群智能算法：粒子群算法解决“旅行商”问题

# 1实验目的

1. 了解群智能算法的产生背景；
2. 了解群智能算法与进化算法的异同；
3. 了解粒子群优化算法基本原理；
4. 了解粒子群优化算法各个参数的影响。

# 2实验要求

本次试验后，要求学生能：

1. 理解粒子群优化算法的基本原理和实现流程；
2. 理解各个参数对粒子群优化算法结果的影响；
3. 学会用粒子群优化算法解决复杂问题；
4. 编写python程序，用改版的粒子群优化算法解决“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群智能算法

在众多智能计算方法中，受动物群体智能启发的算法称为群智能（swarm intelligence，SI）算法。

如蚂蚁搬家、鸟群觅食、蜜蜂筑窝等等，不仅吸引生物学家去研究，也令计算机学家着迷，其中无一不蕴含着深刻的哲理。这些简单个体组成的群落与环境之间以及个体与个体之间的互动行为就被称为“群体智能”，从中演化出的算法就是群智能算法，主要包括粒子群算法、蚁群算法、人工免疫算法等。

群智能算法与进化算法有相似也有不同，它们都是受自然界启发得到的算法，且都是基于种群来搜索。但是进化算法强调达尔文的进化模型，群智能算法则侧重对群体中个体之间相互作用与分布式协同的模拟。

## 3.3粒子群优化算法

### 3.3.1中心思想

粒子群算法通过设计一种无质量的粒子来模拟鸟群中的鸟，粒子仅具有两个属性：速度和位置，速度代表移动的快慢，位置代表移动的方向。

每个粒子在搜索空间中单独的搜寻最优解，并将其记为当前个体极值，并将个体极值与整个粒子群里的其他粒子共享，找到最优的那个个体极值作为整个粒子群的当前全局最优解，粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。

### 3.3.2更新规则

PSO初始化为一群随机粒子(随机解)。然后通过迭代找到最优解。在每一次的迭代中，粒子通过跟踪两个“极值”(pbest，gbest)来更新自己。在找到这两个最优值后，粒子通过下面的公式来更新自己的速度和位置：

i = 1,2,3，……，m ； j = 1,2,3，……，n

公式1

xij表示第i个粒子位置向量的第j个元素；vij表示第i个粒子速度向量的第j个元素。

### 3.3.3参数分析

w为惯性权重因子，若w过小，则速度只取决于粒子当前位置和其历史最好位置gbest以及群最好位置pbest，粒子扩展搜索空间的速度会变慢。

r1为认知能力因子，若没有r1，则粒子没有“自我认知”能力，只会向群最佳位置和速度方向变化，收敛速度更快，但是容易陷入局部最优解。

r2为社会因子，若没有r2，则粒子间没有社会信息的共享，变成N个粒子单独互不干扰地搜索，会很难找到最优解

## 3.4针对“旅行商”问题改版的粒子群优化算法

基本粒子群优化算法在求解连续的函数优化问题中十分方便，对粒子速度求解与例子位置更新都很自然。

然而“旅行商”问题中，各个路径按实数编码后不连续，还不能直接进行加减，需要针对“旅行商”问题的特点，对算法进行改变。

针对旅行商问题的特点，可以用交换子序列法实现粒子群优化搜索。即，把粒子运动的速度转化成一组交换子序列，从而将粒子位置的更新策略改变为：

公式2

其中，v表示速度，是一组交换序列；pbest-xi表示当前粒子到它历史最佳位置的交换序列；gbest-xi表示当前粒子到群最佳位置的交换序列。

公式表示，新的速度为一个由旧速度、pbest-xi、gbest-xi组成的交换序列，其中旧速度中的交换序列以概率w交换，pbest-xi中的交换序列以概率r1交换，gbest-xi中的交换序列以概率r2交换。

例如，当前位置：[1,2,3]，粒子历史最佳位置：[2,1,3]，群最佳位置：[3,1,2]，旧速度：[(1,2)]，则新速度为[w:[(1,2)], r1:[(0,1)], r2:[(0,1),(0,2)]]。

粒子更新：按概率w交换1，2位置上的元素；按概率r1交换0，1位置上的元素；先按概率r2交换0，1位置，再按概率r2交换0，2位置。

# 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实现粒子群优化算法

### 4.2.1初始化粒子

（1）初始化各粒子位置

每个位置向量代表一条路径信息，例如，粒子在[1,2,3,4,5,6,7,8,9,0]这个位置上时，代表该粒子按照0,1,2,3,4,5,6,7,8,9,0的顺序遍历了10个城市（数字为城市的序号）。

先生成pop\_size个0~city\_size的顺序序列，然后分别将它们打乱，作为每个粒子的初始位置。

def init\_xs(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

xi = list(path)

xi.append(0)

pop.append(xi) # 加入种群

return pop

（2）初始化各粒子速度

用一个交换子序列代表速度（参见3.5小节）。例如，位置在[1,2,3,4,5,6,7,8,9, 0]上的粒子按速度[[0,1,0.1], [1,2,0.2]]变化后可能为[2,3,1,4,5,6,7,8,9, 0]（先将第0个元素和第1个元素以概率0.1交换，再将第1个元素和第2个元素以概率0.2交换）。

随机生成一个交换子，作为各个粒子的初始速度。

def init\_vs(pop\_size):

v = []

for i in range(pop\_size):

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

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

vi = [index1, index2, w] # 某个粒子对应的速度，w为交换的概率

v.append([vi])

return v

### 4.2.2更新种群最佳位置和每个粒子的历史最佳位置

（1）计算路径总距离和适应值函数

方法与前一章中相同。

def get\_distance(path):

distance = 0

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

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

return distance

# 分别计算群体中每个例子的适应度，返回适应度向量

def get\_fitness(pop):

f = []

for i in range(pop\_size):

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.append(fi)

return f

（2）更新种群最佳位置（gbest）、粒子历史最佳位置（pbest）

比较各个粒子当前位置和历史最佳位置的适应值，对每个粒子分别更新历史最佳位置。找出fitness最高的粒子，把它作为种群最佳位置。

def set\_best():

global pbest\_fitness, pbest\_list, gbest\_fitness, gbest, fitness, Xs

for k in range(pop\_size):

if pbest\_fitness[k] < fitness[k]:

pbest\_fitness[k] = fitness[k]

pbest\_list[k] = copy.deepcopy(Xs[k])

a = max(fitness)

if gbest\_fitness < a:

index = fitness.index(a)

gbest\_fitness = a

gbest = copy.deepcopy(Xs[index])

return get\_distance(gbest)

### 4.2.3交换操作

（1）交换算子，根据slist中的交换子序列，对path进行交换操作

def swap(path, slist):

for list in slist:

k = list[0]

j = list[1]

r = list[2]

r0 = np.random.random()

if r0 <= r:

path[k], path[j] = path[j], path[k]

return path

（2）求出当前状态到目标状态的交换子序列

遍历当前粒子，如果当前粒子位置向量的第k个元素不等于目标粒子位置向量的第k个元素，说明需要交换，找出当前粒子第k个元组在目标粒子中的序号，从而写出交换子，执行交换操作后继续查找，找到找出完整的交换子序列。

这里用到了numpy的np.where(List==a)函数，可以方便地找出list中为a的元素。

def get\_ss(goal, xi, r):

"""

param goal: pbest 或者 gbest

param xi: 粒子当前解

param r: 进行这种交换的概率

return:交换序列

"""

ss = []

xic = copy.deepcopy(xi) # 先将xi、goal复制，因为之后要进行交换，才能求出下一步的交换子。

goalc = copy.deepcopy(goal)

for k in range(len(xi)):

if xic[k] != goalc[k]: # 如果当前索引的粒子元素不等于目标粒子的对应元素

j = np.where(xic == goalc[k])[0][0] # 找到当前粒子和目标粒子不同的元素所在索引

so = [k, j, r] # 得到交换子,表示以r的概率对i,j进行操作

ss.append(so)

xic[k], xic[j] = xic[j], xic[k] # 执行交换操作

return ss # 返回交换子序列

### 4.2.4更新粒子的位置和速度

利用修改版的更新公式来更新：

其中，pbest-xi是当前粒子到该粒子历史最佳位置的交换子序列，gbest-xi是当前粒子到该种群最佳位置的交换子序列，w为惯性权重因子，r1、r2为两个加速度常数。

当前粒子x(i)的下一个状态x(i+1)就是用v(i+1)这个新计算出的交换子序列来对xi进行一系列的交换操作得到的结果。

def update(i):

global Xs, Vs

# 计算交换序列(更新速度)，即v(i+1) = wv（i）+ r1(pbest-xi) + r2(gbest-xi)

swap\_p = get\_ss(pbest\_list[i], Xs[i], r1)

swap\_g = get\_ss(gbest, Xs[i], r2)

swap\_list = Vs[i] + swap\_p + swap\_g

Xs[i] = swap(Xs[i], swap\_list) # 更新位置

for swapi in swap\_list:

swapi[2] = w # 保留V\_max个交换子，将交换概率改为w，作为下一次更新的初速度

Vs[i] = (swap\_list[-V\_max:]) # 更新速度

### 4.2.4种群优化

def optimize():

global num, fitness, best\_distance

# 迭代num次

i = num

while best\_distance > 22 and i > 0:

i -= 1

# 对每个粒子迭代，更新每个粒子的位置和速度

for j in range(pop\_size):

update(j) # 更新粒子

# 计算新的粒子群的适应度

fitness = get\_fitness(Xs)

# 获取pbest、gbest

best\_distance = set\_best()

a = num - i

return a, best\_distance

## 4.3实验与分析

更改w、r1、r2、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7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。

主函数如下：

if \_\_name\_\_ == '\_\_main\_\_':

city\_size = 10 # 城市大小

pop\_size = 60 # 种群大小

num = 300 # 迭代次数

fitness = [] # 适应值数组，存放每个个体的适应值

w = 0.3 # 加速度

r1 = 0.8 # 认知因子

r2 = 0.8 # 社会因子

pbest\_list = [] # 每一项是一个粒子的历史最佳位置

pbest\_fitness = [] # 每个粒子历史最佳位置对应的适应值

gbest = [] # 种群最佳位置

gbest\_fitness = 0 # 种群最佳位置对应的适应值

best\_distance = 0 # 最佳距离

Xs = [] # 保存粒子的位置信息

Vs = [] # 保存粒子当前交子序列

V\_max = 10

# 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])

parameters = [[0.4, 0.8, 0.8, 60], [0.8, 0.8, 0.8, 60], [0.4, 0.4, 0.4, 60], [0.4, 0.8, 0.4, 60],

[0.4, 0.4, 0.8, 60], [0.4, 0.8, 0.8, 20], [0.4, 0.8, 0.8, 150]]

for parameter in parameters:

w = parameter[0]

r1 = parameter[1]

r2 = parameter[2]

pop\_size = parameter[3]

record = []

best\_distances = []

best\_count = 0

# 对每种参数组合执行50次实验

for count in range(50):

Xs = init\_xs(pop\_size)

Vs = init\_vs(pop\_size)

fitness = get\_fitness(Xs)

pbest\_list = copy.deepcopy(Xs) # 复杂列表要用deepcopy！！！

pbest\_fitness = copy.deepcopy(fitness)

gbest = [] # 种群最佳位置

gbest\_fitness = 0 # 种群最佳位置对应的适应值

best\_distance = set\_best()

# 优化

a, d = optimize()

record.append(a)

best\_distances.append(d)

if d == 21:

best\_count += 1

print("参数为（顺序是w、r1、r2、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(best\_distances))

print(np.median(record))

# 5实验结果

## 5.1单次实验

将w设为0.4，r1设为0.8，r2设为0.8，pop\_size设为60，最大迭代次数设为300，用粒子群优化算法求解旅行商问题，结果如下：

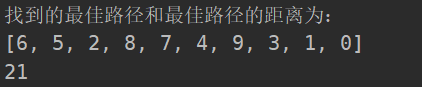


图1 单次实验得到的的解路径和长度

## 5.2多次实验探究参数影响

对7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。实验耗时较长，请耐心等待。

（1）部分实验结果如下：

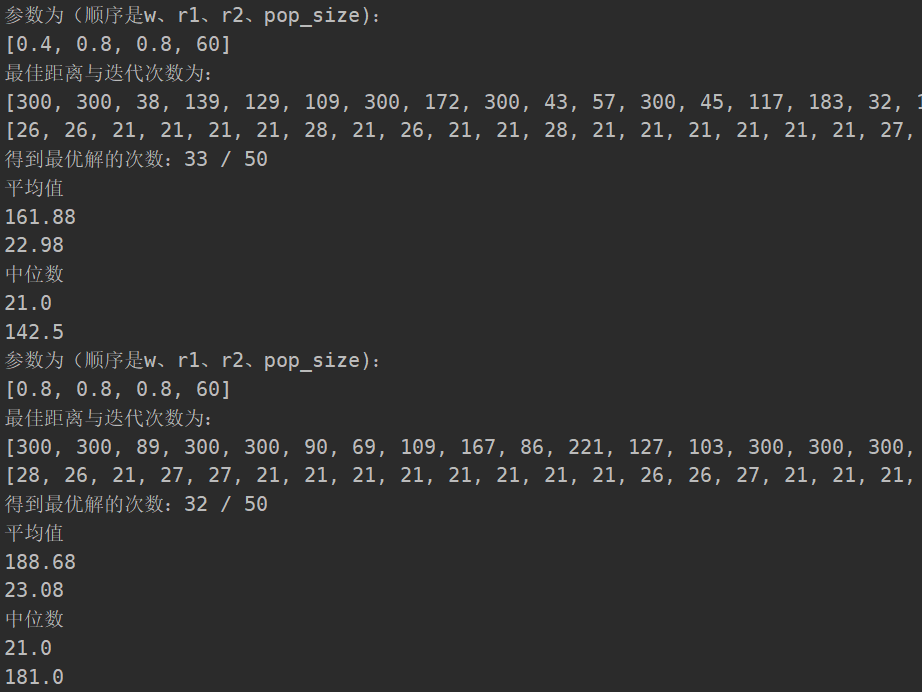


图2 调整参数的部分实验结果

（2）实验统计结果如下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 参数  (w,r1,r2,pop\_zise) | 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 |
| (0.4,0.8,0.8,60) | 21 | 28 | 33 | 161.88 | 22.98 | 142.5 | 21 |
| (0.8,0.8,0.8,60) | 21 | 28 | 32 | 188.68 | 23.08 | 181 | 21 |
| (0.4,0.4,0.4,60) | 21 | 30 | 31 | 166.64 | 23.32 | 150.5 | 21 |
| (0.4,0.8,0.4,60) | 21 | 30 | 32 | 155.9 | 23.26 | 119 | 21 |
| (0.4,0.4,0.8,60) | 21 | 30 | 26 | 196.82 | 24.06 | 243 | 21 |
| (0.4,0.8,0.8,20) | 21 | 32 | 18 | 227.36 | 25.26 | 300 | 26 |
| (0.4,0.8,0.8,150) | 21 | 27 | 44 | 96.96 | 21.88 | 21 | 55.5 |

表1 遗传算法7\*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。