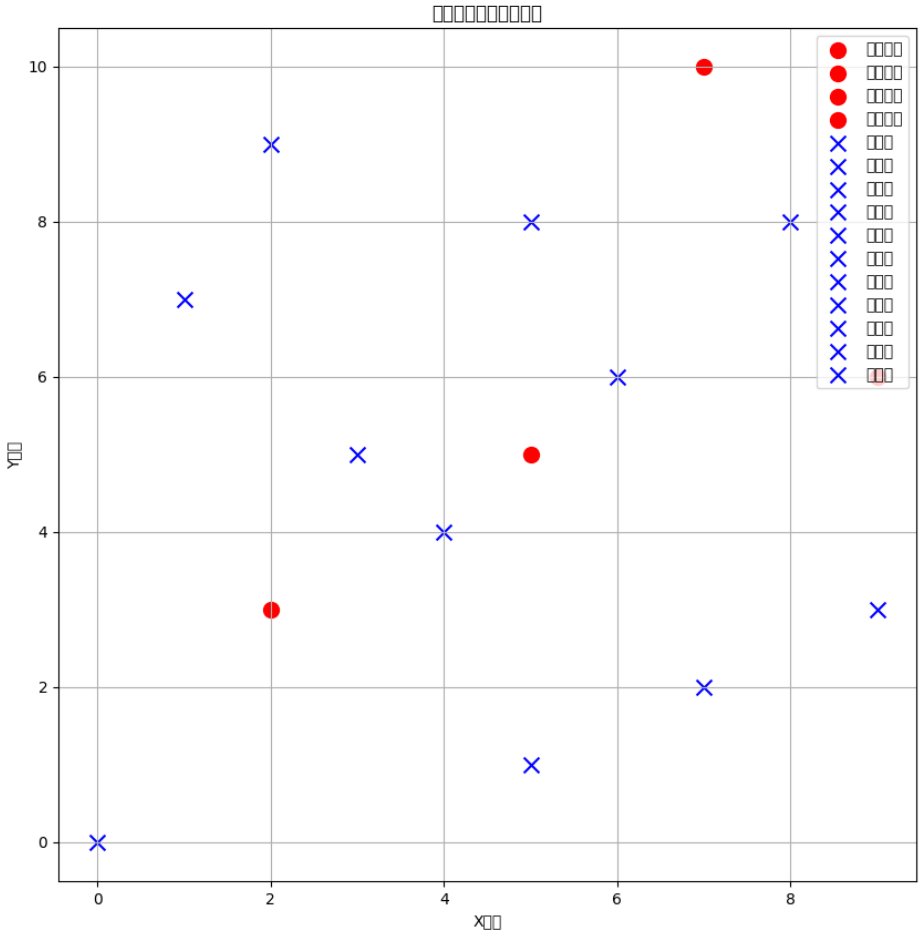
高级算法课程作业

图与数据生成

首先，根据问题需求，先对配送中心位置和卸货点进行规定，本方案中将这两个位置根据现实情况进行约束（如对于武汉市，其物流中心主要集中东北方向的物流中心，或者位于武昌站等附近，但收货驿站较为分散），再通过代码随机生成，选取的点情况如下：

配送中心 = [(2, 3), (9, 6), (7, 10), (5, 5)]
卸货点 = [(0, 0), (5, 1), (8, 8), (6, 6), (1, 7), (3, 5), (7, 2), (4, 4), (9, 3), (5, 8), (2, 9)]其中图的单位为公里，具体情况如下所示，红色的点代表配送中心，蓝色的叉代表卸货点



同时，为了模拟问题中描述的订单情况，对于模拟的一天中订单的情况，假设题目中 m=5，t = 15，即每隔 15 分钟生成一批订单，每个卸货点相对独立，订单数为 0-5，利用代码随机，得出的模拟订单数据部分如下，由于生成的对应一天内的情况，因此数目较多，完整的数据文件随报告附在 order.csv 表格中。

	A	B	C	D
1	priority	location_x	location_y	timestamp
2	一般	5	8	0
3	较紧急	9	3	0
4	紧急	7	2	0
5	较紧急	3	5	15
6	一般	2	3	15
7	一般	1	7	30
8	紧急	2	3	30
9	一般	5	8	30
10	一般	8	8	30
11	较紧急	9	3	45
12	一般	2	3	45
13	一般	1	7	45
14	紧急	2	3	60
15	较紧急	5	1	60
16	紧急	3	5	60
17	一般	5	1	90
18	一般	9	3	90
19	紧急	5	1	90
20	一般	8	8	135
21	紧急	1	7	135
22	一般	3	5	135
23	一般	1	7	135
24	一般	9	3	135

算法设计思路

在对问题进行合理分析后，本方案拟通过结合贪心算法和模拟退火算法构建解决方案。

贪心算法解决路径规划问题的理论分析:

贪心算法是一种逐步构建解决方案的方法，它在每一步选择当前状态下最优的局部选择，以期望最终得到全局最优解。贪心算法具有以下特点和优势:

1. 局部最优性:
   - 每一步选择在当前状态下对问题的局部最优解。
   - 在路径规划中，选择离当前节点最近的下一个订单进行配送。
2. 简单高效:
   - 贪心算法的实现通常比较简单，运行速度较快。
   - 由于不需要回溯或全局搜索，时间复杂度较低。
3. 启发式:
   - 通过贪心选择，能够在很多实际问题中快速获得较优的解，虽然不一定是最优解。

具体到本方案的路径规划问题，贪心算法的设计思路如下:

- 按优先级排序: 通过对订单按优先级排序，确保高优先级订单首先被考虑。
- 选择最近的订单: 在每一步选择当前路径上最近的订单进行配送，这样可以在保证路径长度不超过最大距离的情况下，尽可能多地完成订单。

局限性:

- 全局次优解: 由于贪心算法只关注局部最优，可能会陷入全局次优解，无法保证全局最优。
- 不适用于所有问题: 贪心算法适用于满足贪心选择性质和最优子结构性质的问题，不适用

于所有路径规划问题。

模拟退火算法优化路径的理论分析：

模拟退火算法是一种全局优化算法，源自物理中的退火过程，具有以下特点和优势：

1. 全局搜索能力：
   - 通过在解空间中模拟物理退火过程，在高温时允许较差解，随着温度降低逐步趋向最优解。
   - 可以跳出局部最优，找到全局最优或近似全局最优解。
2. 概率接受机制：
   - 以一定概率接受较差解，避免陷入局部最优。
   - 随着温度降低，接受较差解的概率逐渐减小。
3. 可调参数：
   - 初始温度和降温速率等参数可以调整，以控制搜索过程和精度。

具体到本方案的路径规划问题，贪心算法的设计思路如下：

- 初始路径：从贪心算法获得的初始路径开始。
- 路径扰动：通过交换两个订单的位置来产生新路径（解）。
- 评价函数：计算路径的总距离作为评价标准。
- 接受准则：如果新路径的总距离更短，则接受新路径；否则，以一定概率接受较长路径。

优势：

- 跳出局部最优：通过概率接受较差解，避免陷入局部最优，具有较好的全局搜索能力。
- 灵活性强：适用于多种复杂的优化问题，能够在较大搜索空间中找到近似最优解。

局限性：

- 计算开销大：由于需要多次迭代和计算，时间复杂度较高。
- 参数依赖性强：结果依赖于初始温度和降温速率等参数，参数选择不当可能影响效果。

两种算法综合应用优势：

在路径规划问题中，结合贪心算法和模拟退火算法，可以充分利用二者的优势：利用贪心算法快速得到一个较优的初始解。再通过模拟退火算法对初始解进行优化，进一步提升解的质量。这种组合策略既能保证初始解的快速生成，又能通过全局优化提升最终解的质量，实现高效和高质量的路径规划。

因此，对应算法设计如下：

1. 按优先级排序订单：

   Sort(orders,key=priority)

   对应伪代码如下：

```
Function GreedyPathPlanning(delivery_centers, orders):
    Sort orders by priority
    paths = defaultdict(list)

    For each center in delivery_centers:
        remaining_orders = Copy(orders)

    While remaining_orders is not empty:
        current_location = center
        current_path = []
        current_distance = 0

        While remaining_orders is not empty and current_distance < max_distance:
            next_order = FindOrderWithMinDistance(current_location, remaining_orders)
            distance_to_order = CalculateDistance(current_location, next_order.location)

            If current_distance + distance_to_order + CalculateDistance(next_order.location, center) <= max_distance:
                Append next_order to current_path
                current_distance += distance_to_order
                current_location = next_order.location
                Remove next_order from remaining_orders

            Else:
                Break

        Append current_path to paths[center]
    Return paths
```

2. 选择最近的订单：

   $next\_order = \min_{order \in remaining\_orders} Distance(current\_location, order.location)$

3. 使用模拟退火算法进行路径优化

   初始化 $T = T_{initial}$

```
Function SimulatedAnnealingPathOptimization(initial_path, T_initial, cooling_rate):
    current_path = initial_path
    best_path = initial_path
    best_distance = CalculateTotalDistance(best_path)
    T = T_initial

    While T > 1:
        new_path = SwapTwoOrders(current_path)
        current_distance = CalculateTotalDistance(current_path)
        new_distance = CalculateTotalDistance(new_path)

        If new_distance < current_distance OR Random(0, 1) < Exp((current_distance - new_distance) / T):
            current_path = new_path

        If new_distance < best_distance:
            best_path = new_path
            best_distance = new_distance

        T = T * (1 - cooling_rate)

    Return best_path
```

4. 交换订单的位置与计算总距离：

   $new\_path = Swap(path, i, j)$

   $Distance(path) = \sum_{k=1}^{N} Distance(location_k, location_{k+1})$

   接受新路径的概率：$P = \exp(current\_distance - new\_distance / T)$

总结分析：

贪心算法每一步都选择当前最优解，即选择距离当前无人机位置最近的订单进行配送，以期减少总路径长度。在选择过程中，需确保总路径长度不超过无人机的最大飞行距离。其局部选择公式为：

$next\_order = arg\_min\_Distance_{order \in remaining\_orders}(current\_location, order.location)$

路径长度约束条件为：

current_distance+Distance(current_location,next_order.location)+Distance(next_order.lo

cation,center)≤max_distance

订单排序的时间复杂度为 O(nlogn)，其中 n 是订单的数量。总体上，贪心算法的时间复杂度为 O(nlogn+j·n2)，其中 j 是配送中心的数量

模拟退火算法通过随机扰动当前解（交换两个订单的位置），并使用概率接受机制来跳出局部最优，逐步接近全局最优。其概率接受公式为：

$$P=\exp(\text{current\_distance} - \text{new\_distance} / T)$$

温度逐步降低的公式为：

$$T=T\times(1-\text{cooling\_rate})$$

模拟退火过程中的循环次数取决于初始温度 $T_{initial}$ 和冷却速率 cooling_rate\。设 k 为模拟退火的迭代次数，则总时间复杂度为 O(k·n)。

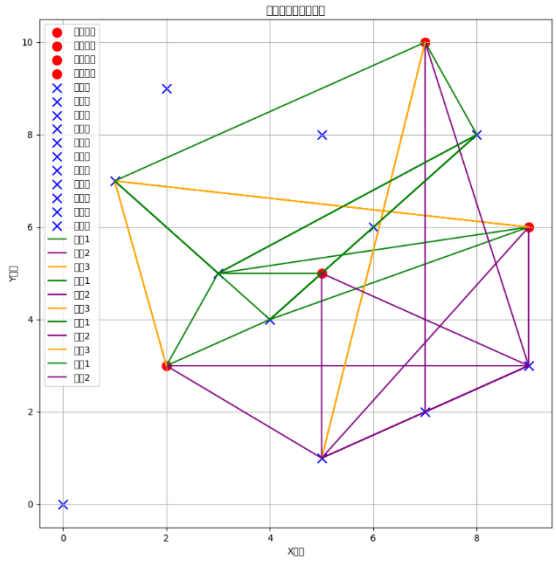
代码实现与实际运行结果

基于以上分析，得出该方案可行，对应具体代码实现关键部分如下：

```python
# 规划路径（贪心算法）
def plan_path(orders):
    orders = sorted(orders, key=lambda x: x.priority)
    paths = defaultdict(list)

    for center in delivery_centers:
        remaining_orders = orders[:]
        while remaining_orders:
            current_location = center
            current_path = []
            current_distance = 0

            while remaining_orders and current_distance < max_distance:
                next_order = min(remaining_orders, key=lambda x: calculate_distance(current_location, x.location))
                distance_to_order = calculate_distance(current_location, next_order.location)

                if current_distance + distance_to_order + calculate_distance(next_order.location,
                                                                             center) <= max_distance:
                    current_path.append(next_order)
                    current_distance += distance_to_order
                    current_location = next_order.location
                    remaining_orders.remove(next_order)
                else:
                    break

            paths[center].append(current_path)

    return paths
```

```python
# 路径优化（模拟退火算法）
def simulated_annealing(path, temperature=10000, cooling_rate=0.003):
    def calculate_total_distance(path):
        total_distance = 0
        current_location = delivery_centers[0]
        for order in path:
            total_distance += calculate_distance(current_location, order.location)
            current_location = order.location
        total_distance += calculate_distance(current_location, delivery_centers[0])
        return total_distance

    def swap_two_orders(path):
        if len(path) < 2:
            return path
        new_path = path[:]
        idx1, idx2 = random.sample(range(len(new_path)), k: 2)
        new_path[idx1], idx2 = new_path[idx2], new_path[idx1]
        return new_path

    current_path = path
    best_path = path
    best_distance = calculate_total_distance(best_path)

    while temperature > 1:
        new_path = swap_two_orders(current_path)
        current_distance = calculate_total_distance(current_path)
        new_distance = calculate_total_distance(new_path)

        if new_distance < current_distance or random.random() < np.exp((current_distance - new_distance) / temperature):
            current_path = new_path

        if new_distance < best_distance:
            best_path = new_path
            best_distance = new_distance

        temperature *= 1 - cooling_rate

    return best_path
```

同时，对于前 2 个小时的数据，模拟生成的路径图如下所示：



其中，图中红色圆圈和蓝色叉分别表示配送中心和卸货点的位置。不同颜色的路径表示从某个配送中心出发的一架或多架无人机的具体路径。从结果可得算法能够合理对路线进行

规划，并满足订单的优先需求。