
- This problem is the optimization version of an NP-complete decision problem.

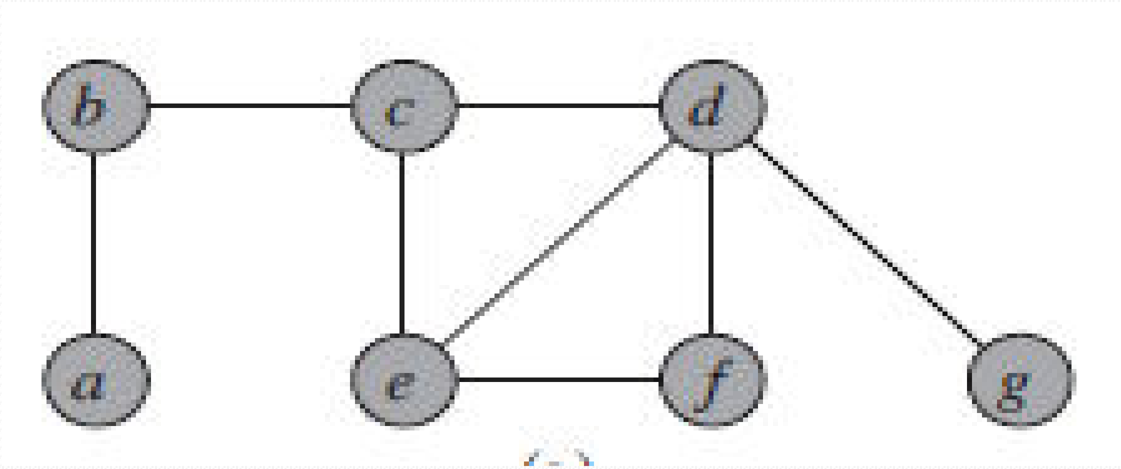
Approximation Algorithm for Vertex Cover Problem

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

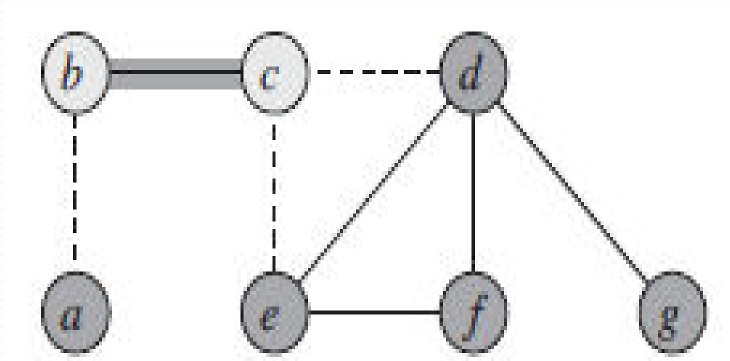
Approximation Algorithm for Vertex Cover Problem

Ex. Consider the following graph:-

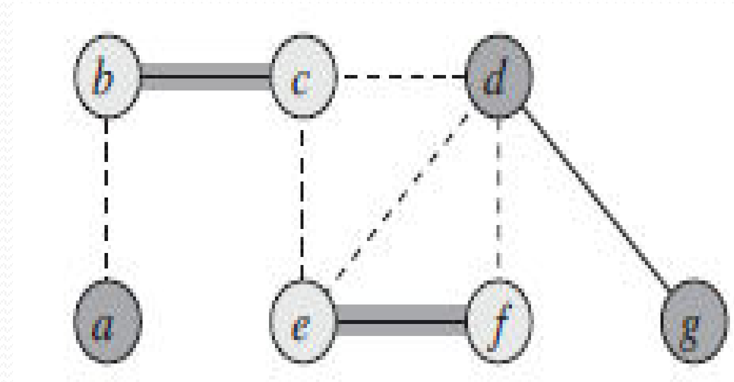


Find the optimal vertex cover of this graph.

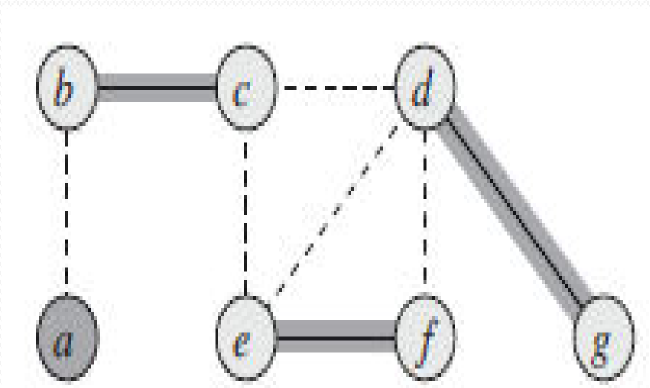
Solution



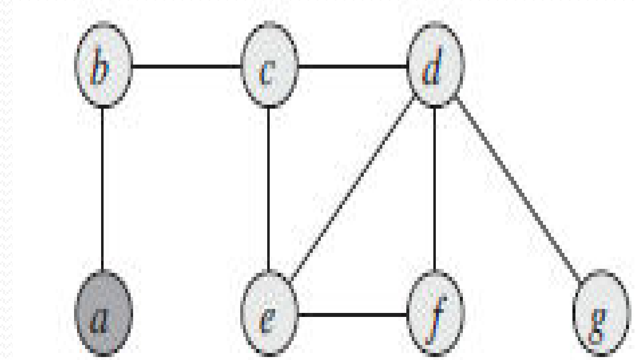
(a)



(b)

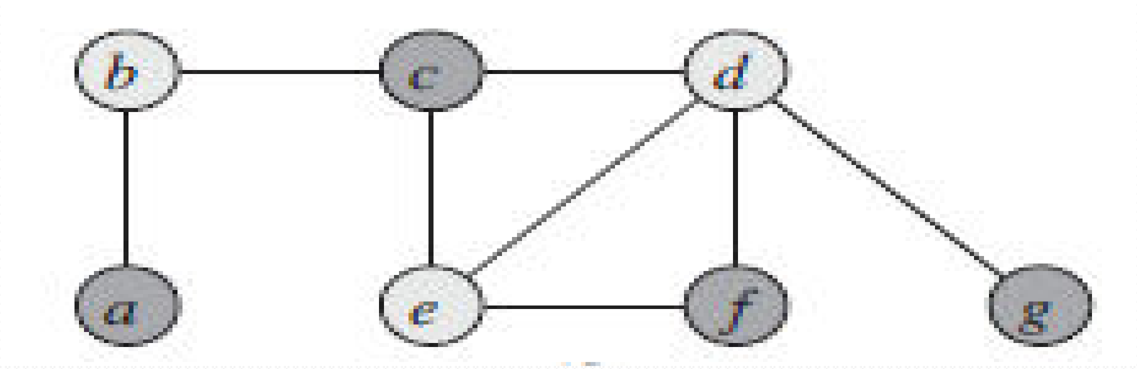


(c)



(d)

Approximation Algorithm for Vertex Cover Problem



Optimal vertex cover for this problem contains only three vertices: b, d, and e.

Note: The running time of this algorithm is O(E+V) using adjacency lists to represent E'.

Traveling-salesman problem

- In the traveling-salesman problem, we are given a complete undirected graph G(V,E) that has a nonnegative integer cost c(u,v) associated with each edge (u,v) ∈ E, and we must find a Hamiltonian cycle (a tour) of G with minimum cost.
- Let c(A) denote the total cost of the edges in the subset $A \subseteq E$.

$$c(A) = \sum_{(u,v) \in A} c(u,v)$$

- Cost function c satisfies the ***triangle inequality*** *if, for all vertices* u,v,w ∈*V,*

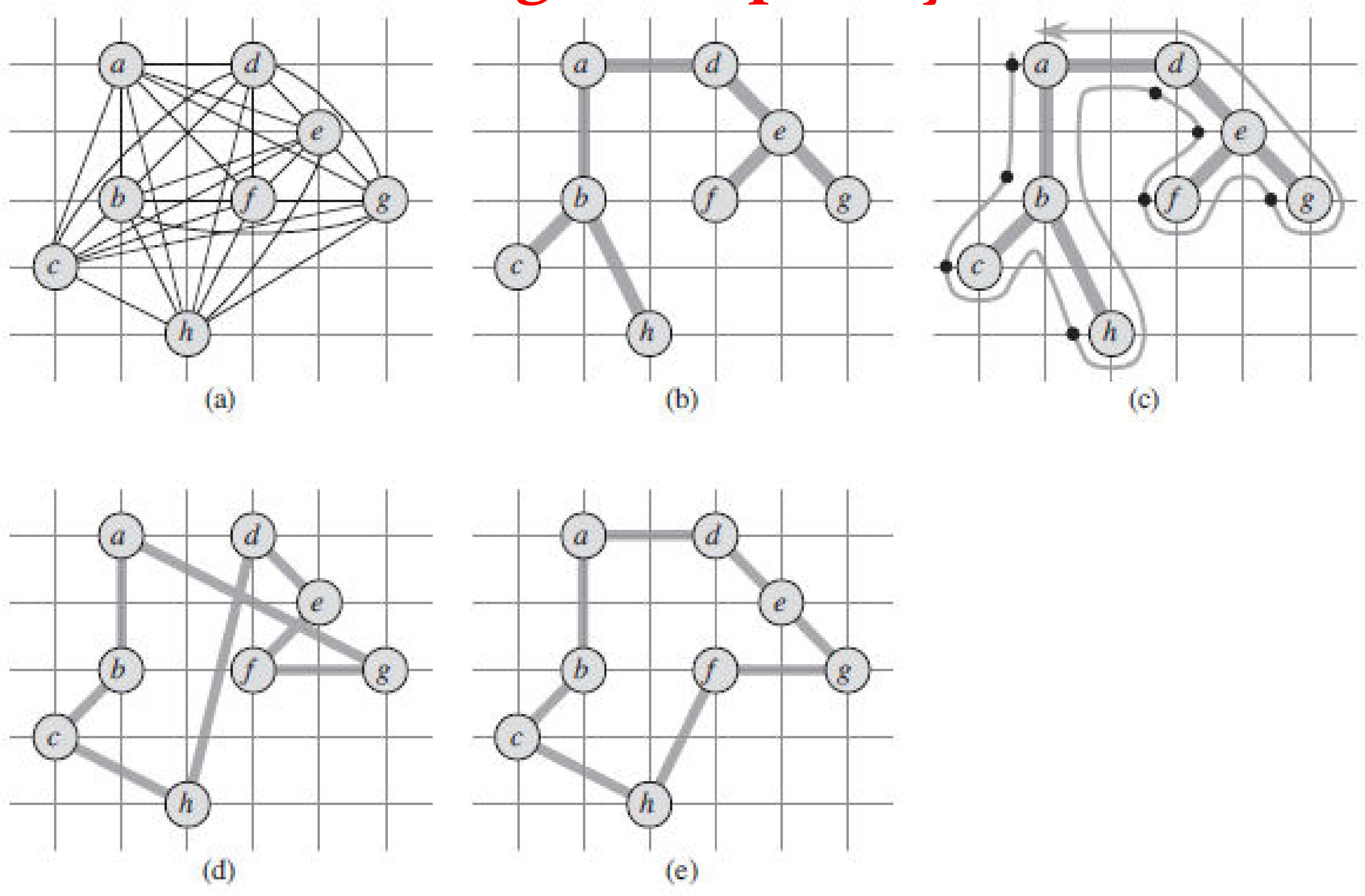
$$c(u,v) \leq c(u,v) + c(v,w)$$

Traveling-salesman problem with the triangle inequality

- The following algorithm computes a near-optimal tour of an undirected graph G, using the minimum-spanning-tree algorithm MST-PRIM.
- Here, the cost function satisfies the triangle inequality.
- The tour that this algorithm returns is no worse than twice as long as an optimal tour.

```
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 with the triangle inequality



Traveling-salesman problem with the triangle inequality

(a) **A complete undirected graph. Vertices lie** on intersections of integer grid lines. For example, f is one unit to the right and two units up from h. The cost function between two points is the ordinary euclidean distance.

(b) **A minimum spanning** tree T of the complete graph, as computed by MST-PRIM. Vertex a is the root vertex. Only edges in the minimum spanning tree are shown. The vertices happen to be labeled in such a way that they are added to the main tree by MST-PRIM in alphabetical order.

(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 with the triangle inequality

(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. Its total cost is approximately 19.074.

(e) An optimal tour H for the original complete graph. Its total cost is approximately 14.715.

Note: The running time of APPROX-TSP-TOUR is $O(V^2)$.