**计算智能 作业2**

2021年10月07日

**要求：**

1. 在文档中说明解题思路、方法实现、求解结果等，必要时需要对结果进行分析和讨论。
2. 将源代码作为附录粘贴于作业文档末尾，并同时作为提交的附件与作业文档一起放在压缩包内备查。
3. 独立完成，严禁抄袭。

**内容：**

1. 编写程序求解下列问题：

旅行商问题（Travelling Salesman Problem, 简记TSP，亦称货郎担问题)：设有n个城市和距离矩阵D=[dij]，其中dij表示城市i到城市j的距离，i，j=1，2 … n，则问题是要找出遍访每个城市恰好一次的一条回路并使其路径长度为最短。说明：

1) 回路：从某个城市出发，最后回到这个城市。

2) 编程语言不限（需提交源代码和运行结果截图）。

3) 作业相关数据说明（15个城市，最短路径距离291）：

a) p01.tsp, the TSP specification of the data.

b) p01\_d.txt, the intercity distance table.

c) p01\_s.txt, an itinerary that minimizes the total distance.

d) p01\_sxy.txt, the XY coordinates of the minimal itinerary.

e) p01\_sxy.png, an image of the minimal itinerary.

f) p01\_xy.txt, a set of XY coordinates for the cities, inferred from the distances.

g) p01\_xy.png, an image of the XY coordinates

3) 截止时间：**10月22日 0:00**。

问题解答：

代码使用了遗传算法来解决问题

代码的输入只采用了各个地点的坐标文件

f) p01\_xy.txt, a set of XY coordinates for the cities, inferred from the distances.

通过坐标我们可以计算得到距离：

coordinates = np.array([[-0.0000000400893815, 0.0000000358808126],  
 [-28.8732862244731230, -0.0000008724121069],  
 [-79.2915791686897506, 21.4033307581457670],  
 [-14.6577381710829471, 43.3895496964974043],  
 [-64.7472605264735108, -21.8981713360336698],  
 [-29.0584693142401171, 43.2167287683090606],  
 [-72.0785319657452987, -0.1815834632498404],  
 [-36.0366489745023770, 21.6135482886620949],  
 [-50.4808382862985496, -7.3744722432402208],  
 [-50.5859026832315024, 21.5881966132975371],  
 [-0.1358203773809326, 28.7292896751977480],  
 [-65.0865638413727368, 36.0624693073746769],  
 [-21.4983260706612533, -7.3194159498090388],  
 [-57.5687244704708050, 43.2505562436354225],  
 [-43.0700258454450875, -14.5548396888330487]])  
  
  
# 得到距离矩阵的函数  
def get\_distance\_matrix(coordinates):  
 num = coordinates.shape[0] # 15个坐标点  
 distmat = np.zeros((15, 15)) # 15X15距离矩阵  
 for i in range(num):  
 for j in range(i, num):  
 distmat[i][j] = distmat[j][i] = round(np.linalg.norm(coordinates[i] - coordinates[j]))  
 return distmat

设置了温度的衰减系数alpha = 0.99，初始温度为100，温度达到0时候停止，同时，设置了最大迭代次数epoch = 1000。

在每一轮遗传算法的迭代中，随机交换除了起始点之外的路线中的两点。

while True: # 产生两个不同的随机数  
 loc1 = int(np.ceil(np.random.rand() \* (num - 1)))  
 loc2 = int(np.ceil(np.random.rand() \* (num - 1)))  
  
 if loc1 != loc2:  
 break  
new\_solution[loc1], new\_solution[loc2] = new\_solution[loc2], new\_solution[loc1]

如果结果更优，则更新迭代，如果结果欠佳，则一定概率接收该解。

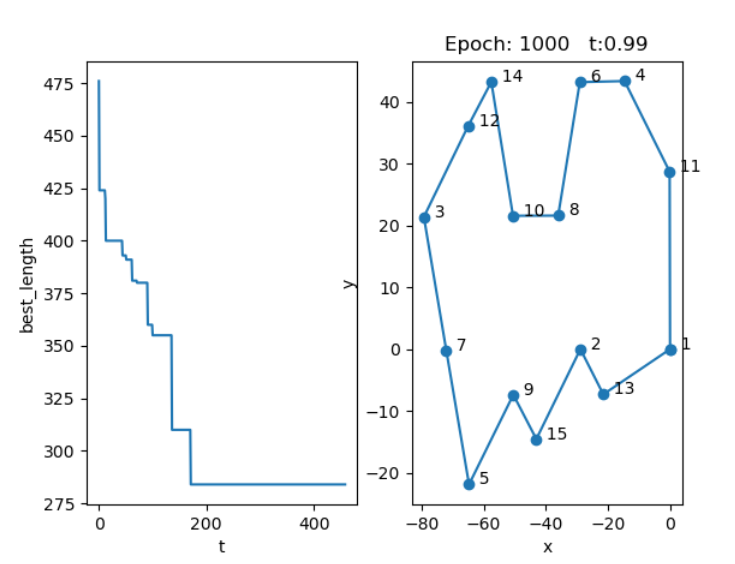
if new\_length < current\_length: # 接受该解  
 # 更新current\_solution  
 current\_length = new\_length  
 current\_solution = new\_solution.copy()  
 # 更新 best\_solution  
 if new\_length < best\_length:  
 best\_length = new\_length  
 best\_solution = new\_solution.copy()  
else: # 按一定的概率接受该解  
 if np.random.rand() < np.exp(-(new\_length - current\_length) / t):  
 current\_length = new\_length  
 current\_solution = new\_solution.copy()  
 else:  
 new\_solution = current\_solution.copy()

代码执行结果：

最优路线：

[ 1 13 2 15 9 5 7 3 12 14 10 8 6 4 11 1]

可视化：



附录：

import numpy as np  
import matplotlib.pyplot as plt  
  
# 旅行商问题 ( TSP , Traveling Salesman Problem )  
coordinates = np.array([[-0.0000000400893815, 0.0000000358808126],  
 [-28.8732862244731230, -0.0000008724121069],  
 [-79.2915791686897506, 21.4033307581457670],  
 [-14.6577381710829471, 43.3895496964974043],  
 [-64.7472605264735108, -21.8981713360336698],  
 [-29.0584693142401171, 43.2167287683090606],  
 [-72.0785319657452987, -0.1815834632498404],  
 [-36.0366489745023770, 21.6135482886620949],  
 [-50.4808382862985496, -7.3744722432402208],  
 [-50.5859026832315024, 21.5881966132975371],  
 [-0.1358203773809326, 28.7292896751977480],  
 [-65.0865638413727368, 36.0624693073746769],  
 [-21.4983260706612533, -7.3194159498090388],  
 [-57.5687244704708050, 43.2505562436354225],  
 [-43.0700258454450875, -14.5548396888330487]])  
  
  
# 得到距离矩阵的函数  
def get\_distance\_matrix(coordinates):  
 num = coordinates.shape[0] # 15个坐标点  
 distmat = np.zeros((15, 15)) # 15X15距离矩阵  
 for i in range(num):  
 for j in range(i, num):  
 distmat[i][j] = distmat[j][i] = round(np.linalg.norm(coordinates[i] - coordinates[j]))  
 return distmat  
  
  
def init\_parameter():  
 alpha = 0.99  
 t = (1, 100)  
 epoch = 1000  
 return alpha, t, epoch  
  
  
num = coordinates.shape[0]  
distmat = get\_distance\_matrix(coordinates) # 得到距离矩阵  
  
new\_solution = np.arange(num)  
  
PATH\_MAX\_LENGTH = 999999  
current\_solution = new\_solution.copy()  
current\_length = PATH\_MAX\_LENGTH  
  
best\_solution = new\_solution.copy()  
best\_length = PATH\_MAX\_LENGTH  
  
alpha, t2, epoch = init\_parameter()  
t = t2[1]  
  
result = [] # 记录迭代过程中的最优解  
while t > t2[0]:  
 for i in np.arange(epoch):  
 while True: # 产生两个不同的随机数  
 loc1 = int(np.ceil(np.random.rand() \* (num - 1)))  
 loc2 = int(np.ceil(np.random.rand() \* (num - 1)))  
  
 if loc1 != loc2:  
 break  
 new\_solution[loc1], new\_solution[loc2] = new\_solution[loc2], new\_solution[loc1]  
  
 new\_length = 0  
 for i in range(num - 1):  
 new\_length += distmat[new\_solution[i]][new\_solution[i + 1]]  
 new\_length += distmat[new\_solution[14]][new\_solution[0]]  
 if new\_length < current\_length: # 接受该解  
 # 更新current\_solution  
 current\_length = new\_length  
 current\_solution = new\_solution.copy()  
 # 更新 best\_solution  
 if new\_length < best\_length:  
 best\_length = new\_length  
 best\_solution = new\_solution.copy()  
 else: # 按一定的概率接受该解  
 if np.random.rand() < np.exp(-(new\_length - current\_length) / t):  
 current\_length = new\_length  
 current\_solution = new\_solution.copy()  
 else:  
 new\_solution = current\_solution.copy()  
 t = alpha \* t  
 result.append(best\_length)  
# 用来显示结果  
solution = np.append(best\_solution, 0)  
solution = solution + np.ones(solution.shape)  
solution = solution.astype(int)  
print(solution)  
  
plt.subplot(1, 2, 1)  
plt.plot(np.array(result))  
plt.ylabel("best\_length")  
plt.xlabel("t")  
plt.subplot(1, 2, 2)  
x, y = [], []  
for i in np.nditer(solution):  
 x.append(coordinates[i - 1][0])  
 y.append(coordinates[i - 1][1])  
plt.scatter(x, y)  
plt.plot(x, y)  
plt.title("Epoch: {} t:{:.2f}".format(epoch, t))  
for i in range(len(coordinates)):  
 plt.text(coordinates[i][0], coordinates[i][1], r' ' + str(i + 1))  
plt.xlabel('x')  
plt.ylabel('y')  
plt.show()