编号：_____

实验	一	二	三	四	五	六	七	八	总评	教师签名
成绩										

武汉大学国家网络安全学院

课程实验（设计）报告

课程名称：高级算法

实验内容：无人机配送路径规划问题

专　　业：网络空间安全

学　　号：2023202210100

姓　　名：姜 蕊

任课教师：林 海

2024 年 6 月 17 日

目录

无人机配送路径规划问题

一、 问题描述与分析

无人机可以快速解决最后 10 公里的配送，本作业要求设计一个算法，实现如下图所示区域的无人机配送的路径规划。在此区域中，共有 j 个配送中心，任意一个配送中心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。该区域同时有 k 个卸货点（无人机只需要将货物放到相应的卸货点即可），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：3 小时内配送到即可；
- 较紧急：1.5 小时内配送到；
- 紧急：0.5 小时内配送到。

我们将时间离散化，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，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，即先完成那个订单，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。

目标：一段时间内（如一天），所有无人机的总配送路径最短

约束条件：满足订单的优先级别要求

假设条件：

1. 无人机一次最多只能携带 n 个物品；
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；

二、 图设置与数据初始化

为了模拟无人机配送的路径规划，我们首先设置实验参数，并初始化配送中心和卸货点的坐标。

2.1 参数设置

实验中的参数如下：

- $I = 5$：配送中心数量
- $J = 50$：卸货点数量
- $max\_items\_per\_drone = 10$：无人机一次最多携带的物品数量
- $max\_flight\_distance = 20$：无人机一次飞行最远路程
- $drone\_speed = 60$：无人机速度 (km/h)
- $order\_interval = 60$：订单生成间隔（分钟）
- $order\_max = 3$：每个卸货点最大订单数量
- $simulation\_duration = 24 \times 60$：模拟时间（分钟）

订单优先级和配送时间要求（分钟）如下：

- 一般（优先级 1）：180 分钟
- 较紧急（优先级 2）：90 分钟
- 紧急（优先级 3）：30 分钟

为了确保实验的可重复性，可以设置随机数种子：

```
np.random.seed(3)
random.seed(3)
```

2.2 坐标初始化

首先为配送中心和卸货点生成随机坐标。构造一个 50× 大小的区域，随机生成配送中心的坐标如下：

```
# 初始化配送中心坐标
xc_depots = np.random.rand(I) * 50
yc_depots = np.random.rand(I) * 50
```

接下来，为卸货点生成坐标，由于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，并且无人机的速度为 60 公里/小时，所以卸货点不能距离配送中心 10 公里以上，否则无人机将无法配送。所以为了确保每个卸货点都在至少一个配送中心的最大飞行距离范围内，在配送中心周围一定范围内随机生成卸货点。

```
# 初始化卸货点坐标
xc_customers = []
yc_customers = []
while len(xc_customers) < J:
    x = np.random.rand() * 50
    y = np.random.rand() * 50
    distances = [np.sqrt((x - xc_depots[i]) ** 2+ (y - yc_depots[i]) ** 2) for
        i in range(I)]
    if min(distances) <= max_flight_distance / 3:
        xc_customers.append(x)
        yc_customers.append(y)
```

最后，将配送中心和卸货点的坐标合并：

```
# 合并坐标
xc = np.concatenate((xc_depots, xc_customers))
yc = np.concatenate((yc_depots, yc_customers))
```

2.3 初始位置绘制

使用以下函数绘制初始配送中心和卸货点的位置：

```
def plot_init(xc, yc):
    # 标记配送中心和卸货点
    for i in range(len(xc)):
        label = f'D{i + 1}' if i < I else f'C{i - I + 1}'
        color = 'red' if i < I else 'blue' # 配送中心用红色，卸货点用蓝色
        plt.scatter(xc[i], yc[i], marker='D', s=10, color=color)
        plt.text(xc[i], yc[i], label, fontsize=7, color='black', ha='right',
            va='bottom')

    plt.xlabel('X坐标')
    plt.ylabel('Y坐标')
    plt.show()

# 绘制初始配送中心和卸货点的位置
plot_init(xc, yc)
```

可以看到图中配送中心和卸货点的位置信息，这里用 D_i 来代表配送中心，C_j 来代表卸货点。

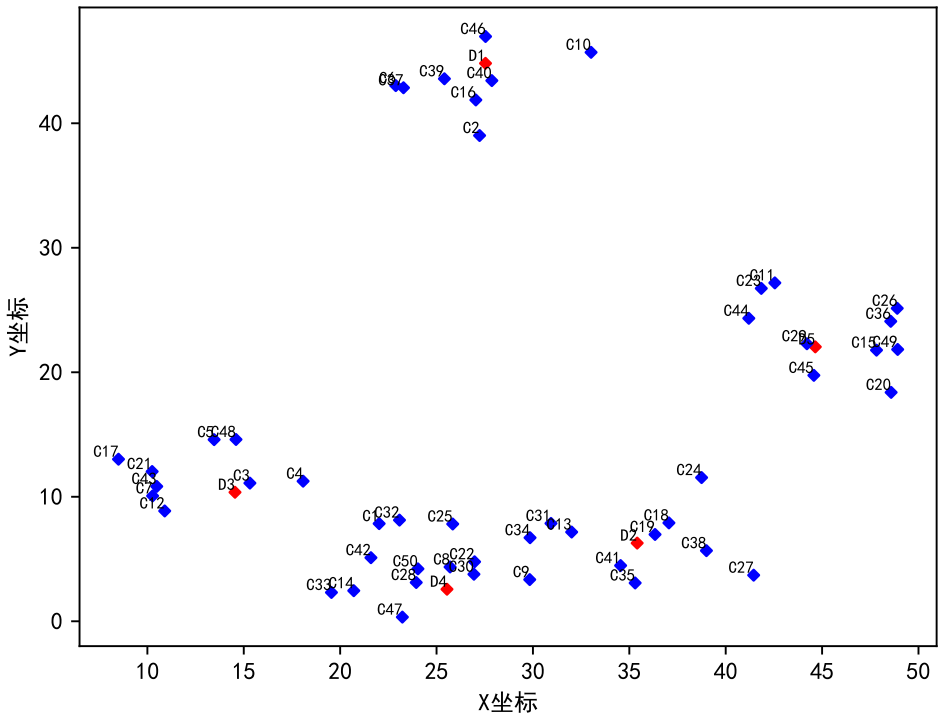

图 1: 初始图

三、 问题分析与算法确定

3.1 问题分析

本实验旨在解决无人机在有限时间内完成配送任务的路径规划问题。我们需要考虑以下几个方面：

1. **订单的优先级别和时间要求：**订单有三个优先级别，并且必须确保订单在对应的时间限制内完成。每隔一段时间（本实验设置一小时）有新的订单生成，系统同时要做出决策。所以这里考虑优先在本阶段处理优先级最高的订单。
2. **无人机的约束条件：**
   - 无人机一次最多只能携带 n 个物品。
   - 无人机一次飞行最远路程为 20 公里（送完货后需要返回配送点）。
   - 配送中心的无人机数量无限，任意一个配送中心都能满足用户的订货需求。
3. **目标：**在满足所有订单优先级和时间要求的前提下，使得一天内所有无人机的总配送路径最短。

无人机的总配送路径越短，也就意味着无人机在一次配送中能处理更多的卸货点。比如有配送中心 D_1 和卸货点 C_1 和 C_2，D_1->C_1->C_2->D_1 的距离小于 D_1->C_1->D_1，D_1->C_2->D_1，因为两边之和大于第三边，除非三者在一条直线上。

本实验选择用贪心算法来解决问题，优先处理高优先级的订单，将时间按照 t = 60 分钟来将一天分成 24 段。在前 23 段中，对于所有最高优先级的订单，贪心搜索可以完成该订单并且在不超出无人机飞行最长距离和携带最多物品数量的约束条件下，还能同时在这条路径中处理的个更多的订单，这些订单可以是高优先级的也可以是低优先级的。余下的没有处理的低优先级订单可以留到下阶段再处理。但是在最后一段时间中，要对所有订单处理，贪心得到最优路径，将所有订单处理完毕。

3.2 贪心算法解决问题

为了解决上述问题，我们设计了一种基于贪心算法的路径规划策略，具体步骤如下：

3.2.1 订单生成

定义一个订单类，包括卸货点的位置，订单优先级，和生成时间。

```
class Order:
    def __init__(self, point, priority, time):
        self.point = point
        self.priority = priority
        self.time = time
```

每隔 $t = 60$ 分钟，各个卸货点会随机生成订单。每个订单有一个优先级和生成时间。订单优先级有三个等级，分别是 1（一般），2（较紧急），和 3（紧急），对应的配送时间要求分别为 180 分钟、90 分钟和 30 分钟。

```
# 生成订单
def generate_orders(current_time):
    orders = deque()
    for j in range(J):
        for _ in range(random.randint(0, order_max)):
            priority = random.choices([1, 2, 3], [0.6, 0.3, 0.1])[0]
            orders.append(Order(point=j, priority=priority, time=current_time))
    return orders
```

3.2.2 优先级订单的调度

对于每一个时间段，系统根据订单的优先级进行调度。首先将所有订单按照优先级从高到低排序，然后选择最高优先级的订单作为新无人机路径的起始配送任务。在无人机配送路径规划问题中，订单的优先级对调度策略有着重要的影响。为了确保高优先级订单能够优先得到处理，实验设计了一种动态调整订单优先级的策略，并基于此策略进行调度。具体步骤如下：

1. **更新订单优先级：**随着时间的推移，如果一个订单没有在其优先级所要求的时间内得到处理，其优先级将会提升一级，直至达到最高优先级（紧急）。在每个时间间隔 t 结束时，我们检查所有未完成的订单，根据其生成时间和当前优先级，动态调整其优先级。

```
# 更新订单优先级
for order in global_orders:
    if order.priority < 3:
        new_priority = order.priority + 1
        if current_time - order.time > priority_levels[new_priority]:
            order.priority = new_priority
```

2. **按优先级排序订单：**在每个时间间隔 t 内，系统对所有订单按照优先级从高到低进行排序。在最后一个时间间隔 t，系统会处理所有剩余订单。在非最后一个时间间隔 t 中，系统只处理当前最高优先级的订单。

```
# 按照优先级排序订单，只处理当前最高优先级的订单
if t < time_intervals - 1:
    global_orders.sort(key=lambda x: x.priority, reverse=True)
    highest_priority = global_orders[0].priority if global_orders else 1
    highest_priority_orders = [order for order in global_orders if
        order.priority == highest_priority]
else:
    highest_priority_orders = global_orders
```

3. **找到最优路径：**对于每个最高优先级订单，系统首先找到离该订单最近的配送中心。然后，使用贪心算法为无人机规划最佳路径。在非最后一个时间间隔 t 中，无人机会先处理当前最高优先级订单，再处理其他订单；在最后一个时间间隔 t 中，无人机会按照贪心算法规划的路径依次处理订单。

```
# 为每个最高优先级订单找到最优路径
for order in highest_priority_orders:
    distances = [dist_matrix[i, order.point + I] for i in range(I)]
    closest_depot = np.argmin(distances)

    # 使用贪心算法找到从该订单出发的最优路径
    if t < time_intervals - 1:
        best_route = greedy_algorithm(global_orders, closest_depot,
            start_point=order.point + I)
    else:
        best_route = greedy_algorithm(global_orders, closest_depot)

    # 将当前最高优先级订单加入路径
    if t < time_intervals - 1:
```

```
        best_route = [order.point + I] + [point for point in best_route if point
            != order.point + I]
```

4. **检查路径和更新全局订单：**系统检查规划出的最佳路径是否超过无人机的最大飞行距离。如果符合条件，则将该路径加入到配送路径中，并从全局订单列表中删除已经完成配送的订单，防止重复和错误处理。

```
# 检查最佳路径是否超过最大飞行距离
total_distance = fitness(best_route, closest_depot)
if total_distance <= max_flight_distance:
    all_paths[closest_depot].append([closest_depot] + best_route +
        [closest_depot])
    converted_route = convert_route_to_labels(best_route)
    print(f"时间 {current_time} 分钟: 配送中心 D{closest_depot + 1} 的最佳路径:
        {converted_route}, 距离: {total_distance}")

    # 累加每个无人机的行驶距离到总行驶距离
    total_distance_all_drones += total_distance

    # 派遣无人机根据最佳路径配送订单，并从全局订单中删除这些订单
    for point in best_route:
        customer_point = point - I
        global_orders = [order for order in global_orders if order.point !=
            customer_point]
```

通过以上步骤，能够有效地调度无人机，使得高优先级订单能够优先得到处理，并在满足订单优先级和时间要求的前提下，尽可能减少总配送距离。

3.2.3 路径规划

为了在满足订单优先级要求的前提下，使所有无人机的总配送路径最短，这里采用了贪心算法来进行路径规划。在每个时间间隔内，无人机从配送中心出发，按照优先级顺序处理订单，规划出最优的配送路径。下面详细描述路径规划的具体步骤和贪心算法的实现。

贪心算法通过逐步选择距离最近的未处理订单点，构建无人机的配送路径。算法过程中需要考虑无人机的最大飞行距离和每次最多携带物品数量的约束。具体实现步骤如下：

1. **初始化路径和剩余距离：**如果有指定的起始点（如当前最高优先级订单的卸货点），则将其加入路径并更新剩余的客户点和总距离。

```
def greedy_algorithm(orders, depot, start_point=None):
    if not orders:
```

```
        return []

    route = []
    total_distance = 0
    customer_orders = defaultdict(int)

    for order in orders:
        customer_orders[order.point + I] += 1

    remaining_customers = set(customer_orders.keys())

    if start_point:
        route.append(start_point)
        remaining_customers.discard(start_point)
        total_distance += dist_matrix[depot, start_point]
```

2. **迭代选择最近的客户点：**在路径未满载（未达到无人机一次最多携带物品数量）且剩余客户点未处理完的情况下，选择当前路径中最后一个点到剩余客户点中最近的点。如果加入该点后路径的总距离不超过无人机的最大飞行距离，则将其加入路径，并更新剩余客户点和总距离。

```
    while remaining_customers and len(route) < max_items_per_drone:
        if not route:
            next_customer = min(remaining_customers, key=lambda x:
                dist_matrix[depot, x])
            route.append(next_customer)
            total_distance += dist_matrix[depot, next_customer]
        else:
            last = route[-1]
            next_customer = min(remaining_customers, key=lambda x:
                dist_matrix[last, x])
            next_distance = dist_matrix[last, next_customer] +
                dist_matrix[next_customer, depot]
            if total_distance + next_distance <= max_flight_distance:
                route.append(next_customer)
                total_distance += dist_matrix[last, next_customer]
            else:
                break

        customer_orders[next_customer] -= 1
        if customer_orders[next_customer] == 0:
            remaining_customers.remove(next_customer)

    return route
```

3. **返回最终路径：**在路径满载或剩余客户点无法在约束条件下加入路径后，返回当前构建的路径。

```
return route
```

通过以上步骤，贪心算法能够在考虑无人机最大飞行距离和携带物品数量的约束下，规划出尽可能优化的配送路径。该算法的核心思想是在每一步选择局部最优解（即最近的未处理订单点），从而逐步构建接近全局最优的路径。尽管贪心算法无法保证全局最优解，但在实际应用中通常能够获得较好的近似解，并且计算复杂度较低，适合于实时决策场景。

3.2.4 无人机调度与路径执行

在每个时间段内，系统根据当前订单情况，调度无人机并执行相应的配送任务。路径执行过程中，系统会持续更新全局订单列表，移除已完成的订单，确保后续时间段的调度和路径规划的准确性。

```
# 为每个配送中心调度无人机并规划路径
for depot in range(I):
    filtered_orders = depot_orders[depot]

    # 在满足无人机约束的情况下批量处理订单
    while filtered_orders:
        batch = filtered_orders[:max_items_per_drone]
        best_route = greedy_algorithm(batch, depot)

        # 检查最佳路径是否超过最大飞行距离
        total_distance = fitness(best_route, depot)
        if total_distance <= max_flight_distance:
            all_paths[depot].append([depot] + best_route + [depot])
            converted_route = convert_route_to_labels(best_route)
            print(f"时间 {current_time} 分钟: 配送中心 D{depot + 1} 的最佳路径:
                {converted_route}, 距离: {total_distance}")

            # 累加每个无人机的行驶距离到总行驶距离
            total_distance_all_drones += total_distance

            # 派遣无人机根据最佳路径配送订单
            for point in best_route:
                customer_point = point - I
                delivered_orders = [order for order in global_orders if
                    order.point == customer_point]
                for order in delivered_orders:
```

```
                global_orders.remove(order)

        # 从过滤后的订单中移除已配送的订单
        filtered_orders = [order for order in filtered_orders if order.point
            + I not in best_route]
```

通过上述步骤和算法设计，我们能够有效地规划无人机的配送路径，确保在满足订单优先级和时间要求的前提下，尽可能地减少总配送距离。

3.2.5 无人机配送路径主循环

无人机调度与路径执行的主函数是模拟主循环。该循环每隔固定时间间隔生成新订单，更新订单优先级，并计算最优配送路径。以下是主循环的详细解释。

模拟主循环是整个路径规划算法的核心部分，它负责生成订单、更新订单优先级、寻找最优路径并派遣无人机执行配送任务。该部分的伪代码如下：

初始化时生成初始订单，并计算整个模拟过程中时间间隔的数量。同时，初始化存储路径的变量和记录总行驶距离的变量。

```
global_orders = generate_orders(0) # 初始化时生成订单
time_intervals = simulation_duration // order_interval

total_distance_all_drones = 0

# 存储每个配送中心的所有路径以便绘图
all_paths = [[] for _ in range(I)]
```

在每个时间间隔中，首先如果已经有剩余订单存在，更新每个订单的优先级。

```
for t in range(time_intervals):
   current_time = t * order_interval
   # 更新订单优先级
   for order in global_orders:
      if order.priority < 3:
         new_priority = order.priority + 1
         if current_time - order.time > priority_levels[new_priority]:
            order.priority = new_priority
```

根据当前时间生成新订单并加入全局订单列表中。然后，检查所有订单是否有违反了配送时间要求。

```
   # 生成新订单
```

```
    new_orders = generate_orders(current_time)
    global_orders.extend(new_orders)

    global_orders = [order for order in global_orders if not is_violated(order,
        current_time)]
```

将订单按照优先级进行排序，选择当前最高优先级的订单进行处理。在最后一个时间间隔中，处理所有剩余订单。

```
    # 按照优先级排序订单，只处理当前最高优先级的订单
    if t < time_intervals - 1:
        global_orders.sort(key=lambda x: x.priority, reverse=True)
        highest_priority = global_orders[0].priority if global_orders else 1
        highest_priority_orders = [order for order in global_orders if
            order.priority == highest_priority]
    else:
        highest_priority_orders = global_orders
```

对于每个最高优先级订单，计算其到所有配送中心的距离，并选择最近的配送中心。

```
    while highest_priority_orders:
        order = highest_priority_orders[0]
        distances = [dist_matrix[i, order.point + I] for i in range(I)]
        closest_depot = np.argmin(distances)
```

根据订单的优先级，使用贪心算法为每个订单找到最优路径，并确保当前订单被包含在路径中（如果不是最后一个时间段）。

```
        # 使用贪心算法找到从该订单出发的最优路径
        if t < time_intervals - 1:
            best_route = greedy_algorithm(global_orders, closest_depot,
                start_point=order.point + I)
        else:
            best_route = greedy_algorithm(global_orders, closest_depot)

        # 将当前最高优先级订单加入路径
        if t < time_intervals - 1:
            best_route = [order.point + I] + [point for point in best_route if
                point != order.point + I]
```

模拟派遣无人机执行配送任务，并从全局订单列表和最高优先级订单列表中删除已配送的订单。

```
        total_distance = fitness(best_route, closest_depot)
```

```
        if total_distance <= max_flight_distance:
            all_paths[closest_depot].append([closest_depot] + best_route +
                [closest_depot])
            converted_route = convert_route_to_labels(best_route)
            print(f"时间 {current_time} 分钟: 配送中心 D{closest_depot + 1}
                的最佳路径: {converted_route}, 距离: {total_distance}")

            # 累加每个无人机的行驶距离到总行驶距离
            total_distance_all_drones += total_distance

            # 派遣无人机根据最佳路径配送订单，并从全局订单中删除这些订单
            for point in best_route:
                customer_point = point - I
                global_orders = [order for order in global_orders if order.point
                    != customer_point]
                highest_priority_orders = [order for order in
                    highest_priority_orders if order.point != customer_point]
```

在每个时间段结束时，可以选择输出总行驶距离和绘制路径图。最后输出全天的总行驶距离。

```
    # 输出每个时间段结束时的总行驶距离
    # print(f"时间 {current_time} 分钟后: 总行驶距离:
        {total_distance_all_drones}")

    # 绘制当前时间段的路径图
    if current_time == 0:
        plot_paths(all_paths, xc, yc, total_distance_all_drones, current_time)
# 最终输出所有时间段结束时的总行驶距离
print("Total Distance Covered by All Drones: ", total_distance_all_drones)
```

四、 运行结果展示

本节展示模拟无人机配送路径规划算法在不同时间间隔下的运行结果。通过实际运行代码，我们收集了总配送距离、各配送中心的最佳路径以及订单的处理情况。

4.1 初始配送中心和卸货点位置

首先，展示配送中心和卸货点的初始位置图。在图 1 中红色的点代表配送中心，蓝色的点代表卸货点。配送中心和卸货点的坐标是随机生成的，且保证每个卸货点距离最近的配送中心不超过无人机最大飞行距离。

4.2 配送路径输出示例

在模拟过程中，我们记录了每个时间段内各配送中心的最佳路径。以下为时间段为 0 分钟时，配送中心的最佳路径示例。

```
时间 0 分钟: 配送中心 D3 的最佳路径: ['C4', 'C3', 'C43', 'C21', 'C12'], 距离: 19.645570819379415
时间 0 分钟: 配送中心 D2 的最佳路径: ['C13', 'C19', 'C19', 'C18', 'C38', 'C35'], 距离: 19.697547410662956
时间 0 分钟: 配送中心 D1 的最佳路径: ['C16'], 距离: 5.968072400782534
时间 0 分钟: 配送中心 D5 的最佳路径: ['C23', 'C36'], 距离: 17.104909284140057
时间 0 分钟: 配送中心 D4 的最佳路径: ['C32', 'C42', 'C30'], 距离: 16.789691908433667
时间 0 分钟: 配送中心 D5 的最佳路径: ['C45'], 距离: 4.577338806793553
```

图 2: 输出配送路径

从结果中可以看到，每个配送中心在当前时间段对应的最佳路径，并且这些路径都满足无人机最大飞行距离的约束。

4.3 总配送距离

在模拟的每个时间段结束时，我们统计了所有无人机的总配送距离。以下为模拟结束时的总配送距离示例：

```
Total Distance Covered by All Drones:  2527.131544370181
```

图 3: 总配送距离

该结果表明，在一天的时间内，所有无人机的总配送距离为 2527.13 公里。通过路径优化算法，我们尽可能减少了总配送距离，从而提高了配送效率。

4.4 路径图绘制

为了直观展示配送路径，我们在每个时间段结束时绘制了路径图。以下为部分时间段的路径图示例。

图 4 和图 5 分别展示了第一个时间间隔和最后一个时间间隔各配送中心的配送路径。红色线条表示从配送中心出发的路径，蓝色点表示卸货点。第一个时间间隔中，可以看到有很多离散的点，这是因为有些订单因为优先级不够高，放到了下一阶段再处理。最后一个时间间隔中，所有订单都得到了配送。(这里离散点只是因为订单是随机生成的，这几个配货点此时没有订单)。

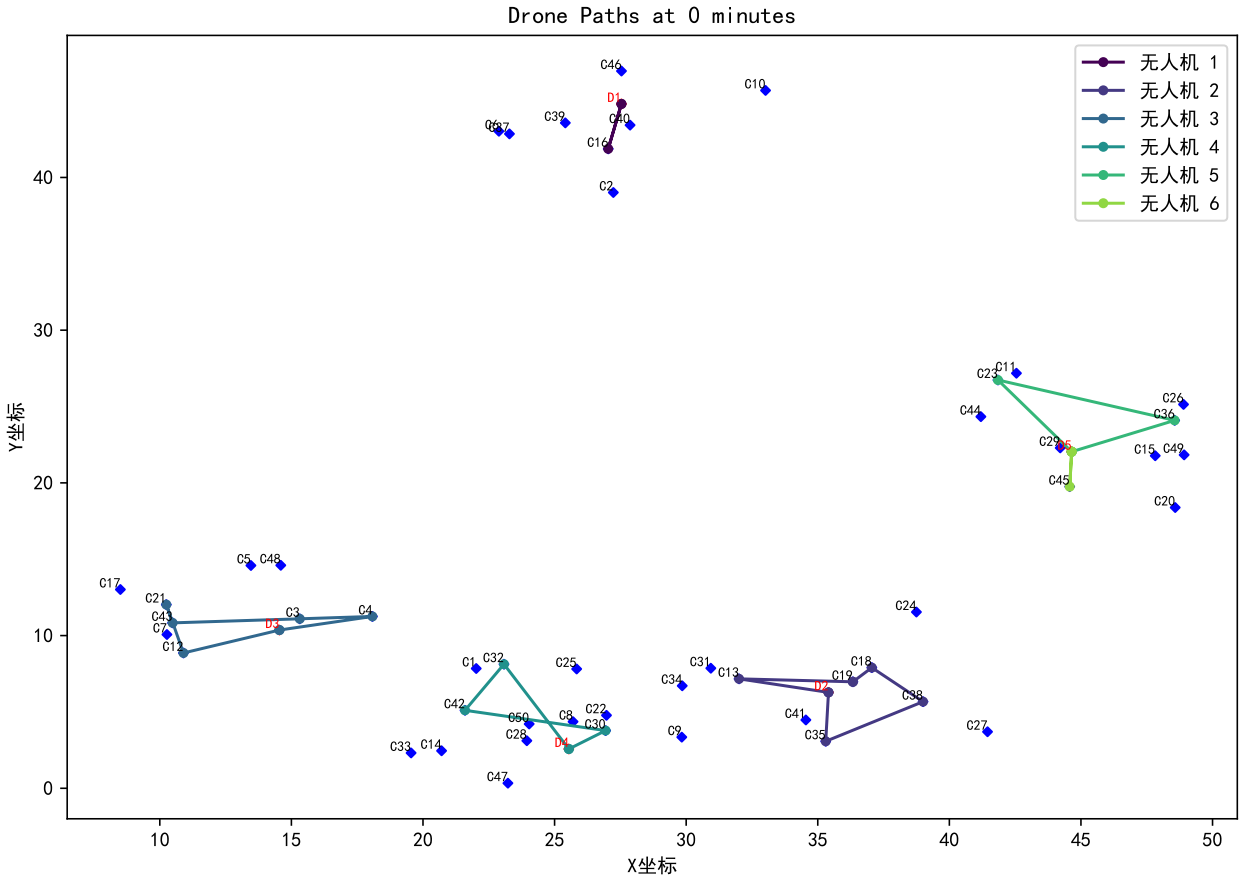
Drone Paths at 0 minutes
无人机 1
无人机 2
无人机 3
无人机 4
无人机 5
无人机 6
X坐标
Y坐标
0
10
20
30
40
10
15
20
25
30
35
40
45
50

图 4: 初始图

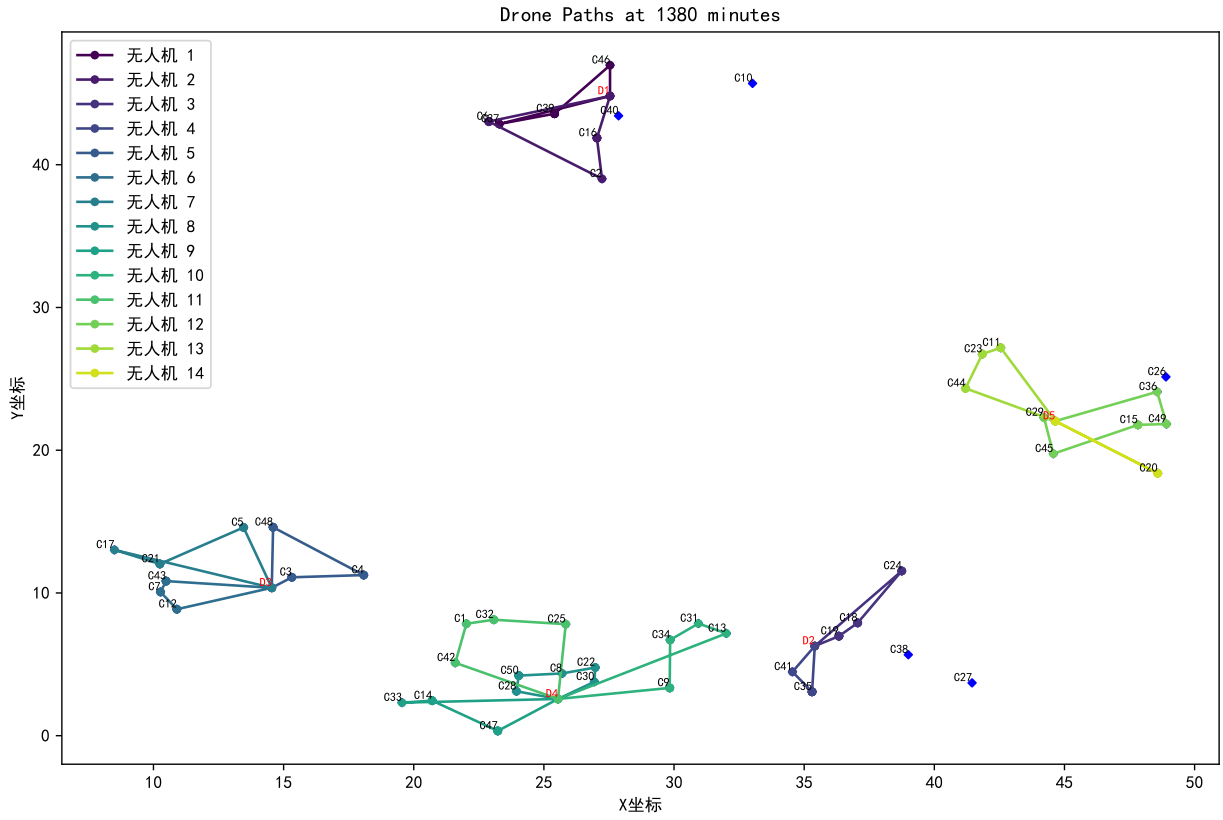

图 5: 最后一个阶段配送图

五、 其他解决算法补充

5.1 聚类后采用启发式算法

前文中的贪心算法，是在不断根据每个订单信息去搜索下一个要到达的地点，得到一条当前的贪心最优路径。而这里可以首先把卸货点用聚类方法分配给各个配送中心，然后分别对各个配送中心求解路径规划问题。这样就问题无人机多配送中心路径规划问题进行了分割和简化。

5.1.1 聚类

计算配送中心和订单的亲密度：亲密度是衡量订单与配送中心之间适合度的一个指标。由于本题目中不限制配送中心的无人机数量，因此亲密度可以根据配送中心与订单之间的距离和订单的时间窗口决定。这里可以对所有本阶段需要处理的订单，按照亲密度将其归到最合适的配送中心类，亲密度越高，联系越紧密。这样每个配送中心有一个带送订单集合，完成聚类。

同样，最后一阶段不仅要处理紧急订单，需要对所有订单聚类。

```
def assign_nodes_to_centers(demand_points, centers):
    adjustedNodes = True
    while adjustedNodes:
        adjustedNodes = False
        for point in demand_points:
            best_center = None
            best_affinity = float('-inf')
            for center in centers:
                affinity = calculate_affinity(point, center)
                if affinity > best_affinity:
                    best_affinity = affinity
                    best_center = center
            if point.assigned_center != best_center:
                point.assigned_center = best_center
                adjustedNodes = True

for order in orders:
    best_center = find_best_center(order)
    assign_order_to_center(order, best_center)
```

5.1.2 启发式规划路径

接下来为每个配送中心规划无人机的配送路径，使得总配送路径最短，满足订单的优先级要求和无人机的飞行距离限制。可以采用遗传算法（GA）对每个配送中心类进行路径规划。

首先描述竞争一插入过程 (C&IP，Competition and Insertion Procedure)：

1. 初始化：选择一个初始节点作为起始点，构建初始路径。
2. 竞争插入：对于每个未插入的节点，计算插入到当前路径中每个位置的代价（如总路径长度增加）。
3. 选择最佳插入位置：将节点插入到使得路径增加最小的位置。
4. 重复步骤 2-3，直到所有节点都被插入路径中。

遗传算法具体过程如下：

- **染色体编码**：使用位置编码，染色体中的基因顺序对应卸货点顺序，基因表示无人机编号和卸货点在这条路径中的位置。每个基因用一个整型数表示，由两部分组成：无人机编号和该卸货点在车辆路径中的位置，如第 4 个基因为 “05011” 表示第 4 个需求点分配给第 5 条路径中的第 11 个服务位置。
- **适应度函数**：设置为无人机的总飞行距离的倒数。
- **初始种群生成**：使用竞争-插入过程（C&IP）构造初始种群。从订单集合中随机选择一个订单，尝试插入到当前所有无人机路径的最佳位置，选择其中插入后总距离增量最小的路径进行插入。如果当前所有无人机路径都不满足约束条件，则新开一个无人机路径。重复上述过程，直至所有订单被分配到路径中。

```
for order in orders:
    best_drone, best_position = find_best_insertion(order)
    insert_into_route(best_drone, best_position, order)
```

- **交叉和变异**：使用轮盘选择法，采用异位交叉，将两条染色体的部分基因交换；对部分基因进行随机变异，增加解的多样性。但是交叉和变异操作可能会生成不合法的解。因此需要检查所有路径，对于不合法的点，重新使用 C&IP 插入路径中。
- **迭代优化**：多次迭代路径规划和改进，得到当前配送中心类中所有订单的最优路径。

5.2 整数线性规划求解问题

此题目还可以使用混合整数线性规划（MILP）模型求解。

5.2.1 变量和参数

定义和参数：

- I: 配送中心的数量
- J: 卸货点的数量

- KS: 无人机的数量
- $U = I \cup J$: 所有节点集合
- $d[i, j]$: 节点 i 到节点 j 的距离。
- D_{max}: 无人机一次飞行的最远距离（20 公里）
- v: 无人机的速度（60 公里/小时）
- Q: 无人机的最大载重
- $a[j]$: 卸货点 j 的服务时间窗口开始时间
- $b[j]$: 卸货点 j 的服务时间窗口结束时间
- $need[j]$: 卸货点 j 的订单需求（可以为多个订单）

需要注意的是，$need[j]$ 可以是当前最高优先级的待处理订单，优先级较低的可以最后一个时间阶段处理。并且题目中实际要求无人机数量是无限的，因此这里的 KS 可以预先设置一个很大的值，实际解的结果中 KS 一定会比这个值小（因为是线性规划最优）。

决策变量：

- $x[i, j, k]$: 若无人机 k 从节点 i 飞到节点 j，则为 1；否则为 0。
- $l[i, k]$: 无人机 k 是否访问点 i , 取值为携带的物品数量，未访问设为 0。
- $t[j, k]$: 无人机 k 到达卸货点 j 的时间。

5.2.2 目标函数

目标函数旨在最小化总配送成本，包括路程成本和平衡成本（用于考虑路径优化的平衡因素）：

$$\text{Minimize} \sum_{(i,j)\in A} \sum_{k\in KS} x[i, j, k] \cdot d[i, j] \tag{1}$$

5.2.3 约束条件

为了确保问题的实际可行性和求解的有效性，我们引入了多个约束条件：

- **无人机不访问自身**：避免无人机从一个节点飞到自身。

$$x[i, i, k] = 0, \quad \forall i \in U, \forall k \in KS \tag{2}$$

- **无人机容量约束**：保证每一个无人机不超过最大携带量。

$$\sum_{j\in J} l[j, k] \leq Q, \quad \forall k \in KS \tag{3}$$

- **每一次卸货点配送的限制**：配送的数量不能多于需求数量。

$$l[j,k] \leq \sum_{i \in U} x[i,j,k] * need[j], \quad \forall j \in J, \forall k \in KS \tag{4}$$

- **订单约束**：保证每一个卸货点的待处理订单都可以被处理。

$$\sum_{k \in KS} l[j,k] = need[j], \quad \forall j \in J \tag{5}$$

- **流量守恒约束**：保证每个点的无人机的出发和到达数量相等，并且最多为 1。

$$\sum_{i \in U} x[i,j,k] = \sum_{i \in U} x[j,i,k] \leq 1, \quad \forall j \in U, \forall k \in KS \tag{6}$$

- **无人机最大飞行距离限制**：无人机从配送中心出发，完成所有订单并返回配送中心的总飞行距离不能超过 20 公里。

$$\sum_{(i,j) \in A} d[i,j] \cdot x[i,j,k] \leq D_{max}, \quad \forall k \in KS \tag{7}$$

- **最多一个配送中心约束**：确保每辆无人机的路径中最多只有一个配送中心。

$$\sum_{i \in IS} \sum_{j \in U} x[i,j,k] = 1, \quad \forall k \in KS \tag{8}$$

$$\sum_{i \in U} \sum_{j \in IS} x[i,j,k] = 1, \quad \forall k \in KS \tag{9}$$

- **MTZ（Miller, Tucker, Zemlin）子路径消除约束**：通过确保路径的连续性来避免子路径的生成。

$$t[j,k] \geq t[i,k] + d[i,j]/v - M \cdot (1 - x[i,j,k]), \quad \forall i,j \in U, \forall k \in KS \tag{10}$$

- **时间窗约束**：确保每个顾客在其指定的时间窗内得到服务。

$$a[j] \leq t[j,k] \leq b[j], \quad \forall j \in J \tag{11}$$

5.2.4 求解线性规划

对于线性规划可以使用 Gurobi 求解器，它可以在合理的时间内找到最优解或接近最优的解。下面是目标函数和约束设置：

```
1
2 # Objective function: Minimize total distance
3 model.setObjective(quicksum(d[i, j] * x[i, j, k] for i, j in A for k in KS),
      GRB.MINIMIZE)
4
5 # Constraints
6
7 # Drones do not visit themselves
```

```
model.addConstrs(x[i, i, k] == 0for i in U for k in KS)

# Capacity constraint
model.addConstrs(quicksum(l[j, k] for j in JS) <= Q for k in KS)

# Delivery limit for each drop-off point
model.addConstrs(l[j, k] <= quicksum(x[i, j, k] * need[j] for i in U) for j in
    JS for k in KS)

# Order fulfillment
model.addConstrs(quicksum(l[j, k] for k in KS) == need[j] for j in JS)

# Flow conservation constraint
model.addConstrs(quicksum(x[i, j, k] for i in U) == quicksum(x[j, i, k] for i
    in U) <= 1for j in U for k in KS)

# Maximum flight distance
model.addConstrs(quicksum(d[i, j] * x[i, j, k] for i, j in A) <= D_max for k
    in KS)

# At most one depot per path
model.addConstrs(quicksum(x[i, j, k] for j in U) == 1for i in IS for k in KS)
model.addConstrs(quicksum(x[i, j, k] for i in U) == 1for j in IS for k in KS)

# MTZ constraints
model.addConstrs(t[j, k] >= t[i, k] + t[i, j] - M * (1 - x[i, j, k]) for i, j
    in A for k in KS)

# Time window constraints
model.addConstrs(a[j] <= t[j, k] for j in JS for k in KS)
model.addConstrs(t[j, k] <= b[j] for j in JS for k in KS)

# Optimize model
model.optimize()
```

六、 总结

通过算法设置和实现结果展示，我们可以看到使用贪心算法的无人机配送路径规划在不同时间间隔下的运行情况。该算法有效地优化了配送路径，减少了总配送距离，并且能够满足订单的优先级要求。此外，本实验还给出了使用聚类 + 启发式算法、以及整数线性规划求解的额外解决方法。未来的研究可以进一步考虑更多的约束条件和优化目标，如考虑配送时间窗口和多目标优化等。