武汉大学课程作业

《无人机配送路径规划问题》

院 系 名 称 ： 国家网络安全学院

专 业 名 称 ： 网络空间安全

课 程 名 称 ： 高级算法设计与分析

姓 名 ： 吕芸娜

学 号 ： 2023202210049

指 导 教 师 ： 林海

二〇二四年六月

目录

第一章 问题描述

1. 问题描述

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，实现区域内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。

2. 实例设置

（1）基本参数

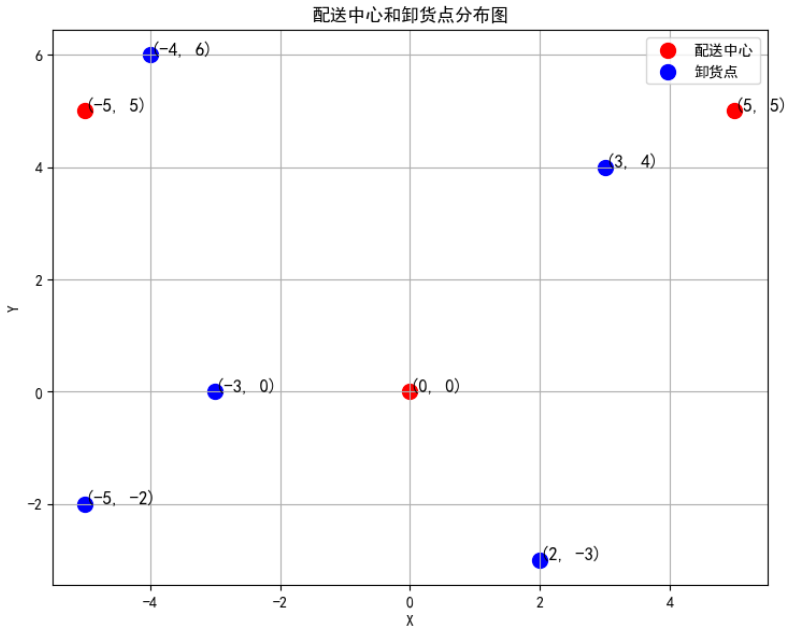
第三章的代码实现中的基本参数如下所示：

配送中心数 j	3 个
卸货点数 k	5 个
决策时间间隔 t	10 分钟
卸货点生成订单数 m	0-5 个
总时长	1 天
无人机容量 n	3 个
无人机一次最远距离	20 公里
无人机速度	60 公里/小时

（2）地理位置

配送中心和卸货点的坐标及分布如下所示：

配送中心坐标	(0, 0), (5, 5), (-5, 5)
卸货点坐标	(2, -3), (-4, 6), (3, 4), (-5, -2), (-3, 0)



各点（包括配送中心和卸货点）的距离如下所示：

	(0, 0)	(5, 5)	(-5, 5)	(2, -3)	(-4, 6)	(3, 4)	(-5, -2)	(-3, 0)
(0, 0)	0.00	7.07	7.07	3.61	7.21	5.00	5.39	3.00
(5, 5)	7.07	0.00	10.00	8.54	9.06	2.24	12.21	9.43
(-5, 5)	7.07	10.00	0.00	10.63	1.41	8.06	7.00	5.39
(2, -3)	3.61	8.54	10.63	0.00	10.82	7.07	7.07	5.83
(-4, 6)	7.21	9.06	1.41	10.82	0.00	7.28	8.06	6.08
(3, 4)	5.00	2.24	8.06	7.07	7.28	0.00	10.00	7.21
(-5, -2)	5.39	12.21	7.00	7.07	8.06	10.00	0.00	2.83
(-3, 0)	3.00	9.43	5.39	5.83	6.08	7.21	2.83	0.00

第二章 算法实现

1. 订单与无人机的模型设计

（1）订单模型：需要记录订单的卸货点、订单的优先级、订单配送剩余时间（需要在剩余时间内将订单送达卸货点）以及订单是否送达目的地。

（2）无人机模型：需要记录无人机对应的配送中心，无人机的路径，无人机的当前负载（确保增加物品时不会超载）以及无人机的路径距离。

2. 订单生成

每隔 t 分钟，算法会为每个卸货点随机生成 0~m 个订单，每个订单具有随机的紧急程度和剩余时间。例如，当紧急程度为“紧急”时，剩余时间设为 30。当规划路径时，若当前轮次没有将该物品运送至卸货点，剩余时间需要减去 t。算法需要保证在剩余时间内可以将订单成功送达卸货点。

3. 路径规划

本题中寻找最小路径的问题可以看作旅行商回路问题。因此，可以采用 Prim 近似算法来得到一条最短路径作为无人机的配送路径。具体地，选择配送中心作为起点，将其加入到已访问节点集合，所有卸货点作为未访问节点集合。之后，选择已访问节点和未访问节点中距离最短的点加入到已访问节点集合（最终路径）中，直到所有卸货点都被访问，返回配送中心。算法如下：

算法 1 旅行商的近似算法

```
输入：G = (V, E)
输出：旅行商回路
T ← Prim 算法得到 G 的最小生成树;
H ← 对 T 进行先序遍历;
return H;
```

4. 配送决策

（1）首先对订单的紧急程度（剩余时间）进行排序。

（2）对于紧急订单（剩余时间少于一定阈值），必须在当前周期内安排配送。具体来说，设置阈值为当前到下一轮配送的时间间隔（t 分钟）加无人机配送的最大时间（20 分钟），当订单的剩余时间少于 20 + t 分钟时，配送该订单。在配送时，当将该订单加入已有无人机的最短配送路径小于等于新增一架无人机进行配送的最短路径时，利用已有的无人机配送该订单；否则新增一架无人机进行配送。

（3）对于非紧急订单，根据无人机的当前负载和剩余容量，决定是否在当前周期内配送或推迟到下一个周期。若加入该非紧急订单后，无人机不会超负载且距离不会超过 20 公里，则配送该物品，利用 Prim 算法寻找一条近似的最短旅行商回路。

算法如下：

算法 2 配送决策

输入：订单 order，无人机 drone
输出：最短路径 total_distance
按订单的剩余时间从小到大排序得到 order;
while order:　# 遍历订单
　if order.remain_time ≤ 20 + t:
　　选择一架 drone.current_load < n 的无人机，使用算法 1 计算旅行商回路，使得旅行商回路距离最小且不超过 20 公里，更新 drone.path 和 drone.distance；否则新增一架无人机;
　else:
　　选择一架 drone.current_load < n 的无人机，使用算法 1 计算旅行商回路，使得旅行商回路距离最小且不超过 20 公里，更新 drone.path 和 drone.distance；否则将订单剩余时间减 t，继续遍历下一个订单;
while drone:　# 遍历无人机
　total_distance += drone.distance;
return total_distance;

5. 算法复杂度分析

订单生成的算法复杂度取决于卸货点数量 k 和订单生成数 m，复杂度为 $O(km)$，路径规划的复杂度取决于 Prim 算法的复杂度，即 $O(k^2 logk)$，配送决策的复杂度取决于订单排序时间复杂度 $O(kmclog(kmc))$（c 为迭代次数）以及订单数、无人机数和 Prim 算法的复杂度 $O(kmcd \cdot k^2 logk)$（d 为无人机数）。所以，总复杂度为 $O(k^3 mcdlogk)$。

第三章 代码实现

1. 订单类

```
class Order:
    def __init__(self, id, x, y, remain_time, priority):
        self.id = id
        self.x = x
        self.y = y
```

```
        self.priority = priority
        self.remain_time = remain_time
        self.flag = False

    def __lt__(self, other):
        return self.remain_time > other.remain_time     # 剩余时间少的在前
```

2. 无人机类

```
class Drone:
    def __init__(self, home_distribution_center):
        self.capacity = 3                   # 无人机一次最多携带的物品数量
        self.max_distance = max_dist        # 最大飞行距离(公里)
        self.speed = 60                     # 速度(公里/小时)
        self.home_distribution_center = home_distribution_center    # 所属配送中心
        self.path = []              # 配送路径 旅行商回路
        self.current_load = 0       # 当前负载
        self.current_distance = 0   # 当前距离
```

判断是否可以继续装入物品：

```
def can_carry(self):
    return self.current_load < self.capacity
```

使用 Prim 算法找到最小的回路：

```
def prim_algorithm(self, pth):
    all_points = list(set(pth))     # 去除重复元素, 但注意 load 得+1
    # 从配送中心开始应用 prim 算法
    start = self.home_distribution_center
    visited = {start}
    unvisited = set(all_points)
    path = []
    while len(visited) <= len(all_points):  # visited 比 all_points 多个配送中心 所以有=
        # 找 visited 和 unvisited 的距离最小的点
        min_dist = 99999
        for point in visited:
            for un_point in unvisited:
                dist = distance(un_point, point)
                if dist < min_dist:
                    min_dist = dist
                    visit_point = un_point  # 该访问这个节点了
        path.append(visit_point)
        visited.add(visit_point)
        unvisited.remove(visit_point)
    return path
```

根据 Prim 算法得到路径和距离，保留符合条件（距离小于 20 公里且不超载）的路径和距离：

```
def path_and_dist(self, pth):
    pth = self.prim_algorithm(pth)  # TSP
    dist = sum(distance(pth[i], pth[i + 1]) for i in range(len(pth) - 1))
    dist = dist + distance(pth[0], self.home_distribution_center) + distanc
e(pth[len(pth) - 1], self.home_distribution_center)    # 回到原点
    return pth, dist

def test_order(self, order):    # 这里只用来计算路径和长度，没有真的更新无人机
(update_order)
    if self.can_carry():
        pth = self.path + [(order.x, order.y)]   # 直接添加这个点
        # 计算当前路径和距离
        pth, dist = self.path_and_dist(pth)     # 计算最短路径，先不更新
self.path 和 self.distance
        if dist <= max_dist:
            return pth, dist
        else:   # 距离太大
            return [], 0
    return [], 0    # 装不下了
```

更新负载、路径和距离：

```
def update_order(self, pth, dist):      # 确定是最小才更新无人机的运送情况
    self.current_load += 1
    self.path = pth
    self.current_distance = dist
```

3. 计算两点间距离

```
def distance(p1, p2):
    distance = math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
    return distance
```

4. 随机生成 0~m 个订单

```
def generate_orders(current_time):
    orders = []
    print(f"时间 {current_time}: 生成的订单信息如下:")
    for up in unloading_points:
        num_orders = random.randint(0, m)   # 为每个卸货点生成 0-m 个订单
        x, y = up                           # 卸货点的坐标
        if num_orders != 0:
            print(f"卸货点: {up}")
        for i in range(num_orders):
            remain_time = random.choice([30, 90, 180])  # 随机生成该订单的紧
急程度
```

```
            if remain_time == 30:
                priority = '紧急'
            elif remain_time == 90:
                priority = '较紧急'
            elif remain_time == 180:
                priority = '一般'
            order = Order(i, x, y, remain_time, priority)
            orders.append(order)
            print(f"订单 ID: {order.id}, 优先级: {order.priority}, 剩余时间: {order.remain_time}")
    return orders
```

5. 配送决策

剩余时间不足 t + 20 分钟的订单必须在本轮送出去；此后，对于剩下的订单，如果无人机还有空位且最短路径不会超过 20 公里的订单可以此轮送走：

（1）对于剩余时间不足 t + 20 分钟的订单：如果还没有无人机，可以找一个离订单最近的配送中心，新建一架无人机；如果有无人机，对比是加入已有无人机中距离最短，还是找一个离订单最近的配送中心，然后新建一架无人机的往返距离最短，选择距离最短的方案。设置 flag = True 代表该订单完成。

（2）对于剩下的订单：尽量保证无人机是装满的，且满足距离不超过 20 公里，以保证订单可以尽早送达目的地。如果有无人机，可以考虑是否把订单加入到这几个无人机中。如果装入无人机，设置 flag = True 代表该订单完成。如果不装入无人机，需要将剩余时间减去 t 分钟。

```
def greedy_delivery(orders, current_time):
    total_distance = 0
    drones = []
    # 按剩余时间对订单进行排序
    orders.sort(reverse=True)
    for order in orders:
        if order.remain_time <= 30:
            # 如果还没有无人机，找一个离订单最近的配送中心，新建一架无人机；
            if len(drones) == 0:
                nearest_dc = min(distribution_centers, key=lambda dc: distance(dc, (order.x, order.y)))
                new_drone = Drone(nearest_dc)  # 从最近的配送中心生成新的无人机
                min_dist = 2 * distance(nearest_dc, (order.x, order.y))
                new_drone.update_order([(order.x, order.y)], min_dist)
                drones.append(new_drone)
```

```
15.             # 如果有无人机，对比是加入已有无人机中距离最短，还是找一个离订单最近的配送中心，新建一架无人机往返的距离最短
16.             else:
17.                 min_dist, ind = 999999, -1
18.                 best_pth = []
19.                 for i, drone in enumerate(drones):
20.                     pth, dist = drone.test_order(order)
21.                     if dist != 0:
22.                         if dist < min_dist:
23.                             min_dist = dist
24.                             ind = i
25.                             best_pth = pth
26.                 nearest_dc = min(distribution_centers, key=lambda dc: distance(dc, (order.x, order.y)))
27.                 new_drone = Drone(nearest_dc)
28.                 dist = 2 * distance(nearest_dc, (order.x, order.y))
29.                 if dist < min_dist:
30.                     min_dist = dist
31.                     new_drone.update_order([(order.x, order.y)], dist)
32.                     drones.append(new_drone)
33.                 else:
34.                     drones[ind].update_order(best_pth, min_dist)
35.             order.flag = True
36.
37.         # 剩下的订单：尽量保证无人机是装满的，且满足 20km 的前提：如果有无人机，可以考虑是否把订单加入到这几个无人机中；
38.         # 没有就算了
39.         # 不装走的话，remain_time - t (间隔时间)
40.         else:
41.             min_dist, ind = 999999, -1
42.             best_pth = []
43.             for i, drone in enumerate(drones):
44.                 pth, dist = drone.test_order(order)
45.                 if dist != 0:
46.                     if dist < min_dist:
47.                         min_dist = dist
48.                         ind = i
49.                         best_pth = pth
50.             if ind != -1:   # 如果找到了可以装这个物品的无人机
51.                 drones[ind].update_order(best_pth, min_dist)
52.                 order.flag = True
53.             else:
54.                 order.remain_time -= t
55.
```

```
    # 最后计算所有无人机的 dist
    tot_dist = 0
    for i, drone in enumerate(drones):
        tot_dist += drone.current_distance
        print(f"{i + 1}号无人机配送情况：配送中心：{drone.home_distribution_center}，"
              f"路径：{drone.path}，路径长度：{drone.current_distance}，物品数：{drone.current_load}")

    return tot_dist
```

6. 主函数

```
def main():
    all_orders = []     # 未解决的订单
    total_distance = 0  # 总路径长度
    current_time = 0    # 当前时刻

    while current_time < total_time:
        # 每隔 t 分钟生成一次订单，进行一次决策
        new_orders = generate_orders(current_time)
        all_orders.extend(new_orders)

        total_distance += greedy_delivery(all_orders, current_time)

        # 移除已经完成的订单
        all_orders = [order for order in all_orders if order.flag == False]

        current_time += t

    print(f"\n 总路径长度：{total_distance}")
```

第四章 实验结果

1. 运行结果

在第一章的基本参数设置下运行代码，得到如下结果：

```
时间 0: 生成的订单信息如下:
卸货点: (2, -3)
订单ID: 0, 优先级: 紧急, 剩余时间: 30
订单ID: 1, 优先级: 一般, 剩余时间: 180
卸货点: (-4, 6)
订单ID: 0, 优先级: 紧急, 剩余时间: 30
订单ID: 1, 优先级: 一般, 剩余时间: 180
订单ID: 2, 优先级: 一般, 剩余时间: 180
订单ID: 3, 优先级: 较紧急, 剩余时间: 90
卸货点: (-5, -2)
订单ID: 0, 优先级: 一般, 剩余时间: 180
订单ID: 1, 优先级: 一般, 剩余时间: 180
订单ID: 2, 优先级: 紧急, 剩余时间: 30
订单ID: 3, 优先级: 一般, 剩余时间: 180
订单ID: 4, 优先级: 一般, 剩余时间: 180
卸货点: (-3, 0)
订单ID: 0, 优先级: 较紧急, 剩余时间: 90
订单ID: 1, 优先级: 一般, 剩余时间: 180
订单ID: 2, 优先级: 一般, 剩余时间: 180
1号无人机配送情况: 配送中心: (0, 0), 路径: [(2, -3), (-5, -2)], 路径长度: 16.061783894463968, 物品数: 3
2号无人机配送情况: 配送中心: (-5, 5), 路径: [(-4, 6)], 路径长度: 2.8284271247461903, 物品数: 3
3号无人机配送情况: 配送中心: (0, 0), 路径: [(-3, 0), (-5, -2)], 路径长度: 11.213591931880693, 物品数: 3

...

卸货点: (-4, 6)
订单ID: 0, 优先级: 较紧急, 剩余时间: 90
订单ID: 1, 优先级: 一般, 剩余时间: 180
订单ID: 2, 优先级: 一般, 剩余时间: 180
订单ID: 3, 优先级: 紧急, 剩余时间: 30
卸货点: (3, 4)
订单ID: 0, 优先级: 较紧急, 剩余时间: 90
订单ID: 1, 优先级: 较紧急, 剩余时间: 90
订单ID: 2, 优先级: 较紧急, 剩余时间: 90
订单ID: 3, 优先级: 一般, 剩余时间: 180
卸货点: (-5, -2)
订单ID: 0, 优先级: 一般, 剩余时间: 180
卸货点: (-3, 0)
订单ID: 0, 优先级: 紧急, 剩余时间: 30
1号无人机配送情况: 配送中心: (0, 0), 路径: [(-5, -2)], 路径长度: 10.770329614269007, 物品数: 3
2号无人机配送情况: 配送中心: (0, 0), 路径: [(-3, 0), (2, -3)], 路径长度: 12.43650317030929, 物品数: 3
3号无人机配送情况: 配送中心: (0, 0), 路径: [(2, -3)], 路径长度: 7.211102550927978, 物品数: 3
4号无人机配送情况: 配送中心: (-5, 5), 路径: [(-4, 6), (-5, -2)], 路径长度: 16.476471310671645, 物品数: 3

总路径长度: 5577.212181448566
```

得到一天内的无人机最短路径长度为 5577.21。

2. 实例解释

以第一个时刻的结果为例进行解释：

在第一个实例中，一共有四个卸货点生成了订单，紧急订单的剩余时间为30分钟。(2, -3) 生成了两个订单，其中ID = 0的订单是紧急订单，需要先处理，为其分配1号无人机。(-4, 6) 生成了四个订单，ID = 0的订单是紧急订单，为其分配2号无人机。(-5, 2) 生成了五个订单，ID = 2的订单是紧急订单，为其分配3号无人机。对于 (-4, 6) 的ID = 2的较紧急订单，将其加入到2号无人机可以达到最短距离最小，故将其加入2号无人机。对于 (-3, 0) 的ID = 0的较紧急订单，将其加入 3 号无人机可以达到最短距离最小，故将其加入 3 号无人机。以此类推，直到所有无人机都装满，或者遍历完所有订单，得到 1 号无人机的路径为 (0, 0) → (2, -3) → (-5, -2) → (0, 0)，路径长度为16.06；2号无人机的路径为 (-5, 5) → (-4, 6) → (-5, 5)，路径长度为2.83；3号无人机路径为 (0, 0) → (-3, 0) → (-5, -2) → (0, 0)，路径长度为11.21。

第五章 总结

在实验过程中，可以将路径的选取看作旅行商回路问题，通过应用 Prim 算法可以得到一个有效的近似解。除此之外，本实验的难点在于如何满足各种约束条件，即需要考虑订单优先级、无人机最远配送距离和无人机装载上限等问题。本算法通过引入一个阈值 t + 20 来决定是否要在本轮配送订单，并利用贪心算法的思想，对于剩余时间超过阈值的物品，判断其是否可以由本轮的无人机配送（不超过负载且不超过最远距离），尽可能地充分利用无人机来达到近似效果，并保证订单可以尽早送达。