UNIT 1 INTRODUCTION TO COMPLEXITYCLASSES

Structure **Page Nos.**

1.0 INTRODUCTION

Until now we have studied a large number of problems and developed efficient solutions using different problem solving techniques. After developing solutions to simple algorithms (sorting, polynomial evaluation, exponent evaluation, GCD) , we developed solutions to more difficult problems (MCST, single source shortest path algorithms, all pair shortest path algorithms, Knapsack problem, chained matrix multiplications, etc) using greedy, divide and conquer technique and dynamic programming techniques. We also formulated some problems as optimization problems, for example, Knapsack and a single source shortest path algorithm. But so far we have not made any serious effort to classify or quantify those problems which cannot be solved efficiently. In this unit as well as in the subsequent unit we will investigate a class of computationally hard problems. In fact, there is a large number of such problems. In the last unit of this block, we will show how a computationally hard problem can be solved through an approximation algorithm.

The structure of this unit is as follows as: In section 1.2 we provide a background for understanding different classes of problems which are introduced in section 1.3. The section highlights differences between tractable and intractable problems, optimization problems and decision problems and deterministic and non-deterministic algorithms. Section 1.4 introduces the first NP Complete problem which is a basis of all other NP complete problems. Problems from graph theory and combinatorics can be formulated as a language recognition problem, for example, pattern matching problems which can be solved by building automata. This issue is addressed in 1.5. The key issues discussed in this unit are summarized in the section1.6 and the solutions to CYPs are included in the section 1.7.

1.1 OBJECTIVES

After studying this unit, you should be able to:

- Differentiate between P, NP, NP-Complete, and NP-Hard problems
- Differentiate between tractable and Intractable problems, optimization and decision problems and deterministic and non-deterministic algorithms
- List P, NP and NP – Complete classes of problems
- Define the concept of reduction in NP-Complete problems
- Explain CNF Satisfiability Problem

1.2 SOME PRELIMINARIES TO COMPLEXITY CLASSES

In this section, some preliminaries are presented to help you understand different class complexities.

1.2.1 Tractable Vs. Intractable Problems

The general view is that the problems are **hard** or **intractable** if they can be solved only in exponential time or factorial time. The opposite view is that the problems having polynomial time solutions are **tractable or easy** problems. Although the exponential time function such as 2^n grows more rapidly than any polynomial time algorithms for an input size n, but for small values of n, intractable problems for with exponential time bounded complexity can be more efficient than polynomial time complexity. But in the asymptotic analysis of algorithm complexity, we always assume that the size of n is very large

It is to be kept in mind that intractability is a characteristic of a problem. It is not related to any problem solving technique. To explain this concept, let us take an example of a chained matrix multiplication problem which was examined earlier. The brute force approach to the solution of this problem is intractable but it can be solved in polynomial time through dynamic programming technique

Whereas intractability is concerned, usually there are two types of problems in this class. The problems which generate a non-polynomial amount of outputs such as TSP and Hamiltonian cycle problem. Consider any complete graph with n number of vertices in which every vertex is connected to other vertices that would generate (n-1)! cycles for examining in both the problems, belong to the first category. Knapsack problem and the sum of subsets problems are also in this category. In the second category, the most well-known problem is Halting problem. The input to this problem is any algorithm which has some input data. Now it is to decide whether the algorithm will ever stop with this input. Alan Turing proved that this type of a problem is **undecidable.**

1.2.2 Optimization Problems Vs. Decision Problems

The Knapsack problem and the single source shortest path problem have been formulated as optimization problems. The former is defined as maximization optimization problem (i.e. maximum profit) and the latter is defined as minimization optimization problem (minimum cost of a path). Depending upon the problem, the optimal value is minimum (single source shortest path) or maximum (Knapsack problem) which is selected among a large number of candidate solutions. Optimization problems can be formally defined as:

An Optimization problem is one in which we are given a set of input values, which are required to be either maximized or minimized w. r. t. some constraints or conditions. Generally an optimization problem has n inputs (call this set as **input domain** or **Candidate set**, C), we are required to obtain a subset of C (call it ***solution set***, S where, $S \subseteq C$) that satisfies the given constraints or conditions. Any subset S⊆C, which satisfies the given constraints, is called a ***feasible*** solution. We need to find a feasible solution that maximizes or minimizes a given objective function. The feasible solution that maximizes or minimizes a given objective function, is called an **optimal solution**. For example, find the shortest path in a graph, find the minimum cost spanning tree, 0/1 knapsack problem and fractional knapsack problem.

Decision Problems:

There is a corresponding decision problem to each optimization problem. Unlike an optimization problem, a decision problem outputs a simple “yes” or “no” answer. Most NP-complete problems that we will discuss will be phrased as decision problems. For example, rather than asking, what is the minimum number of colors needed to color a graph, instead we would phrase this as a decision problem: Given a graph G and an integer k, is it possible to color G with k colors ?

The following examples distinguish between two types of problems.

Example 1: Traveling Salesperson Problem

TSP Optimization Problem- Given a complete graph G = (V,E) and edge weight, the optimization problem find out an optimal path covering all the vertices only once except the starting vertex with the least cost.

TSP Decision Problem – It can be formulated as: Given a complete weighted graph G = (V,E) and an integer K, is there a cycle/path comprising all the vertices only once except the starting vertex and has a total cost ≤ K.

Ex 2: 0-1 Knapsack Problem

The 0-1 Knapsack Optimization Problem- Given the number of items, profit and weight of each item and maximum weight of a Knapsack, the 0-1 Knapsack optimization problem determines the maximal total profit of items that can be placed in the Knapsack

The 0-1 Knapsack Decision Problem-The objective of the decision problem is to determine whether it is possible to earn a profit not less or equal to some amount P while not exceeding the maximum Knapsack capacity.

Example 3: Graph Coloring Problem

Graph Coloring Optimization problem- Given a graph G= (V,E), the graph coloring optimization problem determines minimum numbers of colors needed to color a graph so that no two adjacent vertices of the graph are having the same color.

Graph Coloring Decision Problem- It determines whether a given graph can be colored in at most M colors such that no two adjacent vertices of the graph are having the same color

Example 4: Clique optimization and Decision problem

Given an undirected graph, G= (V, E) clique is a subset of vertices W of V such that all the vertices in W are adjacent to each other consider the following graph.

In this example {V2, V3 and V5} in a clique where as {V2, V3 and V4} is not a clique because V2 is not adjacent to V4. A maximal clique contains maximum number of vertices. For ex. In the given graph a maximal clique is {V1, V2, V3, V5}

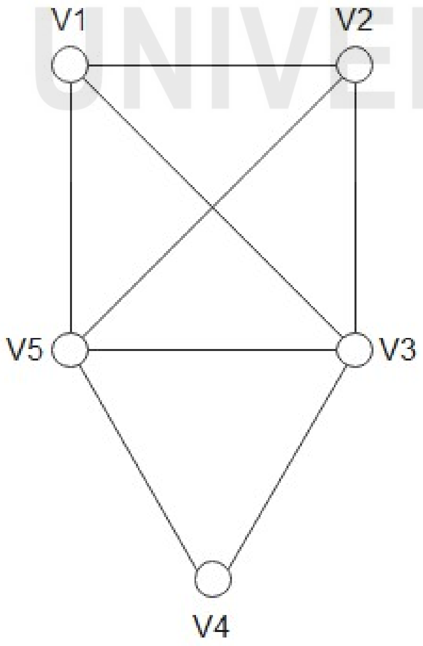

Figure 1: A Problem Instance of a Clique

The clique optimization problem finds out the size of a maximal clique for a given graph. Given a graph and some integer value K, the clique decision problem determines whether there is a clique containing at least K vertices.

We observe that an optimization problem can be formulated as a related decision problem by imposing a bound on the value of optimization. For ex. Graph coloring decision problem determines whether a graph can be colored in at most m colors.

1.2.3 Deterministic Vs. Nondeterministic Algorithms

For a given input, a deterministic algorithm will always produce the same output on an input going through the same sequence of steps, even when run multiple times. All the algorithms that we have seen so far are deterministic.

A non-deterministic algorithm behaves in a completely different way. There may be multiple next steps possible after any given step and the algorithm is allowed to choose any of them in an arbitrary manner. Thus, even for a fixed input, a non-deterministic algorithm may proceed through different sequences of steps on different runs, and may even produce different outputs. It should be understood that non-determinism is a totally abstract theoretical concept and such algorithms do not naturally occur in nature; however, such algorithms have been found to be useful in understanding the behaviour of deterministic algorithms.

1.3 Introduction to P, NP,NP Hard and NP-Complete Problems

The theory of NP-completeness identifies a large class of problems which cannot be solved in polynomial time. Categories the problems into three classes namely P (Polynomial), NP (Non-deterministic Polynomial), and NP complete.

1.3.1 P Class

Almost all problems which we examined so far, belong to polynomial class of problems. For examples, finding out a key in an array, Greatest Common Divisor of two integers, matrix multiplication, sorting algorithms, finding a shortest path in a weighted graph, finding a minimum cost spanning tree, problems implemented through dynamic programming such as, All Pairs Shortest Path Algorithm, Chained Matrix Multiplication and Optimal Binary Search Tree and Binomial Coefficient can be solved in polynomial time.

An algorithm solves a problem in polynomial time if its worst -case time complexity belongs to O(p(n)) where n is a size of a problem and p(n) is polynomial of the problem's input size n. Problems can be classified as tractable and intractable problems. Problems with polynomial time solutions are called **tractable,** problems which do not have polynomial time solutions are called **intractable.**

Problems with the following worst-case time complexities belong to polynomial class of problems:

5n, $5n^3$ +10n, n log^n

Problems with the following worst- case time complexities do not belong to polynomial class of problems:

2^n, n!, $2^{\sqrt{n}}$

Informally P can be defined as a set of problems that has polynomial time solutions. A more formal definition of P is that it includes all decision problems that has yes/no answers. Finally, we can define class P as a class of decision problems that can be solved by deterministic algorithms in polynomial time.

As we have seen in the previous section that many optimization problems can be formulated as decision problems which are easier to implement. For example, consider a graph coloring problem. Instead of asking a minimum number of colors needed to color the vertices of a graph having two adjacent vertices of a graph in different colors, we should ask whether a graph can be colored in no more than m colors where m = 1, 2, …. without coloring the adjacent vertices of a graph in the same color.

1.3.2 NP Class

NP stands for nondeterministic polynomial. It is a class of decision problems that can be solved by nondeterministic polynomial algorithms. Unlike deterministic algorithms which executes the next instruction which is unique because there is no choice for execution of any other instruction, nondeterministic algorithms have a choice among the next instructions.

How do we know that a problem is in NP? The simplest way is to formulate a problem as a decision problem i.e., yes/no question and verify the yes instance of a solution in polynomial time. Most decision problems are in NP. The NP class includes all the problems that have a polynomial time deterministic algorithm. This expression is true. The reason is that if the problem belongs to P class, we can apply deterministic polynomial algorithm to solve / verify it in polynomial time.

An NP algorithm consists of two stages:

(i) Guessing Stage (Nondeterministic stage): Given a problem instance, in this stage a simple string S is produced which can be thought of as a guess (candidate solution) to the problem instance.

(ii) Verification Stage (Deterministic stage). Input to this stage is the problem instance, the distance d and the string S. A deterministic algorithm takes these inputs and outputs yes if S is a correct guess to the problem instance and stops running. In case S is not a correct guess, then the deterministic algorithm outputs no or it may go to an infinite loop and does not halt.

A nondeterministic algorithm is said to be nondeterministic polynomial if the verification stage is completed in polynomial time.

A famous decision problem which is not in NP problem is the **halting** problem, which is defined as follows:

Given an arbitrary program with an arbitrary data, will the program terminate or not? This type of problem is called undecidable problems. Undecidable problems cannot be solved by guessing and checking. Although they are decision problem somehow they cannot be solved by exhaustively checking the solution space

Now Let us write a nondeterministic algorithm for TSP (Traveling Salesperson Problem):

(i) **Nondeterministic Stage**: The following table displays the results of some input strings S generated at the nondeterministic stage for the problem instance graph given in figure 1with the total distance d value that is claimed to be a tour not greater than d(i.e. 18)and passed to the function *verify* as an input.

String S	*Output*	*Reasons*
[A,B,C,D,A]	False	Total weight of a tour(S string)is greater than 18
[A,D,B,C,A]	False	S is not a correct tour
[A,C,B,D,A]	True	S is a tour with weight not exceeding 18

The third string is the correct guess for the tour. Therefore, the nondeterministic stage of the algorithm is satisfied. In general, function *verify*at the verification stage return True if the guess for a particular instance is correct ,otherwise the *verify* function does not return true.

(ii) **Deterministic Stage** : In this stage we write a*verify* function that verifies whether the string S produced is a tour with weight no greater than 18.

```
Pseudocode

Boolean *verify*(weighted graph G, d, S)
{
  if ( S is a tour and the total weight of the edges in S ≤ d)
    return True;
else
return False;
}
```

The algorithm first checks to see whether S is indeed a tour. If the sum of weights is no greater than d, it returns "True".

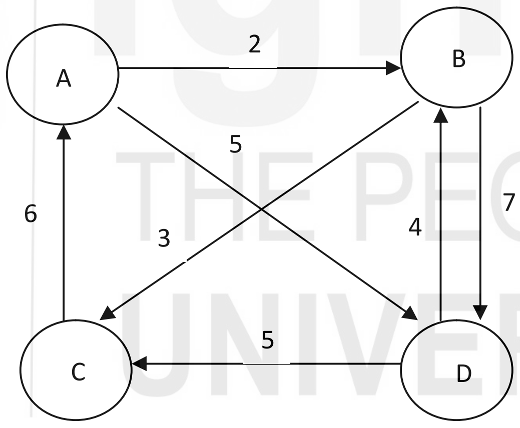

Figure 2: A Problem Instance for TSP

What other decision problem are in NP? There are many such problems. Just to add few examples here: Knapsack, Graph coloring problem, clique, etc.

- Finally there are a large number of problems that are trivially in NP. It means that every problem in P is also in NP as shown in the following diagram(figure3):

T.

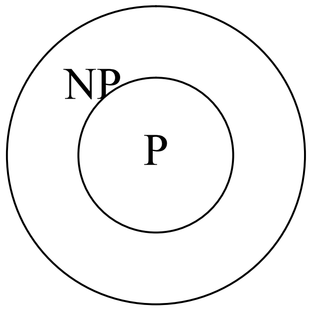

Figure 3: P as a Subset of NP

In this figure NP contains P as a proper subset. However, this is not a case all the time. So far, no proof has been provided that there is a problem in NP that is not in P. Therefore (NP-P) may be completely empty.

Figure 2: Set of all decision problem.

The important issue in theoretical computer science is whether P = NP or P≠NP . If P=NP, we would have polynomial time solution from any known decision problems. For the later case, when P≠NP, we would have to find a problem in NP that is not in P. But to show P=NP, we have to find a polynomial time algorithm for each problem is NP, or find a polynomial-time algorithm for any NP-complete problem.

Check Your Progress – 1

Q1: List the problems belonging to polynomial class

Q2: Formulate Graph Coloring Problem as an optimization and decision problems

Q3: What is Halting Problem?

1.3.3 NP Complete Problems

NP Complete Problems – NP complete problems are the most difficult problem, also called hardest problems in the subset of NP-class. It has been shown that there are large class of problems in NP which are NP-Complete problems as shown in figure 3

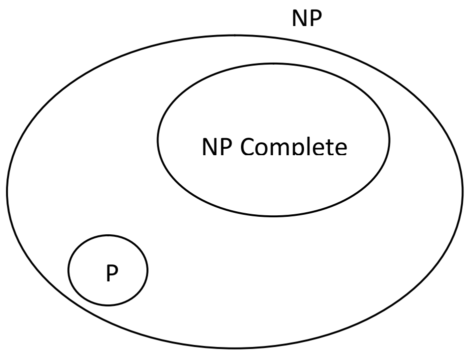

Figure 4: NP Problems containing both P and NP-Complete Problems

The set of NP-Complete problems keeps on growing all the time. It includes problems like Traveling Salesperson Problem, Hamiltonian Circuit Problem, Satisfiability Problem, etc. The list is becoming larger. The common feature is that there is no polynomial time solution for any of these problems exists in the worst cases. In other words, NP-Complete problems usually take super polynomial or exponential time or space in the worst cases.

It is very unlikely that all the NP-Complete problems can be solved in polynomial time but at the same time no one has claimed that NP-Complete problems cannot be solved in polynomial time.

The following is true from the definition of NP-Completeness:

All NP problems can be solved in polynomial time, if any NP-Complete problem is solved in polynomial time.

An important feature of NP-Complete problem definition is that any problem in NP can be **reduced** or mapped to it in polynomial time. To reduce one problem X to another problem Y, we need to do mapping such that a problem instance of X can be transformed to a problem instance of Y, solve Y and then do the mapping of the result back to the original problem. Let us try to understand the process of reduction through one example. You want to add two decimal numbers using our calculator. First the two numbers are entered into a calculator. Then the numbers will be mapped (converted) to binary. The addition operation will be done in binary. Finally, the result will be mapped(reconverted) to decimal for display. All these steps will be done in polynomial time.

The main take away from the above discussion is that if we have two problems: X and Y, X is a NP-Complete problem and Y is known to be in NP. Further, X is polynomially reducible to Y so that we can solve X using Y. Since X is NP-Complete, every problem in NP can be reduced to it in polynomial time

1.4 CNF – Satisfiability Problem-A First NP-Complete Problem

Let us recall that for a new problem to be NP-Complete problem , it must be in NP-class and then a known NP- Complete problem must be polynomially reduced to it. In this unit we will not show any reduction example, because we are interested in the general idea. Some examples of reductions will be covered in the next unit. But one might be wondering how the first NP-Complete problem was proven to be NP-Complete and how the reduction was done. The satisfiability problem was the first NP-Complete problem ever found. The problem can be formulated as follows: Given a Boolean expression, find out whether the expression is satisfiable or not. Whether an assignment to the logical variables that gives a value 1(True)
A logical variable, also called a Boolean variable is a variable having only one of the two values: 1 (True) or False (0). A **literal** is a logical variable or negation of a logical variable. A **clause** combines literals through logical **or**(V) operator(s). ***A conjunctive normal form (CNF)*** combines several clauses through logical **and (∧)** operator(s)

The following is an example of logical expression in CNF having three clauses combined by **and (∧)** operators and logical variables $X_1, X_2, X_3, X_4,$ and complement of X_1, X_2, X_3 and $X_4 (\overline{X_1}, \overline{X_2}, \overline{X_3}$ and $\overline{X_4})$:

$$(X_2 \bigvee \overline{X_3}) \bigwedge (\overline{X_1} \bigvee \overline{X_2} \bigvee X_4) \wedge (X_1 \vee X_3 \vee X_4)$$

Circuit Satisfiability Decision Problem asks for a given logical expression in CNF, Whether some combination of True and False values to the logical variables makes the output of the expression as True.

For instance, if we assign X_1 = True ,X_2=False and X_3 = True, then The following logical expression in CNF satisfiability is Yes because the assignments of True and False values to the logical variables make the Boolean expression True.

$(X_1 \vee X_2 \vee X_3) \wedge (\text{X1} \vee \overline{X_2}) \wedge X_3$

But the assignments of X_1= True, X_2 – True and X_3 – True make the following Boolean expression False.

$(X_1 \vee X_2)\ \wedge \overline{X_2} \wedge X_3$
Therefore the answer to CNF Satisfiability is False.
Since it is not difficult to write a polynomial time algorithm to evaluate a Boolean expression and check whether the result is true, CNF Satisfiability problem is in NP. But to show that CNF satisfiability is NP-Complete, Cook applied the fact that every problem in NP is solvable in polynomial time by a nondeterministic Turing machine. Cook demonstrated that the actions of Turing machine can be simulated by a lengthy and complicated Boolean expression but still in polynomial time. The Boolean

expression would be true if and only if the program being run by a nondeterministic Turing machine produced a Yes answer for its input.
Several new problems such as Hamiltonian Circuit problem, TSP problem, Graph Coloring problem, etc. were proven to be NP-Complete after the satisfiability problem was shown to be NP-Complete.

Check Your Progress-2
Q1: What is P Vs NP problem?
Q2 What is Circuit Satisfiability Decision Problem?

1.5 Summary

In this unit the following topics were discussed:

- Three classes of problems in terms of its time complexities in the worst cases: P, NP, NP-Complete.
- Relationship between P, NP, NP-Complete
- Differences between tractable and intractable problems, optimization and decision problems and deterministic and nondeterministic algorithms
- CNF Satisfiability Problem

1.6 Solution to Check Your Progress

Check Your Progress – 1
Q1: List the problems belonging to polynomial class
Almost all problems which we examined so far, belong to polynomial class of problems. For examples, finding out a key in an array, Greatest Common Divisor of two integers, matrix multiplication, sorting algorithms, finding a shortest path in a weighted graph, finding a minimum cost spanning tree, problems implemented through dynamic programming such as, All Pairs Shortest Path Algorithm, Chained Matrix Multiplication and Optimal Binary Search Tree and Binomial Coefficient can be solved in polynomial time.

Q2: Formulate Graph Coloring Problem as an optimization as well as decision problems

Ans. Graph Coloring Optimization problem- Given a graph G= (V,E), the graph coloring optimization problem determines minimum numbers of colors needed to color a graph so that no two adjacent vertices of the graph are having the same color.

Graph Coloring Decision Problem- It determines whether a given graph can be colored in at most M colors such that no two adjacent vertices of the graph are having the same color

Q3 What is Halting Problem?

Ans. A famous decision problem which is not in NP problem is the **Halting** problem, which is defined as follows:

Given an arbitrary program with an arbitrary data, will the program terminate or not? This type of problem is called undecidable problems. Undecidable problems cannot be solved by guessing and checking. Although they are decision problem somehow they cannot be solved by exhaustively checking the solution space

Check Your Progress-2

Q1: What is P Vs NP problem?

If P=NP, we would have polynomial time solution for many known decision problems. When P≠NP. we would have to find a problem in NP that is not in P. But to show P=NP, we have to find a polynomial time algorithm for each problem is NP.

Q2 What is Circuit Satisfiability Decision Problem?

Ans. Circuit Satisfiability Decision Problem asks for a given logical expression in CNF, Whether some combination of True and False values to the logical variables makes the output of the expression as True

.