武汉大学国家网络安全学院

高级算法设计与分析大作业

课 程 名 称：　高级算法设计与分析

专 业 年 级：　网络空间安全 2023 级

姓　　　名：　柳熩

学　　　号：　2023202210140

协　作　者：　无

实 验 学 期：　2023-2024 学年第二学期

二〇二四年六月二十六日

目　　录

1　实验描述

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；

2　实验准备

2.1　实验目的

在本实验中，设计并实现一个无人机配送路径规划算法，合理选择学过的算法计算最短路径，以确保在满足订单优先级的前提下，最小化所有无人机的总配送路径。

2.2　实验背景

无人机配送能够有效解决最后一公里配送问题，特别是在配送速度和灵活性方面具有显著优势。本实验的目标是通过设计一个路径规划算法，优化无人机的配送路径，以提高配送效率和满足不同优先级订单的要求。

2.3　实验环境

- 编程语言：Python
- 使用库：random, heapq, itertools, matplotlib, networkx

2.4　实验设计

2.4.1　输入数据

本实验中输入的数据包括配送中心位置集合、边的集合、订单集合，具体描述如下：

- 配送中心位置集合：nodes；
- 边的集合：edges，每条边包含两个节点及其之间的距离；
- 订单集合：orders，每个订单包括订单 ID、目的地和优先级

2.4.2　无人机参数

本实验中规定无人机的参数如下所示：

- 最大携带物品数：3；
- 最大飞行距离：20 公里；
- 飞行速度：60 公里/小时；

3 实验步骤

3.1 初始化

首先根据题意，定义一系列常量，例如无人机的速度、时间间隔、单个无人机最大载货量、单个无人机最大飞行距离等，这些参数对于后续无人机的路径规划至关重要。

```
DRONE_SPEED =60 # Drone speed in km/h
TIME_INTERVAL =30 # Time interval in minutes
MAX_CARGO =3 # Max cargo capacity of a drone
MAX_DISTANCE =20 # Max distance a drone can fly in one trip in km
DELIVERY_TIME ={0: 180, 1: 90, 2: 30} # Delivery times in minutes for different
                                               priorities
```

除此之外，我们预定义了图的顶点集 V 和边集 E，其中包含边的起始点和终止点，以及边的权重。这里我们定义了总共 8 个顶点，17 条边，以及每条边的权重。

```
NODES =[0, 1, 2, 3, 4, 5, 6, 7, 8]
EDGES =[
    (0, 1, 5),
    (0, 4, 4),
    (0, 5, 7),
    (0, 3, 4),
    (0, 6, 7),
    (1, 2, 2),
    (1, 3, 6),
    (1, 6, 5),
    (2, 5, 4),
    (2, 6, 3),
    (3, 4, 3),
    (3, 6, 10),
    (4, 5, 2),
    (7, 1, 7),
    (7, 2, 4),
    (8, 3, 3),
    (8, 6, 5)
]
```

在定义常量后，使用邻接列表初始化图，每个节点添加相应的边和权重，以

及生成唯一的订单 ID，定义如下：

```
order_id_generator =itertools.count()

def initialize_graph(nodes, edges):
    # Initialize an empty adjacency list for each node in the graph
    graph ={node: []for node in nodes}

    # Iterate over each edge defined by (u, v, w) where u and v are nodes and w is the weight
    for u, v, w in edges:
        # Append the edge (v, w) to the adjacency list of node u
        graph[u].append((v, w))
        # Append the edge (u, w) to the adjacency list of node v (because the graph is undirected)
        graph[v].append((u, w))

    # Return the constructed graph as an adjacency list
    return graph
```

3.2 循环模拟

在主程序中，总共定义 6 个小时（360 分钟）的运行时间，每隔 30 分钟生成一份新订单，根据生成订单的优先级进行排序，优先级越高的订单越先处理。对于相同优先级内的不同订单，根据订单的目的地选取最近的配送中心，这里采用的是 dijkstra 最短路径搜索算法。在选取到最优的配送中心后，从配送中心到配送终点同样使用 dijkstra 算法选取最优路径。对于选择的路径记录路径点、优先级、消耗时间、做出决定等信息。最后将生成的网络图可视化展示出来。

```
def main():
    # initialize
    graph =initialize_graph(NODES, EDGES)
    orders =[]

    start_time =time.time() # Start time of simulation


    # Simulate six hours of delivery process
    current_time =0
    total_distance =0
    total_flights =0

    while current_time <360: # Six hours in minutes (360 minutes)
        new_orders =generate_orders()
        orders.extend(new_orders)
```

```
    # Sort orders by priority (assuming orders are tuples with (order_id, location, priority))
    orders.sort(key=lambda x: x[2], reverse=True)

    # For each priority level, assign orders to delivery centers and plan paths
    for priority in [2, 1, 0]:
        priority_orders =[order for order in orders if order[2] ==priority]
        while priority_orders:
            center =choose_center(priority_orders[0], graph) # Function to choose delivery center
            paths =plan_path(center, priority_orders, graph) # Function to plan delivery paths
            for path, distance in paths:
                # Extract details for each delivery
                order_ids =[order[0] for order in path]
                path_points =[order[1] for order in path]
                priorities =[order[2] for order in path]
                time_taken =(distance /DRONE_SPEED) *60 # Convert distance to minutes
                decision ="Immediate" if priorities[0] ==2 else "Batch"

                # Print detailed delivery information
                print(
                    f"Current Time: {current_time} minutes, Delivery Center: {center}, Drones Assigned: {len(paths)}, Delivery Path: {path_points}, Destination: {path_points}, Total Distance: {distance} km, Time Taken: {time_taken:.2f} minutes, Priority: {priorities[0]}, Delivery Type: {decision}"
                )

                # Update total distance and number of flights
                total_distance +=distance
                total_flights +=1

            # Update the list of orders after deliveries
```

```
            priority_orders = [
                order
                for order in priority_orders
                if order not in [o for path in paths for o in path[0]]
            ]

        # Increment simulation time
        current_time += TIME_INTERVAL

    # Simulation end time
    end_time = time.time()

    # Calculate total simulation execution time
    execution_time_ms = (end_time - start_time) * 1000

    # Output total delivery stats over six hours
    total_time_taken = (total_distance / DRONE_SPEED) * 60 # Convert total distance
                                                  to minutes
    print(
        f"Total Delivery Distance in Six Hours: {total_distance} km, Total Time
                                                        Taken: {total_time_taken:.2f}
                                                        minutes, Total Flights: {
                                                        total_flights}"
    )
    print(f"Total Execution Time: {execution_time_ms:.2f} ms")

    # Additional code for visualization or further processing can be added here
    draw_graph()
```

3.3 订单生成

首先，初始化一个空列表 new_orders 来存储生成的新订单。然后，随机确定要生成的订单数量，该数量在 0 到 9 之间。这种范围允许在任何给定时间处理订单数量的灵活性。接下来，每个订单通过订单 ID 生成器分配一个唯一标识符。

之后，随机分配每个订单的优先级，优先级可以是三个级别中的一个：0（低）、1（中）或 2（高）。每个订单还会随机分配一个投递点，投递点从可用的点（2、3、4、5、6、7、8）中选择。

生成的订单表示为一个包含 (order_id, point, priority) 元组的形式，并附加到 new_orders 列表中。最后，返回生成的新订单列表。

```
def generate_orders():
    # Initialize an empty list to hold the new orders
    new_orders = []

```

```
    # Generate a random number of orders between 0 and 7
    for _ in range(random.randint(0, 9)):
        # Get the next unique order ID from the generator
        order_id =next(order_id_generator)

        # Randomly choose a priority for the order (0, 1, or 2)
        priority =random.choice([0, 1, 2])

        # Randomly choose a drop point for the order from the available drop points
        point =random.choice([2, 3, 4, 5, 6, 7, 8])

        # Append the order as a tuple (order_id, point, priority) to the list of new orders
        new_orders.append((order_id, point, priority))

    # Return the list of newly generated orders
    return new_orders
```

3.4 配送中心选取

首先，函数接受两个参数：order 和 graph。order 是一个包含订单信息的元组，重点关注订单的目的地；graph 则是一个图的表示形式，其中存储了各个地点之间的距离。

函数内部的第一步是使用 Dijkstra 算法计算从配送中心 0 到所有地点的距离。这一步的结果存储在 distances_from_0 中。接着，函数同样使用 Dijkstra 算法计算从配送中心 1 到所有地点的距离，结果存储在 distances_from_1 中。

在获得两个配送中心到所有地点的距离后，函数将比较这两个中心到订单目的地的距离。如果从配送中心 0 到订单目的地的距离小于或等于从配送中心 1 到订单目的地的距离，函数将返回 0，表示选择配送中心 0。如果从配送中心 1 到订单目的地的距离更近，则函数返回 1，表示选择配送中心 1。

通过这种方法，choose_center 函数能够有效地选择离订单目的地最近的配送中心，从而优化配送效率。这种基于距离的决策过程对于物流和配送系统的运行具有重要意义，能够显著提高配送的及时性和资源的利用效率。

```
def choose_center(order, graph):
    """
    Selects the nearest delivery center based on the order destination.
    This function calculates the distances from two delivery centers to the order destination
    and determines which center is closer using Dijkstra's algorithm.

    Args:
```

```
    order: Tuple containing order information, focusing on the destination.
    graph: Graph representation where distances are stored.

    Returns:
    Delivery center number (0 or 1) depending on which center is closer to the order destination.
    """
    # Calculate distances from delivery center 0 to all locations
    distances_from_0 =dijkstra(graph, 0)
    # Calculate distances from delivery center 1 to all locations
    distances_from_1 =dijkstra(graph, 1)

    # Compare distances from both centers to the order destination
    if distances_from_0[order[1]] <=distances_from_1[order[1]]:
        # If delivery center 0 is closer, return 0
        return 0
    else:
        # If delivery center 1 is closer, return 1
        return 1
```

3.5 路径规划

plan_path 函数旨在从一个中心位置为每个订单规划最优路径。通过计算从中心位置到各订单目的地的最短路径，函数能够有效地组织和分配配送资源，提高整体配送效率。

首先，函数接受三个参数：center、orders 和 graph。center 是路径规划的起始位置，通常是配送中心；orders 是一个包含各订单位置信息和需求的列表；graph 是一个图的表示形式，存储了各个地点之间的距离。

在函数内部，首先初始化一个空列表 path 来存储规划好的路径。接着，进入一个循环，当 orders 列表中还有订单未处理时，循环继续执行。在每次循环中，将当前地点设为中心位置，并初始化一个空列表 cargo 来存储当前配送路径上的订单，同时初始化 trip_distance 为 0 以累计当前路径的总距离。

使用 Dijkstra 算法计算当前地点到所有地点的最短距离，并存储在 distances 中。在一个嵌套的循环中，函数会选择离当前地点最近的订单，并计算到该订单的距离。如果当前路径总距离加上到该订单的距离不超过最大允许距离 MAX_DISTANCE，则将该订单添加到 cargo 列表中，并从 orders 列表中移除。同时，更新 trip_distance 和 current_location，并重新计算从当前地点到所有地点的最短距离。

当 cargo 已满或没有更多订单可以处理时，函数会将返回到中心位置的距离加到 trip_distance 中，并将当前路径的 cargo 和 trip_distance 添加到 path 列表中。这样，每次循环都会生成一条完整的配送路径，直至所有订单处理完毕。

最终，函数返回 path 列表，其中每个元素都是一个包含该次配送的订单和距离的元组。这种路径规划方法能够有效优化配送路径，减少配送时间和成本，提高配送系统的整体效率。

```python
def plan_path(center, orders, graph):
    """
    Plans the optimal paths from a center location to each order.

    Args:
    center: Starting location for path planning.
    orders: List of orders, each consisting of a location and demand.
    graph: Graph representation where distances are stored.

    Returns:
    A list of paths, each containing cargo (orders) and distance for each trip.
    """
    # List to store planned paths
    path =[]
    # While there are still orders to process
    while orders:
        # Set current location to the center
        current_location =center
        # List to store orders for the current trip
        cargo =[]
        # Accumulate distance for the current trip
        trip_distance =0
        # Calculate shortest distances from current location
        distances =dijkstra(graph, current_location)

        # While cargo is not full and there are orders left
        while len(cargo) <MAX_CARGO and orders:
            # Choose the nearest order to the current location
            nearest_order =min(orders, key=lambda o: distances[o[1]])
            # Distance to the nearest order
            distance_to_order =distances[nearest_order[1]]
            # Check if within maximum trip distance
            if trip_distance +distance_to_order <=MAX_DISTANCE:
                # Add the nearest order to cargo
                cargo.append(nearest_order)
                # Remove the chosen order from the list
                orders.remove(nearest_order)
                # Update trip distance
                trip_distance +=distance_to_order
                # Update current location
                current_location =nearest_order[1]
                # Recalculate shortest distances from current location
                distances =dijkstra(graph, current_location)
```

```

        # Add return distance to total trip distance
        trip_distance +=distances[center]
        # Append cargo and distance for the current trip to the path list
        path.append((cargo, trip_distance))

    # Return the planned path list
    return path
```

3.6 图可视化

draw_graph 函数用于创建和绘制一个图形，展示配送中心和投递点之间的连接关系及其权重。通过使用 NetworkX 库，该函数能够可视化配送网络，帮助理解和优化物流系统。

首先，函数创建一个空图 G，并定义配送中心和投递点。配送中心包括节点 0 和 1，而投递点则包括节点 2 至 8。接着，定义一个字典 labels，为每个节点分配标签，标识其在图中的角色（例如配送中心或投递点）。

为了区分不同类型的节点，函数为每个节点分配颜色。如果节点是配送中心，则颜色为绿色（36BA98）；如果是投递点，则颜色为黄色（E9C46A）。

接下来，函数向图中添加节点和边。节点包括所有的配送中心和投递点，而边则由全局变量 EDGES 定义，其中包含节点之间的加权边。然后，函数使用 nx.spring_-layout 方法确定所有节点的位置，这种方法通过模拟弹簧系统布局节点，使得图形更加美观和易读。

函数使用 nx.draw 方法绘制图形，指定节点位置、标签和颜色，并设置节点大小和字体属性。为了展示边的权重，函数创建一个字典 edge_labels，其中包含每条边的权重，并使用 nx.draw_networkx_edge_labels 方法将权重显示在图上。

此外，函数还添加了一个图例，区分配送中心和投递点的颜色和类型。使用 plt.scatter 创建图例标记，并通过 plt.legend 添加图例到图形中。最后，函数设置图形的标题为”Drone Delivery Network” 并显示图形。

```
def draw_graph():
    # Create a graph
    G =nx.Graph()

    # Define delivery centers and drop points
    delivery_centers =[0, 1]
    drop_points =[2, 3, 4, 5, 6, 7, 8]

    # Define labels for the nodes
    labels ={
```

```python
        0: "Delivery Center 0",
        1: "Delivery Center 1",
        2: "Drop Point 2",
        3: "Drop Point 3",
        4: "Drop Point 4",
        5: "Drop Point 5",
        6: "Drop Point 6",
        7: "Drop Point 7",
        8: "Drop Point 8",
    }

    # Assign colors to nodes based on their type
    node_color =["#36BA98" if node in delivery_centers else "#E9C46A" for node in NODES]

    # Add nodes and edges to the graph
    G.add_nodes_from(NODES)
    G.add_weighted_edges_from(EDGES)

    # Determine the positions for all nodes
    pos =nx.spring_layout(G)

    # Draw the graph with the specified node positions, labels, and colors
    nx.draw(G, pos, with_labels=True, labels=labels, node_color=node_color,
            node_size=500, font_size=10,
            font_weight='bold')

    # Create a dictionary for edge labels showing the weights
    edge_labels ={(u, v): d['weight'] for u, v, d in G.edges(data=True)}

    # Draw the edge labels
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='red')

    # Add a legend for the node types
    plt.scatter([], [], c="#36BA98", label="Delivery Center", edgecolors="none", s=100)
    plt.scatter([], [], c="#E9C46A", label="Drop Point", edgecolors="none", s=100)
    plt.legend(scatterpoints=1, frameon=False, labelspacing=1)

    # Set the title of the plot
    plt.title("Drone Delivery Network", color="black")

    # Show the plot
    plt.show()
```

4　实验结果

根据随机生成的订单，程序得出的结果如图4.1所示：

```
(base) PS C:\Users\Leo\OneDrive\研究生\Documents\teaching\课程\高级算法\src> python .\main.py
Current Time: 0 minutes, Delivery Center: 0, Drones Assigned: 1, Delivery Path: [4, 3], Destination: [4, 3], Total Distance: 11 km, Time Taken: 11.00 minutes, Priority: 2, Delivery Type: Immediate
Current Time: 0 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [4, 2, 2], Destination: [4, 2, 2], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 1, Delivery Type: Batc
h
Current Time: 0 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [6], Destination: [6], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 1, Delivery Type: Batch
Current Time: 0 minutes, Delivery Center: 0, Drones Assigned: 1, Delivery Path: [4, 8, 2], Destination: [4, 8, 2], Total Distance: 25 km, Time Taken: 25.00 minutes, Priority: 0, Delivery Type: Batc
h
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [4, 3, 8], Destination: [4, 3, 8], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [2, 2, 2], Destination: [2, 2, 2], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [4, 2, 2], Destination: [4, 2, 2], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 2, Delivery Path: [6], Destination: [6], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 1, Delivery Type: Batch
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [4, 8, 8], Destination: [4, 8, 8], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 0, Delivery Type: Bat
ch
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [2, 6, 8], Destination: [2, 6, 8], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 0, Delivery Type: Bat
ch
Current Time: 30 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [7], Destination: [7], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 0, Delivery Type: Batch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [4, 5, 2], Destination: [4, 5, 2], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [3, 8, 2], Destination: [3, 8, 2], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [2], Destination: [2], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 2, Delivery Type: Immediate
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [4, 4, 3], Destination: [4, 4, 3], Total Distance: 11 km, Time Taken: 11.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [3, 3, 8], Destination: [3, 3, 8], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [6, 2, 2], Destination: [6, 2, 2], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [8, 8, 7], Destination: [8, 8, 7], Total Distance: 30 km, Time Taken: 30.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [4, 8, 8], Destination: [4, 8, 8], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 0, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [2, 6, 8], Destination: [2, 6, 8], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 0, Delivery Type: Bat
ch
Current Time: 60 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [7], Destination: [7], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 0, Delivery Type: Batch
Current Time: 90 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [4, 5, 2], Destination: [4, 5, 2], Total Distance: 17 km, Time Taken: 17.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 90 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [3, 8, 2], Destination: [3, 8, 2], Total Distance: 22 km, Time Taken: 22.00 minutes, Priority: 2, Delivery Type: Imm
ediate
Current Time: 90 minutes, Delivery Center: 0, Drones Assigned: 3, Delivery Path: [2], Destination: [2], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 2, Delivery Type: Immediate
Current Time: 90 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [4, 4, 3], Destination: [4, 4, 3], Total Distance: 11 km, Time Taken: 11.00 minutes, Priority: 1, Delivery Type: Bat
ch
Current Time: 90 minutes, Delivery Center: 0, Drones Assigned: 4, Delivery Path: [3, 3, 8], Destination: [3, 3, 8], Total Distance: 14 km, Time Taken: 14.00 minutes, Priority: 1, Delivery Type: Bat
ch
```

图 4.1　程序执行结果

在本次实验中，定义的无人机路径规划图如图4.2所示，包含 2 个配送中心，以及 7 个卸货点：

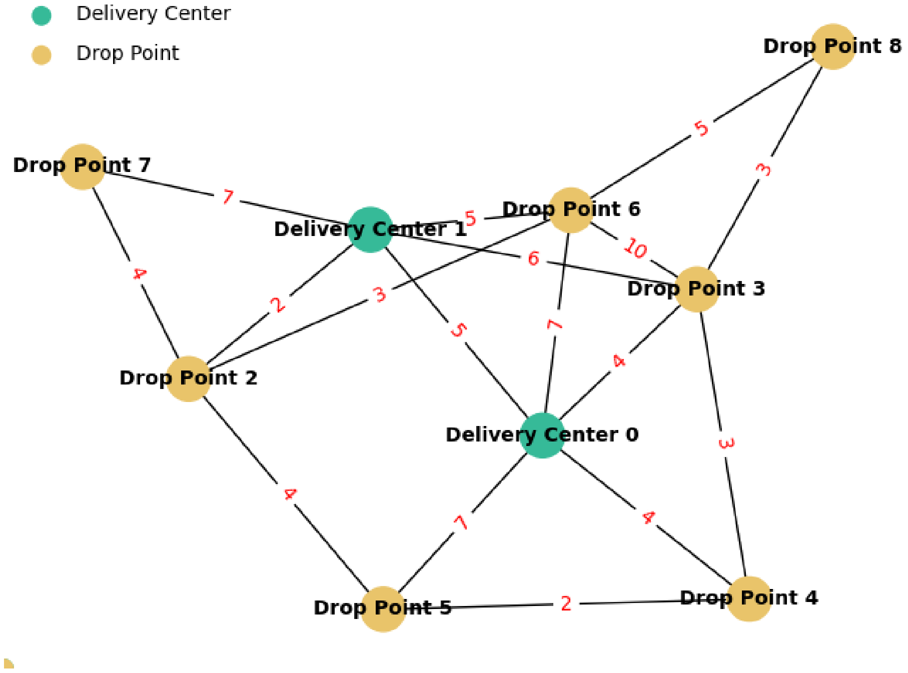

图 4.2　无人机配送路径图