Name: Eirik Studsrød Olsen
username: eirisols

My program is divided into 4 different python files, one with the exhastive search, one with hill climbing, one with the genetic algorithm and one with helper functions that are used a lot called generell_functions.py.

You run the exhaustive search with - python3 exhaustive.py
You run the hill climber with - python3 hill_climbing.py
You run the genetic algorithm with - python3 genetic.py

Exhaustive Search

The shortest tour among 10 cities is (6, 8, 3, 7, 0, 1, 9, 4, 5, 2). I have not implemented a method to print the tour with the actual city names. This tour has a length of 7486.309999999999. My program took 33.73357796669006 seconds to find the shortest tour. The time it takes to search the cities will increase exponentially with each city, so the time it will take to search all 24 cities could be years.

Hill Climbing

The hill climber managed to find the shortest tour between 10 cities at 7486.3099999999995 The hill climber is much faster than the exhaustive search. The hill clumber used 0.9072470664978027 seconds to run 20 times and give the results.

Hill Climbing:10 cities.
Best result: 7486.3099999999995
Worst result: 8349.94
Average result: 7621.763
Standard deviation: 208.1971257726678

Hill climbing: 24 cities.
Best result: 12340.499999999998
Worst result: 15703.449999999997
Average result: 14402.9675
Standard deviation: 846.7234009240376

Genetic Algorithm

My genetic algorithm uses a roulette wheel selection to select parents. It works by assigning a fitness to all the permutations and then checking if the fitness is higher than a global fitness rate, if it is it is selected. The amount of parents made is the population/2. The parents are then used to create children using an order crossover function. The children are then mutated based on a mutationrate using a simple swap mutation. The children are then added to the population and the worst individuals in the population are removed to get the population back to regular size.

I have chosen population sizes of 60, 100 and 200.
The algorithm runs 400 generations with a fitnessrate of 0.01 and a mutationrate of 0.1

Population 60:
Best result: 12396.32
Worst result: 15795.379999999997
Average result: 13456.5245
Standard deviation: 757.7935274235322

Population 100:
Best result: 12325.930000000002
Worst result: 14339.859999999999
Average result: 13020.9125
Standard deviation: 534.5660155960751

Population 200:
Best result: 12334.349999999997
Worst result: 13610.890000000003
Average result: 12925.6795
Standard deviation: 366.7567658827707

You can see that the best results are roughly equal across 60, 100, and 200. But the worst and average results are better with a higher population and the standard deviation is also better. But the speed also goes up with a higher poulation. At 200 it takes 235.26225805282593 seconds to run 20 instances and with 100 population it takes 110.10261607170105 seconds. That still looks like a linear relationship so it’s not that bad. I did not manage to make the plots on my mac.

GA for just 10 cities:
Best result: 7486.309999999999
Worst result: 7503.1
Average result: 7487.989
Standard deviation: 5.167852959453237

You can see from this that the GA found the best tour on nearly all runs.
The running time of the GA is much better than the exhaustive search when you move above 10 cities.