**实验四 智能计算实验**

班级：软工2102 学号： 姓名：Mcrivers

**一、实验目的：**

熟悉和掌握遗传算法和粒子群优化算法的原理和流程，利用遗传算法或粒子群优化算法求解组合优化问题，理解求解TSP问题的流程并测试主要参数对结果的影响。

**二、实验原理：**

遗传算法（Genetic Algorithm，GA）的基本原理是通过作用于染色体上的基因寻找好的染色体来求解问题，它需要对算法所产生的每个染色体进行评价，并基于适应度值来选择染色体，使适应性好的染色体有更多的繁殖机会，在遗传算法中，通过随机方式产生若干个解的数字编码，即染色体，形成初始种群；通过适应度函数给每个个体一个数值评价，淘汰低适应度的个体，选择高适应度的个体参加遗传操作，经过遗传操作后的个体集合形成下一代新的种群，对这个新的种群进行下一轮的进化。

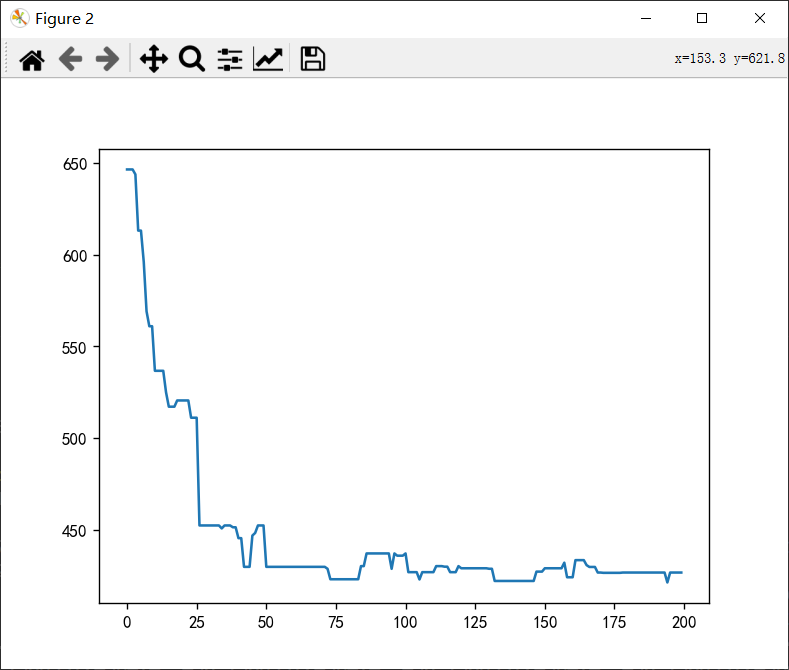
粒子群优化算法(Particle Swarm Optimization, PSO)源于对鸟群捕食的行为研究，其基本思想是通过群体中个体之间的协作和信息共享来寻找最优解。粒子群算法通过设计一种无质量的粒子来模拟鸟群中的鸟，粒子仅具有两个属性：速度和位置，速度代表移动的快慢，位置代表移动的方向。每个粒子在搜索空间中单独的搜寻最优解，并将其记为当前个体极值，并将个体极值与整个粒子群里的其他粒子共享，找到最优的那个个体极值作为整个粒子群的当前全局最优解，粒子群中的所有粒子根据自己找到的当前个体极值和整个粒子群共享的当前全局最优解来调整自己的速度和位置。

旅行商问题，即TSP问题（Traveling Salesman Problem）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，路经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为所有路径之中的最小值。TSP问题是一个组合优化问题。该问题可以被证明具有NPC计算复杂性。因此，任何能使该问题的求解得以简化的方法，都将受到高度的评价和关注。本实验要求利用遗传算法或粒子群优化算法求解TSP问题的最短路径。

**三、实验内容：**

1、参考遗传算法核心代码（无交叉操作），设置种群规模为80，最大迭代步数为200，交叉概率为0.8，变异概率为0.2，采用默认的变异策略（两点互换）求解20个城市的TSP问题（20个城市坐标见参考代码），把结果填入表1，并给出求得的最优路径图和算法收敛图。

1. 无部分匹配交叉操作时，所求得的最优路径图和算法收敛图：



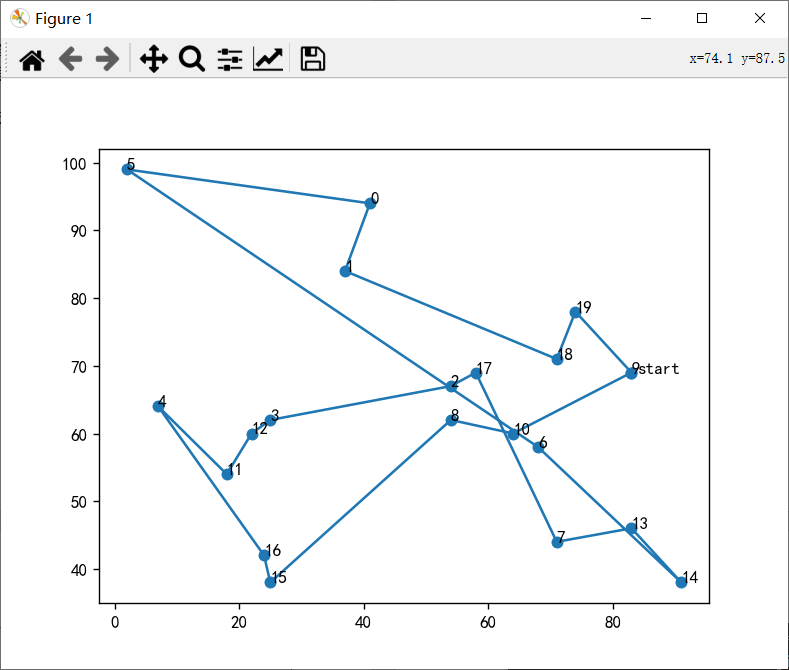


表1 TSP问题（20个城市）的遗传算法求解结果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 交叉操作 | 变异操作 | 最短距离 | 最优个体 | 运行时间 |
| 无 | 两点互换 | 410.8154758938201 | [13, 14, 7, 9, 19, 18, 8, 3, 12, 16, 15, 11, 4, 5, 1, 0, 2, 17, 10, 6] | 0.53125 |
| 部分匹配交叉 | 两点互换 | 377.40296575400527 | [6, 7, 18, 19, 9, 13, 14, 16, 15, 3, 11, 4, 12, 5, 0, 1, 2, 17, 8, 10] | 0.515625 |
| 部分匹配交叉 | 逆转变异 | 364.2499311405987 | [13, 14, 9, 6, 10, 17, 2, 19, 18, 1, 0, 5, 4, 12, 11, 15, 16, 3, 8, 7] | 0.53125 |

2、补充交叉操作的代码，使程序能够解决由于交叉引起的同一个染色体中出现相同基因的问题，即部分匹配交叉策略，算法参数设置和求解的问题都不变，把结果填入表1，并给出求得的最优路径图和算法收敛图。

1) 部分匹配交叉的代码：

            for i in range(len(list1)):

                if i < k1 or i >= k2:

                    while list1[i] in fragment2:

                        index = fragment2.index(list1[i])

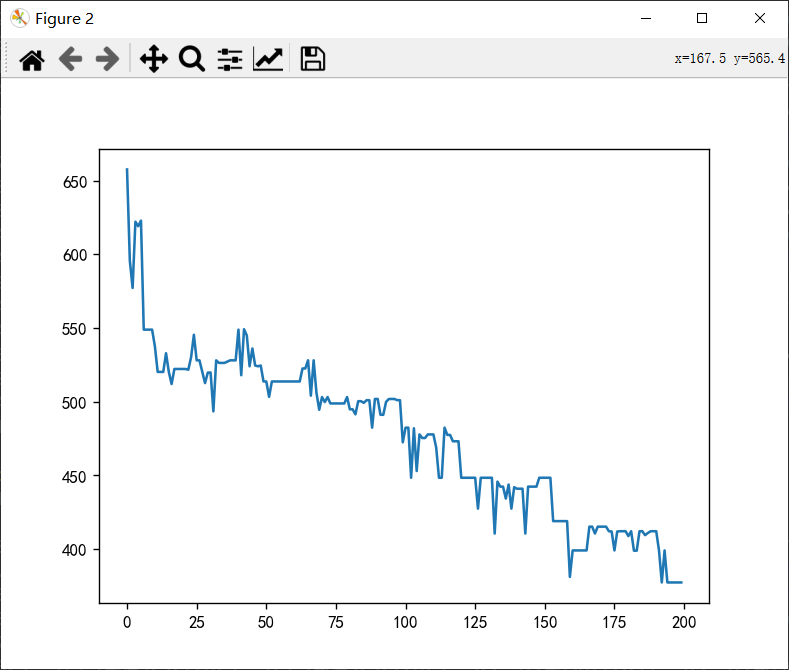
                        list1[i] = fragment1[index]

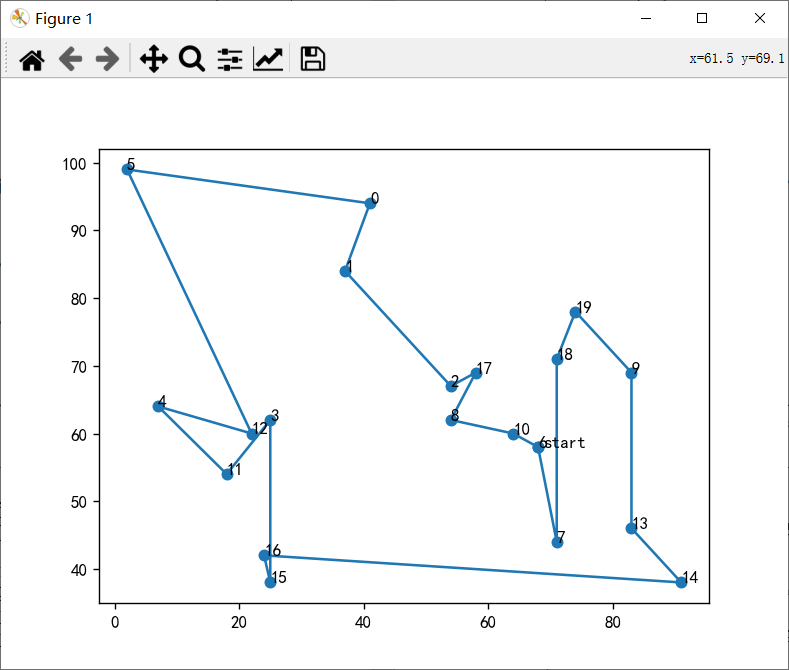
                    while list2[i] in fragment1:

                        index = fragment1.index(list2[i])

                        list2[i] = fragment2[index]

1. 有部分匹配交叉操作时，所求得的最优路径图和算法收敛图：





3、设计不同的变异操作（例如逆转变异或插入变异等），算法参数设置和求解的问题都不变，把结果填入表1，并给出求得的最优路径图和算法收敛图。

1. 所设计的变异操作的代码：

def reverse\_mutation(DNA, MUTA\_RATE):  # 进行变异：逆转变异

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

        mutate\_point1 = np.random.randint(0, DNA\_SIZE)

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

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

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

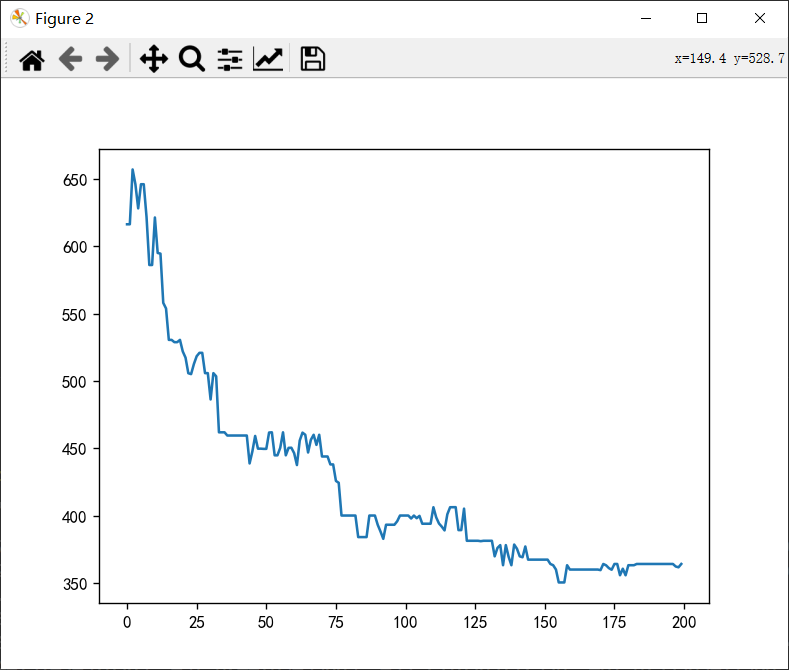
        if mutate\_point1 > mutate\_point2:

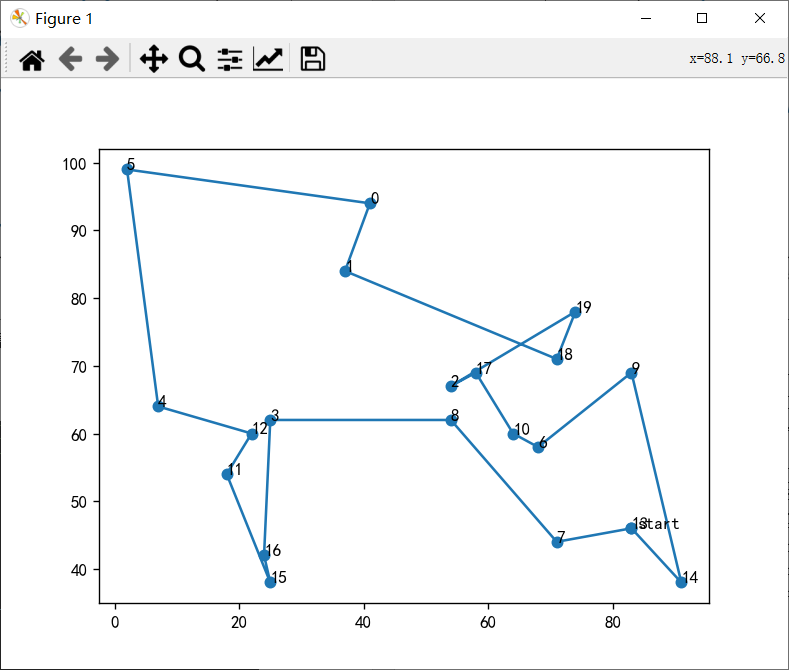
            mutate\_point1, mutate\_point2 = mutate\_point2, mutate\_point1

        # 逆转mutate\_point1和mutate\_point2之间的基因

        DNA[mutate\_point1:mutate\_point2] = DNA[mutate\_point1:mutate\_point2][::-1]

1. 有部分匹配交叉和所设计的变异操作时，求得的最优路径图和算法收敛图：



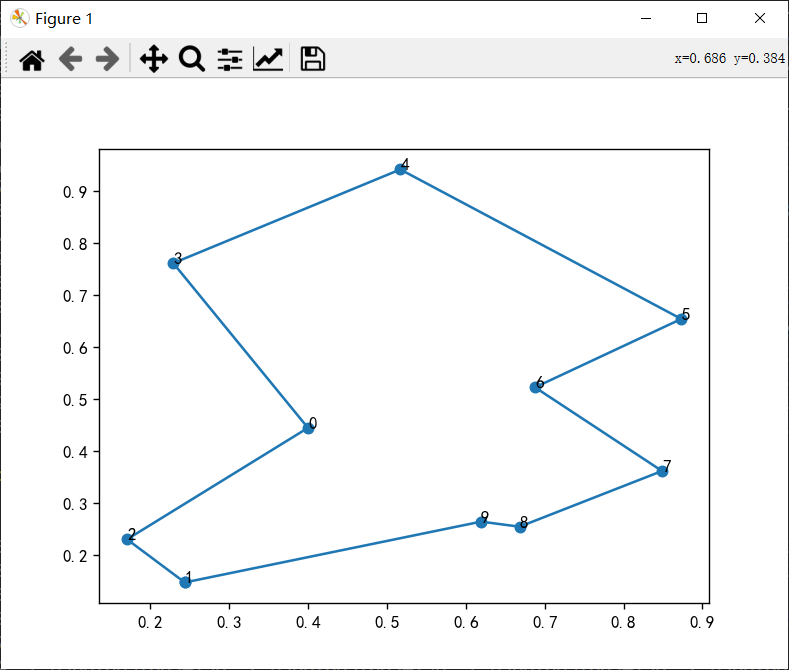


4、要求在相同的种群规模（80）、最大迭代步数（200）、独立运行次数(10)下，用遗传算法求解不同规模（例如10个城市，20个城市，100个城市）的TSP问题，每个城市的坐标可以为随机值，记录10次运行中种群的最好值（最好的最短距离）、最差值（最差的最短距离）和平均值（平均的最短距离），以及平均运行时间，把结果填入表2。同时给出不同城市规模的最优路径图。

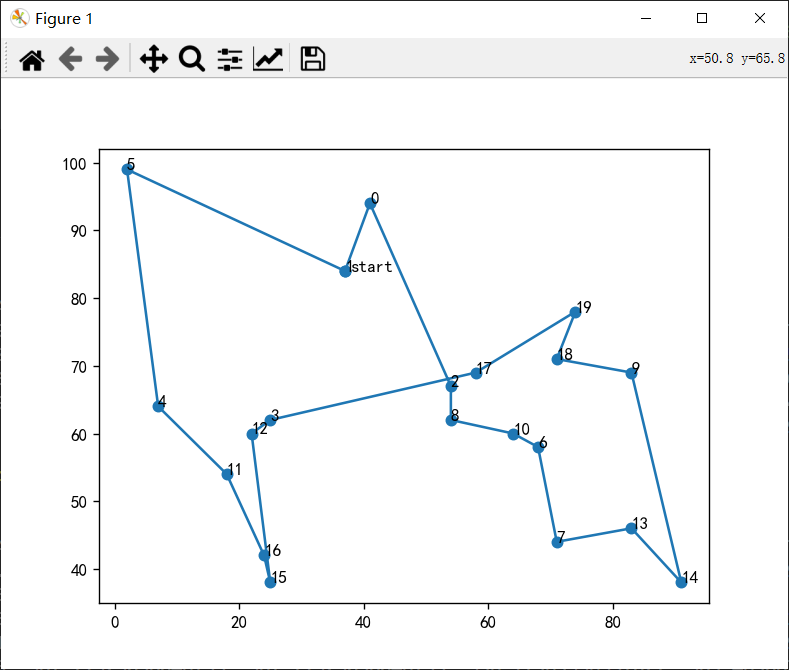
表2 遗传算法求解不同规模的TSP问题的结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 城市规模 | 种群规模 | 最大迭代步数 | 独立运行次数 | 最好值 | 最差值 | 平均值 | 平均运行时间 |
| 10 | 80 | 200 | 10 | 2.690670637009414 | 2.898195562065075 | 2.7944330995372445 | 0.50 |
| 20 | 80 | 200 | 10 | 333.74188345125185 | 405.0692210211662 | 377.541895236 | 0.515625 |
| 100 | 80 | 200 | 10 | 319.8565169477547 | 2163.795165435306 | 2963.15846954 | 2.85625 |

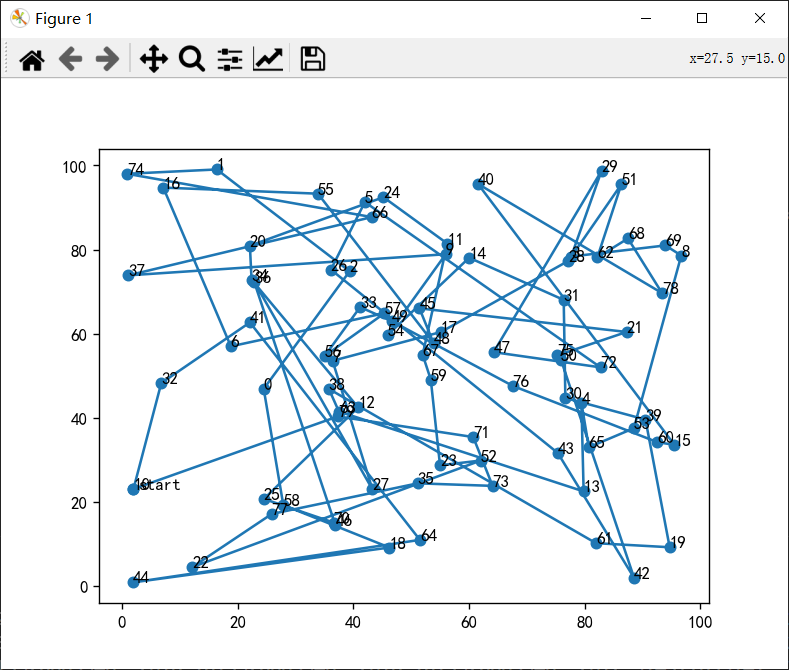
1. 城市规模为10的最优路径图：



1. 城市规模为20的最优路径图：



1. 城市规模为100的最优路径图：



5、现给出中国 34 个省会数据，要求基于此数据设计改进的遗传算法或粒子群优化算法(https://blog.csdn.net/tangshishe/article/details/115769686）解决该TSP问题。要求给出在相同的参数设置（种群规模、最大迭代步数等）下，求得的34个省会的最短路径最好值、最好解、最短路径平均值和算法运行时间，并把结果填入表4，同时给出算法的其它主要参数，例如遗传算法的种群规模、交叉概率变异概率、最大迭代步数等。

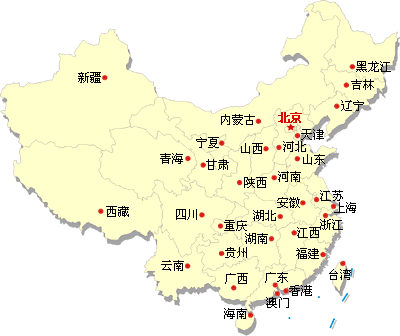


图 1 中国 34 省会位置(参考最短距离为1295.72)

表3 34个省会城市及像素坐标表

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **城市** | 西藏 | 云南 | 四川 | 青海 | 宁夏 | 甘肃 | 内蒙古 | 黑龙江 | 吉林 | 辽宁 | 北京 | 天津 |
| **城市号** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| X坐标 | 100 | 187 | 201 | 187 | 221 | 202 | 258 | 352 | 346 | 336 | 290 | 297 |
| Y坐标 | 211 | 265 | 214 | 158 | 142 | 165 | 121 | 66 | 85 | 106 | 127 | 135 |
| **城市** | 河北 | 山东 | 河南 | 山西 | 陕西 | 安徽 | 江苏 | 上海 | 浙江 | 江西 | 湖北 | 湖南 |
| **城市号** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** |
| X坐标 | 278 | 296 | 274 | 265 | 239 | 302 | 316 | 334 | 325 | 293 | 280 | 271 |
| Y坐标 | 147 | 158 | 177 | 148 | 182 | 203 | 199 | 206 | 215 | 233 | 216 | 238 |
| **城市** | 贵州 | 广西 | 广东 | 福建 | 海南 | 澳门 | 香港 | 台湾 | 重庆 | 新疆 |  |  |
| **城市号** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** |  |  |
| X坐标 | 221 | 233 | 275 | 322 | 250 | 277 | 286 | 342 | 220 | 104 |  |  |
| Y坐标 | 253 | 287 | 285 | 254 | 315 | 293 | 290 | 263 | 226 | 77 |  |  |

表4 不同算法求解34个省会城市的TSP问题的结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 算法及参数 | 种群规模 | 最大迭代步数 | 独立运行次数 | 其他参数 | 最短路径**平均值**（保留小数点后2位） | 最短路径**最好值**（保留小数点后2位） | 最短路径**最差值**（保留小数点后2位） | 最好解（最佳路径） |
| 遗传算法  参数设置： | 80 | 200 | 10 | 交叉率：0.8  变异率：0.2 | 1811.13 | 1735.56 | 2016.22 | [23, 24, 15, 13, 7, 8, 9, 12, 6, 10, 11, 3, 5, 4, 33, 0, 1, 2, 16, 32, 26, 31, 27, 30, 29, 25, 28, 17, 18, 19, 20, 14, 22, 21] |
| \*改进的遗传算法或粒子群优化算法  参数设置： | 80 | 200 | 10 | 惯性权重w = 0.5 | 2875.04 | 2695.23 | 3130.89 | [21 5 4 18 9 19 8 10 17 26 31 7 28 30 16 22 24 14 15 6 11 12 13 20  27 29 25 23 32 2 3 33 0 1] |

备注：\*为选做内容，即改进的遗传算法或粒子群优化算法选其一完成。

\*改进的遗传算法或粒子群算法求解TSP问题的代码：

import numpy as np

import random

import matplotlib.pyplot as plt

import copy

import time

# 城市坐标 City\_Map

City\_Map = [[100, 211], [187, 265], [201, 214], [187, 158], [221, 142], [202, 165], [258, 121], [352, 66], [346, 106],

            [336, 106], [290, 127], [297, 135],

            [278, 147], [296, 158], [274, 177], [265, 148], [239, 182], [302, 203], [316, 199], [334, 206], [325, 215],

            [293, 233], [280, 216],

            [271, 238], [221, 253], [233, 287], [275, 285], [322, 254], [250, 315], [277, 293], [286, 290], [342, 263],

            [220, 226], [104, 77]]

DNA\_SIZE = len(City\_Map)  # 城市数量

POP\_SIZE = 80  # 粒子数量

Iterations = 200  # 迭代次数

class Particle:

    def \_\_init\_\_(self):

        self.position = np.random.permutation(DNA\_SIZE)  # 粒子当前的位置，也就是当前的解

        self.velocity = np.random.permutation(DNA\_SIZE)  # 粒子的速度，表示粒子在解空间中的移动方向和距离

        self.best\_position = self.position.copy()  # 粒子到目前为止找到的最优位置，也就是最优解

        self.best\_fitness = float('inf')  # 粒子到目前为止找到的最优解的适应度值,初始设为无穷大

particles = [Particle() for \_ in range(POP\_SIZE)]

def distance(route):  # 计算路径长度

    dis = 0

    for i in range(len(route) - 1):

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

                (City\_Map[route[i]][1] - City\_Map[route[i + 1]][1]) \*\* 2) \*\* 0.5

    dis += ((City\_Map[route[-1]][0] - City\_Map[route[0]][0]) \*\* 2 +

            (City\_Map[route[-1]][1] - City\_Map[route[0]][1]) \*\* 2) \*\* 0.5

    return dis

def update\_particle\_velocity(particle, global\_best):

    w = 0.5  # 惯性权重

    c1 = 1.5  # 认知分量权重

    c2 = 1.5  # 社会分量权重

    for i in range(DNA\_SIZE):

        r1, r2 = random.random(), random.random()

        particle.velocity[i] = (w \* particle.velocity[i] +

                                c1 \* r1 \* (particle.best\_position[i] - particle.position[i]) +

                                c2 \* r2 \* (global\_best[i] - particle.position[i]))

def update\_particle\_position(particle):

    particle.position = np.argsort(particle.position + particle.velocity)

def run\_pso():

    global\_best = None

    global\_best\_fitness = float('inf')

    shortest\_paths = []

    for \_ in range(Iterations):

        for particle in particles:

            particle\_fitness = distance(particle.position)  # 计算粒子当前位置的适应度值

            if particle\_fitness < particle.best\_fitness:

                particle.best\_fitness = particle\_fitness

                particle.best\_position = particle.position.copy()

            if particle\_fitness < global\_best\_fitness:

                global\_best\_fitness = particle\_fitness

                global\_best = particle.position.copy()

        shortest\_paths.append(global\_best\_fitness)# 记录每次迭代的最短路径

        for particle in particles:

            update\_particle\_velocity(particle, global\_best)

            update\_particle\_position(particle)

    return global\_best, shortest\_paths

if \_\_name\_\_ == "\_\_main\_\_":

    best\_values = []

    worst\_values = []

    avg\_values = []

    run\_times = []

    best\_paths = []

    best\_pop = []

    for \_ in range(10):  # 独立运行10次 PSO 算法

        particles = [Particle() for \_ in range(POP\_SIZE)]  # 重新生成粒子群

        start\_time = time.time()

        best\_route, shortest\_paths = run\_pso()

        shortest\_paths\_np = np.array(shortest\_paths)

        best\_value = np.min(shortest\_paths\_np)

        worst\_value = np.max(shortest\_paths\_np)

        best\_solution = best\_route

        average\_shortest\_path = np.mean(shortest\_paths\_np)

        best\_values.append(best\_value)

        worst\_values.append(worst\_value)

        avg\_values.append(average\_shortest\_path)

        best\_paths.append(best\_route)

        best\_pop.append(shortest\_paths\_np[-1])  # 记录每次迭代的最佳种群

        end\_time = time.time()

        run\_times.append(end\_time - start\_time)

    # 输出10次运行的结果

    print('10次运行中种群的最好值：', min(best\_values))

    print('10次运行中种群的最差值：', min(worst\_values))

    print('10次运行中种群的平均值：', sum(avg\_values) / len(avg\_values))

    print('平均运行时间(单位为秒)：', sum(run\_times) / len(run\_times))

    # 找到最好的路径

    best\_index = best\_values.index(min(best\_values))

    best\_path = best\_paths[best\_index]

    print('最佳路径：', best\_path)

**四、实验报告要求：**

1、分析交叉操作对遗传算法求解20个城市的TSP问题结果的影响。

交叉操作可以在一定程度上优化求得解的质量，通过交叉操作可以更容易获得路径较短的解，而运行时间则没有明显变化。

2、比较求解20个城市的TSP问题时不同变异策略对算法结果的影响。

逆转变异策略得出的结果略优于两点互换策略得出的结果，但不明显

3、分析遗传算法求解不同规模的TSP问题的算法性能。

当问题规模上升时，遗传算法求解所花费的时间明显增长。

\*4、比较分析遗传算法与\*改进的遗传算法或粒子群算法求解34个城市的TSP问题的性能。

改进的粒子群算法得出的结果不如遗传算法，可能是超参数还需要优化所致

5、完成实验小结，并结合实际的优化问题谈谈遗传算法或粒子群算法的推广应用。

在实验中，我深入了解了遗传算法和粒子群优化算法，并在解决旅行商问题时体验了它们的效果。调整参数、观察算法行为和结果，让我意识到参数选择对算法性能的影响。这些算法不仅适用于TSP问题，还可在路线规划、资源分配等实际场景中发挥作用。它们是解决组合优化问题的有力工具，为解决实际挑战提供了新的思路和方法。

遗传算法和粒子群算法在现实中有广泛的应用。对于资源分配、路径规划、调度等问题，这些算法可以提供高效的解决方案。例如，在交通运输领域，它们可以优化物流配送路径，减少运输时间和成本。在生产调度中，它们能够优化生产流程，提高生产效率。此外，在人工智能、金融投资领域，它们也被用来解决优化问题，例如优化神经网络结构或者优化投资组合。这些算法的灵活性和适应性使它们成为处理各种优化问题的有力工具。