Koushik Sahu
118CS0597
Soft Computing Lab – VI

Code:

```
import numpy as np
import random
import operator
import pandas as pd
import matplotlib.pyplot as plt

class City:
   def __init__(self, x, y):
       self.x = x
       self.y = y
   def distance(self, city):
       x_dis = abs(self.x - city.x)
       y_dis = abs(self.y - city.y)
       distance = np.sqrt((x_dis ** 2) + (y_dis ** 2))
       return distance

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

def createRoute(cityList):
   route = random.sample(cityList, len(cityList))
   return route

def initialPopulation(popSize, cityList):
   population = []
   for i in range(0, popSize):
       population.append(createRoute(cityList))
   return population
```

```
def rankRoutes(population):
   fitnessResults = {}
   for i in range(0,len(population)):
       fitnessResults[i] = Fitness(population[i]).routeFitness()
   return sorted(fitnessResults.items(), key =
operator.itemgetter(1), reverse = True)

def selection(popRanked, eliteSize):
   selectionResults = []
   df = pd.DataFrame(np.array(popRanked),
   columns=["Index","Fitness"])
   df['cum_sum'] = df.Fitness.cumsum()
   df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()
   for i in range(0, eliteSize):
       selectionResults.append(popRanked[i][0])
   for i in range(0, len(popRanked) - eliteSize):
       pick = 100*random.random()
   for i in range(0, len(popRanked)):
       if pick <= df.iat[i,3]:
           selectionResults.append(popRanked[i][0])
           break
   return selectionResults

def matingPool(population, selectionResults):
   matingpool = []
   for i in range(0, len(selectionResults)):
       index = selectionResults[i]
       matingpool.append(population[index])
   return matingpool

def breed(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

def breedPopulation(matingpool, eliteSize):
   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 = breed(pool[i], pool[len(matingpool)-i-1])
   children.append(child)
   return children
```

```
def mutate(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

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

def nextGeneration(currentGen, eliteSize, mutationRate):
    popRanked = rankRoutes(currentGen)
    selectionResults = selection(popRanked, eliteSize)
    matingpool = matingPool(currentGen, selectionResults)
    children = breedPopulation(matingpool, eliteSize)
    nextGeneration = mutatePopulation(children, mutationRate)
    return nextGeneration

def geneticAlgorithm(population, cityCoordinateList, popSize,
eliteSize, mutationRate, generations):
    for index, coord in enumerate(cityCoordinateList):
        print(f"City {index}: {coord}")
    pop = initialPopulation(popSize, population)
    print("\nInitial distance: " + str(1 /
    rankRoutes(pop)[0][1]))
    for i in range(0, generations):
        pop = nextGeneration(pop, eliteSize, mutationRate)
    print("Final distance : " + str(1 / rankRoutes(pop)[0][1]))
    bestRouteIndex = rankRoutes(pop)[0][0]
    bestRouteCityList = pop[bestRouteIndex]
    bestRoute = []
    for city in bestRouteCityList:
        bestRoute.append(cityCoordinateList.index((city.x,
city.y)))
    print("")
    for i in range(len(bestRoute)):
        print(f"City {bestRoute[i]} -> ", end="")
        print(f"City {bestRoute[0]}")
    return bestRoute
cityList = []
cityCoordinateList = []
n = 10
for i in range(0, n):
    X, Y = int(random.random() * 200), int(random.random() * 200)
    cityCoordinateList.append((X, Y))
    cityList.append(City(x=X, y=Y))
```

```
bestRouteCityList =
geneticAlgorithm(population=cityList,cityCoordinateList=cityCoordi
nateList, popSize=100, eliteSize=20,
mutationRate=0.01,generations=100)
```

Output:

```
City 0: (26, 64)
City 1: (123, 0)
City 2: (172, 105)
City 3: (186, 8)
City 4: (63, 160)
City 5: (9, 118)
City 6: (24, 186)
City 7: (179, 45)
City 8: (14, 45)
City 9: (69, 53)

Initial distance: 829.5280211986102
Final distance  : 610.4936850550035

City 2 -> City 7 -> City 3 -> City 1 -> City 9 -> City 8 -> City 0 -> City 5 -> City 6 -
> City 4 -> City 2
```