《高级算法设计与分析》大作业实验报告

学号：2023202210143

姓名：张翔宇

一、问题定义

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；

二、算法实现

1. 初始化：

（1）生成地图：

定义一个 `Point` 类，表示一个点对象，包括位置信息 `pos`、编号 `id` 和名称 `name`。配送中心和卸货点的 `id` 都从 0 开始编号，为便于区分，配送中心的名字为 A、B、……；卸货点的名字为 0、1、2……。

```
class Point:
    def __init__(self, pos):
        self.pos = pos
        self.id = 0
        self.name = ''
    def set_name(self, id, name):
        self.id = id
        self.name = name
```

在已知地图大小 `map_size`，配送中心和卸货点数量 `j,k` 的情况下，随机生成两组点集，分别表示配送中心和卸货点的位置。

为了符合实际情况，避免地图上的点在某一部分过于密集，在生成每个点之前，根据点的数量动态计算最小距离 `min_distance` 为 $\frac{\text{地图大小}}{\sqrt{\text{点的数量}}}$，以避免生成过于密集的点。对于卸货点的生成，除了要确保与配送中心的距离足够远外，还要确保新生成的点与已有的卸货点的距离也足够远，以保持卸货点之间的合理分布。

此外，通过预设随机种子保证每次实验地图固定，实验结果可复现，在生成点以后按照距离远点的距离进行排序，便于编号。

参照实际生活中的情况，配送中心数量 `j` ，卸货点数量 `k`，分别设为10，60；地图大小设置为 `-10km` 到 `+10km`。

核心代码：

```
def generate_points(j, k):
    def is_valid_point(x, y, existing_points,
min_distance):
        for pt in existing_points:
            ex, ey = pt.pos
            if distance(x, y, ex, ey) < min_distance:
                return False
        return True

    if map_seed is not None:
        np.random.seed(map_seed)

    points1 = []
    points2 = []
    # 生成配送中心
    while len(points1) < j:
        min_distance = map_size / (int(j**0.5))
        x, y = np.random.uniform(-map_size, map_size, 2)
        if is_valid_point(x, y, points1, min_distance):
            points1.append(Point((int(x), int(y))))

    # 生成卸货点
    while len(points2) < k:
        min_distance = map_size / (int(k**0.5))
        x, y = np.random.uniform(-map_size, map_size, 2)
        # 确保卸货点与配送中心距离合适，并且与自身的点之间的距离也足
够大
        if is_valid_point(x, y, points1, min_distance / 2)
and is_valid_point(x, y, points2, min_distance):
            points2.append(Point((int(x), int(y))))

    # 根据位置的和排序
    points1.sort(key=lambda p: p.pos[0] + p.pos[1])
    points2.sort(key=lambda p: p.pos[0] + p.pos[1])
```

```
    # 给配送中心设置名称，使用字母表示
    for i in range(len(points1)):
        points1[i].set_name(i, chr(ord("A") + i))
    # 给卸货点设置名称，使用数字表示
    for i in range(len(points2)):
        points2[i].set_name(i, str(i))

    return points1, points2
```

(2) 生成订单:

定义`Order`类用来表示一个订单，具有四个属性：`time`表示订单创建的时间，`dp_index`表示派送点的索引，`priority`表示订单的优先级，`deadline`表示订单的截止时间，为订单时间加上优先级对应的时间长度。 在实验中假设`t`为30，即每隔30分钟生成一批订单。

```
class Order:
    def __init__(self, time, dp_index, priority):
        self.time = time
        self.dp_index = dp_index
        self.priority = priority
        self.deadline = time + priority_levels[priority]

    def print_order(self, name):
        #打印订单的指定属性值。
        if hasattr(self, name):
            print(f"{getattr(self, name)}", end=' ')
        else:
            print(f"Invalid attribute name: {name}")
```

然后每隔一段时间生成随机数量和属性的订单列表。它接受一个参数`time`，表示订单生成的时间。固定随机种子为`time + order_seed`，这样既保证在不同时间生成的订单不同，又保证了实验可复现，便于测试结果。每次随机生成订单数量`num_orders`，数量范围在 10 到`orders_per_interval`之间。对于每一个订单，随机选择一个派送点`point`和一个优先级`priority`。使用`Order`类构造订单对象，并将其添加到`orders`列表中。最后返回生成的订单列表。

核心代码

```
def generate_orders(time):
    random.seed(time + order_seed)
    num_orders = random.randint(10, orders_per_interval)
    orders = []
    for _ in range(num_orders):
        point = random.randint(0, k-1)
        priority =
random.choices(list(priority_levels.keys()), [1, 1, 1])[0]
        orders.append(Order(time, point, priority))
    return orders
```

2. 实现方法：

（1）订单暂存：

由于不同优先级的订单最大配送时间分别为`180、90、30`分钟，无人机最大飞行距离为 `20km`，速度为 `60km/h`。因此对于任何订单，只要分配方案能够满足无人机的最大飞行距离限制，从开始配送到送达的时间都不超过 `20` 分钟。由于假设每隔 `30` 分钟生成一批订单。因此对于每个订单，最晚在其截止时间前30分钟，和同批生成的订单一同配送。

因此对于每个订单，应尽可能选择和其距离较近的订单一同配送，如果其距离一个其他订单的距离小于这个订单距离最近配送中心的距离，就让这两个订单同一批配送。由于如果订单尚未到达最晚配送时间，还可以继续暂存，因此，对于希望同一批配送的订单，将其最晚配送时间更改为两个订单中较早的。既可以保证在要求时限内配送完成，又可以累计尽可能多的距离较近的订单，从而缩短无人机配送距离。

核心代码：

```
def store_and_load(all_orders, distribution_centers,
unload_points, current_time):
    # 根据订单的截止时间排序所有订单
    all_orders.sort(key=lambda order: order.deadline)

    # 记录每个订单到最近配送中心的最短距离
    order_min_center = []
    for order in all_orders:
        min_dis = float('inf')
```

```
            for center in distribution_centers:
                distance = calculate_distance(center,
unload_points[order.dp_index])
                if distance < min_dis:
                    min_dis = distance
            order_min_center.append(min_dis)

        # 根据订单之间的距离关系，调整订单的截止时间
        for i in range(len(all_orders)):
            for j in range(i + 1, len(all_orders)):
                distance =
calculate_distance(unload_points[all_orders[i].dp_index],
unload_points[all_orders[j].dp_index])
                if distance <= order_min_center[j]:
                    all_orders[j].deadline =
all_orders[i].deadline

        # 分类订单，根据截止时间划分为当前可处理和临时订单
        orders = []
        temporary_orders = []
        for order in all_orders:
            # 如果订单的截止时间在当前时间+30分钟内就立刻处理，否则暂存
            if order.deadline <= (current_time + 30):
                orders.append(order)
            else:
                temporary_orders.append(order)

        return orders, temporary_orders
```

（2）无人机配送：

对于每一批需要配送的订单，将其分配给若干无人机，同时满足无人机单次携带物品上限和　　单次飞行最远路程。使用启发式算法来求解这个问题。为了全面评估算法的结果，同时使用多种启发式算法，以及暴力和贪心算法作为对照。

a. 暴力算法：

对于每个订单都使用一架无人机，从距离订单卸货点最近的配送中心出发进行配送，因此每架无人机只配送一个订单。地图和订单生成相同的情况下，暴力算法得到的结果为无人机配送距离的上限，即算法的最差结果。

算法的输入为订单列表，所有配送中心和卸货点的位置。输出结果为所有无人机的路径和总距离和。每条无人机路径的数据类型为一个三元组，包括无人机的所有途经点，每个途经点之间的路程，无人机的单次总路程。便于后续可视化和输出。

核心代码

```
# 暴力算法
def brute(orders, distribution_centers, unload_points):
    total_distance = 0  # 总距离初始化为0
    paths = []  # 存储所有订单的路径信息
    for order in orders:
        path_tot_dis = 0  # 单个订单路径的总距离初始化为0
        paths_dis = []  # 存储单个订单路径的每段距离
        # 选择最近的配送中心
        closest_center = min(distribution_centers,
key=lambda center: calculate_distance(center,
unload_points[order.dp_index]))
        one_path = [closest_center,
unload_points[order.dp_index], closest_center]  # 订单的路径，起始点为最近的配送中心，终点为卸货点，返回起始点
        for i in range(len(one_path)-1):
            dis = calculate_distance(one_path[i],
one_path[i+1])  # 计算路径上每段距离
            paths_dis.append(dis)
            path_tot_dis += dis  # 累加每段距离到路径的总距离
        total_distance += path_tot_dis  # 累加订单的路径总距离到系统的总距离
        paths.append((one_path, path_tot_dis, paths_dis))
 # 将订单的路径信息加入到路径列表中
    return paths, total_distance  # 返回所有订单的路径信息列表和系统的总距离
```

b. 贪心算法：

对于每个路径，选择订单列表中距离任何一个配送中心最近的订单，将那个配送中心和订单加入路径，然后每次选择距离当前位置最近的未处理订单加入路径，直到无法继续添加订单或者超出最大距离限制。 同时，如果即将新加入的订单存在比距离当前位置更近的配送中心，也不再添加。

算法的输入输出格式和暴力算法相同。输出同样为包含所有无人机路径的列表和所用路径的距离之和。

核心代码：

```
# 贪婪算法
def greedy(orders, distribution_centers, unload_points):
    total_distance = 0  # 总距离初始化为0
    paths = []  # 存储所有订单的路径信息
    remaining_orders = orders[:]  # 剩余订单列表，初始化为所有订单的副本

    while remaining_orders:
        path = []  # 当前路径的顺序点列表
        path_distance = 0  # 当前路径的总距离
        path_dists = []  # 当前路径每段距离列表

        # 找到离任意配送中心最近的订单
        min_dis = float('inf')  # 最小距离初始化为无穷大
        closest_order = None  # 最近订单初始化为None
        closest_center = None  # 最近配送中心初始化为None

        for order in remaining_orders:
            for center in distribution_centers:
                distance = calculate_distance(center, unload_points[order.dp_index])  # 计算配送中心到订单卸货点的距离
                if distance < min_dis:
                    min_dis = distance
                    closest_order = order
                    closest_center = center

        # 初始化路径起始点
        current_pos = closest_center  # 当前位置设为最近的配送中心
```

```
        path.append(current_pos)  # 加入路径起始点
        path.append(unload_points[closest_order.dp_index])
 # 加入订单的卸货点
        path_dists.append(min_dis)  # 记录起始点到卸货点的距离
        path_distance += min_dis  # 累加路径总距离
        remaining_orders.remove(closest_order)  # 从剩余订
单中移除已处理的订单
        current_capacity = 1  # 当前无人机容量初始化为1（已经载
有一个订单）
        current_pos =
unload_points[closest_order.dp_index]  # 当前位置更新为最后处
理的卸货点

        # 寻找下一个最近的订单，直到无人机装载满或没有更多订单
        while remaining_orders and current_capacity <
max_items_per_drone:
            min_dis = float('inf')  # 最小距离初始化为无穷大
            next_order = None  # 下一个订单初始化为None

            for order in remaining_orders:
                distance = calculate_distance(current_pos,
unload_points[order.dp_index])  # 计算当前位置到订单卸货点的距
离
                if distance < min_dis:
                    min_dis = distance
                    next_order = order

            if next_order:
                # 检查下一个订单是否能回到最近的配送中心
                fg = False
                next_pos =
unload_points[next_order.dp_index]
                for center in distribution_centers:
                    distance = calculate_distance(center,
next_pos)
                    if distance < min_dis:
                        fg = True
                        break
                if fg:
                    break
```

```
                # 检查是否超出最大距离
                if (path_distance + min_dis +
calculate_distance(next_pos, closest_center) <=
max_distance_per_trip):
                    path.append(next_pos)  # 加入路径下一个
卸货点
                    path_dists.append(min_dis)  # 记录到下
一个卸货点的距离
                    path_distance += min_dis  # 累加路径总距
离
                    current_pos = next_pos  # 当前位置更新为
下一个卸货点
                    current_capacity += 1  # 当前无人机容量
加1
                    remaining_orders.remove(next_order)  #
移除已处理的订单
                else:
                    break
            else:
                break

        # 加入返回最近配送中心的路径
        path.append(closest_center)
        path_dists.append(calculate_distance(current_pos,
closest_center))
        path_distance += path_dists[-1]

        paths.append((path, path_distance, path_dists))  #
将当前路径信息加入到路径列表中
        total_distance += path_distance  # 累加当前路径总距离
到系统的总距离

    return paths, total_distance  # 返回所有订单的路径信息列表
和系统的总距离
```

c. 遗传算法：

每个个体的基因为订单列表的一个排列，表示订单的配送顺序。

对于一个排列，表示一条路径将首先配送这个排列的首元素订单，然后按照排列的顺序继续配送剩下的订单，直到达到无人机一次带货上限或是从任何一个配送中心出发和返回都会达到无人机的最大路程。选择使这条路线总距离最短的配送中心作为这条路线的起点和终点。然后在排列剩下的部分上继续分配路径，直到所有的订单都被处理。

因此对于每一个排列，都对应着唯一一种无人机路径。同时由于目标为无人机总路程最短，则尽可能让每条路径更长。因此最优的无人机路径可被表示为一个排列。最优路径可能被遗传算法得到。

将基因转化为无人机路径函数的输入为个体的基因和配送中心，卸货点的坐标；输出与暴力和贪心算法相同，为包含所有无人机路径的列表和所用路径的距离之和。

核心代码

```
def convert_to_path(individual, distribution_centers,
unload_points):
    all_drone_dis = 0  # 用于记录所有无人机的总距离
    paths = []  # 用于存储每条路径的详细信息
    op = 0  # 操作指针，指向当前处理的订单

    # 循环直到处理完所有订单
    while op < len(individual):
        best_distance = float('inf')  # 用于记录最佳路径的最短
距离
        best_trip_len = 0  # 记录最佳路径的长度（订单数）
        best_start = None  # 用于记录最佳路径的起点配送中心
        best_trip_path = []  # 用于记录最佳路径的订单点序列
        best_path_dis = []  # 记录最佳路径的各段距离

        # 遍历每个配送中心，尝试从每个配送中心开始构建路径
        for center in distribution_centers:
            gene_pos = op  # 基因指针，指向当前订单位置
            trip_dis = 0  # 当前路径的总距离
            cur_pos = center  # 当前路径的当前位置
            trip_len = 0  # 当前路径的订单数
            trip_path = [center]  # 当前路径，从配送中心开始
            path_dis = []  # 当前路径的各段距离

            # 构建路径，直到达到订单数量上限或距离上限
```

```
            while gene_pos < len(individual) and \
                    (trip_dis +
calculate_distance(cur_pos,
unload_points[individual[gene_pos]]) +

calculate_distance(unload_points[individual[gene_pos]],
center)) <= max_distance_per_trip and \
                    trip_len < max_items_per_drone:
                # 计算当前点到下一个订单点的距离，并加入路径

 path_dis.append(calculate_distance(cur_pos,
unload_points[individual[gene_pos]]))
                trip_len += 1
                trip_dis += calculate_distance(cur_pos,
unload_points[individual[gene_pos]])
                cur_pos =
unload_points[individual[gene_pos]]

 trip_path.append(unload_points[individual[gene_pos]])
                gene_pos += 1

            # 更新最佳路径
            if best_trip_len < trip_len or (trip_len > 0
and trip_len == best_trip_len and
                                            trip_dis +
calculate_distance(cur_pos, center) < best_distance):
                best_trip_path = trip_path + [center]  #
返回起始点
                best_trip_len = trip_len
                best_distance = trip_dis +
calculate_distance(cur_pos, center)
                best_start = center
                best_path_dis = path_dis +
[calculate_distance(cur_pos, center)]  # 加入返回起始点的距离

        op += best_trip_len  # 更新操作指针
        all_drone_dis += best_distance  # 更新总距离
        paths.append((best_trip_path, best_distance,
best_path_dis))  # 记录当前最佳路径

    return paths, all_drone_dis  # 返回所有路径及总距离
```

生成初始种群

根据设定的种群大小，生成相应数量的订单列表的排列

```
# 生成初始种群
def generate_initial_population(orders, size):
    population = []
    for _ in range(size):
        individual = []
        random.shuffle(orders)
        for order in orders:
            individual.append(order.dp_index)
        population.append(individual)
    return population
```

计算适应度

将个体对应的基因型通过 `convert_to_path` 函数转化为无人机路径得到的总路径和即为个体的适应度。

```
# 计算适应度函数
def calculate_fitness(individual, distribution_centers,
unload_points):
    _, total_distance = convert_to_path(individual,
distribution_centers, unload_points)
    return total_distance
```

选择函数

根据需要选择的种群数量，选择适应度最好的个体

```
  # 选择函数
  def selection(population, fitnesses, num_parents):
        selected_indices = np.argsort(fitnesses)
[:num_parents]
        return [population[i] for i in selected_indices]
```

交叉变异

使用部分匹配交叉算法。选择两个父代基因序列的部分段进行交叉生成两个子代。由于基因型为订单的排列，因此需要对不合法的子代基因型进行修改。分别统计父代和子代基因型中所有数字出现的次数和种类，对子代基因中出现次数过多或过少的位置进行修改，使其成为合理序列。

核心代码

```
def pmx_crossover(parent1, parent2):
    size = len(parent1)  # 获取基因长度

    # 随机选择两个交叉点
    c1 = random.randint(0, size - 2)
    c2 = random.randint(c1 + 1, size - 1)

    # 分别获取父代的前缀、中间段和后缀
    prefix1, middle1, suffix1 = parent1[:c1],
parent1[c1:c2], parent1[c2:]
    prefix2, middle2, suffix2 = parent2[:c1],
parent2[c1:c2], parent2[c2:]

    # 通过交叉生成子代
    child1 = prefix1 + middle2 + suffix1
    child2 = prefix2 + middle1 + suffix2

    # 辅助函数：计算每个数字出现的索引位置
    def count_numbers(numbers):
        count = {}
        for i in range(size):
            if numbers[i] in count:
                count[numbers[i]].append(i)
            else:
                count[numbers[i]] = [i]
        return count

    cnt = count_numbers(parent1)  # 父代中每个数字的位置
    cnt_child = count_numbers(child1)  # 子代中每个数字的位置

    # 修正子代1中的重复元素
    for i in range(size):
        if len(cnt_child[child1[i]]) >
len(cnt[child1[i]]):
```

```
            cnt_child[child1[i]].remove(i)
            for jj in range(size):
                if parent1[jj] not in cnt_child:
                    child1[i] = parent1[jj]
                    cnt_child[parent1[jj]] = [i]
                    break
                elif len(cnt_child[parent1[jj]]) <
len(cnt[parent1[jj]]):
                    child1[i] = parent1[jj]
                    cnt_child[parent1[jj]].append(i)
                    break

    cnt_child = count_numbers(child2)  # 子代2中每个数字的位
置

    # 修正子代2中的重复元素
    for i in range(size):
        if len(cnt_child[child2[i]]) >
len(cnt[child2[i]]):
            cnt_child[child2[i]].remove(i)
            for jj in range(size):
                if parent1[jj] not in cnt_child:
                    child2[i] = parent1[jj]
                    cnt_child[parent1[jj]] = [i]
                    break
                elif len(cnt_child[parent1[jj]]) <
len(cnt[parent1[jj]]):
                    child2[i] = parent1[jj]
                    cnt_child[parent1[jj]].append(i)
                    break

    # 按基因序列排序子代
    child1 = sorted(child1, key=lambda x:
(child1.index(x), x))
    child2 = sorted(child2, key=lambda x:
(child2.index(x), x))

    return child1, child2  # 返回修正后的子代
```

基因突变

在一个基因中，按照一定的概率，随机选择两个位置，交换它们的值

```
    def mutation(individual, mutation_rate):
        size = len(individual)
        for i in range(size):
            if random.random() < mutation_rate:
                swap_idx = random.randint(0, size - 1)
                # 交换两个位置的值
                individual[i], individual[swap_idx] =
individual[swap_idx], individual[i]
        return individual
```

遗传算法主函数：

首先生成初始种群，然后按照指定的迭代次数进行迭代，在每次迭代过程中，选择适应度最好的一半个体进行随机交叉变异生成新个体，然后子代个体按照一定的比例进行基因突变。最终选择适应度最好的个体的基因型，将其转化为无人机路径。

返回值和其他算法相同，为包含所有无人机路径的列表和所用路径的距离之和。

```
def genetic(orders, distribution_centers, unload_points,
pop_size, num_generations):
    mutation_rate = 0.2
    population = generate_initial_population(orders,
pop_size)
    for generation in range(num_generations):
        fitnesses = [calculate_fitness(individual,
distribution_centers, unload_points) for individual in
population]
        # if((generation+1)%10==0):
        #
print("generation",generation+1,population[np.argmin(fitne
sses)] , "best fitness:", np.min(fitnesses))
        parents = selection(population, fitnesses,
pop_size // 2)
        next_population = []
        while len(next_population) < pop_size:
            parent1, parent2 = random.sample(parents, 2)
            child1, child2 = pmx_crossover(parent1,
parent2)
```

```
            child1 = mutation(child1, mutation_rate)
            child2 = mutation(child2, mutation_rate)

            next_population.extend([child1, child2])
        population = next_population

    fitnesses = [calculate_fitness(individual,
distribution_centers, unload_points) for individual in
population]
    best_individual = population[np.argmin(fitnesses)]
    return convert_to_path(best_individual,
distribution_centers, unload_points)
```

d. 粒子群算法

基因型及与路径的对应关系与遗传算法相同，每个个体的基因为订单列表的一个排列，表示订单的配送顺序。每个基因型使用同样的方法转化为无人机路径。因此 `convert_to_path` 函数与遗传算法相同。

生成初始种群

由于基因型与遗传算法相同，因此初始种群生成函数 `generate_initial_population` 与遗传算法相同。

适应度计算

与遗传算法 `calculate_fitness` 相同。

基因交叉

由于粒子群算法的关键在于如何向表现较好的个体学习，标准粒子群算法引入惯性因子 `w` 、自我认知因子 `c1` 、社会认知因子 `c2` 分别作为自身、当代最优解和历史最优解的权重，指导粒子速度和位置的更新。

由于在这个问题中，速度位置的更新则难以直接采用加权的方式进行，因此采用基于遗传算法交叉算子的混合型粒子群算法进行求解，这里采用顺序交叉算子，对惯性因子 `w` 、自我认知因子 `c1` 、社会认知因子 `c2` 来说，子代分别以 $\frac{w}{w+c1+c2}$ ，$\frac{c1}{/}w+c1+c2$，$\frac{c2}{/}w+c1+c2$的概率接受粒子本身、当前最优解、全局最优解交叉的父代之一。

与遗传算法相同，由于基因为订单的排列，因此对子代进行调整，使其数字出现种类和次数与父代一致。

核心代码

```
    def crossover(individual, cur_best_individual,
global_best_individual, w, c1, c2):
        child = [None] * len(individual)
        parent1 = individual

        # 轮盘赌操作选择parent2
        randNum = random.uniform(0, sum([w, c1, c2]))
        if randNum <= w:
            parent2 = [individual[i] for i in
range(len(individual)-1, -1, -1)]
        elif randNum <= w + c1:
            parent2 = cur_best_individual
        else:
            parent2 = global_best_individual

        # parent1 -> child
        start_pos = random.randint(0, len(parent1)-1)
        end_pos = random.randint(0, len(parent1)-1)
        if start_pos > end_pos:
            start_pos, end_pos = end_pos, start_pos
        child[start_pos:end_pos+1] =
parent1[start_pos:end_pos+1].copy()

        # parent2 -> child
        list1 = list(range(0, start_pos))
        list2 = list(range(end_pos+1, len(parent2)))
        list_index = list1 + list2
        j = -1
        for i in list_index:
            for j in range(j+1, len(parent2) + 1):
                if child.count(parent2[j]) <
parent2.count(parent2[j]):
                    child[i] = parent2[j]
                    break

        return child
```

粒子群算法主函数

首先生成初始种群，然后按照指定的迭代次数进行迭代，在每次迭代过程中，每个个体按照一定的概率进行交叉。最终选择适应度最好的个体的基因型，将其转化为无人机路径。

返回值和其他算法相同，为包含所有无人机路径的列表和所用路径的距离之和。

```
def PSO(orders, distribution_centers, unload_points,
pop_size, num_generations):
    w = 0.2 # 惯性因子
    c1 = 0.4 # 自我认知因子
    c2 = 0.4 # 社会认知因子
    cur_best_dis,cur_best_individual =0,[] # 当前最优值、当
前最优解，（自我认知部分）
    global_best_dis,global_best_individual = 0,[] # 全局最
优值、全局最优解，（社会认知部分）
    population = generate_initial_population(orders,
pop_size)
    fitnesses = [calculate_fitness(individual,
distribution_centers, unload_points) for individual in
population] # 计算种群适应度
    global_best_dis = cur_best_dis = min(fitnesses) # 全局
最优值、当前最优值
    global_best_individual = cur_best_individual =
population[fitnesses.index(min(fitnesses))] # 全局最优解、当
前最优解

    for generation in range(num_generations):
        for i in range(len(population)):
            population[i] =
crossover(population[i],cur_best_individual,global_best_in
dividual,w,c1,c2)
            fitnesses[i] =
calculate_fitness(population[i], distribution_centers,
unload_points)

        cur_best_dis,cur_best_individual =
 min(fitnesses),population[fitnesses.index(min(fitnesses))
]
        if min(fitnesses) <= global_best_dis:
```

```
            global_best_dis,global_best_individual =
 min(fitnesses),population[fitnesses.index(min(fitnesses))
]

    return convert_to_path(global_best_individual,
distribution_centers, unload_points)
```

3. 可视化：

对于地图上的配送中心、卸货点、和无人机的位置，使用 `Pygame` 库进行可视化展示。

初始化：

```
# Pygame初始化
pygame.init()  # 初始化Pygame库
screen_size = 800  # 主屏幕大小
info_width = 400  # 信息区域宽度
screen = pygame.display.set_mode((screen_size +
info_width, screen_size))  # 创建屏幕，大小为主屏幕大小加上信息
区域宽度
pygame.display.set_caption("Drone Delivery Simulation")  #
设置窗口标题
```

坐标转换

```
# 坐标转换函数：将地图坐标转换为屏幕坐标
def to_screen_coords(point, map_size, screen_size):
    if type(point) == list or type(point) == tuple:
        return (point[0] + map_size) * screen_size // (2 *
map_size), (point[1] + map_size) * screen_size // (2 *
map_size)
    else:
        return (point.pos[0] + map_size) * screen_size //
(2 * map_size), (point.pos[1] + map_size) * screen_size //
(2 * map_size)
```

绘制地图及点的位置

使用 `pygame` 库中的 `draw.circle` 和 `draw.line` 函数，画不同颜色的圈和线表示配送中心、卸货点、无人机和路径

```
# 画地图函数
def draw_map(distribution_centers, unload_points, orders,
paths, drones, elapsed_time):
    screen.fill(WHITE)  # 用白色填充屏幕背景

    # 画配送中心和卸货点
    font = pygame.font.SysFont(None, 24)  # 创建字体对象
    for i, center in enumerate(distribution_centers):
        pygame.draw.circle(screen, BLUE,
to_screen_coords(center, map_size, screen_size), 5)  # 画
配送中心（蓝色圆圈）
        text = font.render(str(center.name), True, BLACK)
 # 渲染编号文本
        screen.blit(text,
tuple(np.array(to_screen_coords(center, map_size,
screen_size)) + np.array([5, -10])))  # 显示编号

    for order in orders:
        pygame.draw.circle(screen, RED,
to_screen_coords(unload_points[order.dp_index], map_size,
screen_size), 10, 2)  # 画订单位置（红色圆圈）

    for i, point in enumerate(unload_points):
        pygame.draw.circle(screen, RED,
to_screen_coords(point, map_size, screen_size), 5)  # 画卸
货点（红色圆圈）
        text = font.render(str(point.name), True, BLACK)
 # 渲染编号文本
        screen.blit(text,
tuple(np.array(to_screen_coords(point, map_size,
screen_size)) + np.array([5, -10])))  # 显示编号

    # 画路径
    for path, _, __ in paths:
        for i in range(len(path) - 1):
            pygame.draw.line(screen, BLACK,
to_screen_coords(path[i], map_size, screen_size),
to_screen_coords(path[i + 1], map_size, screen_size), 1)
 # 画路径线

    # 画无人机
```

```
    for drone in drones:
        pygame.draw.circle(screen, GREEN,
to_screen_coords(drone, map_size, screen_size), 3)  # 画无人机位置（绿色圆圈）
```

显示信息

使用 `screen.blit` 函数，显示文字。在图的右侧显示时间、订单信息和路径信息。

```
def draw_info(orders, temporary_orders,  paths,
current_time):
    font = pygame.font.SysFont(None, 24)
    info_x_start = screen_size + 20

    # 画背景
    pygame.draw.rect(screen, GREY, (screen_size, 0,
info_width, screen_size))

    y_offset = 20
    screen.blit(font.render(f"Current time:
{int(current_time)}/min", True, BLACK), (info_x_start,
y_offset))
    y_offset += 30

    screen.blit(font.render("Orders:", True, BLACK),
(info_x_start, y_offset))
    y_offset += 30
    sorted_orders = copy.deepcopy(orders)
    sorted_orders.sort(key=lambda order:order.dp_index)
    # 显示订单信息
    fg=0
    for i, order in enumerate(sorted_orders):
        text = f"[{order.dp_index}]: ({order.deadline})"
        screen.blit(font.render(text, True, BLACK),
(info_x_start, y_offset))
        if fg==0:
            info_x_start += 150
            fg+=1
        elif fg==1:
            info_x_start += 150
            fg+=1
```

```
            else:
                y_offset += 25
                info_x_start -= 300
                fg = 0
        if fg!=0:
            y_offset += 25
            info_x_start -= fg*150
        y_offset += 20

        screen.blit(font.render("Temporary Orders:", True,
BLACK), (info_x_start, y_offset))
        y_offset += 30
        sorted_orders = copy.deepcopy(temporary_orders)
        sorted_orders.sort(key=lambda order:order.dp_index)
        # 显示订单信息
        fg=0
        for i, order in enumerate(sorted_orders):
            text = f"[{order.dp_index}]: ({order.deadline})"
            screen.blit(font.render(text, True, BLACK),
(info_x_start, y_offset))
            if fg==0:
                info_x_start += 150
                fg+=1
            elif fg==1:
                info_x_start += 150
                fg+=1
            else:
                y_offset += 25
                info_x_start -= 300
                fg = 0
        if fg!=0:
            y_offset += 25
            info_x_start -= fg*150
        y_offset += 20

        screen.blit(font.render("Drones:", True, BLACK),
(info_x_start, y_offset))
        y_offset += 30

        # # 显示路径信息
        for i, (path, _, __) in enumerate(paths):
```

```
        text = f"Drone {i+1}: {path[0].name}"
        for jj in range(1, len(path)):
            text += f" -> {path[jj].name}"
        screen.blit(font.render(text, True, BLACK),
(info_x_start, y_offset))
        y_offset += 25
```

主函数：

在每轮收到算法返回的无人机路径后，使用 `draw_map` 函数绘制地图， `draw_info` 显示信息。使用 `pygame` 的 `event.get()` 函数读取鼠标和键盘的操作，便于控制。

```
        running = True
        clock = pygame.time.Clock()
        elapsed_time = 0
        paused = False
        cur_pos_idx = [0]*len(paths)
        while running:
            if(elapsed_time >= time_interval):
                running = False

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    running = False
                    exit_flag = False
                elif event.type == pygame.KEYDOWN:
                    # 检查是否按下了空格键
                    if event.key == pygame.K_SPACE:
                        # 切换暂停状态
                        paused = not paused
            if(paused):
                continue
            elapsed_time += clock.get_time() / 1200 #
200,1200 # 加快时间步长
            draw_map(distribution_centers,
unload_points,orders, paths, drones, elapsed_time)
            draw_info(orders, temporary_orders, paths,
elapsed_time + current_time)

            # 更新无人机位置并绘制路径
```

```
            for i in range(len(paths)):
                path, path_tot_dis, paths_dis = paths[i]
                travel_time = sum(paths_dis[:
(cur_pos_idx[i]+1)]) / drone_speed
                tot_time = path_tot_dis / drone_speed
                if elapsed_time < tot_time:
                    if(elapsed_time >= travel_time):
                        cur_pos_idx[i]+=1
                        travel_time = sum(paths_dis[:
(cur_pos_idx[i]+1)]) / drone_speed
                    last_time = sum(paths_dis[:
(cur_pos_idx[i])]) / drone_speed
                    if(travel_time-last_time<=1e-9):
                        ratio=0
                    else:
                        ratio = (elapsed_time-last_time) /
(travel_time-last_time)
                    cur_pos = path[cur_pos_idx[i]].pos
                    cur_to = path[cur_pos_idx[i]+1].pos

                    drone_x = cur_pos[0] + (cur_to[0] -
cur_pos[0]) * ratio
                    drone_y = cur_pos[1] + (cur_to[1] -
cur_pos[1]) * ratio

                    pygame.draw.circle(screen, GREEN,
to_screen_coords((drone_x, drone_y), map_size,
screen_size), 3)

            pygame.display.flip()
            clock.tick(60)  # 控制帧率
```

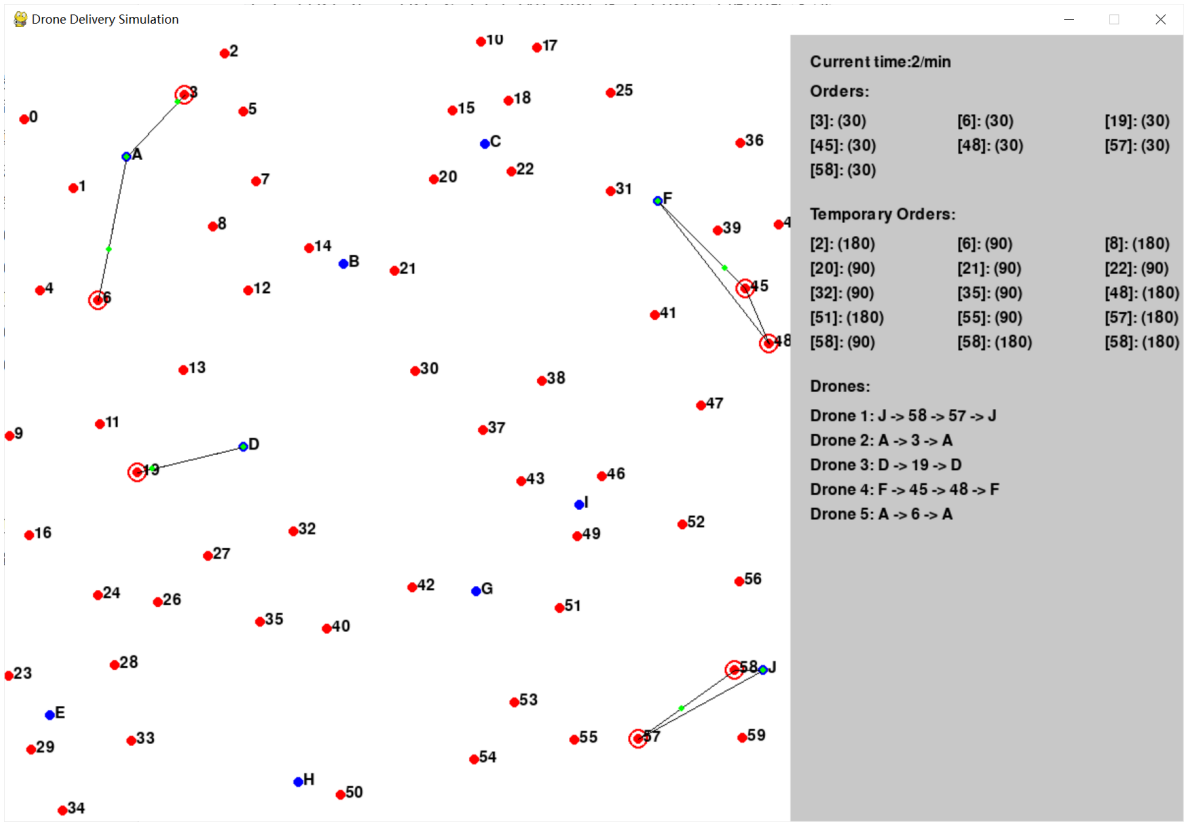
三、实验

代码运行版本为`Python 3.8.16`、`pygame 2.6.0`；

实验运行环境为`Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz`笔记本和`Intel(R) Xeon(R) Gold 6240 CPU @ 2.60GHz` Linux服务器

实验运行截图：

```
(py38) zhangxiangyu@yemang-129-8GPU:~/Algorithm$ python main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
订单数:  22 无人机架次:  7 总距离:  57867.93393286837
订单数:  27 无人机架次:  8 总距离:  78791.59478031663
订单数:  19 无人机架次:  8 总距离:  71501.86204898241
订单数:  16 无人机架次:  7 总距离:  62565.18494159677
订单数:  26 无人机架次:  10 总距离:  74624.42308119622
订单数:  20 无人机架次:  9 总距离:  72586.23368003713
订单数:  14 无人机架次:  7 总距离:  50217.26663282672
订单数:  22 无人机架次:  9 总距离:  67623.5368617351
订单数:  29 无人机架次:  9 总距离:  89227.8962563212
订单数:  15 无人机架次:  8 总距离:  54741.69694920336
订单数:  29 无人机架次:  10 总距离:  78103.05484241199
订单数:  29 无人机架次:  9 总距离:  88955.59707664397
订单数:  17 无人机架次:  6 总距离:  62373.09338856954
订单数:  28 无人机架次:  9 总距离:  91390.41838128229
订单数:  10 无人机架次:  7 总距离:  43827.63623291387
订单数:  13 无人机架次:  5 总距离:  49984.56349955778
订单数:  22 无人机架次:  8 总距离:  64479.844051384134
订单数:  27 无人机架次:  9 总距离:  81119.5309029489
订单数:  29 无人机架次:  10 总距离:  86559.15629859557
订单数:  12 无人机架次:  8 总距离:  45126.934687868394
订单数:  18 无人机架次:  9 总距离:  59665.0116848957
10小时总订单数:  444 总无人机架次:  172 总距离:  1431332.470212156
PSO False
map seed:42, order seed:0
```



实验结果

参数设置：

配送中心数量：10、卸货点数量：60

地图大小：20km × 20km

无人机最大带货数量：5

订单生成时间间隔：30 分钟

每次生成订单数量：10-30

总时长：10小时

遗传算法：

种群数：50、迭代次数：200、变异率：0.2

粒子群算法：

种群数：50、迭代次数：200、惯性因子：0.2、自我认知因子：0.4、社会认知因子：0.4

实验结果

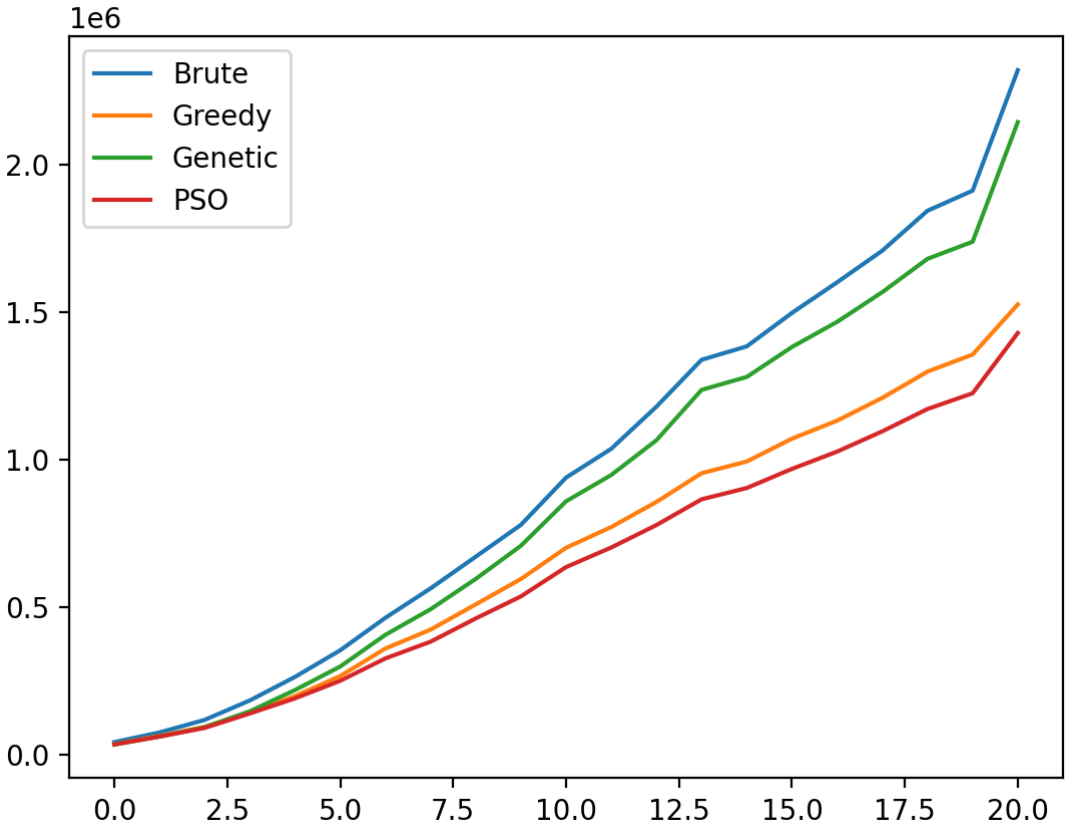
使用订单暂存结果：

订单暂存	暴力	贪心	遗传	粒子群
总距离	2319.695 km	1525.480 km	2143.509 km	1428.417 km
无人机架次	444(总订单数)	236	291	176

不使用订单暂存结果：

无订单暂存	暴力	贪心	遗传	粒子群
总距离	2319.695 km	1573.811 km	2136.311 km	1431.332.417 km
无人机架次	444(总订单数)	250	296	183

随时间变化的无人机总路程如图：



由实验结果可知，暴力算法为算法效果的下限；贪心算法能够得到一个较优解。

在启发式算法中，遗传算法由于在遗传过程中主要依靠随机变异和自由交叉，未能有效从表现较好的个体中遗传基因型，从而促进种族进化；粒子群算法通过引入惯性因子、自我认知因子和社会认知因子，能够有效引导表现不好的个体向表现好的个体学习，从而使得整个种族有效进化。

此外，通过对比是否使用订单暂存方法可知，对于遗传算法来说，由于其最终表现较差，偏向于暴力算法，因此，订单暂存方法并不能有效提升其算法效果，近似于随机影响；而对于表现较好的贪心和粒子群算法，订单暂存方法能够有效降低无人机飞行总架次，同时降低总飞行路径长度。

```
(py38) zhangxiangyu@yemang-129-8GPU:~/Algorithm$ python main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
订单数:  7 暂存订单数:  15 无人机架次:  7 总距离:  41339.93350819373
订单数:  6 暂存订单数:  36 无人机架次:  6 总距离:  32779.505269174726
订单数:  10 暂存订单数:  45 无人机架次:  10 总距离:  42579.3439262033
订单数:  14 暂存订单数:  47 无人机架次:  14 总距离:  65959.22463319794
订单数:  15 暂存订单数:  58 无人机架次:  15 总距离:  79752.38290991218
订单数:  19 暂存订单数:  59 无人机架次:  19 总距离:  90232.04887395685
订单数:  20 暂存订单数:  53 无人机架次:  20 总距离:  110137.93928768014
订单数:  17 暂存订单数:  58 无人机架次:  17 总距离:  99713.77342879328
订单数:  20 暂存订单数:  67 无人机架次:  20 总距离:  107496.41614637
订单数:  25 暂存订单数:  57 无人机架次:  25 总距离:  108019.55333638703
订单数:  31 暂存订单数:  55 无人机架次:  31 总距离:  160222.85552017007
订单数:  19 暂存订单数:  65 无人机架次:  19 总距离:  97644.05499545047
订单数:  26 暂存订单数:  56 无人机架次:  26 总距离:  142695.67827522694
订单数:  32 暂存订单数:  52 无人机架次:  32 总距离:  159049.45599202876
订单数:  10 暂存订单数:  52 无人机架次:  10 总距离:  45423.3754433179
订单数:  22 暂存订单数:  43 无人机架次:  22 总距离:  113468.26695904067
订单数:  18 暂存订单数:  47 无人机架次:  18 总距离:  103737.69966165922
订单数:  19 暂存订单数:  55 无人机架次:  19 总距离:  107361.46769757142
订单数:  23 暂存订单数:  61 无人机架次:  23 总距离:  135301.24765864576
订单数:  14 暂存订单数:  59 无人机架次:  14 总距离:  68145.36685591939
订单数:  77 暂存订单数:  0 无人机架次:  77 总距离:  408636.36836932076
10小时总订单数:  444 总无人机架次:  444 总距离:  2319695.9587482205
brute True
map seed:42, order seed:0
```

```
(py38) zhangxiangyu@yemang-129-8GPU:~/Algorithm$ python main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
订单数:  7 暂存订单数:  15 无人机架次:  5 总距离:  33825.46179164204
订单数:  6 暂存订单数:  36 无人机架次:  5 总距离:  28814.724648510717
订单数:  10 暂存订单数:  45 无人机架次:  7 总距离:  30402.16820999836
订单数:  14 暂存订单数:  47 无人机架次:  9 总距离:  49209.55784122598
订单数:  15 暂存订单数:  58 无人机架次:  7 总距离:  55151.88099072814
订单数:  19 暂存订单数:  59 无人机架次:  12 总距离:  67749.66265448497
订单数:  20 暂存订单数:  53 无人机架次:  12 总距离:  93545.77164408298
订单数:  17 暂存订单数:  58 无人机架次:  9 总距离:  63924.88022380178
订单数:  20 暂存订单数:  67 无人机架次:  12 总距离:  85676.14459852483
订单数:  25 暂存订单数:  57 无人机架次:  19 总距离:  85983.1258623718
订单数:  31 暂存订单数:  55 无人机架次:  15 总距离:  106032.2390293111
订单数:  19 暂存订单数:  65 无人机架次:  11 总距离:  70199.38158026827
订单数:  26 暂存订单数:  56 无人机架次:  13 总距离:  85126.825294954
订单数:  32 暂存订单数:  52 无人机架次:  15 总距离:  97322.97461907347
订单数:  10 暂存订单数:  52 无人机架次:  9 总距离:  39813.578299035406
订单数:  22 暂存订单数:  43 无人机架次:  11 总距离:  77379.03358271322
订单数:  18 暂存订单数:  47 无人机架次:  7 总距离:  61053.78250454912
订单数:  19 暂存订单数:  55 无人机架次:  11 总距离:  77236.00359688443
订单数:  23 暂存订单数:  61 无人机架次:  12 总距离:  88991.74606373835
订单数:  14 暂存订单数:  59 无人机架次:  10 总距离:  57844.25866320289
订单数:  77 暂存订单数:  0 无人机架次:  25 总距离:  170197.7269535176
10小时总订单数:  444 总无人机架次:  236 总距离:  1525480.9286526195
greedy True
map seed:42, order seed:0
```

```
(py38) zhangxiangyu@yemang-129-8GPU:~/Algorithm$ python main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
订单数:  7 暂存订单数:  15 无人机架次:  4 总距离:  33393.8912140662
订单数:  6 暂存订单数:  36 无人机架次:  4 总距离:  26688.149251120292
订单数:  10 暂存订单数:  45 无人机架次:  5 总距离:  31623.26794338678
订单数:  14 暂存订单数:  47 无人机架次:  11 总距离:  54281.800219083794
订单数:  15 暂存订单数:  58 无人机架次:  12 总距离:  71935.35643899793
订单数:  19 暂存订单数:  59 无人机架次:  13 总距离:  79609.9131990312
订单数:  20 暂存订单数:  53 无人机架次:  14 总距离:  107533.31122750779
订单数:  17 暂存订单数:  58 无人机架次:  9 总距离:  86765.05166309465
订单数:  20 暂存订单数:  67 无人机架次:  13 总距离:  102805.14618364401
订单数:  25 暂存订单数:  57 无人机架次:  18 总距离:  112435.45739941891
订单数:  31 暂存订单数:  55 无人机架次:  22 总距离:  150761.19550681332
订单数:  19 暂存订单数:  65 无人机架次:  7 总距离:  89182.912657826
订单数:  26 暂存订单数:  56 无人机架次:  13 总距离:  117775.78106975937
订单数:  32 暂存订单数:  52 无人机架次:  18 总距离:  170828.02621891836
订单数:  10 暂存订单数:  52 无人机架次:  8 总距离:  43814.1116151104
订单数:  22 暂存订单数:  43 无人机架次:  15 总距离:  101494.07580830604
订单数:  18 暂存订单数:  47 无人机架次:  12 总距离:  85212.03042064657
订单数:  19 暂存订单数:  55 无人机架次:  12 总距离:  100778.50684981019
订单数:  23 暂存订单数:  61 无人机架次:  14 总距离:  112928.06539559731
订单数:  14 暂存订单数:  59 无人机架次:  10 总距离:  58073.73698688694
订单数:  77 暂存订单数:  0 无人机架次:  57 总距离:  405589.2498712094
10小时总订单数:  444 总无人机架次:  291 总距离:  2143509.0371402353
genetic True
pop_size: 50 num_generations: 200
map seed:42, order seed:0
```

```
(py38) zhangxiangyu@yemang-129-8GPU:~/Algorithm$ python main.py
pygame 2.6.0 (SDL 2.28.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
订单数:  7 暂存订单数:  15 无人机架次:  4 总距离:  33393.8912140662
订单数:  6 暂存订单数:  36 无人机架次:  4 总距离:  26688.149251120292
订单数:  10 暂存订单数:  45 无人机架次:  6 总距离:  29375.420287957193
订单数:  14 暂存订单数:  47 无人机架次:  7 总距离:  49043.87385063131
订单数:  15 暂存订单数:  58 无人机架次:  6 总距离:  51105.3226116464
订单数:  19 暂存订单数:  59 无人机架次:  8 总距离:  59653.4274878006
订单数:  20 暂存订单数:  53 无人机架次:  6 总距离:  75805.99576847992
订单数:  17 暂存订单数:  58 无人机架次:  6 总距离:  56716.29203532118
订单数:  20 暂存订单数:  67 无人机架次:  8 总距离:  78798.15918187813
订单数:  25 暂存订单数:  57 无人机架次:  11 总距离:  74801.61926422999
订单数:  31 暂存订单数:  55 无人机架次:  12 总距离:  99377.92283409595
订单数:  19 暂存订单数:  65 无人机架次:  7 总距离:  66404.01854563014
订单数:  26 暂存订单数:  56 无人机架次:  10 总距离:  76111.39734780992
订单数:  32 暂存订单数:  52 无人机架次:  10 总距离:  87350.55460495583
订单数:  10 暂存订单数:  52 无人机架次:  7 总距离:  38204.3144708279
订单数:  22 暂存订单数:  43 无人机架次:  7 总距离:  64868.7702947039
订单数:  18 暂存订单数:  47 无人机架次:  5 总距离:  58594.90956664356
订单数:  19 暂存订单数:  55 无人机架次:  7 总距离:  68696.69604742902
订单数:  23 暂存订单数:  61 无人机架次:  8 总距离:  75695.4283729557
订单数:  14 暂存订单数:  59 无人机架次:  8 总距离:  53446.54877286537
订单数:  77 暂存订单数:  0 无人机架次:  29 总距离:  204284.97597671376
10小时总订单数:  444 总无人机架次:  176 总距离:  1428417.6877877624
PSO True
map seed:42, order seed:0
```

个体数及迭代次数的影响：

个体数	迭代次数	遗传算法	粒子群算法
50	200	2143.509 km / 291架次	1428.417 km / 176 架次
100	200	2110.018 km / 275架次	1394.055 km / 172 架次
100	300	2105.520 km / 276架次	1389.475 km / 169 架次

实验结果证明，在设定不变的情况下，增加个体数和迭代次数都能有效提升算法的效果；在设定相同的情况下，粒子群算法能够比遗传算法更好地从表现好的个体中学习，从而促进种群进化，得到更优个体。