Design and Analysis of Algorithms

UNIT-IV

NP-Hard and NP-Complete problems

Dr G. Kalyani

Topics

- **Basic Concepts**
- **Non-Deterministic Algorithms**
- **NP Hard and NP Complete Problems**

Categorization of Problems

- Depending on problem characteristics
- Depending on time complexity
- Depending on the logic used to solve the problem

Types of Problems
(depending on problem characteristics)

- Any problem for which answer is 0 or1 is called decision problem, such algorithm is called decision algorithm.

- Any problem that tries to identify all possible solutions is called enumeration problem, such algorithm is called enumeration algorithm.

- Any problem that involves identification of optimal value of given cost function is known as optimization problem, such algorithm is optimization algorithm.

Types of Problems
(depending on time complexity)

- An algorithm for a given problem is said to be polynomial time algorithm if its time complexity belongs $O(n^k)$, Where k is an integer.

- An algorithm for a given problem is said to be non polynomial time algorithm if it is Not a polynomial time algorithm. TSP($n^2 2^n$)

Types of Problems (depending on logic used)

- Deterministic machine: It is a machine that at any given state during execution, reading Input symbol for next state is known. Algorithms run on deterministic machines are called Deterministic Algorithms.

- If not such machines they are called non deterministic machines and such algorithms are called Nondeterministic Algorithms.

Deterministic & Nondeterministic Search Algorithm

Deterministic

```
A is array of size n

for j= 1 to n
{
   If A[j]=key then
   {
      Write(j);
   }
}
Write(0);
```

Non-Deterministic

```
A is array of size n
for i=1 to n
{
   j:=Choice(1,n)
   If A[j]=key then
   {
        Write(j);
        Success();
     }
}
Write(0);
Failure()
}
```

Nondeterministic Sorting Algorithm

```
Algorithm NSort(A, n)
// Sort n positive integers.
{
    for i := 1 to n do B[i] := 0; // Initialize B[ ].
    for i := 1 to n do
    {
        j := Choice(1, n);
        if B[j] ≠ 0 then Failure();
        B[j] := A[i];
    }
    for i := 1 to n − 1 do  // Verify order.
        if B[i] > B[i + 1] then Failure();
    write (B[1 : n]);
    Success();
}
```

Satisfiability(SAT) Problem

- Let x1,X2,… denote boolean variables (their value is either true or false).
- Let $\bar{xi}$ denote the negation of xi.
- A literal is either a variable or its negation.
- A formula in the propositional calculus is an expression that can be constructed using literals and the operations ***and*** and ***or***.
- Examples of such formulas are $(x_1 \wedge \bar{x}_2) \vee (x_3 \wedge \bar{x}_4)$ and $(x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2)$.
- The symbol ∨ denotes or and ∧ denotes and.

Satisfiability(SAT) Problem contd..

- A formula is in **Conjunctive Normal Form (CNF)** if and only if it is represented as $\wedge_{i=1}^{k} c_i$ where the ci are clauses each represented as $\vee\, l_{ij}$. The l_{ij} are literals.

- It is in **Disjunctive Normal Form(DNF)** if and only if it is represented as $\vee_{i=1}^{k} c_i$ and each clause ci is represented as $\wedge\, l_{ij}$.

- Ex for DNF: $(x_1 \wedge x_2) \vee (x_3 \wedge \bar{x_4})$
- Ex for CNF: $(x_3 \vee \bar{x_4}) \wedge (x_1 \vee \bar{x_2})$

- The satisfiability problem is to determine whether a formula is true for some assignment of truth values to the variables.

- CNF-satisfiability is the satisfiability problem for CNF formulas.

Nondeterministic Algorithm for SAT Problem

```
Algorithm Eval(E, n)
// Determine whether the propositional formula E is
// satisfiable. The variables are x_1, x_2, ..., x_n.
{
    for i := 1 to n do  // Choose a truth value assignment.
        x_i := Choice(false, true);
    if E(x_1, ..., x_n) then Success();
    else Failure();
}
```

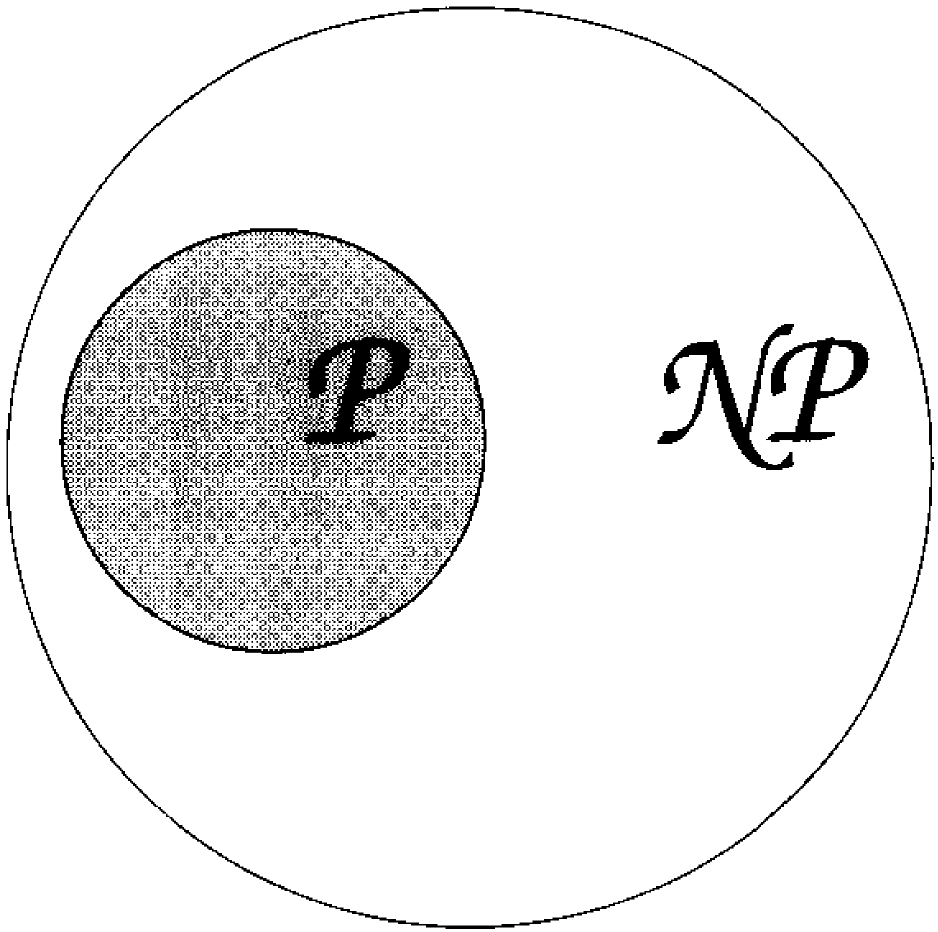
P and NP Problems

- **P:** The class of decision problems that are solvable in polynomial time by deterministic algorithms.
- **NP:** The class of decision problems that are solvable in polynomial time by nondeterministic algorithms.

- Since deterministic algorithms are just a special case of nondeterministic ones $\mathrm{P} \subseteq NP$.

- The most famous unsolved problem in computer science, is whether $P = NP \; or \; P \neq NP$.

Relationship between P and NP

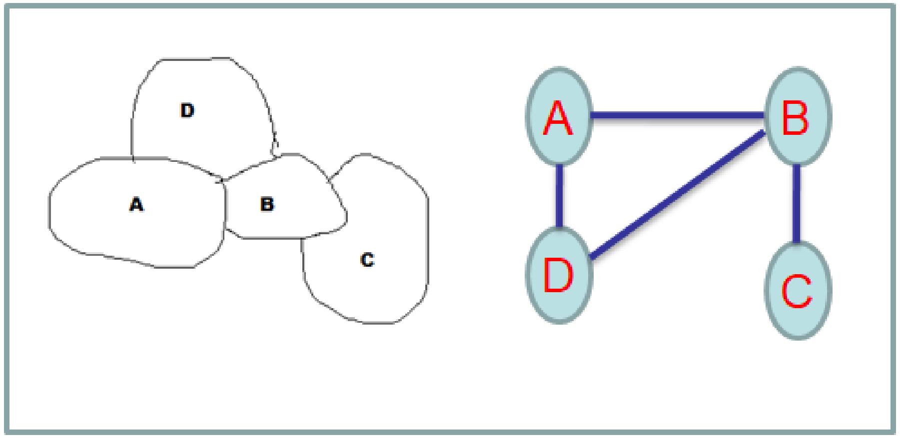


Reducibility

Definition 11.4 Let L_1 and L_2 be problems. Problem L_1 *reduces* to L_2 (also written $L_1 \propto L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time. $\square$

3x+4

ax^2+bx+c

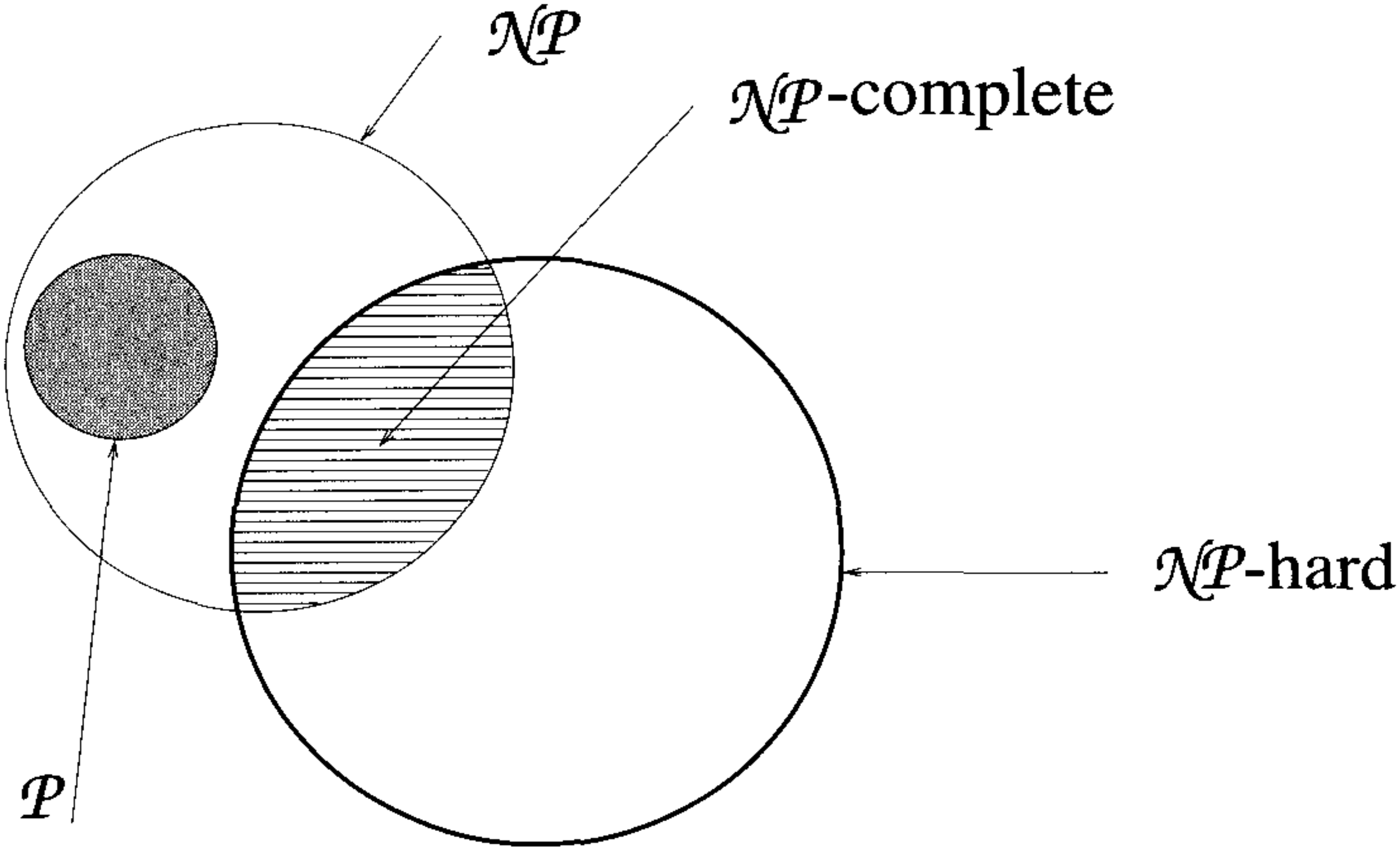


Graph coloring Problem

NP-Hard and NP-Complete

Definition 11.5 A problem L is $\mathcal{NP}$-hard if and only if satisfiability reduces to L (satisfiability $\propto L$). A problem L is $\mathcal{NP}$-complete if and only if L is $\mathcal{NP}$-hard and $L \in \mathcal{NP}$. □

Relationship between P, NP, NP-hard and NP-Complete



Examples of NP-Hard and NP-Complete

- The following problems are NP-Hard
  - The circuit-satisfiability problem
  - Set Cover
  - Vertex Cover
  - Travelling Salesman Problem

- Following are some NP-Complete problems:
  - Determining whether a graph has a Hamiltonian cycle
  - Determining whether a Boolean formula is satisfiable, etc.

NP-Hard and NP-Complete

Definition 11.6 Two problems L_1 and L_2 are said to be *polynomially equivalent* if and only if $L_1 \propto L_2$ and $L_2 \propto L_1$. □

To show that a problem L_2 is $\mathcal{NP}$-hard, it is adequate to show $L_1 \propto L_2$, where L_1 is some problem already known to be $\mathcal{NP}$-hard. Since $\propto$ is a transitive relation, it follows that if satisfiability $\propto L_1$ and $L_1 \propto L_2$, then satisfiability $\propto L_2$. *To show that an $\mathcal{NP}$-hard decision problem is $\mathcal{NP}$-complete, we have just to exhibit a polynomial time nondeterministic algorithm for it.*