

# 武汉大学国家网络安全学院

## 实验报告

课程名称 \_\_\_\_\_ 高级人工智能 \_\_\_\_\_

专业年级 \_\_\_\_\_ 网络空间安全 2023 级硕士 \_\_\_\_\_

姓 名 \_\_\_\_\_ 赵路路 \_\_\_\_\_

学 号 \_\_\_\_\_ 2023202210120 \_\_\_\_\_

实验学期 \_\_\_\_\_ 2023—2024 学年 \_\_\_\_\_ 第一 学期

填写时间 \_\_\_\_\_ 2023 年 \_\_\_\_\_ 12 月 \_\_\_\_\_ 25 日

## 实验概述

**【实验项目名称】：**基于遗传算法和蚁群算法的旅行商问题求解实验

**【实验目的】：**

- (1) 掌握遗传算法和蚁群算法的思想与方法。
- (2) 使用遗传算法和蚁群算法解决旅行商问题。
- (3) 分析比较两种算法的性能与特点。

**【实验环境】：**

- (1) 硬件环境：Mac mini 2023 with Apple M2 16GB
- (2) 操作系统环境：macOS Sonoma 14.1.2
- (3) 测试脚本编程语言：Python
- (4) 被测系统编程语言：Python
- (5) 网络环境：武汉大学校园网
- (6) 其他环境：conda 23.7.4

**【参考资料】：**

- (1) 蔡自兴. 人工智能及其应用[M]. 北京：清华大学出版社, 2016.
- (2) 周博涵. 基于遗传算法实现旅行商问题[EB/OL]. (2023-12-25)[2022-03-10].  
<https://zhuanlan.zhihu.com/p/478964224>

## 实验内容

**【实验方案设计】：**

1. 旅行商问题

(1) 问题定义

旅行商问题 (Traveling Salesman Problem, TSP) 是一类著名的组合优化问题，问题可形式化定义为：给定一组城市  $C = \{1, 2, \dots, n\}$  和它们之间的距离  $d_{ij}$ ，找到一条路径  $P$ ，使得一个旅行商可以从起始城市出发，经过所有城市一次，最终回到起始城市，路径  $P$  的总成本  $\sum_{i=1}^{n-1} d_{P_i P_{i+1}} + d_{P_n P_1}$  最小。

(2) 问题特点

旅行商问题是一个 NP-hard 问题，即在一般情况下，没有已知的有效算法能够在多项式时间内求解最优解。问题规模较小时，可以使用穷举法列举所有可能

的路径并计算其成本，找到最优解。但算法复杂性随着城市数量的增加而急剧增加。因此，解决旅行商问题常采用近似算法，如 K 最近邻法、最小生成树法、模拟退火算法、遗传算法等，虽无法保证找到最优解，但能够在合理的时间内找到次优解。此外，解决旅行商问题也常采用启发式算法，例如蚁群算法、粒子群算法等，这些算法通过模拟自然界的优化过程来寻找解决方案。

### (3) 实验设置

如图 1 所示，在本次实验中，选取中国 34 个省会城市的经纬度坐标作为问题输入，城市间距离使用两点间的欧式距离表示。

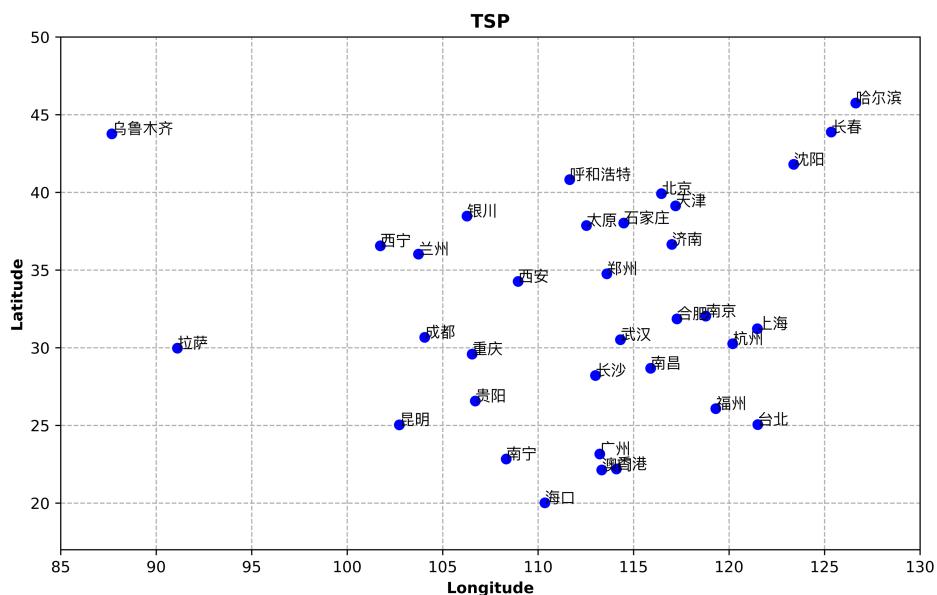


图 1 实验设置示意图

## 2. 遗传算法

遗传算法（Genetic Algorithm, GA）是一种基于自然选择和遗传机制的优化算法，用于在搜索空间中找到问题的近似或最优解。遗传算法的设计灵感来自于生物学中的进化过程，通过模拟遗传、交叉和变异等生物学中的概念，从一组解中生成新的解，并逐步优化。

遗传算法由美国计算机科学家和数学家 John Henry Holland 于 20 世纪 70 年代初提出的。1975，他在《自然系统的适应性基础》（Adaptation in Natural and Artificial Systems）一书中首次详细描述了遗传算法的原理和应用，如表 1 所示，他提出了遗传算法的基本概念，包括基因、染色体、交叉、变异等，这些概念构成了遗传算法的核心框架。

表 1 遗传算法基本概念表

概念	定义
个体	个体是问题的一个解决方案，通常用其体内的基因表示。
基因	基因是个体的编码形式，可以是一个二进制串、整数数组、浮点数数组等，具体形式取决于问题的特性。
适应度函数	适应度函数用于评估个体的优劣，反映了个体在解决问题上的性能。适应度越高，个体越有可能被选择参与后续的进化操作。
选择	选择操作根据个体的适应度值选择个体，高适应度的个体被选中的概率较大。选择模拟了自然选择中适者生存的过程。
交叉	交叉操作模拟了基因的重组过程。选中的个体进行染色体交叉，生成新的个体。
变异	变异操作通过改变个体的染色体中的基因值引入了随机性。变异模拟了基因的突变过程，有助于算法跳出局部最优解。
群体	群体是个体的集合，表示当前解空间中的一组潜在解决方案。群体中的个体通过进化操作逐步优化，形成新的群体。
迭代	遗传算法是一个迭代优化过程，通过一系列的选择、交叉、变异等操作不断演化群体，逐渐逼近最优解。

遗传算法的提出对于解决复杂的优化问题和搜索空间大的问题具有重要意义，成为进化计算领域的开创性工作之一。由于其强大的搜索和优化能力，遗传算法在各种领域都得到了广泛应用，成为进化计算领域的经典算法之一。

如图 2 所示，遗传算法的流程包括：创建初始种群、计算初始适应度、选择/交叉/变异操作、计算操作后适应度、判断终止条件、选择适应度最高个体。

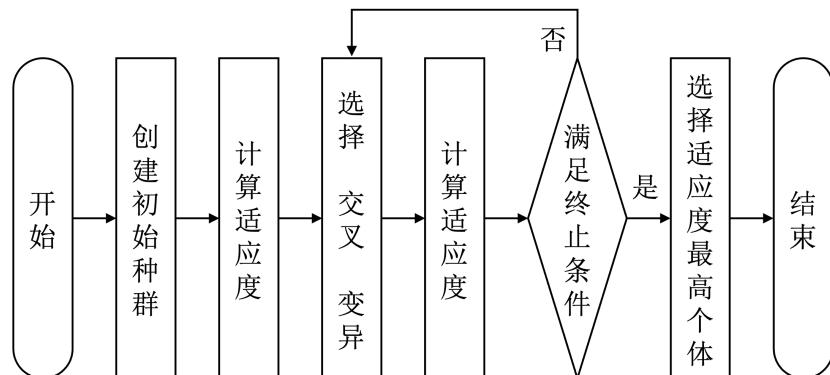


图 2 遗传算法流程图

### 3. 蚁群算法

蚁群算法（Ant Colony Optimization Algorithm, ACOA）是一种基于模拟蚂蚁觅食行为的启发式优化算法，最初由意大利计算机科学家 Marco Dorigo 于 1992 年提出。如表 2 所示，他提出了蚁群算法的基本概念，包括蚂蚁、路径、信息素、选择、更新、迭代等，这些概念构成了遗传算法的核心框架。

表 2 蚁群算法基本概念表

概念	定义
蚂蚁	蚁群算法中的基本个体单元，通过在问题空间中移动来构建解。
路径	蚂蚁在解空间中的移动路径，代表问题解空间中的一个潜在解。
信息素	蚁群算法引入的一种信息传递机制，用于模拟蚂蚁之间的通信。 蚂蚁在路径上释放信息素，信息素的浓度表示路径的好坏。
选择	蚂蚁在选择下一步移动的目标时，根据信息素浓度计算的概率。
更新	每轮迭代后，信息素根据蚂蚁的路径更新，受到蚂蚁行走的距离和信息素的挥发率的影响。
迭代	遗传算法是一个迭代优化过程，通过一系列的选择、更新等操作形成群体协作机制，逐渐逼近最优解。

蚁群算法模拟了蚂蚁群体在寻找食物时的协作行为，通过信息素的释放和更新实现群体智能的路径搜索。信息素在路径上根据蚂蚁行走的距离和信息素的挥发率更新，短路径上的信息素浓度逐渐增加，形成了群体的协作机制。通过迭代优化，蚁群算法能够寻找到问题的较优解，特别适用于组合优化问题。

如图 3 所示，蚁群算法的流程包括：随机初始化蚂蚁位置、选择下一位置、更新信息素矩阵、判断终止条件、选择蚁群中的最短路径。

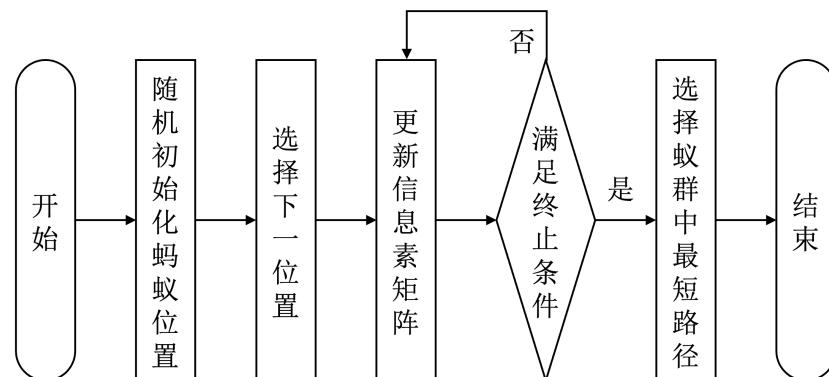


图 3 蚁群算法流程图

## 【实验过程、结果、分析】：

### 1. 遗传算法实验

#### (1) 代码实现

如图 4 所示，在创建初始种群阶段，随机生成一组基因。基因是一个 34 维数组，代表一条途径所有城市的路径。

```
1个用法  ± Simp1e_L
def init_population(self):
    for i in range(self.life_count):
        gene = [x for x in range(self.gene_length)]
        random.shuffle(gene)
        self.lives.append(Life(gene))
```

图 4 创建初始种群代码图

如图 5 所示，在计算适应度阶段，计算该个体体内基因代表的路径的总长度，其适应度函数定义为路径总长度的倒数。

```
3个用法  ± zllwhu
def distance_gene(self, gene):
    distance = 0.0
    for i in range(-1, len(self.cities) - 1):
        index1 = gene[i]
        index2 = gene[i + 1]
        city1 = self.cities[index1]
        city2 = self.cities[index2]
        distance += math.sqrt((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2)
    return distance

1个用法  ± zllwhu
def match_fun_ga(self):
    return lambda life: 1.0 / self.distance_gene(life.gene)
```

图 5 计算适应度代码图

如图 6 所示，在选择操作阶段，根据个体的适应度值，进行轮盘赌选择，即适应度越高的个体被选中的概率越大。

```
2个用法  ± Simp1e_L
def get_one(self):
    r = random.uniform( a: 0, self.bounds)
    for life in self.lives:
        r -= life.score
        if r <= 0:
            return life
```

图 6 选择操作代码图

如图 7 和图 8 所示，在交叉操作阶段，使用两点交叉的策略，即随机选取基因上的两个位点，截断两个父代个体的基因，进行交叉操作。

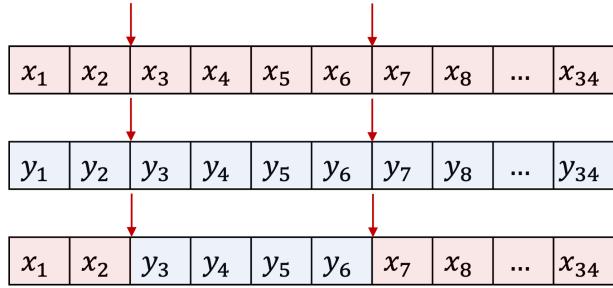


图 7 交叉操作示意图

```
1个用法 __Simp1e_L
def cross(self, parent1, parent2, max_test=120):
    test = 0
    while True:
        new_gene = []
        index1 = random.randint( a: 0, self.gene_length - 1)
        index2 = random.randint(index1, self.gene_length - 1)
        cross_gene = parent2.gene[index1:index2]
        i_flag = 0
        for g in parent1.gene:
            if i_flag == index1:
                new_gene.extend(cross_gene)
                i_flag += 1
            if g not in cross_gene:
                new_gene.append(g)
                i_flag += 1
            if self.match_fun(Life(new_gene)) > max(self.match_fun(parent1), self.match_fun(parent2)):
                self.cross_count += 1
                return new_gene
        if test > max_test:
            self.cross_count += 1
            return new_gene
        test += 1
```

图 8 交叉操作代码图

如图 9 和图 10 所示，在变异操作阶段，使用均匀变异的策略，即在变异过程中，每个基因都有一定概率被随机改变，引入随机性，避免陷入局部最优解。

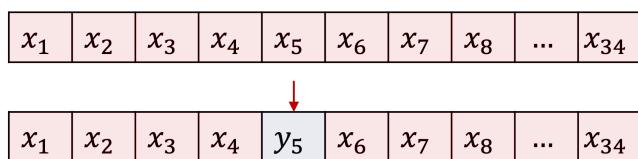


图 9 变异操作示意图

```

1个用法  ± Simple_L
def mutation(self, child):
    new_gene = child.gene[:]
    index1 = random.randint( a: 0, self.gene_length - 1)
    index2 = random.randint( a: 0, self.gene_length - 1)
    new_gene[index1], new_gene[index2] = new_gene[index2], new_gene[index1]
    if self.match_fun(Life(new_gene)) > self.match_fun(child):
        self.mutation_count += 1
        return new_gene
    else:
        rate = random.random()
        if rate < math.exp(-10 / math.sqrt(self.generation)):
            self.mutation_count += 1
            return new_gene
    return child.gene

```

图 10 变异操作代码图

如图 11 所示，在迭代终止条件判断阶段，当迭代次数达到要求时终止程序。

```

1个用法  ± zllwhu
def run_ga(self, max_iter=100, print_process=True):
    while max_iter > 0:
        self.ga.next_iter()
        distance = self.distance_gene(self.ga.best.gene)
        self.distance_record_ga.append(distance)
        if print_process:
            print("Generation: %4d \t\t Distance: %f" % (self.ga.generation - 1, distance))
            print("Optimal path: ", self.ga.best.gene)
        max_iter -= 1

```

图 11 迭代终止条件判断代码图

## (2) 实验结果

如表 3 所示，为本次实验使用的遗传算法超参数设置。

表 3 遗传算法超参数设置表

概念	定义
交叉概率	0.7
变异概率	0.02
群体中个体数量	100
基因长度	34
迭代次数	100

如图 12 所示，为随着迭代次数的增加，最优路径的总长度变化情况；如图 13 所示，为迭代 100 轮后，最优路径的可视化情况。

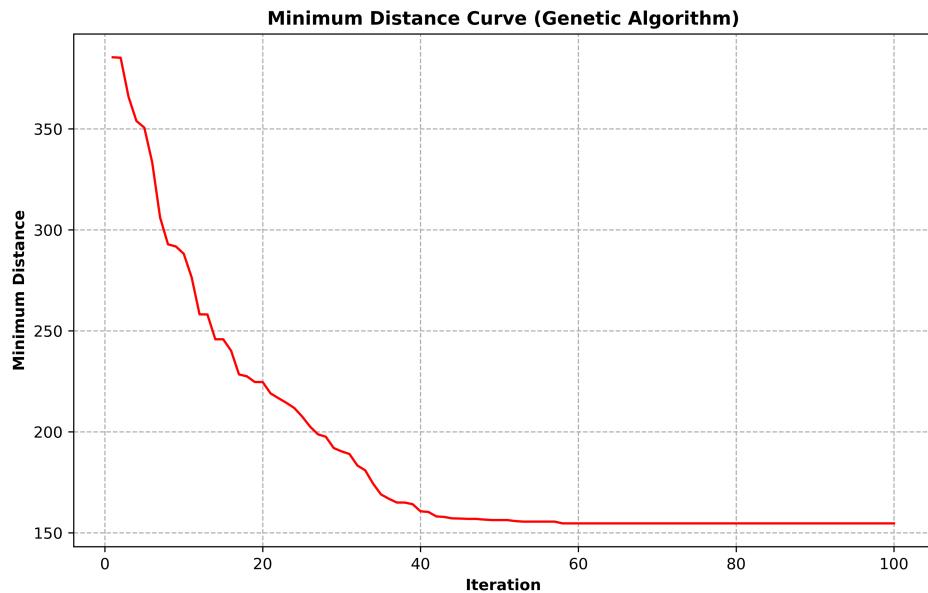


图 12 遗传算法最优路径长度迭代曲线图

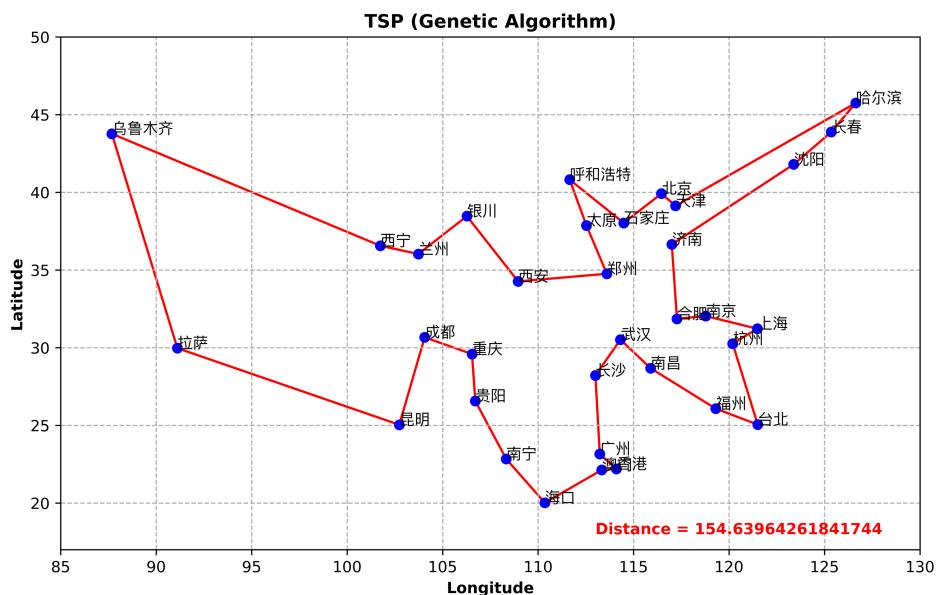


图 13 遗传算法最优路径示意图

## 2. 蚁群算法实验

### (1) 代码实现

如图 14 所示，在随机初始化蚂蚁位置阶段，若蚂蚁个数  $m$  少于或等于城市个数  $n$ ，则从城市索引范围内随机选择  $m$  个不重复的城市作为初始位置；若蚂蚁个数  $m$  多于城市个数  $n$ ，则从城市索引范围内随机选择  $n$  个城市作为初始位置。这段代码确保每只蚂蚁的初始位置在城市中是随机分布的。

```


# zllwhu *
def run(self):
    # 初始化蚂蚁位置
    if self.ant_num <= self.city_num:
        self.path_matrix[:, 0] = np.array(
            random.sample([i for i in range(self.city_num)], self.ant_num))
    else:
        self.path_matrix[:, 0] = np.array(
            [random.randint(0, self.city_num - 1) for _ in range(self.ant_num)])


```

图 14 随机初始化蚂蚁位置代码图

如图 15 所示，在选择下一位置阶段，为每只蚂蚁更新行进路径。计算每一个城市的信息素浓度和启发式信息的比值，归一化得到剩余未达城市的概率分布，采用轮盘赌的策略选择下一城市。

```


# 蚂蚁开始移动
local_distance = np.zeros(self.ant_num)
for k in range(self.ant_num):
    possibility = np.zeros(self.city_num)
    i = -1
    for i in range(self.city_num - 1):
        current_position = self.path_matrix[k, i]
        for next_position in range(self.city_num):
            if next_position in self.path_matrix[k]:
                possibility[next_position] = 0
            else:
                possibility[next_position] = (math.pow(self.pheromone_matrix[current_position, next_position],
                                            self.pheromone_importance) /
                                                math.pow(self.distance_matrix[current_position, next_position],
                                                       self.heuristic_importance))
        possibility = possibility / sum(possibility)
        self.path_matrix[k, i + 1] = roulette(possibility)
    local_distance[k] += self.distance_matrix[self.path_matrix[k, i], self.path_matrix[k, i + 1]]
    local_distance[k] += self.distance_matrix[self.path_matrix[k, i + 1], self.path_matrix[k, 0]]


```

图 15 选择下一位置代码图

如图 16 所示，在更新信息素矩阵阶段，在每只蚂蚁本轮路径上增加信息素浓度，增量为信息素总量/路径长度。

```


# 更新信息素
increment_pheromone_matrix = np.zeros_like(self.pheromone_matrix)
for k in range(self.ant_num):
    i = -1
    for i in range(self.city_num - 1):
        increment_pheromone_matrix[
            self.path_matrix[k, i], self.path_matrix[k, i + 1]] += self.pheromone_constant / local_distance[k]
        increment_pheromone_matrix[
            self.path_matrix[k, i + 1], self.path_matrix[k, i]] += self.pheromone_constant / local_distance[k]
    increment_pheromone_matrix[
        self.path_matrix[k, i + 1], self.path_matrix[k, 0]] += self.pheromone_constant / local_distance[k]
    increment_pheromone_matrix[
        self.path_matrix[k, 0], self.path_matrix[k, i + 1]] += self.pheromone_constant / local_distance[k]


```

图 16 更新信息素矩阵代码图

如图 17 所示，在迭代终止条件判断阶段，当迭代次数达到要求时终止程序。

```
1个用法  zllwhu
def run_acoa(self, max_iter=100, print_process=True):
    flag = 1
    while max_iter > 0:
        distance, path = self.acoa.run()
        self.distance_record_acoa.append(distance)
        if print_process:
            print(f'第{flag}次迭代,最短距离:{distance},最短路径:{path}')
        flag += 1
        max_iter -= 1
    return np.concatenate(
        (self.acoa.global_minima_path, [self.acoa.global_minima_path[0]])), self.acoa.global_minima
```

图 17 迭代终止条件判断代码图

## (2) 实验结果

如表 4 所示，为本次实验使用的蚁群算法超参数设置。

表 4 蚁群算法超参数设置表

概念	定义
蚂蚁数量	40
信息素重要程度	3
启发信息重要程度	2
信息素挥发率	0.2
信息素总量	100

如图 18 所示，为随着迭代次数的增加，最优路径的总长度变化情况。

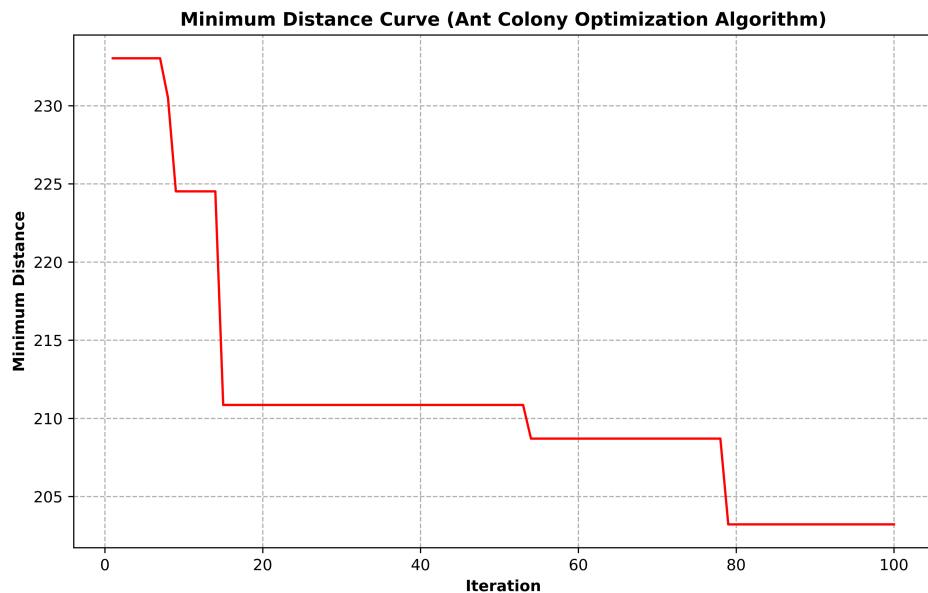


图 18 蚁群算法最优路径长度迭代曲线图

如图 19 所示，为迭代 100 轮后，最优路径的可视化情况。

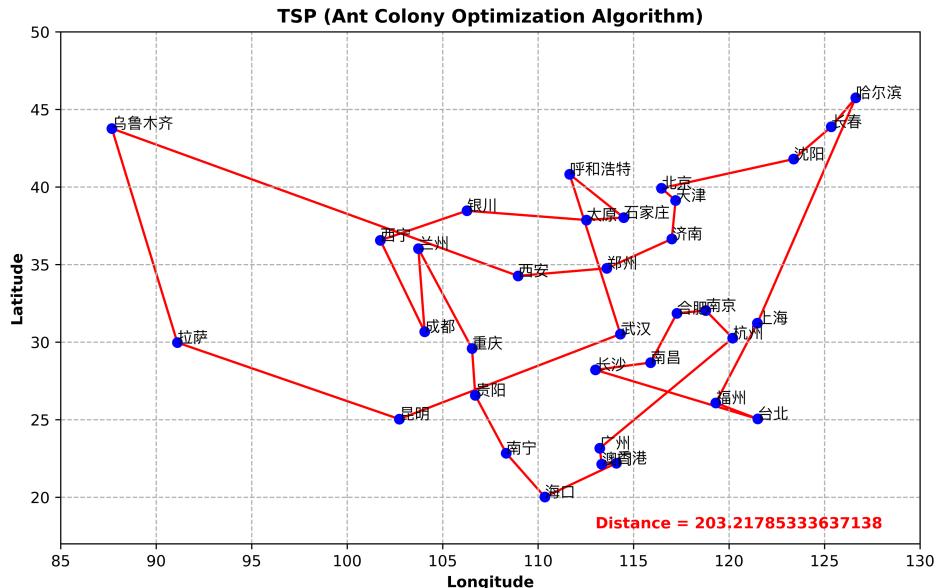


图 19 蚁群算法最优路径示意图

### 【小结】：

#### 1. 准确性分析

遗传算法和蚁群算法都是群体智能算法，通过模拟群体中个体之间的协作和信息共享来寻找解决方案。在被用于解决复杂的组合优化问题，二者都表现出随机性，对于不同的超参数设置、不同的迭代轮次、不同次运行，都可能得到不相同的解，且得到的解不一定是问题的最优解。

对比图 13 和图 15，我们可以看出在迭代轮次都为 100 时，遗传算法获得了更优的解，且此现象可以稳定复现。

**结论 1：**在本实验设计的旅行商问题语义下，遗传算法的准确性优于蚁群算法。这是由于蚁群算法对局部最优解比较敏感，会陷入局部最优解而难以跳出。

#### 2. 收敛性分析

对比图 14 和图 16，我们可以看出遗传算法在迭代 60 次后收敛，蚁群算法在迭代 80 次后才趋于收敛，且此现象可以稳定复现。

**结论 2：**在本实验设计的旅行商问题语义下，蚁群算法通常需要更多的迭代次数才能收敛到一个合适的解。

#### 3. 时间性能分析

通过对迭代耗时进行耗时记录，得到如图 20 的时间开销统计结果，我们可以看出遗传算法的平均耗时为 0.1815 秒，蚁群算法的平均耗时为 0.1329 秒，遗传算法的迭代操作耗时更久。但我们也发现，遗传算法的迭代耗时随轮次增加有明显的上升趋势，这是由于遗传算法中的交叉操作设置了尝试机制：当交叉后得到的子代个体适应度没有比父代两个体高时，重新交叉尝试，直到找到适应度高于两父代个体的子代个体或达到最大尝试次数为止。在迭代轮次增大后，种群中的个体适应度普遍高于初始种群，故尝试次数增多增加了耗时。

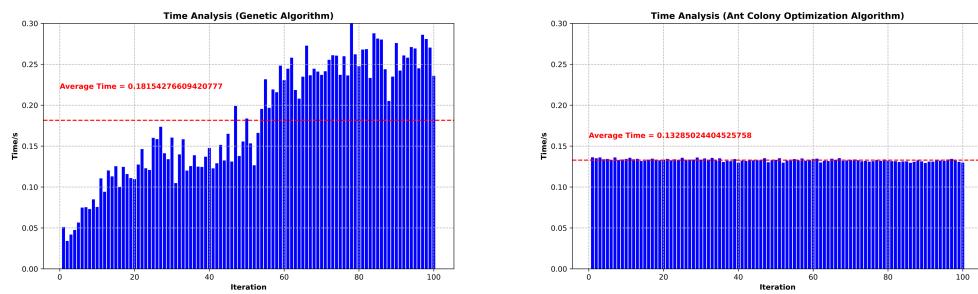


图 20 算法迭代耗时统计图

**结论 3：**蚁群算法较遗传算法具有更小的迭代平均耗时。

#### 4. 实验总结

通过本次实验，我掌握了遗传算法和蚁群算法的思想与方法，并使用遗传算法和蚁群算法解决了旅行商问题。在此基础上，通过实验，分析比较两种算法的性能与特点，得到如下结论：

- ✓ **结论 1：**在本实验设计的旅行商问题语义下，遗传算法的准确性优于蚁群算法。这是由于蚁群算法对局部最优解较敏感，会陷入局部最优解而难以跳出。
- ✓ **结论 2：**在本实验设计的旅行商问题语义下，蚁群算法通常需要更多的迭代次数才能收敛到一个合适的解。
- ✓ **结论 3：**蚁群算法较遗传算法具有更小的迭代平均耗时。

## 指导教师评语及成绩

【评语】：

成绩：

指导教师签名：

批阅日期：

附件：

### 实验报告说明

1. **实验项目名称：**要用最简练的语言反映实验的内容。要求与实验指导书中相一致。
2. **实验目的：**目的要明确，要抓住重点，符合实验任务书中的要求。
3. **实验环境：**实验用的软硬件环境（配置）。
4. **实验方案设计**（思路、步骤和方法等）：这是实验报告极其重要的内容。包括概要设计、详细设计和核心算法说明及分析，系统开发工具等。应同时提交程序或设计电子版。  
对于设计型和综合型实验，在上述内容基础上还应该画出流程图、设计思路和设计方法，再配以相应的文字说明。
- 对于创新型实验，还应注明其创新点、特色。
5. **结论（结果）：**即根据实验过程中所见到的现象和测得的数据，做出结论（可以将部分测试结果进行截屏）。
6. **小结：**对本次实验的心得体会，所遇到的问题及解决方法，其他思考和建议。
7. **指导教师评语及成绩：**指导教师依据学生的实际报告内容，用简练语言给出本次实验报告的评价和价值。