TRAVELING SALESMAN PROBLEM

A COURSE MINI-PROJECT REPORT

By

SARANYA GHOSH (RA2111030010097)
DHRUV CHOPRA (RA2111030010104)
ARNAV SRIVASTAVA (RA2111030010066)

Under the guidance of
Lavanya V

In partial fulfilment for the Course

of

18CSC204J - DESIGN AND ANALYSIS OF ALGORITHMS

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND

TECHNOLOGY

Kattankulathur, Chenpalpattu District

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND

TECHNOLOGY

(Under Section 3 of UGC Act, 1956)
BONAFIDE CERTIFICATE

Certified that this mini project report
'Traveling Salesman Problem'
is the bonafide work of

SARANYA GHOSH (RA2111030010097)
DHRUV CHOPRA (RA2111030010104)
ARNAV SRIVASTAVA (RA2111030010066)

Who carried out the project work under my supervision.

SIGNATURE

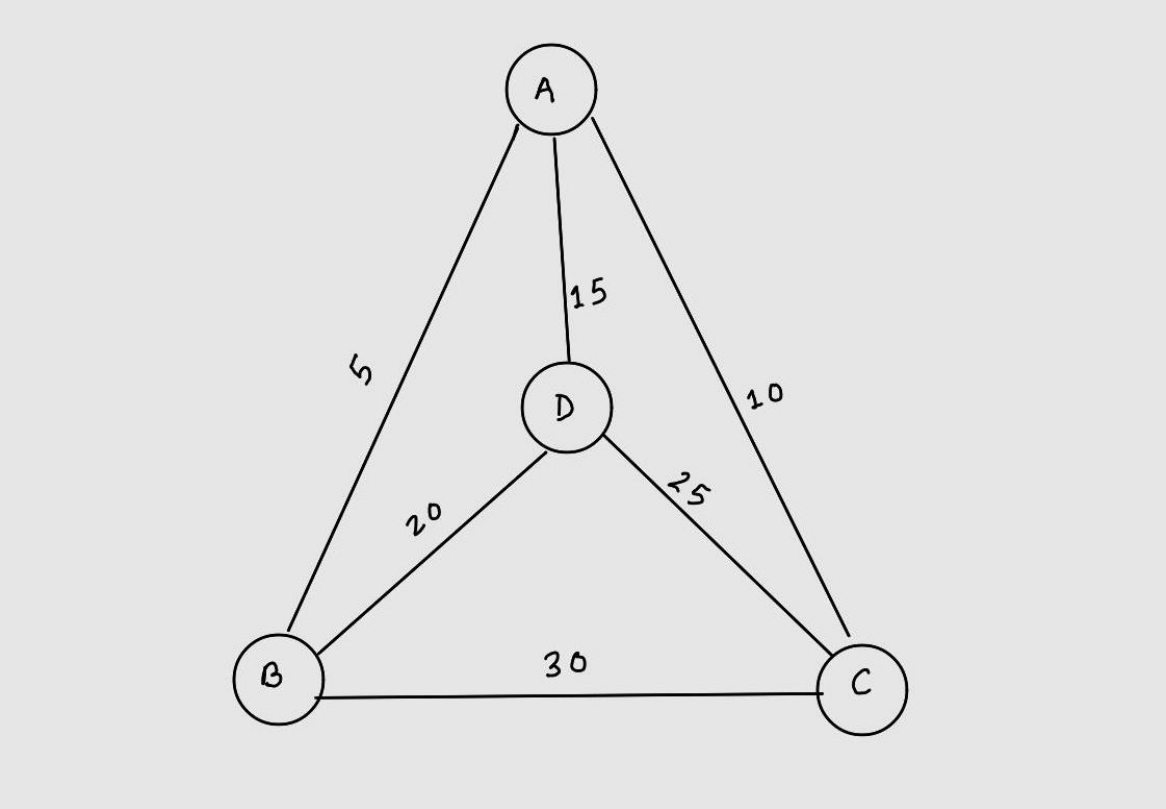
Lavanya V
Assistant Professor
NWC
SRM Institute of Science and Technology

TABLE OF CONTENT

Problem Statement:
Given a set of multiple delivery locations and the distance between every pair of delivery points, the problem is to find the shortest possible route that visits every point exactly once and returns to the starting point. The challenge of the problem is that the delivery person needs to minimize the total length of the trip.

Objective: To solve the traveling salesman problem to find the shortest delivery route using multiple design techniques and find the most optimal technique..

Given a set of delivery locations and distance between every pair of locations, the objective is to find the shortest possible tour that visits every delivery location exactly once and returns to the starting point.



This challenge can be resolved using the concept of 'Traveling Salesman Problem'. The Traveling Salesman Problem (TSP) is the challenge of finding the shortest, most efficient route for a person to take, given a list of specific destinations. It is a well-known algorithmic problem in the fields of computer science and operations research, with important real-world applications for logistics and delivery businesses.

Design Technique 1: Brute force

The Brute Force approach, also known as the Naive Approach, calculates and compares all possible permutations of routes or paths to determine the shortest unique solution. To solve the TSP using the Brute-Force approach, you must calculate the total number of routes and then draw and list all the possible routes. Calculate the distance of each route and then choose the shortest one—this is the optimal solution.

Aim:

To implement the traveling salesman problem to find the shortest delivery route using brute force algorithm

Algorithm

The traveling salesman problem is a permutation problem. There can be n! total ways to solve permutation problems.

In this approach, we'll generate all the possible permutations (routes). And find the one with the shortest distance.

The algorithm is as follows:

Step 1: Choose a city to act as a starting point(let's say city1).
Step 2: Find all possible routes using permutations. For n cities, it will be (n-1)!.
Step 3: Calculate the cost of each route and return the one with the minimum cost.

Pseudo Code:

```
Randomly select , let O<r<m
Randomly select s, let O<s<iXcryuean!
For i-=1 0 m-1 do
    Insert the node N(visited(t), s) after the ith
        node, compute the length L
Find the shortest Lshortest
1f Lshortest < Loriginal)
     Update the tour with the shortest one
End
```

Source Code

```
#include <bits/stdc++.h>
using namespace std;
#define V 4

int TSP_Implement(int Adj_matrix[][V], int s)
{
	vector<int> cities;
	for (int i = 0; i < V; i++)
	if (i != s)
	cities.push_back(i);

	int min_distance = INT_MAX;
	do
	{

		int curr_distance = 0;


		int k = s;
		for (int i = 0; i < cities.size(); i++)
		{
		curr_distance += Adj_matrix[k][cities[i]];
		k = cities[i];
		}
		curr_distance += Adj_matrix[k][s];


		min_distance = min(min_distance, curr_distance);

	}

	while(next_permutation(cities.begin(), cities.end()));

	return min_distance;
}

int main()
{
```

```
        int Adj_matrix[][V] =
        { { 0, 5, 10, 15 },
        { 5, 0, 30, 20 },
        { 10, 30, 0, 25 },
        { 15, 20, 25, 0 } };
        int start = 0;
        cout <<"The minimum cost is: "<<TSP_Implement(Adj_matrix, start) << endl;
        return 0;
}
```

Output

Time Complexity

The time complexity of O(nn), where n is the number of cities as we've to check (n-1)! routes (i.e all permutations).

Result

We successfully implemented the traveling salesman problem to find the shortest delivery route using brute force algorithm.

This is not an optimal solution, so we'll explore some other approaches as well. Branch and bound is one crucial algorithm.

Design Technique 2: Branch And Bound

This method breaks a problem to be solved into several sub-problems. It's a system for solving a series of sub-problems, each of which may have several possible solutions and where the solution selected for one problem may have an effect on the possible solutions of subsequent sub-problems.

Aim

To implement the traveling salesman problem to find the shortest delivery route using branch and bound algorithm

Algorithm

1. Initially, graph is represented by cost matrix C, where

   Cij = cost of edge, if there is a direct path from city i to city j

   Cij = ∞, if there is no direct path from city i to city j.

2. Convert cost matrix to reduced matrix by subtracting minimum values from appropriate rows and columns, such that each row and column contains at least one zero entry.

3. Find cost of reduced matrix. Cost is given by summation of subtracted amount from the cost matrix to convert it in to reduce matrix.

4. Prepare state space tree for the reduce matrix

5. Find least cost valued node A (i.e. E-node), by computing reduced cost node matrix with every remaining node.

6. If <i, j> edge is to be included, then do following :

(a) Set all values in row i and all values in column j of A to ∞

(b) Set A[j, 1] = ∞

(c) Reduce A again, except rows and columns having all ∞ entries.

7. Compute the cost of newly created reduced matrix as,

Cost = L + Cost(i, j) + r

Where, L is cost of original reduced cost matrix and r is A[i, j].

8. If all nodes are not visited then go to step 4.

Pseudo Code

Cost of a tour T = (1/2) * ∑ (Sum of cost of two edges
adjacent to u and in the
tour T)
where u ∈ V
For every vertex u, if we consider two edges through it in T,
and sum their costs. The overall sum for all vertices would
be twice of cost of tour T (We have considered every edge
twice.)

(Sum of two tour edges adjacent to u) >= (sum of minimum weight
two edges adjacent to
u)

Cost of any tour >= 1/2) * ∑ (Sum of cost of two minimum
weight edges adjacent to u)
where u ∈ V

Lower Bound for vertex 1 =
Old lower bound - ((minimum edge cost of 0 +
minimum edge cost of 1) / 2)
+ (edge cost 0-1)
Lower bound(2) =
Old lower bound - ((second minimum edge cost of 1 +
minimum edge cost of 2)/2)
+ edge cost 1-2)

Source Code

```
#include <bits/stdc++.h>
using namespace std;
const int N = 4;

int final_path[N+1];
```

```
bool visited[N];
int final_res = INT_MAX;

void copyToFinal(int curr_path[])
{
    for (int i=0; i<N; i++)
        final_path[i] = curr_path[i];
    final_path[N] = curr_path[0];
}

int firstMin(int adj[N][N], int i)
{
    int min = INT_MAX;
    for (int k=0; k<N; k++)
        if (adj[i][k]<min && i != k)
            min = adj[i][k];
    return min;
}

int secondMin(int adj[N][N], int i)
{
    int first = INT_MAX, second = INT_MAX;
    for (int j=0; j<N; j++)
    {
        if (i == j)
            continue;

        if (adj[i][j] <= first)
        {
            second = first;
            first = adj[i][j];
        }
        else if (adj[i][j] <= second &&
                 adj[i][j] != first)
            second = adj[i][j];
    }
    return second;
}

void TSPRec(int adj[N][N], int curr_bound, int curr_weight,
```

```
        int level, int curr_path[])
{

    if (level==N)
    {

        if (adj[curr_path[level-1]][curr_path[0]] != 0)
        {

            int curr_res = curr_weight +
                    adj[curr_path[level-1]][curr_path[0]];

            if (curr_res < final_res)
            {
                copyToFinal(curr_path);
                final_res = curr_res;
            }
        }
        return;
    }
    for (int i=0; i<N; i++)
    {
        if (adj[curr_path[level-1]][i] != 0 &&
            visited[i] == false)
        {
            int temp = curr_bound;
            curr_weight += adj[curr_path[level-1]][i];

            if (level==1)
              curr_bound -= ((firstMin(adj, curr_path[level-1]) +
                        firstMin(adj, i))/2);
            else
              curr_bound -= ((secondMin(adj, curr_path[level-1]) +
                        firstMin(adj, i))/2);

            if (curr_bound + curr_weight < final_res)
            {
                curr_path[level] = i;
                visited[i] = true;
```

```
            TSPRec(adj, curr_bound, curr_weight, level+1,
                  curr_path);
        }

        curr_weight -= adj[curr_path[level-1]][i];
        curr_bound = temp;

        memset(visited, false, sizeof(visited));
        for (int j=0; j<=level-1; j++)
            visited[curr_path[j]] = true;
      }
   }
}

void TSP(int adj[N][N])
{
   int curr_path[N+1];

   int curr_bound = 0;
   memset(curr_path, -1, sizeof(curr_path));
   memset(visited, 0, sizeof(curr_path));

   for (int i=0; i<N; i++)
      curr_bound += (firstMin(adj, i) +
                  secondMin(adj, i));

   curr_bound = (curr_bound&1)? curr_bound/2 + 1 :
                           curr_bound/2;

   visited[0] = true;
   curr_path[0] = 0;

   TSPRec(adj, curr_bound, 0, 1, curr_path);
}

int main()
{
   int adj[N][N] =  { { 0, 5, 10, 15 },
        { 5, 0, 30, 20 },
        { 10, 30, 0, 25 },
```

```
        { 15, 20, 25, 0 } };

    TSP(adj);

    printf("Minimum cost : %d\n", final_res);
    printf("Path Taken : ");
    for (int i=0; i<=N; i++)
        printf("%d ", final_path[i]);

    return 0;
}
```

Output

Time Complexity

The worst case complexity of Branch and Bound remains the same as that of the Brute Force clearly because in the worst case, we may never get a chance to prune a node.

Result

Successfully implemented the traveling salesman problem to find the shortest delivery route using branch and bound algorithm.

Conclusion:

We have solved the ‘Traveling Salesman Problem to find the optimal delivery route’ problem using two design techniques— Brute Force & Branch and Bound. We observed that the branch and bound approach is more efficient than the brute-force approach. The solution to the TSP problem with the Brute Force algorithm always reaches the optimal solution, but it takes time and a long stage and will get into trouble if the number of cities increases. The Branch and Bound algorithm can solve the TSP problem more efficiently because it does not calculate all possibilities.