Travelling Salesman Problem Heuristic and Exact Algorithms

Dhruv Singh Negi (2021CSB1165)
Virat Jain (2021CSB1220)

Instructor :
Dr. Sudarshan Iyengar

Teaching Assistant :
Poonam Adhikari
Malya Singh
Rajtilak Pal
Nikhil Kumar Tiwary
Piyush Aggarwal
Nitesh Wadhavinde

1. Introduction

- The Travelling Salesman Problem (TSP) is a notoriously difficult problem in computer science.
- The goal of the TSP is to find the shortest route that starts in one city, visits every other city exactly once, and returns back to the starting city.
- The travelling salesman problem has been used as a test case for many optimization algorithms and continues to be a subject of research and study in the field of mathematics and computer science
- The TSP is often referred to as the symmetric TSP, where the distance from city A to city B is the same as the distance from city B to city A.
- The TSP assumes that a direct path between two cities is always the shortest path, and it is called the triangle inequality assumption.

2. TSP as a graph problem

TSP can be represented as an undirected weighted graph in which cities serve as the vertices, paths serve as the edges, and the weight of an edge is determined by the distance of a path. It is a minimization problem that begins and ends at a given vertex after precisely visiting each vertex in between. The model is frequently a complete graph, with an edge connecting each pair of vertices. If there isn't a route connecting two cities, the graph will be finished without changing the ideal route by adding a sufficiently long edge.

3. History and Origin of the Travelling Salesman Problem

Although the idea of the problem was first introduced in a manual in 1832 it described the problem analytically and not mathematically .The term Travelling Salesman was first introduced among the mathematicians by Karl Menger and Hassler Whitney in the 1930s. Mathematician K. Menger defined and wrote about the problem and gave the idea of the brute force solution of the problem and showed how the nearest neighbour approach was not optimal. In 1934 Whitney mentioned the problem of finding the shortest route along the 48 States of America which is similar to the TSP. In 1937 Flood showed how the Hamiltonian paths ,graphs are connected to the TSP and later many mathematicians proposed various methods and heuristics to solve the problem. Later in 1970 Karp showed that the Hamiltonian cycle problem was NP-complete implying that the TSP is also NP-complete.

4. Applications of TSP in Various Fields

1. **Delivery System :** In delivery systems of manufacturing companies, there are stages where goods are transported from the factory to warehouses, wholesalers, and finally to retail shopkeepers and customers.The TSP can be applied to minimise transportation costs in these stages by finding the optimal route.
2. **DNA Sequencing :** A collection of DNA strings, each of length *k*, were embedded in one universal string , with the goal of minimising the length of the universal string. The cities of the TSP are the target strings, and the cost of travel is *k* minus the maximum overlap of the corresponding strings.
3. **Manufacturing of microchips:** Drill Machine is used to drill holes in a PCB of different diameter and at different distances, as the changing of diameter of drill machine is very time consuming and costly , TSP can be used to optimally drill the holes of same diameter first then change the diameter of drill.
4. **Logistics:** TSP can be used to minimise the total cost of order picking in a warehouse by determining an optimal path of picking orders.

5. Why is it one of the most challenging problems in Computer Science ?

The Travelling Salesman Problem (TSP) is one of the most challenging and discussed problems in computer science because it has no efficient solution and it can't be solved using any traditional , existing algorithms in polynomial time. Thus it falls in a group of problems called NP - hard problems and TSP is NP - complete problem as all other NP - hard problems are reducible to it in polynomial time.

To prove a problem is NP-Hard we reduce a well known NP-Hard problem to it. In the TSP case we have the Hamiltonian Cycle problem which is a well known NP-Hard problem. Hamiltonian Cycle problem can be reduced to Travelling Salesman problem as we know that TSP is finding / computing the Hamilton cycle in a directed graph with lowest cost as Hamilton Path covers all the Nodes in The Graph. Considering this we can say that the TSP is NP-Hard.

6. How To Solve It ?

1. Exact vs Heuristic Algorithms

There are two main categories of algorithms available to solve the TSP one is Exact and other is Heuristic Algorithms.Exact and Heuristic Algorithms are a trade off between accuracy and time complexity with Exact being superior in accuracy and Heuristic being superior in time complexity.
Exact algorithms can find the optimal / exact solution but they only work for a small number of cities/nodes as the time complexity is exponential and increases exponentially with the number of cities. Ex- brute-force,branch and bound method etc.
Heuristic algorithms can handle cases with a large number of cities/nodes but it contains some percentage of error and can't guarantee us the optimal / exact solution.Ex- nearest neighbour approach , 2-opt etc.
With advancement in the field of computer science now we have algorithms with a very small margin of error and comparatively better time complexity as compared to the old heuristic and exact algorithms.

2. Exact Algorithms :

1. The brute-force approach

The Most easy and the most intuitive way is the Brute Force Approach also known as the Naive Approach. To solve the TSP using the Brute-Force approach , We take a Starting Point and generate all the permutations of the routes and paths starting from this point and returning to it. We then calculate the distance of each path and then choose the shortest one which is our desired route with minimum cost.

Time complexity = O((n-1)!/2)

Pseudo Code:

Algorithm 1 Brute_Force (Cities):

```
1:  start = first(Cities)
2:  min_path = 0;
3:  cur_path = 0;
4:  for (path :  all possible paths):
5:      cur_path = cost (path);
6:      min_path = min(min_path,cur_path);
7:  return min_path;
```

2. The branch and bound method

Branch and Bound Method is a method created to solve general problems with exponential time constraints. For Branch and Bound in TSP we select the starting node and add it to the tree till the distance is less than bound. Once the distance reaches above the bound we go to the next level and keep doing this until all nodes/arcs are covered. It involves breaking the problem into smaller and smaller problems and then solving it as solving for smaller subproblems is easier and more efficient.

Steps to solve the algorithm :

1) Take a Starting Point let's say 1.
2) Calculate the bound of the current node and now we add see all nodes from 2 to n and if the answer we get along that arc is more than the bound we then we ignore the subtree rooted with the node because obviously moving further will give more answer than the bound.
3) After adding all the nodes/arcs at a level we then move onto the next level and add nodes.
4) We keep doing this until all nodes/arcs are covered and then the min is our answer.

Worst Time complexity = $O(n^2 * 2^n)$

Pseudo Code:

Algorithm 2 Branch_And_Bound (Cities):

```
1:  cur_node = first(Cities)
2:  bound  = int_max
3:  min_path = 0;
4:  while(not all nodes are covered)
5:      cur_bound = bound(cur_node)
6.      For i = 2 to n do
7:          If(answer < cur_bound) Add();
8:          else ignore (subtree);
9:          min_path = min(answer,min_path);
10:  return min_path;
```

Performs Better than Brute Force on Average and More Efficient Overall.

3. Dynamic Programming and the Held-Karp Algorithm

Another way to Solve the Problem is By dynamic programming, also known as the Bellman-Held-Karp algorithm, proposed in 1962 independently by Bellman and by Held and Karp .It solves TSP by systematically evaluating the question for smaller sets of cities and then expanding it to further more cities using the results calculated earlier (from 2 cities to 3 cities and so on). This Prevents Doing the same calculation again and again. It guarantees an optimal solution by considering all possible routes and using the already precomputed values in the dp table. This can be done by both top-down or bottom up approach. In the top down approach we use a recursive formulation combined with memoization to avoid redundant computations and in the bottom up we calculate from smaller to larger number of cities. Here are the essential steps in the Algorithm:

Steps to solve the algorithm :

1) Set up a dynamic programming table to store the optimal distances between subproblems.
2) Then we calculate the cost for smaller tasks (i.e for set size 2,3 and so on)
3) While calculating the result for set size k we use the already calculated results in our dp table for set size less than k by the recurrence formula dp[Set][I] = min {dp[Set - J][J] + distance(J, I)};
4) After filling the entire dp table with the function we check them all using a loop with starting point 1 and ending point i (i goes from 2 to n) and calculate min {dp[({2, 3, ..., n}][i] + distance(1, i)};.The minimum answer is our shortest path.

Time complexity = $O(n^2 * 2^n)$

Pseudo Code:

Algorithm 3 Held-Karp (Graph , n):

```
1:  for i = 2 to n do
2:      dp[Set(i)][i] = distance(1,i);
3:  for i = 2 to n - 1 do
4:      for all Set ⊆ set(2,3,......n) do
5:          for all i ∈ Set do
6:              dp[Set][i] := $\min_{j\neq i, j \in S}$ {dp[Set - i][j] + distance(j,i)};
7:  min_path = $\min_{i\neq 1}$ [dp[set(2, 3, ..., n)][i] + distance(i, 1)]
8:  return min_path;
```

3. Heuristic Algorithms:

1. Nearest Neighbour Algorithm:

One of the earliest algorithms used to approximately solve the travelling salesman issue was the nearest neighbour algorithm. In this method, the person begins in a random location and keeps going to the closest cities until they are all visited.
The approach behind this method is to basically repeatedly try to add the closest edge in a weighted graph to find the optimal tour, rejecting an edge when it either creates a cycle i.e the node is already visited or it creates a vertex with a degree of 3.

Worst Case Time Complexity: $O(n^2)$
Variation from the exact solution in Average Case: +25%.

Pseudo Code:

Algorithm 4 Nearest_Neighbour (Cities):

```
1:  start = first(Cities)
```

```
2:  tour = [start]
3:  unvisited = set(Cities - {start})
4:  while unvisited:
5:      Next = min(unvisited, key=lambda c: distance(c, tour[-1]))
6:      tour.append(Next)
7:      unvisited.remove(Next)
8:  return tour
```

2. Christofides algorithm

It is an heuristic algorithm that guarantees that its solutions of the tsp will be within a factor of 3/2 of the optimal solution length. It is generally used to generate the lower bound answer of the tsp.
The Christofides algorithm is one of the most impressive solutions to the TSP.
On random instances, it tends to be only about 10 percent away from the one tree lower bound. Using the MST and the edges in the minimum weight perfect matching, we can progress towards a solution to the TSP. Nicos Christofides developed an algorithm that uses the MST to generate a good solution to the TSP.

Steps to solve the algorithm:

1) Create a minimum spanning tree (M) of graph (G).
2) Create a set (O) having nodes with odd degree in (M)
3) Find the perfect matching (P) with minimum travel cost from (O)
4) Add the edges in set (P) to the minimum spanning tree to form a new graph (H) with all vertices having even degree.
5) Form the euler cycle of (H) and remove the repeating edges from the euler's path.

Worst Case Time Complexity: O(n^3)
Variation from exact solution in Worst Case: +50%.

Pseudo code:

Algorithm 5 Christofides (Points):

```
1: Graph = complete_graph(points)
2: MST = minimum_spanning_tree(Graph)
3: Odd_Degree_Nodes = []
4: for node in MST:
5:     if (degree(node) % 2 != 0):
6:         Odd_Degree_Nodes.append(node)
7: Perfect_Matching = perfect_matching(Odd_Degree_Nodes)
8: Perfect_matched_edges = []
9: for Pairs in Perfect_Matching:
10:     Perfect_matched_edges += (Pairs[0], Pairs[1])
```

```
MST.add_edges(Perfect_matched_edges)
MinCost = []
for i in eulerian_circuit(MST):
    if i[0] not in MinCost:
        MinCost.append(i[0])
return MinCost
```

3. Local Search and Improving the TSP Solution:

Two opt and k-opt improvements:

The improvement method called "two opt" connects pairs of edges in an alternative configuration that keeps the tour valid.This improvement can be tried for all pairs of edges until no more improvement is possible.
The subsequent graph is then called "opt optimal".The same principle can be extended to three edges, called "three opt" improvement, and falls under the category of k-opt improvements.The larger the k-value, the more likely an improvement can be found, but it might take longer due to the larger number of candidate solutions to try.Random swaps, two opt switches, and three opt switches are common methods for tour improvement in the Travelling Salesman Problem (TSP).

Worst case Time Complexity: O(n^k)

Pseudo Code:

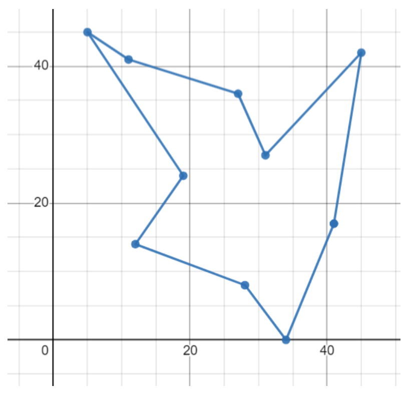
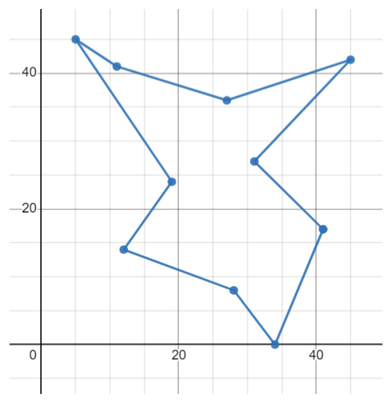
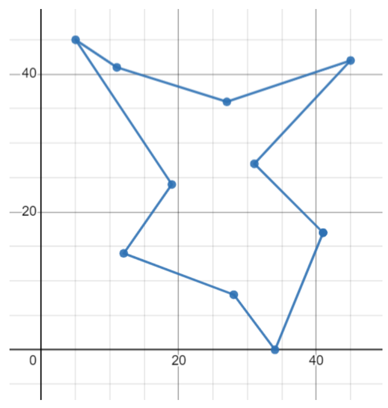
Algorithm 6 Two_Opt (Tour):

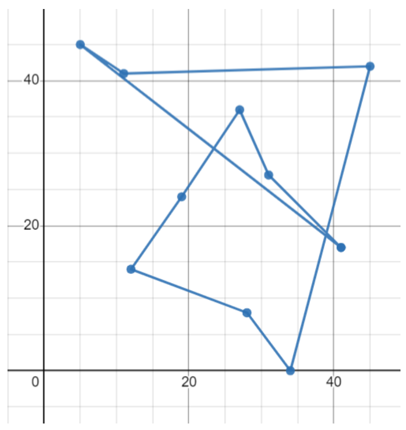
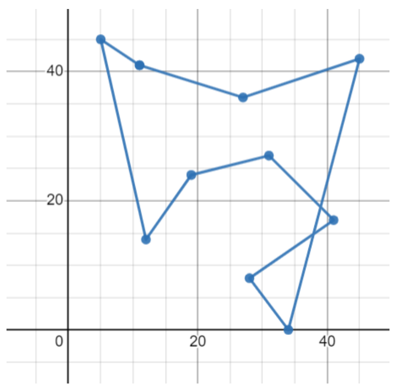
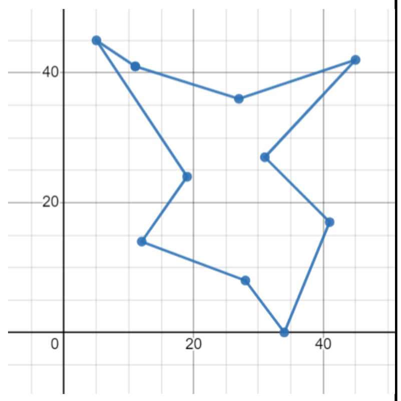
```
min_change = 0
Total_cities = len(Tour)
for i in range(Total_cities - 2):
    for j in range(i + 2, Total_cities - 1):
        change = dist(i, j) + dist(i+1, j+1) - dist(i, i+1) - dist(j, j+1)
        if change < min_change:
            min_change = change
            min_i, min_j = i, j
if min_change < 0 :
    Tour [min_i+1:min_j+1] = Tour [min_i+1:min_j+1][::-1]
return Tour
```

7. Analysing All The Above Approaches for Sample Data Space

Data Set 1

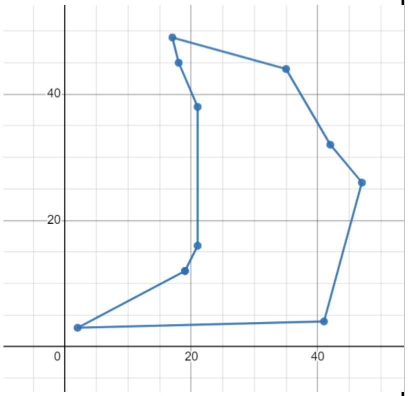
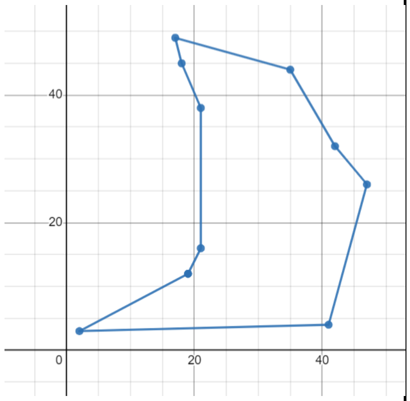
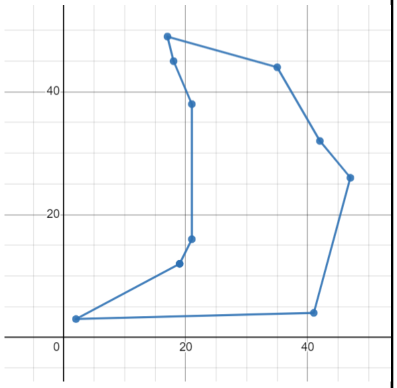
Node	0	1	2	3	4	5	6	7	8	9
X cor	41	34	19	28	12	5	31	11	45	27
Y cor	17	0	24	8	14	45	27	41	42	36

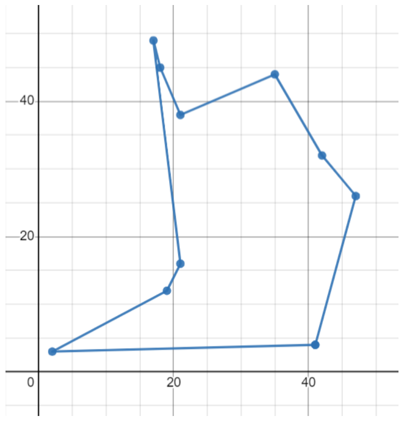
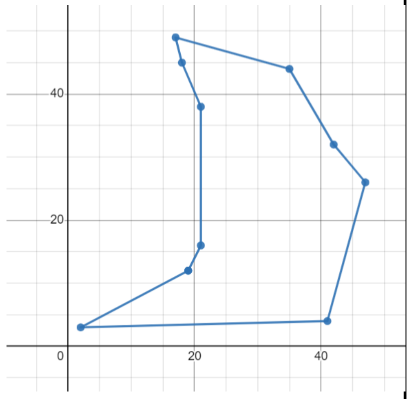
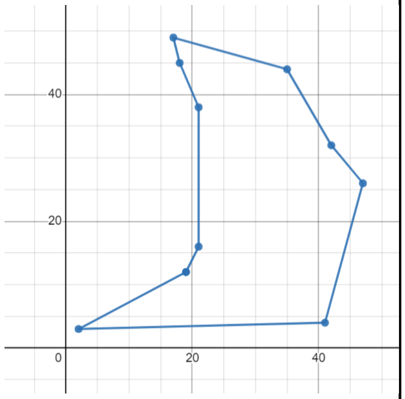
Branch and Bound	Brute Force	DP
Distance - 159	Distance - 157	Distance - 157
Path - 0 1 3 4 2 5 7 9 6 8 0	Path - 0 1 3 4 2 5 7 9 8 6 0	Path - 0 1 3 4 2 5 7 9 8 6 0
Accuracy - 98.727 %	Accuracy - 100 %	Accuracy - 100 %
		

Nearest neighbour	Christofides	2-opt
Distance - 205	Distance - 187	Distance - 157
Path - 0 6 9 2 4 3 1 8 7 5 0	Path - 7 5 8 9 6 2 4 1 3 0 7	Path - 0 1 3 4 2 5 7 9 8 6 0
Accuracy - 69.420 %	Accuracy - 80.9 %	Accuracy - 100 %
		

Data Set 2

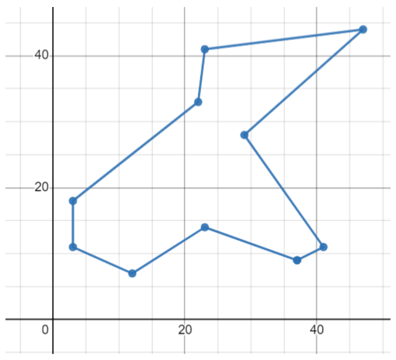
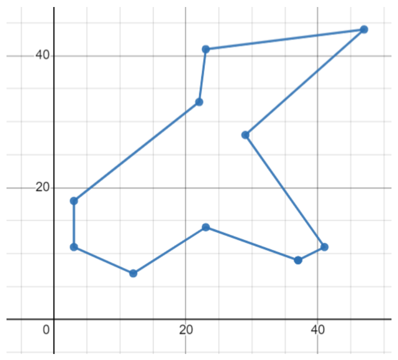
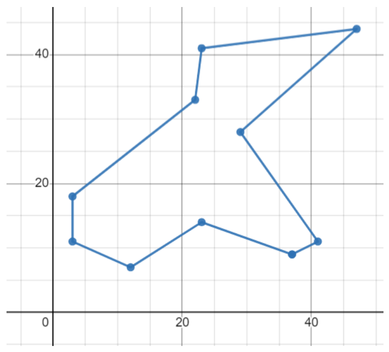
Node	0	1	2	3	4	5	6	7	8	9
X cor	41	2	42	21	18	47	21	19	17	35
Y cor	4	3	32	16	45	26	38	12	49	44

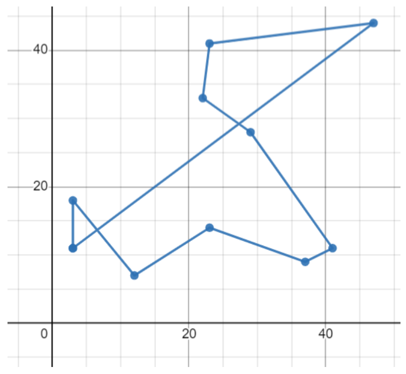
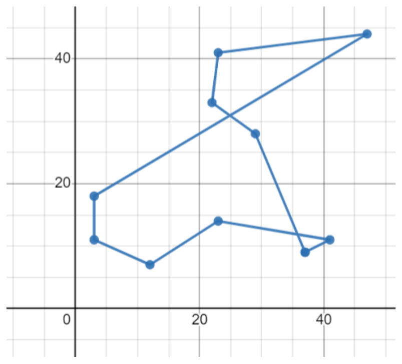
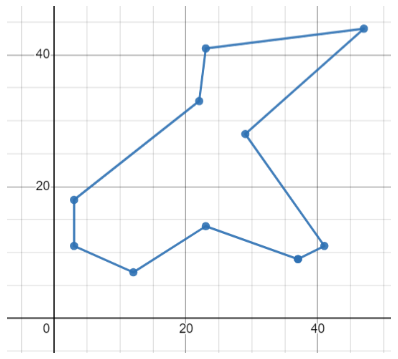
Branch and Bound	Brute Force	DP
Distance - 155	Distance - 155	Distance - 155
Path - 0 1 7 3 6 4 8 9 2 5 0	Path - 0 1 7 3 6 4 8 9 2 5 0	Path - 0 1 7 3 6 4 8 9 2 5 0
Accuracy - 100 %	Accuracy - 100 %	Accuracy - 100 %
		

Nearest neighbour	Christofides	2-opt
Distance - 165	Distance - 155	Distance - 155
Path - 0 5 2 9 6 4 8 3 7 1 0	Path - 0 1 7 3 6 4 8 9 2 5 0	Path - 0 1 7 3 6 4 8 9 2 5 0
Accuracy - 93.548 %	Accuracy - 100 %	Accuracy - 100 %
		

Data Set 3

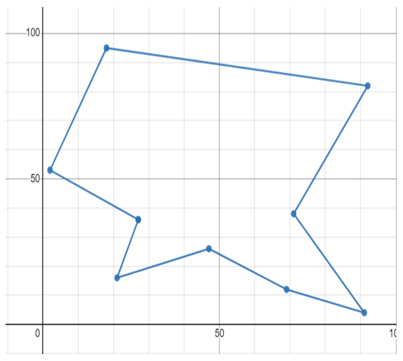
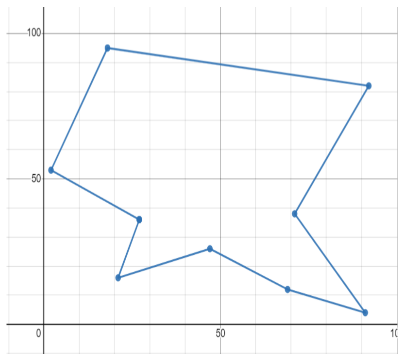
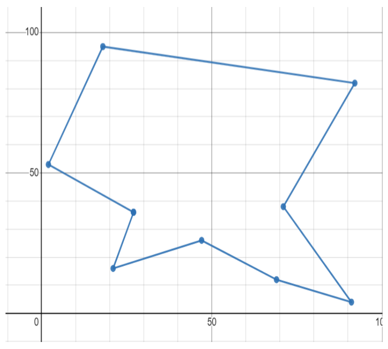
Node	0	1	2	3	4	5	6	7	8	9
X cor	3	22	23	41	3	47	12	37	23	29
Y cor	11	33	14	11	18	44	7	9	41	28

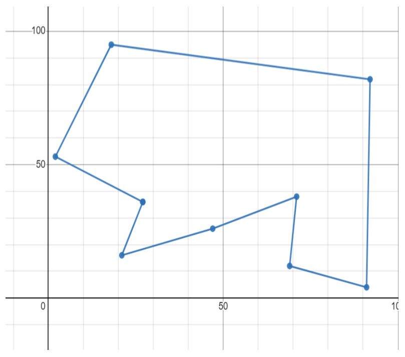
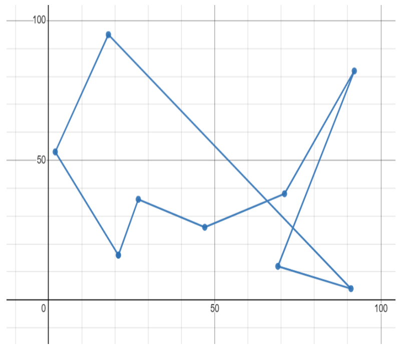
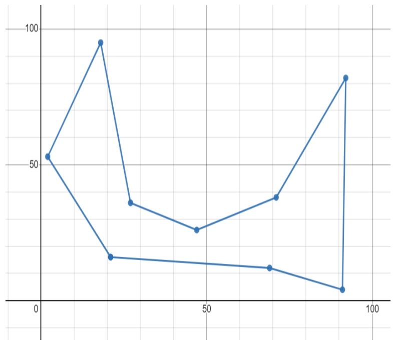
Branch and Bound	Brute Force	DP
Distance - 147	Distance - 147	Distance - 147
Path - 0 4 1 8 5 9 3 7 2 6 0	Path - 0 4 1 8 5 9 3 7 2 6 0	Path - 0 4 1 8 5 9 3 7 2 6 0
Accuracy - 100 %	Accuracy - 100 %	Accuracy - 100 %
		

Nearest neighbour	Christofides	2-opt
Distance - 167	Distance - 162	Distance - 162
Path - 0 5 2 9 6 4 8 3 7 1 0	Path -7 3 2 6 0 4 5 8 1 9 7	Path - 7 3 9 5 8 1 4 0 6 2 7
Accuracy - 86.394 %	Accuracy - 89.7959 %	Accuracy - 100 %
		

Data Set 4

Node	0	1	2	3	4	5	6	7	8
X cor	27	91	2	92	21	18	47	71	69
Y cor	36	4	53	82	16	95	26	38	12

Branch and Bound	Brute Force	DP
Distance - 332	Distance - 332	Distance - 332
Path - 0 2 5 3 7 1 8 6 4 0	Path - 0 2 5 3 7 1 8 6 4 0	Path - 0 2 5 3 7 1 8 6 4 0
Accuracy - 100 %	Accuracy - 100 %	Accuracy - 100 %
		

Nearest neighbour	Christofides	2-opt
Distance - 349	Distance - 399	Distance - 389
Path - 0 4 6 7 8 1 3 5 2 0	Path - 4 0 6 7 3 1 8 5 2 4	Path - 4 8 1 3 7 6 0 5 2 4
Accuracy - 94.879 %	Accuracy - 79.819 %	Accuracy - 82.831 %
		

8. Acknowledgements

We would like to express our sincere gratitude to Dr. Sudarshan Iyengar for giving us the opportunity to work on a subject that interests us. This opportunity allowed us to conduct extensive study and we were able to learn a great deal of fresh information. Additionally, we want to thank all the Teaching Assistant for helping us with any issues we encountered and assisting us at every stage of the project. Without their amazing direction and support, we would not have been able to finish our project.