Traveling Salesman Problem

To tackle this problem, I decided to use mixture of Genetic Algorithm and 2-opt algorithm.

Main parts of Genetic Algorithm:

1. Initial Population
2. Fitness Function
3. Selection of parents
4. Crossover to generate new offsprings
5. Mutation of offsprings
6. Final formation of next generation

1. InitialPopulation

Initial population is created randomly by shuffling the list of cities. First the list of cities from 1 to n is created. Afterwards, random.shuffle is used to generate random instances. The size of initial population can be controlled by changing the variable called **initial_population**.

2. FitnessFunction

Fitness function simply calculates the total distance of the given path. Arguments that are required for fitness function are path which contains list of cities in a desired order and coordinates of each cities. Coordinates of the cities is a dictionary with a name **coor**.

3. SelectionofParents

Selection of parents is performed by tournament selection. K number of random individuals are selected from the population and we pick M best of them to be added to the list of parents. For our case, I decided the total number of parents to be 5*M. So, we use tournament selection 5 times to fill the mating pool. K and M parameters can be changed by changing the values of the variables **selection_k** and **m_best** respectively.

4. Crossovertogeneratenewoffsprings

First, I tried to use single-point crossover to generate offspring. Because each city should be visited precisely once, I got the portion of the first parent and iteratively added cities from the second parent. If the city already exists in offspring, I skipped it. That way paths from both parents are present in a new offspring to certain degree. However, I didn't get very good results from this tactic. Therefore, I decided to use two-point crossover instead. I implemented it essentially in the same way. Two points are randomly generated and using those points portion of the first parent is assigned to the offspring. The rest of the path is filled up with zeros first and then replaced by cities from the second parent.

5. Mutationofoffsprings

Mutation is kept minimal. Only order of two randomly selected cities are replaced. When I

mutated more than two cities, it seriously affected the performance of the algorithm. Therefore, I stick to the minimum.

6. **Finalformationofnextgeneration**

As a result of crossover and mutation new offsprings are generated with the size of twice the number of parents. All these offsprings are preserved for the next generation. Moreover, I used elitism to include top best performing parents in the next generation. The number of elite parents can be manipulated by changing the variable called elite_number. **The size of each new generation is equal to 2*(5*m_best)+ elite_number**.

Global value **f** keeps track of how many times fitness function was called. Usually, fitness function is called 5*m_best times for each generation. Therefore, if we want to generate 500 generations, f=2500*m_best. After fitness function limit is reached, the algorithms returns the best path found so far among all generations.

2-opt Algorithm

Genetic algorithm by itself didn't perform that bad when used with small number of cities. As the number of cities increases, performance went down. To increase the performance even with bigger number of cities, I decided to use hybrid of GA and 2-opt algorithm.

Short definition of 2-opt algorithm: 2-opt is a simple local search algorithm for solving the traveling salesman problem. A complete 2-opt local search will compare every possible valid combination of the swapping mechanism. (Wikipedia)

2-opt algorithm by itself performs well with small number of cities, but as the number of cities increases, the cost of using 2-opt algorithm grows exponentially. Therefore, I didn't use 2-opt algorithm in its raw format. In GA, after the new offsprings are mutated, with the probability of 10%, 5 of the offsprings goes through 2-opt algorithm. Because 2-opt is very costly, instead of going through all edges, I decided to only go through edges between two points in the path that are generated randomly. These techniques are used to make sure that my method of solving TSP doesn't get too costly. Probability of offsprings going through 2-opt can be changed by changing the value of variable called **alpha** between 0 and 1. The number of offsprings that go through 2-opt can also be manipulated by changing the value of **two_opt_offsprings**.

When I checked for 130 cities, my hybrid of 2-opt and GA worked really well. The result I got after 1000 generations was around 6600 and it took only 18 seconds to run. The length of optimal path for those 130 cities is 6100.