《高级人工智能》课程作业

基于遗传算法的无人机配送路径规划

姓名：王秭悦

学号：2023202210114

专业：网络空间安全

班级：学硕 2 班

摘要

无人机可以快速解决最后 10 公里的配送问题。然而，配送路径规划问题是一个 NPC 问题，难以在多项式时间内找到该问题的最优解。并且现实环境还存在多方面的影响因素，比如多配送中心调度问题、订单延迟处理问题都会对增加该问题的复杂性。

本文基于启发式思想的遗传算法，设计并实现了一种高效的无人机配送路径规划算法。该算法支持多中心调度和延迟配送等问题。并且本文设计了一种无人机使用效率评估算法，可以对某条路径对无人机的使用充分程度进行客观的评估，从而帮助系统进行合理决策。本文在 3 个配送中心和 50 个卸货点的分布上模拟并测试了本算法在 24 小时之内的配送规划情况。测试结果显示，所有路径均无订单超时情况出现，且无人机平均使用效率达到了 70%，可见本文设计的基于遗传算法的无人机路径规划问题可以得到较优的可行解。

1 问题描述与分析

1.1 问题描述

无人机可以快速解决最后 10 公里的配送，要求设计一个算法，实现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；

1.2 问题分析

上述问题是一个 NPC 问题，我们很难在多项式时间内求得该问题的最优解。同时，上述问题是一个在线问题，只有在 T 时刻到来的时候，才会知道 T 时刻生成的订单情况。因此，上述问题需要使用近似算法，求解该问题的一个近似解。遗传算法是一种启发式算法，可以类比自然进化过程，逐步淘汰次优解，逐渐靠近最优解。因此，本文采用遗传算法的思想解决无人机配送路径规划。

2 遗传算法基础

遗传算法（Genetic Algorithm, GA）起源于对生物系统所进行的计算机模拟研究。它是模仿自然界生物进化机制发展起来的随机全局搜索和优化方法，借鉴了达尔文的进化论和孟德尔的遗传学说。其本质是一种高效、并行、全局搜索的方法，能自适应地控制搜索过程以求得最佳解。

遗传算法包含两个数据转换操作，一个是从表现型到基因型的转换，将搜索空间中的参数或解转换成遗传空间中的染色体或个体，这个过程称为编码。另一个是从基因型到表现型的转换，即将个体转换成搜索空间中的参数，这个过程称为解码。

遗传算法在求解问题时从多个解开始，然后通过一定的法则进行逐步迭代以产生新的解。

这多个解的集合称为一个种群，记为 p_t，t 表示迭代步。一般地，p_t 中的元素个数在整个演化过程中是不变的，可将群体的规模记为 N。p_t 中的元素称为个体或染色体，在进行演化时，要选择当前解进行交叉以产生新解。这些当前解称为新解的父解，产生的新解称为后代解。

遗传算法中包含了五个基本要素：参数编码、初始群体的设定、适应度函数的设计、遗传操作设计和控制参数设定。

（1）参数编码：由于遗传算法不能直接处理问题空间的参数，因此，必须通过编码将要求解的问题表示成遗传空间的染色体或个体，其由基因按一定结构组成。

（2）初始群体的设定：由于遗传算法是对群体进行操作的，所以必须为遗传操作准备一个由若干初始解组成的初始群体，主要包括随机产生初始种群和确定种群规模。

（3）适应度函数的设计：遗传算法遵循自然界优胜劣汰的原则，在进化搜索中基本上不用外部信息，而是用适应度值表示个体的优劣，作为遗传操作的依据。适应度是评价个体优劣的标准。个体的适应度高，则被选择的概率就高，反之就低。适应度函数是用来区分群体中的个体好坏的标准，是算法演化过程的驱动力，是进行自然选择的唯一依据。改变种群内部结构的操作都是通过适应值加以控制的。

（4）遗传操作设计：遗传操作主要包括选择、交叉和变异。选择操作是从当前群体中按照一定的概率选出优良的个体，使它们有机会作为父代繁殖下一代子孙。交叉操作是指当两个生物体配对或复制时，它们的染色体相互混合，产生一个由双方基因组成的全新的染色体组。变异操作是将个体编码中的一些位进行随机变化，主要目的是维持群体的多样性，为选择交叉过程中可能丢失的某些遗传基因进行修复和补充。

（5）控制参数设定：设定选择概率、交叉概率和变异概率等进化参数。

综上所述，遗传算法的基本步骤如图1所示：

第一步：使用随机方法或者其他方法，产生一个由 N 个染色体（个体）的初始群体；第二步：计算群体中的每一个染色体的适应值；第三步：若满足停止条件，则算法停止，否则以一定概率从种群中随机选择一些染色体构成一个新种群；第四步：以概率 P_c 进行交叉产生一些新的染色体，得到一个新的群体；第五步：以一个较小的概率 P_m 使染色体的一个基因发生变异，形成一个新的群体，返回第二步。

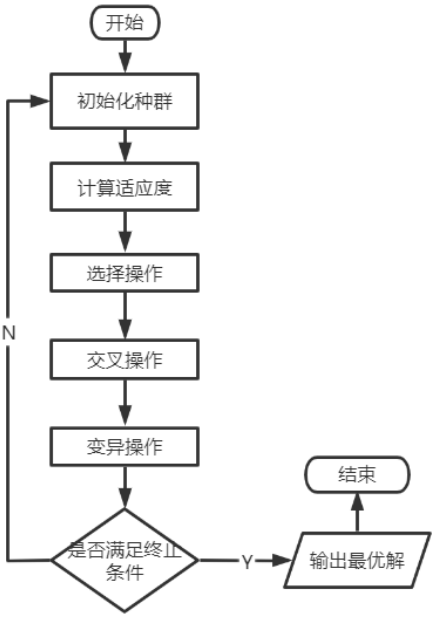


图 1: 遗传算法的基本流程

3 算法实现细节

本文使用遗传算法来求解无人机配送路径规划的最优化问题。为了更加清晰的解释本算法的思路，本节首先以 1 个配送中心为例对算法进行解释。（**注：本文代码全部由本人独立完成**。）

3.1 订单生成

本文使用一个 4 元组来表示各个卸货点产生的订单。

$$(ID,\ (x,y),\ priority,\ time_{order})$$

其中，ID 表示该订单的唯一标识号。(x,y) 唯一对应一个卸货点的坐标。$priority$ 标识该订单的优先级，优先级共分为三个等级，分别是优先级 1（紧急，0.5 小时内送达即可），优先级 2（较紧急，1.5 小时内送达即可），优先级 3（一般，3 小时内送达即可）。$time_{order}$ 表示订单生成的时间。

3.2 染色体编码

对于一个配送中心而言，一个染色体就表示一个配送方案，其中包括了多个无人机飞行路径。本文使用一个 2 维的列表表示染色体的具体结构。如图3所示，这个 2 维矩阵的每一个

元素都是一个订单，代表了染色的每一个基因。每一行代表一个无人机的配送顺序。

由于订单都是随机产生的，因此本文中的染色体第 0 维和第 1 维的长度都不是固定的。一架无人机可以承载 1～n 个物品，代表着染色体第 0 维的长度会在 0 ～ n 这个区间变化。每个配送中心在每次决策时派出的无人机数量是不定的，代表着染色体的第 1 维长度是不固定的。

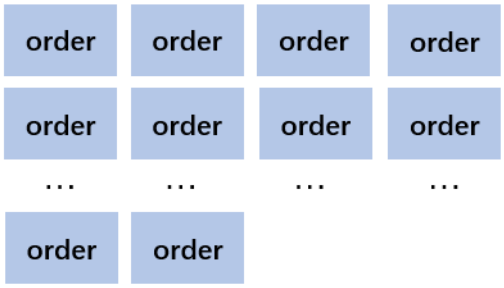


图 2: 染色体结构

3.3 初始种群设定

本文采用随机的方式生成初始种群。将当前时间步生成的所有订单随机排列填入染色体中。并且不需要将每一行都填满，也就是说每一个无人机的配送路径长度都是不固定的。代码如下：

```
# 初始种群生成
def generate_initial_population(orders, population_size):
    population = []
    for _ in range(population_size):
        random.shuffle(orders)
        individual = []
        i = 0
        while i < len(orders):
            route_length = random.randint(DRONE_CAPACITY-1, DRONE_CAPACITY)
            route = orders[i:i+route_length]
            individual.append(route)
            i += route_length
        population.append(individual)
    return population
```

3.4 适应度函数的设计

适应度是评价个体优劣的标准。本文设计了一个适应度评价算法，可以全方面的对一个个体代表的路径进行评估。本文设计的适应度算法有以下要点：

(1) 本算法以当前时间步所有无人机的飞行距离总和作为适应度分数。之后的目标即为最小化该适应度分数。

(2) 本算法会遍历每一架无人机的飞行路径，如果某架无人机的飞行距离不足以支撑起返回原始配送中心，则在原有的适应度分数上增加额外的惩罚。增加的惩罚如下面的公式所示，其中 $Penalty_d$ 表示距离惩罚，d 表示某条飞行路径所需要的距离（包括返程距离），D_{max} 表示无人机的最大飞行距离。

$$Penalty_d = (d - D_{max})^2$$

(3) 本算法会遍历每一个订单送达的时间，如果某个订单送达时间超过了其最迟配送时间，则在原有适应度分数上增加额外的惩罚。增加的惩罚如下面的公式所示，其中 $Penalty_t$ 表示时间惩罚，$time$ 表示某个订单从产生到送达所耗费的时长，$Time_{max}$ 表示该订单最迟配送时长。

$$Penalty_t = (time - Time_{max})^2$$

$$time = time_{elapsed} + time_{delay}$$

其中 $time$ 由两部分时间组成，一部分是无人机从配送中心出发到制定卸货点所需要的时长，另一部分是该订单延迟处理的时长。本文设计的遗传算法支持对某些不紧急的订单延迟配送，即当前时间步产生的订单将会拖后到下个时间步在进行配送。

代码如下：

```
# 适应度函数
def fitness(individual, center, current_time):
    total_distance = 0
    time_violation_penalty = 0
    distance_violation_penalty = 0
    delay_penalty = 0

    for orders in individual:
        if not orders:
            continue
        previous_point = center
        time_elapsed = 0

        for order in orders:
            i, point, priority, order_time = order
            route_distance = calculate_distance(previous_point, point)
            total_distance += route_distance
            # 飞行距离超出惩罚
            if route_distance > DELIVERY_RADIUS:
```

```
                distance_violation_penalty += (route_distance - DELIVERY_RADIUS) * 10
            time_elapsed += route_distance / DRONE_SPEED
            time_delay = current_time - order_time
            # 订单超时惩罚
            if time_elapsed+time_delay > TIME_LIMITS[priority]:
                time_violation_penalty += (time_elapsed+time_delay -
                                           TIME_LIMITS[priority]) * 10
            previous_point = point
        # 回到配送中心
        route_distance = calculate_distance(previous_point, center)
        total_distance += route_distance
        # 飞行距离超出惩罚
        if route_distance > DELIVERY_RADIUS:
            distance_violation_penalty += (route_distance - DELIVERY_RADIUS) * 10

    weighted_total_distance = (total_distance +
                               time_violation_penalty +
                               distance_violation_penalty +
                               delay_penalty)
    return weighted_total_distance  # 返回加权总距离，最小化该值
```

3.5 选择

本文使用了锦标赛算法算法。锦标赛选择算法每次从种群中取出一定数量个体，然后选择其中最好的一个进入子代种群。重复该操作，直到新的种群规模达到原来的种群规模。具体流程为：(1) 从群体中随机选择 M 个个体，计算每个个体的目标函数值 (本次实验选取 M=3)；(2) 计算选择出来的每个个体发的适应度；(3) 选择适应度分数最小的个体 (本文将适应度分数设置为加权飞行距离，因此较小的分数代表较优的个体)。代码如下：

```
def tournament_selection(population, fitnesses, tournament_size=3):
    selected = []
    for _ in range(len(population)):
        tournament = random.sample(list(zip(population, fitnesses)), tournament_size)
        winner = min(tournament, key=lambda x: x[1])
        selected.append(winner[0])
    return selected
```

3.6 交叉

本文采用顺序交叉算法。首先将二维的染色体拉平成一维，如图3所示。然后执行顺序交叉操作。之后再按照原先染色体中每条路径的长度，将新的染色体还原成和原先一样的维度。

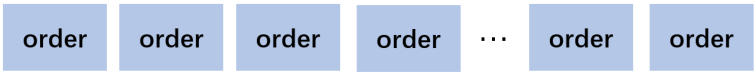


图 3: 染色体结构

顺序交叉（Order Crossover, OX）是一种基于顺序保持的交叉操作，适用于遗传算法中的路径优化问题。OX 可以确保交叉后生成的子代染色体保持父母染色体中的订单顺序，并且不会出现重复订单，如图4所示。

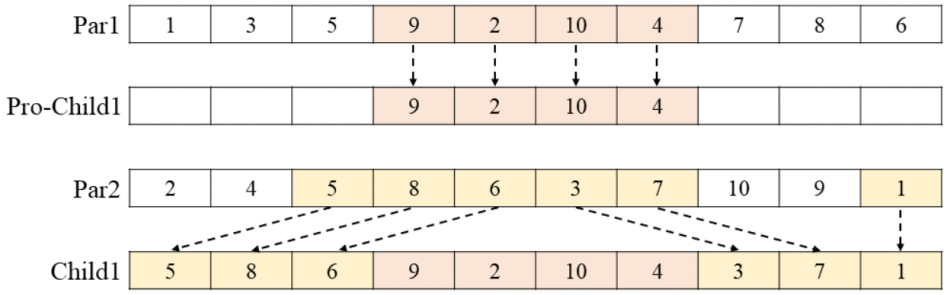


图 4: 顺序交叉

实现代码如下：

```
def ox(route1, route2):
    size = min(len(route1), len(route2))
    # 选择两个交叉点
    p1, p2 = sorted(random.sample(range(size), 2))
    child1, child2 = [None] * size, [None] * size
    # 拷贝到子代
    child1[p1:p2], child2[p1:p2] = route1[p1:p2], route2[p1:p2]
    # 填补其他剩余位置，保持顺序
    current_pos1, current_pos2 = p2, p2
    for i in range(size):
        if route2[(i + p2) % size] not in child1:
            child1[current_pos1 % size] = route2[(i + p2) % size]
            current_pos1 += 1
        if route1[(i + p2) % size] not in child2:
            child2[current_pos2 % size] = route1[(i + p2) % size]
            current_pos2 += 1
    return child1, child2

def order_crossover(parent1, parent2):
    # 染色体拉长
    parent1_merged = merge_lists(parent1)
    parent2_merged = merge_lists(parent2)
    # 顺序交叉
    child1_merged, child2_merged = ox(parent1_merged, parent2_merged)
```

```
    # 染色体还原
    child1 = unmerge_lists(child1_merged, parent1)
    child2 = unmerge_lists(child2_merged, parent2)
    return child1, child2
```

3.7 变异

变异操作会将路径中两个基因的位置进行交换，从而保证一个基因只出现一次。代码如下：

```
def mutate(individual, mutation_rate=0.01):
    new_individual = []
    for route in individual:
        if random.random() < mutation_rate:
            i, j = random.sample(range(len(route)), 2)
            route[i], route[j] = route[j], route[i]
        new_individual.append(route)

    return new_individual
```

3.8 订单延迟配送处理

在每个决策时间步，会存在一些紧急程度不高的订单。如果一架无人机要单独配送一个不太紧急的订单，会导致无人机的使用效率不高，造成资源的浪费。

3.8.1 无人机使用效率评估算法

因此，本文设计了一个无人机使用充分程度评估算法，用于评估一条配送路径对无人机的使用效率。当一条路径对无人机的使用充分程度过低时，将会将该路径上的订单延迟配送。本文设计的无人机使用效率评估分数具体如下：

$$Grade = (w_1 \times G_{num} + w_2 \times G_{distance} + w_3 \times G_{priority})/(w_1 + w_2 + w_3)$$

$$G_{num} = \frac{\text{无人机实际载货数量}}{\text{最大载货数量}}$$

$$G_{distance} = \frac{\text{配送路径距离}}{\text{无人机最大飞行距离}}$$

$$G_{priority} = \frac{\text{高优先级订单数}}{\text{无人机实际载货数量}}$$

(1) G_{num} 表示载货率。无人机一次飞行中，携带的货物越多，该路径对无人机的使用效率越

高。

(2) $G_{distance}$ 表示飞行距离利用率。无人机一次飞行中，飞行路径越长，该路径对无人机的使用效率越高。

(3) $G_{priority}$ 表示高优先级货物比例。无人机一次飞行中，携带的高优先级货物越多，说明该路径对无人机的使用效率越高。根据本文的实验测试，$w_3 = 2$, $w_1 = w_2 = 1$ 时效果较好。

具体代码如下：

```
def grade_of_usage(individual, center):
    grade = []  # 存放某个个体中每条路径的无人机使用充分程度
    for route in individual:
        previous_point = center
        route_distance = 0
        priority_num = 0

        for order in route:
            i, point, priority, order_time = order
            # 计算某架无人机的飞行距离
            distance = calculate_distance(previous_point, point)
            route_distance += distance
            # 计算某架无人机高优先级货物数量
            if priority == 1:
                priority_num += 1
            elif priority == 2:
                priority_num += 0.5
            previous_point = point
        distance = calculate_distance(previous_point, center)
        route_distance += distance
        # 计算某架无人机实际载货数量
        cargo_num = len(route)
        # 计算无人机使用效率分数
        grade_route = {}
        grade_route['cargo_num'] = cargo_num
        grade_route['route_distance'] = route_distance
        grade_route['priority_num'] = priority_num
        grade_route['grade_route'] = (grade_route['cargo_num'] / DRONE_CAPACITY
                                      + grade_route['route_distance'] / DELIVERY_RADIUS
                                      + 2*grade_route['priority_num'] / cargo_num)/4
        grade.append(grade_route)
    return grade
```

3.8.2 订单延迟算法

此外，本文还设计了订单延迟算法。遗传算法迭代得到的最优路径后，该算法会对该路径进行打分，计算最优个体中的每一条路径的分数，找到分数最小的路径。如果该路径中仅

包含低优先级的订单，并且路径中的订单数少于无人机最大载货量，则将该路径中的所有订单延迟到下一个决策时间步在进行处理。**某个若经过一次延迟处理，则其优先级会上升一个等级**。代码如下：

```
def delay_orders(individual, grade_individual):
    min_grade = min(grade_individual, key=lambda d: d['grade_route'])
    min_index = grade_individual.index(min_grade)
    min_grade_route = individual[min_index]
    if len(min_grade_route) == 4:
        return None
    for order in min_grade_route:
        if order[2] == 1:
            return None
    return min_grade_route
```

3.9 多配送中心

现实环境中存在多个配送中心，本文假设配送中心和卸货点的分布情况如图5所示，一共有 3 个配送中心和 50 个卸货点。假设每架无人机的最大飞行距离为 20Km，因此，每个配送中心最远可以覆盖 10Km 以内的区域。超过该范围的卸货点将无法接受该配送中心的服务。故，每个配送中心仅负责距离其 10Km 以内的订单。

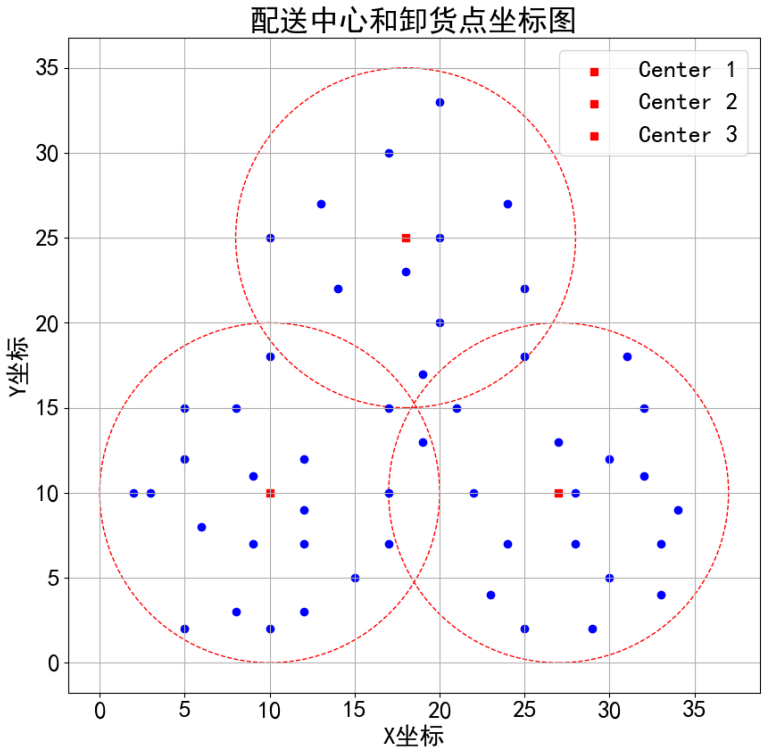


图 5: 配送中心和卸货点坐标图

此外，不同配送中心之间的覆盖范围会存在重叠的现象。针对这一现象，本文采用**就近原则**，由距离最近的一个配送中心负责该卸货点的订单，从而尽可能减少不同配送中心间的调度问题。

3.10 配送路径规划整体流程

图6给出了本文无人机配送路径规划的整体流程。图中显示了 24 个时间步的运行过程，用于模拟一天 24 小时的情况。(1) 每一个时间步都会随机产生订单。并且在后续的实验中，夜晚的订单数量会相对减少。(2) 随机产生订单后，本文还会考虑上一个时间步遗留的订单，本文会将这部分订单的优先级提升一个等级，防止订单无线延迟。(3) 之后使用本文设计的遗传算法跌倒寻找最优路径。(4) 再者，使用本文设计的无人机使用效率评估指标对最优路径中的每一条子路径进行评估，并使用延迟算法对效率低下的路径执行延迟处理。(5) 如此变得到了 T 时间步的最优路径。

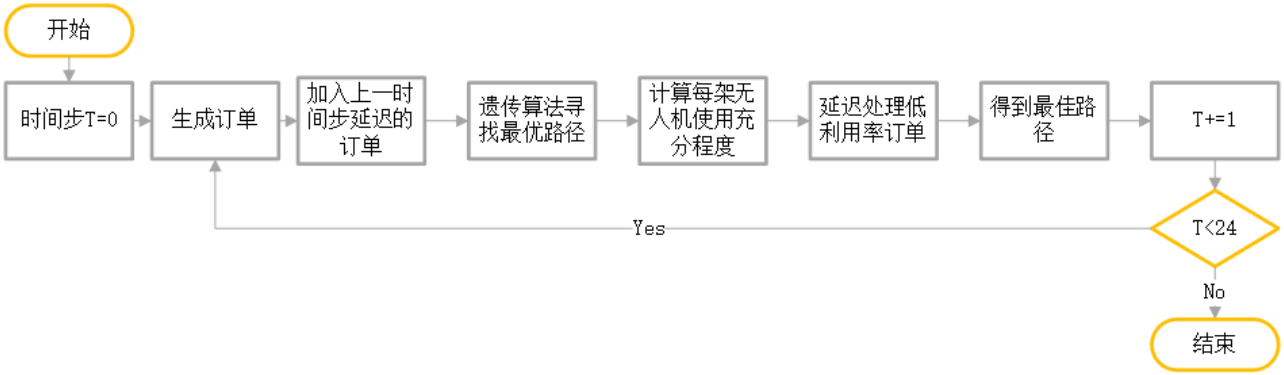


图 6: 配送路径规划整体流程

4 实验与结果

4.1 一个时间步的实验结果

对于其中一个时间步，使用本文设计的随机订单生成函数，生成共 57 个订单：

(0, (5, 12), 2, 0), (1, (19, 13), 3, 0), (2, (2, 10), 3, 0), (3, (12, 12), 2, 0), (4, (10, 2), 1, 0), (5, (5, 15), 3, 0), (6, (9, 7), 2, 0), (7, (12, 3), 1, 0), (8, (15, 5), 3, 0), (9, (12, 12), 1, 0), (10, (17, 15), 1, 0), (11, (9, 11), 1, 0), (12, (9, 7), 1, 0), (13, (12, 3), 3, 0), (14, (15, 5), 3, 0), (15, (9, 11), 2, 0), (16, (12, 3), 1, 0), (17, (17, 10), 1, 0), (18, (10, 2), 2, 0), (19, (2, 10), 3, 0), (0, (30, 5), 2, 0), (1, (27, 13), 1, 0), (2, (25, 18), 2, 0), (3, (32, 15), 1, 0), (4, (34, 9), 1, 0), (5, (32, 15), 1, 0), (6, (33, 4), 1, 0), (7, (25, 18), 3, 0),

(8, (32, 11), 3, 0), (9, (22, 10), 1, 0), (10, (33, 4), 1, 0), (11, (30, 5), 3, 0), (12, (32, 15), 1, 0), (13, (33, 7), 2, 0), (14, (34, 9), 3, 0), (15, (21, 15), 2, 0), (16, (34, 9), 1, 0), (17, (28, 7), 2, 0), (0, (17, 30), 1, 0), (1, (20, 20), 1, 0), (2, (20, 33), 3, 0), (3, (14, 22), 2, 0), (4, (10, 25), 2, 0), (5, (20, 33), 2, 0), (6, (25, 22), 2, 0), (7, (20, 33), 1, 0), (8, (17, 30), 1, 0), (9, (13, 27), 2, 0), (10, (19, 17), 2, 0), (11, (25, 22), 3, 0), (12, (20, 33), 1, 0), (13, (20, 33), 1, 0), (14, (25, 22), 1, 0), (15, (20, 25), 2, 0), (16, (17, 30), 1, 0), (17, (24, 27), 2, 0), (18, (17, 30), 1, 0), (19, (17, 30), 1, 0), (20, (14, 22), 2, 0), (21, (25, 22), 3, 0), (22, (19, 17), 3, 0), (23, (13, 27), 1, 0), (24, (10, 25), 2, 0), (25, (13, 27), 3, 0), (26, (18, 23), 2, 0), (27, (19, 17), 1, 0), (28, (20, 25), 1, 0)

我们对所有的订单按照就近原则分配给对应的配送中心。然后，三个配送中心分别执行本文设计的决策算法实现无人机路径规划。最终可得如图7所示的无人机配送路径。图中每架无人机的配送路径由不同的颜色显示。可以观察到，每个无人机都能以较为合理的路径进行配送并返回原配送中心。

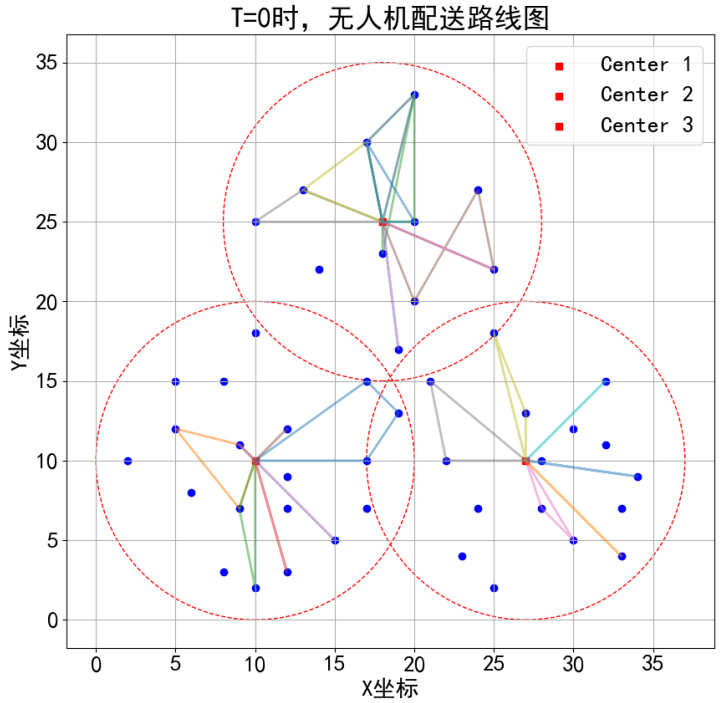
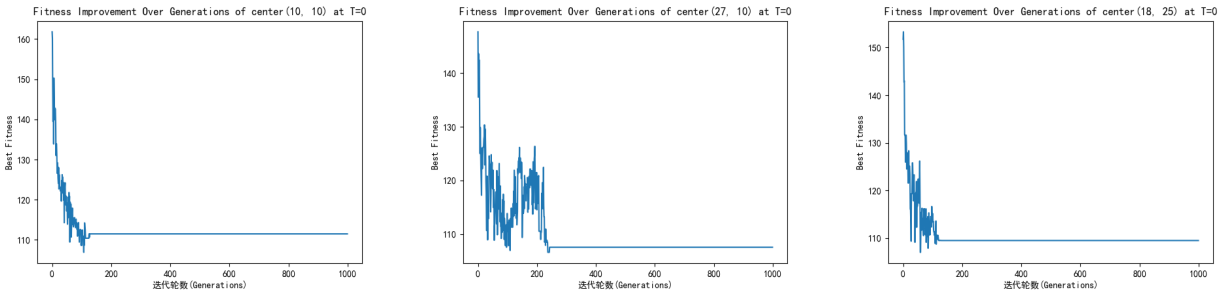


图 7: 无人机配送路径

本文分别绘制了三个配送中心的使用遗传算法迭代优化过程中的适应度函数的下降曲线如图8所示。可以观察到在迭代论述大于 200 时，适应度函数下降幅度放缓甚至不在下降，说

明次数算法已经收敛或者进入局部最优解。



(a) 配送中心 (10,10)　(b) 配送中心 (27,10)　(c) 配送中心 (18,25)

图 8: 适应度函数下降曲线

此外，本文对本次规划的路径进行了定量统计分析，结果如表1所示。从统计分析的数据中，可以发现该路径无订单超时情况，无超最大里程情况，并且无人机使用效率均达到了 80% 以上。可见得出了无人机配送问题的交优的可行解。

表 1: 最佳路径定量分析

配送中心	订单数	配送数	延迟配送数	无人机数	超载数	超距离数	平均使用效率
(10,10)	4	3	1	1	0	0	0.86
(27,10)	23	20	3	7	0	0	0.83
(18,25)	24	21	3	8	0	0	0.80

4.2　24 小时的实验结果

本文还进行了 24 小时的模拟实验。本文将时间离散化，每隔 1 小时就会随机产生订单，同时每隔 1 小时，系统会做出路径规划决策。实验结果如表9b所示。共生成了 869 个订单，共派出了 279 架无人机，均无超载现象，平均无人机评估使用充分程度达到了 70%。路径规划图详见附录。

然而依然会出现个别无人机的路径超过了最大路径限制。分析其原因，猜测是由于遗传算法并没有找到最优解，或者在有限的迭代轮数中实现收敛。为了观察遗传算法的具体性能，本次实验仅展示一次实验的结果。我们可以发现，遗传算法存在不稳定的情况，有时可能会陷入局部最优解。因此，在现实生活中使用遗传算法时，需要进行反复多次实验来防止偶然性。

表 2: 24 小时模拟结果

时间	中心	订单数	配送数	延迟数	无人机数	超载数量	超距离数	平均使用效率
0	(10, 10)	19	17	2	6	0	0	0.677942
	(27, 10)	10	9	1	3	0	1	0.601904
	(18, 25)	1	0	1	0	0	0	null
1	(10, 10)	16	14	2	5	0	0	0.682521
	(27, 10)	23	20	3	7	0	1	0.687866
	(18, 25)	16	15	1	5	0	0	0.676477
2	(10, 10)	5	5	0	2	0	0	0.731067
	(27, 10)	4	3	1	1	0	0	0.576899
	(18, 25)	13	13	0	5	0	0	0.699462
3	(10, 10)	6	6	0	2	0	0	0.947543
	(27, 10)	5	5	0	2	0	0	0.639840
	(18, 25)	19	17	2	6	0	0	0.604118
4	(10, 10)	20	18	2	6	0	0	0.679223
	(27, 10)	5	5	0	2	0	0	0.790357
	(18, 25)	23	21	2	8	0	1	0.646096
5	(10, 10)	3	3	0	1	0	0	0.846164
	(27, 10)	3	0	3	0	0	0	null
	(18, 25)	12	9	3	3	0	0	0.635966
6	(10, 10)	4	4	0	2	0	0	0.693858
	(27, 10)	13	13	0	5	0	1	0.719583
	(18, 25)	17	14	3	5	0	0	0.688132
7	(10, 10)	4	4	0	2	0	0	0.762957
	(27, 10)	7	6	1	2	0	0	0.595872
	(18, 25)	19	16	3	6	0	0	0.676570
8	(10, 10)	23	20	3	7	0	0	0.700202
	(27, 10)	10	7	3	3	0	0	0.672671
	(18, 25)	6	6	0	2	0	0	0.796659
9	(10, 10)	9	9	0	3	0	0	0.722349
	(27, 10)	13	13	0	5	0	0	0.678892
	(18, 25)	7	7	0	3	0	2	0.770114
10	(10, 10)	12	9	3	3	0	0	0.713816
	(27, 10)	2	2	0	1	0	0	0.829568
	(18, 25)	22	20	2	7	0	0	0.631421
11	(10, 10)	12	12	0	4	0	0	0.766068
	(27, 10)	8	5	3	2	0	0	0.667882
	(18, 25)	22	19	3	7	0	0	0.629300

12	(10, 10)	18	15	3	5	0	0	0.642652
	(27, 10)	21	20	1	7	0	0	0.633214
	(18, 25)	6	6	0	2	0	0	0.707919
13	(10, 10)	24	22	2	8	0	0	0.616173
	(27, 10)	9	9	0	3	0	0	0.720014
	(18, 25)	15	12	3	4	0	0	0.686222
14	(10, 10)	16	16	0	6	0	0	0.702928
	(27, 10)	10	8	2	3	0	1	0.662185
	(18, 25)	21	20	1	7	0	0	0.779493
15	(10, 10)	18	16	2	6	0	0	0.600005
	(27, 10)	16	13	3	5	0	0	0.640551
	(18, 25)	11	11	0	4	0	0	0.785959
16	(10, 10)	21	18	3	6	0	0	0.712402
	(27, 10)	22	22	0	8	0	0	0.744095
	(18, 25)	8	6	2	2	0	0	0.618925
17	(10, 10)	15	15	0	5	0	0	0.707040
	(27, 10)	19	17	2	6	0	0	0.640164
	(18, 25)	3	3	0	1	0	0	0.975903
18	(10, 10)	3	3	0	1	0	0	0.592561
	(27, 10)	3	3	0	1	0	0	0.714185
	(18, 25)	12	9	3	3	0	0	0.648783
19	(10, 10)	7	7	0	3	0	0	0.754752
	(27, 10)	4	4	0	2	0	0	0.573406
	(18, 25)	25	23	2	8	0	1	0.695923
20	(10, 10)	22	19	3	7	0	0	0.664004
	(27, 10)	4	2	2	1	0	0	0.411913
	(18, 25)	4	3	1	1	0	0	0.560249
21	(10, 10)	7	5	2	2	0	0	0.709761
	(27, 10)	10	8	2	3	0	1	0.708573
	(18, 25)	14	11	3	4	0	0	0.721884
22	(10, 10)	14	14	0	5	0	0	0.806613
	(27, 10)	16	14	2	5	0	0	0.627496
	(18, 25)	4	3	1	1	0	0	0.706888
23	(10, 10)	5	3	2	1	0	0	0.584730
	(27, 10)	18	18	0	7	0	0	0.729959
	(18, 25)	11	8	3	3	0	1	0.637792

5 总结

无人机可以解决最后 10 公里的配送问题。本文设计了一种基于经典遗传算法的无人机配送路径规划算法，可以实现高效规划无人机配送路径。并设计了一种无人机使用效率评估算法，能够对无人机的利用率做出全方面的评估。本文在 3 个配送中心和 50 个卸货点的分布上模拟并测试了本算法在 24 小时之内的配送规划情况。测试结果显示，所有路径均无订单超时情况出现，且无人机平均使用效率达到了 70%，可见本文设计的基于遗传算法的无人机路径规划问题可以得到较优的可行解。

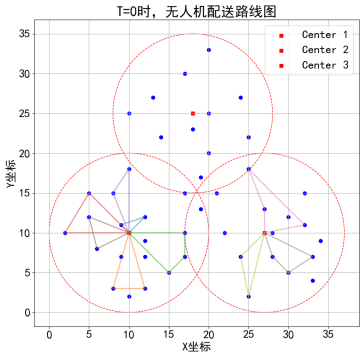
通过高级算法设计与分析这门课，我学到了很多。以前我基本上没有学过算法有关的一些知识，所以在这门课一开始我非常能够从这门课当中学到很多知识。但是，算法这门课毕竟是有门槛的，还是需要一些基础知识才能掌握，所以，一开始上课我都听得很懵。所以，在每次上课之后，我都会复习 PPT，来重温每次上课的内容。虽然，我没法说已经把课程内容掌握的很透彻了，但是我认为已经可以完全理解每次课程 PPT 上的内容了。非常感谢这次课程让我有动机有渠道能够学习到高级算法的一些内容。

在完成大作的过程中，其实遇到了很多问题。比如染色体改怎么设计，适应度函数应该怎么设计，多个配送中心该如何实现调度问题。为了解决无人机配送问题，我在网上查阅了很多资料，也阅读了很多论文。在了解了一些基础设置和方法之后，我发现网络上一些现成的代码都多多少少是有一些问题的。所以，我决定自己搭建系统，完成代码的实现。虽然花费了很多时间，但是这次作业让我对遗传算法和在线算法之类知识的理解不仅仅停留在理论层面上了，而是将其付诸于实践，理解了更多细节和实现的方法。在以后学习和工作当中还能运用。本文的算法还存在很多缺陷，比如当遇到规模较大的数据时运行时间较长，内存开销较大，运行结果不够稳定，有时可能会出现难以收敛的现象。本文的算法完全由我自己独立思考完成，算法流程尚不成熟。后续会更多的参考一些文献，尝试其他路径规划算法，提高算法性能。

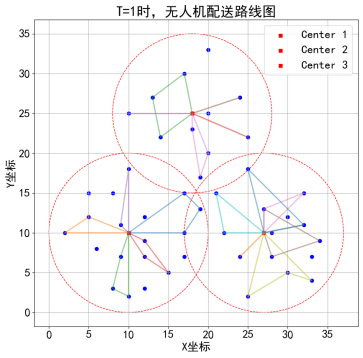
6 附录

图9展示 24 个时间步的路径规划情况。可以观察到，每个无人机都能以较为合理的路径进行配送并返回原配送中心。

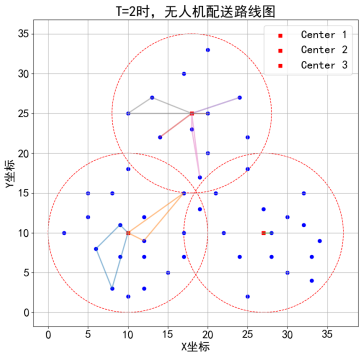
图 9: 24 小时所有时刻的路径规划



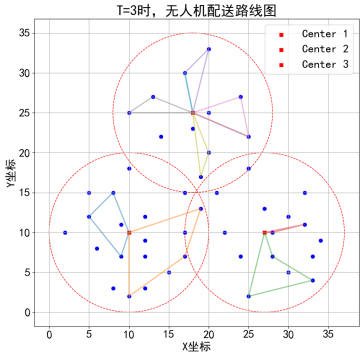
(a) T=0



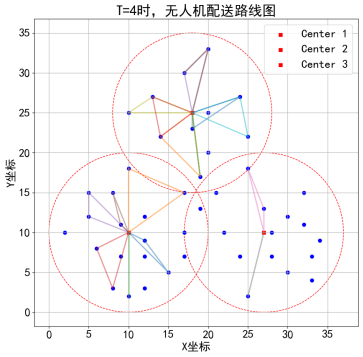
(b) T=1



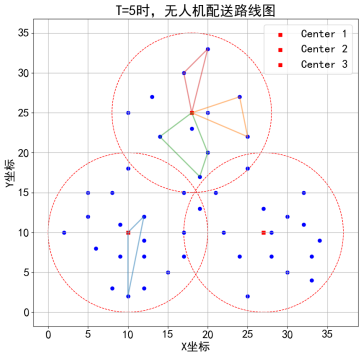
(c) T=2



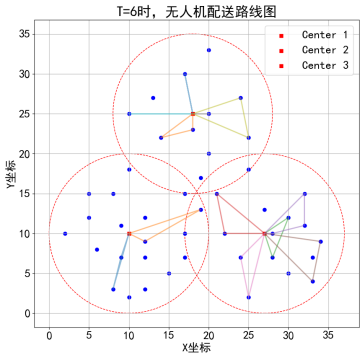
(d) T=3



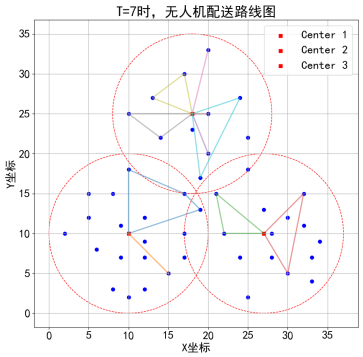
(e) T=4



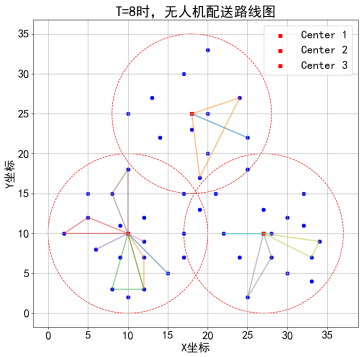
(f) T=5



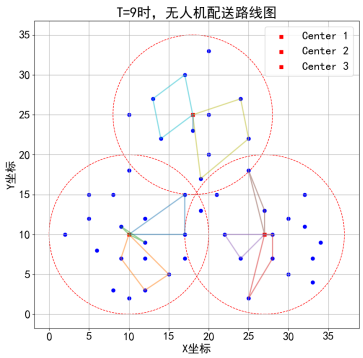
(g) T=6



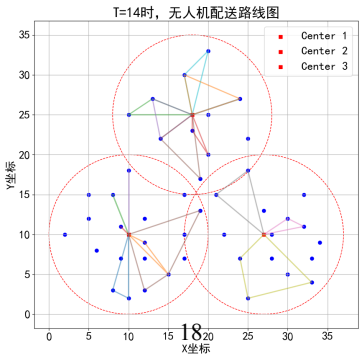
(h) T=7



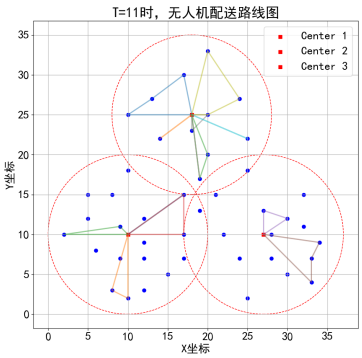
(i) T=8



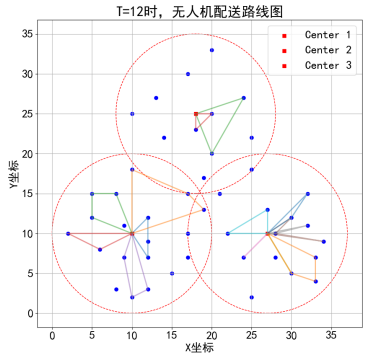
(j) T=9



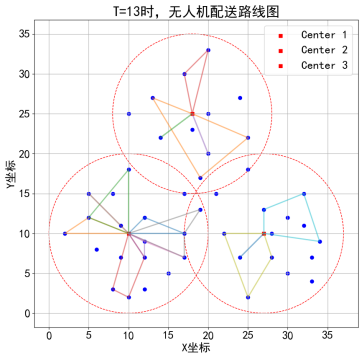
(k) T=10



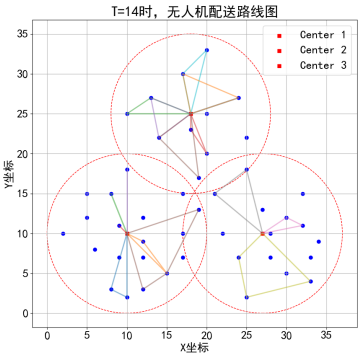
(l) T=11



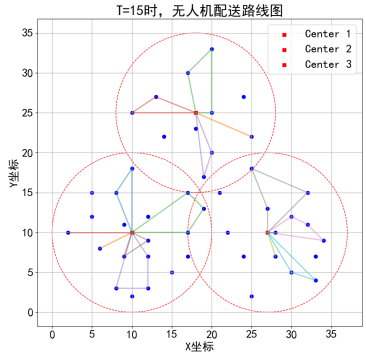
(m) T=12



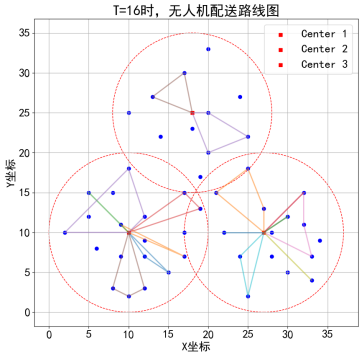
(n) T=13



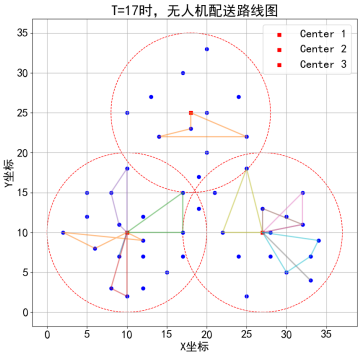
(o) T=14



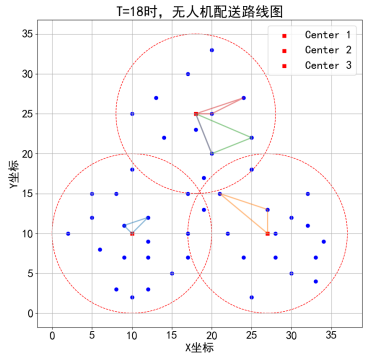
(p) T=15



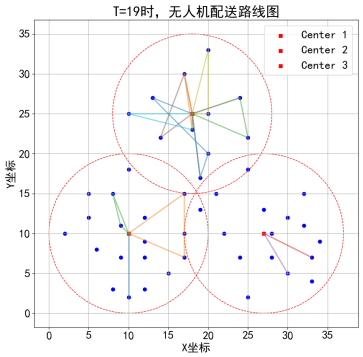
(q) T=16



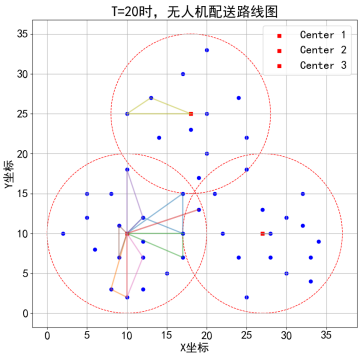
(r) T=17



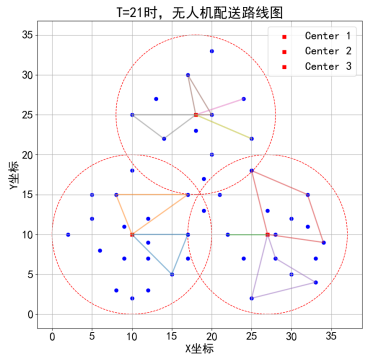
(s) T=18



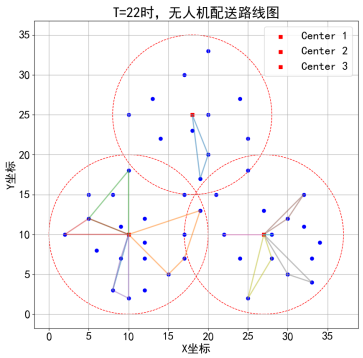
(t) T=19



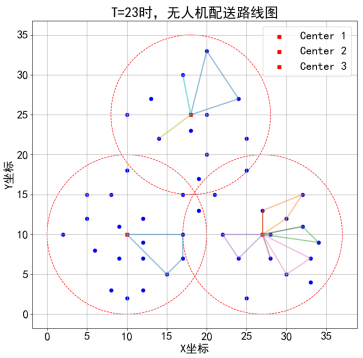
(u) T=20



(v) T=21



(w) T=22



(x) T=23