UNIT 3 HANDLING INTRACTABILITY

Structure

3.0 INTRODUCTION

It has been stated in the previous units that the large class of optimization problems belong to NP -hard class of problems.It is widely accepted that NP-hard problems are intractable problems, i.e., although not yet proven, it appears that such problems do not have polynomial time solutions.Their time complexities are exponential in the worst cases. Some examples of intractable problems are traveling salesman problem, vertex cover problems and graph coloring problems. Besides these combinatorial problems, there are several thousand computational problems in different domains like, biology, data science, finance and operation research fall into NP-hard category.

An exhaustive search can be applied to solve these combinatorial problems but it works when problem instances are quite small. An optimization technique such as dynamic programming may be applied to find solutions for combinatorial problems some such problems but it also has the similar limitation as in exhaustive search that the problem instance should be small. Another limitation of this technique is that the problems must follow the principle of optimality.

Problem solving techniques such as **backtracking and branch and bound techniques** perform better in comparison to exhaustive search. But unlike exhaustive search, these techniques construct the solutions step by step (considering only one element at a time) and performs evaluation of the partial solution. In case the results do not represent a better solution, further exploration of the remaining elements are not considered. Backtracking and branch and bound techniques apply some kinds of intelligence on the exhaustive search. It provides efficient solution if it has good design. But in the worst case it takes exponential time. One real advantage of branch and bound technique is that it can handle a large problem instance.

We can also apply Tabu search or Simulated Annealing which are heuristic local search methods or Genetic Algorithms and Particle Swarm Optimization which are meta heuristic techniques to find out solutions to the optimization problems.

But all these methods in spite of good performance do not ensure rigorous guarantees for achieving the quality of solution i.e., how far the proposed solution is far away from the optimal solution. Sometimes we should ask questions whether there is a possibility of finding near optimal solutions (approximate solution) to combinatorial optimization problems efficiently? Approximation algorithms have been found to be efficient algorithms with good quality solutions which are measured in terms of the maximum distance between the proposed approximate solution and the optimal

solution over all the problem instances. What does it mean? It means that the approximation algorithms always produce solution quite close to the optimal solution.

The focus of the unit will be to discuss few techniques such as backtracking and branch and bound techniques and approximation algorithms to handle intractable problems.

3.1 OBJECTIVES

The main objectives of the unit are to:

- Describe and apply backtracking and branch and bound techniques to combinatorial problems
- Define the concepts of state space tree and approximation ratio
- Formulate Hamiltonian Cycle problem, Subset Sum problem and Knapsack problem
- List approximation ratios of few combinatorial problems

3.2 BACKTRACKING AND BRANCH AND BOUND TECHNIQUES

Design of both techniques is based on state space tree, similar to a binary tree. Each node of the tree represents a partial solution. A node is terminated as soon as it is evident that no solution can be provided by the node's descendants.

How is backtracking different from branch and bound technique?

They differ in terms of types of problems they can solve and how nodes in the state space tree are generated?

Backtracking generally does not apply to optimization problems whereas branch and bound technique can be used to solve optimization problems because it computes a bound and proceeds further with a best bound. Depth first search is used to generate a tree in backtracking whereas there is no such restriction applicable to generate a state space tree using branch and bound.

3.2.1 Backtracking

Backtracking is a general technique for design of algorithms. It is applied to solve problems in which components are selected in sequence from the specified set so that it satisfies some criteria or objective. The backtracking procedure includes depth first search of a state space tree, verifying whether a node is leading to any solution(called promising node)or not but dead ends (called non promising node), does backtracking to the parent of the node if a node is not promising and continuing with the search process on the next child.

In this section we will use backtracking technique to solve two problems: (i) Hamiltonian Circuit problem and (ii) Subset Sum problem.

(i) Hamiltonian circuit problem

Suppose G = (V, E) is a connected graph with n vertices, Hamiltonian circuit problem determines a cycle that visits every vertex of a graph only once except the starting

vertex.$V_1$is a starting vertex of a cycle where$V_1 \in$ G and V_1, V_2……..$V_{n+1}$are distinct vertices in the cycle exceptV_1 and V_{n+1}vertices which are equal.

The following is a problem instance of a graph:

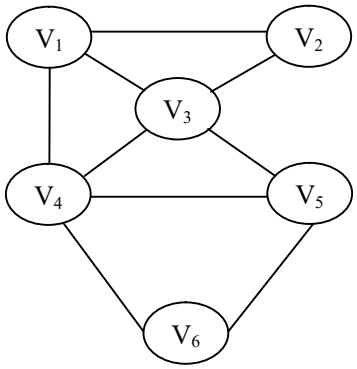

Figure 3.1: A graph for finding Hamiltonian cycle

The following figure represents the state space tree of the above graph(figure 3.1)

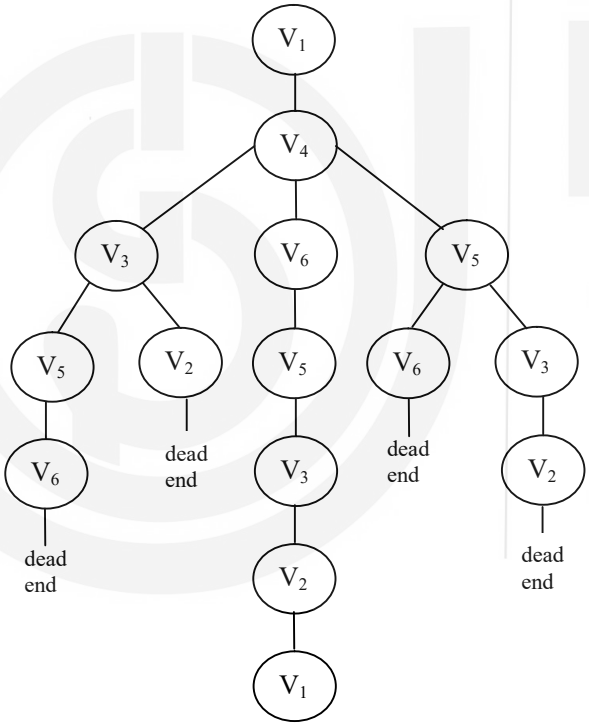

Figure 3.2 : State Space Tree representing Hamiltonian circuit of a graph

Design of a state space tree- Let us pick up V_1 as the starting vertex in a state space tree of a graph given at figure 3.2. Among the three options available to select a node from V_1, we pick upV_4. From V_4 it can go to V_3or V_6 or V_5. If we follow the order of V_6, V_5,V_3, V_2 and V_1, we get a correct Hamiltonian cycle. In other two cases there are dead ends. So the algorithm backtracks. For example in the left most branch of state space tree the algorithm backtracks from V_6 to V_5 to V_3 and then toV_2. This branch also does not provide any result. The algorithm backtracks again to $V_3$and then toV$_4$ and then follows the next path in the tree.

In order to explore another Hamiltonian cycle, the process can re-start from the leaf node of the tree through backtracking .

(ii) Subsetsum Problem

Given a positive integer W and a set S of n positive integer values i.e., S = { s_1, s_2,.. s_n}, the main objective of the Subset Sum problem is to search for all combinations of subsets of integers whose sum is equal to W. As an example let us take S = { 1,4,6,9} and W = 10, there are two solution subsets : {1,9} and {4,6}. In some cases, problem instances may not have any solution subset.

We will assume that elements in the set are in the sorted order, i.e., $s_1 \leq s_2 \leq \cdots \leq s_n$.

Design of state space tree for subset sum problem:

S = {4,6,7,8} and W = 18

The state space tree is designed as a binary tree. The root of the tree does not represent any decision about a node. It is simply a starting point of the tree. Its left and right sub trees include and exclude the element of the set represented by 1 and 0 respectively at every level.

At level 1 in the left branch of the tree, the second element $s_2$is included while in the right subtree it is excluded. A subset of a given set is represented by a path from the root node to the leaf node in the tree. A path from the root node of the tree to a node at the i_{th} level represent the inclusion of the first i numbers in the subset.

Let s' be the sum of numbers of all the nodes included at the i_{th} level. If s' is equal to W then the problem has a final solution. If we want to find out all the subsets, we need to do backtracking to the parent of the node or stop if no further subsets are required.

If the sum s' is not equal to W then the following two inequalities hold and the node can be declared as non-promising node.

(i) $s' + s_{i+1} > W$ (the large value of the sum)
(ii) $s' + \sum_{j=i+1}^{n} s_j < W$ (the small value of the sum)

The complete solution to the problem S = { 4, 6, 7, 8} and W =18 is given in figure 3.3 in form of the state space tree

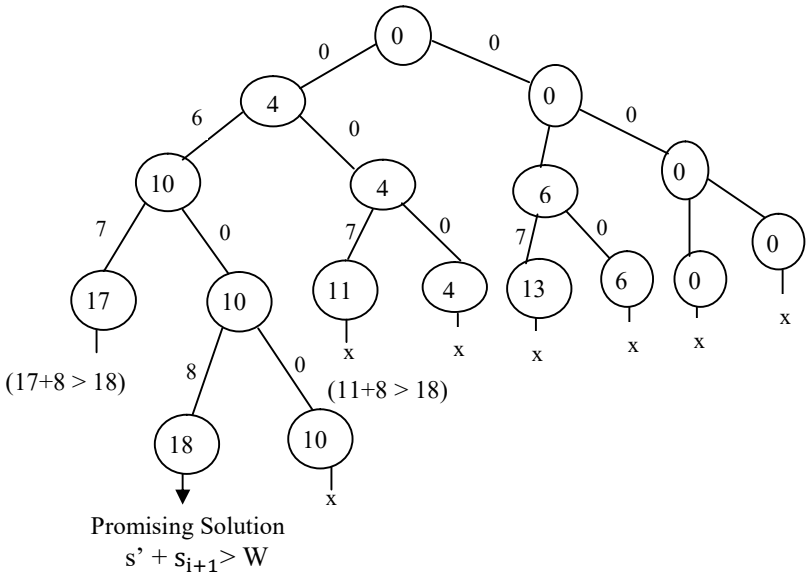

Figure 3.3 State space tree to the problem instance S = { 4, 6, 7, 8} and W =18 through backtracking algorithm

3.2.2 Branch and Bound

The similarity between backtracking and branch and bound techniques is design of state space tree. There are three distinct features of branch and bound technique: (i) unlike backtracking which traverse the tree in DFS, branch and bound does not restrict traversing in a particular way (ii) branch and bound technique computes a bound (value) of a node to decide whether the node is promising or not promising (iii) generally used for optimization problems

In branch and bound technique, search path in state space tree at a particular node is stopped if any of the following reasons occurs:

- Constraint violation
- The value of a bound of a node is inferior to the value of the best solution achieved so far

Let us now apply this technique to solve the Knapsack optimization problem. The knapsack problem is defined as: given a set of objects, its profit p_i, weight w_i , i= 1,2 ….n and W which is a knapsack capacity of Knapsack , obtain subsets of the objects which gives maximum profit and the total weight of the subsets do not exceed knapsack capacity W.

Before designing a state space tree of knapsack problem using branch and bound technique, sorting of objects in descending order by their profit to weight ratio(p_i/w_i) isdone i.e., $p_1/w_1 \geq p_2/w_2 \geq \ldots p_n/w_n$. This sequence will return the first object with the best profit and the last object with the least profit in terms per weight unit respectively. In case ties occur, it should be resolved arbitrarily.

State space tree- It is constructed in form of a binary tree. The root node of the tree is the initial point where no item has been selected (p = 0, w = 0). As in backtracking, the left branch of the tree includes the next object while the right branch excludes it. A node is represented by the three values:

w – total weight of items selected at a node

p – total profit

bound – total profits of any subset of items

One of the simplest way to calculate the bound is given below:

bound = $p + (W-w)(p_{i+1}/w_{i+1})$

The bound at any node is addition of profit p, the total profits of already selected items with the product of the left over capacity of knapsack (W-w) and the best profit per weight unit which is :

p_{i+1}/w_{i+1}.

Example: Given a knapsack problem instance, apply the branch and bound to find a subset which gives the maximum profit. The following is a problem instance:

Object	Profit(Rs.) (p)	Weight(w)	p/w
1	40	4	10
2	42	7	6
3	25	5	5
4	12	3	4

Knapsack capacity = 10

The following is the state space tree (figure 4)of representing the given instance of knapsack problem:

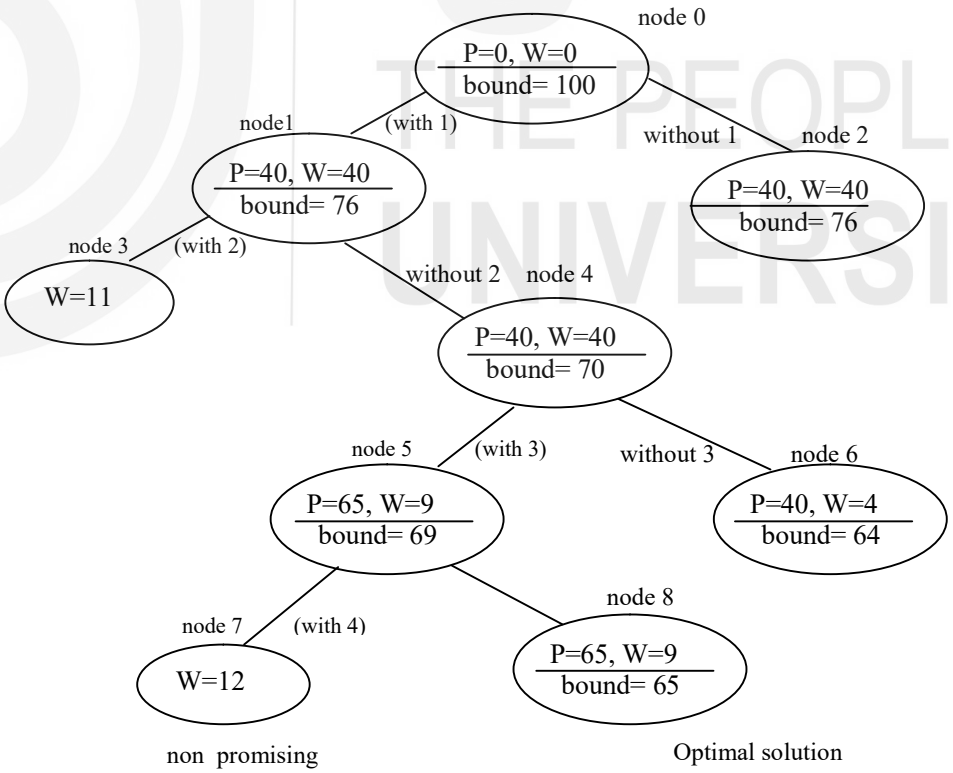

Figure 3.4: State space tree for knapsack problem using branch and bound technique

The root node indicates that no items have been selected as yet. Therefore p=0 and w= 0 and the bound is computed as per equation (i) which is Rs.100(0+ (10-0) (40/4)). The left branch of the tree includes the item1 and which is the only item in the subset while the right branch excludes the item 1. The total profit p and weight w is Rs 40 and 4 respectively. The bound value is 76 (40 + (10 -4)(42/7)). All three values are shown at node 1. Node 2 at the right branch excludes item 1.Therefore p = 0 and w = 0, because no item has been selected in the subset at node 2. Bound at this node is60(0 + (10-0) (42/7)). Compared to node 2, node 1 is more promising for optimization because it has a larger bound. Node 3 and node 4 are child nodes of node 1 have a subset with item 1 and item 2or without item 2 respectively. Let us calculate the total profit(p), total weight(w) and bound at node 3 first. Since w(total weight) of a subset represented by a node 3 exceeds 10 which is a capacity of the knapsack , this node is considered non promising. Since item 2 is not included at node 4 values of p and w are the same as its parent node 2 i.e., p= 40 , w= 4 . The bound= Rs.70(40 + (10-4)(25/5)). Compared to node 2, node 4 is selected for expanding the state space tree because its bound is larger than node 2. Now we move to node 5 and 6 which represent subsets including and excluding item 3 respectively. The total profit(p) , total weight (w) and bound at node 5 are:

p(item1 + item3) = Rs. 40 +Rs 25 = Rs.65

w(item 1 +item3) = 4 + 5 = 9

bound= 65+ (10 -9)(12/3)= 69

We will repeat the computation of p, w and bound at node 6:

P(item 1) = 40

w(item 1) = 4

bound = 40 + (10 – 4)(12/3) = 64

We will continue with node 5 because of its larger bound. Node 6 and node 7 represent a subset with and without item 4 respectively.

Node 7 is a non promising node because its total weight is 12(9 +3) exceeding the knapsack capacity.

p and w at node 8(without item 4) are 65 and 9 respectively, same as its parent

bound = 65 + (10-9)(0) =65

There is a single subset {1,3) represented by node 8. The other two nodes 2 and 6 have lower bounds than node 8. Hence node 8 is the final and optimal solution to the knapsack problem.

Check Your Progress -1

Q1. Define intractable problems in computer science.

Q2 What are the three distinct features of branch and bound technique?

Q3 How is backtracking different from branch and bound technique?

3.3 APPROXIMATION ALGORITHMS BASICS

As discussed in the previous unit, the decision version of hard combinatorial problems belongs to NP- complete complexity class whereas the optimization version of combinatorial problems is in **NP-hard** complexity class. Problems in this category are as difficult as NP-complete complexity class. There is no solution to these problems in polynomial time. Although, they are of practical significance.

The branch and bound technique which was discussed in the previous section is a major breakthrough because it is applicable to hard combinatorial problems with large problem instances. But the main issue is that such good performance cannot be guaranteed.

In this section we present a radically approach to deal with hard combinatorial problems called approximation algorithms which provide a solution to the hard combinatorial problem, reasonably close to optimal solution but in polynomial time. In real life situations, we usually work with inaccurate data. In such cases, getting near optimal solution is accepted and good enough. We require to find a bound that provides a measure how close a proposed solution is to optimal solution. For example, consider a TSP(Traveling Salesperson Optimization Problem) problem which is NP-Hard .

Further suppose that the nodes correspond to points in an Euclidean space, e.g., a 3D room, and the distance between any two points is their Euclidean distance.
There is an algorithm whose solution has the following guarantee: approxvalue is less than twice optimal value (i.e.<2xoptvalue) where, optvalue is the optimal solution for the problem and approxvalue is the approximate solution that the algorithm outputs.

3.3.1 Approximation Ratio

To precisely understand the notion of approximation algorithm, we define the term approximation ratio which can be defined as a ratio between the results obtained by the approximation algorithm and the optimal results of the algorithm.
Consider a minimization optimization problem P in which the main task is to minimize the given objective function. For example, vertex cover problem, TSP problem, graph coloring problem, etc. are combinatorial minimization optimization problem. Assume C is the value returned by a proposed polynomial time approximation algorithm for any such optimization problem on some problem instances whose size is n. Let C^* be the optimal solution for the given problem. Then the approximation ratio$\rho(n)$ is defined as :
$\rho(n)=\frac{C}{C^*}$
For a minimization problem, $C^*$is less and equal to ($\leq$)C and the approximation ratio $\frac{C}{C^*}$ defines the factor by which the cost of the polynomial approximate solution is larger than the cost of the optimal solution. Similarly for maximization problem, C is less and equal ($\leq$)C^* and the approximation ratio $\frac{C^*}{C}$ defines the factor by which the cost of an optimal solution$C^*$is larger than the proposed solution C.

It should be noted that the ratio, as defined above, is always at least 1. A ratio equal to 1 indicates the value returned by the algorithm is always equal to the optimal value, in other words, the algorithm solves the problem exactly. Since we want approximation algorithms that run in polynomial time, under the belief that P is different from NP, it will be not be possible to design an approximation algorithm in polynomial time for any NP-complete problem with 1 approximation ratio. The objective, therefore, becomes to design algorithms with a ratio that is very close to 1.

Check Your Progress-2

Q1 Write two examples of combinatorial optimization problems where approximation ratio can be used for finding near optimal solution.

Q2Define the approximation ratio of a minimization optimization problem. Explain the implication of an algorithm with a ratio that is very close to 1.

3.4 SUMMARY

Backtracking and branch and bound techniques apply some kinds of intelligence on the exhaustive search. It takes exponential time in the worst case for the solution of difficult combinatorial problems but it provides efficient solution if it has good design. But unlike exhaustive search, these techniques construct the solutions step by step, (one element at a time) and perform evaluation of the partial solution. If no solution is achieved by a given result, the tree is not further expanded.

State space tree is a principal mechanism employed by both backtracking and branch and bound techniques which is a binary tree where nodes represent partial solutions. Expansion of the tree is stopped as soon it is ensured that no solutions can be achieved by considering choices that correspond to the node's descendants.

Approximation algorithms are often applied to find near optimal solution to NP-hard optimization problems. Approximation ratio is the main metric to measure the accuracy of the solution to combinatorial optimization problems.

3.4 SOLUTION TO CHECK YOUR PROGRESS

Check Your Progress -1

Q1. Define intractable problems in computer science.

Ans. A problem in computer science is intractable if it cannot be solved it in polynomial time. These problems are hard problems which usually take exponential time in the worst case.

Q2 What are the three distinct features of branch and bound technique?

Ans. There are three distinct features of branch and bound technique: (i) unlike backtracking which traverse the tree in DFS, branch and bound does not restrict traversing in a particular way (ii) branch and bound technique computes a bound (value) of a node to decide whether the node is promising or not promising (iii) generally used for optimization problems

Q3 How is backtracking different from branch and bound technique?

Ans. They differ in terms of types of problems they can solve and how nodes in the state space tree are generated?

(i) Backtracking generally does not apply to optimization problems whereas branch and bound technique can be used to solve optimization problems because it computes a bound on the possible values of the objective function.

(ii) Depth first search is used to generate a tree in backtracking whereas there is no such restriction applied to branch and bound to generate a state space tree.

Check Your Progress-2

Q1 Write two examples of combinatorial optimization problems where approximation ratio can be used for finding near optimal solution.

Ans. 1 (i) Traveling Salesperson Problem- Given a weighted directed graph, the traveling salesperson optimization problem determines a path that starts and ends at the same vertex and visits all the vertices of the graph only once with minimal total weight.

(iii) Vertex Cover optimization problem- Given an undirected graph, the vertex cover optimization problem selects the minimum subset of vertices such that every edge in the input graph contains at least one selected vertex

Suppose an input graph G = (V,E)

The vertex cover optimization problem finds a subset S □ V such that for every edge (x, y) ∈ E ,either x ∈ S or y ∈ S. S should have minimum number of vertices.

Q2 Define the approximation ratio of a minimization optimization problem. Explain the implication of an approximation algorithm with a ratio that is very close to 1.

Ans. a minimization problem ,$0 < C^* \leq$ C and the approximation ratio = $\frac{C}{C^*}$defines the factor by which the cost of the polynomial approximate solution is larger than the cost of the optimal solution. When the approximation ratio is close to 1 indicates the value returned by the algorithm is always equal to the optimal value, in other words, the algorithm solves the problem exactly.