Solving Travelling Salesman Problem with Genetic Algorithm Report

Onur YILMAZ S009604

0. Introduction

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city? In this homework, I ’ll answer this question using a GA to find a solution to the traveling salesman problem (TSP).

1. Algorithm

Before starting to explain methods, below explains the fundamental match between TSP and GA.

Gene: a city represented as x, y coordinates.

Individual: a single route satisfying the conditions above

Population: a collection of possible routes (i.e., collection of individuals)

Parents: two routes that are combined to create a new route

Mating pool: a collection of parents that are used to create our next population (thus creating the next generation of routes)

Fitness: a function that tells us how good each route is (in our case, how short the distance is)

Mutation: a way to introduce variation in our population by randomly swapping two cities in a route

Elitism: a way to carry the best individuals into the next generation

Our GA will proceed in the following steps:

→ Evolve(list of routes)

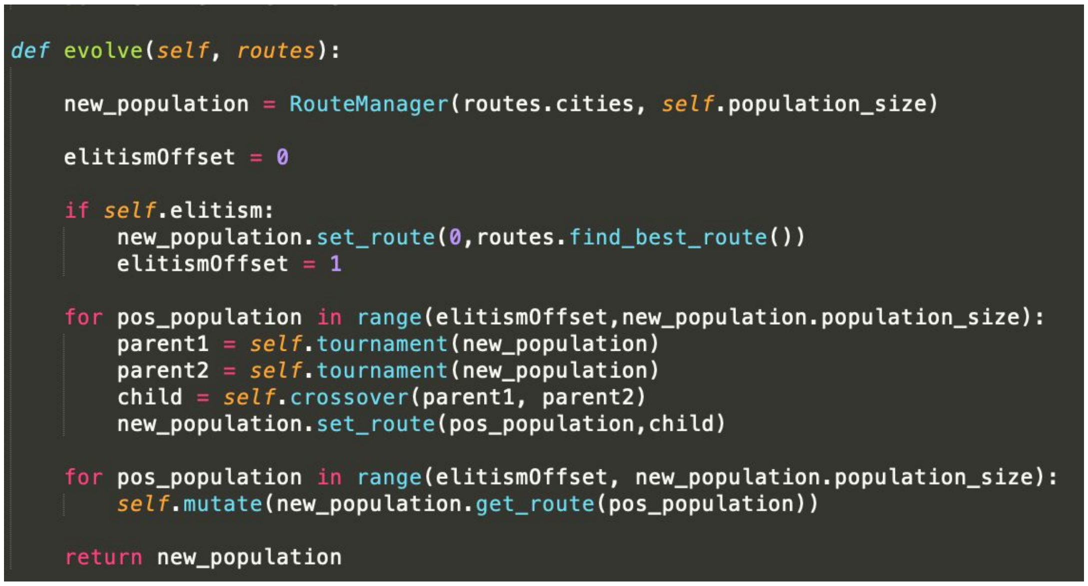
→ Crossover(route1, route2)

→ Mutate(route)

→ Tournament(list of routes) route

3. Implementation Details

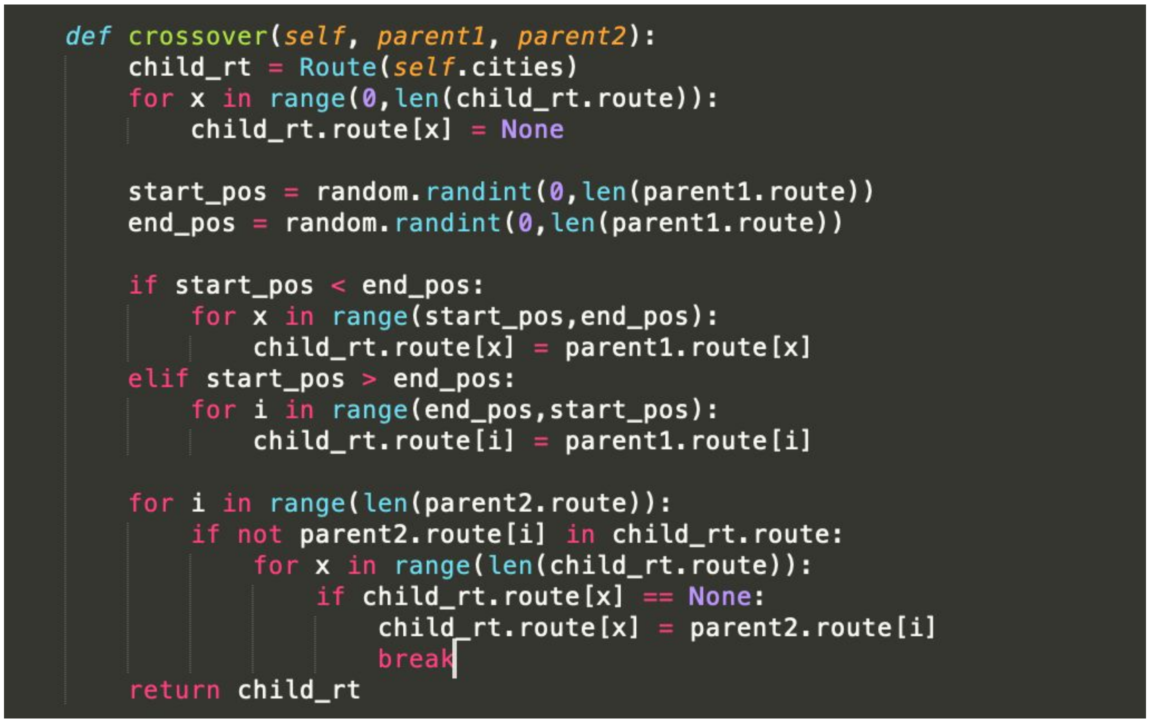
a)Evolve



```
def evolve(self, routes):

    new_population = RouteManager(routes.cities, self.population_size)

    elitismOffset = 0

    if self.elitism:
        new_population.set_route(0,routes.find_best_route())
        elitismOffset = 1

    for pos_population in range(elitismOffset,new_population.population_size):
        parent1 = self.tournament(new_population)
        parent2 = self.tournament(new_population)
        child = self.crossover(parent1, parent2)
        new_population.set_route(pos_population,child)

    for pos_population in range(elitismOffset, new_population.population_size):
        self.mutate(new_population.get_route(pos_population))

    return new_population
```

Our aim is to take a population and evolve it then return the new population. Firstly, we makes a new population, then if we have elitism, set the first of the new population to the fittest of the old.Following, in ‘for loop’ Goes through the new population and fills it with the child of two tournament winners from the previous population.When a child(new route) burn after tournaments, we Fill the population up with children with mutates all the routes.Finally we return new population.

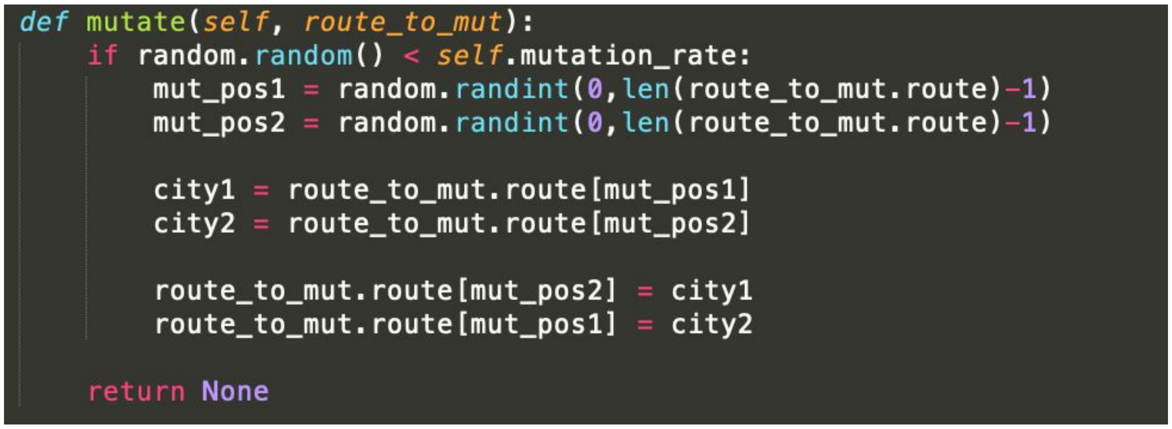
b)Crossover



```
def crossover(self, parent1, parent2):
    child_rt = Route(self.cities)
    for x in range(0,len(child_rt.route)):
        child_rt.route[x] = None

    start_pos = random.randint(0,len(parent1.route))
    end_pos = random.randint(0,len(parent1.route))

    if start_pos < end_pos:
        for x in range(start_pos,end_pos):
            child_rt.route[x] = parent1.route[x]
    elif start_pos > end_pos:
        for i in range(end_pos,start_pos):
            child_rt.route[i] = parent1.route[i]

    for i in range(len(parent2.route)):
        if not parent2.route[i] in child_rt.route:
            for x in range(len(child_rt.route)):
                if child_rt.route[x] == None:
                    child_rt.route[x] = parent2.route[i]
                    break
    return child_rt
```

To put it all in simple terms, we'll create the next generation in a process called crossover with using parent's routes and crossover is done by selecting a random range of parent1 with placing it into the empty child route. Then,two random integer indices of the parent1 are selected to find crossover point randomly.Following, takes the sub-route from parent1 and sticks it in itself looking at the relationship between start pos and end position.Then runs cycles through the parent2 and fills in the child route until all the cities are in the child route. Finally, function will return the child route.

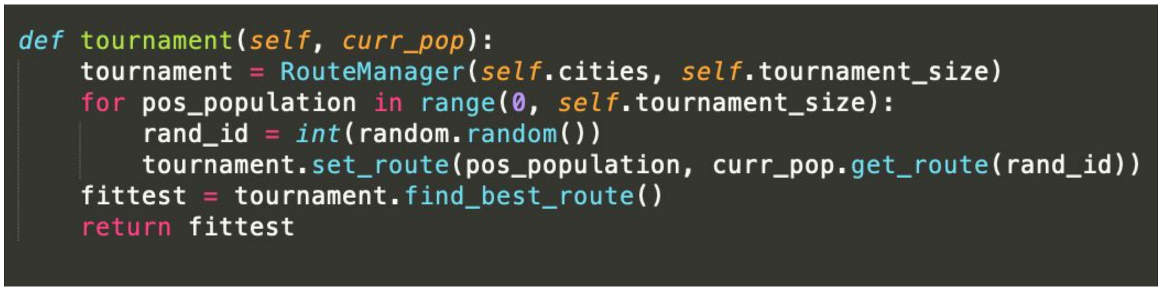
c)Mutate



```python
def mutate(self, route_to_mut):
    if random.random() < self.mutation_rate:
        mut_pos1 = random.randint(0,len(route_to_mut.route)-1)
        mut_pos2 = random.randint(0,len(route_to_mut.route)-1)

        city1 = route_to_mut.route[mut_pos1]
        city2 = route_to_mut.route[mut_pos2]

        route_to_mut.route[mut_pos2] = city1
        route_to_mut.route[mut_pos1] = city2

    return None
```

Mutation helps to allow us to explore other parts of the solution space. We just swap two random indexes in route.

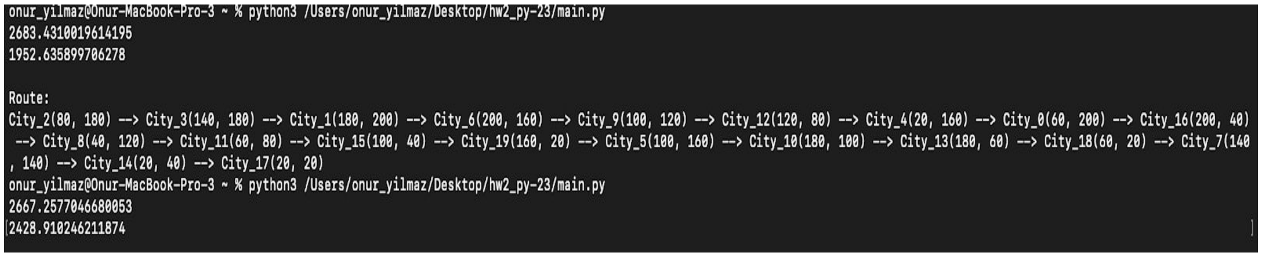
d)Tournament



```python
def tournament(self, curr_pop):
    tournament = RouteManager(self.cities, self.tournament_size)
    for pos_population in range(0, self.tournament_size):
        rand_id = int(random.random())
        tournament.set_route(pos_population, curr_pop.get_route(rand_id))
    fittest = tournament.find_best_route()
    return fittest
```

Tournament function uses a set number of individuals which are randomly selected from the population and it tries to find the fittest route for this individual .

4. Result



```
onur_yilmaz@Onur-MacBook-Pro-3 ~ % python3 /Users/onur_yilmaz/Desktop/hw2_py-23/main.py
2683.4310019614195
1952.635899706278

Route:
City_2(80, 180) --> City_3(140, 180) --> City_1(180, 200) --> City_6(200, 160) --> City_9(100, 120) --> City_12(120, 80) --> City_4(20, 160) --> City_0(60, 200) --> City_16(200, 40)
 --> City_8(40, 120) --> City_11(60, 80) --> City_15(100, 40) --> City_19(160, 20) --> City_5(100, 160) --> City_10(180, 100) --> City_13(180, 60) --> City_18(60, 20) --> City_7(140
, 140) --> City_14(20, 40) --> City_17(20, 20)
onur_yilmaz@Onur-MacBook-Pro-3 ~ % python3 /Users/onur_yilmaz/Desktop/hw2_py-23/main.py
2667.2577046680053
2428.910246211874
```

When we look at the results in general, we observe serious improvements between the results we received before using the genetic algorithms and the results we received after using. As seen in the pictures, we were able to reduce the road distance starting with 2683 to 1952 thanks to this algorithm. Likewise, we observed the results, starting from 2650, in which the distance was reduced to 2500.

5. Conclusion

Here, it can be concluded that. As in our normal life, the stronger/weaker born child is expected because of the stronger/weaker hereditary characteristics of the mother and father. Likewise, the efficiency of newly emerging routes can change entirely depending on the efficiency (genes) of the parent routes. To sum up, a little bit of luck and a little bit of genes combination determine everything.