编程语言：pyhton3.6

**实验1、GA解TSP问题**

基本原理：

遗传算法的主要特点是通过染色体的模拟和模拟染色体上的进化操作，搜索问题的最优解。解被编码成生物个体中的染色体，问题的目标函数对应于生物个体的适应度函数。在此基础上，通过在染色体上进行的生物种群的多代进化，获得最适应环境的染色体作为最优解。

算法中的构成要素：

* 解的编码方法：为了进行遗传计算，需要在问题解的原始表示形式与染色体表示形式之间建立对应关系。
* 确定初始群体和群体规模：一般采用随机法。
* 适应度函数：适应度函数是对生物个体适应性进行度量的函数。通过适应度函数来确定染色体所代表的解的优劣程度，体现了对求解目标的要求。
* 遗传操作：解的搜索是通过染色体上的遗传操作来实现的。基本遗传算法的遗传操作包括选择，交叉和突变。

基本过程：

1. 利用GA算法解TSP问题，将城市序列当做个体的基因串，把遍历的距离的倒数当做适应度函数，确定突变率，交叉率。
2. 随机生成初始群体
3. 计算每个个体适应度
4. （1）选择：按适应度高的原则从群体中选择个体形成新群体，规模与旧群体相当。

（2）交叉：根据交叉率选择两个个体进行交叉形成新的个体，将一个个体的基因序列插到另一个个体上然后删除重复序列

（3）突变：根据突变率选择某个个体突变，将基因序列上的某两位交换

5.若达到最大迭代次数或者群体性能已满足，输出个体适应度最高的结果，否则重复第四步

具体实现：以二维列表表示个体和其对应的城市序列（0到49），城市个数50，个体个数50，交叉率0.7，突变率0.1

1. 初始个体函数

**def** mklife():  
 seq = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,  
 30, 31, 32, 33, 34, 35, 36,37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]  
 random.shuffle(seq)*#将上述列表随机打断并返回* **return** seq

1. 选择程序段

*#选择操作  
#没有进行轮盘操作，简单的选取前5个3次15个2次30个1次作为新群体*j=0  
**while** j<30:  
 ganew[j].list=ga[j].list  
 j+=1  
j = 0  
**while** j < 15:  
 ganew[j+30].list = ga[j].list  
 j += 1  
j = 0  
**while** j < 5:  
 ganew[j+30+15].list = ga[j].list  
 j += 1

1. 交叉函数和操作

*#交叉函数***def** jiaocha(a,b):*#输入两段基因序列* e=[]  
 **for** i **in** a:  
 e.append(i)  
 ci=random.randint(1,50)  
 i=0  
 **while** i<ci:*#将第二段序列的前ci个添加到第一段中* e.append(b[i])  
 i+=1  
 c=[]  
 i=49+ci  
 **while** i>=0:*#从后往前遍历删除重复元素* **if** e[i] **not in** c:  
 c.append(e[i])  
 i-=1  
 d=[]  
 i = 49  
 **while** i >= 0:*#反转基因序列* **if** c[i] **not in** d:  
 d.append(c[i])  
 i -= 1  
 **return** d

*#交叉操作*num=0  
**while** num<50:  
 p=random.random()*#随机生成小数* **if** p<0.7:*#小与交叉率则随机选择两个个体进行交叉* i=random.randint(0,49)  
 j=random.randint(0,49)  
 ganew[num].list=jiaocha(ganew[i].list,ganew[j].list)  
 num+=1

1. 突变函数和操作

*#突变函数***def** tubian(a):  
 i=random.randint(0,49)*#随机选取两个元素* j=random.randint(0,49)  
 tmp=a[i]*#交换* a[i]=a[j]  
 a[j]=tmp  
 **return** a

*# 突变操作*num=0  
**while** num<50:  
 p=random.random()  
 **if** p<0.1:*#小于突变率进行操作* ganew[num].list=tubian(ganew[num].list)  
 num += 1

1. 适应度计算函数和程序段

*#进行适应度计算，但是没有根据适应度排序，因为适应度与距离成反比，所以根据距离升序排列，前面的个体就是适应度高的个体***for** i **in** ganew:  
 i.gadistance = getdistance(i.list)*#计算个体走的距离函数* i.fitt = getfit(i.gadistance)  
cmpfun = operator.attrgetter(**'gadistance'**)  
ganew.sort(key=cmpfun)

*#计算遍历距离函数***def** getdistance(s):  
 gadistance=0.0  
 j=0  
 **while** j<49:  
 gadistance+=distance[s[j]][s[j+1]]  
 j+=1  
 gadistance+=distance[s[0]][s[49]]  
 **return** gadistance  
  
*#计算适应度函数***def** getfit(s):  
 **return** 1/s

实验效果：在后面统一描述

**实验2、ACO解TSP问题**

基本原理：

蚁群优化算法是采用人工蚂蚁的行走路线来选择问题最优解的一种算法。每只人工蚂蚁独立地在问题解空间中搜索，当碰到解的分支路径时，随机地选择某条路径行走，其中信息素浓度更高的路径具有更大的选择概率。

人工蚂蚁在行走的路径上释放出与路径长度有关的信息素。路径越短，信息素浓度越高。随着时间的推移，路径长度短的路径上的信息素浓度越来越高，引导更多的人工蚂蚁通过最优的求解路径，释放出更多的信息素，而在其他路径上的信息素在挥发特性的作用下却逐渐消失，从而形成正反馈效应。

最终整个蚁群在正反馈的作用下集中到代表最优解的路径下，表明找到了最优解。

基本过程：

1. 先定义初始路径信息素浓度，挥发率，蚂蚁个数，城市间距离，城市个数
2. 将m只蚂蚁随机放在n个城市上
3. （1）对于每只蚂蚁计算到达从这个城市去往还没经过的城市的概率，这个概率与信息素浓度成正比，与距离成反比。

（2）使蚂蚁移动到具有最大转移概率的城市

（3）重复（1）（2）直到蚂蚁遍历所有城市

4.更新信息素

5.满足终止条件输出最优结果，否则返回3

具体实现：以二维数组表示蚂蚁单位和其遍历的城市序列，城市个数50，蚂蚁个数50，初始信息浓度1，挥发率0.5，Q=100，α=1，β=-2

1. 初始化蚂蚁所在位置程序段

*#初始化蚂蚁位置*i=0  
**while** i<50:  
 ant[i].append(random.randint(0,49))  
 i+=1

1. 计算最大转移概率城市并转移程序段

i=0  
**for** key **in** ant:  
 num = 0 *# 已经走过的城市个数* **while** num < 49:  
 city = 0  
 p = [0 **for** i **in** range(50)]  
 pall=0  
 nextcity = -1 *# 下一个城市的编号* **while** city < 50:  
 **if** city **not in** key: *# 如果这个蚂蚁没有走过这个城市* p[city] = pow(t[key[-1]][city],1) \* pow(1 / distance[key[-1]][city], 2.0) *# 计算概率t为信息素浓度，distance为两个城市距离* pall+=p[city]  
 city+=1  
 *# 轮盘选择城市* **if** pall > 0:  
 *# 产生一个随机概率* temp\_prob = random.uniform(0.0, pall)  
 **for** j **in** range(50):  
 **if** j **not in** key:  
 *# 轮次相减* temp\_prob -= p[j]  
 **if** temp\_prob < 0.0:  
 nextcity = j  
 **break** distance\_ant[i] += distance[key[-1]][nextcity] *# 计算当前走过的距离* key.append(nextcity) *# 将这个城市加入到已经走过的城市中去* num += 1  
 i+=1

1. 得到目前最优距离程序段

i=0  
**for** key **in** ant:  
 key.append(key[0])*#将初始城市加入到列表末尾* distance\_ant[i]+=distance[key[-1]][key[-2]]*#加上回到初始城市的距离* **if** distance\_ant[i]<distancemin:*#找到最优的城市* distancemin=distance\_ant[i]  
 pathmin=ant[i]  
 i+=1  
print(**'最短距离'**,distancemin,**'迭代次数'**,time,**'路径'**,pathmin)

1. 更新信息素和蚂蚁初始化程序段

*#信息素更新*tchange = [[0 **for** i **in** range(50)] **for** i **in** range(50)]  
j=0  
**for** key **in** ant:*#根据蚂蚁走过的城市增加信息素为100/蚂蚁走过的总距离* i=0  
 **while** i<50:  
 tchange[key[i]][key[i+1]]+=(100/distance\_ant[j])  
 i+=1  
 j+=1  
i=0  
**while** i < 50:*#每条路径更新信息素为0.5\*原信息素浓度+增加信息素浓度* j = 0  
 **while** j < 50:  
 t[i][j] = 0.5 \* t[i][j]+tchange[i][j]+tchange[j][i]  
 j += 1  
 i += 1  
**for** key **in** ant:*#清空蚂蚁走过的城市* i=1  
 **while** i<51:  
 **del** key[1]  
 i+=1

实验效果：在后面统一表述

**实验3、Hopfield解TSP问题**

基本原理：

TSP的解是n个城市的一个排列。为了用神经网络解决这个问题，必须有一个表示问题的方法。使得神经元的输出状态可以编码成n个城市的一个排列，即n个城市在排列中的位置由各自位置的n个神经元的输出状态决定。

对于n个城市需要nxn个神经元构成的nxn阶矩阵来表示。在该换位矩阵中，行表示城市访问的城市，列表示访问的次序，每一列只有一个元素是1，每一行只有一个元素是1，并且整个矩阵中1的个数是n。通过这样的换位矩阵，可以唯一确定一条路线。

Hopfield网络的状态变化分析的核心是对每个网络的状态定义一个能量E，任意一个神经元节点状态发生变化时，能量E将减少。在满足一定的参数条件下，某种能量函数在网络运行过程中是不断降低，最后趋于稳定平衡状态的。当系统稳定后，其最终能量就是所对应的目标函数E的极值。

基本过程：

1. 确定城市个数n，建立n\*n个神经元的离散型霍普菲尔德网络，给定初始状态，第I,j个神经元活跃表示i时刻到达j城市
2. 计算权值w和神经元的输入（按照人工智能教科书P157的公式4-56和4-57）  
   

1. （1）随机选择一个神经元或者一组神经元

（2）根据激活函数改变其状态

（3）重复上述步骤直到网络稳定

4.得到的神经网络对应TSP问题的解

具体实现，城市个数为4，采用串行模式更新状态。

1. 神经元初始化

*#使用二维列表表示神经元状态,1为活跃，-1为抑制*h=[[1,-1,-1,-1],[-1,1,-1,-1],[-1,-1,-1,1],[-1,-1,1,-1]]

1. 权值w和输入函数初始化

*#权值初始化l1=l2=l3=l4=10，q=[[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0],[0,0,0,1,0]]***while**(x<4):  
 y=0  
 **while**(y<4):  
 i=0  
 **while**(i<4):  
 j=0  
 **while**(j<4):  
 w[x][i][y][j]=-(l1\*d[x][y]\*(q[i][j+1]+q[i][j-1])+l2\*q[x][y]\*(1-q[x][y])+l3\*q[i][j]\*(1-q[x][y])+l4\*(1-q[x][y])\*(1-q[i][j]))*#根据公式4-56* j+=1  
 i+=1  
 y+=1  
 x+=1

*#神经元输入初始化*I=[[0 **for** i **in** range(4)]**for** i **in** range(4)]  
x=0  
**while** x<4:  
 i=0  
 **while** i<4:  
 I[x][i]=l4\*n  
 i+=1  
 x+=1

1. 随机选择神经元并改变状态

a=random.randint(0, 3)*#随机选取a行b列的神经元*b=random.randint(0,3)  
s=0  
i=0  
**while** i<4:*#计算输入信号和反馈信号的整合结果* j=0  
 **while** j<4:  
 s+=w[a][b][i][j]\*h[i][j]  
 j+=1  
 i+=1  
s+=I[a][b]  
**if** s>=0:*#根据整合结果对神经元状态进行更新* h[a][b]=1  
**else**:  
 h[a][b]=-1

1. 计算能量

*#计算能量，根据公式4-55*e=0  
x=0  
**while** x<4:  
 i=0  
 **while** i<4:  
 y=0  
 **while** y<4:  
 j=0  
 **while** j<4:  
 e-=0.5\*h[x][i]\*w[x][i][y][j]\*h[y][j]  
 j+=1  
 y+=1  
 i+=1  
 x+=1  
x=0  
**while** x<4:  
 i=0  
 **while** i<4:  
 e-=0.5\*I[x][i]\*h[x][i]  
 i+=1  
 x+=1

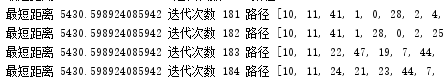
**实验效果和分析：**

分别使用同样的50个城市数据对GA和ASO进行测试（样本在源码文件夹中）：

样本1：

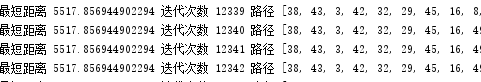
使用蚁群算法时，在迭代150次左右就能够得到最优解

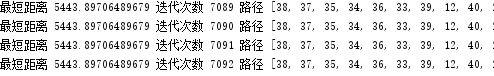




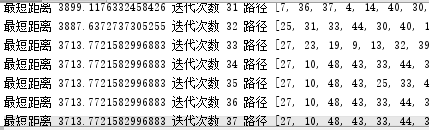
使用遗传算法时，要迭代超过5000次才能得到次优解

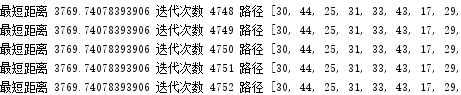
因为第一次迭代效果很差所以显示初始设置的最短距离

并且每次运行的效果差很多，有可能10000次才能得到次优解，也有可能7000次就得到更优解



样本2：

ACO：

GA：

GA算法和ACO算法都能比较快的找到TSP问题的解，两个算法找到最优解花销的时间差不多（在我的代码和设备上如此），虽然GA算法迭代的次数更多但是GA算法每次迭代需要的时间更少。

对于ACO算法，最开始的选择下一个城市的思路是选择前往概率最大的城市，这种思路的效果不好，路径得不到优化，后来采取随机生成某个概率，这个概率小于前往其他城市的概率和，依次与前往其他城市的概率相减，小于0时对应的城市作为下一城市，这种思路得到的效果比较好，个人认为是添加了随机因素，使解的范围更广的原因。

对于4个城市的hopfield方法解决TSP问题，我尝试了多组数据，但是没有得到解，我所得到的连接权值矩阵是对称矩阵且 对角元素为0，能量函数也会逐渐减小到某个值，神经网络趋于稳定后得不到解（不保证每行每列只有一个神经元活动）

