P, NP, NP-Hard & NP-complete problems

Prof. Shaik Naseera

Department of CSE

JNTUACEK, Kalikiri

Objectives

- P, NP, NP-Hard and NP-Complete
- Solving 3-CNF Sat problem
- Discussion of Gate Questions

Types of Problems

- Trackable
- Intrackable
- Decision
- Optimization

Trackable : Problems that can be solvable in a reasonable(polynomial) time.

Intrackable : Some problems are *intractable*, as they grow large, we are unable to solve them in reasonable time.

Tractability

- What constitutes reasonable time?
  - Standard working definition: *polynomial time*
  - On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(2^n)$, $O(n^n)$, $O(n!)$
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given.

Optimization/Decision Problems

- Optimization Problems
  - An optimization problem is one which asks, “What is the optimal solution to problem X?”
  - Examples:
    - 0-1 Knapsack
    - Fractional Knapsack
    - Minimum Spanning Tree
- Decision Problems
  - An decision problem is one with yes/no answer
  - Examples:
    - Does a graph G have a MST of weight $\leq$ W?

Optimization/Decision Problems

- An optimization problem tries to find an optimal solution
- A decision problem tries to answer a yes/no question
- Many problems will have decision and optimization versions
  - Eg: Traveling salesman problem
    - optimization: find hamiltonian cycle of minimum weight
    - decision: is there a hamiltonian cycle of weight $\leq k$

P, NP, NP-Hard, NP-Complete

-Definitions

The Class *P*

P: the class of problems that have polynomial-time deterministic algorithms.

- That is, they are solvable in $O(p(n))$, where $p(n)$ is a polynomial on n
- A deterministic algorithm is (essentially) one that always computes the correct answer

Sample Problems in P

- Fractional Knapsack
- MST
- Sorting
- Others?

The class *NP*

<u>NP</u>: the class of decision problems that are solvable in polynomial time on a *nondeterministic* machine (or with a nondeterministic algorithm)

- (A *<u>determinstic</u>* computer is what we know)
- A *<u>nondeterministic</u>* computer is one that can "guess" the right answer or solution

- Think of a nondeterministic computer as a parallel machine that can freely spawn ***an infinite number*** of processes
- Thus *NP* can also be thought of as the class of problems "whose solutions can be verified in polynomial time"
- Note that *NP* stands for "Nondeterministic Polynomial-time"

Sample Problems in NP

- Fractional Knapsack
- MST
- Others?
  - Traveling Salesman
  - Graph Coloring
  - Satisfiability (SAT)
    - the problem of deciding whether a given Boolean formula is satisfiable

P And NP Summary

- **P** = set of problems that can be solved in polynomial time
  - Examples: Fractional Knapsack, ...
- **NP** = set of problems for which a solution can be verified in polynomial time
  - Examples: Fractional Knapsack,..., TSP, CNF SAT, 3-CNF SAT
- Clearly **P** $\subseteq$ **NP**
- Open question: Does **P = NP**?
  - P $\neq$ NP

NP-hard

- What does NP-hard mean?
  - A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).
- A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, ...
- Ex:- Hamiltonian Cycle

Every problem in NP is reducible to HC in polynomial time. Ex:- TSP is reducible to HC.

B A

Example: lcm(m, n) = m * n / gcd(m, n),

NP-complete problems

- A problem is **NP-complete** if the problem is both
  - NP-hard, and
  - NP.

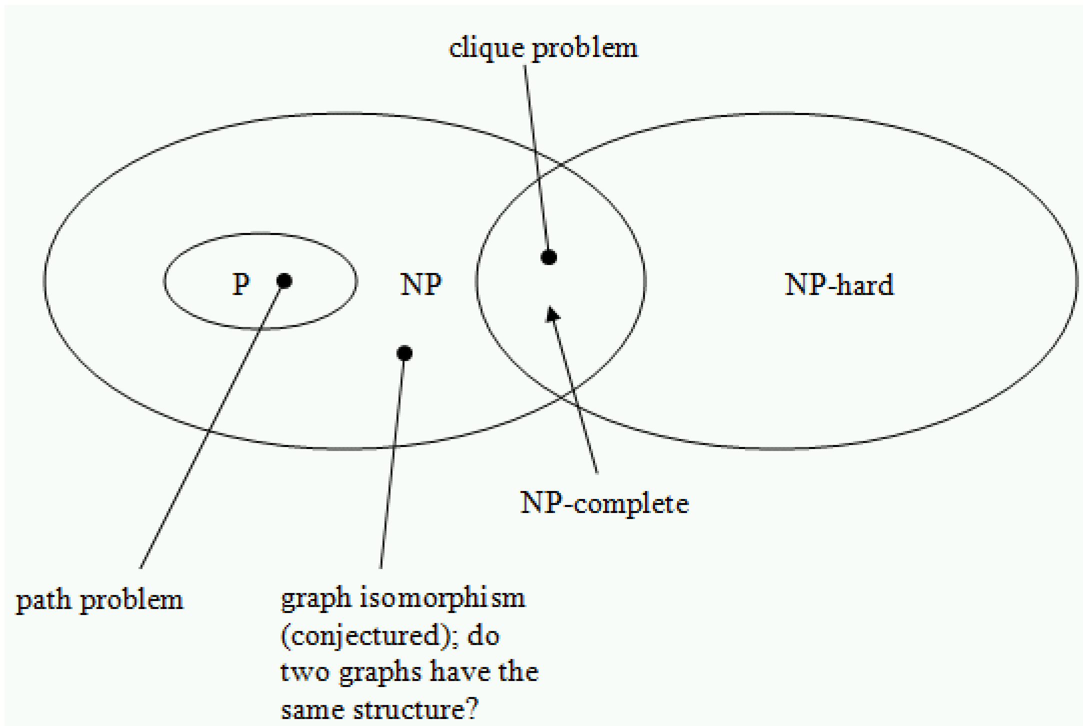
Reduction

- A problem R can be *reduced* to another problem Q if any instance of R can be rephrased to an instance of Q, the solution to which provides a solution to the instance of R
  - This rephrasing is called a *transformation*
- Intuitively: If R reduces in polynomial time to Q, R is “no harder to solve” than Q
- Example: lcm(m, n) = m * n / gcd(m, n),

  lcm(m,n) problem is reduced to gcd(m, n) problem

NP-Hard and NP-Complete

- If R is *polynomial-time reducible* to Q, we denote this $R \leq_p Q$
- Definition of NP-Hard and NP-Complete:
  - If all problems $R \in$ **NP** are *polynomial-time* reducible to Q, then Q is *NP-Hard*
  - We say Q is *NP-Complete* if Q is NP-Hard and $Q \in$ **NP**
- If $R \leq_p Q$ and R is NP-Hard, Q is also NP-Hard


clique problem
P
NP
NP-hard
NP-complete
path problem
graph isomorphism
(conjectured); do
two graphs have the
same structure?

Summary

- P is set of problems that can be solved by a deterministic Turing machine in **P**olynomial time.
- NP is set of problems that can be solved by a **N**on-deterministic Turing Machine in **P**olynomial time. P is subset of NP (any problem that can be solved by deterministic machine in polynomial time can also be solved by non-deterministic machine in polynomial time) but $P \neq NP$.

- Some problems can be translated into one another in such a way that a fast solution to one problem would automatically give us a fast solution to the other.
- There are some problems that every single problem in NP can be translated into, and a fast solution to such a problem would automatically give us a fast solution to every problem in NP. This group of problems are known as **NP-Complete**. Ex:- Clique
- A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.

First NP-complete problem—Circuit Satisfiability (problem definition)

- Boolean combinational circuit
  - Boolean combinational elements, wired together
  - Each element, inputs and outputs (binary)
  - Limit the number of outputs to 1.
  - Called *logic gates*: NOT gate, AND gate, OR gate.
  - *true table*: giving the outputs for each setting of inputs
  - true assignment: a set of boolean inputs.
  - satisfying assignment: a true assignment causing the output to be 1.

Circuit Satisfiability Problem: definition

- Circuit satisfying problem: given a boolean combinational circuit composed of AND, OR, and NOT, is it stisfiable?
- CIRCUIT-SAT={<C>: C is a satisfiable boolean circuit}
- Implication: in the area of computer-aided hardware optimization, if a subcircuit always produces 0, then the subcircuit can be replaced by a simpler subcircuit that omits all gates and just output a 0.

Two instances of circuit satisfiability problems

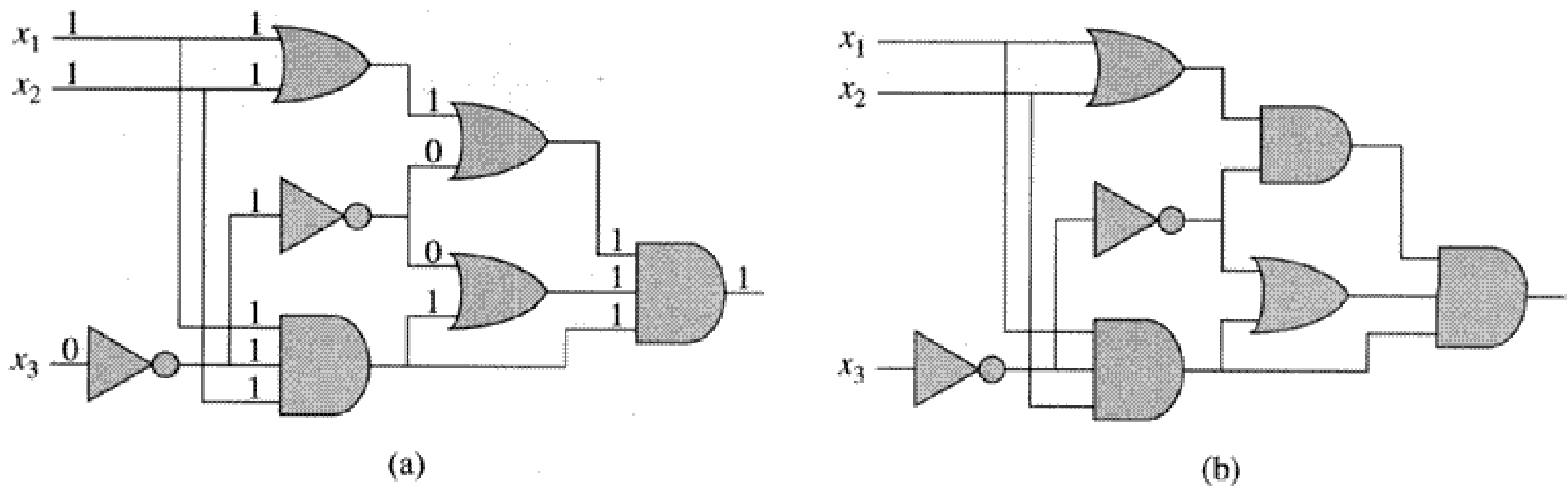


Figure 34.8 Two instances of the circuit-satisfiability problem. **(a)** The assignment $\langle x_1 = 1, x_2 = 1, x_3 = 0\rangle$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable. **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable.

Solving circuit-satisfiability problem

- Intuitive solution:
  - for each possible assignment, check whether it generates 1.
  - suppose the number of inputs is k, then the total possible assignments are 2^k. So the running time is $\Omega(2^k)$. When the size of the problem is $\Theta(k)$, then the running time is not polynomial.

Example of reduction of CIRCUIT-SAT to SAT

$\phi = x_{10} \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))$
$\wedge (x_9 \leftrightarrow (x_6 \vee x_7))$
$\wedge (x_8 \leftrightarrow (x_5 \vee x_6))$
$\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4))$
$\wedge (x_6 \leftrightarrow \neg x_4))$
$\wedge (x_5 \leftrightarrow (x_1 \vee x_2))$
$\wedge (x_4 \leftrightarrow \neg x_3)$

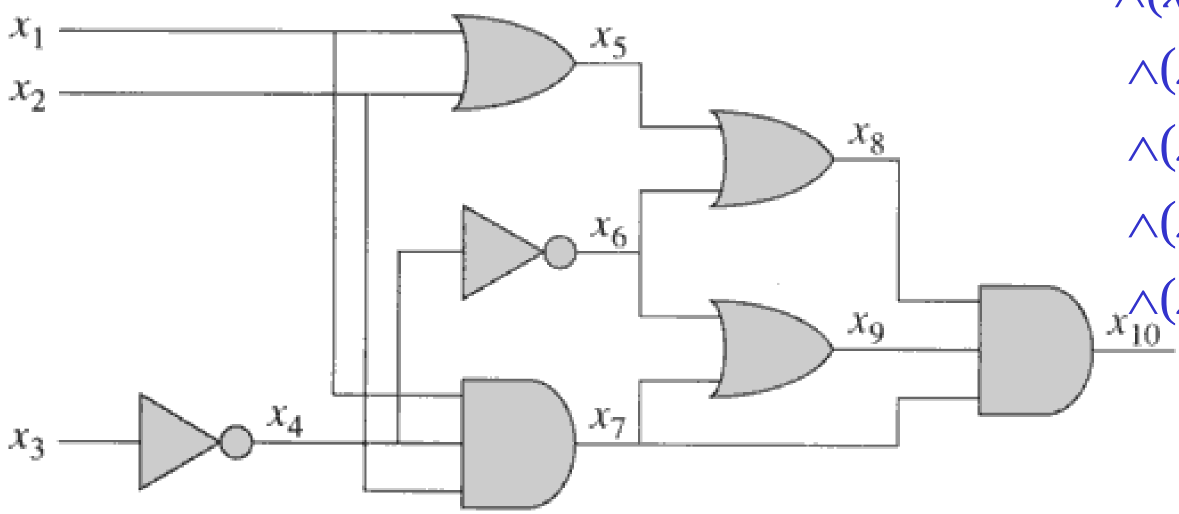


Figure 34.10 Reducing circuit satisfiability to formula satisfiability. The formula produced by the reduction algorithm has a variable for each wire in the circuit.

REDUCTION: $\phi = x_{10} = x_7 \wedge x_8 \wedge x_9 = (x_1 \wedge x_2 \wedge x_4) \wedge (x_5 \vee x_6) \wedge (x_6 \vee x_7)$

$= (x_1 \wedge x_2 \wedge x_4) \wedge ((x_1 \vee x_2) \vee \neg x_4) \wedge (\neg x_4 \vee (x_1 \wedge x_2 \wedge x_4)) = \ldots.$

Conversion to 3 CNF

- The result is that in ϕ', each clause has at most three literals.
- Change each clause into conjunctive normal form as follows:
  - Construct a true table, (small, at most 8 by 4)
  - Write the disjunctive normal form for all true-table items evaluating to 0
  - Using DeMorgan law to change to CNF.
- The resulting ϕ'' is in CNF but each clause has 3 or less literals.
- Change 1 or 2-literal clause into 3-literal clause as follows:
  - If a clause has one literal l, change it to $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$.
  - If a clause has two literals $(l_1 \vee l_2)$, change it to $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$.

Example of a polynomial-time reduction:

We will reduce the

3CNF-satisfiability problem

to the

CLIQUE problem

3CNF formula:

literal

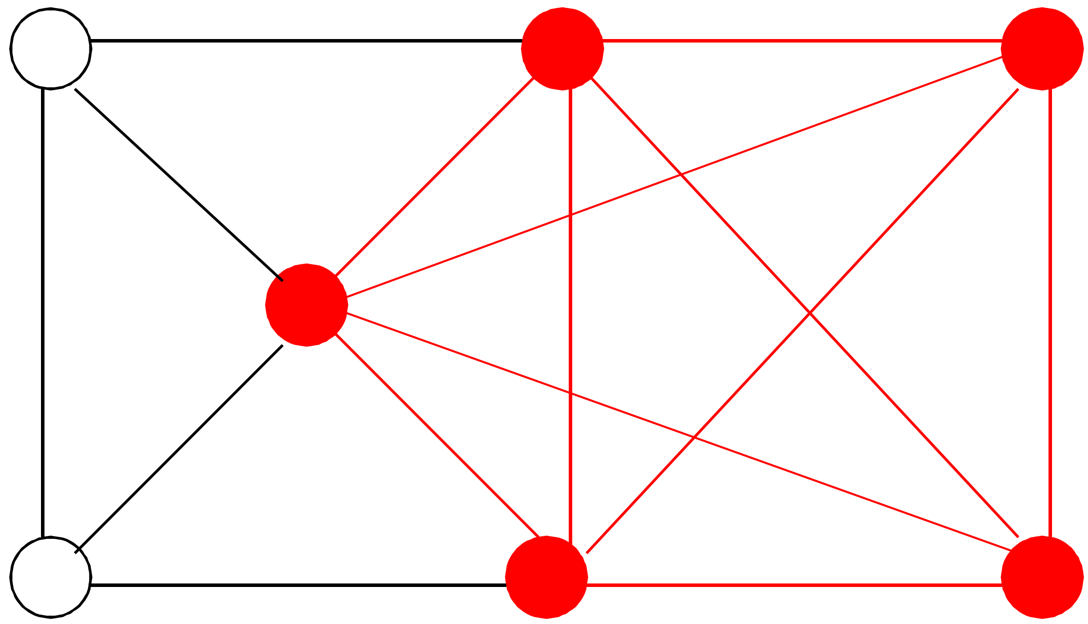
$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \underbrace{(x_3 \vee \overline{x_5} \vee x_6)}_{\text{clause}} \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

Each clause has three literals

Language:

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

A 5-clique in graph G



Language:

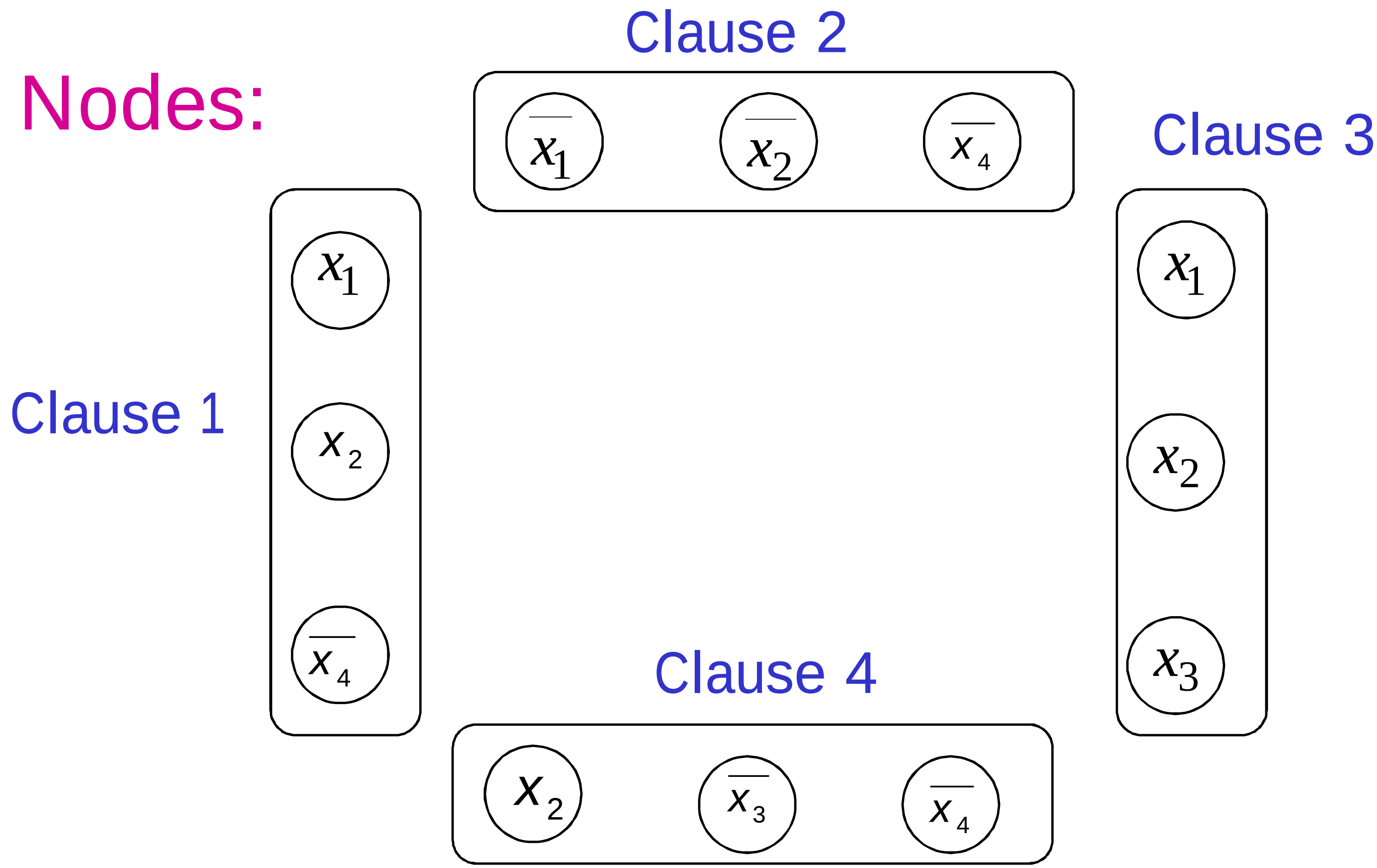
$$\text{CLIQUE} = \{ \langle G, k \rangle : \text{ graph } G \text{ contains a } k\text{-clique} \}$$

Transform formula to graph.

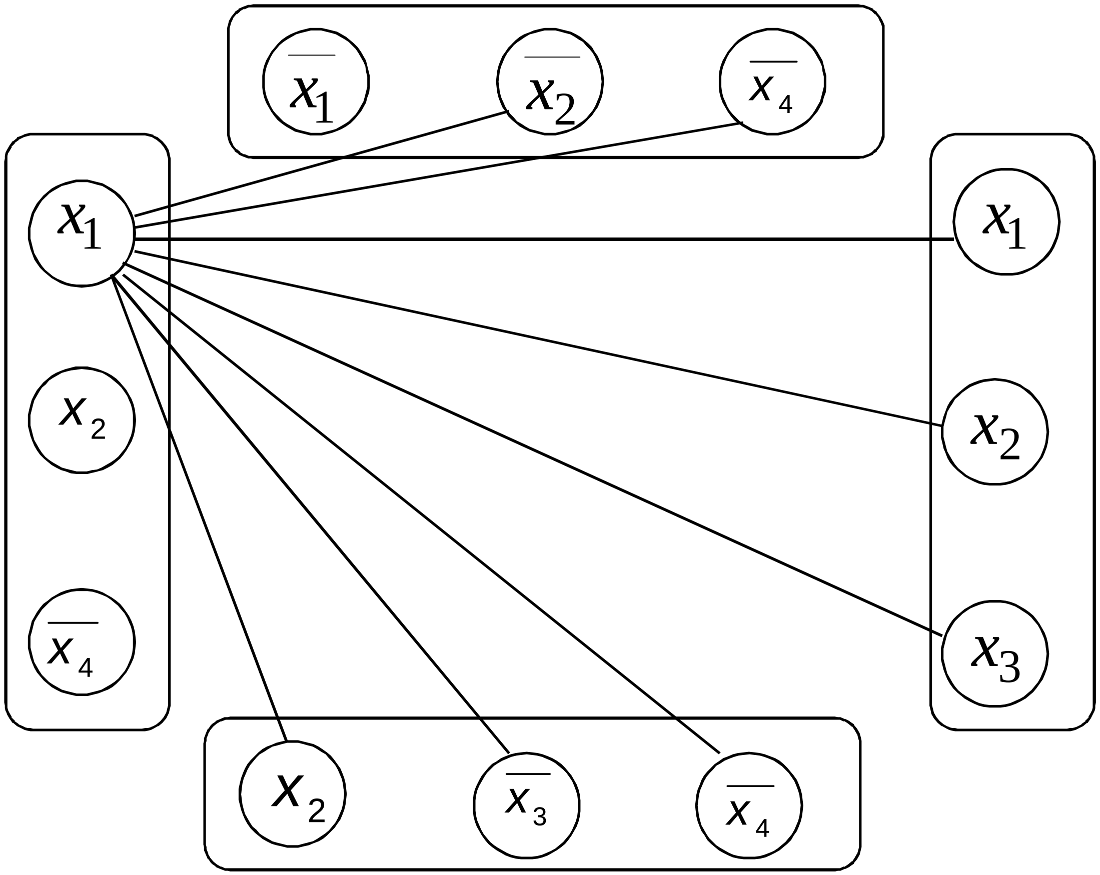
Example:

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

Create Nodes:

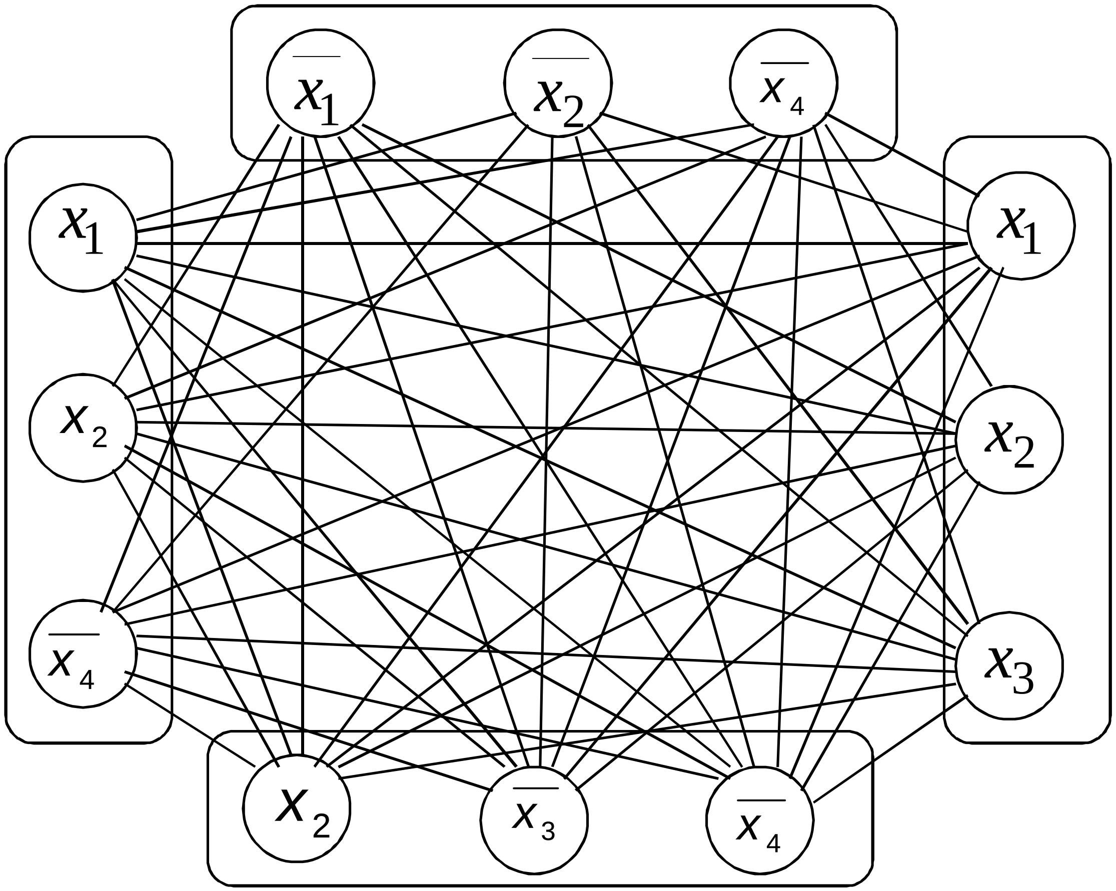


$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



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

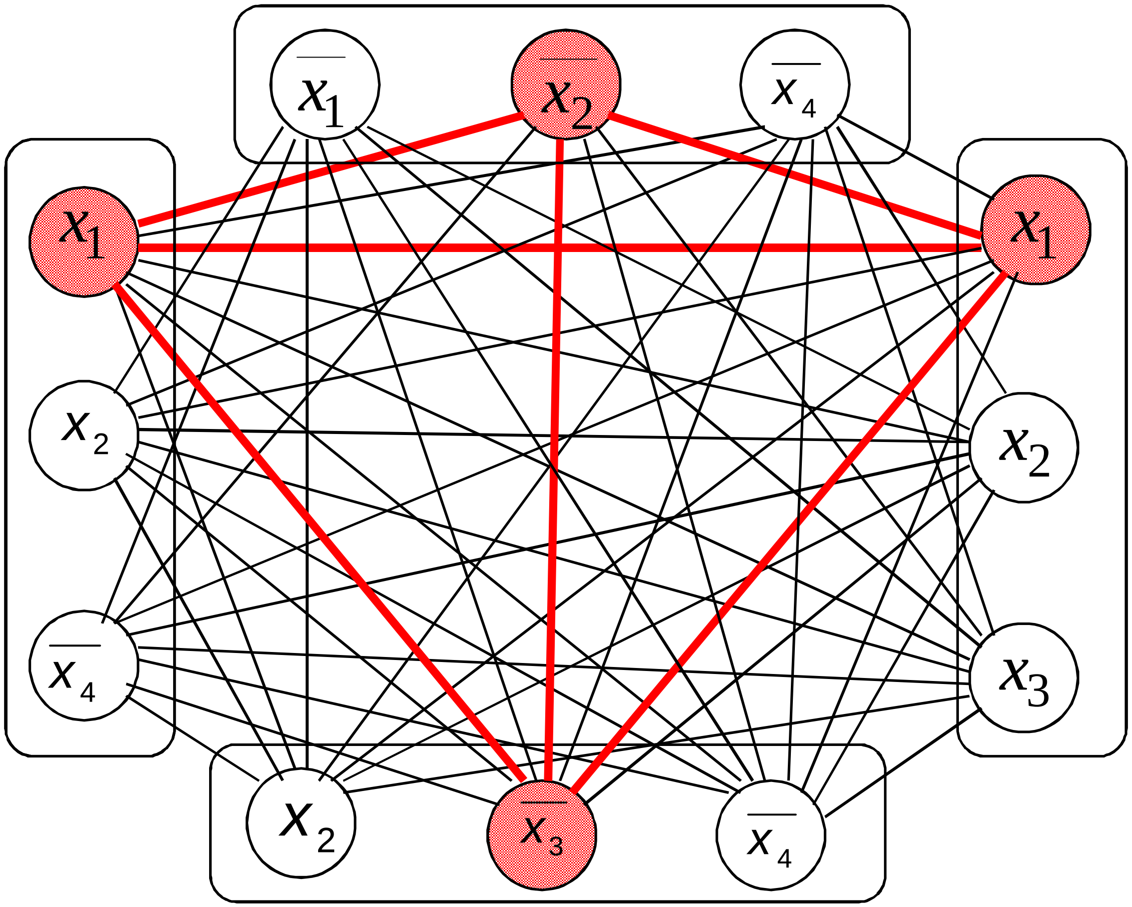
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$x_1 = 1$

$x_2 = 0$

$x_3 = 0$

$x_4 = 1$



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

The objective is to find a clique of size 4, where 4 is the number of clauses.

End of Proof

Theorem:

If: a. Language A is NP-complete

b. Language B is in NP

c. A is polynomial time reducible to B

Then: B is NP-complete

Corollary: CLIQUE is NP-complete

Proof:

a. 3CNF-SAT is NP-complete

b. CLIQUE is in NP

c. 3CNF-SAT is polynomial reducible to CLIQUE (shown earlier)

Apply previous theorem with

A=3CNF-SAT and B=CLIQUE

Previous Gate Questions

1 MCQ (Single Correct Answer)

Q. No. 1

GATE CSE 2015 Set 2

Consider two decision problems Q_1, Q_2 such that Q_1 reduces in polynomial time to $3\text{-}SAT$ and $3\text{-}SAT$ reduces in polynomial time to Q_2. Then which one of the following is consistent with the above statement?

A Q_1 is NP, Q_2 is NP hard.

B Q_2 is NP, Q_1 is NP hard.

C Both Q_1 and Q_2 are in NP.

D Both Q_1 and Q_2 are NP hard.

Q. No. 2

Which of the following statements are TRUE?

1. The problem of determining whether there exists a cycle in an undirected graph is in P.

2. The problem of determining whether there exists a cycle in an undirected graph is in NP.

3. If a problem A is NP−Complete, there exists a non-deterministic polynomial time algorithm to solve A.

A 1 , 2 and 3

B 1 and 2 only

C 2 and 3 only

D 1 and 3 only

Q. No. 3

GATE CSE 2009

Let π_A be a problem that belongs to the class NP. Then which one of the following is TRUE?

- A There is no polynomial time algorithm for π_A
- B If π_A can be solved deterministically in polynomial time, then P = NP
- C If π_A is NP-hard, then it is NP-complete.
- D π_A may be undecidable.

Q. No. 4

GATE CSE 2006

Let S be an NP-complete problem and Q and R be two other problems not known to be in NP. Q is polynomial time reducible to S and S is polynomial-time reducible to R. Which one of the following statements is true?

- A R is NP-complete
- B R is NP-hard
- C Q is NP-complete
- D Q is NP-hard

Q. No. 5

GATE CSE 2004

The problems 3-SAT and 2-SAT are

- (A) both in P
- (B) both NP-complete
- (C) NP-complete and in P respectively
- (D) undecidable and NP-complete respectively

Q. No. 6

GATE CSE 2003

Ram and Shyam have been asked to show that a certain problem Π is NP-complete. Ram shows a polynomial time reduction from the 3-SAT problem to Π, and Shyam shows a polynomial time reduction from Π to 3-SAT. Which of the following can be inferred from these reductions ?

A. Π is NP-hard but not NP-complete

B. Π is in NP, but is not NP-complete

C. Π is NP-complete

D. Π is neither NP-hard, nor in NP

Q. No. 7

GATE CSE 2014 Set 3

Consider the decision problem 2CNFSAT defined as follows:

{ Φ | Φ is a satisfiable propositional formula in CNF with at most two literal per clause}

For example, $\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (x_2 \vee x_4)$ is a Boolean formula and it is in 2CNFSAT.

The decision problem 2CNFSAT is

A. NP-Complete.

B. Solvable in polynomial time by reduction to directed graph reachability.

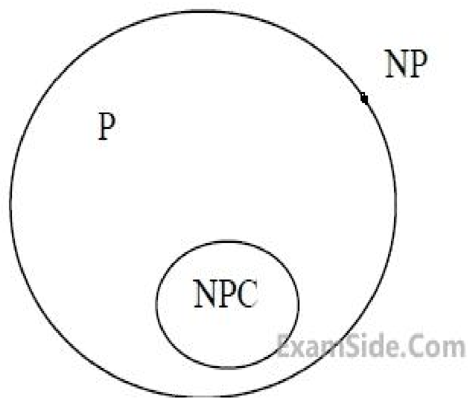
C. Solvable in constant time since any input instance is satisfiable.

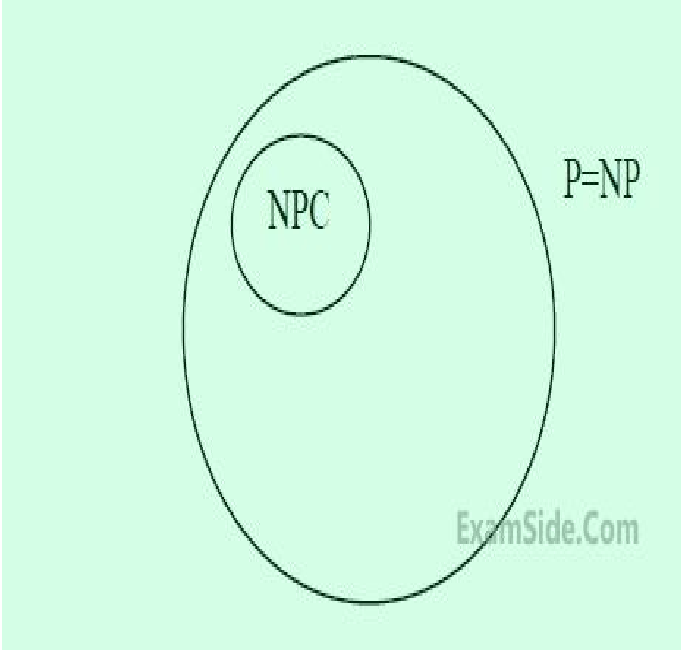
D. NP-Hard, but not NP-complete.

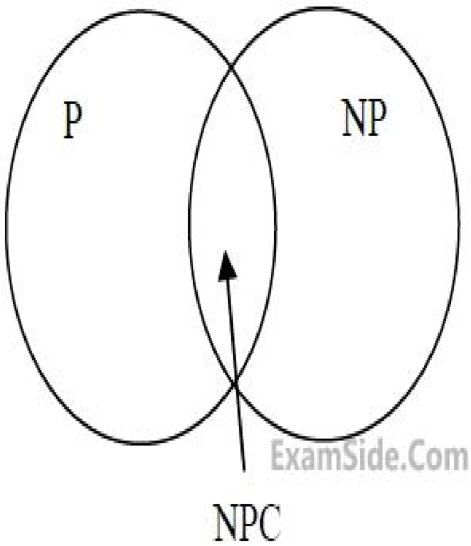
Q. No. 8 GATE CSE 2014 Set 1

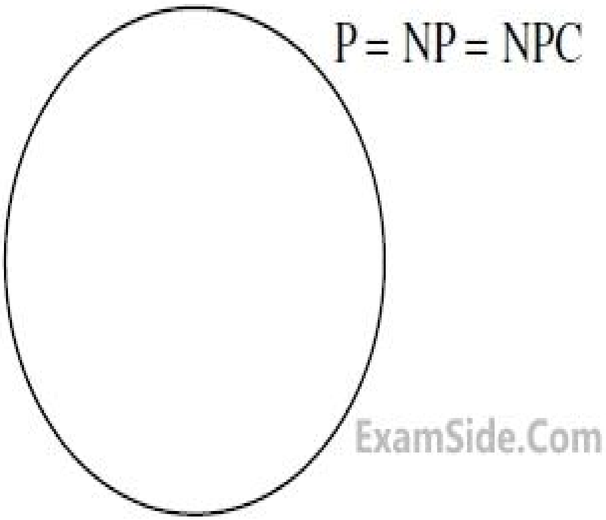
Suppose a polynomial time algorithm is discovered that correctly computes the largest clique in a given graph. In this scenario, which one of the following represents the correct Venn diagram of the complexity classes P, NP and NP Complete (NPC)?

A









Q. No. 10

GATE CSE 2006

Let $SHAM_3$ be the problem of finding a Hamiltonian cycle in a graph G = (V, E) with |V| divisible by 3 and $DHAM_3$ be the problem of determining if a Hamiltonian cycle exists in such graphs. Which one of the following is true?

There exist: search problem

A. Both $DHAM_3$ and $SHAM_3$ are NP-hard

B. $SHAM_3$ is NP-hard, but $DHAM_3$ is not

C. $DHAM_3$ is NP-hard, but $SHAM_3$ is not

D. Neither $DHAM_3$ nor $SHAM_3$ is NP-hard

Explanation: The problem of finding whether there exist a Hamiltonian Cycle or not is NP Hard and NP Complete Both.

Finding a Hamiltonian cycle in a graph G = (V,E) with V divisible by 3 is also NP Hard.

Q. No. 11

GATE CSE 1992

Which of the following problems is not NP-hard?

- A Hamiltonian circuit problem
- B The 0/1 Knapsack problem
- C Finding bi-connected components of a graph
- D The graph coloring problem

Thank You