武汉大学国家网络安全学院

课程报告

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 贺紫怡

学　　号 **2023202210113**

填写时间 **2024** 年 **05** 月 **31** 日

一、 实验概述

1.1 实验背景

电子商务的迅猛发展带来了小型和中型包裹数量的激增，这为快递行业的最后 1 公里配送带来了前所未有的挑战。为了应对这一挑战，业界和学术界都在寻求一种更高效、更环保的配送解决方案。无人机在物流配送领域的应用展现出巨大潜力，具有灵活性、便捷性和低成本的特点。它们能够克服传统车辆配送的限制，如交通拥堵和复杂地形，通过空中飞行缩短配送路线，减少等待时间，降低能耗和成本。此外，无人机配送有助于减少温室气体排放，提高配送效率，并在疫情期间提供了无接触配送的解决方案，减少了人际接触风险。这些优势证明了无人机在快递、外卖和紧急医疗物资配送等方面的广泛应用前景。

当前典型的无人机最后 1 公里配送模式有四种：纯无人机配送模式、无人机与卡车并行配送、无人机与车辆协同配送模式和无人机与车辆混合配送模式。在本次实验中，主要关注第一种模式，即纯无人机配送模式。

1.2 实验要求

本实验要求设计一个算法，采用纯无人机配送模式解决无人机最后一公里配送问题，实现固定区域内的无人机配送的路径规划。在算法中，需要考虑订单的优先级别，优先级别会要求一定的时效。此外还需要考虑无人机的容量和续航问题。

二、相关知识

2.1 多旅行商问题与车辆路径问题

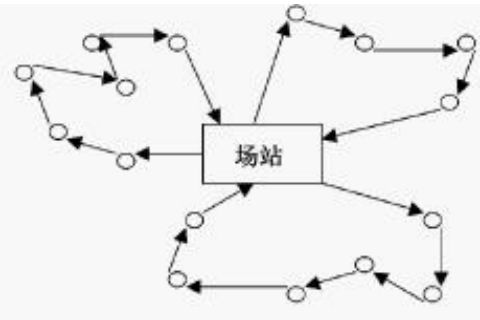
多旅行商问题（Multiple Traveling Salesmen Problem，MTSP）是一种组合优化问题，它是经典的旅行商问题（Traveling Salesman Problem，TSP）的扩展。在 TSP 中，目标是找到一条最短的闭合路径，使得一个旅行商访问所有城市一次后返回起点。而 MTSP 则涉及到多个旅行商和多个起始点，每个旅行商都需要完成类似的任务。

具体来说，假设有 m 个旅行商和 n 个城市；每个城市需要被至少一个旅行商访问一次；每个旅行商都从一个特定的起始点出发，并在访问完所有未被访问

的城市后返回其起始点。目标是最小化所有旅行商路径长度的总和。

车辆路径问题（Vehicle Routing Problem, VRP）最早是由 Dantzig 和 Ramser 于 1959 年首次提出，它是指一定数量的客户，各自有不同数量的货物需求，配送中心向客户提供货物，由一个车队负责分送货物，组织适当的行车路线，目标是使得客户的需求得到满足，并能在一定的约束下，达到诸如路程最短、成本最小、耗费时间最少等目的。

车辆路线问题自 1959 年提出以来，一直是网络优化问题中最基本的问题之一，由于其应用的广泛性和经济上的重大价值，一直受到国内外学者的广泛关注。车辆路线问题可以描述如下：



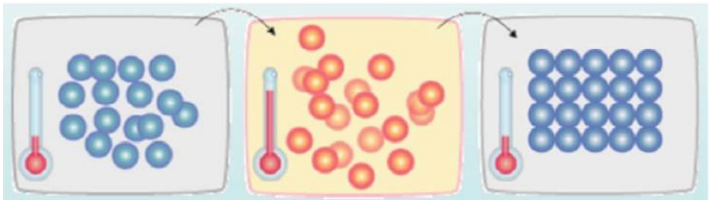
图一 车辆路线问题示意图

设有一场站，共有 M 辆货车，车辆容量为 Q，有 N 位顾客，每位顾客有其需求量 D。车辆从场站出发对客户进行配送服务最后返回场站，要求所有顾客都被配送，每位顾客一次配送完成，且不能违反车辆容量的限制，目的是所有车辆路线的总距离最小。

由此定义不难看出，旅行商问题是 VRP 的特例，由于 Gaery 已证明 TSP 问题是 NP 难题，因此，MTSP 是一个 NP 难问题，VRP 也属于 NP 难题。因为 MTSP 至少和 TSP 一样复杂，而 TSP 本身是已知的 NP 难问题。MTSP 和 VRP 在物流、运输、路径规划等领域有广泛的应用，例如车辆调度、无人机配送路径规划等。

2.2 模拟退火算法

模拟退火算法（Simulated Annealing, SA）是一种用于求解优化问题的概率性算法，灵感来源于金属退火过程中的物理原理。金属退火是指将金属加热到高温，然后缓慢冷却，使其结构逐渐趋于最低能量状态，达到优化效果。模拟退火算法将这一过程应用于优化领域，通过模拟物理退火过程寻找全局最优解。



图二 金属退火过程示意图

模拟退火算法的基本思想是：首先从一个随机初始解开始，并在每一步迭代中，通过随机扰动生成一个新解。新解的好坏通过目标函数来衡量。如果新解比当前解更优，则接受新解；如果新解较差，则以一定概率接受新解，这个概率随着算法的进行逐渐降低。这一接受较差解的机制允许算法跳出局部最优解，从而有更大的概率找到全局最优解。

在具体实现过程中，模拟退火算法包括以下关键步骤：初始温度设定、温度递减策略、邻域解生成策略和接受概率函数。温度递减策略通常采用指数递减或线性递减，而接受概率函数则常用 Metropolis 准则来确定。通过这些步骤的综合作用，模拟退火算法能够在庞大的搜索空间中有效地逼近全局最优解，其适用于求解复杂的组合优化问题，如旅行商问题、生产调度问题等。

其中 Metropolis 准则是指：温度越高，算法接受新解的概率越高。这个根据一定概率选择是否接受差解的方法叫做 Metropolos 准则。

2.3 贪心算法

贪心算法（Greedy Algorithm）是一种逐步构建解决方案的策略，在每一步选择中都做出当前最优的选择，期望通过一系列局部最优选择实现全局最优解。该算法从初始状态开始，通过逐步选择当前最优选项并验证其可行性，直到达到终止条件。尽管贪心算法不总能保证全局最优解，但在许多优化问题中，如最小生成树、单源最短路径、活动选择问题等，能够快速找到近似最优解，且计算复杂度较低。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，在实际应用中广泛使用。

2.4 KD 树

KD 树（K-Dimensional Tree, KD-Tree）是一种高效的多维空间数据结构，适用于快速查询、最近邻搜索和范围搜索等操作。KD 树通过递归地将空间分割成更小的部分，形成层次结构。构建 KD 树时，从根节点开始，选择一个维度并根

据该维度的中位数将数据分成两部分，左子树包含较小的数据点，右子树包含较大的数据点。递归处理子树，直到每个子空间只包含一个数据点或达到指定深度。查询时，利用 KD 树的结构，可以快速排除不可能包含目标的区域，提高搜索效率。KD 树通常用于解决多维空间搜索问题，如范围搜索和最近邻搜索。

三、算法设计

3.1 问题描述及假设

在一个固定区域中，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。

此外将时间离散化，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，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；

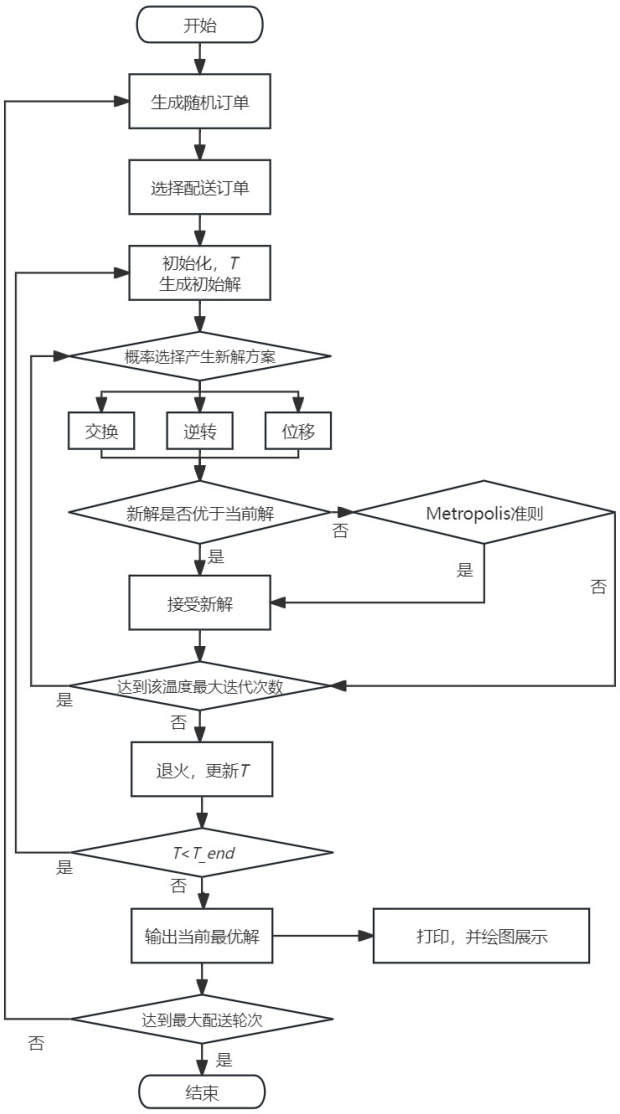
4. 配送中心的无人机数量无限；

5. 任意一个配送中心都能满足用户的订货需求；

最后，我们观察到部分事实并对问题做了如下简化：一架无人机的飞行最远路程为 20 公里，因此每个配送中心最大配送半径为 10 公里，为便于算法设计，假设配送中心均匀分布，因此每个配送中心可以只负责以自己为中心对角线长为 10 公里的正方形区域（在实验代码中，设边长为 14 公里）。

3.2 算法设计及实现

本实验算法是采用分割的思想进行简化算法，通过将多起点的 VRP 问题转换为多个 MTSP 问题。主要采用模拟退火算法计算新解，通过向目标函数中添加约束实现对优先级和无人机容量及最大航程的限制。由于篇幅限制，此处代码不给出注释。具体注释可查看源码。流程图如下：



1. 产生随机订单

根据传入的参数生成随机订单，每个订单均为一个列表[id, x 坐标, y 坐标, 优先级, 该订单货物数量, 该订单生成的时间戳, 该订单剩余等待时间]。其中，时间戳为以 t 为间隔轮次数。其中假设配送中心在(7,7)位置。

```
def gen_random_order(self, count: int, ts: int = 0) -> list:
    orders = [[]]
    orders.append(["s",7.0,7.0,0,0])
    for i in range(count):
        order = ["o_"+str(i+ts*count)]
        x = 7.0
        y = 7.0
        while x == 7.0 and y== 7.0:
            x = round(random.uniform(0, 14), 2)
            y = round(random.uniform(0, 14), 2)
        order.append(x)
        order.append(y)
        order.append(random.randint(0, 2))
        order.append(random.randint(1, 5))
        order.append(ts)
        order.append(self.priority[order[3]])
        orders.append(order)
    return orders
```

2. 订单分拣

算法中需要对订单进行分拣，可以使部分优先级较低的订单推后配送，如此便可以解决某些在先前的订单可以与后产生的相近订单一起配送，此操作可以减少配送数量。此外，为了避免订单堆积至后续轮，可以设置当前轮次最少配送订单数量，而这也应当是在实际中考虑的一点。

```
def select_orders(self, orders: list, ts: int) -> list:
    res_orders = [[],orders[1]]
    orders.pop(0)
    orders.pop(0)
    waiting_orders = []

    for j in range(len(self.waiting_delivery_orders)):
        self.waiting_delivery_orders[j][6] -= self.order_interval
```

```
        if (utils.geo_distance(res_orders[1],self.waiting_delivery_orders[j])/self.speed) > self.priority[self.waiting_delivery_orders[j][3]]-self.order_interval*(ts-self.waiting_delivery_orders[j][5]):
            res_orders.append(self.waiting_delivery_orders[j])
        else:
            waiting_orders.append(self.waiting_delivery_orders[j])
    for i in range(len(orders)):
        orders[i][6] -= self.order_interval
        if (utils.geo_distance(res_orders[1],orders[i])/self.speed) > self.priority[orders[i][3]]-self.order_interval:
            res_orders.append(orders[i])
        else:
            waiting_orders.append(orders[i])
    count = int((len(res_orders)+len(waiting_orders))* 0.3)
    if count <= 0:
        self.waiting_delivery_orders = waiting_orders
    else:
        t_orders = sorted(waiting_orders, key=lambda x: (x[6], -x[4]))
        res_orders.extend(t_orders[:count])
        self.waiting_delivery_orders = t_orders[count:]
    return res_orders
```

3. 多轮模拟退火算法

通过在不同温度下，按照 Metropolos 准则，每个温度下通过产生单轮最优解从而得到该温度的最优解，最后再与全局最优解进行路径评定目标函数值进行比较，从而决定是否接受新解。其中需要注意的是，即使需要满足 Metropolos 准则，也应当满足优先级等约束条件。

```
def sa_process_iterator(self, solution):
        while self.T > self.T_end:
            self.per_iter_solution = solution
            for _ in range(self.Lk):
                current_solu_obj = self.obj_func(solution)
                new_solu = self.generate_new_solu(solution)
                new_solu_obj = self.obj_func(new_solu)
                if new_solu_obj < current_solu_obj:
                    solution = new_solu
                elif new_solu_obj != INF:
                    prob_accept = math.exp(-(new_solu_obj - current_solu_obj) / self.T)
                    p = random.random()
                    if p < prob_accept:
```

```
                        solution = new_solu
            solu_obj = self.obj_func(solution)
            if solu_obj == INF:
                continue
            if solu_obj < self.obj_func(self.per_iter_solution):
                self.per_iter_solution = solution
            if solu_obj < self.obj_func(self.best_solution):
                self.best_solution = solution
            self.all_per_iter_solution.append(self.per_iter_solution)
            self.all_best_solution.append(self.best_solution)

            self.T_list.append(self.T)
            self.T = self.T * self.alpha
```

4. 目标函数

   计算解的目标函数值，同时检查每个无人机的路径长度、优先级和容量是否满足约束。

```
def obj_func(self, solution: list) -> list:
    all_routines, routines_dis = self.get_delivery_distance(solution)
    for single_path_length in routines_dis:
        if single_path_length > self.max_single_path_length:
            return INF
    if not self.check_priority_cap(all_routines):
        return INF
    sum_path = sum(routines_dis)
    max_path = max(routines_dis)
    min_path = min(routines_dis)
    if max_path == 0:
        balance = 0
    else:
        balance = (max_path - min_path) / max_path
    obj = self.distance_weight * sum_path + self.balance_weight * balance
    return obj
```

5. 检查优先级及容量

   检查当前所有路径中是否都满足优先度需求和无人机最大携带容量约束，否则将返回最大值作为惩罚。

```
def check_priority_cap(self, routines: list) -> bool:
    for r in routines:
        if len(r) == 0:
```

```
                continue
            dis = 0
            num = 0
            for i in range(len(r)):
                num += self.current_delivery_orders[r[i]][4]
                if i == 0:
                    dis += utils.get_dis(self.current_delivery_orders[self.start
_index],   self.current_delivery_orders[r[i]])
                elif i + 1 < len(r):
                    dis += utils.get_dis(self.current_delivery_orders[r[i+1]],
                                         self.current_delivery_orders[r[i]])
                if dis/self.speed > self.priority[self.current_delivery_orders[r
[i]][3]] or num>self.cap_num:
                    return False
        return True
```

6. 补充订单

由于最初选择的订单可能生成的路径中的无人机还拥有携带能力，因此在这里用贪心算法尝试加入与路径中每个卸货点最近的点，进行选择扩充使无人机尽可能达到满载，由此可以达到一个局部最优。计算距离选择构建 KD 树，通过由尚未配送的等待订单构成的 KD，可以快速地在大范围内求最小距离，以提高系统的大规模应用能力。

```
def replenish(self):
    best_path, best_dist_list = self.get_delivery_distance(self.best_solution)
    self.waiting_delivery_orders.sort(key=lambda x: (x[1], -x[2]))
    for i in range(len(best_path)):
        if best_dist_list[i] == 0:
            continue
        flag = True
        while best_dist_list[i] < self.max_single_path_length * 0.9 and flag :
            point = np.array([(item[1], item[2]) for item in self.waiting_delivery_orders])
            nn_model = NearestNeighbors(n_neighbors=1, algorithm='kd_tree')
            nn_model.fit(point)
            query_points = []
            for index in range(len(best_path[i])):
                query_points.append([self.current_delivery_orders[best_path[i][index]][1],
                                     self.current_delivery_orders[best_path[i][index]][2]])
            distances, indices = nn_model.kneighbors(query_points)
            dis = []
            for k in range(len(distances)):
                dis.append([distances[k][0],indices[k][0]])
```

```
            dis_sorted = sorted(dis, key=lambda x: x[0])
            j = 0
            for j in range(len(dis_sorted)):
                self.current_delivery_orders.append(self.waiting_delivery_orders[dis_sorted[j][1]])
            best_path[i].insert(j+1,len(self.current_delivery_orders)-1)
                                self.get_distance_one_path(best_path[i]) > self.max_single_path_length
                                or not self.check_priority_cap([best_path[i]]):
                    best_path[i].pop(j+1)
                    best_path[i].insert(j,len(self.current_delivery_orders)-1)
                    if self.get_distance_one_path(best_path[i]) > self.max_single_path_length or not self.check_pr
iority_cap([best_path[i]]):
                        self.current_delivery_orders.pop(len(self.current_delivery_orders)-1)
                        best_path[i].pop(j)
                    else:
                        self.waiting_delivery_orders.pop(dis_sorted[j][1])
                        if j == len(dis_sorted)-1:
                            self.best_solution.insert(self.best_solution.index(best_path[i][j-
1])+1,best_path[i][j])
                        else :
                    self.best_solution.insert(self.best_solution.index(best_path[i][j+1]),best_path[i][j])
                        self.solution_len += 1
                        best_dist_list[i] = self.get_distance_one_path(best_path[i])
                        break
                else:
                    self.waiting_delivery_orders.pop(dis_sorted[j][1])
                    self.best_solution.insert(self.best_solution.index(best_path[i][j])+1,best_path[i][j+1])
                    self.solution_len += 1
                    best_dist_list[i] = self.get_distance_one_path(best_path[i])
                    break
            else:
                flag = False
```

7. 交换产生新解

交换产生新解，与交换变异类似。

```
def swap_solution(self, solution):
    index1 = random.randint(0, self.solution_len - 1)
    index2 = random.randint(0, self.solution_len - 1)
    new_solution = solution[:]
    new_solution[index1], new_solution[index2] = new_solution[index2], new_
solution[index1]
    return new_solution
```

8. 移位产生新解

移位产生新解: 随机选取三个点, 将前两个点之间的点移位到第三个点的后方。

```
1.  def shift_solution(self, solution: list) -> list:
2.          tmp = sorted(random.sample(range(self.solution_len), 3))
3.          index1, index2, index3 = tmp[0], tmp[1], tmp[2]
4.          tmp = solution[index1:index2]
5.          new_solution = []
6.          for i in range(self.solution_len):
7.              if index1 <= i < index2:
8.                  continue
9.              if (i < index1 or i >= index2) and i < index3:
10.                 new_solution.append(solution[i])
11.             elif i == index3:
12.                 new_solution.append(solution[i])
13.                 new_solution.extend(tmp)
14.             else:
15.                 new_solution.append(solution[i])
16.         return new_solution
```

9. 逆转产生新解

逆转: 随机选择两点(可能为同一点), 逆转其中所有的元素。

```
1.  def reverse_solution(self, solution: list) -> list:
2.      index1, index2 = random.randint(0, self.solution_len - 1), random.randin
    t(0, self.solution_len - 1)
3.      if index1 > index2:
4.          index1, index2 = index2, index1
5.      new_solution = solution[:]
6.      tmp = new_solution[index1: index2]
7.      tmp.reverse()
8.      new_solution[index1: index2] = tmp
9.      return new_solution
```

四、实验结果

4.1 实验设置

在本实验中，选择进行五轮订单生成，每轮生成一百个订单，订单的位置随机产生，每个无人机的速度为 60 公里每小时。由于最初的简化，将订单进行按

区域划分至每个配送中心，因此实验只需要展示一个配送中心的配送情况即可。此外，每次订单选择至少 30%，每个无人机尽可能要达到 90%的满载率。

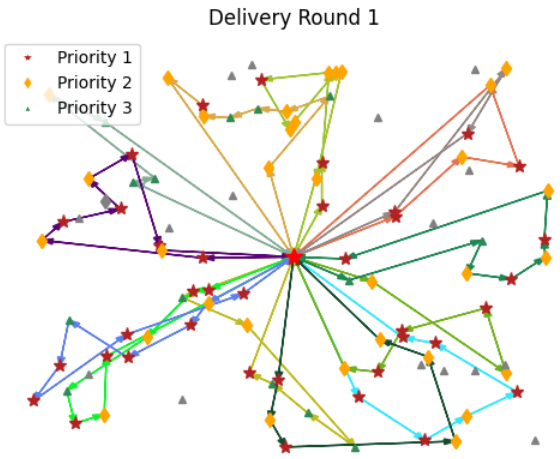
模拟退火的初始初始温度为 200，终止温度为 0.1，每个温度迭代 300 次，采用指数递减，温度衰减洗漱为 0.95。执行交换产生新解概率为 0.1，行逆转产生新解的概率为 0.4，执行位移产生新解的概率为 0.5。其余环境如下表所示：

环境	参数或版本
CPU	i5-1240P 1.70 GHz
Python	3.12.0
scikit-learn	1.5.0

4.2 实验结果

第一轮的结果如下：

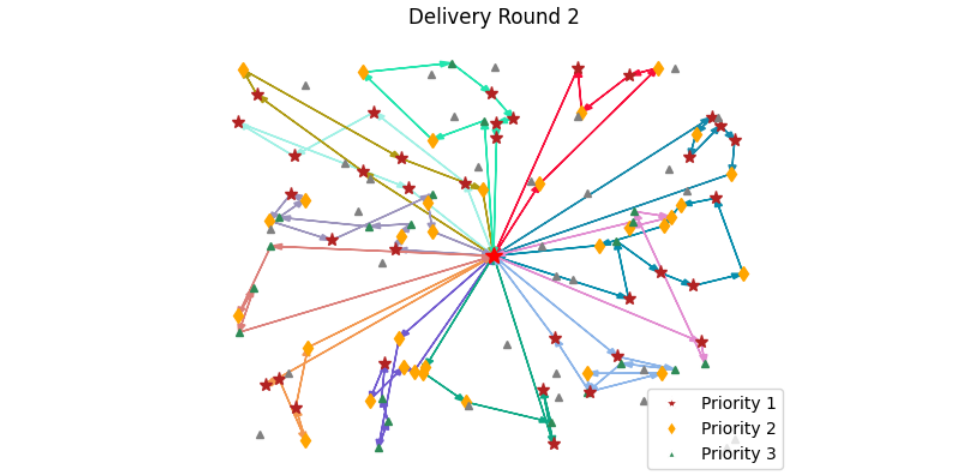
```
第1轮配送订单83个，一对一配送的总配送路程为445.005
无人机所有路线长度为：247.05599999999998
第1架无人机路线长度为：18.586,需要经过6个卸货点,携带19件货物,
第1架无人机路线为：s --> o_65 --> o_6 --> o_41 --> o_21 --> o_29 --> o_94 --> s
第2架无人机路线长度为：18.726,需要经过6个卸货点,携带18件货物,
第2架无人机路线为：s --> o_50 --> o_34 --> o_73 --> o_81 --> o_91 --> o_7 --> s
第3架无人机路线长度为：19.994,需要经过9个卸货点,携带26件货物,
第3架无人机路线为：s --> o_90 --> o_56 --> o_26 --> o_31 --> o_42 --> o_39 --> o_83 --> o_68 --> o_22 --> s
第4架无人机路线长度为：19.046999999999997,需要经过8个卸货点,携带22件货物,
第4架无人机路线为：s --> o_71 --> o_17 --> o_75 --> o_12 --> o_20 --> o_84 --> o_27 --> o_47 --> s
第5架无人机路线长度为：18.68,需要经过4个卸货点,携带12件货物,
第5架无人机路线为：s --> o_38 --> o_19 --> o_80 --> o_97 --> s
第6架无人机路线长度为：19.282,需要经过8个卸货点,携带29件货物,
第6架无人机路线为：s --> o_44 --> o_49 --> o_5 --> o_87 --> o_51 --> o_57 --> o_77 --> o_63 --> s
第7架无人机路线长度为：18.628999999999998,需要经过4个卸货点,携带13件货物,
第7架无人机路线为：s --> o_89 --> o_25 --> o_3 --> o_55 --> s
第8架无人机路线长度为：18.891000000000002,需要经过6个卸货点,携带19件货物,
第8架无人机路线为：s --> o_15 --> o_72 --> o_85 --> o_82 --> o_59 --> o_8 --> s
第9架无人机路线长度为：18.173,需要经过3个卸货点,携带9件货物,
第9架无人机路线为：s --> o_88 --> o_67 --> o_13 --> s
第10架无人机路线长度为：18.314,需要经过7个卸货点,携带13件货物,
第10架无人机路线为：s --> o_48 --> o_11 --> o_36 --> o_45 --> o_46 --> o_61 --> o_86 --> s
第11架无人机路线长度为：19.901,需要经过6个卸货点,携带20件货物,
第11架无人机路线为：s --> o_62 --> o_32 --> o_70 --> o_9 --> o_35 --> o_23 --> s
第12架无人机路线长度为：19.004,需要经过8个卸货点,携带20件货物,
第12架无人机路线为：s --> o_79 --> o_43 --> o_99 --> o_4 --> o_1 --> o_24 --> o_66 --> o_64 --> s
第13架无人机路线长度为：19.829,需要经过8个卸货点,携带25件货物,
第13架无人机路线为：s --> o_98 --> o_30 --> o_69 --> o_92 --> o_18 --> o_58 --> o_40 --> o_0 --> s
```



其中 Priority 1 优先级最高，Priority 优先级最低，灰色表示当前轮次不配送，最中心为配送中心。此外，每一个不同颜色的路径代表不同的无人机。

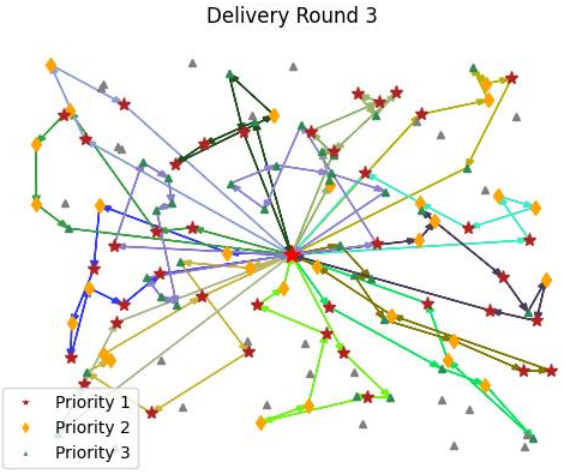
第二轮结果如下：

```
第2轮配送订单87个，一对一配送的总配送路程为477.967
无人机所有路线长度为：250.946
第1架无人机路线长度为：19.42,需要经过5个卸货点,携带17件货物,
第1架无人机路线为：s --> o_126 --> o_103 --> o_188 --> o_177 --> o_192 --> s
第2架无人机路线长度为：18.9,需要经过8个卸货点,携带20件货物,
第2架无人机路线为：s --> o_144 --> o_138 --> o_134 --> o_37 --> o_163 --> o_141 --> o_113 --> o_169 --> s
第3架无人机路线长度为：19.84,需要经过5个卸货点,携带11件货物,
第3架无人机路线为：s --> o_123 --> o_108 --> o_186 --> o_159 --> o_139 --> s
第4架无人机路线长度为：19.455000000000002,需要经过5个卸货点,携带17件货物,
第4架无人机路线为：s --> o_175 --> o_189 --> o_165 --> o_131 --> o_166 --> s
第5架无人机路线长度为：18.767,需要经过6个卸货点,携带17件货物,
第5架无人机路线为：s --> o_118 --> o_130 --> o_106 --> o_154 --> o_127 --> o_136 --> s
第6架无人机路线长度为：19.830000000000002,需要经过5个卸货点,携带17件货物,
第6架无人机路线为：s --> o_190 --> o_161 --> o_121 --> o_148 --> o_162 --> s
第7架无人机路线长度为：19.646,需要经过6个卸货点,携带23件货物,
第7架无人机路线为：s --> o_198 --> o_107 --> o_178 --> o_155 --> o_143 --> o_142 --> s
第8架无人机路线长度为：19.774,需要经过8个卸货点,携带18件货物,
第8架无人机路线为：s --> o_199 --> o_60 --> o_54 --> o_196 --> o_157 --> o_167 --> o_125 --> o_187 --> s
第9架无人机路线长度为：19.521,需要经过9个卸货点,携带28件货物,
第9架无人机路线为：s --> o_119 --> o_197 --> o_115 --> o_182 --> o_153 --> o_184 --> o_140 --> o_158 --> o_174 --> s
第10架无人机路线长度为：19.857,需要经过8个卸货点,携带21件货物,
第10架无人机路线为：s --> o_149 --> o_105 --> o_110 --> o_2 --> o_101 --> o_120 --> o_147 --> o_112 --> s
第11架无人机路线长度为：17.896000000000004,需要经过12个卸货点,携带30件货物,
第11架无人机路线为：s --> o_181 --> o_193 --> o_170 --> o_33 --> o_74 --> o_78 --> o_171 --> o_180 --> o_116 --> o_76 --> o_195 --> o_164 --> s
第12架无人机路线长度为：18.747,需要经过4个卸货点,携带15件货物,
第12架无人机路线为：s --> o_146 --> o_132 --> o_111 --> o_194 --> s
第13架无人机路线长度为：19.293,需要经过6个卸货点,携带21件货物,
第13架无人机路线为：s --> o_151 --> o_16 --> o_133 --> o_109 --> o_160 --> o_14 --> s
```



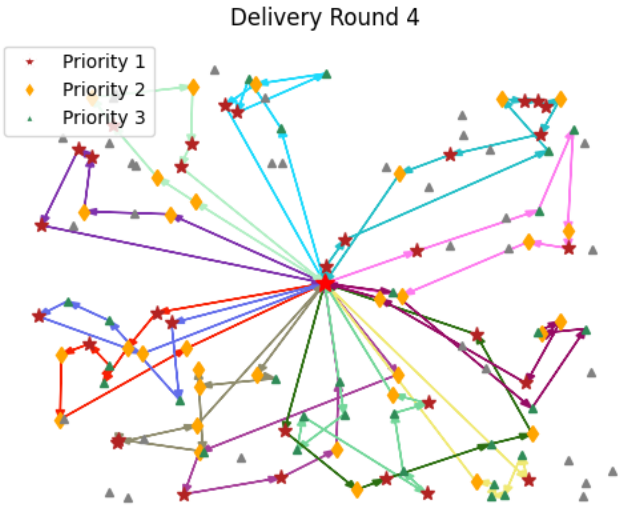
第三轮结果如下：

```
第3轮配送订单104个，一对一配送的总配送路程为522.496
无人机所有路线长度为：285.063
第1架无人机路线长度为：19.911,需要经过8个卸货点,携带26件货物,
第1架无人机路线为：s --> o_229 --> o_250 --> o_202 --> o_245 --> o_282 --> o_295 --> o_212 --> o_228 --> s
第2架无人机路线长度为：19.762999999999998,需要经过7个卸货点,携带26件货物,
第2架无人机路线为：s --> o_273 --> o_289 --> o_219 --> o_267 --> o_269 --> o_244 --> o_191 --> s
第3架无人机路线长度为：19.830000000000002,需要经过6个卸货点,携带20件货物,
第3架无人机路线为：s --> o_265 --> o_266 --> o_95 --> o_218 --> o_236 --> o_96 --> s
第4架无人机路线长度为：18.212,需要经过6个卸货点,携带19件货物,
第4架无人机路线为：s --> o_234 --> o_253 --> o_287 --> o_247 --> o_210 --> o_299 --> s
第5架无人机路线长度为：19.302999999999997,需要经过6个卸货点,携带26件货物,
第5架无人机路线为：s --> o_227 --> o_53 --> o_215 --> o_93 --> o_246 --> o_237 --> s
第6架无人机路线长度为：19.289,需要经过3个卸货点,携带6件货物,
第6架无人机路线为：s --> o_207 --> o_278 --> o_293 --> s
第7架无人机路线长度为：18.175,需要经过9个卸货点,携带20件货物,
第7架无人机路线为：s --> o_117 --> o_135 --> o_137 --> o_286 --> o_216 --> o_283 --> o_294 --> o_220 --> o_279 --> s
第8架无人机路线长度为：16.63,需要经过10个卸货点,携带30件货物,
第8架无人机路线为：s --> o_243 --> o_124 --> o_275 --> o_230 --> o_264 --> o_239 --> o_272 --> o_28 --> o_292 --> o_290 --> s
第9架无人机路线长度为：19.833,需要经过8个卸货点,携带29件货物,
第9架无人机路线为：s --> o_231 --> o_128 --> o_226 --> o_284 --> o_52 --> o_277 --> o_241 --> o_276 --> s
第10架无人机路线长度为：18.346,需要经过9个卸货点,携带18件货物,
第10架无人机路线为：s --> o_252 --> o_258 --> o_168 --> o_235 --> o_297 --> o_203 --> o_256 --> o_224 --> o_238 --> s
第11架无人机路线长度为：18.607,需要经过3个卸货点,携带11件货物,
第11架无人机路线为：s --> o_114 --> o_263 --> o_288 --> s
第12架无人机路线长度为：19.44,需要经过6个卸货点,携带18件货物,
第12架无人机路线为：s --> o_201 --> o_213 --> o_255 --> o_240 --> o_10 --> o_206 --> s
第13架无人机路线长度为：19.285,需要经过8个卸货点,携带26件货物,
第13架无人机路线为：s --> o_222 --> o_268 --> o_274 --> o_204 --> o_261 --> o_280 --> o_221 --> o_270 --> s
第14架无人机路线长度为：18.917,需要经过8个卸货点,携带27件货物,
第14架无人机路线为：s --> o_271 --> o_145 --> o_104 --> o_214 --> o_183 --> o_248 --> o_262 --> o_259 --> s
第15架无人机路线长度为：19.522,需要经过7个卸货点,携带20件货物,
第15架无人机路线为：s --> o_260 --> o_232 --> o_209 --> o_254 --> o_156 --> o_205 --> o_129 --> s
```



第四轮结果如下

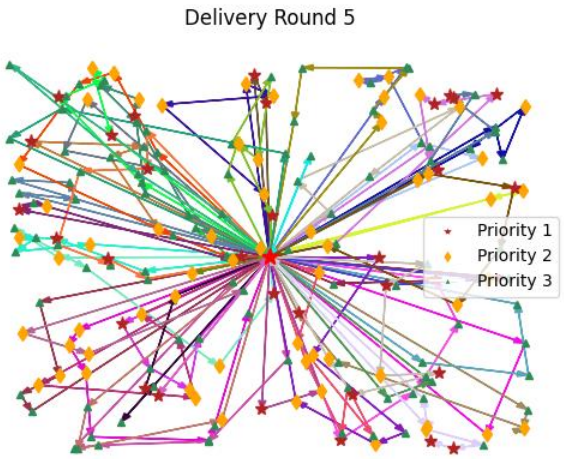
```
第4轮配送订单92个，一对一配送的总配送路程为526.790
无人机所有路线长度为：249.28900000000002
第1架无人机路线长度为：19.762999999999998,需要经过6个卸货点,携带15件货物,
第1架无人机路线为：s --> o_355 --> o_397 --> o_387 --> o_347 --> o_346 --> o_150 --> s
第2架无人机路线长度为：18.765,需要经过7个卸货点,携带17件货物,
第2架无人机路线为：s --> o_339 --> o_185 --> o_102 --> o_345 --> o_332 --> o_379 --> o_368 --> s
第3架无人机路线长度为：19.582,需要经过6个卸货点,携带21件货物,
第3架无人机路线为：s --> o_179 --> o_301 --> o_316 --> o_176 --> o_353 --> o_336 --> s
第4架无人机路线长度为：18.838,需要经过6个卸货点,携带14件货物,
第4架无人机路线为：s --> o_390 --> o_211 --> o_200 --> o_305 --> o_395 --> o_364 --> s
第5架无人机路线长度为：19.474,需要经过8个卸货点,携带29件货物,
第5架无人机路线为：s --> o_249 --> o_122 --> o_308 --> o_312 --> o_208 --> o_152 --> o_394 --> o_342 --> s
第6架无人机路线长度为：18.915,需要经过8个卸货点,携带26件货物,
第6架无人机路线为：s --> o_392 --> o_381 --> o_303 --> o_365 --> o_341 --> o_363 --> o_335 --> o_340 --> s
第7架无人机路线长度为：19.334000000000003,需要经过6个卸货点,携带19件货物,
第7架无人机路线为：s --> o_307 --> o_223 --> o_344 --> o_369 --> o_371 --> o_320 --> s
第8架无人机路线长度为：19.647,需要经过6个卸货点,携带18件货物,
第8架无人机路线为：s --> o_382 --> o_356 --> o_311 --> o_100 --> o_376 --> o_391 --> s
第9架无人机路线长度为：19.761,需要经过11个卸货点,携带26件货物,
第9架无人机路线为：s --> o_377 --> o_354 --> o_321 --> o_373 --> o_304 --> o_319 --> o_309 --> o_351 --> o_352 --> o_337 --> o_350 --> s
第10架无人机路线长度为：18.659,需要经过8个卸货点,携带21件货物,
第10架无人机路线为：s --> o_331 --> o_257 --> o_324 --> o_378 --> o_296 --> o_338 --> o_300 --> o_326 --> s
第11架无人机路线长度为：18.387,需要经过5个卸货点,携带23件货物,
第11架无人机路线为：s --> o_360 --> o_367 --> o_357 --> o_396 --> o_393 --> s
第12架无人机路线长度为：19.537000000000003,需要经过8个卸货点,携带24件货物,
第12架无人机路线为：s --> o_328 --> o_366 --> o_359 --> o_384 --> o_372 --> o_398 --> o_348 --> o_386 --> s
第13架无人机路线长度为：18.627000000000002,需要经过7个卸货点,携带19件货物,
第13架无人机路线为：s --> o_370 --> o_334 --> o_302 --> o_358 --> o_329 --> o_362 --> o_333 --> s
```



第五轮结果如下

```
第5轮配送订单134个，一对一配送的总配送路程为769.630
无人机所有路线长度为：488.991
第1架无人机路线长度为：19.334,需要经过5个卸货点,携带16件货物,
第1架无人机路线为：s --> o_436 --> o_330 --> o_242 --> o_431 --> o_499 --> s
第2架无人机路线长度为：18.692,需要经过4个卸货点,携带10件货物,
第2架无人机路线为：s --> o_414 --> o_420 --> o_417 --> o_466 --> s
第3架无人机路线长度为：17.684,需要经过5个卸货点,携带17件货物,
第3架无人机路线为：s --> o_474 --> o_383 --> o_445 --> o_233 --> o_480 --> s
第4架无人机路线长度为：17.613,需要经过6个卸货点,携带20件货物,
第4架无人机路线为：s --> o_497 --> o_439 --> o_217 --> o_456 --> o_361 --> o_454 --> s
第5架无人机路线长度为：18.300999999999995,需要经过6个卸货点,携带22件货物,
第5架无人机路线为：s --> o_399 --> o_487 --> o_482 --> o_477 --> o_430 --> o_491 --> s
第6架无人机路线长度为：18.557,需要经过3个卸货点,携带13件货物,
第6架无人机路线为：s --> o_343 --> o_447 --> o_424 --> s
第7架无人机路线长度为：19.416,需要经过5个卸货点,携带13件货物,
第7架无人机路线为：s --> o_440 --> o_488 --> o_457 --> o_251 --> o_448 --> s
第8架无人机路线长度为：18.777,需要经过6个卸货点,携带16件货物,
第8架无人机路线为：s --> o_468 --> o_380 --> o_483 --> o_375 --> o_285 --> o_492 --> s
第9架无人机路线长度为：19.907999999999998,需要经过8个卸货点,携带25件货物,
第9架无人机路线为：s --> o_327 --> o_473 --> o_173 --> o_281 --> o_481 --> o_298 --> o_449 --> o_402 --> s
第10架无人机路线长度为：17.87,需要经过3个卸货点,携带11件货物,
第10架无人机路线为：s --> o_425 --> o_446 --> o_484 --> s
第11架无人机路线长度为：16.625,需要经过4个卸货点,携带12件货物,
第11架无人机路线为：s --> o_478 --> o_314 --> o_463 --> o_458 --> s
第12架无人机路线长度为：18.821,需要经过7个卸货点,携带24件货物,
第12架无人机路线为：s --> o_434 --> o_498 --> o_423 --> o_459 --> o_450 --> o_421 --> o_495 --> s
第13架无人机路线长度为：19.464,需要经过4个卸货点,携带13件货物,
第13架无人机路线为：s --> o_493 --> o_306 --> o_475 --> o_291 --> s
第14架无人机路线长度为：19.518,需要经过4个卸货点,携带17件货物,
第14架无人机路线为：s --> o_405 --> o_467 --> o_479 --> o_465 --> s
第15架无人机路线长度为：19.218,需要经过7个卸货点,携带18件货物,
第15架无人机路线为：s --> o_437 --> o_406 --> o_427 --> o_310 --> o_455 --> o_443 --> o_460 --> s
第16架无人机路线长度为：15.871,需要经过3个卸货点,携带10件货物,
第16架无人机路线为：s --> o_438 --> o_470 --> o_476 --> s
第17架无人机路线长度为：17.363,需要经过5个卸货点,携带19件货物,
第17架无人机路线为：s --> o_428 --> o_435 --> o_313 --> o_422 --> o_494 --> s
第18架无人机路线长度为：10.892,需要经过2个卸货点,携带5件货物,
第18架无人机路线为：s --> o_433 --> o_471 --> s
第19架无人机路线长度为：18.963,需要经过3个卸货点,携带8件货物,
第19架无人机路线为：s --> o_444 --> o_432 --> o_401 --> s
第20架无人机路线长度为：13.055,需要经过7个卸货点,携带26件货物,
第20架无人机路线为：s --> o_496 --> o_485 --> o_415 --> o_426 --> o_469 --> o_451 --> o_407 --> s
第21架无人机路线长度为：9.804,需要经过2个卸货点,携带8件货物,
第21架无人机路线为：s --> o_408 --> o_462 --> s
第22架无人机路线长度为：19.379,需要经过5个卸货点,携带15件货物,
第22架无人机路线为：s --> o_409 --> o_411 --> o_442 --> o_461 --> o_388 --> s
```

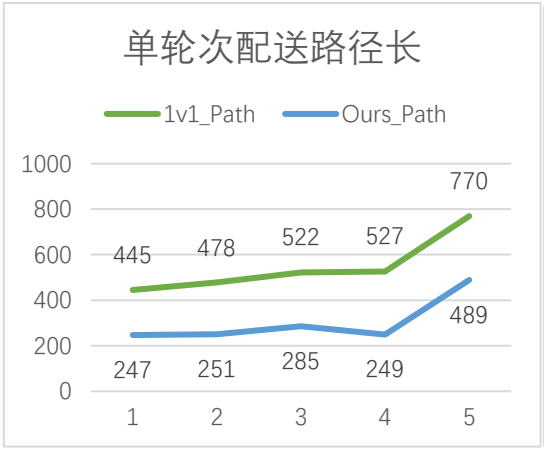
```
第23架无人机路线长度为：18.819,需要经过8个卸货点,携带20件货物,
第23架无人机路线为：s --> o_322 --> o_225 --> o_429 --> o_419 --> o_453 --> o_404 --> o_410 --> o_374 --> s
第24架无人机路线长度为：19.663,需要经过2个卸货点,携带2件货物,
第24架无人机路线为：s --> o_172 --> o_325 --> s
第25架无人机路线长度为：10.33,需要经过2个卸货点,携带8件货物,
第25架无人机路线为：s --> o_403 --> o_385 --> s
第26架无人机路线长度为：16.639000000000003,需要经过6个卸货点,携带24件货物,
第26架无人机路线为：s --> o_464 --> o_418 --> o_318 --> o_452 --> o_400 --> o_315 --> s
第27架无人机路线长度为：19.081000000000003,需要经过6个卸货点,携带23件货物,
第27架无人机路线为：s --> o_412 --> o_349 --> o_489 --> o_416 --> o_441 --> o_472 --> s
第28架无人机路线长度为：19.334,需要经过6个卸货点,携带16件货物,
第28架无人机路线为：s --> o_413 --> o_323 --> o_490 --> o_317 --> o_486 --> o_389 --> s
```

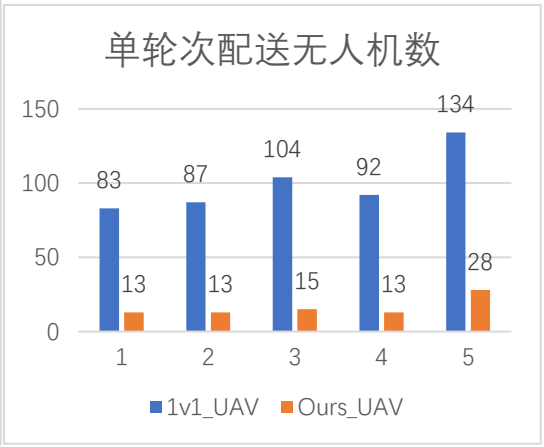


将上述结果整理表格如下：

轮数	配送订单数	一对一配送		本方法		优势	
		UAV 数	总路线长	UAV 数	总路线长	UAV 数	总路线长
1	83	83	445.005	13	247.055	44.48%	84.34%
2	87	87	477.967	13	250.946	47.50%	85.06%
3	104	104	522.496	15	285.063	45.44%	85.58%
4	92	92	526.790	13	249.289	52.68%	85.87%
5	134	134	769.630	28	488.991	36.46%	79.10%
Sum	**500**	**500**	**2741.888**	**82**	**1521.344**	**44.51%**	**83.60%**

将上图数据画图如下：





可以看出，在使用了我们的方法下，配送路径长会下降约 35-53%，而无人机数会下降 80-85%，这不仅提高了配送效率，也大幅度降低了运营成本。

五、实验总结

本实验通过设计并实现一种基于模拟退火算法的无人机配送路径规划方法，

有效地解决了快递行业最后一公里配送的难题。实验结果表明，与传统的一对一配送方式相比，本方法在减少配送路径长度和降低无人机使用数量方面均取得了显著的效果。具体来看，配送路径长度平均下降了约 44.51%，而无人机的使用数量则减少了约 83.60%，这不仅提高了配送效率，也大幅度降低了运营成本。

此外，本方法在考虑订单优先级和无人机容量限制的基础上，通过多轮模拟退火过程，不断优化配送方案，实现了在满足时效性要求的同时，最大化资源利用率。实验中采用的贪心算法和 KD 树数据结构，进一步提高了算法的搜索效率和准确性，尤其是在处理大规模数据时，展现了良好的扩展性和实用性。

然而，实验中也存在一些局限性，如算法的初始参数设置对最终结果有一定影响，且在实际应用中可能面临更加复杂的环境和约束条件。未来的工作可以进一步优化算法参数，探索更加智能的参数调整策略，并考虑实际应用中的动态变化因素，以提高算法的鲁棒性和适应性。总体来说，本实验为无人机配送路径规划提供了一种有效的解决方案，对推动物流配送行业的智能化发展具有重要的理论和实践意义。