Travelling Salesperson Problem – Genetic Algorithm

Lindie Chenou
[Computer Science & Engineering]
Speed School of Engineering
University of Louisville, USA
lcchen01@louisville.edu

1. **Introduction** (What did you do in this project and why?)

The goal of this project is to calculate the optical cost of visiting each city in a data file. The problem is best known as A Traveling Salesperson Problem (TSP). This project is requiring using the Genetic Algorithm to find the shortest route, given a dataset of 100 cities.

2. **Approach** (Describe algorithm you are using for this project)

The genetic algorithm is for solving both constrained and unconstrained optimization problems that are based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. It uses Chromosome, crossover, mutation, and survival of the fittest. One of the biggest advantages of using a Genetic algorithm is the ability to not know how to solve for the optimal solution. What is needed to be able to evaluate the quality of the generated solution coming and through iteration you get a good solution.

The first to do for the genetic algorithm is to randomly generate chromosomes, which are called population. Each chromosome represents an

individual solution. With the initial population created, the evolutionary generational cycle begins.

```
def initialPopulation(popSize, cityList):
    def createRoute(cityList):
        route = random.sample(cityList, len(cityList))
        return route
    population = []

    for i in range(0, popSize):
        population.append(createRoute(cityList))
    return population
```

After the initial population has been randomly created the next function that needs to be implemented is the fitness function. The point of the fitness function is to identify how good an individual is; it determines how to fit an individual is. It will assign a fitness score to each string. With that, the individuals with higher fitness value have a bigger probability of contributing offspring to the next generation.

```
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0

    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromCity = self.route[i]
                toCity = None
                if i + 1 < len(self.route):
                    toCity = self.route[i + 1]
                else:
                    toCity = self.route[0]
                pathDistance += fromCity.distance(toCity)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
```

With the fitness and initial population created, the next function is the selection function. The meaning behind the selection phase is to select the fittest parents and to let them pass on their genes to their children or the next generation. For the selection, two parents will be selected, and parents with a high fitness level will have the chance to reproduce.

The final function being added are the crossover and the mutation function. For the crossover function; in the first parent, a subset will be randomly selected. In which the remaining route of the string will be fill by the second parent where no genes are repeated, and they are inserted in the same order as they were for the second parent. In GA, mutation plays an important role as it helps to prevent local convergence by adding new pathways that will enable the program to explore other parts of the solution space. The swap mutation function will be implemented. With a defined low probability, two cities will swap places on their route. The mutate function is going to implement it for one individual. To compare the value of the crossover and the mutation functions, there will be two of each implementation for both of them. The population size and the mutation rate will both be investigated. Both of these are coming from Eric Stolz and the lecture corresponding with the evolution of a salesman. Using the number of generations as the stopping criteria for the result.

```
def mutatePopulation(population, mutationRate):
    def mutation(individual, mutationRate):
        for swapped in range(len(individual)):
            if(random.random() < mutationRate):
                swapWith = int(random.random() * len(individual))

                city1 = individual[swapped]
                city2 = individual[swapWith]

                individual[swapped] = city2
                individual[swapWith] = city1
        return individual
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutation(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)
    return mutatedPop
```

```
def crossoverPopulation(matingpool, eliteSize):
    def crossover(parent1, parent2):
        child = []
        childP1 = []
        childP2 = []

        geneA = int(random.random() * len(parent1))
        geneB = int(random.random() * len(parent1))

        startGene = min(geneA, geneB)
        endGene = max(geneA, geneB)

        for i in range(startGene, endGene):
            childP1.append(parent1[i])

        childP2 = [item for item in parent2 if item not in childP1]

        child = childP1 + childP2
        return child
    children = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0,eliteSize):
        children.append(matingpool[i])

    for i in range(0, length):
        child = crossover(pool[i], pool[len(matingpool)-i-1])
        children.append(child)
    return children
```

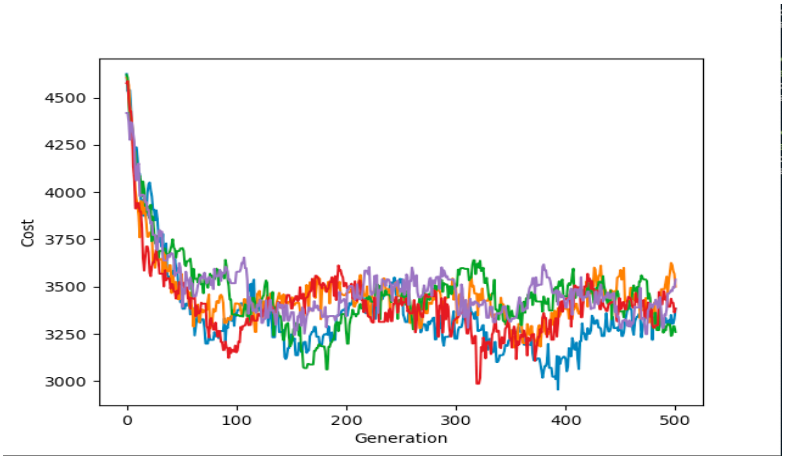
With everything completed for a generation, now it's time to implement another generation that will be produced using the nextGeneration function.

3. **Results** (How well did the algorithm perform?)

3.1 Data (Describe the data you used.)
A set of cities/nodes and their coordinates where provide. The set contained 100 cities and also provided the connection between each city.

Results (Numerical results and any figures or tables.)

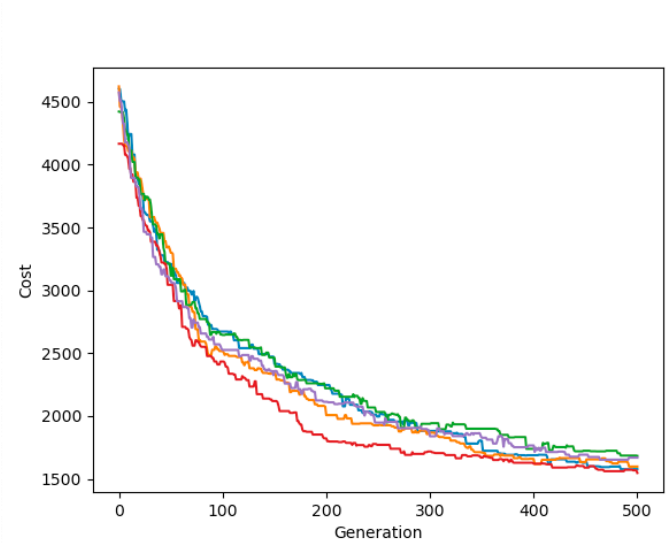
Using four datasets, the result for each one is listed below. The first dataset uses a population size of 100 and a mutation rate being 0.01. with these arguments running it over 50 times the lowest final distance ever receive was 3021. Also showing the graphically presentation of the improvement curves. The standard deviation for the first dataset happens to be 98.17, with the average being 3412.9914, the minimum and the maximum being 3259.666 and 3537.443 respectively.




```
In [14]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4626.053320204926
Final distance: 3384.2893687162878

In [15]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4606.301689097843
Final distance: 3499.724179032896

In [16]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4621.109861188522
Final distance: 3259.6666906501705

In [17]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4577.144319387113
Final distance: 3383.835185472732

In [18]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4419.131267462649
Final distance: 3537.4431472651786
```

```
geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)
```

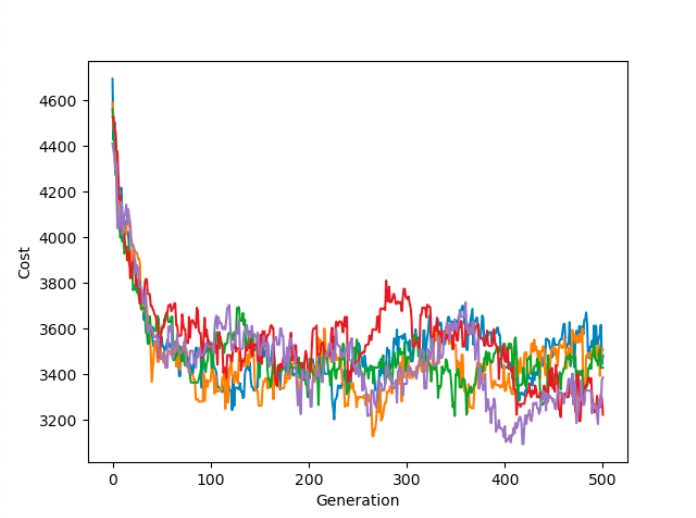
The second dataset uses a population of 100 and a mutation rate of 0.001. The standard deviation for the first dataset happens to be 52.749, with the average being 1615.5732, the minimum and the maximum being 1547.936 and 1683.705 respectively.




```
In [22]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4603.439004015537
Final distance: 1577.5314794014646

In [23]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4621.746414077315
Final distance: 1598.6836980178518

In [24]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4421.662430840254
Final distance: 1683.7058969504615

In [25]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4166.032645377041
Final distance: 1547.939612874433

In [26]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4574.36080074523
Final distance: 1670.0119499713726
```

```
geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20, mutationRate=0.001, generations=500)
```

The Third dataset which has a population of 200 and a mutation rate of 0.01. The standard deviation for the first dataset happens to be 100.978, with the average being 3405.3566 the minimum and the maximum being 3222.039 and 3508.658 respectively.

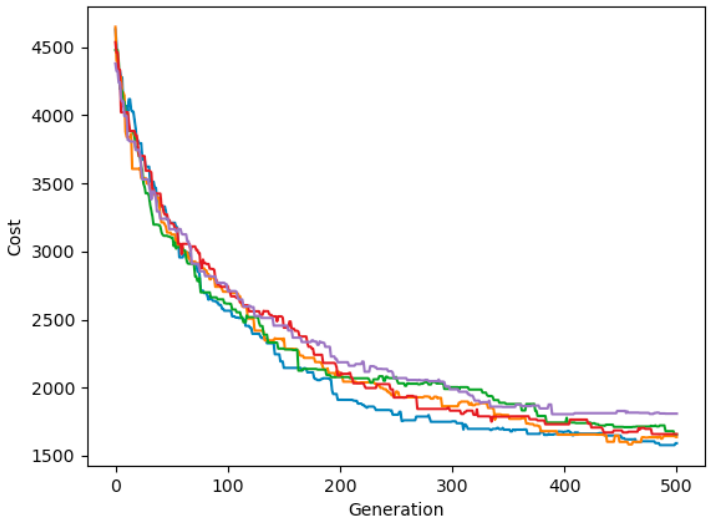



```
In [30]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4694.7519785804
Final distance: 3481.507421244771

In [31]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4589.050177243669
Final distance: 3508.658937873405

In [32]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4561.94049961448
Final distance: 3428.510844399346

In [33]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4528.173387980936
Final distance: 3222.0395679177022

In [34]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4411.514733907369
Final distance: 3386.0697905874013
```

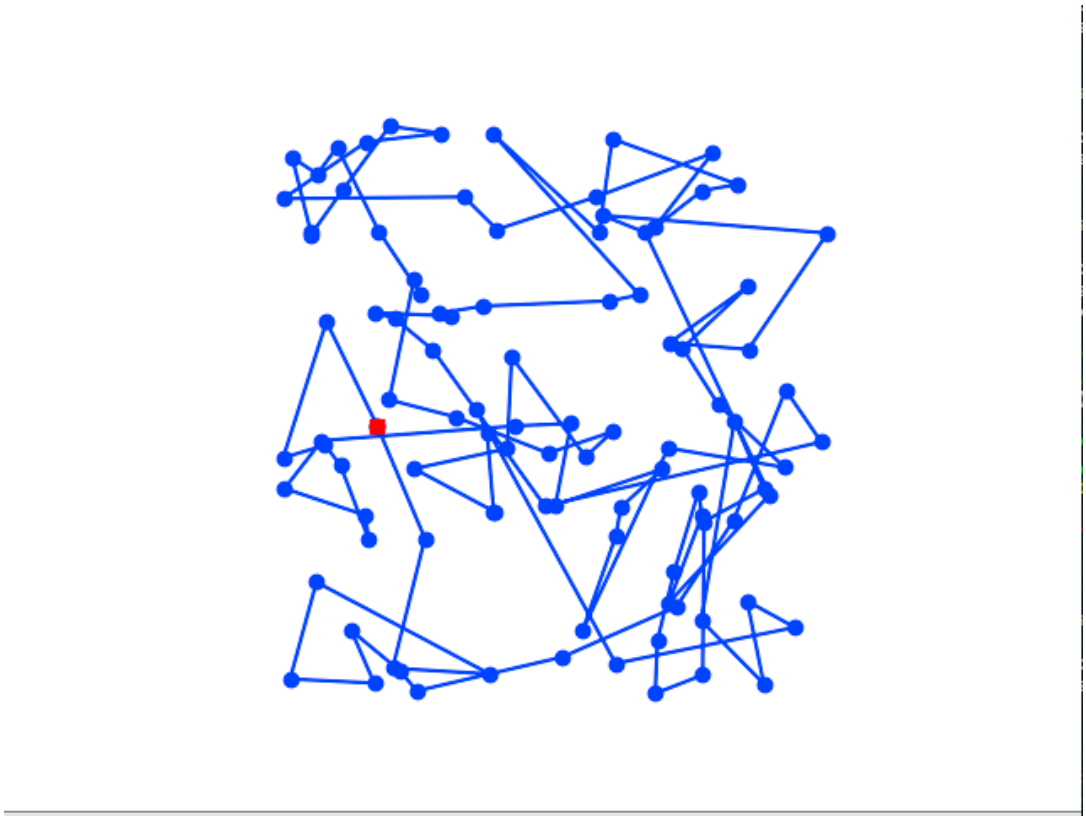
```
geneticAlgorithmPlot(population=cityList, popSize=200, eliteSize=20, mutationRate=0.01, generations=500)
```

The last dataset which has a population of 200 and a mutation rate of 0.001. The standard deviation for the first dataset happens to be 73.087, with the average being 1671.1252 the minimum and the maximum being 1592.540 and 1809.302 respectively.

```
In [36]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4630.299445431291
Final distance: 1592.5404396239128

In [37]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4648.423036520966
Final distance: 1638.2845847829556

In [38]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4478.3339397191485
Final distance: 1656.3752571088644

In [39]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4535.752339282091
Final distance: 1659.1250262219446

In [40]: runfile('/Users/lindiechenou/Desktop/CSE 545/
Project4/project4.py', wdir='/Users/lindiechenou/Desktop/CSE
545/Project4')
Initial distance: 4379.558784252247
Final distance: 1809.3025490656387
```




```
geneticAlgorithmPlot(population=cityList, popSize=200, eliteSize=20, mutationRate=0.001, generations=500)
```



4. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

The biggest improvement regarding the final distance is the mutation rate. The lower the mutation rate happens to be the lower the result will be. While the population size does have a factor in minimizing the final distance, it not a major improvement from the original population size of 100. With the mutation rate, there is about a 2000 drop on the final distance between 0.01 and 0.001. meanwhile to compile the project for each set of datasets it took about a minute for each one.

One thing I wish that I could change is having a tournament selection for the mating pool. Also using a different way of crossing. They are multiple ways to

implement the crossover. The single point crossover, two-point crossover, arithmetic crossover, and lastly the one implemented for this project the uniform crossover.

Genetic algorithms are good at taking and navigation big, potentially massive search spaces, searching for optimal combinations of items, solutions that you would otherwise not find in your lifetime. GA is a family of heuristics that are, in many cases, empirically good at providing a decent response, although they are rarely the best choice for a given domain.

5. **References** (If you used any sources in addition to lectures please include them here.)

https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
https://www.sciencedirect.com/topics/engineering/genetic-algorithm
https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b