武汉大学国家网络安全学院

课程报告

课程名称 高级算法设计与分析

专业年级 网络空间安全 2023 级

姓　　名 王亚龙

学　　号 2023202210116

填写时间 2024 年 06 月 24 日

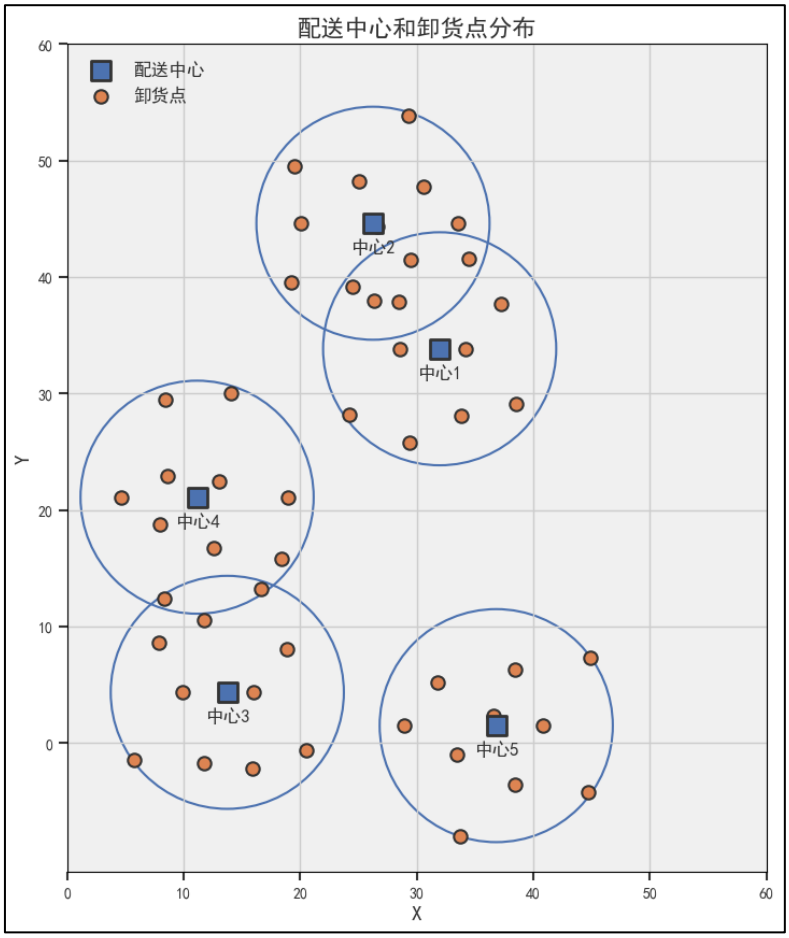
一、实验概述

1.1 问题描述

在现代物流配送中，最后一公里配送是提升效率和客户满意度的关键环节。无人机配送技术被认为是解决这一问题的有效手段，能够快速高效地完成配送任务。研究的主要问题是如何在一个特定区域内，利用多个配送中心的无人机完成卸货点的订单配送，并优化配送路径以最小化总配送距离，同时满足订单的优先级要求。

1.2 区域地图和基本描述

配送区域由多个配送中心和卸货点组成。配送中心用于存储商品和调度无人机，每个配送中心的位置固定，且拥有充足的商品库存和无限的无人机供应。卸货点是无人机配送的终点，每个卸货点会随机生成订单。假设区域内共有j个配送中心和k个卸货点。在具体实验时，设置配送中心数为 5，每个区域有 10 个卸货点，配送中心和卸货点的分布如下图所示：



1.3 订单生成和优先级描述

订单由卸货点生成，每个订单包含一个商品，并分为以下三个优先级别：

- 一般：要求在 3 小时内配送完成；
- 较紧急：要求在 1.5 小时内配送完成；
- 紧急：要求在 0.5 小时内配送完成。

每个订单生成后，有一个生成时间和截止时间，无人机需要在截止时间前完成订单配送。在具体实现时假设：每隔 30 分钟，所有卸货点会随机生成 0-5 个订单。

1.4 系统决策时间间隔和优化目标

为简化问题，时间被离散化为固定的时间间隔t分钟。每个时间间隔，系统将执行一次调度决策，包括以下内容：

1. **无人机调度：**决定哪些配送中心出动多少无人机完成哪些订单；
2. **路径规划：**为每个无人机规划配送路径，即先完成哪个订单，再完成哪个订单，最后返回原配送中心。

优化目标：在保证所有订单在截止时间前送达的前提下，使得无人机的总配送路径最短。

1.5 约束条件

- 无人机负载能力：每次飞行最多携带n个物品，在具体实验中设置为 5；
- 最大飞行距离：每次飞行的总距离不超过 20 公里（包含返回配送中心的距离）；
- 飞行速度：无人机的飞行速度为 60 公里/小时；
- 配送中心能力：每个配送中心的无人机数量和商品供应无限；
- 优先级要求：必须满足订单的优先级别要求，在截止时间前完成配送。

系统在每个时间间隔内做出调度决策，可以选择不对当前的某些订单进行配送，累积后与后面的订单一起配送，以实现路径最短的目标。

二、算法设计

路径规划是无人机配送过程中最关键的部分，目的是在最短路径的基础上，满足所有订单的优先级和截止时间要求。为实现这一目标，设计了一种基于贪心算法的路径规划方法。该方法通过动态选择和调整路径，实现无人机在有限的飞行距离和载重条件下的最优配送。

2.1 订单的生成和更新

1) 订单生成：
   a) 每隔 30 分钟，每个卸货点随机生成$0-5$个订单。订单具有三个优先级：一般（3 小时内配送）、较紧急（1.5 小时内配送）、紧急（0.5 小时内配送）。
   b) 每个订单记录生成时间和截止时间。
2) 订单更新：
   a) 系统每隔30分钟更新一次，包含新的订单生成和当前未完成订单的更新。
   b) 根据当前时间与订单截止时间将未完成订单列表分为立即配送订单列表和缓冲订单列表。
      i. 立即配送订单列表在决策时需要立即配送，确保所有订单在截止时间前完成配送。
      ii. 缓冲订单列表，用于累积当前时刻未配送且不需要立即配送的订单，但是在配送立即配送订单时为了使得最大化利用无人机的载重，决策时也会对缓冲订单列表进行考虑。其中没有配送的缓冲订单列表中的订单会在下一轮调度中与新生成的订单一起进行排序和选择。通过引入缓冲订单列表，系统可以更好地优化路径，减少总配送距离。

2.2 贪心算法路径规划

1) 初始设置：
   a) 为每个配送中心确定其需要配送的订单，具体为其覆盖范围内与其距离最近的卸货点的所有订单。
   b) 初始化路径为空，并将当前配送中心的订单列表复制为未完成订单列表，

表示当前未完成的订单。设置当前无人机的位置为配送中心的位置，并初始化无人机的当前载重、已飞行距离和已使用时间。

2) 订单排序和路径选择

a) 每隔 30 分钟进行订单更新与分类，系统对立即配送订单列表进行排序，按照优先级和与当前无人机位置的距离进行排列。优先选择最紧急且最近的订单。缓冲订单列表的引入，使得系统可以选择不对当前的某些订单进行配送，累积后与后面的订单一起配送，更好的优化路径减少总配送距离。

b) 在选择派送的订单时还需要满足以下条件才可以配送：

i. 无人机在配送该订单后返回配送中心的总距离不超过最大飞行距离。

ii. 无人机在配送该订单的过程中能够在订单的截止时间前完成配送。

3) 更新路径和订单信息：

a) 选择符合条件的订单，将该订单对应的卸货点添加到路径中。

b) 更新无人机的当前载重、已飞行距离和已使用时间。

c) 从未完成的订单列表中移除已配送的订单。

d) 更新无人机的当前位置。

4) 路径终止条件：

a) 当未完成的订单列表为空或者无人机的载重达到最大载重时，停止选择新的订单。

b) 无人机无法再配送其他订单，需要返回配送中心。

三、实验步骤

3.1 数据生成

1. 配送中心和卸货点的初始化：

- 创建 5 个配送中心，记录其位置。
- 每个配送中心创建 10 个卸货点，随机生成其位置。

2. 订单生成：

- 每隔 30 分钟为每个卸货点随机生成 0-5 个订单，订单包含优先级、生成时间和截止时间。生成的订单样例如下：

```
卸货点 4 的订单:
  - 优先级: 较紧急, 生成时间: 0.00时, 截至时间: 1.50时
  - 优先级: 较紧急, 生成时间: 0.00时, 截至时间: 1.50时
  - 优先级: 一般, 生成时间: 1.50时, 截至时间: 4.50时
  - 优先级: 一般, 生成时间: 1.50时, 截至时间: 4.50时
  - 优先级: 紧急, 生成时间: 1.50时, 截至时间: 2.00时
  - 优先级: 紧急, 生成时间: 2.00时, 截至时间: 2.50时
  - 优先级: 较紧急, 生成时间: 2.50时, 截至时间: 4.00时
  - 优先级: 紧急, 生成时间: 2.50时, 截至时间: 3.00时
  - 优先级: 一般, 生成时间: 3.00时, 截至时间: 6.00时
  - 优先级: 一般, 生成时间: 3.00时, 截至时间: 6.00时
  - 优先级: 紧急, 生成时间: 3.00时, 截至时间: 3.50时
  - 优先级: 紧急, 生成时间: 3.50时, 截至时间: 4.00时
```

3.2 配送路径规划算法设计与实现

1. 基于贪心的路径规划：

- 计算无人机的配送路径，并确保其在最大负载和最大距离约束内完成尽可能多的订单。

```python
# 路径规划（贪婪算法）
def plan_path(center, orders, center_index, current_time, max_distance=20, max_items=5):
    path = []
    path_ids = [f"配送中心{center_index+1}"]
    remaining_orders = [order for order in orders if order.timestamp <= current_time]
    current_location = center.location
    current_load = 0
    current_distance = 0
    total_time = current_time
    # 记录开始时间
    start_time = total_time
    while remaining_orders and current_load < max_items:
        # 按截止时间和距离排序
        remaining_orders.sort(key=lambda o: (o.deadline, calculate_distance(current_location, o.unloading_point.location)))
        for order in remaining_orders:
            next_distance = calculate_distance(current_location, order.unloading_point.location)
            return_distance = calculate_distance(order.unloading_point.location, center.location)
            total_distance = current_distance + next_distance + return_distance
            if total_distance <= max_distance and (total_time + next_distance / 60) <= order.deadline:
                path.append(order)
                path_ids.append(f"卸货点{order.unloading_point.index + 1}")
                current_location = order.unloading_point.location
                current_distance += next_distance
                current_load += 1
                total_time += next_distance / 60  # 时间增加，单位小时
                remaining_orders.remove(order)
                break
        else:
            break
    # 回到配送中心
    if path:
        current_distance += calculate_distance(current_location, center.location)
    path_ids.append(f"配送中心{center_index+1}")
    # 记录总配送距离
    total_distance = current_distance
    return path, path_ids, start_time, total_distance  # 返回路径、路径编号、开始时间和总配送距离
```

2. 调度策略：

- 实现无人机调度策略，决定每个配送中心出动多少无人机，每个无人机的路径，记录路径的开始时间和总配送距离。

```python
# 调度策略
def dispatch_drones(delivery_centers, unloading_points, max_distance=20, max_items=5, buffer_time=0.5):
    total_orders = 0
    current_time = 0
    interval = 0.5
    # 记录已处理订单的集合
    processed_orders = set()
    while current_time < 24:
        for center_index, center in enumerate(delivery_centers):
            # 获取该中心覆盖范围内的新订单
            new_orders = [order for point in unloading_points for order in point.orders
                          if calculate_distance(center.location, point.location) <= radius
                          and order.timestamp <= current_time
                          and order not in processed_orders]
            # 更新待处理订单
            center.pending_orders.extend(new_orders)
            processed_orders.update(new_orders)
            # 临时缓冲区订单
            buffer_orders = [order for order in center.pending_orders if order.deadline - current_time > buffer_time]
            immediate_orders = [order for order in center.pending_orders if order not in buffer_orders]
            while immediate_orders:
                path, path_ids, start_time, total_distance, order_count = plan_path(center, immediate_orders + buffer_orders, center_index, current_time, max_distance, max_items)
                if not path:
                    break
                center.drones.append((path, path_ids, start_time, total_distance))
                for order in path:
                    if order in immediate_orders:
                        immediate_orders.remove(order)
                    elif order in buffer_orders:
                        buffer_orders.remove(order)
                    center.pending_orders.remove(order)
                center.total_distance += total_distance  # 累积总配送距离
                total_orders += order_count  # 累积总订单数量
            # 将剩余订单累积到下一个时间段处理
            center.pending_orders = buffer_orders
        current_time += interval
```

下图就是一条配送记录，包含配送路径、配送开始时间与总配送距离，以及配送的订单对应的信息：优先级、卸货点、生成时间和截至时间等信息。

```
无人机 69: 配送中心5 -> 卸货点47 -> 卸货点46 -> 配送中心5
  - 开始时间: 7.50小时, 总距离: 17.29公里
  - 优先级: 紧急, 卸货点: 47, 生成时间: 7.50小时, 截止时间: 8.00小时
  - 优先级: 紧急, 卸货点: 47, 生成时间: 7.50小时, 截止时间: 8.00小时
  - 优先级: 一般, 卸货点: 46, 生成时间: 5.00小时, 截止时间: 8.00小时
  - 优先级: 一般, 卸货点: 46, 生成时间: 5.00小时, 截止时间: 8.00小时
  - 优先级: 较紧急, 卸货点: 46, 生成时间: 6.50小时, 截止时间: 8.00小时
```

3.3 路径规划和调度算法执行

1. 模拟 8 小时内配送中心的无人机调度和路径规划。
2. 输出每个配送中心的无人机配送路径和总配送距离。

四、实验结果与分析

4.1 订单生成情况

实验生成了多个卸货点，每个卸货点随机生成订单，订单包含优先级、生成时间和截止时间。以下是部分卸货点的订单生成情况示例：

```
卸货点 1 的订单:
  - 优先级: 一般, 生成时间: 1.00时, 截至时间: 4.00时
  - 优先级: 较紧急, 生成时间: 1.50时, 截至时间: 3.00时
  - 优先级: 较紧急, 生成时间: 1.50时, 截至时间: 3.00时
  - 优先级: 一般, 生成时间: 2.00时, 截至时间: 5.00时
  - 优先级: 紧急, 生成时间: 2.00时, 截至时间: 2.50时
  - 优先级: 较紧急, 生成时间: 2.00时, 截至时间: 3.50时
  - 优先级: 一般, 生成时间: 2.50时, 截至时间: 5.50时
  - 优先级: 一般, 生成时间: 2.50时, 截至时间: 5.50时
  - 优先级: 一般, 生成时间: 3.00时, 截至时间: 6.00时
  - 优先级: 较紧急, 生成时间: 3.00时, 截至时间: 4.50时
  - 优先级: 一般, 生成时间: 3.00时, 截至时间: 6.00时
卸货点 2 的订单:
  - 优先级: 紧急, 生成时间: 0.50时, 截至时间: 1.00时
  - 优先级: 一般, 生成时间: 0.50时, 截至时间: 3.50时
  - 优先级: 一般, 生成时间: 1.00时, 截至时间: 4.00时
  - 优先级: 较紧急, 生成时间: 1.00时, 截至时间: 2.50时
  - 优先级: 较紧急, 生成时间: 2.00时, 截至时间: 3.50时
  - 优先级: 一般, 生成时间: 2.00时, 截至时间: 5.00时
  - 优先级: 较紧急, 生成时间: 2.50时, 截至时间: 4.00时
  - 优先级: 较紧急, 生成时间: 2.50时, 截至时间: 4.00时
  - 优先级: 较紧急, 生成时间: 3.00时, 截至时间: 4.50时
```

这些订单在实验中分别具有不同的优先级和截止时间，反映了现实中不同客户对配送时间的不同要求。

4.2 路径规划情况

根据实验设置，无人机从各配送中心出发，完成指定的订单配送。以下是部分配送中心的路径规划示例：

i. 存在某些极端情况：

a) 卸货点距离配送中心过远，只能到一个卸货点后就返回，针对这些情况算法会考虑订单的优先级程度，如果优先级程度不为紧急，可能不会先完成该订单而进行累积，在不超过无人机载重限制的前提下将后续到该卸货点的订单进行一起配送。

```
无人机 82: 配送中心5 -> 卸货点50 -> 配送中心5
  - 开始时间: 9.00小时, 总距离: 19.42公里
  - 优先级: 一般, 卸货点: 50, 生成时间: 6.50小时, 截止时间: 9.50小时
  - 优先级: 一般, 卸货点: 50, 生成时间: 7.00小时, 截止时间: 10.00小时
  - 优先级: 一般, 卸货点: 50, 生成时间: 7.00小时, 截止时间: 10.00小时
  - 优先级: 一般, 卸货点: 50, 生成时间: 7.00小时, 截止时间: 10.00小时
  - 优先级: 一般, 卸货点: 50, 生成时间: 7.50小时, 截止时间: 10.50小时
```

例如上图，有五个订单要到卸货点 50，但是卸货点 50 对于配送中心过远，往返距离就已经达到 19.4 公里，无法在前往其他卸货点。此时，会将不紧急的订单进行累积，最终一起配送，但是需要保证累计的订单可以在订单截至时间前配送到。这里订单的截至时间最早为 9.5 时，距离当前时间 9 时仅差半个小时，因此决策为立即配送，并在不超过载重限制的前提下携带上其他到卸货点 50 的订单。

b) 卸货点距离配送中心不远，但它的订单优先级程度不紧急，可能不会先

完成该订单而进行累积，在不超过无人机载重限制的前提下将后续到该卸货点的订单进行一起配送。

```
无人机 28: 配送中心5 -> 卸货点49 -> 配送中心5
  - 开始时间: 3.50小时, 总距离: 10.58公里
  - 优先级: 一般, 卸货点: 49, 生成时间: 1.00小时, 截止时间: 4.00小时
  - 优先级: 一般, 卸货点: 49, 生成时间: 1.00小时, 截止时间: 4.00小时
  - 优先级: 较紧急, 卸货点: 49, 生成时间: 2.50小时, 截止时间: 4.00小时
  - 优先级: 一般, 卸货点: 49, 生成时间: 1.50小时, 截止时间: 4.50小时
  - 优先级: 一般, 卸货点: 49, 生成时间: 1.50小时, 截止时间: 4.50小时
```

例如上图，有五个订单要到卸货点 49，且它们均不紧急，因此会将不紧急的订单进行累积，最终一起配送，但是需要保证累积的订单可以在订单截至时间前配送到。这里对于较紧急的订单它的截至时间为 4 时，距离当前时间 3.5 时仅差半个小时，因此决策为立即配送，携带满 5 个订单后达到载重限制。

ii. 对于常见的类型，在送到某一卸货点后，还可以前往其他卸货点进行配送，此时的约束条件是：订单截止时间、无人机载重、订单优先级以及飞行所需的总距离。

```
无人机 56: 配送中心5 -> 卸货点43 -> 卸货点45 -> 配送中心5
  - 开始时间: 4.50小时, 总距离: 18.12公里
  - 优先级: 紧急, 卸货点: 43, 生成时间: 4.50小时, 截止时间: 5.00小时
  - 优先级: 紧急, 卸货点: 43, 生成时间: 4.50小时, 截止时间: 5.00小时
  - 优先级: 一般, 卸货点: 45, 生成时间: 2.00小时, 截止时间: 5.00小时
  - 优先级: 一般, 卸货点: 45, 生成时间: 2.00小时, 截止时间: 5.00小时
  - 优先级: 较紧急, 卸货点: 45, 生成时间: 3.50小时, 截止时间: 5.00小时
```

A. **算法考虑了订单截止时间与载重的限制：**

- 无人机 56 选择的路径为：配送中心 5 -> 卸货点 43 -> 卸货点 45 -> 配送中心 5。这条路径总距离为 18.12 公里，完成了 5 个订单的配送任务。
- 无人机在 4.5 时开始配送，所有订单的截止时间均为 5 时，说明无人机的出发时间和路径规划考虑到了所有订单的截止时间，确保在订单的截止时间前完成配送。
- 无人机在配送过程中，每次选择的订单都满足最大飞行距离和载重的限制，确保能在规定时间内完成配送。

B. **算法考虑了优先级处理：**

- 紧急订单：两个紧急订单在同一时间生成（4.50 小时），且截止时间为 5.00 小时。无人机 56 在 4.50 小时开始任务，确保在 0.5 小时内（紧急订单的配送时间限制）优先完成了卸货点 43 的紧急订单。
- 较紧急订单：较紧急订单在 3.50 小时生成，截止时间为 5.00 小时。无人机在完成紧急订单后，能够迅速完成该较紧急订单的配送，确保在 1.5 小时的时间限制内完成配送。
- 一般订单：一般订单在较早时间生成（2.00 小时），但截止时间为 5.00 小时。无人机在完成紧急和较紧急订单后，可以在截止时间前完成这些一般订单的配送。

C. 算法考虑了重复卸货点订单的累积处理：

- 从路径信息中可以看出，同一卸货点 43 有多个订单。无人机 56 在一次配送中完成了所有属于卸货点 43 的订单，提高了配送效率，避免了多次往返。

D. 算法考虑了飞行总距离的限制：

- 该无人机完成配送的总距离为 18.12 公里，表明路径规划的有效性。

从上述路径规划中可以看出，无人机的配送路径能够有效覆盖各个卸货点，且在满足订单优先级要求的前提下，尽量缩短了总配送路径。

4.3 总配送距离情况

```
订单总数:2039
配送中心 1 需要配送订单的总数为431, 总配送距离为: 1591.60公里
配送中心 2 需要配送订单的总数为402, 总配送距离为: 1396.88公里
配送中心 3 需要配送订单的总数为372, 总配送距离为: 1388.69公里
配送中心 4 需要配送订单的总数为414, 总配送距离为: 1478.17公里
配送中心 5 需要配送订单的总数为420, 总配送距离为: 1472.88公里
总配送距离: 7328.21公里
```

五个配送中心的总配送距离为 7328.21 公里。平均每个订单的配送距离约为 3.59 公里（7328.21/ 2039）。各配送中心的平均订单配送距离在 3.47 公里到 3.73 公里之间，不存在显著差异。

4.4 分析与讨论

通过实验结果可以看出，使用贪心算法进行路径规划能够有效地满足订单的优先级要求，并尽量缩短总配送路径。在实际应用中，这种算法能够帮助配送企业提高配送效率，降低运营成本。

然而，本实验也存在一些局限性，例如未考虑无人机的充电时间、天气等外部因素对配送的影响。此外，贪心算法虽然简单高效，但在面对复杂的配送需求时，可能无法获得最优解。可以考虑引入更多的优化算法，如遗传算法、模拟退火等，以进一步提升路径规划的效果。对各配送中心的订单可以后续考虑进行更合理的分配，避免某些配送中心过度负担，从而平衡各中心的工作量和总配送距离。当前算法已考虑了订单的累积处理，可以继续优化累积策略，尽量在一次配送中完成更多订单，从而减少总配送次数和距离。

五、实验总结

在本实验中，通过设计并实现路径规划和调度算法，模拟了无人机在配送中心和多个卸货点之间的货物配送过程。

实验结果展示了每个配送中心的无人机路径和总配送距离，反映了系统在不同工作负荷下的表现和效率。贪心算法有效地优化了路径规划过程，使得无人机能够高效地完成订单配送任务，尽可能缩短配送路径，并在订单截止时间内完成配送。通过本次实验，验证了贪心算法在无人机配送路径规划中的一定有效性与合理性。

实验结果表明，该算法能够在满足订单优先级要求的前提下，尽量缩短总配送路径，为无人机配送路径规划提供了一种可行的解决方案。后续可以结合更多实际因素和优化算法，进一步提升无人机配送路径规划的效率和准确性，应对更复杂的配送需求和动态调度场景。