Comparative Analysis of Recursive and Iterative Approaches to the Travelling Salesman Problem (TSP)

Research Team – TSP Analysis Module

May 13, 2025

Abstract

This report documents the analysis and comparison of two algorithmic approaches to solving the Travelling Salesman Problem (TSP): a recursive dynamic programming method and an iterative dynamic programming method. Both implementations are tested across identical fixed scenarios to ensure a fair performance comparison in terms of runtime and space complexity.

1 Problem Overview

The Travelling Salesman Problem (TSP) is a classic NP-hard problem in combinatorial optimization. Given a list of cities and the distances between each pair, the goal is to find the shortest possible route that visits each city exactly once and returns to the origin city.

2 Mathematical Formulation

Formally, the TSP can be modeled as follows:

Given:

- A set of cities $V = \{1, 2, \ldots, n\}$,
- A distance matrix $D = [d_{ij}]$, where d_{ij} is the distance from city i to city j,

Objective:

Minimize the total travel cost:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

Where:

$$x_{ij} = \begin{cases} 1, & \text{if the tour goes directly from city } i \text{ to city } j \\ 0, & \text{otherwise} \end{cases}$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in V \quad \text{(leave each city once)}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \in V \quad \text{(arrive at each city once)}$$

Subtour elimination constraints (to ensure a single tour)

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V$$

Note:

The subtour elimination constraints are typically formulated using additional variables (e.g., Miller–Tucker–Zemlin formulation) in optimization solvers, but are handled implicitly in dynamic programming approaches by managing visited states.

3 Recursive Dynamic Programming Approach

Logic

This approach uses memoization to avoid recomputation. It explores all possible states recursively and stores subproblem results in a 2D memoization table indexed by current node and visited state.

Pseudocode

```
def tsp(i, state):
    if state == FINISHED_STATE:
        return distance[i][START_NODE]
    if memo[i][state] is not None:
        return memo[i][state]

    min_cost = inf
    for next in all unvisited nodes:
        cost = distance[i][next] + tsp(next, state | (1 << next))
        memo[i][state] = min(cost, memo[i][state])
    return memo[i][state]
```

Time and Space Complexity

- Time: $O(n^2 \cdot 2^n)$
- Space: $O(n \cdot 2^n)$

4 Iterative Dynamic Programming Approach

Logic

The iterative approach builds up solutions from small subsets using bitmasking and a memoization table. It eliminates recursion overhead by explicitly constructing the solution via nested loops over subset sizes.

Pseudocode

```
for r in range(3, N+1):
    for subset in combinations(r, N):
        if start not in subset: continue
        for next in subset:
            if next == start: continue
            min_dist = inf
            for end in subset:
                if end == start or end == next: continue
                cost = memo[end][subset ^ (1 << next)] + distance[
                   end][next]
                min_dist = min(min_dist, cost)
            memo[next][subset] = min_dist
```

Time and Space Complexity

- Time: $O(n^2 \cdot 2^n)$
- Space: $O(n \cdot 2^n)$

5 Test Case Design

To ensure consistent comparison:

- Distance matrices were generated symmetrically (undirected graphs).
- Diagonal elements are zero (no self-loops).
- Matrices range from $n = 4$ to $n = 10$ cities.
- Each test case was reused for both implementations.

6 Runtime Analysis

Execution times were recorded using Python's `time.perf_counter()`:

- Recursive and Iterative runtimes were measured per matrix size.
- Data was stored in a DataFrame and exported for visualization.

7 Space Usage Estimation

We estimated peak memory usage using:

- `tracemalloc` to track memory allocations.
- Peak snapshot comparisons between recursive and iterative methods.

8 Results Summary

Preliminary results indicate:

- Recursive version is easier to read but consumes more stack space.
- Iterative version scales better in practice due to no recursion stack.
- Both perform identically in terms of tour cost accuracy.

9 Conclusion

This report has presented a comparative study of two classical TSP approaches. While both share the same time and space complexity in theory, their practical runtime and space behavior differ slightly. The iterative method is more stable in constrained environments.

Next Steps

The results of this analysis will serve as a baseline for comparison with a Quantum-inspired QUBO implementation of the TSP, being handled by another member of the team.