VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Discrete Structures (CO1007)

Assignment

Traveling Salesperson Problem

CC01 – Semester 242

Advisor(s):	Nguyễn Văn Minh Mẫn	Mahidol University
	Nguyễn An Khương	CSE-HCMUT
	Lê Hồng Trang	CSE-HCMUT
	Trần Tuấn Anh	CSE-HCMUT
	Trần Hồng Tài	CSE-HCMUT
	Mai Xuân Toàn	CSE-HCMUT
Student(s):	Lê Quốc Tuấn	2153944

HO CHI MINH CITY, JUNE 2025

Contents

1 Introduction **3**

2 Held-Karp Dynamic Programming (Exact, $n \leq 20$) **3**

2.1 State Definition 3

2.2 Base Case 4

2.3 Transition 4

2.4 Final Cost Computation 4

2.5 Path Reconstruction 4

2.6 Bitmasking Tricks 4

2.7 Complexity 5

3 Heuristic Approach for Large Instances ($n > 20$) **5**

3.1 Phase 1 – Nearest-Neighbour (NN) Construction 5

3.2 Phase 2 – 2-Opt Local Improvement 5

4 Conclusion **5**

1 Introduction

The Traveling Salesperson Problem (TSP) is a classic problem in combinatorial optimization. Given a list of cities and the distances (or costs) between each pair of cities, the problem asks for the shortest possible route that visits each city exactly once and returns to the origin city.

Formally, given a set of n cities and a distance matrix $D = (d_{ij})$, where d_{ij} is the distance from city i to city j, the goal is to find a permutation π of the cities $1, \ldots, n$ that minimizes the total tour length:

$$\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}$$

TSP is an NP-hard problem, meaning that no known polynomial-time algorithm can solve it for all instances. For small numbers of cities, exact algorithms can be used, while for larger instances, heuristic or approximation algorithms are often employed.

Because TSP is NP-hard, **exact** algorithms explode for large n: We therefore combine an exact method (Held-Karp) with an efficient heuristic (Nearest-Neighbour + 2-Opt) and **dispatch at run time**:

$$n \leq 20 \;\Rightarrow\; \text{Exact Method} \qquad n > 20 \;\Rightarrow\; \text{Heuristic Method.}$$

2 Held-Karp Dynamic Programming (Exact, $n \leq 20$)

The Held-Karp algorithm, also known as the Bellman-Held-Karp algorithm, is an exact algorithm for solving the Traveling Salesperson Problem. It uses dynamic programming to systematically explore subproblems and build up to the optimal solution. While still exponential, its complexity of $O(n^2 \cdot 2^n)$ is a significant improvement over the brute-force $O(n!)$ approach.

2.1 State Definition

We define dp[mask][i] as the minimum cost to reach city i having visited all cities in the set represented by ‘mask‘. Here:

- `mask` is an integer bitmask indicating visited cities.

- i is the current city (the last city visited).

2.2 Base Case

Assuming we start at city s, the base case is:

$$\text{dp}[1 \ll s][s] = 0$$

2.3 Transition

For every subset of visited cities (`mask`) and every city i in that subset, we attempt to extend the path by visiting an unvisited city j:

$$\text{dp}[\text{mask} \mid (1 \ll j)][j] = \min\left(\text{dp}[\text{mask}][i] + \text{adjMatrix}[i][j]\right)$$

We also track parent pointers for reconstruction:

$$\text{parent}[\text{mask} \mid (1 \ll j)][j] = i$$

2.4 Final Cost Computation

After filling the table, the minimum tour cost is computed by completing the cycle back to the starting city:

$$\min_{i \neq s}\left(\text{dp}[(1 \ll n) - 1][i] + \text{adjMatrix}[i][s]\right)$$

2.5 Path Reconstruction

To recover the path, we backtrack from the final state using the `parent` table. Starting from the city that gave the minimum total cost and the full bitmask, we trace the path in reverse.

2.6 Bitmasking Tricks

Bitmask operations used:

- $(1 \ll k)$: bitmask with only the k-th city.
- `mask & (1 << k)`: check if k is in the set.
- `mask | (1 << k)`: add k to the set.
- `mask ^ (1 << k)`: remove k from the set.

2.7 Complexity

Time: $\Theta(n^2 2^n)$ – looping over all subsets and possible last cities. **Space:** $\Theta(n2^n)$ – for the `dp` and `parent` tables.

This limits practical use of Held-Karp to roughly $n \leq 20$, where it still guarantees optimality.

3 Heuristic Approach for Large Instances ($n > 20$)

3.1 Phase 1 – Nearest-Neighbour (NN) Construction

Starting from the user-specified start vertex s, repeatedly append the cheapest **unvisited** neighbour. This greedy sweep runs in $O(n^2)$ using the already-allocated adjacency matrix.

3.2 Phase 2 – 2-Opt Local Improvement

Given the NN tour, iteratively perform 2-Opt edge swaps: for every pair of edges (v_i, v_{i+1}), (v_j, v_{j+1}) with $i < j - 1$, swap them when it shortens the route. The worst-case time is $O(n^2)$ per pass; in practice a handful of passes converge. We cap the passes at 100 or ‘n‘ (whichever is smaller) to guarantee termination.

4 Conclusion

For graphs up to 20 vertices we still guarantee optimality via Held-Karp. Beyond that, the NN + 2-Opt heuristic delivers near-optimal tours in quadratic time and linear memory.