Travelling Salesman Problem

November 26, 2019

GAA Project 2019. Course teacher: ***Dr. Vibhor Kant***

Team Member	Roll Number
Ansh Mittal	17UCS028
Anshul Jain	17UCS029
Anshul Kiyawat	17UCS030
Anshu Musaddi	17UCS185

1 Importing the required libraries

[1]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib notebook
```

2 Defining the Problem

Travelling Salesman Problem (TSP) is in which **n cities and distance between each pair is given**.

We have to find a **shortest route to visit each city exactly once and come back to the starting point**.

For example if we have n=3 i.e cities numbered 1 to 3. With a distance matrix containing distances between each pair of cities.

0	1	2	3	4
1	0	50	100	25
2	50	0	150	50
3	100	150	0	75
4	25	50	75	0

Going on route 4 -> 3 -> 1 -> 2 -> 4 will be a **more optimal route than** going on route 1 -> 4 -> 2 -> 3 -> 1.

3 Defining the functions required for Genetic Algorithm

3.1 Fitness Function

It is the Summation of distances from one city to another We have to **minimize** the fitness to achieve the Optimized Output.

We take a chromosome and simply calculate te distance from one city to another and sum them all.

[2]:
```
def fitness(chrm,distances):
    dis = 0
    for i in range(len(chrm)-1):
        dis += distances[chrm[i]][chrm[i+1]]

    dis += distances[chrm[0]][chrm[len(chrm)-1]]

    return dis
```

3.2 Population Initialization

The following function initializes a unique random chromosomes. The population size will be of size n passed as a parameter. The following function will return a Pandas dataframe consisting of Chromosomes and their respective fitness values

[3]:
```
def randinit(n,lenchrm,dis):
    df = pd.DataFrame(np.zeros((n,2)),columns=["Chromosome","Fitness"],␣
 ↪dtype=object)
    i = 0
    arr = list(range(lenchrm))
    while i < n:
        random.shuffle(arr)
        if arr not in list(df["Chromosome"]):
            df.at[i,"Chromosome"] = arr.copy()
            df.at[i,"Fitness"] = fitness(arr,dis)
            i = i+1
    df["Fitness"] = df["Fitness"].astype('int64')
    return df
```

3.3 Distance between Two Cities

Randomly Generating distances between each pair of cities

[4]:
```
def distances(No_Of_Cities):

    distance = np.zeros((No_Of_Cities,No_Of_Cities))

    for city in range(No_Of_Cities):
        cities = [i for i in range(No_Of_Cities) if not i==city]
```

```
        for to_city in cities:
            if(distance[city][to_city]==0 and distance[to_city][city]==0):
                distance[city][to_city] = distance[to_city][city] = random.
↪randint(1000,100000)

    return distance
```

3.4 Crossover

We are using **cut and crossfill** crossover where the crossover point is in the middle of chromosome.

[5]:
```
def crossover(chrm1,chrm2):
    n = len(chrm1)
    chrm3 = chrm1[:int(n/2)]
    for i in range(int(n/2), int(n/2)+n):
        if chrm2[i%n] not in chrm3:
            chrm3.append(chrm2[i%n])
    n = len(chrm2)
    chrm4 = chrm2[:int(n/2)]
    for i in range(int(n/2), int(n/2)+n):
        if chrm1[i%n] not in chrm4:
            chrm4.append(chrm1[i%n])

    return chrm3, chrm4
```

3.5 Mutation

We are performing **random swapping** between any two allele in the chromosome. It is ensured that swapping points are different.

[6]:
```
def mutation(chrm):
    if random.random() < 1/len(chrm):
        n = len(chrm)
        a = random.randint(0, n-1)
        b = random.randint(0, n-1)
        while b==a:
            b = random.randint(0, n-1)
        chrm[a], chrm[b] = chrm[b], chrm[a]
    return chrm
```

3.6 Mating Pool

For parent selection **best 2 out of random 5** chromosomes are chosen with repetition **not** allowed.

[7]:
```
def giveParent(pop):
    n = pop.shape[0]
```

```
    parents = pd.DataFrame(np.zeros((5,2)),columns=['Parents',"Fitness"],␣
↪dtype=object)
    i = 0
    while len(parents) <= 5:
        r = random.randint(0, n-1)
        parent = pop.iloc[r, 0]
        fitness = pop.iloc[r,1]
        if parent not in list(parents["Parents"]):
            parents.at[i,"Parents"] = parent
            parents.at[i,"Fitness"] = fitness
            i = i+1

    parents = parents.sort_values(by=['Fitness'])
    return parents.iloc[0, 0], parents.iloc[1, 0]
```

3.7 Replacing With New Generation

The bottom 2 worst chromosomes are replaced with the new better chromosomes. This is also called **Replace worst**.

[8]:
```
def replaceWorst(p, dis, v=False):
    pop = p.copy()
    p1, p2 = giveParent(pop)
    c1, c2 = crossover(p1, p2)
    c1 = mutation(c1)
    c2 = mutation(c2)
    if v:
        print('Child 1:{0}\tFitness: {1}\nChild 2:{2}\tFitness: {3}'.format(c1,␣
↪fitness(c1,dis), c2, fitness(c2,dis)))

    pop = pop.sort_values(by=['Fitness'])
    pop = pop.reset_index(drop=True)

    if c1 not in list(pop["Chromosome"]):
        pop.loc[p.shape[0]-2] = [c1, fitness(c1,dis)]

    if c2 not in list(pop["Chromosome"]):
        pop.loc[p.shape[0]-1] = [c2, fitness(c2,dis)]

    return pop
```

4 Utility Functions

The following function gives the fitness of the fittest chromosome from the population in return.

[9]:
```
def bestFitness(pop):
    return min(list(pop['Fitness']))
```

The following function gives the fittest chromosome from the population in return.

[10]:
```
def bestChromosome(pop):
    p = pop.sort_values(by=['Fitness'])
    return p.iloc[0, 0]
```

5 Execution of the Genetic Algorithm

This program accepts user chosen population size and number of cities although for representation purposes report is shown on values 200 and 10 respectively for above variables. Moreover a total of 10,000 generations is set as upper limit on number of generations.

[11]:
```
%%time

N = int(input('Value of N: '))
pop_size = int(input('Population Size: '))
show_progress = False
NUMBER_OF_CITIES = N;

dis = distances(N)

generationInfo = pd.DataFrame(np.zeros((10000, 2)), columns=['Chromosome',␣
↪'Fitness'], dtype=object)

for generation in range(10000):
    if generation == 0:
        pop = randinit(pop_size, N, dis)

    generationInfo.loc[generation] = [bestChromosome(pop), bestFitness(pop)]

    if show_progress:
        print("\nGeneration {}".format(generation))
    pop = replaceWorst(pop,dis,show_progress)

Optimal_Solution = [bestChromosome(pop), bestFitness(pop)]
```

```
Value of N: 10
Population Size: 200
CPU times: user 48.3 s, sys: 77.1 ms, total: 48.3 s
Wall time: 52 s
```

6 Generated Optimal Solution

[13]:
```
Optimal_Solution
```

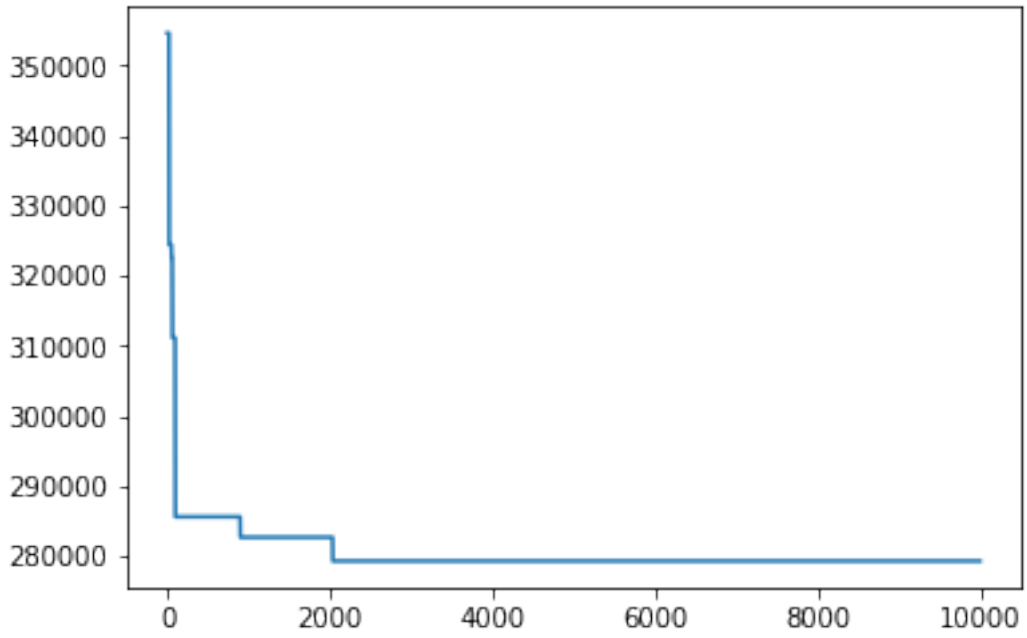
[13]:
```
[[6, 5, 9, 4, 2, 3, 7, 1, 8, 0], 279269.0]
```

7 Performance of the Genetic Algorithm for each Generation

Plot of the best fitness in a generation versus generation number.

[14]:
```
import matplotlib.pyplot as plt
plt.plot(range(generationInfo.shape[0]), list(generationInfo['Fitness']))
```

[14]:
```
[<matplotlib.lines.Line2D at 0x7f64ac769310>]
```



[ ]:
```

```