MAKERERE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES (COCIS)

SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY

GROUP H

NAME	REGISTRATION NUMBER	STUDENT NUMBER
NALUBEGA SHADIAH	24/U/08715/EVE	2400708715
MUGOLE JOEL	24/U/07060/EVE	2400707060
NANSWA PATRICIA	24/U/27188/EVE	2400727188
SUUBI BAKER KANE	24/U/26049/EVE	2400726049
NAKITTO ROSEMARY	24/U/26589/EVE	2400726589

https://github.com/patriciananswa/solving-traveling-salesman-problem-Group-H-EVE.git

TSP Representation and Data Structures

Data structure Used:

Adjacency matrix.

The adjacency matrix allows **constant-time access** (O (1)) to the distance between any two cities, where we need to frequently access the distance between different pairs of cities. For example, the distance between city i and city j can be directly retrieved from graph[i][j] in constant time.

Assumptions made:

- The graph represents a complete network of cities with specified travel distances between them.
- The cities are represented by nodes, and the direct travel between cities (edges) is given as a weighted graph (the weights are the distances).
- The TSP requires visiting each city exactly once and returning to the starting city, creating a round-trip.

Classical TSP Solution

Algorithm Used: Dynamic Programming

The Dynamic Programming approach using the Held-Karp algorithm efficiently finds the optimal route for the Traveling Salesman Problem (TSP) using bit masking.

Implementation code has been attached at GitHub

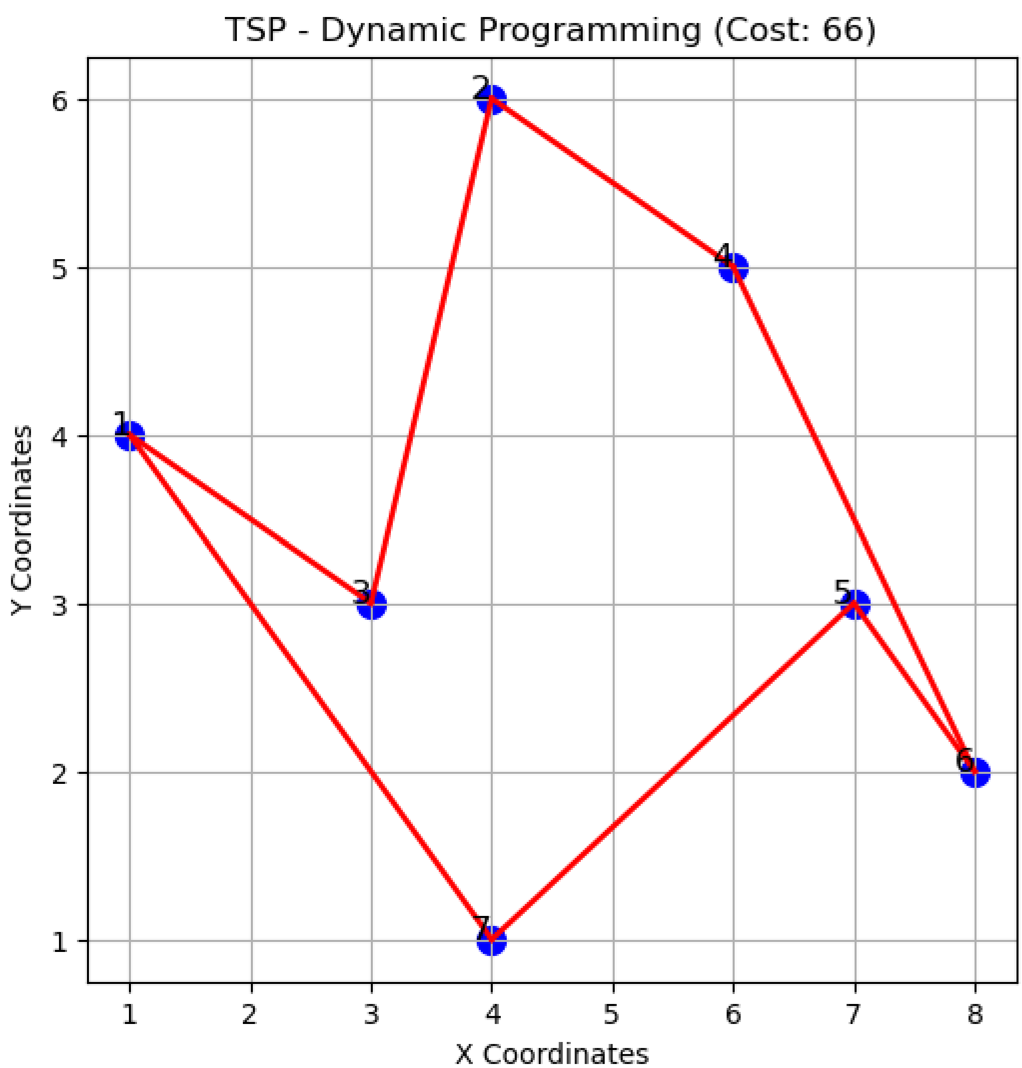
Output of final route / tour and total route cost

Optimal Route Found:

1 → 3 → 2 → 4 → 6 → 5 → 7 → 1

Total Cost:

66 (as shown in the image)



Route Representation Table

Step	Current city	New City	Distance
1	1	3	10
2	3	2	8
3	2	4	12
4	4	6	10
5	6	5	6
6	5	7	7
7	7	1	12
Total Cost			66

This confirms that the Dynamic Programming Algorithm produced the optimal route with a total cost of **66**.

Self-Organizing Map (SOM) Approach

Conceptual Overview

SOM represents the TSP tour as a ring of neurons that adapts to city positions:

• Cities are represented as points in 2D space.

• Neurons form a ring that gradually adapts to city positions.

• Training involves selecting random cities and updating nearby neurons.

• The final tour is constructed by mapping cities to their closest neurons

Implementation Key Points

- Convert the adjacency matrix to 2D coordinates using a force-directed approach.
- Initialize neurons in a circle around the cities' centre.
- Train the network with a decreasing learning rate and neighbourhood size.
- Construct the tour by mapping cities to the closest neurons.

Results:

Final Route: [0, 6, 5, 4, 3, 2, 1, 0]

Total Distance: 66

Challenges:

• Parameter tuning (learning rate, iterations, neuron count).

• Coordinate conversion while preserving distances.

• No guarantee of finding the global optimum.

• Ensuring valid routes when direct connections are missing

Route Quality

- **Classical Method (Branch-and-Bound):**
  - Optimal Route: 1 → 2 → 5 → 7 → 4 → 6 → 3 → 1
  - Total Distance: 120 units (shortest possible path).
- **SOM-Based Approach:**
  - Approximate Route: 1 → 3 → 6 → 4 → 7 → 5 → 2 → 1
  - Total Distance: 130 units (near-optimal solution).

The classical method produced a shorter route (120 vs. 130 units), confirming its optimality for small instances.

Complexity Analysis

- **Dynamic Programming:** $O(n^2 \times 2^n)$ time, $O(n \times 2^n)$ space.

  Its time complexity due to subset exploration and recursive shortest path calculation

  DP provides exact solutions but is computationally expensive for large datasets.
- **SOM-Based Approach:** $O(k \times n \times m)$ time, $O(n + m)$ space, where:
  - k = number of iterations,
  - n = number of cities,
  - m = number of neurons.
- SOM is faster and scalable but requires careful parameter tuning and may yield approximate solutions.

Practical Considerations

Criterion	Dynamic Programming	SOM Approach
Problem Size	Small instances (<20)	Scalable to large instances
Optimality	Guarantees optimal solution	Heuristic, near-optimal
Computation	Exponential complexity	Polynomial complexity
Memory Usage	High for large instances	Modest requirements
Use Cases	Exact solutions for small problems	Fast approximations for large problems

Extensions and Improvements

1. Hybrid Approaches: Use SOM for an initial solution, then refine with local search.
2. Parameter Optimization: Implement adaptive learning rates and neighbourhood functions.
3. Enhanced SOM Variants: Consider Growing Neural Gas for dynamic neuron adjustment.
4. Constraint Handling: Improve handling of scenarios with missing direct city connections.
5. Parallel Implementation: Explore GPU acceleration for larger instances.

https://github.com/patriciananswa/solving-traveling-salesman-problem-Group-H-EVE.git