INT202
Complexity of Algorithms

NP-Completeness

Year 2020-2021

Optimization Problems

Optimization Problems

- We have some problem instance x that has many feasible “solutions”.
- We are trying to minimize (or maximize) some cost function c(S) for a “solution” S to x. For example,
  - Finding a minimum spanning tree of a graph
  - Finding a smallest vertex cover of a graph

Optimization Problems

Solution Approach

- for each instance x we have a set of feasible solutions F(x)
- each solution $s \in F(x)$ has a cost c(s)
- the optimum cost OPT(x) = min { c(s): $s \in F(x)$} (or max)

Many computational problems are NP-complete optimization problems

- find a node cover with the smallest number of nodes
- find the shortest TSP (c(s) = length of the path)

Approximation Ratios

- How to cope with NP-Completeness?
  - at least try to find as good solution as possible
- How to measure “goodness” of a solution
  - how close to the optimal solution

    approximation ratio
- Let the algorithm returns solution T(x)
  - one way is to look at c(T(x))/c(OPT(x)) (for minimization problems)

Approximation Ratios

An approximation produces a solution T

- T is a **k-approximation** to the optimal solution OPT if $c(T)/c(OPT) \leq k$ (assuming a min. problem)
- T is a **k-approximation** to the optimal solution OPT if $c(OPT)/c(T) \leq k$ (assuming a max. problem)
- The value of k is never less than 1.
- A 1-approximation algorithm is the optimal solution.
- The goal is to find a polynomial-time approximation algorithm with small constant approximation ratios.

Polynomial-Time Approximation Schemes

- Approximation scheme is an approximation algorithm that takes $\epsilon>0$ as an input such that for any fixed $\epsilon>0$ the scheme is $(1+\epsilon)$-approximation algorithm.
- A problem L has a **polynomial-time approximation scheme (PTAS)** if it has a polynomial-time $(1+\epsilon)$-approximation algorithm, for any fixed $\epsilon>0$ (this value can appear in the running time).
- As the ϵ decreases the running time of the algorithm can increase rapidly:
    - For example it might be $O(n^{2/\epsilon})$

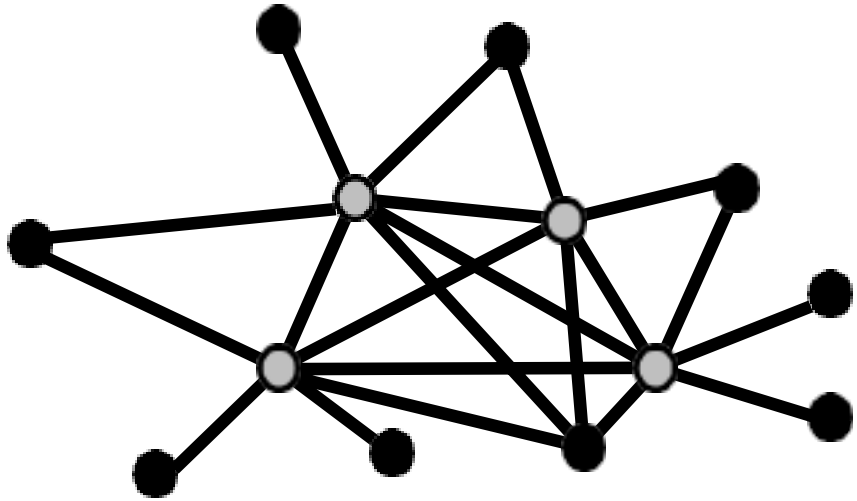
Fully Polynomial-Time Approximation Schemes

- We have **fully polynomial-time approximation scheme** when its running time is polynomial not only in n but also in $1/\epsilon$

- For example it could be $O((1/\epsilon)^3 n^2)$.

Vertex Cover

- G = (V, E) is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both (there is a vertex in V' "covering" every edge in E).
- OPT-VERTEX-COVER: Given an graph G, find a vertex cover of G with smallest size.
- OPT-VERTEX-COVER is NP-hard.



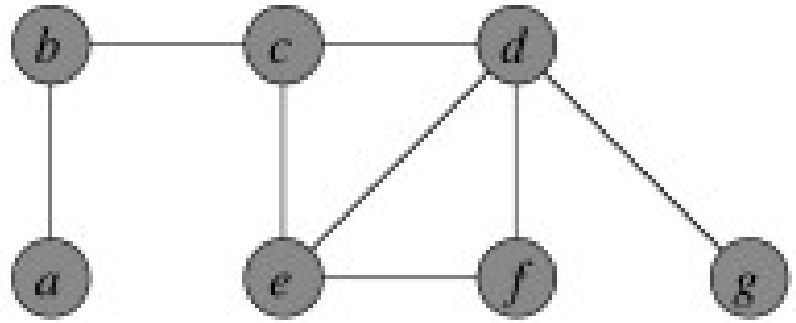
Approx-Vertex-Cover

Vertex Cover can be approximated by the following surprisingly simple algorithm, which iterately chooses an edge that is not covered yet and covers it using both incident vertices:

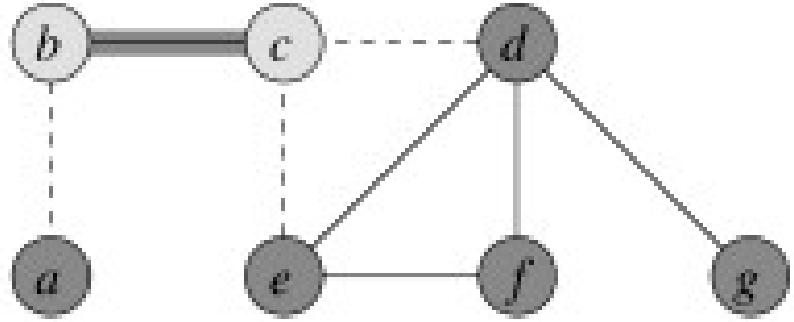
```
APPROX-VERTEX-COVER(G)
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```

Approx-Vertex-Cover

Suppose we have this input graph:

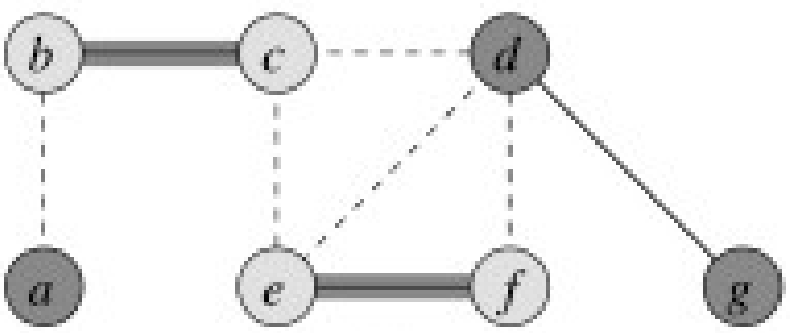


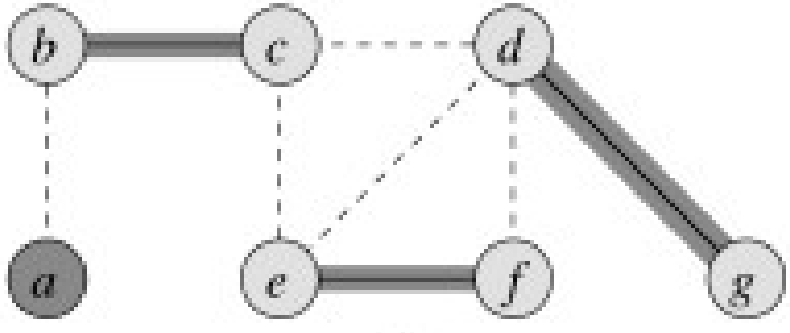
Suppose then that edge {b, c} is chosen. The two incident vertices are added to the cover and all other incident edges are removed from consideration:



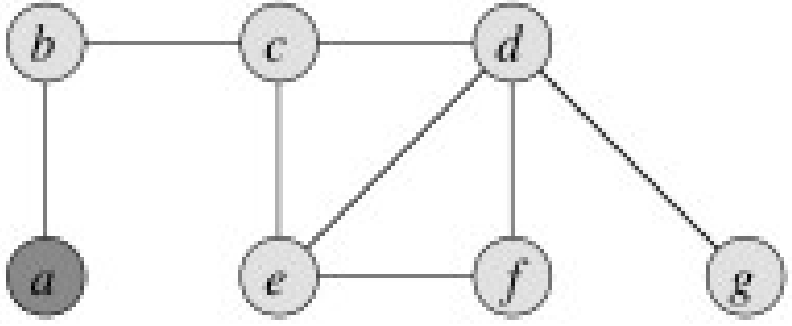
Approx-Vertex-Cover

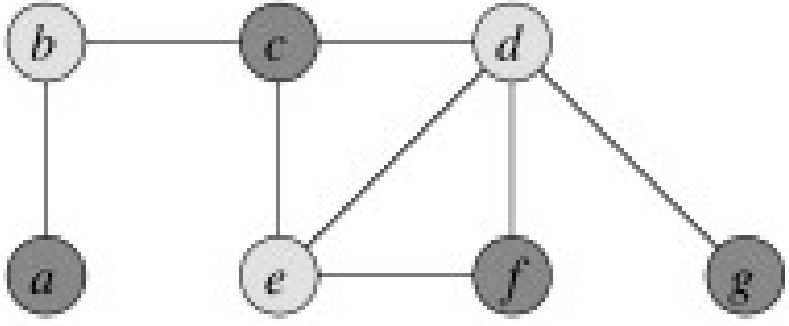
Iterating now for edges {e, f} and then {d, g}:





The resulting vertex cover is shown on the left and the optimal vertex on the right (the lighter colored vertices are in the cover):





Approx-Vertex-Cover

How good is the approximation? We can show that the solution is within a factor of 2 of optimal.

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

2-Approximation: to show that the solution is no more than twice the size of the optimal cover. We'll do so by finding a lower bound on the optimal solution C*.

Approx-Vertex-Cover

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

- Let A be the set of edges chosen in line 4 of the algorithm. Any vertex cover must cover at least one endpoint of every edge in A.
- No two edges in A share a vertex, because for any two edges one of them must be selected first (line 4), and all edges sharing vertices with the selected edge are deleted from E′ in line 6, so are no longer candidates on the next iteration of line 4.
- Therefore, in order to cover A, the optimal solution C* must have at least as many vertices as edges in A:

$$|A| \leq |C^*|$$

Approx-Vertex-Cover

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

- Since each execution of line 4 picks an edge for which neither endpoint is yet in C and line 5 adds these two vertices to C, then we know that: $|C| = 2|A|$
- Therefore: $|C| \leq 2|C^*|$
- That is, |C| cannot be larger than twice the optimal, so is a 2-approximation algorithm for Vertex Cover.
- This is a common strategy in approximation proofs: we don't know the size of the optimal solution, but we can set a lower bound on the optimal solution and relate the obtained solution to this lower bound.

TSP Approximations

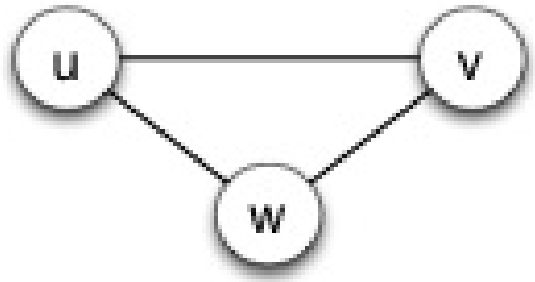
- In the Traveling Salesperson Problem (TSP) we are given a complete undirected graph G = (V, E) (representing, for example, routes between cities) that has a nonnegative integer cost c(u, v) for each edge {u, v} (representing distances between cities), and must find a Hamiltonian cycle or tour with minimum cost.
    - A Hamiltonian cycle is a closed loop on a graph where every node (vertex) is visited exactly once
- We define the cost of such a cycle A to be the sum of the costs of edges:

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

Triangle Inequality TSP

- The unrestricted TSP is very hard, so we'll start by looking at a common restriction.
- In many applications (e.g., Euclidean distances on two dimensional surfaces), the TSP cost function satisfies the triangle inequality:

$$c(u, v) \leq c(u, w) + c(w, v), \quad \forall\, u, v, w \in V.$$



- Essentially this means that it is no more costly to go directly from u to v than it would be to go between them via a third point w.

Triangle Inequality TSP

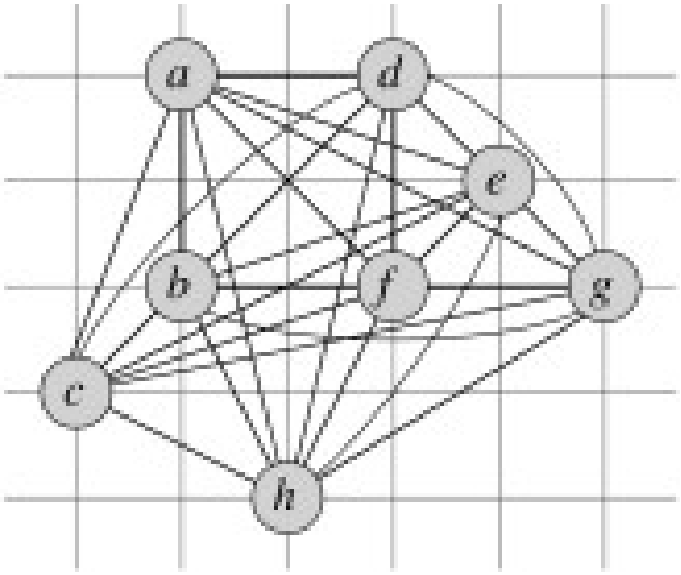
- The triangle inequality TSP is still NP-Complete, but there is a 2-approximation algorithm for it. The algorithm finds a minimum spanning tree, and then apply pre-order traversal:

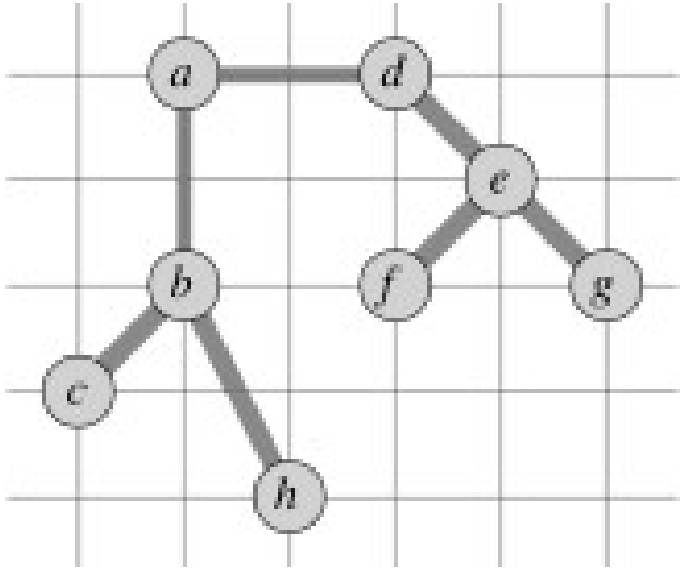
APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H

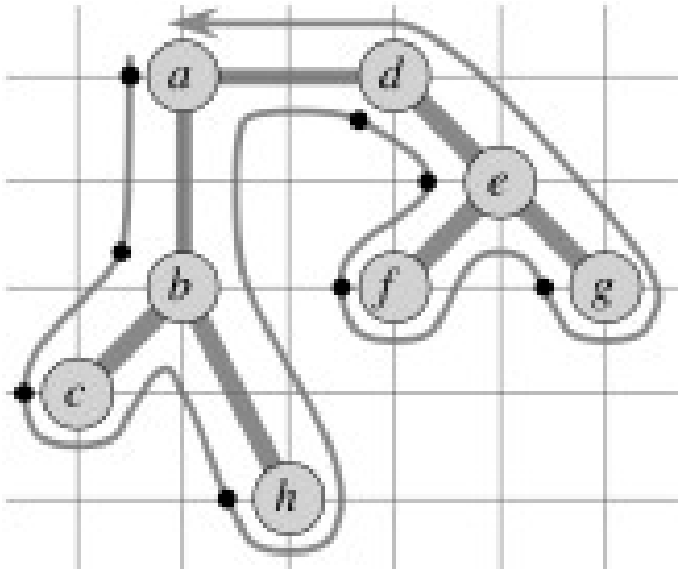
Triangle Inequality TSP

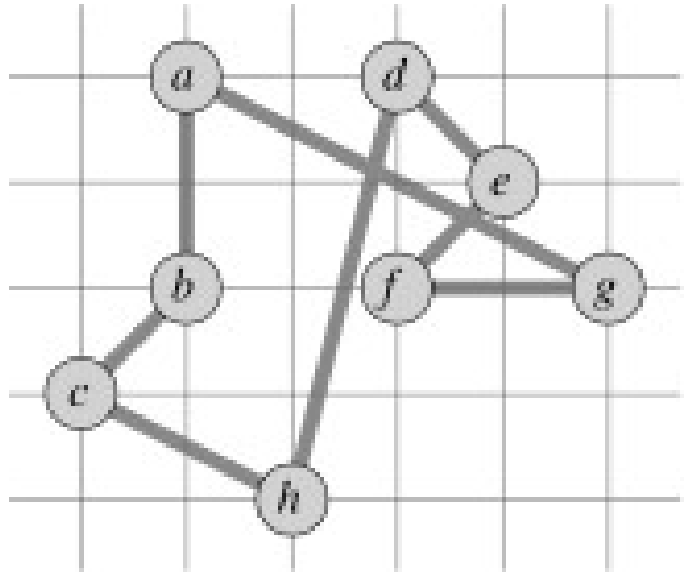
- Suppose we are working on the graph shown below to the left.
- Vertices are placed on a grid so you can compute distances if you wish.
- The MST starting with vertex a is shown to the right.



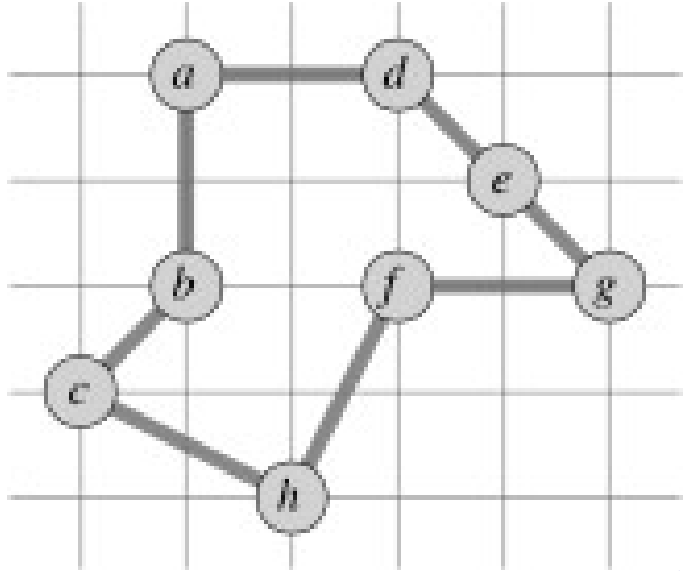


- Starting with vertex a, the preorder walk visits vertices in order a, b, c, h, d, e, f, g (cost 19.074).





- The optimal solution is shown below (cost 14.715).



Triangle Inequality TSP

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H

- Theorem: Approx-TSP-Tour is a polynomial time 2-approximation algorithm for TSP with triangle inequality.
- Let T be the spanning tree found in line 2, H be the tour found and H* be an optimal tour for a given problem.
- If we delete any edge from H*, we get a spanning tree that can be no cheaper than the minimum spanning tree T, because H* has one more (nonegative cost) edge than T:

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

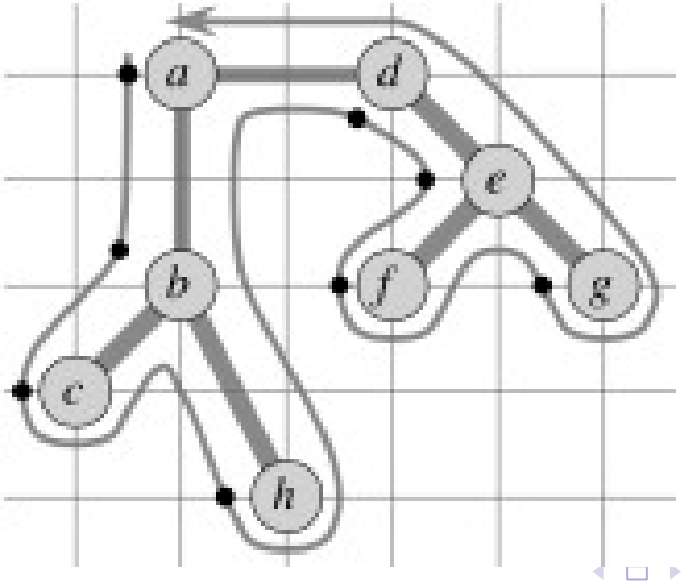
Triangle Inequality TSP

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H

- Consider the cost of the full walk W that traverses the edges of T exactly twice starting at the root. (For our example, W is ⟨{a, b}, {b, c}, {c, b}, {b, h}, {h, b}, {b, a}, {a, d}, ... {d, a}⟩.) Since each edge in T is traversed twice in W:

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

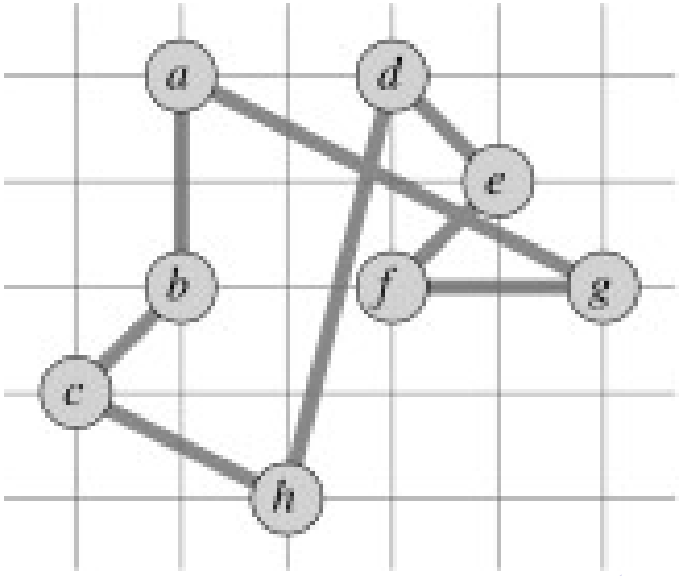


Triangle Inequality TSP

APPROX-TSP-TOUR(G, c)

1 select a vertex $r \in G.V$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let H be a list of vertices, ordered according to when they are first visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H

- This walk W visits some vertices more than once, so we can skip the redundant visits to vertices once we have visited them, producing the same tour H as in line 3.
- For example, instead of ⟨{a, b}, {b, c}, {c, b}, {b, h}, ... ⟩, go direct: ⟨{a, b}, {b, c}, {c, h}, ... ⟩.



Triangle Inequality TSP

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

- By the triangle inequality, which says it can't cost any more to go direct between two vertices,

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

- Noting that H is the tour constructed by Approx-TSP-Tour, and putting all of these together:

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

- So: $c(H) \leq 2\,c(H^*)$
- Thus Approx-TSP-Tour is a 2-approximation algorithm for TSP.

The General TSP

- Above we got our results using a restriction on the TSP. Unfortunately, the general problem is harder.

- Theorem: If $P \neq NP$, then for any constant $k \geq 1$ there is no polynomial time approximation algorithm with ratio k for the general TSP.

Summary of Strategies

- Faced with an NP Hard optimization problem, your options include:
  - Use a known exponential algorithm and stick to small problems.
  - Figure out whether you can restrict your problem to a special case for which polynomial solutions are known
    - e.g.: using a DAG instead of a general graph.
  - Give up on optimality, and find or design an approximation algorithm that gives "good enough" results.
    - Some of the possible options:
      - Design a clever approximation using some heuristic (e.g., as for vertex cover and TSP in this lecture).
      - Get lucky and show that randomly choosing a solution is good enough.