CS451 Assignment 2 Report

Can Yılankıran S011483

In this assignment, the goal was successfully implementing a genetic algorithm for Travelling Salesman problem which is an NP-Hard problem. The used function in the algorithm as follows: Evolve, crossover, mutate and tournament.

Genetic Algorithm:

Firstly, I want to start with evolve function. In the evolve function the main goal is basically generating a new set of routes according to the cities and population size. After that we will call the tournament function according to the elitism parameter. If elitism is false, we will pass through the whole list of routes. If elitism parameter is chosen to be true, we will not pass the route having the best fitness score. Elitism provides us to keep the best solution in the list after the evolve operation.

Secondly, the selection procedure must be done after evolving. The main idea behind the selection, we try to reduce the possible solutions by choosing the best ones according to the fitness value. There are lots of ways of selection process such as, tournament selection, roulette wheel selection, etc. The tournament selection is used in this assignment. In tournament selection, the routes will be randomly chosen and the route having the best fitness score will be chosen.

Thirdly, lets continue with crossover function. I randomly defined some start and ending points. According to these starts and ending points the cities assigned from two different routes, into a new route. So, we can say that 2 routes coming from the tournament function is crossed in order to generate a new one. Also, I would like to mention if the elitism parameter selected, elite route doesn't passed into the crossover function same as selection and evolve.

Lastly, according to the mutation rate, which is given as a parameter, each route will be changed if the random number gets lower than the mutation rate. So, we can basically say that the portion of routes list is going to be changed almost the same value as the mutation rate. For example, if mutation rate is 0.2, the %20 of the routes list is going to be changed.

Implementation Details:

Let's start with evolve function because all of the other function called in this function. Evolve function takes and returns RouteManager object. Firstly, I generated the new population of routes with the usage of

new_routes = RouteManager(self.cities, self.population_size)

After that, according to elitism parameter the function acts in two different ways. As I mentioned above, if the elitism is selected to be true, the elite route which has the most fitness value, does not passed through other functions. Then, in order the process the tournament selection we build a for loop starting from the first element(Route object) of the routes to the last element of the routes(Route object). (Note: in the case of elitism is false, if elitism is true for loops will not take the first element of routes.)

```
for i in range(0, new_routes.population_size):
        chosen1 = self.tournament(routes)
        chosen2 = self.tournament(routes)
```

After the selection procedure is done, we can call the crossover function.

```
new_routes.set_route(i, self.crossover(chosen1, chosen2))
```

After the crossover function, we need to mutate the current population (Routes). To be able to that with the for loop we send all Route objects in the new_routes to the mutate function. (Note: the elitism parameter acts like same as selection operation, if the elitism parameter is true, we don't send the elite route into the mutation.)

Now let us focus on tournament function. In the tournament function I crated a RouteManager object which is called tournament_routes according to the self.cities and self.tournament_size. Because the functions of find_best_route() and set_route is already given by our assistants, so if I create a RouteManager object and randomly add Route objects to the tournament_routes from the routes list, then select the one having best fitness value, selection operation will be done.

```
tournament_routes = RouteManager(self.cities, self.tournament_size)
count = 0
for i in np.random.randint(0, routes.population_size,size=self.tournament_size):
        tournament_routes.set_route(count, routes.get_route(i))
        count += 1
return tournament_routes.find_best_route()
```

Let us continue with the crossover function. We randomly generate the start end the end position according to the limit of given routes length. Then we add cities to the result from the route1 with the limit of start to end. At last, we add cities from the route2 while checking is the result contains the city or not and checking the result have null city object in result(Route object.) The reason behind the check of containing or not, prevent duplicates. If the result has null city object in it after the assigned cities from route1, route2 will complete it.

```
for i in range(start, end):
    result.assign_city(i, route_1.get_city(i))
```

The above code will add the cities from route1.

```
For i in range(route_2.__len__()):
        if not result.__contains__(route_2.get_city(i)):
                stop = True
                x = 0
                while x < result.__len__() and stop is not False:
                        if not result.get_city(x):
                        result.assign_city(x, route_2.get_city(i))
                        stop = False
        x += 1
```

The above code will add the missing cities on the result route from the route2 while checking the result list contains the new assigned city or not.

Lastly, the mutate function. If the generated random variable comes less than self.mutation_rate, the route will be changed according to a random number.

```
if np.random.sample() < self.mutation_rate:
        random_location = np.random.randint(route.__len__())

        temp_city = route.get_city(i)
        temp_city2 = route.get_city(random_location)

        route.assign_city(random_location, temp_city)
        route.assign_city(i, temp_city2)
```

At the above code I generated the random variables using NumPy. Then changed the routes according to the random_location.

Results:

The default parameters were,

population_size=50, mutation_rate=0.2, tournament_size=10, elitism=False

According to these parameters, the algorithm come up with approx. **1539** route distance(route cost). Let us observe each parameters effect one by one.

Population Size: I will only change the population size the other parameters will remain as the same as default.

- Population Size = 500. After making it 500 there was no big change in the route cost, it was approx. **1500-1700**
- Population Size = 10000. After making it 500 there was no big change in the route cost, it was approx. **1500-1700**
- Now let us lower it. Population Size = 25. After making it 25 there was no big change in the route cost, it was approx. **1500-1700**
- Population Size = 5. After making it 25 there was no big change in the route cost, it was approx. **1500-1700**

Mutation Rate: I will only change the mutation rate the other parameters will remain as the same as default.

- Mutation rate = 0.4. After making it 0.4 there a big change in the route cost, it was approx. **1800-2000**
- Mutation rate = 0.8. After making it 0.4 there a no big change with respect to the 0.4 in the route cost, it remains approx. **1800-2000**
- Now let us lower it. Mutation rate = 0.2. there is a big change with respect to 0.4, it was approx**. 1400-1700**

- Mutation rate = 0.05. This change makes a huge impact on the algorithm. The results were approx. **900-1000**

Tournament Size: I will only change the tournament size the other parameters will remain as the same as default.

- Tournament Size = 50. After making it 50, there is no difference in the route cost but even a human can feel that the algorithm gets slower. I added the time library to see the difference, and the results are as it follows: The elapsed time with the default values is approx. **4.50** seconds, the elapsed time after the change in the tournament size(50) is approx. **13.4** seconds.
- Tournament Size = 100. After making it 100. The cost was lowered to the approx. **1300-1500**. But the elapsed time get into the point of approx. **23.4** seconds.

Elitism: I will only change the elitism parameter; the other parameters will remain as the same as default. After I changed the elitism parameter to the true, the route cost lowered to the approx. **1000-1200.**

Now let us try to make a combination of parameters to get the best result. According to the results we gained, I will try to lower the mutation rate and leave elitism true all the time. With the parameters as it follows.

population_size=20, mutation_rate=0.05, tournament_size=5, elitism=True

I gained the result of approx**. 800-1050** with the 2 seconds of elapsed time.

Conclusion

I have learned the how genetic algorithms acts and the implementation details of genetic algorithms generally. Also, according to the results, mutation rate needs to be low, I believe the reason behind this, how high we adjust the mutation rate, the algorithm starts to act as a random search. So, it should be kept at low levels. Also, elitism parameter should be True all the time. Even without looking at the results it feels right, because keeping the route having the highest fitness value cannot harm our algorithm. tournament size does not affect the path cost too much but it affects the runtime of the algorithm, so it should be kept at optimal levels.