

A
CIA REPORT
ON
Travelling Salesman Problem (TSP)

Submitted by,
03 Walke Akshata
04 Unde Ashwini
05 Attar Rehan
(TY BTECH COMPUTER)
[DIVISION A]

Guided by,
Prof. P. B. Dhanwate



Subject- Design and Analysis of Algorithms
In Academic Year 2024-25
Department of Computer Engineering
Sanjivani College of Engineering, Kopergaon

Sanjivani College of Engineering, Kopargaon

CERTIFICATE

This is to certify that

03 Walke Akshata

04 Unde Ashwini

05 Attar Rehan

(T.Y. Computer)

Successfully completed their CIA Report on

Travelling Salesman Problem (TSP)

Towards the partial fulfilment of

Bachelor's Degree in Computer Engineering

During the academic year 2022-23

Prof. P.B.Dhanwate
[Guide]

Dr. D. B. KSHIRSAGAR
[H.O.D. Comp Engg]

Dr. A. G. THAKUR
[Director]

CONTENTS

Sr. No.	Chapter	Page No.
1.	Introduction	04
2.	Problem Statement & description	05
3.	Problem Pre requisite	07
4.	Requirement analysis of problem	08
5.	Solved example of given application	09
6.	Control abstraction of the Algorithm: Flowchart	14
7.	Explanation of algorithmic steps or code of entire program	15
8.	Implementation	19
9.	Time and space complexity of problem statement	21
10.	Conclusion	23
11.	References	23

1. Introduction

The Traveling Salesman Problem (TSP) is a cornerstone problem in optimization, mathematics, and theoretical computer science. It seeks to determine the shortest possible route that allows a traveler to visit a given set of cities exactly once, returning to the starting city. Formally, it is modeled as a graph, where cities are nodes, and edges represent the paths between them, weighted by distances or costs.

Key Characteristics:

- Objective: Minimize the total distance or cost of the tour.
- Constraints: Each city must be visited exactly once, and the tour must return to the starting city.

Importance:

- TSP is a classic combinatorial optimization problem and is categorized as NP-hard, meaning no polynomial-time algorithm exists to solve it for large inputs. As the number of cities increases, the number of possible routes grows factorially, making exhaustive solutions computationally impractical.

Applications:

- TSP has wide-ranging applications in real-world scenarios, including:
- Logistics and Transportation: Optimizing delivery routes to minimize fuel consumption and time.
- Manufacturing: Sequencing tasks in operations such as drilling or circuit board assembly.
- Telecommunications: Designing network layouts for efficient data transmission.
- Biology: Sequencing genomes in bioinformatics research.

2. Problem Statement & Description

Problem Statement:

The Travelling Salesman Problem (TSP) is a classic puzzle in the world of mathematics and computer science. It's about a salesman who needs to visit a list of cities, and the challenge is to find the shortest possible route that allows the salesman to visit each city exactly once and return to the starting point. This problem might seem simple at first, but as the number of cities increases, finding the best route becomes extremely complicated. The problem can be applied to many real-world scenarios, such as logistics, route planning, manufacturing, and circuit design.

Given:

- A set of n cities.
- A distance matrix $D[i][j]$, where $D[i][j]$ represents the distance between city i and city j .

Find:

- The shortest path (tour) that visits each city exactly once and returns to the starting point.

Description:

The **Traveling Salesman Problem (TSP)** is a well-known puzzle in math and computer science. It asks a simple question: If a salesman has to visit a list of cities and knows the distance between each pair of cities, what is the shortest route that visits all the cities exactly once and returns to the starting point?

At first, this might seem easy, but it becomes much harder as the number of cities increases. For a small number of cities, you could try all possible routes to find the shortest one. But as the number of cities grows, the number of possible routes grows rapidly. For example, if you have just 10 cities, there are already nearly 400,000 possible routes! For 20 cities, this number jumps to about 60 trillion. This huge number of possible routes makes it very hard to find the best solution quickly.

Despite the challenge, the TSP is important because it has many real-world uses. Here are a few examples:

- Delivery and logistics: Companies need to plan the shortest delivery routes to save time and fuel.
- Circuit design: In electronics, TSP helps in designing efficient layouts for circuits.

- DNA sequencing: In biology, TSP is used in organizing genetic data to map genomes more efficiently.
- Travel planning: When planning trips, finding the best route to visit multiple places can be solved using ideas from TSP.

Because the problem is so difficult for larger numbers of cities, exact methods that find the best possible solution are often only useful for small cases. These methods can guarantee the best solution but take too long to solve for many cities.

For larger problems, we often use heuristic methods that don't always find the perfect solution but can give a good solution quickly. Some examples include:

- Nearest Neighbour Method: This simple method always goes to the closest city next. It's fast but doesn't always give the best solution.
- Genetic Algorithms: These use ideas from biology, like evolution, to improve solutions over time.
- Simulated Annealing: A technique that tries to avoid getting stuck in bad solutions by occasionally accepting worse routes in hopes of finding a better one later.

Overall, the TSP is both a challenging and important problem with practical applications in many areas. Even though it's hard to solve perfectly for large numbers of cities, we can still find good solutions using various methods.

3. Problem Prerequisite

To fully grasp and solve the Traveling Salesman Problem (TSP), it is helpful to have knowledge of the following topics:

1. Graph Theory:

- Basic concepts such as nodes (vertices), edges, weighted graphs, and complete graphs.
- Understanding how a graph can represent cities (nodes) and paths (edges with weights).

2. Combinatorics:

- Familiarity with permutations and combinations to understand the number of possible routes.
- Understanding factorial growth to comprehend the complexity of solving TSP exhaustively.

3. Algorithms:

- Knowledge of dynamic programming (e.g., Bellman-Held-Karp algorithm).
- Understanding of greedy algorithms and their limitations for finding optimal solutions.

4. Optimization:

- Basics of optimization problems, including objectives (minimization or maximization) and constraints.
- Familiarity with heuristics and metaheuristics, like Nearest Neighbor, Genetic Algorithms, and Simulated Annealing.

5. Complexity Theory:

- Concepts like NP-hardness and why finding an exact solution for TSP is computationally expensive.
- Difference between polynomial-time algorithms and exponential-time algorithms.

6. Mathematics:

- Ability to work with matrices (e.g., distance matrices).
- Understanding of geometric distances (Euclidean or Manhattan distance).

7. Programming Skills:

- Practical experience with coding and implementing algorithms.
- Familiarity with libraries and tools for graph manipulation (e.g., NetworkX in Python).

4. Requirement Analysis of Problem

1. Input Requirements

- Number of Cities (n): The total number of cities to be visited.
- Distance Matrix: A 2D matrix $D[i][j]$ representing the distance between city i and city j . The matrix can be symmetric or asymmetric.

2. Output Requirements

- Optimal Tour (Path): The sequence of cities that the salesman should visit, starting and ending at the same city.
- Total Distance (Cost): The total distance of the optimal tour.

3. Constraints

- Visit Each City Once: Every city must be visited exactly one time.
- Return to Starting City: The tour must return to the starting city after visiting all other cities.
- Non-Negative Distances: All distances between cities must be non-negative.

4. Optimization Goals

- Minimize Total Distance: The objective is to minimize the total distance of the tour while visiting all cities and returning to the starting point.

5. Performance Considerations

- Scalability: For small numbers of cities, exact methods (e.g., dynamic programming) can be used, but for larger instances, approximation algorithms or heuristics (e.g., genetic algorithms, ant colony optimization) may be required due to the NP-hard nature of the problem.

6. Domain-Specific Requirements (Optional)

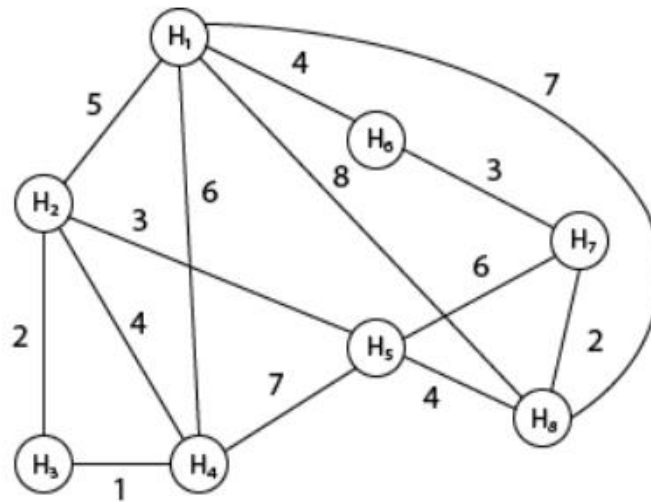
- Real-World Factors: Depending on the application, additional constraints like city accessibility, dynamic conditions (traffic, roadblocks), or time windows may need to be considered.

5.Solved example of given application

Example of TSP:

A newspaper agent daily drops the newspaper to the area assigned in such a manner that he has to cover all the houses in the respective area with minimum travel cost. Compute the minimum travel cost.

The area assigned to the agent where he has to drop the newspaper is shown in fig:



Solution: The cost- adjacency matrix of graph G is as follows:

cost_{ij} =

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

The tour starts from area H1 and then select the minimum cost area reachable from H1.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H6 because it is the minimum cost area reachable from H1 and then select minimum cost area reachable from H6.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H7 because it is the minimum cost area reachable from H6 and then select minimum cost area reachable from H7.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H8 because it is the minimum cost area reachable from H8.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H5 because it is the minimum cost area reachable from H5.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H2 because it is the minimum cost area reachable from H2.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

Mark area H3 because it is the minimum cost area reachable from H3.

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈
H ₁	0	5	0	6	0	4	0	7
H ₂	5	0	2	4	3	0	0	0
H ₃	0	2	0	1	0	0	0	0
H ₄	6	4	1	0	7	0	0	0
H ₅	0	3	0	7	0	0	6	4
H ₆	4	0	0	0	0	0	3	0
H ₇	0	0	0	0	6	3	0	2
H ₈	7	0	0	0	4	0	2	0

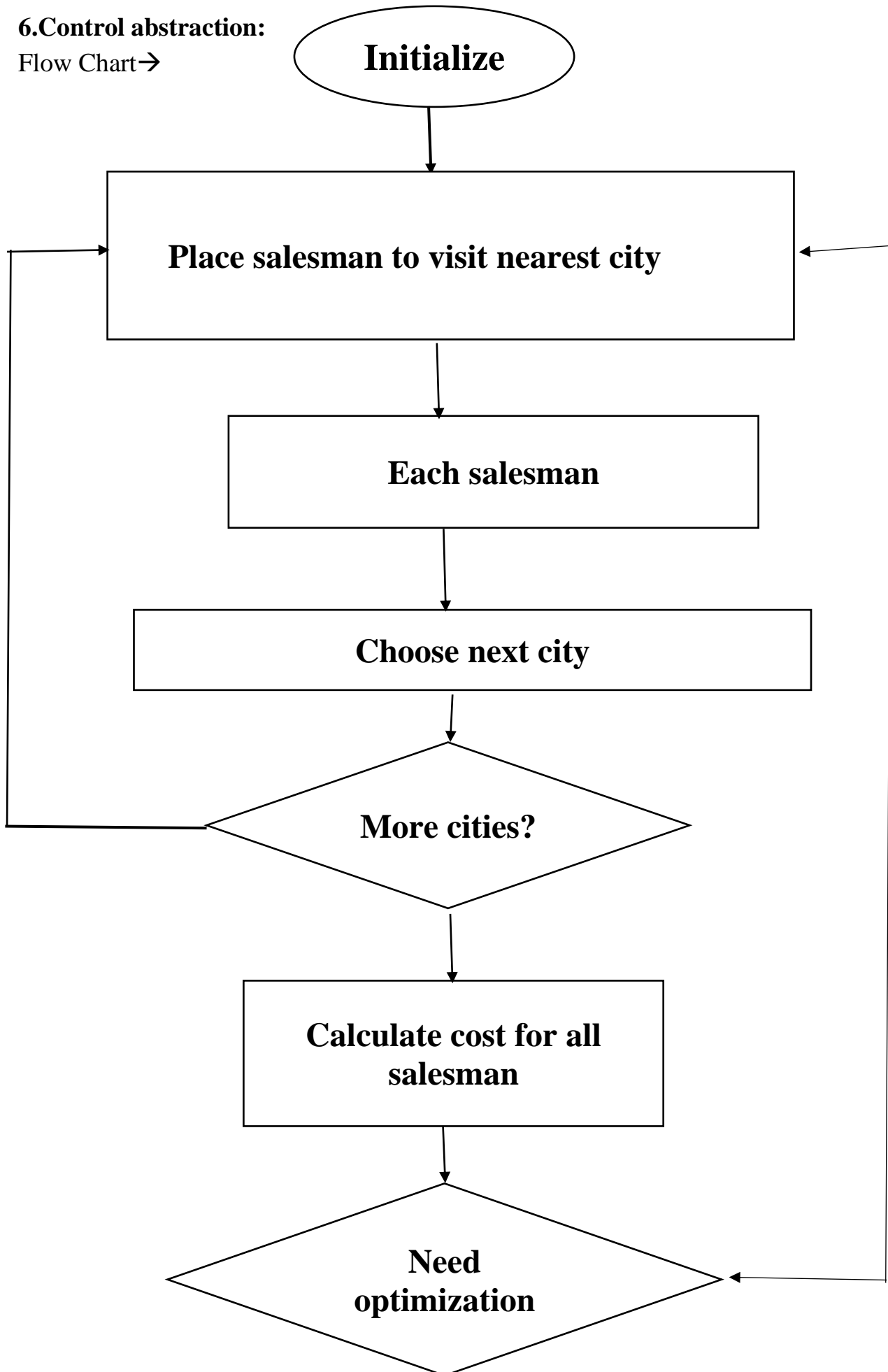
Mark area H4 and then select the minimum cost area reachable from H4 it is H1. So, using the greedy strategy, we get the following.

4 3 2 4 3 2 1 6

H1 → H6 → H7 → H8 → H5 → H2 → H3 → H4 → H1.

Thus the minimum travel cost = 4 + 3 + 2 + 4 + 3 + 2 + 1 + 6 = 25

6.Control abstraction:
Flow Chart→



7.Explanation of algorithmic steps or code of entire program

1. Greedy Strategy

Overview: The Greedy Strategy aims to build a solution piece by piece, always choosing the next piece that offers the most immediate benefit. For TSP, this typically means visiting the nearest unvisited city.

Steps:

1. Select a Starting City:
 - Choose any city to start the tour.
2. Create a List of Unvisited Cities:
 - Maintain a list of cities that have not yet been visited.
3. Visit the Nearest Unvisited City:
 - From the current city, calculate the distances to all unvisited cities.
 - Choose the nearest unvisited city and move to that city.
4. Mark the City as Visited:
 - Add the newly visited city to the list of visited cities and remove it from the unvisited list.
5. Repeat Until All Cities are Visited:
 - Continue the process of visiting the nearest unvisited city until all cities are visited.
6. Return to the Starting City:
 - After visiting all cities, return to the starting city to complete the tour.
7. Output the Tour and Total Distance:
 - The result will be the order of cities visited and the total distance of the tour.

Example:

- Cities: A, B, C, D
- Distances:
 - A-B: 10, A-C: 15, A-D: 20
 - B-C: 35, B-D: 25
 - C-D: 30

Starting from A, the nearest city might be B (10), then D (25), then C (35), and finally return to A.

2. Dynamic Programming

Steps Explained Simply:

1. Define the Problem:

- You have a list of cities and the distances between them. The goal is to find the shortest path that visits each city once and goes back to the starting point.

2. Create a Table:

- Set up a table dp where $dp[i][j]$ represents the shortest distance to visit all cities in the first i cities, ending at city j .

3. Initialization:

- Start with the distance from the starting city to itself, which is zero: $dp[1][0] = 0$
- This means if we are at the starting city (let's say city A), the cost is zero because we haven't moved.

4. Fill the Table:

- For each possible number of cities visited:

- For each city that could be the last visited city:
 - For each other city that could have been visited before this last city:
 - Update the distance: $dp[i][j] = \min(dp[i-1][k] + \text{distance}(k, j))$
 - Here, k is any other city that was visited before j. This means we are checking all possible paths that lead to city j and keeping track of the shortest one.

5. Calculate the Final Result:

- After filling in the table, find the shortest route that returns to the starting city:
 - Check the distances for all cities being the last visited city and add the distance back to the starting city.
 - This gives you the minimum distance for the complete tour.

6. Return the Result:

- The value you calculate in this step is the shortest route for the TSP.

Example:

Imagine you have three cities: A, B, and C with the following distances:

- A to B: 10
- A to C: 15
- B to C: 20

1. Start with the distance from A to A, which is zero.
2. Fill the table by considering routes that go from A to B, A to C, and then back to A through different combinations.
3. Finally, calculate the distance for routes that cover all cities and return to A.

In addition to using a **Greedy strategy** and **Dynamic programming** to solve the Traveling Salesman Problem, there are also other algorithms that can find approximations to the shortest route.

These algorithms are popular because they are much more effective than to actually check all possible solutions, but as with the greedy algorithm above, they do not find the overall shortest route.

Algorithms used to find a near-optimal solution to the Traveling Salesman Problem include:

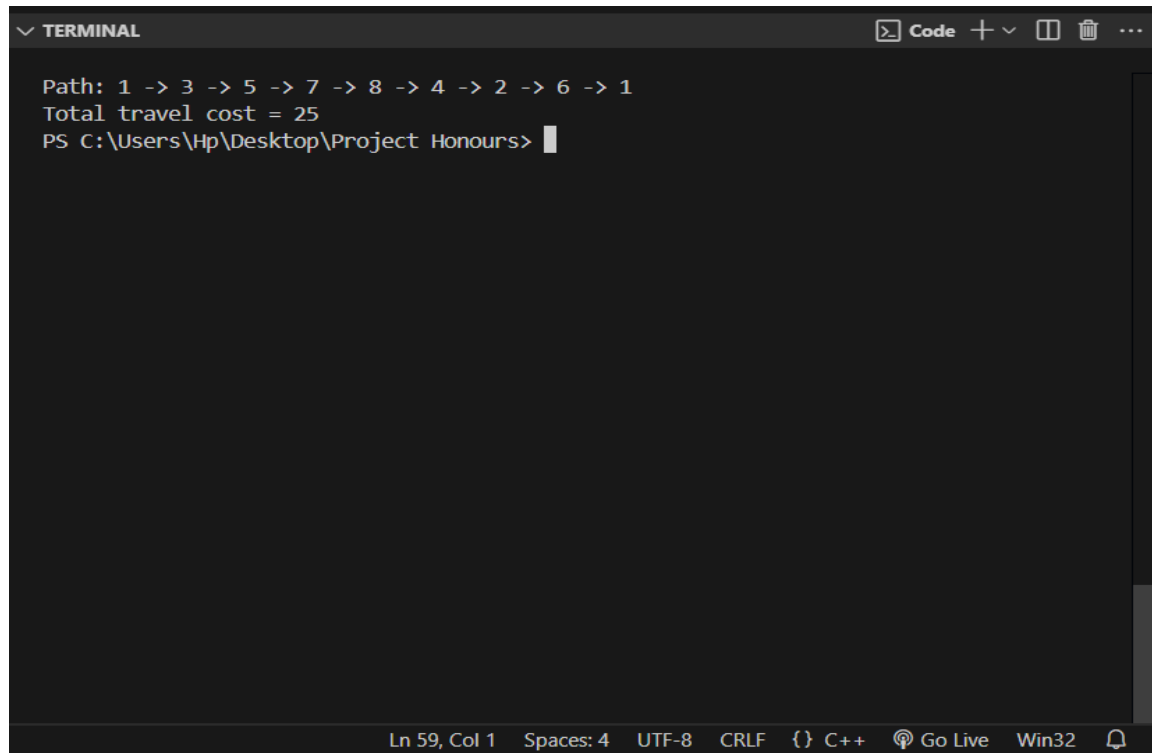
- **2-opt Heuristic:** An algorithm that improves the solution step-by-step, in each step removing two edges and reconnecting the two paths in a different way to reduce the total path length.
- **Genetic Algorithm:** This is a type of algorithm inspired by the process of natural selection and use techniques such as selection, mutation, and crossover to evolve solutions to problems, including the TSP.
- **Simulated Annealing:** This method is inspired by the process of annealing in metallurgy. It involves heating and then slowly cooling a material to decrease defects. In the context of TSP, it's used to find a near-optimal solution by exploring the solution space in a way that allows for occasional moves to worse solutions, which helps to avoid getting stuck in local minima.
- **Ant Colony Optimization:** This algorithm is inspired by the behaviour of ants in finding paths from the colony to food sources. It's a more complex probabilistic technique for solving computational problems which can be mapped to finding good paths through graphs.

8. Implementation

```
Go Run Terminal Help  ← →  Untitled (Workspace)
Welcome  tsp.cpp ×
Project Honours > tsp.cpp > ...
1  #include <iostream>
2  #include <climits>
3  using namespace std;
4
5  const int N = 8; // Number of cities
6
7  // Distance matrix
8  int dist[N][N] = {
9      {0, 4, 1, 3, 6, 7, 8, 2},
10     {4, 0, 5, 2, 7, 6, 3, 4},
11     {1, 5, 0, 6, 3, 5, 4, 7},
12     {3, 2, 6, 0, 5, 2, 6, 3},
13     {6, 7, 3, 5, 0, 4, 2, 6},
14     {7, 6, 5, 2, 4, 0, 5, 3},
15     {8, 3, 4, 6, 2, 5, 0, 1},
16     {2, 4, 7, 3, 6, 3, 1, 0}
17 };
18
19 // Greedy TSP function
20 void tspGreedy() {
21     bool visited[N] = {false}; // Track visited cities
22     int totalCost = 0;          // Total travel cost
23     int currentCity = 0;        // Start from city 0
24
25     cout << "Path: " << currentCity + 1;
26
27     visited[currentCity] = true;
28
29     for (int i = 1; i < N; i++) {
30         int nearestCity = -1;
31         int minDist = INT_MAX;
32
```

```
33         // Find the nearest unvisited city
34         for (int j = 0; j < N; j++) {
35             if (!visited[j] && dist[currentCity][j] < minDist) {
36                 minDist = dist[currentCity][j];
37                 nearestCity = j;
38             }
39         }
40
41         visited[nearestCity] = true;
42         totalCost += minDist;
43         currentCity = nearestCity;
44
45         cout << " -> " << currentCity + 1;
46     }
47
48     // Return to the starting city
49     totalCost += dist[currentCity][0];
50     cout << " -> " << 1 << endl;
51
52     cout << "Total travel cost = " << totalCost << endl;
53 }
54
55 int main() {
56     tspGreedy();
57     return 0;
58 }
59
```

Output



```
✓ TERMINAL Code + - [ ] [ ] ...  
Path: 1 -> 3 -> 5 -> 7 -> 8 -> 4 -> 2 -> 6 -> 1  
Total travel cost = 25  
PS C:\Users\Hp\Desktop\Project Honours> |  
Ln 59, Col 1 Spaces: 4 UTF-8 CRLF {} C++ Go Live Win32 [ ]
```

The image shows a terminal window with a dark background. The title bar at the top reads "✓ TERMINAL" and includes icons for "Code", a plus sign, a minus sign, a square, a trash can, and a menu. The terminal content displays the output of a program: "Path: 1 -> 3 -> 5 -> 7 -> 8 -> 4 -> 2 -> 6 -> 1" and "Total travel cost = 25". Below this, the command prompt "PS C:\Users\Hp\Desktop\Project Honours>" is shown with a cursor. The status bar at the bottom indicates "Ln 59, Col 1", "Spaces: 4", "UTF-8", "CRLF", "{} C++", "Go Live", "Win32", and a bell icon.

9. Time and space complexity of problem statement

1. Brute Force (Naive Approach):

- How it works: Try all possible routes and choose the one with the least travel cost.
- Time Complexity: $O(n!)$ (too slow for large inputs)
- Space Complexity: $O(n^2)$ (for storing distances between houses)

2. Dynamic Programming (Efficient Approach):

- How it works:
 - We break down the problem into smaller subproblems (subsets of houses) and solve them step by step, storing the results so that we don't have to calculate them again.
 - The dynamic programming approach to TSP is called the Held-Karp algorithm.
 - For example, it calculates the shortest path for visiting groups of houses first, and then uses that information to find the shortest path for the full set of houses.
 - It uses a bit masking technique to represent which houses have been visited.
- Time Complexity: $O(n^2 * 2^n)$, which is much better than brute force but still gets slow as the number of houses increases.
- Space Complexity: $O(n * 2^n)$ for storing the results of subproblems.

3. Greedy Method (Fast but not Always Optimal):

- How it works:
 - The greedy method makes decisions step by step, picking the shortest immediate travel option (the house that is closest to the current location).
 - The idea is to always go to the nearest unvisited house.

- However, this doesn't always guarantee the best overall solution—it's like taking the locally best option without thinking of the whole picture.
- Time Complexity: $O(n^2)$, much faster because it just looks at the next closest house.
- Space Complexity: $O(n^2)$ for storing the distances between houses.

Comparison of Approaches:

Method	How it works	Time Complexity	Space Complexity
Brute Force	Try every possible route	$O(n!)$	$O(n^2)$
Dynamic Programming	Break the problem into smaller parts and use past results (Held-Karp algorithm)	$O(n^2 * 2^n)$	$O(n * 2^n)$
Greedy Method	Go to the nearest house at each step	$O(n^2)$	$O(n^2)$

Which One to Use?

- Brute Force: Not practical for more than a few houses, as it takes too long.
- Dynamic Programming (DP): Gives the optimal solution but is still a bit slow for large inputs. Works well for up to 20-30 houses.
- Greedy Method: Very fast, but may not give the best solution. Works well if you need a quick solution for many houses, but you should be okay with some inaccuracy.

10. Conclusion

1.The Traveling Salesman Problem (TSP) is a cornerstone in the field of optimization and computational mathematics, encapsulating the challenges of finding efficient solutions to complex real-world problems. Despite its simplicity in formulation, TSP is computationally intensive due to its NP-hard nature, making exact solutions infeasible for large instances.

2.The significance of TSP lies in its practical applications, from logistics and supply chain management to circuit design and bioinformatics, where optimizing routes and sequences is crucial. While exact algorithms like dynamic programming and branch-and-bound provide solutions for small-scale instances, heuristic and metaheuristic approaches offer practical approximations for larger problems.

3.TSP serves as a gateway to understanding combinatorial optimization and inspires the development of innovative algorithms and computational techniques. It exemplifies the balance between theoretical complexity and practical applicability, making it a subject of ongoing research and a benchmark for algorithmic innovation.

11. References

1. GeeksforGeeks : <https://www.geeksforgeeks.org/traveling-salesman-problem-using-dynamic-programming/>
2. Javatpoint : <https://www.javatpoint.com/travelling-salesman-problem>
3. TutorialsPoint : <https://www.tutorialspoint.com/Travelling-Salesman-Problem>

