Advanced Algorithms

Lecture 11
Approximation Algorithms
for NP-Complete Problems

Tung Kieu
tungkvt@cs.aau.dk

ILO of Lecture 11

- Approximation algorithms
  - to understand the concepts of **approximation ratio** and **approximation algorithm**;
  - to understand the examples of approximation algorithms for the problems of vertex-cover and traveling-salesman.

Agenda

- P, NP, and NP-complete
- Approximation ratio, approximation algorithm, and approximation scheme
- Approximation algorithm for vertex-cover
- Approximation algorithm for traveling-salesman

P, NP, NP-complete

- P
  - Problems that are solvable in polynomial time, $n^{O(1)}$.
- NP
  - Problems that are verifiable in polynomial time, $n^{O(1)}$.
- NP-complete
  - A problem is in NP, and is as hard as any problem in NP.
  - No polynomial-time algorithm has yet been discovered.
  - Nobody has yet been able to determine conclusively whether NP-complete problems are in fact solvable in polynomial time
- $P = NP$ or that $P \neq NP$.
  - ?

Example

- Subset sum problem
  - Given a set of n integers, is there a non-empty subset whose sum is x, e.g., 0?
    - Consider set {−3, −2, 1, 5, 8}
- NP?
  - Yes, given any subset, you can verify if its sum is *x* in linear time $O(n)$.
    - Is sum of {1, 5, 8} = 10?
- P?
  - No, in the worst case, in order to identify a non-empty subset whose sum is x, we need to enumerate all 2^n possible subsets, thus having exponential runtime

Summary

Problems	Verifiable in Polynomial time	Solvable in polynomial time
P	Yes	Yes
NP	Yes	Yes or Unknown
NP-Complete	Yes	Unknown

Handling NP-complete problems

- Many interesting and important problems are NP-complete.
  - Knapsack problem
  - Travelling salesman problem
- NO! We have some ways to deal with an NP-complete problem.
  - If the actual inputs are small, an algorithm with exponential running time may be acceptable.
  - Come up approaches to find *near-optimal*-- approximation algorithm solutions in polynomial time.
  - Use *heuristics* to speed up exponential running time.

Agenda

- P, NP, and NP-complete
- Approximation ratio, approximation algorithm, and approximation scheme
- Approximation algorithm for vertex-cover
- Approximation algorithm for traveling-salesman

Approximation ratios

- Suppose that
  - we are working on an *optimization* problem with input size n;
  - each solution has a *cost* value, and we want to identify the optimal solution, i.e., the one with the minimum or maximum possible cost;
  - optimal solution is C^*, returned by an **exact** algorithm that runs in **exponential** time;
  - approximate solution is C, returned by an **approximation** algorithm that runs in **polynomial** time.
- Maximization problem:
  - $0 < C \leq C^*$, C^*/C gives a factor.
  - E.g., C^*=100, C=90, C^*/C = 10/9
- Minimization problem:
  - $0 < C^* \leq C$, C/C^* gives a factor.
  - E.g., C*=100, C=110, C/C^* = 11/10

Approximation ratios

- A ρ(n)-approximation algorithm has an approximation ratio ρ(n), if, for any input size of n, it satisfies $\max(\frac{C}{C*}, \frac{C*}{C}) \leq$ ρ(n) .
  - C is control by ratio ρ(n).
  - It provides a guarantee on the performance of an approximation algorithm.
    - Consider a 1.2-approximation algorithm with optimal cost C*=100.
    - For a minimization problem, the algorithm returns a value that is no larger than 100*1.2=120.
    - For a maximization problem, the algorithm returns a value that is no smaller than 100/1.2=83.3.
- Approximation ratio is never smaller than 1.
- 1-approximation algorithm produces the optimal solution.

Approximation scheme

- An **approximation scheme** for an optimization problem is an **approximation algorithm** that takes as input
  - The problem and a value $\varepsilon > 0$.
  - Then, the scheme is a $(1+\varepsilon)$-approximation algorithm.
- Polynomial-time approximation scheme, PTAS
  - Scheme runs in polynomial time of input size *n* for any fixed $\varepsilon > 0$, e.g., $O(n^{2/\varepsilon})$
- Fully polynomial-time approximation scheme, FPTAS
  - Scheme runs in polynomial time of both input size n and $1/\varepsilon$, e.g., $O((1/\varepsilon)^2 n^3)$

Exam 2018

5. Take a careful look at the following statements and decide if they are correct.

5.1 (*2 points*) Consider an approximation algorithm with approximation ratio 1.1 for solving a NP-complete problem P. Assume that P is a **maximization** problem and its optimal solution is 100. Then, the approximation algorithm may return a value 105.

☐ 1) Correct ☐ 2) Wrong

5.2 (*2 points*) Consider an approximation algorithm with approximation ratio 2 for solving a NP-complete problem P. Assume that P is a **minimization** problem and its optimal solution is 100. Then, the approximation algorithm may return a value 201.

☐ 1) Correct ☐ 2) Wrong

Exam 2018

5. Take a careful look at the following statements and decide if they are correct.

5.1 (*2 points*) Consider an approximation algorithm with approximation ratio 1.1 for solving a NP-complete problem P. Assume that P is a **maximization** problem and its optimal solution is 100. Then, the approximation algorithm may return a value 105.

☐ **1)** Correct ☐ **2)** Wrong

5.2 (*2 points*) Consider an approximation algorithm with approximation ratio 2 for solving a NP-complete problem P. Assume that P is a **minimization** problem and its optimal solution is 100. Then, the approximation algorithm may return a value 201.

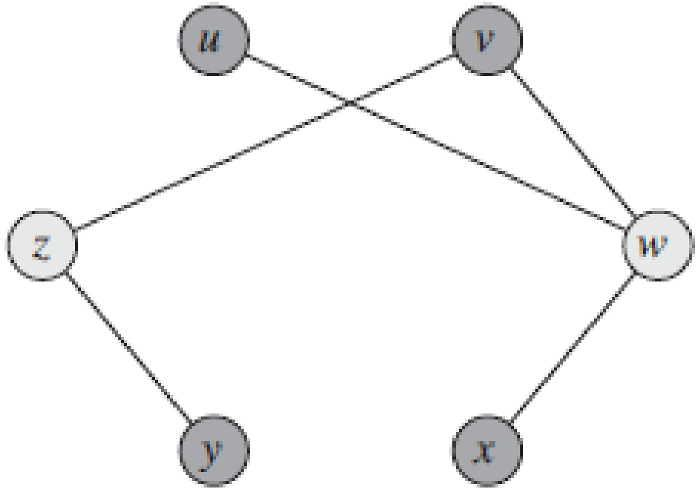
☐ **1)** Correct ☐ **2)** Wrong

Agenda

- P, NP, and NP-complete
- Approximation ratio, approximation algorithm, and approximation scheme
- Approximation algorithm for vertex-cover
- Approximation algorithm for traveling-salesman

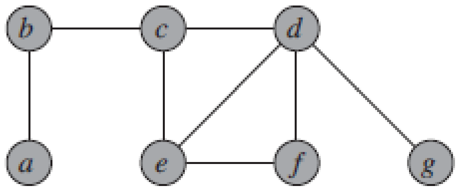
The vertex-cover problem

- Given an undirected graph $G = (V, E)$
- A vertex cover of G is a ***subset of vertices*** $V' \subseteq V$, s.t.
  - For each $(u, v) \in E$, we have $u \in V'$ or $v \in V'$ or both.
  - $V_1' = \{u, v, w, x, y, z\}$
  - $V_2' = \{w, z\}$
  - $V_3' = \{u, v, y, x\}$
- The size of a vertex cover is the number of vertices in it.
  - Sizes of V_1', V_2', and V_3' are 6, 2, and 4, respectively.
- Vertex-cover problem: find a vertex cover of minimum size.

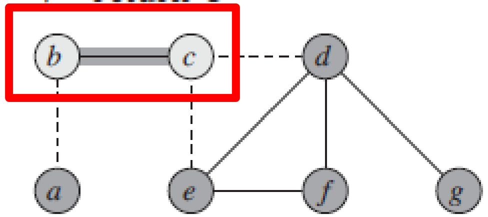


Approximation algorithm

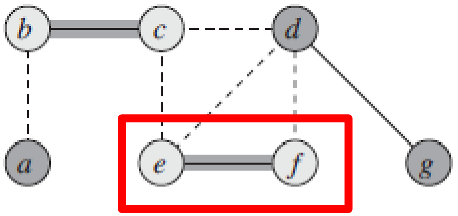
```
APPROX-VERTEX-COVER(G)
C = ∅
E′ = G.E
while E′ ≠ ∅
    let (u, v) be an arbitrary edge of E′
    C = C ∪ {u, v}
    remove from E′ every edge incident on either u or v
return C
```



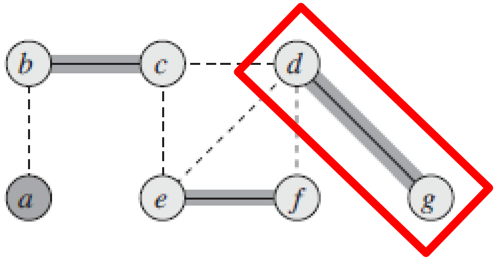
$C = \{\emptyset\}$
$E' = \{ab, bc, cd, ce, de, df, dg, ef\}$



$C = \{b, c\}$
$E' = \{de, df, dg, ef\}$



$C = \{b, c, e, f\}$
$E' = \{dg\}$



$C = \{b, c, e, f, d, g\}$
$E' = \{\emptyset\}$

Run time: $O(V + E)$
$C^* = \{b, d, e\}$

Approximation ratio

```
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
```

- Approximation ratio
  - $C^* = \{b, d, e\}, C = \{b, c, e, f, d, g\}$
  - $\frac{C}{C^*} = \frac{6}{3} = 2$
- What if we are lucky (i.e., having a lucky order in line 4), can we get a better solution or even exact solution?
  - Visit (d, e) first. Then $(b, c) \rightarrow C = \{d, e, b, c\}$
  - $\frac{C}{C^*} = \frac{4}{3}$ better than 2.
- What is the approximation ratio then?
  - Observing from the two examples, it should be at least 2.
  - Then, we need to prove the ratio.

Approximation ratio

- Let A denote the set of edges that line 4 picked. $A \subseteq E$
- To cover the edges in A, any vertex cover must include at least one endpoint of each edge in A.
  - This is due to the definition of a vertex cover: a vertex cover contains at least one vertex of each edge.
  - The optimal vertex cover C^* should also include at least one endpoint of each edge in A.
- No two edges in A share an endpoint
  - Once an edge is picked in line 4 and is added into A, all the edges that share the edge's endpoints are deleted from E' in line 6.
- Thus, we have the lower bound $|C^*| \geq |A|$.

Approximation ratio

- When line 4 picks an edge, both endpoints of the edge are added into C.
  - We have $|C| = 2|A|$
- Considering the lower bound $|C^*| \geq |A|$, we have
  - $|C| = 2|A| \leq 2|C^*|$
  - Approximation ratio: $\frac{|C|}{|C^*|} \leq 2$
- Conclusion: we have a 2-approximation algorithm.

Reflection on approximation ratio proof

- How can we possibly prove the approximation ratio without even knowing the size of an optimal solution?
  - Instead of knowing the exact size of an optimal solution, we rely on a lower bound on the size of an optimal solution.
    - Vertex-cover problem: $|C^*| \geq |A|$
  - Next, we consider the relationship between the result returned by an approximation and the lower bound.
    - $|C| = 2|A|$
- This is a common ***methodology*** used in approximation ratio proof.

Agenda

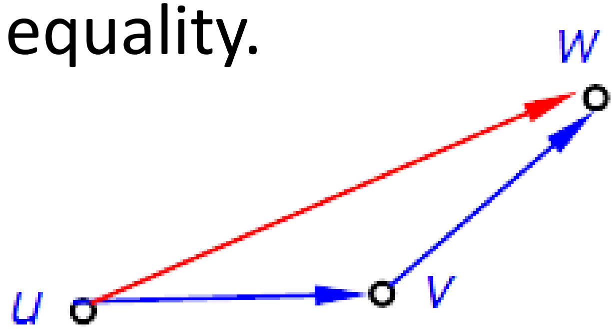
- P, NP, and NP-complete
- Approximation ratio, approximation algorithm, and approximation scheme
- Approximation algorithm for vertex-cover
- Approximation algorithm for traveling-salesman

Traveling-salesman problem (TSP)

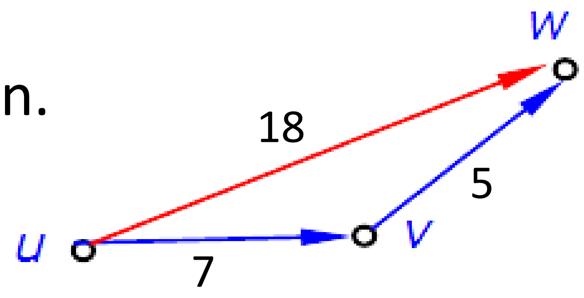
- Given
  - A list of cities and the distance between each pair of cities.
- Compute
  - the shortest *path that visits each city exactly once and returns to the origin city* or shortest *simple cycle with all vertices*?
- Given a **complete** undirected graph $G = (V, E)$
  - Every pair of vertices is connected by an edge.
  - Each vertex has $V - 1$ edges to all remaining vertices.
- For each edge $(\mathrm{u}, \mathrm{v}) \in E$, it has a nonnegative integer cost $c(u, v)$, e.g., the Euclidean distance.
- Identify a Hamiltonian cycle of G with minimum cost.
  - A Hamiltonian cycle is a simple cycle that contains each vertex in V.
  - A simple cycle is a path $(v_0, v_1, v_2, \dots, v_k)$ where $v_0 = v_k$ and $v_1, v_2, \dots, v_k$ are distinct.

Simplified TSP

- The cost function c satisfies triangle inequality.
- For all $u, v, w \in V$:
  - $c(u,w) \ \le \ c(u,v) \ + \ c(v,w)$



- These are natural simplifications
  - *Vertices* – points in the plane.
  - *Cost of an edge* – Euclidean distance between the two vertices of the edge.
- General TSP
  - Without the triangle inequality assumption.



Another simpler TSP

- A **spanning tree** T for a connected graph *G* is a tree that includes all the vertices of G. Then we want to find a spanning tree of minimum cost---**minimum spanning trees** problem.
- Is Hamiltonian cycle a tree?
  - No, because it is a cycle.
- Can we change a Hamiltonian cycle to a tree?
  - Yes, by deleting an edge to break the cycle.

Approximation algorithm

```
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
```

- Choose a vertex, say vertex r, as root.
- Compute a MST from the chosen root r.
- Preorder tree work on the MST.
    - Visits each vertex before visiting its children.

MST-Prim’s algorithm

```
MST-Prim(G,r)
01 for each vertex u ∊ G.V()
02    u.setkey(∞)
03    u.setparent(NIL)
04 r.setkey(0)
05 Q.init(G.V())     // Q is a priority queue ADT
06 while not Q.isEmpty()
07    u ← Q.extractMin()    // making u part of T
08    for each v ∊ u.adjacent() do
09       if v ∊ Q and G.w(u,v) < v.key() then
10          v.setkey(G.w(u,v))
11          Q.modifyKey(v)
12          v.setparent(u)
```

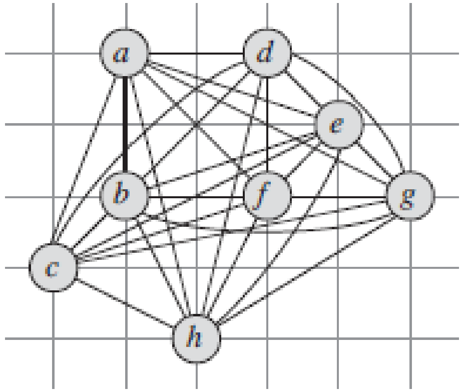
Initialize all vertices: $O(|V|)$

Initialize a priority queue with $|V|$ elements. $O(|V|)$

While loop: $|V|$ times
Line 7: Each Q.**extractMin**() takes $O(\lg |V|)$, in total $O(|V| \lg |V|)$.
Line 8: For loop: given a vertex, it iterates on all its adjacent vertices.
Together with the while loop, in total it iterates $|E|$.
Each iteration it calls Q.**modifyKey**, which takes $O(\lg |V|)$.
In total, $O(|E| \lg |V| + |V| \lg |V|) = O(|E| \lg |V|)$.

Example

- Choose vertex a as the root
- Identify the MST from a.



a	b	c	d	e	f	g	h
0/-	∞	∞	∞	∞	∞	∞	∞

b	d	c	e	f	g	h
$2/a$	$2/\mathrm{a}$	…	…	…	…	…

c	d	h	e	f	g
$\sqrt{2}/b$	$2/\mathrm{a}$	$\sqrt{5}/b$	…	…	…

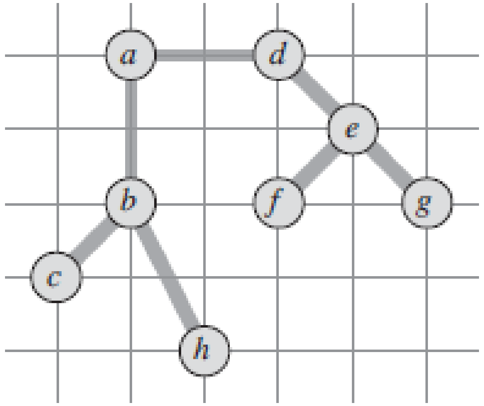
d	h	f	g	e
$2/\mathrm{a}$	$\sqrt{5}/b$	…	…	…

e	f	h	g
$\sqrt{2}/\mathrm{d}$	$2/\mathrm{d}$	$\sqrt{5}/b$	…

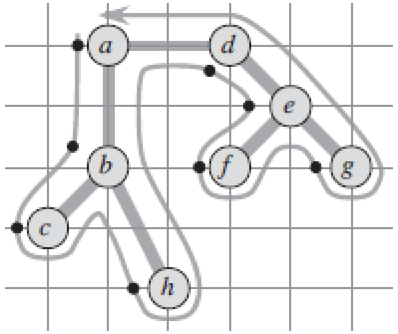
f	g	h
$\sqrt{2}/\mathrm{e}$	$\sqrt{2}/\mathrm{e}$	$\sqrt{5}/b$

g	h
$\sqrt{2}/\mathrm{e}$	$\sqrt{5}/b$

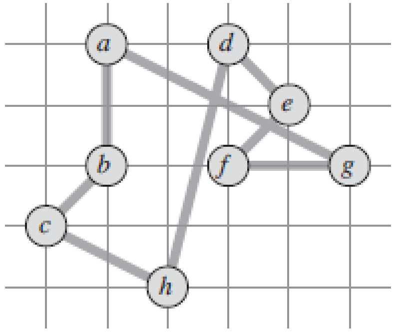
h
$\sqrt{5}/b$



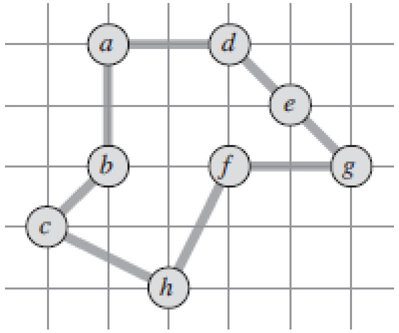
Pre-order tree walk on the MST



- $\{a, b, c, d, e, f, g, h\} \rightarrow \{a, b, c, h, d, e, f, g\}$
- Add the root a to the end, so we have $\langle a, b, c, h, d, e, f, g, a \rangle$



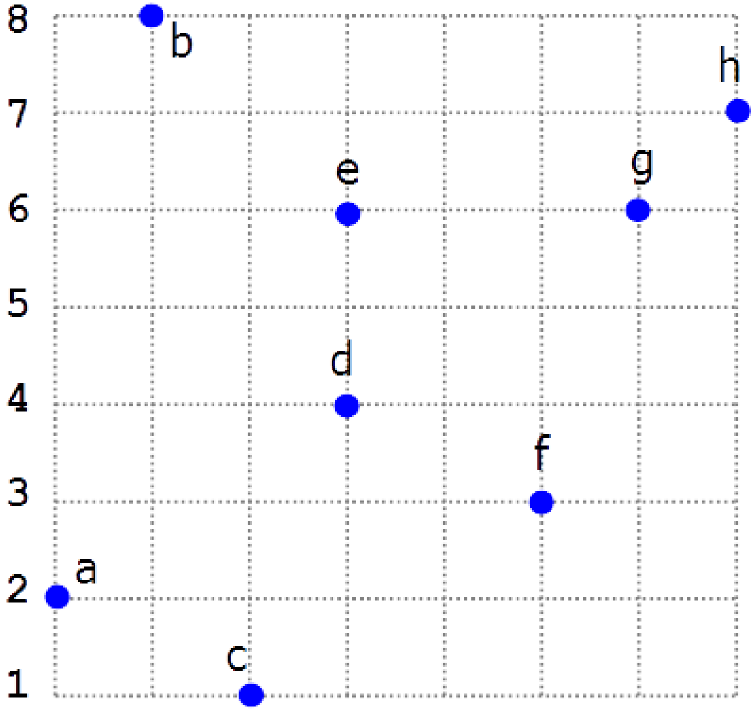
C=19.074



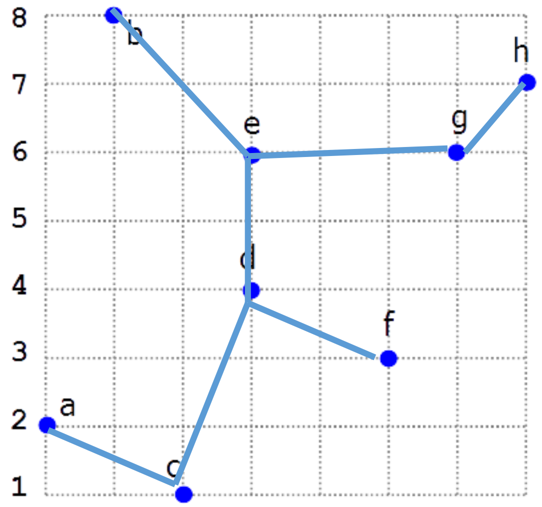
C*=14.715

Mini quiz (also on Moodle)

- Compute an approximate TSP tour:
  - Use vertex *a* as the starting vertex
  - When there is a choice (in Prim's and the pre-order tree walk), choose the alphabetically “smaller” vertex.



Solution



f	b	g	h
$\sqrt{5}$/d	$\sqrt{8}/\mathrm{e}$	3/e	...

b	g	h
$\sqrt{8}/\mathrm{e}$	3/e	...

g	h
3/e	...

h
$1/g$

a	b	c	d	e	f	g	h
0/-	∞	∞	∞	∞	∞	∞	∞

c	d	e	f	b	g	h
$\sqrt{5}/a$	$\sqrt{13}/\mathrm{a}$	5/a	$\sqrt{26}/\mathrm{a}$	$\sqrt{37}/\mathrm{a}$	...	...

d	f	e	b	g	h
$\sqrt{10}/\mathrm{c}$	$\sqrt{13}/\mathrm{c}$	5/a	$\sqrt{37}/\mathrm{a}$	...	...

e	f	b	g	h
2/d	$\sqrt{5}$/d	$\sqrt{20}/\mathrm{d}$	...	...

Pre-order tree walk:
$\{a, c, d, e, f, b, g, h\} \rightarrow \{a, c, d, e, b, g, h, f\}$

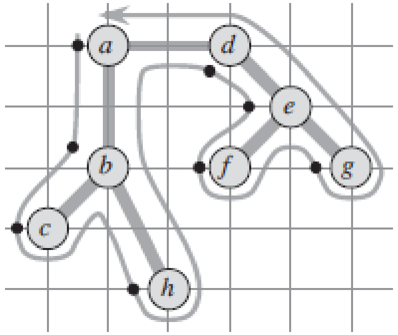
Approximate result:
$\langle a, c, d, e, b, g, h, f, a \rangle$

Approximation ratio

- Let H^* denote an optimal cycle and H denote the cycle identified by our approximation algorithm. Let T be a minimum spanning tree.
- By deleting any edge from H^*, we will get a spanning tree.
- Thus, we have $c(T) \leq c(H^*)$.
  - This is the lower bound.
- What is the relationship between $c(H)$ and the lower bound $c(T)$?
  - To this end, we introduce a new concept called **full walk**.

Approximation ratio

- A ***full walk*** of a tree T lists the vertices when they are first visited and also whenever they are returned to after a visit to a sub-tree.
- Full walk, denoted as W:
  - $[a, b, c, b, h, b, a, d, e, f, e, g, e, d, a]$.
  - $\langle a, b\rangle, \langle b, c\rangle, \langle c, b\rangle, \langle b, h\rangle, \langle h, b\rangle, \langle b, a\rangle, \langle a, d\rangle, \langle a, d\rangle,$ $\langle d, e\rangle, \langle e, f\rangle, \langle f, e\rangle, \langle e, g\rangle, \langle e, d\rangle, \langle d, a\rangle$.
  - $c(W) = 2c(T)$, as it contains every edge in the MST T twice.



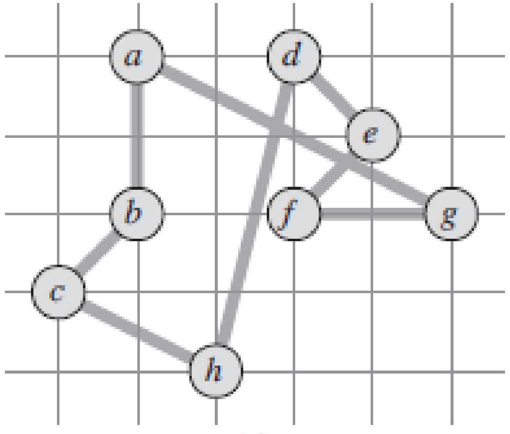
- Consider the lower bound $c(T) \leq c(H^*)$, we have
  - $c(W) \leq 2c(H^*)$
- What is the relationship between the full walk W and the approximated cycle H?
  - $c(W)$ and $c(H)$?

Approximation ratio

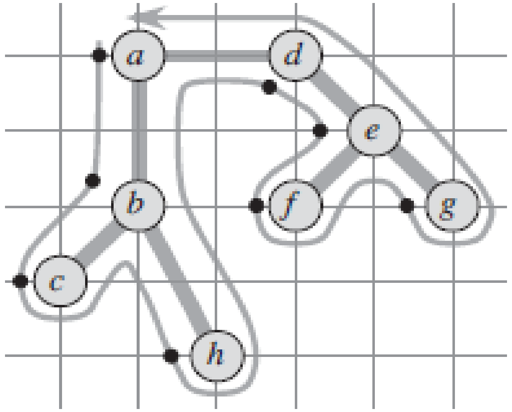
- Pre-order walk: $\{a, b, c, h, d, e, f, g\}$
  - $H = \langle a, b\rangle, \langle b, c\rangle, \langle c, h\rangle, \langle h, d\rangle, \langle d, e\rangle, \langle e, f\rangle, \langle f, g\rangle, \langle g, a\rangle$
  - $W = \langle a, b\rangle, \langle b, c\rangle, \langle c, b\rangle, \langle b, h\rangle, \langle h, b\rangle, \langle b, a\rangle, \langle a, d\rangle, \langle d, e\rangle, \langle e, f\rangle,$
    $\langle f, e\rangle, \langle e, g\rangle, \langle g, e\rangle, \langle e, d\rangle, \langle d, a\rangle.$

H	W
$\langle a, b\rangle$	$\langle a, b\rangle$
$\langle b, c\rangle$	$\langle b, c\rangle$
$\langle c, h\rangle$	$\langle c, b\rangle, \langle b, h\rangle$
$\langle h, d\rangle$	$\langle h, b\rangle, \langle b, a\rangle, \langle a, d\rangle$
$\langle d, e\rangle$	$\langle d, e\rangle$
$\langle e, f\rangle$	$\langle e, f\rangle$
$\langle f, g\rangle$	$\langle f, e\rangle, \langle e, g\rangle$
$\langle g, a\rangle$	$\langle g, e\rangle, \langle e, d\rangle, \langle d, a\rangle$

H



W



Due to triangle inequality, we have
$c(H) \leq c(W)$. Thus, $c(H) \leq c(W) \leq 2c(H^*)$

Approximation ratio

- From $c(H) \leq c(W) \leq 2c(H^*)$, we have
  - $\frac{c(H)}{c(H^*)} \leq 2 \rightarrow$ Thus, approximation ratio is 2.
- This means that the approximate cycle will never have more than twice distance of the optimal cycle.

No efficient ρ-approximation

- Do all NP-complete problems have polynomial ρ-approximation algorithms (where ρ is a constant)?
  - No!
- Next, we will prove that the general TSP problem cannot have a polynomial ρ-approximation algorithm, unless $P = NP$.
- In the general TSP problem, we drop the assumption that the cost function c satisfies the triangle inequality.
    - E.g., use travel times as costs, but not Euclidean distances.

Proof sketch

- Given a graph $G = (V, E)$, a ***Hamiltonian cycle*** of G is a simple cycle that contains each vertex in V.
- What is the Hamiltonian-cycle problem?
  - A decision problem: does a graph G have a Hamiltonian cycle?
  - It is a NP-complete problem, Theorem 34.13.
  - Solving it in polynomial time implies $P = NP$, Theorem 34.4.
- Proof by *contradiction*:
  - Since the Hamiltonian cycle problem is NP-complete, no polynomial time algorithms exist unless $P = NP$.
    - If there exists a ***polynomial*** ρ-approximation algorithm A for solving general TSP, we are also able to use A to solve the Hamiltonian-cycle problem.
    - Recall that A is polynomial. This means that we use A to solve the Hamiltonian cycle problem in ***polynomial*** time.
    - This is a contradiction, unless $P = NP$. In other words, if $P \neq NP$, this is a contradiction.

Hamiltonian cycle problem to General TSP

- Suppose we have a polynomial time, approximation algorithm A with approximation ratio ρ for general TSP.
  - Assume ρ is an integer.
- We now show how to use A to solve the Hamiltonian cycle problem.
  - Given a graph $G = (V, E)$, whether or not there is a Hamiltonian cycle in G.
- We turn G into a ***complete*** graph $G' = (V, E')$
  - Assign an integer cost to each edge in E'.

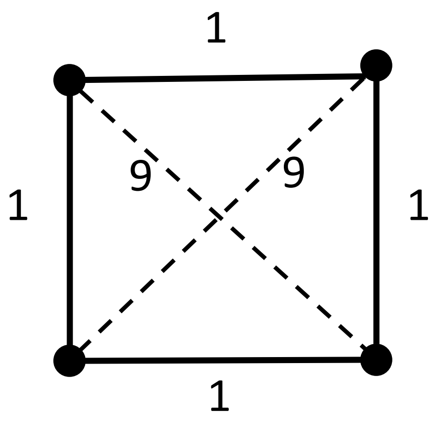
$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \rho|V|+1 & \text{otherwise} \end{cases}$$

  - For example, assuming that we have $\rho = 2$, then we have the edge weights of $2|V| + 1$ for all newly added edges in E'.
- Now we consider a general TSP on G' with cost function c.

Hamiltonian cycle problem to General TSP

$$c(u,v)=\begin{cases} 1 & \text{if } (u,v)\in E \\ \rho|V|+1 & \text{otherwise} \end{cases}$$

- Assume $\rho = 2.$ $|V| = 4.$



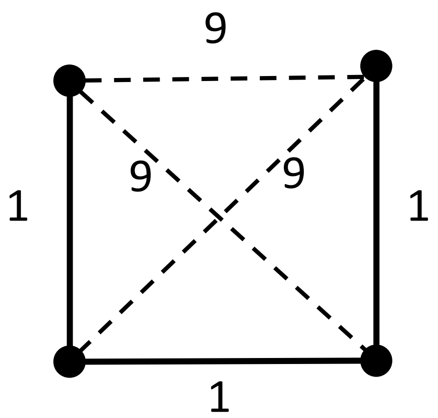
The weights do not satisfy triangle-inequality anymore.

- If the original graph G has a Hamiltonian cycle H^*
  - Each edge should have cost 1 and in total $|V|$ edges. Thus, H^*'s cost is $|V|$, i.e., $c(H^*) = |V|$.
    - This example: $c(H^*) = 4.$
  - If we use the ρ-approximation algorithm A, it will return a cycle H with cost at most $\rho|V|$, i.e., $c(H) \le \rho(H^*) = \rho|V|$.
    - This example: $c(H) \le 8.$

Hamiltonian cycle problem to General TSP

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ \rho|V|+1 & \text{otherwise} \end{cases}$$

- Assume $\rho = 2$. $|V| = 4$.



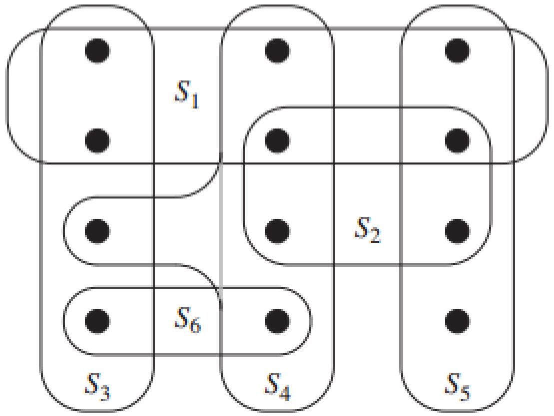
- If the original graph G does not have a Hamiltonian cycle
  - Then any Hamiltonian cycle in G' must use some (at least one) edges that are not in E, i.e., some newly added edges.
    - In the best case, we use only one newly added edge, we have $(\rho|V| + 1) + (|V| - 1) = \rho|V| + |V| > \rho|V|$
    - This example: $1 + 1 + 1 + 9 = 12 > \rho|V| = 8$

Hamiltonian cycle problem to General TSP

- So this means that, if the ρ-approximation algorithm A returns
  - A cycle whose cost is at most $\rho|V|$, G has a Hamiltonian cycle.
  - A cycle whose cost is more than $\rho|V|$, G has no Hamiltonian cycle.
- Therefore, we can use A to solve the Hamiltonian-cycle in polynomial time because A is a polynomial approximation algorithm.
- Since the Hamiltonian-cycle problem is NP-complete, there does not exist a polynomial time algorithm unless $P = NP$.
- This is a contradiction unless $P = NP$.

Set-covering problem

- The set-covering problem
- Given a finite set X and a family F of subsets of X. The problem is to find a minimum-size subset $C \subseteq F$ whose members cover all of X.
  - Black dots are the elements in X.
  - $F = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ each S_i contains some elements in X (black dots).
  - $C^* = \{S_3, S_4, S_5\}$



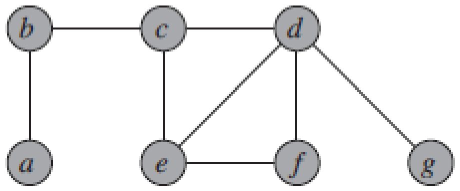
- A greedy approximation algorithm
  - At each stage, picking up the set S that covers the greatest number of remaining uncovered elements.
  - $C = \{S_1, S_4, S_5, S_3\}$
  - $(\ln |X| + 1)$-approximation algorithm
    - Approximation ratio is not a constant anymore.

Set-covering vs. vertex cover

2 In Lecture 11, we have seen a 2-approximation algorithm (denoted as ALG1) for solving the **vertex cover** problem. We also briefly talked about a $(\ln|X|+1)$-approximation algorithm (denoted as ALG2) for solving the **set cover** problem.

- (*10 points*) Actually, the **set cover** problem can be regarded as a generalization of the vertex cover problem. Show how can you transform a vertex cover problem into a set cover problem.

- X represents all edges.
- Each vertex is a subset of X, which contains the edges that are incident to the vertex.



- $X = \{ab, bc, cd, ce, de, df, dg, ef\}$
- $S_c = \{bc, cd, ce\}$

ILO of Lecture 11

- Approximation algorithms
  - to understand the concepts of **approximation ratio**, **approximation scheme**, **approximation algorithm**;
  - to understand the examples of approximation algorithms for the problems of vertex-cover and traveling-salesman.