INSTRUCTOR: MECIT YAMAN |Department of Aeronautical Engineering

[redacted]

MURAT BATUHAN GÜNAYDIN

140121028

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Travelling salesman is a problem which asks the following question; how to travel on a route of a given cities with given distances to each other. As city numbers increases solution time and solution clearness increases. In this project Turkey cities were used as cities that the salesman will travel. Since there are 81 cities on Turkish map, it can be said that it is almost impossible to find the best path with brute force. But the optimum value for the best path can be estimated by using algorithms such as genetic algorithm (GA) and simulated annealing (SA).

In this Project firstly the optimum values were calculated with genetic algorithm by coding on python, and then simulated annealing algorithm was used to find the optimum path and distance. Results and performance of both algorithms were compared.

GENETIC ALGORITHM

Genetic algorithm is the one of the best algorithm to calculate the best path for salesman. This algorithm mainly based on starting from a set of lists and by crossing over and mutation processes creating new list sets. Then sustaining this process until the values are converge.

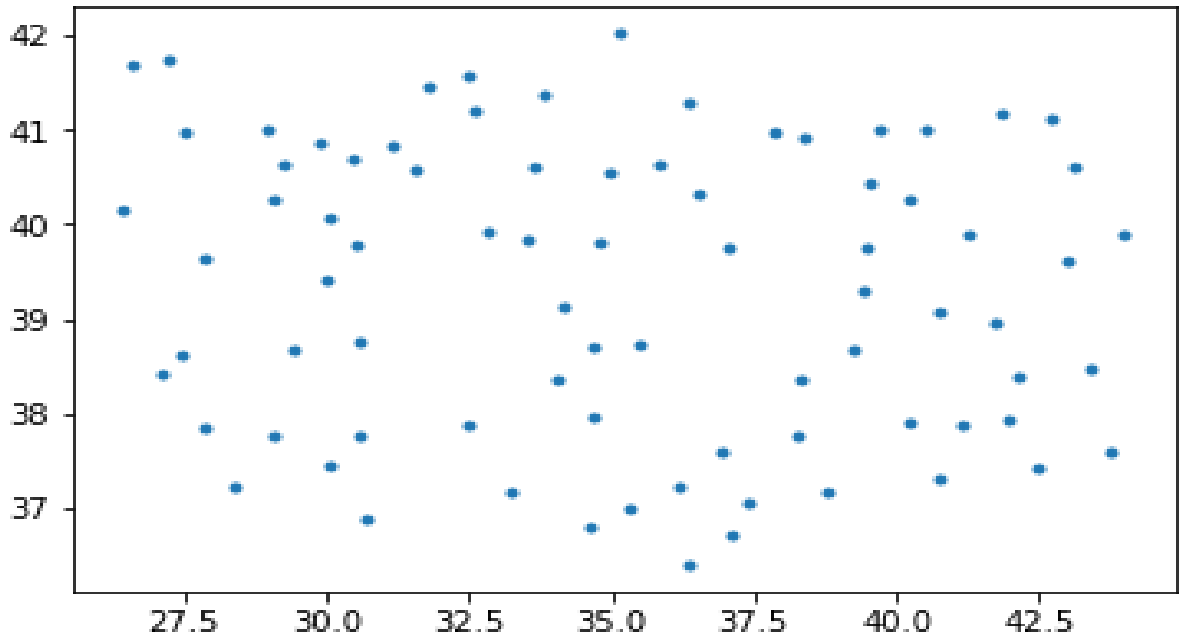


Figure 1 - City Locations

Figure 2- Turkey Map

Steps of Genetic Algorithm Used in This Project:

1- Defining coordinates of 81 cities of Turkey to the Python, and storing them as numpy arrays as x and y

2- Assigning Ankara as the beginning and ending point, also shuffling the path.

3- Defining distances between the cities to the python as numpy arrays.

4- Function for drawing path for the salesman

5- Crossing over process which means the creating new generation paths from older generation paths by adding them each other, subtracting doubled cities and adding missing cities. Also making mutations which means, increasing the diversity of populations by randomly changing two cities on a route

6- Creating initial population for salesman.

7- Creating the new generations from older ones by using the functions written before,

8- Finally running the code with for loop with number of maximum iterations of 10000, plotting the fitness and path plots and calculating the optimum distance in iteration.

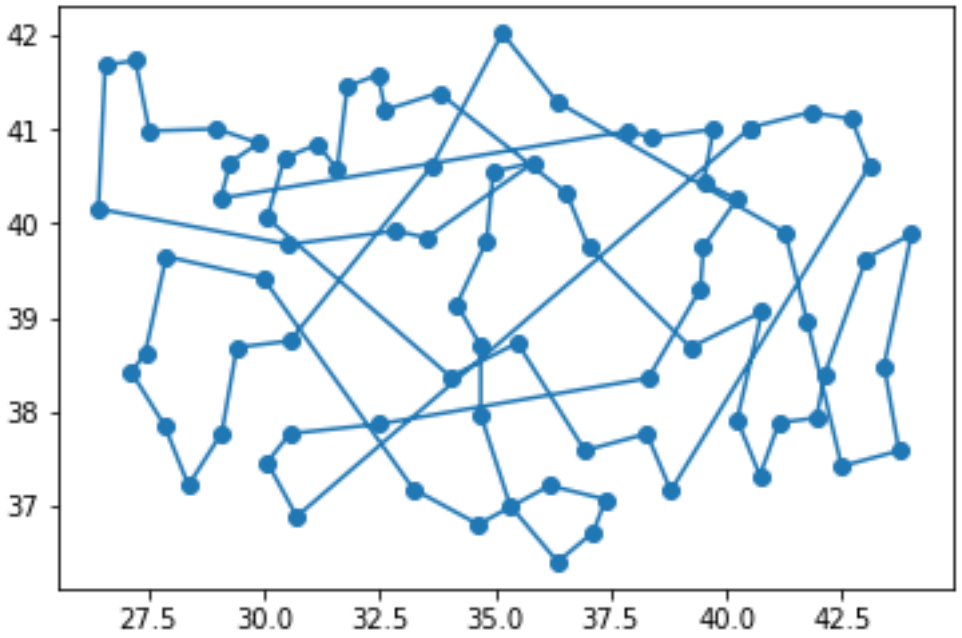


Figure 3 – Optimum Path according to GA

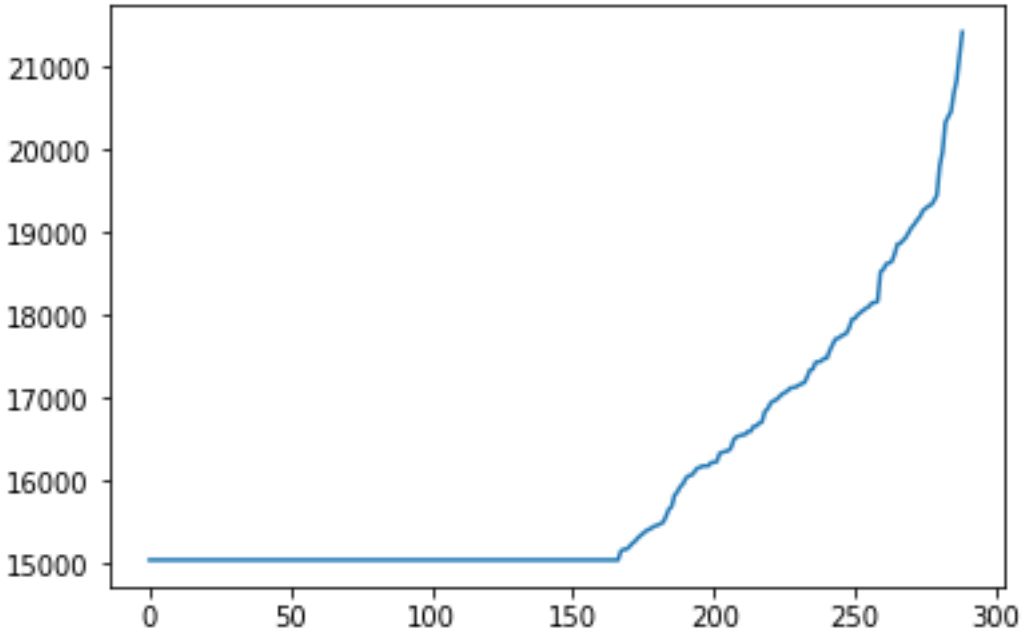
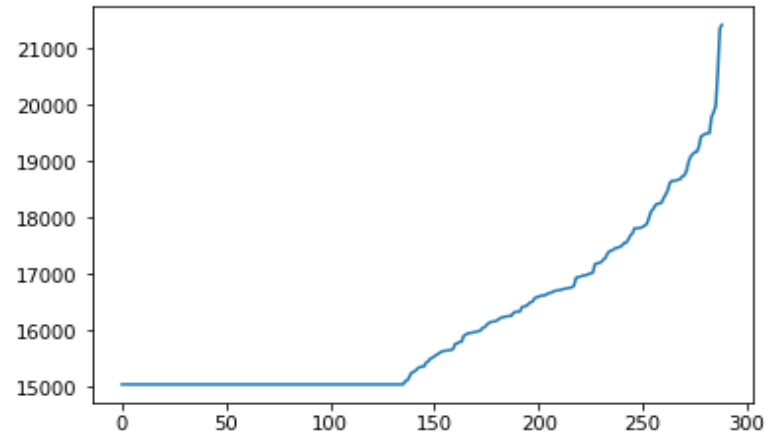
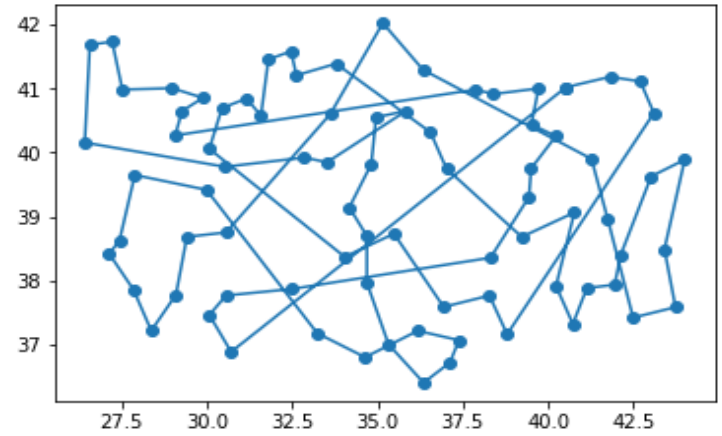


Figure 4 - Fitness Graph




```
iteration 671 distance is:  15047.0
```



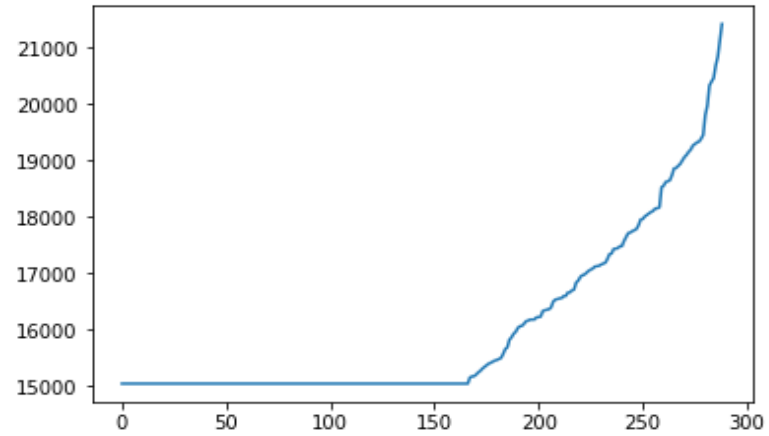


Figure 5- GA Output

The optimum distance was found as 15047 km with number of iterations, the population number was arranged as 300.

From the figure 5 it was observed that, fitness did not become exactly fit with 671 iterations, and it took about 20 minute with a Toshiba satellite i7 CPU 2.50 GHz computer to solve algorithm with 671 iterations.

SIMULATED ANNEALING ALGORITHM

Initially we set it to high and let 'cool' slowly while the algorithm was running. Even though this temperature variable is high, the algorithm will be allowed more often to accept solutions that are worse than the current solution. This gives the algorithm the ability to exit from any local optimal state at which the execution finds itself early. As the temperature decreases, there is a chance to accept worse solutions, so the algorithm is gradually allowed to focus on a search field where it can be found close to an optimum solution.

Steps of Simulated Annealing Used in This Project

1- Defining coordinates of 81 cities of Turkey to the Python, and storing them as numpy arrays as x and y
2- Defining Ankara as the beginning point for the salesman
3- Define the First Temperature and Cooling Speed.
4- Create a random path.
5- Create a new path by changing two points of the current path.
6- Calculate the distance of each path.
7- Update temperature using equation
8- Loop until the condition you want is met:

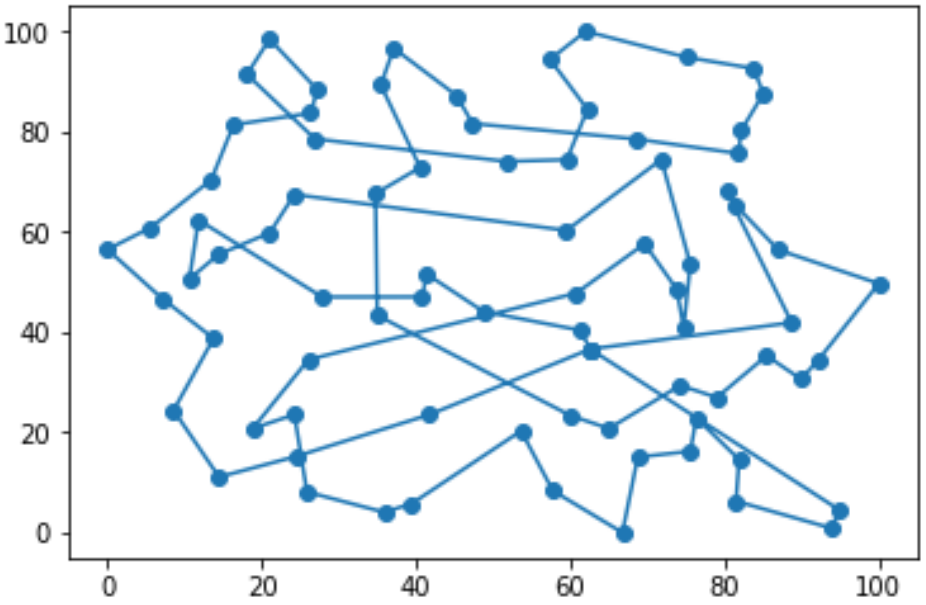
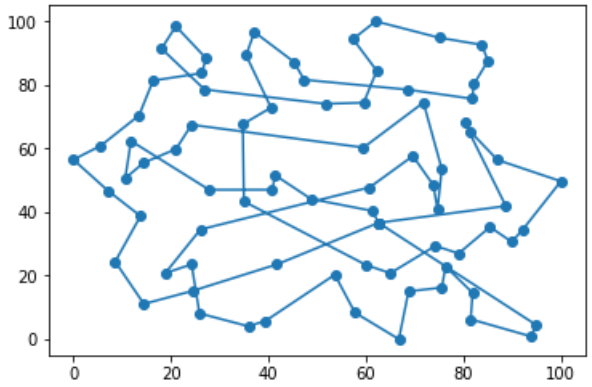


Figure 6- SA Path

```
In [2]: runfile('C:/Users/S2B/Desktop/Python/simulated.py', wdir='C:/Users/
S2B/Desktop/Python')
```

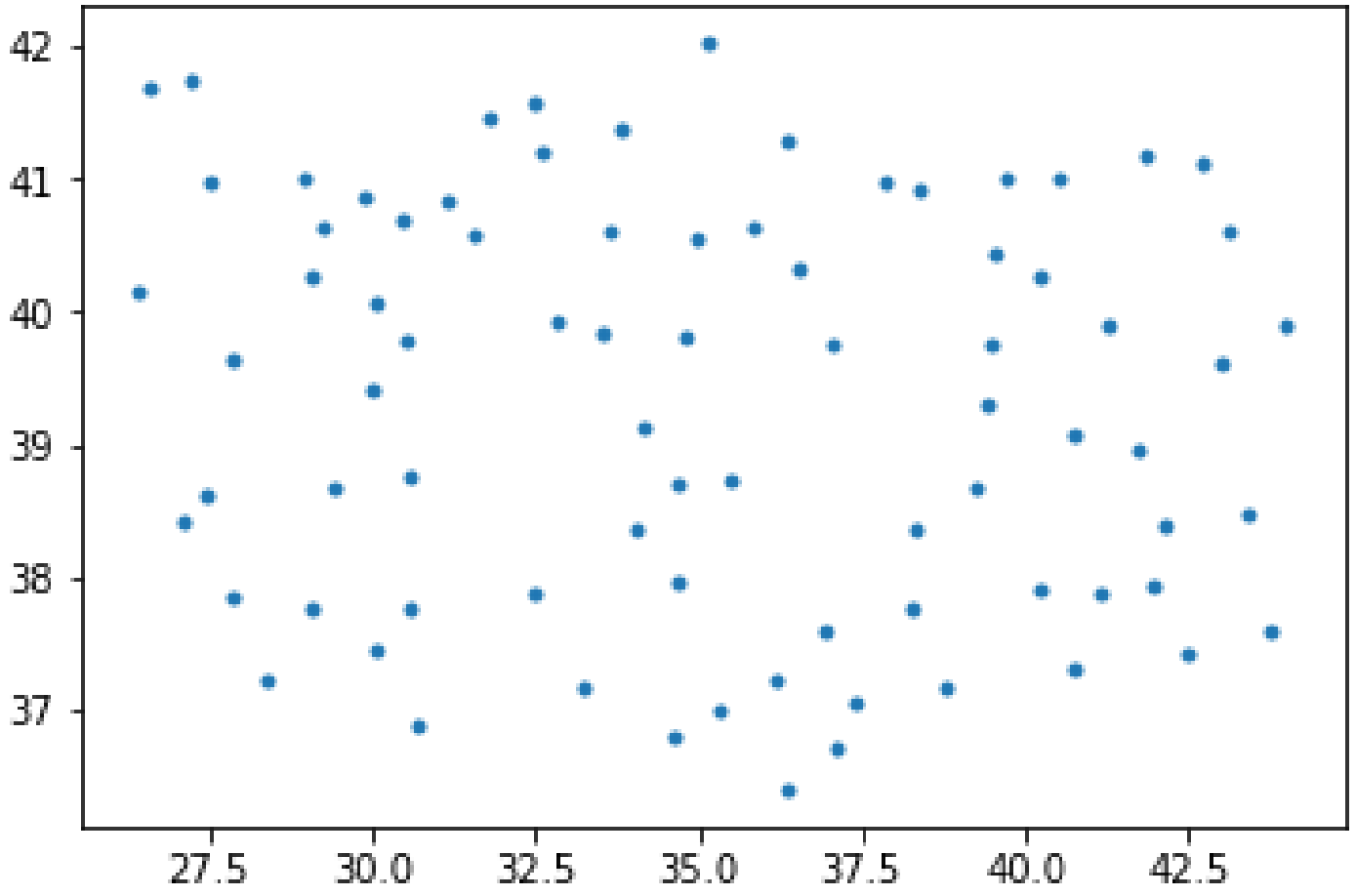



```
best path is=  16043.0
```

Figure 7- SA Output

COMPARISON:

Optimum distance that was calculated by Genetic Algorithm (GA) was found as 15047 km and optimum distance that was calculated by Simulated Annealing (SA) was found as 16043 km. Hence from the code that I wrote, it can be said that Genetic algorithm gave more optimum path. But the solution time of the SA algorithm were shorter. Simulated Annealing algorithm has the advantage of jumping out of the local optimum and Genetic Algorithm has advantage of mutation, these features gives to GA and SA flexibility to look from bigger perspective and not to stuck on a local region.


42
41
40
39
38
37
27.5
30.0
32.5
35.0
37.5
40.0
42.5

APPENDICES

GENETIC ALGORITHM (GA)

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 22 20:36:57 2019

@author: S2B
"""

import numpy as np
import matplotlib.pyplot as plt

def get_coordinates():

    call = np.genfromtxt('Coordinates.csv', dtype='str',delimiter=',', encoding='utf-8').T

    cities = call[0]
    x = call[1].astype(float)
    y = call[2].astype(float)
    points=np.vstack((x,y))
    return points, x, y,cities

def get_path():

  f_c = np.where(cities=='Ankara') # first city
  n = points.shape[1]
  l = list(range(n))
  l=np.delete(l,f_c)
  np.random.shuffle(l)
  l=np.insert(l,0,f_c)
  return l


def get_path_length(path):

    call = np.genfromtxt('distancematrix.csv', dtype='str',delimiter=',', encoding='utf-8')

    def distance(i,j):
        return float(call[i+1, j+1])

    total_length = 0
```

```
    for i,j in zip(path[:-1],path[1:]):
        d=distance(i,j)
        total_length = total_length + d
    return total_length

def draw_path(path):
  path = np.append(path,path[0])
  plt.plot(y[path],x[path],'-o')

def cross_over(gene1,gene2, mutation=0.5):

  r = np.random.randint(len(gene1)) # cross over location
  newgene = np.append(gene1[:r],gene2[r:]) # may be a defunct gene
  missing = set(gene1)-set(newgene)
  elements, count = np.unique(newgene, return_counts=True)
  duplicates = elements[count==2]
  duplicate_indices=(newgene[:, None] == duplicates).argmax(axis=0)
  newgene[duplicate_indices]=list(missing) # now proper.

  if np.random.rand()<mutation:
    i1,i2 = np.random.randint(0,len(newgene),2)
    newgene[[i1,i2]] = newgene[[i2,i1]]
  return newgene

def create_initial_population(m):
  population = []
  fitness = []
  for i in range(m):
    gene = get_path()
    path_length = get_path_length(gene)
    population.append(gene)
    fitness.append(path_length)

  population = np.array(population)
  fitness = np.array(fitness)
  sortedindex = np.argsort(fitness)
  return population[sortedindex], fitness[sortedindex]

def next_generation(population):
  pop = []
  fit = []
  i=0
  f=int(np.sqrt(len(population)))
  for gene1 in population[:f]:
```

```
    for gene2 in population[:f]:
      i=i+1
      x =  cross_over(gene1,gene2)
      l = get_path_length(x)
      pop.append(x)
      fit.append(l)

  population = np.array(pop)
  fitness = np.array(fit)
  sortedindex = np.argsort(fitness)
  return population[sortedindex], fitness[sortedindex]

n_population=300
points,x,y,cities=get_coordinates()
population, fitness  = create_initial_population(n_population)

for i in range(10000):
  population, fitness=next_generation(population)
      #print(fitness.min(),fitness.mean())
      print('iteration', i+1, 'distance is: ', fitness.min())
      best_path = population[0]
      draw_path(best_path)
      plt.show()
      plt.plot(fitness)
      plt.show()
```

SIMULATED ANNELAING (SA)

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 22 15:39:46 2019

@author: Murat Batuhan Günaydın
"""

import random, numpy as np, math
import matplotlib.pyplot as plt
import copy as cp

call = np.genfromtxt('Coordinates.csv', dtype='str',delimiter=',', encoding='utf-8').T
names=call[0]
x = call[1].astype(float)
```

```
y = call[2].astype(float)

def get_path_length(apath):
    # this function returns the total distance of a path
    call = np.genfromtxt('distancematrix.csv', dtype='str',delimiter=',', encoding='utf-8')

    call[call==''] = 0
    def distance(i,j):
        return float(call[i+1, j+1])

    total = 0
    for i,j in zip(apath[:-1],apath[1:]):
        d=distance(i,j)
        total = total + d
    return total

city = np.vstack((x, y)).T
cn=len(city)

xmin = min(pair[0] for pair in city)
aa= [pair[0] for pair in city]
xmax = max(pair[0] for pair in city)
ymin = min(pair[1] for pair in city)
ymax = max(pair[1] for pair in city)

def transform(pair):
    x = pair[0]
    y = pair[1]
    return [(x-xmin)*100/(xmax - xmin), (y-ymin)*100/(ymax - ymin)]


fc=np.where(names=='Ankara')

city = [ transform(b) for b in city]
path = random.sample(range(cn),cn);
path=np.delete(path,fc)
path=np.insert(np.asarray(path),0,fc)
path=path.tolist()

for temperature in np.logspace(0,5,num=100000)[::-1]:
    [i,j] = sorted(random.sample(range(cn),2));

    newpath =  path[:i] + path[j:j+1] +  path[i+1:j] + path[i:i+1] + path[j+1:];

```

```
58.     if math.exp( ( sum([ math.sqrt(sum([(city[path[(k+1) % cn]][d] - city[path[k % cn]]
59.     [d])**2 for d in [0,1] ])) for k in [j,j-1,i,i-1]]) - sum([math.sqrt(sum([(city[newpath[
        (k+1) % cn]][d]
60.     - city[newpath[k % cn]][d])**2 for d in [0,1] ])) for k in [j,j-1,i,i-1]])) / temperature
        ) > random.random():
61.         path = cp.copy(newpath);
62.
63. path=np.delete(path,fc)
64. path=np.insert(np.asarray(path),0,fc)
65. path=path.tolist()
66. distance=get_path_length(path)
67.
68. plt.plot([city[path[i % cn]][0] for i in range(cn+1)], [city[path[i % cn]][1] for i in rang
        e(cn+1)], 'o-');
69. plt.show()
70. print('best path is= ',distance)
```