武漢大學

# 国家网络安全学院

## 研究生课程论文

# 自适应无人机配送路径规划算法设计

评阅人签名：_____复核人签名：_____得分：_____


姓　　名　陈诺

学　　号　2023202210011

专业/班级　网络空间安全


课程名称　高级算法设计与分析

指导教师　林海老师

2024 年 06 月 06 日

# 目 录

# 1 引　言

## 1.1 背景介绍

在现代物流配送领域中，最后 10 公里的配送一直是一个挑战。传统的物流配送方案常受限于交通拥塞、区域配送员数量不足等问题，进而导致配送效率低、成本高昂、服务质量难以保证等情况的出现。无人机配送方式因其高效快速、灵活性强的特点，逐渐走进人们的生活中。高配送效率与高质量以及低配送成本，为电商、医药、食品等领域广泛的应用场景带来了新的机遇和发展空间。

## 1.2 问题定义

需要设计一种无人机配送路径规划算法，该算法要求在满足订单优先级要求下，最小化总配送路径，以提高配送效率、降低成本，并保证配送服务质量。在下图所示的区域中，共有 j 个配送中心（j=3），任意一个配送中心都有用户所需要的商品，其数量无限，同时任一配送中心的无人机数量无限。该区域共有 k 个卸货点（k=9），无人机只需要将货物放到相应的卸货点即可。

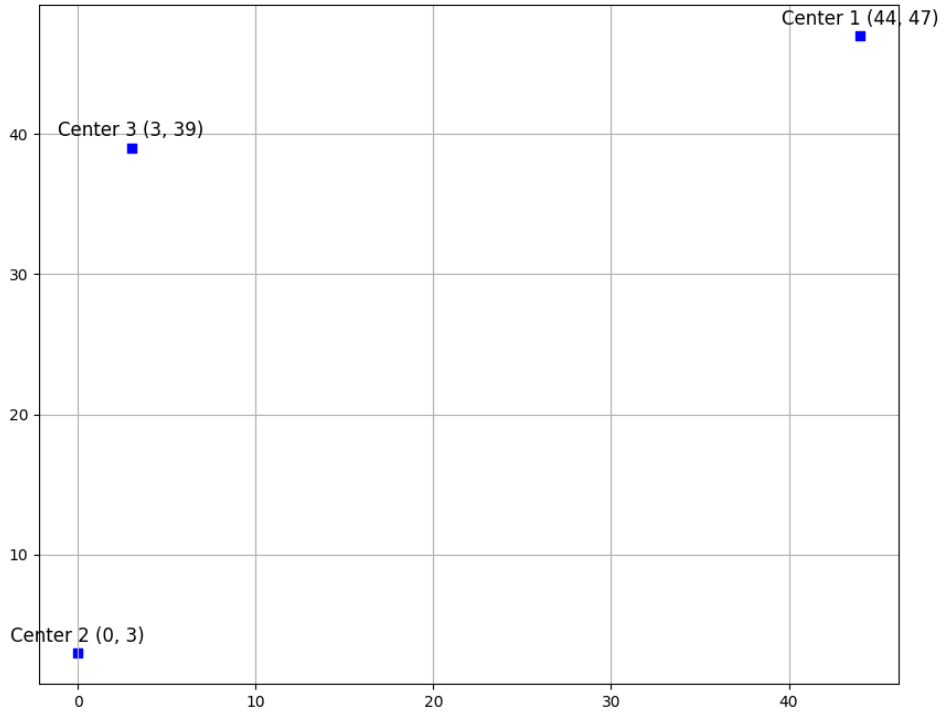


图 1-1 配送中心位置

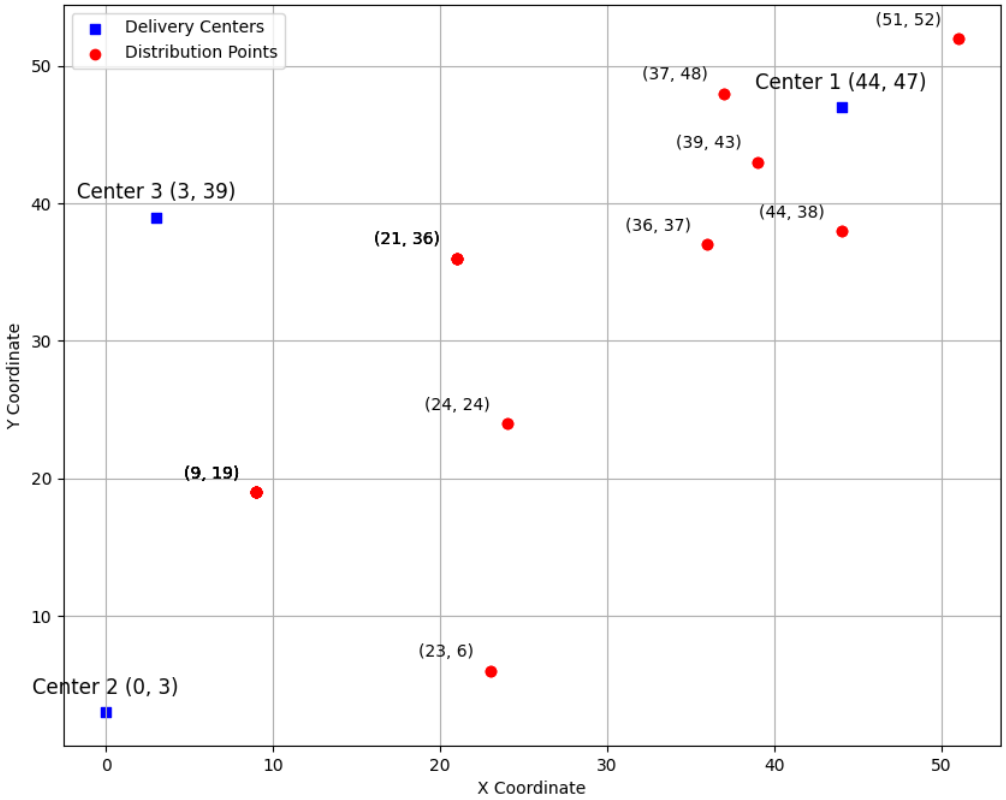


图 1-2 卸货点位置

假设每个卸货点会随机生成订单，一个订单只有一个商品，但这些订单有优先级别，分为三个优先级别（用户下订单时，会选择优先级别，优先级别高的付费高）：

- 一般（General）：3 小时内配送到即可；
- 较紧急（Urgent）：1.5 小时内配送到；
- 紧急（Very Urgent）：0.5 小时内配送到。

在真实应用场景中，需要将时间离散化，也就是每隔 t 分钟，所有的卸货点会生成订单（0-m 个订单），同时每隔 t 分钟，系统要做成决策。决策内容包括：

1. 哪些配送中心出动多少无人机完成哪些订单；
2. 每个无人机的路径规划，即先完成哪个订单再完成哪个订单，...，最后返回原来的配送中心；

【注意】系统做决策时，可以不对当前的某些订单进行配送，因为当前某些订单可能紧急程度不高，可以累积后和后面的订单一起配送。

问题的目标是使得一段时间内（如一天），所有无人机的总配送路径最短。问题的约束条件是需要满足订单的优先级要求。假设条件为：

1. 无人机一次最多只能携带 n 个物品（n=2）；
2. 无人机一次飞行最远路程为 20 公里（无人机送完货后需要返回配送点）；
3. 无人机的速度为 60 公里/小时；

4. 配送中心的无人机数量无限；

5. 任意一个配送中心都能满足用户的订货需求；

设计算法的输入是配送中心的位置、订单信息列表，输出是配送中心负责的订单列表和所有无人机的配送路径。

## 1.3 解决思路

个人认为无人机配送路径规划问题非常类似于 1972 年 Karp 提出的 21 个 NPC 问题里的 Job sequencing 优化。该问题也被称为 Job Scheduling Problem。其中有一系列需要完成的作业，每个作业都有一个截止时间和收益值。目标是确定一个作业的执行顺序，使得在不超过任何作业截止时间的前提下，最大化总收益。这里需要完成的作业与无人机配送问题中的订单对应，作业的截止时间和订单最大允许配送时间对应。基于订单优先级和实际配送时间进行收益值累积。因此，本实验考虑将基于贪心算法的近似求解思路作为对比实验，并提出多层次的决策机制来优化配送任务，以确保在有限资源下尽可能满足所有订单的交付要求，具体解决方案如下：

系统为应对动态变化的网络环境，同样在无人机网络中考虑使用切片式瞬时网络路由的方法。这种方法在卫星路由中非常常见且有效，通过将网络进行分片后，对瞬时的网络拓扑进行路由分析和规划，系统可以更灵活地适应网络环境的变化。

针对每个网络拓扑，先根据距离进行聚类，将订单点和配送中心进行配对后，再帮助每个配送中心进行决策。根据无人机每次最多可携带物品数量的限制，处理当前待处理的订单，而不只是新生成的订单。换言之，这些待处理的订单既包括新生成的订单，也包括累积的订单。这一策略可以在某些订单紧急程度较低或相同配送点位置的订单数量过多的情况下，优化配送过程，提高整体效率。此外，为了保证订单在规定的截止交付时间内完成配送避免被长时间搁置，系统会动态调整订单的优先级。总的来说，订单排序是先根据订单生成时的紧急程度进行初始排序，再根据订单是否较长时间未被成功配送来提升优先级，以提供实时有效的订单排序队列。按如上策略进行订单排序后再进行路径规划。本报告提供的算法使用多种启发式算法包括遗传算法、蚁群优化和粒子群优化算法，综合其优势，以尝试给出更为合理的路径规划。

## 1.4 报告结构

本报告将深入探究无人机配送路径规划问题。第二章将探讨解决该问题的技术选型思路，解释选择特定方案和算法的原因，并比较不同方法的优缺点。第三章将详细介绍算法设计，包括订单生成与分类、聚类算法、路径优化、无人机调度等。在第四章实验章节，则将描述实验的具体细节，包括新提出的混合路径规划优化算法和进行对比的基于贪心算法的配送方案，并对实验结果进行分析和评价。最后，会在总结章节中，对整个算法的性能、局限性以及未来的改进方向进行探讨，并进行展望和拓展。

# 2 技术选型

为了更好地解决该无人机配送路径规划问题，实验选择了多种先进技术和优化方法。这些技术在不同的方面提供支撑，以使得整体算法更加高效和可靠。

## 2.1 DBSCAN 聚类算法

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是一个比较有代表性的基于密度的聚类算法。相比于划分和层次聚类方法，该种方法将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇。簇的形成需要两个参数，一个是扫描半径，另一个是最小包含点数。具体原理是从任意一个未被访问的点开始，找出与其距离在扫描半径内的所有附近点。不断地用同样的算法去处理未被访问的点，直到所有的点都被访问。

相比于我们熟知的 K-means 聚类算法，DBSCAN 算法不需要预先指定簇的数量。此外，DBSCAN 还可以发现任意形状的簇。同时，该算法还能有效处理噪声点，分离密集区域和噪声点。两个算法的具体对比总结如下：

表 2-1　　DBSCAN 算法和 K-means 算法对比

| | DBSCAN | K-means |
|---|---|---|
| 簇形状 | 可以处理任意形状的簇 | 假设簇是球状的 |
| 噪声处理 | 可以识别并忽略噪声点 | 无法识别噪声，所有点都会被分配到某个簇 |
| 参数 | 需要选择扫描半径和最小包含点数 | 需要预先指定簇的数量 K |
| 稳定性 | 结果更稳定，不依赖随机初始化 | 结果依赖初始中心点，可能不稳定 |
| 应用场景 | 适用于不规则分布和含有噪声的数据 | 适用于簇为凸形的数据分布和聚类数量已知或可预测 |

因此，在无人机配送路径规划这样一个配送中心和订单位置可能不规则分布的场景中，DBSCAN 算法可能更加适用。此外，采用 DBSCAN 算法也能更好应对异常的配送订单位置等噪声值，从而得到更合理的聚类结果和路径规划方案。

## 2.2 路径优化算法

路径优化问题属于 NP-hard 问题，单一的优化算法可能难以在合理时间找到最优解。运用多种优化算法实现的混合算法可以结合多种算法优势，提高解的质量和搜索效率。混合优化算法主要结合了遗传算法（GA）、蚁群优化（ACO）和粒子群优化（PSO）等多种启发式和元启发式算法的特点，对每个配送区内的订单进行路径优化，来确保路径长度最短。

### 2.1.1 遗传算法

作为一种基于自然选择和遗传学原理的优化技术，遗传算法通过模拟生物进化过程中的选择、交叉和变异等机制，来寻找问题的最优解。首先生成一个初始种群，每个个体代表一个可能的解，然后通过适应度函数评估每个个体的优劣。适应度高的个体被选为父代，通过交叉和变异操作生成新一代的个体。这个过程会一直迭代重复，直到找到满意的解或达到预设的终止条件为止。

在这个无人机配送路径优化问题中，遗传算法可以有效地帮助找到最优的配送路径。首先，生成一组随机的配送路径，每条路径对应一个个体。然后，根据路径的总距离或者总耗时等指标，来计算每条路径的适应度值，适应值越低，即路径越短或耗时越少，表示该路径越优越。接下来，将根据适应度值选择较优的个体作为父代，通过交叉操作来生成新的路径个体。交叉之后，通过变异操作随机调整某些路径，以进一步增加解的多样性和避免局部最优。最终，收敛到的即为最优或最接近最优的配送路径。

### 2.1.2 蚁群优化

蚁群优化算法是一种仿生计算技术，通过参照动物界中蚂蚁在寻找食物时通过信息素来传递信息的行为，来解决这个优化问题。该算法模拟了蚂蚁在路径上留下信息素，并根据信息素浓度来指导后续蚂蚁选择路径，从而逐步找到全局最优解。蚂蚁在选择路径时，不仅依据路径上已有的信息素浓度，还会结合局部启发式信息（如距离或时间）进行决策。在每次蚂蚁完成路径后，会根据具体的路径优劣情况更新信息素浓度，使得更优的路径上信息素浓度增加，同时信息素也会随时间逐渐挥发。通过这一机制，蚁群优化算法能够有效地解决复杂的无人机配送路径在内的组合优化问题。因此，从原理上来看，利用蚁群优化算法来寻找最短配送路径，以提高配送效率并降低成本也是一个可行的解决方案。

### 2.1.3 粒子群优化

粒子群优化算法（Paritcle Swarm Optimization, PSO）是一种源自自然界群体行为的启发式优化技术。该算法模拟了粒子在搜索空间中的移动和相互作用，每个粒子代

表一个可能的解。通过不断地调整粒子的速度和位置，以及根据个人最佳位置（Pbest）和全局最佳位置（Gbest）的引导，PSO 算法能够逐步逼近问题的最优解。

在无人机配送路径优化问题中，粒子群优化算法能够帮助无人机有效规划最优路径，从而提高配送效率并降低成本。此外，PSO 算法的应用不仅限于单一目标的优化，还可以考虑多个因素的综合优化，如最短路径、最小能耗、最小时间等。通过调整适应度函数，可以平衡不同目标之间的权衡关系，使得无人机在配送过程中既能够尽快到目的地，又能够节省能源和时间成本。因此，粒子群优化算法在无人机配送路径优化中具有灵活性和多样性，能够适应不同的配送需求和环境条件。

2.1.4 混合路径优化算法

混合路径优化算法融合了以上三种算法，一定程度上克服了单一算法在解决无人机配送路径优化问题中的可能存在的局限性。新的路径优化算法通过执行三种优化算法后，在最终选择配送路径时，通过比较它们的结果，选择总距离最短的路径作为最终的配送方案。这一混合路径优化算法结合了三种算法，以有效利用它们各自的优点，弥补彼此的缺点，从而更有效地解决无人配送路径优化问题。

1> 遗传算法具有广泛的适用性，能够有效地探索大范围的解空间，并且易于实现。它通过种群的进化，不断优化个体，从而寻找到问题的全局最优解。然而，在一些特殊情况下如搜索空间非常复杂等情况下，遗传算法可能会陷入局部最优解。

2> 蚁群优化算法模拟了蚂蚁在寻找食物过程中的行为，利用信息素和启发式信息引导蚂蚁的移动，有助于发现全局最优解。蚁群算法具有较好的鲁棒性和并行性，并且能够自适应地搜索解空间。但是，蚁群算法可能受到启发式信息的局限，导致在复杂问题中收敛速度较慢。

3> 粒子群优化算法的设计思想基于群体智能，通过模拟粒子的速度和位置调整，以及个体和全局最佳位置的引导，来搜索最优解。该种算法具有简单易实现、收敛速度快的特点，适用于高维优化问题。但是，粒子群算法容易陷入局部最优解，并且对参数设置敏感。

综合利用不同算法的优点，有利于提高搜索效率和全局搜索能力。通过这种方式提高算法的鲁棒性和适应性，特别有助于帮助跳出局部最优解，有利于增加解空间的探索范围。

# 3 算法设计

## 3.1 订单生成与聚类

这一部分主要完成了订单的生成和分类工作。首先，定义了三个配送中心位置和名称，以及九个订单的位置和优先级。然后使用 DBSCAN 聚类算法对订单进行聚类，将相近的订单分配给同一配送中心，并返回了每个配送中心对应的订单列表。这一部分代码的主要目的是为了实现订单的分配，以便后续的路径规划和无人机调度。

```python
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from scipy.spatial.distance import cdist
from deap import base, creator, tools, algorithms

# 配送中心位置和名称
delivery_center_locations = np.array([[44, 47], [0, 3], [3, 39]])
delivery_center_names = ['Center 1', 'Center 2', 'Center 3']

# 订单位置和优先级
orders = [((9, 19), 'Very Urgent'), ((21, 36), 'Very Urgent'), ((21, 36), 'Urgent'),
          ((9, 19), 'Urgent'), ((9, 19), 'Urgent'), ((9, 19), 'Urgent'), ((24, 24), 'Urgent'),
          ((9, 19), 'General'), ((21, 36), 'General'), ((23, 6), 'General'), ((37, 48), 'Urgent'),
          ((36, 37), 'General'), ((39, 43), 'Very Urgent'), ((51, 52), 'General'), ((44, 38), 'Urgent')]

# DBSCAN聚类算法进行配送中心和订单的聚类
def cluster_orders(delivery_center_locations, orders, eps=20):
    print("\n正在进行路径规划...")
    order_locations = np.array([order[0] for order in orders])
    order_priorities = np.array([order[1] for order in orders])
    order_features = np.column_stack((order_locations, order_priorities))
    dbscan = DBSCAN(eps=eps, min_samples=1)
    order_labels = dbscan.fit_predict(order_features)  # 在聚类时考虑优先级信息
    center_orders = {i: [] for i in range(len(delivery_center_locations))}

    for label in set(order_labels):
        cluster_orders = order_features[order_labels == label]
        center_distances = cdist(cluster_orders[:, :-1], delivery_center_locations)  # 只考虑位置信息计算距离
        closest_centers = np.argmin(center_distances, axis=1)

        for order_idx, center_idx in enumerate(closest_centers):
            center_orders[center_idx].append(orders[np.where(order_labels == label)[0][order_idx]])

    return center_orders
```

图 3-1 配送中心及订单聚类代码

## 3.2 路径优化

目标函数`total_distance`用于计算无人机总配送距离。该函数接受一个个体的输入，个人表示订单分配给各个无人机的情况。根据紧急程度对任务进行排序后分批任务，保证无人机一次最多携带两个物品。最后，将每批订单的配送距离累加得到总的无人机

配送任务距离。该方法的主要作用是为了评估每个配送方案的效果，为后续的混合优化算法选用具体算法种类提供评价指标。

```python
def total_distance(individual):
    # 初始化无人机的配送任务
    num_drones = len(delivery_center_locations)
    drones = [[] for _ in range(num_drones)]
    # 分配订单给无人机
    for order_idx, center_idx in enumerate(individual):
        drones[center_idx].append(order_idx)
    total_distance = 0
    for drone_idx, drone_tasks in enumerate(drones):
        if drone_tasks:
            current_location = delivery_center_locations[drone_idx] # 无人机从配送中心出发
            total_drone_distance = 0
            sorted_tasks = sorted(drone_tasks, key=lambda x: {'Very Urgent': 0, 'Urgent': 1, 'General': 2}[orders[x][1]])
            batches = [sorted_tasks[i:i + 2] for i in range(0, len(sorted_tasks), 2)] # 任务分批，每次最多2个订单
            for batch in batches:
                for order_idx in batch:
                    order_location = orders[order_idx][0]
                    order_priority = orders[order_idx][1]
                    distance_to_order = np.linalg.norm(current_location - np.array(order_location))
                    total_drone_distance += distance_to_order
                    current_location = order_location
                    # 根据订单的紧急程度调整飞行距离的惩罚
                    if order_priority == 'Very Urgent':
                        penalty_factor = 3  # 非常紧急订单的惩罚系数
                    elif order_priority == 'Urgent':
                        penalty_factor = 2  # 紧急订单的惩罚系数
                    else: # order_priority == 'General'
                        penalty_factor = 1  # 普通订单的惩罚系数
                    if distance_to_order > 20:
                        penalty = (distance_to_order - 20) * penalty_factor  # 超出部分按惩罚系数进行惩罚
                        total_drone_distance += penalty
                distance_back_to_center = np.linalg.norm(current_location - delivery_center_locations[drone_idx])
                total_drone_distance += distance_back_to_center
                current_location = delivery_center_locations[drone_idx] # 返回配送中心后更新当前位置为配送中心
            total_distance += total_drone_distance
    return total_distance,
```

图 3-2 方案总配送距离计算代码

## 3.3 无人机调度

### 3.3.1 遗传算法设计与实现

遗传算法通过模拟自然选择和遗传机制来搜索问题的最优解。在每一代中，通过交叉操作来产生新的个体，然后通过变异操作对个体进行随机改变，最后通过选择操作保留适应度较高的个体，从而逐步优化整个种群。在无人机路径配送问题中，个体表示配送方案，适应度函数评估方案的优劣，过程中通过交叉、变异和选择等操作逐步优化配送方案，找到最优解。

在具体算法实现中，使用 DEAP 库提供的工具箱实现。首先，使用`creator`创建适应度类和个体类，然后创建了工具箱，注册了个体的初始化方法、适应度评价函数、交叉、变异和选择等操作。接着使用`algorithms.eaSimple()`函数来执行遗传算法的主要迭代过程。

```python
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
toolbox = base.Toolbox()
toolbox.register("individual", tools.initRepeat, creator.Individual, lambda: random.randint(0, len(delivery_center_locations) - 1), n=len(orders))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", total_distance)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low=0, up=len(delivery_center_locations) - 1, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 遗传算法
def genetic_algorithm():
    pop = toolbox.population(n=100)
    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("min", np.min)
    algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=50, stats=stats, halloffame=hof, verbose=True)
    best_individual = hof[0]
    return best_individual
```

图 3-3 遗传算法实现代码

### 3.3.2 蚁群优化算法设计与实现

蚁群优化算法也用于寻找一组无人机的最优配送策略，以最小化总配送距离即总成本。该算法通过不断迭代，更新信息素浓度和蚂蚁的移动策略，使得蚂蚁倾向于选择路径上信息素浓度高的地方，从而使得整体搜索更加集中在有利的区域，达到优化配送路线的目的。在具体代码中，蚁群优化算法被实现为`ant_colony_optimazation`函数。首先，初始化一些包括蚂蚁数量、迭代次数、信息素挥发率等在内的参数。然后多次迭代，每只蚂蚁根据信息素浓度和距离等因素选择路径，并更新信息素浓度。结束后，根据蚂蚁的选择结果和距离计算出最优的路径。最终，算法返回最优路径。

```python
# 蚁群优化算法
def ant_colony_optimization():
    # 实现ACO算法
    num_orders = len(orders)
    num_ants = 10
    num_iterations = 100
    evaporation_rate = 0.5
    pheromone = np.ones((num_orders, num_orders))  # 信息素矩阵
    best_individual = [0] * num_orders
    best_distance = float('inf')
    for _ in range(num_iterations):
        all_routes = []
        all_distances = []
        for _ in range(num_ants):
            route = list(np.random.permutation(num_orders))
            distance = sum([np.linalg.norm(np.array(orders[route[i]][0]) - np.array(orders[route[(i + 1) % num_orders]][0])) for i in range(num_orders)])
            all_routes.append(route)
            all_distances.append(distance)
            if distance < best_distance:
                best_distance = distance
                best_individual = route
        # 更新信息素
        pheromone *= (1 - evaporation_rate)
        for route, distance in zip(all_routes, all_distances):
            for i in range(num_orders):
                pheromone[route[i]][route[(i + 1) % num_orders]] += 1.0 / distance
    # 保证返回的个体在0到2之间
    best_individual = [i % len(delivery_center_locations) for i in best_individual]
    return best_individual
```

图 3-4 蚁群优化算法实现代码

### 3.3.3 粒子群优化算法设计与实现

作为一种基于群体智能的优化算法，粒子群优化算法（Particle Swarm Optimization,PSO）也可用来参与最小化总配送距离或其他指标的方案设计。通过个体经验和群体经验不断更新自身位置和速度，从而逐步找到最优解。具体实现中，每个粒

子代表一个订单分配方案，其位置表示无人机的路径顺序，速度则用来调整路径的搜索方向。通过计算每个粒子的适应度（本问题中则为总配送距离），并不断调整粒子的位置和速度，直至达到停止条件或最大迭代次数。最终，算法将收敛于一个最优的无人机路径方案，从而实现了无人机路径配送问题的优化。

在具体代码实现过程中，PSO 算法通过模拟粒子群体相互协作和竞争来寻找最优路径。首先初始化了粒子群，每个粒子表示一个订单路径的排列。然后在多次迭代中，每个粒子的路径总配送距离（适应度）被计算出来，并与其历史最优路径和群体的全局最优路径进行比较和更新。粒子的速度和位置根据惯性项、认知项（粒子自身历史最优位置）以及社会项（群体历史最优位置）进行调整，以引导粒子逐步逼近最优解。最终，返回适应度最高的粒子路径，作为 PSO 算法给出的无人机最优配送路线。

```python
# 粒子群优化算法
def particle_swarm_optimization():
    # 实现PSO算法
    num_particles = 30
    num_iterations = 100
    inertia = 0.5
    cognitive = 1.0
    social = 2.0
    best_individual = [0] * len(orders)
    best_distance = float('inf')

    particles = [list(np.random.permutation(len(orders))) for _ in range(num_particles)]
    velocities = [np.random.rand(len(orders)) for _ in range(num_particles)]
    personal_best_positions = particles[:]
    personal_best_distances = [float('inf')] * num_particles

    for _ in range(num_iterations):
        for i, particle in enumerate(particles):
            distance = sum([np.linalg.norm(np.array(orders[particle[j]][0]) - np.array(orders[particle[(j + 1) % len(orders)]][0])) for j in range(len(orders))])
            if distance < personal_best_distances[i]:
                personal_best_distances[i] = distance
                personal_best_positions[i] = particle[:]
            if distance < best_distance:
                best_distance = distance
                best_individual = particle[:]

        for i, particle in enumerate(particles):
            for j in range(len(orders)):
                velocities[i][j] = (inertia * velocities[i][j] +
                                    cognitive * random.random() * (personal_best_positions[i][j] - particle[j]) +
                                    social * random.random() * (best_individual[j] - particle[j]))
                particle[j] = int(particle[j] + velocities[i][j]) % len(orders)

    # 保证返回的个体在0到2之间
    best_individual = [i % len(delivery_center_locations) for i in best_individual]
    return best_individual
```

图 3-5 粒子群优化算法实现代码

### 3.3.4 混合路径优化算法设计与实现

混合算法结合了三种算法的优势来优化无人机路径配送问题。首先，通过遗传算法生成一个初始种群，每个个体代表订单路径的一个排列，经过选择、交叉和变异操作，迭代优化得到最佳个体；然后，使用蚁群优化算法，通过模拟蚂蚁在路径上留下的信息素来选择和优化路径，经过多次迭代更新信息素矩阵，找到最优路径；最后，粒子群优化算法通过模拟粒子在搜索空间中的飞行，利用粒子间的相互合作和个体的历史最佳位置来调整和优化路径。

三种算法独立运行后，混合算法综合考虑各自的最优解，选择总距离最短的路径作为最终结果。这样，混合路径优化算法一定程度吸纳了三种算法的优势，提高了路径优化的效果和效率。

# 4 实验效果

## 4.1 实验结果

使用 DBSCAN 聚类算法对订单进行聚类，具体的聚类效果如下图所示。可以看见，聚类的结果将订单分成了几个组，每个组内的订单距离彼此比较近，而不同组间的订单距离较远。通过如下方式将订单分配给不同的配送中心，可以更有效地进行配送。

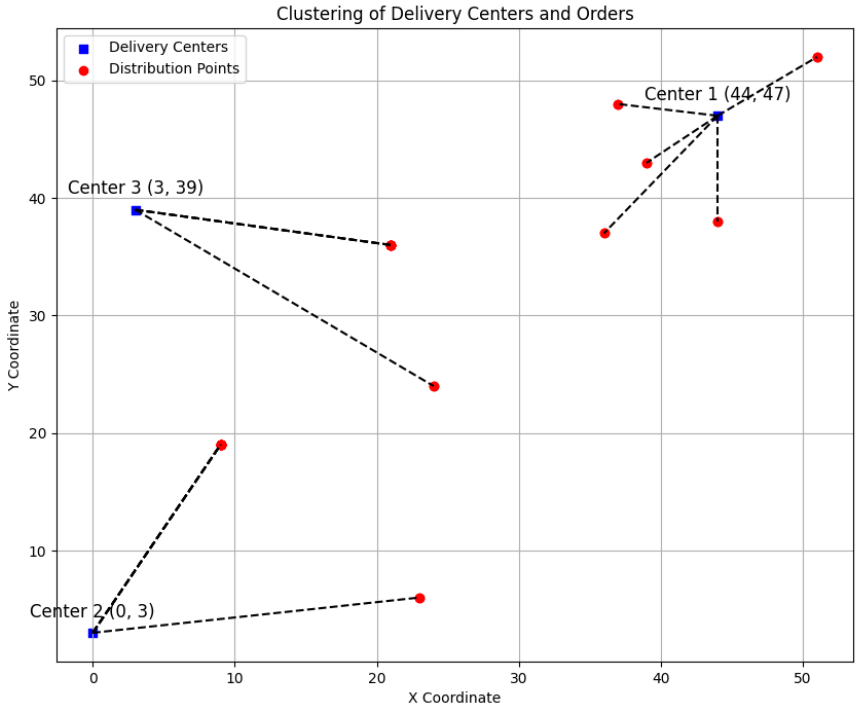


图 4-1 配送中心和订单的聚类效果

在最终的配送路径中，每架无人机负责执行一条路径，根据订单的位置和优先级进行分配。在配送路径规划过程中，紧急程度较高的订单会被优先考虑。如图所示，每架无人机一次配送过程中最多携带的物品数不超过 2 个，且完成配送后返回到配送中心，符合实验假设，实验结果满足预期。

```
最终的配送路径:
无人机 1 的路径:
((44, 47), 'Depart from Center') -> ((37, 48), 'Urgent') -> ((36, 37), 'General') -> ((44, 47), 'Return to Center')
无人机 2 的路径:
((44, 47), 'Depart from Center') -> ((39, 43), 'Very Urgent') -> ((51, 52), 'General') -> ((44, 47), 'Return to Center')
无人机 3 的路径:
((44, 47), 'Depart from Center') -> ((44, 38), 'Urgent') -> ((44, 47), 'Return to Center')
无人机 4 的路径:
((0, 3), 'Depart from Center') -> ((9, 19), 'Very Urgent') -> ((9, 19), 'Urgent') -> ((0, 3), 'Return to Center')
无人机 5 的路径:
((0, 3), 'Depart from Center') -> ((9, 19), 'Urgent') -> ((9, 19), 'Urgent') -> ((0, 3), 'Return to Center')
无人机 6 的路径:
((0, 3), 'Depart from Center') -> ((9, 19), 'General') -> ((23, 6), 'General') -> ((0, 3), 'Return to Center')
无人机 7 的路径:
((3, 39), 'Depart from Center') -> ((21, 36), 'Very Urgent') -> ((21, 36), 'Urgent') -> ((3, 39), 'Return to Center')
无人机 8 的路径:
((3, 39), 'Depart from Center') -> ((24, 24), 'Urgent') -> ((21, 36), 'General') -> ((3, 39), 'Return to Center')
```

图 4-2 混合路径优化算法-无人机最终配送路径

下图展示具体的无人机配送路径。当无人机一次性最多能携带的物品数量能够覆盖附近的配送任务且较快完成配送，无人机将会在一次配送过程中完成所有配送。若附近订单数多于单次所能携带的物品数，则会分多批进行配送，如用路径 4、5、6 来处理配送点（9，19）产生的 5 个订单。其中通过先配送具有较高优先级的物品来保证订单优先级别方面的要求。另外，尽管每个配送中心的无人机数量无限，但配送中心在保证较少的总配送时间的前提下，仍会尽可能减少完成所有配送任务的无人机数量。

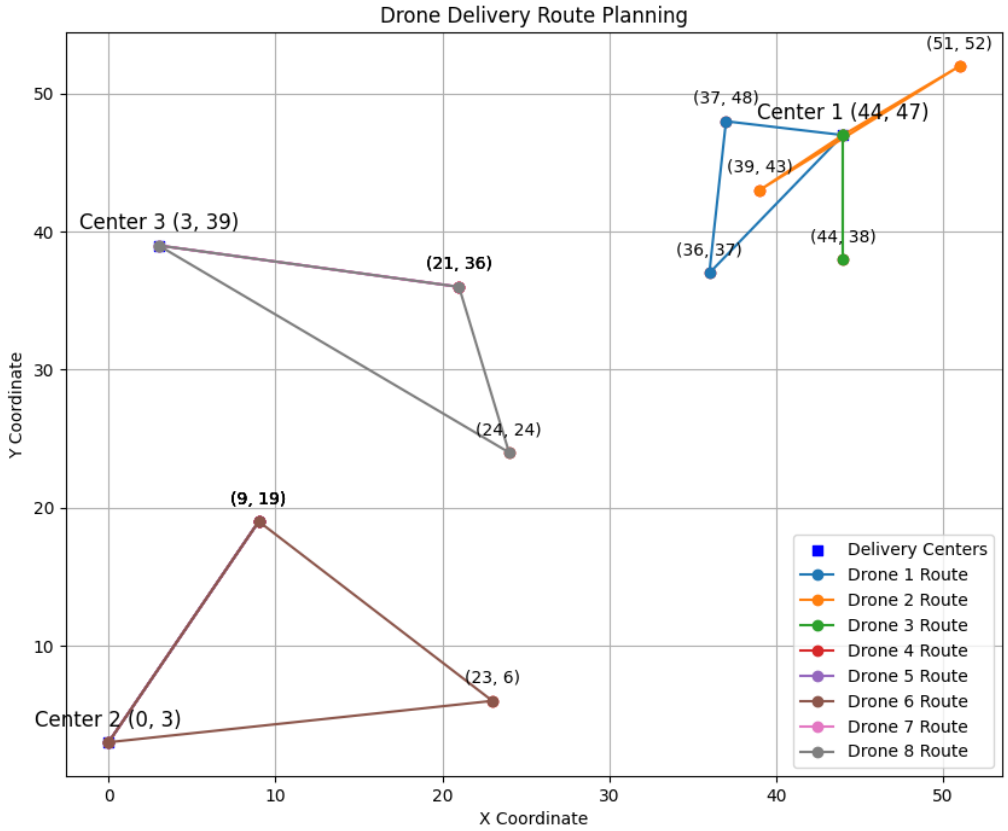


图 4-3 混合路径优化算法-无人机配送路径二维展示

## 4.2 对比实验

```
最终的配送路径:
无人机 1 的路径:
(44, 47) (Depart from Center) -> (39, 43) (Very Urgent) -> (37, 48) (Urgent) -> (44, 47) (Return to Center)
无人机 2 的路径:
(44, 47) (Depart from Center) -> (44, 38) (Urgent) -> (36, 37) (General) -> (44, 47) (Return to Center)
无人机 3 的路径:
(44, 47) (Depart from Center) -> (51, 52) (General) -> (44, 47) (Return to Center)
无人机 4 的路径:
(0, 3) (Depart from Center) -> (9, 19) (Very Urgent) -> (9, 19) (Urgent) -> (0, 3) (Return to Center)
无人机 5 的路径:
(0, 3) (Depart from Center) -> (9, 19) (Urgent) -> (9, 19) (Urgent) -> (0, 3) (Return to Center)
无人机 6 的路径:
(0, 3) (Depart from Center) -> (9, 19) (General) -> (23, 6) (General) -> (0, 3) (Return to Center)
无人机 7 的路径:
(3, 39) (Depart from Center) -> (21, 36) (Very Urgent) -> (21, 36) (Urgent) -> (3, 39) (Return to Center)
无人机 8 的路径:
(3, 39) (Depart from Center) -> (21, 36) (General) -> (24, 24) (Urgent) -> (3, 39) (Return to Center)
```

图 4-4 基于贪心策略的优化算法-无人机最终配送路径

对比的路径规划算法是一种基于贪心策略的方法。首先，同样使用DBSCAN算法将订单按位置聚类分配给最近的配送中心。然后，对于每个配送中心的订单，无人机会先按照优先级排序后选择距离当前无人机位置最近的订单进行配送，直到无人机容量耗尽或所有订单处理完毕。每次完成订单配送后，无人机也会返回配送中心。

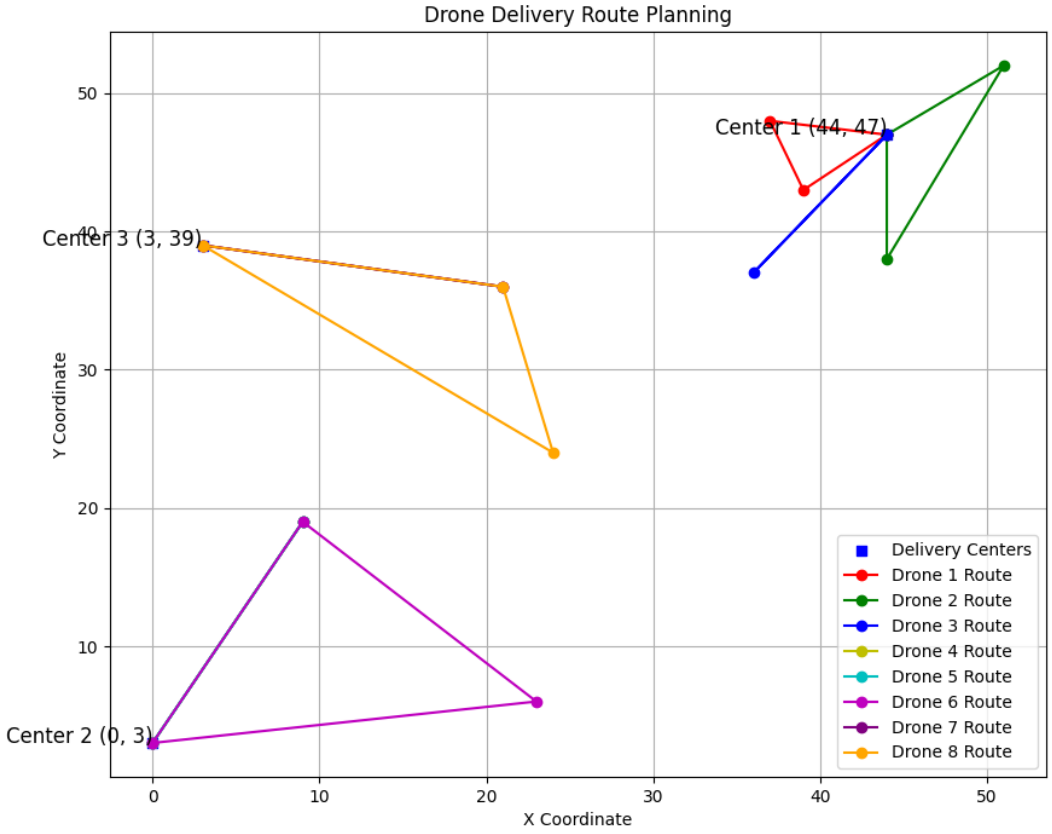


图 4-5 基于贪心策略的优化算法-无人机配送路径二维展示

基于贪心策略的路径规划算法在每一步都选择当前最优解，即最接近并且具有最高优先级的订单。这种方法由于严格按照优先级和距离选择，有时路径规划较僵化，不能灵活处理实际情况。总的来说，虽然基于贪心策略的路径规划算法更为简单直观，容易实现和理解，且对于单个订单的处理速度很快，但由于其局部最优性，可能会导致资源利用不够充分，系统整体的效率不高。

相比之下，混合路径优化算法更注重整体路径的优化和资源的合理利用。它会考虑多个订单之间的空间分布和优先级，在降低最小化总体路径长度和时间成本的同时，减少整体系统的资源浪费。例如，无人机 1 和无人机 2 的路径相比于基于贪心策略的路径规划算法更为合理，减少了空载和不必要的返回。贪心策略主要按照优先级排序后选择最近的订单。优先级较低但距离较近的订单可能被优先处理，从而导致高优先级订单被延迟。但混合路径优化算法综合考虑了订单的优先级和配送距离，更好地平衡优先级和距离，提高整体配送效率。换言之，混合路径优化算法是站在全局视角下的路径优化，可以减少无人机的空载飞行时间，有效提高资源利用效率和系统的整体性能。

# 5 总结与展望

针对无人机配送路径规划问题，本文提出一种自适应混合路径优化算法，该算法综合了多种启发式算法的优势，不受单一算法的局限性。通过结合遗传算法、蚁群优化和粒子群优化等多种算法，该算法能够在确认最终路径前进行比较，从而更有可能找到全局最优或接近最优的路径方案。综合利用各自的优点，如全局搜索能力、局部搜索能力和收敛速度等，从而避免陷入局部最优解。此外，算法还利用 DBSCAN 等聚类算法对订单进行聚类，进一步优化了路径规划过程。实验结果表明，与基于贪心策略的算法相比，混合路径优化算法更有可能找到全局最优或接近最优的路径方案。不单基于某一启发式算法，混合路径优化算法具有更强的鲁棒性和适应性，能够更好地适应不同场景下的路径规划需求，表现出很好的稳定性。

混合路径优化算法在解决包括无人机路径规划在内的物流配送问题方面表现良好，但仍具有一定的局限性。例如，在处理大规模数据时，算法的计算复杂度可能会增加，导致运行时间较长。此外，算法的参数设置对结果影响较大，需要进行提前的精确调整。但随着对相关算法的进一步研究和改进，可以期待其在实际应用中发挥更加重要和广泛的作用。