INF3490 Oblig 1

joakikt

September 2018

1 The program

Made with python 3
Requirements:
numpy
matplotlib

Running:

```
usage: main.py [-h] [-n NUMBER_OF_CITIES] [-m MUTATION_RATE]
               [-g GENERATIONS] [-p POPULATION_COUNT]
               [-r RUN_COUNT]
               {exhaustive,hillclimb,genetic,report}

positional arguments:
  {exhaustive,hillclimb,genetic,report}

optional arguments:
  -h, --help            show this help message and exit
  -n NUMBER_OF_CITIES, --number-of-cities NUMBER_OF_CITIES
  -m MUTATION_RATE, --mutation-rate MUTATION_RATE
  -g GENERATIONS, --generations GENERATIONS
  -p POPULATION_COUNT, --population-count POPULATION_COUNT
  -r RUN_COUNT, --run-count RUN_COUNT
```

2 Exhaustive search

This was created by simply creating all permutations of cities and looping over them and comparing the distance while keeping the shortest path

- What is the shortest tour among the first 10 cities?
  - $Copenhagen \rightarrow Hamburg \rightarrow Brussels \rightarrow Dublin \rightarrow Barcelona \rightarrow Belgrade \rightarrow Istanbul \rightarrow Bucharest \rightarrow Budapest \rightarrow Berlin$
  - Distance: 7486.31
- How long did your program take to find it?
  - 40.320 seconds
- How long would you expect it to take with all 24 cities??
  - Based on the timings of the runs i did it takes ca. 10 times as long for each city added
    So for 10 it takes 40 and if my assumptions are correct it will take around 400s for 11 cities. So $40 \times 10^{14} = 4000000000000000s =$ 126839167.93 years

3 Hill climbing

The hillclimber is made by generating a random permutation of cities, then using itertools combination function i create all possible swaps of that permutation, i then loop over them and swap the cities in the permutation and compare them. I keep the best neighbour and continue until none of the neighbours are better than the current best permutation.

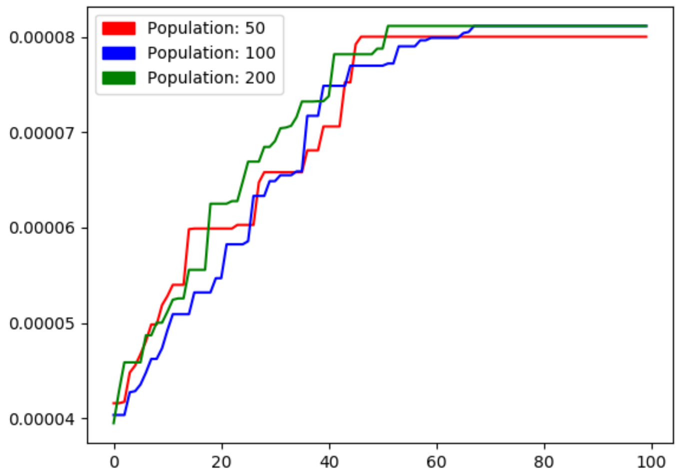
- How well does the hill climber perform?
  - On 10 cities it uses 0.2s each run
- Reports:
  - 10 cities
    - Best: 7486.31
    - Worst: 7737.95
    - Mean: 7513.15
    - σ: 75.09
  - 24 cities
    - Best: 12633.05
    - Worst: 16367.21
    - Mean: 14498.27
    - σ: 914.70

4 Genetic algorithm

The genetic algorithm starts by creating *population_count* random permutations. It then uses order crossover to generate offspring from parents, then mutates the offspring using swap mutation where it swaps two alleles at random. The mutation-rate option decides the chance for offspring to mutate. Then it selects the population of the next generation using fitness-based replacement and $(\mu + \lambda)$-selection.

- Reports:
- Variable population over 100 generations with a mutation rate of 10 %
  - Population 50:
    - Best: 12325.93
    - Worst: 14517.74
    - Mean: 13153.75
    - σ: 615.35
  - Population 100:
    - Best: 12441.27
    - Worst: 13788.32
    - Mean: 13056.48
    - σ: 337.76
  - Population 200:
    - Best: 12340.50
    - Worst: 13743.56
    - Mean: 12909.55
    - σ: 360.72
- Among the first 10 cities, did your GA find the shortest tour?
  - Yes
- How did the running time of your GA compare to that of the exhaustive search?
  - 10 cities: GA: 2.289s Exhaustive: 40.320
  - 24 cities: GA: 3.381s Exhaustive: Long
- How many tours were inspected by your GA as compared to by the exhaustive search?
  - GA: $(population \times 2) \times generations$ This case: 20000
    Exhaustive: 24!

Plot of fitness of the best fit individual in each generation:



Judging by this the population 50 converges faster, but on a lower fitness. Population 100 takes more generations to converge. Population 200 converges faster than 100 and on a higher fitness than 50. So in this case population 200 is the best.