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：电子信息

姓　　名：谢柳

学　　号：2023282210237

任课教师：林海

2024 年 6 月 30 日

目录

无人机配送路径规划问题

一、 实验简介

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，实现自定义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。

每隔 t 分钟，所有的卸货点会生成订单（0-m 个订单），同时系统要做成决策，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、 实验环境

本次实验的编程语言是 Python3.6.10

库：random, numpy, itertools, matplotlib, seaborn 等。

三、 实验代码

3.1 实验参数参数

无人机参数：定义无人机最大携带物品数为 10；最大飞行距离为 20 公里；飞行速度为 60 公里/小时。

仿真时间参数：定义仿真时长为 600min，每个时间间隔 t 为 10min。

```
DRONE_ENDURANCE = 20/ 60# Defined the endurance of drones to 30minutes
DRONE_TRAVEL_RADIUS = 20# Defined the drone’s longest distance

# defined Speeds in miles per hour for drone
DRONE_SPEED = 60
# Defined the drone’s capacity
MAX_WEIGHT = 10

# simulated time
current_time = 0
end_time = 600
time_interval = 15
```

3.2 构建图

定义 7 个节点，节点对应的坐标存放在 city_list 中，定义 City 0 和 City 1 是配送中心，其余五个节点是顾客所在地。节点之间的距离是它们坐标之间的欧几里得距离。

```
# Created a list of arrays for storing the list of coordinates for cities
city_list = np.array([
   [0, 0],    # City 0
   [-3, -3], # City 1
   [1, -6],   # City 2
   [3, 4],    # City 3
   [-6, 3],   # City 4
   [2,1],     # City 5
   [-4, -5]   # City 6
])
```

节点之间的位置关系如下图：

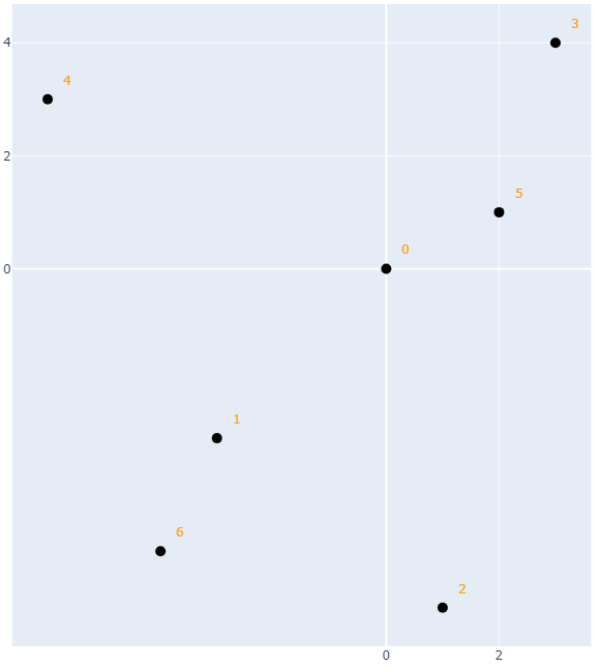

图 1: 节点之间的位置关系

3.3 生成订单

上述代码用于生成随机订单，每个订单包含订单 ID、顾客位置、订单优先级、货物重量和订单剩余时间。

```
# Initialize an order ID generator using itertools.count, which returns
    consecutive integers starting from 0
order_id_generator = itertools.count()

# A dictionary that maps priority levels to their corresponding time limits
dic = {0: 30, 1: 90, 2: 180}

# Function to generate random orders
def generate_orders():
    new_orders = [] # List to store the generated orders
    # Generate a random number of orders between 5and 9
    for _ in range(random.randint(5, 9)):
        order_id = next(order_id_generator) # Get the next order ID
        priority = random.choice([0, 1, 2]) # Randomly choose a priority (0
            or 1)
```

```
        last_time = dic[priority] # Get the corresponding time limit for the
            chosen priority
        n_list = [i for i in range(2, n)] # Generate a list of possible
            positions from 2to n-1
        pos = random.choice(n_list) # Randomly choose a position from the list
        weight = random.choice([1, 2, 3, 4]) # Randomly choose a weight
        # Append the generated order details to the new_orders list
        new_orders.append([order_id, pos, priority, weight, last_time])
    return new_orders # Return the list of generated orders
```

3.4 将订单分配到不同的配送中心

根据订单的配送地选择最近的配送中心，将订单分配到不同的配送中心，之后配送中心再对他们进行处理。

```
def order_allocation(orders, delivery_center0_order,
    delivery_center1_order):
    for order in orders:
        city = order[1]

        if(helper.euclidean_distance(city_list[city], city_list[0]) <=
            helper.euclidean_distance(city_list[city], city_list[1])):
            delivery_center0_order.append(order)
        else:
            delivery_center1_order.append(order)
    return delivery_center0_order, delivery_center1_order
```

3.5 配送中心选择当前需配送订单

get_orders_delivered 用于处理订单并确定需要多少无人机来完成配送。其主要步骤如下：

检查是否有订单，如果没有则返回原始订单和 0 个无人机。

按剩余时间对订单进行升序排序。

找出所有剩余时间小于等于 30 分钟的订单。

初始化一个列表 all_drone_tour_cities 来存储每个无人机的配送路线。

遍历优先订单，根据重量和配送距离决定是否需要添加新的无人机。如果需要添加新的无人机，则记录当前的配送路线，并重置相关变量以开始新一轮决策。对于紧急订单，无人机一定会派无人机立即配送，并且总路程小于 20km，根据无人机的飞行速度可以知道，紧急订单一定会在规定时间内送达。

将剩余的城市添加到配送路线中。

使用最近邻算法生成并打印每个无人机的初始配送路线。

更新剩余订单的时间。

返回剩余订单和所需的无人机数量。

```
def get_orders_delivered(orders, delivery_center, time_interval):
    # If there are no orders, return the original orders and 0drones needed
    if not orders:
        return orders, 0

    drone_num = 0
    # Sort the orders by their remaining time in ascending order
    orders.sort(key=lambda x: x[4])

    # Find all orders with a remaining time of 30minutes or less
    priority_orders = [order for order in orders if order[-1] <= 30]

    all_drone_tour_cities = []

    cities = []
    cities.append(delivery_center)
    cur_weight = 0
    k = len(priority_orders)
    i = 0

    # Iterate through the priority orders
    while i < k:
        cur_weight += priority_orders[i][3] # Add the order's weight to the
            current weight
        if priority_orders[i][1] not in cities:
            cities.append(priority_orders[i][1]) # Add the order's destination
                city if not already in the list
```

```
        tmp_tour = nearest_neighbor(cities) # Generate a tour using the
            nearest neighbor algorithm

        # Check if a new drone is needed based on weight or tour distance
        if cur_weight > MAX_WEIGHT or cal_tour_distance(tmp_tour) > 20:
            i -= 1# Step back one order

            if delivery_center not in cities:
                cities.insert(0, delivery_center) # Ensure delivery center is
                    the starting point
            if len(cities[:-1]) > 1:
                all_drone_tour_cities.append(cities[:-1]) # Save the current
                    cities list excluding the last one

            cities = [delivery_center] # Reset cities list with the delivery
                center
            cur_weight = 0# Reset current weight

        i += 1

    # Add remaining cities to the tours if any
    if delivery_center not in cities:
        cities.insert(0, delivery_center)
    if len(cities) >= 1:
        all_drone_tour_cities.append(cities)
    print("all_drone_tour_cities", all_drone_tour_cities)

    # Generate and print initial tours for each set of cities
    for cs in all_drone_tour_cities:
        cities = list(set(cs))
        initial_tour = nearest_neighbor(cities)
        print("The delivery center is: ", delivery_center)
        print("Initial tour using nearest neighbor, ", initial_tour)

    # Update the remaining orders and their times
    orders = orders[k:]
    for order in orders:
```

```
        order[-1] -= time_interval

    drone_num = len(all_drone_tour_cities) # Number of drones needed
    return orders, drone_num
```

3.6 配送中心选择配送路径

基于贪心算法构建配送路线，需要注意的是，路径的起点和终点都是对应的配送中心。

```
# Function to construct an initial tour using the Nearest Neighbor method.
def nearest_neighbor(cities):
    if(len(cities) == 0):
        return []
    unvisited_cities = deepcopy(cities)
    start_city = cities[0]
    current_city = start_city
    tour = [current_city]


    while len(unvisited_cities) > 1:
        unvisited_cities.remove(current_city)
        nearest_city = min(unvisited_cities, key=lambda city:
            helper.euclidean_distance(city_list[current_city],
            city_list[city]))
        tour.append(nearest_city)
        current_city = nearest_city

    # Add the starting city again to complete the tour
    tour.append(start_city)
    return tour
```

3.7 实验结果

第一行是随机生成的订单，第二行是配送中心 0 号在本次的派出的无人机对应需要经过的城市，可以看到，0 号配送中心要排出两台无人机，分别经过 0-4-0 和 0-3-0，不能同时配送 3、4 号地点订单的原因是 0-3-4-0 的距离之和大于 20，所以只能用两台无人机分别配送。

```
订单:  [[0, 4, 0, 3, 30], [2, 3, 0, 2, 30], [5, 4, 2, 2, 180], [6, 5, 2, 3, 180]] [[1, 2, 1, 2, 90], [3, 6, 0, 1, 30], [4, 6, 0, 2, 30]]
all_drone_tour_cities [[0, 4], [0, 3]]
The delivery center is:  0
Initial tour using nearest neighbor,  [0, 4, 0]
The delivery center is:  0
Initial tour using nearest neighbor,  [0, 3, 0]
all_drone_tour_cities [[1, 6]]
The delivery center is:  1
Initial tour using nearest neighbor,  [1, 6, 1]
------------------------------------------------------------
```

图 2: t 时刻生成的订单和配送中心的决策

下图是 0-5-4-0 路径的可视化图：

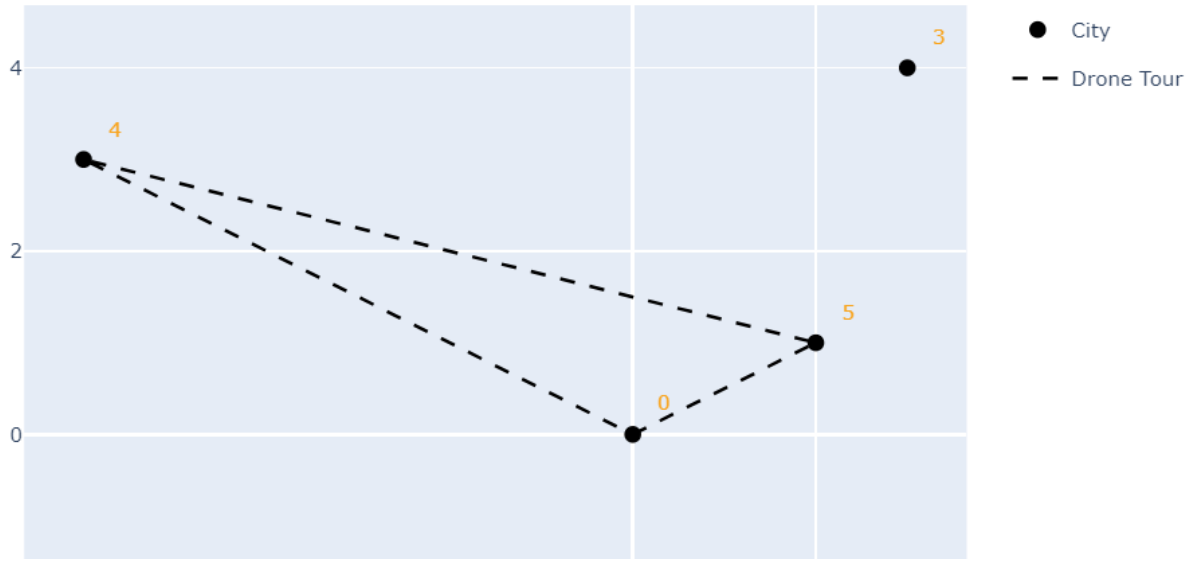

图 3: 无人机运行轨迹示意图

四、 总结

本实验的目标是设计一个算法来解决无人机在指定区域内的配送路径规划问题。该区域有多个配送中心和多个卸货点，每个卸货点会随机生成不同优先级别的订单。无人机需要在满足订单优先级别要求的前提下，最短路径地完成配送任务。

本次实验使用一个全局订单队列保存所有未完成的订单，并根据优先级和剩余时间排序。

采用最近邻算法为无人机规划路径，对当前所有订单按剩余配送时间从小到大排序，优先处理紧急订单。初始化一个无人机的当前负载和路径列表，从最近的配送中心出发。遍历排序后的订单，将符合重量和距离限制的订单依次添加到当前无人机的路径中。如果当前无人机无法再添加订单，则记录当前路径并派出新无人机继续处理剩余订单。

每隔 t 分钟，系统根据当前订单队列和无人机状态做出决策，确定派出多少无人机以及每个无人机的路径。对于非紧急订单，可以选择暂时不处理，等待下一次决策时合并处理。

本次实验让我学习了如何模拟无人机配送系统，处理订单生成、优先级排序、路径规划和系统决策等一系列实际问题。在代码实现过程中遇到了一些问题，通过调试和优化，增强了我对于算法在实际物流配送中应用的理解，提高了我的代码能力。