Approximation Algorithms

Dr Hamed Vahdat-Nejad

NP-completeness

2 approaches to getting around NP-completeness

- When the actual inputs are small, an algorithm with exponential running time may be perfectly satisfactory.
- It may still be possible to find *near-optimal solutions* *in polynomial* time.

Approximation Algorithm

- An algorithm that returns near-optimal solutions is called an ***approximation algorithm***.
- An algorithm for a problem has an ***approximation ratio of ρ(n)*** ***if, for any input of*** size *n, the cost C of the solution produced by the algorithm is within a factor of ρ(n) of the* cost *C* of an optimal solution:*

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

- We call an algorithm that achieves an approximation ratio of $\rho(n)$ a ***ρ (n) approximation algorithm***.
- These definitions apply for both minimization and maximization problems.
- For a maximization problem, $0 < C \leq C^*$.
- For a minimization problem, $0 < C^* \leq C$.
- A 1-approximation algorithm produces an optimal solution.

- An ***approximation scheme*** for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\Delta > 0$ *such that for any* fixed Δ, the scheme is a $(1+\Delta)$-approximation algorithm.
- An approximation scheme is a ***polynomial-time approximation scheme*** if for any fixed $\Delta > 0$, the scheme runs in time polynomial in the size n of its input instance.

- The running time of a polynomial-time approximation scheme can increase very rapidly as Δ decreases.
- For example, the running time of a polynomial-time approximation scheme might be $O(n^{2/\Delta})$.
- An approximation scheme is a ***fully polynomial-time approximation scheme*** if it is an approximation scheme and its running time is polynomial both in $1/\Delta$ and in the size n of the input instance.

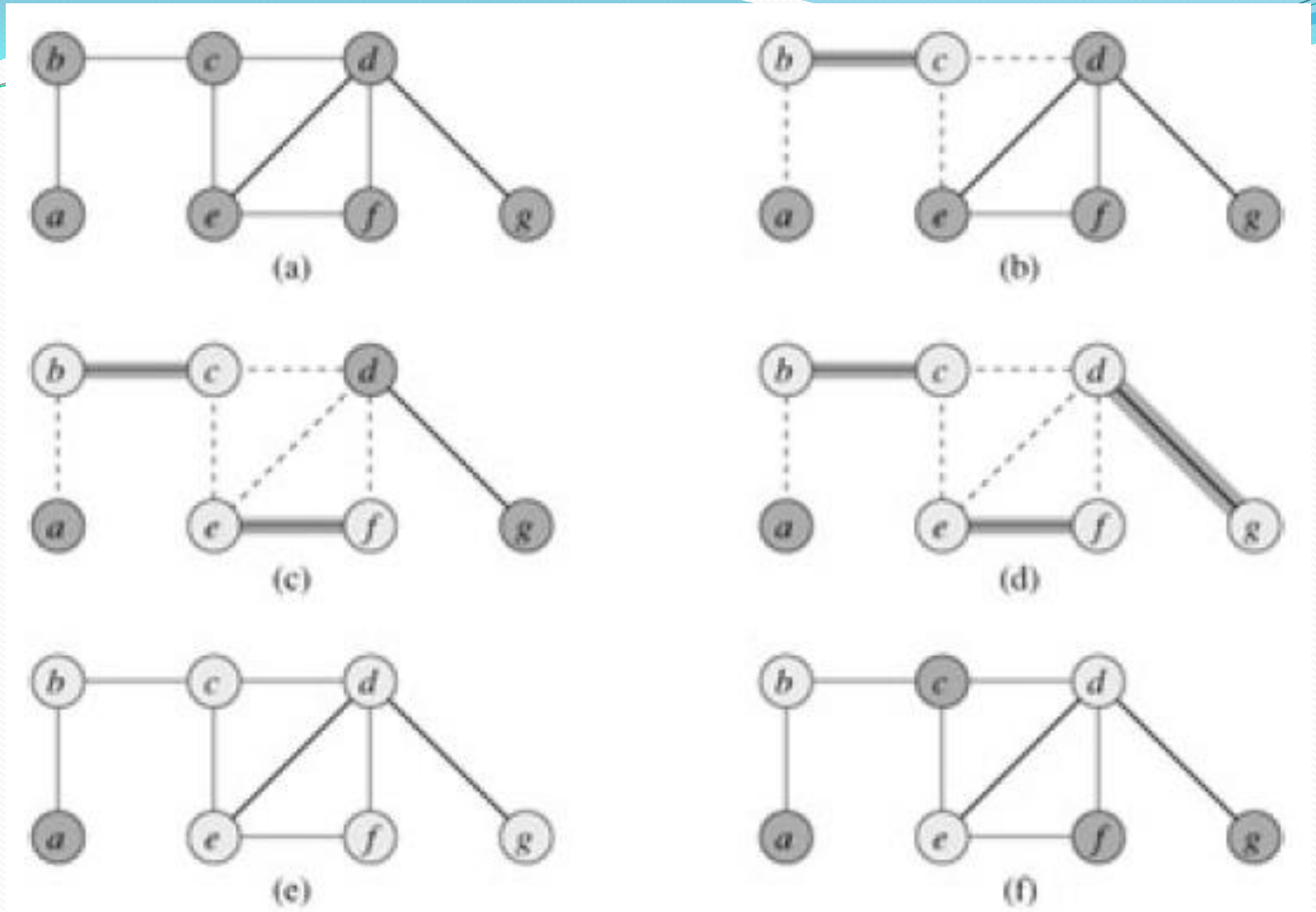
- For example, the scheme might have a running time of $O((1/\Delta)^2 n^3)$.
- With such a scheme, any constant-factor decrease in Δ can be achieved with a corresponding constant-factor increase in the running time.

The vertex-cover problem

- The vertex-cover problem was defined and proven NP-hard.
- A vertex cover of an undirected graph G = (V, E) is a subset $V' \in V$ 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.
- We call such a vertex cover an optimal vertex cover.

- Even though it may be difficult to find an optimal vertex cover in a graph *G*, it is not too hard to find a vertex cover that is near-optimal.
- The approximation algorithm takes as input an undirected graph *G* and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

- APPROX-VERTEX-COVER(*G*)
  - $C \leftarrow \emptyset$
  - $E' \leftarrow E[G]$
  - **while** $E' \neq \emptyset$ **do**
    - let (u, v) *be an arbitrary edge of* E'
    - $C \leftarrow C \cup \{u, v\}$
    - remove from E' *every edge incident on either* u *or* v
  - **return** C



- The running time of this algorithm is $O(V + E)$.

- APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.
- ***Proof***
  - It is obvious that APPROX-VERTEX-COVER runs in polynomial time.
  - The set *C of vertices that is returned by APPROX-VERTEX-COVER is a vertex cover, since* the algorithm loops until every edge in E[G] has been covered by some vertex in C.

- let *A* denote the set of edges that were picked by the algorithm.
- In order to cover the edges in *A*, an optimal cover C* must include at least one endpoint of each edge in A.
- No two edges in A share an endpoint.
- Thus, no two edges in *A* are covered by the same vertex from C*, so we have $|C^*| \geq |A|$
- On the other hand, $|C| = 2|A|$

- So

$$\begin{aligned} |C| &= 2|A| \\ &\leq 2|C^*| \end{aligned}$$

- Hence, proving the theorem.

The traveling-salesman problem

Problem

- Given a complete undirected graph $G = (V, E)$ that has a nonnegative integer cost c(u, v) associated with each edge $(u, v) \in E$. Find a hamiltonian cycle (a tour) of G with minimum cost.
- let $c(A)$ denote the total cost of the edges in the subset $A \subset E$:

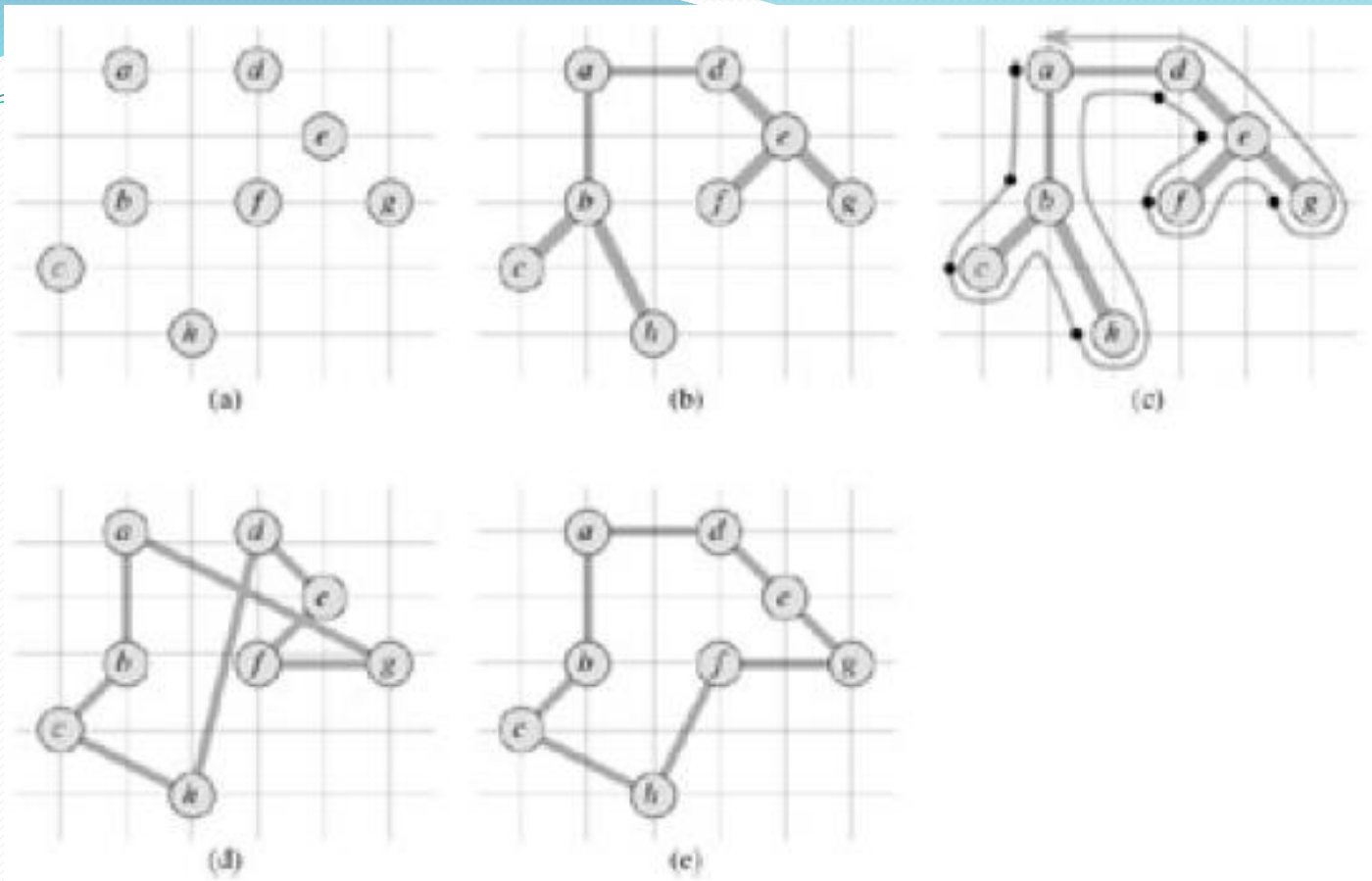
$$c(A) = \sum_{(u,v) \in A} c(u, v)$$

- The cost function c satisfies the **triangle inequality** if for all vertices u, v, w $\in$ V
  - $c(u, w) \leq c(u, v) + c(v, w)$.
- if the vertices of the graph are points in the plane and the cost of traveling between two vertices is the ordinary euclidean distance between them, then the triangle inequality is satisfied.

Algorithm

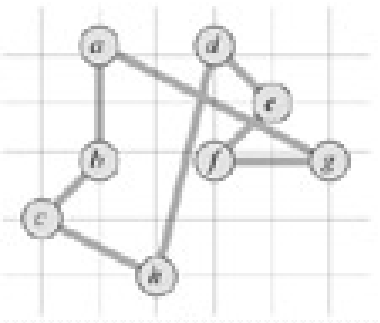
- APPROX-TSP-TOUR(G)
  - select a vertex r ∈ V [G] to be a "root" vertex.
  - compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r).
  - let L be the list of vertices visited in a preorder tree walk of T.
  - **return** the hamiltonian cycle H that visits the vertices in the order L.

- MST-PRIM algorithm returns a minimum-spanning-tree.



- (b) shows the minimum spanning tree T grown from root vertex a by MST-PRIM.
- (c) shows how the vertices are visited by a preorder walk of T .
- (d) displays the corresponding tour.
- (e) displays an optimal tour, which is about 23% shorter.

- 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 hamilton cycle corresponding to the walk lists a vertex just when it is first encountered, yielding the ordering *a, b, c, h, d, e, f, g*.
- The total cost is approximately 19.074.
- The total cost of the optimal tour H* for the given set of vertices is approximately 14.715.

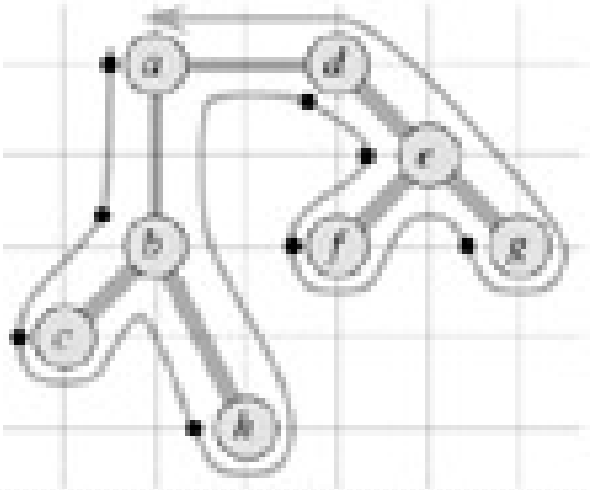


- The total time for Prim's algorithm is $O(E \log V)$.
- Even with a simple implementation of MST-PRIM, the running time of APPROX-TSP-TOUR is 0398(V^2).
- *Theorem* APPROX-TSP-TOUR is a polynomial-time 2-approximation algorithm for the traveling salesman problem with the triangle inequality.
- ***Proof*** We showed that APPROX-TSP-TOUR runs in polynomial time.
- Let H^* denote an optimal tour for the given set of vertices.

- Since we obtain a spanning tree by deleting any edge from a tour, the weight of the minimum spanning tree *T is a lower bound on* the cost of an optimal tour;

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

- A full walk of T lists the vertices when they are first visited and also whenever they are returned to after a visit to a subtree.
  - *a, b, c, b, h, b, a, d, e, f, e, g, e, d, a.*



- Since the full walk traverses every edge of T exactly twice, we have

$$c(W) = 2c(T)$$

- Hence

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

- Unfortunately, W is generally not a tour, since it visits some vertices more than once.
- By the triangle inequality, however, we can delete a visit to any vertex from W and the cost does not increase.
- If a vertex v is deleted from W between visits to u and w, the resulting ordering specifies going directly from u to w.
- By repeatedly applying this operation, we can remove from W all but the first visit to each vertex.
- In our example, this leaves the ordering

  a, b, c, h, d, e, f, g.

- Let *H be the cycle* corresponding to this preorder walk.
- It is a hamiltonian cycle, since every vertex is visited exactly once.
- It is the cycle computed by APPROX-TSP-TOUR.
- Since *H is* obtained by deleting vertices from the full walk *W , we have*

$$c(H) \leq c(W)$$

- Therefore $c(H) \leq 2c(H^*)$

Reference

- Cormen, et.al, **Introduction to Algorithms**, Second Edition, MIT Press,2001.

Any question?