NP Completeness

Andreas Klappenecker

[partially based on slides by Jennifer Welch]

Dealing with NP-Complete Problems

Dealing with NP-Completeness

Suppose the problem you need to solve is NP-complete. What do you do?

You might be able to show that bad running time does not happen for inputs of interest

You might find heuristics to improve running time in many cases (but no guarantees)

You may be able to find a polynomial time algorithm that is guaranteed to give an answer close to optimal

Optimization Problems

- Concentrate on approximation algorithms for optimization problems:
  - every candidate solution has a positive cost
- Minimization problem: goal is to find smallest cost solution
  - Ex: Vertex cover problem, cost is size of VC
- Maximization problem: goal is to find largest cost solution
  - Ex: Clique problem, cost is size of clique

Approximation Algorithms

An approximation algorithm for an optimization problem

- runs in polynomial time and
- always returns a candidate solution

Ratio Bound

Ratio bound: Bound the ratio of the cost of the solution returned by the approximation algorithm and the cost of an optimal solution

- minimization problem:

  cost of approx solution / cost of optimal solution

- maximization problem:

  cost of optimal solution / cost of approx solution

So ratio is always at least 1, goal is to get it as close to 1 as we can

Approximation Algorithms

A poly-time algorithm A is called a δ-approximation algorithm for a minimization problem P if and only if for every problem instance I of P with an optimal solution value OPT(I), it delivers a solution of value A(I) satisfying

$$A(I) \leq \delta OPT(I).$$

Approximation Algorithms

A poly-time algorithm A is called a δ-approximation algorithm for a maximization problem P if and only if for every problem instance I of P with an optimal solution value OPT(I), it delivers a solution of value A(I) satisfying

$$\delta A(I) \geq OPT(I).$$

Vertex Cover

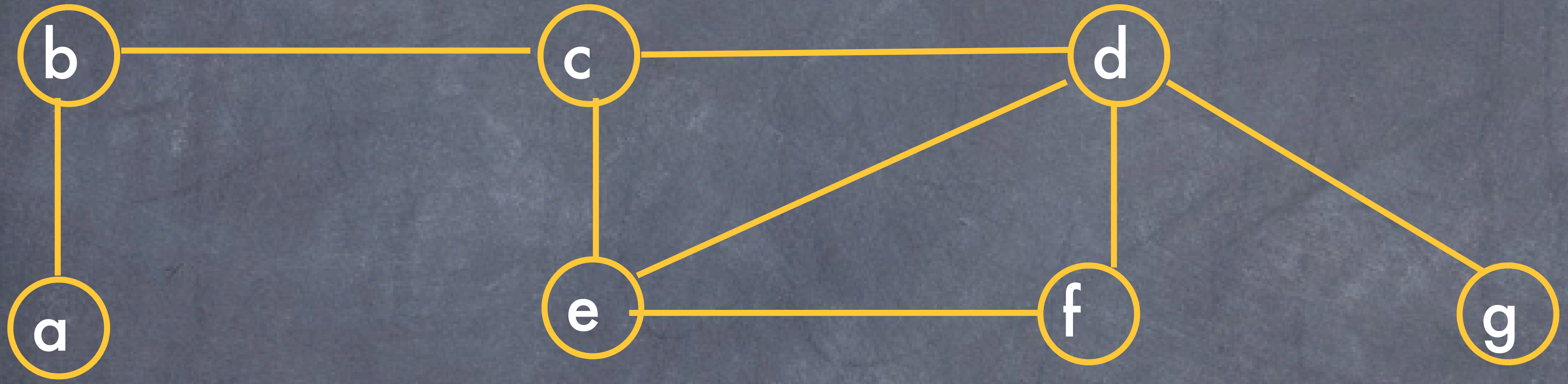
Approximation Algorithm for Minimum Vertex Cover Problem

```
input:  G = (V,E)
C := ∅
E' := E
while E' ≠ ∅ do
    pick any (u,v) in E'
    C := C U {u,v}
    remove from E' every edge incident on u or v
endwhile
return C
```

Min VC Approx Algorithm

- Time is O(E), which is polynomial.
- How good an approximation does it provide?
- Let's look at an example.

Min VC Approx Alg Example



choose (b,c): remove (b,c), (b,a), (c,e), (c,d)
choose (e,f): remove (e,f), (e,d), (d,f)
Answer: {b,c,e,f,d,g}
Optimal answer: {b,d,e}
Algorithm's ratio bound is 6/3 = 2.

Ratio Bound of Min VC Alg

Theorem: Min VC approximation algorithm has ratio bound of 2.

Proof: Let A be the total set of edges chosen to be removed.

- Size of VC returned is 2*|A| since no two edges in A share an endpoint.
- Size of A is at most size of a min VC since min VC must contain at least one node for each edge in A.
- Thus cost of approx solution is at most twice cost of optimal solution

More on Min VC Approx Alg

- Why not run the approx alg and then divide by 2 to get the optimal cost?
- Because answer is not always exactly twice the optimal, just never more than twice the optimal.
- For instance, a different choice of edges to remove gives a different answer:
    - Choosing (d,e) and then (b,c) produces answer {b,c,d,e} with cost 4 as opposed to optimal cost 3

Metric TSP

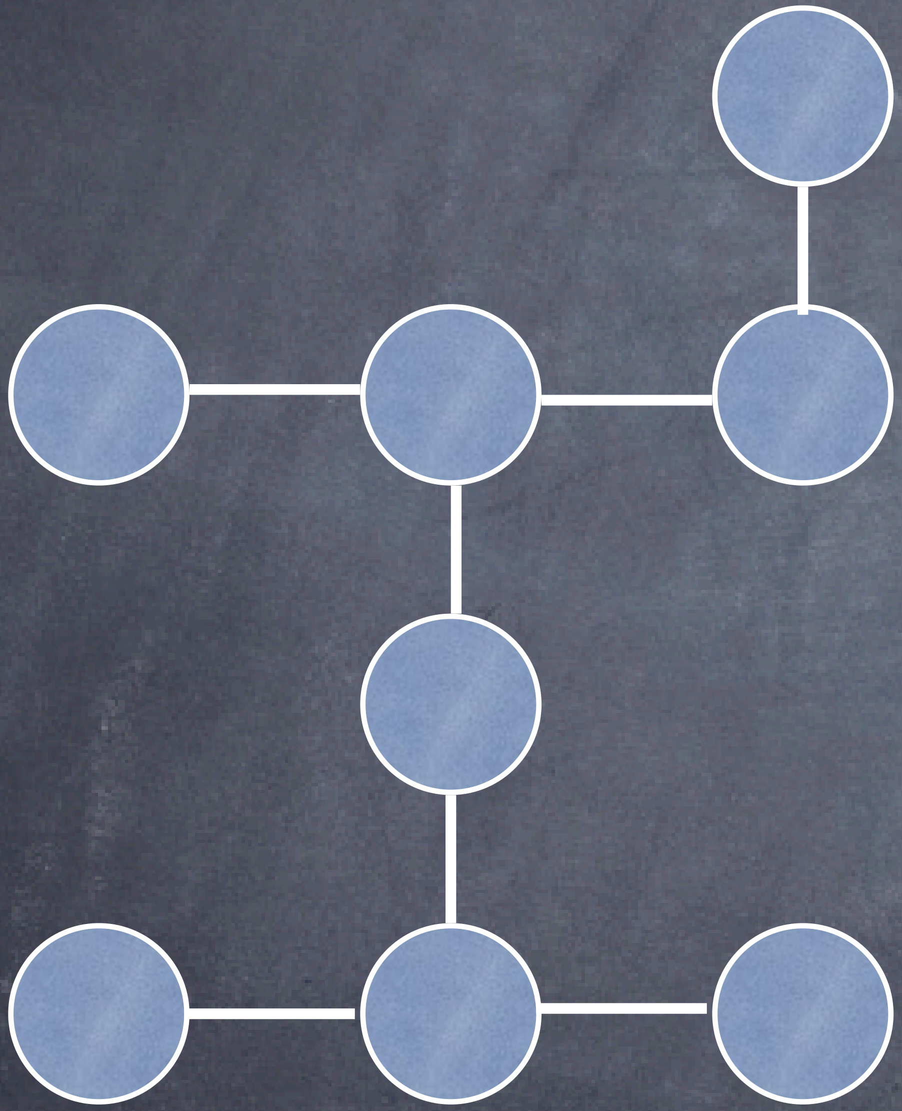
Triangle Inequality

- Assume TSP inputs with the triangle inequality:
  - distances satisfy property that for all cities a, b, and c, $\text{dist}(a,c) \leq \text{dist}(a,b) + \text{dist}(b,c)$
  - i.e., shortest path between 2 cities is the direct route
  - this metric TSP is still NP-hard
- Depending on what you are modeling with the distances, an application might or might satisfy this condition.

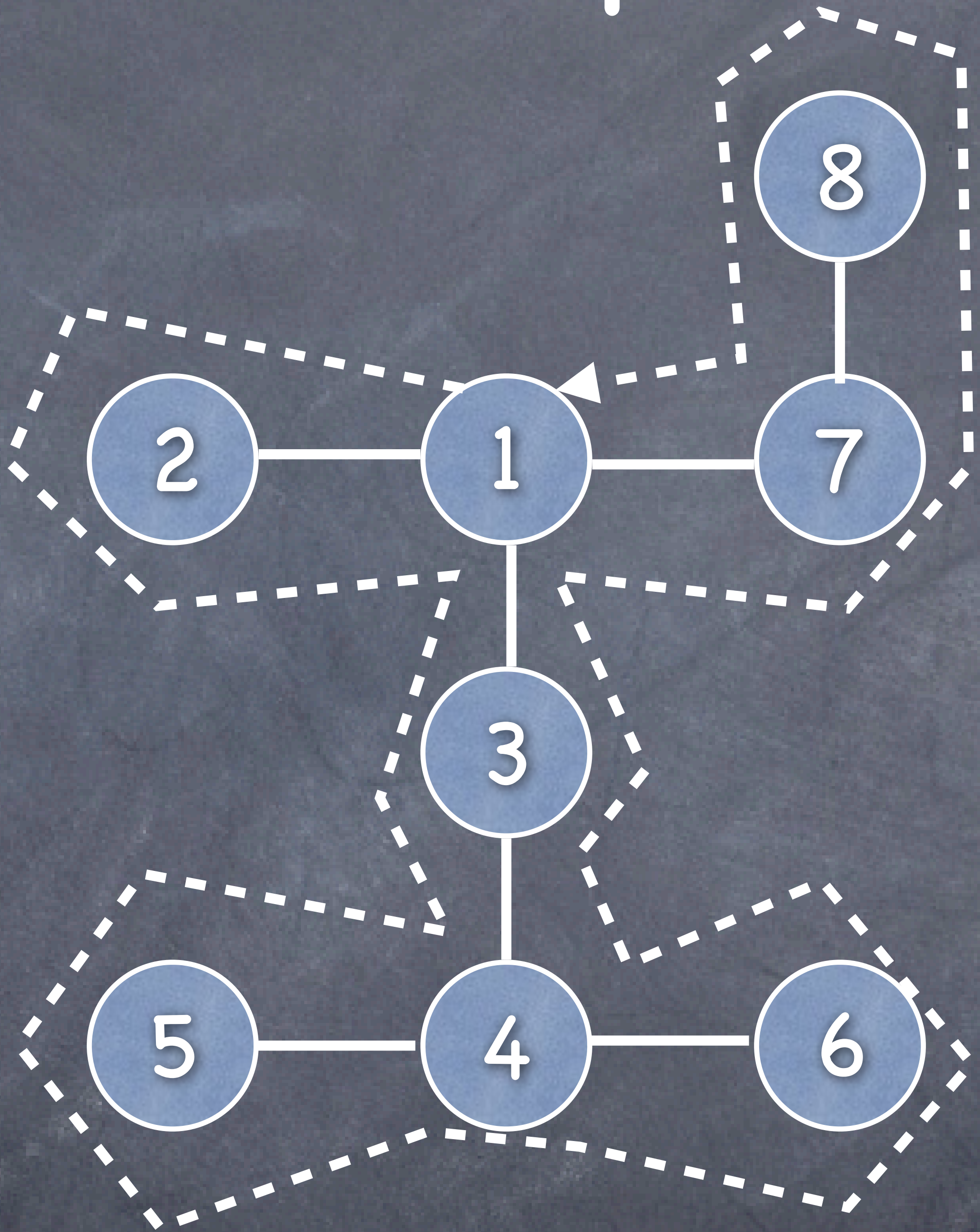
TSP Approximation Algorithm

- input: set of cities and distances b/w them that satisfy the triangle inequality
- create complete graph G = (V,E), where V is set of cities and weight on edge (a,b) is dist(a,b)
- compute MST of G
- Go twice around the MST to get a tour (that will have duplicates)
- Remove duplicates to avoid visiting a city more than once

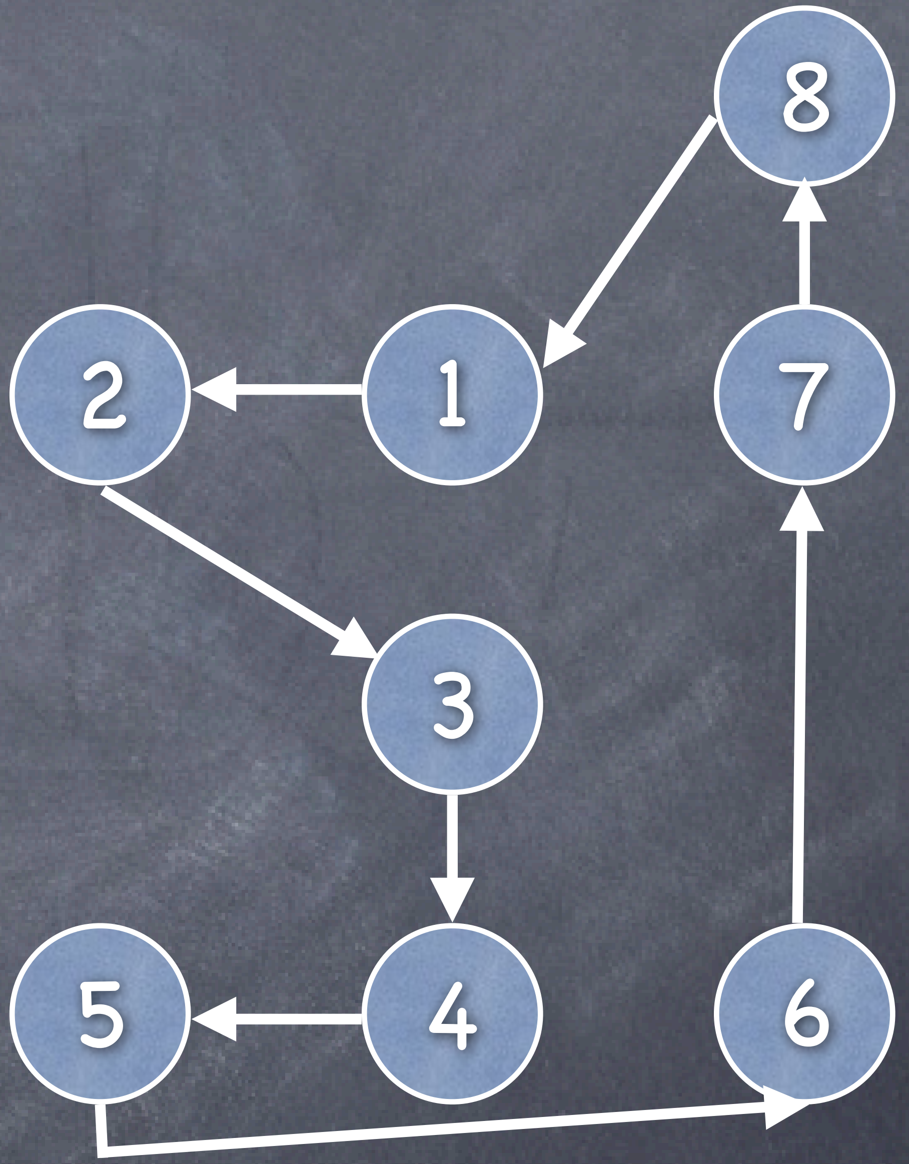
Example



Form MST



DFS Traversal, Double Edges, Eulerian Tour of 2-Multigraph



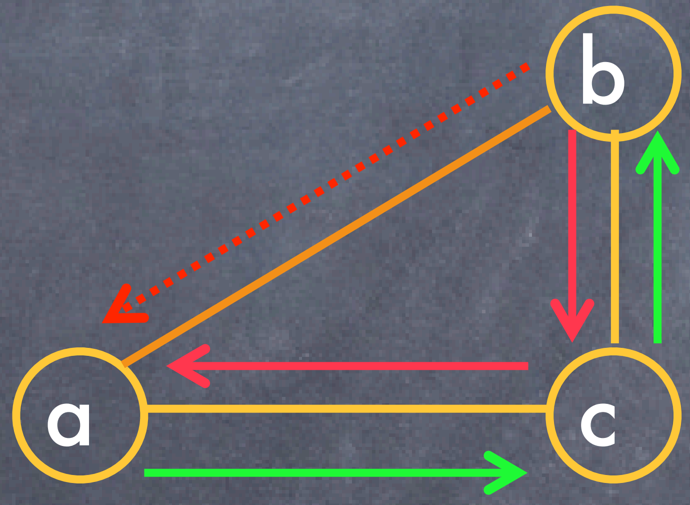
Skip cities that were already visited

Analysis of TSP Approx Alg

- Running time is polynomial (creating complete graph takes $O(V^2)$ time, Kruskal's MST algorithm takes time $O(E \log E) = O(V^2 \log V)$.
- How good is the quality of the solution?

Analysis of TSP Approx Alg

- cost of approx solution ≤ 2*weight of MST, by triangle inequality
- Why?



when tour created by going around the MST is adjusted to remove duplicate nodes, the two red edges are replaced with the dashed diagonal edge

Analysis of TSP Approx Alg

- weight of MST < length of min tour
- Why?
- Min tour minus one edge is a spanning tree T, whose weight must be at least the weight of MST.
- And weight of min tour is greater than weight of T.

Analysis of TSP Approx Alg

- Putting the pieces together:
- cost of approx solution ≤ 2*weight of MST

  ≤ 2*cost of min tour
- So approx ratio is at most 2.

(non-metric) TSP

TSP Without Triangle Inequality

Theorem: If $P \neq NP$, then no polynomial time approximation algorithm for TSP (w/o triangle inequality) can have a constant ratio bound.

Proof: We will show that if there is such an approximation algorithm, then we could solve a known NP-complete problem (Hamiltonian cycle) in polynomial time, so P would equal NP.

HC Exact Algorithm using TSP Approximation Algorithm

Input: $G = (V,E)$

1. convert G to this TSP input:
   - one city for each node in V
   - distance between cities u and v is 1 if (u,v) is in E
   - distance between cities u and v is r*|V| if (u,v) is not in E, where r is the ratio bound of the TSP approx alg
   - Note: This TSP input does not satisfy the triangle inequality

HC (Exact) Algorithm Using TSP Approximation Algorithm

2. run TSP approx alg on the input just created

3. if cost of approx solution returned in step 2 is ≤ r*|V| then return YES else return NO

Running time is polynomial.

Correctness of HC Algorithm

- If G has a HC, then optimal tour in TSP input constructed corresponds to that cycle and has weight |V|.
- Approx algorithm returns answer with cost at most r*|V|.
- So if G has HC, then algorithm returns YES.

Correctness of HC Algorithm

- If G has no HC, then optimal tour for TSP input constructed must use at least one edge not in G, which has weight r*|V|.
- So weight of optimal tour is > r*|V|, and answer returned by approx alg has weight > r*|V|.
- So if G has not HC, then algorithm returns NO.

Set Cover

Set Cover

Given a universe U of n elements, a collection S = $\{S_1, S_2, \ldots, S_k\}$ of subsets of U, and a cost function c: S->Q^+, find a minimum cost subcollection of S that covers all the elements of U.

Example

We might want to select a committee consisting of people who have combined all skills.

Cost-Effectiveness

We are going to pick a set according to its cost effectiveness.

Let C be the set of elements that are already covered.

The cost effectiveness of a set S is the average cost at which it covers new elements: $c(S)/|S-C|$.

Greedy Set Cover

- $C := \emptyset$
- while $C \neq U$ do
  - Find most cost effective set in current iteration, say S, and pick it.
  - For each e in S, set price(e)= c(S)/|S-C|.
  - $C := C \cup S$
- Output C

Theorem

Greedy Set Cover is an H_m-approximation algorithm, where $m = \max\{ |S_i| : 1 <= i <= k\}$.

Lemma

For all sets T in S, we have

$\Sigma_{e \text{ in } T}$ price(e) <= c(T) H_x with x=|T|

Proof: Let e in $T \cap (S_i \setminus \bigcup_{j<i} S_j)$ and

$V_i = T \setminus \bigcup_{j<i} S_j$ be the remaining part of T before being covered by the greedy cover.

Lemma (2)

Then the greedy property implies that

price(e) <= $c(T)/|V_i|$

Let $e_1,\dots,e_m$ be the elements of T in the order chosen by the greedy algorithm.

It follows that

price(e_k) <= $w(T)/(|T|-k+1)$.

Summing over all k yields the claim.

Proof of the Theorem

- Let A be the optimal set cover and B the set cover returned by the greedy algorithm.
- Σ price(e) <= $\Sigma_{S \text{ in } A}$ $\Sigma_{e \text{ in } S}$ price(e)
  By the lemma, this is bounded by
- $\Sigma_{T \text{ in } A}$ $c(T)H_{|T|}$
- The latter sum is bounded by $\Sigma_{T \text{ in } A}$ c(T) times the Harmonic number of the cardinality of the largest set in S.

Example

- Let $U = \{e_1,\ldots,e_n\}$
- $S = \{ \{e_1\},\ldots,\{e_n\}, \{e_1,\ldots,e_n \} \}$
- $c(\{e_i\}) = 1/i$
- $c(\{e_1,\ldots,e_n \})=1+\varepsilon$
- The greedy algorithm computes a cover of cost H_n and the optimal cover is $1+\varepsilon$