MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 39

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Mandal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55

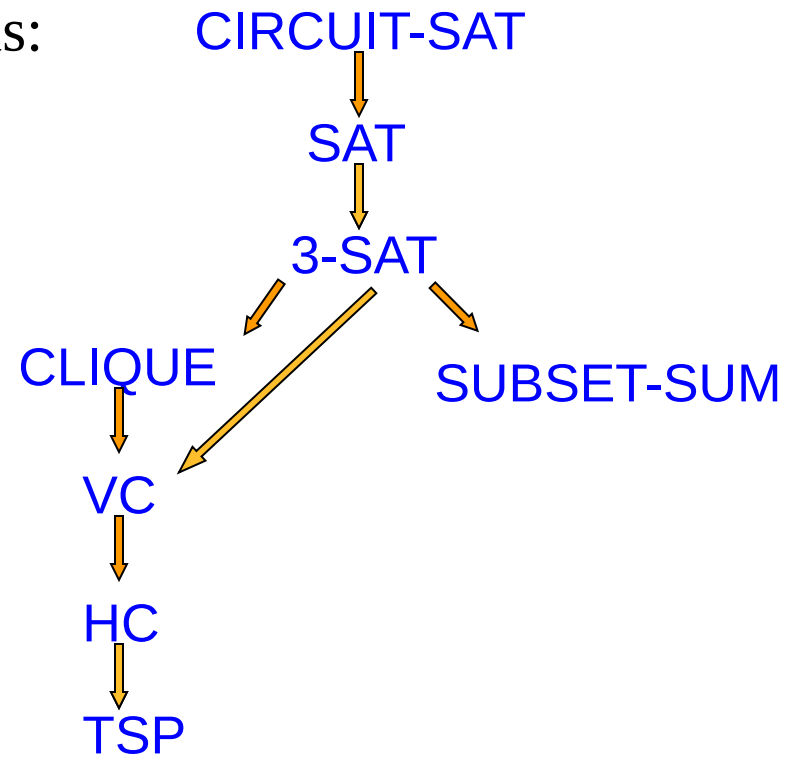
Class Room : 2101

Some NP-Complete Problems

- SAT
- 3-SAT
- VC
- TSP
- CLIQUE (does G contain a completely connected subgraph of size at least K?)
- HC (does G have a Hamiltonian cycle?)
- SUBSET-SUM (given a set S of natural numbers and integer t, is there a subset of S that sum to t?)

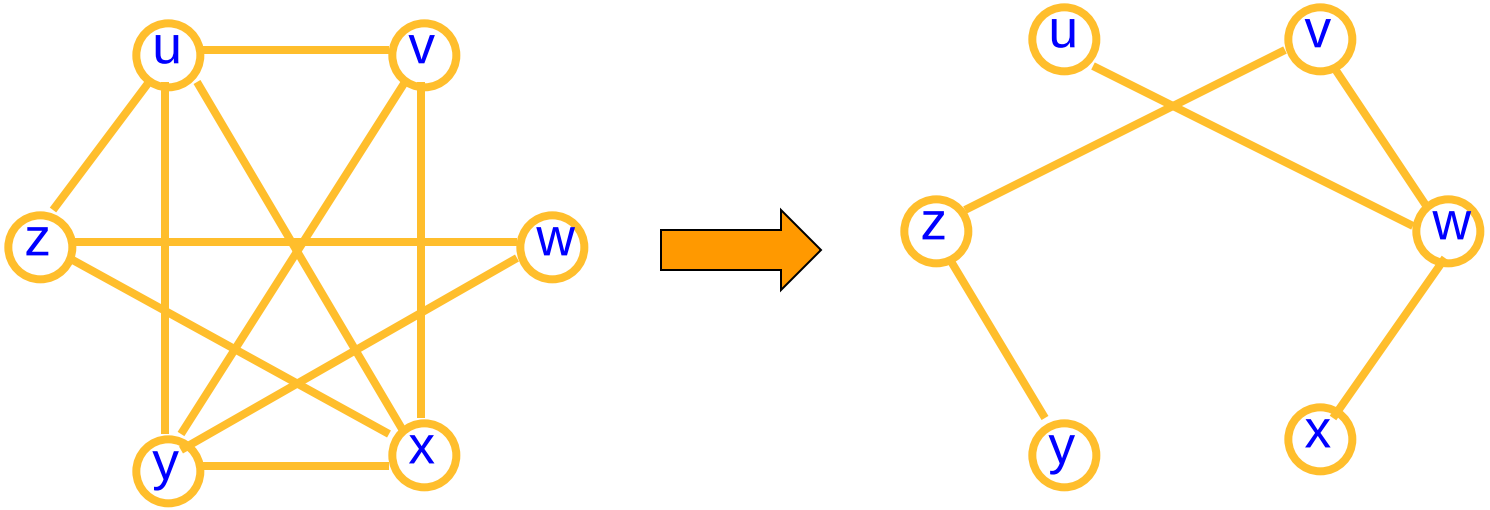
Relationship Between Some NP-Complete Problems

- Textbook shows NP-completeness using this tree of reductions:



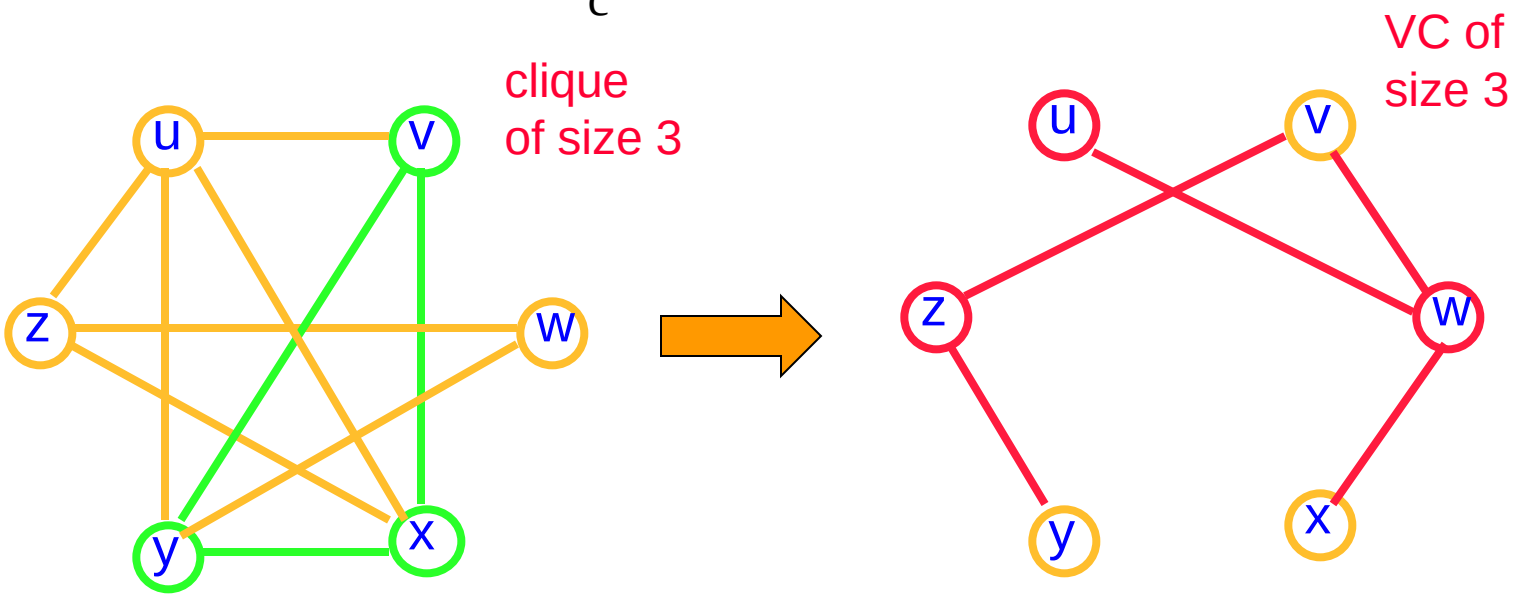
CLIQUE vs. VC

- The complement of graph G = (V,E) is the graph $G_c = (V,E_c)$, where E_c consists of all the edges that are missing in G.



CLIQUE vs. VC

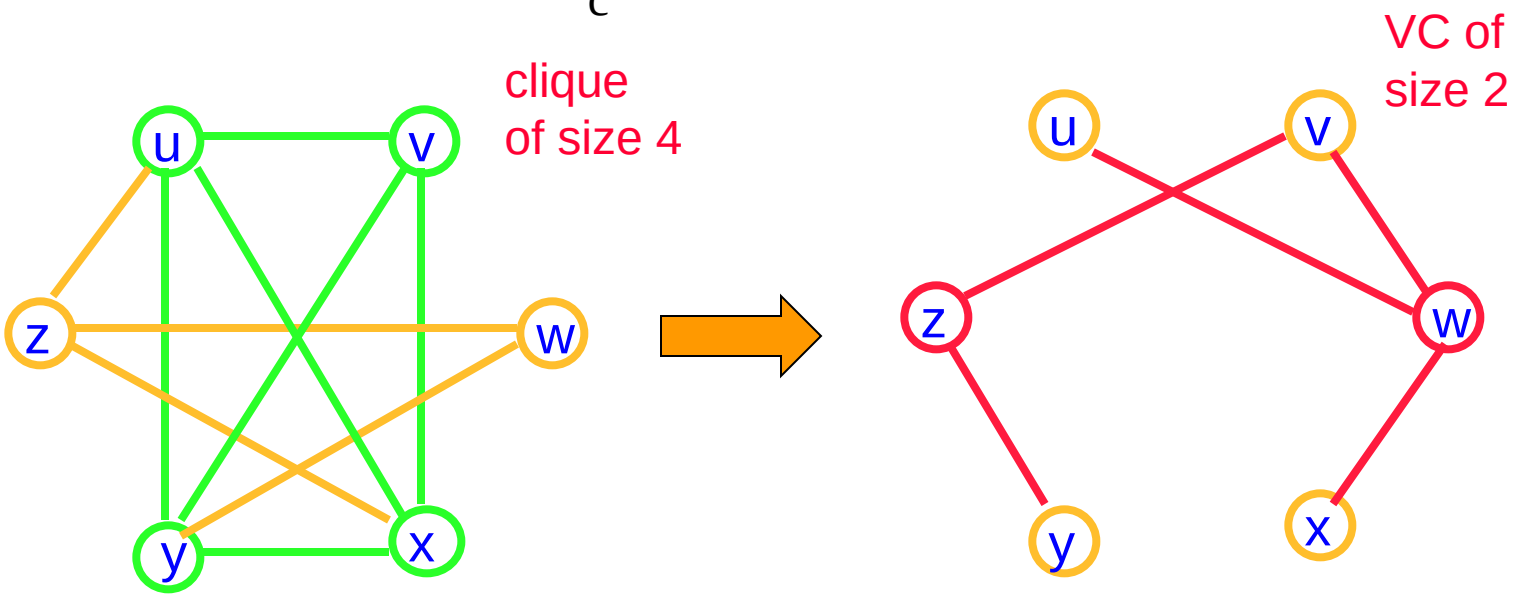
- Theorem: V' is a clique of G if and only if V – V' is a vertex cover of G_c.



the nodes in V' would only "cover" missing edges and thus are not needed in G_c

CLIQUE vs. VC

- Theorem: V' is a clique of G if and only if V – V' is a vertex cover of G_c.



the nodes in V' would only "cover" missing edges and thus are not needed in G_c

VC and CLIQUE

- Can use previous observation to show that VC $\leq_p$ CLIQUE and also to show that CLIQUE $\leq_p$ VC.

Approximation Algorithms for NP-Complete Problems

- Additional source: *Computers and Intractability, A Guide to the Theory of Intractability*, M. Garey and D. Johnson, W. H. Freeman and Co., 1979

Dealing with NP-Completeness

- Suppose the problem you need to solve is NP-complete. What do you do next ?
- hope/show bad running time does not happen for inputs of interest
- find heuristics to improve running time in many cases (but no guarantees)
- 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

Measuring How Good an Approximation Algorithm Is

- 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

Alternative Measurements

- Relative error:

  |cost of approx soln – cost of optimal soln|/cost of optimal soln

- Difference:

  |cost of approx soln – cost of optimal soln|

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*

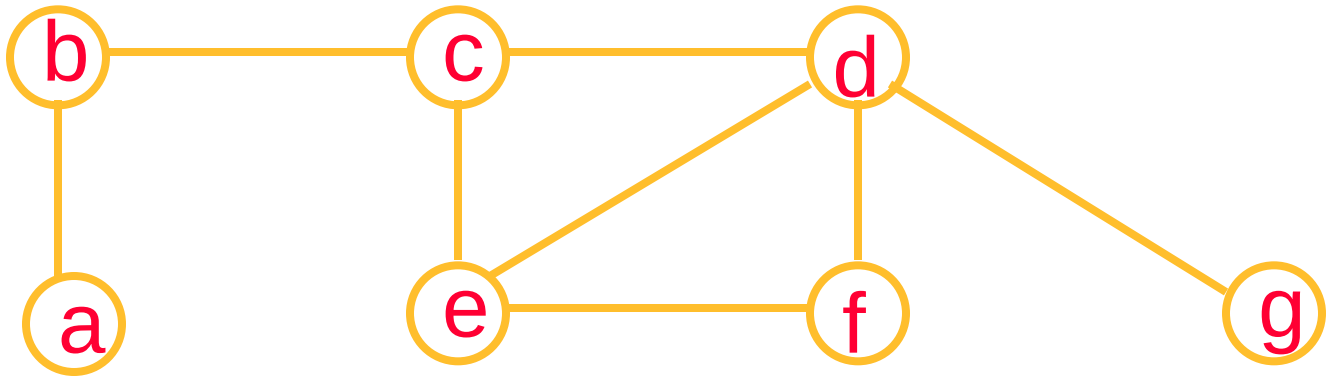
endwhile

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:

- Every chosen edge e has both ends in C
- But e must be covered by an optimal cover; hence, one end of e must be in OPT
- Thus, there is at most twice as many vertices in C as in OPT.
- Thus cost of approx solution is at most twice cost of optimal solution.
- That is, C is a 2-approx. of OPT

More on Min VC Approx Alg

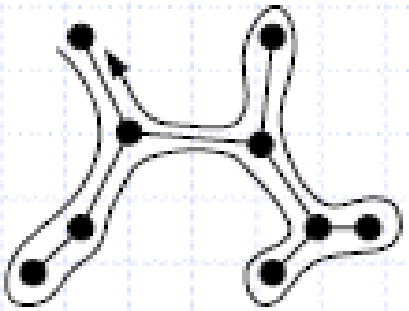
- Why not run the approx alg and then divide by 2 to get the optimal cost ?
- Because answer is not always exactly twice the optimal, just never more than twice the optimal.
- For instance, a different choice of edges to remove gives a different answer:
  - Choosing (d,e) and then (b,c) produces answer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

Triangle Inequality

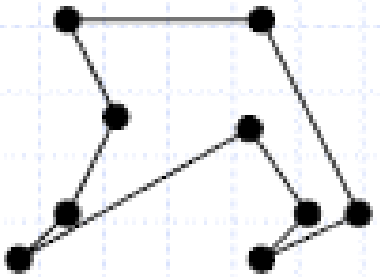
- Assume TSP inputs with the triangle inequality:
  - distances satisfy property that for all cities a, b, and c, dist(a,c) ≤ dist(a,b) + dist(b,c)
  - i.e., shortest path between 2 cities is direct route
- Depending on what you are modeling with the distances, an application might or might satisfy this 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



Euler tour P of MST M



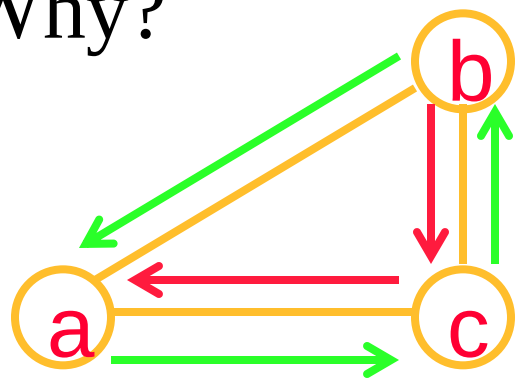
Output tour T

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
green 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 $\leq$ 2*weight of MST

  $\leq$ 2*cost of min tour
- So approx ratio is at most 2.

wish you

all the very best. . .