INF3490 Oblig nr. 1

Emir Zamwa
UiO-brukernavn: emirz

10. september 2018

Content

Pictures

1 Travelling Salesman Problem

1.1 Instructions on running the program

To run the program, make sure all python packages are installed on computer. This includes csv, sys, time, itertools, random, statistics and matplotlib.pyplot. Once this is done, go into the code and find the main() method. This is on the top. Here you can change how many runs you want, how many cities, also population and generations. Generations is the criteria for ending the GA loop. You can also here choose which algorithms to run. You should see comments guiding you. After this, to run, write python3 $INF3490\_Oblig1.py$ in the terminal.

1.2 Answers to questions from assignment

1.2.1 Exhaustive Search

The shortest tour among the first 10 cities is $7486,31$ km. Barcelona -> Belgrade -> Istanbul -> Bucharest -> Budapest -> Berlin -> Copenhagen -> Hamburg -> Brussels -> Dublin -> back to Barcelona. The program spent around 21 seconds to find it. This is also shown in the terminal once program is run. Amount of times tours were inspected: 3628800
To run the algorithm with all 24 cities; out of my speed of the program, it would approximately take around 108 billion years to finish!

1.2.2 Hill Climbing

The answer for the question given is written in terminal once the program is run. The results under is copied from terminal:

```
——— HILL CLIMBING 10 CITIES ———
Best tour length: 7486.31 km
Permutation for best run: [7, 3, 8, 6, 2, 5, 4, 9, 1, 0]
Worst tour length: 8391.05 km
Permutation for worst run: [3, 8, 2, 4, 9, 1, 5, 6, 7, 0]
Mean for all runs: 7615.683 km
Standard deviation for all runs: 272.955
Time: 39.502 ms

——— HILL CLIMBING 24 CITIES ———
Best tour length: 12416.51 km
Permutation for best run: [15, 13, 18, 0, 12, 7, 11, 16, 3, 8, 6, 21, 19, 14, 10, 4, 9, 20, 1, 5, 22,
17, 23, 2]
Worst tour length: 16484.51 km
Permutation for worst run: [0, 18, 20, 9, 4, 1, 3, 11, 16, 13, 15, 19, 14, 10, 5, 22, 23, 21, 6, 2, 17,
8, 7, 12]
Mean for all runs: 14292.604 km
Standard deviation for all runs: 1039.238
Time: 1320.885 ms
```

1.2.3 Genetic Algorithm

For the genetic algorithm, I have chosen population sizes of 50, 100 and 300 for the test. Also, my genetic algorithms "stop" call is based on generations. You choose how many generations the algorithm will run before it is done. For the sake of this test, I have chosen 300 generations. This can easily be changed in the code if needed.
Again, the results are copied from the terminal (3 runs):

```
——— GENETIC ALGORITHM 24 CITIES ———
Running 20 times...
Best tour length: 15499.63 km
Permutation for best run: [13, 15, 3, 11, 7, 12, 0, 18, 16, 8, 2, 10, 9, 4, 20, 1, 22, 5, 23, 14, 19, 21, 6, 17]
Worst tour length: 19067.66 km
Permutation for worst run: [9, 20, 5, 15, 2, 17, 3, 8, 23, 4, 18, 0, 12, 16, 7, 11, 13, 6, 21, 14, 19, 1, 22, 10]
Mean for all runs: 17557.633 km
Standard deviation for all runs: 979.428
Time: 7359.065 ms
```

```
——— GENETIC ALGORITHM 24 CITIES ———
Running 20 times...
Best tour length: 14312.47 km
Permutation for best run: [9, 20, 4, 1, 18, 0, 12, 7, 11, 16, 8, 2, 23, 10, 14, 19, 21, 6, 3, 13, 15, 22, 17, 5]
Worst tour length: 18886.71 km
Permutation for worst run: [12, 7, 3, 8, 2, 16, 11, 15, 13, 18, 5, 1, 20, 10, 21, 6, 23, 17, 9, 4, 14, 19, 22, 0]
Mean for all runs: 16090.267 km
Standard deviation for all runs: 1200.641
Time: 15823.388 ms
```

```
——— GENETIC ALGORITHM 24 CITIES ———
Running 20 times...
Best tour length: 13170.21 km
Permutation for best run: [0, 13, 15, 17, 22, 5, 18, 1, 20, 9, 4, 23, 10, 14, 19, 21, 6, 2, 8, 7, 11, 3, 16, 12]
Worst tour length: 16278.7 km
Permutation for worst run: [4, 9, 20, 22, 17, 15, 13, 16, 12, 0, 3, 7, 11, 8, 2, 5, 1, 18, 6, 21, 19, 23, 14, 10]
Mean for all runs: 14938.67 km
Standard deviation for all runs: 990.608
Time: 46807.856 ms
```

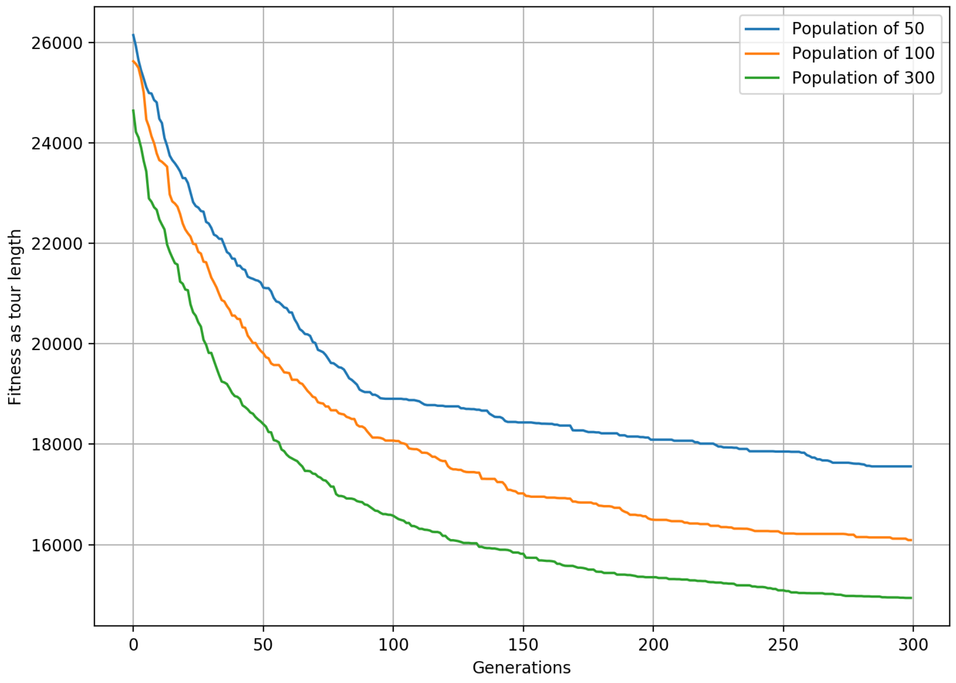

Figure 1: Plot of the evolution of the 3 runs (24 cities)

Out of the plot, we can see that the higher the population is, the lower the tour length becomes. This concludes that higher population is better.
My genetic algorithm found the shortest tour like the exhaustive search. Depending on the population and generations; as seen in the results above, we can see that the GA is much quicker to find the results than the exhaustive search. Both with 10 cities and 20 cities.
The exhaustive search with 10 cities inspected tours 3628800 times! With the GA, it totally depends on population. With the amounts tested above, it gave:
Amount of times tours were inspected: 73020 (50 population)
Amount of times tours were inspected: 152000 (100 population)
Amount of times tours were inspected: 456000 (300 population)

1.3 Briefly what I have done

1.3.1 Exhaustive Search

With the ES, I have made a list with every possible permutation of the cities. Once this list is made, the search will then go thorugh every permutation and calculate the tour length. In the end, it picks the best one and prints it out.

1.3.2 Hill Climber

With the HC, I have two methods. One to start the climb, and the other as the climb. Here, everytime the algorithm is run, it starts by making a random permutation of the cities. After this, it makes neighbour permutations and making sure it doesn't have duplicate permutations. This list is then returned and checked for lowest tour length. The algorithm then, out of all the runs, saves the best ones and i the end picks the best one.

1.3.3 Genetic Algorithm

The genetic algorithm starts by making a population of random permutations. This population, say it is 100, is then sent for choosing parents. I have made the program such that the amount of parents always is population/2. Cases where the parent amount isn't even numbers, an extra parent is made. The parent selection is random. So in this case, out of 100, 50 parents are made. These parents are then sent to get in pairs. The pairs that are made, are 'indexily' selected from the parent list. Then, I have made so the program also makes one child for every parentpairs. In this case, from 50 parents, you get 25 children. The children made is done by ordered crossover. Now, this list is sent to mutate. I have set the mutate rate to be 30%, so there is 30% chance for every kid to get mutated. If mutated, two genes are swapped. Now, after all this, the children are sent back and put in the new population list. Remember we had 100 in the first place? Now it is 125. This list is then sorted with the individuals fitness, where highest tour length is further back in the list. We end up having a list from best to worst of 125 individuals. After the sort, the list cuts away the last 25 individuals. And then, this new population list, now containing 100 individuals again, is back in the loop to go through all the steps above again. I make sure the population is at 100 all the time. The loop goes for the amount of generations set by the user. After all this, the best one is selected for each run like the hill climber, and best one is picked.