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: Held-Karp Algorithm Report

**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 to the Traveling Salesperson Problem (TSP)

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 $\pi$ 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.

# 2 The Held-Karp Algorithm (Dynamic Programming Approach)

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 Problem Formulation and State Definition

The core idea of dynamic programming is to break down a complex problem into smaller, overlapping subproblems. For Held-Karp, a subproblem is defined by:

- A subset of cities that have already been visited.
- The last city visited in that subset.

Let $n$ be the total number of cities. We define our dynamic programming state as:

$$\text{dp}[\text{mask}][i]$$

where:

- mask is an integer representing a bitmask of visited cities. If the $k$-th bit of mask is set (i.e., 1), it means city $k$ has been visited. Otherwise, city $k$ has not been visited.
- $i$ is the index of the city that was **last visited** in the path corresponding to the mask.

The value stored in dp[mask][$i$] is the minimum cost to visit all cities represented by mask, ending at city $i$.

## 2.2 Base Case

The base case for our dynamic programming relation is when we start at a particular city. Let's assume city 0 is our starting city (after mapping character vertices to indices). The cost of visiting only the starting city and ending at the starting city is 0.

$$\text{dp}[1 \ll \text{start\_vertex\_idx}][\text{start\_vertex\_idx}] = 0$$

Here, $1 \ll$ start_vertex_idx creates a mask where only the bit corresponding to the starting city's index is set.

## 2.3 Recursive Relation / Transitions

To compute dp[mask][$i$], we consider all possible cities $j$ that could have been visited immediately before city $i$ in the path represented by mask. The transition formula is:

$$\text{dp}[\text{mask}][i] = \min_{j \in \text{mask}, j \neq i} (\text{dp}[\text{mask} \setminus \{i\}][j] + \text{distance}(j, i))$$

In an iterative implementation, we build up the solution by iterating through masks and then through possible current and next cities. For each mask (representing a set of visited cities) and each city $i$ in that mask (representing the current end city): If dp[mask][$i$] is a valid (reachable) state: We try to extend the path to an unvisited city $j$.

- A city $j$ is unvisited if its bit is not set in mask (i.e., $\neg(\text{mask}\&(1 \ll j))$).
- There must be a direct edge from city $i$ to city $j$.
- The new mask will be next_mask = mask | $(1 \ll j)$.
- The cost to reach next_mask ending at $j$ via $i$ is dp[mask][$i$] + adjMatrix[$i$][$j$].

We update dp[next_mask][$j$] if this 'new_cost' is less than the currently stored value, and record $i$ as the parent of $j$ for path reconstruction:

$$\text{dp[next\_mask][}j\text{]} = \min(\text{dp[next\_mask][}j\text{]}, \text{dp[mask][}i\text{]} + \text{adjMatrix[}i\text{][}j\text{]})$$

$$\text{parent[next\_mask][}j\text{]} = i$$

### 2.4 Finding the Minimum Tour Cost

After filling the entire dp table, the minimum total tour cost is found by considering all possible last cities $i$ in the full tour (where all cities are visited, represented by final_mask = $(1 \ll n) - 1$). For each such $i$, we add the cost of returning from $i$ to the starting city:

$$\text{min\_total\_cost} = \min_{i=0\ldots n-1} (\text{dp[final\_mask][}i\text{]} + \text{adjMatrix[}i\text{][start\_vertex\_idx]})$$

### 2.5 Path Reconstruction

To reconstruct the actual path, we use the 'parent' table. Starting from the 'last_vertex_in_tour' (the city $i$ that yielded the 'min_total_cost') and the 'final_mask', we backtrack using the 'parent' pointers until we reach the starting city and its base mask. The path is then reversed to get the correct order.

## 3 Bitmasking in Held-Karp

Bitmasking is a technique where an integer's binary representation is used to store a set of boolean flags. In Held-Karp, it efficiently represents the subset of cities visited.

- Each bit position corresponds to a city's index (e.g., bit 0 for city 0, bit 1 for city 1, etc.).
- A bit set to 1 means the city is in the subset.
- A bit set to 0 means the city is not in the subset.

Common bitwise operations used:

- '(1 « k)': Creates a mask with only the $k$-th bit set.
- 'mask & (1 « k)': Checks if the $k$-th bit is set in 'mask'.
- 'mask | (1 « k)': Adds city $k$ to the set represented by 'mask'.
- 'mask $\oplus$ (1 « k)': Removes city $k$ from the set represented by 'mask'.

This allows for compact storage and efficient operations on subsets of cities.

# 4 Complexity Analysis

- **Time Complexity:**
  - The outer loop iterates through $2^n$ possible masks.
  - The next loop iterates through $n$ possible current end cities ($i$).
  - The innermost loop iterates through $n$ possible next cities ($j$).

  Therefore, the total time complexity is $O(n \cdot n \cdot 2^n) = O(n^2 \cdot 2^n)$.
- **Space Complexity:**
  - The 'dp' table has dimensions $2^n \times n$.
  - The 'parent' table also has dimensions $2^n \times n$.

  Thus, the total space complexity is $O(n \cdot 2^n)$.

The exponential complexity limits the practical applicability of Held-Karp to instances with a small number of cities (typically $n \leq 20 - 25$).

# 5 Conclusion

The Held-Karp algorithm offers an exact solution to the Traveling Salesperson Problem using dynamic programming and bitmasking. While its exponential time and space complexity restrict its use to small graphs, it remains a fundamental algorithm for understanding exact solutions to NP-hard problems. Its elegance lies in systematically building optimal paths for subsets of cities, ultimately leading to the shortest possible Hamiltonian cycle. For larger instances of TSP, approximation algorithms or heuristics are typically preferred.