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

DISCRETE STRUCTURE FOR COMPUTER SCIENCE

Large assignment

Finding the shortest path

Instructor: Ph.D Tran Tuan Anh
Student(s): Dao Duy Dat - 2352223

HO CHI MINH CITY, JUNE 2025

Contents

1 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classic optimization problem where a person must find the shortest possible route that visits each of N cities exactly once and returns to the starting city. The goal is to minimize the total travel cost.

In this exercise, I implemented two different approaches to solve TSP depending on the input size:

- **Bitmask Dynamic Programming:** Used for small instances ($N \leq 20$), where we apply DP with bitmasking to explore all subsets of cities efficiently.
- **Genetic Algorithm:** Used for larger instances (e.g., $N = 25$), where we evolve a population of candidate solutions using selection, crossover, and mutation to find a near-optimal tour.

1.1 Bitmask Dynamic Programming Approach

Goal: Find the shortest Hamiltonian cycle that visits all V cities and returns to the starting point.

State Representation:

Let:

- $mask$ be a bitmask representing the set of visited vertices.
- u be the current vertex.

What is bitmask? Bitmask is a binary number used for representing something. For e.g: we have five bulbs, which are numbered from 1 to 5, to show that the bulbs at position 1 and 4 are on, we can write a bitmask: 10010.

Some operations with bitmask:

- **Set the i^{th} bit:**
  $$b \mid (1 \ll i)$$
  This sets the bit at position i (0-indexed from the right) to 1.

  Example: Let $b =$ `0b10011` (19 in decimal), set bit 3:
  $$\texttt{0b10011} \mid (1 \ll 3) = \texttt{0b11011}$$
- **Clear the i^{th} bit:**
  $$b \ \& \ \sim (1 \ll i)$$
  This clears (sets to 0) the bit at position i.

  Example: Let $b =$ `0b10111`, clear bit 4:
  $$\texttt{0b10111} \ \& \ \sim (1 \ll 4) = \texttt{0b00111}$$
- **Toggle the i^{th} bit:**
  $$b \ \oplus \ (1 \ll i)$$
  This flips the bit at position i.

  Example: Let $b =$ `0b10001`, toggle bit 2:
  $$\texttt{0b10001} \ \oplus \ (1 \ll 2) = \texttt{0b10101}$$
- **Check if the i^{th} bit is set:**
  $$b \ \& \ (1 \ll i)$$
  If the result is non-zero, the i^{th} bit is set to 1.

  Example: Let $b =$ `0b10010`, check bit 3:
  $$\texttt{0b10010} \ \& \ (1 \ll 3) = 0$$

- **Left Shift:**
$$1 \ll i$$
Shifts 1 to the left by i positions, setting the i^{th} bit from the right.

- **Right Shift:**
$$b \gg i$$
Shifts the bits of b to the right by i positions, effectively dividing by 2^i.

Bitmask takes a vital role of storing keep track of which cities have been visited without explicitly storing the actual set. Suppose we have n cities (vertices) labeled from 0 to n-1. We represent the set of visited cities using a binary mask of length n, where each bit corresponds to a city.
If the i-th bit is 1: the city has been visited.
If the i-th bit is 0: the city has not been visited.
We define:

$$dp[mask][u] = \text{Minimum cost to reach vertex } u \text{ after visiting all cities in } mask$$

Transition:
For each unvisited vertex v:

$$dp[mask \cup \{v\}][v] = \min\left(dp[mask \cup \{v\}][v],\ dp[mask][u] + cost[u][v]\right)$$

Initialization:

$$dp[1 << start][start] = 0$$

Final Cost:
Once all cities are visited:

$$\min_{i \in [0, V-1]} \left(dp[(1 << V) - 1][i] + cost[i][start]\right)$$

Time Complexity:

$$O(V^2 \cdot 2^V)$$

This approach is efficient for small values of V (up to 20).

1.2 Genetic Algorithm Approach

Goal: Find an approximate solution to TSP for larger V using a population-based evolutionary strategy.

Chromosome Representation: Each individual is a permutation of $\{0, 1, ..., V-1\}$ representing a travel route.

Algorithm Steps:

1. **Initialization:**
   - The first individual is generated using the Nearest Neighbor heuristic.
   - The rest are randomly generated permutations.
2. **Fitness Function:**
$$fitness(ind) = \text{Total travel cost of the path}$$
3. **Selection:** Tournament selection between two randomly chosen individuals.
4. **Crossover (Order Crossover - OX):**
   - Select two cut points l and r.
   - Copy segment $[l, r]$ from parent 1 to child.
   - Fill the remaining values from parent 2 in order, avoiding duplicates.
5. **Mutation:** With a small probability, swap two cities in the tour.
6. **Elitism:** Carry the best individual to the next generation.
7. **Evolution:** Repeat the selection, crossover, mutation, and fitness update for a fixed number of generations.

Time Complexity:

$$O(POP\_SIZE \cdot GENERATIONS \cdot V)$$

This method scales better for larger values of V (e.g., 25 cities).