# 实验七 遗传算法求TSP问题实验

## 实验目的

熟悉和掌握遗传算法的原理、流程和编码策略，理解求解TSP问题的流程并测试主要参数对结果的影响，掌握遗传算法的基本实现方法

## 二、实验内容

用遗传算法求解不用规模（如10个城市，20个城市，100个城市）的TSP问题。

## 三、实验要求

1用遗传算法求解不用规模（如10个城市，20个城市，100个城市）的TSP问题

2.对于用一个TSP问题，设置不同的种群规模、交叉概率和变异概率。

3设置种群规模为100，交叉概率为0.85，变异概率为0.15，然后增加一种变异策略（如相邻两点互换变异、逆转变异或插入变异）和1种个体选择概率分配策略用于求解同一TSP问题。

## 四、实验报告要求

1.画出遗传算法求解TSP问题的流程图。

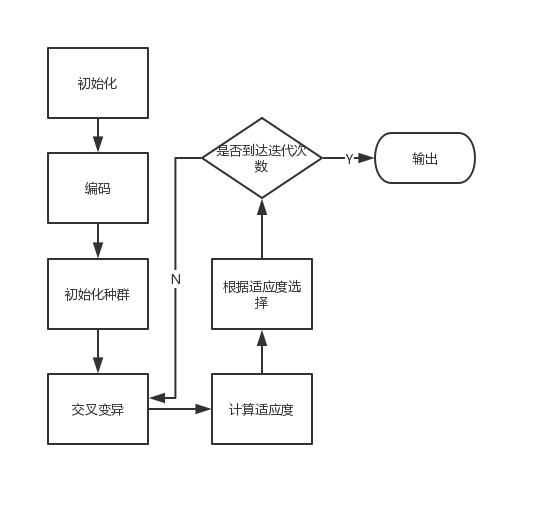


图 1 遗传算法

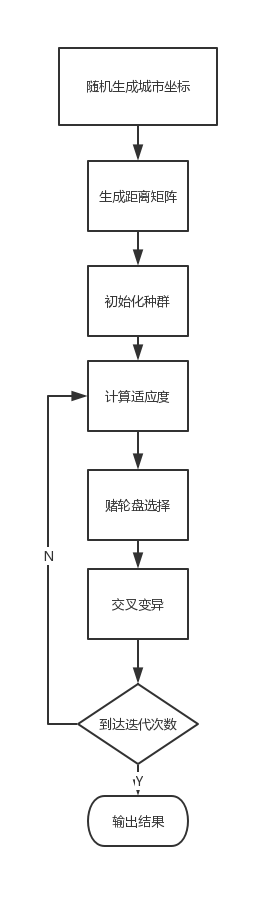


图 2 TSP流程图

1. 分析遗传算法求解不用规模的TSP问题的算法性能。

城市数量越大或者群体规模越大或者最大迭代数越大，运行等待结果的时间越。TSP规模越大，用时越长，但是当规模大到一定程度时，遗传算法的求解性能优于广度/宽度优先搜索算法，用时比一般的搜索算法少很多，对自身来说规模越大结果越精确。

1. 对于用一个TSP问题，分析种群规模、交叉概率和变异概率对算法结果的影响。

（1）种群规模对结果的影响：规模越大算法结果越精确，适应度越好，但是运行时间越久。

设置 城市数量10 变异概率0.15 交叉概率0.95 最大迭代数500

种群规模设置为50时：2组得到了最优解

（2）交叉概率对结果的影响：交叉概率过低将得不到最优解，交叉概率越高平均适应度越好。

设置 城市数量10 变异概率0.15 最大迭代数500 种群规模500

交叉概率设置为0.99时：得到17组最优解

（3）变异概率对结果的影响：变异概率过高或者过低都影响得到最优解

设置 城市数量10 种群规模500 交叉概率0.95 最大迭代数500

变异概率0.01时：12组得到了最优解

1. 增加1中变异策略和1中个体选择概率分配策略，比较求解同一TSP问题时不同变异策略及不同个体选择分配策略对算法结果的影响。

变异概率调整，对结果影响变化不大时，是受了种群数目和迭代次数的限制，总共发生的变异次数太少，但是设置迭代次数与规模增大时，会导致运行过于缓慢，等待结果时间太久。

1. 提交源程序。

import numpy as np

import random

import matplotlib.pyplot as plt

import copy

#各个城市的坐标

# City\_Map = [[106.54,29.59]

# ,[91.11,29.97]

# ,[87.68,43.77]

# ,[106.27,38.47]

# ,[111.65,40.82]

# ,[108.33,22.84]

# ,[126.63,45.75]

# ,[125.35,43.88]

# ,[123.38,41.8]

# ,[114.48,38.03]

# ,[112.53,37.87]

# ,[101.74,36.56]

# ,[117,36.65]

# ,[113.6,34.76]

# ,[118.78,32.04]

# ,[117.27,31.86]]

City\_Map = 100 \* np.random.rand(20, 2)#随机产生20个城市

DNA\_SIZE = len(City\_Map) #编码长度

POP\_SIZE = 100 #种群大小

CROSS\_RATE = 0.85 #交叉率

MUTA\_RATE = 0.15 #变异率

Iterations = 1000 #迭代次数

def distance(DNA):#根据DNA的路线计算距离

dis = 0

temp = City\_Map[DNA[0]]

for i in DNA[1:]:

dis = dis + ((City\_Map[i][0]-temp[0])\*\*2+(City\_Map[i][1]-temp[1])\*\*2)\*\*0.5

temp = City\_Map[i]

return dis+((temp[0]-City\_Map[DNA[0]][0])\*\*2+(temp[1]-City\_Map[DNA[0]][1])\*\*2)\*\*0.5

def getfitness(pop):#计算种群适应度，这里适应度用距离的倒数表示

temp = []

for i in range(len(pop)):

temp.append(1/(distance(pop[i])))

return temp-np.min(temp)

def select(pop, fitness): # 根据适应度选择，以赌轮盘的形式，适应度越大的个体被选中的概率越大

s = fitness.sum()

temp = np.random.choice(np.arange(len(pop)), size=POP\_SIZE, replace=True,p=(fitness/s))

p = []

for i in temp:

p.append(pop[i])

return p

def mutation(DNA, MUTA\_RATE):#进行变异

if np.random.rand() < MUTA\_RATE: #以MUTA\_RATE的概率进行变异

mutate\_point1 = np.random.randint(0, DNA\_SIZE)#随机产生一个实数，代表要变异基因的位置

mutate\_point2 = np.random.randint(0,DNA\_SIZE)#随机产生一个实数，代表要变异基因的位置

while(mutate\_point1 == mutate\_point2):#保证2个所选位置不相等

mutate\_point2 = np.random.randint(0,DNA\_SIZE)

DNA[mutate\_point1],DNA[mutate\_point2] = DNA[mutate\_point2],DNA[mutate\_point1] #2个所选位置进行互换

def crossmuta(pop, CROSS\_RATE):#交叉变异

new\_pop = []

for i in range(len(pop)):#遍历种群中的每一个个体，将该个体作为父代

n=np.random.rand()

if n>=CROSS\_RATE:#大于交叉概率时不发生变异，该子代直接进入下一代

temp = pop[i].copy()

new\_pop.append(temp)

if n<CROSS\_RATE:#小于交叉概率时发生变异

list1 = pop[i].copy()

list2 = pop[np.random.randint(POP\_SIZE)].copy()#选取种群中另一个个体进行交叉

status = True

while status:#产生2个不相等的节点，中间部分作为交叉段，采用部分匹配交叉

k1 = random.randint(0, len(list1) - 1)

k2 = random.randint(0, len(list2) - 1)

if k1 < k2:

status = False

k11 = k1

fragment1 = list1[k1: k2]

fragment2 = list2[k1: k2]

list1[k1: k2] = fragment2

list2[k1: k2] = fragment1

del list1[k1: k2]

left1 = list1

offspring1 = []

for pos in left1:

if pos in fragment2:

pos = fragment1[fragment2.index(pos)]

while pos in fragment2:

pos = fragment1[fragment2.index(pos)]

offspring1.append(pos)

continue

offspring1.append(pos)

for i in range(0, len(fragment2)):

offspring1.insert(k11, fragment2[i])

k11 += 1

temp = offspring1.copy()

mutation(temp,MUTA\_RATE)

new\_pop.append(temp)#把部分匹配交叉后形成的合法个体加入到下一代种群

return new\_pop

def print\_info(pop):#用于输出结果

fitness = getfitness(pop)

maxfitness = np.argmax(fitness)#得到种群中最大适应度个体的索引

#打印结果

print("最优的基因型：", pop[maxfitness])

print("最短距离：",distance(pop[maxfitness]))

#按最优结果顺序把地图上的点加入到best\_map列表中

best\_map = []

for i in pop[maxfitness]:

best\_map.append(City\_Map[i])

best\_map.append(City\_Map[pop[maxfitness][0]])

X = np.array((best\_map))[:,0]

Y = np.array((best\_map))[:,1]

#绘制地图以及路线

plt.figure()

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.scatter(X,Y)

for dot in range(len(X)-1):

plt.annotate(pop[maxfitness][dot],xy=(X[dot],Y[dot]),xytext = (X[dot],Y[dot]))

plt.annotate('start',xy=(X[0],Y[0]),xytext = (X[0]+1,Y[0]))

plt.plot(X,Y)

if \_\_name\_\_ == "\_\_main\_\_":#主循环

#生成初代种群pop

pop = []

list = list(range(DNA\_SIZE))

for i in range(POP\_SIZE):

random.shuffle(list)

l = list.copy()

pop.append(l)

best\_dis= []

#进行选择，交叉，变异，并把每代的最优个体保存在best\_dis中

for i in range(Iterations): # 迭代N代

pop = crossmuta(pop, CROSS\_RATE)

fitness = getfitness(pop)

maxfitness = np.argmax(fitness)

best\_dis.append(distance(pop[maxfitness]))

pop = select(pop, fitness) # 选择生成新的种群

print\_info(pop)#打印信息

print('逐代的最小距离：',best\_dis)

#画图

plt.figure()

plt.plot(range(Iterations),best\_dis)

plt.show()

plt.close()

6.总结实验心得体会。

这次遗传算法求解TSP问题的实验，让我体会了参数对实验结果的影响。本次实验针对种群数目，迭代次数，变异概率，交叉概率等参数进行修改设定与测试来获取最优解情况分析参数对结果的影响。在设置新函数新参数时，灵活应用程序设计结构，对遗传算法和python语言都有了进一步的理解。以后求解规模较大的TSP问题时，如果使用搜索与松弛策略用时太久，时间复杂度过高。可以考虑使用遗传算法求解，但要保证结果的精确，因为遗传算法在规模较小时可能得到的结果不是最优解。