武汉大学国家网络安全学院

《高级算法设计与分析》课程报告

基于凸优化和启发式算法的无人机配送路径规划问题实验报告

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 李妍

学　　号 2023202210161

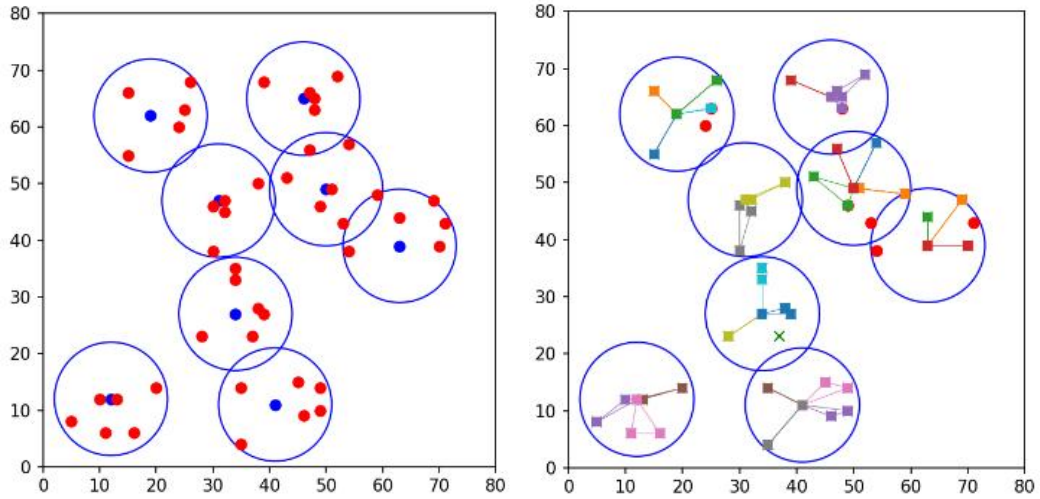
实验学期 2023 - 2024 学年 春季 学期

填写时间 2024 年 06 月 12 日

目录

1 问题描述

无人机可以快速解决最后 10 公里的配送，本实验将设计一个算法，实现一定区域内的无人机配送的路径规划。坐标地图中含有多个配送区域，其中配送区域以无人机在配送区域中单向最远路程为半径作圆划分，圆心即为配送中心，标记为蓝色；在圆内有多个卸货点，标注为红色，如图 1.1(a)所示。算法结果如图 1.1(b)所示，在配送区域内无人机单次配送的路径被标注为同一颜色。



(a) 配送区域　　(b) 算法结果

图 1.1 无人机配送问题配送图及结果

第一章首先对问题进行详细描述，包括配送区域、卸货点生成订单、订单优先级别、算法假设条件、算法目标、算法决策和参数设置七个部分。

1.1 配送区域

在配送区域中，共有 j 个**配送中心**（本实验设置 j = 4、5、6、7），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 6 个**卸货点**，无人机只需要将货物放到相应的卸货点。为考虑情况更为全面，本实验进行了配送中心数量分别为 4、5、6、7 四种情况的实验，其中配送中心数量为 6 的无人机配送区域如图 1.2 所示。

1.2 卸货点生成订单

实验题目将时间离散化，本实验设置每隔 5 分钟，所有的卸货点会生成 0 - 18 个订单。

1.3 订单优先级别

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，用户下订单时，会选择优先级别，优先级别高的付费高。优先级分为三个级别，每种优先级订单最多各六个，优先级具体数据如下：

- **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。

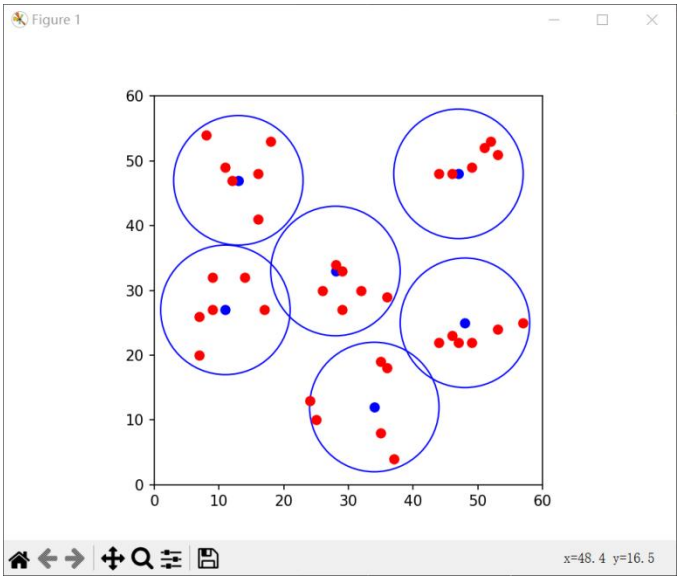


图 1.2 无人机配送区域

1.4 算法假设条件

算法包含如下 5 个假设条件：

- 无人机一次最多只能携带 20 个物品；
- 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；
- 无人机的速度为 60 公里/小时；
- 配送中心的无人机数量无限；
- 任意一个配送中心都能满足用户的订货需求。

1.5 算法目标

本算法的目标是在满足订单优先级别要求的前提下，在一段时间内，所有无人机的总配送路径最短。

1.6 算法决策

每隔 5 分钟，系统做成决策。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。算法决策包括：

- 哪些配送中心出动多少无人机完成哪些订单；
- 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。

1.7 参数设置

上述对题目的具体参数设置已经融入各条件中写出，所有参数设置总结在下表 1.1 中。

表 1.1 参数设置

参数符号	参数值	参数说明
j	4、5、6、7	配送中心数量
k	7	每个配送中心的卸货点数量
t	5	生成订单间隔分钟数
m	18	单次最多生成的订单数
n	20	无人机单次最多携带的物品数

上表展示了实验中具体参数设置，其中 j 有 4 个参数值，表示后续对这 4 个参数分别做了实验，具体实验结果在第 6 章。

2 实验背景

本实验在不考虑优先级等的限制下，可以视作为车辆路径问题（Vehicle Routing Problem, VRP）以及带时间窗的车辆路径问题 VRPTW（Vehicle Routing Problem with Time Windows）的确定性算法问题。因此本实验从基本问题出发，再延伸到具有更多限制性的本特殊问题。下面分别对车辆路径问题、带时间窗的车辆路径问题以及相关求解方法进行介绍。

2.1 车辆路径问题

车辆路径规划问题一般指的是：对一系列发货点和收货点，组织调用一定的车辆，安排适当的行车路线，使车辆有序地通过它们，在满足指定的约束条件下，例如：货物的需求量与发货量，交发货时间，车辆容量限制，行驶里程限制，行驶时间限制等，力争实现一定的目标，如车辆空驶总里程最短，运输总费用最低，车辆按一定时间到达，使用的车辆数最小等。车辆路径规划问题示例如图 2.1 所示，其中蓝色方块表示仓库，黑色圆点表示顾客。一次配送可以在限制条件内同时配送多位顾客，算法目标是使总配送路径最短。

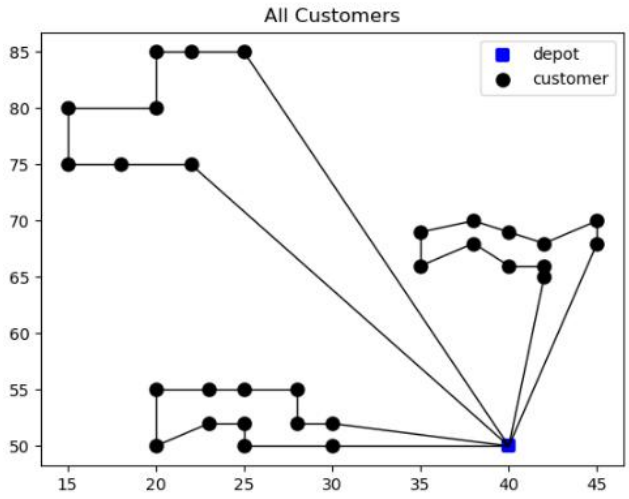


图 2.1 车辆路径规划问题示例

VRP 问题是一个组合优化问题，其中需要考虑多个因素，包括车辆容量限制、路径长度限制等，目标是找到一组最短路径来满足所有的约束条件。建立线性模型表述 VPR 问题，相关决策变量如表 2.1 所示：

表 2.1 VRP 问题模型数学符号

数学符号	表示含义
z	最短路径
c_{ij}	点 i 到点 j 的距离
x_{ijk}	车辆 k 由点 i 驶向点 j 的事件是否发生
r_i	点 i 的货运任务重量
y_{ik}	点 i 的货运任务由车辆 k 来完成的事件是否发生
w_k	车辆 k 的最大承重量

VRP 数学模型如下：

$$min\ z\ =\ \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{m}c_{ij}x_{ijk},$$

$$s.t.\ \sum_{i=1}^{n}r_i y_{ik} \le w_k, k = 1,2,\ldots\ldots,m,$$

$$\sum_{k=1}^{m}y_{ik} = 1, i = 1,2,\ldots\ldots,n,$$

$$\sum_{k=1}^{m}x_{ijk} = y_{jk}, j = 0,1,2,\ldots\ldots,n;\ k = 1,2,\ldots\ldots,m,$$

$$\sum_{k=1}^{m}x_{ijk} = y_{ik}, i = 0,1,2,\ldots\ldots,n;\ k = 1,2,\ldots\ldots,m,$$

$$y_{ik} = 1\ or\ 0, i = 0,1,2,\ldots\ldots,n;\ k = 1,2,\ldots\ldots,m,$$

$$x_{ijk} = 1\ or\ 0, i,j = 0,1,2,\ldots\ldots,n;\ k = 1,2,\ldots\ldots,m.$$

在上述模型数学表达式释义如下

- 目标函数是为了最小化所有车辆的总行驶距离成本；
- 约束 1 是为了保证每辆车装载的货运总量不得超过自身的最大承重量；
- 约束 2 表示每个需求点由且仅由一辆车送货；
- 约束 3 表示若客户点 j 由车辆 k 送货，则车辆 k 必由某点 i 到达点 j；
- 约束 4 表示若客户点 i 由车辆 k 送货，则车辆 k 送完该点的货后必到达另一点 j；
- 约束 5 中y_{ik}表示客户点 i 的货运任务由车辆 k 来完成；当事件发生时取 1，否则取 0；
- 约束 6 中x_{ijk}表示车辆 k 由点 i 驶向点 j，当事件发生时取 1，否则取 0。

根据上述模型可以看出，VRP 问题主要关注于车辆容量限制和最短路径限制。而本题中每个卸货点会随机生成数量不定的订单，且需要考虑订单的优先级别，需要在 VRP 问题上做进一步改进才能使算法做出符合本问题的订单配送决策。

2.2 带时间窗的车辆路径问题

实验题目中增加了优先级的约束，不同优先级对应不同配送时间，因此需要考虑带时间窗相应问题。为了考虑配送时间要求，带时间窗的车辆路径规划问题（VRPTW）应运而生。VRPTW 不仅考虑 VRP 的所有约束，还需要考虑时间窗约束，也就是每个顾客对应一个时间窗，时间窗的起始值分别代表该点的最早到达时间和最晚到达时间。顾客的需求必须要在其时间窗内被送达。

VRPTW 可以建模为一个混合整数规划问题，先引入下面的决策变量，如表 2.2 所示。

表 2.2 VRPTW 问题模型数学符号

数学符号	表示含义
c_{ij}	点 i 到点 j 的距离
x_{ijk}	车辆 k 由点 i 驶向点 j 的事件是否发生
C	车辆起始点的集合
K	车辆的集合
q_i	第 i 个顾客的需求车容量
Q	车容量的集合
s_{ik}	车辆 k 到达点 i 的时间
t_{ij}	从点 i 驶向点 j 花费的时间
M	一个足够大的正整数
A	(i,j)形成的集合
V	从点 i 到点 j 边形成的集合

综合上面引出的决策变量，VRPTW 的标准模型如下：

$$min \sum_{k\in K} \sum_{i\in V} \sum_{j\in V} c_{ij} x_{ijk},$$

$$s.t.\sum_{k\in K}\sum_{j\in V}x_{ijk}=1,\forall i\in C,$$

$$\sum_{j\in V}x_{0jk}=1,\forall k\in K,$$

$$\sum_{i\in V}x_{ihk}-\sum_{j\in V}x_{hjk}=0,\forall h\in C,\forall k\in K,$$

$$\sum_{i\in V}x_{i,n+1,k}=1,\forall k\in K$$

$$\sum_{i\in C}q_i\sum_{j\in V}x_{ijk}\leq Q,\forall k\in K,$$

$$s_{ik}+t_{ij}-M(1-x_{ijk})\leq s_{jk},\forall(i,j)\in A,\forall k\in K,$$

$$e_i\leq s_{ik}\leq l_i,\forall i\in V,\forall k\in K,$$

$$x_{ijk}\in\{0,1\},\forall(i,j)\in A,\forall k\in K$$

VRPTW 数学模型如下：

- 目标函数是为了最小化所有车辆的总行驶距离成本；
- 约束 1 至 4 保证了每辆车必须从仓库出发，经过不同点，最终返回仓库；
- 约束 5 为车辆的容量约束；
- 约束 6 至 7 是时间窗约束，保证了车辆到达每个顾客点的时间均在时间窗内。

根据上述模型可以看出，VRPTW 模型在 VRP 模型的基础上添加了时间窗的限制。而本题中每隔 5 分钟，每个卸货点会随机生成数量不定的订单，且需要考虑无人机飞行的最长路程以及最多携带的物品量。VRPTW 模型中的时间窗约束恰好对应了题目中的优先级约束，并且需要根据无人机的速度以及配货点与卸货点之间的距离进行计算。

2.3 求解方法

2.3.1 传统算法

1. 传统算法概述

综合有关车辆路线问题的求解方法，可以分为精确算法（exact algorithm）与启发式解法（heuristics），其中精密算法有分支界限法、分支切割法、集合涵盖法等；启发式解法有节约法、模拟退火法、确定性退火法、禁忌搜寻法、基因

算法、神经网络、蚂蚁殖民算法等。1995 年，Fisher 曾将求解车辆路线问题的算法分成三个阶段。第一阶段是从 1960 年到 1970 年，属于简单启发式方式，包括有各种局部改善启发式算法和贪婪法（Greedy）等；第二阶段是从 1970 年到 1980 年，属于一种以数学规划为主的启发式解法，包括指派法、集合分割法和集合涵盖法；第三阶段是从 1990 开始至今，属于较新的方法，包括利用严谨启发式方法、人工智能方法等。

2. 启发式算法

由于 VRP 是 NP-hard 问题，难以用精确算发求解，启发式算法是求解车辆运输问题的主要方法，多年来许多学者对车辆运输问题进行了研究，提出了各种各样的启发式方法。车辆运输问题的启发式方法可以分为简单启发式算法、两阶段启发式算法、人工智能方法建立的启发式方法。

简单启发式方法包括节省法或插入法、路线内间节点交换法、贪婪法和局部搜索法等方法。节省法或插入法（savings or insertion）是在求解过程中使用节省成本最大的可行方式构造路线，直到无法节省为止。交换法则是依赖其他方法产生一个起始路线，然后以迭代的方式利用交换改善法减少路线距离，直到不能改善为止。1960 年，Clarke 和 Wright 首先提出一种启发式节省法（savings methods）来建立车队配送路线。简单启发式方法简单易懂、求解速度快，但只适合求解小型、简单的 VRP 问题。

两阶段方法包括先分组后定路线（clusterfirst-route second）和先定路线后分组（routefirst-cluster second）两种启发式策略。前者是先将所有需求点大略分为几个组，然后再对各个组分别进行路线排序；后者则是先将所有的需求点建构成一条路线，再根据车辆的容量将这一路线分割成许多适合的单独路线。

人工智能方法自 1990 年来，在解决组合优化问题上显示出强大功能，在各个领域得到充分应用，很多学者也将人工智能引入车辆路线问题的求解中，并构造了大量的基于人工智能的启发式算法。禁忌搜索法（TS）基本上是属于一种人工智能型（AI）的局部搜寻方法，Willard 首先将此算法用来求解 VRP ，随后亦有许多位学者也发表了求解 VRP 的 TS 算法。西南交通大学的袁庆达等设计了考虑时间窗口和不同车辆类型的禁忌算法，这种算法主要采用 GENIUS 方法产生初始解，然后禁忌算法对初始解优化。模拟退火方法具有收敛速度快，全局

搜索的特点，Osman 对 VRP 的模拟退火算法进行了研究，他提出的模拟退火方法主要适合于解决路线分组。遗传算法具有求解组合优化问题的良好特性，Holland 首先采用遗传算法（GA）编码解决 VRPTW 问题。现在多数学者采用混合策略，分别采用两种人工智能方法进行路线分组和路线优化。Ombuki 提出了用遗传算法进行路线分组，然后用禁忌搜索方法进行路线优化的混合算法。Bent 和 Van Hentenryck 则首先用模拟退火算法将车辆路线的数量最小化，然后用大邻域搜索法（largneighborhood search）将运输费用降到最低。

总结几种人工智能方法可以看出：

- 禁忌搜索算法所得到的解最接近最优解，但其运算时间也最长，是遗传算法的 2～3 倍，SA 算法的近 20 倍；
- 遗传算法也能较好的逼近最优解，同时使运算时间大大缩短，所以遗传算法能兼顾运算时间和效率两方面，是具有较好的发展前途的方法；
- 模拟退火算法求解速度非常快，也能提供一定程度上的优化方案在求解较小规模问题上具有较好效果。

2.3.2 运筹优化求解器

原问题可以建模为旅行商问题，而旅行商问题是 NPC 问题，找到原问题的多项式时间算法是不可能的，因此需要在性能与时间中折衷，虽然求解器并不能保证多项式时间求解，但其可以保证与每个配送中心的最优解的差距在 1e-3%以下。对于原问题来说，本实验算法更关注性能，而求解时间只需对问题规模而言是可接受的。且对于我们设定的问题规模（8 个中心 64 个配送点），最终设计的算法在家用 PC 上每轮规划不超过 20s，对于题目中的场景而言是可接受的。

针对 VRP 问题以及 VRPTW 问题，具有较多运筹优化求解器，求解器是闭源的。闭源求解器通常在追求精度的同时，不得不牺牲一定的计算速度。这是因为为了确保问题的准确解决，常常需要使用复杂的算法和精细的优化技术，这些算法往往不能在多项式时间内完成计算，因此必须接受一定的时间成本以换取问题解的高精度。下面对相关求解器作以介绍。

1. Gurobi

Gurobi 是由美国 Gurobi Optimization 公司开发新一代大规模优化器，提供 C++, Java, Python, .Net, Matlab 和 R 等多种接口，支持求解以下模型：

- 连续和混合整数线性问题；
- 凸目标或约束连续和混合整数二次问题；
- 非凸目标或约束连续和混合整数二次问题；
- 含有对数、指数、三角函数、高阶多项式目标或约束，以及任何形式的分段约束的非线性问题；
- 含有绝对值、最大值、最小值、逻辑与或非目标或约束的非线性问题。

2. COPT

杉数求解器 COPT（Cardinal Optimizer）是杉数自主研发的针对大规模优化问题的高效数学规划求解器套件，也是支撑杉数端到端供应链平台的核心组件，是目前同时具备大规模混合整数规划、线性规划（单纯形法和内点法）、半定规划、混合整数二阶锥规划以及混合整数凸二次规划和混合整数凸二次约束规划问题求解能力的综合性能数学规划求解器，为企业应对高性能求解的需求提供了更多选择。COPT 支持所有主流编程接口：C、C++、C#、Python、Julia、Java、AMPL、GAMS、Pyomo、PuLP、CVXPY。

3. SCIP

SCIP 起源于 ZIB (Zuse Institute Berlin)，由 Tobias Achterberg 奠定整个框架，是目前用于混合整数规划（MIP）和混合整数非线性规划（MINLP）的最快的非商业解算器之一，它允许用户对求解过程进行全面控制，支持自定义搜索树中的各个模块，在分支限界 (Branch and Bound) 过程中添加变量等功能。SCIP 支持 Python、Java、AMPL、GAMS、MATLAB 等编程语言。

3 实验思路

实验思路从问题本身出发，再逐步结合相关限制条件。首先在不考虑会随机生成订单，用 VRPTW 算法来求解，得到最短路径；然后根据优先级别和距离限制对目标点聚类，得出无人机下一次飞行的目标点。本章对实验思路进行介绍，主要包括问题建模、距离决策的启发式算法和时间决策的启发式算法三个部分。

3.1 问题建模

本实验原问题与 VRPTW 相似，如果只考虑某一时刻已经的订单，而不考虑未来的订单，则可以建模为 VRPTW 问题，使用 Gurahipy 求解器求解，返回最优静态解。在每一时刻，会有一些订单产生，订单与物品需求、优先级、地点这三个元素有关，将其建立成三元组。在不考虑随机生成订单的时候，已知需要配送的订单，配送地点为订单地点，时间窗为 0 到相应优先级的阈值，可以将实验问题建模为 VRPTW，根据 VRPTW 的数学模型，引入下面的决策变量，如表 3.1 所示。

表 3.1 无人机配送路径规划问题决策变量

数学符号	表示含义
c_{ij}	点 i 到点 j 的距离
x_{ijk}	无人机 k 由点 i 驶向点 j 的事件是否发生
C	无人机配送中心的集合
K	无人机集合
q_i	第 i 个顾客的订单需求
Q	无人机容量的集合
s_{ik}	无人机 k 到达点 i 的时间
t_{ij}	从点 i 驶向点 j 花费的时间
T	无人机的最长续航时间
M	一个足够大的正整数
A	(i,j)形成的集合
V	从点 i 到点 j 边形成的集合

综合上面引出的决策变量，VRPTW 的标准模型如下：

$$min \sum_{k\in K} \sum_{i\in V} \sum_{j\in V} c_{ij} x_{ijk},$$

$$s.t. \sum_{k\in K} \sum_{j\in V} x_{ijk} = 1, \forall i \in C,$$

$$\sum_{j\in V} x_{0jk} = 1, \forall k \in K,$$

$$\sum_{i\in V} x_{ihk} - \sum_{j\in V} x_{hjk} = 0, \forall h \in C, \forall k \in K,$$

$$\sum_{i\in V} x_{i,n+1,k} = 1, \forall k \in K$$

$$\sum_{i\in C} q_i \sum_{j\in V} x_{ijk} \leq Q, \forall k \in K,$$

$$s_{ik} + t_{ij} - M(1 - x_{ijk}) \leq s_{jk}, \forall (i,j) \in A, \forall k \in K,$$

$$e_i \leq s_{ik} \leq l_i, \forall i \in V, \forall k \in K,$$

$$\sum_{i\in C} \sum_{j\in V} t_{ij} \leq T, \forall (i,j) \in A$$

$$x_{ijk} \in \{0,1\}, \forall (i,j) \in A, \forall k \in K$$

VRPTW 数学模型如下：

- 目标函数是为了最小化所有无人机的总路程；
- 约束 1 至 4 保证了每辆无人机必须从配送点出发，经过不同点，最终返回配送点；
- 约束 5 为无人机携带物品的数量约束，即容量约束；
- 约束 6 至 7 是时间窗约束，保证了车辆到达每个顾客点的时间均在时间窗内；
- 约束 8 是无人机的续航约束，需保证无人机的往返总续航时间在范围内。

原问题每个时刻的配送情况都需要满足上述条件，在 VRPTW 的基础上增加了无人机的续航约束，同时满足其优先级，即时间窗限制。

此时原问题转变为，每一轮需要处理哪些订单，说明由于存在未来未知因素，现在还无法达到最优解，所以还需采用启发式算法。对于目标点聚类，即决策无人机下一次飞行经过的所有目标点，根据题意由优先级别和距离远近这两个因素做出聚类决策。本实验根据距离和优先级设计了两个启发式算法，分别在

3.2 和 3.3 节进行叙述，即优先级较高、即很紧急的订单，以及距离无人机得近的订单会被优先考虑被选取。若存在一个点被任意一个启发式函数选中，就将其加入到本轮需要送的集合中，然后遍历所有点后，对被选中的点视为一个 VRPTW 求解。

3.2 启发式算法 1：距离决策

第一个启发式算法是由距离决策的，即离得近的可以一起送，当距离相近的配送点内有多个点需要配送可以一起配送，如图 3.1 中左下两个蓝色点相隔较近，所以一次配送即可完成。但问题是配送意义上的“离得近”与常规聚类得到的离得近不同，常规聚类的结果不一定能一起送，如图 3.1 中与配送中心距离较近的绿色点与配送中心以下的黄色点及蓝色点距离都较近，但是如果这四个点聚类为一个订单，那么较远的、在配送范围边界的点则需要单独派出无人机配送，这显然是低效的。即边界的两个点即使离得很近，如果将这两个订单同时配送，就会超出 20km 续航限制，这样情况下对这两个点进行聚类是“错误”的，因为并不能省路程，所以本实验提出一种新的“聚类方法”，基于路径的聚类。

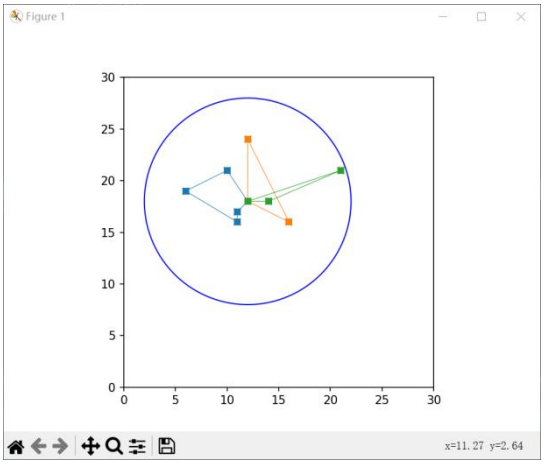


图 3.1 订单配送示例

基于路径聚类的距离决策方法思路是：模拟一次配送，假设配送无人机载重无限，每个配送点只需要一个货物，那么在这种情况下无人机会尽量将相近的卸货点订单一起配送，这样会被一起被配送订单的卸货点就会被认为离得很近，一起配送可以减少总路程。而没有一起送的卸货点则被认为它们离得不近，一起配送比分开送的路程要更多。

对于上述思路，在算法求解的过程中，根据线性规划求解结果进行选择，将一次无人机出行经过卸货点的阈值设为 3，即一个出行内如果有超过 3 个卸货点，

以 1 的概率全选；若有 2 个卸货点，以 60%概率选择该线路；若只有 1 个卸货点，但距离很远时（设置超过半径的 80%），则选择单独送该线路。

3.3 启发式算法 2：时间决策

除去上述距离因素的决策外，实验题目中还有对优先级的约束，若有紧急的但较远的订单，依然需要优先派送，即根据订单的剩余时间进行配送排序，时间越紧急的订单配送概率最大。因此，将每个点被配送的概率设计成 sigmoid 函数，随着剩余时间减少逐渐增加到 1，以订单剩余时间的相应概率去卸货点配送。

对于优先级决策的算法实现，实验中设置最长配送时间 180 分钟。由于配送范围半径为 10km，因此临界情况即卸货点靠近配送边界时，需要一台无人机单独往返。同时考虑到无人机若一次配送多个订单，相对单个订单则会绕路，设置最短配送时间为 20 分钟。对于剩余最长时间 180 分钟的一般不紧急订单，设置配送概率为 0.1；对于剩余 20 分钟的紧急订单，设置配送概率为 1，即所剩时间越短的订单，决策被选择的概率越大，通过这样配送概率的设置保证了每个订单都能配送到。

4 实验配置

实验环境部分主要包括关键库配置和处理器配置，下面进行具体介绍。

4.1 关键库配置

实验涉及的**关键库配置**如下：

- Python3.8
- Gurobipy11.0.2
- Random

4.2 处理器配置

实验涉及的**处理器配置**如下：

- Windows 10
- Intel 9750h

5 代码分析

本章主要介绍实验的主体部分，包括生成无人机配送区域、路径规划和结果可视化三个部分，下面进行具体介绍。

5.1 生成无人机配送区域

本节主要介绍生成无人机配送区域的设计和代码实现，包括参数设置、生成配送中心、生成卸货点和配送中心卸货点可视化四个部分。

5.1.1 参数设置

根据第一章中实验概述中的数据进行参数设置，包括地图边长为 60 单位，有 6 个配送中心，共有 36 个卸货点，根据题意“最后 10 公里”设置配送半径为 10，无人机最长飞行距离为 20，每隔 5 分钟随机生成订单需求，具体代码实现如图 5.1 所示。

```
MAP_SIZE = 60
NUM_CENTERS = 6
NUM_POINTS = 36
RADIUS = 10
max_distance = 20
max_demand = 5
```

图 5.1 参数设置代码

5.1.2 生成配送中心

```
def generate_distribution_centers(map_size, num_centers, radius):
    centers = []
    attempts = 0
    while len(centers) < num_centers and attempts < 1000:
        attempts += 1
        new_center = (round(random.uniform(radius, map_size - radius)), round(random.uniform(radius, map_size - radius)))
        if all(math.sqrt((new_center[0] - c[0]) ** 2 + (new_center[1] - c[1]) ** 2) >= 1.6 * radius for c in centers):
            centers.append(new_center)
    return centers
```

图 5.2 生成配送中心位置代码实现

对于生成配送中心位置，定义 generate_distribution_centers()函数，需要确保每个配送中心之间的距离足够远。首先初始化一个空列表来存储生成的配送中心坐标，并且初始化尝试计数器。当生成的中心数量小于要求的数量并且尝试次数少于 1000 时，继续循环。每次循环增加尝试次数，随机生成一个新的配送中心坐标，至少离边界 radius 的距离，确保不在边界上。检查新生成的配送中心与已经存在的所有配送中心的距离是否都大于等于 1.6 倍的 radius，确保配送中心之

间相隔距离足够大。如果距离条件满足，将新中心添加到列表中。最后，返回生成的配送中心列表。具体代码实现如图 5.2 所示。

5.1.3 生成卸货点

对于生成卸货点的函数，定义 generate_delivery_points()函数，确保这些卸货点围绕在给定的配送中心周围，并且不会与配送中心重合。首先初始化一个空集合 points 来存储生成的卸货点，使用集合是为了自动去重，因为集合不允许重复元素。当生成的卸货点数量小于要求的数量时，继续循环。选择一个配送中心作为当前生成卸货点的参考中心，用模运算来循环选择中心，这样可以均匀分布卸货点到不同的中心。随机生成一个角度 angle，范围从 0 到 2π，用来确定卸货点相对于中心的方向；随机生成一个距离 r，范围从 0 到给定的 radius，用来确定卸货点相对于中心的距离。计算新卸货点坐标，将计算结果四舍五入为整数，并且检查新生成的点是否在地图范围内，确保新点不与任何配送中心重合。果上述条件都满足，将新点添加到 points 集合中，最后返回生成好的卸货点列表，将集合转换为列表形式。

函数通过在给定半径范围内围绕配送中心随机生成角度和距离，来创建卸货点。每生成一个点后，会检查其是否在地图范围内且不与配送中心重合。如果条件符合，就将该点加入集合，直到达到指定数量的卸货点。其中，使用集合来自动去重，以确保每个生成的点都是唯一的。具体代码实现如图 5.3 所示。

```
def generate_delivery_points(map_size, num_points, centers, radius):
    points = set()
    while len(points) < num_points:
        center = centers[len(points) % len(centers)]
        angle = random.uniform(0, 2 * math.pi)
        r = random.uniform(0, radius)
        new_point = (round(center[0] + r * math.cos(angle)), round(center[1] + r * math.sin(angle)))
        if 0 <= new_point[0] <= map_size and 0 <= new_point[1] <= map_size and new_point not in centers:
            points.add(new_point)
    return list(points)
```

图 5.3 生成卸货点代码实现

5.1.4 配送中心卸货点可视化

对于可视化配送中心和卸货点在地图上的分布，定义 plot_map()函数。首先创建一个新的绘图窗口和一对轴对象，用于绘制地图，设置 x 轴和 y 轴的范围为从 0 到 map_size，确保整个地图都能显示在图中；设置轴的纵横比为 1:1， x 轴和 y 轴的单位长度相同，确保绘制的圆形不会变形。遍历每一个配送中心的坐标，

使用 plt.Circle 创建一个以当前中心为圆心、半径为 radius 的圆形对象。将创建好的圆形对象添加到轴对象 ax 中进行绘制，在当前配送中心的位置绘制一个蓝色的点，遍历每一个卸货点的坐标，在配送中心的位置绘制一个红色的点，最后显示绘制好的图形。

函数通过 Matplotlib 库来绘制一个地图，其中包含配送中心及其周围一定半径的圆形区域，以及卸货点。配送中心用蓝色表示，卸货点用红色表示。通过将这些元素绘制在统一的地图上，可以直观地看到配送中心和配送点的分布情况。具体代码实现如图 5.4 所示

```
def plot_map(centers, points, radius, map_size):
    fig, ax = plt.subplots()
    ax.set_xlim(0, map_size)
    ax.set_ylim(0, map_size)
    ax.set_aspect('equal', adjustable='box')

    for center in centers:
        circle = plt.Circle(center, radius, color='blue', fill=False)
        ax.add_artist(circle)
        ax.plot(center[0], center[1], 'bo')

    for point in points:
        ax.plot(point[0], point[1], 'ro')

    plt.show()
```

图 5.4 配送中心卸货点可视化代码实现

5.2 路径规划

本节主要介绍生成路径规划的设计和代码实现，包括对应配送中心和卸货点、不含优先级的路径规划、包含优先级的路径规划三个部分。

5.2.1 对应配送中心与卸货点

根据配送中心和配送点的坐标计算出哪些配送点位于离各个配送中心一定范围内。首先创建一个空列表 incident，用来存储每个配送中心附近的配送点索引，遍历每一个配送中心的坐标。然后，创建一个空列表 incident_points，用来存储当前配送中心附近的配送点索引，遍历每一个配送点的坐标，并使用 enumerate 函数获取到对应的索引值 i。接着，计算当前配送中心与当前配送点之间的距离，判断是否小于等于 10，如果距离满足条件，则将当前配送点的索引 i 添加到 incident_points 列表中，并且将存储了当前配送中心附近的配送点索引的列表 incident_points 添加到 incident 列表中。

遍历每个配送中心，找出距离它一定范围内的配送点，并记录它们的索引。最终生成一个列表 incident，其中每个元素是一个列表，包含了对应配送中心附近的配送点的索引。具体代码实现如图 5.5 所示。

```
incident = []
for center in centers:
    incident_points = []
    for i, point in enumerate(points):
        if math.sqrt((center[0] - point[0]) ** 2 + (center[1] - point[1]) ** 2) <= 10:
            incident_points.append(i)
    incident.append(incident_points)
```

图 5.5 计算配送中心与卸货点对应关系

5.2.2 不含优先级的路径规划

本小节主要介绍不含优先级的路径规划的设计和代码实现，包括构建 VRPTW 模型、路径规划和根据配送中心生成卸货点索引三个部分。

1. 构建 VRPTW 模型

根据实验题目中的限制条件，需要对无人机的路径进行规划约束，这里采用了基于车辆路径问题的路线规划函数，并且结合题目进一步增加限制条件。

首先，初始化了一些字典和列表，用于存储各个节点的信息，如服务时间、横纵坐标等。然后，遍历 points 列表，为每个配送点创建一个唯一的 id 并加入 N 列表中，将每个配送点的坐标、需求量、开始和结束时间等信息存储到相应的字典中。将配送中心 id 存储到 depot 变量中，计算每对节点之间的距离，并将它们存储到 Cost 和 TT 字典中，TT 是通过距离除以速度得到的旅行时间，无人机的时速为 60，构建 VRPTW（Vehicle Routing Problem with Time Windows）模型，设置模型的时间限制为 1500 秒，并设置优化容忍度参数为 1e-2，调用 optimize() 方法对模型进行求解。具体代码实现如图 5.6 所示。

```
depot = N[0]
CAP = 30
Cost = {}
K = list(range(1, NUM_POINTS))
for f_n in N:
    for t_n in N:
        dist = math.sqrt((x_coord[f_n] - x_coord[t_n]) ** 2 + (y_coord[f_n] - y_coord[t_n]) ** 2)
        Cost[f_n, t_n] = dist
        TT[f_n, t_n] = dist / 1
vrptw_model, X, T = build_model(N, Q, TT, ET, LT, ST, Cost, CAP, K, max_distance)
vrptw_model.Params.TimeLimit = 1500
vrptw_model.setParam('OptimalityTol', 1e-2)
vrptw_model.optimize()
route_list = []
```

图 5.6 构建 VRPTW 模型代码实现

2. 路径规划

在模型构建完毕后，遍历所有无人机，初始化一个路径 route，将节点加入到路径中。从剩余节点中找到下一个要访问的节点并添加到路径中，然后从剩余节点列表中移除。循环执行以上步骤，直到回到初始节点。将无人机编号 k 插入到路径的开头，并将路径加入到 route_list 列表中。具体代码如图 5.7 所示。

```
for k in K:
    route = [depot]
    cur_node = depot
    cur_k = None
    for j in N[1:]:
        if X[depot, j, k].x > 0:
            cur_node = j
            cur_k = k
            route.append(j)
            N.remove(j)
            break
    if cur_k is None:
        continue
    while cur_node != depot:
        for j in N:
            if X[cur_node, j, cur_k].x > 0:
                cur_node = j
                route.append(j)
                if j != depot:
                    N.remove(j)
                break
    route.insert(0, k)
    route_list.append(route)

return route_list, vrptw_model.objVal
```

图 5.7 路径规划代码实现

3. 根据配送中心生成卸货点索引

为每个配送中心生成多个子类，每个子类包含符合一定条件的配送点的索引。首先，创建一个空列表 center_sub_classes，用于存储每个配送中心的子类。然后，使用 enumerate 函数遍历每一个配送中心的坐标，并获取对应的索引 i 和中心点 center，并且为当前的配送中心创建一个空列表 sub_class，用于存储该配送中心的子类。接着，使用列表推导式，创建一个新的列表 selected_points，其中包含当前配送中心的所有相关配送点的信息。然后，调用函数 get_route，计算出从当前配送中心 center 到 selected_points 中每个配送点的路径，遍历 route_list 中的每个路径 route，使用列表推导式，将路径中除了起点和终点的节点的索引转换回原来的配送点索引，并添加到 sub_class 列表中。最后，将当前配送中心的子类列表 sub_class 添加到 center_sub_classes 列表中。

根据每个配送中心的相关配送点，使用特定的路径计算方法生成多个路径子类。每个子类包含一系列配送点的索引，这些索引表示在该路径上配送的点。最终，这些子类被存储在 center_sub_classes 列表中，每个元素代表一个配送中心的子类集合。具体代码实现如图 5.8 所示。

```
center_sub_classes = []
for i, center in enumerate(centers):
    sub_class = []
    selected_points = [[points[ex], 1, 999] for ex in incident[i]]
    route_list, _ = get_route(center, selected_points, max_distance)
    for route in route_list:
        sub_class.append([incident[i][ex-1] for ex in route[2:-1]])
    center_sub_classes.append(sub_class)
```

图 5.8 根据配送中心生成卸货点索引代码实现

5.2.3 包含优先级的路径规划

本小节主要介绍包含优先级的路径规划的设计和代码实现，包括模拟动态需求、合并节点需求和添加子类节点三个部分。

1. 模拟动态需求

根据实验题目，需要模拟一个动态的需求生成过程，每隔一定时间步长产生新的需求。首先，初始化时间变量，初始化一个空列表 total_demand 来存储所有的需求，设置时间步长为 5，即每隔 5 个时间单位进行一次需求更新。然后运行循环生成需求，创建一个新的需求列表 total_demand_new，更新现有需求的剩余时间，遍历所有节点，生成随机需求量，如果生成的随机需求量大于 0，并且另一个随机概率 0.5，则向 total_demand 列表中添加一个新的需求。然后，第二次生成随机需求量，以同样的方式，如果随机需求量大于 0 并且随机概率大于 0.5，则将新的需求追加到 total_demand 列表中，但这次固定的服务时间是 90。接着，第三次生成随机需求量，以同样的方式，如果随机需求量大于 0 并且随机概率大于 0.5，则将新的需求追加到 total_demand 列表中，但这次固定的服务时间是 180。

每次迭代，所有现有需求的剩余时间都会减少 time_step。每个节点有三次机会生成一个随机需求，每次生成的需求会有不同的固定服务时间（30, 90 或 180），循环产生动态变化的需求列表。具体代码实现如图 5.9 所示。

```
time = 0
total_demand = []
time_step = 5
while True:
    total_demand_new = [[ex[0], ex[1], ex[2]-time_step, ex[3]] for ex in total_demand]
    total_demand = total_demand_new
    for i in range(NUM_POINTS):
        random_demand = random.randint(0, max_demand)
        if (random_demand > 0) and (random.random() > 0.5):
            total_demand.append([points[i], random_demand, 30, i])
        random_demand = random.randint(0, max_demand)
        if (random_demand > 0) and (random.random() > 0.5):
            total_demand.append([points[i], random_demand, 90, i])
        random_demand = random.randint(0, max_demand)
        if (random_demand > 0) and (random.random() > 0.5):
            total_demand.append([points[i], random_demand, 180, i])
```

图 5.9 模拟动态需求代码实现

2. 合并节点需求

承接上述模拟优先级需求生成后，需要更新 total_demand_new 列表，以反映每个节点的累计需求和最短服务时间限制。首先，初始化一个空列表，用于存储更新后的需求信息。然后，遍历所有节点，初始化时间限制为 180，即每个节点的初始时间限制为 180。遍历所有存在的需求，如果需求对应的节点索引等于当前节点 i，则将该需求的数量加到 need 上，累计该节点所有需求的总量；如果该需求的剩余时间小于当前的 time_limit，则更新 time_limit 为该需求的剩余时间，这样可以确保 time_limit 是当前节点所有需求中最短的服务时间限制。最后，如果该节点有需求，则将该节点的需求信息添加到 total_demand_new 中。

合并同一节点的所有需求，并更新该节点的累计需求量和最短服务时间限制。通过遍历所有节点 i 和所有需求 total_demand，代码将每个节点的需求累加起来，并确定该节点所有需求中的最短服务时间限制。最终，将具有正需求量的节点的信息追加到 total_demand_new 列表中。代码实现如图 5.10 所示。

```
    total_demand_new = []
    for i in range(NUM_POINTS):
        time_limit = 180
        need = 0
        for demand in total_demand:
            if demand[3] == i:
                need = need + demand[1]
                if demand[2] < time_limit:
                    time_limit = demand[2]
        if need > 0:
            total_demand_new.append([points[i], need, time_limit, i])
```

图 5.10 合并节点需求代码实现

3. 添加子类节点

承接上述合并节点需求，需要根据需求和子类中心点的信息来确定哪些子类节点需要添加。首先，将前一段代码计算得到的新的需求列表进行赋值；初始化列表 possi_add，用于记录每个节点是否有可能被添加；初始化空列表 added，用于记录已经添加过的节点；初始化列表 demanded，用于记录每个节点是否有需求；提取 total_demand 中的所有需求点坐标，形成 demanded_points 列表。然后，遍历所有子类中心点 center_sub_classes，对每个子类中心点，初始化一个空列表 added_point。遍历子类中心点中的每个子类 sub_class，对每个子类中的每个点 point，如果该点有需求且未被添加，则累计需求量 total_sub_demanded。

根据累计需求量 total_sub_demanded 来决定是否添加子类中的点：如果 total_sub_demanded 大于 2 或需求比例超过 80%，则将子类中的所有点添加到 added_point 和 added 列表中。否则，如果 total_sub_demanded 大于 1 且随机事件发生概率 66%，则将子类中的所有点添加到 added_point 和 added 列表中。最后，将每个子类中心点的添加节点列表 added_point 添加到 center_added_point 中。

```
total_demand = total_demand_new
possi_add = [0] * NUM_POINTS
added = []
demanded = [0] * NUM_POINTS
demanded_points = [ex[0] for ex in total_demand]
for i, point in enumerate(points):
    if point in demanded_points:
        demanded[i] = 1
center_added_point = []
for center_sub_class in center_sub_classes:
    added_point = []
    for sub_class in center_sub_class:
        total_sub_demanded = 0
        for point in sub_class:
            if demanded[point] == 1 and point not in added:
                total_sub_demanded = total_sub_demanded + 1
        if total_sub_demanded > 2 or (total_sub_demanded / len(sub_class)) > 0.8:
            for point in sub_class:
                if point not in added:
                    added_point.append(point)
                    added.append(point)
        elif total_sub_demanded > 1 and random.random() < 0.66:
            for point in sub_class:
                if point not in added:
                    added_point.append(point)
                    added.append(point)
    center_added_point.append(added_point)
```

图 5.11 合并节点需求代码实现

根据需求和子类中心点的信息，决定哪些节点需要添加。具体是按照需求总量和随机因素来确定是否添加某些子类中的节点。最终，center_added_point 列表将包含所有需要添加的节点，按照子类中心点的分类进行组织。代码实现如图 5.11 所示。

根据需求的概率和随机数，将符合条件的需求点添加到相应的 incident 中。遍历所有需求点 total_demand，对于每个需求点，通过调用 cal_possi 函数计算需求点的概率，并与一个随机数比较。如果随机数小于需求点的概率，且需求点在某个 incident 中且未被添加过，则将该需求点添加到对应的 incident 中。通过遍历 incident 列表来找到符合条件的 incident，并将需求点添加到 center_added_point 中的对应位置。具体代码实现如图 5.12 所示。

```
for demand in total_demand:
    if random.random() < cal_possi(demand[2]):
        for i, incident_1 in enumerate(incident):
            if demand[3] in incident_1 and demand[3] not in added:
                center_added_point[i].append(demand[3])
```

图 5.12 根据概率添加需求列表代码实现

5.3 结果可视化

本节主要介绍结果可视化代码实现，包括节点添加图和结果路线图两个部分。

1. 节点添加图

首先需要绘制一张图，展示哪些节点被添加了，哪些节点没有被添加，并且通过不同的颜色和标记来区分这两类节点。使用 matplotlib.pyplot 创建一个带有子图的图形对象 fig 和轴对象 ax。设置绘图区域的 x 轴和 y 轴范围，将轴的长宽比例设置为相等，并允许调整框架，以确保图形不失真。遍历所有节点 points。如果节点索引 i 在 added 列表中，则使用绿色标记绘制该点；否则，使用红色圆圈标记绘制该点。

通过遍历所有节点，根据节点是否已被添加，用不同的颜色和标记将它们绘制在一张二维平面图上。被添加的节点用绿色标记，而未被添加的节点用红色圆圈标记。通过这种可视化方式，可以直观地看到哪些节点被选择添加，哪些节点没有被选择。具体代码实现如图 5.13 所示。

```
opt = 0
fig, ax = plt.subplots()
ax.set_xlim(0, MAP_SIZE)
ax.set_ylim(0, MAP_SIZE)
ax.set_aspect('equal', adjustable='box')
for i, point in enumerate(points):
    if i in added:
        ax.plot(point[0], point[1], 'gx')
    else:
        ax.plot(point[0], point[1], 'ro')
```

图 5.13 节点添加图代码实现

2. 结果路线图

接着，根据一些条件筛选和处理需求数据，并将符合条件的需求数据与相关路线绘制在图上。首先，创建一个空列表 demand_after_deleted，用于存储删除后的需求数据。然后，遍历所有中心点，获取当前中心点的已添加点列表 added_point，用于筛选需求数据，创建一个空列表 selected_demands，用于存储符合条件的需求数据，遍历所有需求数据，如果需求数据中记录的点索引在 added_point 中，则将该需求数据添加到 selected_demands 中。接着，使用列表推导式，从原始需求数据中移除已选中的需求数据，调用函数 get_route()，根据当前中心点、选中的需求数据和最大距离计算出相关路线列表 route_list 和最优中心点 opt_center，将最优中心点的值累加到变量 opt 上，并且创建一个以当前中心点为圆心、半径为 10 的蓝色圆环对象 circle，将圆环对象添加到图形对象 ax 上。

随后，遍历所有路线数据，创建空列表 path_x 和 path_y，用于存储路径各点的 x 坐标和 y 坐标，并添加当前中心点的坐标作为路径的起点。遍历路线中除去起点和终点的节点索引 n，添加对应需求数据的点坐标作为路径的中间点，再次添加当前中心点的坐标作为路径的终点。最后，使用 plt.plot() 将路径绘制在图上，使用正方形标记表示路径节点，并设置线宽为 0.5，标记大小为 5。

根据一些条件筛选和处理需求数据，并计算出相关路线。每个中心点对应一个圆环，且根据最优中心点的计算结果进行颜色填充。每条路线由一系列节点组成，通过绘制线段和节点标记，将路线展示在图上，并且也会输出最优路径距离。具体代码实现如图 5.14 所示。

```
    demand_after_deleted = []
    for i, center in enumerate(centers):
        added_point = center_added_point[i]
        selected_demands = []
        for demand in total_demand:
            if demand[3] in added_point:
                selected_demands.append(demand)
        total_demand = [demand for demand in total_demand if demand not in selected_demands]
        route_list, opt_center = get_route(center, selected_demands, max_distance)
        opt = opt + opt_center
        circle = plt.Circle(center, 10, color='blue', fill=False)
        ax.add_artist(circle)
        for route in route_list:
            path_x = []
            path_y = []
            path_x.append(center[0])
            path_y.append(center[1])
            for n in route[2:-1]:
                path_x.append(points[selected_demands[n-1][3]][0])
                path_y.append(points[selected_demands[n-1][3]][1])
            path_x.append(center[0])
            path_y.append(center[1])
            plt.plot(path_x, path_y, linewidth=0.5, marker='s', ms=5)

    plt.show()
    print("end")
    print("最优路径距离:", opt)
```

图 5.14 结果路线图及输出代码实现

6 实验结果

本章主要承接第五章实验设计与实现的实验结果，主要包括不含优先级的路径规划、包含优先级的路径规划、不同参数实验结果和实验结果总结四个部分，下面进行具体介绍。

6.1 不含优先级的路径规划

在不考虑随机生成订单的情况下，即在没有考虑优先级的情况下，本问题可以用 VRPTW 算法来求解，以得到最短路径。Gurahipy 求解器求解线性规划问题，通过建立线性规划目标以及约束条件，返回最优静态解。以单图为例，创建半径为 10 的配送区域，每个配送内有 8 个卸货点，根据题目要求建立订单及无人机续航等约束，进行求解，尽可能无人机总路程最短，可得输出如图 6.1 所示。其中，无人机可以同一次派送的订单标注为相同颜色。

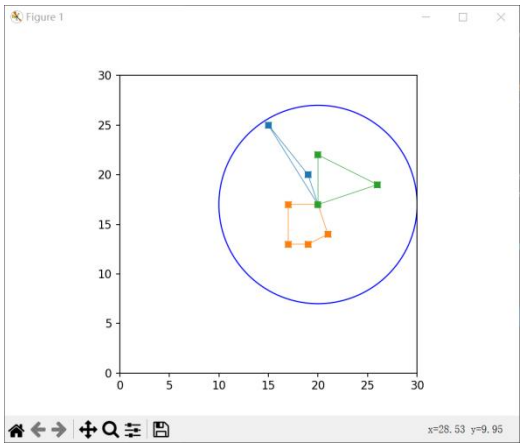


图 6.1 不含优先级的路径规划实验结果图

使用 VRPTW 算法的命令行输出，如图 6.2 所示。从算法输出可以看出，在容忍度参数设置为 10 的-4 次方时，经由 0.01 秒算法迭代了 264 次，中间表格中 Gap 表示当前解与最优解的差距百分比。

6.2 包含优先级的路径规划

考虑优先级时，以路径距离以及紧急程度进一步限制约束，根据相关情况赋予概率，使得无人机优先情况紧急及距离较近的订单，算法实现结果如图 6.3 所示。左图为原始生成图，右图为包含优先级的路径规划实验结果图。

```
Presolve time: 0.07s
Presolved: 985 rows, 1190 columns, 7070 nonzeros
Variable types: 175 continuous, 1015 integer (1015 binary)

Root relaxation: objective 2.315298e+01, 264 iterations, 0.01 seconds (0.01 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   23.15298    0    8   33.93379   23.15298  31.8%     -    0s
H    0     0                      30.1958330   23.15298  23.3%     -    0s
H    0     0                      30.0436786   23.15298  22.9%     -    0s
H    0     0                      29.7193617   23.15298  22.1%     -    0s
     0     0   23.15298    0    8   29.71936   23.15298  22.1%     -    0s
H    0     0                      29.2991738   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     0   23.15298    0    8   29.29917   23.15298  21.0%     -    0s
     0     2   24.88771    0    8   29.29917   24.88771  15.1%     -    0s

Cutting planes:
  Implied bound: 17
  MIR: 4
  RLT: 30

Explored 21 nodes (1139 simplex iterations) in 0.32 seconds (0.20 work units)
Thread count was 12 (of 12 available processors)

Solution count 5: 29.2992 29.7194 30.0437 ... 33.9338

Optimal solution found (tolerance 1.00e-04)
Best objective 2.929917378445e+01, best bound 2.929917378445e+01, gap 0.0000%
```

图 6.2 不含优先级的路径规划命令行输出

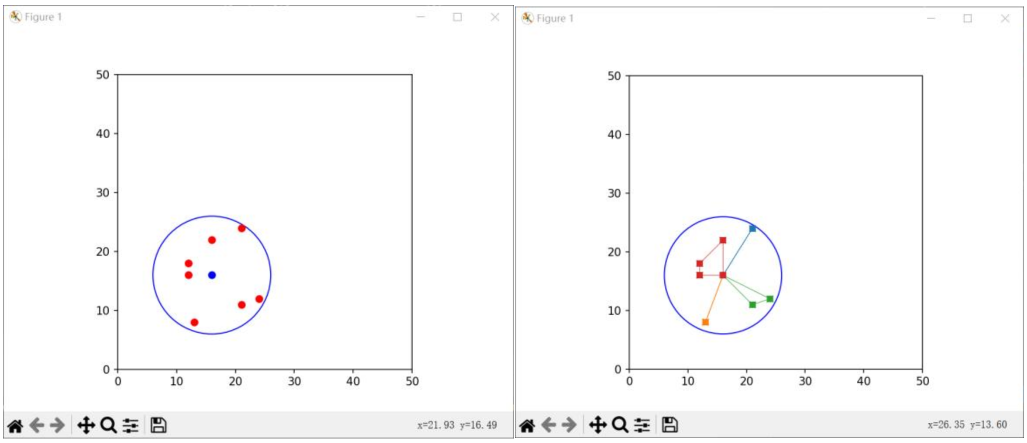


图 6.3 包含优先级的路径规划实验结果对比图

使用 VRPTW 算法的命令行输出，如图 6.4 所示。从算法输出可以看出，在容忍度参数设置为 10 的-4 次方时，经由算法迭代了 80 次，中间表格中 Gap 表示当前解与最优解的差距百分比。同时，算法也输出了最优路径距离的数值，为 60.20。

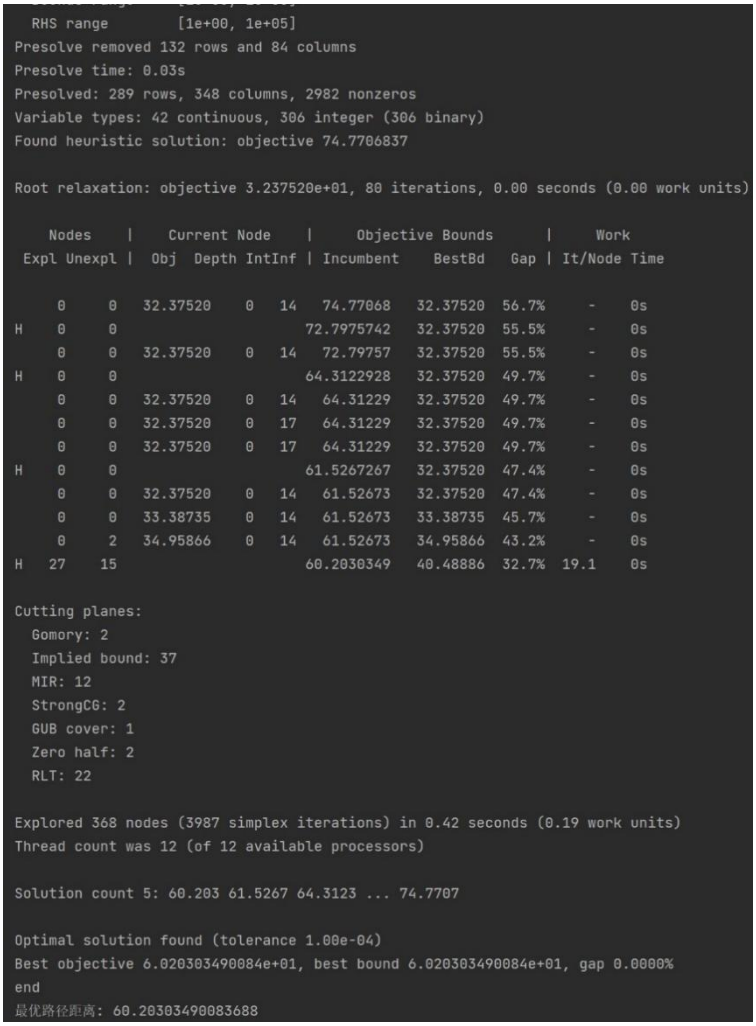


图 6.4 包含优先级的路径规划命令行输出

6.3 不同参数实验结果

调整生成配送中心的数量分别为 4、5、6、7 进行对比实验，输出原始配送图及前四轮迭代结果，并对实验结果进行分析。

1. 四个配送中心实验结果

四个配送中心的原始配送图，如图 6.5 所示，从图中可以看出四个配送区域具有重叠部分。

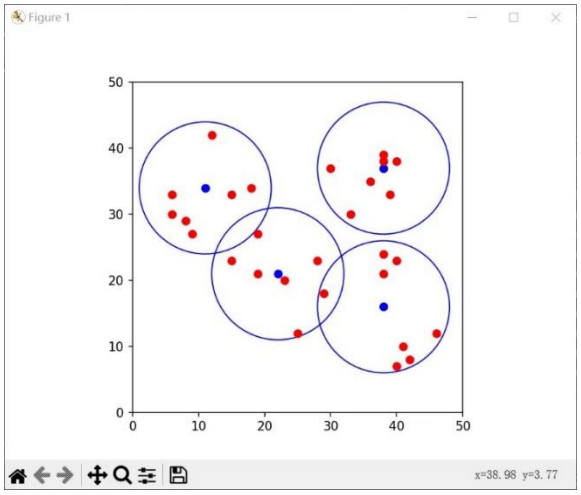


图 6.5 四个配送中心原始配送图

在该配送图的条件下，进行四次迭代，命令行输出及相应路径可视化图如图 6.6、6.7、6.8、6.9 所示。

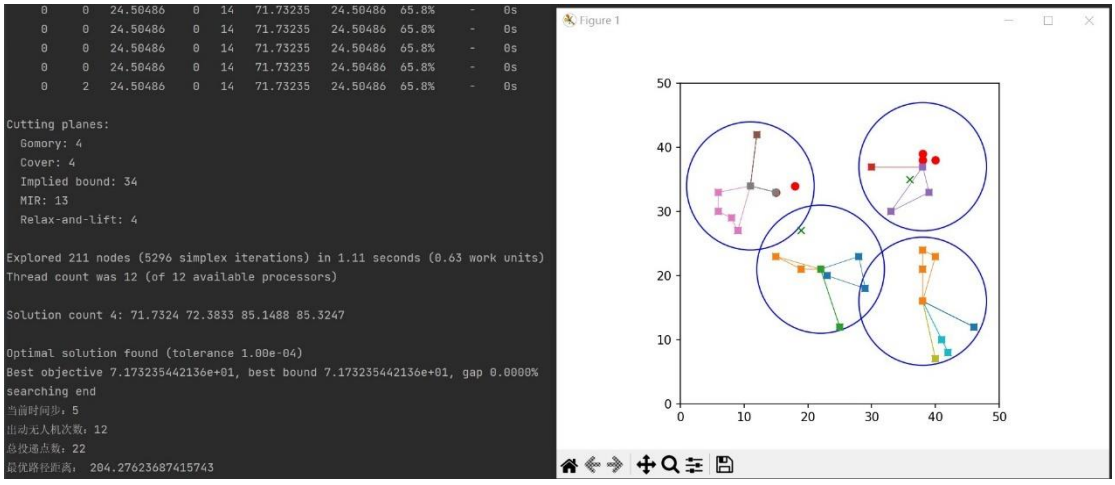


图 6.6 四个配送中心第一次迭代实验结果

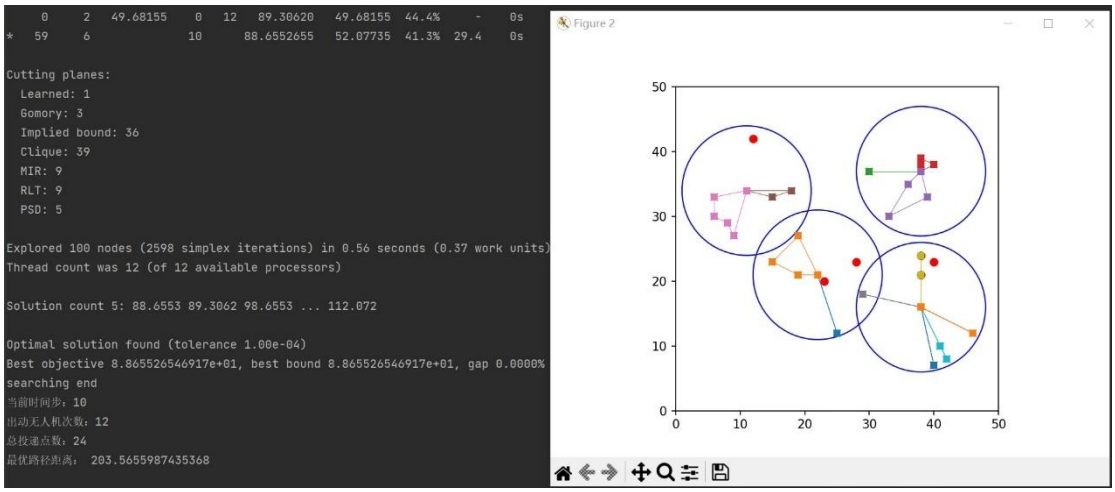


图 6.7 四个配送中心第二次迭代实验结果

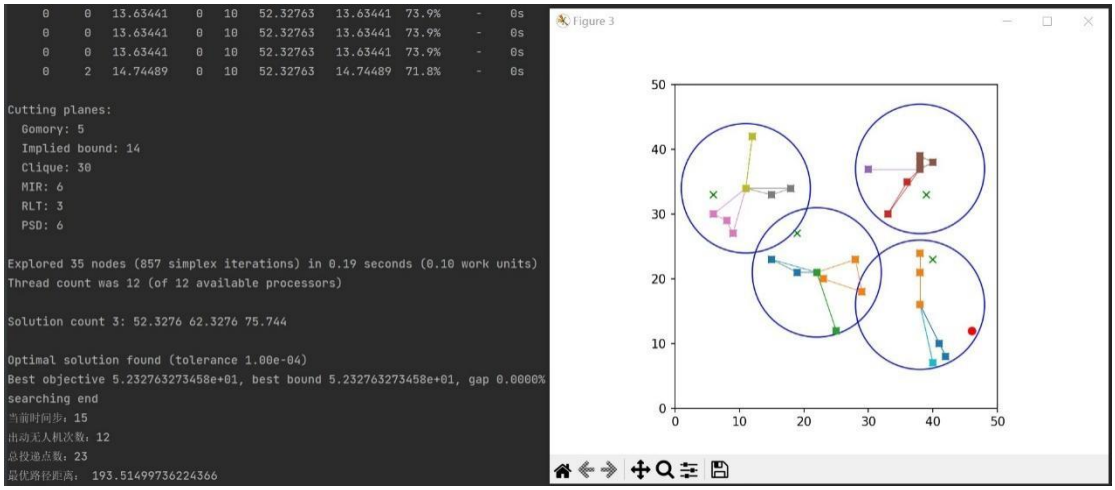


图 6.8 四个配送中心第三次迭代实验结果

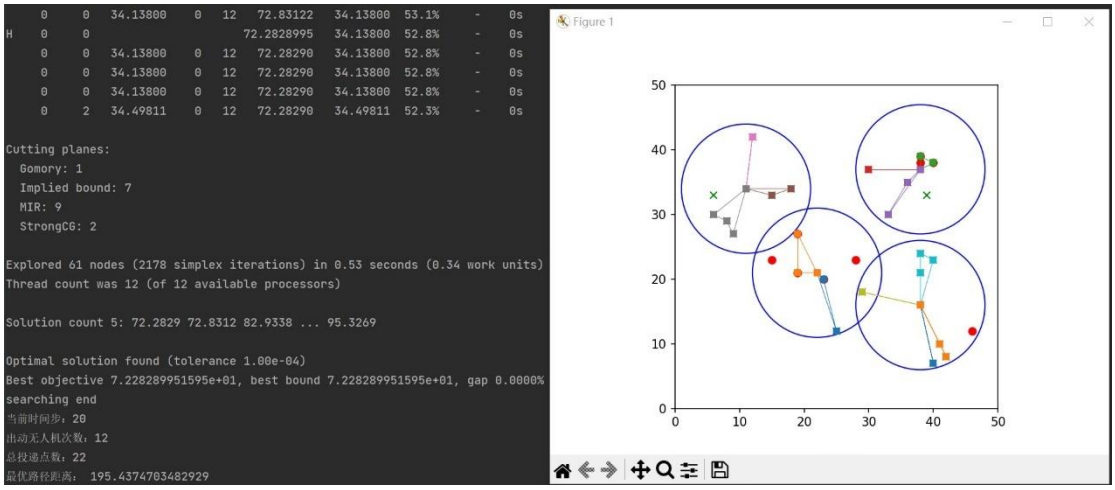


图 6.9 四个配送中心第四次迭代实验结果

根据上述实验结果，每一次迭代后生成新的订单，时间步长增加，重新规划出动无人机次数和投递点，生成最优路径距离。圆圈内红点表示未产生新订单需求的卸货点，绿色叉表示该卸货点订单由于距离较远或优先级不高而稍后再送。在第四次迭代新生成订单后，实验结果显示出动无人机次数为 12，覆盖了 22 个投递点，当前最优路径距离为 195.44。

2. 五个配送中心实验结果

五个配送中心的原始配送图，如图 6.10 所示，从图中可以看出五个配送区域具有重叠部分。

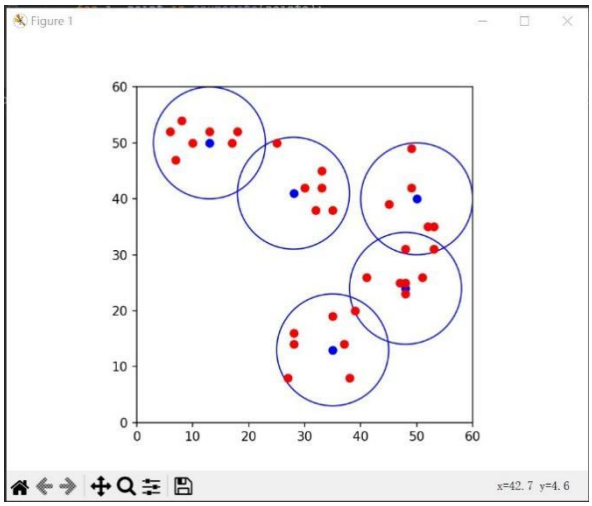


图 6.10 五个配送中心原始配送图

在该配送图的条件下，进行四次迭代，命令行输出及相应路径可视化图如图 6.11、6.12、6.13、6.14 所示。

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   16.14214    0    4   18.56022   16.14214  13.0%     -    0s
H    0     0                       17.3511777   16.14214  6.97%     -    0s
H    0     0                       16.6868409   16.14214  3.26%     -    0s
     0     0   16.14214    0    4   16.68684   16.14214  3.26%     -    0s

Cutting planes:
  Clique: 1
  RLT: 2

Explored 1 nodes (159 simplex iterations) in 0.07 seconds (0.03 work units)
Thread count was 12 (of 12 available processors)

Solution count 3: 16.6868 17.3512 18.5602

Optimal solution found (tolerance 1.00e-04)
Best objective 1.668684091773e+01, best bound 1.668684091773e+01, gap 0.0000%
searching end
当前时间步：5
出动无人机次数：14
总投递点数：27
最优路径距离： 225.7613811069948
```

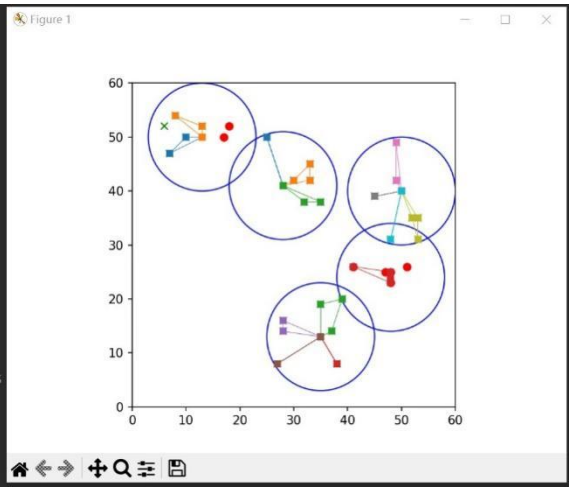


图 6.11 五个配送中心第一次迭代实验结果

```
     0     0   20.65081    0    8   23.32385   20.65081  11.5%     -    0s
     0     0   20.73622    0    8   23.32385   20.73622  11.1%     -    0s
     0     0   20.80175    0    8   23.32385   20.80175  10.8%     -    0s
     0     0   20.80175    0    8   23.32385   20.80175  10.8%     -    0s
     0     2   21.84296    0    8   23.32385   21.84296  6.35%     -    0s

Cutting planes:
  Gomory: 1
  Implied bound: 12
  MIR: 9
  RLT: 16
  PSD: 3

Explored 14 nodes (695 simplex iterations) in 0.17 seconds (0.09 work units)
Thread count was 12 (of 12 available processors)

Solution count 3: 23.3239 23.6253 23.9882

Optimal solution found (tolerance 1.00e-04)
Best objective 2.332385174946e+01, best bound 2.332385174946e+01, gap 0.0000%
searching end
当前时间步：10
出动无人机次数：14
总投递点数：24
最优路径距离： 211.72692488994227
```

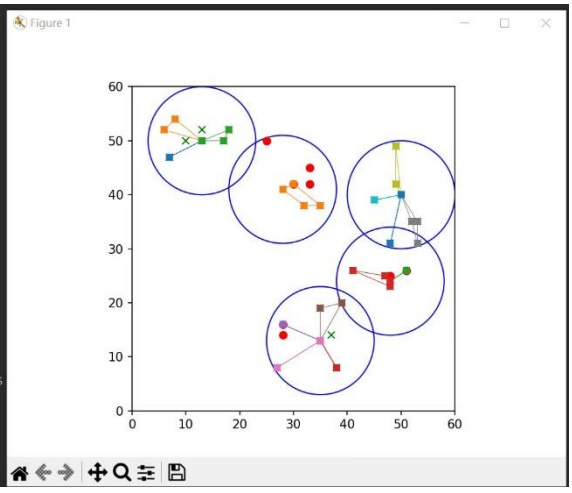


图 6.12 五个配送中心第二次迭代实验结果

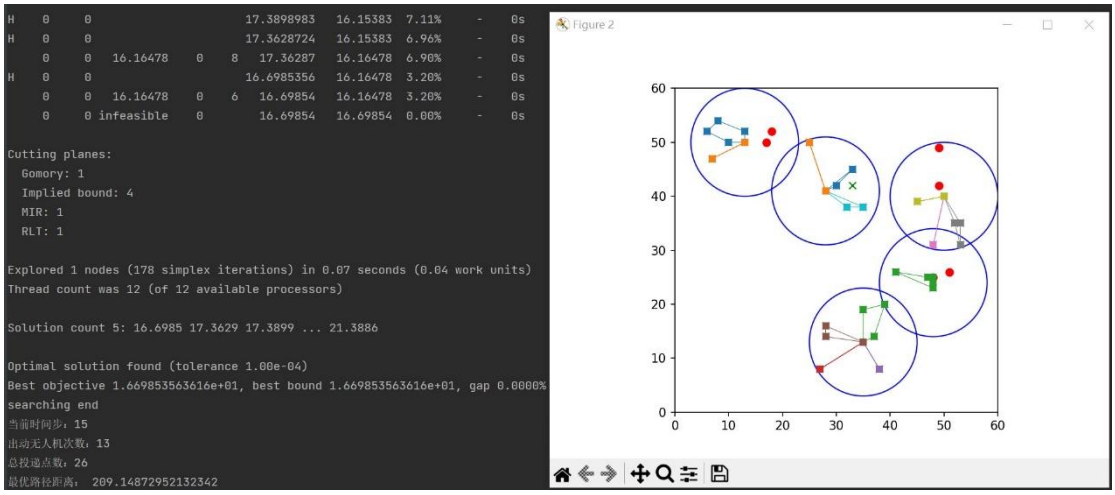


图 6.13 五个配送中心第三次迭代实验结果

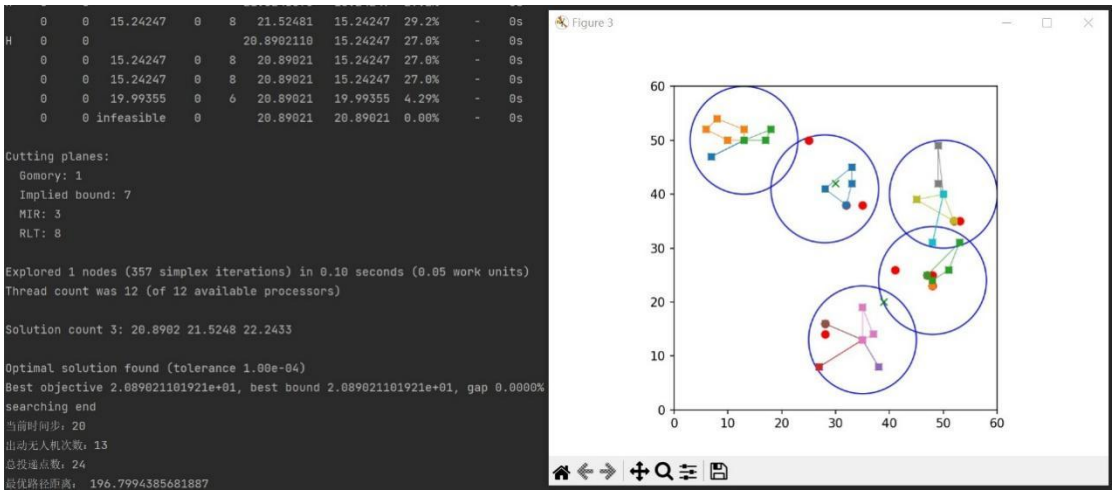


图 6.14 五个配送中心第四次迭代实验结果

根据上述实验结果，每一次迭代后生成新的订单，时间步长增加，重新规划出动无人机次数和投递点，生成最优路径距离。圆圈内红点表示未产生新订单需求的卸货点，绿色叉表示该卸货点订单由于距离较远或优先级不高而稍后再送。在第四次迭代新生成订单后，实验结果显示出动无人机次数为 13，覆盖了 24 个投递点，当前最优路径距离为 196.80。

3. 六个配送中心实验结果

六个配送中心的原始配送图，如图 6.15 所示，从图中可以看出六个配送区域具有重叠部分。

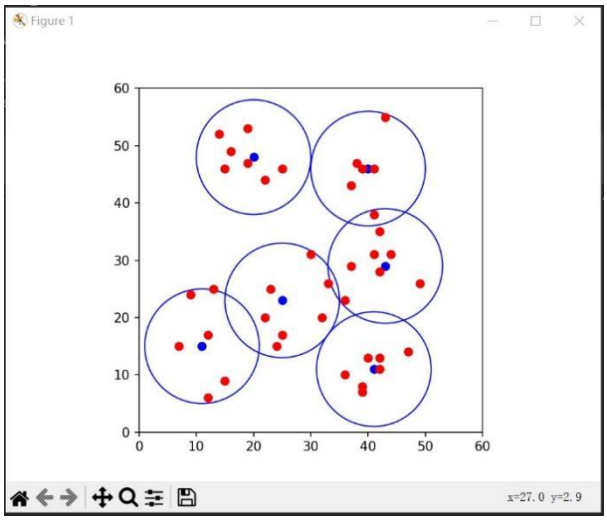


图 6.15 六个配送中心原始配送图

在该配送图的条件下，进行四次迭代，命令行输出及相应路径可视化图如图 6.16、6.17、6.18、6.19 所示。

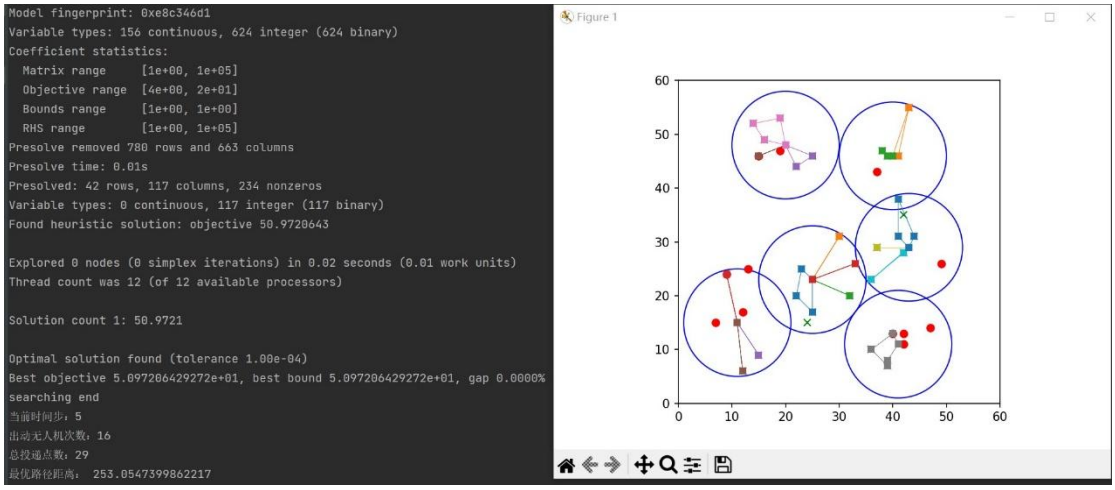


图 6.16 六个配送中心第一次迭代实验结果

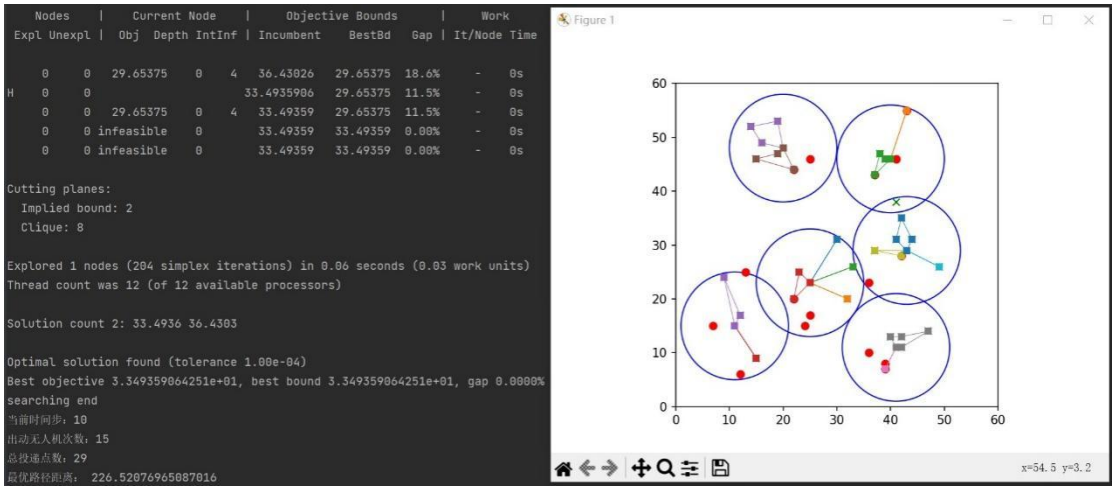


图 6.17 六个配送中心第二次迭代实验结果

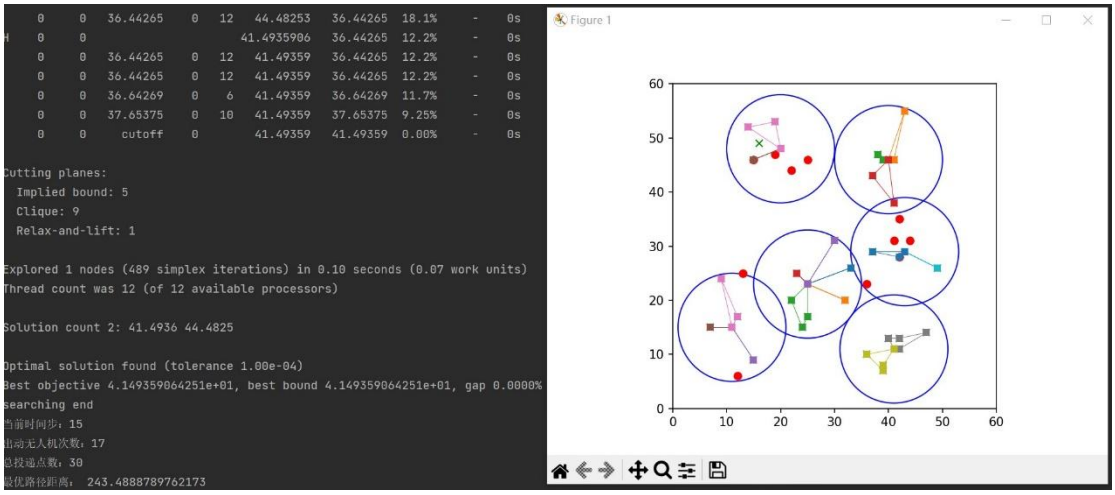


图 6.18 六个配送中心第三次迭代实验结果

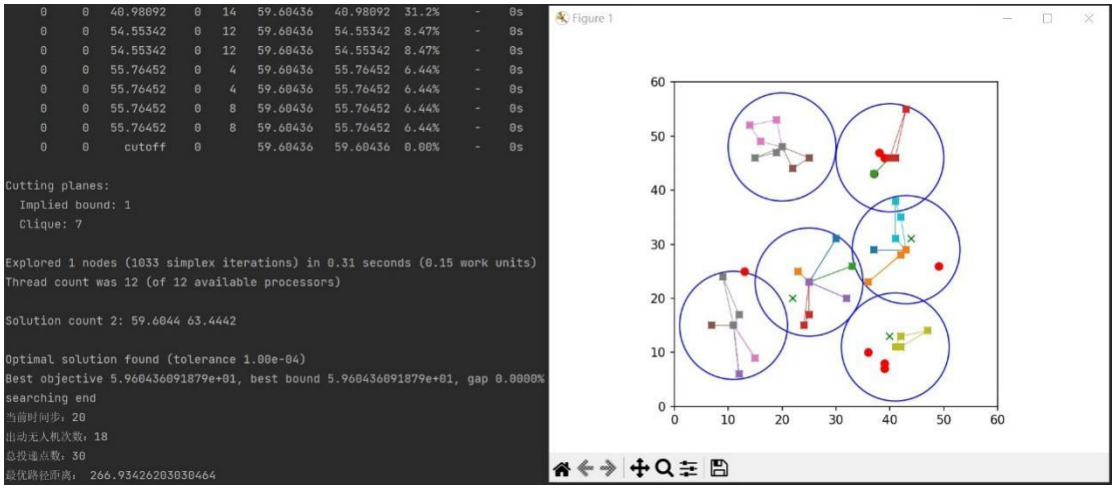


图 6.19 六个配送中心第四次迭代实验结果

根据上述实验结果，每一次迭代后生成新的订单，时间步长增加，重新规划出动无人机次数和投递点，生成最优路径距离。圆圈内红点表示未产生新订单需求的卸货点，绿色叉表示该卸货点订单由于距离较远或优先级不高而稍后再送。在第四次迭代新生成订单后，实验结果显示出动无人机次数为 18，覆盖了 30 个投递点，当前最优路径距离为 266.93。

4. 七个配送中心实验结果

七个配送中心的原始配送图，如图 6.20 所示，从图中可以看出七个配送区域具有重叠部分。在该配送图的条件下，进行四次迭代，命令行输出及相应路径可视化图如图 6.21、6.22、6.23、6.24 所示。

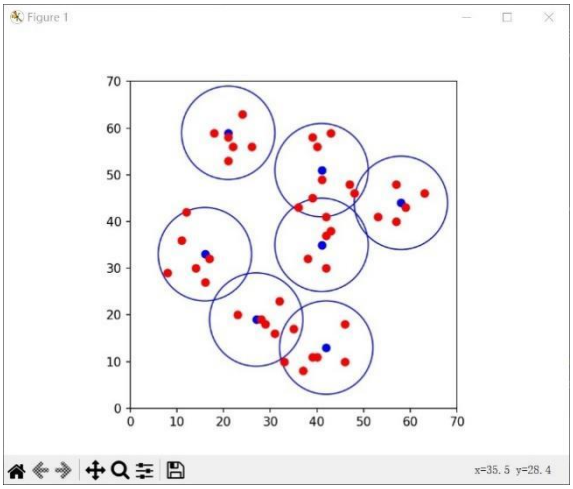


图 6.20 七个配送中心原始配送图

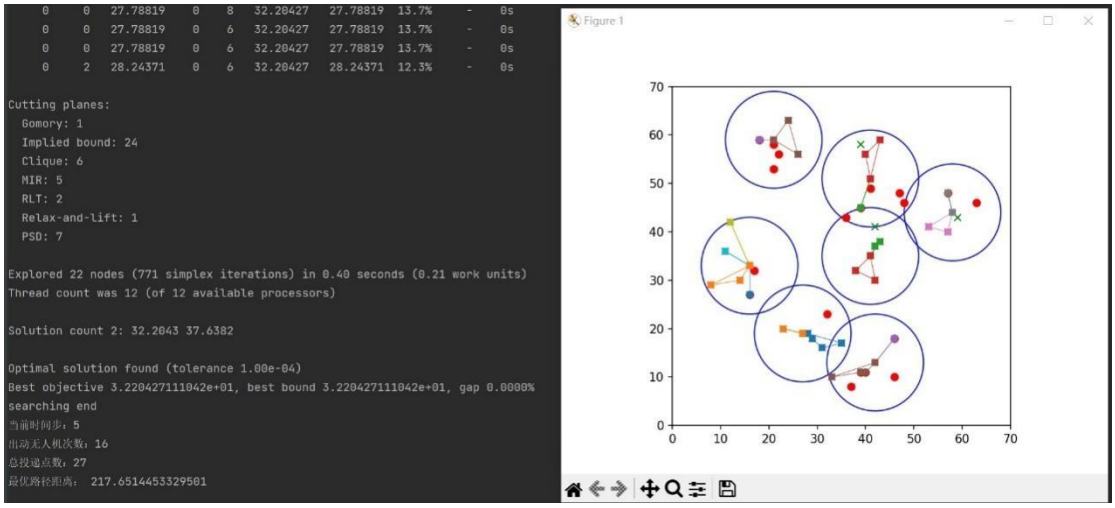


图 6.21 七个配送中心第一次迭代实验结果

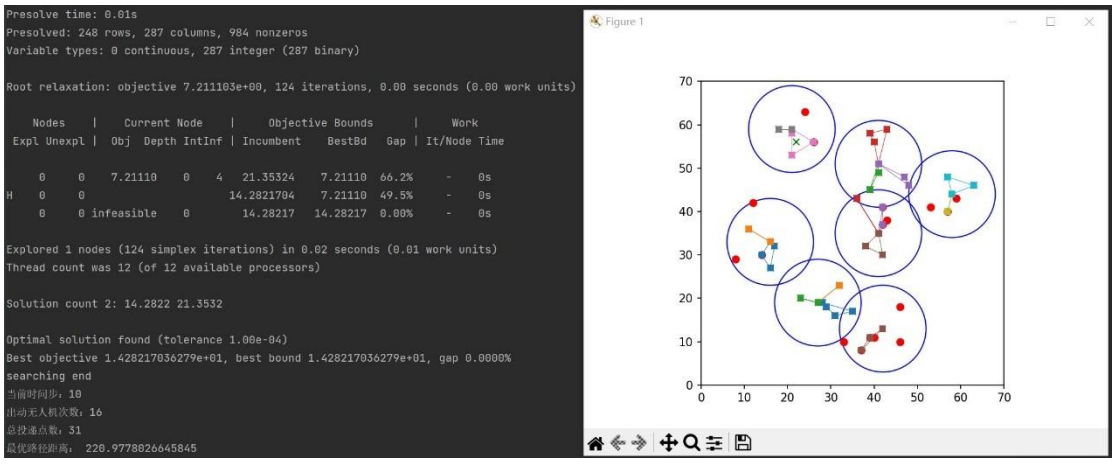


图 6.22 七个配送中心第二次迭代实验结果

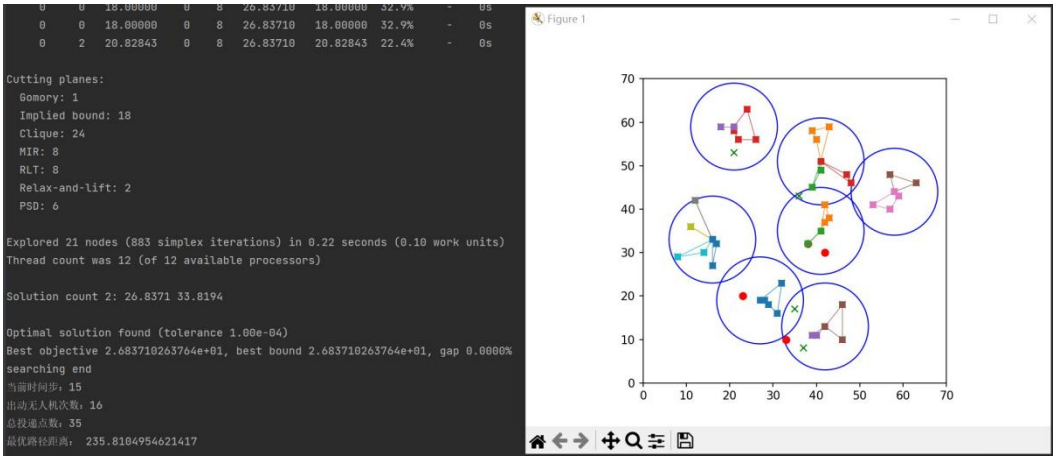


图 6.23 七个配送中心第三次迭代实验结果

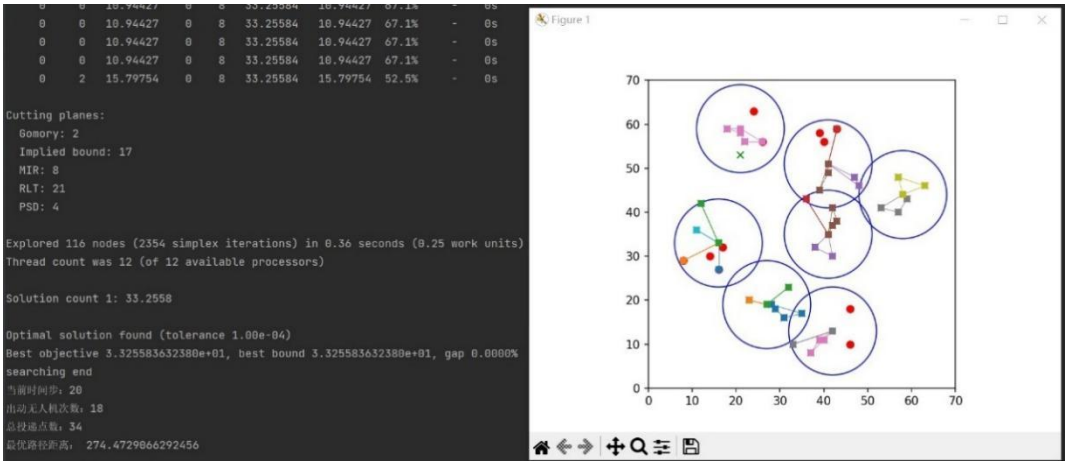


图 6.24 七个配送中心第四次迭代实验结果

根据上述实验结果，每一次迭代后生成新的订单，时间步长增加，重新规划出动无人机次数和投递点，生成最优路径距离。圆圈内红点表示未产生新订单需求的卸货点，绿色叉表示该卸货点订单由于距离较远或优先级不高而稍后再送。在第四次迭代新生成订单后，实验结果显示出动无人机次数为 18，覆盖了 34 个投递点，当前最优路径距离为 274.47。

6.4 实验结果总结

通过实验可以看出，当配送中心变多时，配送区域更加密集，但是算法依然能得到较好的输出结果，即无人机路径规划策略，证明了算法的有效性。

通过对一段时间内的订单生成和无人机调度的模拟，实验得到了以下结果：

- 无人机成功按优先级完成了所有订单的配送。
- 总配送路径长度得到显著优化，确保了资源的高效利用。
- 系统能够灵活调度策略，合理推迟低优先级订单，提高整体效率。

7 实验总结

本实验对无人机在特定区域内的路径规划问题进行了算法设计，该区域包括多个配送中心和卸货点。每个配送中心可以无限供应商品和派遣无人机，而卸货点则生成不同优先级的订单。在此实验中，我设计并实现了一种算法，用于规划无人机在给定区域内的路径，以最小化总配送路径长度，同时满足订单的优先级要求。其中，借助带时间窗的车辆路径规划问题（VRPTW）进行数学建模，用 Gurobi 求解器进行求解，然后设计了基于距离和基于时间的两个启发式算法，一旦存在某一卸货点满足其中一个启发式算法的条件，该点即被选中。

本实验设计的算法有效地解决了无人机配送路径规划问题，在约束条件下最小化了总路径长度，并保证了订单优先级的要求。实验结果表明，即使在配送区域密集的情况下，但是算法依然能得到较好的输出结果，能得出较优的无人机路径规划策略，证明了算法的有效性。

在实验中，我也遇到了一些问题，在最开始时进行建模，数学模型过于复杂，后来找到时间窗的车辆路径规划问题，才得以有了思路。在方法的选择也进行了利弊权衡，启发式函数的确定也是结合实际应用分析，在距离聚类上进行重写改进。虽然在本次实验之前也没有具体接触过应用问题建模的算法实现，但是通过深入学习相关背景知识，我能够自主解决其中存在的问题并且完成实验。本次实验使我增强了算法设计和实现能力，同时也提高了我的问题解决能力和编程能力。总之，本次实验是一个非常有意义和有价值的学习过程。