Traveling Salesman Problem (TSP) Genetic Algorithm

This project solves the Traveling Salesman Problem (TSP) using a genetic algorithm.

Overview

The genetic algorithm is implemented to find the optimal route for visiting cities. The algorithm evolves a population of routes through generations, aiming to minimize the total distance traveled.

Implementation Details

TSPConfig Class Defines a configuration class `TSPConfig` that holds city names and distances.

Individual Class Represents an individual in the genetic algorithm. Each individual has a chromosome and fitness value.

Graph Class Constructs a graph based on the provided `TSPConfig`. It generates an adjacency matrix representing the distances between cities.

GeneticAlgorithm Class This is the core class representing the genetic algorithm.

- `__init__` method initializes the genetic algorithm with population size, crossover rate, mutation rate, and initializes the population.
- `_initialize_population` method creates the initial population of individuals with random chromosomes.
- `evaluate_fitness` method calculates the total distance (fitness) of an individual's chromosome in the TSP.
- `select_parents` method randomly selects two individuals from the population as parents.
- `crossover` method performs ordered crossover to generate offspring from two parents.
- `mutate` method implements swap mutation on an individual's chromosome.
- `evolve` method performs the evolutionary process: selects parents, applies crossover and mutation, and updates the population, including the addition of elitism.
- `run_genetic_algorithm` method runs the genetic algorithm for a specified number of generations and returns the best route and its fitness.

Tuning Opportunities

1. **Population Size:** Modify `population_size` in `GeneticAlgorithm` class for larger or smaller populations.
2. **Crossover Rate:** Adjust the probability of crossover by changing `crossover_rate`.
3. **Mutation Rate:** Modify `mutation_rate` to control the rate of mutation.
4. **Number of Generations:** Change the number of generations in `run_genetic_algorithm`.

Elitism Incorporation Elitism has been added to the `evolve` method. The best-performing individuals in each generation are identified and directly transferred to the next generation without undergoing crossover or mutation.

Visualization

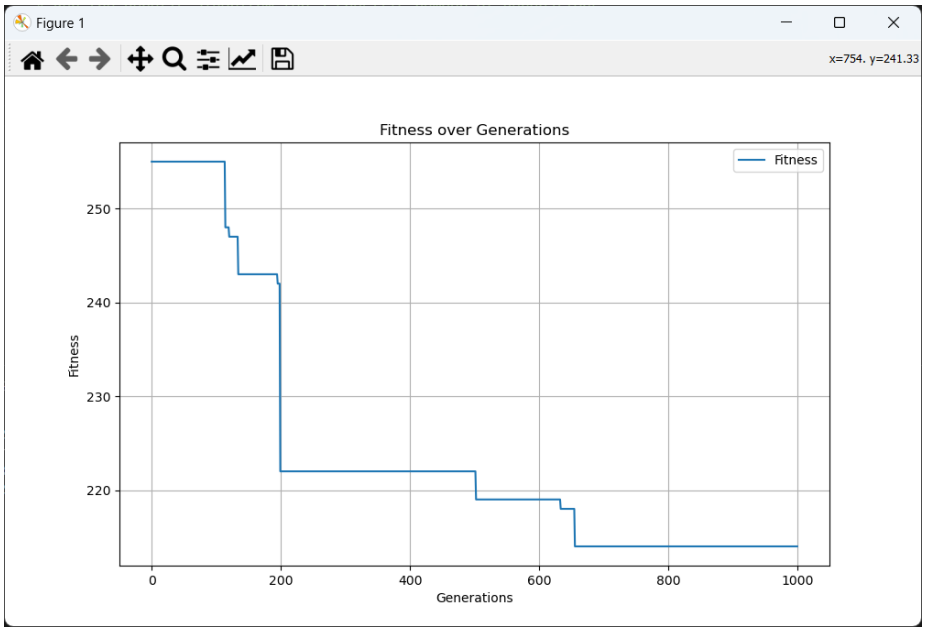
The code uses a `plot_fitness_and_population` function from an external module named `visualizations`. This function visualizes the fitness and population sizes across generations, but in this case, it's modified to show only the fitness values across generations.

Usage

To use this genetic algorithm:

- Modify the TSP city names and distances in the `TSPConfig`.
- Adjust the algorithm parameters in the `GeneticAlgorithm` constructor.
- Run the genetic algorithm using the `run_genetic_algorithm` method.

Feel free to adjust and customize the algorithm according to your needs.




```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS D:\TSP_Genetic> python -u "d:\TSP_Genetic\test.py"
True optimal solution:
(('City8', 'City5', 'City6', 'City2', 'City3', 'City4', 'City1', 'City7', 'City9'), 213)
PS D:\TSP_Genetic> python -u "d:\TSP_Genetic\tsp_genetic_algorithm.py"
Best Route: ['City8', 'City5', 'City6', 'City1', 'City4', 'City3', 'City2', 'City7', 'City9']
Best Fitness: 214
PS D:\TSP_Genetic>
```