**任课教师：黄文清**

**《智能信息处理及应用》**

**（ 2020 -- 2021学年 第一学期）**

**实**

**验**

**报**

**告**

**学号：2018329621262**

**2018329621277**

**姓名：刘恒玮**

**朱若晨**

**班级：智能科学与技术（2）班**

**实验一 遗传算法解决TSP问题**

**一、实验目的**

1. 掌握遗传算法的基本原理和算法流程。
2. 理解根据特定问题的性质，构建合理的编码方案、适应度函数、遗传操作、控制参数的基本思想和步骤。
3. 理解TSP问题的基本概念，能根据TSP问题的特点，给出合理的编码方案、适应度函数定义方法、遗传操作方案和控制参数。
4. 复习python语言，能熟练使用python语言编写程序。

**二、实验环境与设备**

本次实验以小组形式进行。

实验环境：python语言。

实验设备：win10 Intel i7 2.21GHz

**三、实验原理**

1）遗传算法的基本原理分析；

遗传算法是一种基于自然选择和群体遗传机理的搜索算法,它模拟了自然选择和自然遗传过程中的繁殖、杂交和突变现象。再利用遗传算法求解问题时,问题的每一个可能解都被编码成一个“染色体”,即个体,若干个个体构成了群体(所有可能解)。在遗传算法开始时,总是随机的产生一些个体(即初始解),根据预定的目标函数对每一个个体进行评估,给出一个适应度值,基于此适应度值,选择一些个体用来产生下一代,选择操作体现了“适者生存”的原理,“好”的个体被用来产生下一代,“坏”的个体则被淘汰,然后选择出来的个体,经过交叉和变异算子进行再组合生成新的一代,这一代的个体由于继承了上一代的一些优良性状,因而在性能上要优于上一代,这样逐步朝着最优解的方向进化。因此,遗传算法可以看成是一个由可行解组成的群体初步进化的过程。

2）TSP问题的描述；

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，路径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路程为所有路径之中的最小值，用遗传算法，模拟染色体的遗传过程，进行求解。

**四、实验内容**

1）核心代码 不断杂交变异找出最优解

population\_cur = init\_population() # 计算当前种群的适应值

    fitness = get\_fitness(population\_cur)

    time\_start = time.time()# 记录进化开始时间

    # 终止条件

    while generation < max\_generation:

        # 父代最好的留1/4活下来

        population\_next = select\_sorted\_population(fitness, population\_cur, population\_size // 4)

        # 杂交

        for i in range(population\_size):

            p1, p2 = selection(fitness, 2) # 使用轮盘赌选择算子 随机选两个作为父母

            child1, child2 = crossover(population\_cur[p1], population\_cur[p2])

            # 对孩子进行变异

            if random.random() < p\_mutation:

                child1 = select\_best\_mutaion(child1, distmat)

            if random.random() < p\_mutation:

                child2 = select\_best\_mutaion(child2, distmat)

            population\_next.append(child1)

            population\_next.append(child2)

        # 选出下一代的种群

        population\_next = select\_sorted\_population(get\_fitness(population\_next), population\_next, population\_size)

        # 找出精英记录下来

        pre\_max\_fitness, pre\_max\_individual = get\_elite(fitness, population\_cur)

        record(pre\_max\_fitness)

        # 换代

        population\_cur = population\_next

        generation += 1

        # 更新fitness

        fitness = get\_fitness(population\_cur)

1. 选出最好的变异方法，对一个待变异的个体A，分别进行滑动变异，翻转变异，irgibnnm变异，取三种变异结果最好的那种。具体实现过程见代码及注释：

# 选择出最好的突变结果

def select\_best\_mutaion(s, distmat):

    s\_res = [slide\_mutation(s[:]), inversion\_mutation(s[:]), irgibnnm\_mutation(s[:], distmat)] #三种突变中选出最好的突变

    res = [get\_distances(s\_res[0], distmat), get\_distances(s\_res[1], distmat), get\_distances(s\_res[2], distmat)]

    min\_index = res.index(min(res))

    return s\_res[min\_index]

# 滑动变异

def slide\_mutation(s):

    a, b = get\_two\_randint(len(s))

    t = s[a]

    for i in range(a + 1, b + 1):

        s[i - 1] = s[i]

    s[b] = t

    return s

# 获得一个旅行路径的距离

def get\_distances(sequence, distmat):

    cost = 0

    for i in range(len(sequence)):

        cost += distmat[sequence[i - 1]][sequence[i]]

    return cost

# 倒置变异 随机生成两个下标a,b ; 将s[a]到s[b]之间的序列倒过来

def inversion\_mutation(s):

    # 自己手写的2变换

    a, b = get\_two\_randint(len(s))

    for i in range(a, (a + b) // 2 + 1):

        s[i], s[b + a - i] = s[b + a - i], s[i]

    return s

# 返回（小，大）两个随机数

def get\_two\_randint(size):

    b = a = random.randint(0, size - 1)

    while a == b:

        b = random.randint(0, size - 1)

    if a > b:

        return b, a

    return a, b

# irgibnnm变异 先对序列进行一次翻转变异，随机挑选一个城市，把他交换到地图距离与他最近的城市附近去

def irgibnnm\_mutation(s, distmat):

    a, b = get\_two\_randint(len(s))

    # 先倒置

    for i in range(a, (a + b) // 2 + 1):

        s[i], s[b + a - i] = s[b + a - i], s[i]

    # 再移动

    b = (b + 1) % len(s)

    min = b - 1

    for i in range(len(s)):

        if i == b:

            continue

        if distmat[b][min] > distmat[b][i]:

            min = i

    s[b], s[min - 4] = s[min - 4], s[b]

    return s

1. 画出距离和最大进化代数的关系和旅行的具体路径。

# 画出距离和最大进化代数的关系  和旅行的具体路径

def plot(sequnce):

    global record\_distance, coordinates

    plt.figure(figsize=(15, 6))

    plt.subplot(121)

    plt.plot(record\_distance)

    plt.ylabel('distance')

    plt.xlabel('iteration ')

    plt.subplot(122)

    x\_list = []

    y\_list = []

    for i in range(len(sequnce)):

        x\_list.append(coordinates[sequnce[i]][1])

        y\_list.append(coordinates[sequnce[i]][0])

    x\_list.append(coordinates[sequnce[0]][1])

    y\_list.append(coordinates[sequnce[0]][0])

    plt.plot(x\_list, y\_list, 'c-', label='Route')

    plt.plot(x\_list, y\_list, 'ro', label='Location')

    # 防止科学计数法

    ax = plt.gca()

    ax.xaxis.set\_major\_formatter(FormatStrFormatter('%.2f'))

    ax.yaxis.set\_major\_formatter(FormatStrFormatter('%.2f'))

    plt.xlabel("Longitude")

    plt.ylabel("Latitude")

    plt.title("Tsp Route")

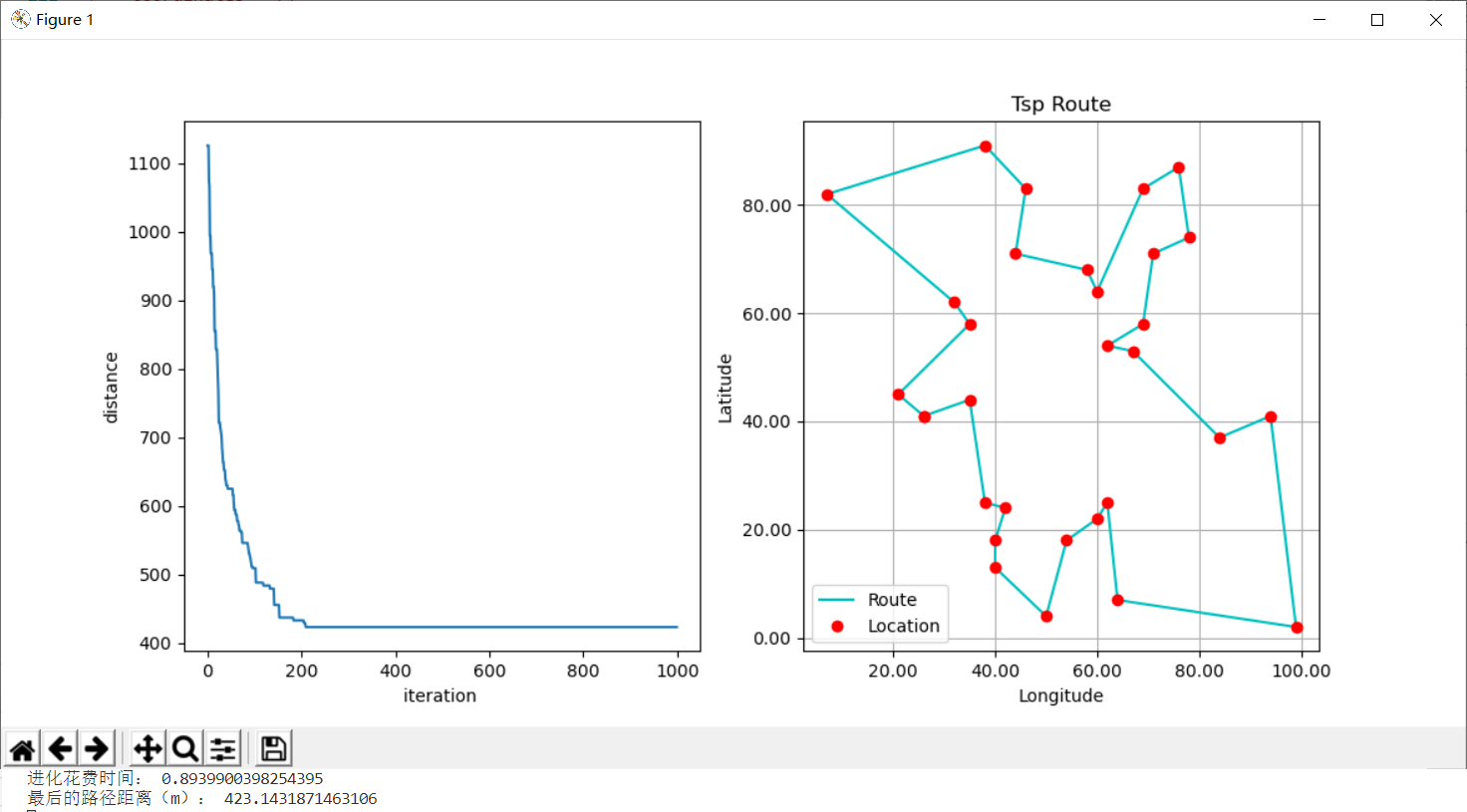
    plt.grid(True)

    plt.legend()

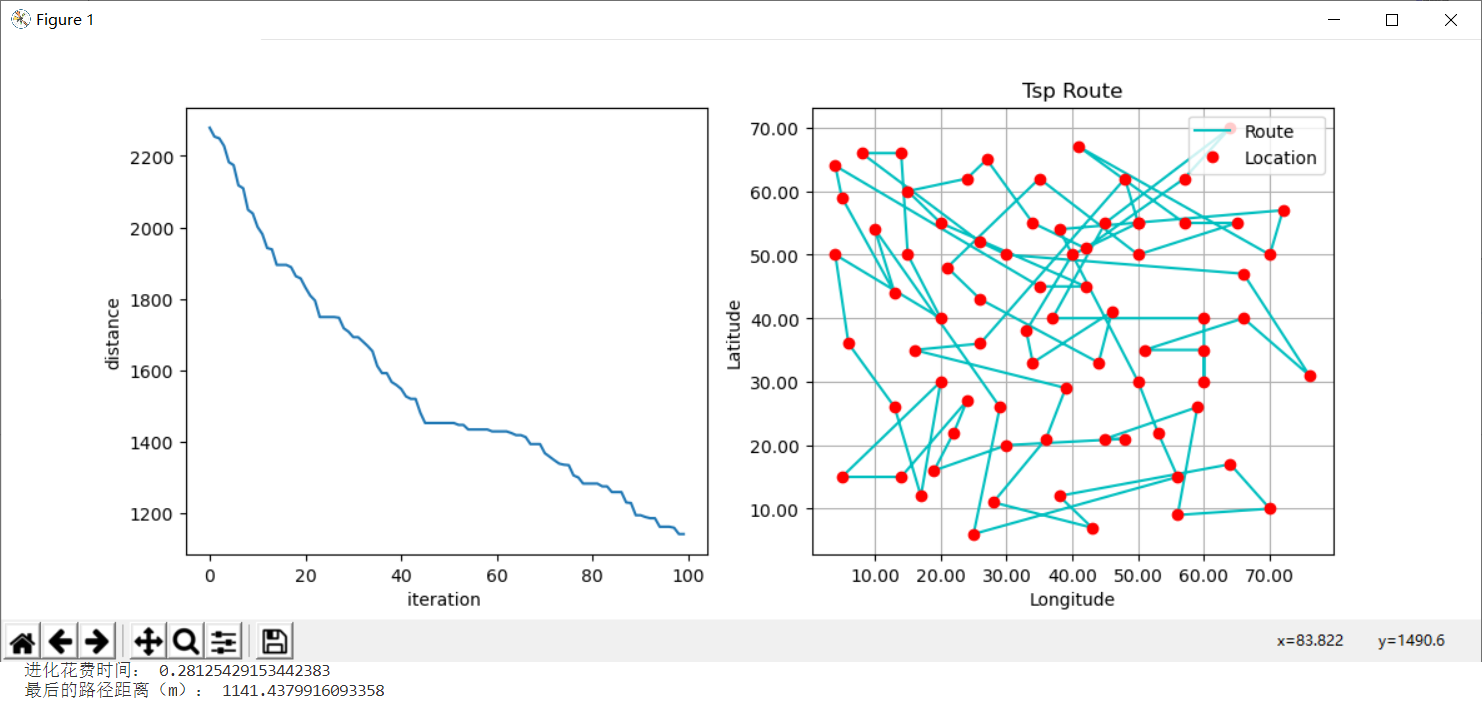
plt.show()

**五、实验结果**

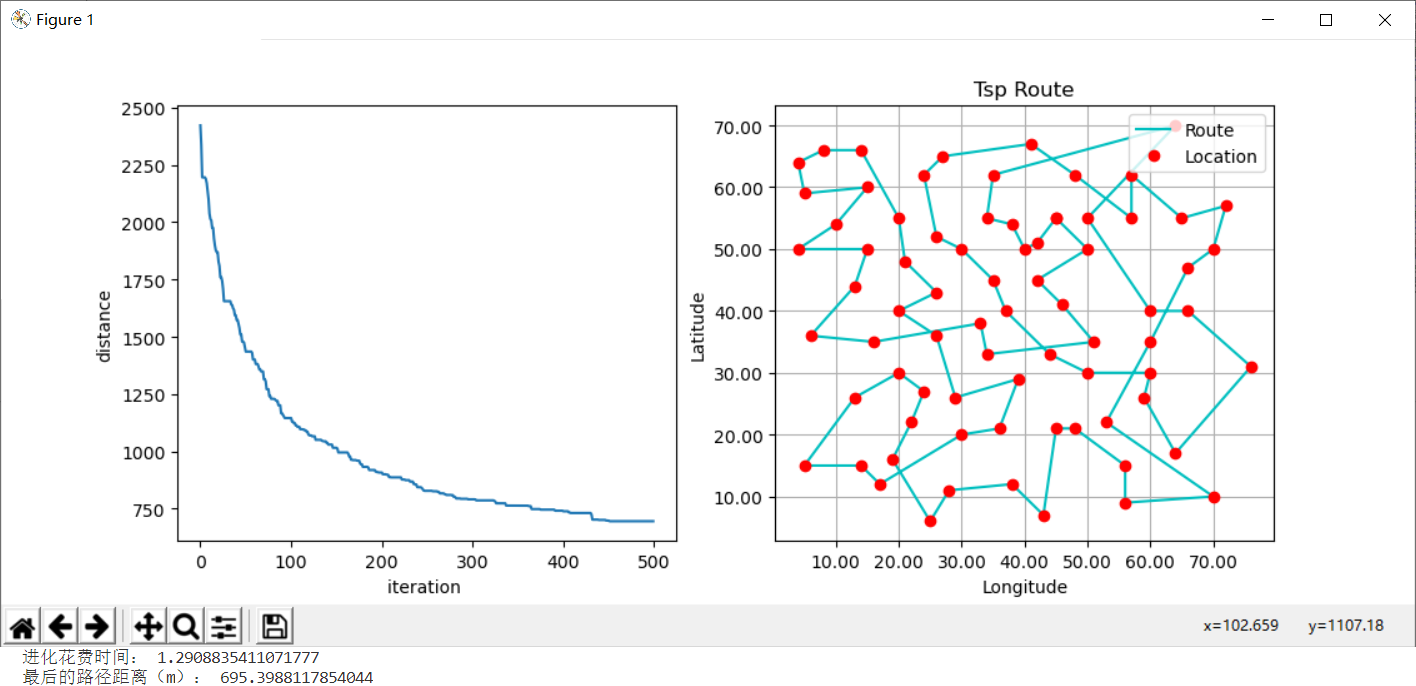
1）30个城市TSP问题得到的最优路径(最大进化代数大约在300以上就可以得到最优解)



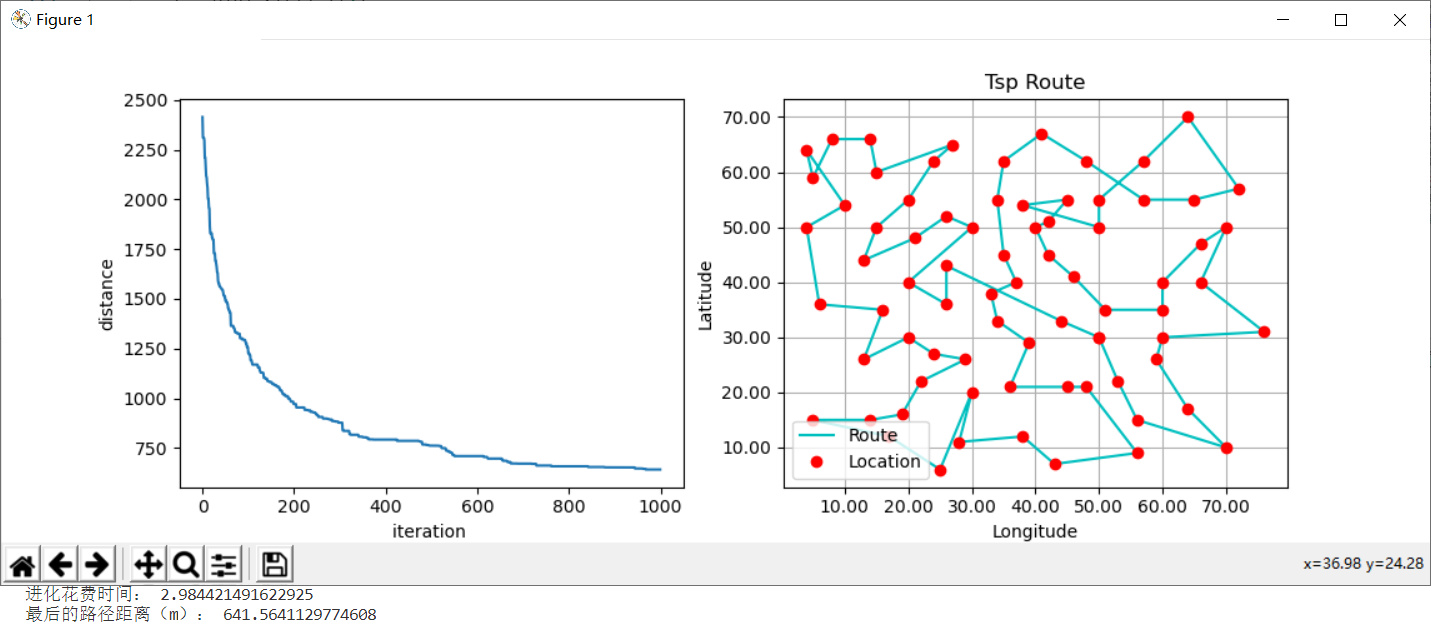
2）75个城市的TSP问题



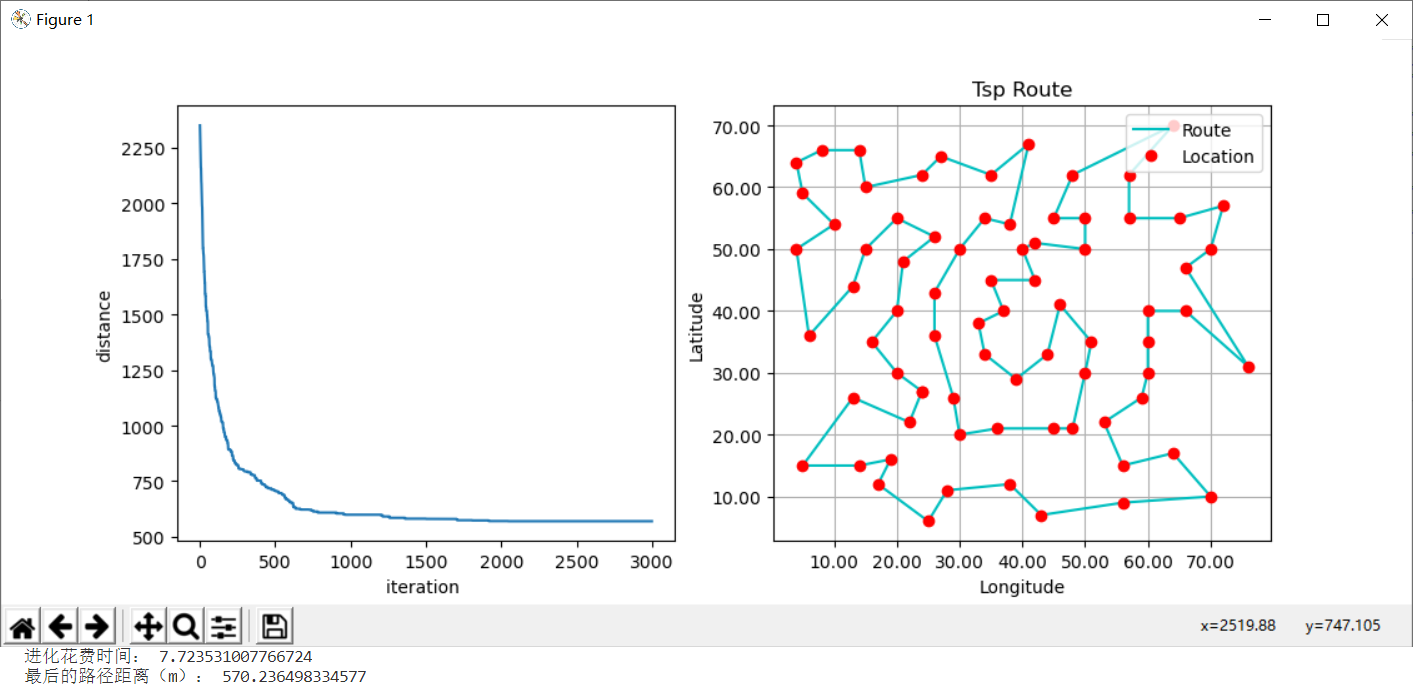
**最大进化代数等于100时，得到的路径杂乱无章**



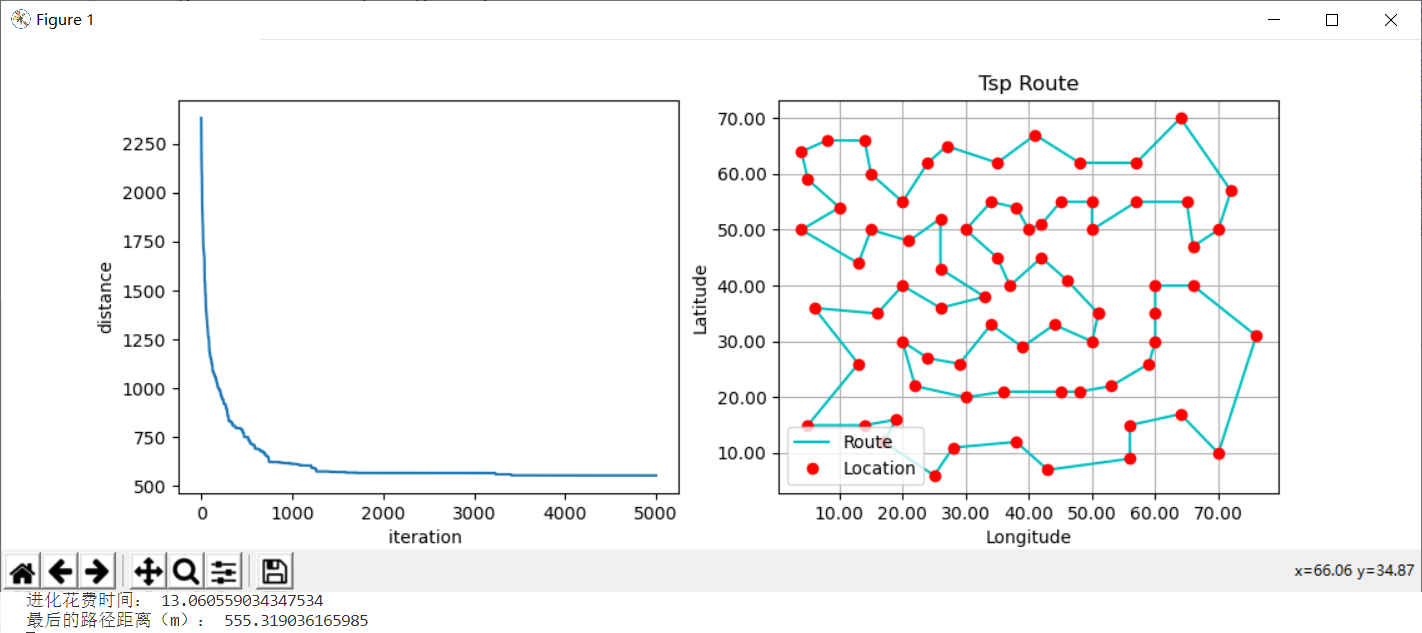
**最大进化代数等于500时，得到的路径仍有较多交叉**



**最大进化代数等于1000时，得到的路径较为准确，但仍有交叉**



**最大进化代数等于3000时，路径已不存在交叉**



**最大进化代数等于5000时，路径接近最优解**

**最后通过多次运行比较，我们小组所得到的最短的路径（最优解）：**



**实验讨论：**

影响实验结果的因素有：城市数量和最大进化代数以及选择的变异方法。城市越多，路线越复杂，越难以得到最优解；在一定限度内，最大进化代数越大，路径越准确，当最短路径接近准确值时，最大进化代数越大，最短路径在准确值附近波动，实验中所选择的三种变异方法各有优劣。

**六、实验总结**

- **问题1**：在75城市的数据时寻找的最短路径不能确定是否是最优解，并且即便最大进化代数取值很大也难以确保得到最优解。

解决方案：因为不知道准确的答案是多少，因此只能使最大进化代数尽可能大来提高精确度，在接近准确值之后，即便最大进化代数继续提高，最短路径会在准确值上下波动，因此只能是多次运行得到相对的最优解。

- **问题2**：在查阅相关文献和网上的代码之后了解到的irgibnnm变异不是非常理解。

解决方案：在与队友的不断讨论之后，对该变异方法是先对序列进行一次翻转变异，随机挑选一个城市，把他交换到地图距离与他最近的城市附近去，虽然在调试的过程中发现该方法所得到的效果并没有达到预期效果。

**参考资料：**

1. 《遗传算法解决TSP问题》实验报告指导书
2. 《智能信息处理技术原理与应用》
3. 《人工智能》（第3版）