高级算法设计与分析

课程大作业

院（系）名 称: 国家网络安全学院

专 业 名 称 ： 网络空间安全

学 生 姓 名 ： 林 晨

授 课 教 师 ： 林 海

二〇二四年六月

一、背景介绍

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、数据生成

2.1 配送中心和配送点生成

固定随机种子为 42，生成 6 个配送中心与 12 个配送点：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 设定随机种子以确保结果可重复
np.random.seed(42)

# 生成 6 个配送中心的坐标 (x, y)
delivery_centers = np.random.uniform(0, 10, (6, 2))

# 生成 12 个配送点的坐标 (x, y)
drop_off_points = np.random.uniform(0, 10, (12, 2))
```

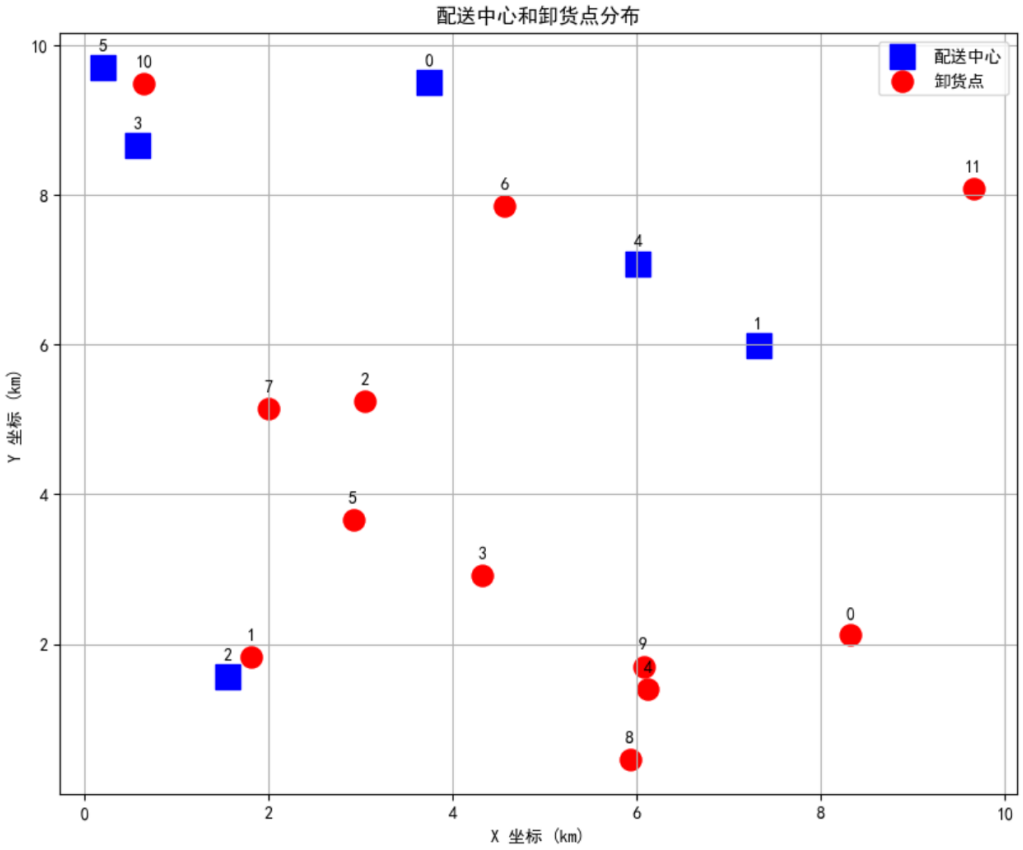
查看数据表格并绘制配送中心和卸货点分布图：

```
# 创建数据表格
centers_df_cn = pd.DataFrame(delivery_centers, columns=['X 坐标 (km)', 'Y 坐标 (km)'])
centers_df_cn.index.name = '配送中心 ID'
drop_off_df_cn = pd.DataFrame(drop_off_points, columns=['X 坐标 (km)', 'Y 坐标 (km)'])
drop_off_df_cn.index.name = '卸货点 ID'

# 正常显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
# 绘制图形
plt.figure(figsize=(10, 8))
plt.scatter(centers_df_cn['X 坐标 (km)'], centers_df_cn['Y 坐标 (km)'], color='blue', label='配送中心', marker='s', s=200)
plt.scatter(drop_off_df_cn['X 坐标 (km)'], drop_off_df_cn['Y 坐标 (km)'], color='red', label='卸货点', marker='o', s=150)
for i, txt in enumerate(centers_df_cn.index):
    plt.annotate(txt, (centers_df_cn['X 坐标 (km)'][i], centers_df_cn['Y 坐标 (km)'][i]), textcoords="offset points", xytext=(0, 10), ha='center')
for i, txt in enumerate(drop_off_df_cn.index):
    plt.annotate(txt, (drop_off_df_cn['X 坐标 (km)'][i], drop_off_df_cn['Y 坐标 (km)'][i]), textcoords="offset points", xytext=(0, 10), ha='center')
plt.xlabel('X 坐标 (km)')
plt.ylabel('Y 坐标 (km)')
```

```
plt.title('配送中心和卸货点分布')
plt.legend()
plt.grid(True)
plt.show()
```

生成的 6 个配送中心和 12 个卸货点分布图如下所示：



6 个配送中心和 12 个卸货点具体的数据表格如下所示：

卸货点 ID	X 坐标 (km)	Y 坐标 (km)
0	8.324426	2.123391
1	1.818250	1.834045
2	3.042422	5.247564
3	4.319450	2.912291
4	6.118529	1.394939
5	2.921446	3.663618
6	4.560700	7.851760
7	1.996738	5.142344
8	5.924146	0.464504
9	6.075449	1.705241
10	0.650516	9.488855
11	9.656320	8.083973

配送中心 ID	X 坐标 (km)	Y 坐标 (km)
0	3.745401	9.507143
1	7.319939	5.986585
2	1.560186	1.559945
3	0.580836	8.661761
4	6.011150	7.080726
5	0.205845	9.699099

2.2 订单生成

确定每隔 180 分钟，所有的卸货点会生成 m 个订单，m 的取值范围为 0 到 10 的随机数，每个订单包含：

- 配送中心 ID；
- 卸货点 ID；
- 优先级（一般、较紧急、紧急）；
- 订单生成时间；
- 截止配送时间（依据优先级确定）。

生成一天内的数据：

```
import random
import datetime

# 设定一天的时间范围
start_time = datetime.datetime(2024, 6, 19, 0, 0, 0)
end_time = datetime.datetime(2024, 6, 19, 23, 59, 59)

# 每隔 180 分钟生成订单
time_interval = datetime.timedelta(minutes=180)

# 订单优先级及其对应的截止配送时间（小时）
priority_levels = {
    "一般": 3,
    "较紧急": 1.5,
    "紧急": 0.5
}

orders = []

current_time = start_time
while current_time <= end_time:
    for drop_off_id in range(len(drop_off_points)):
        # 随机生成 m 个订单
        m = random.randint(0, 10)
        for _ in range(m):
            # 随机选择配送中心 ID
            delivery_center_id = random.choice(range(len(delivery_centers)))
            # 随机选择优先级
            priority = random.choice(list(priority_levels.keys()))
            # 计算截止配送时间
```

```
            deadline = current_time +
datetime.timedelta(hours=priority_levels[priority])
            # 生成订单
            order = {
                "配送中心 ID": delivery_center_id,
                "卸货点 ID": drop_off_id,
                "优先级": priority,
                "订单生成时间": current_time,
                "截止配送时间": deadline
            }
            orders.append(order)
    current_time += time_interval

# 转换为 DataFrame 并显示
orders_df = pd.DataFrame(orders)
orders_df
```

生成的数据如下表所示：

	配送中心ID	卸货点ID	优先级	订单生成时间	截止配送时间
0	3	1	紧急	2024-06-19 00:00:00	2024-06-19 00:30:00
1	2	1	紧急	2024-06-19 00:00:00	2024-06-19 00:30:00
2	2	1	紧急	2024-06-19 00:00:00	2024-06-19 00:30:00
3	1	1	较紧急	2024-06-19 00:00:00	2024-06-19 01:30:00
4	4	1	较紧急	2024-06-19 00:00:00	2024-06-19 01:30:00
...	...	...	...	...	...
487	2	11	较紧急	2024-06-19 21:00:00	2024-06-19 22:30:00
488	2	11	较紧急	2024-06-19 21:00:00	2024-06-19 22:30:00
489	2	11	较紧急	2024-06-19 21:00:00	2024-06-19 22:30:00
490	1	11	一般	2024-06-19 21:00:00	2024-06-20 00:00:00
491	2	11	紧急	2024-06-19 21:00:00	2024-06-19 21:30:00

492 rows × 5 columns

三、问题分析与思路

3.1 问题分析

这是一个带有时间窗和优先级的路径规划问题，可以归类为车辆路径问题的变体，需要考虑以下几个方面：

- 多配送中心：存在多个配送中心，无人机可以从任何一个中心出发；
- 时间窗约束：每个订单都有特定的送达时间要求，无人机需要在规定时间内完成配送；
- 优先级约束：订单根据紧急程度分为不同的优先级，需要优先处理高优先级的订单；
- 容量限制：无人机一次最多只能带 n 个商品；
- 距离限制：无人机一次飞行最远路程为 20 公里（包括往返距离）；
- 速度限制：无人机有固定的飞行速度。

3.2 求解思路

考虑采用启发式策略来对候选路径集合进行修剪，同时对每一个时间段的订单采用贪心策略进行求解，算法流程如下：

1. 订单生成与预处理：
- 每隔 180 分钟生成订单，记录订单的生成时间和截止配送时间；
- 将订单按照优先级和生成时间进行排序，高优先级订单优先处理。

2. 订单分配：
- 对于当前时间段的订单进行分组，优先级高的订单必须在当前时间段内处理，优先级较低的订单选择性处理；
- 采用一个静态和动态结合的策略来处理订单，静态策略指的是在单个时间段内根据优先级处理订单，动态策略指的是在多个时间段内进行订单的合并和重新分配。

3. 路径规划：
- 对于一个订单，优先采用距离最近的配送中心的无人机进行配送（如果该无人机可以配送）；
- 合并同一条路线上的订单，根据无人机的载重进行负载均衡，保证配送距离最短；
- 通过对路径进行迭代优化，保证在满足时间窗和优先级的前提下，路径尽可能短。

4. 无人机调度：

- 对于每个选择路径，按照路径长短进行汇总并排序，选择优选集中的最优路径；
- 无人机在完成订单后返回配送中心，等待下一个订单。

通过上述思路，能够有效地对无人机配送进行规划，在满足时间窗和优先级要求的前提下，最小化总配送路径，提升配送效率。

四、算法实现

具体实现的代码如下所示：

```
# 计算两点之间的欧氏距离
def calculate_distance(point1, point2):
    return np.linalg.norm(point1 - point2)

# 路径规划函数（考虑路径聚合和路线合并）
def improved_plan_delivery_routes(orders, delivery_centers, drop_off_points,
max_items_per_drone=5, max_distance=20):
    # 使用贪婪算法生成初始路径
    routes = []
    for index, order in orders.iterrows():
        delivery_center = delivery_centers[order['配送中心 ID']]
        drop_off_point = drop_off_points[order['卸货点 ID']]
        distance_to_drop_off = calculate_distance(delivery_center,
drop_off_point)

        # 如果距离超过最大距离，无法完成配送
        if distance_to_drop_off > max_distance:
            continue

        # 计算需要几架无人机来完成订单
        num_drones = (order['数量'] + max_items_per_drone - 1) //
max_items_per_drone

        # 计算每架无人机的路径
        for drone_id in range(num_drones):
            # 简单贪婪算法：选择距离最近的卸货点
            drone_route = [delivery_center, drop_off_point, delivery_center]
            routes.append(drone_route)

    # 路线聚合和路线合并
    aggregated_routes = aggregate_routes(routes)
```

```
    merged_routes = merge_routes(aggregated_routes, max_distance,
max_items_per_drone)

    # 生成最终的路径详细信息
    final_routes = []
    for route in merged_routes:
        route_details = []
        for i in range(len(route) - 1):
            start_point = route[i]
            end_point = route[i + 1]
            delivery_center_id = find_nearest_delivery_center(start_point,
delivery_centers)
            drop_off_id = find_nearest_drop_off(end_point, drop_off_points)
            route_details.append({
                "起始点": start_point,
                "终点": end_point,
                "配送中心 ID": delivery_center_id,
                "卸货点 ID": drop_off_id
            })
        final_routes.append(route_details)

    return final_routes

# 路线聚合函数（按起始点聚合）
def aggregate_routes(routes):
    aggregated_routes = {}
    for route in routes:
        start_point = tuple(route[0])
        end_point = tuple(route[-2])  # 倒数第二个是卸货点，倒数第一个是起始点
        if (start_point, end_point) not in aggregated_routes:
            aggregated_routes[(start_point, end_point)] = []
        aggregated_routes[(start_point, end_point)].append(route)

    return aggregated_routes

# 路线合并函数（基于距离和载重考虑）
def merge_routes(aggregated_routes, max_distance, max_items_per_drone):
    merged_routes = []
    for key in aggregated_routes:
        start_point, end_point = key
        routes = aggregated_routes[key]

        total_distance = sum(calculate_distance(route[0], route[-2]) for route
in routes)
```

```
        total_items = sum(len(route) - 2 for route in routes)  # 总共的物品数量

        if total_distance <= max_distance and total_items <= len(routes) *
max_items_per_drone:
            merged_route = []
            for route in routes:
                merged_route.extend(route[1:-1])  # 不包括起始点和终点
            merged_route.insert(0, routes[0][0])  # 添加起始点
            merged_route.append(routes[-1][-2])  # 添加终点
            merged_routes.append(merged_route)
        else:
            merged_routes.extend(routes)  # 无法合并，保留原路线

    return merged_routes

# 找到最近的配送中心
def find_nearest_delivery_center(point, delivery_centers):
    min_distance = np.inf
    closest_center_id = None
    for i, center in enumerate(delivery_centers):
        distance = calculate_distance(point, center)
        if distance < min_distance:
            min_distance = distance
            closest_center_id = i
    return closest_center_id

# 找到最近的卸货点
def find_nearest_drop_off(point, drop_off_points):
    min_distance = np.inf
    closest_drop_off_id = None
    for i, drop_off in enumerate(drop_off_points):
        distance = calculate_distance(point, drop_off)
        if distance < min_distance:
            min_distance = distance
            closest_drop_off_id = i
    return closest_drop_off_id

# 生成订单
orders_df = generate_orders(delivery_centers, drop_off_points, start_time,
end_time, time_interval, priority_levels)

# 执行路径规划
final_routes = improved_plan_delivery_routes(orders_df, delivery_centers,
drop_off_points)
```

```
# 输出结果
for i, route in enumerate(final_routes):
    print(f"路径{i + 1}:")
    for step in route:
        print(f"    起始点: {step['起始点']}, 终点: {step['终点']}, 配送中心 ID:
{step['配送中心 ID']}, 卸货点 ID: {step['卸货点 ID']}")
```

算法具体实现中考虑了路径聚合：

1. 首先，通过起始点进行路线聚合，即将具有相同起始点和终点的路线合并为一个较长的路径。

2. 检查两条路线能否合并的标准包括：

- 路线的终点相同；
- 合并后的路线总距离不超过最大允许距离；
- 合并后的路线的载重不超过最大允许载重。

考虑路径聚合方法能够减少总体配送距离，提高配送效率，同时考虑了简单的距离和载重限制，确保合并后的路线仍然满足所有的配送需求和约束条件。

算法求解后得出了多条路径，包含起始点、终点、配送中心等信息：

```
路径1:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径2:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径3:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径4:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径5:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径6:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径7:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
路径8:
    起始点: [7.31993942 5.98658484], 终点: [8.32442641 2.12339111], 配送中心ID: 1, 卸货点ID: 0
    起始点: [8.32442641 2.12339111], 终点: [7.31993942 5.98658484], 配送中心ID: 1, 卸货点ID: 11
```