Approximation Algorithms

Combinatorial Optimization problems

- The problems in which the goal is to find the "best" object from within some finite space of objects, e.g.
  - Shortest path:
    - Given a graph with edge costs and a pair of nodes, find the shortest path (least costs) between them
  - Traveling salesman:
    - Given a complete graph with nonnegative edge costs, find a minimum cost cycle visiting every vertex exactly once
  - Maximum Network Lifetime:
    - Given a wireless sensor networks and a set of targets, find a schedule of these sensors to maximize network lifetime

Optimization Problems: Formalization

- Given:
  - A problem instance I.
- Goal:
  - With respect to a defined parameter, we wish
    - to find the optimum solution …..that may be maximum or minimum
    - i.e. to minimize (or maximize) some cost function c(S) for a "solution" S to the problem instance I.
- e.g.
  - INDEPENDENT SET……maximum number of vertices in a graph..
  - Shortest Path…..minimum cumulative edges weights in a path…..
  - TSP…..smallest number of edges that span a specific vertices

Optimization Problems…

- However, now we are interested in problems that are not solvable by a deterministic machine in polynomial time.

Class P and NP

- Informal Definitions:
  - The class P consists of those problems that are solvable in polynomial time, i.e. $O(n^k)$ for some constant k where n is the size of the input.
  - The class NP consists of those problems that are "verifiable" in polynomial time

What is an NP problem ?

- A problem that belongs to class NP !!!
- When can a problem can be said to be belonging to class NP ?
  - a problem is assigned to the class NP if it is solvable in polynomial time by a nondeterministic Turing machine.
  - why do we have to bring in a nondeterministic Turing machine?
  - loosely
    - the class of decision problems for which a given proposed solution for a given input can be checked/certified quickly in polynomial time.
- Polynomial time verification/certification….

Polynomial time verification

- Given ……
  - An algorithm C(s, t) is a certifier for problem X if for every string s, $s \in X$ there exists a string t such that C(s, t) = yes.
  - Certification algorithm intuition.
    - Certifier views things from "managerial" viewpoint.
    - Certifier doesn't determine whether $s \in X$ on its own……..rather, it merely checks or certifies a proposed proof t that $s \in X$.

Decision Problems

- Decision Problems…..
  - only output a YES-NO
- Decision version of TSP
  - Given n cities, is there a TSP tour of length at most L?
- Why Decision Problem? What relation between the optimization problem and its decision?
- Decision is not "harder" than that of its optimization problem
  - If the decision is "hard", then the optimization problem should be "hard"

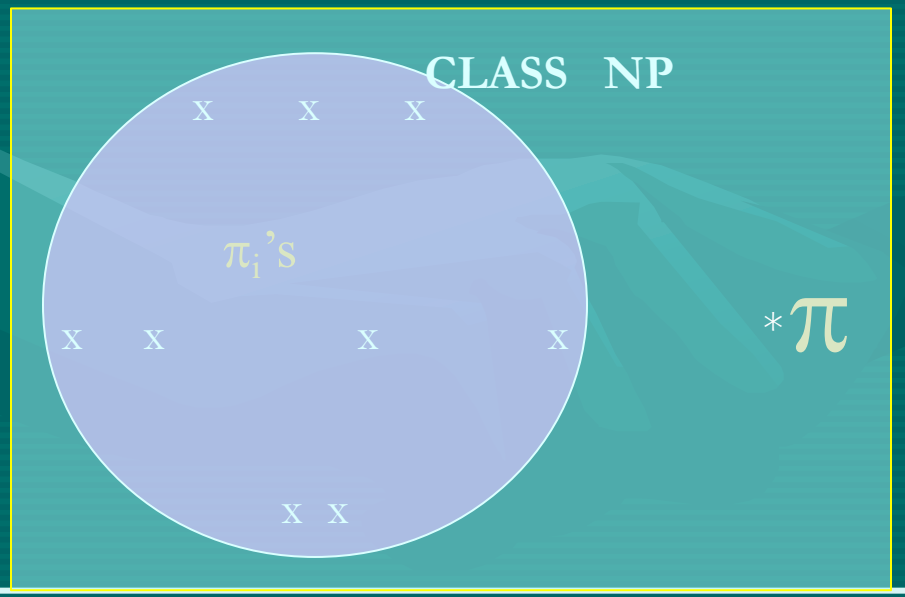
Types of Optimization Problems

- An optimization problem may be
  - P,
  - NP-Hard or
  - NP-Complete
  - …..

NP-Hard Problem

- A problem π is NP-Hard if for every problem $\pi_1 \in$ NP, π_1 reduces to π.
  - i.e. π is at least as difficult as π_1 OR
  - i.e. π is at least as difficult as every problem in class NP.

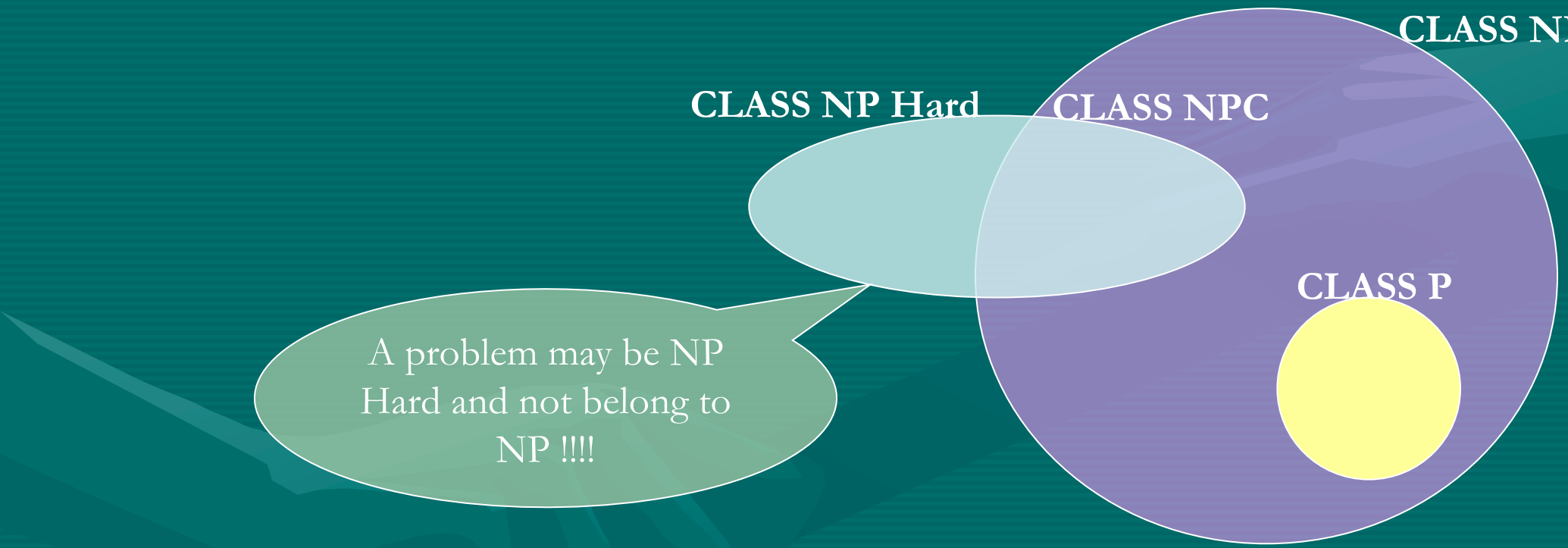
- Graphically,



Here, every π_i in CLASS NP is polynomially reducible to a problem π that is outside NP

NP-Complete problem

- A decision problem is NP-complete
  - when it is both in class NP and is NP-hard.



Classes of Discrete Optimization Problems

- Class 1
  - problems have polynomial-time algorithms for solving the problems optimally.
  - e.g. Minimum Spanning Tree problem, SSSP problem….
- Class 2 …NP-hard problems
  - No polynomial-time algorithm is known….
  - it is very likely that there is none.
  - e.g. Traveling Salesman Problem

Dealing with NP problems

- However, NP problems need solutions in real-life
- Up to now…..
  - the best algorithm for solving a hard optimization problem requires exponential time in the worst case.
- Thus, the question is ……How to deal with NP….*ness* ?

Dealing with limitations of algorithm computations…

- Three main directions to solve NP-hard discrete optimization problems:
  - Integer programming techniques
  - Heuristics
  - Approximation algorithms

Approximation : Solution for NP problems

- To reduce the time required for solving a problem…….
  - we can relax the problem, and
  - obtain a feasible solution "close" to an optimal solution
- We compromise on optimality for a good feasible solution…… No heuristics!
- In Hochbaum's words…..
  - "Trading-off optimality in favour of tractability is the paradigm of approximation algorithms".

Approximation vs Heuristics

- Ideally,
  - we would like to have a guarantee on how close to optimal, the solution given by our algorithm is…...
    - e. g. the case of Euclidean TSP algorithm …….
      - a tour whose length at most a factor ρ times the minimum length of a tour, for a (hopefully small) value of ρ.

- Heuristics
  - may produce good solutions but do not come with a guarantee on the quality of their solution.

Approximation Ratios

- Basic terminology
  - Let, OPT(I) denote the value of an optimal solution to the problem under consideration for input I.
  - e.g.
    - OPT(G) in TSP denotes……
    - OPT(H) in the independent-set problem denotes……

Approximation ratio, informally

- The approximation ratio ρ is defined as the ratio

$$\frac{result\ obtained\ by\ the\ algorithm}{the\ optimal\ cost\ or\ profit}$$

- What should be the value of this ratio relative to 1?
- Typically this ratio is ……
  - e.g. in TSP…...
  - e.g. in an airline reservation example……

Approximation ratio, informally…

- A k-approximation algorithm is an algorithm with approximation ratio k
  - e.g. both the instances above are 2-approximation algorithms.

Approximation Ratio: Minimization problem

- Consider a minimization problem.
  - Let OPT(I)= value of the optimal solution….a positive integer.
  - consider an algorithm A that gives an output solution A(I), on an instance I.
- Then, the appro^{n} ratio of the algorithm A is $\rho = \max_I \frac{A(I)}{OPT(I)}$
- That is, $OPT(I) \leq A(I)$
  - however, closer the value of the ρ is to one, better is the appro^{n} algorithm A(I)

Approximation Ratio: Maximization problem

- Consider a maximization problem.
  - Let OPT(I)= value of the optimal solution….a positive integer.
  - consider an algorithm A that gives an output solution A(I), on an instance I.
- Then, the appro^{n} ratio of the algorithm A is $\rho = \max_I \frac{OPT(I)}{A(I)}$
- That is, $A(I) \leq OPT(I)$
  - however, closer the value of the ρ is to one, better is the appro^{n} algorithm A(I)

Approximation Ratio

$$1 \leq \text{Max}\,(A(I)/OPT(I), OPT(I)/A(I)) \leq \rho(n)$$

Minimization problem

Maximization problem

Formal definition of AA

- def:
  - An ρ-*approximation algorithm* is a polynomial-time algorithm that always produces a solution of value within ρ times the value of an optimal solution.
  - that is, for any instance of the problem

    $A(I)/OPT(I) \leq \rho$, ……..*for a minimization problem*

    $OPT(I)/A(I) \leq \rho$, ……..*for a maximization problem*

- ρ is called the *approximation guarantee* (or *factor*) of the algorithm.

Approximation Scheme

- ρ may be a function of the input size.
- When an algorithm A achieves an approximation ratio ρ we call it a ρ - approximation algorithm.
- An approximation scheme for an optimization problem
  - is an approximation algorithm that takes as inputs
    - an instance of the problem and
    - a value $\in > 0$,
  - such that for a fixed $\in$, the scheme is a $(1 + \in)$ - approximation algorithm.

Characteristics of AAs

- Approximation algorithms ……..
  - are time-efficient ……sometimes not as efficient as heuristics
  - don't guarantee optimal solution, however
    - guarantee a "good" solution within some factor of the optimum
  - rigorous mathematical analysis is required to prove the approximation guarantee
  - often use algorithms for related problems as subroutines

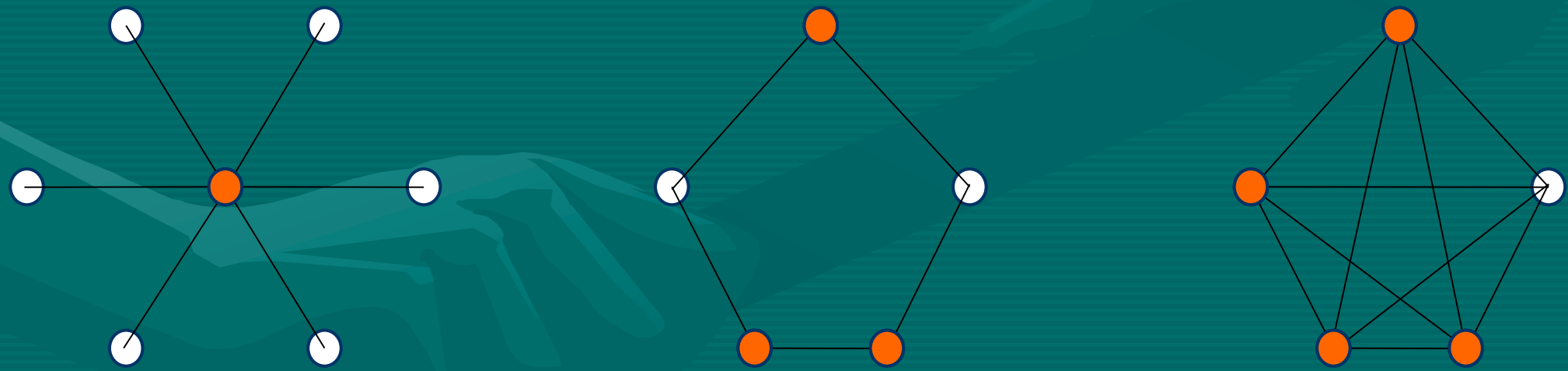
The Vertex Cover Problem

- VERTEX COVER
  - informally a subset of vertices that include all the edges…..an edge is covered if one of its endpoint is chosen.

- Given a graph G = (V, E) and an integer k,
  - is there a subset of vertices $S \subseteq V$ such that for each edge, at least one of its endpoints is in S?

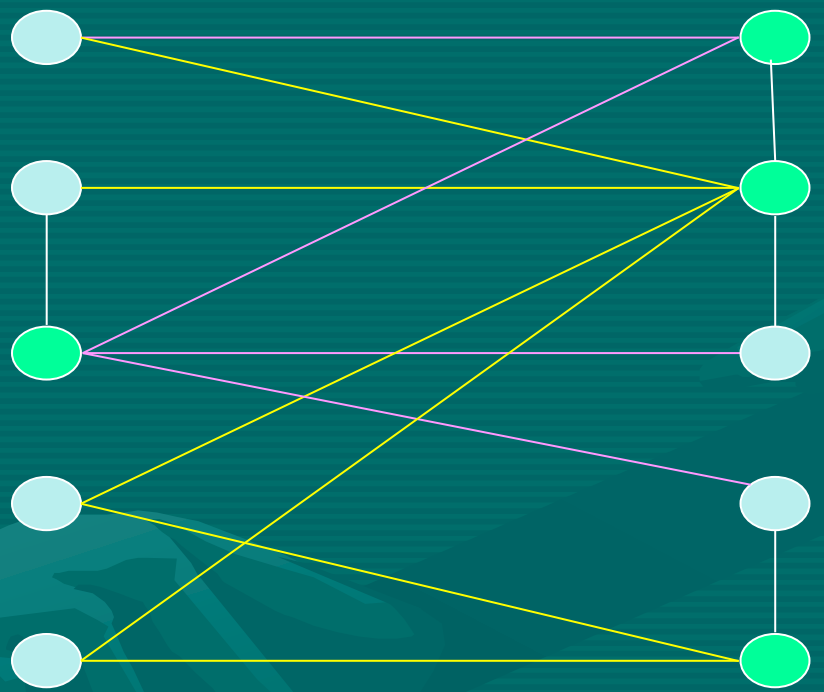
The Vertex Cover Problem…

- The Minimum Vertex Cover
  - Find a vertex cover with minimum number of vertices.
  - Given a graph G = (V, E) and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?



The Vertex Cover Problem

- Ex. Is there a vertex cover of size ≤ 4? Yes. size ≤ 3? No.



vertex cover

The Vertex Cover Problem is NPC

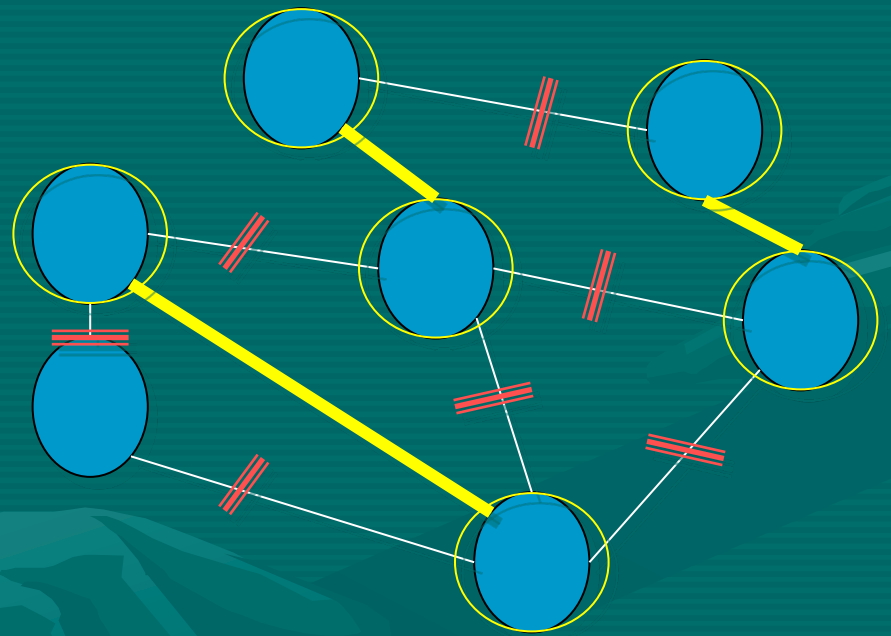
- The problem
  - of finding a vertex cover i.e. optimal vertex cover is a classical optimization problem
  - is a typical example of an NP-hard optimization problem

- However, has an approximation algorithm to sub optimally solve it i.e. one that returns a near-optimal vertex cover

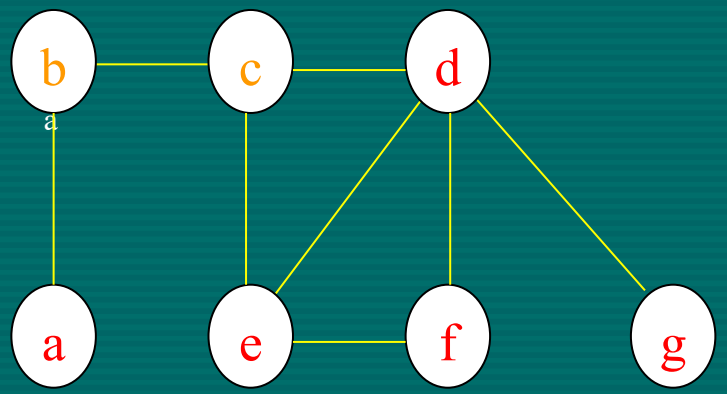
Approximate Vertex Cover

- Algorithm Outline
  - Let S be the cover….initialized to Φ
  - Pick an edge (u,v) in the graph
  - Add it's end-points u and v to S
  - Remove any edge that neighbors u or v i.e. one that is incident on either u or v.
  - Repeat until there are no edges left

Approximate Vertex Cover: An Example…



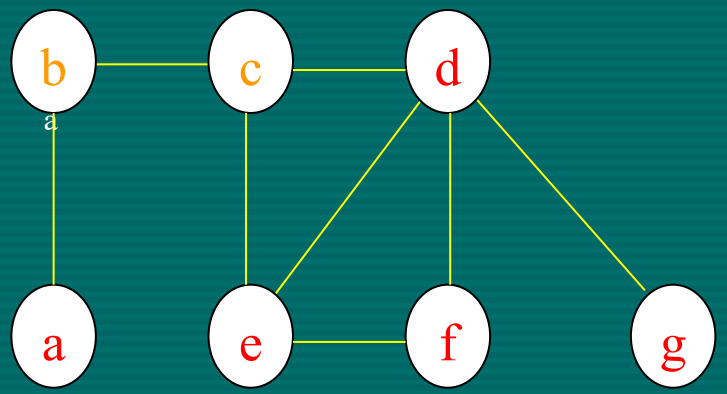
Approximate Vertex Cover: An Example



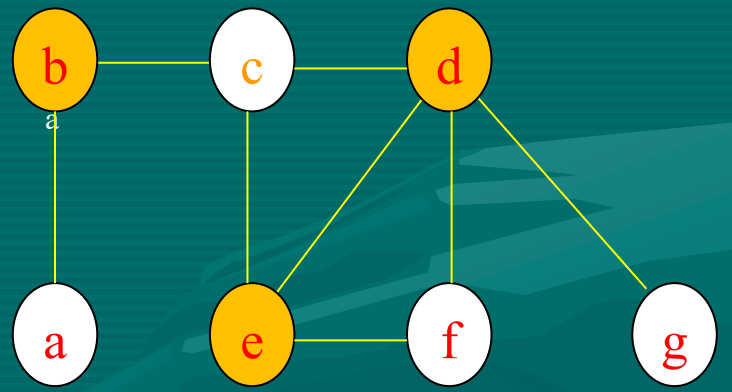
Input Graph

What is the optimal vertex cover size ? Which vertices?

Approximate Vertex Cover: An Example

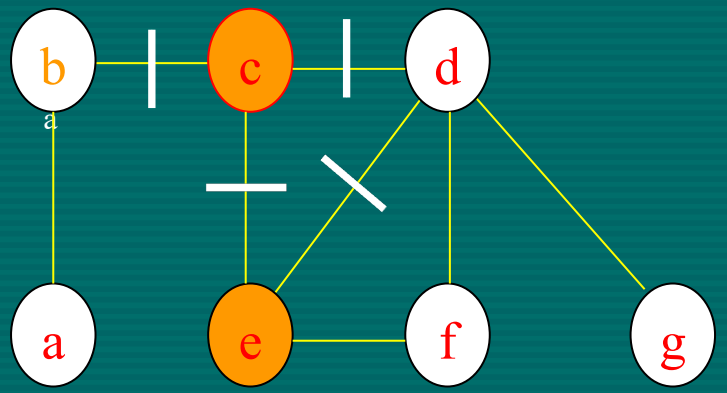


Input Graph

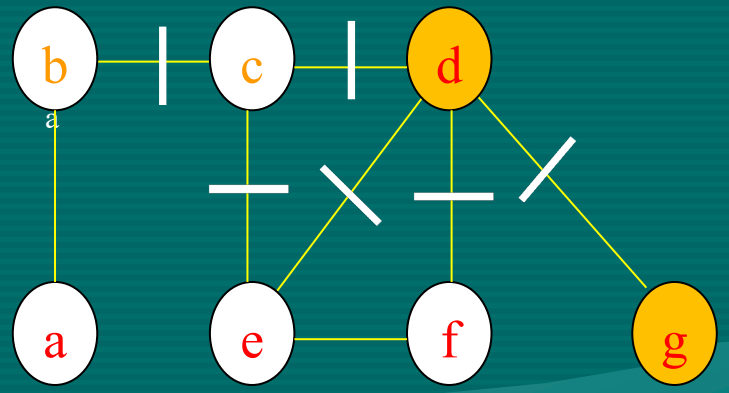


Optimal result, Size 3

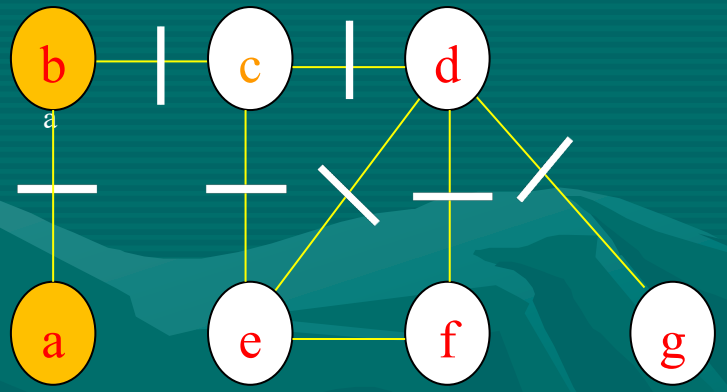
Approximate Vertex Cover: An Example...



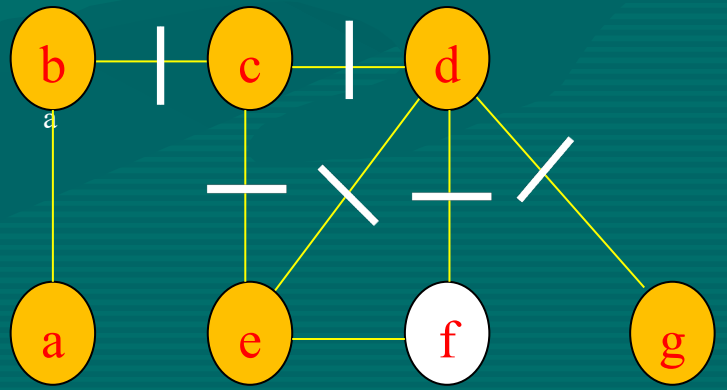
Step 1: choose edge (c,e)



Step 2: choose edge (d,g)



Step 3: choose edge (a,b)



Result : size = 6

Approximate Vertex Cover Algorithm

APPROX-VERTEX COVER (G)

1. C = ϕ
2. E' = E[G]
3. while E' != ϕ
4. let (u,v) be an arbitrary edge of E'
5. C = C u {u,v}
6. remove from E' every edge incident on either u or v
7. return C

AVCA: Time Complexity

APPROX-VERTEX COVER (G)

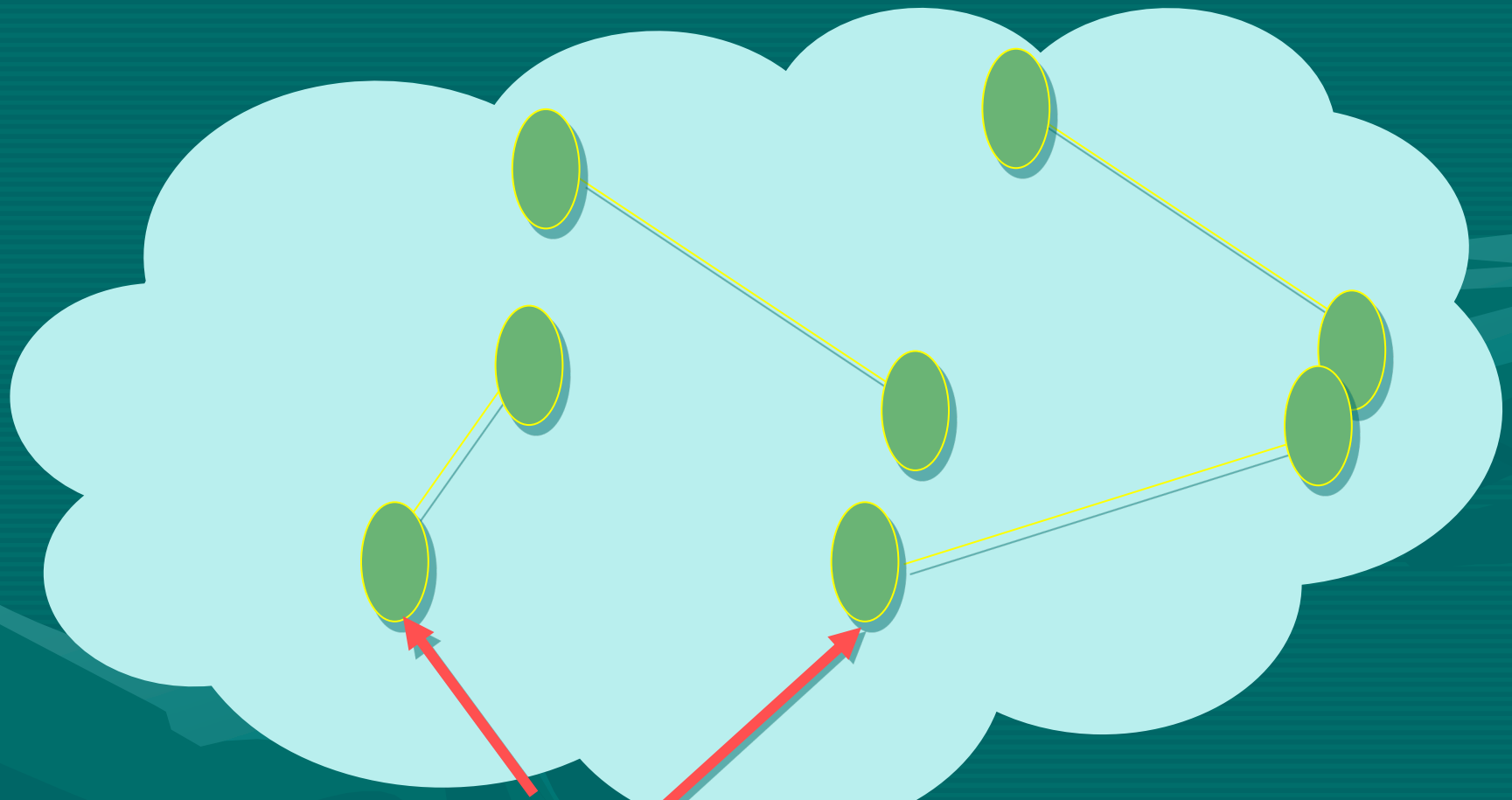
1. C = ϕ
2. E' = E[G] } $O(n^2)$
3. while E' != ϕ
4. let (u,v) be an arbitrary edge of E' } $O(1)$
5. C = C ∪ {u,v}
6. remove from E' every edge incident on either u or v } $O(n)$
7. return C

$O(n^2)$

- The algorithm runs in O(V + E) time.

How good the approximation is ?

Observe the set of edges our algorithm chooses



no common vertices! $\Rightarrow$ any VC contains 1 in each

our VC contains both, hence at most twice as large

Approximate Vertex Cover…

- OPT must cover every edge so that……
  - either u or v must be in the cover
  - this implies that => OPT > ½|S|……..where S is the vertex cover
  - this implies that 2*OPT ≥ |S|
- Therefore, we have a 2-approximation.

Proof of Approximate Vertex Cover

- Theorem: APPROX-VERTEX-COVER is a 2-approximation algorithm.
- Proof:
  - The algorithm clearly returns a vertex cover, since it loops until all edges have been removed.
  - Each edge removed in line 6 was covered by some vertex of an edge removed in line 4.
  - Let A denote the set of edges removed in line 4:
    - The optimal cover OPT(I) must contain at least one endpoint of each of the edges in A.
  - No two edges in A share an endpoint because all edges that share an endpoint with an edge in A are removed in line 6.

Proof of Approximate Vertex Cover …

- Proof: …..continued
  - Since no two edges in A are covered by the same vertex in OPT(I) , we have that

    $|OPT(I)| \geq |A|$ …… (1)

    is a lower bound on the size of OPT(I).
  - Each execution of line 4 picks an edge, both of whose endpoints are added to the approximate cover C.
  - This gives us an upper bound on the size of C:
    - $|C| = 2|A|$ ……(2)
  - Combining equations (1) and (2), we get $|C| \leq 2|OPT(I)|$

Set Cover Problem (SCP): Illus^{tn}#0

- Consider a scenario…..
  - Say you'd like to send some message to a large list of people (e.g. the entire campus)
  - The Central Computer Centre provides you the existing available mailing-lists
    - However, the moderator of each list charges Rs 100 for each message sent
  - Hence, one would like to find the smallest set of lists that covers all recipients

Set Cover Problem (SCP): Illus^{tn}#1

- Consider a scenario…..
  - on a geometric plane, a collection of towns representing a province are marked as points.
  - the planners want to build schools for the entire province.
- Constraints
  - each school should be located in a town and
  - no one in any town should travel more than 5 kms to get to a school.
- The problem
  - What is the minimum number of schools required?

SCP : Illus^{tn}#1 - More formally ...

- For each town t……
  - let S_t = set of towns within 5 kms of it.
  - a school at t covers (itself, of course) these other towns.
- Question: How many sets S_t must be picked in order to cover all the towns in the province ?

SCP: Illus^{tn}#1 - More formally ...

- Formally:
  - Input: A set of elements X; and also given are sets $S_1, \cdots, S_m \subseteq X$
  - Output: A selection of the sets S_k whose union is X.
  - Cost: Number of sets S_k picked.
- The objective is to minimize the cost.

SCP : Illustration#2

- Consider another scenario…..
  - Execution of a project requires a certain number of skills (set X).
  - Each team member possesses a subset of the skills.
  - Select a team that together covers all the skills.
  - Objective: Minimize the number of members selected.

A greedy strategy for SCP

- DO UNTIL all elements of the skills set X are covered…….
  - Pick the set S_i with the largest number of uncovered elements.
- Easy observation: The greedy strategy is not optimal.
- How bad can be the greedy solution?
  - Luckily, not far from the optimal value.