UNIT – V

NP Hard and NP Complete Problems: Basic Concepts

NP Hard Graph Problems: Clique Decision Problem (CDP), Traveling Salesperson Decision Problem (TSP).

……

Deterministic Algorithms

Deterministic algorithms are the algorithms whose operations are defined in a unique manner and these algorithms are executed successfully generating the desired outcome.

Non-deterministic Algorithms

Non-deterministic algorithm is an algorithm in which each operation generates one or more result. That is this algorithm does not produce unique result.

There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in polynomial time. For example: searching of an element from the list O(log n), sorting of elements O(log n).

The second group consists of problems that can be solved in non-deterministic polynomial time. For example: Knapsack problem $O(2^{n/2})$ and Travelling Salesperson problem ($O(n^2 \cdot 2^n)$).

Any problem for which answer is either yes or no is called **decision problem**. The algorithm for decision problem is called **decision algorithm**.

Any problem that involves the identification of optimal cost (minimum or maximum) is called **optimization problem**. The algorithm for optimization problem is called **optimization algorithm**.

• **Definition of P** – Problems that can be solved in polynomial time. ("P" stands for polynomial).
Examples – Searching of key element, Sorting of elements, All pair shortest path.

• **Definition of NP** – It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".

Examples – Travelling Salesperson problem, Graph coloring problem, Knapsack problem, Hamiltonian circuit problems.

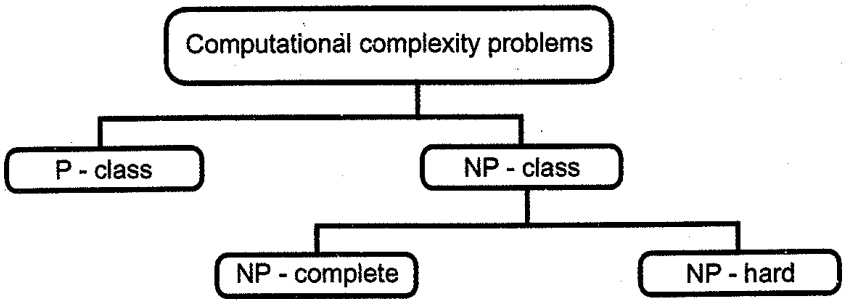
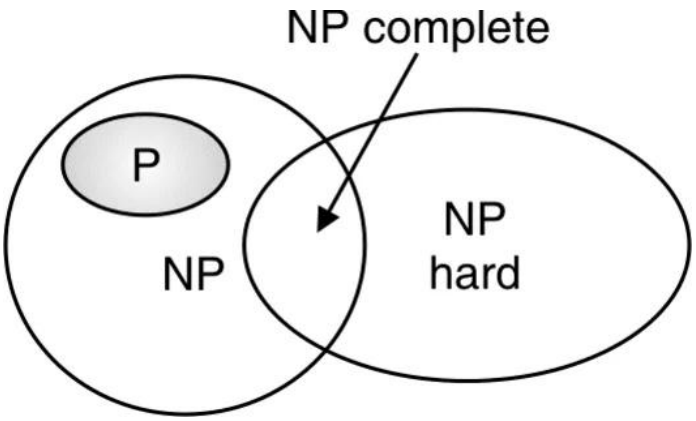


Fig. 9.1.1 Classification of problems

The NP class problems can be further categorized into **NP-complete** and **NP-hard** problems.

- A problem D is called **NP-complete** if –
    - i) It belongs to class NP
    - ii) Every problem in NP can also be solved in polynomial time.
- If an NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.

- All NP-complete problems are NP-hard but all NP-hard problems **cannot be NP-complete**.



The **NP class problems** are the decision problems that can be solved by non-deterministic polynomial algorithms.

Q1. Explain NP complete problems.

Answer:

A problem is said to be NP-complete if and only if it satisfies the following two conditions:

(i) Problem should be present in class NP.

(ii) Each problem should also be solved in polynomial time (i.e., NP-hard problem).

Q2. List some examples of NP-complete problems.

Answer:
Some of the examples of NP-complete problems are:

1. Graph coloring
2. Graph isomorphism
3. Knapsack problem
4. Travelling salesman problem

Q3. What is NP-hard problem?

Answer:

A problem is said to be NP-hard if and only if it is solved in polynomial time or reduced to an NP-hard problem.

Whenever an NP-hard problem is solvable (in polynomial time), then each NP-complete problem can also be solved (in polynomial time).

Q4. List some examples of NP-hard problems.

Answer:
Some of the examples of NP-hard problems are:

1. Halting problem
2. Decision subset sum problem
3. Boolean satisfiability
4. Travelling salesman problem

Q14. Explain about different types of NP problems.

Answer :

The different types of NP problems are,

(a) Problems that have polynomial-time-algorithms.

(b) Problems that does not have polynomial-time-algorithms.

The problems that can be solved by applying the best algorithms are categorized into two categories,

1. First category contain problems with solutions, bounded by polynomials of small degree.

Example

Ordered searching – $O(\log n)$

Sorting – $O(n)$

2. Second category contain problems with solution algorithms of non-polynomials.

Example

Travelling sales person problem – $O(n^2 2^n)$

Knapsack problem – $O(2^{n/2})$.

It does not have the capability of developing polynomial algorithms for the problems of second category. Also, it has not been proved yet that the problems in the second category do not have polynomial-time-algorithms. The theory of NP-completeness simply tells that many of the problems which do not have known polynomial-time-algorithms yet are computationally related and hence classified into two groups. They are,

1. NP-hard
2. NP-complete.

A property states that if an NP-complete problem is able to be solved in polynomial time then the remaining NP-complete problems will also have the polynomial-time-algorithms. Moreover, these NP-complete problems can have polynomial-time algorithms, if a solution is developed to an NP-hard problem in polynomial time.

Hence, it is to be noted that all NP-complete problems can be considered as NP-hard problems, but some NP-hard problems are not considered as NP-complete problems.

Q) Explain the P, NP, NP-Hard and NP- complete classes with suitable examples.

a) P Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a "yes" or "no" answer) that can be solved by a deterministic machine (our computers) in polynomial time.

Features:

- The solution to **P problem**s is easy to find.
- **P** is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

Most of the coding problems that we solve fall in this category like the below.

1. Calculating the greatest common divisor.
2. Finding Minimum spanning Tree
3. Merge Sort

b) NP Class

The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine (note that our computers are deterministic) in polynomial time.

Features:

- The solutions of the NP class might be hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.

- Problems of NP can be verified by a deterministic machine in polynomial time.

This class contains many problems that one would like to be able to solve effectively:

1. Boolean Satisfiability Problem (SAT).
2. Hamiltonian Path Problem.
3. Graph coloring.

c) NP-hard class

An NP-hard problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.

Features:

- All NP-hard problems are not in NP.
- It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.
- A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Some of the examples of problems in Np-hard are:

1. Halting problem.
2. Qualified Boolean formulas.
3. No Hamiltonian cycle.

d) NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP.

Features:

- NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.
- If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem in polynomial time.

Some example problems include:

1. Hamiltonian Cycle.
2. Satisfiability.
3. Vertex cover.

Q) .Give examples of some deterministic algorithms. Justify.

A deterministic algorithm is one that, given a particular input, will always produce the same output following the same sequence of computational steps.

1. Binary Search Algorithm

- **Justification**:
  Binary search follows a fixed sequence of steps to divide the sorted array and locate the element. For the same input, the path of comparisons and the final result will always be the same.
- **Time Complexity**: O(log n)

2. Dijkstra's Algorithm

- **Justification**:
  It finds the shortest path from a source node to all other nodes in a weighted graph (with non-negative weights). The same path will always be produced for the same input graph.
- **Time Complexity**: $O(V^2)$ or $O((V + E) \log V)$ using a min-heap

3. Merge Sort

- **Justification**:
  It follows a fixed divide-and-conquer approach. For the same array input, it will always split and merge in the same way, producing the same sorted array.
- **Time Complexity**: O(n log n)

4. Euclidean Algorithm (GCD)

- **Justification**:
  Used to compute the greatest common divisor of two numbers using fixed subtraction/modulo operations. The steps are predictable and consistent.
- **Time Complexity**: O(log min(a, b))

5. Prim's Algorithm

- **Justification**:
  A deterministic approach to find the minimum spanning tree of a weighted graph. It always selects the minimum weight edge connecting a visited node to an unvisited node.
- **Time Complexity**: $O(V^2)$ or $O((V + E) \log V)$

Q) Write about non deterministic algorithms and choice, failure and success functions with search example.

The algorithm in which every operation is uniquely defined is called deterministic algorithm.

The algorithm in which every operation may not have unique result, rather there can be specified set of possibilities for every operation. Such an algorithm is called non-deterministic algorithm.
Non-deterministic means that no particular rule is followed to make the guess.

The non-deterministic algorithm is a two-stage algorithm:

1. **Non-deterministic (Guessing) stage** –
   Generate an arbitrary string that can be thought of as a candidate solution.
2. **Deterministic ("Verification") stage** –
   In this stage, it takes as input the candidate solution and the instance to the problem and returns *yes* if the candidate solution represents the actual solution.

Example 11.1 Consider the problem of searching for an element x in a given set of elements $A[1:n]$, $n \geq 1$. We are required to determine an index j such that $A[j] = x$ or $j = 0$ if x is not in A. A nondeterministic algorithm for this is Algorithm 11.1.

```
1   j := Choice(1, n);
2   if A[j] = x then {write (j); Success();}
3   write (0); Failure();
```

Algorithm 11.1 Nondeterministic search

In the above given non-deterministic algorithm there are three functions used -

1) Choose - Arbitrarily chooses one of the element from given input set.

2) Fail - Indicates the unsuccessful completion.

3) Success - Indicates successful completion.

The algorithm is of non-deterministic complexity O(1), when A is not ordered then the deterministic search algorithm has a complexity Ω (n).

Some more example are as follows

```
1   Algorithm NSort(A, n)
2   // Sort n positive integers.
3   {
4       for i := 1 to n do B[i] := 0; // Initialize B[ ].
5       for i := 1 to n do
6       {
7           j := Choice(1, n);
8           if B[j] ≠ 0 then Failure();
9           B[j] := A[i];
10      }
11      for i := 1 to n − 1 do  // Verify order.
12          if B[i] > B[i + 1] then Failure();
13      write (B[1 : n]);
14      Success();
15  }
```

Algorithm 11.2 Nondeterministic sorting

Because we're in the non-deterministic model, the "guess" (Choice) is treated as a single step. Thus NSort runs in linear time.

The fourth line executes in O(n), and fifth line in $O(n^2)$ time

The 12th line takes another O(n) times.

Total time: O(n)+ O(n)+ $O(n^2)$

Q) Explain the strategy to prove that a problem is NP hard.

The strategy for proving the problem is NP-hard is as follows,

Step-1

First select the problem L_1 that is a problem of type NP-hard.

Step-2

Explain the derivation of an instance I' of L_2 from I of L_1. This is done in order to provide solutions for I of L_1 from the solution of I' from L_2.

Step-3

From step-2, it can be concluded that $L_1 \propto L_2$.

Step-4

From transitive property (i.e., step-1 and step-3), it can be concluded that L_2 is a problem of type NP-hard.

Q) What is Satisfiability problem?

Satisfiability (often abbreviated SAT) is the problem of determining whether there exists an interpretation that satisfies the formula. In other words, it establishes whether the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to true.

Example:

Consider the Boolean formula: $F = A \wedge \neg B$

This formula is satisfiable because if we assign $A = \text{TRUE}$ and $B = \text{FALSE}$, then F evaluates to $\text{TRUE} \wedge \neg\text{FALSE} = \text{TRUE} \wedge \text{TRUE} = \text{TRUE}$.

Consider another formula: $G = A \wedge \neg A$

This formula is unsatisfiable because no matter whether A is assigned TRUE or FALSE, $A \wedge \neg A$ will always evaluate to FALSE.

Q) Briefly explain Cooks-theorem. Give the applications of Cook's theorem.

Cook's theorem states that if $P = NP$, then satisfiability lies in P.

It is known that satisfiability is in NP. Hence, if $P = NP$ then satisfiability is in P.

And also vice versa i.e., if satisfiability is in P, then $P = NP$.

To prove this, $Q(A, I)$ has to be formulated from polynomial time nondeterministic algorithm A and input I.

When the length of input 'I' is n and the time complexity of algorithm 'A' = $P(n)$.

Then the length of $Q \Rightarrow O(p^3(n) \log n) = O(p^4(n))$. The deterministic algorithm determines Q and whether state Q is satisfiable or not.

If $O(q(m))$ is time required to formulate length m is satisfiable, then the complexity of the algorithm O is, $O(p^3(n) \log n + q(p^3(n) \log n)$.

If P, is satisfiable then the complexity of polynomial time function will to $O(r(n))$.

Hence, if satisfiability is fourd P, then a deterministic 'D' in P can be obtained for each of the nondeterministic algorithm A is in NP.

Thus, if satisfiability is in P then,

$$P = NP.$$

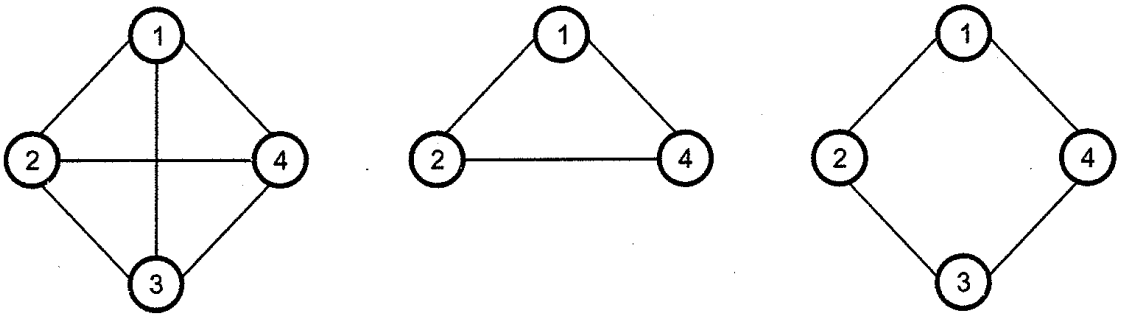
Applications:

Its applications span various areas, including problem classification, reduction techniques, algorithm design, and more.

Q) Clique Decision Problem (CDP)

Clique is nothing but a maximal complete subgraph of a graph G. The graph G is a set of (V, E) where V is a set of vertices and E is a set of edges. The size of Clique is number of vertices present in it.

For example :



Graph G Clique size = 3 Clique size = 4

Note that Clique of size = 3 and 4 are complete subgraph of graph G.

Max Clique Problem - It is an optimization problem in which the largest size Clique is to be obtained.

In the decision problem of Clique, we have to determine whether G has a Clique of size at least k for given k.

Let, DClique (G, k) is a deterministic decision algorithm for determining whether graph G has Clique or not of size atleast k. Then we have to apply DClique for k = n, n – 1, n – 2,... until the output is 1. The max Clique can be obtained in time $\leq n \cdot F(n)$

where F(n) is the time complexity of DClique. If Clique decision problem is solved in **polynomial time** then **max Clique** problem can be solved in **polynomial time.**

Theorem : The Clique Decision Problem (CDP) is NP-complete.

Proof : Let, F be a formula for CNF which is satisfiable.

$$F = C_1 \wedge C_2 \wedge \ldots C_k$$

where C is a clause. Every clause in CNF is denoted by a_i where $1 \leq i \leq n$. If length of F is F and is obtained in time O(m) then we can obtain polynomial time algorithm in CDP.

Let us design a graph G = (V, E) with set of vertices

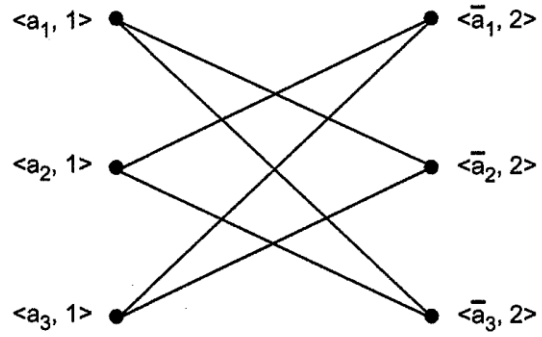
$V = \{ <\sigma, i> | \sigma \text{ is a literal in } C_i \}$ and set of edges

$E = \{ (<\sigma, i>, <\delta, j>) | i \neq j \text{ and } \sigma \neq \bar{\delta} \}$

For example

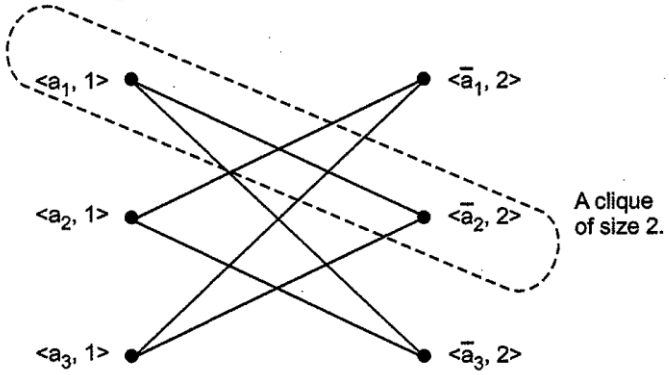
$$F = \left(\underbrace{(a_1 \vee a_2 \vee a_3)}_{C_1} \right) \wedge \left(\underbrace{(\bar{a}_1 \vee \bar{a}_2 \vee \bar{a}_3)}_{C_2} \right)$$

The graph G can be drawn as follows



The formula F is satisfiable if and only if G has a clique of size > k. The F will be satisfiable only when at least one literal σ in C_i is true.

Thus the graph as six cliques of size two. Note that F is satisfiable as well.



As CNF satisfiability is NP complete CDP is also NP complete.

Q) Traveling Salesperson Decision Problem (TSP).

The travelling salesman decision problem can be stated as follows - Given n number of cities and a travelling salesman has to visit each city. Then find the shortest tour so that the travelling salesman can visit each and every city only once.

Theorem : Prove that a directed Hamiltonian cycle ∝ the travelling salesman decision problem.

Proof

The directed Hamiltonian cycle is a cycle in which each vertex of a directed graph is visited only once. To prove that directed Hamiltonian cycle (DHC) ∝ Travelling Salesperson Problem (TSP) we will apply following steps -

If there are two problems A and B and if we wish to prove that B is NP complete then -

1) Pick up a known NP complete problem A

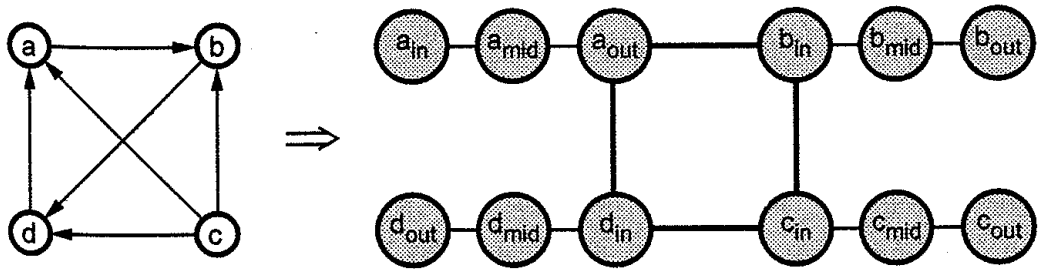
2) Reduce A to B. This can be done in following steps

a) Describe a transformation that maps the instance of A to instance of B in such a manner that "yes" for B = "yes" for A.

b) Prove that this transformation runs in polynomial time.

3) This proves that problem B is also NP complete.

To prove that DHC ∝ TSP we will reduce Directed Hamiltonian cycle to undirected Hamiltonian Cycle. The conversion is as follows -



In this undirected graph if there exists a Hamiltonian cycle, then that denotes visiting cities in the order of nodes being visited in Hamiltonian cycle.

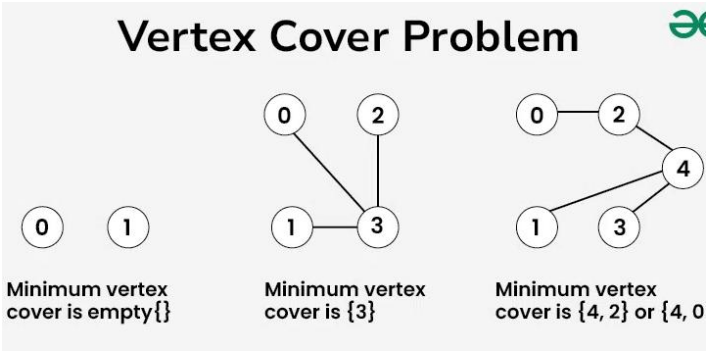
If no Hamiltonian cycle exists that means TSP has no solution. The Hamiltonian cycle problem is NP complete. This proves that TSP is also NP complete.

Q) Write short note on the following:

a) Vertex cover b) Independent set c) Set cover d) Steiner tree

Answer:

A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either 'u' or 'v' is in the vertex cover. Although the name is Vertex Cover, the set covers all edges of the given graph. *Given an undirected graph, the vertex cover problem is to find minimum size vertex cover.*
The following are some examples.



b) Independent Set: In a graph, an independent set is a subset of vertices where no two vertices in the set are connected by an edge.

c) Set Cover: Given a universe of elements U and a collection of sets S1,S2,S3……Sn where each set is a subset of U, a set cover is a subset of the collection S such that every element in U is covered by at least one set in the subset.

d) Steiner Tree: In a weighted graph, a Steiner tree is a minimum-weight tree that connects a specified set of "terminal" vertices. The Steiner tree can include additional vertices, called "Steiner points", that are not terminals but help to minimize the total weight of the tree