**蚁群算法**

蚁群算法是一种模拟进化算法，主要目的是在图中寻找优化路径的机率算法。

蚁群算法最早是为了解决TSP问题（即旅行商问题）。

TSP问题的要求：

1. 路径的限制是每个城市只能拜访一次；
2. 最后要回到原来发出的城市。
3. 求得的路径路程为所有路径之中的最小值。

概念原型：各个蚂蚁在没有实现告诉他们食物在什么地方的前提下开始寻找食物。当一只找到食物以后，它会向环境释放一种挥发性分泌物（信息素）来实现，吸引其他的蚂蚁过来，这样越来越多的蚂蚁会找到食物。有些蚂蚁并没有像其它蚂蚁一样总重复同样的路，他们会另辟蹊径，如果另开辟的道路比原来的其他道路更短，那么，渐渐地，更多的蚂蚁被吸引到这条较短的路上来。

最后，经过一段时间运行，就可能会出现一条最短的路径被大多数蚂蚁重复着。

蚁群算法的基本原理：

1. 蚂蚁在路径上释放信息素。
2. 碰到还没走过的路口，就随机挑选一条路走。同时，释放与路径长度有关的信息素。
3. 信息素浓度与路径长度成反比，后来的蚂蚁再次碰到该路口时，就选择信息素浓度较高路径。
4. 最优路径上的信息素浓度越来越大。
5. 最终蚁群找到最优寻食路径。

基于TSP问题的基本蚁群算法：

TSP求解中，假设蚁群算法中的每只蚂蚁是具有以下特征的简单智能体：每次周游，每只蚂蚁在其经过的支路（i,j）上都留下信息素‚蚂蚁选择城市的概率与城市之间的距离和当前连接支路上所包含的信息素余量有关。ƒ为了强制蚂蚁进行合法的周游，直到一次周游完成后，才允许蚂蚁游走已访问过的城市（这可由禁忌表来控制）。

1. 状态转移

为了避免残留信息素过多而淹没启发信息，在每只蚂蚁走完一步或者完成对所有n个城市的遍历(也即一个循环结束)后，要对残留信息进行更新处理

(2)信息素更新模型

蚁周模型（Ant-Cycle模型）

蚁量模型（Ant-Quantity模型）

蚁密模型（Ant-Density模型）

区别：1.蚁周模型利用的是全局信息，即蚂蚁完成一个循环后更新所有路径上的信息素；

2.蚁量和蚁密模型利用的是局部信息，即蚂蚁完成一步后更新路径上的信息素。

开始

初始化Nc=0

循环次数Nc-Nc+1

N

Y

满足结束条件

N

Y

残留信息量更新

结束

输出程序计算结果

蚂蚁个数k=1

K=k+1

计算状态转移概率，选择下一元素

修改禁忌表

K>=m? 

蚁群算法中主要参数的选择

α-定义：信息启发式因子，取值范围[0,5]

α值越大，蚂蚁选择之前走过的路径可能性就越大，搜索路径的随机性减弱，α越小，蚁群搜索范围就会减少，容易陷入局部最优

β-定义：期望启发式因子,取值范围[0,5]

β值越大，蚁群就越容易选择局部较短路径，这时算法的收敛速度是加快了，但是随机性缺不高，容易得到局部的相对最优

m-定义：蚁群数量, 取值范围[10,10000]

m数量越多，得到的最优解就越精确， 但是会产生不少重复解，随着算法接近最优质的的收敛，信息正反馈作用降低，大量的重复工作，消耗了资源，增加了时间复杂度。

ρ-定义：信息挥发因子，1-ρ表示残留因子, 取值范围[0.1,0.99]

ρ过小时，在各路径上残留的信息素过多，导致无效的路径继续被搜索，影响到算法的收敛速度；ρ过大，无效的路径虽然可以被排除搜索，但是不能保证有效的路径也会被放弃搜索，影响到最优值的搜索。

代码：实现求50个城市之间最短距离的代码

*# -\*- coding: utf-8 -\*-***import** random  
**import** copy  
**import** time  
**import** sys  
**import** math  
**import** tkinter *#//GUI模块***import** threading  
**from** functools **import** reduce(ALPHA, BETA, RHO, Q) = (1.0,2.0,0.5,100.0)  
*# 城市数，蚁群*(city\_num, ant\_num) = (50,50)  
distance\_x = [  
 178,272,176,171,650,499,267,703,408,437,491,74,532,  
 416,626,42,271,359,163,508,229,576,147,560,35,714,  
 757,517,64,314,675,690,391,628,87,240,705,699,258,  
 428,614,36,360,482,666,597,209,201,492,294]  
distance\_y = [  
 170,395,198,151,242,556,57,401,305,421,267,105,525,  
 381,244,330,395,169,141,380,153,442,528,329,232,48,  
 498,265,343,120,165,50,433,63,491,275,348,222,288,  
 490,213,524,244,114,104,552,70,425,227,331]  
*#城市距离和信息素*distance\_graph = [ [0.0 **for** col **in** range(city\_num)] **for** raw **in** range(city\_num)]  
pheromone\_graph = [ [1.0 **for** col **in** range(city\_num)] **for** raw **in** range(city\_num)]  
*#----------- 蚂蚁 -----------***class** Ant(object):  
 *# 初始化* **def** \_\_init\_\_(self,ID):  
 self.ID = ID *# ID* self.\_\_clean\_data() *# 随机初始化出生点  
 # 初始数据* **def** \_\_clean\_data(self):  
 self.path = [] *# 当前蚂蚁的路径* self.total\_distance = 0.0 *# 当前路径的总距离* self.move\_count = 0 *# 移动次数* self.current\_city = -1 *# 当前停留的城市* self.open\_table\_city = [**True for** i **in** range(city\_num)] *# 探索城市的状态* city\_index = random.randint(0,city\_num-1) *# 随机初始出生点* self.current\_city = city\_index  
 self.path.append(city\_index)  
 self.open\_table\_city[city\_index] = **False** self.move\_count = 1  
 *# 选择下一个城市* **def** \_\_choice\_next\_city(self):  
 next\_city = -1  
 select\_citys\_prob = [0.0 **for** i **in** range(city\_num)] *#存储去下个城市的概率* total\_prob = 0.0  
 *# 获取去下一个城市的概率* **for** i **in** range(city\_num):  
 **if** self.open\_table\_city[i]:  
 **try** :  
 *# 计算概率：与信息素浓度成正比，与距离成反比* select\_citys\_prob[i] = pow(pheromone\_graph[self.current\_city][i], ALPHA) \* pow((1.0/distance\_graph[self.current\_city][i]), BETA)  
 total\_prob += select\_citys\_prob[i]  
 **except** ZeroDivisionError **as** e:  
 print (**'Ant ID: {ID}, current city: {current}, target city: {target}'**.format(ID = self.ID, current = self.current\_city, target = i))  
 sys.exit(1)  
 *# 轮盘选择城市* **if** total\_prob > 0.0:  
 *# 产生一个随机概率,0.0-total\_prob* temp\_prob = random.uniform(0.0, total\_prob)  
 **for** i **in** range(city\_num):  
 **if** self.open\_table\_city[i]:  
 *# 轮次相减* temp\_prob -= select\_citys\_prob[i]  
 **if** temp\_prob < 0.0:  
 next\_city = i  
 **break** *# 未从概率产生，顺序选择一个未访问城市  
 # if next\_city == -1:  
 # for i in range(city\_num):  
 # if self.open\_table\_city[i]:  
 # next\_city = i  
 # break* **if** (next\_city == -1):  
 next\_city = random.randint(0, city\_num - 1)  
 **while** ((self.open\_table\_city[next\_city]) == **False**): *# if==False,说明已经遍历过* next\_city = random.randint(0, city\_num - 1)  
 *# 返回下一个城市序号* **return** next\_city  
 *# 计算路径总距离* **def** \_\_cal\_total\_distance(self):  
 temp\_distance = 0.0  
  
 **for** i **in** range(1, city\_num):  
 start, end = self.path[i], self.path[i-1]  
 temp\_distance += distance\_graph[start][end]  
 *# 回路* end = self.path[0]  
 temp\_distance += distance\_graph[start][end]  
 self.total\_distance = temp\_distance  
 *# 移动操作* **def** \_\_move(self, next\_city):  
 self.path.append(next\_city)  
 self.open\_table\_city[next\_city] = **False** self.total\_distance += distance\_graph[self.current\_city][next\_city]  
 self.current\_city = next\_city  
 self.move\_count += 1  
 *# 搜索路径* **def** search\_path(self):  
 *# 初始化数据* self.\_\_clean\_data()  
 *# 搜素路径，遍历完所有城市为止* **while** self.move\_count < city\_num:  
 *# 移动到下一个城市* next\_city = self.\_\_choice\_next\_city()  
 self.\_\_move(next\_city)  
 *# 计算路径总长度* self.\_\_cal\_total\_distance()  
*#----------- TSP问题 -----------***class** TSP(object):  
 **def** \_\_init\_\_(self, root, width = 800, height = 600, n = city\_num):  
 *# 创建画布* self.root = root  
 self.width = width  
 self.height = height  
 *# 城市数目初始化为city\_num* self.n = n  
 *# tkinter.Canvas* self.canvas = tkinter.Canvas(  
 root,  
 width = self.width,  
 height = self.height,  
 bg = **"#EBEBEB"**, *# 背景白色* xscrollincrement = 1,  
 yscrollincrement = 1  
 )  
 self.canvas.pack(expand = tkinter.YES, fill = tkinter.BOTH)  
 self.title(**"TSP蚁群算法(n:初始化 e:开始搜索 s:停止搜索 q:退出程序)"**)  
 self.\_\_r = 5  
 self.\_\_lock = threading.RLock() *# 线程锁* self.\_\_bindEvents()  
 self.new()  
 *# 计算城市之间的距离* **for** i **in** range(city\_num):  
 **for** j **in** range(city\_num):  
 temp\_distance = pow((distance\_x[i] - distance\_x[j]), 2) + pow((distance\_y[i] - distance\_y[j]), 2)  
 temp\_distance = pow(temp\_distance, 0.5)  
 distance\_graph[i][j] =float(int(temp\_distance + 0.5))  
 *# 按键响应程序* **def** \_\_bindEvents(self):  
 self.root.bind(**"q"**, self.quite) *# 退出程序* self.root.bind(**"n"**, self.new) *# 初始化* self.root.bind(**"e"**, self.search\_path) *# 开始搜索* self.root.bind(**"s"**, self.stop) *# 停止搜索  
 # 更改标题* **def** title(self, s):  
 self.root.title(s)  
 *# 初始化* **def** new(self, evt = **None**):  
 *# 停止线程* self.\_\_lock.acquire()  
 self.\_\_running = **False** self.\_\_lock.release()  
 self.clear() *# 清除信息* self.nodes = [] *# 节点坐标* self.nodes2 = [] *# 节点对象  
 # 初始化城市节点* **for** i **in** range(len(distance\_x)):  
 *# 在画布上随机初始坐标* x = distance\_x[i]  
 y = distance\_y[i]  
 self.nodes.append((x, y))  
 *# 生成节点椭圆，半径为self.\_\_r* node = self.canvas.create\_oval(x - self.\_\_r,  
 y - self.\_\_r, x + self.\_\_r, y + self.\_\_r,  
 fill = **"#ff0000"**, *# 填充红色* outline = **"#000000"**, *# 轮廓白色* tags = **"node"**,  
 )  
 self.nodes2.append(node)  
 *# 显示坐标* self.canvas.create\_text(x,y-10, *# 使用create\_text方法在坐标（302，77）处绘制文字* text = **'('**+str(x)+**','**+str(y)+**')'**, *# 所绘制文字的内容* fill = **'black'** *# 所绘制文字的颜色为灰色* )  
 *# 顺序连接城市  
 #self.line(range(city\_num))  
 # 初始城市之间的距离和信息素* **for** i **in** range(city\_num):  
 **for** j **in** range(city\_num):  
 pheromone\_graph[i][j] = 1.0  
 self.ants = [Ant(ID) **for** ID **in** range(ant\_num)] *# 初始蚁群* self.best\_ant = Ant(-1) *# 初始最优解* self.best\_ant.total\_distance = 1 << 31 *# 初始最大距离* self.iter = 1 *# 初始化迭代次数  
 # 将节点按order顺序连线* **def** line(self, order):  
 *# 删除原线* self.canvas.delete(**"line"**)  
 **def** line2(i1, i2):  
 p1, p2 = self.nodes[i1], self.nodes[i2]  
 self.canvas.create\_line(p1, p2, fill = **"#000000"**, tags = **"line"**)  
 **return** i2  
 *# order[-1]为初始值* reduce(line2, order, order[-1])  
 *# 清除画布* **def** clear(self):  
 **for** item **in** self.canvas.find\_all():  
 self.canvas.delete(item)  
 *# 退出程序* **def** quite(self, evt):  
 self.\_\_lock.acquire()  
 self.\_\_running = **False** self.\_\_lock.release()  
 self.root.destroy()  
 print (**u"\n程序已退出..."**)  
 sys.exit()  
 *# 停止搜索* **def** stop(self, evt):  
 self.\_\_lock.acquire()  
 self.\_\_running = **False** self.\_\_lock.release()  
 *# 开始搜索* **def** search\_path(self, evt = **None**):  
 *# 开启线程* self.\_\_lock.acquire()  
 self.\_\_running = **True** self.\_\_lock.release()  
 **while** self.\_\_running:  
 *# 遍历每一只蚂蚁* **for** ant **in** self.ants:  
 *# 搜索一条路径* ant.search\_path()  
 *# 与当前最优蚂蚁比较* **if** ant.total\_distance < self.best\_ant.total\_distance:  
 *# 更新最优解* self.best\_ant = copy.deepcopy(ant)  
 *# 更新信息素* self.\_\_update\_pheromone\_gragh()  
 print (**u"迭代次数："**,self.iter,**u"最佳路径总距离："**,int(self.best\_ant.total\_distance))  
 *# 连线* self.line(self.best\_ant.path)  
 *# 设置标题* self.title(**"TSP蚁群算法(n:随机初始 e:开始搜索 s:停止搜索 q:退出程序) 迭代次数: %d"** % self.iter)self.canvas.update() *# 更新画布*  
 self.iter += 1**def** \_\_update\_pheromone\_gragh(self): *# 更新信息素*  
 *# 获取每只蚂蚁在其路径上留下的信息素* temp\_pheromone = [[0.0 **for** col **in** range(city\_num)] **for** raw **in** range(city\_num)]  
 **for** ant **in** self.ants:  
 **for** i **in** range(1,city\_num):  
 start, end = ant.path[i-1], ant.path[i]  
 *# 在路径上的每两个相邻城市间留下信息素，与路径总距离反比* temp\_pheromone[start][end] += Q / ant.total\_distance  
 temp\_pheromone[end][start] = temp\_pheromone[start][end]  
 *# 更新所有城市之间的信息素，旧信息素衰减加上新迭代信息素* **for** i **in** range(city\_num):  
 **for** j **in** range(city\_num):  
 pheromone\_graph[i][j] = pheromone\_graph[i][j] \* RHO + temp\_pheromone[i][j]  
 *# 主循环* **def** mainloop(self):  
 self.root.mainloop()  
*#----------- 程序的入口处 -----------***if** \_\_name\_\_ == **'\_\_main\_\_'**:TSP(tkinter.Tk()).mainloop()