进化算法：遗传算法解决“旅行商”问题

# 1实验目的

1. 了解进化算法的产生与发展；
2. 了解进化算法的设计原则；
3. 了解选择、交叉、变异三种基本的遗传算子；
4. 了解遗传算法的基本框架；
5. 学会用遗传算法解决复杂问题。

# 2实验要求

本次试验后，要求学生能：

1. 理解遗传算法的原理；
2. 理解遗传算法的实现流程；
3. 能编程实现选择、交叉、变异三种基本的遗传算子；
4. 编写python程序，用遗传算法解决“旅行商”问题。

# 3实验原理

## 3.1实验介绍

本实验是运用遗传算法解决“旅行商”问题：

一位旅行商要去往n个城市经商，恰好访问每个城市一次，并最终回到出发城市。设旅行商从第i个城市前往第j个城市的开销为COSTij，周游完全部n个城市并回到原点的总开销为COST总，市编写程序求COST总的最小值。

## 3.2进化算法

进化算法是一种基于自然选择、自然遗传等生物进化机制的搜索算法。借鉴了大自然中生物的进化操作，它一般包括基因编码、种群初始化、交叉变异算子、选择机制等基本操作。与传统的基于微积分的方法和穷举方法等优化算法相比，进化计算是一种成熟的具有高鲁棒性和广泛适用性的全局优化方法，具有自组织、自适应、自学习的特性，能够不受问题性质的限制，有效地处理传统优化算法难以解决的复杂问题（比如NP难优化问题）。

下面介绍一种最基本的进化算法框架。

## 3.3基本遗传算法

### 3.3.1编码

即将问题空间的参数通过编码表示成遗传空间的染色体或个体。主要有串位编码、实数编码、多参数级联编码等方式。本次实验是采用了实数编码，更加直观清晰，将旅行商的一种周游路径看作一条染色体，路径上的城市序号即对应染色体上的基因。需要注意的是，本问题中染色体上的基因为1~n且不可重复，否则不满足旅行商问题的条件。

### 3.3.2适应度函数

遗传算啊遵循自然界优胜劣汰的原则，在进化搜索上不用其他外部信息，而是用适应度表示个体的优劣，作为选择操作的依据。一般用目标函数映射为适应度函数，例如本题中，为了得到最短的路径，取路径总长度的倒数为适应度函数。

为了避免适应度过高的个体，影响遗传的随机性，应该对适应度函数做一定的制度变化，让适应度函数的值较均匀地分布在一个范围较小的值域内。

### 3.3.3选择

选择操作是从当前群体中，按照一个与适应度相关的概率选出优良的个体。让他们繁衍下一代。

本实验是是采用适应度比例法（选择个体的概率为当前个体适应度占适应度总和的比例），用轮盘赌选的方式选择出可以繁衍下一代的个体。

### 3.3.4交叉

模拟生物的交配，将从上一步选择出的待繁殖的种群中依次随机取出两个染色体，让他们相互混合，产生一对由双方基因组成的新染色体。交叉产生的个体可能比上一代更优秀，也可能比上一代更差，但是基因越好的个体，越有概率把自己的优良基因传给下一代。

本实验是按概率pc产生交叉，采用了两点交换，在两条亲本染色体上随机生成两个交叉点，将交叉点内的部分交换。

根据具体问题不同，交叉之后需要对不合法的染色体进行修正，例如，本实验中染色体代表旅行商周游城市的路径，每一个城市经过且只经过一次，而交换两条路径中的部分路径必然会导致交换后的路径中出现重复的城市，或出现没访问的城市，需要按照一定策略进行修正（详见4.5）

### 3.3.5异变

变异是将个体编码中的一些为进行随机变化，属于辅助性的搜索操作，一般变异的概率设置得较小。

本实验是按pm发生变异，随机生成两个变异点，将变异点上的基因交换。

# 4实验步骤

## 4.1安装依赖

本实验需要用到python的扩展程序库Numpy进行数值计算，如果没有安装过这个库，使用国内的清华pip源安装，命令如下：

pip3 install numpy scipy matplotlib -i <https://pypi.tuna.tsinghua.edu.cn/simple>

导入依赖库，其中copy是python内建模块，本实验要用到copy.deepcopy()进行深度复制，无需额外下载，可以直接import。

import numpy as np

import copy

## 4.2实现遗传算法

### 4.2.1生成初始种群

每一个染色体代表一条路径。生成1 ~city\_size的数组，打乱顺序后加入pop中，直到pop中的个体数达到pop\_size个。

def create\_pop(pop\_size):

pop = []

for i in range(pop\_size):

path = np.arange(1, map.shape[0])

np.random.shuffle(path) # 随机打乱数组

pop.append(np.append(path, 0)) # 加入种群

return np.array(pop)

### 4.2.2计算个体的适应值

（1）先计算该个体染色体代表的路径的距离

需要注意的是，旅行商问题是求旅行商周游一整圈的总距离，即，走到最后一个城市后还要回到原点。

def get\_distance(path):

distance = 0

# i从-1开始，先计算了从最后一个城市回来的距离

for i in range(- 1, len(path) - 1):

distance += map[path[i]][path[i + 1]]

return distance

（2）计算每个个体的适应值

总路径越短越好，所以距离越短，适应值越高，从而可以得出适应值应和距离成反比。将计算出的适应值保存在fitness数组中。

def get\_fitness(pop):

f = np.array([])

for i in range(pop.shape[0]):

path = pop[i]

fi = 1 / get\_distance(path) # 因为要求最短路径，所以适应度函数设为路径总距离的倒数

f = np.append(f, fi)

return f

### 4.2.3选择

每次按概率从种群中选择适应值高的个体保存下来繁衍下一代。由于是按概率选择，适应值高的并不一定被保留，适应值低的并不一定被淘汰，符合自然规律，也可以避免陷入局部最佳值，保证了种群的随机性。

这里用到numpy下的np.choice()函数，从当前的pop中按概率数组possibility = f / f.sum()选择pop\_size个个体组成新种群。

def select\_pop(pop, f):

possibility = f / f.sum()

chosen = np.random.choice(np.arange(pop.shape[0]), size = pop\_size, replace = False, p = possibility)

pop\_new = pop[chosen, :]

return pop\_new

### 4.2.4交叉

（1）找出个体中重复的基因

即找出个体所代表的路径中重复经过的城市，用于修正交叉后的染色体，因为旅行商问题中不允许重复访问城市，每个城市都要访问且仅访问一遍

def find\_repeat(path, index1, index2):

for k in path:

if path.count(k) > 1:

return path.index(k)

return None

（2）判断是否交叉

按照概率pc进行交叉操作，生成一个0~1的随机数，如果随机数小于pc，则不进行交叉操作。

（3）两点交叉

两点交叉是在染色体上随机生成两个交叉点a、b，将两条染色体在交叉点（a，b）内的部分互换

（4）修正交叉出的新染色体

两点交换就会导致路径中出现重复的城市，在旅行商问题中这是不被允许的，所以要去除重复城市。

是因为交换才产生的重复，所以一个染色体中重复出现的城市就是与它交换的染色体所缺少的。修正思路就是找到这两个序列的重复的部分，然后将它们交换回去。

（5）交叉操作代码

def cross\_pop(p1, p2):

if np.random.random() > pc:

return p1, p2

else:

# S1:交换

# 随机生成两个交叉点

index1 = np.random.randint(0, city\_size - 3) # randint(a, b)函数返回的是[a, b)中的随机整数

index2 = np.random.randint(index1 + 2, city\_size) # 保证两个交叉点中至少有一个元素

# 将交叉点内的部分交换

p1[index1: index2], p2[index1: index2] = p2[index1: index2], p1[index1: index2]

# S2：修正（去重复）

# 找出重复部分

i, j = find\_repeat(p1, index1, index2), find\_repeat(p2, index1, index2)

# 不断交换，直到不再有重复

while i is not None and j is not None:

p1[i], p2[j] = p2[j], p1[i]

i, j = find\_repeat(p1, index1, index2), find\_repeat(p2, index1, index2)

return p1, p2

### 4.2.5变异

按概率pm进行异变，这里采用的是逆转异变，将基因中随机一段序列以逆向排序插回原序列。

def mut\_pop(path):

if np.random.random() > pm:

return path

else:

# 异变

index1 = np.random.randint(0, city\_size - 3)

index2 = np.random.randint(index1 + 2, city\_size - 1)

while (index2 - index1) >= 0:

path[index1], path[index2] = path[index2], path[index1]

index1 += 1

index2 -= 1

return path

### 4.2.6进化

根据迭代上限进行迭代进化，直到迭代次数到达上限或找到可接受的解时停止循环。

每次循环先计算种群的适应值数组、更新最佳路径，然后对种群依次进行选择、交叉、变异操作。

def evolution(pop):

global num

# 迭代num次

i = num

best\_distance = 100 # 刚开始时把best\_distance初始化为一个较大的数

while best\_distance > 22 and i > 0: # 找到可接受的解或迭代次数到达上限时停止

i -= 1

fitness = get\_fitness(pop)

# 计算当前种群中最佳适应的个体、对应路径、路径总长度

local\_best\_index = np.argmax(fitness)

local\_best\_path = copy.deepcopy(pop[local\_best\_index])

local\_best\_distance = get\_distance(local\_best\_path)

# 每次循环前，更新最佳路径及其总长度

if local\_best\_distance < best\_distance:

best\_path = copy.deepcopy(local\_best\_path)

best\_distance = local\_best\_distance

# 正式开始进化

# 选择

pop = select\_pop(pop, fitness)

# 交叉

# 每次从选择出的个体中选两个进行交配

pool = list(range(pop\_size))

for n in range(int(pop\_size / 2)):

parent1 = np.random.choice(pool)

pool.remove(parent1)

parent2 = np.random.choice(pool)

pool.remove(parent2)

child1, child2 = cross\_pop(list(pop[parent1]), list(pop[parent2]))

pop = np.append(pop, [child1], axis=0)

pop = np.append(pop, [child2], axis=0)

# 变异

for j in range(pop\_size):

pop[j] = np.array(mut\_pop(pop[j]))

print("结束！最后得到最佳路径为")

print(best\_path)

print("最佳路径长度为")

print(best\_distance)

return num - i, best\_distance

## 4.3用遗传算法解决旅行商问题

### 4.3.1定义参数和地图矩阵

city\_size = 10

pop\_size = 60

pop = np.array([]) # 种群数组

fitness = np.zeros(pop\_size) # 适应值数组，存放每个个体的适应值

pc = 0.85 # 交叉概率

pm = 0.1 # 变异概率

num = 300 # 最大迭代次数

# 地图数组

map = np.array([

[0, 4, 11, 7, 15, 9, 1, 8, 10, 5],

[4, 0, 12, 3, 6, 5, 10, 7, 4, 7],

[11, 12, 0, 14, 7, 2, 3, 1, 2, 11],

[7, 3, 14, 0, 6, 13, 8, 15, 7, 2],

[15, 6, 7, 6, 0, 6, 5, 2, 16, 1],

[9, 5, 2, 13, 6, 0, 2, 18, 10, 9],

[1, 10, 3, 8, 5, 2, 0, 14, 5, 3],

[8, 7, 1, 15, 2, 18, 14, 0, 2, 8],

[10, 4, 2, 7, 16, 10, 5, 2, 0, 14],

[5, 7, 11, 2, 1, 9, 3, 8, 14, 0]

], dtype='int').reshape([city\_size, city\_size])

### 4.3.2实验与分析

更改pc、pm、pop\_size等参数，进行多次实验，观察各个参数对算法执行效率、搜索准确性的影响。这里设置了5种不同的参数组合方式，对每种参数组合方式进行50次实验。

最后统计最大迭代次数为300时，不同参数组合方式**各自**对应的50次实验中，最好解的距离、最坏解的距离、最好解次数、迭代次数、迭代次数的中位数、搜索到解的距离平均值、搜索到的解的距离中位数等数学特征，进行比较。

parameters = [[0.85, 0.1, 60], [0.45, 0.1, 60], [0.85, 0.45, 60], [0.85, 0.1, 20], [0.85, 0.1, 150]]

for parameter in parameters:

pc = parameter[0]

pm = parameter[1]

pop\_size = parameter[2]

record = [] # 记录每次实验的迭代次数

best\_distances = [] # 记录每次实验得到的最短路径长度

best\_count = 0 # 记录50次实验中，最佳路径出现次数

# 对每种参数组合执行50次实验

for j in range(50):

pop = create\_pop(pop\_size)

# 进化

a, d = evolution(pop)

record.append(a)

best\_distances.append(d)

if d == 21:

best\_count += 1

print("参数为（顺序是pc、pm、pop\_size)：")

print(parameter)

print("最佳距离与迭代次数为：")

print(record)

print(best\_distances)

print("得到最优解的次数：%d / 50" % best\_count)

print("平均值")

print(np.mean(record))

print(np.mean(best\_distances))

print("中位数")

print(np.median(record))

print(np.median(best\_distances))

# 5实验结果

## 5.1单次实验

将pc设为0.85，pm设为0.1，pop\_size设为60，最大迭代次数设为300，用遗传算法求解旅行商问题，结果如下：

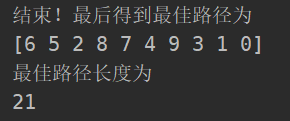


图1 单次实验得到的解路径和长度

## 5.2多次实验探究参数影响

对5种不同的参数组合方式，分别进行50次遗传算法实验，每次实验的最大迭代次数为300次。

（1）部分实验结果如下：

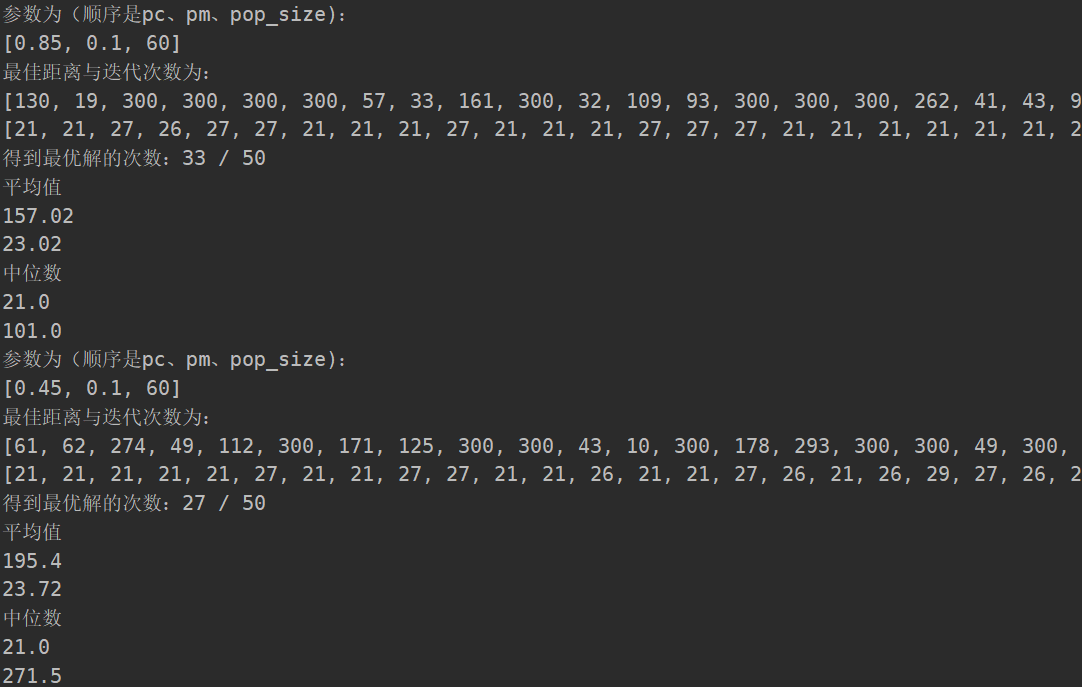


图2 调整参数的部分实验结果

（2）实验统计结果如下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 参数  (pc,pm,pop\_size) | 最好解 | 最坏解 | 最好解次数 | 平均迭代次数 | 平均解 | 迭代次数中位数 | 解中位数 |
| (0.85,0.1,60) | 21 | 27 | 36 | 141.76 | 22.64 | 92 | 21 |
| (0.45,0.1,60) | 21 | 28 | 30 | 174.62 | 23.26 | 151.5 | 21 |
| (0.85,0.45,60) | 21 | 27 | 37 | 154.34 | 22.4 | 136 | 21 |
| (0.85,0.1,20) | 21 | 30 | 21 | 222.56 | 24.86 | 300 | 27 |
| (0.85,0.1,150) | 21 | 26 | 49 | 58.04 | 21.1 | 44 | 21 |

表1 遗传算法5\*50次实验运行结果

鼓励同学们尝试设置更多的参数组合进行多次实验，深入探究如果设置参数可以使算法得到更优秀的性能。

同学们还可以进一步尝试改变遗传算法中的选择、交叉、变异等算子，使用收敛速度更快的遗传算子，观察对算法性能的影响。