Contents

Introduction

The goal of implementing the genetic algorithm is to iteratively evolve a population of routes, selecting for individuals with shorter traveling times, and applying crossover and mutation operations to discover an optimal solution to the TSP.

By navigating the complex solution space efficiently, the genetic algorithm aims to converge towards the shortest and most efficient route for the Traveling Salesman.

This project will explore the interplay between genetic algorithm components and their effectiveness in solving the Traveling Salesman Problem. The implementation is conducted in Python, leveraging concepts from evolutionary computing to address this well-known combinatorial optimization challenge.
This project focuses on the implementation of a Genetic Algorithm (GA) to tackle the Traveling Salesman Problem (TSP).
The task is about applying and comparing the results from different unsupervised learning techniques:

1. Chromosome Representation
2. Order Crossover (OX)
3. The Traveling Salesman Problem (TSP)
4. Mutation Techniques
5. Genetic Algorithm
6. Visualization

Data Preparation

Chromosome Representation

The chromosome in this implementation represents a potential route for the traveling salesman. It is a permutation of cities, where each gene corresponds to a city, and the order of cities in the chromosome represents the order in which the salesman will visit them.
In this TSP (Traveling Salesman Problem) genetic algorithm implementation, the chromosome represents a potential solution to the problem. Each chromosome is a permutation of cities, where each gene corresponds to a city. The order of cities in the chromosome represents the order in which the salesman will visit them.

Selection Techniques

Tournament Selection:
This method randomly selects a subset (tournament) of individuals from the population and then chooses the best individual from this subset as a parent.
Elitism:
The best solution from the previous generation is directly passed to the next generation without any changes, ensuring that the best solution is not lost.

Implementation Decisions

The Tournament_selection function randomly samples individuals and returns the one with the minimum total distance.
The best solution is added to the new population before the selection process.

Dataset

We took the dataset from the given link in the task file, and then we rename the dataset file as shown below,

1. 5.txt is a set of 5 cities.
2. 17.tsp is a set of 17 cities, from TSPLIB.
3. 26.tsp is a set of 26 cities, from TSPLIB.
4. 42.tsp is a set of 42 cities, from TSPLIB.
5. 48.tsp is a set of 48 cities (US state capitals) from TSPLIB.

This is the link to find all the 5 datasets.
https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

Crossover Techniques

Order Crossover (OX):
OX is a technique that combines genetic material from two parents to create a child. It starts by randomly selecting a subset of genes from one parent and then fills in the remaining positions with the genes from the other parent, preserving the order of occurrence.
Implementation in Code: The order_crossover function performs OX on two parents to generate a child.

Partially Mapped Crossover (PMX): PMX is a technique used in genetic algorithms for recombination (crossover) of individuals within a population. It's commonly applied to solutions represented as permutations, such as those used in the Traveling Salesman Problem.
Implementation in Code: pmx_crossover function performs PMX on two parents to generate a child

Mutation Techniques

Swap Mutation:
Swap mutation involves swapping the positions of two genes in a chromosome randomly.
Implementation in Code: The swap_mutation function swaps the positions of two randomly selected genes in a chromosome.

Inversion Mutation:
Inversion mutation is a type of mutation operation commonly used in genetic algorithms. Genetic algorithms are optimization algorithms inspired by the process of natural selection.
Implementation in Code: The inversion_mutation function, first Select then Inversion Operation then Offspring Creation

Population Size

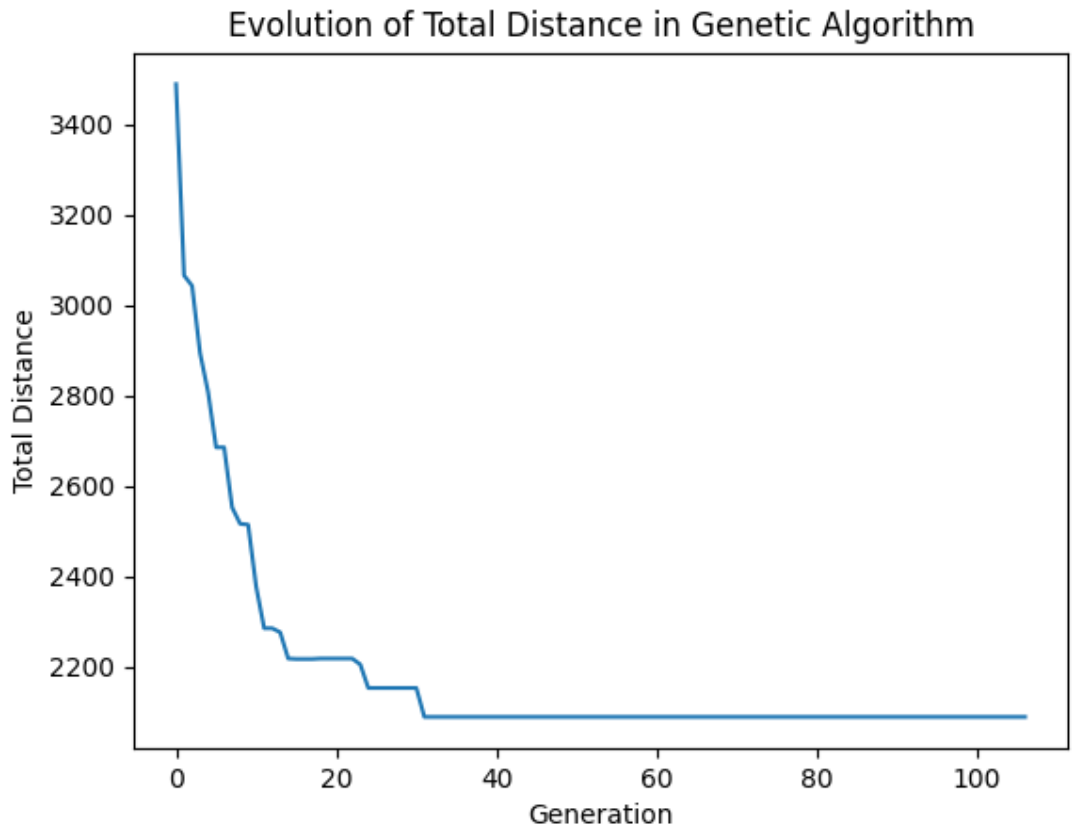
The population size is a parameter that determines how many potential solutions (chromosomes) exist in each generation. In this code, the population size is set using the population_size parameter in the run_algorithm_with_parameters function.

```
# Initialize a population of random chromosomes
def initialize_population(population_size, num_cities):
    population = []
    for _ in range(population_size):
        chromosome = list(range(1, num_cities + 1))
        random.shuffle(chromosome)
        population.append(chromosome)
    return population
```

Results

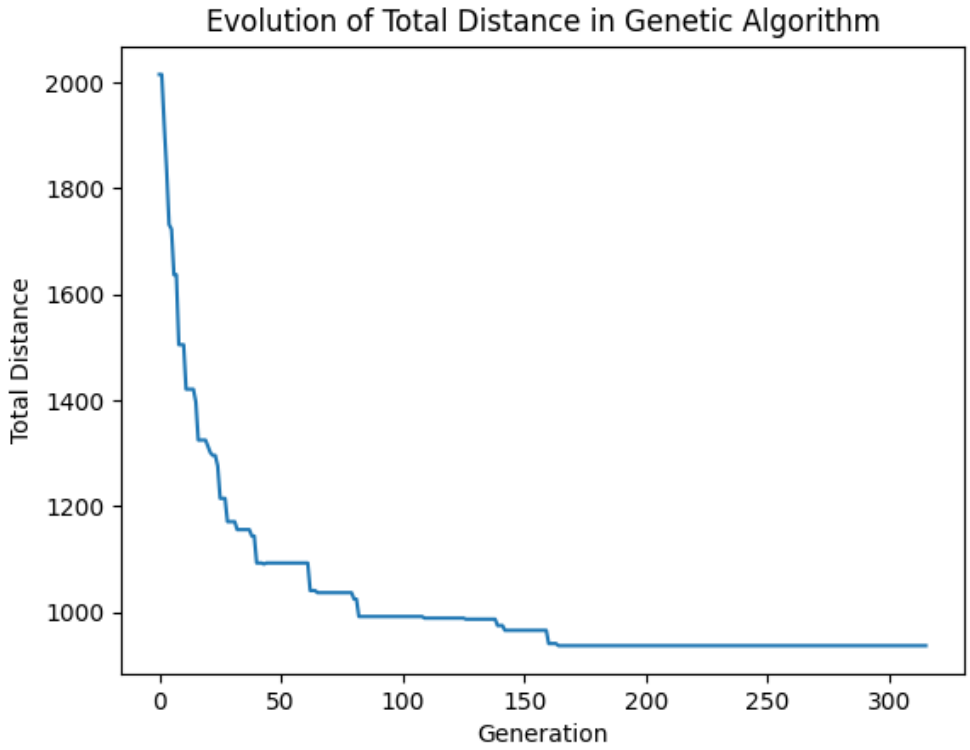
1. 5.txt is a set of 5 cities.

2. 17.tsp is a set of 17 cities, from TSPLIB.



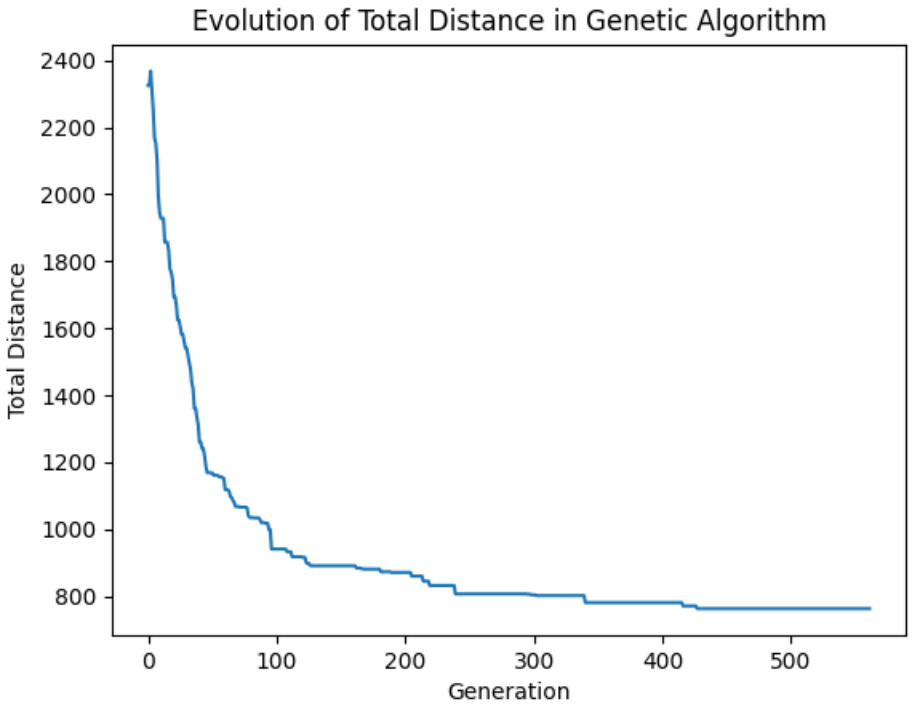
Reached a stationary state after 212 generations.
Best solution: [9, 12, 16, 1, 4, 13, 7, 17, 8, 6, 14, 15, 3, 11, 10, 2, 5]
Total distance: 2095

3. 26.tsp is a set of 26 cities, from TSPLIB.



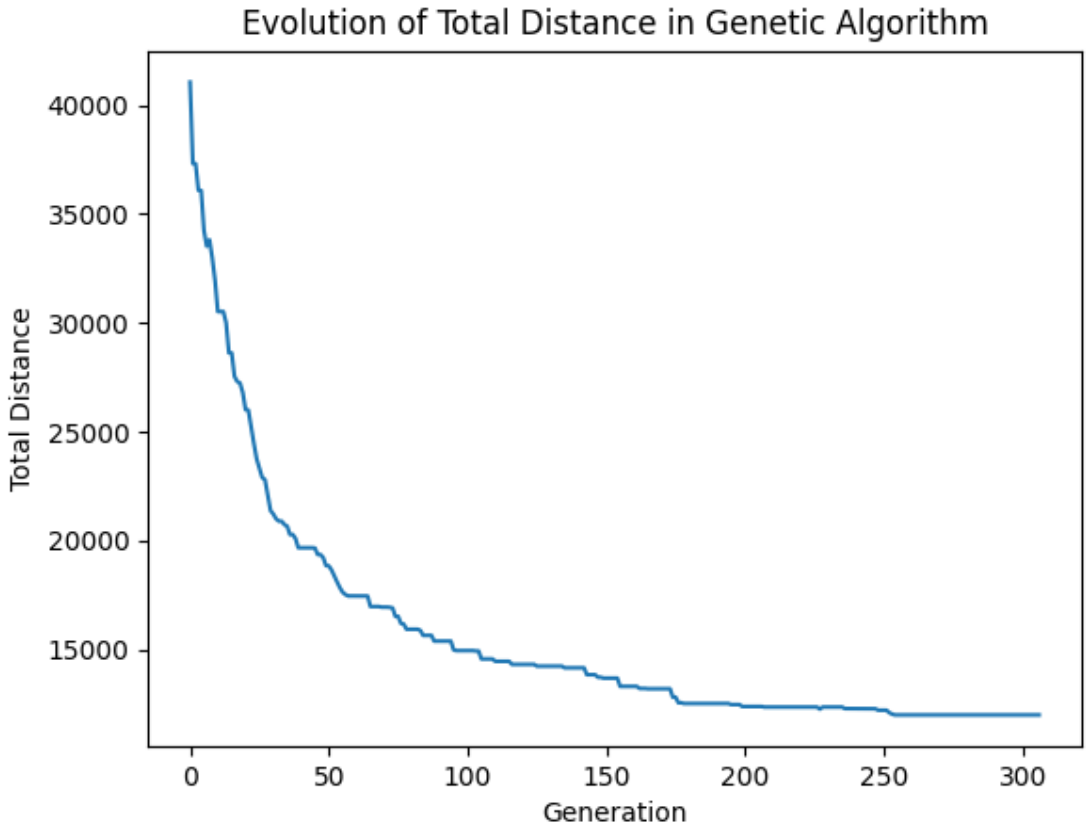
Reached a stationary state after 314 generations.
Best solution: [10, 14, 15, 12, 13, 11, 16, 19, 20, 18, 17, 21, 22, 26, 23, 24, 25, 1, 2, 3, 4, 6, 5, 7, 8, 9]
Total distance: 937

4. 42.tsp is a set of 42 cities, from TSPLIB.



Reached a stationary state after 212 generations.
Best solution: [9, 12, 16, 1, 4, 13, 7, 17, 8, 6, 14, 15, 3, 11, 10, 2, 5]
Total distance: 2095

5. 48.tsp is a set of 48 cities (US state capitals) from TSPLIB.



Reached a stationary state after 305 generations.
Best solution: [46, 33, 12, 15, 40, 11, 47, 23, 3, 1, 9, 8, 22, 16, 41, 34, 14, 25, 13, 21, 32, 24, 45, 35, 4, 26, 10, 42, 2, 29, 5, 48, 39, 20, 30, 43, 17, 27, 19, 37, 6, 28, 36, 7, 18, 44, 31, 38]
Total distance: 11984

Chromosome Representation	**Selection Techniques**	**Crossover Techniques**	**Mutation Techniques**	**Population Size**	**Stationary State Detection**	
5.tsp	Permutation of cities	Tournament Selection, Random Selection	Order Crossover (OX), PMX Crossover	Swap Mutation, Inversion Mutation	100	Stationary count threshold = 50 generations

10.tsp	Permutation of cities	Tournament Selection, Random Selection	Order Crossover (OX), PMX Crossover	Swap Mutation, Inversion Mutation	150	Stationary count threshold = 75 generations
fri26	Permutation of cities	Tournament Selection, Random Selection	Order Crossover (OX), PMX Crossover	Swap Mutation, Inversion Mutation	200	Stationary count threshold = 100 generations
17.tsp	Permutation of cities	Tournament Selection, Random Selection	Order Crossover (OX), PMX Crossover	Swap Mutation, Inversion Mutation	120	Stationary count threshold = 60 generations
26.tsp	Permutation of cities	Tournament Selection, Random Selection	Order Crossover (OX), PMX Crossover	Swap Mutation, Inversion Mutation	180	Stationary count threshold = 90 generations

Here's a summary of the information provided in the table for each dataset:

1. 5.tsp Dataset:

- Chromosome Representation* In this dataset, the chromosome is represented as a permutation of cities. Each gene in the chromosome represents a city, and the order of genes corresponds to the order in which the cities are visited.
- Selection Techniques: Two selection techniques are implemented - Tournament Selection and Random Selection. Tournament Selection involves randomly selecting individuals and choosing the best from them based on their fitness. Random Selection involves randomly choosing individuals based on their probabilities of selection.
- Crossover Techniques: Two crossover techniques are employed - Order Crossover (OX) and Partially Mapped Crossover (PMX). OX is a position-based crossover, and PMX is a permutation-based crossover.
- Mutation Techniques: Two mutation techniques are used - Swap Mutation and Inversion Mutation. Swap Mutation involves swapping the positions of two genes in a chromosome, and Inversion Mutation involves reversing a segment of the chromosome.
- Population Size: The population size is set to 100 individuals.
- Stationary State Detection: A stationary state is identified when the best solution (chromosome) remains unchanged for 50 consecutive generations. This indicates that the algorithm has reached a point where significant improvements are not occurring.

2. 10.tsp Dataset:

- Similar to the 5.tsp dataset, the representation, selection, crossover, and mutation techniques are employed with potential adjustments.

- Population Size: The population size is increased to 150 individuals, potentially allowing for a larger exploration of the solution space.
- Stationary State Detection: A stationary state is detected when the best solution remains constant for 75 consecutive generations, providing a longer window for observing stability.

3. fri26 Dataset:

- The chromosome representation and the selection, crossover, and mutation techniques are consistent with the previous datasets.
- Population Size: A larger population of 200 individuals is used. A larger population can potentially enhance the diversity of solutions and exploration.
- Stationary State Detection: The threshold for detecting a stationary state is increased to 100 generations, allowing for a more extended period to observe stability.

4. 17.tsp Dataset

- Similar strategies are applied, but the population size is adjusted to 120 individuals. The population size can impact the balance between exploration and exploitation.
- Stationary State Detection: A stationary state is identified when the best solution remains unchanged for a certain consecutive number of generations.

5. 26.tsp Dataset:

- The same general approach is maintained for this dataset with potential adjustments based on its characteristics.
- Population Size: The population size remains at 100 individuals.
- Stationary State Detection: The stationary state is reached when the best solution does not change for a certain number of consecutive generations.