学号： 2023282210196

高级算法课程作业

无人机配送问题

院(系)名 称 ： 国家网络空间安全学院

专 业 名 称 ： 电子信息

学 生 姓 名 ： 王启河

指 导 教 师 ： 林海

二〇二四年六月

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 模型建立

2.1 地图建立

对于本问题的地图，主要有配送中心和卸货点组成，采用保存坐标的方式保存题图。设 $\{C_1, C_2, ..., C_M\}$ 为 M 个配送中心坐标的集合；$\{U_1, U_2, ..., U_N\}$ 为 N 个卸货点坐标的集合，$B_{ij}(i = 1, 2, ...M, j = 1, 2, ...N)$ 表示配送中心 C_i 和卸货点 U_j 的距离；$D_{ij}(i = 1, 2...N, j = 1, 2...N)$ 表示卸货点 U_i 和卸货点 U_j 的距离，距离都采用欧式距离计算。

2.2 订单建立

本问题要求每隔 t 分钟，所有卸货点生成一次订单，目标是考虑一段时间内的订单处理，因此可以设会产生 s 轮订单；$L_{ij}(i = 1, 2, ...s, j = 1, 2, ..N)$ 表示卸货点 U_j 在第 i 轮产生订单的订单数量；$T_{ij}(i = 1, 2, ...s, j = 1, 2, ..N)$ 表示卸货点 U_j 在第 i 轮产生订单的剩余时间，取值为 0.5、1.5、3。

2.3 无人机建立

设一架无人机配送的最大容量为 max_take，无人机配送的最远距离为 max_length；对于一架无人机，用 $A_j(j = 1, 2, ...N)$ 表示本架无人机在卸货点 U_j 投放的物品数量，P_j 的取值为 $0 - max\_take$，若为 0，则表示无人机不经过卸货点 U_j；用 $P = \{(c, (u1, u2, ...ul)\}(1 \leq l \leq N)$ 表示该无人机的配送路径，该路径的配送中心为 C_c，经过的卸货点顺序为 $U_{u1} -> U_{u2} -> ... -> U_{ul}$，即无人机路径为 $C_c -> U_{u1} -> U_{u2} -> ... -> U_{ul} -> C_c$。

2.4 问题分解转化

根据问题描述，我们需要求的是一段时间内所有无人机的配送最短路径，如果我们将整个时间段的订单进行同时处理，这是不符合实际情况的。在真实情况中，我们只能知道当前时刻和之前时刻的订单生成情况，不能知道后续时刻的订单生成情况。因此我们应该设计的是一个在线算法，每轮订单都做出当前时刻的决策即可。

在问题描述中提到，我们在做决策时可以先不对当前订单进行配送，可以累积到后面做一起做决策。所以我们采用每次都只派送紧急订单，其余订单移动到对应的紧急轮次的方法。我设置每隔 30 分钟生成一次订单，因此只要有订单成为紧急订单，则需要立即派送。可以使用 T_{ij} 和 L_{ij} 在线计算出第 i 轮订单，卸货点卸货点 U_j 必须派送的订单 $O_{ij}(i = 1, 2, ...s, j = 1, 2, ...N)$。

要使得所有订单的配送路径和最小，我们可以将求整个时段的问题分解成求出每轮订单最小配送路径，最后再将所有轮次的路径和求和即可。所以我们的问题可以分解转化成：考虑第 i 轮的必发订单 $\{O_{i1}, O_{i2}, ...O_{iN}\}(i = 1, 2, ...s)$，求满足约束条件的配送路径，使得所有的笔法订单都能派送，且无人机的配送路径和最小。对于分解转化后的问题，我设计实现了一种基于贪心算法的求解算法。

3　基于贪心的算法设计与实现

3.1　算法整体思路

对于找最小路径问题，基于贪心算法的设计整体思路是先通过 T_{ij} 和 L_{ij} 计算出所有轮订单的必发订单 O_{ij}。然后找到所有满足约束条件的合法回路，然后通过基于贪心算法的单轮订单派送算法获得此轮订单的派送方案，统计完此轮路径总长和无人机数量后，进行下一轮。当所有轮订单都处理好后，输出相关信息即可。算法整体流程图如图 3.1 所示。

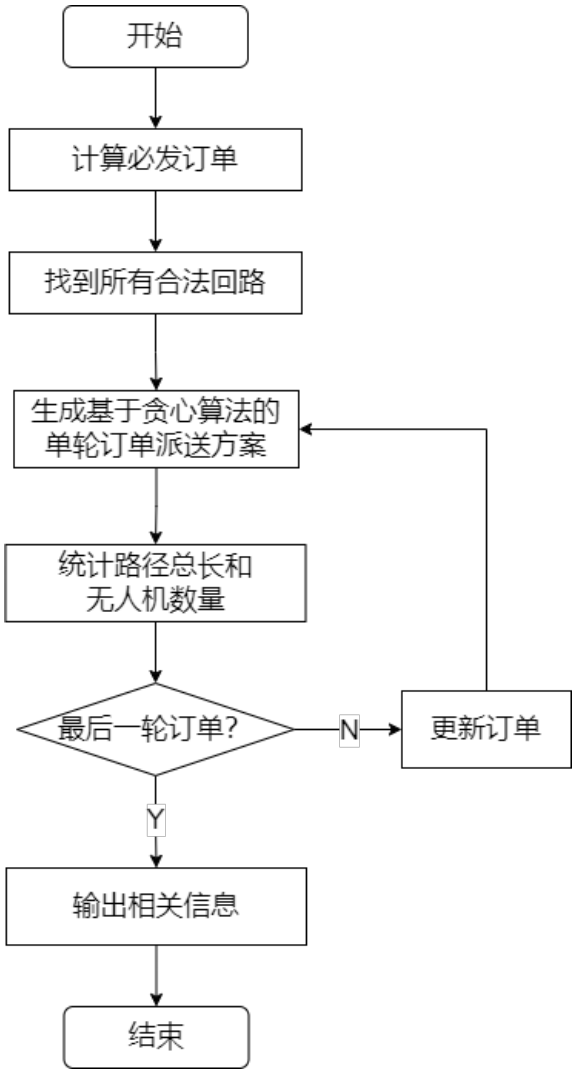

图 3.1　算法整体思路流程图

整体算法中包含了计算必发订单、找到合法回路和基于贪心的单轮订单派送算法三部分，接下来详细介绍这三部分。

3.2 必发订单生成算法

由于问题描述中设置了紧急、较紧急和一般三种级别，分别对应 0.5 小时、1.5 小时和 3 小时送到，我们是半小时生成一次订单。因此生成的紧急订单当前轮次就需要立马发送，生成的较紧急订单则 3 轮次后发送，一般订单则是 6 轮次后生成。我们设定当一段时间到最后一轮时，需要把前面所有订单都发送出去。算法实现代码如图 3.2 所示。

```
def GetOrders(order_num, order_time):
    """ 根据订单数量和订单时间生成每轮必须发的订单
        若只剩0.5小时，则必须发
        输入：order_num 订单数量
             order_time 订单剩余时间 (0.5, 1.5. 3)对应 (1, 3, 6)
        输出：orders是必发订单"""
    m = len(order_num)
    n = len(order_num[1])
    # 初始化一个必发订单
    orders = [[0 for i in range(n)] for i in range(m)]
    # 计算必发订单
    for i in range(m):
        for j in range(n):
            # 计算当前订单最晚发的时间
            must_round = min(i + order_time[i][j], m) - 1
            orders[must_round][j] += order_num[i][j]
    return orders
```

图 3.2 必发订单生成算法实现代码

3.3 合法回路寻找算法

寻找合法回路可以通过对每个回路进行检查，从而确定是否将回路添加到合法回路序列中。首先判断回路长度是否小于 *max_length* 以及节点数量是否小于无人机最大载重 *max_take*(保证每个卸货点至少能分配一个订单)。同时还要避免出现相同的回路，只是方向相反的情况出现，只有回路及其逆序都不在已有回路中才添加。实现代码如图 3.3 所示。

```python
def find_cycles(centers, unloads, max_length, max_take):
    """ 找到所有以 'centers' 点为起点和终点的回路，
        经过至少一个给定坐标点，
        长度小于 'max_length'且节点数小于max_take的所有回路
        输入：centers 配送中心
             unloads 卸货点
             max_length 无人机最长飞行距离
             max_take 无人机最大载重
        输出：
             cycles合法回路"""
    n = len(unloads)
    cycles = set()  # 使用集合存储唯一的回路及其长度
    seen_paths = set()  # 记录已经添加过的路径及其逆序

    # 遍历所有起点
    for s in range(len(centers)):
        # 遍历所有坐标的排列，考虑每个可能的回路
        for perm in itertools.permutations(list(range(0, n))):
            # 从起点开始构建回路，经过排列中的每个坐标点，再回到起点
            for i in range(n):
                path = list(perm[:i + 1])
                size = len(path)
                length = 0
                # 计算回路的总长度
                for j in range(size - 1):
                    length += distance(unloads[path[j]], unloads[path[j + 1]])
                length += distance(unloads[path[0]], centers[s]) + distance(unloads[path[size - 1]], centers[s])
                # 检查回路是否有效且长度小于 max_length
                if length < max_length and size <= max_take:
                    # 将路径和其逆序分别转换为元组
                    path = [s] + path + [s]
                    forward_path = tuple(path)
                    backward_path = tuple(path[::-1])
                    # 如果路径和其逆序都没有出现过，则将路径及其逆序添加到集合中
                    if forward_path not in seen_paths and backward_path not in seen_paths:
                        cycles.add((s, forward_path[1:-1], length))
                        seen_paths.add(forward_path)
                        seen_paths.add(backward_path)
    return cycles
```

图 3.3　寻找合法回路算法实现代码

3.4　基于贪心的单轮订单派送算法

3.4.1　整体流程

对于第 i 轮必须要发送的订单 $\{O_{i1}, O_{i2}, ...O_{iN}\}$，派送几架无人机，每架无人机在每个卸货点的投放订单数量如何分配是此问题核心。对此，我设计了一种基于贪心思想的算法，即每次派出一架无人机时，通过所有卸货点剩余的必发订单数量来对所有回路进行一个排序，找出最好的回路（使用贪心方法排序）。排序前需要删除所有包含无订单的卸货点，因为最优路径的无订单的卸货点无人机一定不会经过。排序后需要根据最优回路，对此次无人机在最优回路上的卸货点进行货物分配。最后将最优路径、物品分情况等信息加入无人机信息列表。循环上述过程直到剩余必发总数量为 0 就得到了第 i 轮所有派送无人机的路径和物品分配情况。基于贪心的单轮订单派送算法整体流程图如图 3.4，实现代码如图 3.5。

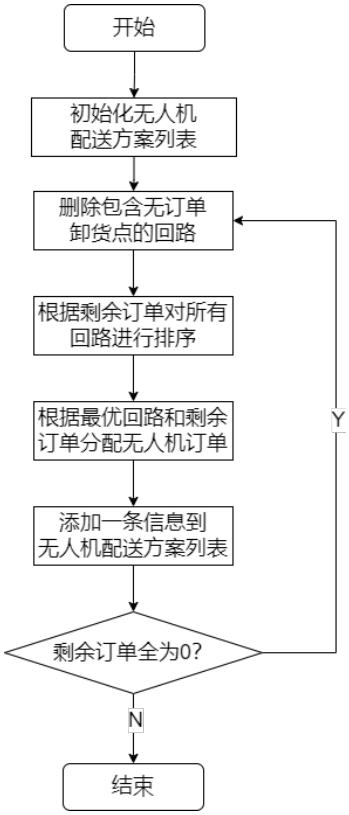

图 3.4　单轮无人机派送算法整体流程图

```python
def delivery(order, cycles, max_take):
    """ 单时刻需发送订单的无人机派送算法
        输入：
            oreder[i]为i卸货点需要派送的订单数量
            cycles是所有回路
            max_take为无人机最大载重
        输出：
            deliver_messages无人机派送回路相关信息：分配中心编号，回路路径，回路距离， 物品分配"""
    tmp_cycles = cycles.copy()
    deliver_messages = []
    # 卸货点还有订单，则还需派出无人机
    while sum(order) != 0:
        # 删除存在无订单卸货点的回路
        tmp_cycles = delete_cycle(order, tmp_cycles)
        # 根据剩余订单对回路排序
        tmp_cycles = cycles_sort(order, tmp_cycles, max_take)
        best_cycle = tmp_cycles[0]
        # 根据最优回路和剩余订单对当前无人机载货进行分配
        center = best_cycle[0]
        path = best_cycle[1]
        path_dis = best_cycle[2]
        act, order = allocation(order, best_cycle[1], max_take)
        # 添加无人机配送方案
        deliver_messages.append((center, path, path_dis, act))
    return deliver_messages
```

图 3.5　单轮无人机派送算法实现代码

3.4.2 删除回路算法

此算法比较简单，只要根据剩余订单数量进行判断即可。若为卸货点订单剩余数量为 0，则删除所有包含此卸货点的所有回路。

删除回路算法代码实现如图 3.6。

```
def delete_cycle(order, cycles):
    """ 删除订单数量为0的卸货点所在的所有回路"""
    cycles_to_remove = set()  # 使用集合来收集需要删除的回路
    for i in range(len(order)):
        if order[i] == 0:
            for cycle in cycles:
                if i in cycle[1]:
                    cycles_to_remove.add(cycle)
    # 删除回路
    for cycle in cycles_to_remove:
        cycles.remove(cycle)
    return cycles
```

图 3.6　删除回路算法实现代码

3.4.3 回路排序算法

回路排序算法的核心是怎么评价一条回路的好坏。假设现在有一条合法回路 $P = \{(c,(u1,u2,...ul)\}(1 \le l \le N)$，其路径总长为 $P\_dis = \sum_{i=1}^{l-1} D_{ui,u(i+1)} + C_{c,u1} + C_{c,ul}$。一个直观的想法是用路径总长与卸货点的数量的比值 $\frac{P\_dis}{l}$ 作为评价标准，但这样没有使用上剩余订单情况的信息，不能很好地评判该回路在当前剩余订单情况下的优劣。

因此考虑采用 $\frac{P\_dis}{\sum_{i=1}^{l} ui}$ 作为评价标准表示了每个订单的平均路径，值越小越好。但又考虑到无人机有最大载重限制，所以还需考虑无人机在该回路上的订单分配情况，如果该无人机在此路径上分配结束后，有 z 个卸货点订单清零了，则除以 z+1（避免出现除 0）。

所以最终的评价标准计算公式为 $F = \frac{P\_dis}{\sum_{i=1}^{l} ui \times (z+1)}$。最后根据计算的评价标准从小到大排序。回路排序算法实现代码如图 3.7 所示。

```python
def cycles_sort(order, cycles, max_take):
    """ 计算cycles权重，并排序返回"""
    weighted_cycles = []
    for cycle in cycles:
        unloads_weight = 0
        unlodas = cycle[1]
        tmp_order = order.copy()
        # 获得无人机经过回路分配后，归零卸货点订单数量z
        act, tmp_order, z = allocation(tmp_order, unlodas, max_take)
        for unload in unlodas:
            unloads_weight += order[unload]
        weight = cycle[2]
        # 计算权重
        if unloads_weight <= max_take:
            weight /= (unloads_weight * (z + 1))
        else:
            weight /= (max_take * (z + 1))
        # 添加权重信息到元组中
        weighted_cycles.append((cycle[0], cycle[1], cycle[2], weight))
    # 排序
    sorted_cycles = sorted(weighted_cycles, key=lambda x: x[3])
    return sorted_cycles
```

图 3.7　回路排序算法实现代码

3.4.4　无人机订单分配算法

对于一条合法回路 $P = \{(c, (u1, u2, ...ul)\}(1 \le l \le N)$ 和一个剩余订单 $\{S_1, S_2, ...S_N\}$，一架无人机最多能完成 max_take 个订单，我们需要考虑如何分配这些订单到卸货点上。我采用的方法是：如果整个回路上订单总数小于等于 max_take，则每个卸货点的订单都可完成；如果如果整个回路上订单总数大于 max_take，则尽可能均匀地分配订单，可以使得订单能清零的卸货点尽可能多的同时还确保有大量订单的卸货点也能较快完成订单。无人机订单分配算法实现代码如图 3.8 所示。

```python
def allocation(order, cycle_unloads, max_take):
    """ 一条回路上一台无人机载重分配
        输入：
            order[i]为i卸货点需要派送的订单数量
            cycle_unloads是回路上的卸货点
            max_take为无人机最大载重
        输出：
            act[j]: 无人机第j个卸货点卸货数量
            order: 剩余订单数量
            zero_num: 分配后订单清零节点数量"""
    n = len(order)
    act = [0] * n
    # 如果一台无人机可以送完回路所有卸货点订单，无人机分配就按订单数量分配
    for unload in cycle_unloads:
        act[unload] = order[unload]
    # 如果一台无人机无法送完回路所有卸货点订单，就尽量均匀地分配到所有订货点
    if sum(act) > max_take:
        i = 0
        act = [0] * n
        while max_take:
            unload = cycle_unloads[i]
            if order[unload] > act[unload]:
                act[unload] += 1
                max_take -= 1
            i = (i + 1) % len(cycle_unloads)
    zero_num = 0
    for i in range(n):
        if order[i] == act[i] and order[i] != 0:
            zero_num += 1
        order[i] -= act[i]
    return act, order, zero_num
```

图 3.8　无人机订单分配算法实现代码

4　实验结果与总结

4.1　实验结果

我设置了 3 个配送中心，8 个卸货点，无人机最大订单携带量为 20，随机产生 15 轮订单（7.5h），其中订单数量在 0-10 之间随机生成，订单级别按照概率分别为 0.8，0.1，0.1 生成紧急、较紧急和一般订单（事实上在此方法中可以直接随机生成必发订单即可）。相关参数具体设置情况如图 4.1 所示。

```
""" 相关参数 """
# 配送中心数量
center_num = 3
# 卸货点数量
unloda_num = 8
# 无人机最大携带物品数量
max_take = 20
# 无人机最大飞行距离(km)
max_dis = 20
# 无人机飞行速度(km/h)
speed = 60
# 订单轮数
order_round = 15
# 设置随机种子以确保结果可重复
np.random.seed(15)
# order_num[i][j]表示第i轮订单，j卸货点的订单数量
order_num = np.random.randint(0, 10, size=(order_round, unloda_num))
# order_time[i][j]表示第i轮订单，j卸货点订单剩余时间,1为0.5小时，3为1.5小时，6为3小时
order_time = np.random.choice([1, 3, 6], size=(order_round, unloda_num), p=[0.8, 0.1, 0.1])
# # centers配送中心坐标
centers = np.array([[2.4, 1.8], [12.1, 2.5], [8.1, 12.2]])
# # unlodas卸货点坐标
unlodas = np.array([[1.3, 6.1],
                    [5.2, 6.1],
                    [4.2, 9.7],
                    [7.1, 3.8],
                    [10.2, 6.5],
                    [11.5, 9.5],
                    [4.5, 12.5],
                    [8.5, 8.4]])
```

图 4.1　相关参数设置情况

对于每轮结果，我做了可视化展示，如图 4.2-4.16 所示。

程序在控制台输出了每轮的结果以及最后总体的相关信息，一共派出 36 无人机，派送距离为 517.19km。控制台输出如图 4.17 所示。

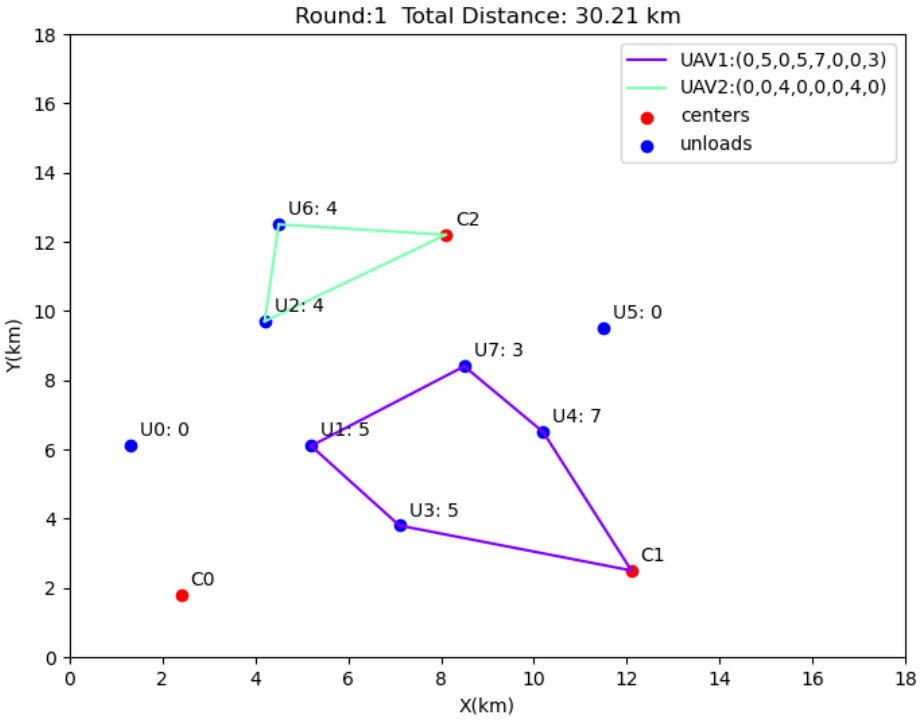

图 4.2　第 1 轮无人机派送结果

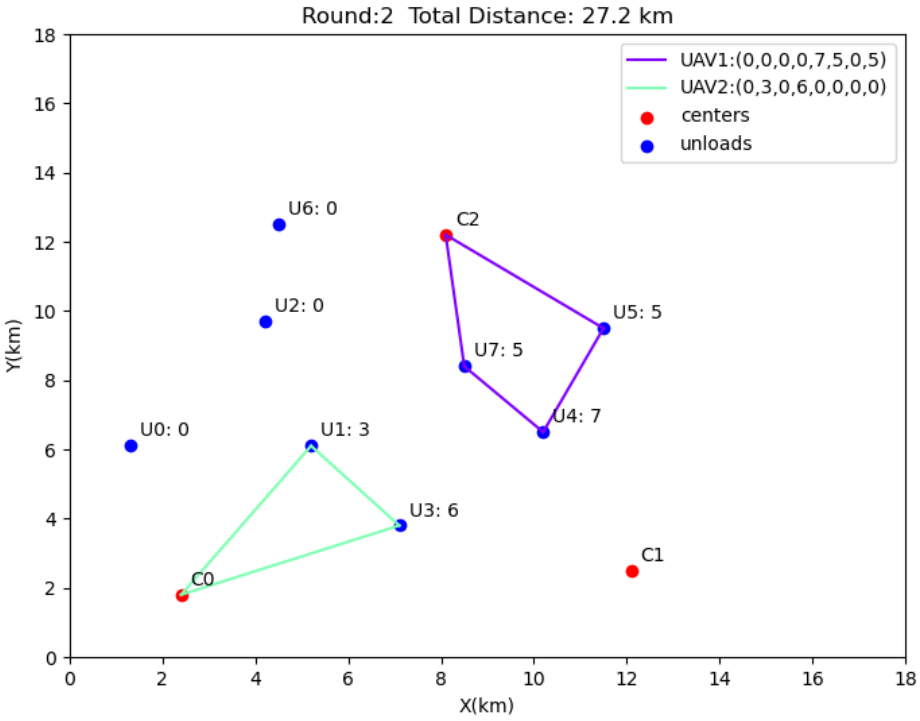

图 4.3　第 2 轮无人机派送结果

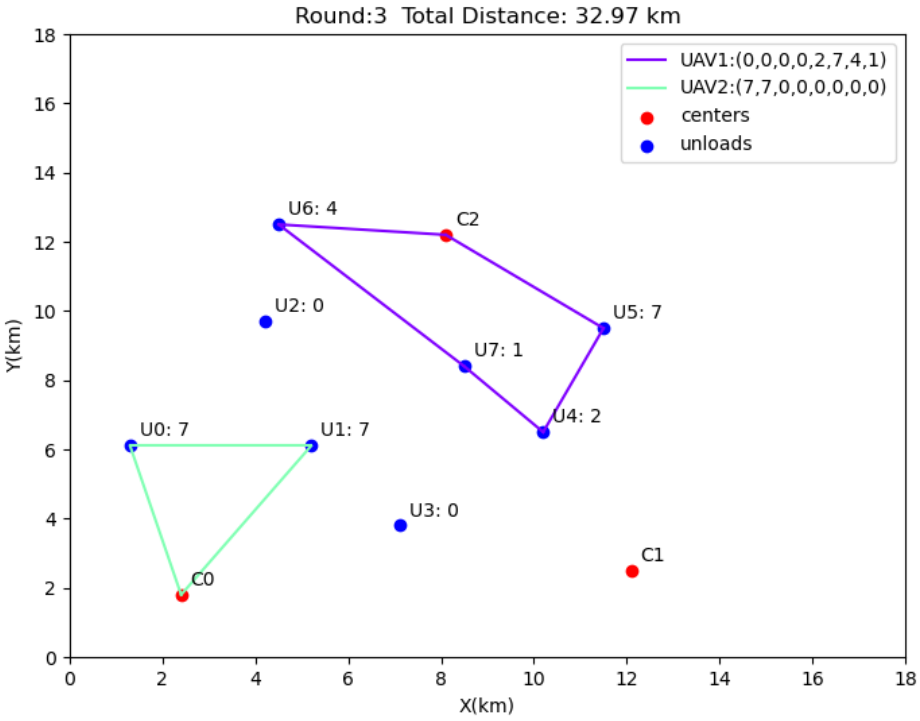

图 4.4　第 3 轮无人机派送结果

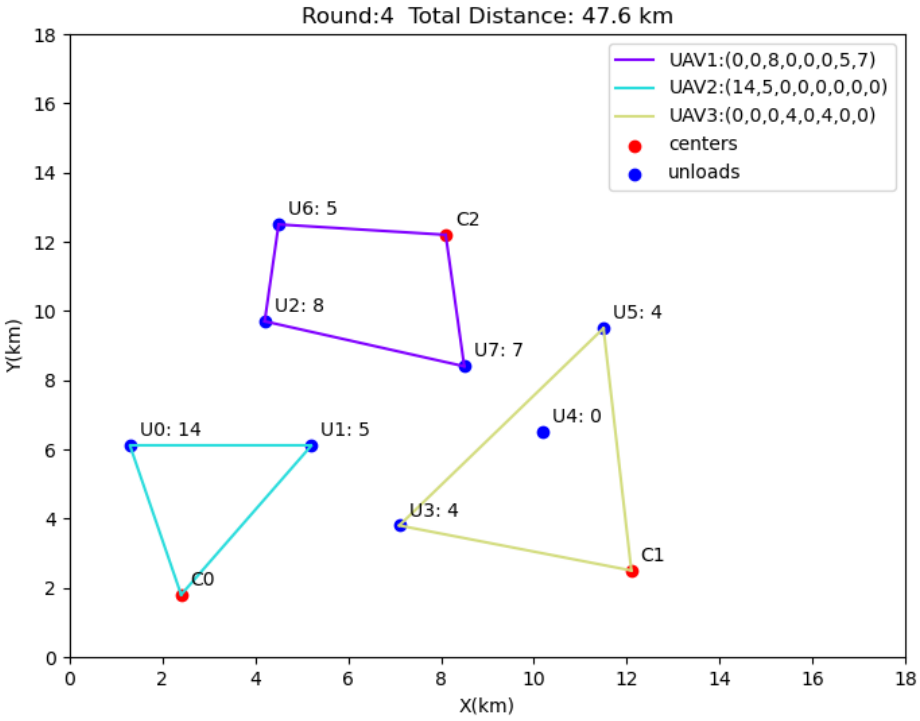

图 4.5　第 4 轮无人机派送结果

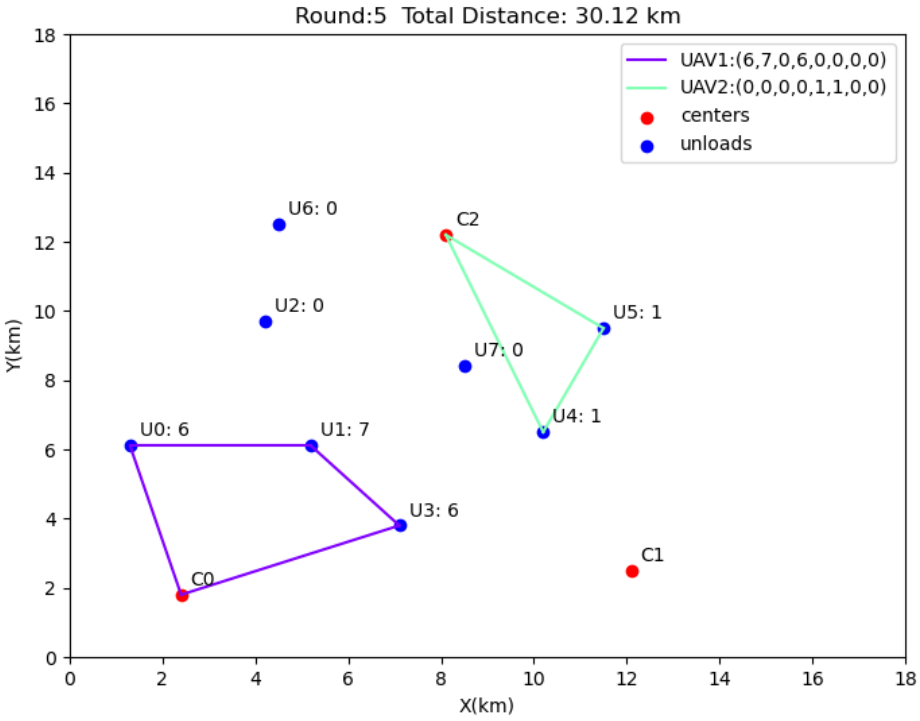

图 4.6　第 5 轮无人机派送结果

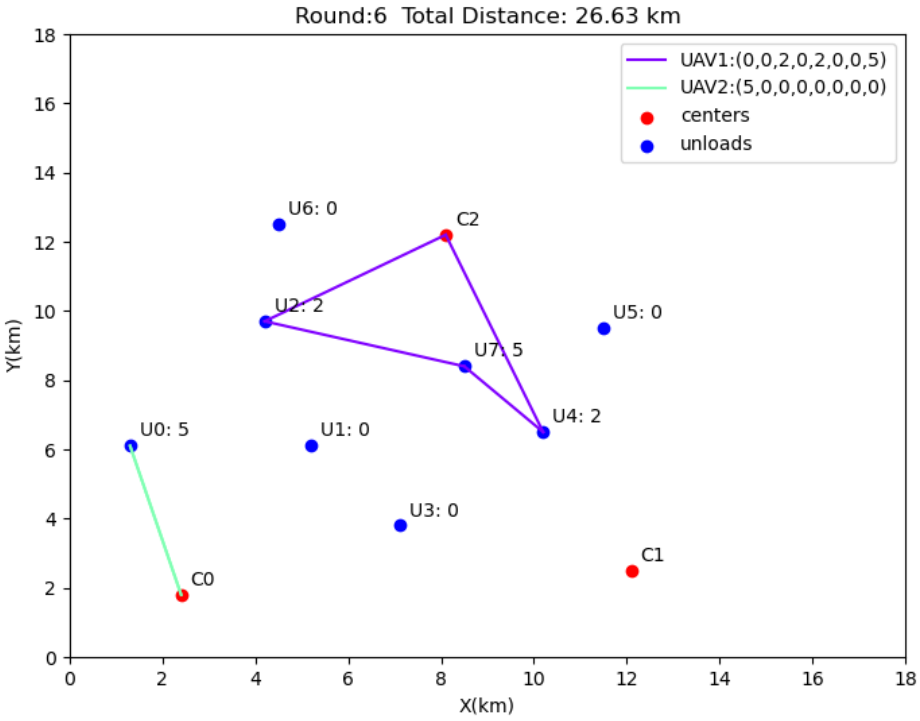

图 4.7　第 6 轮无人机派送结果

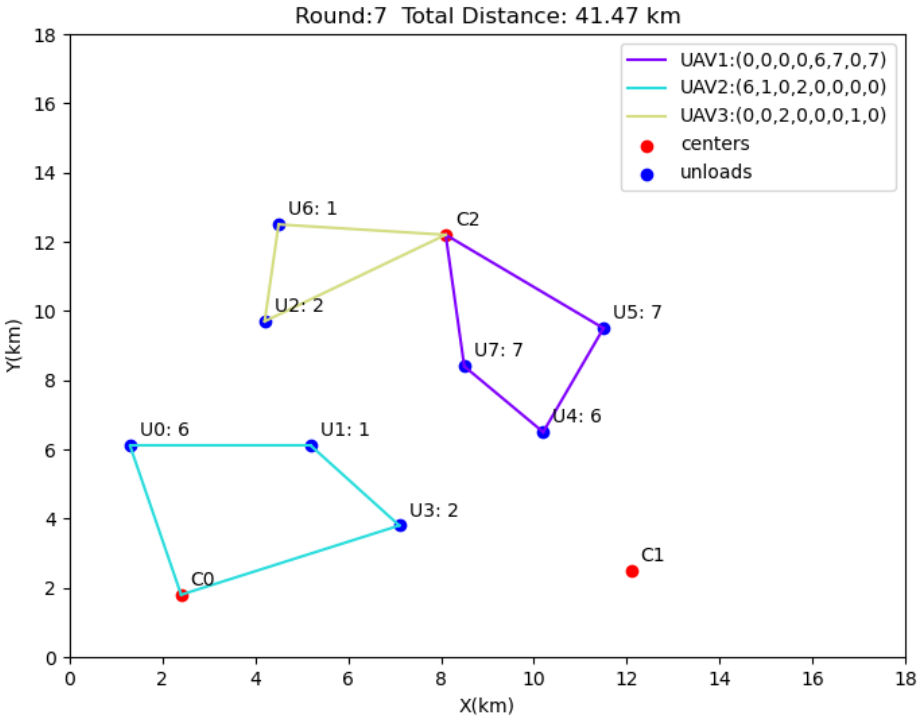

图 4.8　第 7 轮无人机派送结果

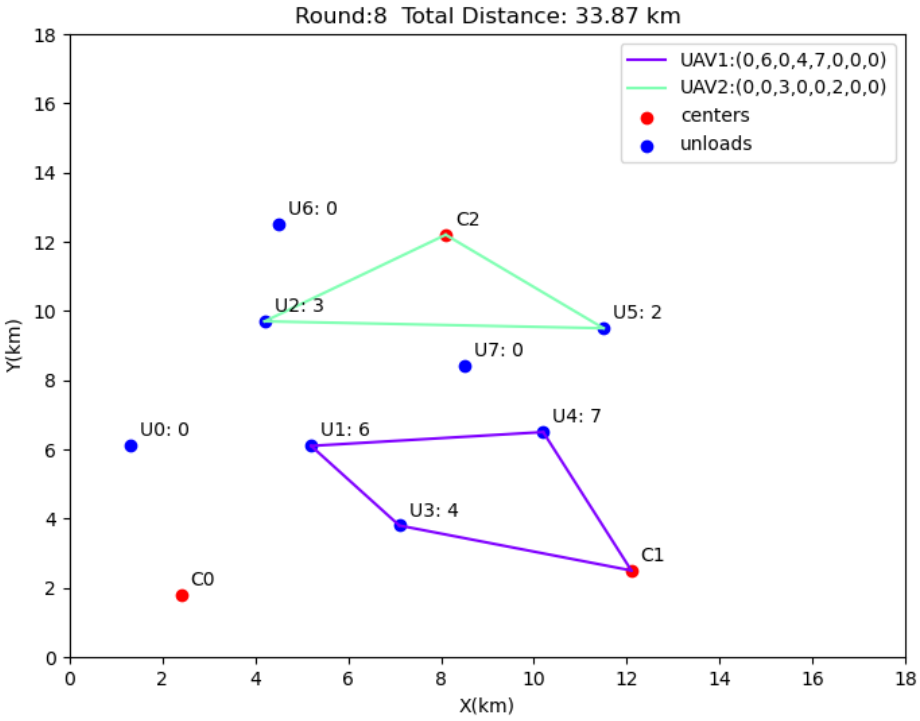

图 4.9　第 8 轮无人机派送结果

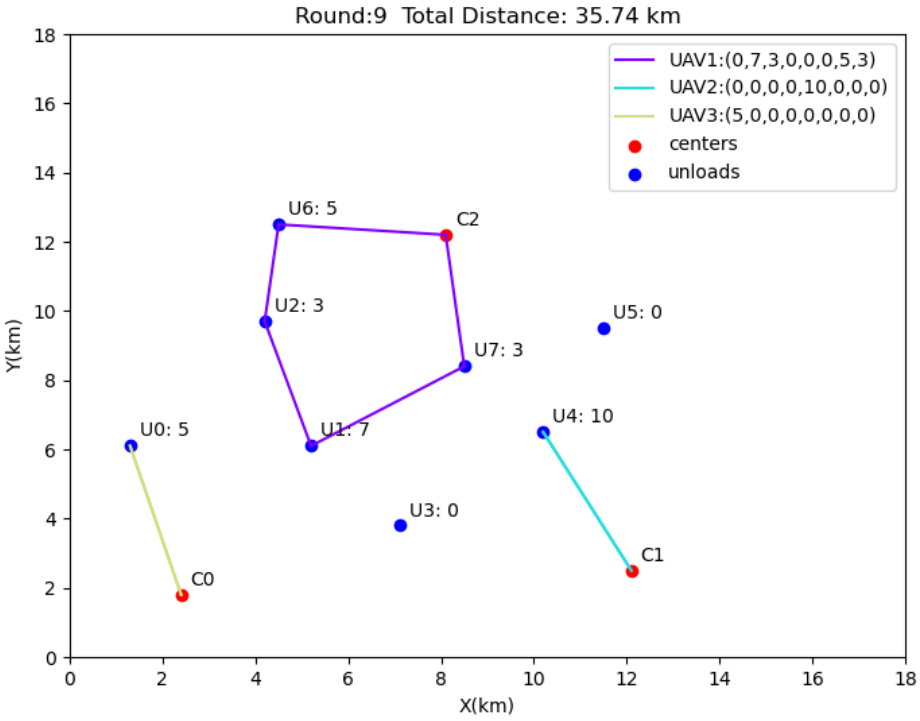

图 4.10　第 9 轮无人机派送结果

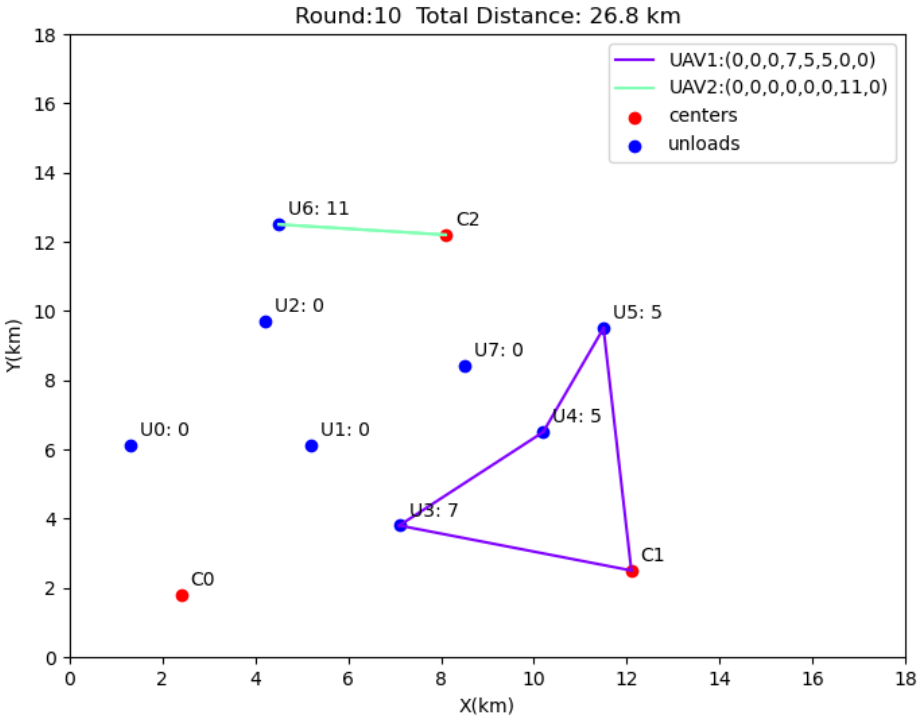

图 4.11　第 10 轮无人机派送结果

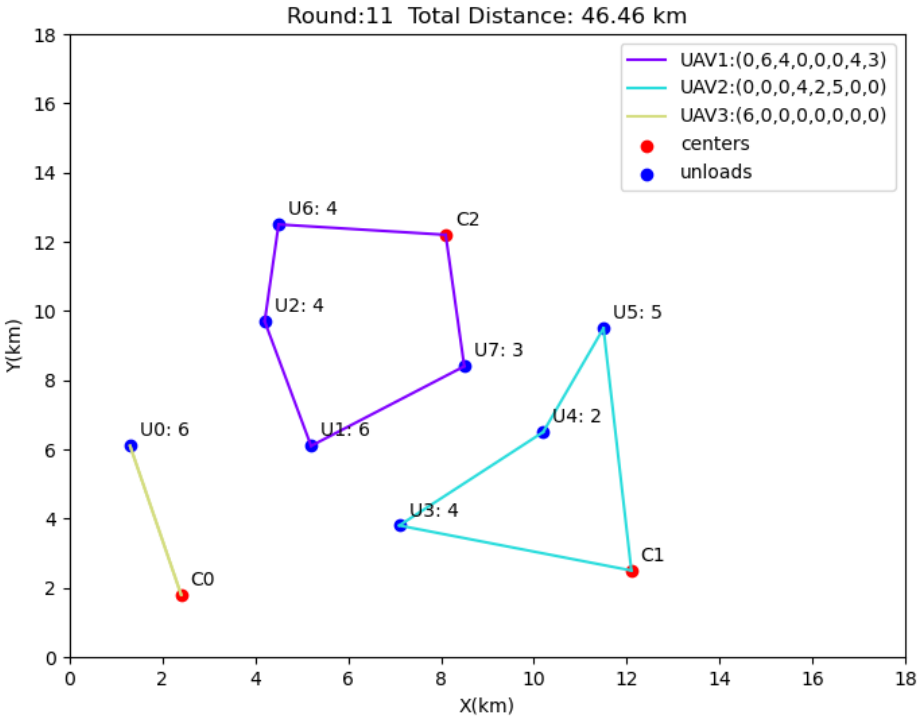

图 4.12　第 11 轮无人机派送结果

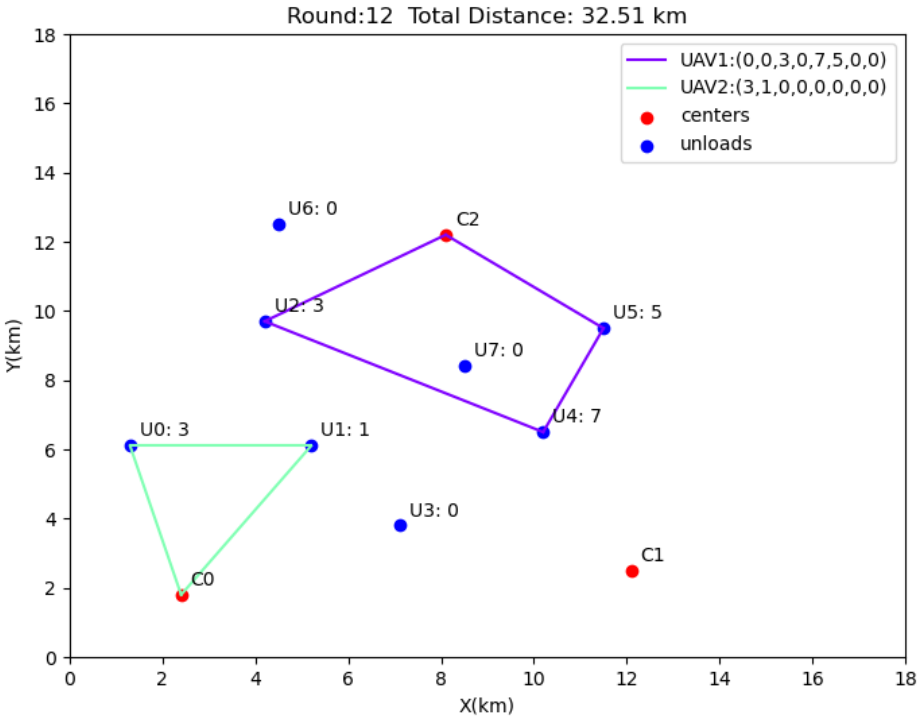

图 4.13　第 12 轮无人机派送结果

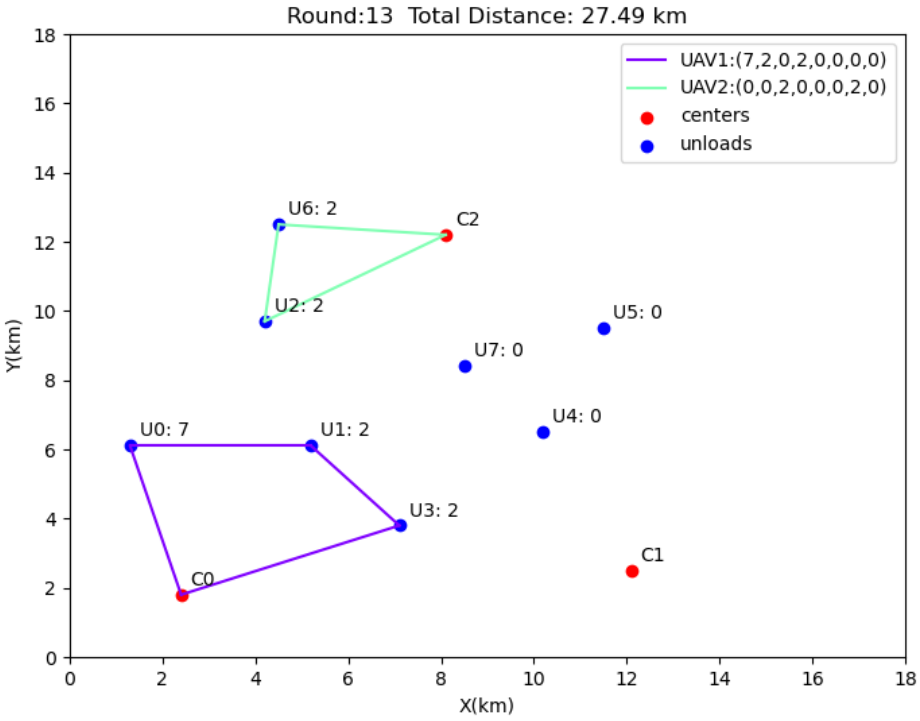

图 4.14　第 13 轮无人机派送结果

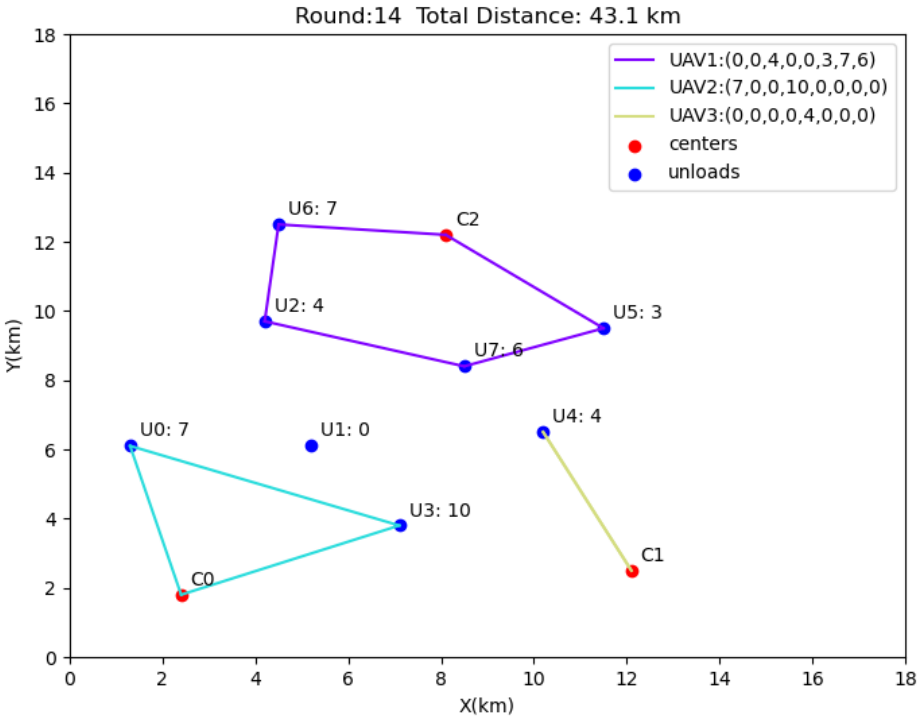

图 4.15　第 14 轮无人机派送结果

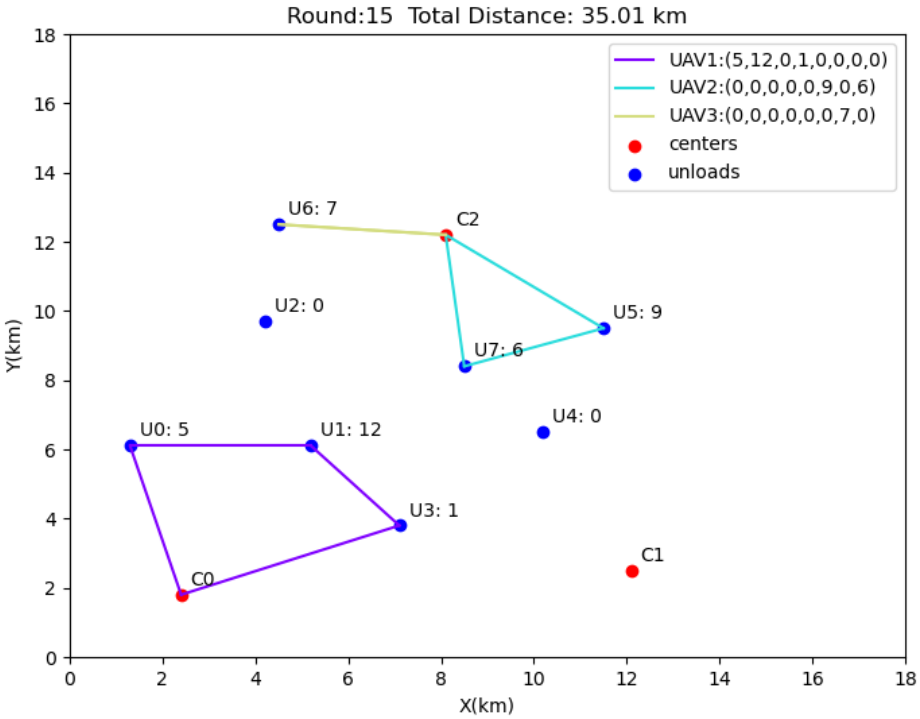

图 4.16　第 15 轮无人机派送结果

```
第14轮共派出3架无人机, 配送总长为43.1 km
###############第15轮无人机配送情况###############
卸货点订单数量: [5, 12, 0, 1, 0, 9, 7, 6]
--------------center0派出一架无人机------------------
无人机卸货方案:
center0 -> (unload0,5) -> (unload1,12) -> (unload3,1) -> center0
配送距离: 16.429592097918544
--------------center2派出一架无人机------------------
无人机卸货方案:
center2 -> (unload5,9) -> (unload7,6) -> center2
配送距离: 11.357962365861132
--------------center2派出一架无人机------------------
无人机卸货方案:
center2 -> (unload6,7) -> center2
配送距离: 7.224956747275376
..................................................
第15轮共派出3架无人机, 配送总长为35.01 km
**************全局情况********************
一共派出36架无人机，派送总距离为: 517.19 km
```

图 4.17　控制台输出

4.2 总结

本次大作业提出的算法主要包括三个关键部分：必发订单生成、合法回路寻找和基于贪心的单轮订单派送算法。首先通过精确的订单生成机制和对不同级别订单的处理，保证了系统在各轮次内的灵活性和高效性。然后通过寻找合法回路并应用基于贪心策略的派送算法，有效地优化了派送路径和无人机资源的利用，从而最大化满足了派送要求。

算法设计中对于回路的评价标准和订单分配策略的细致考量，使得每一步决策都基于当前剩余订单情况进行了合理的优化选择，确保了每轮派送的高效性和可行性。此外，算法中的删除回路和回路排序等辅助功能，进一步增强了系统的鲁棒性和稳定性。

但考虑到实际应用中可能配送中心和卸货点规模较大，再采用纯贪心类的算法，可能不能较快地完成任务。因此在之后的改进中，可以选择加入一些启发性算法，如遗传算法等。这样可以使得算法的实时性更好。

通过本次大作业，我完成了对多配送中心的无人机配送问题的算法设计和实现。本次大作业的完成度较高，在解决整个问题的过程中，通过多方面的思考我不断地完善自己的想法，最终完成了此次大作业，这个过程让我收获颇丰。