Assignment: Genetic Algorithm Project Report

Author: Cecilia Y. Sui

Course: Artificial Intelligence

Language used: Python

Files submitted: Graph.py GA.py test1.txt test2.txt test3.txt

Genetic Algorithm Steps:

Step 1: Create an initial population of P chromosomes (generation 0)

```
def createPopulation(self, nodes):
```

The algorithm creates the initial population based on the graph consisted of nodes from the inputted file. The createPopulation function uses permutation to avoid duplicates and returns a list of P (population size — sizeP) chromosomes.

Step 2: Evaluate the fitness of each chromosome.

```
def evaluateFitness(self, graph, population):
```

This function calculates the cost (i.e. distance) for each chromosome and the fitness value for each chromosome, which is the cost of the chromosome over the total cost of all chromosomes in the current population. The fitness values are then flattened using the "wheel of fortune" concept, resulting in a list of values in the range [0,1] which are used later as the proportional probabilities. The evaluateFitness function returns two lists: cost, containing the costs for all chromosomes; and fit, containing the fitness values.

Step 3: Choose N parents from the current population via proportional selection (i.e. selection probability is proportional to the fitness function)

```
def chooseParent(self, graph, population):
```

The function randomly chooses N (selection size — sizeN) parents from the current population via proportional selection based on the fitness values, and returns the best one (minimum cost / distance) among those N parents.

Step 4: Randomly select two parents to create offspring using crossover operator.

```
def crossover(self, parent1, parent2):
```

The function takes two parents and randomly generates a breakpoint where the crossover operation is performed. Duplicates are avoided. It returns the offspring as a list of nodes.

Step 5: Apply mutation operators for minor changes in the results.

```
def mutation(self, chromosome):
```

The mutation operation is performed based on the mutation rate chosen by the user. The mutation is conducted via swapping two nodes at randomly generated indices. Mutation is not performed on already preserved "best" ones.

Step 6: Repeat Steps 4 and 5 until all parents are selected and mated.
Step 7: Replace old population of chromosomes with new one.

The algorithm keeps the best ones based on a chosen rate, and replaces the rest of the old population with newly generated ones.

Step 8: Evaluate the fitness of each chromosome in the new population.
Step 9: Terminate if the number of generations meets some upper bound; otherwise go to Step 3.

The algorithm terminates either when the number of chosen "generations" are reached, or it has converged to a local minimum. Either way, it prints out the final path of the fittest route along with the cost for that path.

Algorithm Analysis:

Test Case No.1: test1.txt
Number of cities: 5
Minimal cost: 19
Configuration: GA(generations=20, sizeP=10, mutationR=0.1, bestOnes=0.1, sizeN=5)
Results: always converged to a local minimum within the first 20 generations.

Test Case No.2: test2.txt

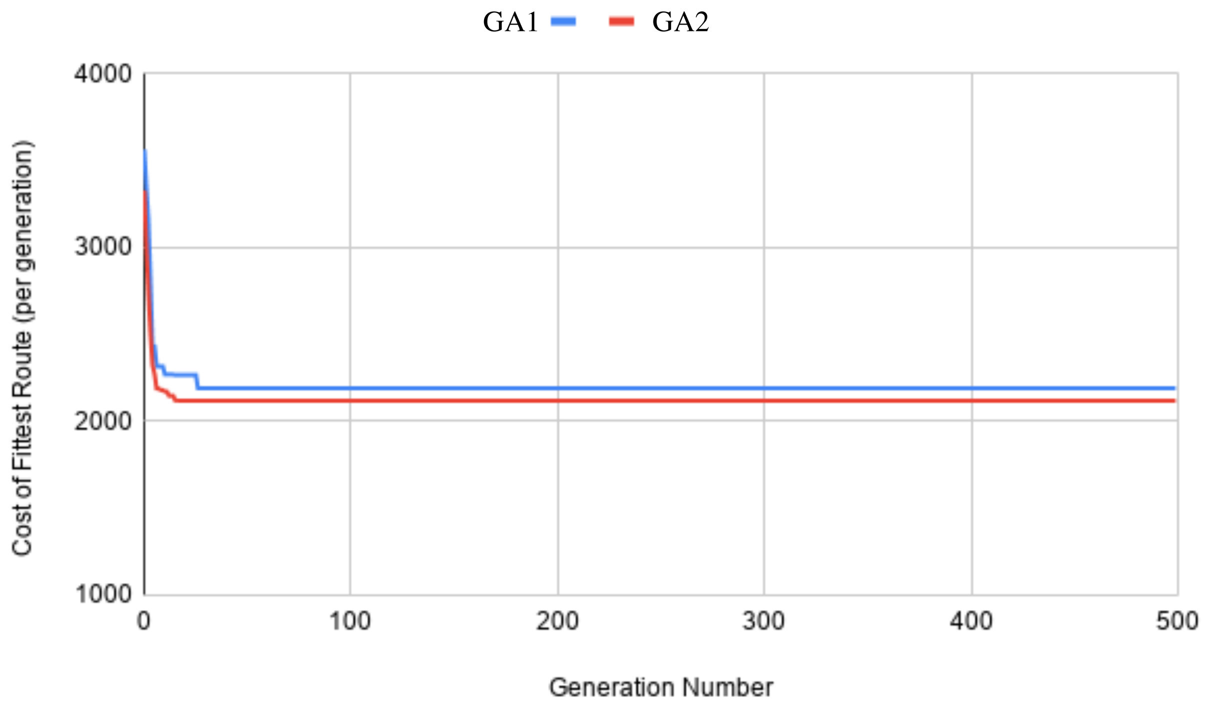
Number of cities: 17

Minimal cost: 2085

Configuration Examples:

GA1 (generations=500, sizeP=300, mutationR=0.1, bestOnes=0.2, sizeN=100)

GA2 (generations=1000, sizeP=500, mutationR=0.4, bestOnes=0.4, sizeN=200)



GA1 never converged to the minimal cost 2085, but reached 2192 on the 26th generation, where it stopped making any progress. Below is the final result returned.

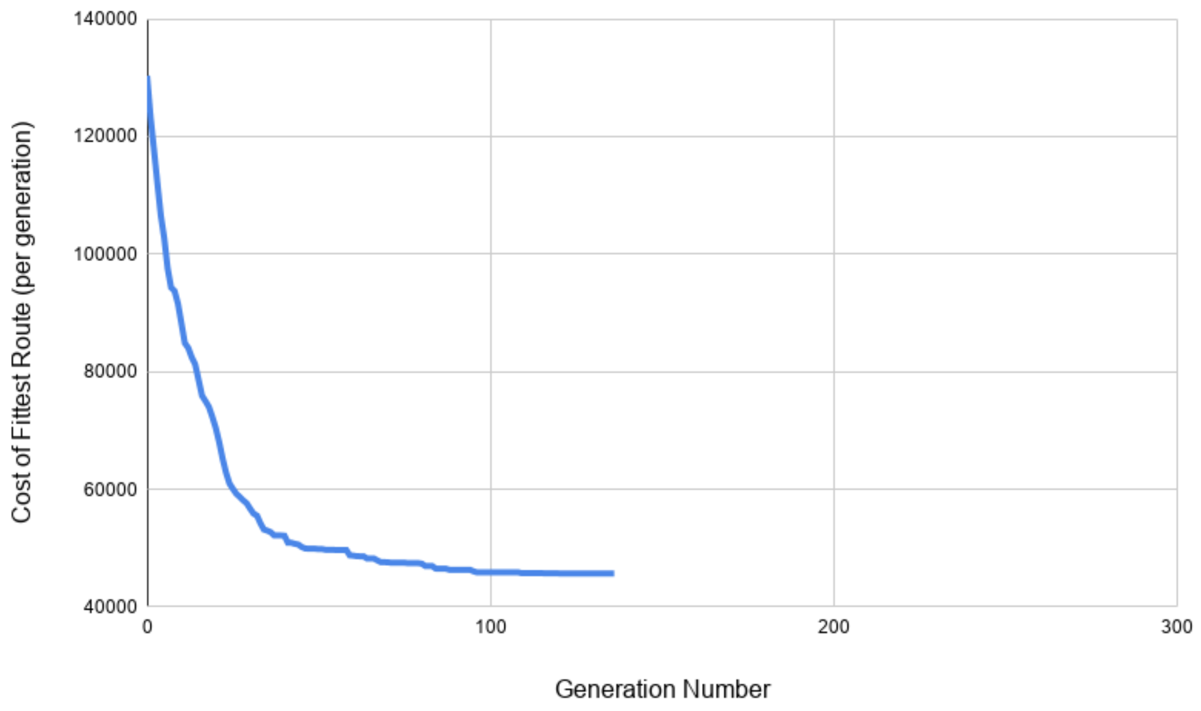
GA2 never converged to the minimal cost 2085 either, but reached 2120 on the 18th generation, where it stopped making any progress.

Test Case No.3: test3.txt

Number of cities: 48

Minimal cost: 33523

Configuration: GA(generations=1000, sizeP=500, mutationR=0.1, bestOnes=0.2, sizeN=200)



The result after running for an hour is shown above. Although the algorithm has not converged to the minimal cost yet, it is slowly approaching the correct value.