《人工智能导论》

**基于蚁群算法求解旅行商问题编程作业**

作业类型：个人作业

指导老师：冀俊忠

报告人：高立扬 21071003

报告人签字：

2023年12月 日

目录

[实验环境 2](#_Toc25285)

[问题描述 2](#_Toc19070)

[1.作业基础——旅行商问题 2](#_Toc31284)

[2. 算法基础——蚁群算法 3](#_Toc5222)

[主要算法思想和设计 3](#_Toc28675)

[1.约束 3](#_Toc11261)

[2.参数设计 4](#_Toc25416)

[2.1 核心变量 4](#_Toc13483)

[2.2 其他变量及计算 4](#_Toc28649)

[3.算法思想 5](#_Toc12399)

[4.算法设计 5](#_Toc22220)

[4.1 初始化 5](#_Toc12244)

[4.2 ACO算法 7](#_Toc29838)

[4.3 绘图 10](#_Toc23131)

[程序框图 11](#_Toc25270)

[运行结果说明 11](#_Toc23372)

# 实验环境

IDE：VSCODE 最新版

内部库：math, random

外部库：matplotlib, numpy

python版本：3.10.12

# 问题描述

1.作业基础——旅行商问题

旅行商问题，简称TSP。问题描述如下：假设有一个旅行商人要拜访N个城市，他必须选择所要走的路径，路径的限制是，每个城市只能拜访一次，而且最后需要回到原来出发的城市。路径的选择目标，是要求得的路径路程为所有路径之中的最小值。TSP的历史很悠久，其雏形可以追溯到1759年欧拉研究的骑士周游问题。从图论的角度来说，TSP其实是在一个带权的完全无向图中，寻找一个权值最小的哈密顿回路。

该问题是一个典型的NP-hard问题，即求得一个哈密顿回路很容易，但是求得权值小于的哈密顿回路的回路，在最坏的情况下可能需要检查很多很多回路，这将是一个天文数字！时间复杂度达到了O(n!)！显然我们不能依靠暴力搜索来解决TSP问题，因此在计算机科学与技术发展的早期，科学家们用动态规划法、线性规划法、分支界定法等，然而随着城市数的增加，这些方法变得乏力起来，随着科学技术的发展，诞生了遗传算法、蚁群算法、模拟退火法、神经网络等来解决城市数目更多的TSP，由于该问题是NP-hard问题，因此所有的算法只能说是求近似解，并不能求得最优解（这是由科学家定义的）。本报告将用蚁群算法来尝试求得TSP优解。

旅行商问题的应用领域十分广，并且至今都发挥着重要作用。在交通领域，它可以辅助规划合理的交通网，减少拥堵；在物流领域，它可以帮助企业更好地规划物流，提高经济效益，减少运营成本；在互联网领域，它可以辅助网络节点设计，让信息更好地流动。

1. 算法基础——蚁群算法

仿生学的研究给人类自然科学技术的发展和进步带来了许多启示和重要的贡献，群集智能作为一种新兴的广为研究的演化计算技术，成为计算智能中的一个研究领域。蚁群算法是其中的一个研究热点，它是模仿蚂蚁群在觅食过程中所体现出的智能行为而提出的一种搜索方法。

蚂蚁在觅食的过程中虽然没有直接的语言交流，但是却总能找到从食物源到蚁穴之间的最短道路，这是因为蚁群通过信息素来完成了信息的传播。蚂蚁在觅食过程中会在经过路径上留下一定浓度的信息素，某条路径上信息素的浓度与通过该路径的蚂蚁数量成正比，而蚂蚁倾向于通过信息素浓度高的路径，这就形成了正反馈，这就确保了较短的觅食路径能迅速积累信息素。

蚁群算法的思想正是蚂蚁觅食，释放信息素的过程。详细算法思想与设计将在下文说明。

蚁群算法的优点是具有良好的鲁棒性，模仿了蚁群觅食中的信息素正反馈过程，还有分布式并计算的优势。但是它自身也存在着一些缺点，比如当城市大于100时，迭代次数会过多，且算法容易陷入局部最佳，精度欠佳。个人认为，想要针对弊端来改良蚁群算法，科学家应当进一步研究蚂蚁的觅食行为，设置对照实验；同时算法的设计不仅要基于现实而且要超越于现实，结合实际应用而开发新的模型、设置新的约束等。

# 主要算法思想和设计

1.约束

正如上文提到，蚁群算法是对自然界蚁群的觅食寻径方式进行模似而得出的一种仿生算法——蚁群在运动过程中，能够在它所经过的路径上，留下一种叫做息素的物质进行信息传递。蚁群中的每一只蚂蚁，在运动过程中都能够感知这种物质，并以此指导自己的运动方向。

为了更好地实现算法，我们需要先对算法里的“蚂蚁”施加一些约束，如下：

①蚂蚁完成一次路径搜寻后，会在行进的路线上释放信息素，信息素的量与解的质量成正比

②蚂蚁在一次完整的寻路过程中，不会重复访问已经过的“城市”，因此需要设置一个表来进行标记，确保蚂蚁寻路的合法性

③蚂蚁知道任意两个城市之间的距离

④蚂蚁在选择路径的时候，会同时考虑到信息素浓度和两点之间的距离，采用随机局部搜索策略，从而确保距离较短的边、信息素浓度较大的边，被选择的概率也越大

⑤所有的蚂蚁都搜索完一次，代表着迭代次数加一，此时才能更新信息素，随后新的一轮蚂蚁进行下一次迭代。因此，同一代蚂蚁不会互相影响，计算④中的概率时用到的是上一代蚂蚁留下的信息素。规定第一代蚂蚁迭代时，城市的信息素浓度都为1

⑥更新信息素包含了原有信息素的“挥发”和该轮蚂蚁经过的路线上信息素的“释放”

⑦若没有达到预定的迭代次数，算法陷入局部最优解，则终止算法，以局部最优解为问题答案。

2.参数设计

有了对蚂蚁的约束，下面就可以开始设计算法中所需要用到的参数以及计算公式等，如下所述

2.1 核心变量

①信息素浓度 ②i城市到j城市的距离（）

③启发信息 ④蚂蚁数量M，它一般设置为城市数目的1.5倍

2.2 其他变量及计算

①一只蚂蚁从城市i走向城市j的转移概率为



该方法用到了“轮盘赌”的思路

其中Uk为当前蚂蚁可经过（也就是未经过）的城市的集合

α为信息素因子，反映了蚂蚁觅食过程中，路径上积累的信息素在指导蚁群搜索中的重要程度，取值一般在[1,4]，取值过大会导致蚂蚁重复选择前辈认为“较优”的路径，随机性降低；取值过小会导致随机性过大，不利于较优路径的选择

β为启发信息因子，反映了启发式信息在蚂蚁觅食过程中的重要程度，取值一般在[0,5]，取值过大会导致收敛加快，容易陷入局部最优；取值过小会导致搜索随机性过大，不利于较优路径的选择

②信息素的迭代更新





其中ρ是信息素挥发因子，常用取值为[0.5, 0.8]，取值过大会导致信息素挥发过快，导致较优路径被“遗忘”，取值过小会导致各个路径的信息素浓度差异变小，不利于寻优

每只蚂蚁在各自周游中所产生的信息素总量为Q，取值一般在[10, 1000]，Lk为路径长度，可见路径过长会导致路径信息素浓度变低。Q过大会导致过早收敛，导致局部最优；过小会导致各个路径信息素浓度差别过小，系统陷入混沌状态

③迭代次数一般取[100,500]，常用200

3.算法思想

算法的主要思想是：通过蚂蚁觅食释放信息素的正反馈机制，控制迭代次数使蚂蚁们通过一次又一次的迭代，不断地寻找到较优的路径，最后拟合出较为接近最优解的解。

4.算法设计

4.1 初始化

顾名思义，在算法开始之前，先要对变量和参数进行初始化，由于算法体量较小，因此直接设置为全局变量，方便蚁群算法的访问与更新。

没有定义任何类，因为python的list使用起来非常灵活，完全足够解决本问题，定义类反而复杂。

一些变量或参数的定义用到了numpy外部库，这是一个强大的数组工具，可以进行多种多样的list操作，是python最火的外部库之一，可谓家喻户晓，美名远扬，自从大一接触深度学习，一直到现在，numpy仍不断展现着它的惊艳。

一共构建了30个城市。

ρ设置为0.1，这是为了先确保实验快速推进，观察算法大题运转过程。

下表1.展示了初始化的变量和参数，及其变量类型和介绍

表1.初始化

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 简介 |
| city\_name | list | 存放城市的名字（1,2,3……）便于后续画图 |
| city\_xy | list | 存放城市的坐标 |
| city\_num | int | 城市的个数 |
| dis | numpy list | 城市之间的距离，二维矩阵，主对角线为无穷小 |
| M | int | 蚂蚁的数量 |
| alpha | int | 信息素因子α |
| beta | int | 启发信息因子β |
| rou | float | 信息素挥发因子ρ |
| iter | int | 当前迭代轮数，初始为0 |
| epoch | int | 总迭代轮数 |
| Q | int | 信息素总量 |
| phe | numpy list | 信息素浓度表，初始为全1，对应τ |
| candidate | numpy list | 蚂蚁路径候选表 |
| path\_best | numpy list | 最佳路径表 |
| dis\_best | numpy list | 最佳回路长度表 |
| eta | numpy list | dis矩阵所有元素进行取倒数操作，对应η |

下面是完整代码：

city\_name = []

city\_xy = []

with open('data2.txt','r',encoding='UTF-8') as f:

lines = f.readlines()

for line in lines:

line = line.split('\n')[0]

line = line.split(',')

city\_name.append(line[0])

city\_xy.append([float(line[1]), float(line[2])])

city\_xy = np.array(city\_xy)

# 读取城市距离

city\_num = len(city\_name)

dis = np.zeros((city\_num, city\_num))

for i in range(city\_num):

for j in range(city\_num):

if i != j:

dis[i][j] = math.sqrt((city\_xy[i][0] - city\_xy[j][0]) \*\* 2 + (city\_xy[i][1] - city\_xy[j][1]) \*\* 2)

else:

dis[i][j] = 100000

# 蚂蚁数量

M = 100

# 城市数量

alpha = 1

beta = 2

rou = 0.1 #挥发速度

iter = 0

epoch = 200

Q = 100

# 初始信息素矩阵

phe = np.ones((city\_num, city\_num))

# 候选集列表,存放蚁群的路径(一只蚂蚁一个路径)，一共是蚂蚁数量\*城市数量

candidate = np.zeros((M, city\_num)).astype(int)

# path\_best存放的是相应的，每次迭代后的最优路径，每次迭代只有一个值

path\_best = np.zeros((epoch, city\_num))

# 存放每次迭代的最优距离

dis\_best = np.zeros(epoch)

# 倒数矩阵

eta = 1.0 / dis

4.2 ACO算法

Step1.蚂蚁初始路径选择

在每一轮迭代开始的时候，需要随机为蚂蚁选择出发点，该部分主要用到了numpy的random.permutation函数，该函数将指定列表内的元素进行随机打乱。那么它为什么在蚂蚁初始路径选择时起到了核心作用？如果我们有一个从0到n的，存放所有城市节点下标的列表，此时将其用上述函数进行随机打乱，那么它就可以作为每一次迭代开始时，蚂蚁的出发点。

然而蚂蚁的数量不可能正好等于城市数目，因此需要分两种情况进行讨论，我们还需要一个存放蚂蚁下标的列表：如果蚂蚁数量小于等于城市数目，那么被打乱的列表，也存放所有蚂蚁的下标即可；反之，则需要将蚂蚁下标列表切块，进行上述操作，这就会出现至少存在一个城市，在迭代开始的时候是至少两只蚂蚁的出发点的情况。

具体代码实现如下：

if M <= city\_num:

# 生成一个随机起点列表，并且起点不重复

candidate[:, 0] = np.random.permutation(range(city\_num))[:M]

else:

# 蚂蚁比城市多，那就按照城市数量，进行分块处理

delta = M - city\_num

flag = 0

while delta > city\_num:

candidate[city\_num\*flag:city\_num\*(flag+1),0] = np.random.permutation(range(city\_num))[:]

delta = delta - city\_num

flag = flag + 1

candidate[city\_num\*(flag+1):M,0]=np.random.permutation(range(city\_num))[:delta]

Step2.城市选择

选择好蚂蚁的出发点，下面需要模拟蚂蚁的“觅食”了。蚂蚁在选择城市的时候，会用到上文提到的概率计算公式，还会用到轮盘赌法。什么是轮盘赌法？

设群体的规模为 N ，F(xi)(i=1, ..., N)是其中每个个体的适应值,则第 i 个个体被选中的概率由下式给出：



（1）r=random(0, 1)，s=0，i=0；

（2）如果s≥r，则转（4）；

（3）s=s+p(xi)，i=i+1, 转（2）

（4）xi 即为被选中的染色体，输出i

（5）结束.

同时，在蚂蚁觅食的过程中，还需要有一个数组来存储每一只蚂蚁觅食路线的长度，便于算法的后续运算。

具体代码如下：

# Step2：每一只蚂蚁的下一个城市选择

for i in range(M):

# uk：未曾访问的城市

uk = list(range(city\_num))

# 蚂蚁的起点

visit = candidate[i, 0]

uk.remove(visit)

for j in range(1, city\_num):#访问剩下的城市

protrans = np.zeros(len(uk))#每次循环都更改当前没有访问的城市的转移概率矩阵1\*30,1\*29,1\*28...

# 下一城市的概率函数

for k in range(len(uk)):

# 计算概率公式的分子

protrans[k] = np.power(phe[visit][uk[k]], alpha) \* np.power(

eta[visit][uk[k]], beta)

# 累计概率，轮盘赌选择法

finalpro = (protrans / sum(protrans)).cumsum()

finalpro -= np.random.rand()

# 求出离随机数产生最近的索引值，这便是轮盘赌中第一个被选中的数值

k = uk[list(finalpro > 0).index(True)]

# 下一个访问城市的索引值

candidate[i, j] = k

uk.remove(k)

length[i] += dis[visit][k]

visit = k # 更改出发点，继续选择下一个到达点

length[i] += dis[visit][candidate[i, 0]]#最后一个城市和第一个城市的距离值也要加进去

Step3.更新最短路径

当本迭代波次的蚂蚁觅食完毕，就需要根据蚂蚁们的觅食路径进行统计，更新最短路径。本环节难度不大，只需要考虑到三种情况：

①第一次迭代，直接选择最短的一条路径作为最优路径

②非第一次迭代，若本次迭代所有的路径都不如上一次的最优路径短，那么就将本次迭代的最优路径设置为上一次的最优路径

③非第一次迭代，若本次迭代中最短的路径比上一次的最优路径短，那么选择此路径为最优路径

具体代码如下：

# Step3：更新最短路径

# 第一次迭代直接选择最短路径

if iter == 0:

dis\_best[iter] = length.min()

path\_best[iter] = candidate[length.argmin()].copy()

else:

# 当前路径都不如上一次的短，那就替换本次路径为上一次的

if length.min() > dis\_best[iter - 1]:

dis\_best[iter] = dis\_best[iter - 1]

path\_best[iter] = path\_best[iter - 1].copy()

# 否则直接选择最短路径

else:

dis\_best[iter] = length.min()

path\_best[iter] = candidate[length.argmin()].copy()

Step4.信息素浓度更新

按照公式进行信息素浓度更新即可，在最后需要将迭代器加一，防止死循环。

具体代码如下：

#Step4：信息素更新

#信息素的增加量矩阵

delta\_phe = np.zeros((city\_num, city\_num))

# 城市之间的信息素增量

for i in range(M):

for j in range(city\_num - 1):

delta\_phe[candidate[i, j]][candidate[i][j + 1]] += Q / length[i]

#最后一个城市和第一个城市的信息素增加量

delta\_phe[candidate[i, j + 1]][candidate[i, 0]] += Q / length[i]

#信息素更新的公式：

phe = rou \* phe + delta\_phe

iter += 1

4.3 绘图

信息可视化是科学研究中，帮助学者进行理解的重要环节，通过matplotlib外部库进行绘图，具体代码如下：

print("最优路径",path\_best[-1]+1)

print("迭代", epoch,"次后","求得最优解",dis\_best[-1])

# 路线图绘制

fig = plt.figure()

plt.title("最佳路径")

x = []

y = []

path = []

for i in range(len(path\_best[-1])):

x.append(city\_xy[int(path\_best[-1][i])][0])

y.append(city\_xy[int(path\_best[-1][i])][1])

path.append(int(path\_best[-1][i])+1)

x.append(x[0])

y.append(y[0])

path.append(path[0])

for i in range(len(x)):

plt.annotate(path[i], xy=(x[i], y[i]), xytext=(x[i] + 0.3, y[i] + 0.3))

plt.plot(x, y,'-o')

fig = plt.figure()

plt.title("迭代-距离图")

plt.plot(range(1, len(dis\_best) + 1), dis\_best)

plt.xlabel("迭代次数")

plt.ylabel("距离值")

plt.show()

# 程序框图

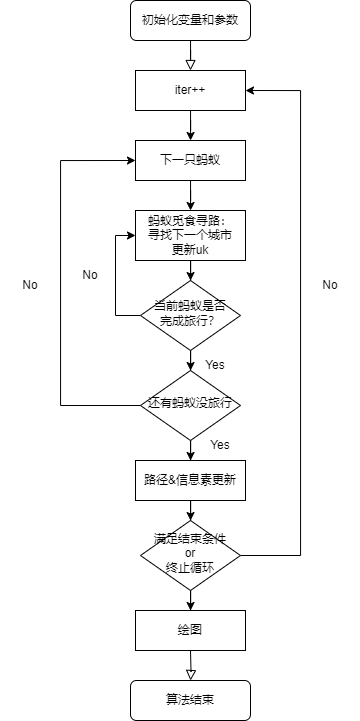


图1.流程图

# 运行结果说明

下面将节选一次运行结果图，参数设置详见算法设计4.1初始化的代码，用时35.5秒完成200次迭代

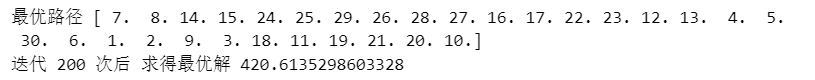


图2.迭代结果

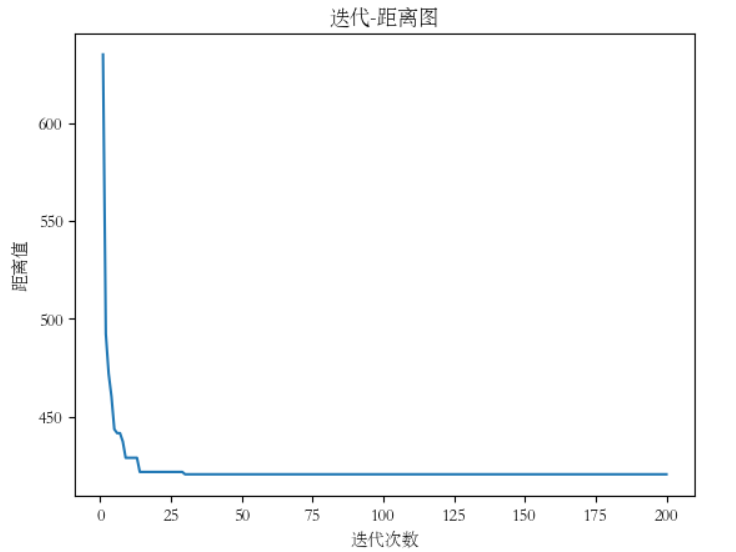
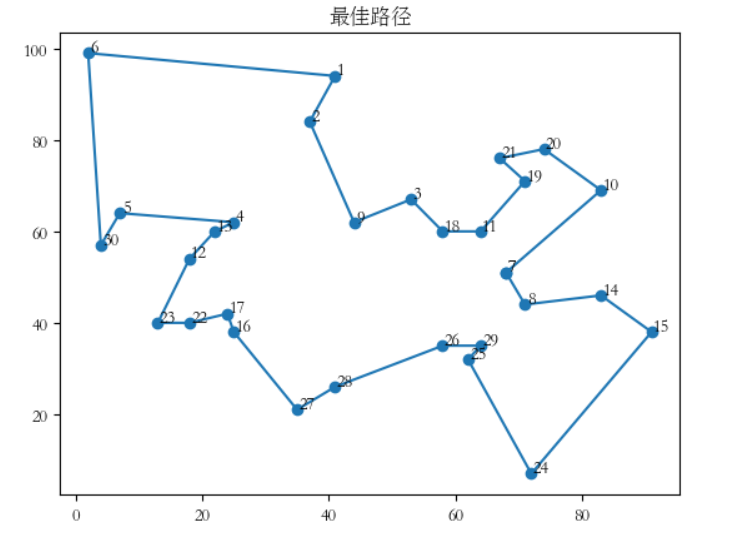


图3-4 路线图&迭代-距离曲线

如图可见，100只蚂蚁30个城市的情况下，通过参数的调整，曲线可以迅速收敛，反复实验证明算法的结果不会出现跳变，基本都稳定在420左右，误差不到1，有一次试验甚至在15次不到的迭代过后就收敛了。

因此，需要在每次迭代的时候，判断蚂蚁们是否陷入了局部最优解，及时进行中断循环，可以节省很多时间。对此，我设置了一个stopit变量，在每次迭代更新最优解时，连续碰到局部最优解就进行计数，否则立刻置零，当stopit变量累计到10，就立刻break出程序，经测试，运行时间缩短到了4.6秒（15次迭代收敛的情况下）

# 算法改进

1. 信息素增量模型

从课堂所学中，我了解到了几种信息素增量模型：

①蚁量模型： ②蚁密模型： ③蚁环模型：

④蚁恒模型：

其中模型④我认为最接近显示情况，它能体现在一次周游中蚂蚁所消耗能量Q在各段路径上的分配，还能反映各段路径长度的不同，一只蚂蚁在同次周游中留在不同路径上的信息素增量各不相同的事实。而且对于蚂蚁的任何一次周游，都有



也就是说，用到此模型的算法，是遵守能量守恒定律的，非常科学。

1. 算法改进

根据上述蚁恒模型，更改信息素更新代码，重新测试运行结果，其中一次测试结果如图5-7，运行结果为5.4秒，参数仍不变

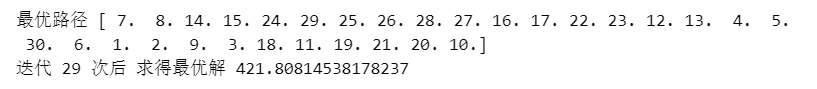


图5.迭代了30次不到

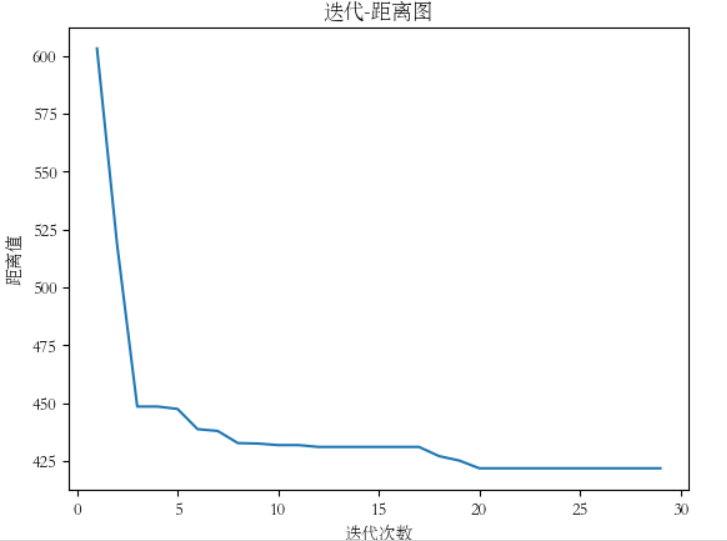
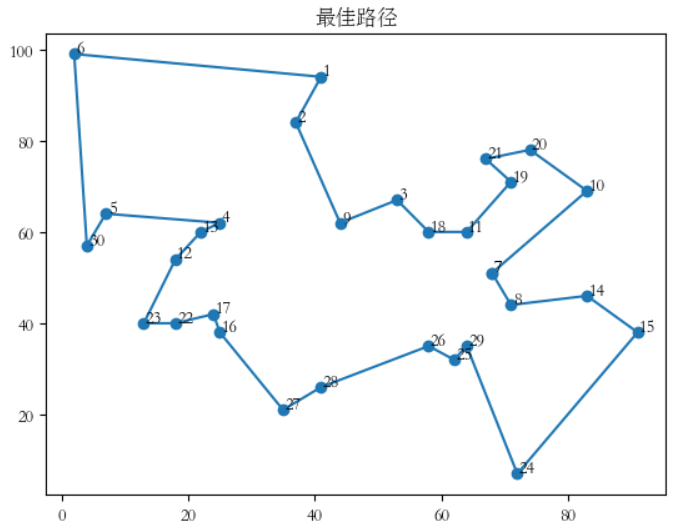


图6-7 效果图

结果和原模型差距不大，而且重复实验发现结果波动较大，最大的一次求得的最优解是426。

思考和反思发现，这可能是因为参数设置不变导致的，因为新的模型需要新的适配参数，蚁群算法中有如此多的参数，不进行寻优肯定是不行的，而且我的数据集的复杂程度可能不足以该模型施展其才能。

由于参数众多，决定挑选我认为最重要的三个因子：ρ、α和β，进行参数寻优，首先对ρ进行寻优，结果如下表2.所示

表2.ρ寻优

|  |  |  |
| --- | --- | --- |
| ρ的取值 | 局部最优迭代次数 | 最优解 |
| 0.1 | 21 | 434.18 |
| 0.2 | 48 | 421.81 |
| 0.3 | 36 | 420.61 |
| 0.4 | 32 | 424.83 |
| 0.5 | 31 | 426.05 |
| 0.6 | 26 | 420.61 |
| 0.7 | 23 | 434.38 |
| 0.8 | 33 | 433.87 |
| 0.9 | 19 | 420.61 |

由于算法的随机性，迭代次数和最优解似乎无规律，因此需要重复试验进行统计，由于时间问题，不进行多次实验，综合考虑选择ρ为0.9，继续进行参数寻优，这次选择α，如表3.

表3.α寻优

|  |  |  |
| --- | --- | --- |
| α的取值 | 局部最优迭代次数 | 最优解 |
| 1 | 19 | 440.90 |
| 1.5 | 20 | 427.60 |
| 2 | 56 | 421.81 |
| 2.5 | 26 | 437.24 |
| 3 | 13 | 446.11 |
| 3.5 | 12 | 442.44 |
| 4 | 12 | 465.25 |

明显可以看出α过大过小都会导致过早陷入局部最优，过大还会导致路线出现交叉现象如图8所示。

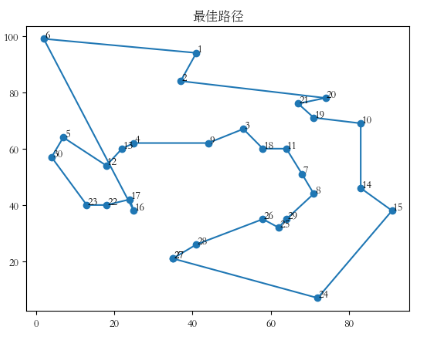


图8.交叉线路

因此选择α为2，下面对β进行寻优，如表4.

表4.β寻优

|  |  |  |
| --- | --- | --- |
| β的取值 | 局部最优迭代次数 | 最优解 |
| 0 | 12 | 1017.45 |
| 0.5 | 32 | 540.02 |
| 1 | 17 | 441.11 |
| 1.5 | 18 | 427.76 |
| 2 | 15 | 427.61 |
| 2.5 | 22 | 432.96 |
| 3 | 18 | 420.61 |
| 3.5 | 25 | 421.81 |
| 4 | 67 | 420.61 |
| 4.5 | 199 | 420.61 |
| 5 | 16 | 431.80 |

可见，β过小带来的后果是灾难性的，如图9所示，蚂蚁似乎像无头苍蝇一样胡乱寻路。β过大会导致随机性过大，这一点可以从变化不定的迭代次数和最优解看出来。我认为β取3比较合适。

综上所述，我认为蚁群算法的一组较好取值为ρ=0.5，α=2，β=3

1. 仍可以改进的地方

本报告撰写的蚁群算法并不完善，还有很多情况没有考虑，下面是一些可以改进的地方

①同一迭代波次中，考虑到蚂蚁信息素的相互作用（八邻域）

②考虑到信息素向四周扩散，扩散浓度与两点间距离和释放信息素的量有关

③在蚂蚁选择下一个城市的时候，不仅要考虑该城市的信息素浓度，还要考虑到附近几个城市信息素的浓度（八邻域、一定距离内）

# 总结与思考

通过本次个人作业，我更加深入地了解到了TSP问题，并且再次温习了蚁群算法，并得以付诸于实践。

一开始在课堂学习的过程中，我对蚁群算法的概念理解非常模糊，在本次实验的过程中，我查阅了资料，详细了解了蚁群迭代、信息素更新、路程更新、轮盘赌思想等，这帮助了我完成了算法撰写。

实现算法后，我进行了参数寻优，获得了一组较好的参数，经过思考找出了一些可以优化算法的方案。

总的来说，本次实验使我将理论付诸于实践，从而对理论知识有了更深的认识和理解，我受益匪浅。我从大一就开始接触人工智能、深度学习，并且获得了人工智能相关课题的星火优秀项目，后来加入北京工业大学PIP战队视觉组，进行机器人视觉算法研究，到如今和中科院软件所国重组内进行学习，可以说，我对人工智能一直有着十分的热情，而在求学路上的一次次实验和实战，都是养成我严谨态度、科学思维的宝贵财富，而每一次反思和总结是帮助我进步的阶梯，感谢《人工智能导论》课的两个大作业，让我重新审视了自己对科学的态度，燃烧我付诸于科研的热情！