INF3490 Mandatory Assignment 1

Travelling Salesman Problem

Name: Haiyue Chen

Username: haiyuec

Instructions on how to run my programs:

All my programs are written for python 3.31 and above.

- Exhaustive search:
  python exhaustive_search.py [number_of_cities]

  Provide the number of cities you would want the algorithm to compute as the command line argument, starting from the beginning of the european_cities.csv file.

- Hill climbing:
  python hill_climber.py [number_of_cities] [number_of_attempts]

  Provide the number of cities you would want the algorithm to compute as the first command line argument, then the number of hill climbing attempts that you would like the program to compute as the second command line argument.

- Genetic algorithm:
  python genetic_algorithm.py [number_of_cities] [population_size]

  Provide the number of cities you would want the program to compute as the first command line argument, then the population size.

PS:

The results of the programs are not showing the last part of the tour, e.g. the route from A to B to C and back to A is: [A,B,C]. The last part from C to A is not shown but the distance is calculated with the last part of the tour.

Questions and implementation explanation

Exhaustive search:

I used the itertools python library to generate all the possible permutations of cities in the for of lists, and used another function to calculate the distance of each route.

1. **What is the shortest tour among the first 10 cities?**

   The shortest tour among the first 10 cities is:

   Copenhagen -> Hamburg -> Brussels -> Dublin -> Barcelona -> Belgrad -> Istanbul -> Bucharest -> Budapest -> Berlin -> Copenhagen

   The tour distance is 7486.31km.

2. **How long would you expect it to take with all 24 cities?**

   It took about 209 seconds for the program to compute for the first 10 cities. There are 362880 possible route with in 10 cities and 6.204484 * 10^{23} route within 24 cities. I estimate the running time for 24 cities with exhaustive search would be about 1.2 * 10^{13} years.

Hill climber:

I used the random module in python to shuffle the cities as the initial route for every attempt with hill climber. My hill climber algorithm would check neighbors of the current route, and if a better route was found, do the algorithm again on the new route. The neighbors are generated by changing the position of two randomly chosen cities in a route. The stopping condition when the algorithm has check 100 neighbors and no better route was found. I have considered to make this parameter as a command line argument, but I found rather running the algorithm multiple times was a better and more "hill climber ish" way of using the algorithm.

I have implemented the statistics in the result of my program and I will simply refer to that.
Running the program with the command:

```
python hill_climber.py 10 20
```

will run the hill climbing algorithm 20 times on the first 10 cities, each run with randomly generated starting route.

One of the runs with 20 attempts gave the following result:
Shortest tour:

Bucharest -> Istanbul -> Belgrade -> Budapest -> Brussels -> Barcelona -> Dublin -> Hamburg -> Copenhagen -> Berlin -> Bucharest

Distance: 8243.97 km

Longest tour:

Berlin -> Barcelona -> Dublin -> Brussels -> Hamburg -> Belgrade -> Istanbul -> Budapest -> Bucharest -> Copenhagen

Distance: 9899.37 km

Average distance: 9018.451 km
Standard deviation: 457.464 km
Running time: 0.0643 seconds

One of the runs with 24 cities gave the following result:
Shortest tour:

Kiev -> Bucharest -> Milan -> Madrid -> Paris -> Barcelona -> Dublin -> London -> Hamburg -> Moscow -> Stockholm -> Saint Petersburg -> Berlin-> Brussels -> Budapest -> Vienna -> Belgrade -> Munich -> Copenhagen -> Prague -> Rome -> Sofia -> Istanbul -> Warsaw -> Kiev

Distance: 21944.54km

Longest tour:

Prague -> Barcelona -> Berlin -> Brussels -> Saint Petersburg -> Moscow -> Sofia -> London -> Hamburg -> Madrid -> Munich -> Rome -> Milan -> Dublin -> Paris -> Istanbul -> Belgrade -> Budapest -> Kiev -> Copenhagen -> Stockholm -> Bucharest -> Warsaw -> Vienna -> Prague

Distance: 26802.33km

Average distance: 25155.598 km
Standard deviation: 996.758 km
Running time 0.1766 seconds

Genetic algorithm:

I created each of the individuals as a TSPsolver object and had a evolve-method to evolve the population. The parents are given a reproduction chance based on their fitness and the total fitness in the parent-pool. The parents are chosen at random with the reproduction chance, and children are created by using the PMX algorithm and mutated after the mapping. There is an equal chance of the mutation being swap mutation, insert mutation, scramble mutation and inversion mutation. This process creates two children by calling the PMX function with parent1 and then calling the PMX function with parent2. This process happens [population_size] / 2 times every generation, and thus doubling the population. The program would terminate after running 50 evolutions without improvement.

I chose the 10, 50 and 100 as values for population size. Since I defined the algorithm to terminal after 50 generations without improvement, the algorithm would not terminate on the same generation count every time. I will use the average generation count as the generation number for each of the variants. I will also use the generation when the algorithm terminated, not the generation where it found the best result.

Population 10:

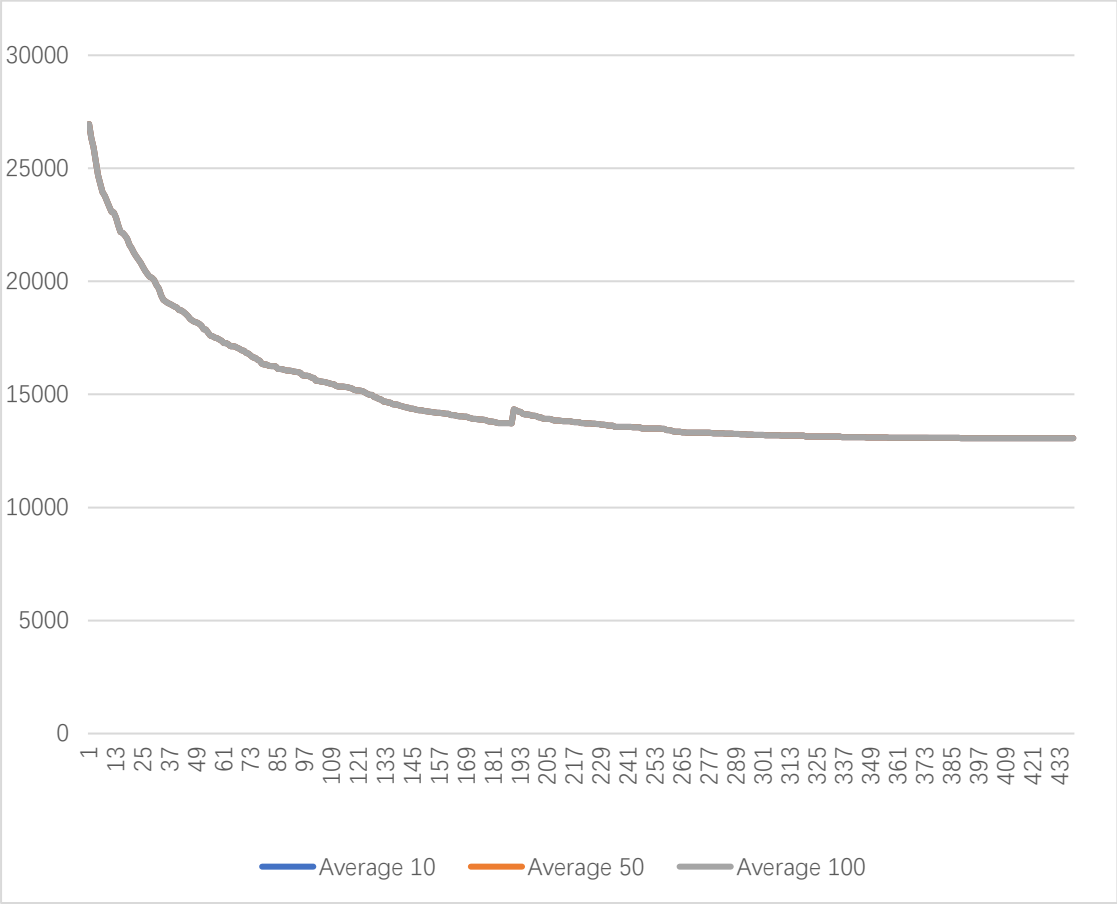
Average tour length: 13096.884 km
Max tour length: 14348.390 km
Min tour length: 12287.000 km
Standard deviation: 591.630 km

Population 50:

Average tour length: 12536.877 km
Max tour length: 13244.960 km
Min tour length: 12287.070 km
Standard deviation: 292.953 km

Population 100:

Average tour length: 12450.435 km
Max tour length: 13195.320 km
Min tour length: 12287.070 km
Standard deviation: 224.107 km



There are three graphs in the picture above, but because the results are so similar, they overlap in the picture. I have included the excel file with the data as well in the delivery if there is a need for it. With this result I concluded the population number does not affect the quality of my genetic algorithm. This means in this case the optimal population size for the traveling salesman problem would be 10, as the program would produce a result must quicker. It took 94 seconds with population size 100, 34 seconds with population size 50 and under 1 second with population size 10 on my laptop computer.

My algorithm manages to find the best tour among the first 10 cities, and it does it very quickly compared to the exhaustive approach.
The genetic algorithm used 0.07 seconds and the exhaustive search algorithm used 209 seconds. The genetic algorithm used 0.23 seconds to find a potential best tour among all the 24 cities, and the exhaustive search algorithm would in theory find the absolute best tour among the 24 cities, but it is safe too say that would take too much time.

I ran the genetic algorithm on 24 cities with population size of 10. The particular run ran for 266 generations, assuming the algorithm does not generate duplicate children, this means about 2600 to 2700 tours were evaluated. The exhaustive search would had to evaluate $6.204484 * 10^{23}$ tours.