# 实验报告 使用遗传算法解决旅行商问题（TSP）

## 一、实验目的

1. **熟悉和掌握遗传算法的基本概念和基本思想**：
   * 通过实际编程实践，加深对遗传算法（Genetic Algorithm, GA）的理解，包括其在解决优化问题中的应用。
2. **理解和掌握遗传算法的各个操作算子**：
   * 探究遗传算法中的选择（Selection）、交叉（Crossover）、变异（Mutation）等核心操作算子的实现方法。
3. **培养利用遗传算法进行问题求解的基本技能**：
   * 通过解决旅行商问题（Traveling Salesman Problem, TSP），培养使用遗传算法解决实际问题的能力。

## 二、实验预习内容

1. **旅行商问题的表示**：
   * TSP是寻找最短可能路径，访问一系列城市并返回起点的问题。
2. **遗传算法的基本概念和基本算子**：
   * 理解遗传算法中的基本概念，如染色体、适应度、选择、交叉、变异等。

## 三、实验内容

**以N个节点的TSP问题为例，应用遗传算法进行求解，求出问题的最优解**。

### 1. 旅行商问题（TSP）

* TSP是一个组合优化问题，其中目标是找到访问所有城市并返回起点的最短路径。
* 给定的实验代码解决了一个含有30个节点的TSP问题。
* 也可取前10个城市的坐标（如表2）进行测试：
* 表2 Oliver TSP问题的10个城市位置坐标

| 城市编号 | 坐标 |
| --- | --- |
| 1 | (87，7) |
| 2 | (91，38) |
| 3 | (83，46) |
| 4 | (71，44) |
| 5 | (64，60) |
| 6 | (68，58) |
| 7 | (83，69) |
| 8 | (87，76) |
| 9 | (74，78) |
| 10 | (71，71) |

* 有人求得的最优路径为： 0 3 5 4 9 8 7 6 2 1 0
* 路径长度是166.541336
* 下面给出30个城市的位置信息：
* 表1 Oliver TSP问题的30个城市位置坐标

| 城市编号 | 坐标 | 城市编号 | 坐标 | 城市编号 | 坐标 |
| --- | --- | --- | --- | --- | --- |
| 1 | (87，7) | 11 | (58，69) | 21 | (4，50) |
| 2 | (91，38) | 12 | (54，62) | 22 | (13，40) |
| 3 | (83，46) | 13 | (51，67) | 23 | (18，40) |
| 4 | (71，44) | 14 | (37，84) | 24 | (24，42) |
| 5 | (64，60) | 15 | (41，94) | 25 | (25，38) |
| 6 | (68，58) | 16 | (2，99) | 26 | (41，26) |
| 7 | (83，69) | 17 | (7，64) | 27 | (45，21) |
| 8 | (87，76) | 18 | (22，60) | 28 | (44，35) |
| 9 | (74，78) | 19 | (25，62) | 29 | (58，35) |
| 10 | (71，71) | 20 | (18，54) | 30 | (62，32) |

* 最优路径为：1 2 3 4 6 5 7 8 9 10 11 12 13 14 15 16 17 19 18 20 21 22 23 24 25 28 26 27 29 30
* 其路径长度为：424.869292

### 2. 问题描述

* 应用遗传算法求解30/10个节点的TSP（旅行商问题）问题，求问题的最优解。

## 四、实验要求

1. **掌握遗传算法的基本原理、各个遗传操作和算法步骤**：
   * 通过实验，深入理解遗传算法的工作原理和核心操作。
2. **求出问题最优解，若得不出最优解，请分析原因**：
   * 在实验中记录求解过程，分析未能得出最优解的可能原因。
3. **对实验中的几个算法控制参数进行仔细定义**：
   * 定义种群大小、变异率、交叉率等参数，并通过实验找到最佳值。
4. **界面显示每次迭代求出的局部最优解和最终求出的全局最优解**：
   * 在程序运行过程中，记录并显示每次迭代的最优解和最终的全局最优解。

## 五、实验结果分析

### 实验环境和参数设置

* **实验环境**：Python 3.10, Numpy, Matplotlib
* **参数设置**：
  + 种群大小：100
  + 精英大小：20
  + 变异率：0.01
  + 迭代代数：500-1200

### 实验过程

1. **初始化**：根据给定的城市坐标，初始化一个种群，每个个体是一条随机生成的路径。

### 1. 种群 (Population)

* 在遗传算法中，种群是一个包含多个个体的集合，每个个体代表了问题的一个可能解。
* def initial\_population(self):  
   return [self.create\_route(self.population) for \_ in range(self.pop\_size)]
* 这里的 initial\_population 方法生成初始种群，每个个体是通过 create\_route 方法随机生成的路径。在TSP问题中，每个个体（路径）是城市的一个特定排列。

1. **适应度计算**：为每条路径计算适应度，即路径长度的倒数。

### 2. 适应度 (Fitness)

* 适应度是衡量个体优劣的标准，通常是问题特定的。在TSP中，适应度是路径长度的倒数。
* # Fitness 子类：用于评估路径适应度  
   class Fitness:  
   def \_\_init\_\_(self, route):  
   self.route = route # 路径  
   self.distance = 0 # 路径长度  
   self.fitness = 0.0 # 路径的适应度值  
    
   # 计算路径的总距离  
   def route\_distance(self):  
   if self.distance == 0:  
   path\_distance = 0  
   for i in range(len(self.route)):  
   from\_city = self.route[i]  
   to\_city = self.route[i + 1] if i + 1 < len(self.route) else self.route[0]  
   path\_distance += from\_city.distance(to\_city)  
   self.distance = path\_distance  
   return self.distance  
    
   # 计算路径的适应度  
   def route\_fitness(self):  
   if self.fitness == 0:  
   self.fitness = 1 / float(self.route\_distance()) # 适应度是路径长度的倒数  
   return self.fitness
* 在 Fitness 类中，route\_fitness 方法计算了一条路径的适应度。路径越短，其适应度值越高。

1. **遗传操作**：应用选择、交叉和变异算子生成新的种群。

### 3. 选择 (Selection)

* 选择是遗传算法中用于决定哪些个体将被保留下来繁殖后代的过程。
* # 选择函数  
  def selection(self, pop\_ranked):  
   selection\_results = []  
   df = pd.DataFrame(np.array(pop\_ranked), columns=["Index", "Fitness"])  
   df['cum\_sum'] = df.Fitness.cumsum()  
   df['cum\_perc'] = 100 \* df.cum\_sum / df.Fitness.sum()  
    
   for i in range(self.elite\_size):  
   selection\_results.append(pop\_ranked[i][0])  
   for \_ in range(len(pop\_ranked) - self.elite\_size):  
   pick = 100 \* random.random()  
   for i in range(len(pop\_ranked)):  
   if pick <= df.iat[i, 3]:  
   selection\_results.append(pop\_ranked[i][0])  
   break  
   return selection\_results
* 在 selection 方法中，先是根据适应度保留一部分精英个体，然后使用轮盘赌选择算法选取其他个体。
  + df = pd.DataFrame(...): 创建一个 DataFrame，用于存储每个路径的索引和适应度，以便进行后续的选择操作。
  + df['cum\_sum'] 和 df['cum\_perc']: 这两行计算适应度的累积和及其累积百分比，这是轮盘赌选择方法的一部分。
  + 前两个 for 循环是选择操作的核心。首先选择适应度最高的个体，然后通过轮盘赌算法选择剩余的个体。

### 4. 交叉 (Crossover)

* 交叉是遗传算法中的一种遗传操作，用于结合两个父代个体的信息，生成新的后代个体。
* def breed(parent1, parent2):  
   geneA, geneB = random.sample(range(len(parent1)), 2)  
   start\_gene, end\_gene = sorted([geneA, geneB])  
   childP1 = parent1[start\_gene:end\_gene]  
   childP2 = [item for item in parent2 if item not in childP1]  
   return childP1 + childP2
* 在 breed 方法中，通过从两个父代路径中选择一段并结合起来，形成新的子代路径。
  + geneA, geneB = random.sample(...): 从父代路径中随机选择两个点作为交叉的起止点。
  + childP1 = ...: 从一个父代中选择一个子序列。
  + childP2 = ...: 从另一个父代中选择不在 childP1 中的其余部分。
  + return childP1 + childP2: 将两部分组合成新的子代路径。

### 5. 变异 (Mutation)

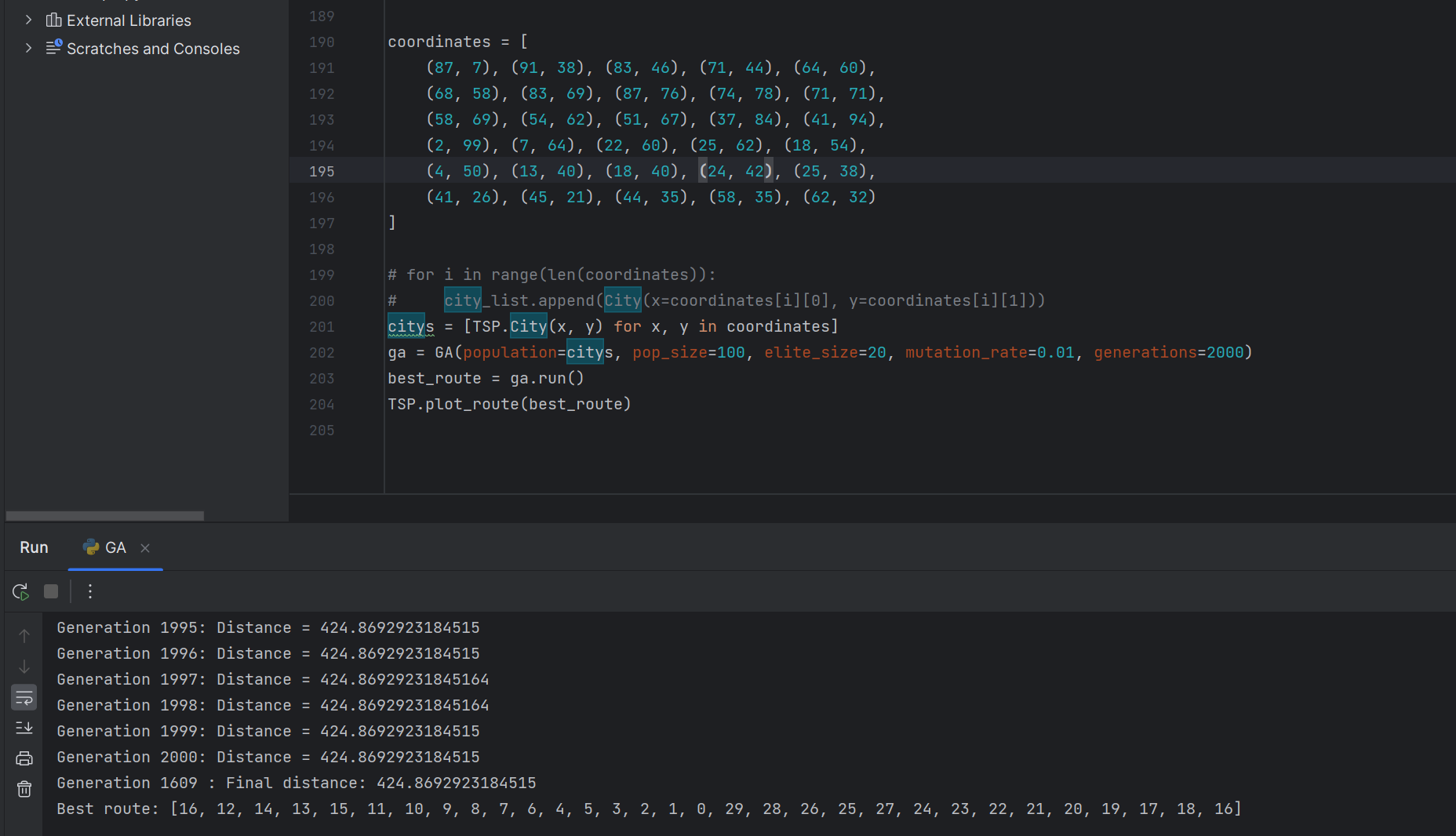
* 变异是引入新特征的过程，有助于维持种群的多样性，防止算法过早收敛于局部最优解。
* def mutate(individual, mutation\_rate):  
   for swapped in range(len(individual)):  
   if random.random() < mutation\_rate:  
   swap\_with = int(random.random() \* len(individual))  
   individual[swapped], individual[swap\_with] = individual[swap\_with], individual[swapped]  
   return individual
* 在 mutate 方法中，通过随机交换路径中的两个城市来引入变异。
  + for swapped in range(len(individual)): 遍历路径中的每个城市。
  + if random.random() < mutation\_rate: 以一定概率执行变异操作。
  + individual[swapped], individual[swap\_with] = ...: 交换两个城市的位置，引入新的特性。

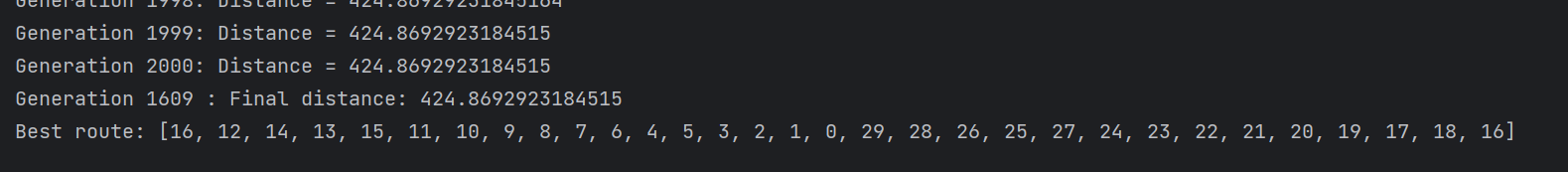
1. **迭代优化**：重复执行上述步骤，直到达到设定的迭代次数。

* def next\_generation(self, current\_gen):  
   pop\_ranked = self.rank\_routes(current\_gen)  
   selection\_results = self.selection(pop\_ranked)  
   matingpool = self.mating\_pool(current\_gen, selection\_results)  
   children = self.breed\_population(matingpool)  
   return self.mutate\_population(children)
  + pop\_ranked = self.rank\_routes(current\_gen): 对当前种群进行排名。
  + selection\_results = self.selection(pop\_ranked): 选择最优个体以及其他个体。
  + matingpool = ...: 创建配对池。
  + children = self.breed\_population(matingpool): 应用交叉操作产生新一代。
  + return self.mutate\_population(children): 对新一代应用变异操作。

### 实验结果

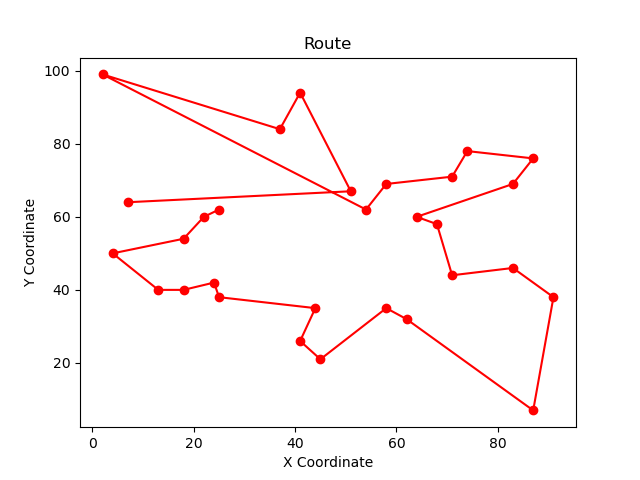
* 实验中记录了每次迭代的局部最优解和最终的全局最优解。
* 最终获得的最短路径和其长度显示在界面上。
* 首先是30个城市的结果，30个城市迭代多次才能找到最优解



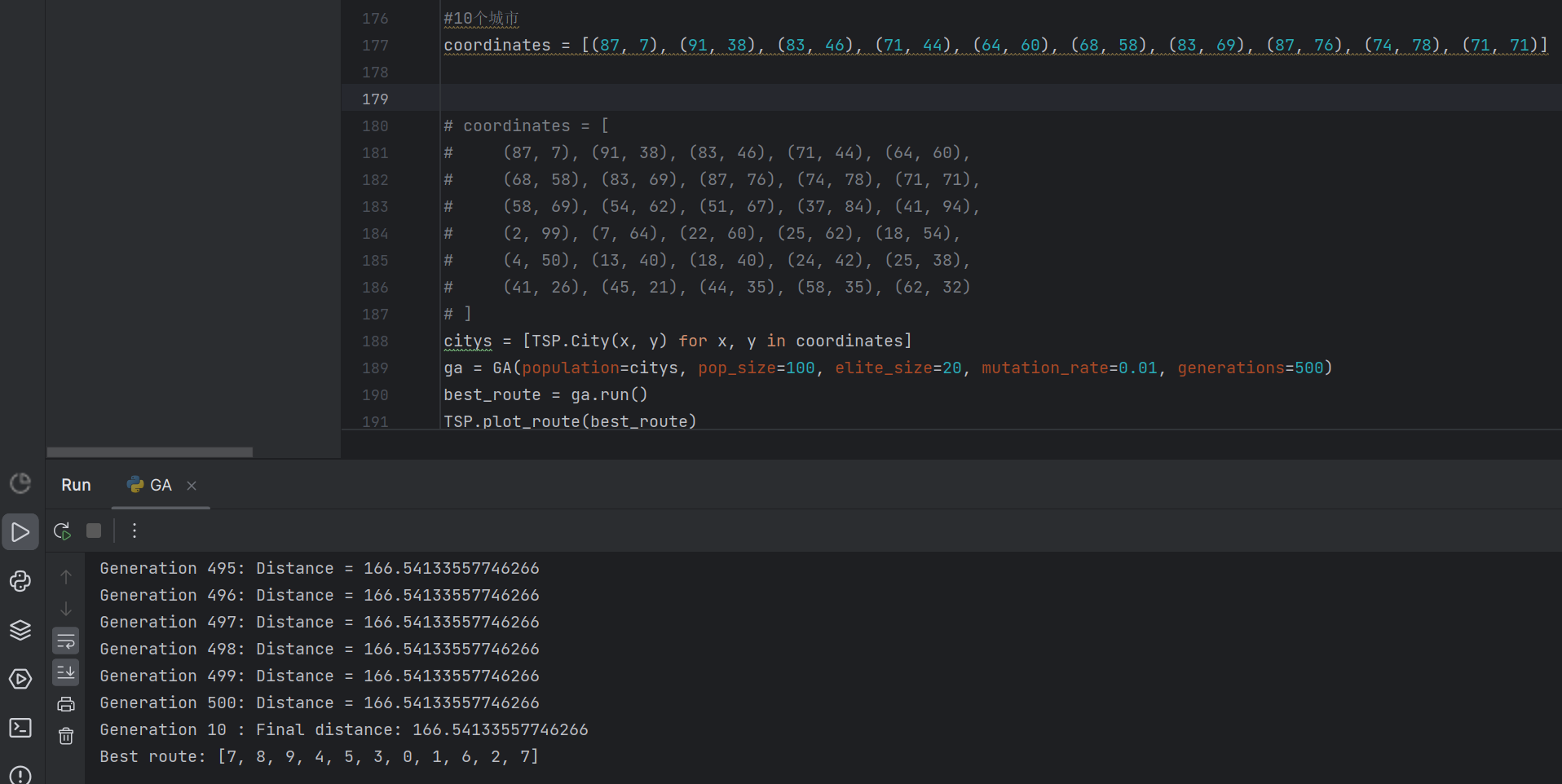


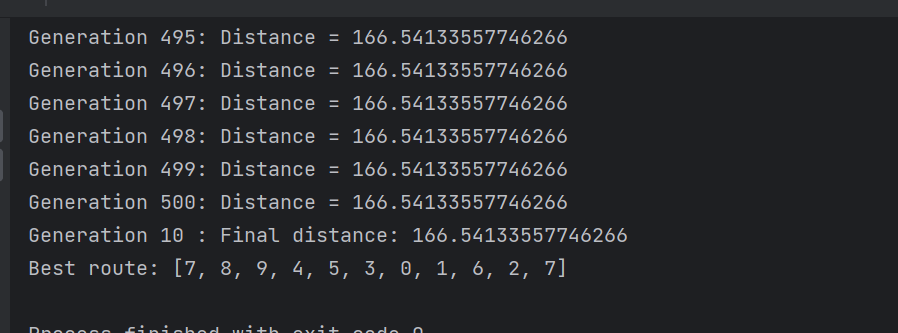
可以看到最终最短距离和实验预先给出的结果相同，路径类似，毕竟路径是一个循环路径，有多种结果。

以下是画出的图

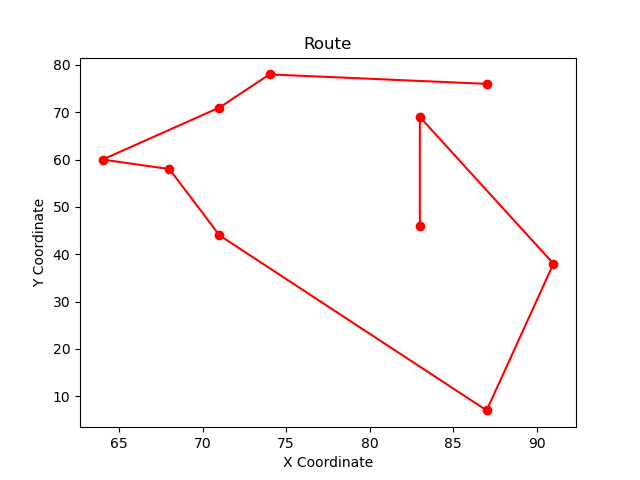


* 现在是10个城市的迭代结果，可以看到10个城市很快找到了最优解





可以看到10个城市很快找到了最优解。路径图如下



### 结果分析

* 实验结果显示遗传算法能够有效地逼近最优解。
* 由于遗传算法的随机性，不同的运行可能会产生不同的结果。
* 如果未达到理论上的最优解，可能是由于种群多样性不足、参数设置不合理或早熟收敛导致。

## 六、实验结论

本次实验通过使用遗传算法解决了一个具体的TSP问题。实验结果表明，遗传算法是一个强大的工具，可以有效地处理复杂的优化问题。通过合理的参数设置和足够的迭代次数，遗传算法能够找到非常接近最优解的解决方案。实验还展示了遗传算法的核心概念，如染色体表示、适应度计算、遗传操作等。总的来说，这次实验成功地达到了教学目标，加深了对遗传算法的理解并提高了使用遗传算法解决问题的能力。