CHAPTER 9

Introduction To NP Completeness

Syllabus :

- The class P and NP
- Polynomial reduction
- NP- Completeness Problem
- NP-Hard Problems
- Travelling Salesman problem
- Hamiltonian problem
- Approximation algorithms

9.1 The Class P and NP Problems

GTU - Dec. 2010, May 2011, Dec. 2011, May 2012, Dec. 2012, May 2013, Dec. 2013, May 2014, May 2015

We can say that algorithm solves give problem in polynomial time if worst case running time of algorithm belong to O(p(n)), where p(n) is polynomial in n, where n indicates problem size. If problem is solvable in polynomial time, it is called **tractable**. Problems which cannot be solved in polynomial time are called **intractable**.

Any problem can be classified in one of the following category :

P Problem :

- P problems are set of problems that can be solved in polynomial time by deterministic algorithms.
- P problems are simple to solve and easy to verify
- Most of the problems we have discussed so far are P problems.

- It excludes all the problems which cannot be solved in polynomial time. Knapsack problem using brute force approach cannot be solved in polynomial time. Hence, knapsack is not P problem.
- There exist many important problems whose solution is not found in polynomial time so far, nor has it been proved that such solution dose not exists. TSP, Graph colouring, partition problem, knapsack etc are examples of such case
- Examples of P problems are searching an element in array (O(n)), inserting element at the end in linked list (O(n)), sorting data using selection sort($O(n^2)$), finding height of tree ($O(\log_2 n)$), sort data using merge sort($O(n\log_2 n)$) etc.

NP Problem :

- NP problems are such problems that can be solved in non deterministic polynomial time. NP does not mean non polynomial, it stands for Non Deterministic Polynomial time.
- Non deterministic algorithm operates in two stages.
- **Nondeterministic (guessing) stage :** For input instance I, some solution string S is generated, which can be thought as a candidate solution.
- **Deterministic (verification) stage :** I and S are given as a input to deterministic algorithm, which returns "yes" if S is a solution of input instance I.
- Solution to NP problems cannot be obtained in polynomial time, but given the solution, it can be verified in polynomial time.
- NP includes all problems of P, i.e. $P \subseteq NP$
- Examples of NP problems are, knapsack problem ($O(2^n)$), Travelling salesman problem (O(n!)), Tower of Hanoi ($O(2^n - 1)$), Hamiltonian cycle (O(n!)) are example of such problems.
- NP Problems are further classified in NP complete and NP hard categories.
- Fig. 9.1.1 shows the taxonomy of complexity classes

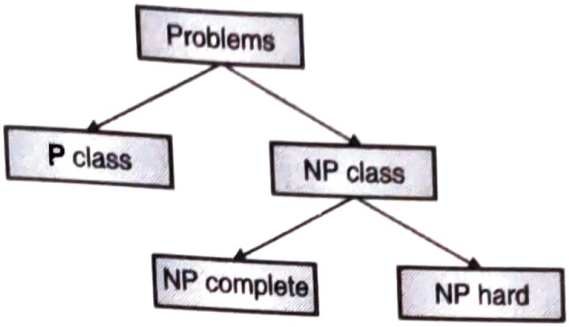


Fig. 9.1.1 : Taxonomy of complexity classes

9.1.1 Example of P Class Problem

Selection sort : We already have discussed the working principle of selection sort. Initially data is in random order. Selection sort divides the list in two parts from the beginning. In every iteration, size of sorted list is incremented by one and size of unsorted list reduces by one.

In each iteration, minimum element from unsorted list is replaced with the first element. To find minimum element from list of size n, we should perform (n – 1) comparisons. Algorithm for selection sort is described below :

Algorithm :

```
Algorithm SELECTION_SORT(A)
// A is unsorted input array of size n

for i ← 1 to n – 1 do
    min ← i
    for j ← i +1 to n do
        if ( A[j] < A[min]) do
            min ← j
        end
    end
swap(A[i], A[min])
end
```

Ex. 9.1.1 : Sort the letters of word "DESIGN" in alphabetical order using selection sort.

Soln. :

Following Fig. P. 9.1.1 shows the simulation to sort characters of word "DESIGN"

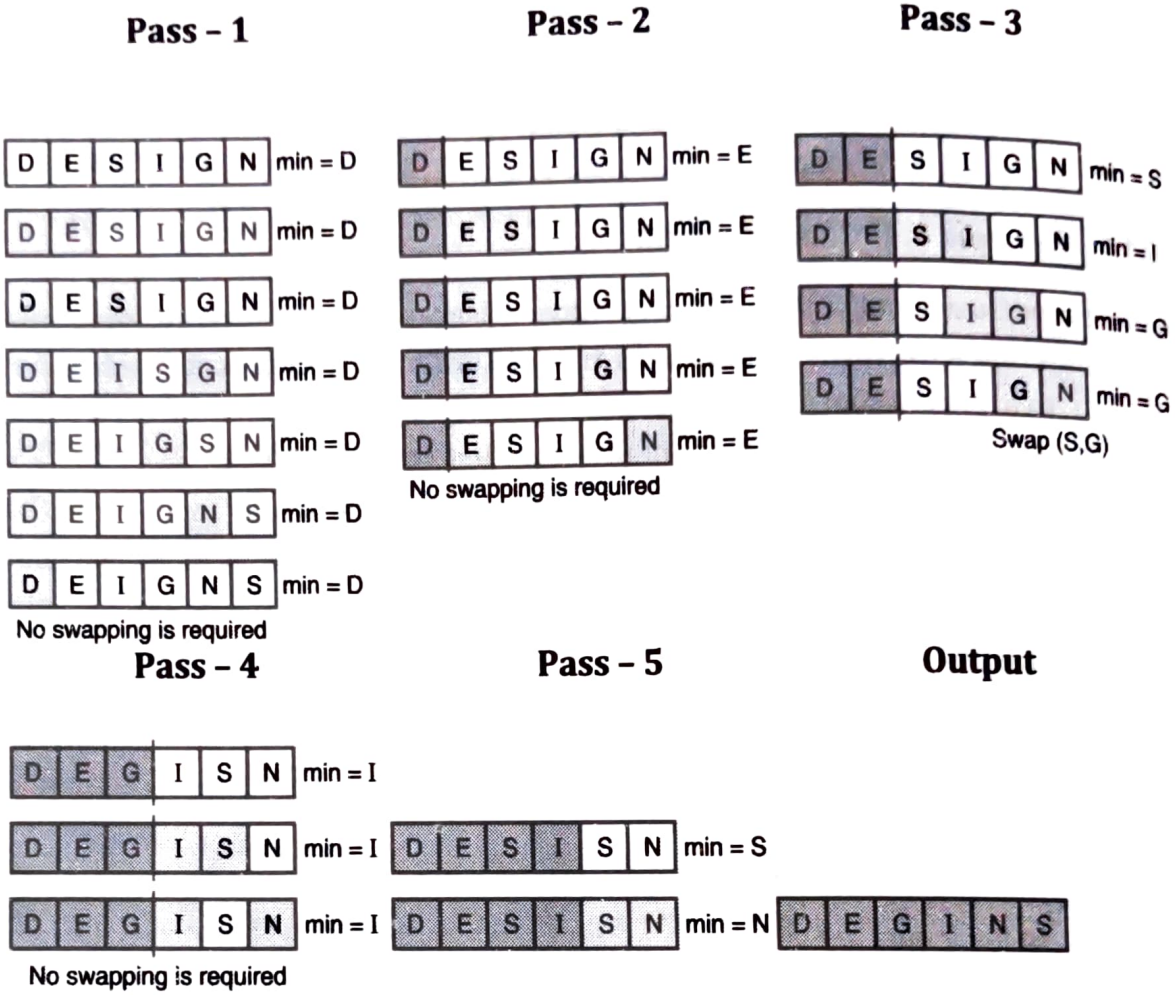


Fig. P. 9.1.1 Simulation of selection sort

Complexity analysis :

- Selection sort iterates both loops irrespective of input data pattern. Unlike bubble sort, selection sort cannot detect sorted sequence. So running time of selection sort in best, average and worst case is $O(n^2)$. We can come to this conclusion by adding number of comparisons just like bubble sort.
- In first iteration it performs $(n - 1)$ comparisons to find minimum element, in next iteration it does $(n - 2)$ comparisons and so on. Hence running time of selection sort is given as,

  $T(n) = (n - 1) + (n - 2) + ... + 3 + 2 + 1$

$T(n) = \Sigma (n - 1) = (n^2 - n) / 2 = O(n^2)$

So running time of selection sort is in polynomial.

9.1.2 Example of NP Class Problem

- **Travelling salesman problem :** This is well explored problem by mathematicians and researchers. Problem is stated as "Given n cities and distance between them, find out the path that covers all the cities with minimum distance."
- Consider the Fig. 9.1.2, where A, B, C, D, E indicates cities and weighted edge joining them indicates distance between two cities. Goal is to find a path that covers all the cities exactly once and gives minimum distance cost.

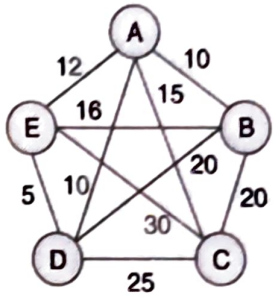


Fig. 9.1.2 Graph for TSP

- There exists many path (n! at least !) with different cost. Path A-B-C-D-E-A has cost 72. Path A-C-B-E-D-A has cost 66. If we enumerate all paths, we get n! such different paths. So finding minimum path does not belong to polynomial time, and hence it is NP problem.

9.2 Polynomial Reduction

GTU - May 2011, Dec. 2011, Dec. 2014

- Polynomial reduction is a way of solving problem A by the hypothetical routine for solving different problem B, which runs in polynomial time. Basically, polynomial reduction proves that problem A is not more difficult than the problem B.
- For example, we have some hypothetical algorithm, which can sort numerical data in some polynomial time. Input to algorithm can only be in numeric form. Suppose we have a new problem to sort names of the cities across country. What can be done? Suppose we don't have efficient algorithms to handle string data. We can apply some hashing function on city names to map them to numeric values. Now this is an identical to the first approach.
- Thus, polynomial reduction is the way of turning one problem into another problem whose solution can be found in polynomial time.
- Let us consider two decision problems A and B. Problem is called **decision problem** if its answer is "Yes" or "No". Reduction from A to B transforms input of A to equivalent input of B. So given an input x to A, polynomial

reduction algorithm produces intermediate result R(x) to problem B such that R(x) to B returns "yes" only if input x to A returns "Yes". Fig. 9.2.1 shows the scenario.

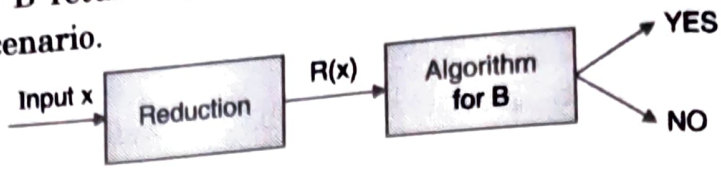


Fig. 9.2.1 : Polynomial reduction

9.3 NP- Completeness Problem

GTU - May 2011, Dec. 2013, May 2014

- Decision problem p is called NP complete if it has following two properties :
- It belongs to class NP
- Every other problem in NP can be transformed to P in polynomial time.
- These two facts prove that NP complete problems are the harder problems in class NP. They are often referred as NPC.

Factoid :

- If any NP complete problem belongs to class P, then P = NP
- However, solution of any NP complete problem can be verified in polynomial time, but cannot be obtained in polynomial time. NP complete problems are often solved using randomization algorithms, heuristic approach or approximation algorithms.
- Some of the well known NP complete problems are listed here :
  - Boolean satisfiability problem
  - Knapsack problem
  - Hamiltonian path problem
  - Travelling salesman problem
  - Sub set sum problem
  - Vertex cover problem
  - Graph coloring problem
  - Clique problem

9.4 NP-Hard Problems

GTU - May 2011, Dec. 2013, Dec. 2014

- Formally, a decision problem p is called NP hard, if every problem in NP can be reduced to p in polynomial time. NP hard is super set of all problems. NPC are in NP hard, but converse may not be true.
- NP hard problems are at least as hard as the hardest problems in NP. If we can solve any NP hard problem in polynomial time, we would be able to solve all the problems in NP in polynomial time. NP hard problems do not have to be in NP. Even they may not be decision problem.
- Subset sub problem, travelling salesman problem are NPC and also belong to NP hard. There are certain problems which belong to NP hard but they are not NP complete. A well known example of NP hard problem is Halting problem.
- Halting problem is stated as, "Given an algorithm and set of inputs, will it run forever ?" Answer to this question is Yes or No, so this is decision problem. There does not exist any known algorithm which can decide the answer for any given input in polynomial time. So halting problem is NP hard problem.
- Different mathematicians have given different relationship considering possibilities of P = NP and P ≠ NP. Relationship between P, NP, NP Complete and NP hard is described in below Fig. 9.4.1 assuming that P ≠ NP. This is widely accepted relationship among these complexity classes.

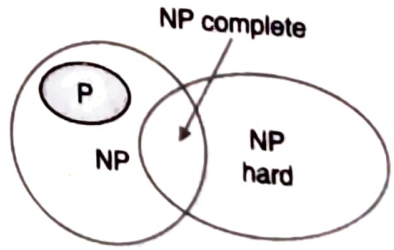


Fig. 9.4.1

9.4.1 Compare : NP Hard and NP Complete

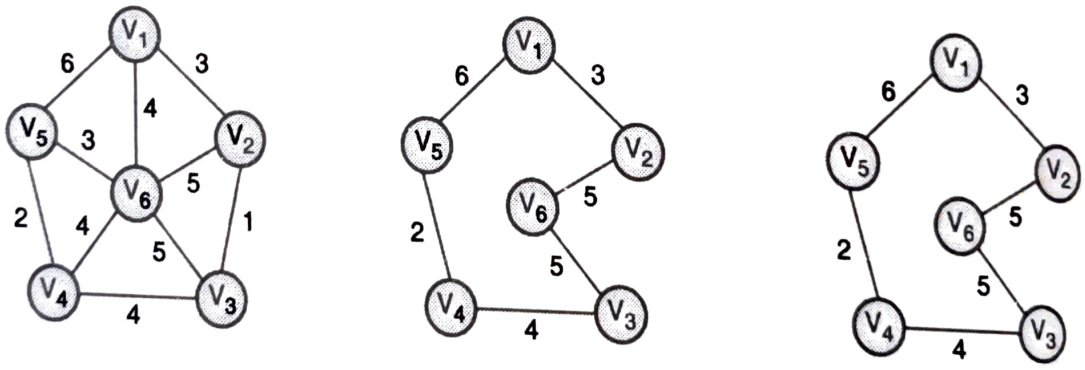
GTU - Dec. 2010

Sr. No.	NP Complete	NP Hard
1.	NP complete problems are decision problem.	NP hard problem might not be decision problem.
2.	NP complete problems are harder problems.	NP hard problems are hardest problem.
3.	NP complete problem are in NP.	NP hard problems may not be in NP.

Sr. No.	NP Complete	NP Hard
4.	Problem X is NP complete if NP problem Y is reducible to X in polynomial time.	Problem X is NP hard if NP complete problem Y is reducible to X in polynomial time.
5.	Ex. 3 SAT problem.	Ex. Halting problem.

9.5 Travelling Salesman Problem

- **Problem :** "Given n cities and distance between each pair of cities, salesman has to visit all n cities with the objective of minimizing travelling distance. Salesman has to come back to the start city after visiting all cities exactly once."
- TSP problem is special case of Hamiltonian cycle problem. With n cities, n! different paths are possible. Finding best path using brute force approach leads to O(n!) time. So TSP is NP problem.



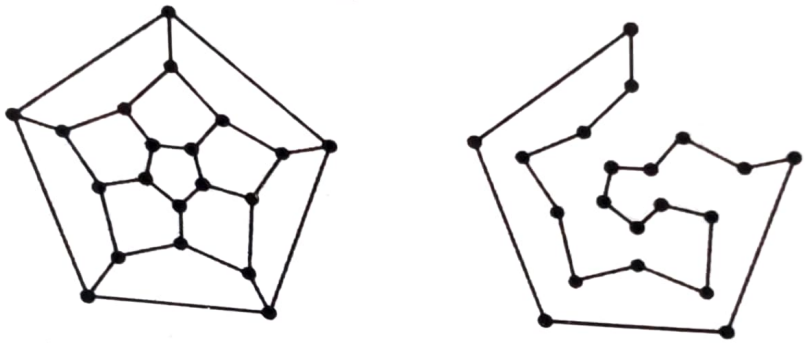
(a) TSP Graph (b) Path length: 25 (c) Path length: 17

Fig. 9.5.1: TSP and its two solutions

- Given some path sequence S, verification of member ship of all vertices V in S is done in polynomial time. Comparison of two paths and selecting best of two can also be done in polynomial time, but finding shortest path from given Fig. 9.5.1 in polynomial time is not possible. So travelling salesman problem is NP complete problem.
- Nearest neighbour or minimum spanning tree approach are good approximations of this optimization problem. Both of this approach provides quick sub optimal solution.

9.6 Hamiltonian Problem

- Undirected graph G = (V, E) is called Hamiltonian if there exists a cycle which covers all the vertices in V exactly once except starting vertex. Following Fig. 9.6.1 describes the Hamiltonian cycle.
- Hamiltonian path is formed by removing one edge from the Hamiltonian cycle. Hamiltonian path visits every vertex in a graph but does not form cycle. To construct Hamiltonian cycle from Hamiltonian path, we may require adding more than one edge.



(a) Input graph (b) Output cycle

Fig. 9.6.1 : Graph and its Hamiltonian cycle

- This problem is well known as Hamiltonian cycle problem. To check if there exist a Hamiltonian cycle, permutations of all vertices are examined. Such n! different permutations are possible for graph having n vertices. Checking the graph for Hamiltonian cycle is very slow using brute force approach. We cannot find the solution in deterministic polynomial time. This is NP problem.

Few examples of Hamiltonian graphs :

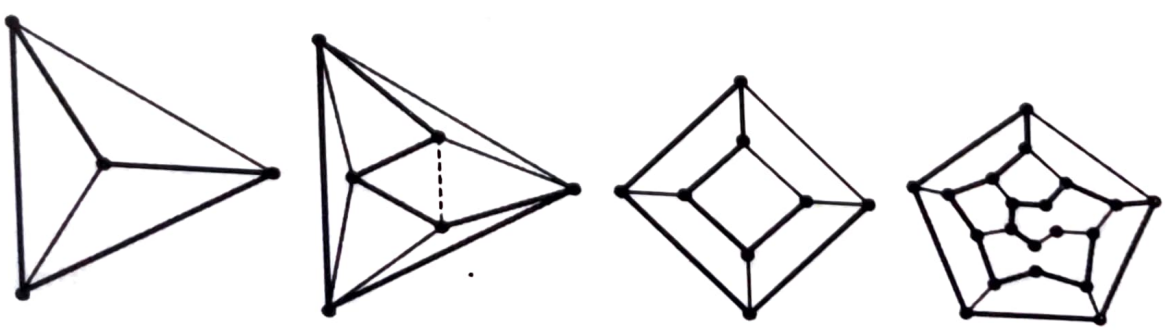


Fig. 9.6.2

9.7 Approximation Algorithms

- Many real time problems are NP complete. Their running time is not acceptable for practical implementation. To make them useful, we have certain ways:

Exponential time algorithms :

An optimal solution is available, but can handle only small input.

General optimization methods :

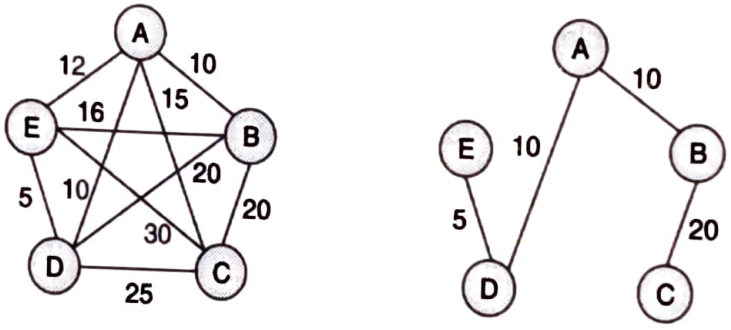
Genetic algorithms, branch and bound techniques, neural network etc are general optimization methods. But it is quite difficult to prove how good they are.

Approximation algorithms :

- Approximation algorithms provide solution quickly, but do not guarantee the optimal solution. However, solutions of approximation algorithms can be proved close to optimal one.
- Finding optimal solution for NP complete is not polynomial bounded. Solving NPC for best solution may not be acceptable for real time scenario. Approximation algorithms apply some heuristic to cut out the further computation in each step.
- Heuristic enables existing algorithm to run quickly for large instance. However heuristic may not work equally efficiently for all algorithms.
- Constraints in approximation algorithms are somewhat relaxed. Requirement of optimal solution is reduced to feasible solution which is close enough to optimal solution. Feasible solution near to optimal solution is called **approximate solution**.
- For many NP problems, where time is important, approximate solution provides good balance between time and accuracy. Consider the following cases,
  - In TSP, optimization problem is to find shortest path, whereas approximation problem is to find short path.
  - In vertex cover problem, optimization problem is to find fewest vertices, whereas approximation problem is to find few vertices.

Travelling salesman problem :

- Travelling salesman problem is stated as, "Given set of n cities and distance between each pair of cities, find the minimum length path such that it covers each city exactly once and terminates the tour at starting city."
- It is not difficult to show that this problem is NP complete problem. There exists n! paths, search of optimal path becomes very slow when n is considerable large.
- Approximate solution to this problem is to find minimum spanning tree of given graph. This approach would greatly reduce the time. Consider the following example :



(a) Grpah G for TSP **(b) MST of graph G**

Fig. 9.7.1 : TSP graph and its minimum spanning tree

Some random path length : E-C-D-B-A-E: 30 + 25 + 20 + 10 + 12 = 97

Path length using MST : E-D-A-B-C-E: 5 + 10 + 10 + 20 + 30 = 65

9.8 University Questions

Dec. 2010

Q. 1 Explain P and NP Problems giving examples. **(Section 9.1)** **(3 Marks)**

Q. 2 Compare NP-Hard with NP-Complete problems. **(Section 9.4.1)** **(3 ½ Marks)**

May 2011

Q. 3 What is polynomially Turing reducible problem? Explain with example how problem A can be polynomially Turing reduced to problem B. **(Section 9.2)** **(6 Marks)**

Q. 4 Define P, NP, NP complete and NP-Hard problems. **(Sections 9.1, 9.3 and 9.4)** **(4 Marks)**

Dec. 2011

Q. 5 Explain different class of problems. **(Section 9.1)** **(3 ½ Marks)**

Q. 6 Explain polynomial reduction. **(Section 9.2)** **(3 ½ Marks)**

May 2012

Q. 7 Explain in Brief: P Problem, NP Problem. **(Section 9.1)** **(4 Marks)**

Dec. 2012

Q. 8 Explain in Brief: P Problem, NP Problem. **(Section 9.1)** **(4 Marks)**

May 2013

Q. 9 Explain P and NP Problems. **(Section 9.1)** **(7 Marks)**

Dec. 2013

Q. 10 Define P, NP, NP complete and NP-Hard problems.
(Sections 9.1, 9.3 and 9.4) **($3\frac{1}{2}$ Marks)**

May 2014

Q. 11 Write a brief note on NP-completeness and the classes-P, NP and NPC (Pending).
(Sections 9.1 and 9.3) **(7 Marks)**

Dec. 2014

Q. 12 Explain in Brief : NP Hard Problem, Polynomial reduction.
(Sections 9.2 and 9.4) **(7 Marks)**

May 2015

Q. 13 Explain class P and class NP. **(Section 9.1)** **(4 Marks)**

□□□