CS451/551 – Introduction to Artificial Intelligence
Assignment-1

HW2

Solving Travelling Salesman Problem with Genetic Algorithm

TOLGA GUMUSCU

S017901

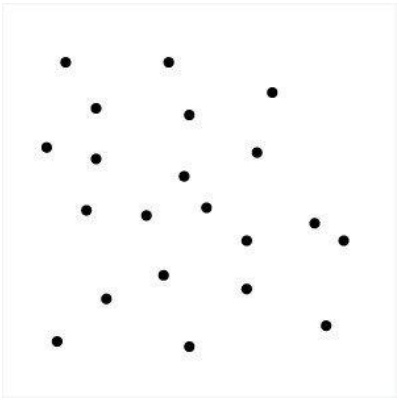
Table of Contents

Introduction

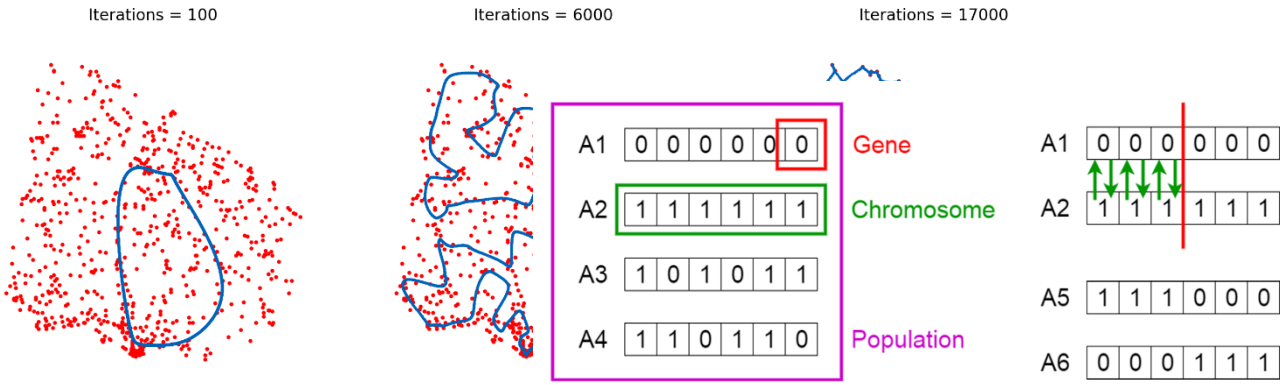
The homework asked from us in CS451's second assignment is about the Traveling Salesman Problem. During this homework, we will try to solve this problem using the Genetic Algorithm. We will fill in the methods of GA from the 5 classes given to us. In this context, there are 4 different methods we need to develop.

Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is the challenge of finding the shortest yet most efficient route for a person to take given a list of specific destinations. It is a well-known algorithmic problem in the fields of computer science and operations research.



There are obviously a lot of different routes to choose from but finding the best one—the one that will require the least distance or cost—is what mathematicians and computer scientists have spent decades trying to solve for.



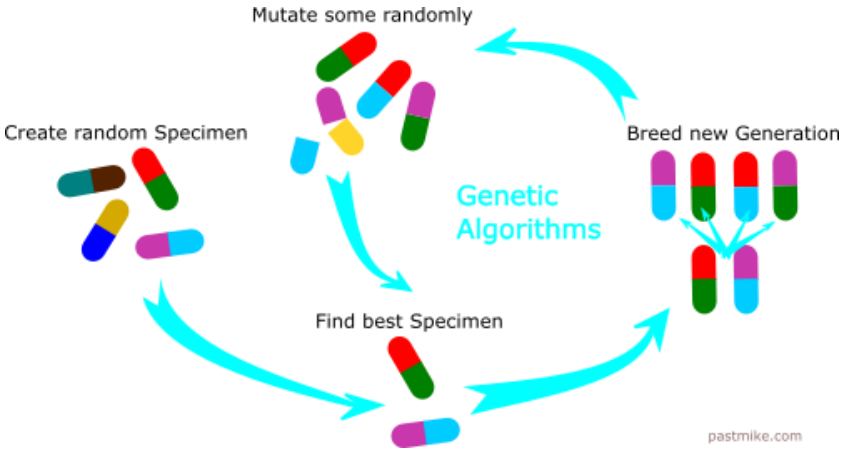
Genetic Algorithm

The process of natural selection starts with the selection of fittest individuals from a population. They produce offspring which inherit the characteristics of the parents and will be added to the next generation. If parents have better fitness, their offspring will be better than parents and have a better chance at surviving. This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

This notion can be applied for a search problem. We consider a set of solutions for a problem and select the set of best ones out of them.

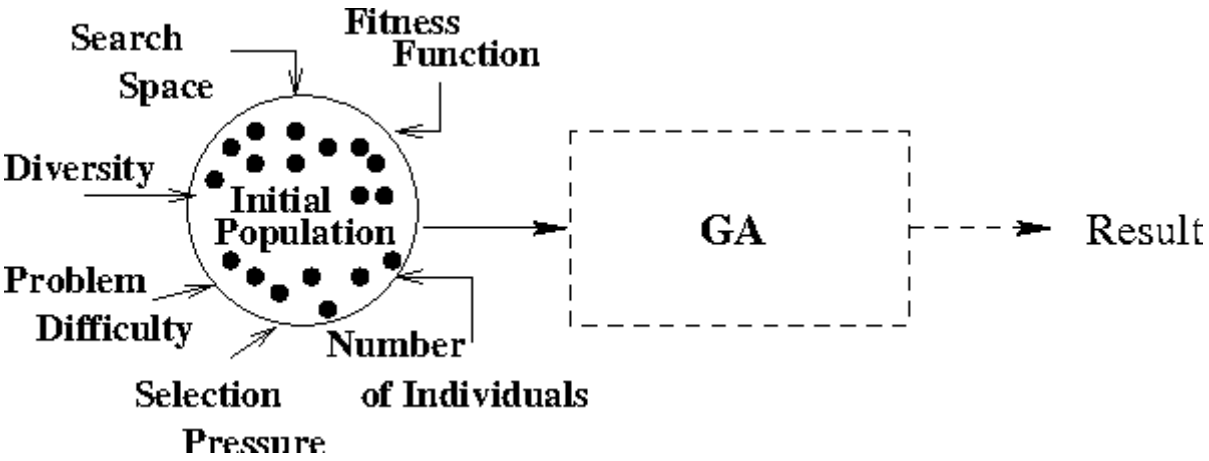
Five phases are considered in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

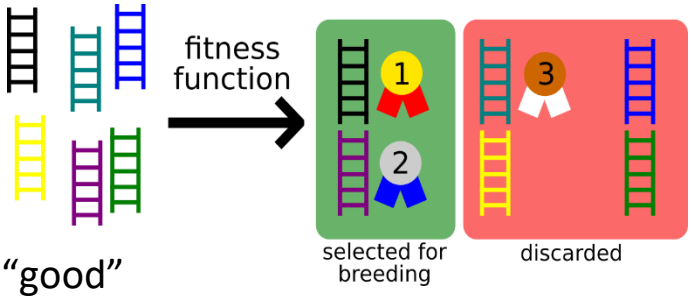


1) Initial Population

Population is a subset of solutions in the current generation. Population **P** can also be defined as a set of chromosomes. The initial population **P(0)**, which is the first generation is usually created randomly. In an iterative process, populations **P(t)** at generation **t (t =1,2,….)** are constituted.



2) Fitness Function

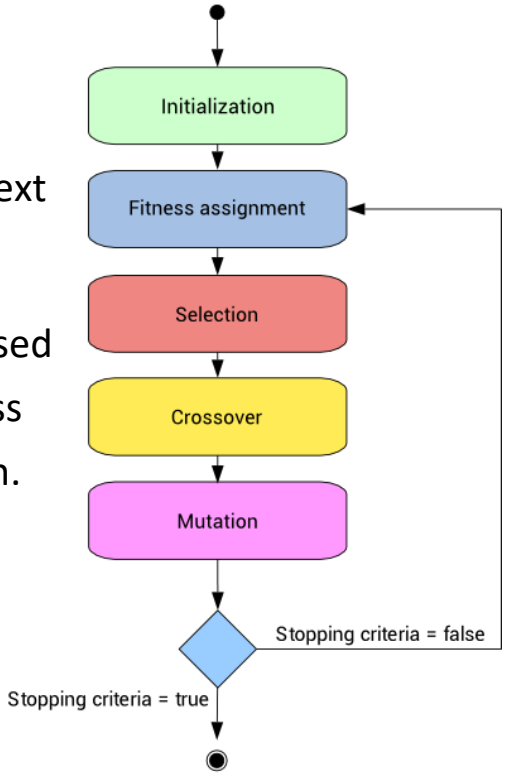


The fitness function simply defined is a function which takes a **candidate solution to the problem as input and produces as output** how "fit" our how "good" the solution is with respect to the problem in consideration.

A fitness function should possess the following characteristics −

- The fitness function should be sufficiently fast to compute.
- It must quantitatively measure how fit a given solution is or how fit individuals can be produced from the given solution.

3) Selection



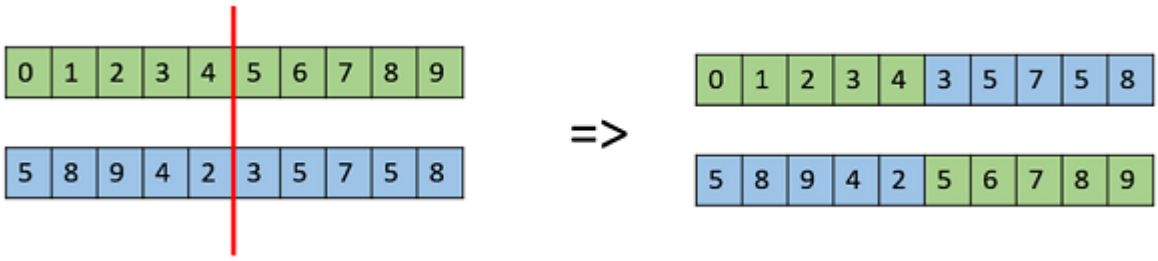
The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

4) Crossover

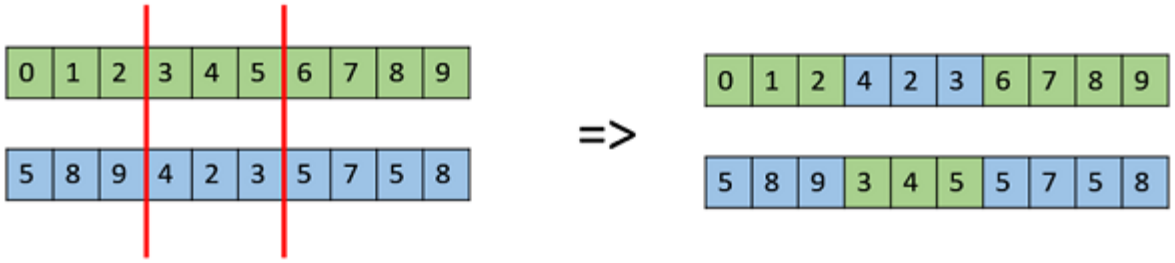
A. One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.

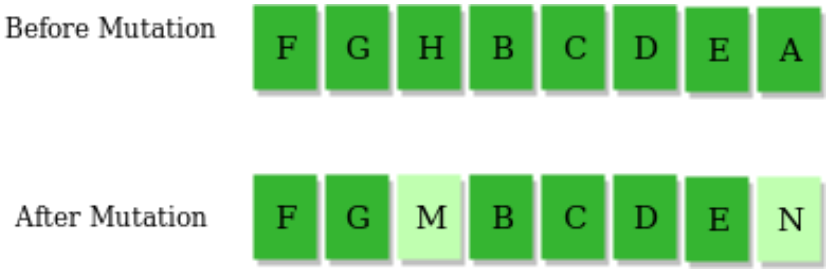


B. Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



5) Mutation



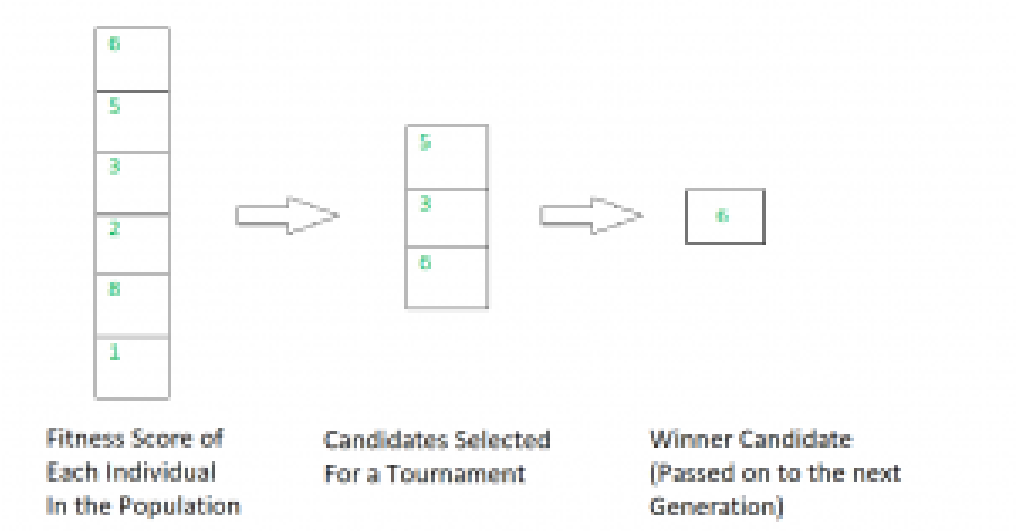
In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped.

Mutation occurs to maintain diversity within the population and prevent premature convergence.

6) Tournament Selection

Tournament Selection is a Selection Strategy used for selecting the fittest candidates from the current generation in a Genetic

Algorithm. These selected candidates are then passed on to the next generation. In a K-way tournament selection, we select k-individuals and run a tournament among them. Only the fittest candidate amongst those selected candidates is chosen and is passed on to the next generation.



Implementation Details

Crossover

```
def crossover(self, route_1, route_2):
    # YOUR CODE HERE
    newChild = Route(self.cities)
    randNum1 = random.randint(0, len(self.cities) - 2)
    randNum2 = random.randint(randNum1 + 1, len(self.cities) - 1)

    for target in range(randNum1, randNum2):
        newChild.assign_city(target, route_1.get_city(target))

    # we can write 20 instead of "len(self.cities)" because
    # we know that we have 20 cities, I obtain this  from CityManager class

    for target in range(randNum2, len(self.cities)):
        if newChild.get_city(target) is None:
            a = 0
            while a < randNum2:
                value = route_2.get_city(a)
                if value not in newChild:
                    newChild.assign_city(target, value)
                    break
                a += 1
```

```
    # here I am using "20" because it will occur same results
    for target in range(randNum2, 20):
        if newChild.get_city(target) is None:
            a = 0
            while a < randNum2:
                value = route_2.get_city(a)
                if value not in newChild:
                    newChild.assign_city(target, value)
                    break
                a += 1
    # we can either use FOR or WHILE loops, thats why I used them both
    # to proof it
    for target in range(0, len(self.cities)):
        if newChild.get_city(target) is None:
            for a in range(0, 20):
                value = route_2.get_city(a)
                if value not in newChild:
                    newChild.assign_city(target, value)
                    break

    potentialChange = random.random()
    if potentialChange <= self.mutation_rate:
        self.mutate(newChild)

    return newChild
```

The type of our crossover method is the Multi-Point crossover, we need to have two different spots to use this type of crossover, in this context I create 2 different numbers here, I randomly created the numbers I created. We change the values between the points we have determined with the help of crossover, just like a DNA, by swapping. As I mentioned in the code with comment, it has a structure that allows us to use both while and for. In this context, I wrote in both ways to show both.

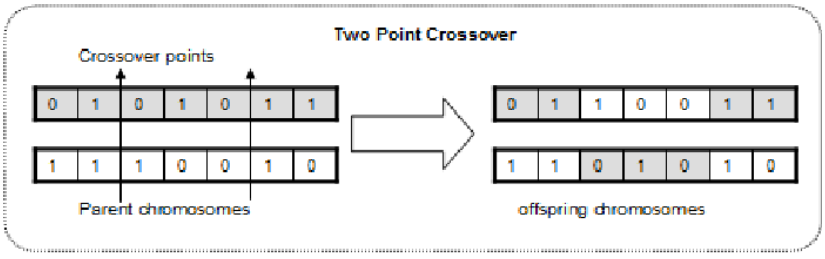


Figure 2. Two point crossover

Evolve

```
def evolve(self, routes):
    # YOUR CODE HERE
    # population_size == 10
    bb = 0
    while bb < self.population_size:
        aa, cc = 0, 0
        rot1, rot2, index, fit = [], [], [], []
        while cc < 2:
            for aa in range(0, self.tournament_size):
                ind = random.randint(0, self.population_size - 1)
                check = (ind in index)
                if check:
                    continue
                index.append(ind)
                route = routes.get_route(ind)
                fit.append(route)
                aa += 1
            if cc == 0:
                rot1 = self.tournament(fit)
            elif cc == 1:
                rot2 = self.tournament(fit)
            cc += 1
        update = self.crossover(rot1, rot2)
        routes.set_route(bb, update)
        bb += 1
    return routes
```

To use our crossover method, we need two different route variables, in this context, we create the routes that we will use in the evolve method. Then we run the loops here until “tournament_size”, since the value given to us is 10, we run 10 times. Here we create a route pool and randomly choose one from this route pool, but I have defined the necessary conditions to avoid duplication, thanks to which we do not face a situation such as repetition.

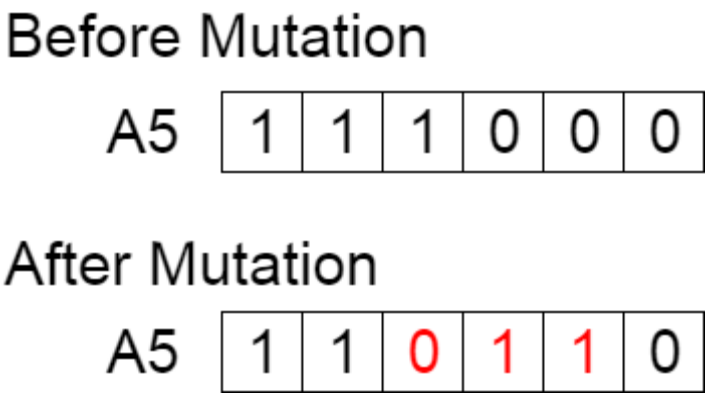
Tournament

```
def tournament(self, routes):
    # YOUR CODE HERE
    champ = None
    for a in range(0, len(routes)):
        aa = (champ is None)
        if aa or routes[a].calc_fitness() > champ.calc_fitness():
            champ = routes[a]
    # we are trying to find more fitter route in here
    return champ
```

With the Tournament method, he chooses the most optimal value for us from among the most suitable candidates, and the champion of this place deserves to be the crossover. There is no best or optimum value in this algorithm, a genetic algorithm is a type of algorithm that does its best.

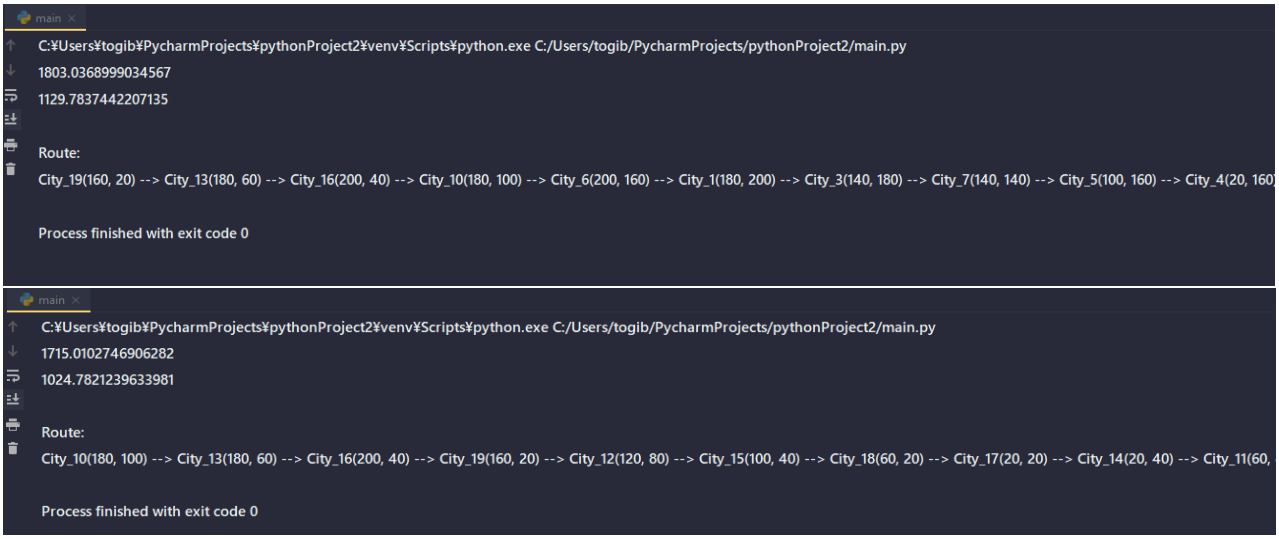
Mutate

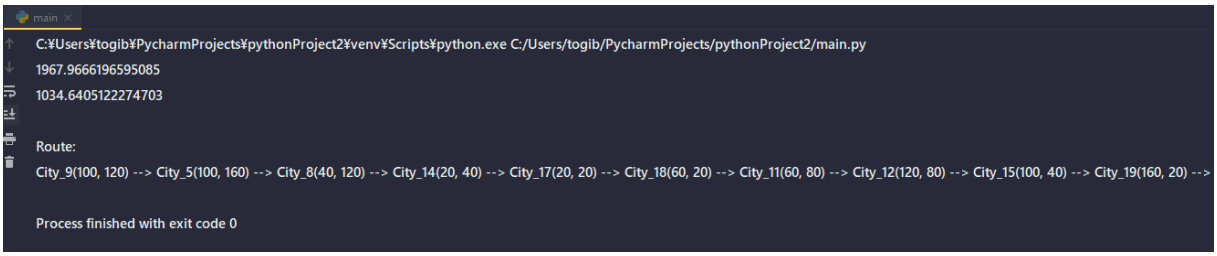
```
def mutate(self, route):
    # YOUR CODE HERE
    factor_1 = random.randint(0, route.__len__() - 1)
    factor_2 = random.randint(0, route.__len__() - 1)
    aa = route.get_city(factor_1)
    bb = route.get_city(factor_2)
    # basically what we are doing is swap operation
    route.assign_city(factor_1, bb)
    route.assign_city(factor_2, aa)
```



As a working principle, we can compare the mutate function to the swap operation. We are doing change operation with the random "factors" that we have defined here, and thus our mutation process is completed.

Results





Reference

https://blog.routific.com/travelling-salesman-problem

https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_fitness_function.htm#:~:text=The%20fitness%20function%20simply%20defined,to%20the%20problem%20in%20consideration.&text=The%20fitness%20function%20should%20be%20sufficiently%20fast%20to%20compute.

https://medium.datadriveninvestor.com/population-initialization-in-genetic-algorithms-ddb037da6773#:~:text=Population%20is%20a%20subset%20of,1%2C2%2C%E2%80%A6.)

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3#:~:text=A%20genetic%20algorithm%20is%20a,offspring%20of%20the%20next%20generation.

https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

https://www.geeksforgeeks.org/tournament-selection-ga/