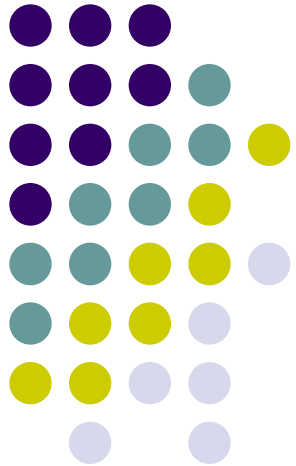
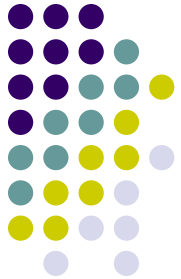
Approximation Algorithms

Analysis of Algorithms

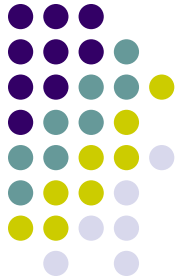


Motivation: Getting around NP-completeness



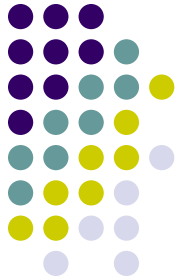
- Exponential time may be acceptable for small inputs [Brute Force]
- Isolate special cases that can run in polynomial time [Divide-and-Conquer]
- Near-optimal solutions may be acceptable [Approximation Algorithms]

Performance Ratios



- C* is the cost of the optimal solution
- C is the cost of the solution produced by an approximation algorithm
- An algorithm has an approximation ratio of ρ (n) if for an input of size n, C is within a factor of C*

Performance Ratios [cont.]



- Maximization problem
  - $0 < C <= C^*$
  - C^* / C factor by which the cost of the optimal solution is larger than the cost of the approximate solution
- Minimization problem
  - $0 < C^* <= C$
  - C / C^* factor by which the cost of the approximate solution is larger than the cost of the optimal solution

Definitions

- *Approximation Algorithm*
  - produces "near-optimal" solution
- Algorithm has *approximation ratio* $\rho(n)$ if:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n)$$

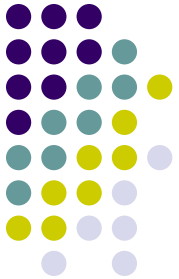
C = cost of algorithm's solution

C^* = cost of optimal solution

n = number of inputs = size of instance

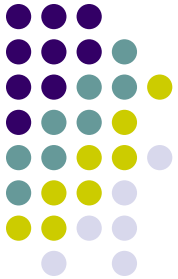
- *Approximation Scheme*
  - $(1+\varepsilon)$-approximation algorithm for fixed ε
    - $\varepsilon > 0$ is an input
  - *Polynomial-Time Approximation Scheme*
    - time is polynomial in n

How do approximations algorithms work?



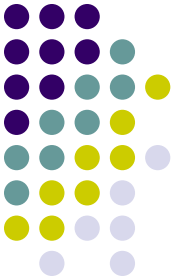
- Exploit the nature of the problem
- Use greedy techniques
- Use linear programming
- Use dynamic programming
- Use random assignments

Overview



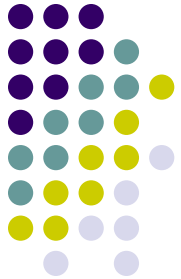
- VERTEX-COVER
  - Polynomial-time 2-approximation algorithm
- TSP
  - TSP with triangle inequality
    - Polynomial-time 2-approximation algorithm
  - TSP without triangle inequality
    - Negative result on polynomial-time $\rho(n)$-approximation algorithm
- SET-COVER
    - polynomial-time $\rho(n)$-approximation algorithm
      - $\rho(n)$ is a logarithmic function of set size
- SUBSET-SUM
  - Fully polynomial-time approximation scheme
- MAX-3-CNF Satisfiability
  - Randomized $\rho(n)$-approximation algorithm

The vertex-cover problem



- Let G=(V,E) be an undirected graph
- A vertex cover
  - subset V' of V
  - such that if (u,v) is an edge of G, then either u belongs in V' or v belongs in V' (or both)
- The size of a vertex cover is the number of vertices in it

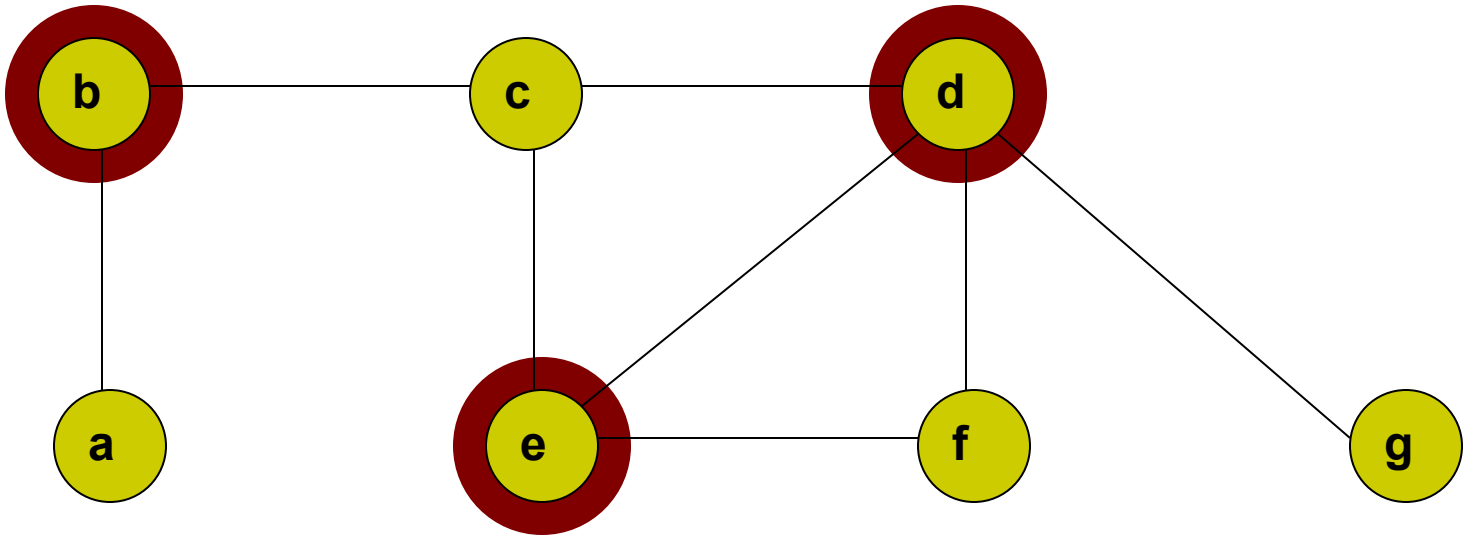
The vertex-cover problem [cont.]



- The vertex-cover problem is to find a vertex cover of minimum size
- Such a vertex-cover is called an optimal vertex cover

- **This problem is NP-complete**

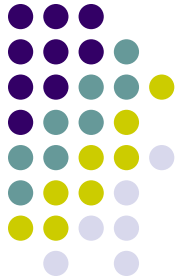
Consider the graph:



By inspection, optimal vertex cover is {b,d,e}

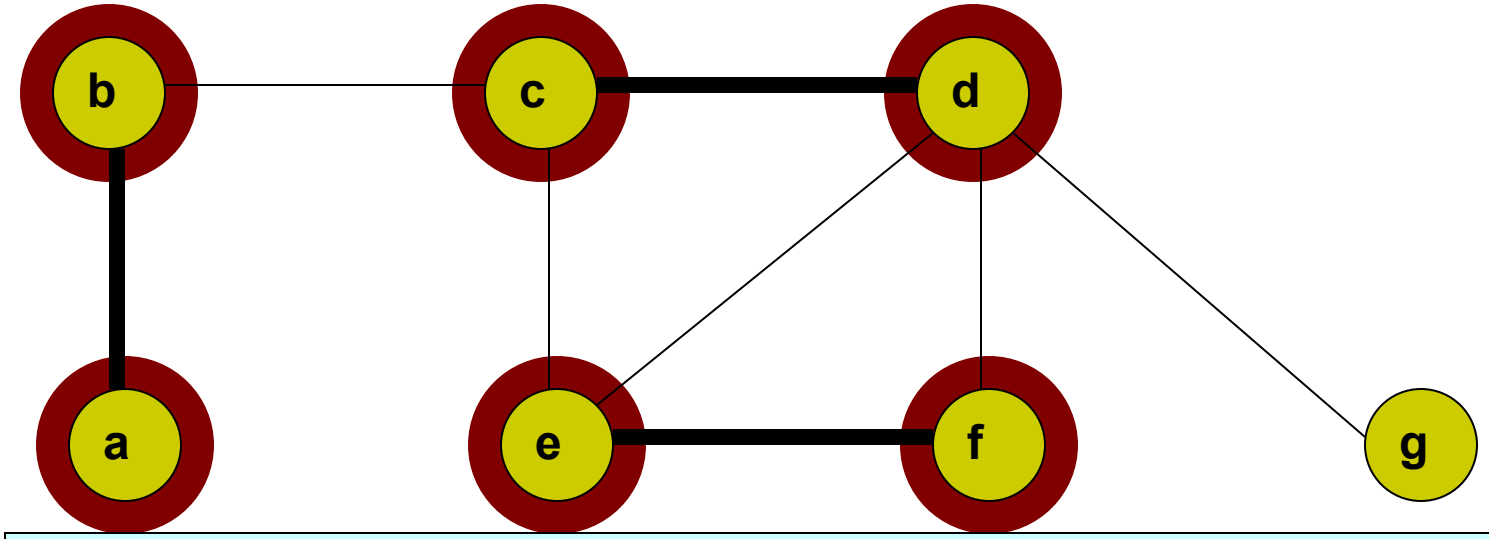
C* = size of optimal solution = 3

Approx-Vertex-Cover (G)



1. C □ NULL //C contains results of vertex cover
2. E' □ E[G] //Initially all edges are stored here
3. while E' != NULL //all edges that are not considered yet
4. do let (u,v) be an edge of E' //select an edge from E'
5. C □ C U {u,v} //add vertices to C of the selected edge
6. remove from E' all edges incident on u or v
   //delete all edges related to the vertices "u" and "v"
7. return C

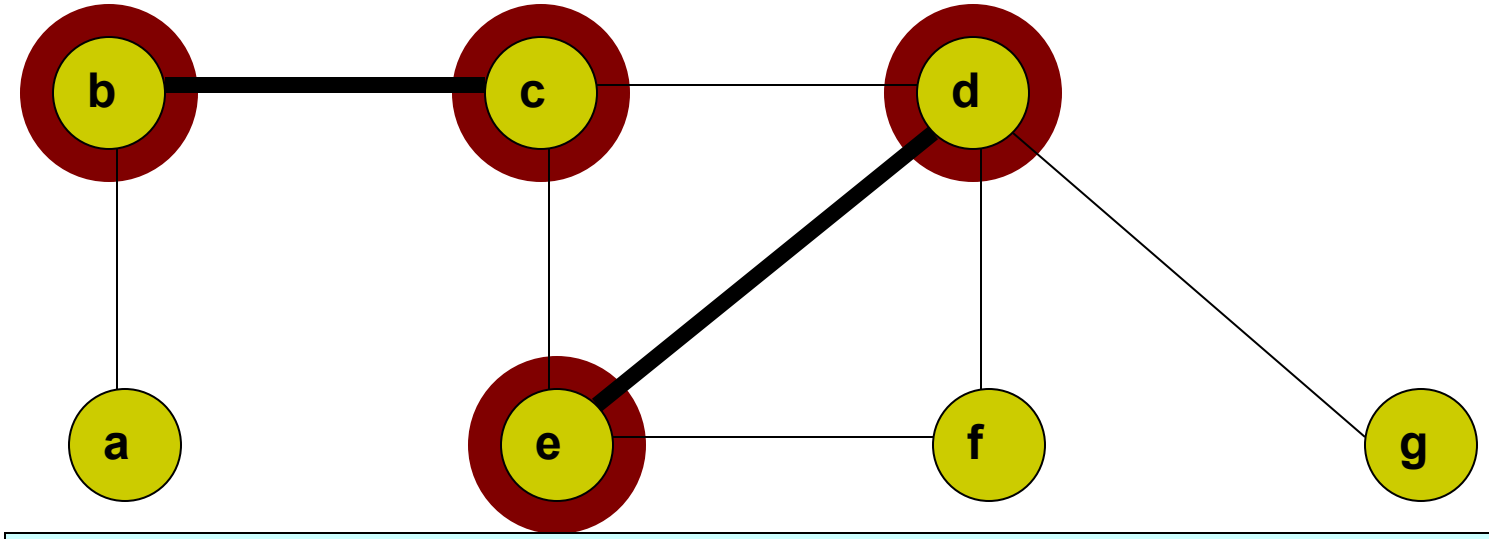
Back to our graph:



C : a b c d e f Approximate vertex cover. C=6

E' : (a,b) (b,c) (c,d) (c,e) (d,e) (d,f) (b,d) (e,f)

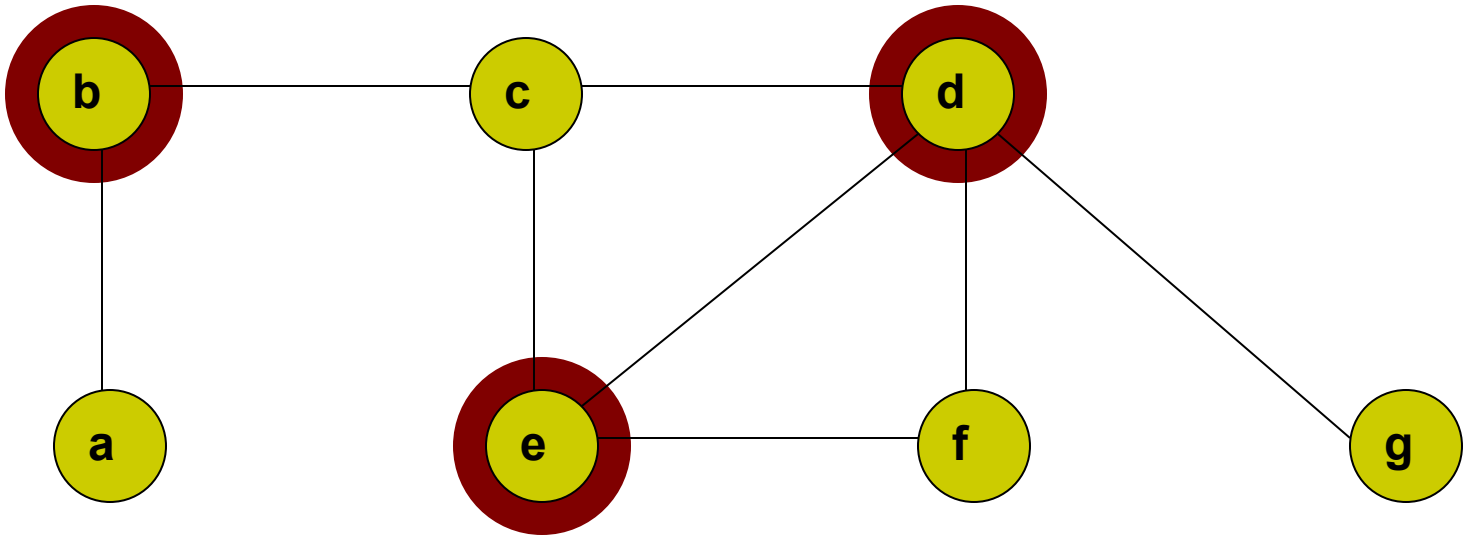
Improved version:



C : d e b c Approximate vertex cover. C=4

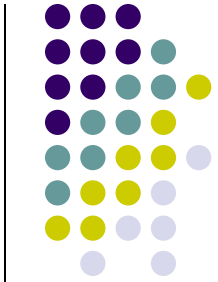
E' : (d,e) (c,d) (c,e) (d,f) (d,g) (e,f) (b,c) (a,b)

Is it possible to approximate the optimal solution?



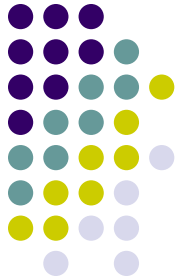
In this case, and with our algorithm, NO!

Analysis of Algorithm



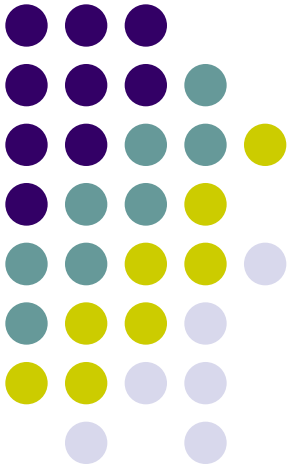
- Running time:
  - O(V+E): if we use adjacency list.
- 2-Approximation Algorithm
  - Minimization problem C* = 3; C = 6
  - Factor = C/C* = 2

Theorem: APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm



- Proof:
  - Polynomial time algorithm because : Complexity = O(V+E)
  - Let, A = set of edges picked by approximation algorithm
  - No 2 edges in A share an endpoint.
  - Thus no 2 edges in A are covered by the same vertex in C*
  - **So, Lower bound: |C*| >= |A|**
  - We pick an edge for which neither of its endpoints are already in C
  - **So, Upper bound: |C| = 2|A|**
  - **Therefore: |C| = 2|A| <= 2|C*|**

Traveling Salesman



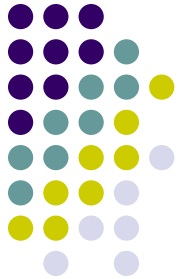
TSP with triangle inequality

Polynomial-time 2-approximation algorithm

TSP without triangle inequality

Negative result on polynomial-time ρ (n)-approximation algorithm

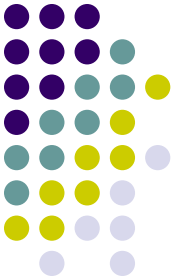
The traveling-salesperson problem.



- Given a complete undirected graph G=(V,E)
- Each edge has a nonnegative integer cost c(u,v)
- Find a Hamiltonian cycle of G with minimum cost.

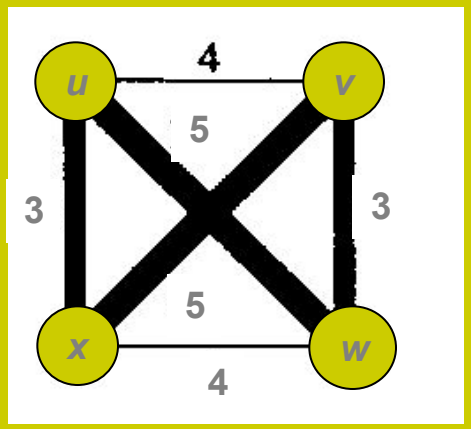
- This is a NP complete problem

Triangle Inequality



- The cost function satisfies the triangle inequality for all vertices u, v, w in V
  - c(u,w) <= c(u,v) + c(v,w)

TSP with Triangle Inequality



Cost Function Satisfies

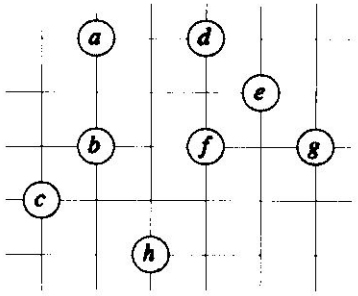
Triangle Inequality

$$\forall u, v, w \in V$$

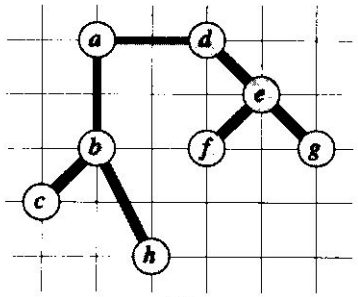
$$c(u, w) \leq c(u, v) + c(v, w)$$

APPROX-TSP-TOUR(G, c)

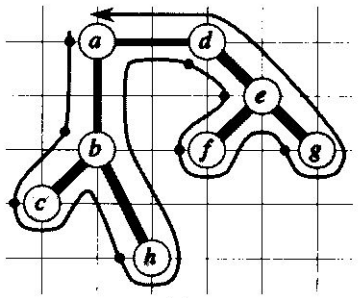
1 select a vertex $r \in V[G]$ to be a "root" vertex
2 compute a minimum spanning tree T for G from root r using MST-PRIM(G, c, r)
3 let L be the list of vertices visited in a preorder tree walk of T
4 **return** the hamiltonian cycle H that visits the vertices in the order L



(a)

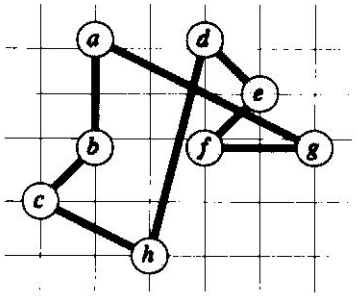


(b)



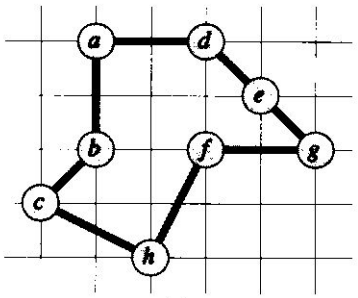
(c)

"full walk" = abcbhbadefegeda



(d)

final approximate tour

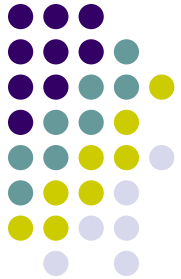


(e)

optimal tour

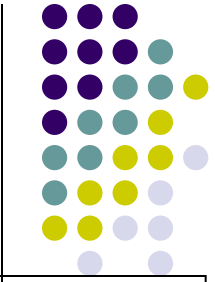
source: 91.503 textbook Cormen et al.

TSP with Triangle Inequality



- Time Complexity:
  - MST-Prim()'s running time: $O(V^2)$
  - So, APPROX-TSP-TOUR's running time: $O(V^2)$

TSP with Triangle Inequality



Theorem: ***APPROX-TSP-TOUR*** *is a* ***polynomial-time 2-approximation algorithm for TSP with triangle inequality****.*

Proof: Algorithm runs in time polynomial in *n*.

Let H^* be an optimal tour and T be MST

$c(T) \le c(H^*)$ (since deleting 1 edge from a tour creates a spanning tree)

Let W be a full walk of T (lists vertices when they are first visited and when returned to after visiting subtree)

$c(W) = 2c(T)$ (since full walk traverses each edge of T twice)

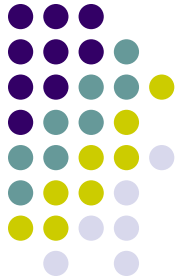
$c(W) \le 2c(H^*)$

Now make W into a tour H using triangle inequality.

$c(H) \le c(W)$ (New inequality holds since H is formed by deleting duplicate vertices from W.)

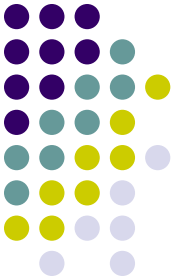
$$c(H) \le 2c(H^*)$$

TSP without Triangle Inequality



- If $P \neq NP$, then for any constant $\rho \geq 1$, there is no polynomial time approximation algorithm with approximation ratio ρ for the general traveling-salesman problem
- Prove: try yourself

TSP without Triangle Inequality



***Theorem**: If* $P \neq NP$, *then for any constant* $\rho \geq 1$

~~***polynomial-time approximation algorithm with ratio*** ρ~~

***for TSP without triangle inequality**.*

***Proof**: (by contradiction)* Suppose there **is** one --- call it *A*

Showing how to use *A* to solve NP-complete Hamiltonian Cycle problem ⇒ contradiction!

Convert instance G of Hamiltonian Cycle into instance of TSP (in polynomial time):

$$E' = \{(u,v) : u,v \in V, u \neq v\}$$ (G'=(V,E') is complete graph on V)

$$c(u,v) = \begin{cases} 1 & (u,v) \in E \\ \rho|V|+1 & otherwise \end{cases}$$ (assign integer cost to each edge in E')

For TSP problem (G',c):

G has Hamiltonian Cycle ⇒ (G',c) contains tour of cost |V|

G does not have Hamiltonian Cycle ⇒ Tour of G' must use some edge not in E

Cost of that tour of G' $\geq (\rho|V|+1)+(|V|-1) = \rho|V|+|V| > \rho|V|$

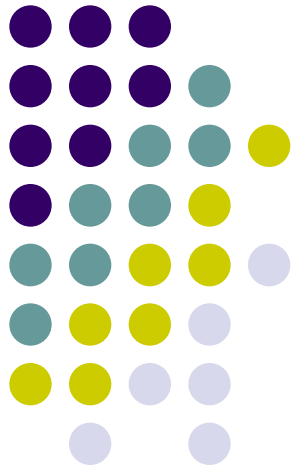
Can use *A* on (G',c)! *A finds tour of cost at most ρ(length of optimal tour)*

G has Hamiltonian Cycle ⇒ *A* finds tour of cost at most ρ|V| ⇒ cost = |V|

G does not have Hamiltonian Cycle ⇒ *A* finds tour of cost > ρ |V|

$P \neq NP$

MAX-3-CNF Satisfiability

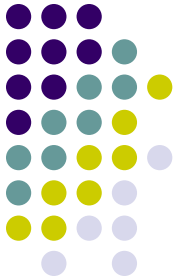


3-CNF Satisfiability Background

Randomized Algorithms

Randomized MAX-3-CNF SAT Approximation Algorithm

MAX-3-CNF Satisfiability



- Background on Boolean Formula Satisfiability
  - Boolean Formula Satisfiability: Instance of language **SAT** is a boolean formula φ consisting of:
    - *n* boolean variables: $x_1, x_2, \ldots, x_n$
    - *m* boolean connectives: boolean function with 1 or 2 inputs and 1 output
      - e.g. AND, OR, NOT, implication, iff

        $\vee \quad \wedge \quad \neg \quad \rightarrow \quad \leftrightarrow$
    - parentheses
    - truth, satisfying assignments notions apply

$$SAT = \{< \phi >: \phi \text{ is a satisfiable boolean formula}\}$$

NP-Complete

source: 91.503 textbook Cormen et al.

MAX-3-CNF Satisfiability
(continued)

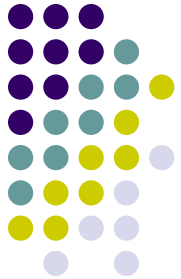
- Background on 3-CNF-Satisfiability
  - Instance of language **SAT** is a boolean formula φ consisting of:
    - *literal:* variable or its negation
    - CNF = conjunctive normal form
      - *conjunction*: AND of clauses
      - *clause*: OR of literal(s)
    - 3-CNF: each clause has **exactly 3** distinct literals

MAX-3-CNF Satisfiability: optimization version of 3-CNF-SAT
- Maximization: satisfy as many clauses as possible
- Input Restrictions:
  - exactly 3 literals/clause
  - no clause contains both variable and its negation

NP-Complete

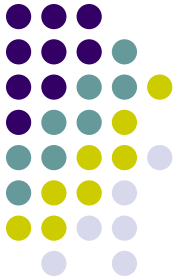
source: 91.503 textbook Cormen et al.

Randomized approximation algorithm for MAX-3-CNF-SAT



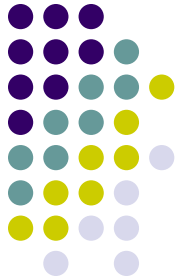
- Each clause has 3 distinct literals. Also assume that no clause contains a variable and its negation.
- **Problem**: Return an assignment of variables which maximizes the number of clauses which evaluate to true.

The random approximation algorithm



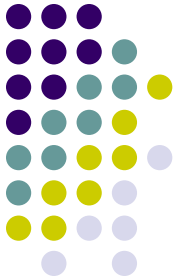
- **Randomly assign a 0 or a 1 value to all variables.**

Analysis



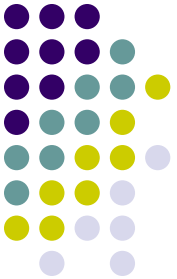
- We have set each variable to 1 with a ½ probability.
- We have set each variable to 0 with a ½ probability.
- We define an indicator random variable
  - Y_i = I {clause i is satisfied}
- Y_i = 1 as long as at least one of the literals in the i^{th} clause has been set to 1.
- A clause is not satisfied iff all it's literals are 0.
  - Pr {clause i is not satisfied} = $(½)^3$ = 1/8

Analysis [cont.]



- So Pr {clause i is satisfied} = 1 – 1/8 = 7/8
- Lemma 5.1 (CLRS)
  - Given a sample space S and an event A in the sample space, let $X_A = I\{A\}$. Then $E[X_A] = Pr\{A\}$
- $E[Y_i] = 7/8$

Analysis [cont.]



- Let Y = number of satisfied clauses.
  - $Y = Y_1 + Y_2 + \ldots\ldots + \ldots\ldots Y_m$
- Then we have:

$$
\begin{aligned}
E[Y] &= E\left[\sum_{i=1}^{m} Y_i\right] \\
&= \sum_{i=1}^{m} E[Y_i] \\
&= \sum_{i=1}^{m} 7/8 \\
&= 7m/8
\end{aligned}
$$

- Since m is the upper bound on the number of satisfied clauses, the approximation ratio is m/(7m/8) = 8/7
- Therefore this is a 8/7 approximation algorithm