Chapter-8

NP hard and NP Complete problems

8.1 Basic Concepts

The computing times of algorithms fall into two groups.

Group1– consists of problems whose solutions are bounded by the polynomial of small degree.

Example – Binary search o (log n) , sorting o(n log n), matrix multiplication 0(n 2.81).

NP –HARD AND NP – COMPLETE PROBLEMS

Group2 – contains problems whose best known algorithms are non polynomial.

Example –Traveling salesperson problem 0(n22n), knapsack problem 0(2n/2) etc.
There are two classes of non polynomial time problems

1. NP- hard
2. NP-complete

A problem which is NP complete will have the property that it can be solved in polynomial time iff all other NP – complete problems can also be solved in polynomial time.
The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle.''

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is correct once you see it.

Example... Does matrix A have LU decomposition?
No guarantee if answer is ``no''.

Another way of thinking of NP is it is the set of problems that can solved efficiently by a really good guesser.
The guesser essentially picks the accepting certificate out of the air (Non-deterministic Polynomial time). It can then convince itself that it is correct using a polynomial time algorithm. (Like a right-brain, left-brain sort of thing.)
Clearly this isn't a practically useful characterization: how could we build such a machine?

Exponential Upper bound

Another useful property of the class NP is that all NP problems can be solved in exponential time (EXP).
This is because we can always list out all short certificates in exponential time and check all O (2nk) of them.
Thus, P is in NP, and NP is in EXP. Although we know that P is not equal to EXP, it is possible that NP = P, or EXP, or neither. Frustrating!

NP-hardness

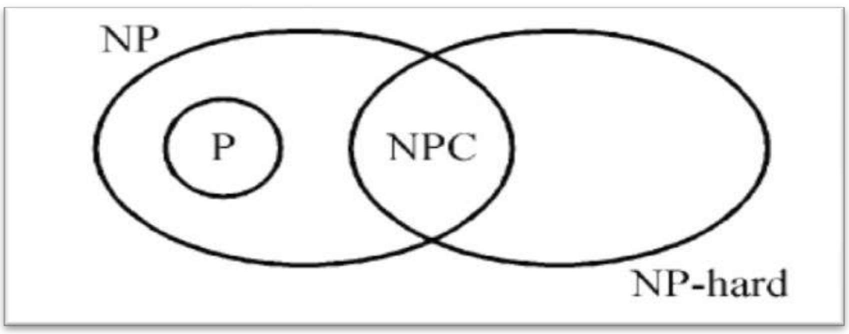
As we will see, some problems are at least as hard to solve as any problem in NP. We call such problems NP-hard.
How might we argue that problem X is at least as hard (to within a polynomial factor) as problem Y?
If X is at least as hard as Y, how would we expect an algorithm that is able to solve X to behave?

NP –HARD and NP – Complete Problems Basic Concepts

If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.

All NP-complete problems are NP-hard, but all NP- hard problems are not NP-complete.

The class of NP-hard problems is very rich in the sense that it contains many problems from a wide variety of disciplines.



P: The class of problems which can be solved by a deterministic polynomial algorithm.

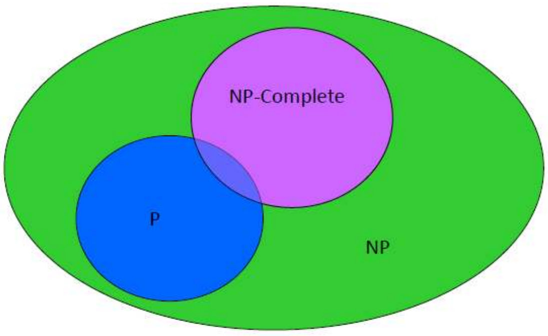
NP: The class of decision problem which can be solved by a non-deterministic polynomial algorithm.

NP-hard: The class of problems to which every NP problem reduces

NP-complete (NPC): the class of problems which are NP-hard and belong to NP.

NP-Competence

- How we would you define NP-Complete
- They are the "hardest" problems in NP



8.2 Deterministic and Nondeterministic Algorithms

- Algorithmswiththepropertythattheresultofeveryoperationisuniquelydefinedaretermeddeterministic
- Such algorithms agree with the way programs are executed on a computer.
- In a theoretical framework, we can allow algorithms to contain operations whose outcome are not uniquely defined but are limited to a specified set of possibilities.
- Themachineexecutingsuchoperationsareallowedtochooseanyoneoftheseoutcomessubjecttoaterminationcondition.
- This leads to the concept of non deterministic algorithms.
- To specify such algorithms in SPARKS, we introduce three statements
  Choice (s) ……… arbitrarily chooses one of the elements of the set S.
  Failure …. Signals an unsuccessful completion.
  Success: Signals a successful completion.
- Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.
- A non deterministic algorithm terminates unsuccessfully if and only if there exists no set of choices leading to a successful signal.
- A machine capable of executing an on deterministic algorithm is called an un deterministic machine.
- While non deterministic machines do not exist in practice they will provide strong intuitive reason to conclude that certain problems cannot be solved by fast deterministic algorithms.

Nondeterministic algorithms

A non deterministic algorithm consists of
Phase 1: guessing
Phase 2: checking

- If the checking stage of a non deterministic algorithm is of polynomial time-complexity, then this algorithm is called an NP (nondeterministic polynomial) algorithm.
- NP problems : (must be decision problems)
  –e.g. searching, MST
  Sorting
  Satisfy ability problem (SAT)
  travelling salesperson problem (TSP)
  Example of a non deterministic algorithm
  // The problem is to search for an element x //
  // Output j such that A(j) =x; or j=0 if x is not in A //
  j choice (1 :n)
  if A(j) =x then print(j) ; success endif
  print (‘0’) ; failure
  complexity 0(1);

Non-deterministic decision algorithms generate a zero or one
as their output.

Deterministic search algorithm complexity. (n)

- Many optimization problems can be recast into decision problems with the property that the decision problem can be solved in polynomial time iff the corresponding optimization problem can.
- The decision is to determine if there is a 0/1 assignment of values to xi $1 \leq i \leq n$ such that $\sum p_i x_i \geq R$, and $\sum w_i x_i \leq M$, R, M are given numbers $p_i, w_i \geq 0$, $1 \leq i \leq n$.
- It is easy to obtain polynomial time nondeterministic algorithms for many problems that can be deterministically solved by a systematic search of a solutions pace of exponential size.

8.3 Satisfiability

- Letx_1, x_2, x_3…. x_n denotes Boolean variables.
- Let x_i denotes the relation of xi.
- A literal is either a variable or its negation.
- A formula in the prepositional calculus is an expression that can be constructed using literals and the operators and ^ or v.
- A clause is a formula with at least one positive literal.
- The satisfy ability problem is to determine if a formula is true for some assignment of truth values to the variables.
- It is easy to obtain a polynomial time non determination algorithm that terminates s successfully if and only if a given prepositional formula E(x_1, x2……x_n) is satiable.
- Such an algorithm could proceed by simply choosing (non deterministically) one of the 2n possible assignment so f truth values to (x_1, x_2…x_n) and verify that E(x_1,x_2…x_n) is true for that assignment.

The satisfy ability problem

The logical formula:

x_1v x_2 v x_3
& -x_1
& -x_2
the assignment : x_1 ← F , x2 ← F , x3 ← T will make the above formula true .
(-x1, -x2, x3) represents x1 ← F, x2 ← F, x3 ← T
If there is at least one assignment which satisfies a formula, then we say that this formula is satisfiable; otherwise, it is un satisfiable.
An un satisfiable formula:

x1vx2
&x1v-x2
&-x1vx2
&-x1v-x2

Definition of the satisfiability problem:
Given a Boolean formula, determine whether this formula is satisfiable or not.
Aliteral: xi or-xi
Aclause:x1vx2v-x3Ci
A formula: conjunctive normal form (CNF)
C1&C2&…&Cm

8.4 Some NP-hard Graph Problems

ThestrategytoshowthataproblemL2isNP-hardis

1. Pick a problem L1 already known to be NP-hard.
2. Show how to obtain an instance I1 of L2m from any instance I of L1 such that from the solution of I1 We can determine (in polynomial deterministic time) thesolutiontoinstanceIofL1
3. Conclude from (ii) that L1L2.
4. Conclude from (1),(2), and the transitivity of that
   Satisfiability L1 L1L2
   Satisfiability L2
   L2is NP-hard

1. Chromatic Number Decision Problem (CNP)
   a. A coloring of a graph G = (V,E) is a function f : V { 1,2, …, k} i V
   b. If (U, V) E then f(u) f(v).
   c. The CNP is to determine if G has a coloring for a given K.
   d. Satisfiability with at most three literals per clause chromatic number problem. CNP is NP-hard.

2. Directed Hamiltonian Cycle(DHC)
   Let G=(V,E) be a directed graph and length n=1V1
   TheDHCisacyclethatgoesthrougheveryvertexexactlyonceandthenreturnstothestartingvertex.
   The DHC problem is to determine if G has a directed Hamiltonian Cycle.
   Theorem: CNF (Conjunctive Normal Form) satisfiability DHC
   DHC is NP-hard.

3. Travelling Salesperson Decision Problem (TSP) :
   1. The problem is to determine if a complete directed graph G = (V,E) with edge costs C(u,v) has a tour of cost at most M.

Theorem: Directed Hamiltonian Cycle (DHC) TSP

2. But from problem (2) satisfiability DHC Satisfiability TSP
   TSP is NP-hard.

8.5 Sum of subsets

TheproblemistodetermineifA={a1,a2,……,an}(a1,a2,………,anarepositiveintegers) has a subset S that sum s to a given integer M.

Scheduling identical processors

Let Pi1 ≤ i ≤m be identical processors or machines Pi.
Let Ji 1≤ i ≤ n be n jobs.
Jobs Ji requires ti processing time

A schedule S is an assignment of jobs to processors.

For each job Ji, S specifies the time interval s and the processors on which this job i is to be processed.
A job cannot be processed by more than one process or at any given time.

The problem is to find a minimum finish time non-preemptive schedule. The finish time of S is FT(S) = max {Ti}1≤i≤m. Where Ti is the time at which processor Pi finishes processing all jobs (or job segments) assigned to it

An NP-hard problem L cannot be solved in deterministic polynomial time.

By placing enough restrictions on any NP hard problem, we can arrive at a polynomials solvable problem.

Examples

CNF-Satisfy ability with at most three literals per clause is NP-hard. If each clause is restricted to have at most two literals then CNF-satisfy ability is polynomial solvable. Generating optimal code for a parallel assignment statement is NP-hard, However if the expressions are restricted to be simple variables, then optimal code can be generated in polynomial time.

Generating optimal code for level one directed a-cyclic graphs is NP-hard but optimal code for trees can be generated in polynomial time.

Determining if a planner graph is three color able is NP-Hard

Todetermineifitistwocolorableisapolynomialcomplexityproblem.
(Weonlyhavetoseeifitisbipartite)

General definitions - P, NP, NP-hard, NP-easy, and NP-complete... - Polynomial-time reduction • Examples of NP-complete problems

P - Decision problems (decision problems) that can be solved in polynomial time - can be solved “efficiently”

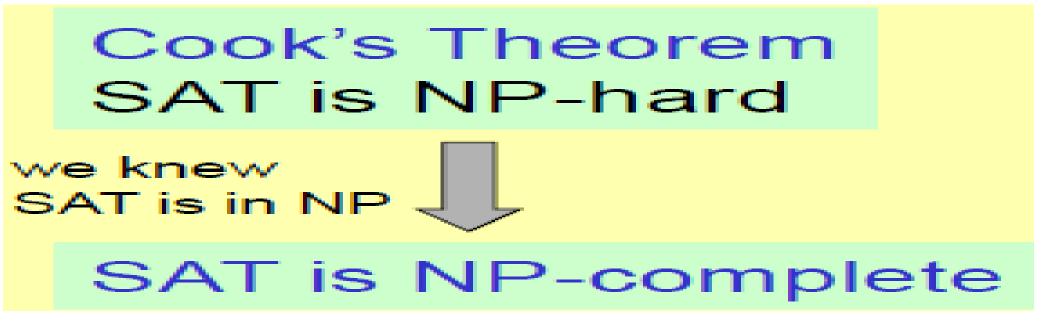
NP - Decision problems whose “YES” answer can be verified in polynomial time, if we already have the proof (or witness)

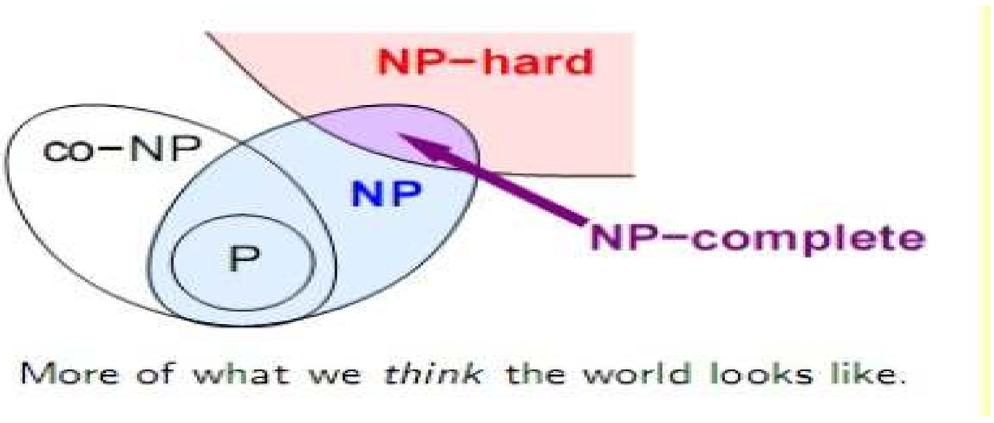
Co-NP - Decision problems whose “NO” answer can be verified in polynomial time, if we already have the proof (or witness)
E.g. the satisfy ability problem (SAT) - Given a Boolean formula is it possible to assign the input x1...x9, so that the formula evaluates to TRUE?

If the answer is YES with a proof (i.e. an assignment of input value), then we can check the proof in polynomial time (SAT is in NP). We may not be able to check the NO answer in polynomial time. (Nobody really knows.)

•NP-hard
A problem is NP-hard iff an polynomial-time algorithm for it implies a polynomial-time algorithm for every problem in NPNP-hard problems are at least *as hard as* NP problems

•NP-complete
A problem is NP-complete if it is NP-hard, and is an element of NP (NP-easy)





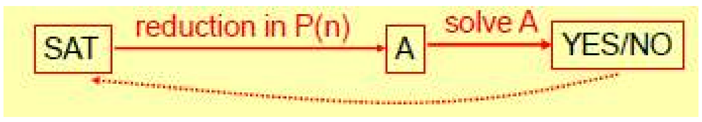
More of what we *think* the world looks like.

Relationship between decision problems and optimization problems every optimization problem has a corresponding decision problem

Optimization: minimize x, subject to constraints yes/no: is there a solution, such that x is less than c? an optimization problem is NP-hard (NP-complete) if its corresponding decision problem is NP-hard (NP-complete)

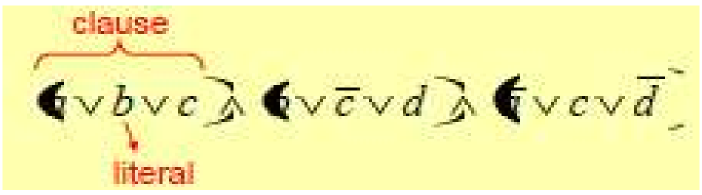
Polynomial-time reductions

How to know another problem, A, is NP-complete?
To prove that A is NP-complete, reduce a known NP-complete problem to A

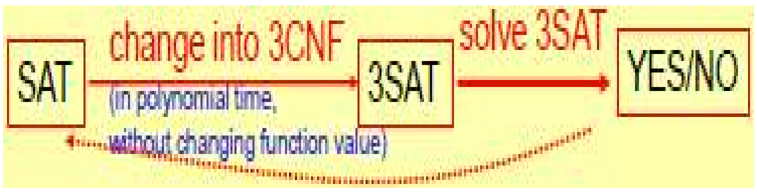


Requirement for Reduction
Polynomial time
YES to A also implies YES to SAT, while
NO to A also implies No to SAT (Note that A must also have short proof for YES answer)

An example of reduction 3CNF



3SAT: is a boolean formula in *3CNF* has a feasible assignment of inputs so that it evaluates to TRUE?
reduction from 3SAT to SAT (3SAT is NP-complete)



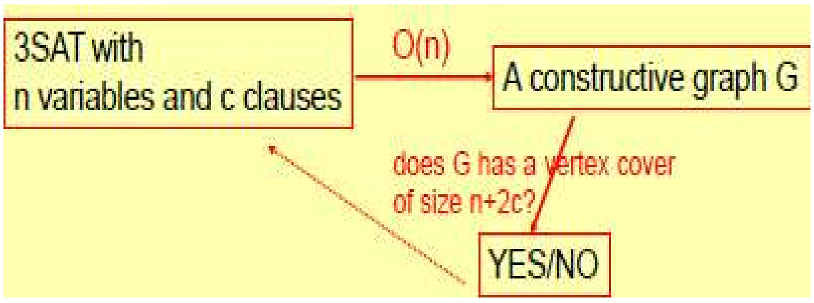
Examples of NP-complete problems

Vertex cover
Independent set
Set cover
Steiner tree

Vertex cover
Given a graph G = (V, E), find the *smallest* number of vertexes that cover *each edge*

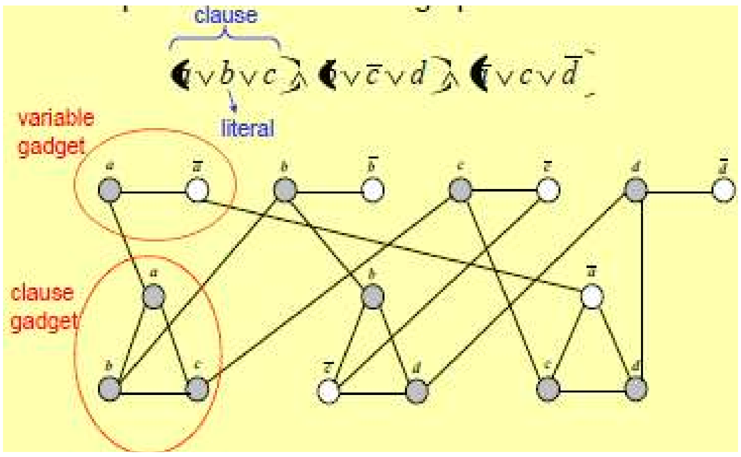
Decision problem: is the graph has a vertex cover of size K?

Reduction

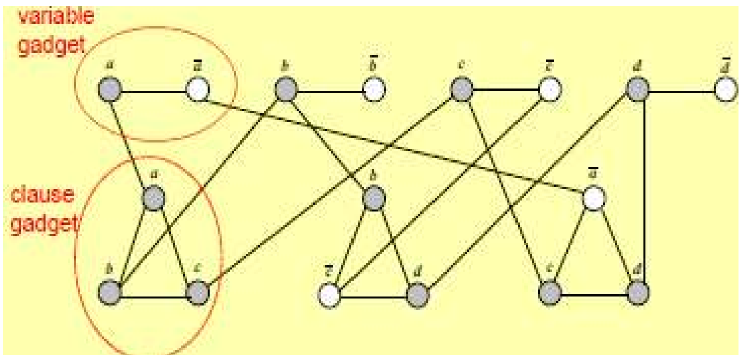


Vertex cover

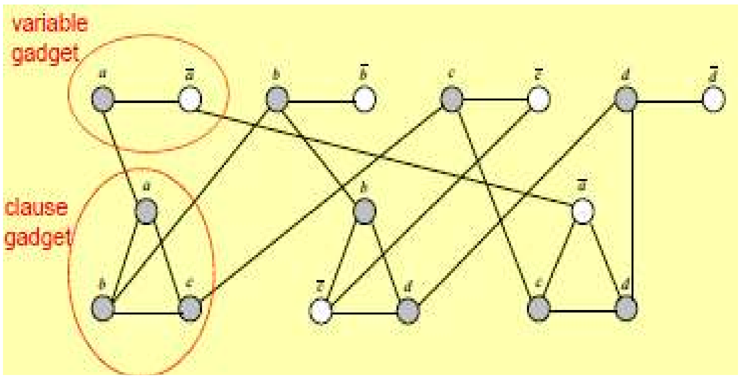
An example of the constructive graph



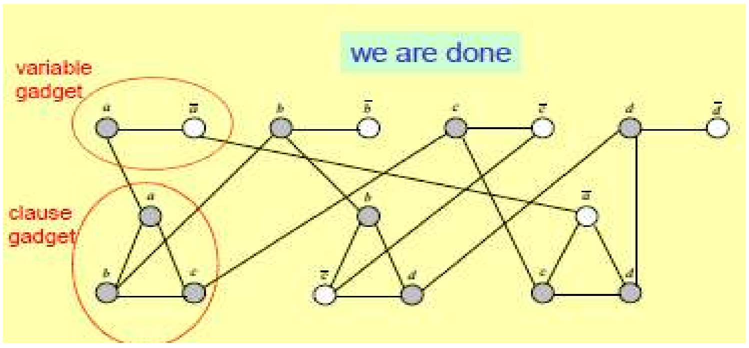
Vertex cover
We must prove: the graph has a n+2c vertex cover, if and only if the 3SAT is satisfiable (to make the two problem has the same YES/NO answer!)



Vertex cover
- If the graph has a n+2c vertex cover
1) There must be 1 vertex per variable gadget, and 2 per clause gadget
2) In each clause gadget, set the remaining one literal to be true

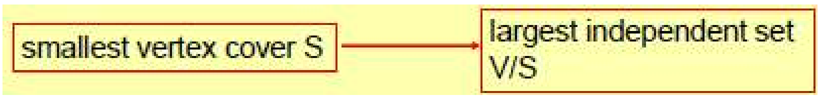


• Vertex cover
 If the 3SAT is satisfiable
1) Choose the TURE literal in each variable gadget
2) Choose the remaining two literal in each clause gadget



Independent set

Independent set: a set of vertices in the graph with no edges between *each pairof* nodes. given a graph G=(V,E), find the *largest independent set*
reduction from vertex cover:

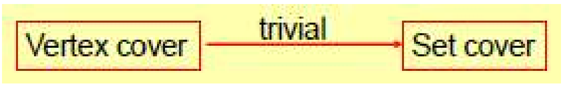


Independent set

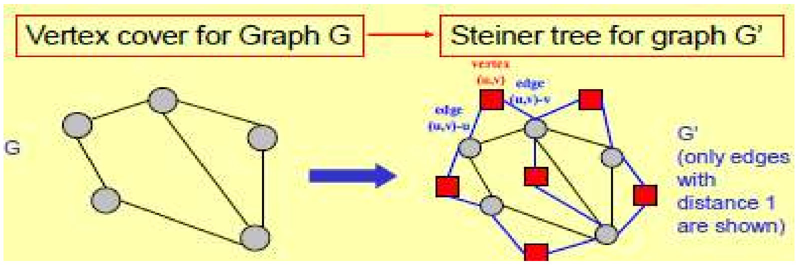
If G has a vertex cover S, then V/S is an independent set
Proof: consider two nodes in V/S: if there is an edge connecting them, then one of them must be in S, which means one of them is not in V/S
If G has an independent set I, then V/I is a vertex cover
Proof: consider one edge in G:
If it is not covered by any node in V/I, then its two end vertices must be both in I, which means I is not an independent set

Given a universal set *U*, and several subsets *S1…Sn*
find the least number of subsets that contains each elements in the universal set
vertex cover is a special case of set cover:

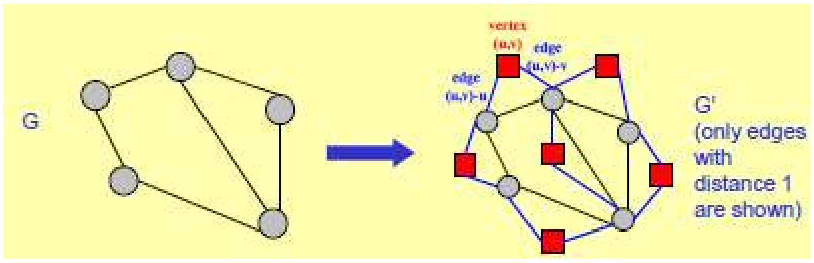
1) the universal set contains all the edges
2) each vertex corresponds to a subset, containing the edges it covers



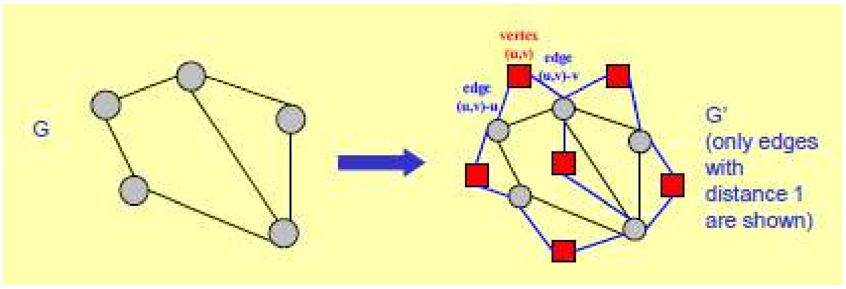
Steiner tree
Given a graph G = (V, E), and a subset C of V
find the minimum tree to connect each vertex in C reduction


Vertex cover for Graph G
Steiner tree for graph G'
vertex (u,v)
edge (u,v)-v
edge (u,v)-u
G
G' (only edges with distance 1 are shown)

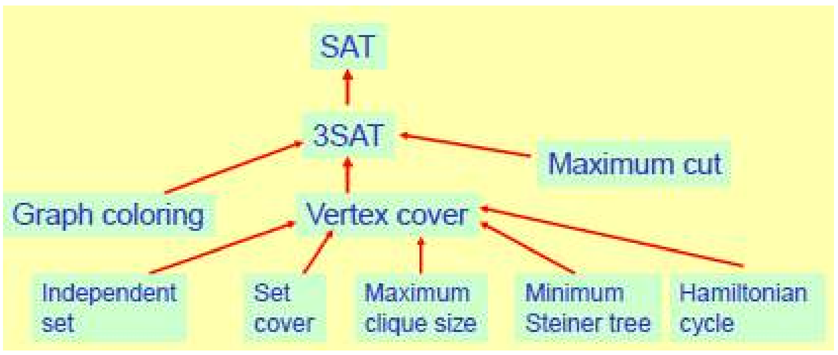
Steiner tree
- G' is a *complete graph*
- for every node u in G, create a node u in G'
- for every edge (u, v) in G, create a node (u, v) in G'
- in G', every node (u, v) is connected to u and v with distance 1
- in G', every node u and v is connected with distance 1
- other edges in G' are of distance 2



In the Steiner tree problem for G', choose C to be the set of all nodes (u, v)
G' has a minimum Steiner tree of close m+k-1 iff G has a minimum vertex cover of size k



Examples of NP-complete problems


SAT
3SAT
Maximum cut
Graph coloring
Vertex cover
Independent set
Set cover
Maximum clique size
Minimum Steiner tree
Hamiltonian cycle

****