Approximation algorithms

Map for today

- By the end of today you will be able to:
  - Define what is an approximation algorithm
  - Explain how to get a 2 approximation for vertex cover
  - Explain how to get a 2 approximation for metric TSP
  - Understand why, in general, no approximation is possible for general TSP
  - See the new problem Set-Cover

Approximation Algorithms

- Motivation:
  - Solving NP-hard problems exactly is very costly
  - What if we can do with non-exact solutions?

- Approximation algorithms:
  – An algorithm returning a near-optimal solution.

Vertex-cover problem

- Vertex cover: given an undirected graph $G=(V,E)$, then a subset $V' \subseteq V$ such that if $(u,v) \in E$, then $u \in V'$ or $v \in V'$ (or both).
- Size of a vertex cover: the number of vertices in it.
- Vertex-cover problem: find a vertex-cover of minimum size.
- For small k we have a good solution (Why?) . What if k is large?

Obvious algorithms?

- Can you come up with non exact but fast algorithms for Vertex-Cover?

APPROXIMATION RATIO

For a problem with input of length n:

- C* - the cost of optimal solution
- C - the cost obtained by an approximation algorithm
- **max(C/C*, C*/C) ≤ρ(*n*)**, where ρ(*n*) is a function
- If ρ(*n*)=1, then the algorithm is **optimal**
- The larger ρ(*n*), the worse the algorithm

In other words

- ρ (n) implies that for any input of size n, the solution the algorithm outputs C is within factor ρ (n) of outcome of optimal solution C^*, i.e

$$1 \leq \text{Max}\ (C/C^*,\ C^*/C) \leq \rho(n)$$

Minimization problem

Maximization problem

Approximation of vertex cover

Algorithm 1: APPROX-VERTEX-COVER(G)

---

1 $C \leftarrow \emptyset$
2 while $E \neq \emptyset$

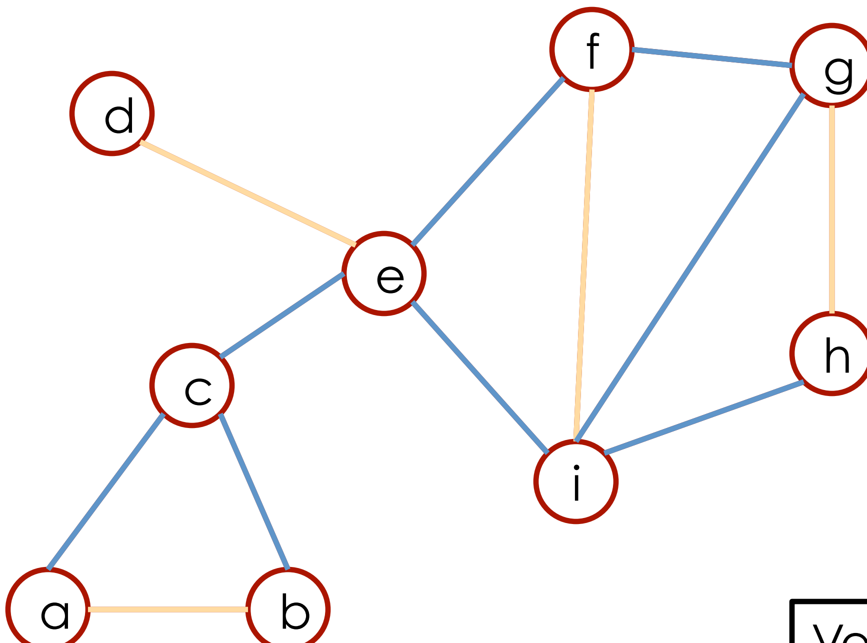
   pick any $\{u, v\} \in E$

   $C \leftarrow C \cup \{u, v\}$

   delete all eges incident to either u or v

return C

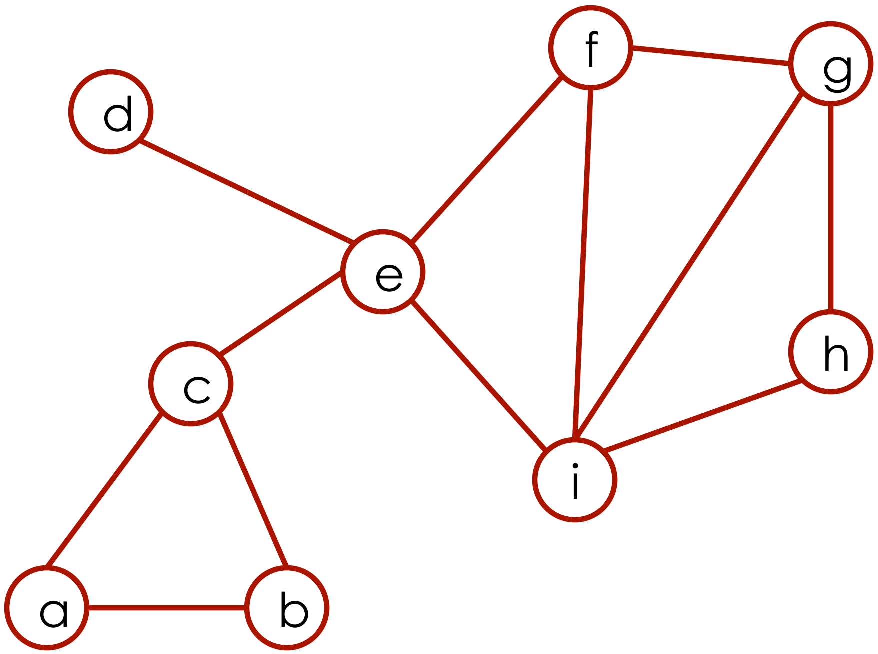
Example of approximate vertex-cover



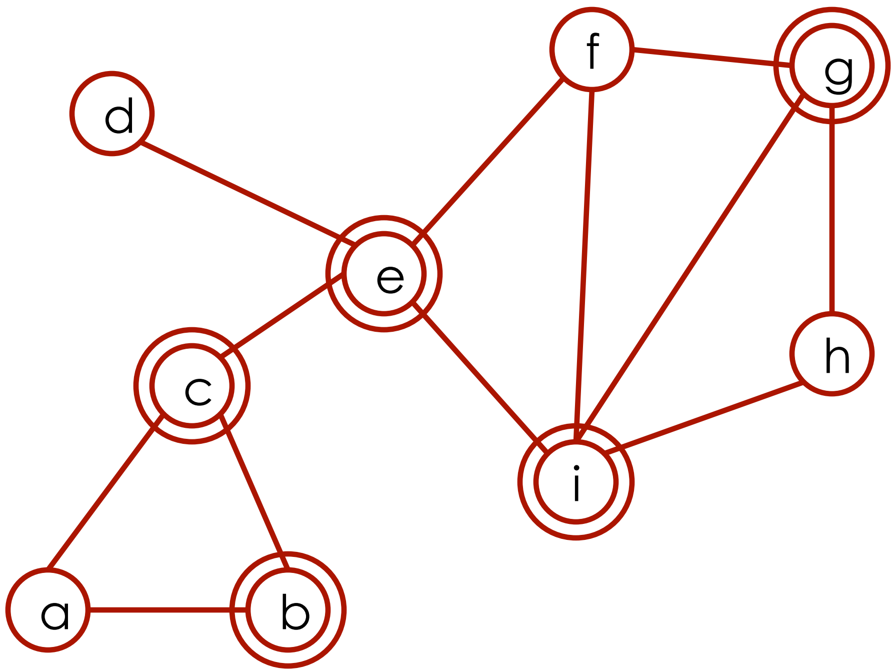
Vertex cover of 8 vertices

What is the optimal?

- We have 8, can we do better?



Optimal – 5 vertices



That was easy!

- Like most approximation algorithms, the algorithm is straightforward, the argument for the bound is the interesting point

2-approximate vertex-cover

- Theorem 35.1 (page 1026).
  - APPROXIMATE-VERTEX-COVER is a polynomial time 2-approximation algorithm
- We first note:
  - It runs in polynomial time
  - The returned C is a vertex-cover.

C is a vertex cover

- Given a graph G=(V,E) a matching M in G is a set of pairwise non-adjacent edges.
  - Meaning: No two edges share a common vertex
- The algorithms finds a **maximal** matching
  - Maximal = can not be extended, not maximum!
  - If the

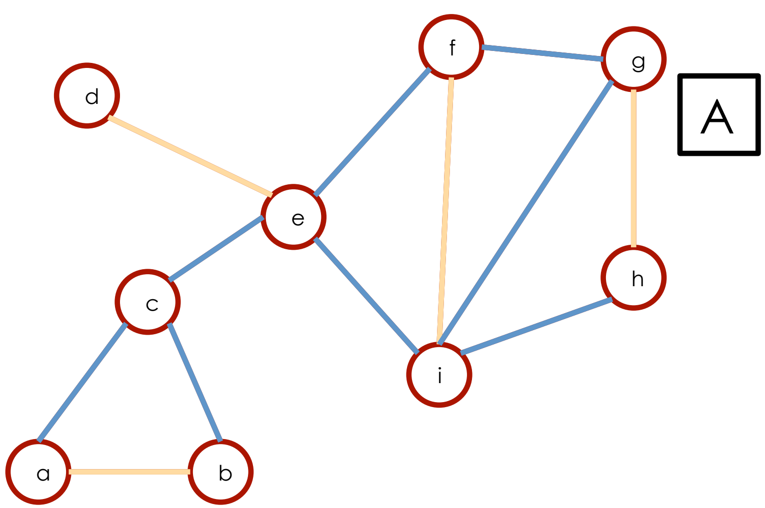
What do we have to make sure

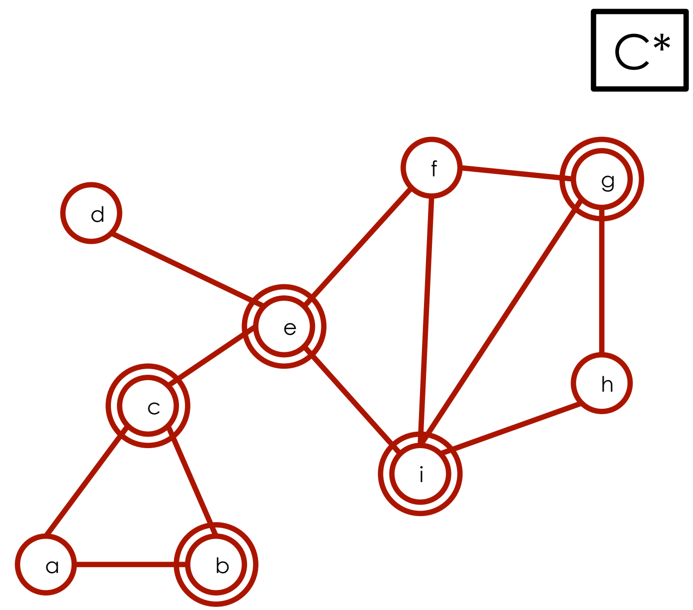
APPROXIMATE-VERTEX-COVER is a polynomial time 2-approximation algorithm = For every instance of Vertex Cover, the result of the algorithm is not more than twice as much as the optimal

2 approx. follow up

Let A be the set of edges picked in a single iteration and C* be the optimal vertex-cover.

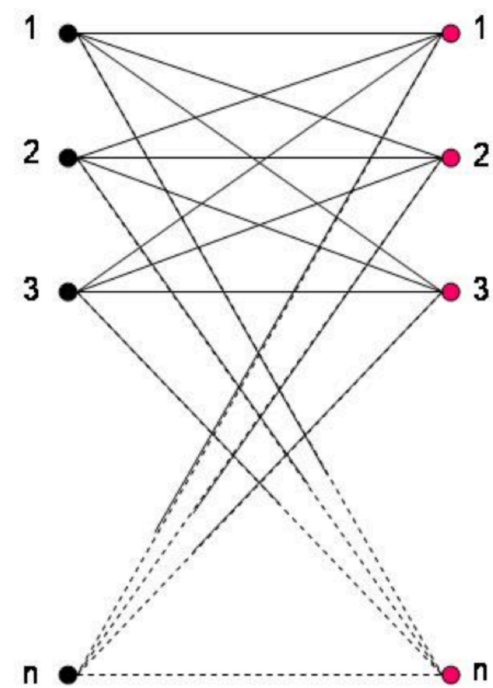
- C* must include at least one end of each edge in A, and no two edges in A are covered by the same vertex in C*, so $|C^*| \geq |A|$.
- Moreover, $|C|=2|A|$, so $|C| \leq 2|C^*|$





An instance that forces 2 approximation

- Consider a complete bipartite graph of n black nodes on one side and n red nodes on the other side, denoted $K_{n,n}$
- In this case the returned algorithm will always return 2n
- The optimal is n



Remember TSP

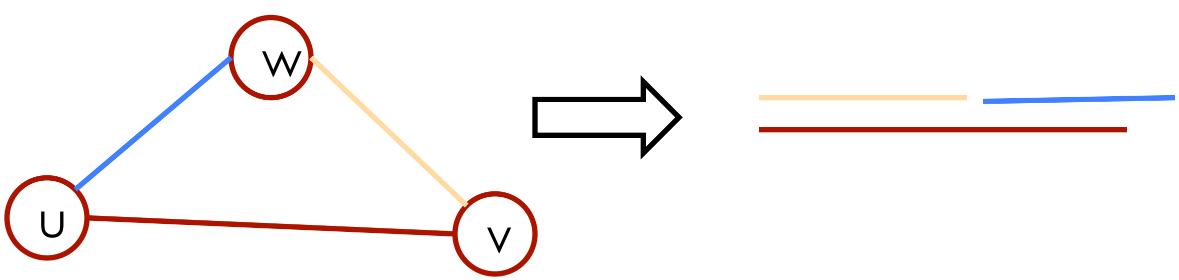
Traveling Salesman Problem (TSP).

- Input: a **complete**, undirected graph $G = (V, E)$, with edge weights (costs) $w : E \rightarrow R+$, $|V| = n$.
- Output: a tour (cycle that visits all n vertices exactly once each, and returning to starting vertex) of minimum cost.

Approximating TSP

- In the plane "*triangle inequality*" holds.
- *Triangle inequality*: cutting out an intermediate stop never increases the cost.
- Formally: for any three vertices u,v,w in G:

$$\text{cost}(u,w) \leq \text{cost}(u,v)+\text{cost}(v,w)$$



We show these results:

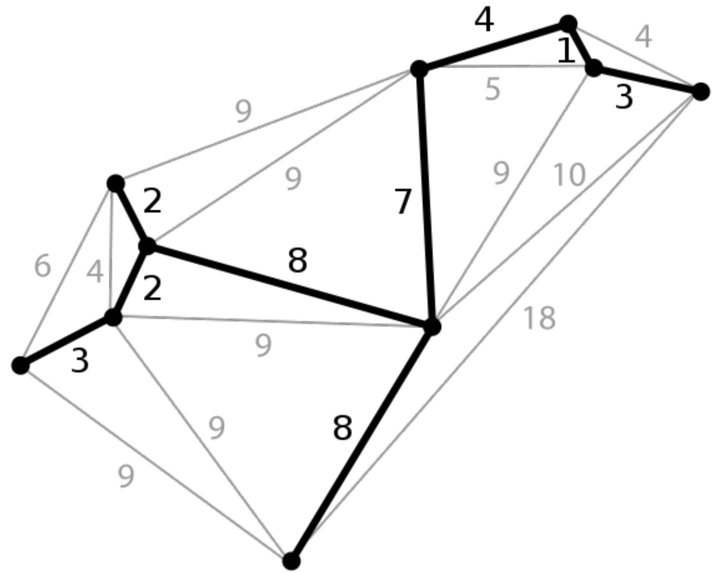
- If the graph satisfies the triangle inequality, then TSP can be approximated efficiently within a factor 2.
- Otherwise, the general problem of approximating TSP without triangle inequality is NP-hard.

Recall: Minimum spanning tree (MST)

Given a weighted connected undirected graph G , **a spanning tree** of G is subgraph of G that is a tree that connects all the vertices

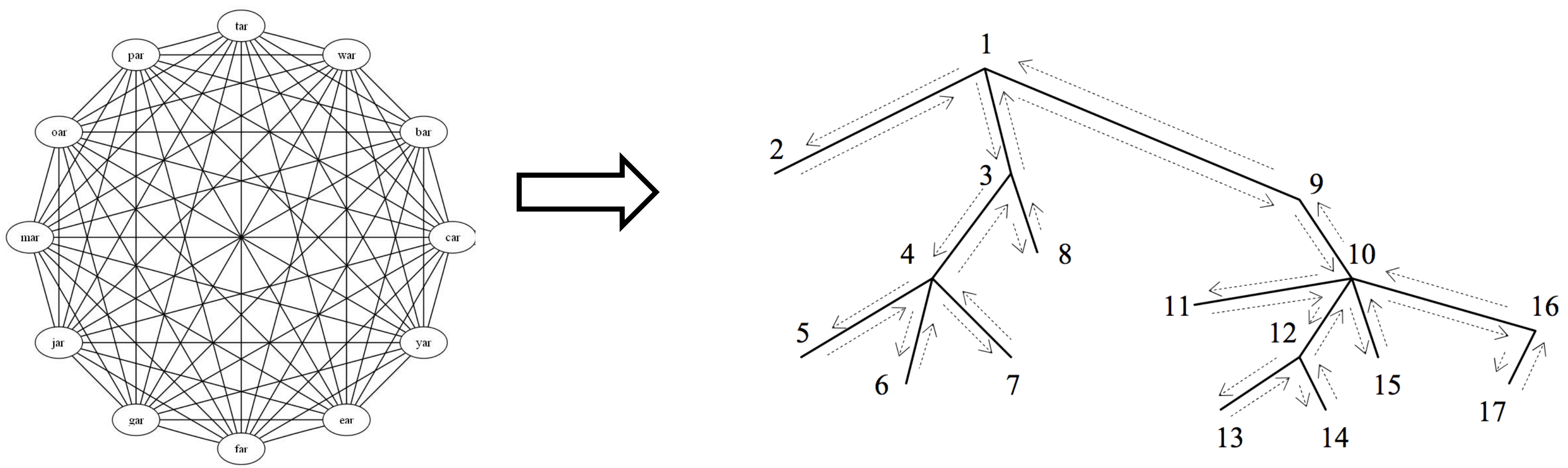
MST is the smallest weighted spanning tree.

Prim finds the MST runs in $O(|V|^2)$



The algorithm

1 Compute a weighted MST of G.
2 Root MST arbitrarily and traverse in pre-order: $v_1, v_2, \ldots, v_n$.
3 Output tour: $v_1 \to v_2 \to \cdots \to v_n \to v_1$.



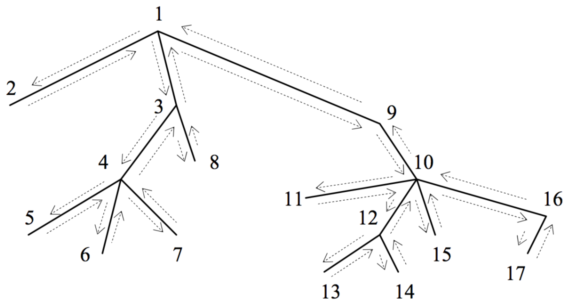
Our goal in mind

- We have a simple algorithm
- Prove that the result is not more than twice the optimum

Approx-TSP-Tour is a 2 approximation

For the problem instance I of TSP let: 1. **A(I)**- the tour length returned by Approx-TSP-Tour 2.**OPT(I)** – the optimal tour length and   3.**MST(I)** -the weight of the MST produced

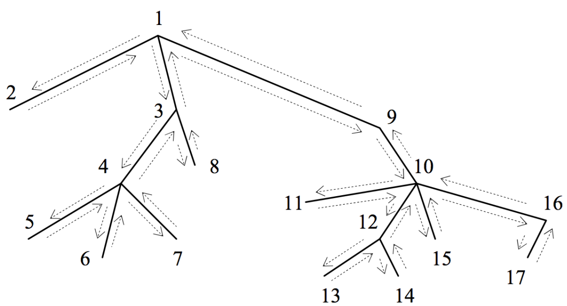
- Claim: For every instance I, A(I) ≤ 2 OPT(I)
- Proof:
  - We first note that MST(I) ≤ OPT(I)
    - Since *An optimal tour minus one edge is a spanning tree, and we have the minimal in our hand (as seen 2 lectures ago)*



Approx-TSP-Tour is a 2 approximation

A(I)- the tour length returned by Approx-TSP-Tour **OPT(I)** – the optimal tour length
MST(I) -the weight of the MST produced

- Proof(part 2) :
  - Let σ be a full walk along the MST in pre-order (that is, we revisit vertices as we backtrack through them).
  - The cost(σ) = 2 × MST(I).



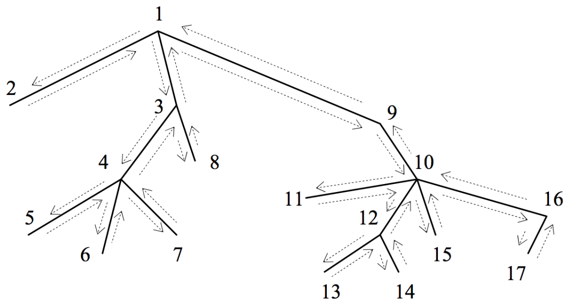
Approx-TSP-Tour is a 2 approximation

A(I)- the tour length returned by Approx-TSP-Tour **OPT(I)** – the optimal tour length
MST(I) -the weight of the MST produced σ - full walk along the MST in pre-order

- Proof(part 3) :
  - A's result is at most the full walk σ, so by the triangle inequality:

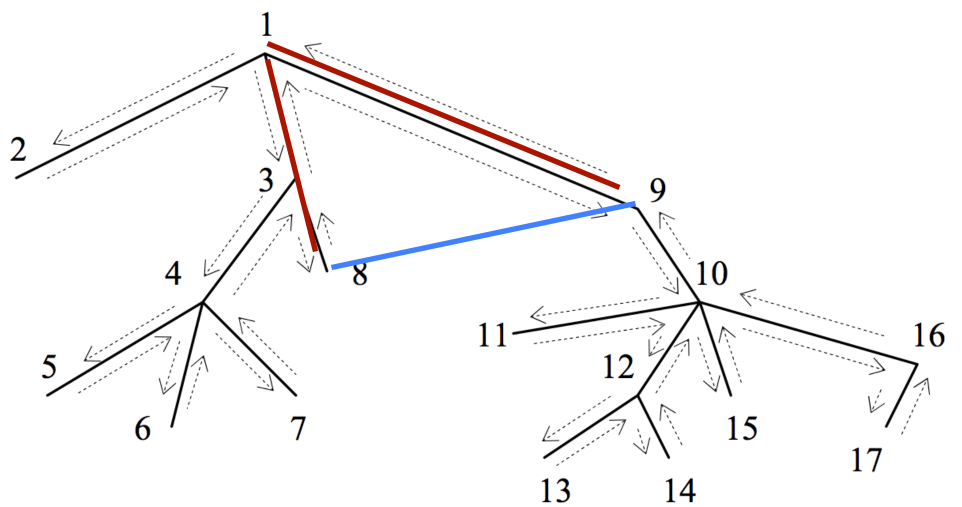
    $$A(I) \le cost(\sigma) \le 2 \times MST(I) \le 2 \times OPT(I)$$

  - $cost(\sigma) = 2 \cdot cost(MST)$.
  - Finally, by triangle inequality, shortcutting previously visited vertices does not increase the cost. Hence we have $cost(\sigma) \le 2 \cdot cost(MST) \le OPT$.



Illustrative example:

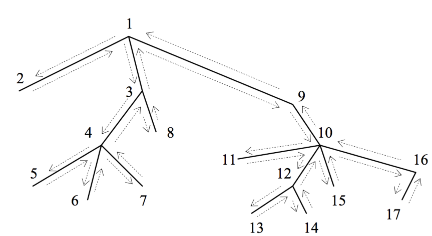
- $A(I) \le cost(\sigma) = 2 \times MST(I)$
- The edge e_{89} is taken in A(I) $e_{89} \le e_{83} + e_{31} + e_{19}$ by triangle inequality



All together now

- Claim : For every instance I, A(I) ≤ 2 × MST(I)
- Proof:
  - MST(I) ≤ OPT(I)
    - Since *An optimal tour minus one edge is a spanning tree*
  - Let σ be a full walk along the MST in pre-order (that is, we revisit vertices as we backtrack through them).
  - The cost(σ) ≤ 2 × MST(I).
  - A's result is a subsequence of the full walk σ, so by the triangle inequality:

$$A(I) \leq cost(\sigma) = 2 \times MST(I) \leq 2 \times OPT(I)$$



Happy news done, sad news arrive

Recall Hamiltonian cycle (HC)

Input: an undirected (not necessarily complete) graph G = (V, E).

Output: YES if G has a Hamiltonian cycle, NO otherwise

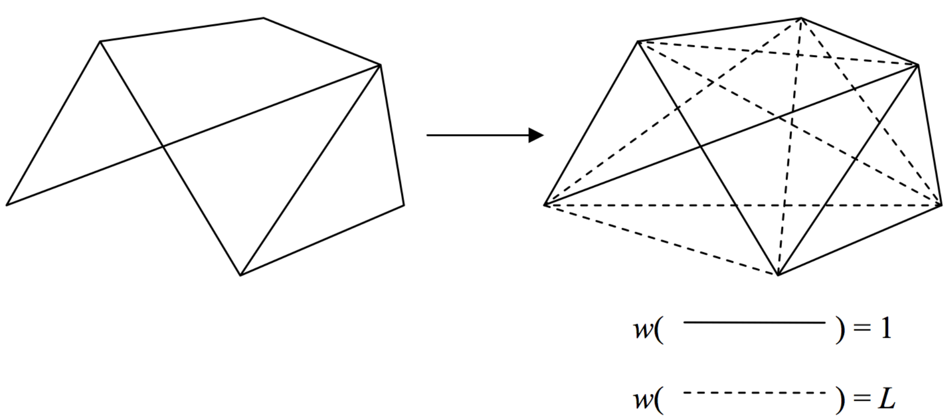
- NP complete problem

General TSP is NP-hard to approximate

- Theorem: For any constant k, it is NP-hard to approximate TSP to a factor of k.
- Plan (Our old friend reduction):
  - Assume A is a k-approximation algorithm for TSP.
  - Use A to solve Hamiltonian cycle in polynomial time
  - P = NP.

The reduction

- HC gets G = (V, E) as input HC
- Modify G to G′ = (V ′ , E′) with weight function w by:
  - All edges of G have weight 1 in G’
  - All other edges in the complete graph G’ get a weight L >k*n, Say L=2k

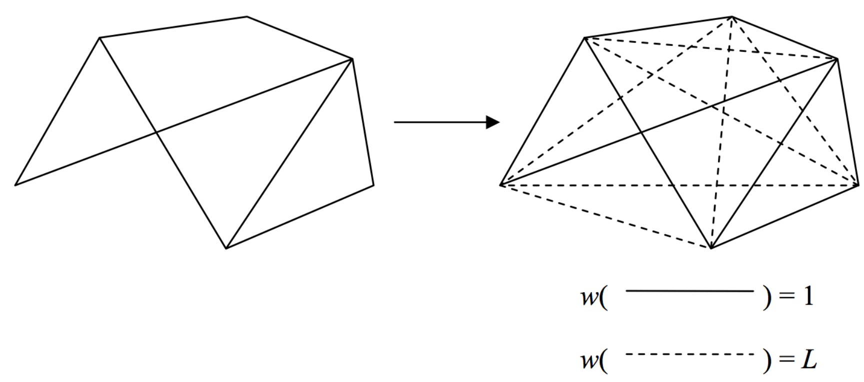


The reduction (cont.)

- Now run the following algorithm:

Algorithm 2: HC-Reduction(G)

Construct G' as described above.
if $A(G')$ returns a 'small' cost tour ($\leq kn$) **then**
  return YES
if $A(G')$ returns a 'large' cost tour ($\geq L$) **then**
  return NO



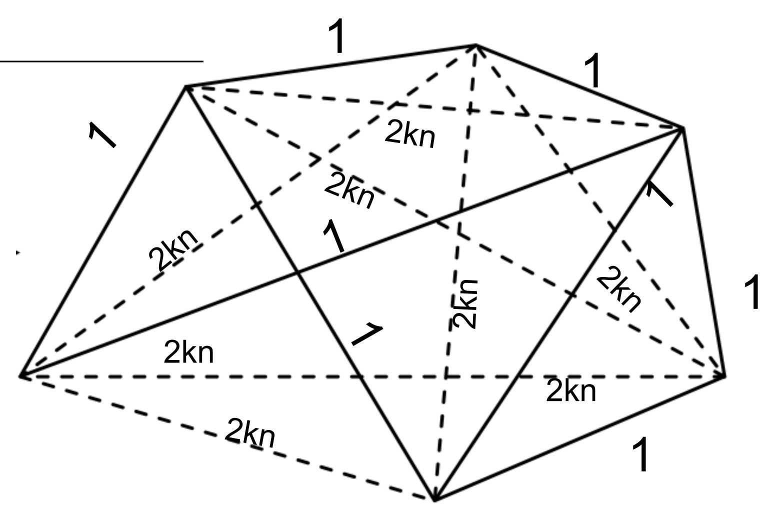
Class ex.

- Why does the reduction works?

Algorithm 2: HC-Reduction(G)

---

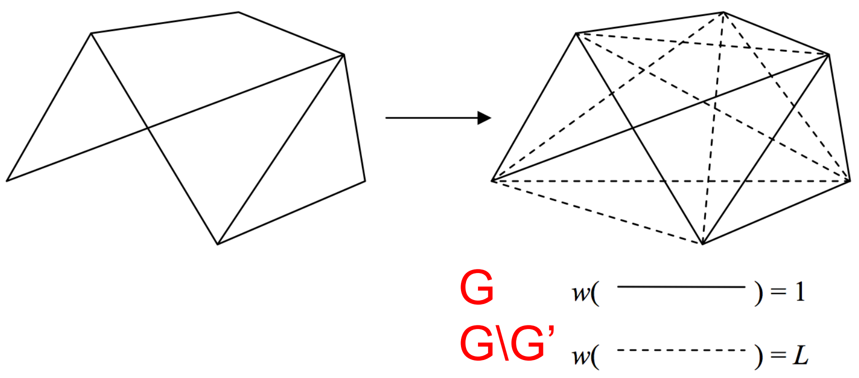
1 Construct G' as described above.
2 **if** $A(G')$ returns a 'small' cost tour ($\leq kn$) **then**
3   **return** YES
4 **if** $A(G')$ returns a 'large' cost tour ($\geq L$) **then**
5   **return** NO



Answer

- The weight of any edge e ∈(G'\G) is twice the weight of all edges in G
- Opt(I) ≤ A(I)×k
- If A(I)<= n×k, no edge in G'\G can participate, and that tour is a euclidian tour in G
  - If A(I)>n×k (L) even Opt(I) has to contain an edge from G'\G
    - That means that G did not

      have a Hamiltonian cycle



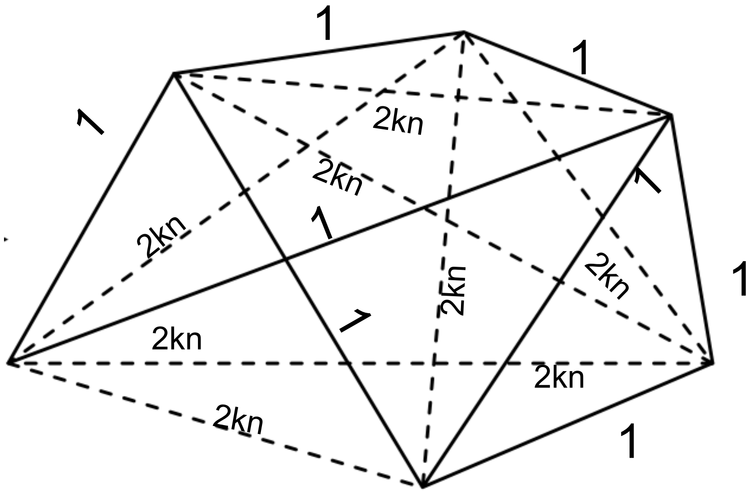
In other words

- The algorithm is k-approximation
  - Therefore: 1 ≤ A(I)/Opt(I) ≤ k
- If the answer was yes A(I) ≤ k*n -> Opt(I) ≤ n
  - Since it is a tour of edges of at least n edges of weight 1 Opt(I)
  - The tour never went through

one of the "big" edges ->

there is a Hamiltonian path

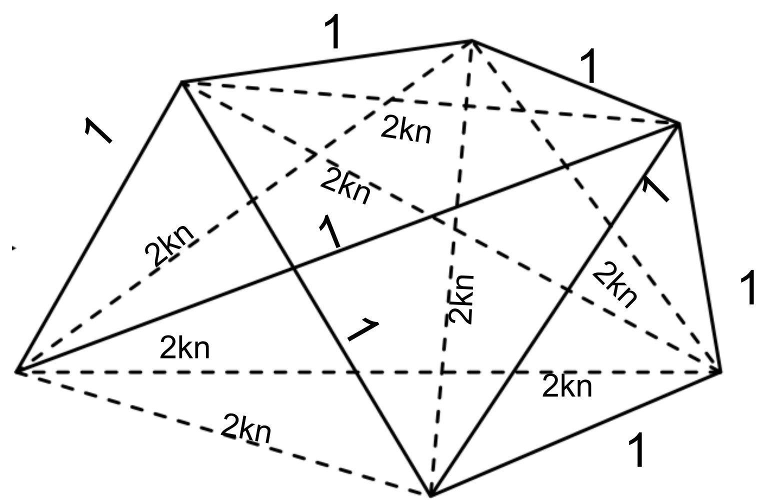
in the original graph



In other words …

- The algorithm is k-approximation
  - Therefore: $1 \leq A(I)/Opt(I) \leq k$
- If the answer is no $A(I) > k*n$ -> $Opt(I) > n$
  - The tour went at least through one “big” edge, which is larger than the sum of all edges in the original graph ->

It was not possible to construct a Hamiltonian path in the original graph



Set Cover

- *Definition*: Given a finite set X and subsets of X, find the minimum number of these subsets whose union is X.
- Many applications (*can you think of any*?)
- Generalises vertex-cover, hence NP-complete (*Why? A vertex can be seen as the set of edges which it covers.*)
- Approx. algorithm (*greedy*): select at each step a set which covers as many still uncovered elements as possible.

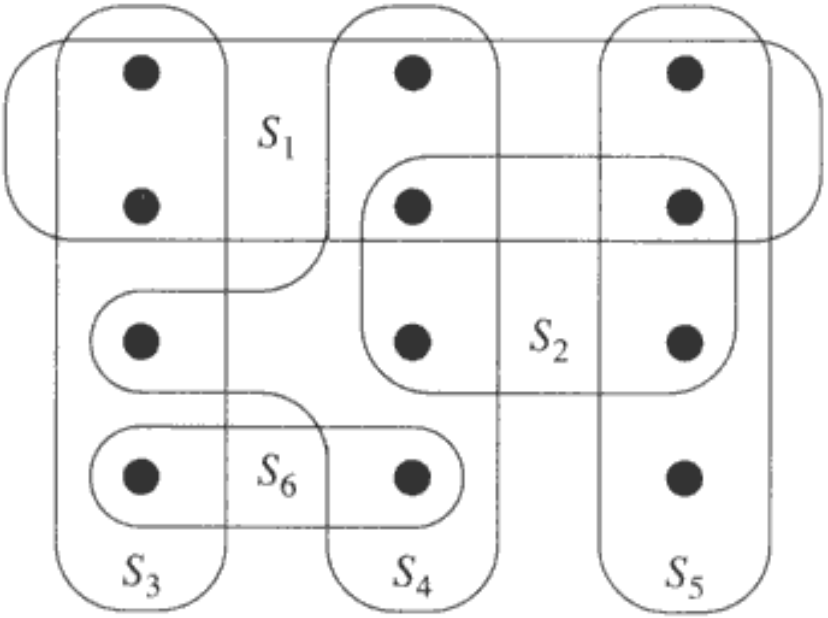


Figure 35.3 An instance $(X, \mathcal{F})$ of the set-covering problem, where X consists of the 12 black points and $\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. A minimum-size set cover is $\mathcal{C} = \{S_3, S_4, S_5\}$. The greedy algorithm produces a cover of size 4 by selecting the sets S_1, S_4, S_5, and S_3 in order.

GREEDY-SET-COVER$(X, \mathcal{F})$

$U \leftarrow X$
$\mathcal{C} \leftarrow \emptyset$
while $U \neq \emptyset$
  do select an $S \in \mathcal{F}$ that maximizes $|S \cap U|$
    $U \leftarrow U - S$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
return $\mathcal{C}$

Analysis of Greedy-Set-Cover

- In next class …