*# -\*- coding: utf-8 -\*-*

*"""*

*Created on Wed Jun 16 15:21:03 2018*

*@author: SYSTEM*

*"""*

**import** os

os.getcwd()

*#返回当前工作目录*

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

*# % pylab*

*#初始化城市坐标，总共52个城市*

coordinates = np.array([[565.0, 575.0], [25.0, 185.0], [345.0, 750.0], [945.0, 685.0], [845.0, 655.0],

[880.0, 660.0], [25.0, 230.0], [525.0, 1000.0], [580.0, 1175.0], [650.0, 1130.0],

[1605.0, 620.0], [1220.0, 580.0], [1465.0, 200.0], [1530.0, 5.0], [845.0, 680.0],

[725.0, 370.0], [145.0, 665.0], [415.0, 635.0], [510.0, 875.0], [560.0, 365.0],

[300.0, 465.0], [520.0, 585.0], [480.0, 415.0], [835.0, 625.0], [975.0, 580.0],

[1215.0, 245.0], [1320.0, 315.0], [1250.0, 400.0], [660.0, 180.0], [410.0, 250.0],

[420.0, 555.0], [575.0, 665.0], [1150.0, 1160.0], [700.0, 580.0], [685.0, 595.0],

[685.0, 610.0], [770.0, 610.0], [795.0, 645.0], [720.0, 635.0], [760.0, 650.0],

[475.0, 960.0], [95.0, 260.0], [875.0, 920.0], [700.0, 500.0], [555.0, 815.0],

[830.0, 485.0], [1170.0, 65.0], [830.0, 610.0], [605.0, 625.0], [595.0, 360.0],

[1340.0, 725.0], [1740.0, 245.0]])

*#计算52个城市间的欧式距离*

**def** getdistmat(coordinates):

num = coordinates.shape[0]

distmat = np.zeros((52, 52))

*# 初始化生成52\*52的矩阵*

**for** i **in** range(num):

**for** j **in** range(i, num):

distmat[i][j] = distmat[j][i] = np.linalg.norm(coordinates[i] - coordinates[j])

**return** distmat

*#返回城市距离矩阵*

distmat = getdistmat(coordinates)

numant = 60 *# 蚂蚁个数*

numcity = coordinates.shape[0]

*# shape[0]=52 城市个数,也就是任务个数*

alpha = 1 *# 信息素重要程度因子*

beta = 5 *# 启发函数重要程度因子*

rho = 0.1 *# 信息素的挥发速度*

Q = 1 *# 完成率*

iter = 0 *#迭代初始*

itermax = 150 *#迭代总数*

etatable = 1.0 / (distmat + np.diag([1e10] \* numcity))

*#diag(),将一维数组转化为方阵 启发函数矩阵，表示蚂蚁从城市i转移到城市j的期望程度*

pheromonetable = np.ones((numcity, numcity))

*# 信息素矩阵 52\*52*

pathtable = np.zeros((numant, numcity)).astype(int)

*# 路径记录表，转化成整型 40\*52*

distmat = getdistmat(coordinates)

*# 城市的距离矩阵 52\*52*

lengthaver = np.zeros(itermax) *# 迭代50次，存放每次迭代后，路径的平均长度 50\*1*

lengthbest = np.zeros(itermax) *# 迭代50次，存放每次迭代后，最佳路径长度 50\*1*

pathbest = np.zeros((itermax, numcity)) *# 迭代50次，存放每次迭代后，最佳路径城市的坐标 50\*52*

**while** iter < itermax:

*#迭代总数*

*#40个蚂蚁随机放置于52个城市中*

**if** numant <= numcity: *# 城市数比蚂蚁数多，不用管*

pathtable[:, 0] = np.random.permutation(range(numcity))[:numant]

*#返回一个打乱的40\*52矩阵，但是并不改变原来的数组,把这个数组的第一列(40个元素)放到路径表的第一列中*

*#矩阵的意思是哪个蚂蚁在哪个城市,矩阵元素不大于52*

**else**: *# 蚂蚁数比城市数多，需要有城市放多个蚂蚁*

pathtable[:numcity, 0] = np.random.permutation(range(numcity))[:]

*# 先放52个*

pathtable[numcity:, 0] = np.random.permutation(range(numcity))[:numant - numcity]

*# 再把剩下的放完*

*# print(pathtable[:,0])*

length = np.zeros(numant) *# 1\*40的数组*

*#本段程序算出每只/第i只蚂蚁转移到下一个城市的概率*

**for** i **in** range(numant):

*# i=0*

visiting = pathtable[i, 0] *# 当前所在的城市*

*# set()创建一个无序不重复元素集合*

*# visited = set() #已访问过的城市，防止重复*

*# visited.add(visiting) #增加元素*

unvisited = set(range(numcity))

*#未访问的城市集合*

*#剔除重复的元素*

unvisited.remove(visiting) *# 删除已经访问过的城市元素*

**for** j **in** range(1, numcity): *# 循环numcity-1次，访问剩余的所有numcity-1个城市*

*# j=1*

*# 每次用轮盘法选择下一个要访问的城市*

listunvisited = list(unvisited)

*#未访问城市数,list*

probtrans = np.zeros(len(listunvisited))

*#每次循环都初始化转移概率矩阵1\*52,1\*51,1\*50,1\*49....*

*#以下是计算转移概率*

**for** k **in** range(len(listunvisited)):

probtrans[k] = np.power(pheromonetable[visiting][listunvisited[k]], alpha) \

\* np.power(etatable[visiting][listunvisited[k]], alpha)

*#eta-从城市i到城市j的启发因子 这是概率公式的分母 其中[visiting][listunvis[k]]是从本城市到k城市的信息素*

cumsumprobtrans = (probtrans / sum(probtrans)).cumsum()

*#求出本只蚂蚁的转移到各个城市的概率斐波衲挈数列*

cumsumprobtrans -= np.random.rand()

*# 随机生成下个城市的转移概率，再用区间比较*

*# k = listunvisited[find(cumsumprobtrans > 0)[0]]*

k = listunvisited[list(cumsumprobtrans > 0).index(**True**)]

*# k = listunvisited[np.where(cumsumprobtrans > 0)[0]]*

*# where 函数选出符合cumsumprobtans>0的数*

*# 下一个要访问的城市*

pathtable[i, j] = k

*#采用禁忌表来记录蚂蚁i当前走过的第j城市的坐标，这里走了第j个城市.k是中间值*

unvisited.remove(k)

*# visited.add(k)*

*#将未访问城市列表中的K城市删去，增加到已访问城市列表中*

length[i] += distmat[visiting][k]

*#计算本城市到K城市的距离*

visiting = k

length[i] += distmat[visiting][pathtable[i, 0]]

*# 计算本只蚂蚁的总的路径距离，包括最后一个城市和第一个城市的距离*

*# print("ants all length:",length)*

*# 包含所有蚂蚁的一个迭代结束后，统计本次迭代的若干统计参数*

lengthaver[iter] = length.mean()

*#本轮的平均路径*

*#本部分是为了求出最佳路径*

**if** iter == 0:

lengthbest[iter] = length.min()

pathbest[iter] = pathtable[length.argmin()].copy()

*#如果是第一轮路径，则选择本轮最短的路径,并返回索引值下标，并将其记录*

**else**:

*#后面几轮的情况，更新最佳路径*

**if** length.min() > lengthbest[iter - 1]:

lengthbest[iter] = lengthbest[iter - 1]

pathbest[iter] = pathbest[iter - 1].copy()

*# 如果是第一轮路径，则选择本轮最短的路径,并返回索引值下标，并将其记录*

**else**:

lengthbest[iter] = length.min()

pathbest[iter] = pathtable[length.argmin()].copy()

*#此部分是为了更新信息素*

changepheromonetable = np.zeros((numcity, numcity))

**for** i **in** range(numant):*#更新所有的蚂蚁*

**for** j **in** range(numcity - 1):

changepheromonetable[pathtable[i, j]][pathtable[i, j + 1]] += Q / distmat[pathtable[i, j]][pathtable[i, j + 1]]

*#根据公式更新本只蚂蚁改变的城市间的信息素Q/d其中d是从第j个城市到第j+1个城市的距离*

changepheromonetable[pathtable[i, j + 1]][pathtable[i, 0]] += Q / distmat[pathtable[i, j + 1]][pathtable[i, 0]]

*#首城市到最后一个城市 所有蚂蚁改变的信息素总和*

*#信息素更新公式p=(1-挥发速率)\*现有信息素+改变的信息素*

pheromonetable = (1 - rho) \* pheromonetable + changepheromonetable

iter += 1 *# 迭代次数指示器+1*

print(**"this iteration end："**,iter)

*# 观察程序执行进度，该功能是非必须的*

**if** (iter - 1) % 20 == 0:

print(**"schedule:"**,iter - 1)

*#迭代完成*

*#以下是做图部分*

*#做出平均路径长度和最优路径长度*

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12, 10))

axes[0].plot(lengthaver, **'k'**, marker=**'\*'**)

axes[0].set\_title(**'Average Length'**)

axes[0].set\_xlabel(**u'iteration'**)

*#线条颜色black https://blog.csdn.net/ywjun0919/article/details/8692018*

axes[1].plot(lengthbest, **'k'**, marker=**'<'**)

axes[1].set\_title(**'Best Length'**)

axes[1].set\_xlabel(**u'iteration'**)

*#fig.savefig('Average\_Best.png', dpi=500, bbox\_inches='tight')*

plt.close()

fig.show()

*# 作出找到的最优路径图*

bestpath = pathbest[-1]

plt.plot(coordinates[:, 0], coordinates[:, 1], **'r.'**, marker=**'>'**)

plt.xlim([-100, 2000])

*#x范围*

plt.ylim([-100, 1500])

*#y范围*

**for** i **in** range(numcity - 1):

*#按坐标绘出最佳两两城市间路径*

m, n = int(bestpath[i]), int(bestpath[i + 1])

*#打印最佳路径*

print(**"best\_path:"**,m, n)

plt.plot([coordinates[m][0],coordinates[n][0]], [coordinates[m][1], coordinates[n][1]], **'k'**)

plt.plot([coordinates[int(bestpath[0])][0],coordinates[int(bestpath[51])][0]], [coordinates[int(bestpath[0])][1],coordinates[int(bestpath[50])][1]] ,**'b'**)

ax = plt.gca()

ax.set\_title(**"Best Path"**)

ax.set\_xlabel(**'X\_axis'**)

ax.set\_ylabel(**'Y\_axis'**)

*#plt.savefig('Best Path.png', dpi=500, bbox\_inches='tight')*

plt.show()

plt.close()