MAKERERE UNIVERSITY

Traveling Salesman Problem: Classical and SOM-Based Methods

March 18, 2025

Participants

Name	Registration Number	Student Number
Mujuni Innocent	24/U/07155/PS	2400707155
Wangoda Francis	24/U/11855/PS	2400711855
Bwanika Robert	24/U/23908/PSA	2400723908
Kinda Kelsey Naluwafu	24/U/05862/PS	2400705862
Naluyima Pauline Oliver	24/U/08847/PS	2400708847

Introduction

The Traveling Salesman Problem (TSP) requires determining the shortest route that visits each city exactly once and returns to the starting point. In this assignment, we solve a TSP instance defined on a graph of 7 cities (with City 1 as the origin) using two approaches: a classical exact method via dynamic programming (Held-Karp algorithm) and a heuristic method based on Self-Organizing Maps (SOM). Detailed code is provided in the separate files `tsp_dynamic_programming_solution.py` and `tsp_som.py`.

TSP Representation and Data Structures

Graph Representation

```
adjacency_matrix = [
    [0, 12, 10, inf, inf, inf, 12],   # Node 1-start (index 0)
    [12, 0, 8, 12, inf, inf, inf],    # Node 2 (index 1)
    [10, 8, 0, 11, 3, inf, 9],        # Node 3 (index 2)
    [inf, 12, 11, 0, 11, 10, inf],    # Node 4 (index 3)
    [inf, inf, 3, 11, 0, 6, 7],       # Node 5 (index 4)
    [inf, inf, inf, 10, 6, 0, 9],     # Node 6 (index 5)
    [12, inf, 9, inf, 7, 9, 0]        # Node 7 (index 6)
]
```

Justification

Adjacency matrix provides $O(1)$ lookup time for distances between cities. For this small problem (7 cities), the $O(n^2)$ space complexity is acceptable, and the representation naturally handles the symmetric, bidirectional nature of the routes.

Problem Statement

The formal problem statement is: *"Visit each city exactly once and return to the starting city while minimizing the total travel distance."*

Assumptions

- The graph is undirected and symmetric; hence, the distance from city A to city B equals the distance from city B to city A.
- All distances are non-negative.
- Although the adjacency matrix uses `inf` to denote missing direct connections, it is assumed that the graph is connected ensuring a valid tour exists.

- Each city is visited exactly once, and the tour concludes at the starting city.
- The problem size (7 cities) is small enough for the dynamic programming approach to be computationally feasible.

Classical TSP Solution

Algorithm: Dynamic Programming

- Uses state representation (mask, city) where the mask tracks visited cities.
- Builds solutions incrementally using memoization to avoid recalculations.
- Guarantees an optimal solution by exploring all possible paths systematically.

Results

```
Route: 1 > 3 > 5 > 7 > 6 > 4 > 2
Total Distance: 63
```

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

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

Results

```
Route (Random): 1 > 7 > 5 > 6 > 4 > 2 > 3 > 1
Total Distance (Average): 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.

Analysis and Comparison

Route Quality

The dynamic programming approach guarantees the optimal route, while the SOM method yields a near-optimal solution.

Complexity Analysis

- **Dynamic Programming**: $O(n^2 \times 2^n)$ time, $O(n \times 2^n)$ space.
- **SOM**: $O(k \times n \times m)$ time, $O(n + m)$ space, where k = iterations, n = cities, m = neurons.

Practical Considerations

Criterion	**Dynamic Programming**	**SOM Approach**
Problem Size	Small instances (<20 cities)	Scalable to larger instances
Optimality	Guarantees optimal solution	Heuristic, potentially suboptimal
Computation	Exponential complexity	Polynomial complexity
Memory	High for larger instances	Modest requirements
Use Cases	Exact solutions for small problems	Fast approximations for larger 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 neighborhood functions.
3. **Enhanced SOM Variants**: Consider Growing Neural Gas for dynamic neuron adjustment.
4. **Constraint Handling**: Improve handling of situations with no direct connection between cities.
5. **Parallel Implementation**: Explore GPU acceleration for larger instances.