

武汉大学国家网络安全学院

实验报告

课程名称 高级算法设计

专业年级 网络空间安全 2023 级

姓 名 袁子昕

学 号 2023202210087

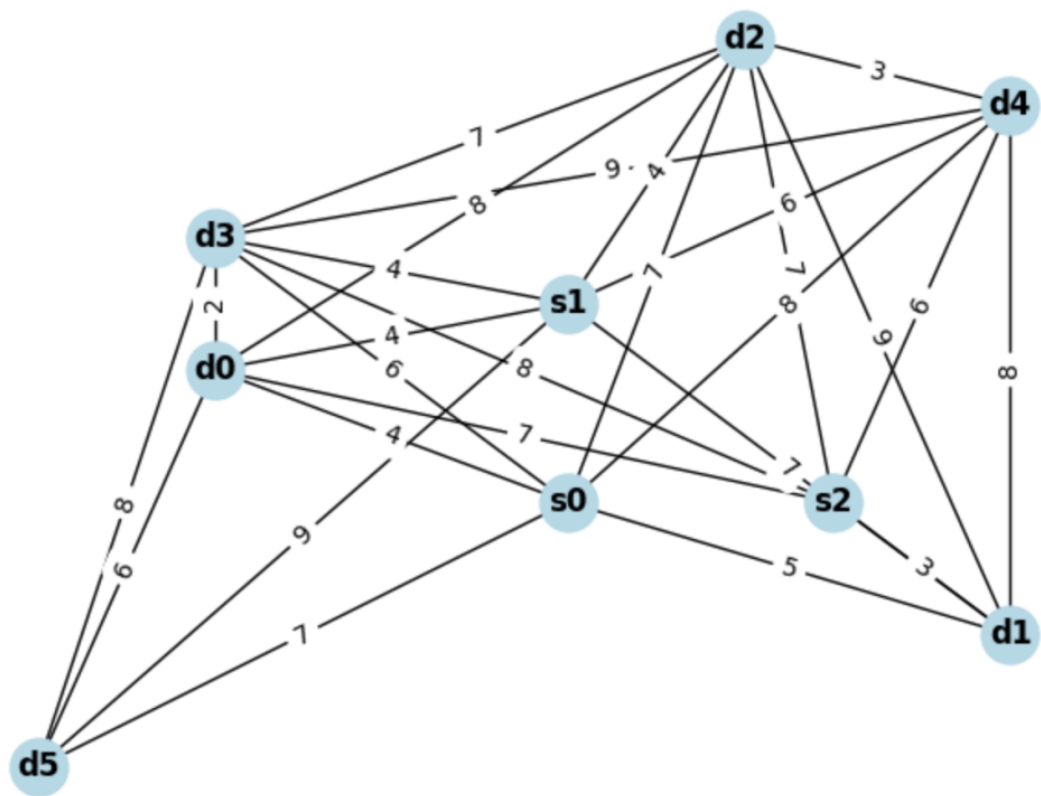
学 期 2023-2024 学年 第二 学期

填写时间 2024 年 6 月 22 日

一、场景初始化

根据要求，本算法要实现一个区域内无人机配送路径规划问题，首先将配送中心和卸货点抽象为一个图中的不同节点，由于无人机一次飞行最远路程为 20 公里，因此在每个卸货点 10 公里内必然有一个配送中心，如下图所示，该图满足以下几个条件：

- 图中 s_n 为配送中心， d_n 为卸货点，共设置了 3 个配送中心和 6 个卸货点；
- 每个卸货点都有能够完成配送的配送中心；
- 边的权重为两点之间的距离（km），只存在距离小于等于 10km 的边。



图生成的代码如下所示，使用了 `networkx` 包，根据初始化给定的配送中心坐标字典 `centers` 和卸货点坐标字典 `points`，先生成节点，再根据点坐标计算出配送中心到达每一个卸货点的边以及卸货点之间的边。在添加边时，有两种边不加入图中：

- 超过 10km 的边不添加：因为此时无人机已经不能完成一个往返了；
- 配送中心之间的边不添加：因为每个配送中心都有无限量的无人机和可满足订单的货物。

```
Python 复制代码

1 def cal_distance(point1, point2):
2     """
3     计算两个点之间的欧几里得距离，这样能保证满足三角不等式
4     :param point1: 第一个点的坐标，形式为(x, y)的元组
5     :param point2: 第二个点的坐标，形式为(x, y)的元组
6     :return 两个点之间的欧几里得距离，取整数
7     """
8     distance = ((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2) **
0.5
9     return round(distance)
10
11
12 def init_graph(centers, points):
13     """
14     初始化图结构，并根据给定的中心点和点集构建图。
15     :param centers: 配送中心的字典，{中心标识: 中心坐标}
16     :param points: 卸货点的字典，{点标识: 点坐标}
17     :return G: 构建好的图
18     """
19     # 创建一个空的无向图
20     G = nx.Graph()
21
22     # 添加配送中心和卸货点节点
23     for node, pos in centers.items():
24         G.add_node(node, pos=pos)
25     for node, pos in points.items():
26         G.add_node(node, pos=pos)
27
28     # 添加配送中心和卸货点之间的边（假设所有配送中心都可以向所有卸货点配送货物）
29     for center in centers:
30         for point in points:
31             # 计算配送中心到卸货点的距离
32             distance = cal_distance(centers[center], points[point])
33             if distance < 10:
34                 G.add_edge(center, point, weight=distance)
35
36     for point1 in points:
37         for point2 in points:
38             if point1 != point2:
39                 # 计算卸货到卸货点的距离
40                 distance = cal_distance(points[point1], points[point2])
41                 if distance < 10:
42                     G.add_edge(point1, point2, weight=distance)
43
```

```

44     # 绘制图
45     if plt.get_fignums():
46         plt.close('all') # 关闭所有打开的图像，避免干扰
47         pos = nx.get_node_attributes(G, 'pos')
48         nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=500, font
_weight='bold')
49
50         labels = nx.get_edge_attributes(G, 'weight')
51         nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
52         plt.show()
53
54     return G

```

二、卸货点生成订单

每个卸货点每隔时间 t 会生成一批订单，每个订单有自己的优先级：

- 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。

因此设计 Order 类用于表示订单：

Python | 复制代码

```

1 class Order:
2     """
3     订单类
4     order_id: 订单id
5     deadline: 订单需要在截止时间前送达
6     priority: 订单优先级
7     * 0: 紧急, 0.5小时内配送到
8     * 1: 较紧急, 1.5小时内配送到
9     * 2: 一般, 3小时内配送到即可
10    point_id: 订单目的地
11    """
12    times = [30, 90, 180] # 静态变量, 优先级对应的时间
13
14    def __init__(self, current_time, order_id, priority, point_id):
15        self.order_id = order_id # 分配唯一的订单ID
16        self.deadline = current_time + Order.times[priority]
17        self.priority = priority
18        self.point_id = point_id

```

并设计了每个卸货点处的订单生成函数，每个卸货点会随机生成 1-5 个订单，每个订单的优先级也是随机生成的，每个订单的订单目的地即为当前卸货点。

```
Python | 复制代码

1 def simulate_order_generation(points, minute):
2     """
3     模拟订单生成。
4     :param points: 需要生成订单的点的列表
5     :param minute: 订单生成的时刻（也就是current_time）
6     :return: 一个包含所有生成订单的列表
7     """
8     random.seed() # 初始化随机数生成器
9     orders = []
10    for point in points:
11        for i in range(random.randint(1, 5)): # 随机生成1到5个订单
12            new_order = Order(minute, f"order_{minute}_{i}", random.randint(0,
13                                2), point)
14            orders.append(new_order)
15    return orders
```

三、决策函数

决策函数用于决策每个配送中心的应该派出的无人机数量以及配送的订单，并完成每个无人机的路径规划，函数定义如下。

```
Python | 复制代码

1 def make_decisions(G, orders, current_time, max_flight_distance=20, drone_capacity=3):
2     """
3     根据给定的图形G、订单列表和无人机参数，决定无人机的分配和飞行路线。
4     :param G: 图形，表示配送区域中配送中心和卸货点之间的关系
5     :param orders: 订单列表，每个订单包含截止时间、优先级和配送点信息
6     :param max_flight_distance: 无人机的最大飞行距离，默认为20
7     :param drone_capacity: 无人机的载货量，默认为3
8     :param current_time: 当前时间
9     :return drone_assignments: 无人机分配字典，键为配送中心，值为分配给该中心的订单列表。
10    :return reserved_orders: 由于超出了无人机能力而保留的订单列表。
11    """
```

我们首先根据订单的时效和优先级对订单进行排序，然后将订单分配到最近的配送中心。在分配订单时，我们先将订单分配给离目的地最近的配送中心。

```

1  # 初始化无人机的分配
2  drone_assignments = {center: [] for center in G.nodes() if "s" in center}
3  reserved_orders = []
4
5  # 对订单按照时效和优先级排序
6  orders.sort(key=lambda x: (x.deadline - current_time, x.priority))
7
8  # 分配订单到最近的配送中心
9  for order in orders:
10     shortest_distance = float('inf')
11     selected_center = None
12     for center in G.nodes():
13         if "s" in center:
14             if G.has_edge(center, order.point_id):
15                 distance = G[center][order.point_id]["weight"]
16                 if distance < shortest_distance:
17                     shortest_distance = distance
18                     selected_center = center
19     drone_assignments[selected_center].append(order)

```

然后，我们根据无人机的容量和订单数量决定派出多少架无人机，并为每架无人机规划一条路线，确保订单在其时效内配送完成。路径规划使用了 `networkx` 包中的 Christofides 算法，它是一个近似解法，可能不会找到绝对最优解，但在实际应用中通常能提供足够好的解决方案。首先将配送中心以及其分配到的订单中的目的地作为节点，建立一张子图，在这个子图中求出 TSP 路径，即为该无人机的飞行路径。若规划的路径超过了无人机的最大飞行路径，则驱逐订单列表中 deadline 最晚优先级最低的订单，留到下一次再派送。

```

1  for center, orders in drone_assignments.items():
2      num_drones = (len(orders) + drone_capacity - 1) // drone_capacity # 向上取整
3      drone_routes = []
4
5      # 为每架无人机规划路线
6      for i in range(num_drones):
7          drone_orders = orders[i * drone_capacity:(i + 1) * drone_capacity]
8          # 创建一个子图，只包含配送中心和需要配送的卸货点
9          sub_G = G.copy()
10         for node in list(sub_G.nodes()):
11             if node not in [center] + [order.point_id for order in drone_orders]:
12                 sub_G.remove_node(node)

```

```

14         # 使用Christofides算法找到近似最优路径
15         tsp_path = traveling_salesman_problem(sub_G, cycle=True)
16         tsp_path_length = sum(sub_G[tsp_path[i]][tsp_path[i + 1]]['weight'] for i
    in range(len(tsp_path) - 1))
17     if tsp_path_length > max_flight_distance:
18         # 如果路径长度超过最大飞行距离，则删除一个优先级最小的订单并重新规划路径
19         orders.sort(key=lambda x: (x.deadline - current_time, x.priority), re
    verse=True)
20         reserved_orders.append(orders.pop())
21         tsp_path = traveling_salesman_problem(sub_G, cycle=True)
22         tsp_path_length = sum(sub_G[tsp_path[i]][tsp_path[i + 1]]['weight'] f
    or i in range(len(tsp_path) - 1))
23         drone_routes.append((drone_orders, tsp_path, tsp_path_length))
24
25     print(f"配送中心 {center} 将派出 {num_drones} 架无人机。")
26     idx = 1
27     for drone_orders, tsp_path, tsp_path_length in drone_routes:
28         print(
29             f"无人机{idx}将配送以下订单: {'', '.join([order.order_id for order in dron
    e_orders])}\n"
30             f"路径为: {tsp_path}\n路径长度为: {tsp_path_length}km\n")
31         idx += 1
32

```

四、主函数

主函数首先设置了配送中心和配送点的坐标位置，并调用 `init_graph()` 初始化区域图。接着使用 `while` 循环模拟每隔 10 分钟生成订单并进行决策。决策后会返回无人机分配列表以及保留未配送的订单，将订单添加到 `orders` 列表中参加下一轮决策。

```

1 def main():
2     # 定义配送中心和卸货点的位置
3     centers = {
4         "s0": (0, 0),
5         "s1": (0, 3),
6         "s2": (3, 0)
7     }
8
9     points = {
10         "d0": (-4, 2),
11         "d1": (5, -2),
12         "d2": (2, 7),
13         "d3": (-4, 4),
14         "d4": (5, 6),
15         "d5": (-6, -4)
16     }
17
18     G = init_graph(centers, points)

```

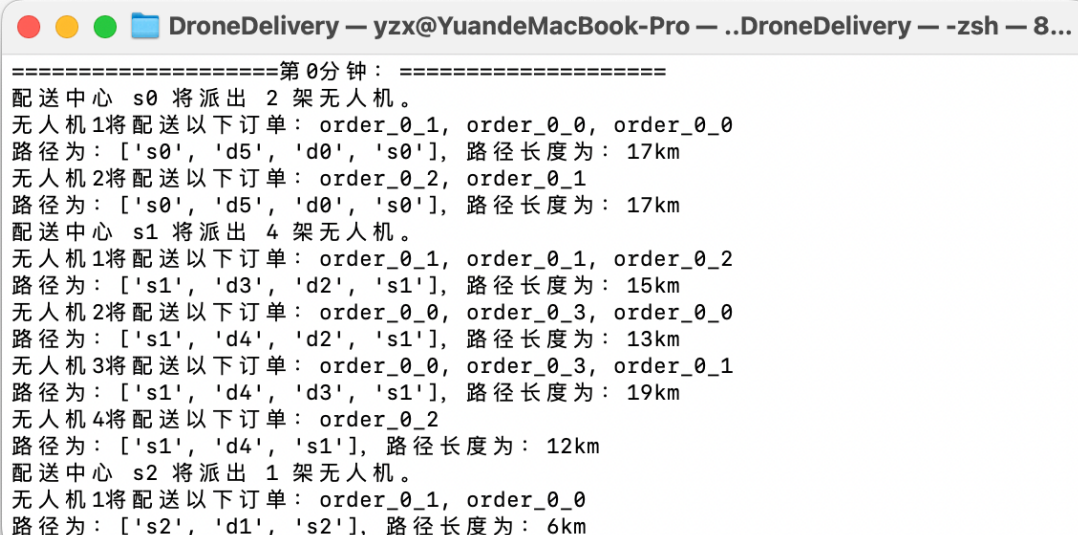
```

20     # 模拟每隔10分钟生成订单并进行决策
21     for minute in range(0, 180, 10):
22         # 生成订单
23         orders = simulate_order_generation(points, minute)
24
25         print(f"=====第{minute}分钟: =====")
26         # 进行决策
27         drone_assignments, reversed_orders = make_decisions(G, orders, current_time=minute)
28         orders += reversed_orders
29
30         # 打印决策结果
31         print("")
32         for center, orders in drone_assignments.items():
33             if len(orders) == 0:
34                 continue
35             print(f"配送中心 {center} 的无人机将完成以下订单: ")
36             for order in orders:
37                 print(f"订单ID: {order.order_id}, d: {order.point_id}, 优先级: {order.priority}")
38
39         # 等待
40         # import time
41         # time.sleep(10)

```

五、结果展示

执行结果如下图所示，仅展示部分结果：



```

DroneDelivery — yzx@YuandeMacBook-Pro — ..DroneDelivery — -zsh — 8...
=====第 0 分钟: =====
配送中心 s0 将派出 2 架无人机。
无人机1将配送以下订单: order_0_1, order_0_0, order_0_0
路径为: ['s0', 'd5', 'd0', 's0'], 路径长度为: 17km
无人机2将配送以下订单: order_0_2, order_0_1
路径为: ['s0', 'd5', 'd0', 's0'], 路径长度为: 17km
配送中心 s1 将派出 4 架无人机。
无人机1将配送以下订单: order_0_1, order_0_1, order_0_2
路径为: ['s1', 'd3', 'd2', 's1'], 路径长度为: 15km
无人机2将配送以下订单: order_0_0, order_0_3, order_0_0
路径为: ['s1', 'd4', 'd2', 's1'], 路径长度为: 13km
无人机3将配送以下订单: order_0_0, order_0_3, order_0_1
路径为: ['s1', 'd4', 'd3', 's1'], 路径长度为: 19km
无人机4将配送以下订单: order_0_2
路径为: ['s1', 'd4', 's1'], 路径长度为: 12km
配送中心 s2 将派出 1 架无人机。
无人机1将配送以下订单: order_0_1, order_0_0
路径为: ['s2', 'd1', 's2'], 路径长度为: 6km

```


DroneDelivery — yzx@YuandeMacBook-Pro — ..DroneDelivery — -zsh — 8...

```
配送中心 s0 的无人机将完成以下订单：
订单 ID: order_0_1, d: d0, 优先级: 0
订单 ID: order_0_0, d: d5, 优先级: 0
订单 ID: order_0_0, d: d0, 优先级: 1
订单 ID: order_0_2, d: d0, 优先级: 1
订单 ID: order_0_1, d: d5, 优先级: 2
配送中心 s1 的无人机将完成以下订单：
订单 ID: order_0_1, d: d2, 优先级: 0
订单 ID: order_0_1, d: d3, 优先级: 0
订单 ID: order_0_2, d: d3, 优先级: 0
订单 ID: order_0_0, d: d4, 优先级: 0
订单 ID: order_0_3, d: d4, 优先级: 0
订单 ID: order_0_0, d: d2, 优先级: 1
订单 ID: order_0_0, d: d3, 优先级: 1
订单 ID: order_0_3, d: d3, 优先级: 1
订单 ID: order_0_1, d: d4, 优先级: 2
订单 ID: order_0_2, d: d4, 优先级: 2
配送中心 s2 的无人机将完成以下订单：
订单 ID: order_0_1, d: d1, 优先级: 0
订单 ID: order_0_0, d: d1, 优先级: 1
```

DroneDelivery — yzx@YuandeMacBook-Pro — ..DroneDelivery — -zsh — 8...

```
=====第10分钟：=====
配送中心 s0 将派出 4 架无人机。
无人机1将配送以下订单： order_10_0, order_10_3, order_10_4
路径为： ['s0', 'd0', 's0'], 路径长度为： 8km
无人机2将配送以下订单： order_10_1, order_10_3, order_10_1
路径为： ['s0', 'd5', 'd0', 's0'], 路径长度为： 17km
无人机3将配送以下订单： order_10_2, order_10_2, order_10_0
路径为： ['s0', 'd5', 'd0', 's0'], 路径长度为： 17km
无人机4将配送以下订单： order_10_4
路径为： ['s0', 'd5', 's0'], 路径长度为： 14km
配送中心 s1 将派出 3 架无人机。
无人机1将配送以下订单： order_10_0, order_10_0, order_10_1
路径为： ['s1', 'd3', 'd4', 'd2', 's1'], 路径长度为： 20km
无人机2将配送以下订单： order_10_2, order_10_3, order_10_1
路径为： ['s1', 'd4', 'd2', 's1'], 路径长度为： 13km
无人机3将配送以下订单： order_10_0
路径为： ['s1', 'd2', 's1'], 路径长度为： 8km
配送中心 s2 将派出 2 架无人机。
无人机1将配送以下订单： order_10_1, order_10_4, order_10_0
路径为： ['s2', 'd1', 's2'], 路径长度为： 6km
无人机2将配送以下订单： order_10_2, order_10_3
路径为： ['s2', 'd1', 's2'], 路径长度为： 6km
```

配送中心 s0 的无人机将完成以下订单：

订单 ID: order_10_0, d: d0, 优先级: 0
订单 ID: order_10_3, d: d0, 优先级: 0
订单 ID: order_10_4, d: d0, 优先级: 0
订单 ID: order_10_1, d: d5, 优先级: 0
订单 ID: order_10_3, d: d5, 优先级: 0
订单 ID: order_10_1, d: d0, 优先级: 1
订单 ID: order_10_2, d: d0, 优先级: 1
订单 ID: order_10_2, d: d5, 优先级: 1
订单 ID: order_10_0, d: d5, 优先级: 2
订单 ID: order_10_4, d: d5, 优先级: 2

配送中心 s1 的无人机将完成以下订单：

订单 ID: order_10_0, d: d3, 优先级: 0
订单 ID: order_10_0, d: d4, 优先级: 0
订单 ID: order_10_1, d: d2, 优先级: 1
订单 ID: order_10_2, d: d2, 优先级: 1
订单 ID: order_10_3, d: d2, 优先级: 1
订单 ID: order_10_1, d: d4, 优先级: 1
订单 ID: order_10_0, d: d2, 优先级: 2

配送中心 s2 的无人机将完成以下订单：

订单 ID: order_10_1, d: d1, 优先级: 0
订单 ID: order_10_4, d: d1, 优先级: 0
订单 ID: order_10_0, d: d1, 优先级: 1
订单 ID: order_10_2, d: d1, 优先级: 2
订单 ID: order_10_3, d: d1, 优先级: 2