*Discrete Mathematics Project Report on*

# Implementation of Algorithms for Traveling Salesman Problem

**Alina Melony Pinto – 191IT206**

**Gayathri Nisha – 191IT116**

**K Sakshi Thimmaiah – 191IT124**

**Nishka Kotian – 191IT136**

**Department of Information Technology, NITK Surathkal**

***Date of Submission: 13 November 2020***

**in partial fulfillment for the award of the degree of**

**Bachelor of Technology**

**In**

**Information Technology**


**At**

**Department of Information Technology National Institute**

**Technology Karnataka, Surathkal**

**November 2020**

# IMPLEMENTATION OF ALGORITHMS FOR TRAVELING SALESMAN PROBLEM

---

## ABSTRACT

The Traveling Salesman problem is one of the very important problems in Computer Science and Operations Research. It is used to find the minimum cost of doing work while covering the entire area or scope of the work in concern. In this paper, we will review the past work done in solving the travelling salesman problem using three different techniques- Naïve solution, Dynamic programming, Genetic algorithm, and Branch & Bound.

---

## INTRODUCTION

The Traveling Salesman Problem (TSP) is related to the real-life analogy of a salesman who has to visit a number of cities while taking the shortest possible path. The four different techniques that will be used in this paper i.e. Naïve solution, Dynamic programming, Genetic algorithm, and Branch & Bound have different approaches and different parameters for solving the travelling salesman problem.

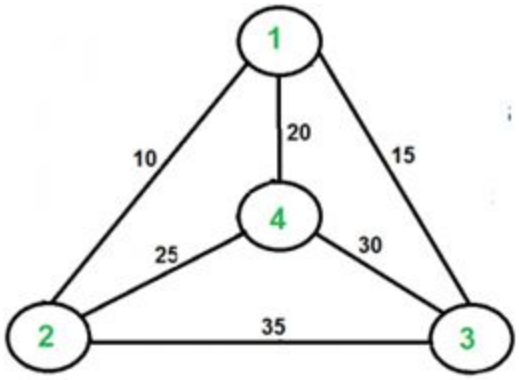
- The Brute Force approach, also known as the Naive Approach, calculates and compares all possible permutations of routes or paths to determine the shortest unique solution.
- Genetic algorithms (GA) are based on the survival of the fittest chromosome among the species which are generated by random changes in their gene-structure in evolutionary biology.

- In the Branch and Bound method, for the current node in the tree, we compute a bound on the best possible solution that we can get if we down this node.
- In Dynamic Programming (DP) we build the solution as we go along. In our case, this means that our initial state will be any first node to visit, and then we expand each state by adding every possible node to make a path of size 2, and so on.

---

## WORK DONE

The salesman's goal is to keep both the travel costs and the distance travelled as low as possible. Focused on optimization, TSP is often used in computer science to find the most efficient route for data to travel between various nodes. Consider the graph shown below:




It has 4 cities or nodes to be visited: 1,2,3,4. If the starting point is 1, it needs to visit all the cities and come back to 1 eventually. The TSP tour here would be 1-2-4-3-1 with the cost of the tour being 10+25+30+15 = 80.

In this project, we have tried to solve this problem by implementing 4 different algorithms namely:

1) *Naïve solution*

2) *Dynamic programming*

3) *Genetic algorithm*

4) *Branch & Bound*

Here is the detailed discussion for each one of the algorithms:

## *1) Naïve solution:*

The naïve or brute force solution aims at computing and verifying all the possible permutations and thereby choosing the minimum/shortest possible solution.

These are the steps to be followed to implement this approach:

Step 1: Choose any one of the cities as the starting and ending point.

Step 2: Generate all possible permutations i.e., (n-1)! Permutations

Step 3: Calculate the cost of each of these possible permutations and choose the one with the minimum cost.

Step 4: Return the permutation with the minimum cost (or here the shortest path).

This approach has a time complexity of O(N!) which is not very efficient for large datasets.

## *2) Dynamic programming:*

In the Dynamic Programming approach, we do not compute all possible permutations of the path, instead, the solution is built as we go along the path i.e., the problem is divided into the smallest possible subproblems. We figure out the solutions for them and then integrate these solutions to form the solution for the larger, more complicated problem/subproblem. In this way, we arrive at the final solution to the TSP.

Step 1: The problem is divided into subproblems.

Step 2: For each subproblem, the cost of the path is calculated recursively.

Step 3: The minimum cost path is chosen simultaneously.

Step 4 : $g(i,S) = \min\{C_{ik} + g(k,S-\{k\}\}$

where i is the starting point, S is a set of points, k is a point belonging to S, g(i,S) is the minimum cost of path and $C_{ik}$ is the cost of traveling from i to k.

There are at most $2^n n$ subproblems and each takes linear time. This approach has a time complexity of $O(2^n n^2)$.

## *3) <u>Genetic algorithm:</u>*

The genetic algorithm is a technique based on methods adapted from the field of genetics in biology. The algorithm is designed to replicate the real-world phenomenon of evolution. Here, the possible model behaviours are encoded into genes (here, cities are taken as genes). The natural process of selection or survival of the fittest is being replicated here, where the shortest possible paths thrive when compared with the other routes. The proposed algorithm is motivated by the observation that genes common to all the individuals of a GA have a high probability of surviving evolution and ending up being part of the final solution; as such, they can be saved

away to eliminate the redundant computations at the later generations of a GA. We have aimed to eliminate these redundant computations at the early generations of a GA so that the computation time can be significantly reduced while at the same time retaining or enhancing the quality of the end result. Researches on genetic algorithms focus not only on improving the quality of the end result but also on reducing the computation time of GA.

Here are the steps included in this process/algorithm:

Step 1: Creating an initial population.

Step 2: Use the fitness function to select the fitter chromosomes.

Step 3: Select the best genes and then apply the crossover and mutation operators in order.

Step 4: Find new mutations until a solution is arrived at.

Step 5: Stop and output the best chromosome

## *4) Branch & Bound:*

The Branch & Bound method is based on the idea that the sets are divided into 2 disjoint subsets at every step of the process – branching. Furthermore, one subset contains the path between the two selected towns and the other subset does not. For each of these subsets, the lower restriction (bound) is calculated for the duration or for travel expenses. Finally, the subset which exceeds the estimated lower bound is eliminated.

Steps:

1.We start enumerating all possible nodes (in lexicographical order, starting at vertex 0 for which lower bound is calculated)

2.Each time we branch, we try to infer additional decisions regarding which edges must be included or excluded from tours represented by those nodes.

3.When we branch, we compute lower bounds for both children. If the lower bound for a child is greater than the lowest cost found so far, we prune that child and need not consider its descendants.

4.When there are no more nodes to explore, the current best solution would be the optimal solution.

---

## RESULTS AND ANALYSIS

| | Branch And Bound | Naive approach | Dynamic Programming | Genetic Algorithm |
|---|---|---|---|---|
| Time (s) | 0.00784 | 0.31445 | 0.04545 | 0.19831 |
| Distance | 1513 | 1513 | 1513 | - |
| Path | 1,2,6,5,4,8, 9,7,10,3,1 | 1,2,6,5,4,8, 9,7,10,3,1 | 1,2,6,5,4,8,9,7, 10,3,1 | - |

Table 1. Comparison of algorithm considering 10 nodes.

**Naive approach**: The Naive, or the brute-force approach compares all the possible permutations of paths to discover the best shortest path. Given n cities, it results in (n−1)!/2 possible unique cycles. The starting city has n-1 possibilities for the second city. The second city will have n-2 possibilities and so on. As the paths are bidirectional, there will be

duplicates. Removing them, we get the total number of possible paths as (n-1)!/2. Hence the time complexity is O(n!) due to which it is not suitable for real-world applications. This approach always gives the exact solution but comes at the cost of huge runtimes.

**Branch and bound**:

Using the branch and bound method,

Cost of any tour= $\frac{1}{2}\sum_{v\in V}(\text{Sum of the costs of the two tour edges adjacent to } v)$

Now,

The sum of the two tour edges adjacent to a given vertex v >= sum of the two edges of least cost that are adjacent to v

Therefore:

Cost of any tour $\geq \frac{1}{2}\sum_{v\in V}(\text{Sum of the costs of the two least cost edges adjacent to } v)$

This technique is also an *exact* algorithm. The only issue is that it is not guaranteed to have low running time. It is quick on some instances, and slow on others. The worst-case complexity of Branch and Bound is the same as that of the Naive solution because it is possible that we may never get a chance to prune a node.

**Dynamic Programming:** Dynamic programming divides the problem into $2^n n$ subproblems each of which has linear time complexity

making the algorithm time complexity $O(2^n n^2)$. This approach gives the shortest optimal path.

**Genetic Algorithm**: Genetic Algorithm does not always give the exact shortest path, but it provides an optimal solution that is close to the actual shortest path. It works well even for a larger number of cities compared to all the algorithms mentioned above.
We can observe that with just 100 generations, we could find a route about half the distance of the original and probably very close to the optimal solution.

The output for 7 and 10 city problems along the cost matrix is shown below

[[0,3,6,7,5,3,5],

[6,0,2,9,1,2,7],

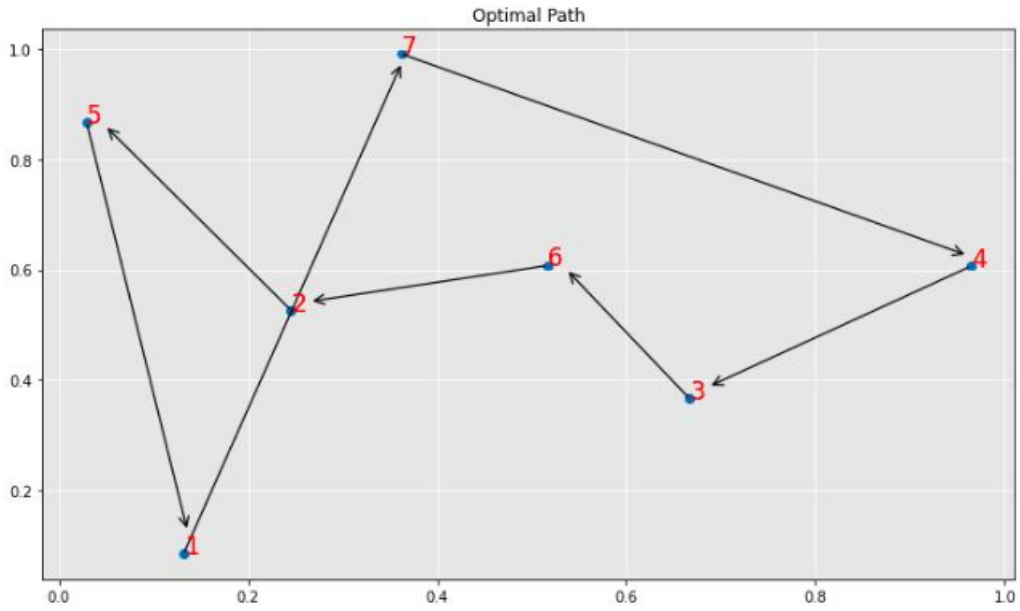
[0,9,0,3,6,0,6],

[2,6,1,0,8,7,9],

[2,0,2,3,0,7,5],

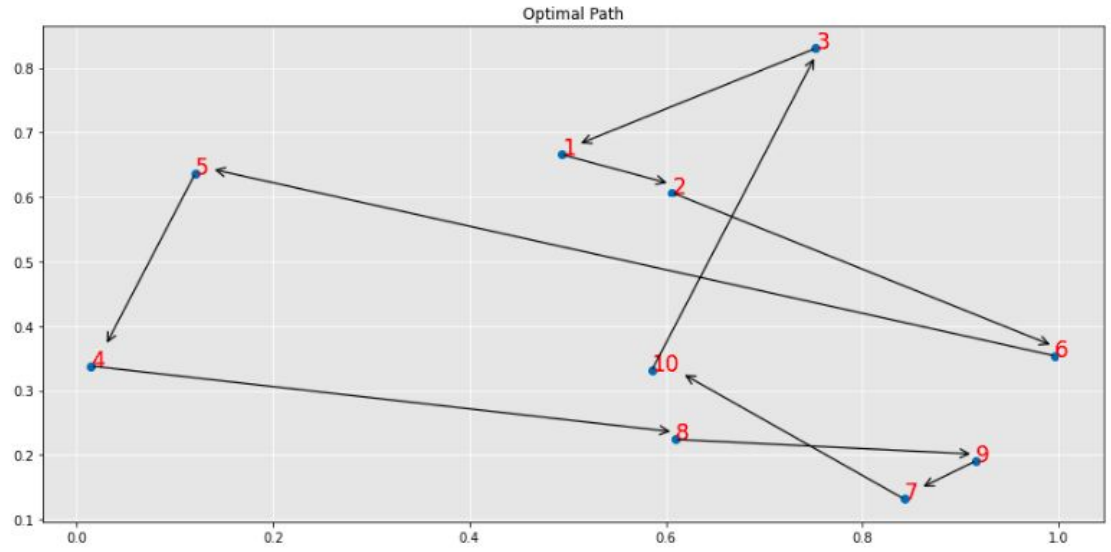
[9,2,2,8,9,0,7],

[3,6,1,2,9,3,0]]




```
Total Nodes: 7
Distance : 3.794
Cost:  13
Path Taken :  1 7 4 3 6 2 5 1
```

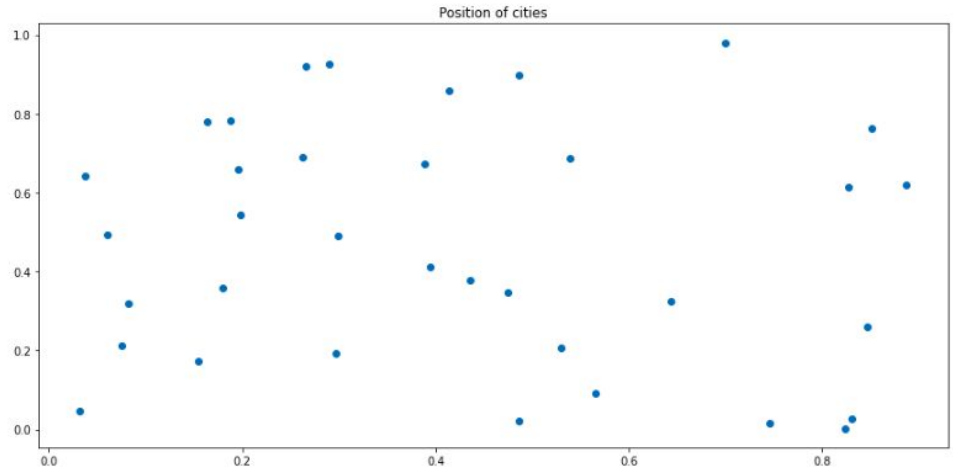
[[0,383,886,777,915,793,335,386,492,649],
[421,0,362,27,690,59,763,926,540,426],
[172,736,0,211,368,567,429,782,530,862],
[123,67,135,0,929,802,22,58,69,167],
[393,456,11,42,0,229,373,421,919,784],
[537,198,324,315,370,0,413,526,91,980],
[956,873,862,170,996,281,0,305,925,84],
[327,336,505,846,729,313,857,0,124,895],
[582,545,814,367,434,364,43,750,0,87],
[808,276,178,788,584,403,651,754,399,0]]

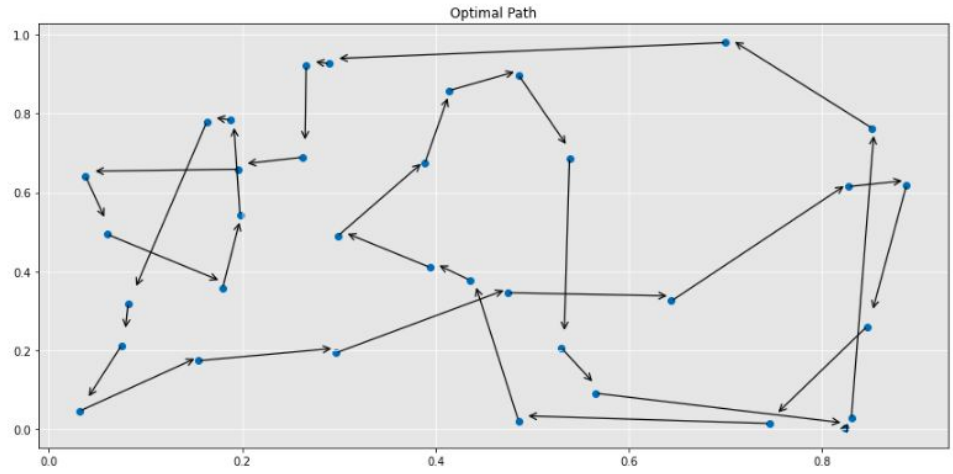



```
Total Nodes: 10
Distance : 3.994
Cost:  1513
Path Taken :  1 2 6 5 4 8 9 7 10 3 1
```

Result obtained for TSP with 35 cities using Genetic Algorithm

```
Initial distance: 15.386392710267332
Finished
Final distance: 7.567568246735564
Solution:
```




```
Time taken for  Genetic algorithm for 50 iterations: 1.6927677049999978
```

## CONCLUSION

This report compares various algorithms used to solve the Traveling Salesman problem. We can conclude that for a large number of nodes the Genetic algorithm is the fastest while the Naive approach is the slowest among the four algorithms. Even though dynamic programming and branch and bound algorithm may appear faster when a smaller number of points are taken, it is slower for a larger number of points.

Dynamic programming approach works well for number of nodes ranging upto 30.

Considering the optimal path, dynamic programming approach, branch and bound and naive approach give the shortest path while genetic approach give less favourable paths.

---