NP HARD DAN NP COMPLETE

ANALISIS ALGORITMA DAN KOMPLEKSITAS

DECISION PROBLEMS VS. OPTIMIZATION PROBLEMS

- **The problems to solve are basically of two kinds.**
  - In *decision problems* we are trying to decide whether a statement is true or false.
  - In *optimization problems* we are trying to find the solution with the best possible score according to some scoring scheme.
- **Optimization problems can be either:**
  - *maximization problems*, where we are trying to maximize a certain score, or
  - *minimization problems*, where we are trying to minimize a cost function.

EXAMPLE 1: HAMILTONIAN CYCLES

- **Given a directed graph, we want to decide whether or not there is a Hamiltonian cycle in this graph.**
- **This is a decision problem.**

EXAMPLE 2: TSP - THE TRAVELING SALESMAN PROBLEM

- **Given a complete graph and an assignment of weights to the edges, find a Hamiltonian cycle of minimum weight.**
- **This is the *optimization version* of the problem.**
- **In the *decision version*, we are given a weighted complete graph and a real number c, and we want to know whether or not there exists a Hamiltonian cycle whose combined weight of edges does not exceed c.**

EXAMPLE 3: PAIRWISE SEQUENCE ALIGNMENT, OPTIMIZATION VERSION

- **Pairwise sequence alignment methods are used to find the best-matching of two query sequences.**
- **Given two sequences and a scoring scheme, find the highest-scoring alignment of these sequences.**
- **This is an optimization problem.**
- **Similarly for multiple sequence alignment.**

EXAMPLE 3: PAIRWISE SEQUENCE ALIGNMENT, DECISION VERSION

- **Given two sequences, a scoring scheme, and a cutoff score c, decide whether there exists an alignment with a score that is higher than c.**
- **Similarly for multiple sequence alignment.**

IMPORTANT OBSERVATION:

EACH OPTIMIZATION PROBLEM HAS A CORRESPONDING DECISION PROBLEM.

INPUTS

- **Each of the problems discussed above has its characteristic input.**
  - For example, for the optimization version of the TSP, the input consists of a weighted complete graph;
  - For the decision version of the TSP, the input consists of a weighted complete graph and a real number.
- **The input data of a problem are often called the *instance* of the problem.**
  - Each instance has a characteristic *size;* which is the amount of computer memory needed to describe the instance.

THE CLASS P

- **A decision problem *D* is *solvable in polynomial time* or *in the class P*, if there exists an algorithm *A* such that**
  - *A* takes instances of *D* as inputs.
  - *A* always outputs the correct answer "Yes" or "No".
  - There exists a polynomial *p* such that the execution of *A* on inputs of size *n* always terminates in *p(n)* or fewer steps.
- **The class P consists of those problems that are solvable in polynomial time.**
- **More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem**

THE CLASS NP

- **NP is not the same as non-polynomial complexity/ running time.**
  - NP does not stand for not polynomial.
- **NP = Non-Deterministic polynomial time**
- **NP means verifiable in polynomial time by some non deterministic Turing machine**
  - If we are given a solution we can verify the legitimacy in polynomial time

THE RELATION WITH AUTOMATA

- **Problem is in NP iff it is decidable by some non deterministic Turing machine in polynomial time.**
- **It is provable (not discussed in this lecture) that a Non Deterministic Turing Machine is equivalent to a Deterministic Turing Machine**
- **The deterministic version of a poly time non deterministic Turing machine will run in exponential time (worst case)**

HAMILTONIAN CYCLES

- **Determining whether a directed graph has a Hamiltonian cycle does not have a polynomial time algorithm**
- **However if someone was to give you a sequence of vertices, determining whether or not that sequence forms a Hamiltonian cycle can be done in polynomial time**
- **Therefore Hamiltonian cycles are in NP**

SAT

- **A boolean formula is *satisfiable* if there exists some assignment of the values 0 and 1 to its variables that causes it to evaluate to 1.**
- **CNF – Conjunctive Normal Form: ANDing of clauses of ORs**

$$(x_0 \vee x_2) \wedge (\neg x_0 \vee x_1) \wedge (x_0 \vee x_1 \vee \neg x_2)$$

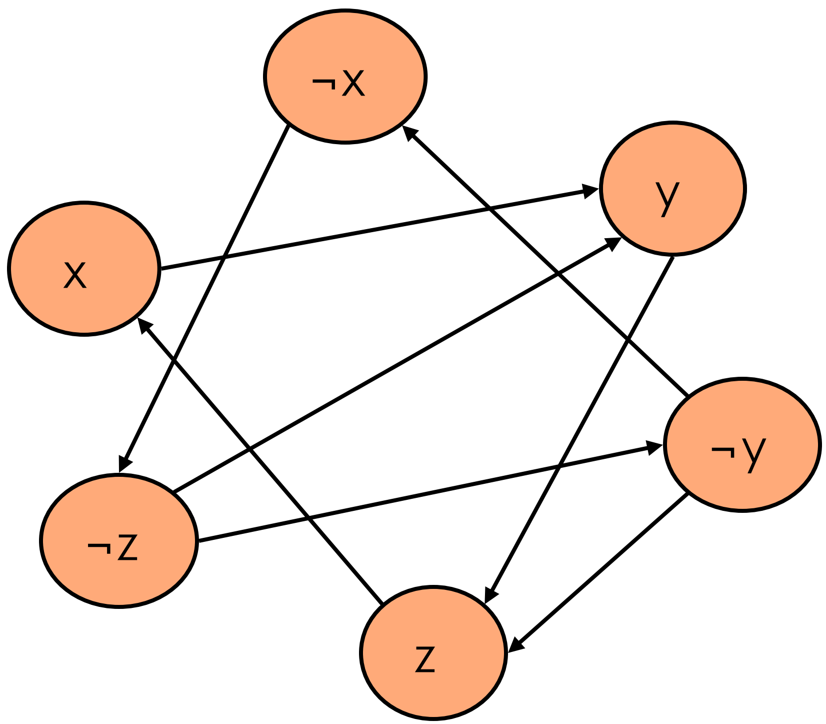
2-CNF SAT

- **Each or operation has two arguments that are either variables or negation of variables**
- **The problem in 2 CNF SAT is to find true/false (0 or 1) assignments to the variables in order to make the entire formula true.**
- **Any of the OR clauses can be converted to implication clauses**

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

2-SAT IS IN P

- **Create the implication graph**
  - $(p \vee q) \Leftrightarrow (\neg p \rightarrow q) \Leftrightarrow (\neg q \rightarrow p)$



$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$

$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$

SATISFIABILITY VIA PATH FINDING

- **If there is a path from** $x \text{ to } \neg x$
- **And if there is a path from** $\neg x \text{ to } x$
- **Then FAIL!**
- **How to find paths in graphs?**
  - DFS/BFS and modifications thereof

3 CNF SAT (3 SAT)

- **Determining the satisfiability of a formula in conjunctive normal form where each clause is limited to at most three literals**
- **Not so easy anymore.**
- **Implication graph cannot be constructed**
- **No known polytime algorithm**
- **Is it NP?**
  - If someone gives you a solution how long does it take to verify it?
  - Make one pass through the formula and check
- **This is an NP problem**

P = NP?

- **P is a subset of NP**
  - Since it takes polynomial time to run the program, just run the program and get a solution → polynomial time to verify the solution
- **But is NP a subset of P?**
- **No one knows if P = NP or not**
- **Solve for a million dollars!**
  - http://www.claymath.org/millennium-problems
  - The Poincare conjecture is solved today

AMUSING ANALOGY

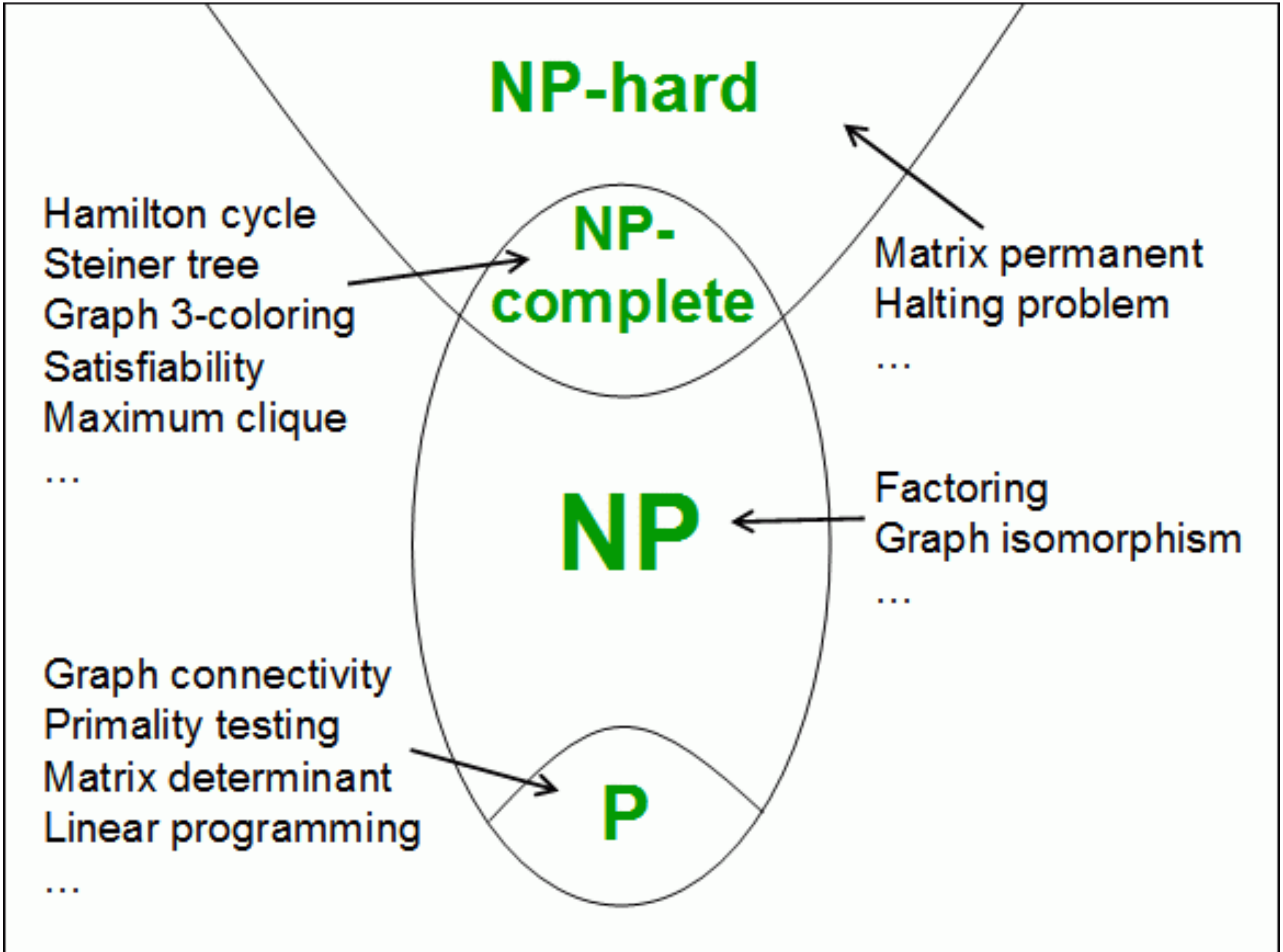
- **Students believe that every problem assigned to them is NP-complete in difficulty level, as they have to find the solutions.**
- **Teaching Assistants, on the other hand, find that their job is only as hard as NP, as they only have to verify the student's answers.**
- **When some students confound the TAs, even verification becomes hard**

WHAT IS NOT IN NP?

- **Undecidable problems**
- **Example: Given a polynomial with integer coefficients, does it have integer roots**
  - Hilbert's 10^{th} problem
  - Impossible to check for all the integers
  - Even a non-deterministic TM has to have a finite number of states!
  - → undecidable problem

CLASSES OF NP

- **NP:** Class of computational problems for which a given solution can be verified as a solution in polynomial time by a non-deterministic Turing machine (or *solvable* by a *non-deterministic* Turing machine in polynomial time).
- **NP-hard:** Class of problems which are at least as hard as the hardest problems in NP. Problems that are NP-hard do not have to be elements of NP; indeed, they may not even be decidable.
- **NP-complete:** Class of problems which contains the hardest problems in NP. Each NP-complete problem has to be in NP.
- **NP-easy:** At most as hard as NP, but not necessarily in NP, since they may not be decision problems.
- **NP-equivalent:** Problems that are both NP-hard and NP-easy, but not necessarily in NP, since they may not be decision problems.
- **NP-intermediate:** If P and NP are different, then there exist problems in the region of NP that fall between P and the NP-complete problems. (If P and NP are the same class, then NP-intermediate problems do not exist.)


NP-hard
NP-complete
NP
P
Hamilton cycle
Steiner tree
Graph 3-coloring
Satisfiability
Maximum clique
…
Matrix permanent
Halting problem
…
Factoring
Graph isomorphism
…
Graph connectivity
Primality testing
Matrix determinant
Linear programming
…

REDUCIBILITY

- **A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephrased" as an instance of Q', the solution to which provides a solution to the instance of Q**
- **Is a linear equation reducible to a quadratic equation?**
  - Sure! Let coefficient of the square term be 0

REDUCIBILITY NATOTATION

$$L_1 \leq_p L_2$$

- **That notation means that L_1 is reducible in polynomial time to L_2 .**
- **The less than symbol basically means that the time taken to solve L_1 is no worse than a polynomial factor away from the time taken to solve L_2.**

NP-HARD

- **A problem (a language) is said to be NP-hard if every problem in NP can be poly time reduced to it.**

$$L' \leq_p L \text{ for every } L' \in NP$$

- **L is one of the hardest problems in NP.**

NP-COMPLETE

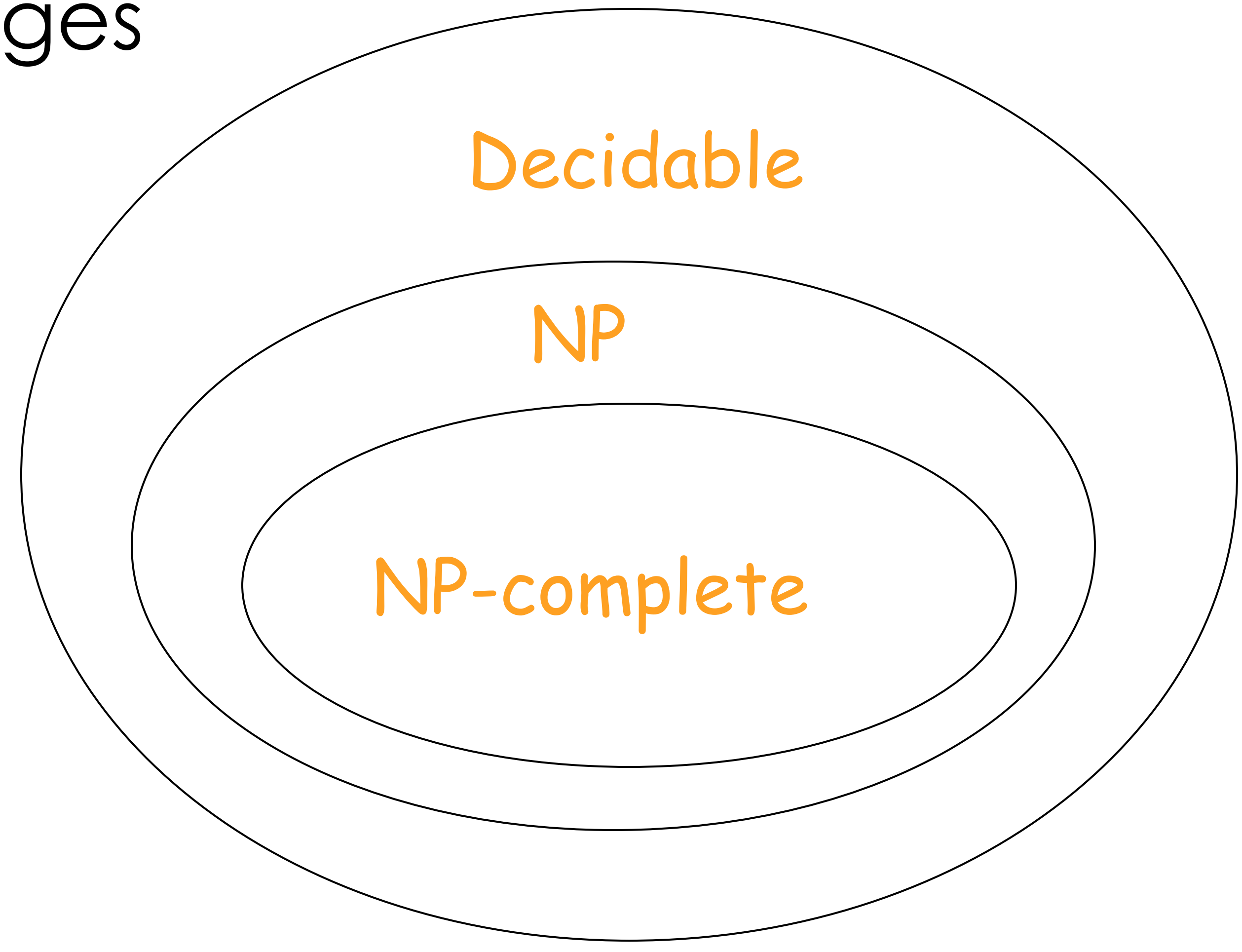
A language $\mathcal{L}$ is NP-complete if:

- $\mathcal{L}$ is in NP, and

- Every language in NP is reduced to $\mathcal{L}$ in polynomial time

CLASS OF NP-COMPLETE LANGUAGES

We define the class of NP-complete languages



NP COMPLETE PROBLEMS/ LANGUAGES

- **Requirements:**
  - Need to be in NP
  - Need to be in NP-Hard
- **If both are satisfied then it is an NP complete problem**
- **Reducibility is a transitive relation.**
- **If we know a single problem in NP-Complete, that helps when we are asked to prove some other problem is NP-Complete**
  - Assume problem P is NP Complete
  - All NP problems are reducible to this problem
  - Now given a different problem P’
  - If we show P reducible to P’
  - Then by transitivity all NP problems are reducible to P’

WHAT IS IN NP-COMPLETE

- **3-CNF SAT**
  - Actually any boolean formula can be reduced to 3-CNF form
- **Vertex cover**
- **Clique**
- **Hamiltonian circuit**
- **Partition problem**

The "Hardest" Sets in NP

Sudoku

Clique

SAT

Independent-Set

HAM

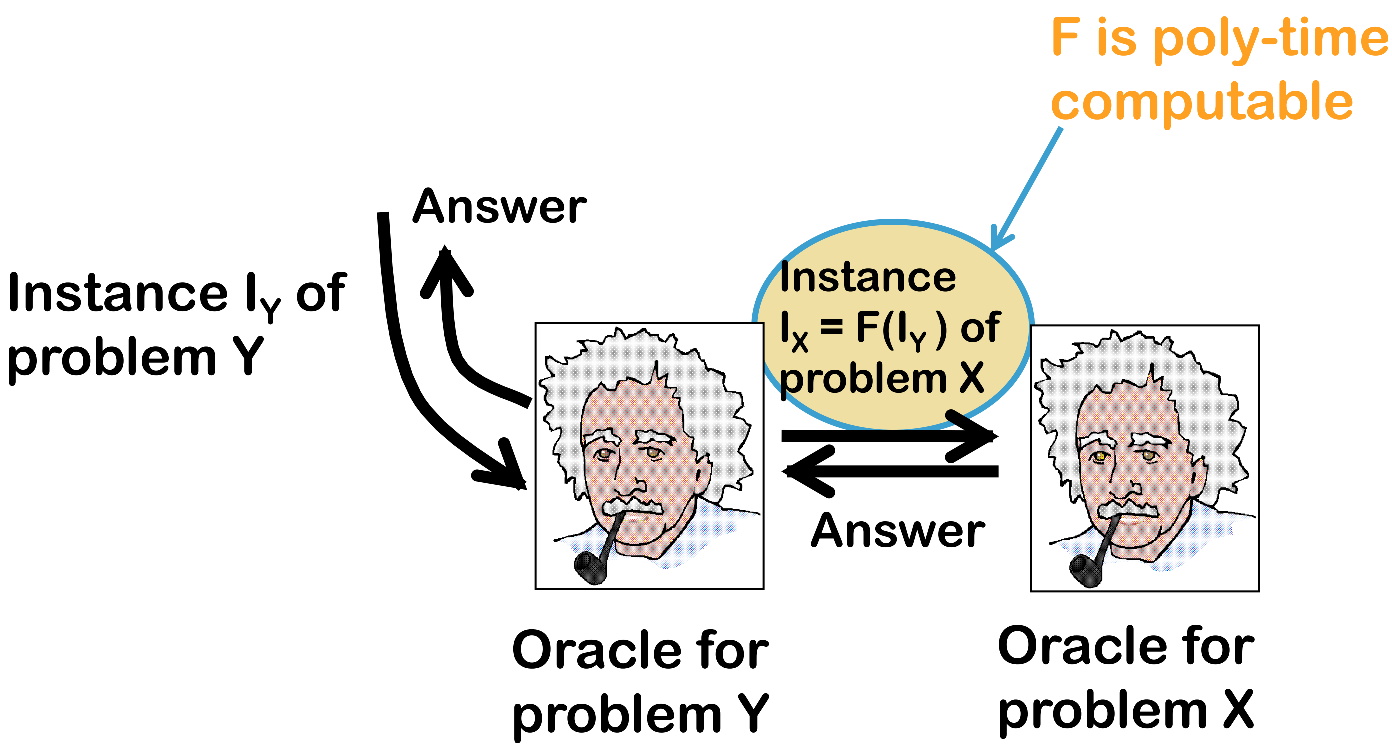
3-Colorability

These problems are all "polynomial-time equivalent".

i.e., each of these can be reduced to any of the others in poly-time

“Poly-time reducible to each other”

Reducing problem Y to problem X in poly-time



Theorem [Cook/Levin]:

SAT is one language in NP, such that if we can show SAT is in P, then we have shown NP $\subseteq$ P.

SAT is a language in NP that can capture all other languages in NP.

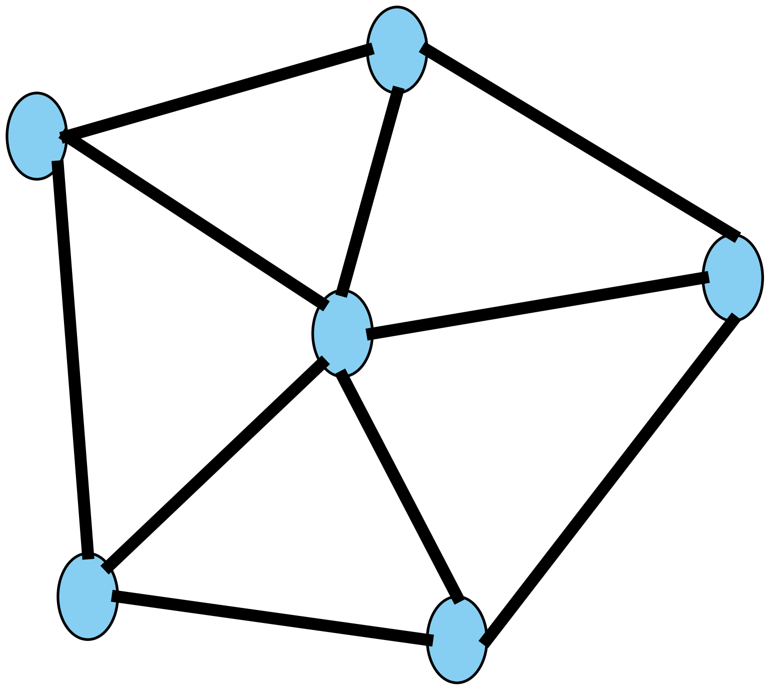
We say SAT is NP-complete.

SAT AND 3COLOR

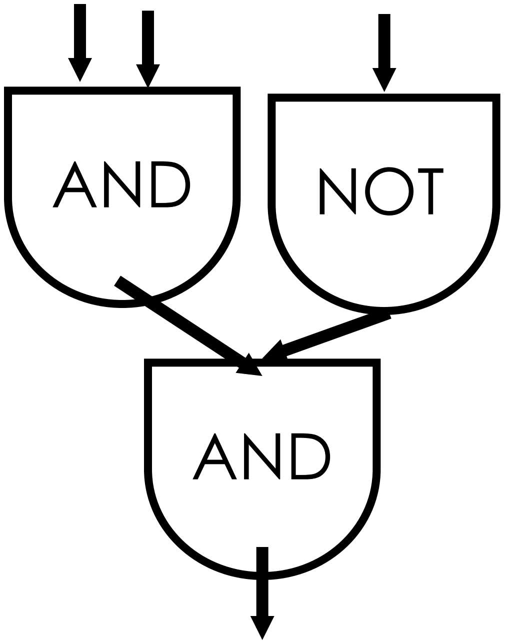
SAT and 3COLOR: Two problems that seem quite different, but are substantially the same.

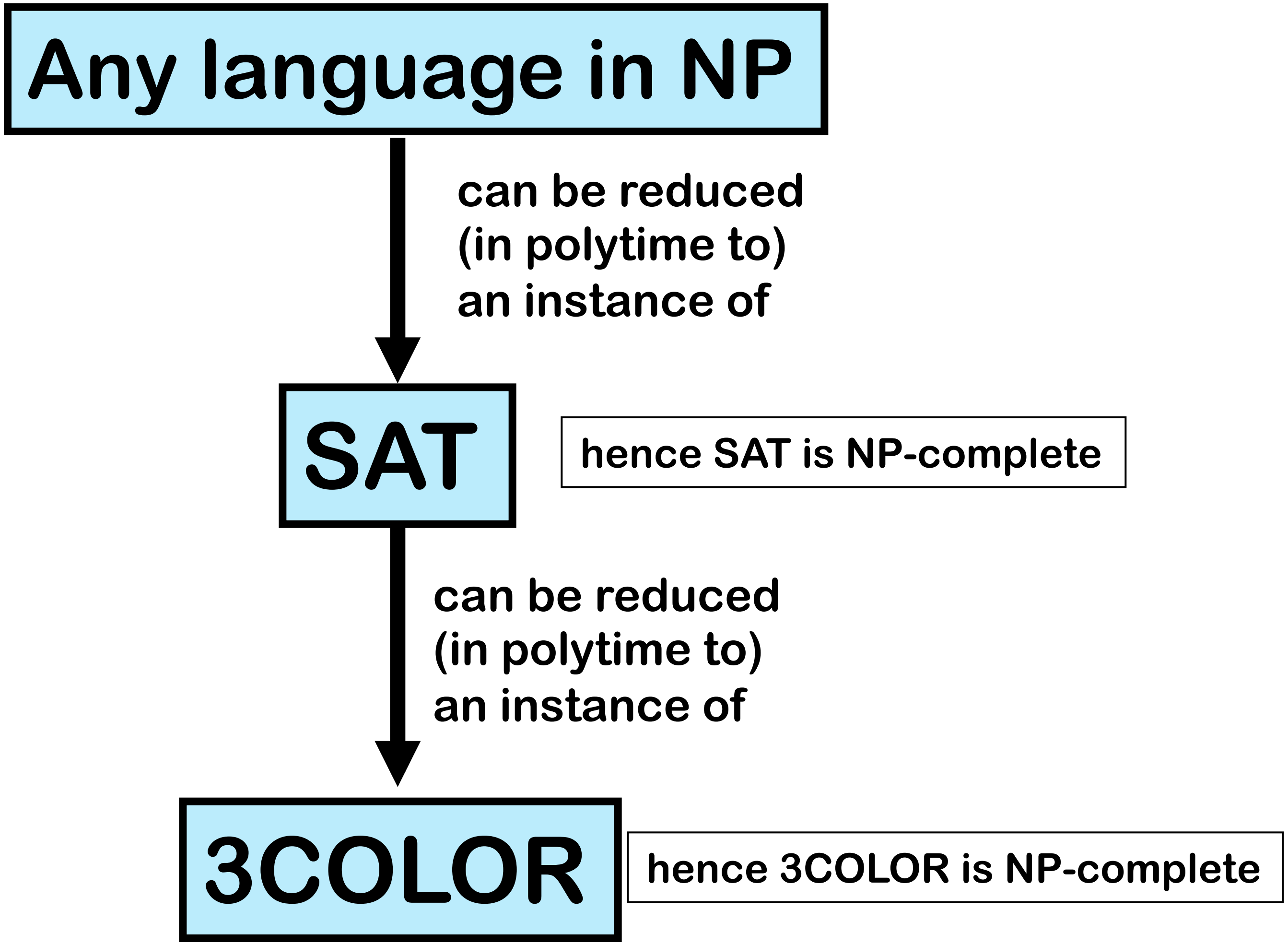
If you get a polynomial-time algorithm for one, you get a polynomial-time algorithm for ALL.

3-colorability



Circuit Satisfiability




Any language in NP
can be reduced
(in polytime to)
an instance of
SAT
hence SAT is NP-complete
can be reduced
(in polytime to)
an instance of
3COLOR
hence 3COLOR is NP-complete

AN EXAMPLE OF REDUCTION

- **CLIQUE problem**
- **A clique in an undirected graph is a subset of vertices such that each pair is connected by an edge (complete sub-graphs)**
- **We want to take a problem instance in 3-CNF SAT and convert it to CLIQUE finding.**

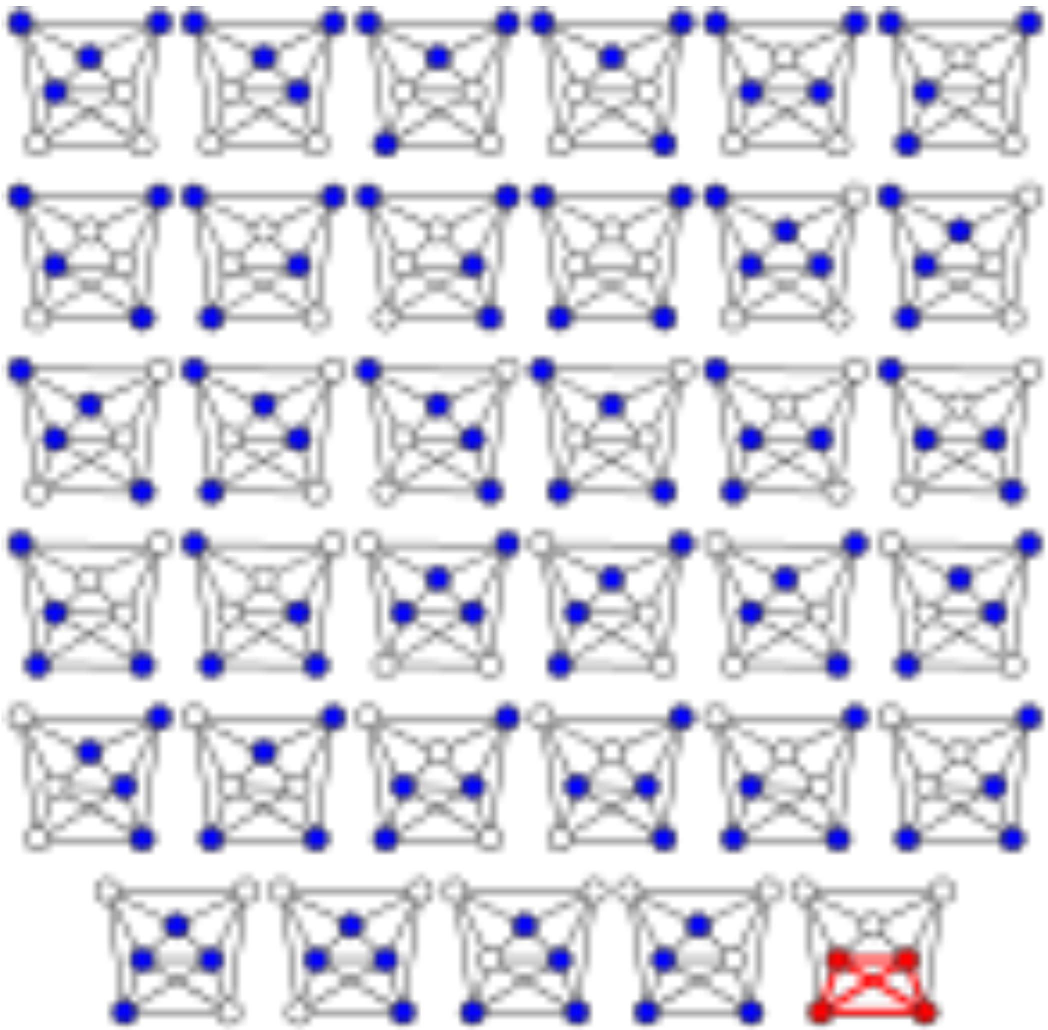
CLIQUE

- **An instance of a clique problem gives you 2 things as input**
  - Graph
  - Some positive integer k
- **Question being asked = do we have a clique of size k in this graph**

DECISION PROBLEMS VERSUS OPTIMIZATION PROBLEMS

- **Finding the maximum sized clique is an optimization problem**
- **But we can reduce it to a series of decision problems**
  - Can we find a clique of size k
  - Can we find a clique of size k-1
  - etc

CLIQUES IN A 7-VERTEX-GRAPH



REDUCING 3CNF SAT TO CLIQUE

- **Given – A boolean formula in 3 CNF SAT**
- **Goal – Produce a graph (in polynomial time) such that**

$$\text{Satisfiabiality} \Leftrightarrow \text{Clique of a certain size}$$

- **We will construct a graph where satisfying formula with k clauses is equivalent to finding a k vertex clique.**

3CNF formula:

literal variable or its complement

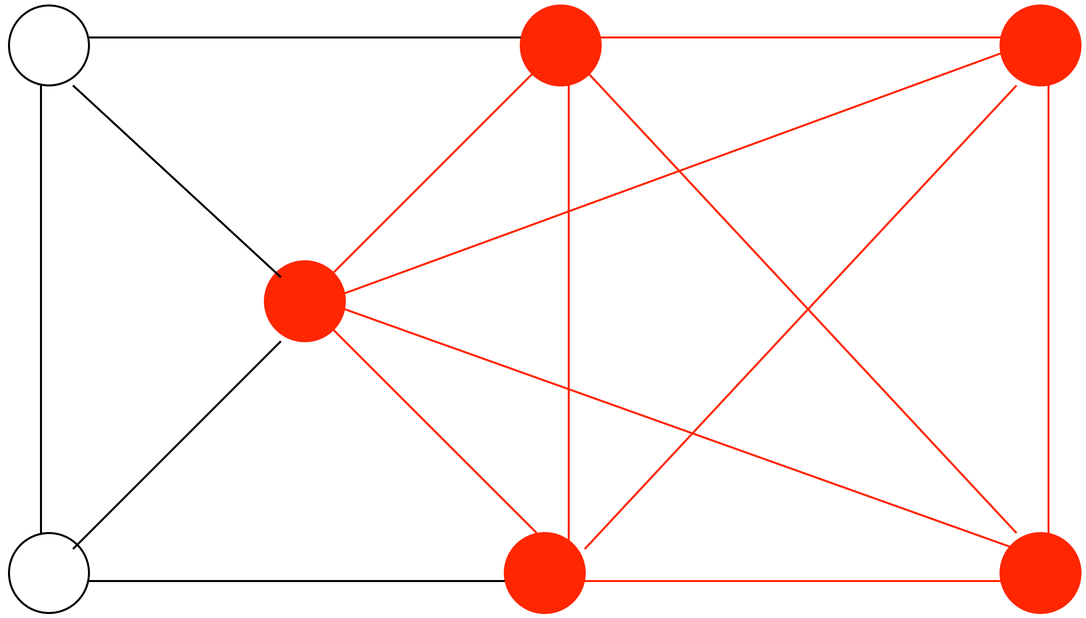
$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \underbrace{(x_3 \vee \overline{x_5} \vee x_6)}_{\text{clause}} \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

Each clause has three literals

Language:

3CNF-SAT ={ w : w is a satisfiable 3CNF formula}

A 5-clique in graph G



Language:

CLIQUE = $\{< G,k >$: graph G contains a k-clique$\}$

Theorem: 3CNF-SAT is polynomial time reducible to CLIQUE

Proof: give a polynomial time reduction of one problem to the other

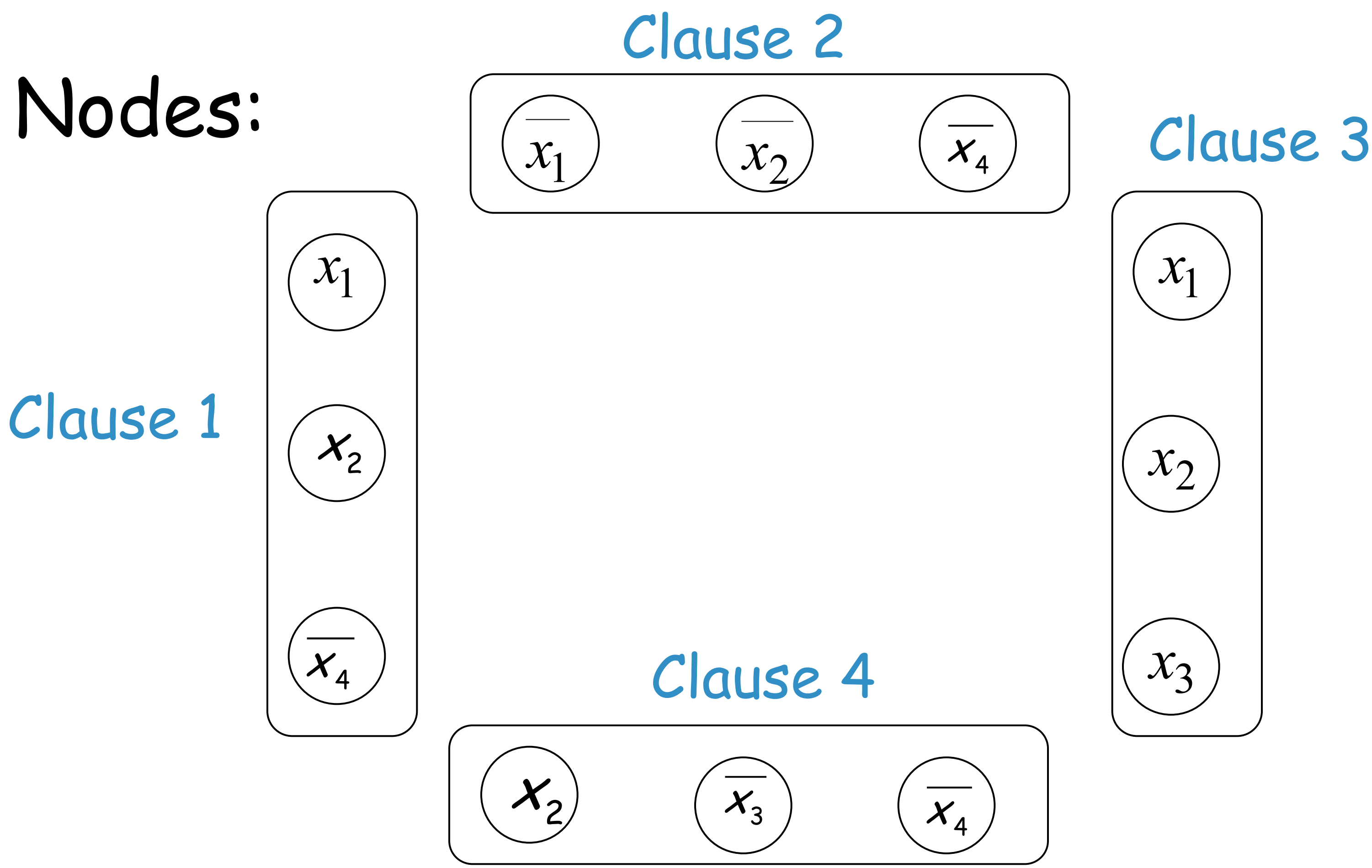
Transform formula to graph

Transform formula to graph.

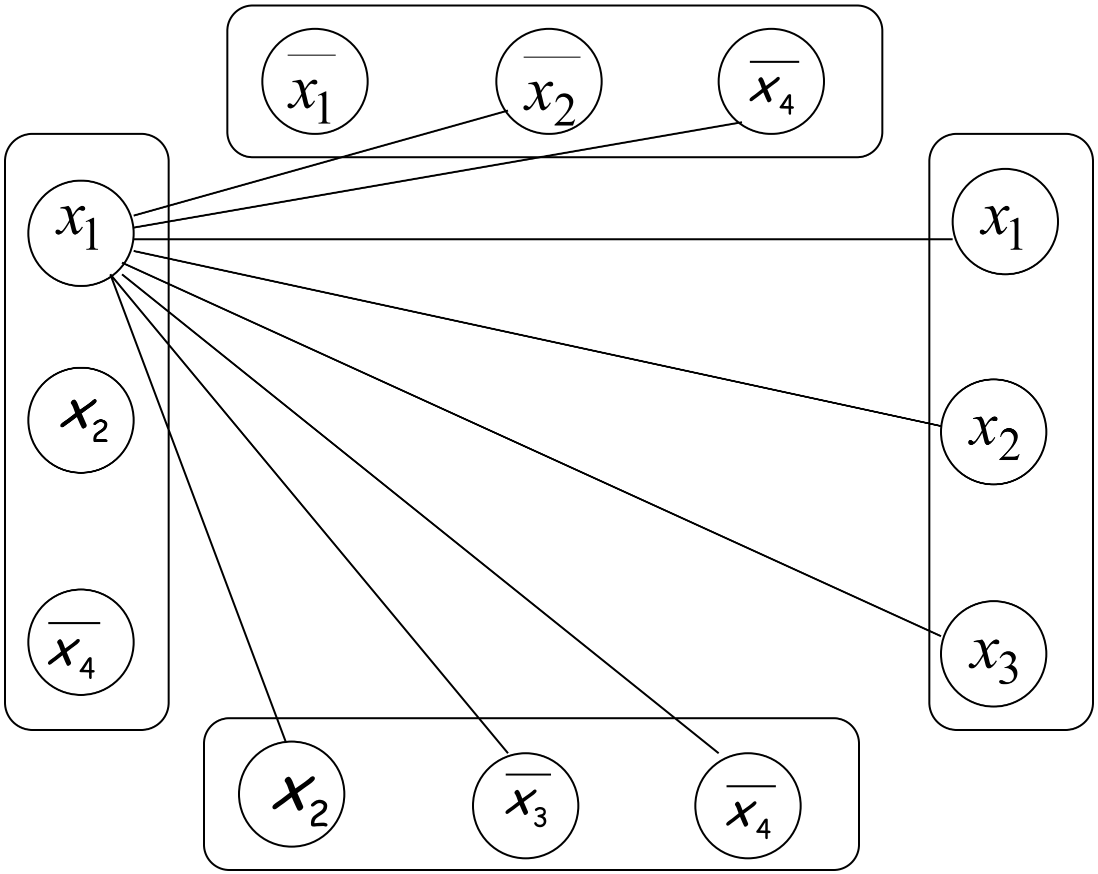
Example:

$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$

Create Nodes:

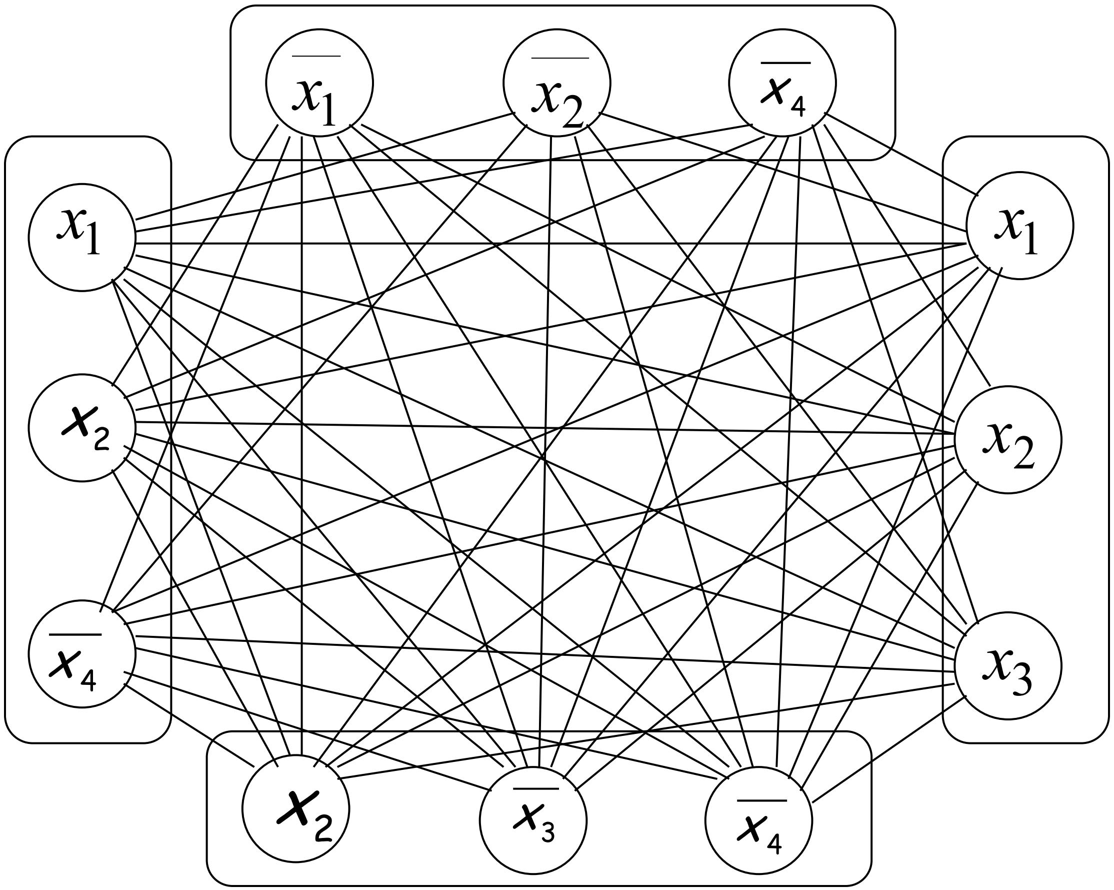


$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Add link from a literal ξ to a literal in every other clause, except the complement $\overline{\xi}$

$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Resulting Graph

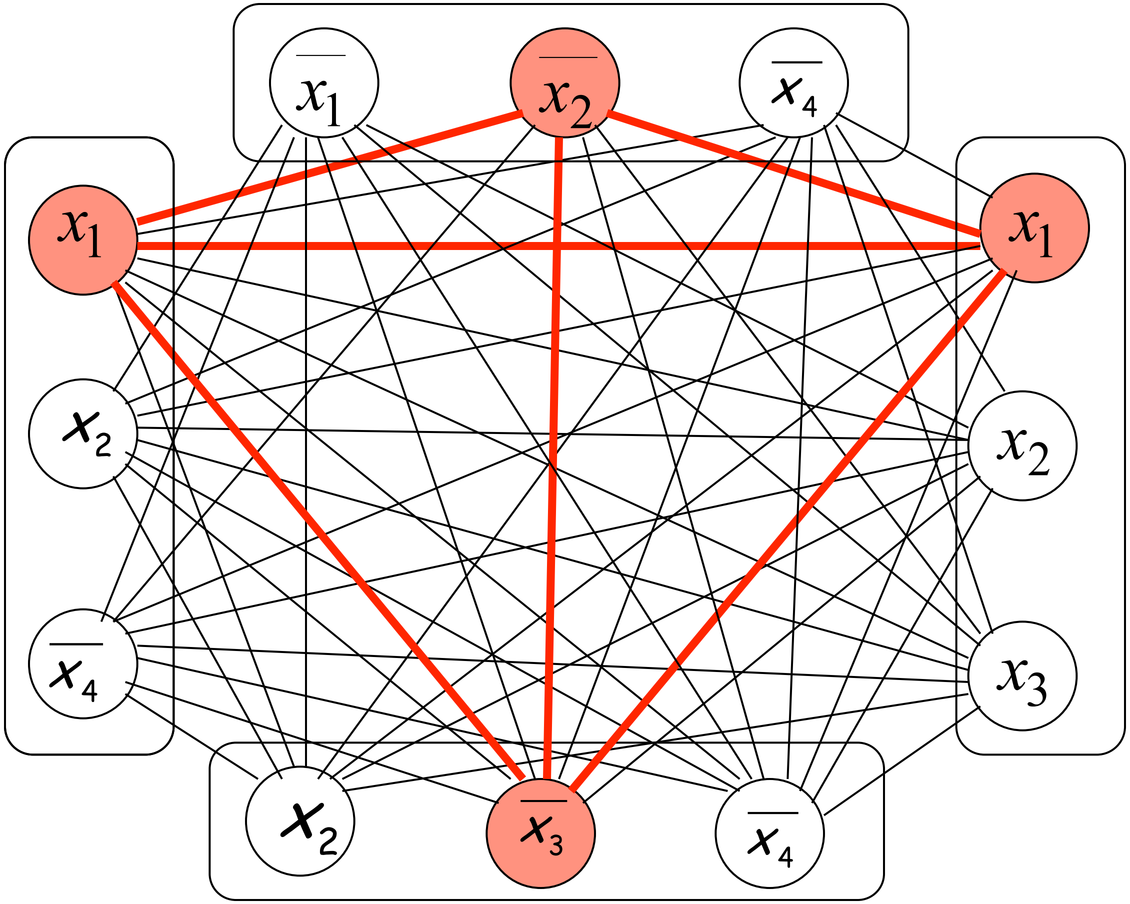
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 1$



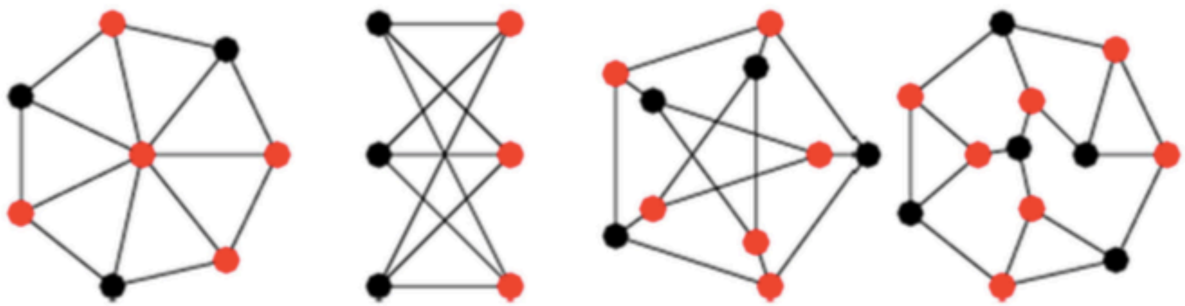
The formula is satisfied if and only if the Graph has a 4-clique

End of Proof

VERTEX COVER PROBLEM

- **A vertex cover of an undirected graph G=(V,E) is a subset of vertices such that every edge is incident to at least one of the vertices**
  - We're typically interested in finding the minimum sized vertex cover

Example

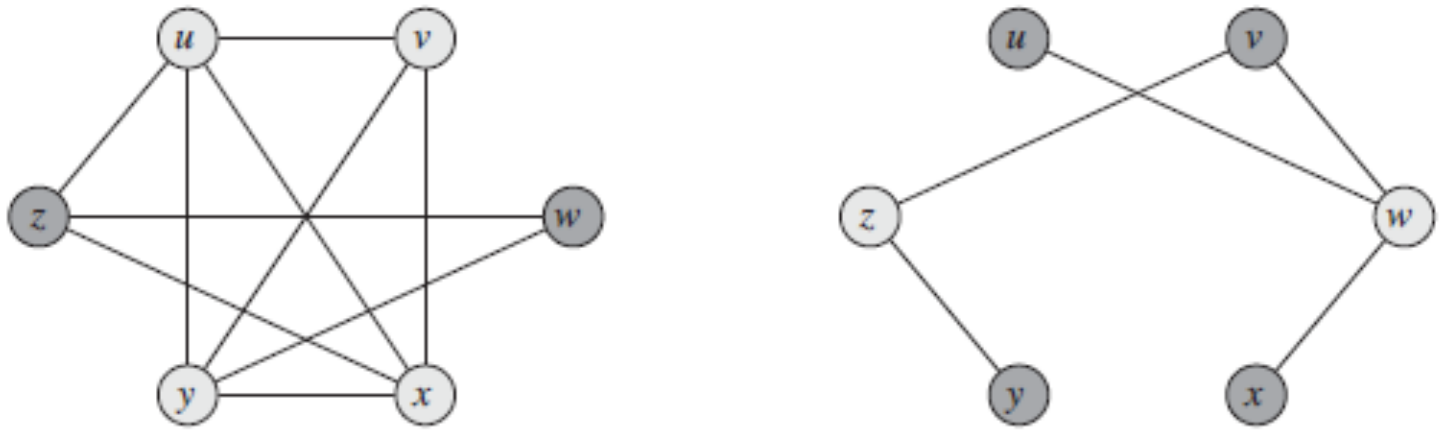


IS VERTEX COVER NP-COMPLETE?

- **To show vertex cover is NP-complete**
  - What problem should we try to reduce to it?
    - It sounds like the 'reverse' of CLIQUE
  - Reduction is done from CLIQUE to vertex cover

G

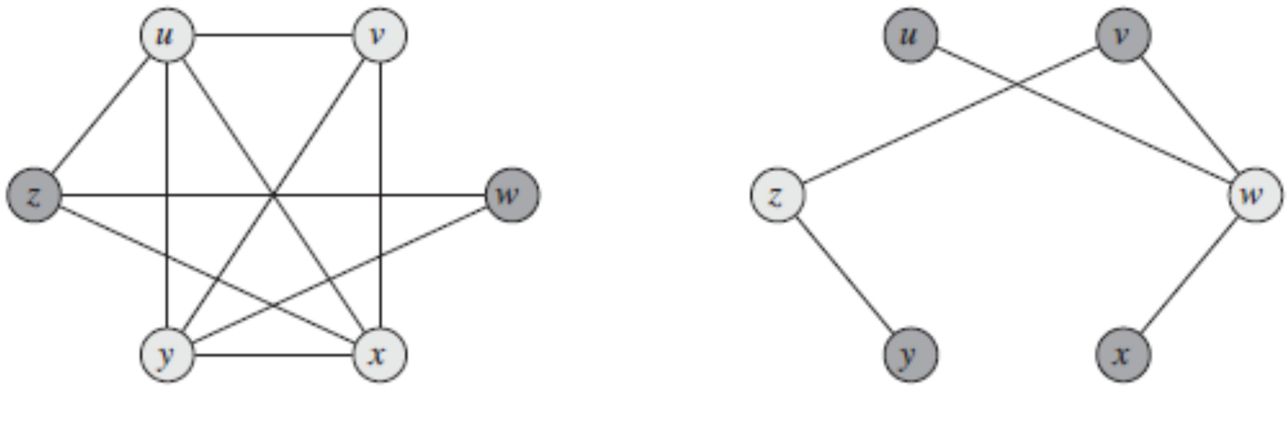
G' = Complement of G



(a) (b)

Clique of size k in G exists iff a vertex cover of size |V| - k exists in G' where G' is the complement graph (vertices that had an edge between then in G do not have one in G' and vice versa)

$$\text{CLIQUE} \leq_p \text{VC}$$



(a) (b)

The original graph has a u,v,x,y CLIQUE. That is a clique of size 4

The complement graph has a vertex cover of size 6 (number of vertices) – 4 (clique size). z,w is one such vertex cover.

TOWARDS ALTERNATIVE PERFORMANCE MEASURES

- **So far we have been talking about algorithms that**
  - run in polynomial time on *all* instances
  - always find the solution with the *best* score/ cost.
- **As we have seen, such algorithms may be too much to ask for.**
- **We will now briefly discuss how one can meaningfully relax the above requirements.**

WORST CASE VS. AVERAGE PERFORMANCE

- So far, we have been insisting that there exists a polynomial *p* such the running time of an algorithm is bounded by *p(n)* for *all* instances of size *n*.
- However, for many practical purposes, it may be sufficient to have an algorithm whose *average* running time for instances of size *n* is bounded by a polynomial.
- Such an algorithm may still be unacceptably slow for some particularly bad instances, but such bad instances will necessarily be very rare and may be of little practical relevance.

APPROXIMATION ALGORITHMS

- **While optimal solutions to optimization problems are clearly best, "reasonably" good solutions are also of value.**
- **Let us say that an algorithm for a minimization problem *D* has a *performance guarantee of* $1 + \varepsilon$ if for each instance *I* of the problem it finds a solution whose cost is at most $(1 + \varepsilon)$ times the cost of the optimal solution for instance *I*.**
- **While *D* may be *NP*-hard, it may still be possible to find, for some $\varepsilon > 0$, polynomial-time algorithms for *D* with performance guarantee $1 + \varepsilon$.**
- **Such algorithms are called *approximation algorithms*.**
- **For maximization problems, the notion of an approximation algorithm is defined similarly.**

POLYNOMIAL-TIME APPROXIMATION SCHEMES

- **We say that a minimization problem *D* has a *polynomial-time approximation scheme (PTAS)* if for every $\varepsilon > 0$ there exists a polynomial-time algorithm for *D* with performance guarantee $1 + \varepsilon$.**
- **While *D* may be *NP*-hard, it may still be have a PTAS.**

APPROXIMATION ALGORITHM FOR VERTEX COVER

C←∅

while E = ∅

pick any {u, v} ∈ E

C ← C ∪ {u, v}

delete all edges incident to either u or v

return C

→ not produce optimal solution most of the times

HEURISTIC ALGORITHMS

- Quite often, bioinformaticians rely on *heuristic algorithms* for solving NP-hard optimization problems.
- These are algorithms that appear to run reasonably fast on the average instance, appear to find, most of the time, solutions within *$(1 + \varepsilon)$* of optimum for reasonably small ε.
- However, it is not always easy or possible to mathematically analyze the performance of a heuristic algorithm.

OTHER NP COMPLETE PROBLEMS

- **Subset sum**
  - Given a set of positive integers and some target t > 0, do we have a subset that sums up to that target set
  - Why is the naïve algorithm going to be bad?
- **Richard Karp proved 21 problems to be NP complete in a seminal 1971 paper**
  - Not that hard to read actually!
  - Definitely not hard to read it to the point of knowing what these problems are.