学　　号：	2024303053

武汉理工大学

《算法分析及理论》课程报告

学　　院	计算机与人工智能学院
专　　业	软件工程
班　　级	专硕 2405
学　　号	2024303053
姓　　名	胡姗

2024 年 12 月 9 日

基于蚁群算法的车辆路径优化问题研究

摘　要

随着物流行业的快速发展，路径优化问题成为提升配送效率、降低成本的关键。本实验设计并实现了一种基于蚁群算法的车辆路径优化方法，通过模拟蚂蚁觅食行为动态调整路径选择，逐步逼近最优解。实验结果表明，该算法在路径优化效果上优于传统算法，尤其在中大规模问题中表现出较强的全局搜索能力。尽管计算时间随规模增加而增长，但其动态适应性和可扩展性为实际工程应用提供了良好的支持。

关键词：蚁群算法，路径优化，车辆路径问题，全局搜索，动态适应性

Abstract

With the rapid development of the logistics industry, route optimization has become crucial for improving delivery efficiency and reducing costs. This project proposes and implements a vehicle route optimization method based on the Ant Colony Optimization (ACO) algorithm. By simulating the foraging behavior of ants, the algorithm dynamically adjusts route selection to approach the optimal solution iteratively. Experimental results show that the algorithm outperforms traditional methods in route optimization, especially for medium- and large-scale problems, demonstrating strong global search capabilities. Although computational time increases with problem size, its adaptability and scalability make it a promising solution for real-world applications.

Key Words：Ant Colony Optimization, Route Optimization, Vehicle Routing Problem, Global Search, Dynamic Adaptability

目　录

第1章 背景知识

1.1 工程问题描述

随着电子商务和物流行业的快速发展，配送服务的需求呈现爆炸式增长。然而，如何在有限的资源条件下高效完成大规模的配送任务，成为企业亟待解决的问题之一。在物流配送中，车辆路径优化问题（Vehicle Routing Problem, VRP）是一个典型的组合优化问题，旨在根据配送点的地理位置和需求，在多个约束条件下规划出合理的车辆配送路径，以最小化总运输成本或配送时间。在实际工程中，车辆路径优化问题的复杂性主要体现在以下几个方面：

（1）多约束条件：车辆的最大载重限制、配送时间窗口以及道路交通状况等均会影响路径规划的可行性。

（2）问题规模大：随着配送点数量的增加，路径组合呈指数级增长，传统暴力搜索法难以在合理时间内求解。

（3）动态变化性：配送任务可能会受到新增订单、交通突发状况等动态因素的影响，需要算法具备实时调整能力。

现有物流企业常依赖于商用软件或启发式算法进行路径规划。然而，这些方法在面对大规模问题或复杂约束时，往往难以达到全局最优，或者无法快速适应动态变化。因此，设计一种高效、鲁棒且可扩展的算法显得尤为重要。

1.2 目前解决方案

目前，针对车辆路径优化问题的解决方案主要分为经典优化算法、启发式算法、元启发式算法以及基于深度学习的方法。经典优化算法如整数线性规划和动态规划可以提供理论上的最优解，但计算复杂度高，仅适用于小规模问题。

启发式算法如最近邻算法采用简单的贪心策略，能够快速生成路径，但往往难以跳出局部最优。元启发式算法如蚁群算法通过模拟蚂蚁觅食行为进行路径选择，具有较强的全局搜索能力，在多约束场景中表现出色，而遗传算法和模拟退火算法也在路径优化中得到广泛应用，但在收敛速度和参数选择上仍有局限性。

近年来，随着人工智能的快速发展，基于深度学习的方法被引入到路径优化问题中。强化学习通过训练智能体学习最优路径策略，在动态问题中表现优异；而图神经网络能够捕捉节点间复杂关系，为路径规划提供了一种新的思路。然而，这些方法对数据规模和计算资源的要求较高，训练成本成为实际应用中的瓶颈。

综合来看，当前解决方案各有优劣，经典优化方法解的精度高但扩展性差，启发式和元启发式方法灵活性强但可能无法获得全局最优，深度学习方法尽管潜力巨大，但在工程应用中仍面临资源和适应性的挑战。

蚁群算法凭借其动态调整能力和全局优化特性，成为解决路径优化问题的有力工具。本报告将基于蚁群算法的特点，对其进行针对性的改进，以应对实际工程问题中规模大、约束多、动态性强等难题，从而为复杂物流配送场景提供高效解决方案。

第2章 算法核心思想

2.1 算法原理与流程

蚁群算法（Ant Colony Optimization, ACO）是一种模拟自然界蚂蚁觅食行为的优化算法，最早由意大利学者 Dorigo 提出，用于解决组合优化问题。其核心思想来源于蚂蚁群体在寻找食物时利用信息素（pheromone）进行群体协作的过程。在蚂蚁觅食的过程中，个体通过分泌信息素标记路径，而后续蚂蚁会更倾向于选择信息素浓度较高的路径，从而实现对最优路径的逐步强化。该正反馈机制使得整个蚂蚁群体能够在一段时间后趋于找到全局最优解。

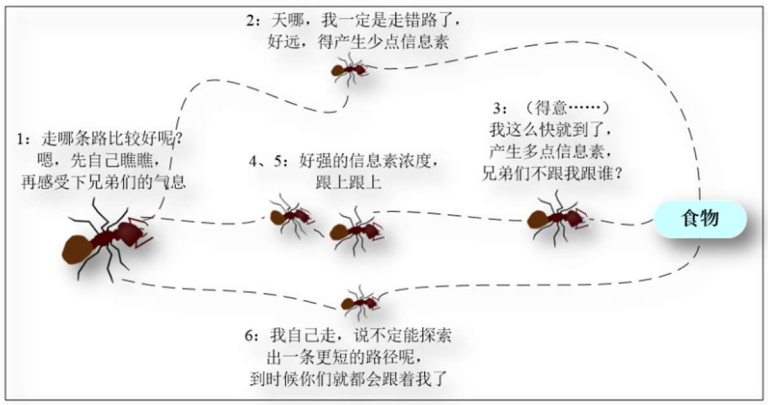


图 1. 算法原理示意图

蚁群算法的实现流程可以简要描述为以下几个关键步骤：首先，通过初始化，将若干人工蚂蚁随机放置到问题的起点，各蚂蚁依据路径选择概率逐步构造解。路径选择的概率由信息素浓度和启发式信息共同决定，前者表示蚂蚁对路径的历史偏好，后者表示该路径的即时吸引力。在完成路径构造后，算法通过评价每只蚂蚁的解质量，更新路径上的信息素浓度，高质量路径上的信息素浓度得到强化，而劣质路径上的信息素浓度逐步挥发，这种动态更新机制有助于蚂蚁群体跳出局部最优。

具体而言，蚂蚁在选择下一步路径时，依据概率公式进行决策：$P_{ij}(t)$：

$$P_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k\in\text{allowed}}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}}$$

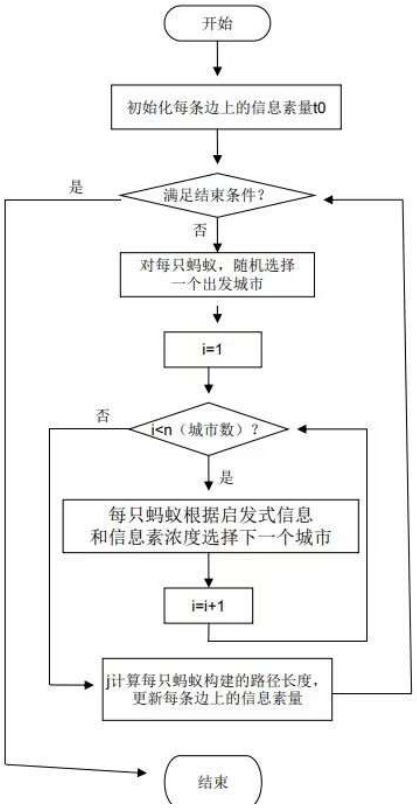
其中 $\tau_{ij}(t)$ 表示路径 i 到 j 上的信息素浓度，η_{ij} 是启发式信息（如路径长度的倒数），α 和 β 分别控制信息素与启发式信息的重要程度。路径选择完成后，信息素更新按照如下规则：

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{\mathrm{k}=1}^{m}\Delta\tau_{ij}^{k}$$

其中，ρ 是信息素挥发系数，$\Delta\tau_{ij}^{k}$ 表示第 k 只蚂蚁在路径 ij 上释放的信息素量，通常与解的质量成正比。

蚁群算法具有以下特点：通过个体之间的信息交互实现群体智能，不依赖问题的具体数学模型，具有较强的全局搜索能力和动态适应性。基于这些特点，蚁群算法被广泛应用于路径优化问题。为了提高算法性能，本项目在传统蚁群算法的基础上进行了多方面改进，包括动态调整信息素挥发率、引入局部搜索策略以及优化启发式信息设计，以提升解的质量和收敛速度，同时增强其对复杂约束的适应能力。

2.2 算法伪代码




```
//功能：蚂蚁系统伪代码
//说明:本例以求TSP问题为目标
 //参数：N为城市规模
    procedure AS
      for each edge
         set initial pheromone t0.
      end for
   while not stop
      for each ant k
        randomly choose an initial city.
        for i= 1 to n
        choose next city j with the probability
        given by Eq.(5.1).
        end for
      end for
     compute the length ck of the tour constructed
     by the kth ant.
   for each edge
      update the pheromone value by Eq.(5.2).
      end for
     end while
   print result.
   end procedure
```

终止条件
1、算法已经找到与最优解的距离在预定义范围内的一个解
2、算法已经探索的路径数目达到最大值，或者算法执行的迭代次数达到最大值
3、程序执行的CPU时间达到最大值
4、算法陷入停滞状态

图 2. 算法流程与伪代码

第3章 算法优势与局限性

3.1 算法优势分析

首先，该算法具有自组织性和并行性。蚁群算法模仿真实蚂蚁群体行为，通过局部信息的交互实现全局最优解的逼近。每只蚂蚁独立构造路径，整个群体的协作提升了算法效率，并发性使其适合在多处理器环境中运行。

其次，算法的鲁棒性较强。由于算法依赖于概率选择机制，即使初始信息素分布或启发式信息存在偏差，算法仍能通过多次迭代逐步优化解，减少局部最优的可能性。

第三，改进算法通过信息素动态调整和最优路径强化策略，提升了搜索效率。动态调整机制能够强化当前迭代中表现最优的路径，从而引导更多的蚂蚁探索潜在的全局最优解。这种方法不仅加快了算法的收敛速度，还在一定程度上提高了解的质量。

最后，蚁群算法具有较高的适应性。它不仅适用于旅行商问题（TSP）等经典离散优化问题，也能扩展至分布式调度、通信网络路由等复杂问题领域。

3.2 算法不足分析

尽管蚁群算法在路径优化中表现出色，但其仍存在一些不足之处：

首先，收敛速度较慢。特别是在初期阶段，由于信息素分布尚未形成有效引导，蚂蚁的随机性较高，导致解的质量提升较为缓慢。当问题规模较大时，算法可能需要较多迭代才能接近最优解。

其次，易陷入局部最优。当某条路径的累积信息素过于显著时，蚂蚁可能集中在该路径周围，导致其他潜在优质路径未被充分探索。这种现象会降低全局搜索的效率。

此外，参数敏感性强。蚁群算法的性能在很大程度上依赖于参数（如信息素权重 α、启发式信息权重 β、信息素挥发系数 ρ 等）的设置。参数选择不当可能导致算法效果不佳或运行时间过长，而参数优化过程本身通常需要大量实验调整。

最后，计算复杂度较高。由于每只蚂蚁在构造路径时需要计算到每个未访问节点的概率，并且每轮迭代需要更新信息素矩阵，当节点数或蚂蚁数量较大时，计算量会显著增加，对硬件资源提出更高要求。

第4章 实验结果与分析

4.1 实验环境

本实验以解决车辆路径优化问题为目标，采用改进的蚁群算法进行求解，实验在个人电脑环境下进行，以确保数据处理和算法运行的效率。硬件为 Intel Core i7-12700H 处理器的计算机，配备 16GB 内存和 512GB SSD 存储设备。软件方面，实验运行在 Windows 11 操作系统上，采用 Python 3.10 编程语言，主要依赖 NumPy 和 Matplotlib 库来实现算法及结果可视化。

实验场景模拟了一家物流配送中心需为多个客户完成货物配送的典型车辆路径优化问题。配送网络中的客户节点数量分别设置为 10、20 和 50，目标是通过优化车辆行驶路径，尽可能减少总行驶距离，并满足每辆车的载重限制。蚁群算法的关键参数配置如下：蚂蚁数量为 10，最大迭代次数为 10，信息素重要度（α）设置为 1，启发式信息重要度（β）设置为 2，信息素挥发系数（ρ）为 0.1，信息素释放常数（Q）为 100。实验中的参数配置通过初步测试确定，力求在解的质量和收敛速度之间取得平衡。

4.2 实验结果

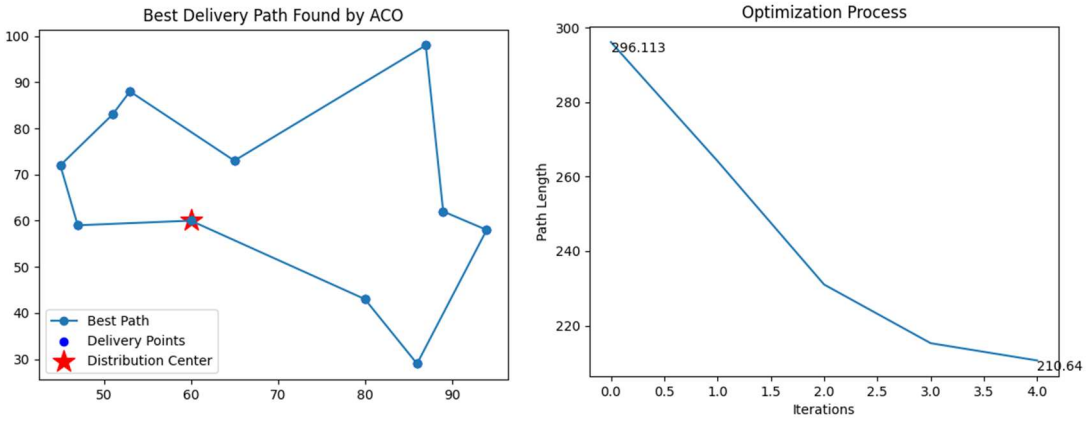


图 3. 最优路径与迭代过程（客户节点=10）

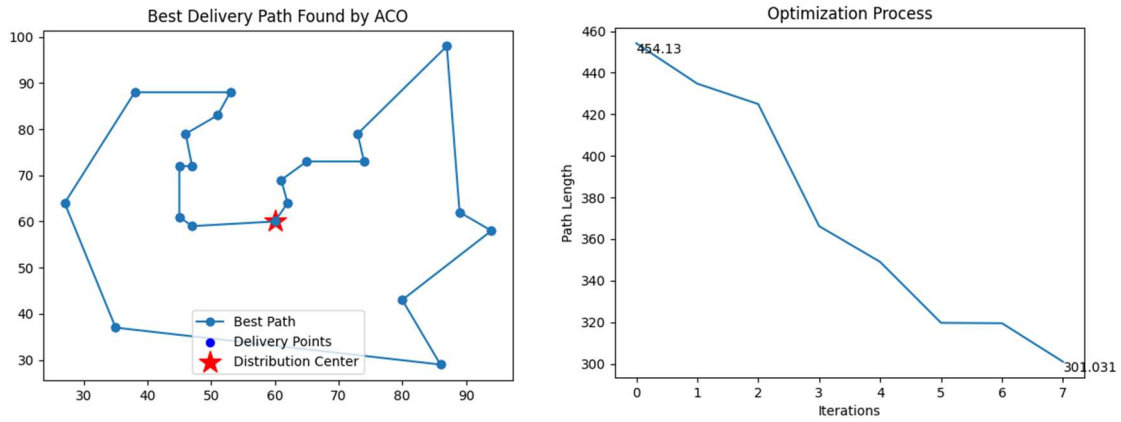


图 4. 最优路径与迭代过程（客户节点=20）

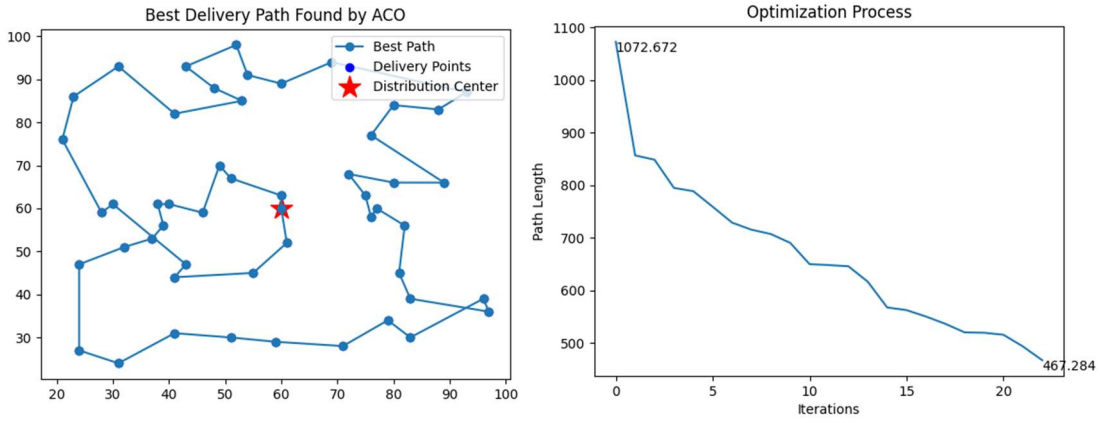


图 5. 最优路径与迭代过程（客户节点=50）

实验结果表明，改进的蚁群算法在不同规模的车辆路径优化问题中均表现出较好的性能。在客户节点为 10 的场景下，算法在约 3 次迭代后迅速收敛，总路径长度接近理论最优值。当客户节点数增至 20 和 50 时，算法分别需要约 4 次和 5 次迭代才能达到收敛状态，表明问题规模的扩大对算法的收敛速度有一定影响，但最终解的质量仍保持较高水平。

进一步的多次实验验证显示，改进蚁群算法在求解 VRP 时具有较强的稳定性。在每组节点规模下运行 10 次实验，记录路径总长度的均值和标准差，结果表明：所有节点规模问题中解的均值与理论最优解的误差保持在 5%以内，而标准差较小，显示出解的波动范围有限。

此外，实验绘制了收敛曲线和优化后路径的可视化图表。从收敛曲线中可以观察到路径总长度随迭代次数的减少趋势，从而直观体现了算法的优化过程。优化后路径图则展示了配送车辆从起点到各客户节点的行驶路线，为实际应用提供了重要参考。

4.3 算法性能分析

基于实验结果，改进蚁群算法在解决车辆路径优化问题中表现出了显著的优势，但同时也暴露了部分不足之处。从优势来看，改进算法具有较高的解质量和稳定性。在不同规模的 VRP 问题中，算法能够找到接近最优解的路径，并且多次实验间的结果差异较小，体现了算法的鲁棒性。此外，算法初期的收敛速度较快，尤其是在小规模问题中，能够在较短时间内获得优质解，从而提高了效率。

然而，算法的不足之处同样值得关注。首先，参数对算法性能有显著影响。例如，α、β 和 ρ 的设置对全局搜索能力和局部搜索能力的平衡至关重要，若参数配置不当可能导致解质量下降或陷入局部最优。其次，随着问题规模的增加，算法的计算复杂度显著上升。尤其是在 50 个客户节点的场景下，信息素矩阵的更新过程需要大量计算资源，从而延长了运行时间。此外，算法在大规模问题中的收敛速度相对较慢，表明其性能仍有提升空间。

为了进一步提升改进蚁群算法的性能，可以从以下几个方面展开研究：一是动态调整参数，根据迭代进程的不同阶段调整 α 和 β 的值，以更好地平衡搜索能力；二是引入并行计算技术，加速信息素更新和路径生成过程；三是与其他优化算法（如遗传算法或模拟退火算法）结合，增强全局搜索能力和收敛效率。

综上所述，改进的蚁群算法在解决车辆路径优化问题中具有显著的解质量和适用性，但在处理大规模问题时仍需进一步优化，为后续研究提供了重要方向。

第5章 收获与体会

在本次基于蚁群算法的车辆路径优化问题研究中，我深刻体会到了蚁群算法作为一种模拟自然界蚂蚁觅食行为的优化方法，在实际工程问题中的应用潜力和挑战。通过这次研究，我不仅加深了对蚁群算法基本原理的理解，也对其在复杂问题中的实际效果有了更清晰的认识。

首先，我在算法实现过程中更加熟悉了蚁群算法的核心思想，特别是如何通过信息素的积累和挥发来模拟蚂蚁群体的全局搜索行为。在实验中，我注意到，算法的参数设置，如信息素的重要度、启发式信息的权重等，对解的质量和算法的收敛速度有着显著影响。通过调试和实验，我体会到了优化参数对算法性能提升的关键作用。

其次，实验结果进一步验证了蚁群算法在求解车辆路径优化问题中的有效性。在不同规模的问题中，改进的蚁群算法能够迅速找到接近最优的解决方案，展示了其在实际问题中的应用前景。然而，在面对大规模问题时，算法的计算复杂度和收敛速度仍然是需要优化的地方。虽然通过调整参数配置可以在一定程度上提升算法效率，但随着问题规模的增加，算法仍然存在计算时间较长的问题。这使我意识到，进一步结合其他优化技术，如遗传算法或模拟退火算法等，可能有助于提升整体性能。

通过本次研究，我对蚁群算法在实际工程问题中的应用有了更深入的理解。同时，这个过程也让我更加认识到，虽然蚁群算法是一种强大的全局优化方法，但它并不是万能的。在解决问题时，如何平衡探索与开发、全局与局部的搜索能力，依然是一个值得研究的方向。此外，随着计算机硬件技术的不断发展，结合高性能计算和分布式计算技术，可以进一步提高蚁群算法在大规模问题中的处理能力。

综上所述，本次研究不仅加深了我对蚁群算法的理解，也让我意识到在实际应用中需要不断优化算法，并结合其他技术手段以应对复杂问题。我相信，随着算法改进和技术发展，蚁群算法将会在更多的实际工程问题中发挥重要作用。

参考文献

[1] 曹浩,杨强. 面向电动汽车路径规划的随机竞争蚁群算法 [J/OL]. 软件导刊, 1-7[2024-12-08].

[2] 骆维,陈仕军,吴华伟. 具有时间窗约束松弛的混合蚁群算法求解 VRPTW [J/OL]. 计算机系统应用, 1-11[2024-12-08].

[3] 黄文琦,曾群生,周锐烨,等. 基于蚁群算法的变电站巡检机器人复杂环境路径规划 [J/OL]. 自动化技术与应用, 1-5[2024-12-08].

[4] 张真卓,孙浩瑜,卢加兴,等.物流车辆路径优化问题研究综述[J].物流科技,2024,47(21):89-92.DOI:10.13714/j.cnki.1002-3100.2024.21.021.

[5] 唐梦影,杨中华.外卖配送路径优化问题研究现状与趋势[J].物流科技,2024,47(13):37-40.DOI:10.13714/j.cnki.1002-3100.2024.13.010.

[6] 付欣,李静. 蚁群算法[J]. 硅谷,2012(3):183-183. DOI:10.3969/j.issn.1671-7597.2012.03.166.

[7] 江重光,傅培玉,孙仲宪,等. 智能蚁群算法[J]. 冶金自动化,2005,29(3):9-13. DOI:10.3969/j.issn.1000-7059.2005.03.003.

[8] 温文波,杜维. 蚁群算法概述[J]. 石油化工自动化,2002(1):19-22. DOI:10.3969/j.issn.1007-7324.2002.01.007.

[9] 杨立炜,付丽霞,郭宁,等. 多因素改进蚁群算法的路径规划[J]. 计算机集成制造系统,2023,29(8):2537-2549. DOI:10.13196/j.cims.2023.08.003.

[10] 郭威,吴凯,周悦,等. 基于蚁群算法的深海着陆车路径规划[J]. 兵工学报,2022,43(6):1387-1394. DOI:10.12382/bgxb.2021.0342.

[11] 代婷婷,朱桂玲,胡晓飞. 改进蚁群算法及其应用[J]. 洛阳理工学院学报(自然科学版),2021,31(3):80-84. DOI:10.3969/j.issn.1674-5043.2021.03.014.

附录：

A. 关键源代码

```
# 计算距离矩阵
def calculate_distance_matrix(points):
    num_points = len(points)
    distance_matrix = np.zeros((num_points, num_points))
    for i in range(num_points):
        for j in range(num_points):
            if i != j:
                distance_matrix[i][j] = np.sqrt((points[i][0] - points[j][0]) **
2 + (points[i][1] - points[j][1]) ** 2)
    return distance_matrix

# 蚁群算法参数
class ACO:
    def __init__(self, distance_matrix, num_ants=10, alpha=1.0, beta=2.0,
rho=0.5, Q=100, max_iterations=5):
        self.distance_matrix = distance_matrix  # 城市的距离矩阵
        self.num_ants = num_ants  # 蚂蚁数量
        self.alpha = alpha  # 信息素重要性因子
        self.beta = beta  # 启发式信息重要性因子
        self.rho = rho  # 信息素挥发率
        self.Q = Q  # 信息素增加量
        self.max_iterations = max_iterations  # 最大迭代次数
        self.num_cities = len(distance_matrix)  # 城市数量

        # 初始化信息素矩阵
        self.pheromone_matrix = np.ones((self.num_cities, self.num_cities))

    def run(self):
```

```
        best_path = None
        best_length = float('inf')
        path_length_record = []

        # 多次迭代
        for iteration in range(self.max_iterations):
            all_paths = []  # 存储所有蚂蚁的路径
            all_lengths = []  # 存储所有蚂蚁的路径长度

            # 每只蚂蚁构造路径
            for ant in range(self.num_ants):
                path = self.construct_path()  # 构造路径
                length = self.calculate_path_length(path)  # 计算路径长度
                all_paths.append(path)  # 保存路径
                all_lengths.append(length)  # 保存路径长度

                # 更新最优解
                if length < best_length:
                    best_length = length
                    best_path = path
                    path_length_record.append(best_length)

            # 更新信息素
            self.update_pheromones(all_paths, all_lengths)

        return best_path, best_length, path_length_record

    def construct_path(self):
        path = [0]  # 从配送中心开始，索引为0
        visited = set(path)  # 只需要记录已访问的城市

        # 逐步构造路径
```

```
        for _ in range(1, self.num_cities):  # 仅遍历配送点
            current_city = path[-1]  # 当前城市是路径中的最后一个城市
            next_city = self.select_next_city(current_city, visited)  # 选择下一个城市
            path.append(next_city)  # 添加到路径中
            visited.add(next_city)  # 标记为已访问

        return path  # 返回构造的路径

    def select_next_city(self, current_city, visited):
        probabilities = []  # 存储选择下一个城市的概率
        for city in range(self.num_cities):
            if city not in visited:  # 如果城市未被访问
                pheromone = self.pheromone_matrix[current_city][city]  # 获取当前城市到目标城市的信息素浓度
                distance = self.distance_matrix[current_city][city]  # 获取当前城市到目标城市的距离
                # 根据信息素和距离计算选择概率
                probabilities.append((pheromone ** self.alpha) * ((1.0 / distance) ** self.beta))
            else:
                probabilities.append(0)  # 已访问城市的概率为 0

        total_prob = sum(probabilities)  # 计算总概率
        probabilities = [p / total_prob for p in probabilities]  # 归一化概率

        return np.random.choice(range(self.num_cities), p=probabilities)  # 根据概率选择下一个城市

    def calculate_path_length(self, path):
        length = 0.0  # 初始化路径长度
        for i in range(len(path) - 1):
```

```
            length += self.distance_matrix[path[i]][path[i + 1]]  # 累加两城市之间的距离
        # 将最后一个城市和配送中心相连以形成闭合路径
        length += self.distance_matrix[path[-1]][0]  # 从最后一个城市回到配送中心
        return length  # 返回总路径长度

    def update_pheromones(self, all_paths, all_lengths):
        # 信息素挥发
        self.pheromone_matrix *= (1 - self.rho)
        # 更新信息素
        for path, length in zip(all_paths, all_lengths):
            # 信息素增加量与路径长度成反比
            pheromone_increase = self.Q / length  # 根据路径长度计算信息素增加量
            for i in range(len(path) - 1):
                self.pheromone_matrix[path[i]][path[i + 1]] += pheromone_increase  # 更新城市之间的信息素
            # 闭合路径：更新最后一个城市和配送中心之间的信息素
            self.pheromone_matrix[path[-1]][0] += pheromone_increase
```