Approximate Algorithms

Introduction:

An Approximate Algorithm is a way of approach **NP-COMPLETENESS** for the optimization problem. 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. Such algorithms are called approximation algorithm or heuristic algorithm.

- For the traveling salesperson problem, the optimization problem is to find the shortest cycle, and the approximation problem is to find a short cycle.
- For the vertex cover problem, the optimization problem is to find the vertex cover with fewest vertices, and the approximation problem is to find the vertex cover with few vertices.

Performance Ratios

Suppose we work on an optimization problem where every solution carries a cost. An Approximate Algorithm returns a legal solution, but the cost of that legal solution may not be optimal.

For Example, suppose we are considering for a **minimum size vertex-cover (VC)**. An approximate algorithm returns a VC for us, but the size (cost) may not be minimized.

Another Example is we are considering for a **maximum size Independent set (IS)**. An approximate Algorithm returns an IS for us, but the size (cost) may not be maximum. Let C be the cost of the solution returned by an approximate algorithm, and C* is the cost of the optimal solution.

We say the approximate algorithm has an approximate ratio P (n) for an input size n, where

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

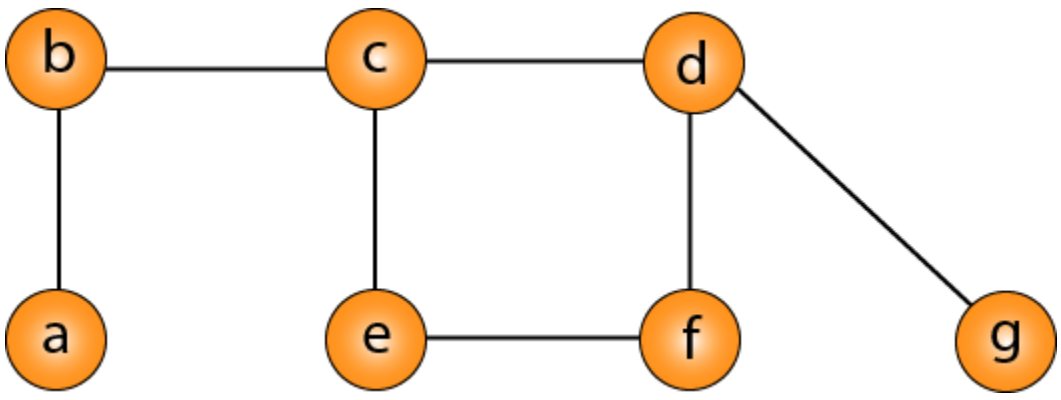
Intuitively, the approximation ratio measures how bad the approximate solution is distinguished with the optimal solution. A large (small) approximation ratio measures the solution is much worse than (more or less the same as) an optimal solution.

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. Some problems have polynomial-time approximation algorithm with small constant approximate ratios, while others have best-known polynomial time approximation algorithms whose approximate ratios grow with n.

Vertex Cover

A Vertex Cover of a graph G is a set of vertices such that each edge in G is incident to at least one of these vertices.

The decision vertex-cover problem was proven NPC. Now, we want to solve the optimal version of the vertex cover problem, i.e., we want to find a minimum size vertex cover of a given graph. We call such vertex cover an optimal vertex cover C*.

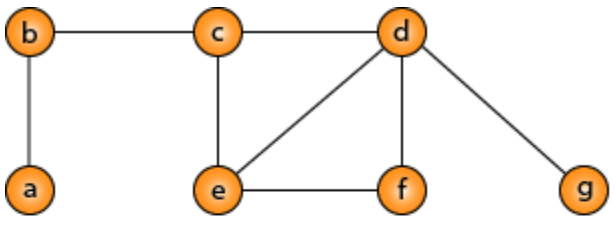


An approximate algorithm for vertex cover:

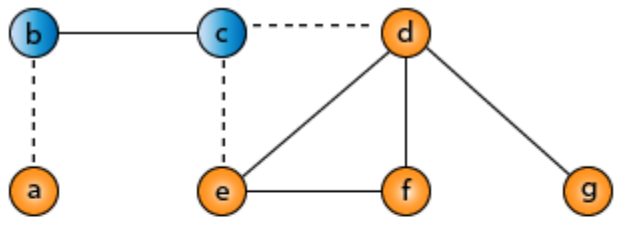
```
1. Approx-Vertex-Cover (G = (V, E))
2. {
3.         C = empty-set;
4.       E'= E;
5.       While E' is not empty do
6.         {
```

7. Let (u, v) be any edge in E': (*)
8. Add u and v to C;
9. Remove from E' all edges incident to
10. u or v;
11. }
12. Return C;
13. }

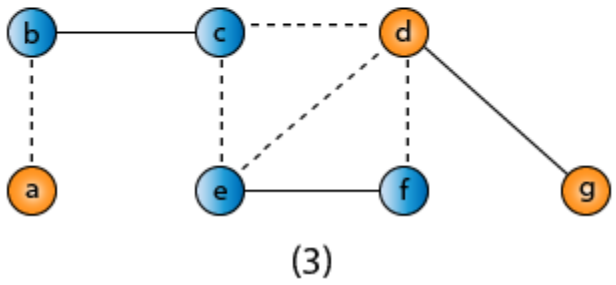
The idea is to take an edge (u, v) one by one, put both vertices to C, and remove all the edges incident to u or v. We carry on until all edges have been removed. C is a VC. But how good is C?



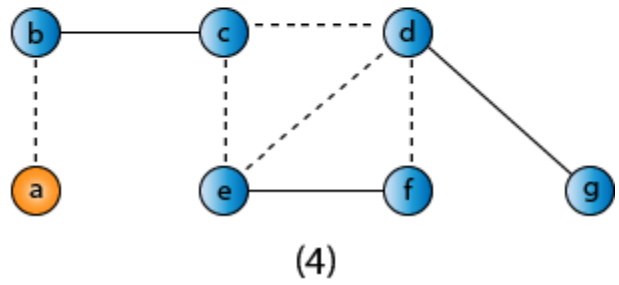
(1)



(2)



(3)



(4)

VC = {b, c, d, e, f, g}

Traveling-salesman Problem

In the traveling salesman Problem, a salesman must visits n cities. We can say that salesman wishes to make a tour or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. There is a non-negative cost c (i, j) to travel from the city i to city j. The goal is to find a tour of minimum cost. We assume that every two cities are connected. Such problems are called Traveling-salesman problem (TSP).

We can model the cities as a complete graph of n vertices, where each vertex represents a city.

It can be shown that TSP is NPC.

If we assume the cost function c satisfies the triangle inequality, then we can use the following approximate algorithm.

Triangle inequality

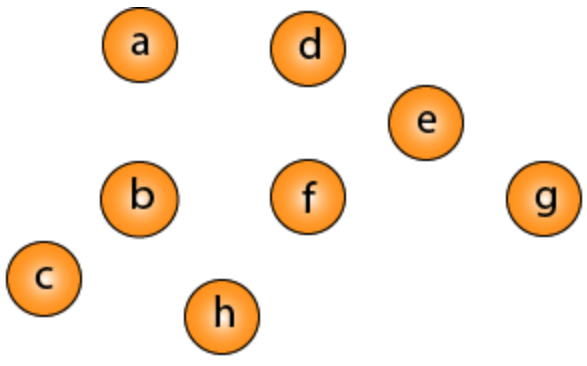
Let u, v, w be any three vertices, we have

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

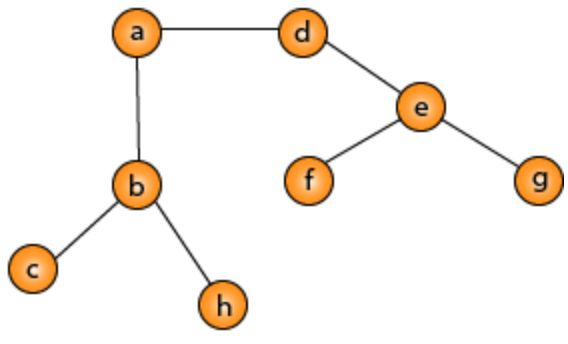
One important observation to develop an approximate solution is if we remove an edge from H*, the tour becomes a spanning tree.

1. Approx-TSP (G= (V, E))
2. {
3. 1. Compute a MST T of G;
4. 2. Select any vertex r is the root of the tree;
5. 3. Let L be the list of vertices visited in a preorder tree walk of T;
6. 4. Return the Hamiltonian cycle H that visits the vertices in the order L;
7. }

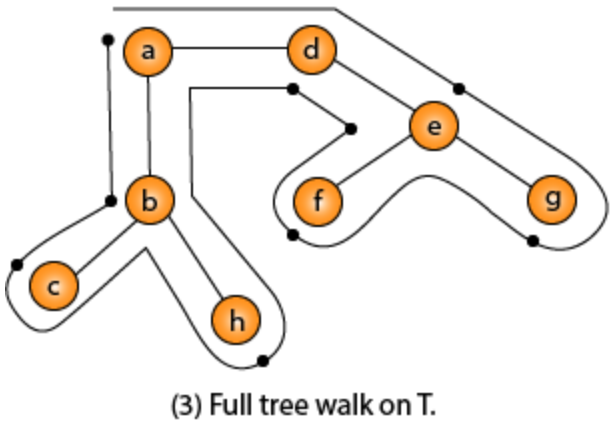
Traveling-salesman Problem



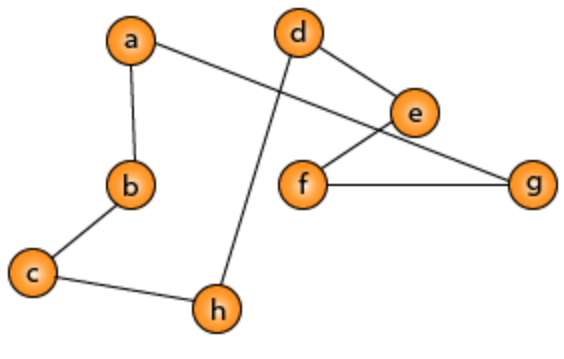
(1) A given set of points



(2) MST T



(3) Full tree walk on T.



(4) A preorder sequence gives a tour H.

Intuitively, Approx-TSP first makes a full walk of MST T, which visits each edge exactly two times. To create a Hamiltonian cycle from the full walk, it bypasses some vertices (which corresponds to making a shortcut)