**Traveling Salesman Problem using Genetic Algorithm**

The code I made is mine. I have made the “Genetic Algorithm” code completely from scratch. Though I took two functions from other sources. They are the function to count the total distance and the function to draw the graph using matplotlib. The code is not working perfectly as it seems to be completely random. It does get better when it finds a better child, but it does not improve on every generation.

To run it, first you must determine how many cities will be put into this code. Then change the cityAmount on line 35 according to how many cities will be used for the code. After that, run the code and it should output nothing yet. It is now asking for the city index and their coordinates using the format of (‘index’ ‘x’ ‘y’) separated by spaces. After inputting the cities, it should run the calculations for the minimum path cost and the path that generates that amount of cost. It will print both the original cost, the calculated cost, and the path for the calculated cost.

I implemented three ways to make new children from the picked parents for this code. One is the mutation where it takes a part of a parent and randomizes it. The second one is a crossover where it mixes two parents together by taking a part of one parent and giving it to the other parent and vice versa. And the third one is a completely randomized one to give the generation something new to use. These are the only Genetic Algorithm methods I implemented as I wrote the code from scratch, and I also had no prior python experience by the time I wrote the code.

The code prints out the best cost every ten generations just to show that it is doing something instead of being stuck. And in the end, it will print out the initial path cost, the calculated minimum path cost, and its path. It will then use matplotlib to draw the graph.

**List of other Sources:**

* Function for finding distance between cities (<https://www.geeksforgeeks.org/program-calculate-distance-two-points/>)
* Function for matplotlib (<https://github.com/hassanzadehmahdi/Traveling-Salesman-Problem-using-Genetic-Algorithm>)
* Reference for GA Methods (<https://www.youtube.com/watch?v=3GAfjE_ChRI>)

1. import random

2. import math

3. import matplotlib.pyplot as plt

4.

5. # calculate distance between two cities

6. def distance(x1, y1, x2, y2):

7.     return (((x2 - x1)\*\*2 +(y2 - y1)\*\*2)\*\*0.5)

8.

9. def totalDistance(cCounted):

10.     cost = 0.0

11.     for i in range(len(cCounted) - 1):

12.         cost += distance(cCounted[i][1], cCounted[i][2], cCounted[i+1][1], cCounted[i+1][2])

13.     return cost

14.

15. def drawMap(city, answer):

16.     for j in city:

17.         plt.plot(j[1], j[2], "ro")

18.         plt.annotate(j[0], (j[1], j[2]))

19.

20.     for i in range(len(answer[1])):

21.         try:

22.             first = answer[1][i]

23.             secend = answer[1][i + 1]

24.

25.             plt.plot([first[1], secend[1]], [first[2], secend[2]], "black")

26.         except:

27.             continue

28.

29.     first = answer[1][0]

30.     secend = answer[1][-1]

31.     plt.plot([first[1], secend[1]], [first[2], secend[2]], "black")

32.

33.     plt.show()

34.

35. cityAmount = 100

36. populationSize = 100

37. generationAmount = 1000

38.

39. city = []

40.

41. # inputting city data into the list

42. for i in range(0, cityAmount):

43.     n = input()

44.     split = n.split()

45.     city.append([int(split[0]), float(split[1]), float(split[2])])

46.

47. chromosome = []

48.

49. for i in range(0, populationSize):

50.     copy = city.copy()

51.     random.shuffle(copy)

52.     copy.append(copy[0])

53.     cost = totalDistance(copy)

54.     chromosome.append([cost, copy])

55.

56. chromosome.sort(key = lambda x: x[0])

57. original = chromosome[0][0]

58.

59. print("Initial Best Distance: " + str(original))

60.

61. # "Genetic Algorithm"

62. for i in range(0, generationAmount):

63.     dis = 0

64.     newPop = []

65.     p1 = chromosome[0][1]

66.     for loc in chromosome:

67.         if loc[1] != p1:

68.             p2 = chromosome[1][1]

69.             break

70.

71.     if p1 not in newPop:

72.         newPop.append([chromosome[0][0], chromosome[0][1]])

73.

74.     chromosome = []

75.     for j in range(0, int(populationSize/4)):

76.         randomizer = []

77.         while n:

78.             tempChild = []

79.             randomizer = []

80.             while len(randomizer) < int(len(p1)/2):

81.                 randomNumber = random.randint(1, len(p1)-1)

82.                 if(randomNumber not in randomizer):

83.                     randomizer.append(randomNumber)

84.

85.             for k in p1:

86.                 if k[0] in randomizer:

87.                     tempChild.append(k)

88.             for k in p2:

89.                 if k not in tempChild:

90.                     tempChild.append(k)

91.             tempChild.append(tempChild[0])

92.

93.             dis = totalDistance(tempChild)

94.             if(tempChild not in newPop):

95.                 newPop.append([dis, tempChild])

96.                 break

97.

98.

99.         while n:

100.             tempChild = []

101.             randomizer = []

102.

103.             while len(randomizer) < int(len(p1)/2):

104.                 randomNumber = random.randint(1, len(p1)-1)

105.                 if(randomNumber not in randomizer):

106.                     randomizer.append(randomNumber)

107.

108.             for k in p2:

109.                 if k[0] in randomizer:

110.                     tempChild.append(k)

111.             for k in p1:

112.                 if k not in tempChild:

113.                     tempChild.append(k)

114.             tempChild.append(tempChild[0])

115.

116.             dis = totalDistance(tempChild)

117.             if(tempChild not in newPop):

118.                 newPop.append([dis, tempChild])

119.                 break

120.

121.         tempChild = []

122.         randomizer = []

123.

124.         # Mix parent1 into parent2

125.         while n:

126.             tempChild = []

127.             for k in range(0, int(len(p1)/2)):

128.                 tempChild.append(p1[k])

129.             random.shuffle(tempChild)

130.             for k in p2:

131.                 if k not in tempChild:

132.                     tempChild.append(k)

133.             tempChild.append(tempChild[0])

134.             dis = totalDistance(tempChild)

135.             if tempChild not in newPop:

136.                 newPop.append([dis, tempChild])

137.                 break

138.

139.         tempChild = []

140.

141.         while n:

142.             tempChild = []

143.             # Mix parent2 into parent1

144.             for k in range(0, int(len(p2)/2)):

145.                 tempChild.append(p2[k])

146.             random.shuffle(tempChild)

147.             for k in p1:

148.                 if k not in tempChild:

149.                     tempChild.append(k)

150.             tempChild.append(tempChild[0])

151.             dis = totalDistance(tempChild)

152.             if tempChild not in newPop:

153.                 newPop.append([dis, tempChild])

154.                 break

155.

156.         tempChild = []

157.

158.     # # ----------------Complete Randomizer---------------

159.     for j in range(0, math.ceil(populationSize / 10)):

160.         cp1 = p1.copy()

161.         cp1.pop()

162.         random.shuffle(cp1)

163.         cp1.append(cp1[0])

164.         dis = totalDistance(cp1)

165.         if cp1 not in newPop:

166.             newPop.append([dis,cp1])

167.             break

168.     newPop.sort(key = lambda x : x[0])

169.     chromosome = newPop.copy()

170.     newPop = []

171.     if(i % 10 == 0):

172.         print("Current best distance of Gen " + str(i) + ": " + str(chromosome[0][0]))

173.

174. print("\n\nBest cost of initial population: " + str(original))

175. print("\n\nlo and behold, The final result after training is: ")

176. print(chromosome[0][0])

177. path = []

178. for i in range(0, int(cityAmount+1)):

179.     path.append(chromosome[0][1][i][0])

180. print("\n\nwith the path of: \n")

181. print(path)

182. drawMap(city, chromosome[0])