from math import floor

import numpy as np

import matplotlib.pyplot as plt # 导入所需要的库

class Gena\_TSP(object):

def \_\_init\_\_(self ,data ,maxgen=200 ,size\_pop=200 ,cross\_prob=0.9 ,pmuta\_prob=0.01 ,select\_prob=0.8):

self.maxgen = maxgen # 最大迭代次数

self.size\_pop = size\_pop # 群体个数

self.cross\_prob = cross\_prob # 交叉概率

self.pmuta\_prob = pmuta\_prob # 变异概率

self.select\_prob = select\_prob # 选择概率

self.data = data # 城市的左边数据

self.num =len(data) # 城市个数 对应染色体长度

# 距离矩阵n\*n, 第[i,j]个元素表示城市i到j距离matrix\_dis函数见下文

self.matrix\_distance = self.matrix\_dis()

# 通过选择概率确定子代的选择个数

self.select\_num = max(floor(self.size\_pop \* self.select\_prob + 0.5), 2)

# 父代和子代群体的初始化（不直接用np.zeros是为了保证单个染色体的编码为整数，np.zeros对应的数据类型为浮点型）

self.chrom = np.array([0] \* self.size\_pop \* self.num).reshape(self.size\_pop, self.num)#父 print(chrom.shape)(200, 14)

self.sub\_sel = np.array([0] \* self.select\_num \* self.num).reshape(self.select\_num, self.num)#子 (160, 14)

# 存储群体中每个染色体的路径总长度，对应单个染色体的适应度就是其倒数 #print(fitness.shape)#(200,)

self.fitness = np.zeros(self.size\_pop)

self.best\_fit = []##最优距离

self.best\_path = []#最优路径

# 计算城市间的距离函数 n\*n, 第[i,j]个元素表示城市i到j距离

def matrix\_dis(self):

res = np.zeros((self.num, self.num))

for i in range(self.num):

for j in range(i + 1, self.num):

res[i, j] = np.linalg.norm(self.data[i, :] - self.data[j, :])#求二阶范数 就是距离公式

res[j, i] = res[i, j]

return res

# 随机产生初始化群体函数

def rand\_chrom(self):

rand\_ch = np.array(range(self.num)) ## num 城市个数 对应染色体长度 =14

for i in range(self.size\_pop):#size\_pop # 群体个数 200

np.random.shuffle(rand\_ch)#打乱城市染色体编码

self.chrom[i, :] = rand\_ch

self.fitness[i] = self.comp\_fit(rand\_ch)

# 计算单个染色体的路径距离值，可利用该函数更新fittness

def comp\_fit(self, one\_path):

res = 0

for i in range(self.num - 1):

res += self.matrix\_distance[one\_path[i], one\_path[i + 1]] #matrix\_distance n\*n, 第[i,j]个元素表示城市i到j距离

res += self.matrix\_distance[one\_path[-1], one\_path[0]]#最后一个城市 到起点距离

return res

# 路径可视化函数

def out\_path(self, one\_path):

res = str(one\_path[0] + 1) + '-->'

for i in range(1, self.num):

res += str(one\_path[i] + 1) + '-->'

res += str(one\_path[0] + 1) + '\n'

print(res)

# 子代选取，根据选中概率与对应的适应度函数，采用随机遍历选择方法

def select\_sub(self):

fit = 1. / (self.fitness) # 适应度函数

cumsum\_fit = np.cumsum(fit)#累积求和 a = np.array([1,2,3]) b = np.cumsum(a) b=1 3 6

pick = cumsum\_fit[-1] / self.select\_num \* (np.random.rand() + np.array(range(self.select\_num)))#select\_num 为子代选择个数 160

i, j = 0, 0

index = []

while i < self.size\_pop and j < self.select\_num:

if cumsum\_fit[i] >= pick[j]:

index.append(i)

j += 1

else:

i += 1

self.sub\_sel = self.chrom[index, :]#chrom 父

# 交叉，依概率对子代个体进行交叉操作

def cross\_sub(self):

if self.select\_num % 2 == 0:#select\_num160

num = range(0, self.select\_num, 2)

else:

num = range(0, self.select\_num - 1, 2)

for i in num:

if self.cross\_prob >= np.random.rand():

self.sub\_sel[i, :], self.sub\_sel[i + 1, :] = self.intercross(self.sub\_sel[i, :], self.sub\_sel[i + 1, :])

def intercross(self, ind\_a, ind\_b):#ind\_a，ind\_b 父代染色体 shape=(1,14) 14=14个城市

r1 = np.random.randint(self.num)#在num内随机生成一个整数 ，num=14.即随机生成一个小于14的数

r2 = np.random.randint(self.num)

while r2 == r1:#如果r1==r2

r2 = np.random.randint(self.num)#r2重新生成

left, right = min(r1, r2), max(r1, r2)#left 为r1,r2小值 ，r2为大值

ind\_a1 = ind\_a.copy()#父亲

ind\_b1 = ind\_b.copy()# 母亲

for i in range(left, right + 1):

ind\_a2 = ind\_a.copy()

ind\_b2 = ind\_b.copy()

ind\_a[i] = ind\_b1[i]#交叉 （即ind\_a （1,14） 中有个元素 和ind\_b互换

ind\_b[i] = ind\_a1[i]

x = np.argwhere(ind\_a == ind\_a[i])

y = np.argwhere(ind\_b == ind\_b[i])

"""

下面的代码意思是 假如 两个父辈的染色体编码为【1234】，【4321】

交叉后为【1334】，【4221】

交叉后的结果是不满足条件的，重复个数为2个

需要修改为【1234】【4321】（即修改会来

"""

if len(x) == 2:

ind\_a[x[x != i]] = ind\_a2[i]#查找ind\_a 中元素=- ind\_a[i] 的索引

if len(y) == 2:

ind\_b[y[y != i]] = ind\_b2[i]

return ind\_a, ind\_b

# 变异模块 在变异概率的控制下，对单个染色体随机交换两个点的位置。

def mutation\_sub(self):

for i in range(self.select\_num):#遍历每一个 选择的子代

if np.random.rand() <= self.cross\_prob:#如果随机数小于变异概率

r1 = np.random.randint(self.num)#随机生成小于num==14 的数

r2 = np.random.randint(self.num)

while r2 == r1:#如果相同

r2 = np.random.randint(self.num)#r2再生成一次

self.sub\_sel[i, [r1, r2]] = self.sub\_sel[i, [r2, r1]]#随机交换两个点的位置。

# 进化逆转 将选择的染色体随机选择两个位置r1:r2 ，将 r1:r2 的元素翻转为 r2:r1 ，如果翻转后的适应度更高，则替换原染色体，否则不变

def reverse\_sub(self):

for i in range(self.select\_num):#遍历每一个 选择的子代

r1 = np.random.randint(self.num)#随机生成小于num==14 的数

r2 = np.random.randint(self.num)

while r2 == r1:#如果相同

r2 = np.random.randint(self.num)#r2再生成一次

left, right = min(r1, r2), max(r1, r2)#left取r1 r2中小值，r2取大值

sel = self.sub\_sel[i, :].copy()#sel 为父辈染色体 shape=（1,14）

sel[left:right + 1] = self.sub\_sel[i, left:right + 1][::-1]#将染色体中(r1:r2)片段 翻转为（r2:r1)

if self.comp\_fit(sel) < self.comp\_fit(self.sub\_sel[i, :]):#如果翻转后的适应度小于原染色体，则不变

self.sub\_sel[i, :] = sel

# 子代插入父代，得到相同规模的新群体

def reins(self):

index = np.argsort(self.fitness)[::-1]#替换最差的（倒序）

self.chrom[index[:self.select\_num], :] = self.sub\_sel

#路径坐标

data = np.array([16.47,96.10,16.47,94.44,20.09,92.54,

22.39,93.37,25.23,97.24,22.00,96.05,20.47,97.02,

17.20,96.29,16.30,97.38,14.05,98.12,16.53,97.38,

21.52,95.59,19.41,97.13,20.09,92.55]).reshape(14,2)

def main(data):

Path\_short = Gena\_TSP(data) # 根据位置坐标，生成一个遗传算法类

Path\_short.rand\_chrom() # 初始化父类

## 绘制初始化的路径图

fig, ax = plt.subplots()

x = data[:, 0]

y = data[:, 1]

ax.scatter(x, y, linewidths=0.1)

for i, txt in enumerate(range(1, len(data) + 1)):

ax.annotate(txt, (x[i], y[i]))

res0 = Path\_short.chrom[0]

x0 = x[res0]

y0 = y[res0]

for i in range(len(data) - 1):

plt.quiver(x0[i], y0[i], x0[i + 1] - x0[i], y0[i + 1] - y0[i], color='r', width=0.005, angles='xy', scale=1,

scale\_units='xy')

plt.quiver(x0[-1], y0[-1], x0[0] - x0[-1], y0[0] - y0[-1], color='r', width=0.005, angles='xy', scale=1,

scale\_units='xy')

plt.show()

print('初始染色体的路程: ' + str(Path\_short.fitness[0]))

# 循环迭代遗传过程

for i in range(Path\_short.maxgen):

Path\_short.select\_sub() # 选择子代

Path\_short.cross\_sub() # 交叉

Path\_short.mutation\_sub() # 变异

Path\_short.reverse\_sub() # 进化逆转

Path\_short.reins() # 子代插入

# 重新计算新群体的距离值

for j in range(Path\_short.size\_pop):

Path\_short.fitness[j] = Path\_short.comp\_fit(Path\_short.chrom[j, :])

# 每隔三十步显示当前群体的最优路径

index = Path\_short.fitness.argmin()

if (i + 1) % 30 == 0:

print('第' + str(i + 1) + '步后的最短的路程: ' + str(Path\_short.fitness[index]))

print('第' + str(i + 1) + '步后的最优路径:')

Path\_short.out\_path(Path\_short.chrom[index, :]) # 显示每一步的最优路径

# 存储每一步的最优路径及距离

Path\_short.best\_fit.append(Path\_short.fitness[index])

Path\_short.best\_path.append(Path\_short.chrom[index, :])

res1 = Path\_short.chrom[0]

x0 = x[res1]

y0 = y[res1]

for i in range(len(data) - 1):

plt.quiver(x0[i], y0[i], x0[i + 1] - x0[i], y0[i + 1] - y0[i], color='r', width=0.005, angles='xy', scale=1,

scale\_units='xy')

plt.quiver(x0[-1], y0[-1], x0[0] - x0[-1], y0[0] - y0[-1], color='r', width=0.005, angles='xy', scale=1,

scale\_units='xy')

plt.show()

return Path\_short # 返回遗传算法结果类

Path\_short=main(data)

print(Path\_short)