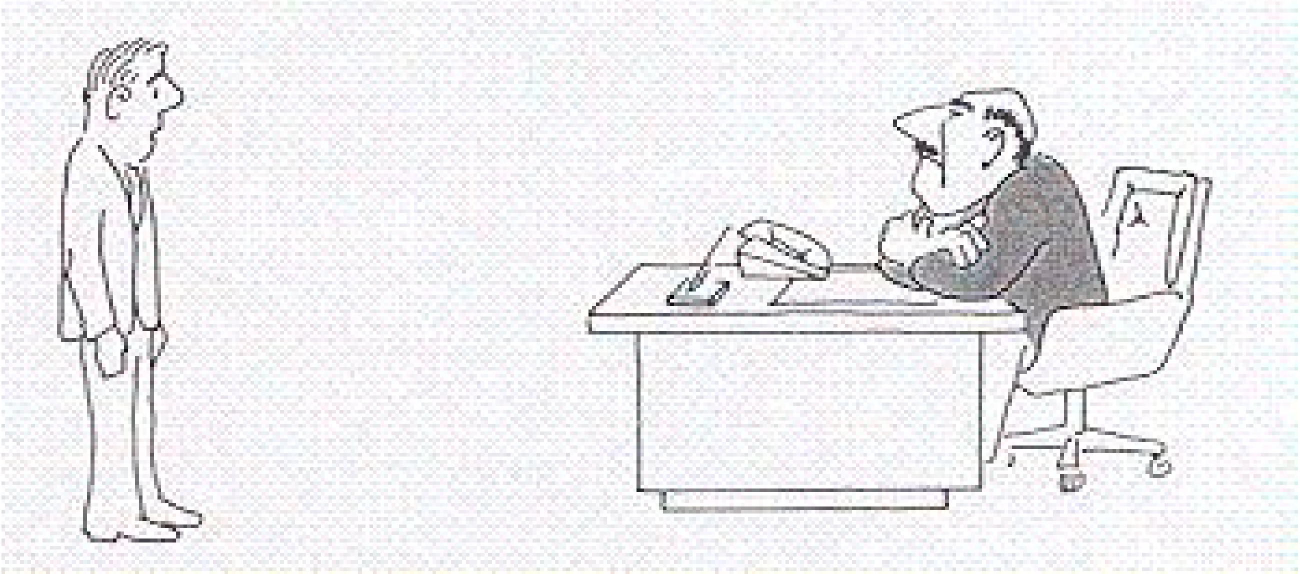
NP Completeness

Polynomial Solutions

- Polynomial complexity
  - An algorithm is said to be of polynomial time if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm
    - $T(n) = O(n^k)$ for some constant k
- Pseudo Polynomial Complexity
  - A numeric algorithm runs in pseudo-polynomial time if its running time is polynomial in the numeric value of the input (which is exponential in the length of the input – its number of digits)
    - e.g running time of 0-1 knapsack – O(nW), where length of W is proportional to bits in W ie logW

Introduction

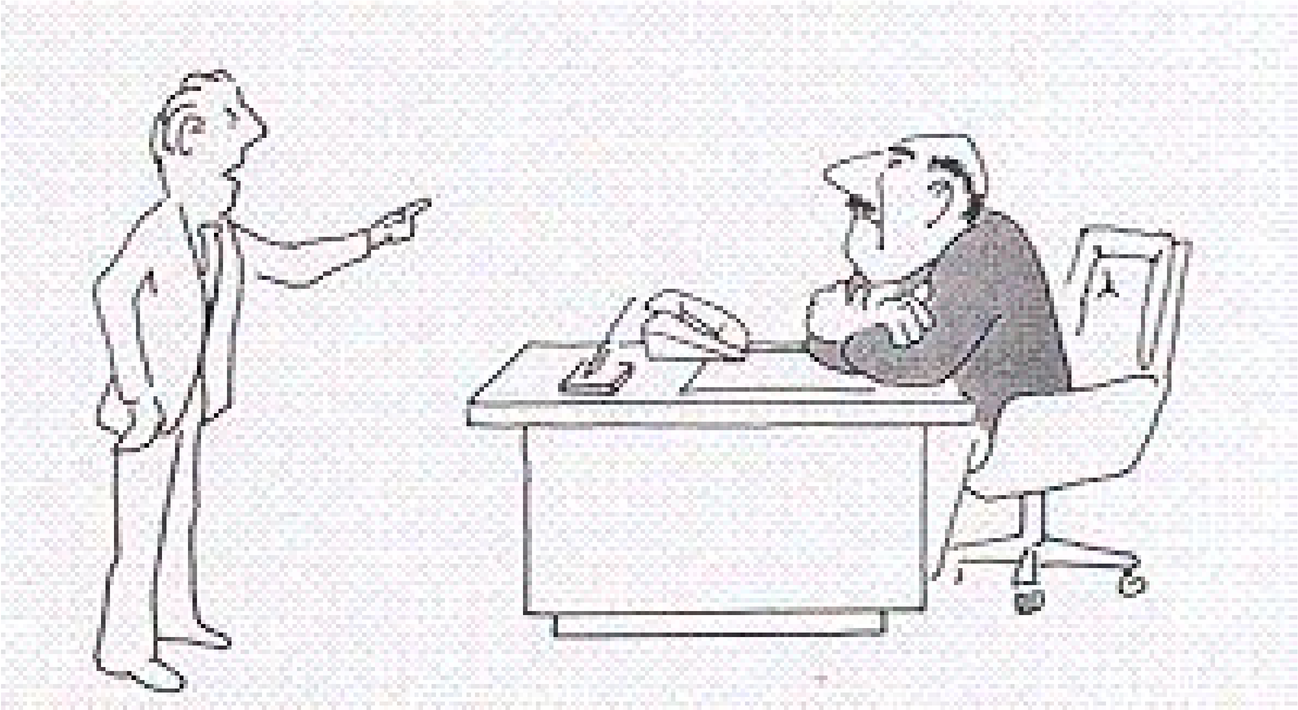
- Some computational problems are really hard to solve
  - Cant find a solution in polynomial time



I can't find an efficient algorithm, I guess I'm just too dumb

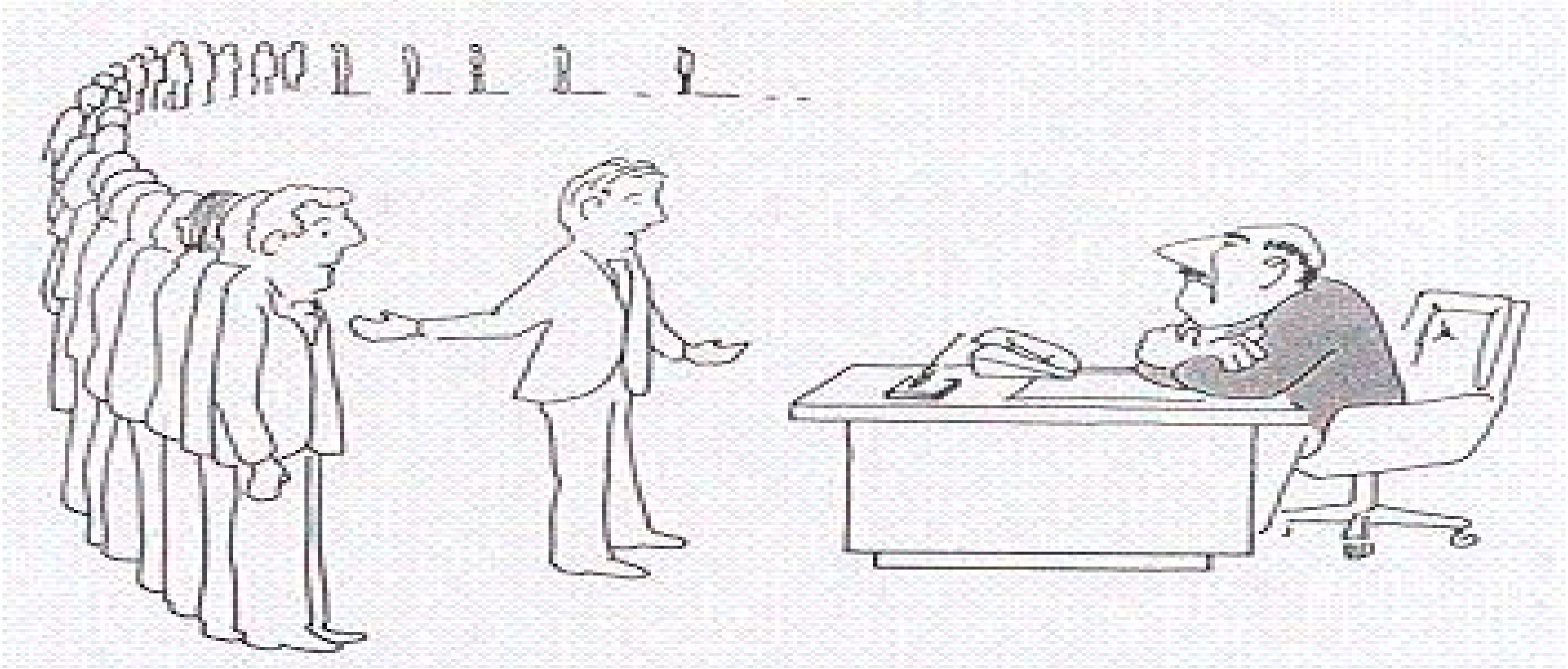
http://max.cs.kzoo.edu/~kschultz/CS510/ClassPresentations/NPCartoons.html

- Would be better if we could prove that no efficient algorithm exists for this problem
    - i.e., no algorithm can solve it quickly
- Proving intractability is equally hard



I can't find an efficient algorithm, because no such algorithm is possible.

- Theory of NP Completeness
  - provides straightforward techniques for proving that a given problem is "just as hard" as a large number of other problems that are widely recognized as being difficult and that have been confounding the experts for years



I can't find an efficient algorithm, but neither can all these famous people

Decision Problems

- Computational problems for which the output is either true or false
  - Given a string T and a string P, does P appear as a substring of T?
  - Given a weighted graph G, and an integer k, does G have a minimum spanning tree of weight of atmost k?
- Can turn any optimization problem into a decision problem
  - Introduce parameter k and ask if the optimal value to the problem ia atmost or atleast k

Complexity Class P

- P is the class of computational problems which are "efficiently solvable" or "tractable"
  - Have reasonably efficient algorithms
- They are polynomially bounded
  - Its worst case complexity is bounded by a polynomial function of the input size
  - Algorithms built from several polynomial algorithms will also be polynomially bounded
- Example problems in P
  - Finding minimum spanning tree
  - Finding s-t connectivity or reachability
  - Fractional knapsack problem

Complexity Class NP

- NP – Nondeterministic polynomial time
- NP is the set of all decision problems for which the instances where the answer is "yes" have efficiently verifiable proofs of the fact that the answer is indeed "yes."
  - A proposed solution can be checked quickly in polynomial time if it is a solution to the problem
    - Non deterministic algorithm can exhibit different behaviors on different runs

Complexity Class Np

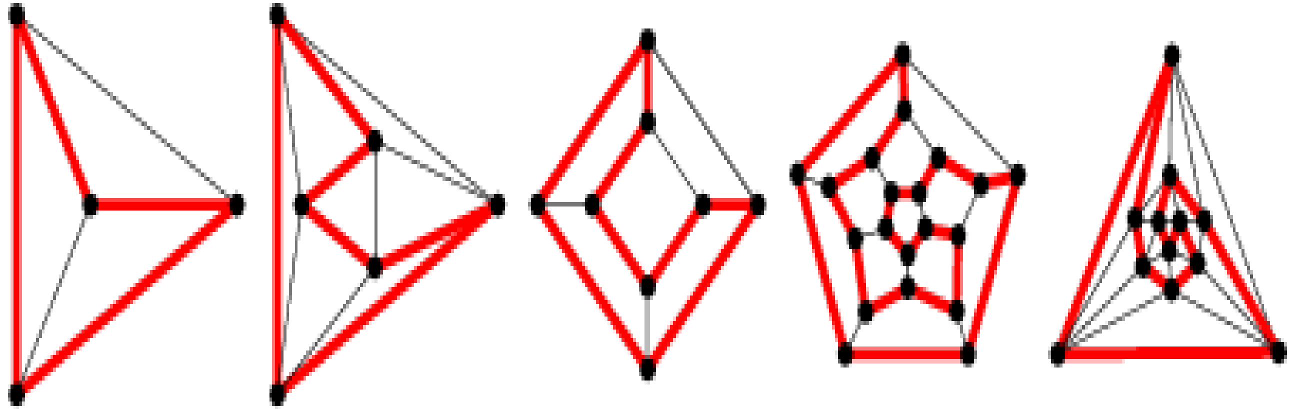
- Complexity class NP is set of decision problems L that can be
  - Nondeterministically accepted in polynomial time
  - Or is the set of all decision problems for which the instances where the answer is "yes" have efficiently verifiable proofs of the fact that the answer is indeed "yes"
    - Proofs verified in polynomial time
- Complement of L need not be in NP

Polynomial Time Verifiability

- If a solution to a decision solution can be verified if it outputs true in polynomial time, then the problem is polynomial time verifiable
  - Guess a solution to the given problem and design an algorithm to check if the output is true or false
- NP is set of all decision problems can be verified in polynomial time

Hamiltonian Cycle is in NP

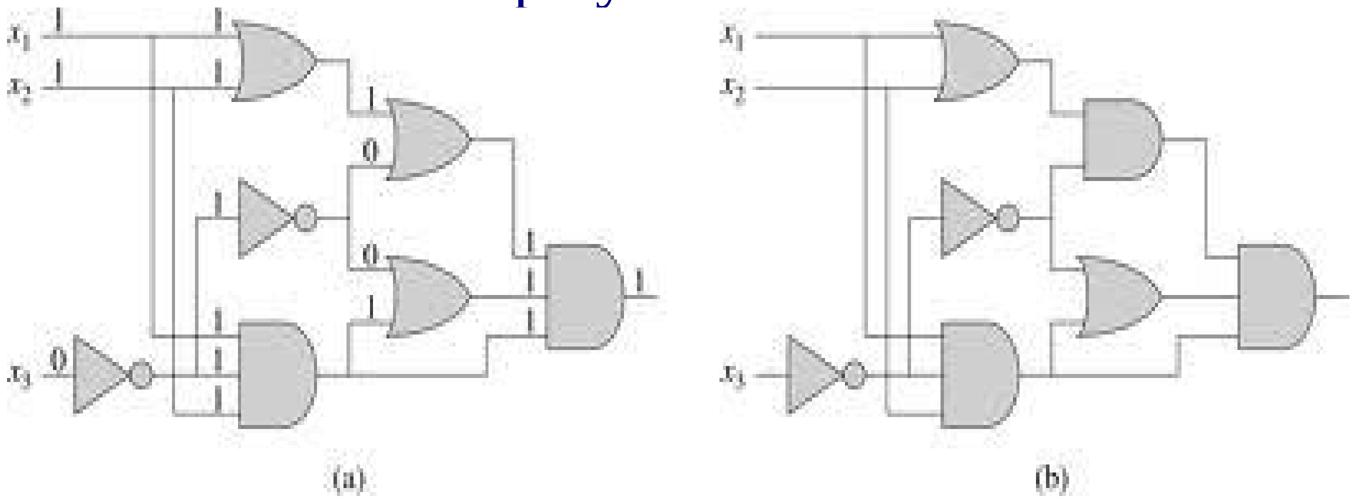
- Takes a graph G as input and asks if there is a simple cycle in G that visits each vertex exactly once returning to the start vertex
  - Cycle called Hamiltonian cycle
  - Take nodes in order from 1 to N returning to 1,
    - Check they occur exactly once, and forms a cycle in G, if yes outputs 1 – takes polynomial time



http://mathworld.wolfram.com/images/eps-gif/HamiltonianPlatonicCycles_751.gif

Circuit SAT

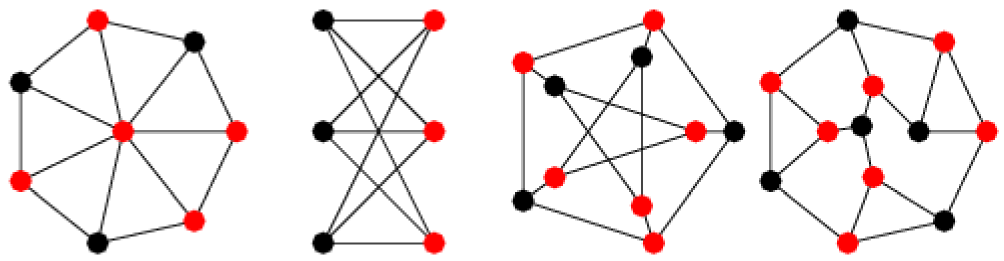
- Takes as input a Boolean circuit with single output node and asks whether there is an assignment of values to the inputs so that its output is 1
  - Guess some inputs and check the output is 1
    - Takes polynomial time to calculate



https://www.cs.indiana.edu/~achauhan/Teaching/B403/LectureNotes/images/12-circuitsat.jpg

Vertex Cover is in NP

- Takes graph G and integer k and asks if there is a vertex cover for G containing atmost k vertices
  - Select a collection C of k vertices
  - Examine each edge in G and check if one of the end points in C,
    - Takes polynomial time to verify



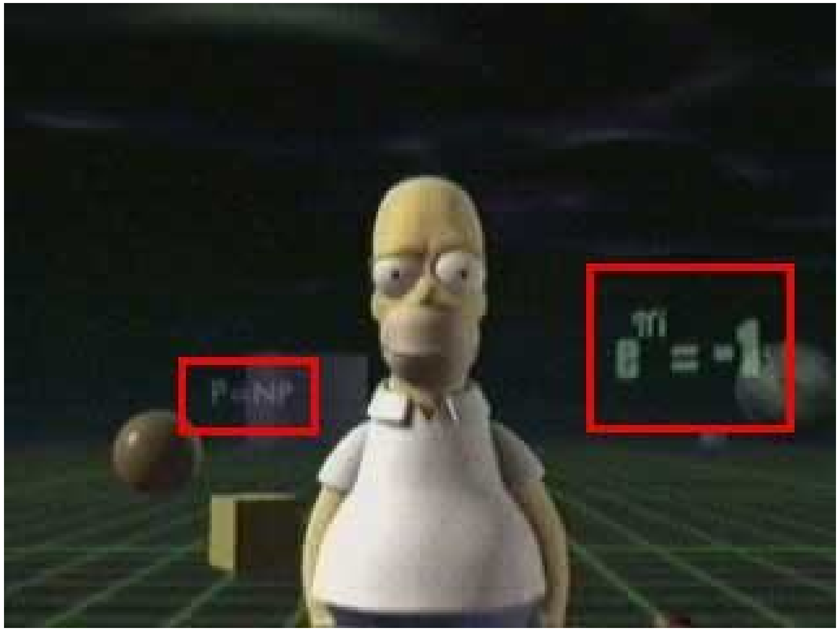
http://mathworld.wolfram.com/images/eps-gif/VertexCover_1000.gif

Exercise: Show they are in NP

- Traveling Salesman Problem:
  - given n vertices 1, . . . , n and all $n(n − 1)/2$ distances between them,and integer k, is there a cycle that passes through every vertex exactly once, of total cost k or less
- Longest Path Problem
  - given a graph G with nonnegative edge weights and two vertices s and t, and integer k is there a path from s to t with total weight at least k

P = NP Question

- Not yet known if P=NP
  - Most believe they are not equal
  - Believe P is different than NP and co-NP



Polynomial Time Reducibility

- The formal definition of NP-completeness uses reductions or transformations from one problem to another
- Consider a problem L that we want to solve, and we have an algorithm for another problem M
  - There is a function f that takes an input x of L and transforms it to an input f(x) of M such that the correct answer for L on x is yes iff the correct answer for M on f(x) is yes.
  - By composing f and the algorithm for M, an algorithm for L can be designed

Polynomial Time Reducibility

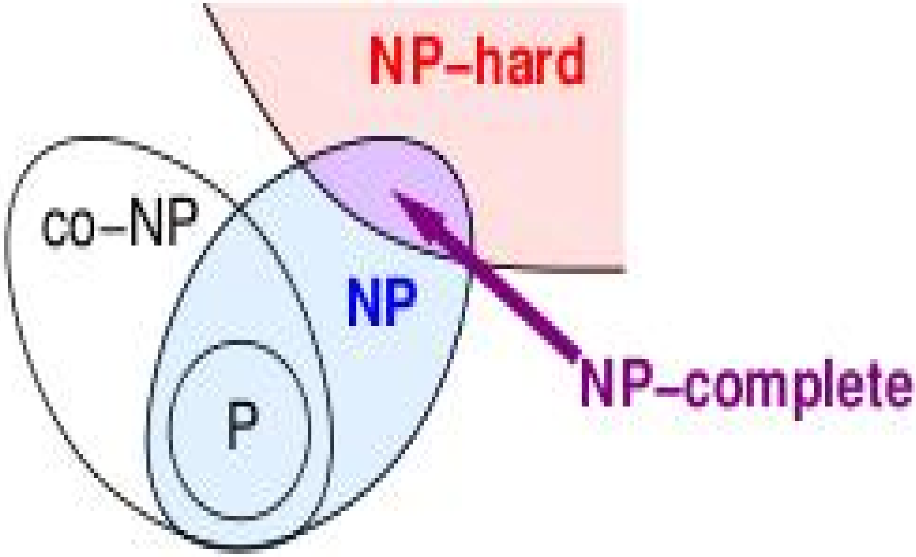
- Let f be a function from input set for L into input set for M. f is a polynomial reduction if
  - f can be computed in polynomial time
  - For every string x, if x is a yes input for L, then f(x) is a yes input for M
  - For every string x, if x is a no for L, then f(x) is a no input for M

NP Hard and NP Complete

- A decision problem M is **NP-Hard** if every other problem L in NP is polynomial time reducible to M
  - M is atleast as hard as any other problem in NP
- A decision problem M is **NP-Complete** if *M is in NP* and *M is NP-Hard*
  - ie one of the toughest problems in NP
- If an NP-Complete problem is solvable in polynomial time, then every other problem in NP is solvable in polynomial time

Is P = NP?

- An unanswered question



Cook Levine Theorem

- Circuit SAT is NP-Complete
  - We have shown Circuit SAT is in NP
  - Prove Circuit SAT is NP-Hard
    - For this we need to reduce known NP-Hard problem to the given problem
    - But we only know this problem
  - It can be proved that every problem in NP can be reduced to a circuit SAT in polynomial time
    - Every problem at each step computes and creates and solves a Boolean formula
    - Covert it into a circuit and ask if it is satisfiable

Problem Reduction

- A problem (language) L is NP-hard if every problem in NP is polynomial-time reducible to L.
- A problem (language) is NP-complete if it is in NP and it is NP-hard.
- CIRCUIT-SAT is NP-complete:
  - CIRCUIT-SAT is in NP
  - For every M in NP, M⇒CIRCUIT-SAT in polynomial time

CNF-SAT

- CNF- Conjunctive Normal Form
  - Collection of subexpressions called clauses that are combined using AND
  - Each clause formed as the OR of Boolean variables or their negatives (literals)
  - e.g
    - $(a+b+\neg d+e)(\neg a+\neg c)(\neg b+c+d+e)(a+\neg c+\neg e)$
    - OR: +, AND: (times), NOT: ¬
- SAT: Given a Boolean formula S, is S satisfiable, that is, can we assign 0's and 1's to the variables so that S is 1 ("true")?

CNF-SAT is NP-Complete

- CNF-SAT is in NP
    - Non-deterministically choose an assignment of 0's and 1's to the variables and then evaluate each clause. If they are all 1 ("true"), then the formula is satisfiable.
- Prove it is NP-Hard
    - Reduce Circuit SAT to CNF-SAT
        - Given Boolean circuit make variable for every input and gate
        - Create sub-formula for each gate and form the formula as the output variable AND-ed with all these sub-formulas
        - The formula is satisfiable iff the Boolean circuit m is satisfiable

CNF-SAT

- Construction of formula S equivalent to C
  - Create variable x_i for each input to C
  - Create variable y_i for each output of a gate in C
  - Create a formula B_g corresponding to each gate g
    - g- AND gate: $B_g = (c \leftrightarrow a.b)$
    - g – OR gate $B_g = (c \leftrightarrow a+b)$
    - g – NOT gate $B_g = (b \leftrightarrow \neg a)$
    - Convert each B_g to be in CNF and combine these formulas by AND to get a CNF formula

CNF Construction

- draw a truth table for every gate and write down the CNF formula
  - $c \leftrightarrow a.b$

c	a	b	$c \leftrightarrow a.b$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	0	1	1
0	1	0	1
0	0	0	1

  - write down equivalent disjunctive normal form (DNF) formula for all true-table items evaluating to 0
    - $(c.a.\neg b)+(c.\neg a.b)+(c.\neg a.\neg b)+(\neg c.a.b)$
  - Use Demorgan's Law to get CNF
    - $(\neg c+\neg a+b)(\neg c+a+\neg b)(\neg c+a+b)(c+\neg a+\neg b)$

3-SAT

- The problem
  - Takes a Boolean formula in CNF form where each clause has exactly three literals and asks if it is satisfiable
  - e.g $(\neg x_1+x_2+x_3)(x_2+\neg x_3+x_4)(x_1+\neg x_2+\neg x_4)$
- It is a restricted version of CNF-SAT problem
- 3-SAT is NP-Complete
  - However 2-SAT is solvable in polynomial time

3-SAT is NP Complete

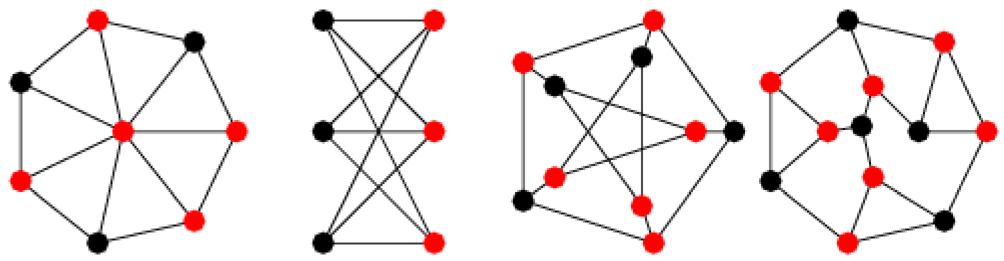
- Prove 3-SAT is in NP
  - Construct a nondeterministic polynomial time algorithm that takes a CNF formula S with 3 literals per clause and evaluates S to see if it is 1
- Prove 3-SAT is NP Hard
  - Reduce CNF-SAT to 3-SAT
    - Reduce a CNF-SAT formula C and convert it to 3-SAT form S
    - S is satisfiable iff the corresponding CNF-SAT C is satisfiable and vice versa

Converting CNF to 3-CNF

- Perform the local replacement for each clause C_i in C
  - If $C_i = (a)$ ie., has one term, replace C_i by $S_i = (a+b+c)(a+\neg b+c)(a+b+\neg c)(a+\neg b+\neg c)$, where b and c not used anywhere else
  - If $C_i = (a+b)$ ie., has two terms, replace C_i by $S_i = (a+b+c)(a+b+\neg c)$, where c not used anywhere else
  - If $C_i = (a_1+a_2+\ldots+a_k)$ ie., has k terms, replace C_i by $S_i = (a_1+a_2+b_1)(\neg b_1+a_3+b_2)(\neg b_2+a_4+b_3)\ldots(\neg b_{k-3}+a_{k-1}+a_k)$, where $b_1,b_2,\ldots,b_k$ not used anywhere else
- Clause C_i is 1 iff S_i is also 1

Some Interesting Graph Problems

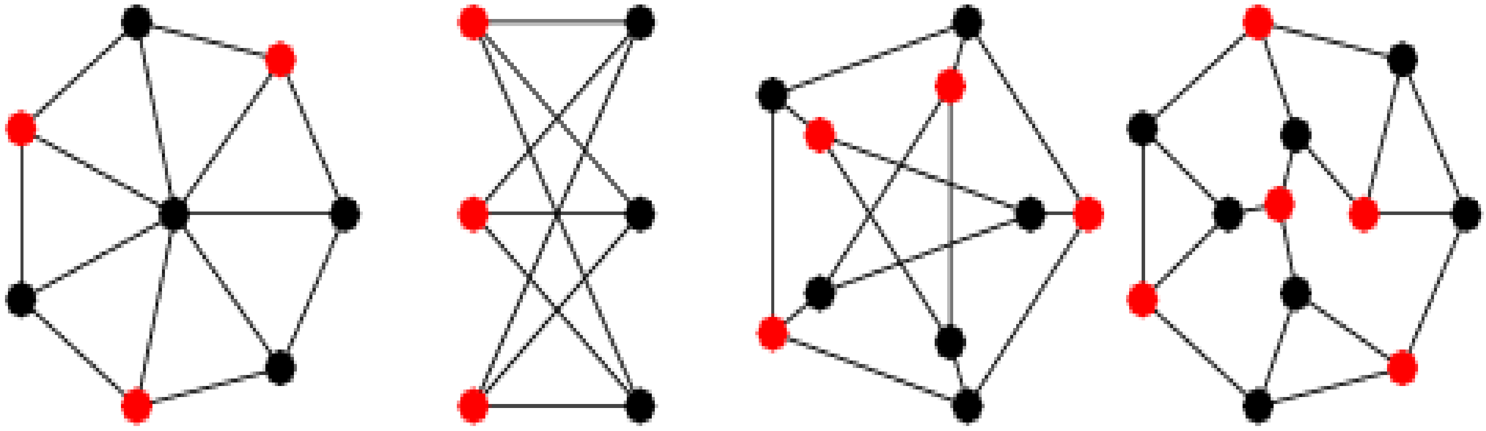
- Vertex Cover
  - Set of vertices such that each edge of the graph is incident to at least one vertex of the set
- Minimum Vertex Cover
  - Takes graph G and integer k and asks if there is a vertex cover for G containing atmost k vertices



http://mathworld.wolfram.com/images/eps-gif/VertexCover_1000.gif

Independent Set

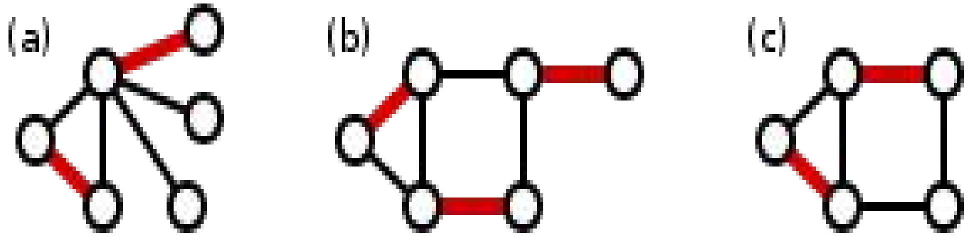
- Independent Set
  - set of vertices in a graph s.t that no two vertices in the set are adjacent
- Maximum Independent Set
  - is a largest independent set for a given graph G and its size is denoted by α(G)



http://mathworld.wolfram.com/images/eps-gif/IndependentSet_900.gif

Matching

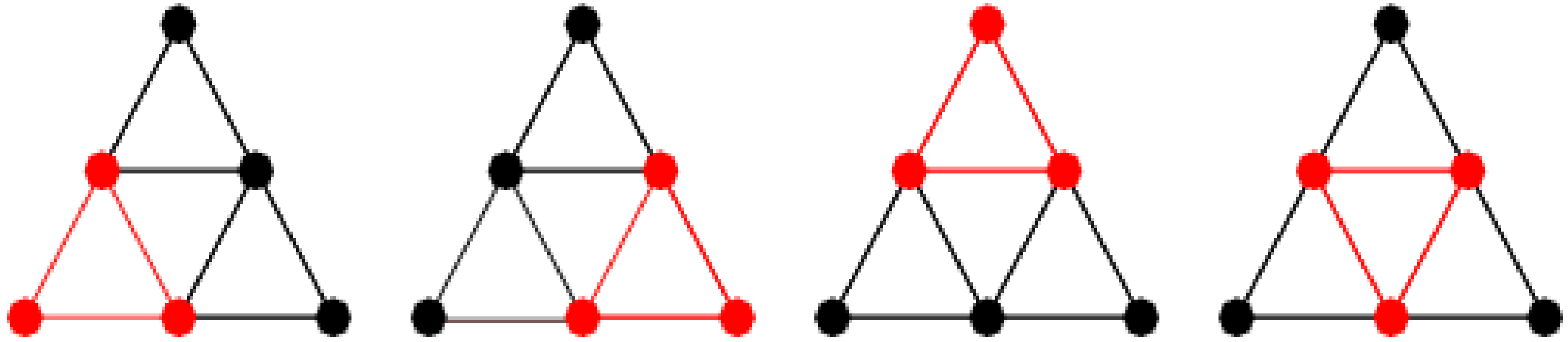
- Matching
  - set of edges s.t no two edges in the set are adjacent
- Maximum Matching
  - Given a graph G and integer k, is there a matching of size atleast k
  - Perfect Matching: every vertex of the graph is incident to exactly one edge of the matching



http://upload.wikimedia.org/wikipedia/commons/thumb/9/98/Maximum-matching-labels.svg/300px-Maximum-matching-labels.svg.png

Clique

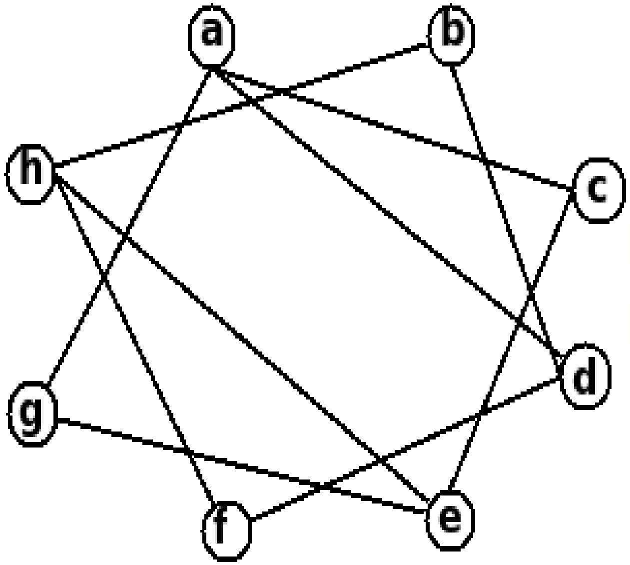
- Clique
  - subset of its vertices such that every two vertices in the subset are connected by an edge
- Maximum Clique
  - Given graph G, and integer k, is there a clique in G with atleast k vertices



http://mathworld.wolfram.com/images/eps-gif/Clique_950.gif

Exercise

- For the following graph
  - Find a vertex cover, edge cover, matching, independent set and clique
  - Find the
    - Minimum vertex cover
    - Maximum matching
    - Maximum Independent Set
    - Maximum Clique



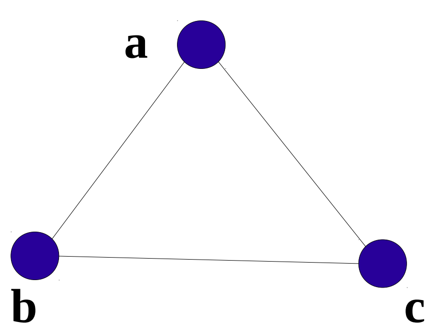
Interesting Properties

- A set of vertices is a vertex cover, if and only if its complement is an independent set
  - number of vertices of a graph is equal to its vertex cover number plus the size of a maximum independent set
- A set is independent if and only if it is a clique in the graph’s complement
  - Edges in complement when there is no edge in G
- A perfect matching is always a minimum edge covering.

Vertex Cover is NP-Complete

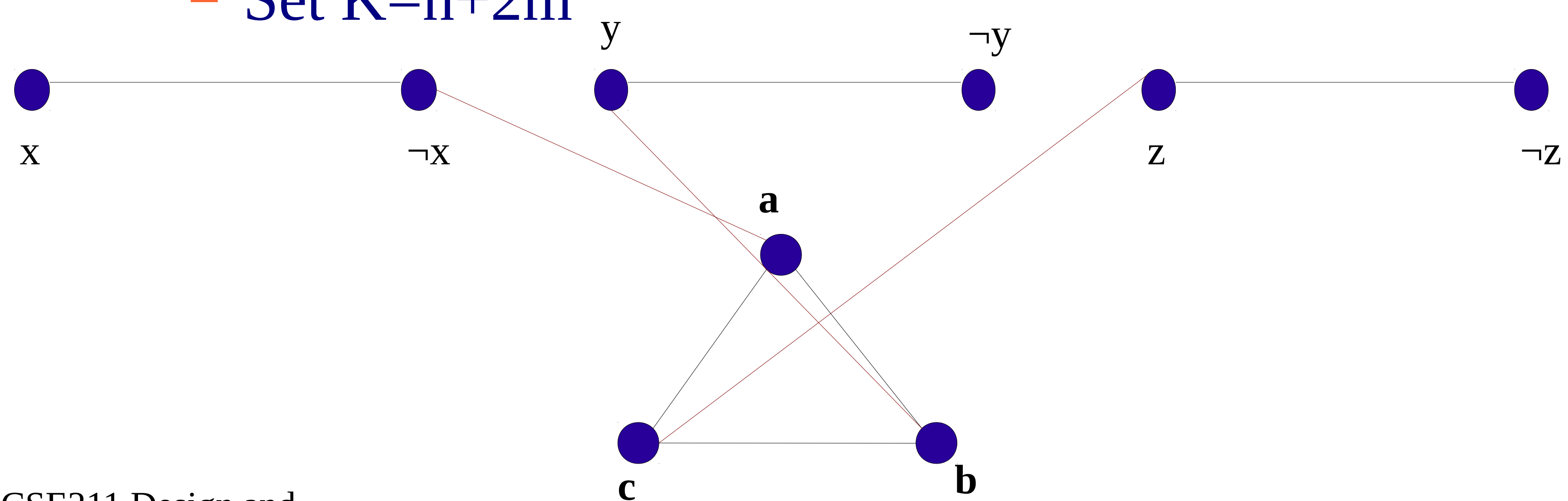
- Show Vertex Cover is in NP
- Reduce 3SAT to Vertex Cover
  - Let S be a Boolean formula in CNF with each clause having 3 literals.
  - For each variable x, create a node for x and ¬x, and connect these two:

  - For each clause (a+b+c), create a triangle and connect these three nodes.



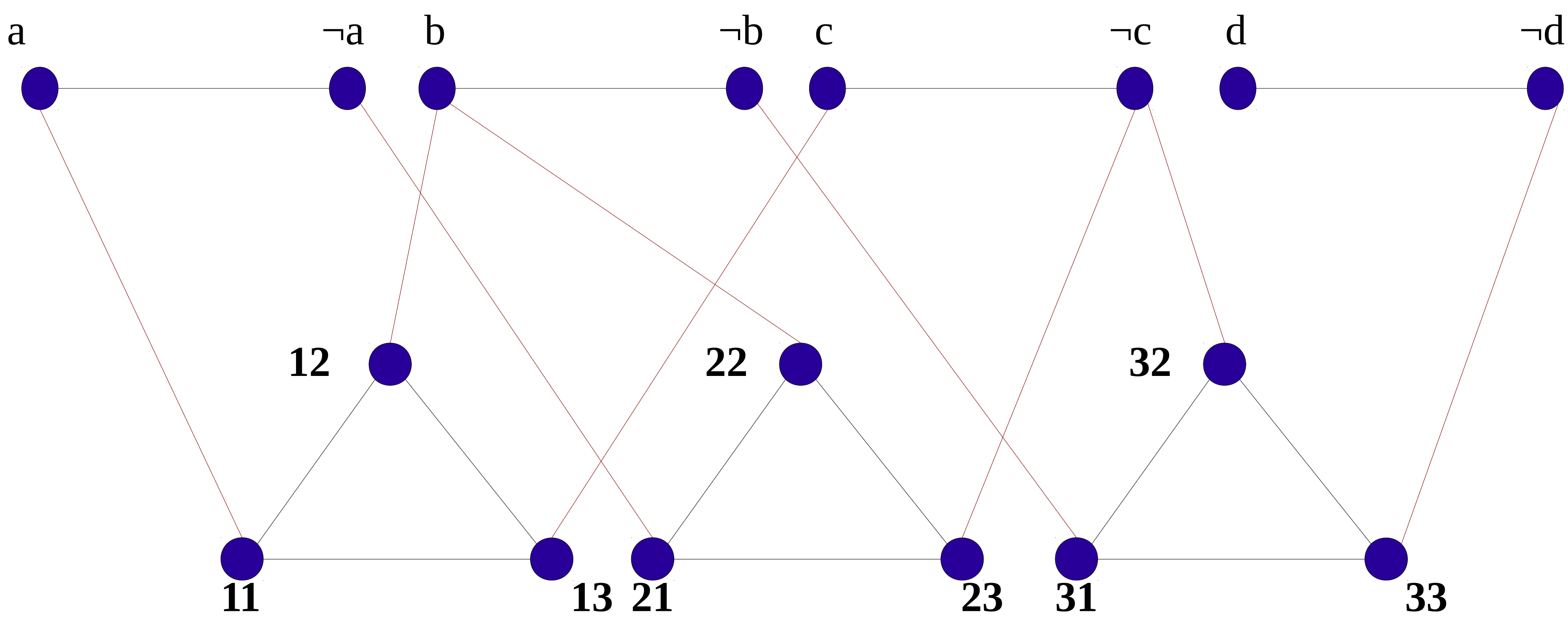
Vertex Cover is NP-Complete

- Completing the construction
  - Connect each literal in a clause triangle to its copy in a variable pair.
    - E.g., a clause (¬x+y+z)
- Let n=# of variables, Let m=# of clauses
  - Set K=n+2m



Vertex Cover is NP-Complete

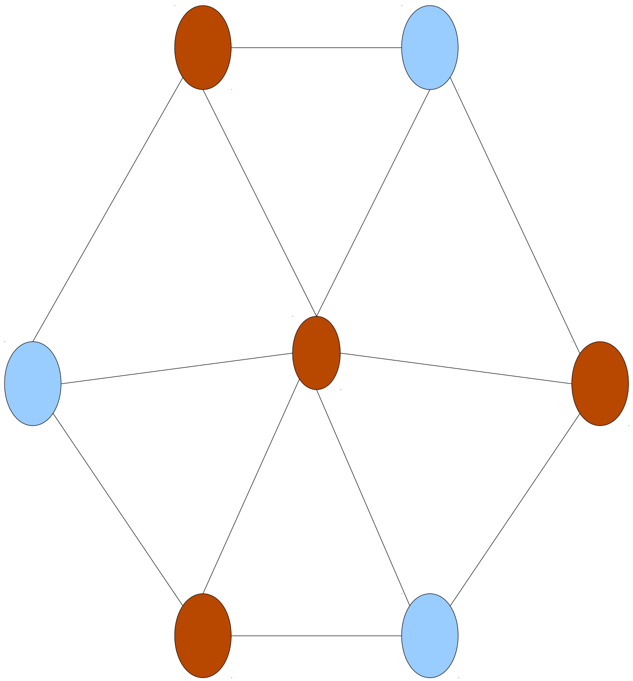
- Graph has vertex cover of size K iff formula is satisfiable.
  - e.g (a+b+c)(¬a+b+¬c)(¬b+¬c+¬d)
  - Here k=4+6=10

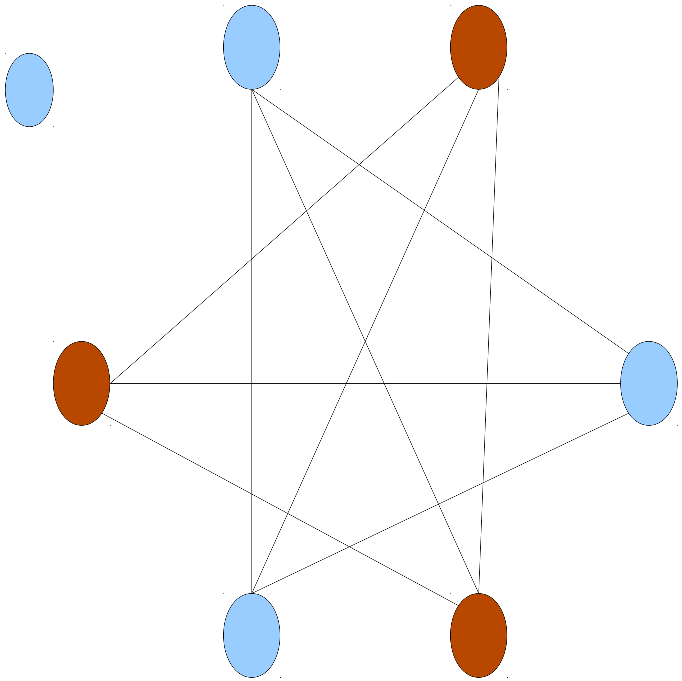


Max Clique is NP-Complete

- Show Max Clique problem is in NP
- Prove it is NP-Hard
  - Reduce Vertex Cover problem (G,k) to it
  - Construct G^c – the complementary graph of G
  - G^c has a clique of size atleast n-k if and only if G has a vertex cover of size atmost k
  - Hence Max Clique is NP Hard
- Max Clique problem is NP-Complete!

Max Clique is NP Complete





Exercise

- Referring to existing proofs (other than that given in Goodrich or Cormen)
  - Prove Maximum Independent Set is NP-Complete
  - Prove Hamiltonian Cycle is NP-Complete
  - Prove Set-Cover Problem is NP-Complete

Other NP Complete Problems

- Subset Sum
  - Given a set of integers and a distinguished integer K, is there a subset of the integers that sums to K?
  - Proof by reduction from Vertex Cover
- 0/1 Knapsack
  - Given a collection of items with weights and benefits, is there a subset of weight at most W and benefit at least K?
  - Proof by reduction Subset-Sum problem

Other NP-Complete Problems

- Hamiltonian-Cycle
  - Given an graph G, is there a cycle in G that visits each vertex exactly once?
- Traveling Salesman Problem
  - Given a complete weighted graph G, is there a cycle that visits each vertex and has total cost at most K?
  - Proof: reduction from Hamiltonian-Cycle
    - It contains Hamiltonian cycle problem as special case
    - Set cost of edges as 1, and k=number of vertices in G