TRAVELLING SALESMAN PROBELM

PROBLEM:

Given a set of vertices(cities) and distance between every pair of vertices(cities), the problem is to find the shortest possible tour that visits every vertex(city) exactly once and returns to the starting point.

SOLUTION:

We have implemented the travelling salesman problem using two important methods:

1. Branch and Bound Algorithm
2. Backtracking

1.BRANCH AND BOUND ALGORITHM

We have used Branch and Bound Algorithm to implement travelling salesman problem as search problem. To generate the tree two data structures are used: a stack and a circular queue. For testing, the connection matrix is generated randomly. We have used stack in place of recursion to track each event in the program. In Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node. In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Traveling salesman problem:

$$\text{Cost of a tour } T = \sum (\text{sum of cost of two minimum weight edges adjacent to u and in the tour T}), \text{ Where u belong to V.}$$

Algorithm:

1. To generate the connection matrix randomly.
2. Start with an initial solution from city 1. Enqueue all the live nodes into the queue.
3. Now, start exploring a branch of the factorial tree. It has four possibilities:
   a) Save the better solutions; the nodes which satisfies the cost function better.
   b) Leaving nodes when there are no more branches.
   c) If bottom of stack is reached then Stops, means no more nodes to be visited.
   d) Explore an alternate branch.
4. Finally, after exploring all the branches, we will get an optimal cycle or tour and optimal values.

We have implemented the search problem using Branch and Bound Algorithm. We have also used stacks instead of recursions to know the values of each variables at each step and help in determining the optimal solution easily.

2.BACKTRACKING

In this case, we first create all Hamiltonian Cycles using Deep Backtracking Concepts. Then among these Hamiltonian cycles, the travelling salesman problem corresponds to minimum cost cycles.

Cost of Cycle = ∑ weight of edges between vertices in the cycle.

Algorithm:

1. First, we have created the Graph data structure using Adjacency List to store the neighbourhood of each vertices.
2. Generated the random weight or connections matrix.
3. Then we extract all the Hamiltonian cycles using the function AllHamiltionPath() in our program. This function works as:
    a) First, we sent the starting vertex as argument in function.
    b) Now, next piece of code process only on unvisited vertices. We have also maintained Path data structure to store the Hamiltonian path. Path visits each vertex exactly once.
    c) Check if adding vertex to the path leads to solution or not, if it is contributing then consider it in Hamiltonian path.
    d) Otherwise Backtrack until you recover.
4. We have also implemented the code for getting optimal path inside the function itself. We check for the path, if it is the path then we calculate the cost of cycles and compare it with global cycle till now. And if it satisfied then we store the optimal path in data structure to print it after the end of program.

APPLICATIONS

1. Drilling of printed circuit boards
2. Overhauling gas turbine engines
3. X-Ray crystallography
4. The order-picking problem in warehouses
5. Vehicle routing
6. Mask plotting in PCB production
7. School bus routing problem