Travelling Salesman Problem

using

Genetic Algorithms

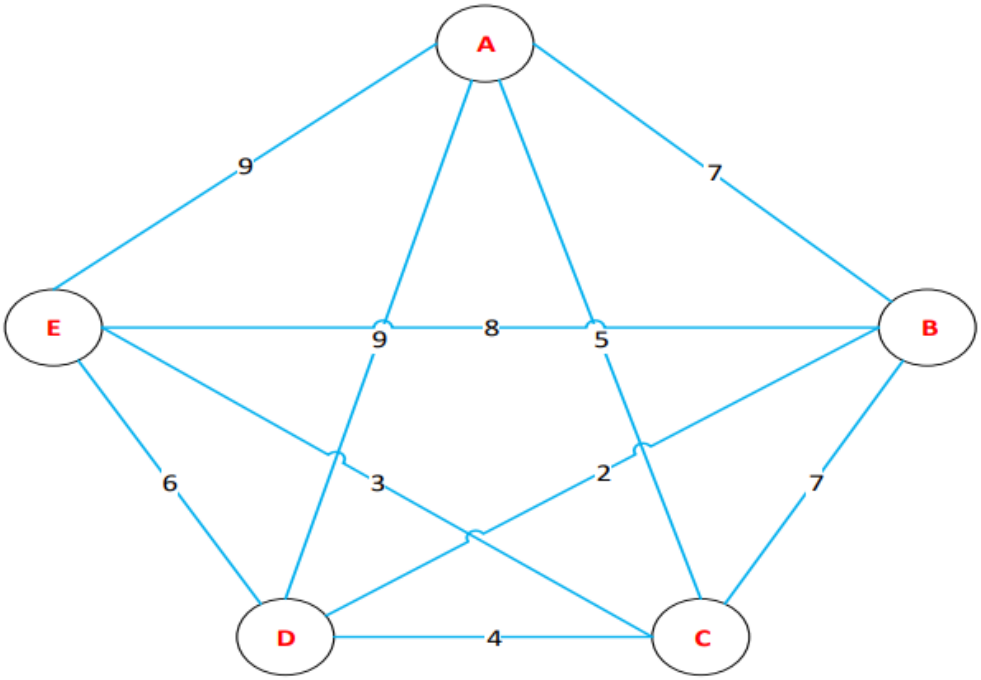
By: Muhammad Irfan

Introduction

Genetic algorithm is the branch of Artificial Intelligence. It is mainly used for solving optimization problems. The idea behind the genetic algorithm is the theory of evolution of natural selection. It describes that the fittest species will survive and the weaker species will be eaten or die. So, the best solution from the available solution for the given problem is selected while other all solutions are discarded [1][4]. In genetic algorithms, we try to find the best solution among the different available solutions. In this process, raw data is encoded into some genetic algorithm scheme, this process is called Genotype. While the decoding of this data is called Phenotype. Many daily life problems are being solved by genetic algorithms. For example, 8-Queens problem, 0-1 Knapsack Problem, N Queens Problem, Train Gear Problem, Image Reconstruction and Travelling Salesman Problem, etc. The main purpose of using genetic algorithms for solving these daily life problems is to bring optimization to the solutions of these problems [2]. In this paper we will solve the traveling salesman problem using the genetic algorithm the implementation language will be python and the implementation tool is Jupyter notebook, anaconda 3.0.

Basic Idea of Traveling Salesman Problem

The traveling salesman problem is a classical combinatorial optimization problem. It is a minimization problem. It is the problem that is extensively used for understanding optimization problems [4][6]. In traveling salesman problem, for traveling from one place to another we need to minimize many factors. These factors can be time, cost, and distance. In the traveling salesman problem, the salesman has to travel to different cities and has to be back to the city from where the salesman has started the journey. The objective function is to minimize the overall distance of the whole tour (this is depicted in figure TSP-1). It is an NP-hard problem. Like other NP-hard problems, the traveling salesman problem also cannot be solved in a polynomial way. The salesman can take many different paths for visiting different cities, but she has to select the best route in terms of the shortest distance. By applying a genetic algorithm to solving the Traveling Salesman Problem, the amount of distance among different cities can be optimized [2].



TSP -1

(*Courtesy to Zhou, Ai-Hua, at el.*)

There are two types of Traveling salesman problems:

1. **Symmetric Travelling Salesman Problem**

   In the symmetric Travelling Salesman Problem, the cost of traveling from city 1 to city 2 is the same as if the salesman travels back from city 2 to city1. It can be denoted as:

   $A \rightarrow B = X$

   $B \rightarrow A = X$

2. **Asymmetric Travelling Salesman Problem**
3. In Asymmetric Travelling Salesman Problem, the cost of traveling from city 1 to city 2 is not the same as if the salesman travels back from city 2 to city1. It can be denoted as:

   $A \rightarrow B \neq X$

   $B \rightarrow A \neq X$

In this paper, we have used the Symmetric Travelling Salesman Problem. The identity matrix is used to calculate the distance for Symmetric Travelling Salesman Problem. An example of an identity matrix is shown in the following figure (Figure-2).

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Figure -2

In the following table Table-1, example distances for different four cities have been supposed to demonstrate the problem easily. It can also be noted that the distance for the same city is zero which demonstrates the behavior of the identity matrix. And the symmetric behavior of the data can also be observed. For example, the distance from Karachi to Hyderabad is equal to 200 KM, which is the same as the distance from Hyderabad to Karachi.

	Karachi	**Hyderabad**	**Badin**	**Thatta**
Karachi	0	200	190	205
Hyderabad	200	0	210	225
Badin	190	210	0	15
Thatta	205	225	15	0

Table-1

If we suppose the above four cities. Then the possible path will depend on four cities and the ways a salesman can travel would be as follow:

$$4! = 24$$

A possible number of paths a salesman can opt for completing the tour is obtained by the factorial of the number of cities in the problem. If there are four cities, 24 possible paths would be there that a salesman can opt to complete the whole tour. In the following figure (Figure-3) the possible path is shown for the four cities if the salesman starts traveling from the city1.

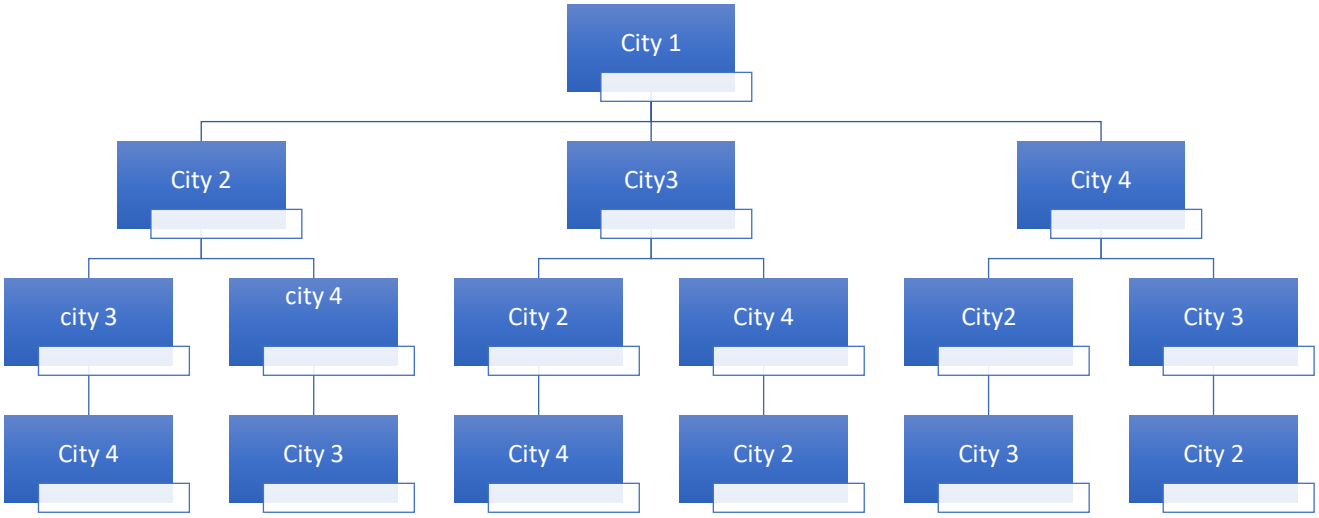


Figure -3

Implementation

Encoding & Implementation

For encoding we convert the problem space into solution space. For encoding traveling salesman problem we have used following steps:

- Gene: Any single city which the salesman will visit is the gene in this example.
- Chromosome: The one proposed tour is the chromosome. The chromosome will contain order in which the salesman will travel from one city to another for completing his journey.

2	6	9	5	3	8

- The overall aim of the paper is to select the distance form one city to another. For that Pythagoras theorem is used for finding the distance between the cities.

$$\text{Distance A} \rightarrow \text{B} = \sqrt{x^2 - y^2}$$

$$= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

The above formula has also been used by the [5]

- As a rule of thumb in genetic algorithms the population size is kept smaller, otherwise the genetic algorithm will slow down. Therefore, The population size for this demonstration is 20.
- The Chromosomes size is 20.
- Roulette Wheel selection method is used for the parents' selection. Roulette Wheel calculates the probability for any particular chromosome to be selected.

$$\text{Pi} = \frac{Fi}{\sum_{j1}^{n} f j}$$

Fi the fitness value of the same chromosome divided by the summation of all the fitness values of all the chromosomes.

In roulette wheel selection, the chromosome with the highest value has more chances of selection.

- Single point cross over is used in this paper.
- There are three mutation operators i.e., bit flip, swap bits and reversing. Bit flip is used in binary encoding while in traveling salesman problem every gene is holding individual city number not the binary value. Therefore, swap bits are used for the mutation operator.

Initial Population Generation

The following code is used for the initial population generation. The function uses random generation between 1 to 20 for the first time and populates in the first chromosome. It also checks the duplicate value. If duplicate exists it will replace the value with the new one.

```
def generateFirstPopulation():
    for _ in range(1, POPULATION_SIZE + 1):
        generatePossiblePath()
```

Following is the first randomly generated population using the above formula

```
generateFirstPopulation()
population
```

```
[[2, 20, 8, 18, 4, 10, 9, 19, 7, 14, 17, 1, 12, 5, 13, 3, 6, 11, 15, 16],
 [14, 19, 1, 3, 4, 8, 7, 2, 18, 17, 12, 15, 20, 10, 5, 11, 13, 16, 6, 9],
 [8, 4, 16, 14, 18, 12, 7, 11, 1, 13, 6, 2, 9, 19, 17, 3, 10, 20, 15, 5],
 [12, 9, 3, 8, 20, 17, 5, 6, 18, 14, 13, 16, 15, 11, 2, 1, 10, 4, 7, 19],
 [1, 8, 18, 17, 2, 11, 20, 15, 6, 14, 4, 9, 5, 16, 13, 19, 12, 10, 7, 3],
 [10, 4, 8, 20, 5, 13, 19, 11, 17, 3, 12, 6, 16, 14, 1, 9, 2, 7, 18, 15],
 [8, 11, 18, 19, 3, 5, 1, 13, 14, 9, 17, 15, 12, 7, 10, 16, 4, 2, 20, 6],
 [13, 19, 18, 7, 1, 12, 6, 17, 9, 4, 2, 15, 14, 20, 3, 16, 10, 11, 8, 5],
 [19, 15, 3, 11, 18, 12, 20, 2, 4, 9, 7, 16, 1, 10, 17, 8, 14, 13, 6, 5],
 [1, 9, 14, 11, 16, 2, 17, 6, 5, 13, 19, 12, 7, 15, 8, 18, 3, 4, 20, 10],
 [5, 10, 4, 8, 9, 11, 3, 19, 20, 13, 17, 6, 12, 2, 14, 7, 1, 18, 16, 15],
 [20, 16, 19, 2, 17, 7, 14, 13, 10, 18, 11, 9, 4, 3, 15, 5, 12, 6, 8, 1],
 [20, 4, 2, 6, 18, 13, 14, 17, 15, 12, 11, 10, 9, 16, 5, 8, 1, 7, 3, 19],
 [20, 17, 5, 14, 11, 10, 6, 16, 4, 2, 12, 9, 3, 19, 8, 1, 15, 7, 18, 13],
 [8, 1, 14, 6, 19, 18, 10, 20, 5, 15, 13, 9, 2, 11, 16, 4, 7, 12, 17, 3],
 [13, 17, 7, 5, 8, 19, 15, 9, 4, 1, 6, 20, 2, 11, 18, 10, 3, 14, 16, 12],
 [11, 3, 7, 12, 1, 6, 14, 19, 10, 17, 9, 4, 5, 13, 2, 16, 15, 8, 20, 18],
 [19, 20, 14, 7, 9, 2, 8, 4, 16, 15, 12, 5, 17, 18, 3, 6, 10, 1, 11, 13],
 [8, 20, 11, 16, 7, 5, 10, 3, 14, 19, 6, 4, 18, 2, 12, 1, 17, 13, 15, 9],
 [1, 17, 19, 12, 10, 2, 14, 5, 20, 7, 11, 3, 16, 15, 9, 4, 13, 18, 8, 6],
 [3, 13, 10, 1, 20, 4, 15, 11, 17, 5, 2, 14, 9, 16, 6, 18, 19, 7, 8, 12]]
```

Fitness Calculation

The following two function of the code will calculation the fitness value for each chromosome and the fitness value for the identity matrix. In simple words fitness function is calculation the distance among the cities in which the salesman will travel.

```
def fitnessFunction():
    for i in range(len(population)):
        for j in range(len(population)):
            dCidade[i][j] = round(math.sqrt(((x[i]-x[j])**2)+ ((y[i]-y[j])**2)),4 )
    return calculateDistances()
```

```
def calculateDistances():
    global distances
    distances = [0 for x in range(CITIES_SIZE)]
    for i in range(len(population)):
        for j in range(len(population[i])):
            firstPos = 19 if tour[i][j] == 20 else tour[i][j]
            secondPos = 19 if tour[i][j+1] == 20 else tour[i][j+1]
            distances[i] += round(dCidade[firstPos][secondPos], 4)
    dict_dist = {i:distances[i] for i in range(0, len(distances))}
    distances = copy.deepcopy(dict_dist)
    return sorted(distances.items(), key=lambda kv: kv[1])
```

Selection Operator

Roulette Wheel selection method is used for the parents' selection. Roulette Wheel calculates the probability for any particular chromosome to be selected. Following two function are being used for the parents selection.

```
def rouletteFunction(sorted_x):
    global parentsOne
    global parentsTwo
    arr = []
    rouletteArr = []
    for i in range(10):
        arr.append(sorted_x[i][0])
    for j in range(len(arr)):
        for _ in range(10 - j):
            rouletteArr.append(arr[j])
    parentsOne = createParents(rouletteArr)
    parentsTwo = createParents(rouletteArr)
```

```
def createParents(rouletteArr):
    parentArr = []
    for _ in range(5):
        parentArr.append(rouletteArr[random.randint(0,54)])
    return parentArr
```

Crossover Operation

Single point crossover operation is used in this travelling salesman problem. Following is the code which will perform the crossover operation.

```
for i in range(5):
    parentsOneAux = parentsOne[i]
    parentsTwoAux = parentsTwo[i]
    usedIndexes = []

    randomIndexInsidecromosom = random.randint(0, POPULATION_SIZE - 1)

    usedIndexes.append(randomIndexInsidecromosom)

    childOne = copy.deepcopy(population[parentsOneAux])
    childTwo = copy.deepcopy(population[parentsTwoAux])

    valAuxOne = childOne[randomIndexInsidecromosom]
    valAuxTwo = childTwo[randomIndexInsidecromosom]

    childOne[randomIndexInsidecromosom] = valAuxTwo
    childTwo[randomIndexInsidecromosom] = valAuxOne
```

Mutation Operation

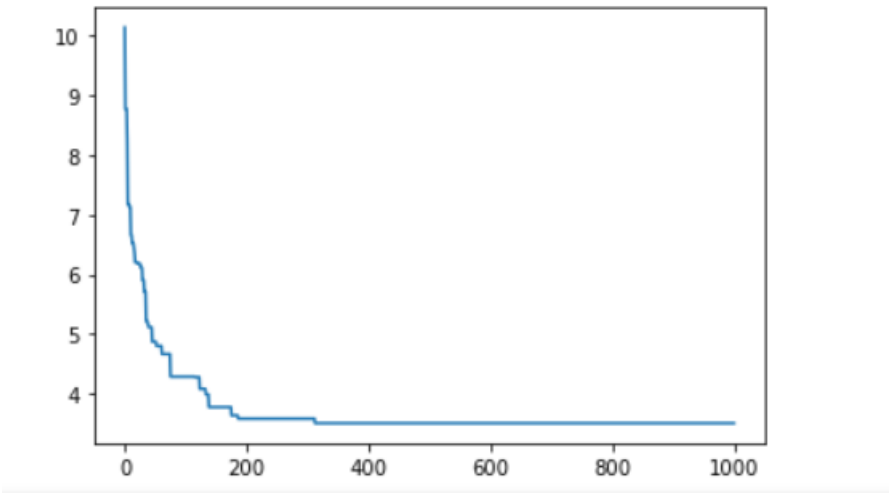
There are three mutation operators i.e., bit flip, swap bits and reversing. Bit flip is used in binary encoding while in traveling salesman problem every gene is holding individual city number not the binary value. Therefore, swap bits are used for the mutation operator.

- In this example we have used 5% mutation probability. Which means there are 5% percent chances that the new offspring will be mutated or not. Following code implements the mutation operation

```
def mutate(matrix):
    for i in range(0, len(matrix)):
        for _ in range(0, len(matrix[i])):
            ranNum = random.randint(1,100)
            if ranNum >= 1 and ranNum <= 5:
                indexOne = random.randint(0,19)
                indexTwo = random.randint(0,19)
                auxOne = matrix[i][indexOne]
                auxTwo = matrix[i][indexTwo]
                matrix[i][indexOne] = auxTwo
                matrix[i][indexTwo] = auxOne
```

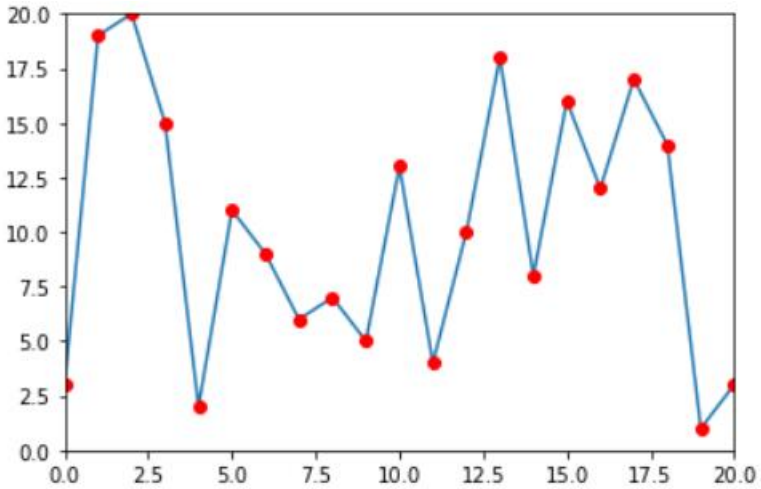
Results and Discussion

Since the termination condition is the number of cycles the code will execute in start it was set to 9999. But after few numbers of execution it was observed that the optimized results is being obtained after 360 cycles so that the maximum execution was done till 1000 cycles.



The best path that the Traveling salesman can opt is calculated as shown in the following two images.

```
Total Polulation: 20
Mutation Probability: 5%
Number of cities: 3.5047000000000006
Best Route: [3, 19, 20, 15, 2, 11, 9, 6, 7, 5, 13, 4, 10, 18, 8, 16, 12, 17, 14, 1]
```



References

1. Furqan, Mhd & Sitompul, Opim & Mawengkang, Herman & Siahaan, Andysah Putera Utama & Donni Lesmana Siahaan, Muhammad & Wanayumini, & Nasution, Nazaruddin. (2018). A Review of Prim and Genetic Algorithms in Finding and Determining Routes on Connected Weighted Graphs. International Journal of Civil Engineering and Technology. 9. 1755-1765.

2. Zhou, Ai-Hua, Li-Peng Zhu, Bin Hu, Song Deng, Yan Song, Hongbin Qiu, and Sen Pan. 2019. "Traveling-Salesman-Problem Algorithm Based on Simulated Annealing and Gene-Expression Programming" *Information* 10, no. 1: 7. https://doi.org/10.3390/info10010007

3. Hariyadi, Putri Mutira, Phong Thanh Nguyen, Iswanto Iswanto, and Dadang Sudrajat. "Traveling salesman problem solution using genetic algorithm." Journal of Critical Reviews 7, no. 1 (2020): 56-61.

4. El Hassani, Hicham & Said, Benkachcha & Benhra, Jamal. (2015). New Genetic Operator (Jump Crossover) for the Traveling Salesman Problem. International Journal of Applied Metaheuristic Computing. 6. 10.4018/IJAMC.2015040103.

5. Alsalibi, Bisan & Babaeianjelodar, Marzieh & Venkat, Ibrahim. (2012). A Comparative Study between the Nearest Neighbor and Genetic Algorithms: A revisit to the Traveling Salesman Problem. nternational Journal of Computer Science and Electronics Engineering. 1. 34-38.

6. Arora, Khushboo, Samiksha Agarwal and Rohit Tanwar. “Solving TSP using Genetic Algorithm and Nearest Neighbour Algorithm and their Comparison.” (2016).