in3050 Assignment1

Cornelia D. Vassbotn, icvassbo

vår 2021

Introduction

How to run all programs in terminal:
python exhaustive_search.py
python hill_climbing.py
python gentic.py

Exhaustive Search

For the exhaustive search, I made an alghorithm that inspected every possible tour amoung the first 6 cities. See Listing 1 for output from program.

The shortest tour among the first 10 cities is 'Copenhagen', 'Hamburg', 'Brussels', 'Dublin', 'Barcelona', 'Belgrade', 'Istanbul', 'Bucharest', 'Budapest', 'Berlin', whith the distance 7486.309.

Listing 1: Terminal output

```
Nbr of cities:  6
Time:  0.005877017974853516 seconds

Nbr of cities:  7
Time:  0.037139892578125 seconds

Nbr of cities:  8
Time:  0.25395894050598145 seconds

Nbr of cities:  9
Time:  2.282039165496826 seconds

Nbr of cities:  10
Time:  25.658502101898193 seconds
```

With 10 cities, there are $10! = 3.628.800$ combinations, and the program needs to find all tours. This explains why the program took 25 seconds, which is quite long. With 24 cities, there are $24! = 6.204484017 * 10^{23}$ combinations of different tours. To get an approximately calculation of how long it will take to run the program with 24 cities, we can assume that it will take $\frac{24!}{10!} = 1.7097895 * 10^{17}$ times higher than for the case with 10 cities.

Hill Climbing

The hill climbing algorithm finds the shortest tour amoung all cities. The algorithm works much faster than the Exhaustive Search, and is capable to find the shortest tour with all 24 cities, within a reasonable time period. With 10 cities the program takes 0.0021109580 seconds. Compared to the Exhaustive Search it works 25.2003839015/0.0021109580993652344 = 11938 times faster with 10 cities. See Listing 2 for output from program.

Listing 2: Terminal output

```
Nbr of runs:  20
Nbr of cities:  10
Mean:  7581.4985
Std:  202.29858289852174

Best tour:  ['Copenhagen', 'Hamburg', 'Brussels', 'Dublin', 'Barcelona',
Best length: 7486.309999999999

Worst tour:  ['Budapest', 'Belgrade', 'Istanbul', 'Bucharest', 'Copenhag
Worst length: 8349.94
Time of last run 0.003149747848510742
_______________
Nbr of runs:  20
Nbr of cities:  24
Mean:  14166.501500000002
Std:  735.2285171786726

Best tour:  ['Istanbul', 'Bucharest', 'Berlin', 'Copenhagen', 'Hamburg',
Best length: 12606.32

Worst tour:  ['Istanbul', 'Bucharest', 'Budapest', 'Berlin', 'Copenhagen
Worst length: 15363.519999999999
Time of last run 0.08754801750183105
```

Genetic Algorithm

For the genetic algorithm I chose to use Partially Mapped Crossover to make new children. To make mutations I chose to use Swap permutation, with the assumption that by swapping to random cities, there is a chance to get a better solution. The chance of getting a mutation is 10%. In other words, of all children 10% will be mutated.

When we see the figure over average fit individual, with 10 cities, we can conclude that there is a benefit of having a population of 100, rather than 50. However we can also conclude that it does not matter if we use a population of size 150, or 100. By the figure we see that after approximately 35 generations, there are not any difference between the population of size 100 and 150.

By the graph over average fit individual, with 24 cities, we see a greater difference in fitness. Here the population of size 150, is continually better than the population of size 100.

Among the 10 cities, the GA found the shortest tour, but now in every run with a population of 50 individuals. We can see this in Figure 1, as the average is higher than of the exhaustive search. However, the algorithm average after 80 generation did come close.

The running time to the GA, with population size= 50, and number of cities = 10, it used 0.677132 seconds. Compared to the Exhaustive Search, the GA is $\frac{25.658}{0.677132} = 37.8$ times faster.

The running time to the GA, with population size= 50, and number of cities = 24, it used 1.352599 seconds. Compared to the Exhaustive Search, the GA is capable to find the best route with a reasonable time.

At most the GA inspected at 250 individuals two times each generation, which give us $150 * 80 * 2 = 24.000$ inspections. This is a much lower number than with the exhaustive search.

Listing 3: Terminal output

```
population_size:  50
nbr_cities:  10
nbr_generations:  80
Mean:  7584.193249999999
Std:  117.80310514981136
Time of last run 0.6771321296691895
Best length:  7486.309999999999
Worst length: 7915.15


population_size:  100
nbr_cities:  10
nbr_generations:  80
Mean:  7617.328899999999
Std:  270.46075335210844
Time of last run 1.3765947818756104
Best length:  7486.3099999999995
Worst length: 8391.050000000001


population_size:  150
```

```
nbr_cities:  10
nbr_generations:  80
Mean:  7548.2935
Std:  204.40312833405955
Time of last run 1.9811899662017822
Best length:  7486.3099999999995
Worst length: 8407.18


population_size:  50
nbr_cities:  24
nbr_generations:  80
Mean:  18315.54205
Std:  985.7702655042383
Time of last run 1.3525991439819336
Best length:  16409.3
Worst length: 22617.250000000004


population_size:  100
nbr_cities:  24
nbr_generations:  80
Mean:  17073.9953
Std:  1219.0368606231355
Time of last run 2.8233590126037598
Best length:  15479.470000000001
Worst length: 21462.899999999998


population_size:  150
nbr_cities:  24
nbr_generations:  80
Mean:  16561.56035
Std:  1036.6037364308388
Time of last run 4.5352277755737305
Best length:  14760.519999999999
Worst length: 19384.440000000002
```

Graph

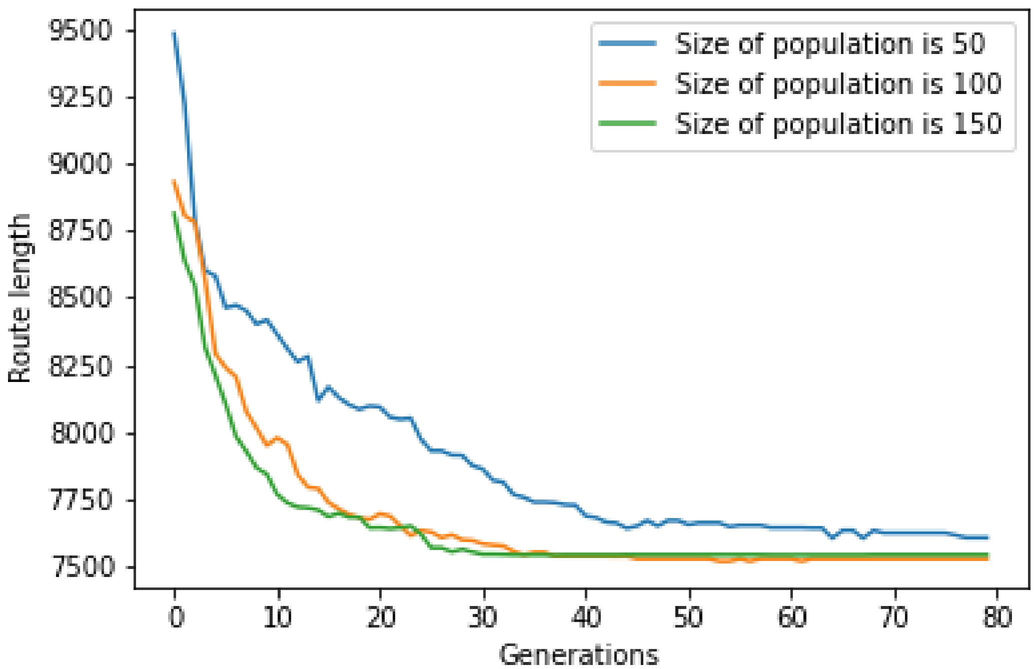

Figure 1: Average fitness of the best fit individual in each generation, with 10 cities.

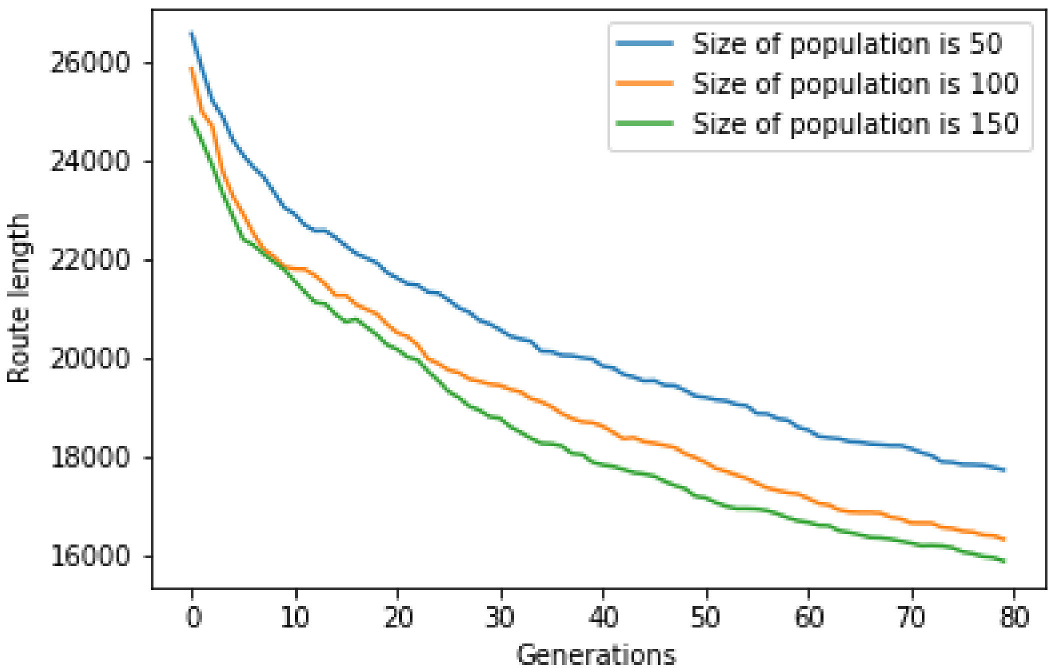

Figure 2: Average fitness of the best fit individual in each generation, with 24 cities.