OBJECTIVE:To implement travelling salesman problem using Genetics Algorithm-

import math import random

class City:

def \_\_init\_\_(self, x=None, y=None): self.x = None self.y = None if x is not None:

self.x = x else:

self.x = int(random.random() \* 200)

if y is not None:

self.y = y else: self.y = int(random.random() \* 200)

def getX(self):

return self.x

def getY(self):

return self.y

def distanceTo(self, city): xDistance = abs(self.getX() - city.getX()) yDistance = abs(self.getY() - city.getY())

distance=math.sqrt((xDistance\*xDistance)+(yDistance\*yDistance) ) return distance

def \_\_repr\_\_(self): return str(self.getX()) + ", " + str(self.getY())

class TourManager: destinationCities = []

def addCity(self, city): self.destinationCities.append(city)

def getCity(self, index): return self.destinationCities[index]

def numberOfCities(self): return len(self.destinationCities)

class Tour:

def \_\_init\_\_(self, tourmanager, tour=None): self.tourmanager = tourmanager

self.tour = [] self.fitness = 0.0 self.distance = 0 if tour is not None:

self.tour = tour

else:

for i in range(0, self.tourmanager.numberOfCities()): self.tour.append(None)

def \_\_len\_\_(self): return len(self.tour)

def \_\_getitem\_\_(self, index):

return self.tour[index]

def \_\_setitem\_\_(self, key, value): self.tour[key] = value

def \_\_repr\_\_(self): geneString = "|" for i in range(0, self.tourSize()):

geneString += str(self.getCity(i)) + "|"

return geneString

def generateIndividual(self):

for cityIndex in range(0, self.tourmanager.numberOfCities()):

self.setCity(cityIndex, self.tourmanager.getCity(cityIndex))

random.shuffle(self.tour)

def getCity(self, tourPosition):

return self.tour[tourPosition]

def setCity(self, tourPosition, city): self.tour[tourPosition] = city self.fitness = 0.0 self.distance = 0

def getFitness(self):

if self.fitness == 0:

self.fitness = 1/float(self.getDistance())

return self.fitness

def getDistance(self): if self.distance == 0: tourDistance = 0 for cityIndex in range(0, self.tourSize()):

fromCity = self.getCity(cityIndex) destinationCity = None if cityIndex+1 < self.tourSize():

destinationCity = self.getCity(cityIndex+1)

else: destinationCity = self.getCity(0)

tourDistance += fromCity.distanceTo(destinationCity)

self.distance = tourDistance

return self.distance

def tourSize(self): return len(self.tour)

def containsCity(self, city):

return city in self.tour

class Population:

def \_\_init\_\_(self, tourmanager, populationSize, initialise): self.tours = [] for i in range(0, populationSize): self.tours.append(None)

if initialise:

for i in range(0, populationSize): newTour = Tour(tourmanager) newTour.generateIndividual() self.saveTour(i, newTour)

def \_\_setitem\_\_(self, key, value): self.tours[key] = value

def \_\_getitem\_\_(self, index):

return self.tours[index]

def saveTour(self, index, tour): self.tours[index] = tour

def getTour(self, index):

return self.tours[index]

def getFittest(self):

fittest = self.tours[0] for i in range(0, self.populationSize()):

if fittest.getFitness() <= self.getTour(i).getFitness(): fittest = self.getTour(i)

return fittest

def populationSize(self): return len(self.tours)

class GA:

def \_\_init\_\_(self, tourmanager): self.tourmanager = tourmanager self.mutationRate = 0.015 self.tournamentSize = 5 self.elitism = True

def evolvePopulation(self, pop): newPopulation = Population(self.tourmanager, pop.populationSize(),

False) elitismOffset = 0 if self.elitism: newPopulation.saveTour(0, pop.getFittest()) elitismOffset = 1

for i in range(elitismOffset, newPopulation.populationSize()): parent1 = self.tournamentSelection(pop)

parent2 = self.tournamentSelection(pop) child = self.crossover(parent1, parent2) newPopulation.saveTour(i, child)

for i in range(elitismOffset, newPopulation.populationSize()): self.mutate(newPopulation.getTour(i)) return newPopulation

def crossover(self, parent1, parent2): child = Tour(self.tourmanager)

startPos = int(random.random() \* parent1.tourSize()) endPos = int(random.random() \* parent1.tourSize())

for i in range(0, child.tourSize()):

if startPos < endPos and i > startPos and i < endPos: child.setCity(i, parent1.getCity(i))

elif startPos > endPos:

if not (i < startPos and i > endPos):

child.setCity(i, parent1.getCity(i))

for i in range(0, parent2.tourSize()):

if not child.containsCity(parent2.getCity(i)):

for ii in range(0, child.tourSize()):

if child.getCity(ii) == None:

child.setCity(ii, parent2.getCity(i)) break

return child

def mutate(self, tour):

for tourPos1 in range(0, tour.tourSize()):

if random.random() < self.mutationRate: tourPos2 = int(tour.tourSize() \* random.random())

city1 = tour.getCity(tourPos1) city2 = tour.getCity(tourPos2)

tour.setCity(tourPos2, city1) tour.setCity(tourPos1, city2)

def tournamentSelection(self, pop):

tournament = Population(self.tourmanager, self.tournamentSize,

False) for i in range(0, self.tournamentSize):

randomId = int(random.random() \* pop.populationSize()) tournament.saveTour(i, pop.getTour(randomId))

fittest = tournament.getFittest() return fittest

if \_\_name\_\_ == '\_\_main\_\_': tourmanager = TourManager()

# Create and add our cities city = City(60, 200) tourmanager.addCity(city) city2 = City(180, 200) tourmanager.addCity(city2) city3 = City(80, 180) tourmanager.addCity(city3) city4 = City(140, 180) tourmanager.addCity(city4) city5 = City(20, 160) tourmanager.addCity(city5) city6 = City(100, 160) tourmanager.addCity(city6) city7 = City(200, 160) tourmanager.addCity(city7) city8 = City(140, 140)

tourmanager.addCity(city8) city9 = City(40, 120) tourmanager.addCity(city9) city10 = City(100, 120) tourmanager.addCity(city10) city11 = City(180, 100) tourmanager.addCity(city11) city12 = City(60, 80) tourmanager.addCity(city12) city13 = City(120, 80) tourmanager.addCity(city13) city14 = City(180, 60)

tourmanager.addCity(city14) city15 = City(20, 40) tourmanager.addCity(city15) city16 = City(100, 40) tourmanager.addCity(city16) city17 = City(200, 40)

tourmanager.addCity(city17) city18 = City(20, 20) tourmanager.addCity(city18) city19 = City(60, 20) tourmanager.addCity(city19) city20 = City(160, 20) tourmanager.addCity(city20)

# Initialize population pop = Population(tourmanager, 50, True);

print "Initial distance: " + str(pop.getFittest().getDistance())

# Evolve population for 50 generations ga = GA(tourmanager) pop = ga.evolvePopulation(pop) for i in range(0, 100):

pop = ga.evolvePopulation(pop)

# Print final results print "Finished"

print "Final distance: " + str(pop.getFittest().getDistance()) print "Solution:" print pop.getFittest()



