## **广州大学学生实验报告**

**开课学院及实验室：**计算机科学与工程实验室 **2021年11月4日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机科学与网络工程学院** | **年级/专业/班** | **网络193** | **姓名** | 吴伟俊 | **学号** | 1906200107 |
| **实验课程名称** | **人工智能导论实验** | | | | | **成绩** |  |
| **实验项目名称** | **TSP问题的遗传算法实现** | | | | | **指导老师** | 张少宏 |

(\*\*\*报告只能为文字和图片,老师评语将添加到此处,学生请勿作答\*\*\*)

**一、实验内容**

**问题描述**：旅行商问题，即TSP问题（Traveling Salesman Problem）又译为旅行推销员问题、货郎担问题，是数学领域中著名问题之一。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。

**内容提要**：以N个节点的TSP（旅行 商问题）问题为例，应用遗传算法并用选定的编程语言，设计简单的遗传优化系统对问题进行求解，求出问题的最优解，通过实验培养学生利用遗传算法进行问题求解的基本技能。

**二、实验设备**

1. 实验设备：计算机；

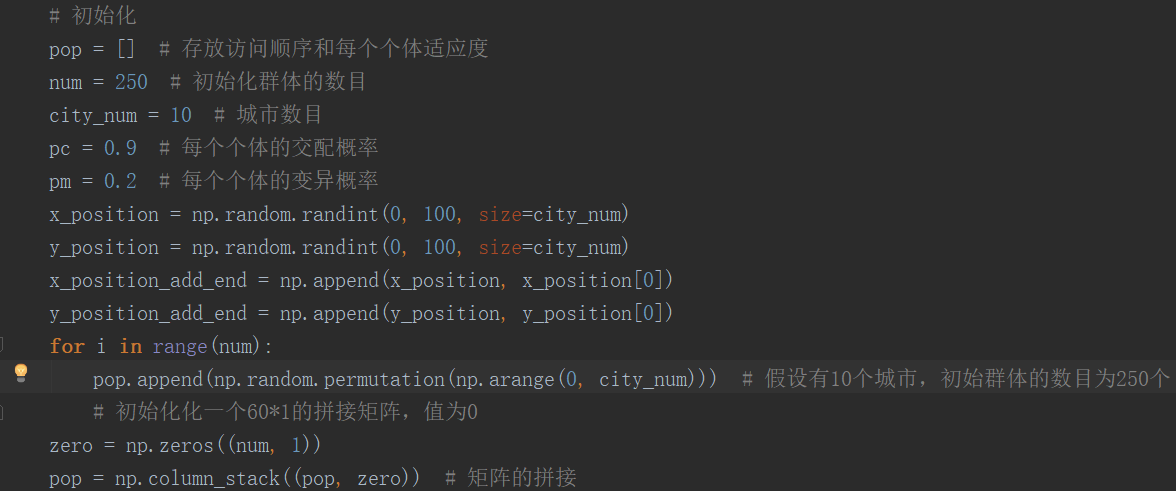
2. 平台：Windows操作系统，vscode

**三、实验步骤**

* 随机生成N个二维坐标节点。

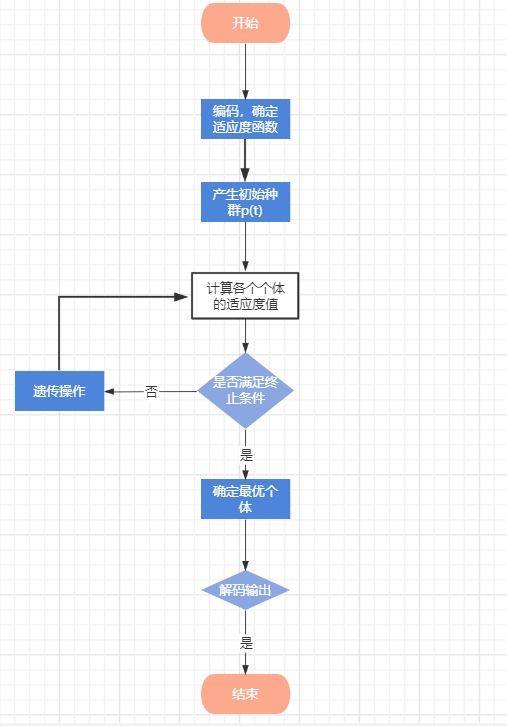
先进行初始化，随机生成矩阵pop，设置10个城市，250个群体数目

代码如下：



* 应用遗传算法并用选定的编程语言，设计简单的遗传优化系统对问题进行求解，求出问题的最优解。

算法流程图如下：

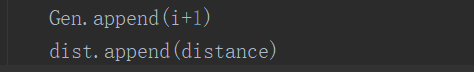


* 选择适当可视化方法显示结果。

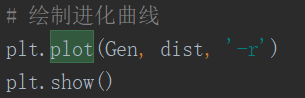
引入了matplotlib库，内含可视化的各种方法，比如，调用plt.show()方法可以显示绘制图像。

相关代码：

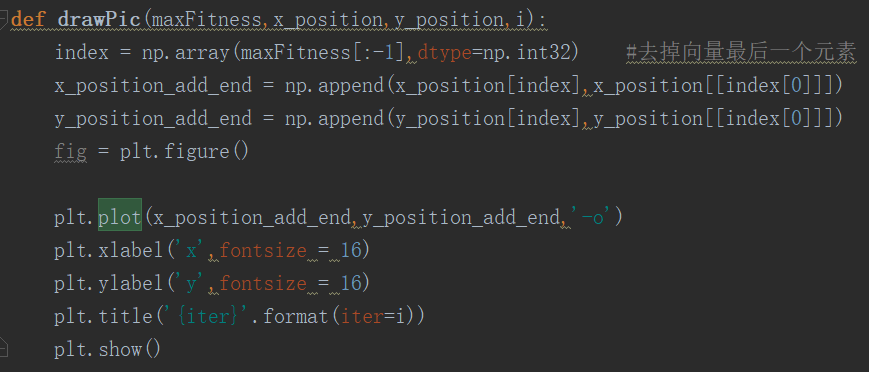
将遗传代数，每次的最优距离存储到数组中



绘制图像：



同时，也可以输出每次的各个城市的分布：



* 分析适应度函数对启发式搜索算法的影响。

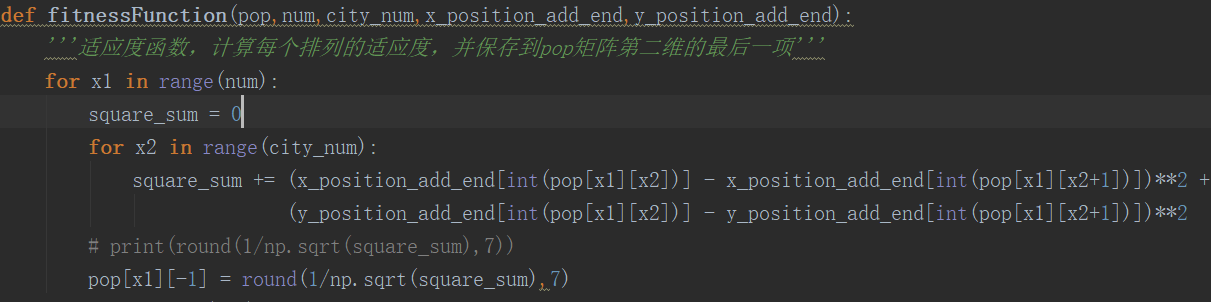
由于m，n城市之间的距离是sqrt((xm-xn)^2+(ym-yn)^2)，且m，n城市之间有t-1组，故目标函数为sqrt((xm-xn)^2+(ym-yn)^2)从1~t-2组的结果之和。

遗传算法仅以适应度函数为依据,利用种群中每个个体的适应度值来进行搜索。适应度函数即为路径总长度的倒数。因此适应度函数的值越小，路径总长度越大。

其次，适应度函数是对群体个体的评价指标，能反应该个体的好坏。我们根据这个指标就可以判断群体中的个体是否好坏，从而将一些个体淘汰，进而进行自然选择。

//以下相关代码是将每个排列的适应度都存储在pop矩阵的最后一项中，这样能方便后边求取最大适应度，以及输出最优解，并绘制图像。

相关代码：

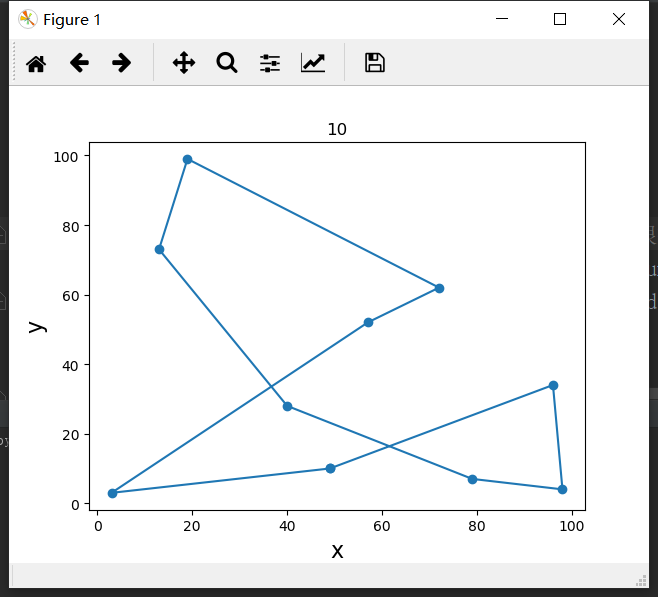


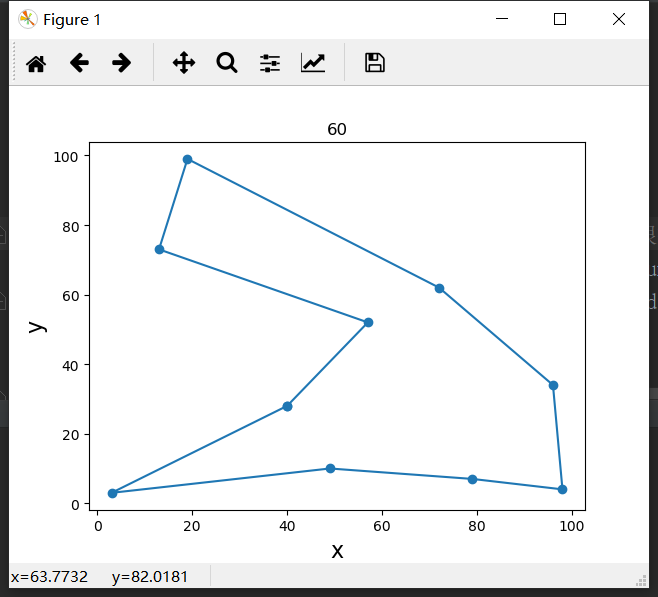
* \*扩展选做题：考虑不同数值N对最终结果和求解性能的影响。对于比较大的N，是否设计更快速的近似方法代替原有算法。

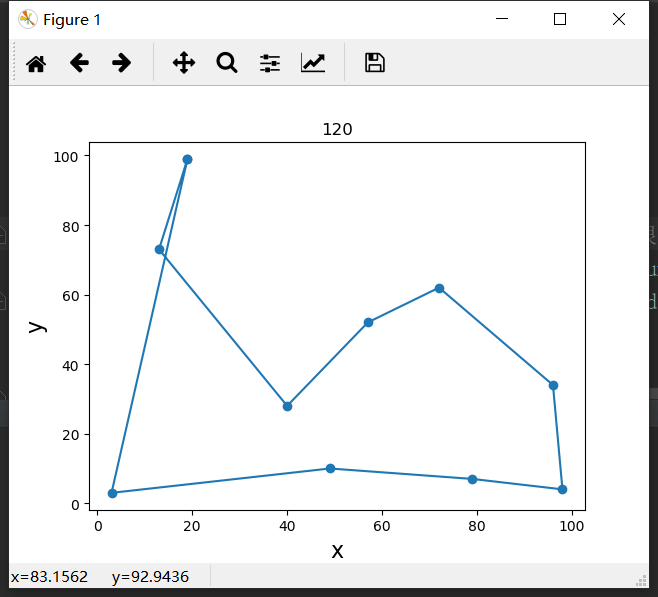
**四、分析说明**

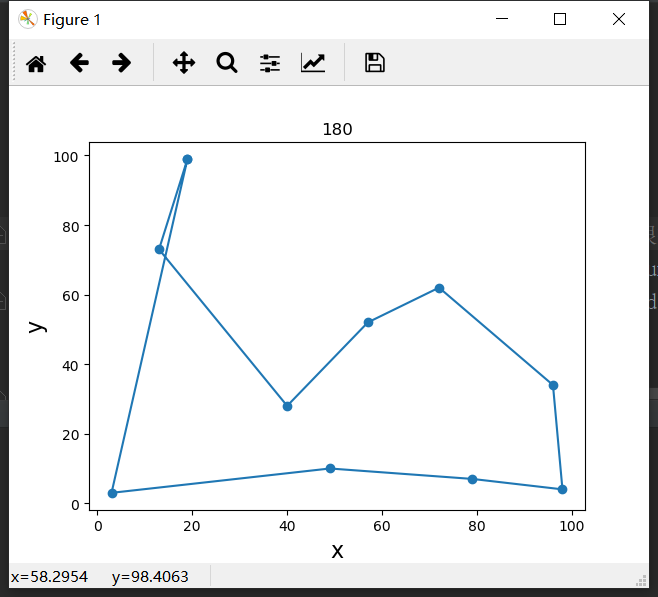
* 实验结果：

遗传代数以及相应城市的分布图，以10，60，120，180代为例：

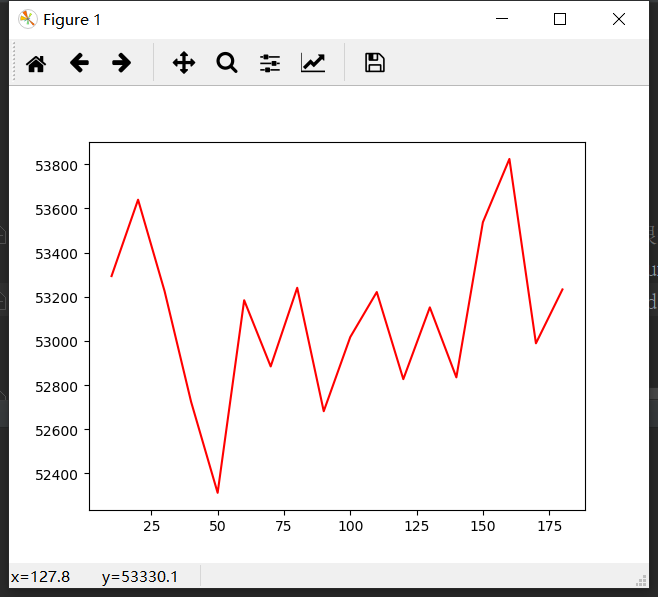




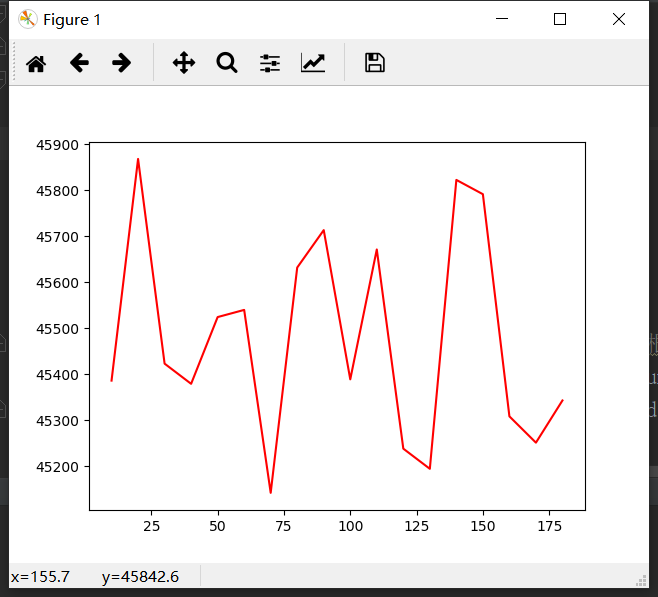
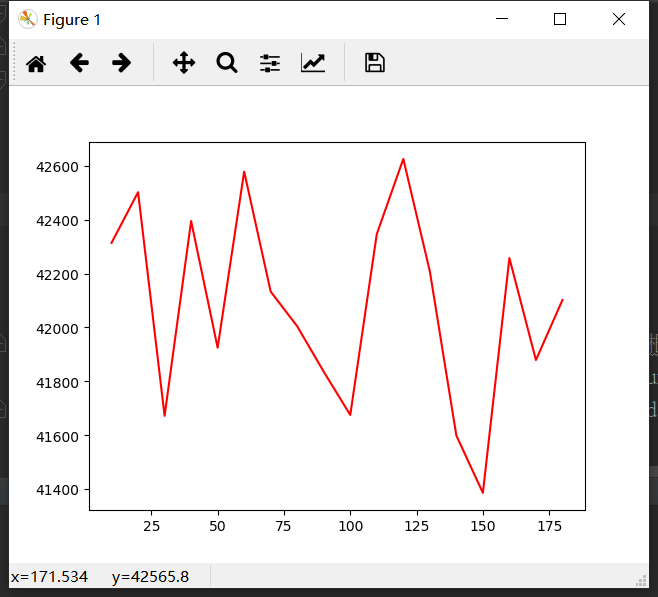




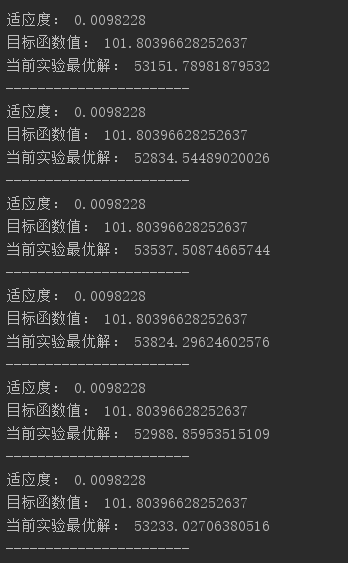
最优解以及相应的遗传代数的关系：



此处我多实验了几次，发现总是趋于某个最优解。



可以看出，随着遗传代数的增加，虽然最优解有波动，但逐步趋于某个值。



部分值如图所示，可见实验最优解趋于53000.

**实验总结：**

通过本次实验，我了解并熟悉遗传算法的原理以及使用，并通过遗传算法解决了TSP问题，并体会到了适应度函数对遗传算法的影响。学习到了遗传算法的思想，优胜略汰，自然选择。同时，该算法的适应度函数是运用了伪欧氏距离进行求解，和上一次实验有了联系，让我觉得每做一个实验就是在筑起一个地基。总而言之，希望能在今后的实验中学到更多有用的知识。

**函数源代码如下：**

#from itertools import permutations

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

#from itertools import combinations, permutations

def fitnessFunction(pop,num,city\_num,x\_position\_add\_end,y\_position\_add\_end):

'''适应度函数，计算每个排列的适应度，并保存到pop矩阵第二维的最后一项'''

for x1 in range(num):

square\_sum = 0

for x2 in range(city\_num):

square\_sum += (x\_position\_add\_end[int(pop[x1][x2])] - x\_position\_add\_end[int(pop[x1][x2+1])])\*\*2 +\

(y\_position\_add\_end[int(pop[x1][x2])] - y\_position\_add\_end[int(pop[x1][x2+1])])\*\*2

# print(round(1/np.sqrt(square\_sum),7))

pop[x1][-1] = round(1/np.sqrt(square\_sum),7)

def choiceFuction(pop):

'''

这里的做法：比如A当前种群中的最优解，B为经过交叉、变异后的最差解，把A作为最当前代中的最优解保存下来作为这一代的最优解，同时A也参与交叉

和变异。经过交叉、变异后的最差解为B，那么我再用A替代B。

:argument pop矩阵

:return 本代适应度最低的个体的索引值和本代适应度最高的个体

'''

yield np.argmin(pop[:, -1])

yield pop[np.argmax(pop[:, -1])]

def choice(pop,num,city\_num,x\_position\_add\_end,y\_position\_add\_end,b):

fitnessFunction(pop,num,city\_num,x\_position\_add\_end,y\_position\_add\_end)

c,d =choiceFuction(pop)

# 上一代的最优值替代本代中的最差值

pop[c] = b

return pop

def drawPic(maxFitness,x\_position,y\_position,i):

index = np.array(maxFitness[:-1],dtype=np.int32) #去掉向量最后一个元素

x\_position\_add\_end = np.append(x\_position[index],x\_position[[index[0]]])

y\_position\_add\_end = np.append(y\_position[index],y\_position[[index[0]]])

fig = plt.figure()

plt.plot(x\_position\_add\_end,y\_position\_add\_end,'-o')

plt.xlabel('x',fontsize = 16)

plt.ylabel('y',fontsize = 16)

plt.title('{iter}'.format(iter=i))

plt.show()

def matuingFuction(pop, pc, city\_num, pm, num):

mating\_matrix = np.array(1 - (np.random.rand(num) > pc), dtype=np.bool) # 交配矩阵，如果为true则进行交配

a = list(pop[mating\_matrix][:, :-1]) # 进行交配的个体

b = list(pop[np.array(1 - mating\_matrix, dtype=bool)][:, :-1]) # 未进行交配的个体,直接放到下一代

b = [list(i) for i in b] # 对b进行类型转换，避免下面numpy.array 没有index属性

# print(a)

if len(a) % 2 != 0:

b.append(a.pop())

# print('ab的长度：',len(a),len(b))

for i in range(int(len(a) / 2)):

# 随机初始化两个交配点,这里写得不好，这边的两个点初始化都是一个在中间位置偏左，一个在中间位置偏右

p1 = np.random.randint(1, int(city\_num / 2) + 1)

p2 = np.random.randint(int(city\_num / 2) + 1, city\_num)

x1 = list(a.pop())

x2 = list(a.pop())

matuting(x1, x2, p1, p2)

# 交配之后产生的个体进行一定概率上的变异

variationFunction(x1, pm, city\_num)

variationFunction(x2, pm, city\_num)

b.append(x1)

b.append(x2)

zero = np.zeros((num, 1))

# print('b的形状：',len(b))

temp = np.column\_stack((b, zero))

return temp

def matuting(x1, x2, p1, p2):

# 以下进行交配

# 左边交换位置

temp = x1[:p1]

x1[:p1] = x2[:p1]

x2[:p1] = temp

# 右边交换位置

temp = x1[p2:]

x1[p2:] = x2[p2:]

x2[p2:] = temp

# 寻找重复的元素

center1 = x1[p1:p2]

center2 = x2[p1:p2]

while True: # x1左边

for i in x1[:p1]:

if i in center1:

# print(center1.index(i)) # 根据值找到索引

x1[x1[:p1].index(i)] = center2[center1.index(i)]

break

if np.intersect1d(x1[:p1], center1).size == 0: # 如果不存在交集，则循环结束

break

while True: # x1右边

for i in x1[p2:]:

if i in center1:

# print(center1.index(i)) # 根据值找到索引

x1[x1[p2:].index(i) + p2] = center2[center1.index(i)]

# print(x1)

break

if np.intersect1d(x1[p2:], center1).size == 0: # 如果不存在交集，则循环结束

break

while True: # x2左边

for i in x2[:p1]:

if i in center2:

# print(center2.index(i)) # 根据值找到索引

x2[x2[:p1].index(i)] = center1[center2.index(i)]

break

if np.intersect1d(x2[:p1], center2).size == 0: # 如果不存在交集，则循环结束

break

while True: # x2右边

for i in x2[p2:]:

if i in center2:

# print(center2.index(i)) # 根据值找到索引

x2[x2[p2:].index(i) + p2] = center1[center2.index(i)]

# print(x2)

break

if np.intersect1d(x2[p2:], center2).size == 0: # 如果不存在交集，则循环结束

break

def variationFunction(list\_a, pm, city\_num):

'''变异函数'''

if np.random.rand() < pm:

p1 = np.random.randint(1, int(city\_num / 2) + 1)

p2 = np.random.randint(int(city\_num / 2) + 1, city\_num)

# print(p1,p2)

temp = list\_a[p1:p2]

temp.reverse()

list\_a[p1:p2] = temp

# print(list\_a)

def findDistance(pop):

sum=0

for temp in pop:

sum += 1/temp[-1]

return sum

def main():

Gen = [] # 代数

dist = [] # 每一代的最优距离

# 初始化

pop = [] # 存放访问顺序和每个个体适应度

num = 250 # 初始化群体的数目

city\_num = 10 # 城市数目

pc = 0.9 # 每个个体的交配概率

pm = 0.2 # 每个个体的变异概率

x\_position = np.random.randint(0, 100, size=city\_num)

y\_position = np.random.randint(0, 100, size=city\_num)

x\_position\_add\_end = np.append(x\_position, x\_position[0])

y\_position\_add\_end = np.append(y\_position, y\_position[0])

for i in range(num):

pop.append(np.random.permutation(np.arange(0, city\_num))) # 假设有10个城市，初始群体的数目为250个

# 初始化化一个60\*1的拼接矩阵，值为0

zero = np.zeros((num, 1))

pop = np.column\_stack((pop, zero)) # 矩阵的拼接

fitnessFunction(pop, num, city\_num, x\_position\_add\_end, y\_position\_add\_end)

for i in range(180):

a, b = choiceFuction(pop) # a 为当代适应度最小的个体的索引，b为当代适应度最大的个体,这边要保留的是b

# pop[a]=b

# print(x\_position\_add\_end," ",y\_position\_add\_end)

if (i + 1) % 10 == 0:

distance = findDistance(pop)

print("适应度：", b[-1]);

print("目标函数值：", 1 / b[-1]);

print("当前实验最优解：",distance)

print("-----------------------")

Gen.append(i+1)

dist.append(distance)

#drawPic(b, x\_position, y\_position, i + 1) # 根据本代中的适应度最大的个体画图

pop\_temp = matuingFuction(pop, pc, city\_num, pm, num) # 交配变异

pop = choice(pop\_temp, num, city\_num, x\_position\_add\_end, y\_position\_add\_end, b)

# 绘制进化曲线

plt.plot(Gen, dist, '-r')

plt.show()

main()