The Burman Traveling Salesman problem

HEIG-VD

Machine Learning (MLG)

Lab 8

13.06.2022

Authors

Berney Alec

Ferrari Teo

6.1. Briefly explain the problem and your solution (1/2 page).

We must solve the travelling salesman problem (TSP). We start with a list of cities and their respective distances. The goal is to find the shortest path that makes us pass through each city only once and brings us back to the starting point. More precisely, in our case, we are given 14 pairs of coordinates.

We have to calculate ourselves the distances between the cities while considering the curvature of the earth. For this we use a python library that uses the Vincenty formula.

We produce a matrix containing all distances between a city “I” and another city “y”.

Our solution is based on a genetic algorithm where we try to minimize the distance of the path between all cities. To minimize this path, we use a fitness function that evaluates the performance based on the path length.

6.2 Provide the better route you found and the shortest path in kilometres. Is it the optimal shortest path? explain.

We found a lot of different paths, but they all have the same shortest path that is 3346.761973 km.

All the paths found are the same but with a different start or in the other way.

Here is a list of different paths found:

[12, 7, 10, 8, 9, 0, 1, 13, 2, 3, 4, 5, 11, 6]

[13, 2, 3, 4, 5, 11, 6, 12, 7, 10, 8, 9, 0, 1]

[12, 7, 10, 8, 9, 0, 1, 13, 2, 3, 4, 5, 11, 6]

[2, 3, 4, 5, 11, 6, 12, 7, 10, 8, 9, 0, 1, 13]

[2, 3, 4, 5, 11, 6, 12, 7, 10, 8, 9, 0, 1, 13]

[8, 10, 7, 12, 6, 11, 5, 4, 3, 2, 13, 1, 0, 9]

We can't know if it is the shortest path because we can't explore every road to check if they are the right ones. The only thing we can assume is that it is a good path result that minimize a lot the travel length.

6.3 Describe your fitness function

Here is our function used to calculate the fitness:

```python
def fitness(genome):
    global distances
    score = 0.0

    for i in range(len(genome)):
        j = (i + 1) % len(genome)
        score += distances[genome[i], genome[j]]
    return score
```

Other functions used in this code can be found in the notebook.

Firstly, we have a function that compute the distance with a library called vincenty using the vincenty formula.

After that, also created a function creating a matrix with all distances between cities.

In the final fitness function, for the path given, we sum all the distances between each city of the path. The score is equal to the total distance. The distances are obtained with the matrix of distances created earlier.

As it is a circular path, the last city is also the first one, when we compute the last distance between two cities, we use a modulo operation to get the first one again.

6.4 Explain the way you encoded the solution, give a chromosome example.

First, we used a "G1DList" to create our chromosome. This list contains the index of the city from the longitude and latitude table given in the lab information.

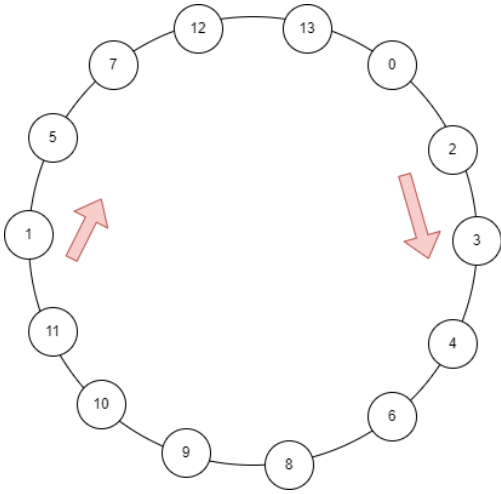
Of course, the first city is at index 0 and the last one at index 13.

Finally, the chromosome is a list containing the cities visited in the order of the path found.

An important point is that the city at the index 13 of the list from the chromosome is connected to the first city at index 0.

Here is an example of a possible path encoded in a chromosome:

[1, 5, 7, 12, 13, 0, 2, 3, 4, 6, 8, 9, 10, 11]



1	5	7	12	13	0	2	3	4	6	8	9	10	11

So, it gives a path like the picture on the right.

To know the distance between two cities, we remember you that the distance_matrix stocked and had already compute all the distances between cities and his used in our algorithm.

6.5 Provide the configuration of the GA you finally used to find your better results: mutation, crossover, population size, type of selection, mutation, crossover used, number of generations. Describe the methodology or experiments performed in order to get your better results.

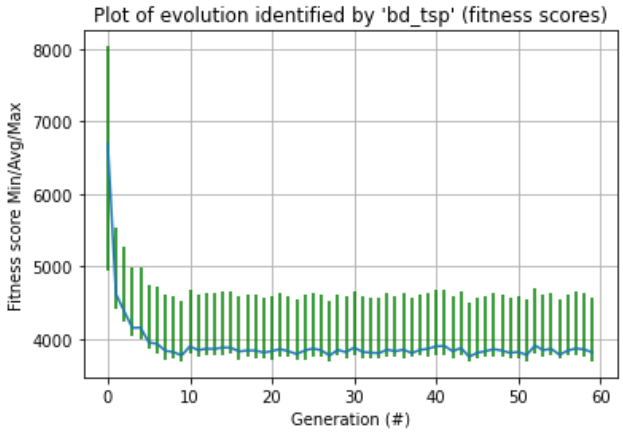
Here our final parameters:

- Mutation rate: 0.03
- Crossover rate: 0.9
- Population size: 300
- Type of selection: Tournament selection (Selectors.GTournamentSelector)
- Type of mutation: Swap (Mutators.G1DListMutatorSwap)
- Type of crossover: SinglePoint (G1DListCrossoverSinglePoint)
- Number of generations: 60

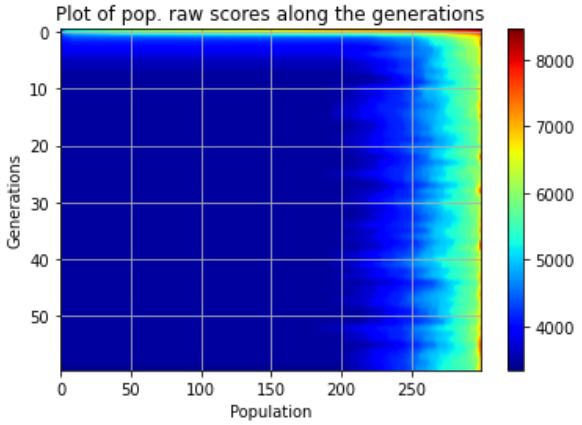
To start, we've set population size, number of generations at high values such as 1'000. The goal was to focus our choice on crossover rate and mutation rate. After we found good parameters for these values, we tried to decrease the values of population size and number of generations until the results began to be worst.

"Tournament Selection" has been chosen because "Rank Selection" has problem with limiting the evolution of chromosome. We didn't choose "Roulette Wheel Selection", because it's already used in "Tournament Selection", so it makes a good compromise between these solutions.

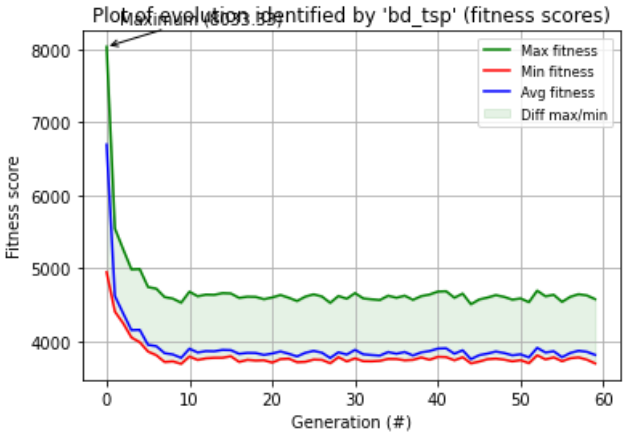
6.6 Provide relevant plots of your experiments and explanations.



The algorithm quickly finds a good result, but we need more generations to have more chance that he finds the best results. These generations are the 10 to 60.



This graph shows the same as the first one, but we it is a different way to represent it.



The most important there is that our average is close to the minimum fitness. It indicates that we have good parameters for the algorithm.

We can also notice that the difference between min and max fitness is small.

6.7 Conclusions (1/2 page) + [optional] eventual supplementary comments (1/2 page)

Finally, while we don't know the final solution, we feel like we got a good approximation of the best result without having to compute too many iterations. We discussed the results with our colleagues, and we all seem to have arrived at approximately the same result.

We had to do a lot of testing to define the best parameters for the training.

Genetic algorithms seem like a good way to solve NP complete problems or problems with an unknown heuristic.