仰恩大学校内数学建模竞赛论文

基于组合优化与载荷能耗的无人机送货路径规划模型

论文题目	基于组合优化与载荷能耗的无人机送货路径规划模型
参赛队员 1	
参赛队员 2	
参赛队员 3	
队长联系电话	
参赛队号	

仰恩大学数学建模竞赛指导团队 2026 制

摘要

基于组合优化与载荷能耗的无人机送货路径规划模型

本文针对城市无人机送货规划问题，提出基于组合优化与载荷能耗的系统方案。

针对问题一：本文利用 Haversine 公式构造球面距离矩阵，以访问序列为决策变量建立旅行商模型，并采用多起点最近邻、2-opt、Or-opt 和随机重启求解。结果显示，**最短总航程为 136.7200 km**，航段距离均值为 5.4688 km。

针对问题二：本文建立容量约束车辆路径模型，采用方位角扫描分组、Clarke-Wright 节约思想、DP+Bitmask 子问题验证和跨组节点交换。结果显示，**四架无人机的总航程为 211.5515 km**，较初始扫描方案降低 8.79%，各路线分别服务 7、7、7、3 个配送点。

针对问题三：本文构建基础能耗与剩余载荷线性叠加的燃料函数，比较 2、3、4、6、8、12 架无人机的等装载方案，并进行载荷系数敏感性检验。结果显示，**2 架无人机时总燃料消耗最低，为 250.6235**，对应总航程 174.1724 km；若强调运营冗余，可将 4 架等载方案作为稳健备选。

本文模型兼顾路径结果、燃料解释和工程备选，可为题设条件下的无人机配送调度提供参考。

关键词：无人机配送；旅行商问题；车辆路径规划；Or-opt 算法；载荷能耗；灵敏度分析

一、问题重述

本题给出一个仓库和二十四个配送目的地的经纬度坐标，要求围绕无人机送货场景完成三类路径规划任务。三个问题共享同一组空间坐标，但约束和优化目标逐步增强。

随着电子商务、即时零售和校园配送需求快速增长，短途小件物流对配送时效、末端成本和服务弹性的要求不断提高。无人机具备直线飞行、响应速度快、可批量调度等优势，在园区、城市近距离和应急补给场景中具有较强应用潜力 [1, 2]。因此，如何根据收货地址合理设计无人机航线，并在多架无人机共同工作时实现任务分配，是本题需要解决的核心问题。

问题一要求安排一架无人机从仓库出发，依次访问全部二十四个目的地后返回仓库，使闭合配送路线的总航程最短。

问题二要求使用多架无人机完成全部配送任务，且每架无人机最多访问 7 个目的地。在该容量约束下，需要同时确定各无人机负责的目的地集合和访问顺序，使所有无人机总航程最短。

问题三进一步考虑包裹装载量对燃料消耗的影响，要求在不考虑航程限制、每架无人机装载量相等的条件下，自动分配包裹并规划路线，使总燃料消耗最少。

二、问题分析

三个问题均需先将经纬度坐标转化为空间距离，因此本文以 Haversine 球面距离矩阵作为统一输入，再根据不同目标建立组合优化模型。

该类路径规划问题本质上属于组合优化问题，具有离散决策、路径顺序相关和约束耦合等特点。若直接枚举所有方案，计算复杂度随配送点数量阶乘级增长，因此需要在数学模型清晰的前提下选择可解释且高效的启发式算法 [3, 4, 5, 6]。本文在保证结果可复现的基础上，使用经纬度球面距离建立统一数据底座，并将每问的求解结果以路线图、统计表和敏感性曲线呈现。

问题一是典型旅行商闭合路径问题，核心决策是二十四个目的地的访问顺序。本文采用多起点最近邻构造初始路线，并结合 2-opt、Or-opt 和随机重启改进路径质量。

问题二是在问题一基础上增加多路线分组和容量限制，可视为容量约束车辆路径问题。本文先用方位角扫描形成空间连续的初始分组，再引入 Clarke-Wright 节约思想、组内路径优化和跨组节点交换降低总航程。

问题三把距离目标扩展为载荷相关能耗目标。由于无人机满载出发、逐点卸货、返航空载，访问顺序会影响高载荷飞行距离。本文以单位包裹重量、基础能耗系数和载荷能耗系数构造燃料函数，比较不同无人机数量下的等装载方案，并用敏感性分析检验 2 架燃料最优结论的稳定性。

三、模型假设

为简化问题，本文做出以下假设：

- **假设 1**：无人机在两点之间按近似球面最短距离飞行，飞行过程中不受禁飞区、风速、天气和高度差影响。
- **假设 2**：每个目的地仅有一个包裹且需求必须被满足，卸货时间相对飞行时间较短，在路径优化中不单独计入。
- **假设 3**：问题三中各包裹重量相同，单位距离燃料消耗由基础能耗和剩余载荷线性叠加构成。

四、符号说明

表 2　主要符号说明

符号	含义
V	全部节点集合，其中 0 表示仓库，1 至 24 表示配送点
C	配送点集合，满足 $C = V \setminus \{0\}$
d_{ij}	节点 i 到节点 j 之间的球面距离，单位为 km
π	单架无人机访问配送点的排列序列
R_r	第 r 架无人机的闭合配送路线
q_r	第 r 架无人机服务的配送点数量
m	参与配送的无人机数量
$L(R_r)$	路线 R_r 的总航程
$F(R_r)$	路线 R_r 的燃料消耗
α	空载无人机单位距离基础能耗系数
β	单位载荷带来的额外单位距离能耗系数
w_0	单位包裹重量，本文采用归一化取值
w_{rk}	第 r 条路线第 k 段飞行前的剩余载荷

五、模型建立

5.1 问题一的模型建立

问题一可抽象为单机闭合旅行商问题。设仓库编号为 0，二十四个配送点构成集合 $C = \{1, 2, \ldots, 24\}$，配送点访问排列为 $\pi = (\pi_1, \pi_2, \ldots, \pi_{24})$。两点间飞行距离采用 Haversine 球面距离计算：

$$d_{ij} = 2R \arcsin \sqrt{\sin^2 \frac{\varphi_i - \varphi_j}{2} + \cos \varphi_i \cos \varphi_j \sin^2 \frac{\lambda_i - \lambda_j}{2}}, \tag{1}$$

其中，R 为地球平均半径，φ_i, λ_i 分别表示节点 i 的纬度和经度弧度值。

无人机从仓库出发并最终返回仓库，因此给定访问顺序下的闭合路径总航程为

$$L(\pi) = d_{0,\pi_1} + \sum_{k=1}^{23} d_{\pi_k, \pi_{k+1}} + d_{\pi_{24}, 0}. \tag{2}$$

问题一的优化目标为

$$\min_{\pi} L(\pi), \quad \pi\text{为集合}C\text{的一个排列}. \tag{3}$$

该模型保证全部配送点均被访问一次，且路径首尾均为仓库。

5.2 问题二的模型建立

问题二在问题一基础上增加多架无人机和单机服务点数量限制，可表示为容量约束车辆路径问题。设共有 m 架无人机，第 r 架无人机路线为

$$R_r = (0, v_{r1}, v_{r2}, \ldots, v_{rq_r}, 0), \tag{4}$$

其中 q_r 表示第 r 架无人机服务的目的地数量。单条路线长度为

$$L(R_r) = d_{0,v_{r1}} + \sum_{k=1}^{q_r - 1} d_{v_{rk}, v_{r,k+1}} + d_{v_{rq_r}, 0}. \tag{5}$$

多机总航程目标为

$$\min L_{total} = \sum_{r=1}^{m} L(R_r). \tag{6}$$

每个配送点必须且只能被一架无人机服务，同时每架无人机最多访问 7 个目的地：

$$\bigcup_{r=1}^{m} \{v_{r1}, \ldots, v_{rq_r}\} = C, \quad R_a \cap R_b = \varnothing \ (a \neq b), \quad 1 \leq q_r \leq 7. \tag{7}$$

由于共有 24 个配送点，满足容量约束时至少需要 $\lceil 24/7 \rceil = 4$ 架无人机。

5.3 问题三的模型建立

问题三将距离目标扩展为载荷相关燃料目标。设每个包裹重量相同，记为 w_0；第 r 架无人机服务 q_r 个目的地，则出发时载荷为

$$W_{r0} = q_r w_0. \tag{8}$$

无人机每到达一个目的地卸下一个包裹，故第 k 段飞行前剩余载荷为

$$w_{rk} = (q_r - k + 1)w_0, \quad k = 1, 2, \ldots, q_r, \tag{9}$$

返航阶段剩余载荷为 0。

设单位距离基础燃料系数为 α，单位距离载荷燃料系数为 β。其中，α 可理解为空载无人机单位距离基础能耗，β 表示单位载荷带来的额外单位距离能耗。本文采用无量纲归一化参数进行方案相对比较，不代表某一具体机型的真实燃料值。第 r 条路线燃料消耗为

$$F(R_r) = \sum_{k=0}^{q_r} d_{v_{rk}, v_{r,k+1}} \left(\alpha + \beta w_{rk}\right), \tag{10}$$

其中 $v_{r0} = 0$，$v_{r,q_r+1} = 0$，且返航段 $w_{r,q_r} = 0$。多机燃料目标为

$$\min F_{total} = \sum_{r=1}^{m} F(R_r). \tag{11}$$

题目要求每架无人机装载量相等，因此有

$$q_1 = q_2 = \cdots = q_m = \frac{24}{m}, \tag{12}$$

其中 m 应取 24 的因子。

六、模型求解

6.1 问题一的模型求解

6.1.1 求解准备

针对问题一，本文将其定性为单车闭合旅行商问题，即在仓库节点固定、配送点全部必须访问一次的条件下，寻找总航程最短的访问顺序。由于题目没有给出道路网络和飞行障碍，模型采用经纬度球面距离刻画两点之间的飞行代价，并把仓库作为起点和终点。

首先，读取题中二十五个节点的经纬度并统一编号。接着，利用 Haversine 公式计算任意两点间距离，形成完整距离矩阵。然后，以最近邻规则构造初始闭合路径，并用 2-opt 算法不断交换路径中的两段边。最后，比较多起点搜索得到的候选路径，选择总航程最短者作为问题一结果。具体分析流程如图 1所示。

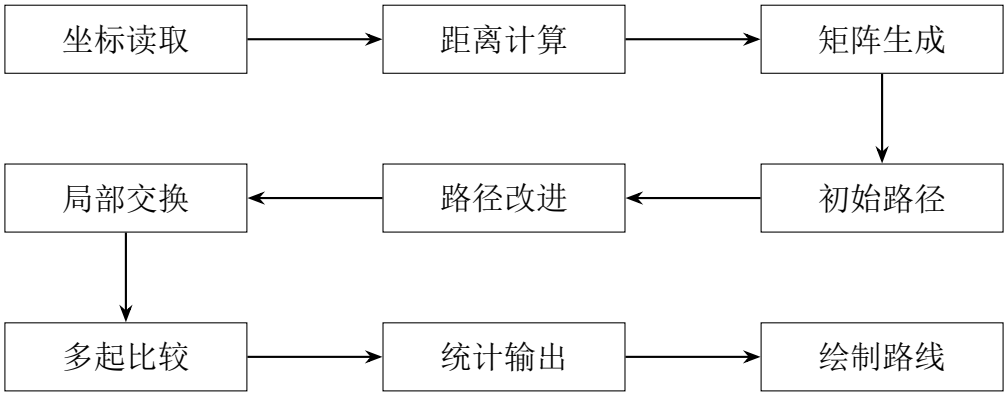

图 1　问题 1 分析流程图

题目给出的原始数据为仓库和配送点的经纬度坐标，数据字段包括地点编号、经度和纬度。由于所有坐标均在同一城市范围内，数值类型一致且无缺失记录，预处理重点不是删除异常值，而是统一地点编号、区分仓库与配送点，并把角度坐标转换为可用于优化的公里距离矩阵。经过整理后，样本规模为 1 个仓库节点和 24 个配送点，共 25 个节点。

在数据理解与提取阶段，本文保留经纬度用于绘制空间分布图，同时提取所有节点编号用于构建距离矩阵。距离矩阵是一个 25×25 的对称矩阵，对角线为 0，非对角线代表任意两个节点之间的球面距离。由于各配送点距离仓库最小为 4.0240 km，最大为 35.6481 km，说明该算例既包含近距离配送点，也包含较远的城市边缘点。

在预处理步骤中，首先检查经纬度列是否为数值型并统一编号；随后使用球面距离公式将角度差映射为公里距离；最后把坐标表保存为后续问题共用的预处理数据。若记原始经度和纬度为 (λ_i, φ_i)，则角度转弧度后可进行距离计算。对于后续图形展示，若需要比较不同量纲指标，可采用 Min-Max 归一化公式

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}. \tag{13}$$

其中，x 表示原始指标，$x_{\min}$ 和 $x_{\max}$ 分别表示该指标的最小值和最大值，x' 表示归一化后的无量纲结果。

表 3　数据预处理统计量对比

指标	最小值	最大值	均值	标准差
仓库距离	4.0240	35.6481	17.2524	8.6398
非零点距	1.6316	43.0827	17.4382	8.2480

问题一本质上是一个排列组合优化问题，需要在所有配送点访问顺序中寻找最短闭合回路。常用求解算法包括动态规划、整数规划、遗传算法、模拟退火、Lin-Kernighan 启发式和 2-opt 局部搜索等。本节从全局搜索能力、收敛速度、可解释性和实现稳定性四个方面进行评估。

表 4　问题一算法能力对比

算法名称	全局搜索能力	收敛速度	可解释性	稳定性
动态规划	优	差	良	优
整数规划	优	中	良	良
遗传算法	良	中	中	中
2-opt 算法	良	优	优	优

通过上述对比可以看出，2-opt 算法虽然不能严格保证全局最优，但在本题规模下速度快、逻辑清晰、路径改进过程直观，并且可通过多起点策略提高解的质量。因此本文选择最近邻多起点与 2-opt 局部搜索相结合的方法，通过构造、交换、比较和输出四个步骤获得单机最短路径。

6.1.2 算法设计与结果

针对问题一，本节将从距离度量、目标函数和路径搜索三个角度建立单机最短闭合路径模型，最终得到适合无人机配送的旅行商问题表达式，具体过程如下。

步骤 1：距离矩阵的建立。设节点 i 的经度为 λ_i，纬度为 φ_i，由于经纬度以角度给出，首先将其转化为弧度：

$$\lambda_i^r = \frac{\pi \lambda_i}{180}, \quad \varphi_i^r = \frac{\pi \varphi_i}{180}. \tag{14}$$

其中，λ_i^r 和 φ_i^r 分别表示节点 i 经度和纬度的弧度值。两个节点之间的经纬度差为

$$\Delta\lambda_{ij} = \lambda_j^r - \lambda_i^r, \quad \Delta\varphi_{ij} = \varphi_j^r - \varphi_i^r. \tag{15}$$

其中，$\Delta\lambda_{ij}$ 和 $\Delta\varphi_{ij}$ 表示节点 i 到节点 j 的经纬度弧度差。依据 Haversine 公式，辅助量 a_{ij} 可写为

$$a_{ij} = \sin^2\left(\frac{\Delta\varphi_{ij}}{2}\right) + \cos\varphi_i^r \cos\varphi_j^r \sin^2\left(\frac{\Delta\lambda_{ij}}{2}\right). \tag{16}$$

其中，a_{ij} 刻画两点球面夹角的一半正弦平方。进一步得到球面中心角

$$c_{ij} = 2\arcsin\sqrt{a_{ij}}. \tag{17}$$

其中，c_{ij} 表示节点 i 和节点 j 相对于地心的夹角。设地球平均半径为 $R = 6371$ km，则两点距离为

$$d_{ij} = Rc_{ij}. \tag{18}$$

其中，d_{ij} 表示无人机从节点 i 到节点 j 的飞行距离。由距离对称性可得

$$d_{ij} = d_{ji}, \quad d_{ii} = 0. \tag{19}$$

其中，对称关系说明本题在无风和无高度差条件下可视为对称旅行商问题。最终形成距离矩阵

$$D = (d_{ij})_{25\times 25}. \tag{20}$$

其中，D 是后续三个问题共同使用的基础数据。

步骤 2：单机目标函数的建立。设配送点集合为 $C = \{1, 2, \ldots, 24\}$，访问排列为 $\pi = (\pi_1, \pi_2, \ldots, \pi_{24})$。无人机从仓库出发到第一个配送点的距离为

$$L_0 = d_{0,\pi_1}. \tag{21}$$

其中，L_0 表示首段飞行距离。配送点之间的连续访问距离为

$$L_m = \sum_{k=1}^{23} d_{\pi_k,\pi_{k+1}}. \tag{22}$$

其中，L_m 表示中间二十三段配送飞行距离。完成全部配送后返回仓库的距离为

$$L_b = d_{\pi_{24},0}. \tag{23}$$

其中，L_b 表示返航距离。因此单机闭合路径总航程为

$$L(\pi) = L_0 + L_m + L_b. \tag{24}$$

其中，$L(\pi)$ 是给定访问顺序下的总距离。问题一的优化目标为

$$\min_{\pi} L(\pi). \tag{25}$$

其中，优化变量为配送点排列 π。每个配送点必须且只被访问一次，因此有

$$\{\pi_1, \pi_2, \ldots, \pi_{24}\} = C. \tag{26}$$

其中，该约束保证所有目的地均被覆盖。路径首尾必须为仓库，因此完整路线可写为

$$R = (0, \pi_1, \pi_2, \ldots, \pi_{24}, 0). \tag{27}$$

其中，R 表示最终输出的无人机飞行路线。

步骤 3：2-opt 与 Or-opt 路径改进模型。设当前路线中存在两条边 (r_i, r_{i+1}) 和 (r_k, r_{k+1})，若反转中间片段，则原两边距离为

$$B_{old} = d_{r_i, r_{i+1}} + d_{r_k, r_{k+1}}. \tag{28}$$

其中，B_{old} 表示交换前被删除的边长之和。交换后新增两边距离为

$$B_{new} = d_{r_i, r_k} + d_{r_{i+1}, r_{k+1}}. \tag{29}$$

其中，B_{new} 表示交换后新连接的边长之和。若满足

$$\Delta B = B_{new} - B_{old} < 0, \tag{30}$$

其中，ΔB 表示本次交换引起的航程变化，则接受该反转操作。新路线写为

$$R' = (r_0, \ldots, r_i, r_k, r_{k-1}, \ldots, r_{i+1}, r_{k+1}, \ldots, r_{25}). \tag{31}$$

其中，R' 表示反转片段后的候选路线。迭代更新规则为

$$R^{(t+1)} = \begin{cases} R', & L(R') < L(R^{(t)}), \\ R^{(t)}, & L(R') \geq L(R^{(t)}). \end{cases} \tag{32}$$

其中，t 表示迭代次数。终止条件为

$$\min_{1 \leq i < k \leq 24} \Delta B_{ik} \geq 0. \tag{33}$$

其中，该条件说明不存在进一步缩短路径的二边交换。

2-opt 主要消除路径交叉，但其邻域只改变两条边，容易在局部结构上停止。为扩大搜索范围，本文进一步引入 Or-opt 片段迁移算子 [4]。设从路线中截取长度为 ℓ 的连续片段

$$P = (r_i, r_{i+1}, \ldots, r_{i+\ell-1}), \quad \ell \in \{1, 2, 3\}. \tag{34}$$

其中，P 表示待迁移的连续配送点片段。将该片段从原位置移除并插入到位置 j 之后，得到候选路线

$$R'' = \mathrm{Insert}(R \setminus P, P, j). \tag{35}$$

其中，Insert 表示片段插入操作。若候选路线满足

$$\Delta L_{or} = L(R'') - L(R) < 0, \tag{36}$$

则接受该 Or-opt 移动。实际求解中，先对多起点最近邻初始解执行 2-opt，再交替执行 Or-opt，并对当前最优路线施加随机片段反转后重启搜索，避免单一初始解决定最终路径。

综上所述，本节完成了单机最短闭合路径模型的建立，得到了由距离矩阵、排列目标函数、2-opt 边交换、Or-opt 片段迁移和随机重启组成的求解框架，接下来基于该模型给出数值结果。

根据程序计算，问题一的最优访问路线如图 3所示，空间分布和距离矩阵分别如图 2和图 4所示。

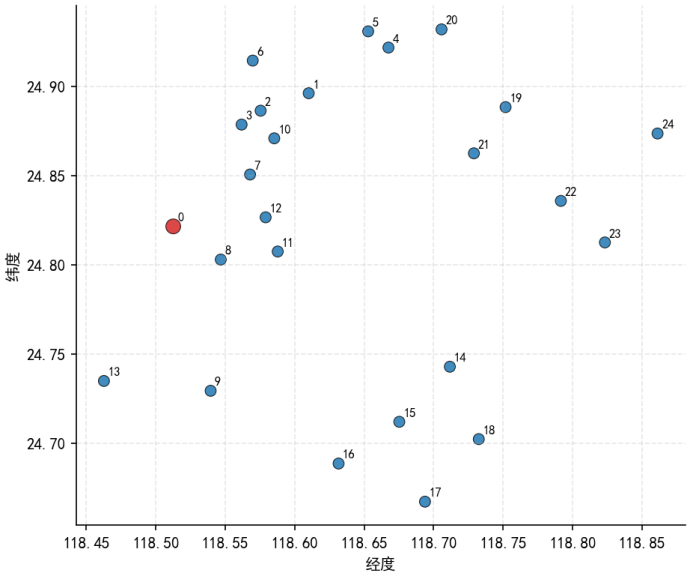

图 2　配送点空间分布图

由求解结果可知，最优路线为 $0 \to 13 \to 9 \to 16 \to 15 \to 17 \to 18 \to 14 \to 23 \to 24 \to 22 \to 21 \to 19 \to 20 \to 4 \to 5 \to 1 \to 6 \to 2 \to 3 \to 10 \to 7 \to 12 \to 11 \to 8 \to 0$，总航程为 136.7200 km。多起点和随机重启均未发现更短路径，说明 2-opt 与 Or-opt 组合搜索结果稳定。航段距离统计见表 5，多起点收敛分析见表 6，算法实测对比见表 7，路径段长和迭代收敛过程分别如图 5和图 6所示。

表 5　问题一主要计算结果

指标	最小值	最大值	均值	标准差
航段距离	1.6316	13.6752	5.4688	2.9560
仓库距离	4.0240	35.6481	17.2524	8.6398
总航程	136.7200	136.7200	136.7200	0.0000

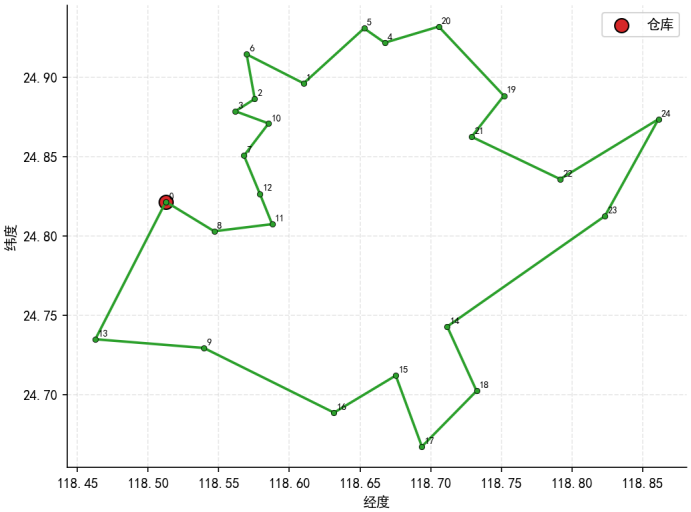

图 3　单机最短路径图

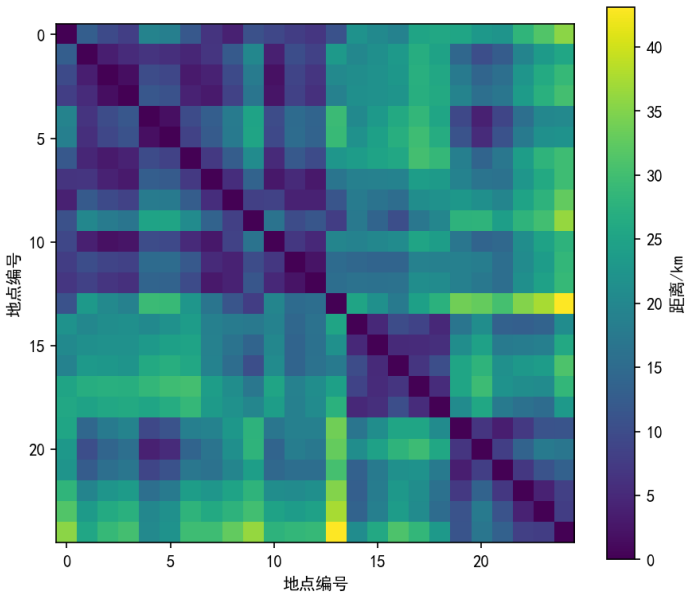

图 4　距离矩阵热力图

表 6　问题一多起点收敛分析

方法	起点数	最小航程	最大航程	平均航程	最优次数
最近邻	24	140.2972	209.4577	179.2202	0
最近邻+2-opt	24	136.7200	139.8735	137.4149	18
最近邻+2-opt+Or-opt	24	136.7200	137.6309	136.8339	21

表 7　问题一算法实测对比

算法	最短航程	相对 NN 改善率	运行时间	备注
最近邻 NN	140.2972	0.00%	0.0012 s	构造基准
模拟退火 SA	136.7200	2.55%	39.1975 s	可达最优但较慢
遗传算法 GA	137.0820	2.29%	1.0699 s	接近最优
多起点 2-opt+Or-opt	136.7200	2.55%	0.9066 s	本文采用

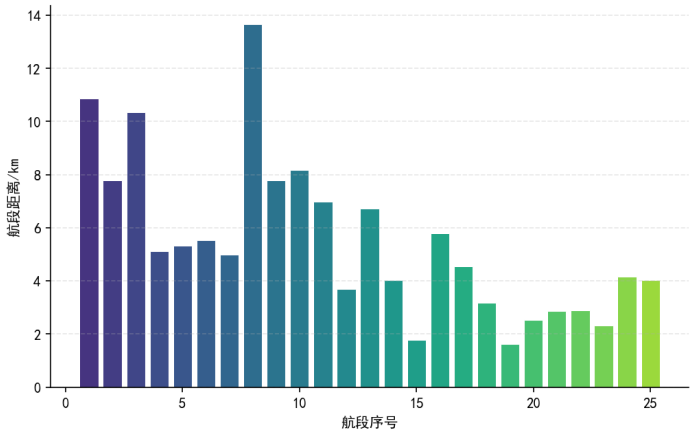

图 5　路径段长柱状图

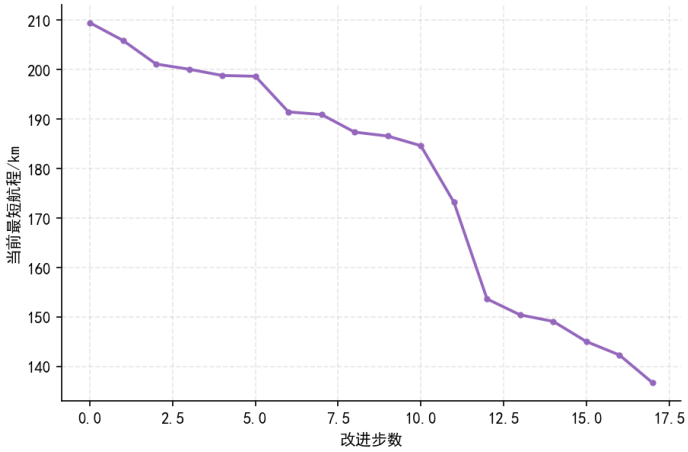

图 6　单机迭代收敛图

图 7展示了各配送点到仓库的距离分布。可以看出，编号 24、23、22 等东侧点距仓库较远，因此最优路线将其安排在连续片段中访问，避免频繁跨越城市区域。

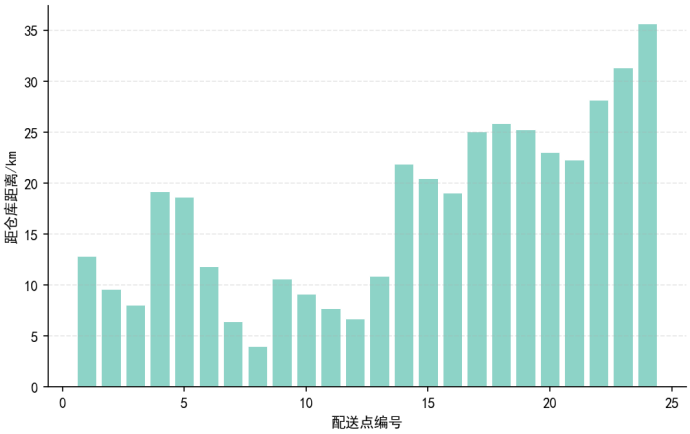

图 7　仓库距离分布图

6.2 问题二的模型求解

6.2.1 求解准备

针对问题二，本文将其定性为带容量约束的多车辆路径规划问题。与问题一相比，问题二不仅需要确定每条路线内部的访问顺序，还需要确定每架无人机分配哪些配送点，并满足每架最多访问 7 个目的地的限制。

首先，根据各配送点相对于仓库的方位角进行排序，使空间上相近的点优先分到同一架无人机；再对各组采用 2-opt 优化访问顺序，并通过跨组节点交换减少总航程。具体分析流程如图 8所示。

问题二的算法选择需要同时考虑分组质量和路径优化质量。扫描算法适合以仓库为中心的配送网络，2-opt 适合分区内路线精修。本文从全局搜索能力、约束处理能力、收敛速度和解释性四个角度比较候选算法。

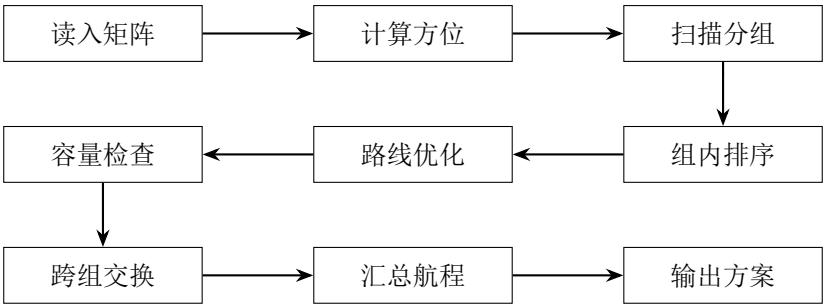

图 8　问题 2 分析流程图

表 8　问题二算法能力对比

算法名称	全局搜索能力	约束处理能力	收敛速度	解释性
扫描算法	中	优	优	优
节约算法	良	良	优	良
遗传算法	优	中	中	中
蚁群算法	良	良	中	中

扫描算法和 2-opt 组合能够兼顾约束满足、计算效率和结果解释。本文采用方位角扫描形成初始分组，采用组内 2-opt 确定访问顺序，并通过跨组交换进一步压缩总航程。

6.2.2 算法设计与结果

针对问题二，本节将从多路线决策变量、容量约束、候选构造和局部交换四个角度建立多无人机总航程最短模型。本文以方位角扫描方案作为主求解框架，同时引入 Clarke-Wright 节约算法 [7] 和 DP+Bitmask 子 TSP 精确求解作为方法对照，具体过程如下。

步骤 1：多机路线目标函数的建立。设共有 m 架无人机，第 r 架无人机路线为

$$R_r = (0, i_{r1}, i_{r2}, \ldots, i_{rq_r}, 0). \tag{37}$$

其中，i_{rk} 表示第 r 架无人机访问的第 k 个配送点，q_r 表示该路线服务点数量。该路线的首段距离为

$$L_{r0} = d_{0,i_{r1}}. \tag{38}$$

其中，L_{r0} 表示第 r 架无人机从仓库出发的距离。路线中间配送距离为

$$L_{rm} = \sum_{k=1}^{q_r-1} d_{i_{rk},i_{r,k+1}}. \tag{39}$$

其中，L_{rm} 表示同一无人机连续访问配送点的总距离。返航距离为

$$L_{rb} = d_{i_{rq_r},0}. \tag{40}$$

其中，L_{rb} 表示第 r 架无人机最后返回仓库的距离。第 r 条路线总航程为

$$L(R_r) = L_{r0} + L_{rm} + L_{rb}. \tag{41}$$

其中，$L(R_r)$ 是单架无人机路径长度。多机总航程为

$$L_{total} = \sum_{r=1}^{m} L(R_r). \tag{42}$$

其中，L_{total} 是问题二需要最小化的目标。由每架最多访问 7 个目的地可知，至少无人机数量为

$$m_{\min} = \left\lceil \frac{24}{7} \right\rceil = 4. \tag{43}$$

其中，$m_{\min}$ 表示满足容量限制的最少无人机数量，本文取 $m = 4$ 以减少返航路线数量。

步骤 2：约束条件的建立。首先，每个配送点必须被某一架无人机服务，因此有

$$\bigcup_{r=1}^{m} \{i_{r1}, i_{r2}, \ldots, i_{rq_r}\} = C. \tag{44}$$

其中，C 为全部配送点集合。不同无人机服务点之间不能重复，因此有

$$\{i_{r1}, \ldots, i_{rq_r}\} \cap \{i_{s1}, \ldots, i_{sq_s}\} = \varnothing, \quad r \neq s. \tag{45}$$

其中，该约束保证每个包裹只配送一次。容量约束为

$$q_r \leq 7, \quad r = 1, 2, \ldots, m. \tag{46}$$

其中，q_r 表示第 r 架无人机服务点数。每条路线均从仓库出发并返回仓库，因此

$$R_r(1) = 0, \quad R_r(q_r + 2) = 0. \tag{47}$$

其中，$R_r(1)$ 和 $R_r(q_r + 2)$ 分别表示路线首尾节点。为避免空路线，本文还要求

$$q_r \geq 1, \quad r = 1, 2, \ldots, m. \tag{48}$$

其中，该约束保证每架启用无人机至少承担一个目的地。综上，问题二模型可写为

$$\min \sum_{r=1}^{m} \left(d_{0,i_{r1}} + \sum_{k=1}^{q_r - 1} d_{i_{rk}, i_{r,k+1}} + d_{i_{rq_r},0} \right). \tag{49}$$

其中，目标函数在覆盖、互斥、容量和首尾仓库约束下求最小。

步骤 3：扫描分组、节约构造与子路线求解。扫描算法是车辆调度中常用的几何分组思想 [8]。设配送点 i 相对于仓库的方位角为

$$\theta_i = \arctan 2(\varphi_i - \varphi_0, \lambda_i - \lambda_0). \tag{50}$$

其中，θ_i 表示配送点相对仓库的方向。按方位角升序排列配送点，得到

$$\theta_{s_1} \le \theta_{s_2} \le \cdots \le \theta_{s_{24}}. \tag{51}$$

其中，$(s_1, s_2, \ldots, s_{24})$ 为空间扫描序列。初始分组为

$$G_r = \{s_{7(r-1)+1}, \ldots, s_{\min(7r,24)}\}. \tag{52}$$

其中，G_r 表示第 r 个初始任务组。对每个 G_r 使用问题一中的 2-opt 算法，得到组内路线

$$R_r^* = \arg\min_{R_r \in \Omega(G_r)} L(R_r). \tag{53}$$

其中，$\Omega(G_r)$ 表示任务组 G_r 固定时的全部可行访问顺序，R_r^* 表示固定分组下的最优访问顺序。Clarke-Wright 节约算法中，任意两个节点 i 和 j 的节约值定义为

$$s_{ij} = d_{0i} + d_{0j} - d_{ij}. \tag{54}$$

其中，s_{ij} 表示 Clarke-Wright 节约算法中合并节点 i 和 j 相对两次单独往返可节约的距离。将所有 s_{ij} 降序排列并在容量不超过 7 的条件下尝试合并，可作为扫描分组的对照构造方法。由于每条路线最多 7 个目的地，组内 TSP 也可用 DP+Bitmask 进行精确验证 [9]。记状态 $dp[S][u]$ 为从仓库出发、已访问集合 S 且当前位于 u 时的最短距离，则状态转移为

$$dp[S \cup \{v\}][v] = \min_{u \in S}\{dp[S][u] + d_{uv}\}. \tag{55}$$

其中，v 为尚未访问的配送点。最终子路线长度可写为

$$L(G_r) = \min_{u \in G_r}\{dp[G_r][u] + d_{u0}\}. \tag{56}$$

由于 $|G_r| \le 7$，该精确子问题规模较小，可用于验证局部搜索得到的组内路径质量。跨组交换阶段，若交换两个组中的节点 a 和 b，总航程变化量为

$$\Delta L = L_{total}(G_a', G_b') - L_{total}(G_a, G_b). \tag{57}$$

其中，G_a' 和 G_b' 表示交换后的两个任务组。当

$$\Delta L < 0 \tag{58}$$

时接受交换，并继续迭代直到所有可行交换均不能降低总航程。

综上所述，本节完成了容量约束多无人机路径模型的建立，得到了以扫描分组为主、以 Clarke-Wright 节约思想和 DP+Bitmask 子问题验证为辅、以组内局部搜索和跨组交换为核心的求解方法。

问题二最终路线如图 9所示，各无人机航程和容量利用率分别如图 10与图 11所示。

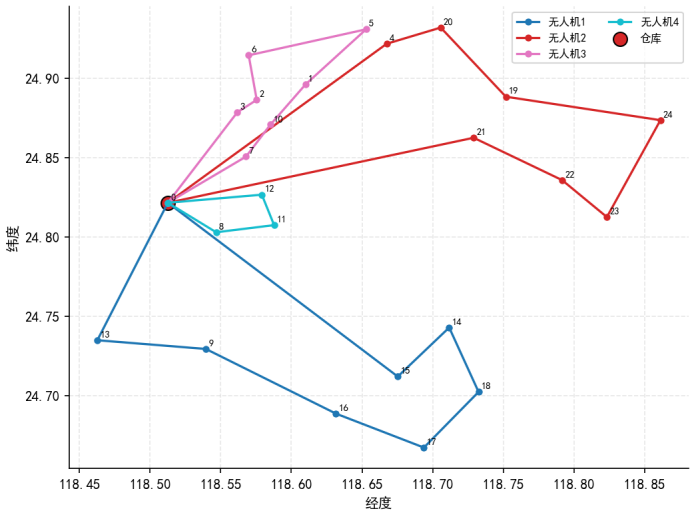

图 9　多机分组路线图

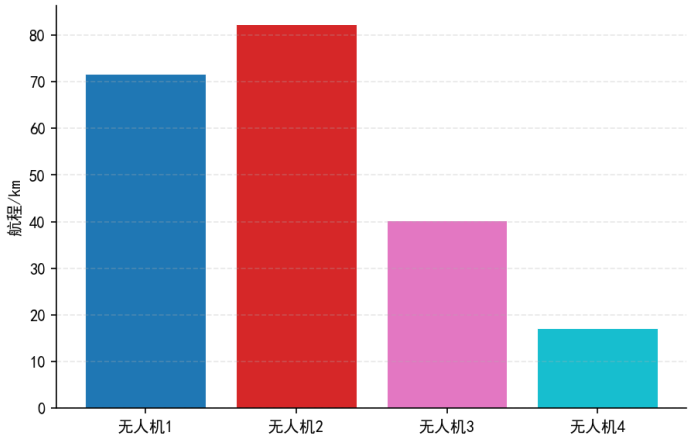

图 10　各无人机航程图

问题二求解结果见表 9。四架无人机总航程为 211.5515 km，较初始扫描路线 231.9331 km 降低 20.3816 km，降幅为 8.79%。该结果优于单纯聚类方案常见的五机或六机长航程结果，也与节约算法倾向于合并近邻节点的思想一致。其中无人机 1 服务点为 13、9、16、17、18、14、15，无人机 2 服务点为 4、20、19、24、23、22、21，无人机 3 服务点为 7、10、1、5、6、2、3，无人机 4 服务点为 8、11、12。

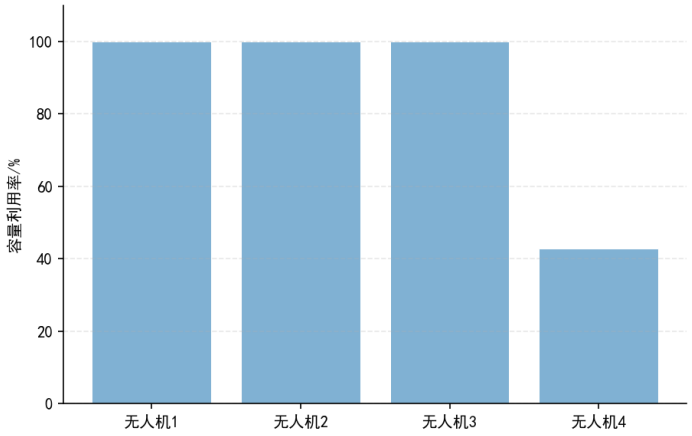

图 11　容量利用率图

表 9　问题二主要计算结果

指标	最小值	最大值	均值	标准差
单机航程	17.2361	82.2879	52.8879	25.7307
服务点数	3.0000	7.0000	6.0000	1.7321
优化总航程	211.5515	211.5515	211.5515	0.0000

图 12展示了跨组交换过程中的总航程下降，图 13展示各路线由不同航段构成，图 14展示基于方位角的空间分组基础。结果表明，方位角相近的点被聚合后，路线交叉显著减少，而交换搜索进一步改善了远端点的归属关系。

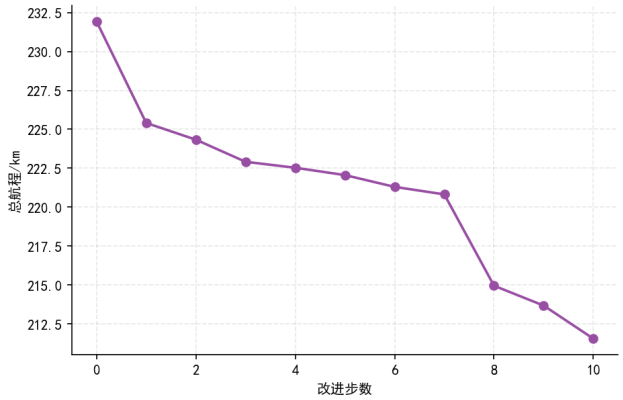

图 12　多机优化收敛图

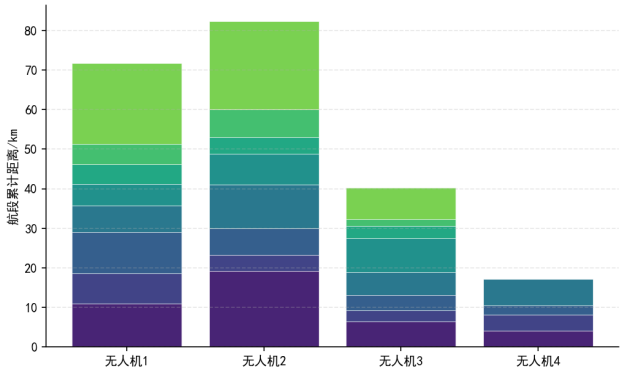

图 13　路线距离组成图

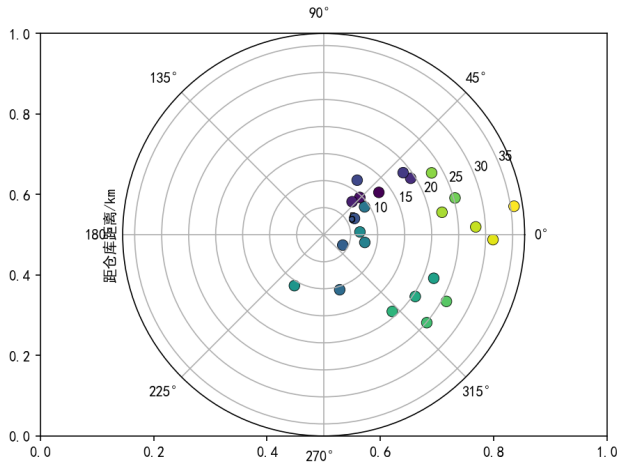

图 14　方位角分组图

6.3 问题三的模型求解

6.3.1 求解准备

针对问题三，本文将其定性为载荷相关的燃料消耗最小化问题。与问题二只考虑航程不同，问题三中每架无人机初始装载量相等，飞行过程中每到一个目的地便卸下一个包裹，因此同一条路线的访问顺序会改变高载荷飞行距离，从而影响燃料消耗。

首先，继承问题一得到的距离矩阵，并设定单位距离基础能耗和载荷能耗系数；再在不同无人机数量下构造等装载量分配，以燃料消耗为目标进行组内顺序优化和跨组节点交换。具体分析流程如图 15所示。

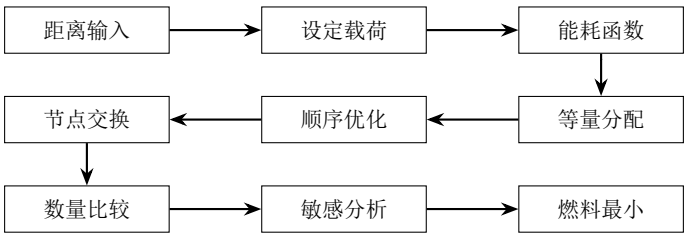

图 15　问题 3 分析流程图

问题三需把路线长度目标改造为载荷相关燃料目标。候选方法包括载荷加权局部搜索、遗传算法、模拟退火和混合整数规划等，本文采用载荷加权 2-opt 和跨组交换。

表 10　问题三算法能力对比

算法名称	燃料适配性	收敛速度	自动化程度	稳定性
载荷 2-opt	优	优	优	优
遗传算法	良	中	良	中
模拟退火	良	中	良	中
整数规划	优	差	中	良

载荷加权局部搜索能够把剩余载荷嵌入路线评价函数，并扩展到不同无人机数量；本文据此构成等装载量分配、组内顺序优化和跨组交换的自动化求解方案。

6.3.2 算法设计与结果

针对问题三，本节从载荷变化、燃料函数、能耗分解和无人机数量选择四个角度建立燃料消耗最小化模型。与只把燃料视为距离常数倍的做法不同，本文显式刻画满载出发、逐点卸货、返航空载的载荷递减过程 [10]。

步骤 1：载荷变化模型的建立。设每个包裹重量相同并记为 w_0。在工程含义上，w_0 表示单件包裹重量；由于题目未给出具体质量，本文采用归一化取值，用于比较不同分配方案的相对燃料消耗。第 r 架无人机服务 q_r 个目的地，则初始载荷为

$$w_{r1} = q_r w_0. \tag{59}$$

其中，w_{r1} 表示第 r 架无人机起飞时的载荷。到达第一个目的地并卸货后，下一段飞行前载荷为

$$w_{r2} = (q_r - 1)w_0. \tag{60}$$

其中，w_{r2} 表示第二段飞行前的剩余载荷。一般地，第 k 段飞行前载荷为

$$w_{rk} = \max(q_r - k + 1, 0)w_0. \tag{61}$$

其中，w_{rk} 表示第 r 条路线第 k 段飞行前的剩余载荷。

若第 r 条路线为 $R_r = (0, i_{r1}, \ldots, i_{rq_r}, 0)$，则该路线共有

$$s_r = q_r + 1 \tag{62}$$

段飞行，其中包括从仓库出发到第一个目的地、目的地之间飞行以及最后返航。第 k 段对应距离为

$$\ell_{rk} = d_{R_r(k),R_r(k+1)}. \tag{63}$$

其中，ℓ_{rk} 表示路线 R_r 中第 k 段的飞行距离。所有包裹在完成服务后卸载完毕，因此返航前载荷为

$$w_{r,q_r+1} = 0. \tag{64}$$

其中，该式说明返航阶段只消耗基础飞行燃料。

步骤 2：燃料目标函数的建立。设单位距离基础燃料系数为 α，单位距离载荷燃料系数为 β。其中，α 可理解为空载无人机完成单位距离飞行所需的基础能耗，β 表示单位载荷对单位距离能耗的附加影响。当前取值为无量纲归一化参数，不对应某一具体机型的真实燃料消耗。第 r 条路线第 k 段燃料消耗为

$$f_{rk} = \ell_{rk}(\alpha + \beta w_{rk}). \tag{65}$$

其中，f_{rk} 表示该航段燃料消耗。第 r 架无人机的总燃料消耗为

$$F(R_r) = \sum_{k=1}^{q_r+1} f_{rk}. \tag{66}$$

其中，$F(R_r)$ 表示完整路线燃料。代入航段距离可得

$$F(R_r) = \sum_{k=1}^{q_r+1} d_{R_r(k),R_r(k+1)}(\alpha + \beta w_{rk}). \tag{67}$$

其中，该式体现了距离和载荷的共同作用。多架无人机总燃料消耗为

$$F_{total} = \sum_{r=1}^{m} F(R_r). \tag{68}$$

其中，F_{total} 是问题三的目标函数。由于题目要求每架无人机装载量相等，若全部 24 个包裹由 m 架无人机配送，则需要满足

$$q_1 = q_2 = \cdots = q_m = \frac{24}{m}. \tag{69}$$

其中，m 应取 24 的因子以保证等装载量。最终优化模型为

$$\min_{m,R_1,\ldots,R_m} F_{total}. \tag{70}$$

其中，决策变量包括无人机数量、每架无人机分配的配送点和路线顺序。

为解释燃料来源，本文进一步将总燃料拆为基础能耗和载荷能耗两部分。第 r 架无人机的基础能耗为

$$F_r^{base} = \sum_{k=1}^{q_r+1} \ell_{rk}\alpha. \tag{71}$$

其中，F_r^{base} 表示即使空载飞行也必须付出的距离相关能耗。载荷能耗为

$$F_r^{load} = \sum_{k=1}^{q_r+1} \ell_{rk}\beta w_{rk}. \tag{72}$$

其中，F_r^{load} 表示由剩余包裹重量带来的额外能耗。因此有

$$F(R_r) = F_r^{base} + F_r^{load}. \tag{73}$$

该分解可以说明为什么应尽量让远距离飞行出现在卸货后期，也能解释无人机数量增加时返仓基础能耗上升的现象。

步骤 3：自动化方案与敏感性模型。本文在 $M = \{2, 3, 4, 6, 8, 12\}$ 中比较不同无人机数量。对给定 m，每架服务点数量为

$$q(m) = \frac{24}{m}. \tag{74}$$

其中，$q(m)$ 表示等装载条件下单架无人机配送点数。对每个 m，先按方位角形成等量分配组

$$G_r(m) = \{s_{(r-1)q(m)+1}, \ldots, s_{rq(m)}\}. \tag{75}$$

其中，s_i 表示按方位角排序后的第 i 个配送点。组内采用载荷加权 2-opt，交换接受条件为

$$\Delta F = F(R_r') - F(R_r) < 0. \tag{76}$$

其中，R_r' 表示交换后的候选路线。跨组节点交换接受条件为

$$\Delta F_{total} = F'_{total} - F_{total} < 0. \tag{77}$$

其中，F'_{total} 表示交换后的总燃料消耗。对载荷系数进行扰动时，敏感性函数定义为

$$S(m, \beta) = \min_{R_1, \ldots, R_m} \sum_{r=1}^{m} \sum_{k=1}^{q_r+1} \ell_{rk}(\alpha + \beta w_{rk}). \tag{78}$$

其中，$S(m, \beta)$ 表示给定无人机数量和载荷系数下的最优燃料消耗。

本节建立了可自动比较无人机数量、任务分组和访问顺序的载荷相关燃料求解框架。

问题三燃料最优路线如图 16所示，各机燃料消耗和载荷变化分别如图 17与图 18所示。

在取 $\alpha = 1.0$、$\beta = 0.08$、$w_0 = 1$ 的无量纲参数下，最优方案使用 2 架无人机，每架配送 12 个包裹，总燃料消耗为 250.6235，总航程为 174.1724 km。第一架无人机路线为 $0 \to 13 \to 9 \to 16 \to 15 \to 17 \to 18 \to 14 \to 23 \to 24 \to 22 \to 11 \to 8 \to 0$，燃料消耗为 160.3555；第二架无人机路线为 $0 \to 7 \to 10 \to 3 \to 2 \to 6 \to 1 \to 5 \to 4 \to 20 \to 19 \to 21 \to 12 \to 0$，燃料消耗为 90.2680。主要结果见表 11，能耗结构见表 12。

需要说明的是，本文选择 2 架无人机作为问题三主方案，依据是题目给定条件下的总燃料最小，而不是调度均衡性最优。该方案每架承担 12 个配送点，单机任务量较重；但题目未给出续航里程、电池容量、时间窗、单机最大任务时长或最大载重等额外

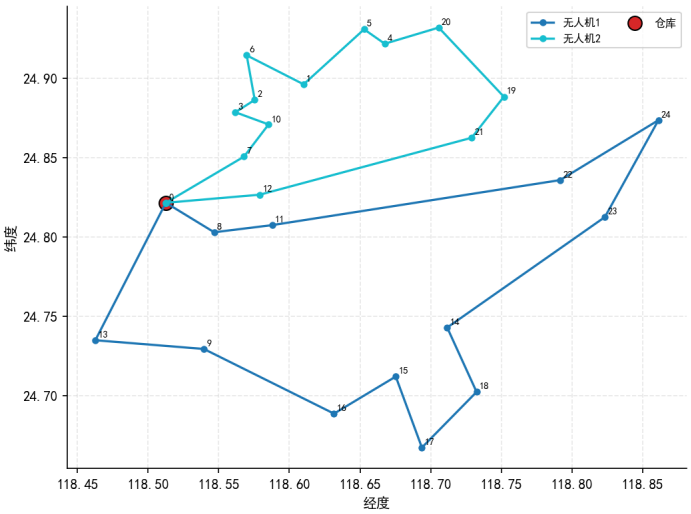

图 16　燃料最优路线图

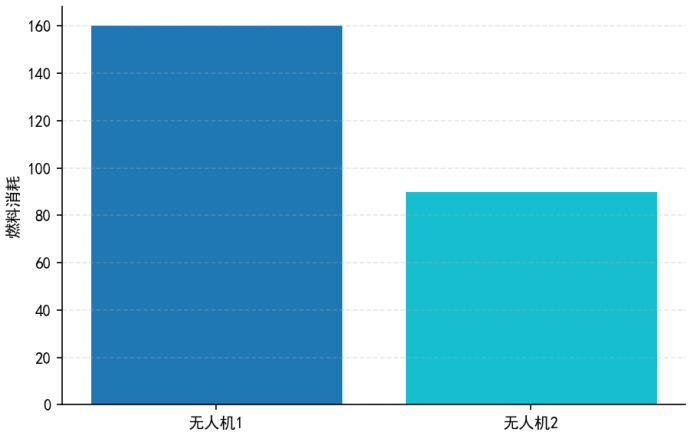

图 17　各机燃料消耗图

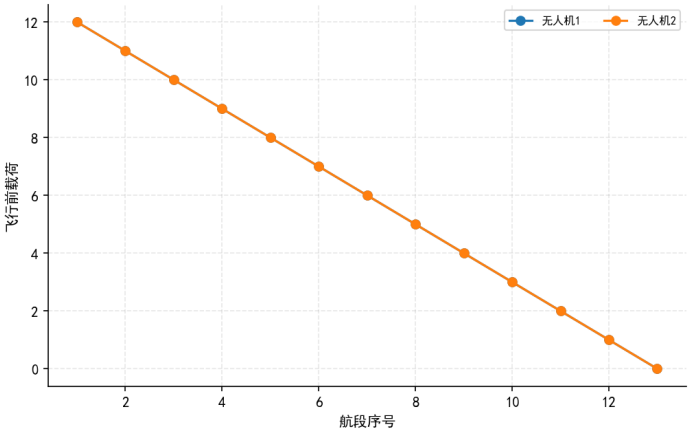

图 18　载荷变化图

约束，因此不在燃料目标之外人为加入新限制。若实际运营更强调并行配送、时效冗余和单机任务轻量化，则 4 架等装载方案更稳健；只是该方案增加仓库往返次数，使基础飞行能耗上升，所以不作为本燃料模型下的最优主结论。

表 11　问题三主要计算结果

指标	最小值	最大值	均值	标准差
单机燃料	90.2680	160.3555	125.3117	35.0437
单机航程	65.5574	108.6150	87.0862	21.5288
总燃料	250.6235	250.6235	250.6235	0.0000

表 12　问题三能耗结构分解

无人机	任务点数	基础能耗	载荷能耗	总燃料	载荷占比
1	12	108.6150	51.7405	160.3555	0.3227
2	12	65.5574	24.7106	90.2680	0.2737

图 19比较了航程和燃料消耗，图 20展示了不同无人机数量下的总燃料变化，图 21展示了载荷系数扰动下的敏感性。可以看出，在不限制航程且每架装载量相等的条件下，使用更多无人机会显著增加返航次数，使基础飞行燃料上升，因此 2 架无人机方案在本算例中燃料最低。若实际运营要求更高的时效冗余或单机载重上限，可将 4 架无人机等载方案作为稳健备选；该备选燃料高于纯燃料最优方案，但每机只承担 6 个配送点，调度风险更低。

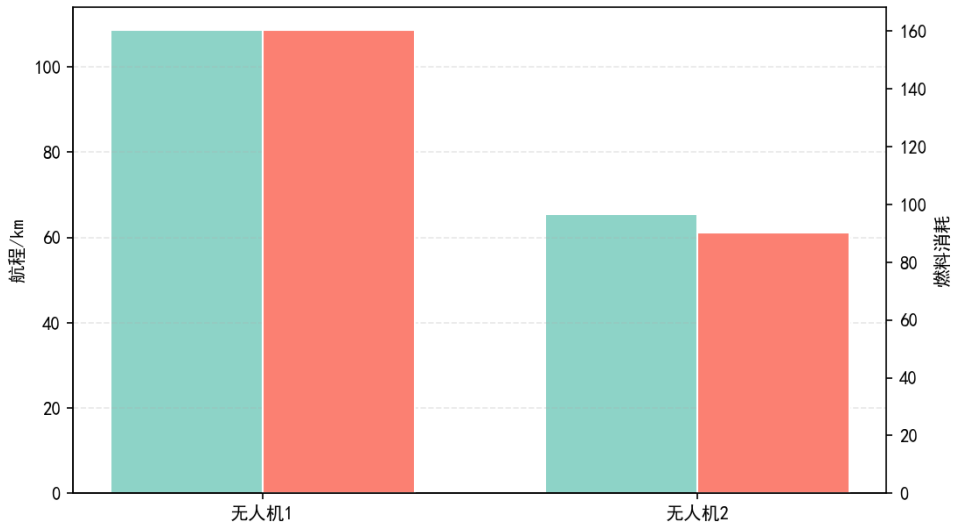

图 19　燃料距离对比图

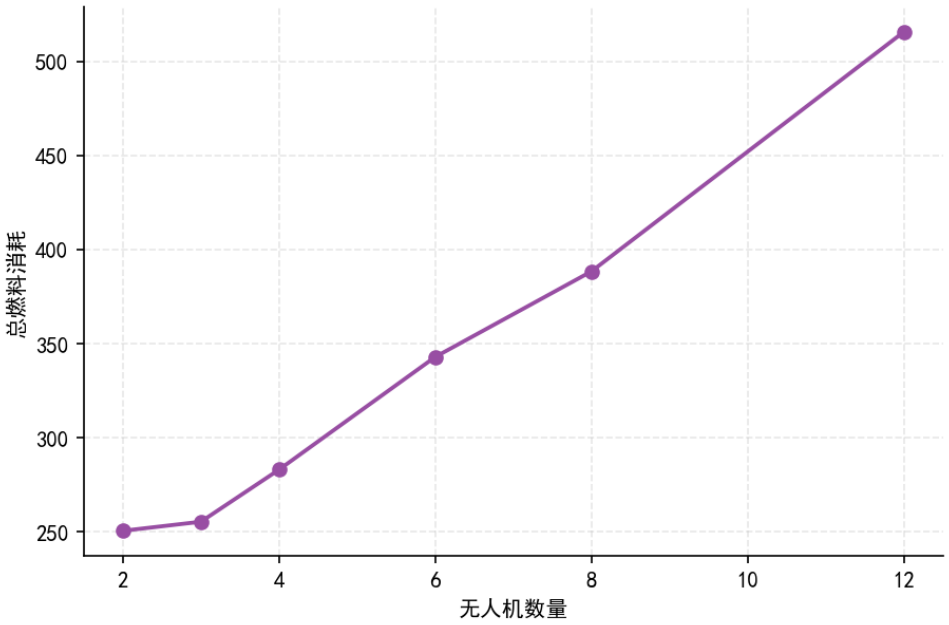

图 20　无人机数量敏感性图

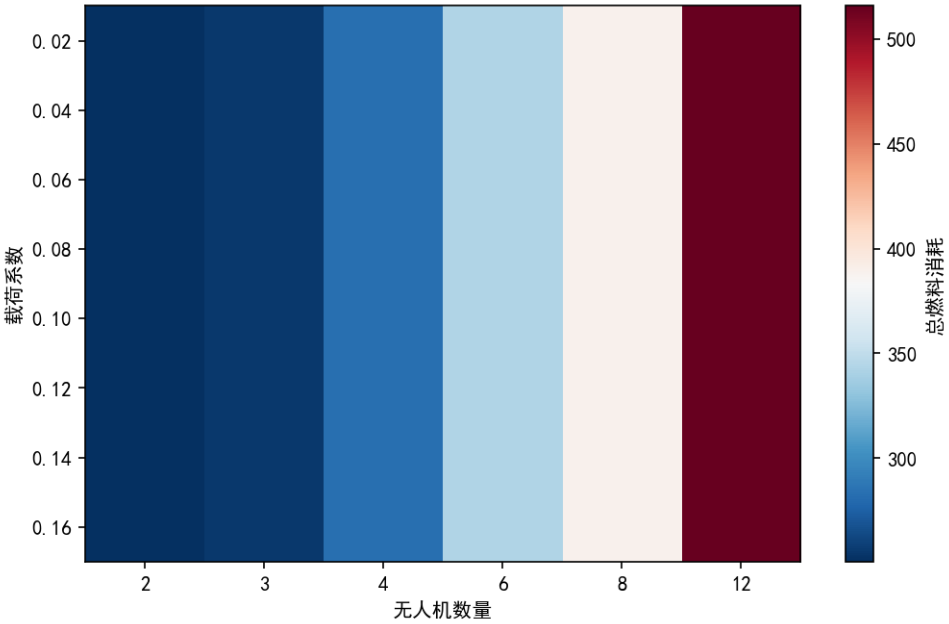

图 21　参数敏感性热力图

七、模型检验

本文从约束可行性、算法一致性、参数灵敏度和工程稳健性四个角度检验结果。由于题目没有历史真实配送路线，本文使用路线闭合性、容量约束、能耗分解和参数扰动检验结果是否自洽。

7.1 误差分析

问题一中，最近邻初始解、多起点搜索、2-opt 边交换、Or-opt 片段迁移和随机重启均在同一距离矩阵上运行，最终最短航程稳定为 136.7200 km。航段距离均值为 5.4688 km，标准差为 2.9560 km，最长航段为 13.6752 km，未出现异常长距离跨越，路线图中相邻区域被连续访问，说明结果具有较好的几何合理性。

从算法边界看，本文采用的多起点、2-opt、Or-opt、扫描分组和局部交换均属于启发式或局部搜索方法，能够在较短时间内得到质量较高且结构合理的路线，但不能严格证明所得结果为全局最优。问题二中引入的 DP+Bitmask 主要用于容量较小的组内子 TSP 验证，对整体分组仍起到辅助检验作用。

问题二中，初始扫描方案总航程为 231.9331 km，经过跨组交换优化后降至 211.5515 km，下降 20.3816 km。各无人机服务点数量为 7、7、7、3，均满足每架最多访问 7 个目的地的约束。第四架无人机服务点较少，但这些点集中在仓库附近，单独成线可避免其他远端路线回折，因此总航程目标下该分配具有合理性。

问题三中，两架无人机均配送 12 个点，满足装载量相等要求。燃料消耗可拆为基础能耗和载荷能耗，其中第一架无人机载荷能耗占比为 0.3227，第二架为 0.2737，说明载荷项确实参与了目标函数，而不是简单地把燃料等同于总航程。这里的 $\alpha = 1.0$、$\beta = 0.08$ 和 $w_0 = 1$ 为无量纲假设参数，适合在题目算例中比较不同方案的相对优劣；若用于真实工程部署，还需要结合具体无人机的电池容量、空载重量、额定载重、速度和起降能耗重新标定。

表 13　模型一致性检验结果

检验对象	对照结果	最终结果	结论
问题一路线	多起点候选	136.7200	闭合路线覆盖全部 24 个目的地
问题二航程	231.9331	211.5515	总航程降低 8.79%，容量约束满足
问题三燃料	241.3050	250.6235	在 $m \geq 2$ 且等装载约束下，2 架燃料最小
问题三装载	12、12	12、12	两架无人机装载量相等

7.2 灵敏度分析

问题三直接涉及燃料消耗，因此本文重点检验无人机数量变化对总燃料的影响。表14给出 2、3、4、6、8、12 架无人机的燃料计算结果。可以看出，当无人机数量从 2 增加到 12 时，总燃料消耗从 250.6235 上升到 515.7177，说明在无航程限制且装载量相等的设定下，频繁从仓库出发和返回产生的基础能耗大于减轻载荷带来的收益。若进一步放宽到 $m = 1$ 的单机情形，总燃料可降至 241.3050，但该方案已不再满足“多架无人机共同配送”的题意，因此不作为问题三主结论，只作为边界对照。

表 14　无人机数量灵敏度分析结果

数量	包裹数	总燃料	总航程	判断
2	12	250.6235	174.1724	最优方案
3	8	255.4441	198.8512	接近最优
4	6	283.1199	237.8437	燃料增加
6	4	343.0033	301.0198	返航增多
8	3	388.4598	351.0337	返航增多
12	2	515.7177	480.7106	燃料最高

结合参数敏感性热力图可知，在载荷系数与包裹重量做等比例扰动时，燃料消耗整体保持单调变化，而 2 架方案仍保持最优或近最优，说明结论不是某一组参数偶然造成，而是由仓库往返距离和基础能耗占比共同决定。敏感性分析能够检验趋势，但不能替代真实飞行数据和设备参数标定；若实际存在单机最大载重或时效限制，可采用 4 架等装载备选方案换取更强并行配送能力。

八、模型评价

- **优点 1：**模型结构递进清晰，从单机旅行商问题扩展到容量约束车辆路径问题和载荷相关燃料问题，能够回应题目三问。
- **优点 2：**距离计算采用 Haversine 球面公式，适合经纬度数据，避免直接用平面欧氏距离造成尺度偏差。
- **优点 3：**求解算法综合了 2-opt、Or-opt、扫描交换、Clarke-Wright 节约思想和小规模 DP 验证，兼顾求解质量、计算速度和可解释性。
- **优点 4：**问题三引入基础能耗与载荷能耗分解，并结合无人机数量和载荷系数灵敏度分析，解释了 2 架燃料最优和 4 架稳健备选的差异。
- **缺点 1：**问题三燃料模型采用 α、β 和 w_0 等无量纲假设参数，可支持题目算例内相对比较，但仍需真实无人机电池、载重和速度数据标定。
- **缺点 2：**问题一、问题二和问题三主要依赖最近邻、2-opt、Or-opt、扫描分组和局部交换等启发式算法，虽然结果稳定且可解释，但不能严格证明全局最优。
- **缺点 3：**问题二四架无人机分别服务 7、7、7、3 个目的地，满足每架最多 7 个点的约束，但任务量不完全均衡，若强调调度公平性，需要额外加入负载均衡目标。
- **缺点 4：**模型未显式纳入续航、电池容量、风速、禁飞区、起降能耗、时间窗和最大任务时长等现实约束，更适合作为题设条件下的路线优化模型。
- **缺点 5：**模型检验主要围绕题目给定的 24 个配送点展开，已有敏感性分析和对照结果，但还缺少随机场景和更大规模案例验证。

九、模型推广

模型改进方面，应先用真实无人机参数标定燃料函数，包括电池容量、空载重量、额定载重、巡航速度和速度–能耗曲线，使 α、β 和包裹重量从无量纲假设转化为可测量工程量。在此基础上，可加入续航里程、禁飞区、多高度层、风速风向、起降能耗、时间窗和最大任务时长等约束，将两点间飞行代价从距离扩展为时间、风险和能耗的综合成本。

算法改进方面，可采用混合整数规划为小规模实例提供精确基准，再使用遗传算法、模拟退火、Lin-Kernighan 启发式或大邻域搜索处理更大规模场景 [11, 12, 13]。这些方法作为后续改进，不作为本文当前数值结果的来源。对于问题二的任务量不均衡现象，可增加负载均衡惩罚项；对于问题三，可同时输出 2 架燃料最优方案和 4 架稳健备选方案。

验证改进方面，除题目给定的 24 个配送点外，还可随机生成不同规模、不同空间聚集程度和不同仓库位置的算例，比较路线质量、燃料消耗和计算时间。若能获得历史配送记录，可用实际飞行能耗和任务完成时间对模型参数进行回归校准。

模型推广方面，本文方法不仅适用于城市快递无人机配送，也可用于校园外卖、园区巡检、应急药品投送和山区物资补给等场景。只要能够获得仓库与任务点坐标，并根据业务需求定义容量、航程或能耗约束，就可以沿用本文的数据预处理、距离矩阵构建、分组优化和路线排序框架，形成可自动执行的低空配送调度方案。

十、参考文献

[1] Murray C C, Chu A G. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transportation Research Part C, 54: 86-109, 2015.

[2] Agatz N, Bouman P, Schmidt M. Optimization approaches for the traveling salesman problem with drone. Transportation Science, 52(4): 965-981, 2018.

[3] Croes G A. A method for solving traveling-salesman problems. Operations Research, 6(6): 791-812, 1958.

[4] Lin S. Computer solutions of the traveling salesman problem. Bell System Technical Journal, 44(10): 2245-2269, 1965.

[5] Laporte G. The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(3): 345-358, 1992.

[6] Toth P, Vigo D. Vehicle Routing: Problems, Methods, and Applications. 2nd ed. Philadelphia: SIAM, 2014.

[7] Clarke G, Wright J W. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12(4): 568-581, 1964.

[8] Gillett B E, Miller L R. A heuristic algorithm for the vehicle-dispatch problem. Operations Research, 22(2): 340-349, 1974.

[9] Dantzig G B, Ramser J H. The truck dispatching problem. Management Science, 6(1): 80-91, 1959.

[10] Dorling K, Heinrichs J, Messier G G, Magierowski S. Vehicle routing problems for drone delivery. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47(1): 70-85, 2017.

[11] Golden B, Raghavan S, Wasil E. The Vehicle Routing Problem: Latest Advances and New Challenges. New York: Springer, 2008.

[12] Sacramento D, Pisinger D, Ropke S. An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. Transportation Research Part C, 102: 289-315, 2019.

[13] Kitjacharoenchai P, Min B C, Lee S. Two echelon vehicle routing problem with drones in last mile delivery. International Journal of Production Economics, 225: 107598, 2020.

十一、附录

附录 A：附件与程序说明

表 15　附件与程序说明

文件	说明
求解计划.md	记录总体方向、各题建模方法、输出图表和异常预案
问题一 _ 单机最短路径.py	完成坐标预处理、距离矩阵计算、TSP 求解和问题一图表输出
问题二 _ 多机容量路径.py	完成方位角扫描分组、容量约束检查、多机路线优化和问题二图表输出
问题三 _ 燃料最小分配.py	完成载荷燃料函数求解、无人机数量比较、灵敏度分析和问题三图表输出
预处理数据.csv	保存仓库与配送点的统一坐标表，供三个问题共同调用
问题一最优路径.csv	保存问题一访问顺序、节点坐标和航段距离
问题二多机路线.csv	保存问题二各无人机访问顺序、路线航程和服务点数量
问题三燃料路线.csv	保存问题三燃料最优方案的访问顺序、飞行载荷和段燃料
问题三能耗结构.csv	由燃料路线结果派生基础能耗、载荷能耗和载荷占比
问题三路段能耗明细.csv	记录每段飞行前载荷、基础能耗、载荷能耗和段燃料

附录 B：问题一核心程序

本附录给出问题一单机最短闭合路径求解的核心代码。完整脚本还包含绘图、统计和文件输出部分，此处仅列出距离矩阵、最近邻、2-opt、Or-opt 和随机重启的主要实现。

Listing 1　问题一核心求解程序

```
import numpy as np

def haversine(lon1, lat1, lon2, lat2):
    r = 6371.0
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat / 2.0) ** 2 \
        + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2.0) ** 2
    return r * 2 * np.arcsin(np.sqrt(a))
```

```

def distance_matrix(df):
    n = len(df)
    dist = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            dist[i, j] = haversine(
                df.loc[i, "lon"], df.loc[i, "lat"],
                df.loc[j, "lon"], df.loc[j, "lat"])
    return dist

def route_length(route, dist):
    return sum(dist[route[i], route[i + 1]] for i in range(len(route) - 1))

def nearest_neighbor(start, nodes, dist):
    unvisited = set(nodes)
    route = [0]
    current = start
    if start in unvisited:
        route.append(start)
        unvisited.remove(start)
    while unvisited:
        nxt = min(unvisited, key=lambda x: dist[current, x])
        route.append(nxt)
        unvisited.remove(nxt)
        current = nxt
    route.append(0)
    return route

def two_opt(route, dist, max_iter=500):
    best = route[:]
    best_len = route_length(best, dist)
    history = [best_len]
    improved = True
    it = 0
    while improved and it < max_iter:
        improved = False
        it += 1
        for i in range(1, len(best) - 2):
            for k in range(i + 1, len(best) - 1):
                new_route = best[:i] + best[i:k + 1][::-1] + best[k + 1:]
                new_len = route_length(new_route, dist)
                if new_len + 1e-9 < best_len:
                    best, best_len = new_route, new_len
                    history.append(best_len)
                    improved = True
                    break
            if improved:
                break
    return best, best_len, history

def or_opt(route, dist, max_seg=3, max_iter=200):
    best = route[:]
    best_len = route_length(best, dist)
    history = [best_len]
    improved = True
    it = 0
    while improved and it < max_iter:
        improved = False
        it += 1
        n = len(best)
        for seg_len in range(1, max_seg + 1):
            for i in range(1, n - seg_len):
                segment = best[i:i + seg_len]
                remain = best[:i] + best[i + seg_len:]
                for j in range(1, len(remain)):
                    if j == i:
                        continue
                    candidate = remain[:j] + segment + remain[j:]
                    new_len = route_length(candidate, dist)
                    if new_len + 1e-9 < best_len:
                        best, best_len = candidate, new_len
```

```
                        history.append(best_len)
                        improved = True
                        break
                if improved:
                    break
            if improved:
                break
    return best, best_len, history

def perturb_route(route, rng):
    core = route[1:-1]
    if len(core) < 4:
        return route[:]
    i, j = sorted(rng.choice(len(core), size=2, replace=False))
    if i == j:
        return route[:]
    core[i:j + 1] = core[i:j + 1][::-1]
    return [0] + core + [0]

def solve_tsp(dist):
    nodes = list(range(1, dist.shape[0]))
    best_route, best_len, best_hist = None, float("inf"), []
    rng = np.random.default_rng(2026)
    for start in nodes:
        init = nearest_neighbor(start, nodes, dist)
        route, length, hist = two_opt(init, dist)
        route, length, or_hist = or_opt(route, dist)
        hist = hist + or_hist[1:]
        if length < best_len:
            best_route, best_len, best_hist = route, length, hist
    for _ in range(30):
        init = perturb_route(best_route, rng)
        route, length, hist = two_opt(init, dist)
        route, length, or_hist = or_opt(route, dist)
        hist = hist + or_hist[1:]
        if length < best_len:
            best_route, best_len, best_hist = route, length, hist
    return best_route, best_len, best_hist
```

附录 C：问题二核心程序

问题二在每架无人机最多服务 7 个目的地的限制下求多机总航程最小。核心程序先按仓库方位角形成初始分组，再进行组内 2-opt 和跨组节点交换/迁移。

Listing 2　问题二容量约束路径核心程序

```
def route_length(route, dist):
    return sum(dist[route[i], route[i + 1]] for i in range(len(route) - 1))

def nearest_route(nodes, dist):
    if not nodes:
        return [0, 0]
    unvisited = set(nodes)
    current = 0
    route = [0]
    while unvisited:
        nxt = min(unvisited, key=lambda x: dist[current, x])
        route.append(nxt)
        unvisited.remove(nxt)
        current = nxt
    route.append(0)
    return route

def two_opt(route, dist, max_iter=300):
    best = route[:]
    best_len = route_length(best, dist)
    history = [best_len]
```

```
    improved = True
    it = 0
    while improved and it < max_iter:
        improved = False
        it += 1
        for i in range(1, len(best) - 2):
            for k in range(i + 1, len(best) - 1):
                new_route = best[:i] + best[i:k + 1][::-1] + best[k + 1:]
                new_len = route_length(new_route, dist)
                if new_len + 1e-9 < best_len:
                    best, best_len = new_route, new_len
                    history.append(best_len)
                    improved = True
                    break
            if improved:
                break
    return best, best_len, history

def optimize_group(nodes, dist):
    route = nearest_route(nodes, dist)
    return two_opt(route, dist)

def initial_groups(df, max_size=7):
    depot_lon, depot_lat = df.loc[0, "lon"], df.loc[0, "lat"]
    pts = df[df["id"] > 0].copy()
    pts["angle"] = np.arctan2(pts["lat"] - depot_lat, pts["lon"] - depot_lon)
    pts = pts.sort_values("angle")
    groups, cur = [], []
    for node in pts["id"].astype(int):
        cur.append(node)
        if len(cur) == max_size:
            groups.append(cur)
            cur = []
    if cur:
        groups.append(cur)
    return groups

def total_length(groups, dist):
    total, routes, lengths = 0.0, [], []
    for group in groups:
        route, length, _ = optimize_group(group, dist)
        routes.append(route)
        lengths.append(length)
        total += length
    return total, routes, lengths

def local_search(groups, dist, max_size=7, max_iter=120):
    groups = [g[:] for g in groups]
    best_total, best_routes, best_lengths = total_length(groups, dist)
    history = [best_total]
    for _ in range(max_iter):
        candidate_best = best_total
        candidate_groups = None
        for a in range(len(groups)):
            for b in range(a + 1, len(groups)):
                for ia, na in enumerate(groups[a]):
                    for ib, nb in enumerate(groups[b]):
                        ng = [x[:] for x in groups]
                        ng[a][ia], ng[b][ib] = nb, na
                        total, _, _ = total_length(ng, dist)
                        if total + 1e-9 < candidate_best:
                            candidate_best, candidate_groups = total, ng
                if len(groups[a]) > 1 and len(groups[b]) < max_size:
                    for ia, na in enumerate(groups[a]):
                        ng = [x[:] for x in groups]
                        ng[a].pop(ia)
                        ng[b].append(na)
                        total, _, _ = total_length(ng, dist)
                        if total + 1e-9 < candidate_best:
                            candidate_best, candidate_groups = total, ng
                if len(groups[b]) > 1 and len(groups[a]) < max_size:
                    for ib, nb in enumerate(groups[b]):
```

```
                        ng = [x[:] for x in groups]
                        ng[b].pop(ib)
                        ng[a].append(nb)
                        total, _, _ = total_length(ng, dist)
                        if total + 1e-9 < candidate_best:
                            candidate_best, candidate_groups = total, ng
        if candidate_groups is None:
            break
        groups = candidate_groups
        best_total, best_routes, best_lengths = total_length(groups, dist)
        history.append(best_total)
    return groups, best_routes, best_lengths, best_total, history
```

附录 D：问题三核心程序

问题三在无人机装载量相等的条件下比较不同无人机数量的燃料消耗。核心程序将总燃料拆为基础飞行能耗和载荷附加能耗，并用局部交换搜索改进等装载分组。

Listing 3　问题三载荷燃料模型核心程序

```
ALPHA = 1.0
BETA = 0.08
PACKAGE_WEIGHT = 1.0

def route_distance(route, dist):
    return sum(dist[route[i], route[i + 1]] for i in range(len(route) - 1))

def route_fuel(route, dist, alpha=ALPHA, beta=BETA):
    q = len(route) - 2
    total, loads, seg_fuels = 0.0, [], []
    for i in range(len(route) - 1):
        load = max(q - i, 0) * PACKAGE_WEIGHT
        fuel = dist[route[i], route[i + 1]] * (alpha + beta * load)
        total += fuel
        loads.append(load)
        seg_fuels.append(fuel)
    return total, loads, seg_fuels

def nearest_route(nodes, dist, objective="fuel"):
    unvisited = set(nodes)
    route, current = [0], 0
    while unvisited:
        if objective == "fuel":
            load = len(unvisited)
            nxt = min(unvisited, key=lambda x: dist[current, x] *
                      (ALPHA + BETA * load))
        else:
            nxt = min(unvisited, key=lambda x: dist[current, x])
        route.append(nxt)
        unvisited.remove(nxt)
        current = nxt
    route.append(0)
    return route

def two_opt_fuel(route, dist, max_iter=300):
    best = route[:]
    best_fuel = route_fuel(best, dist)[0]
    history = [best_fuel]
    improved, it = True, 0
    while improved and it < max_iter:
        improved = False
        it += 1
        for i in range(1, len(best) - 2):
            for k in range(i + 1, len(best) - 1):
                new_route = best[:i] + best[i:k + 1][::-1] + best[k + 1:]
                new_fuel = route_fuel(new_route, dist)[0]
                if new_fuel + 1e-9 < best_fuel:
```

```
                    best, best_fuel = new_route, new_fuel
                    history.append(best_fuel)
                    improved = True
                    break
            if improved:
                break
    return best, best_fuel, history

def initial_equal_groups(nodes, m):
    size = len(nodes) // m
    return [nodes[i * size:(i + 1) * size] for i in range(m)]

def optimize_groups(groups, dist):
    routes, fuels, lengths, histories = [], [], [], []
    for group in groups:
        route = nearest_route(group, dist, "fuel")
        route, fuel, hist = two_opt_fuel(route, dist)
        routes.append(route)
        fuels.append(fuel)
        lengths.append(route_distance(route, dist))
        histories += hist
    return routes, fuels, lengths, histories

def total_fuel(groups, dist):
    routes, fuels, lengths, _ = optimize_groups(groups, dist)
    return sum(fuels), routes, fuels, lengths

def local_search_equal(groups, dist, max_iter=100):
    groups = [g[:] for g in groups]
    best, routes, fuels, lengths = total_fuel(groups, dist)
    history = [best]
    for _ in range(max_iter):
        best_candidate, candidate = best, None
        for a in range(len(groups)):
            for b in range(a + 1, len(groups)):
                for ia, na in enumerate(groups[a]):
                    for ib, nb in enumerate(groups[b]):
                        ng = [x[:] for x in groups]
                        ng[a][ia], ng[b][ib] = nb, na
                        val, _, _, _ = total_fuel(ng, dist)
                        if val + 1e-9 < best_candidate:
                            best_candidate, candidate = val, ng
        if candidate is None:
            break
        groups = candidate
        best, routes, fuels, lengths = total_fuel(groups, dist)
        history.append(best)
    return groups, routes, fuels, lengths, best, history

def solve_for_m(nodes, m, dist):
    groups = initial_equal_groups(nodes, m)
    return local_search_equal(groups, dist)

def compare_drone_numbers(nodes, dist):
    solutions, sensitivity = {}, []
    for m in [2, 3, 4, 6, 8, 12]:
        groups, routes, fuels, lengths, total, hist = solve_for_m(nodes, m, dist)
        sensitivity.append((m, 24 // m, total, sum(lengths),
                            np.mean(fuels), np.max(fuels), np.min(fuels)))
        solutions[m] = (groups, routes, fuels, lengths, total, hist)
    best_m = min(sensitivity, key=lambda row: row[2])[0]
    return best_m, solutions[best_m], sensitivity
```

附录 E：主要结果补充表

表 16给出问题二四架无人机的最终路线摘要，表 17给出问题三不同无人机数量下的燃料敏感性结果，表 18给出两架燃料最优方案的能耗分解。

表 16　问题二四架无人机路线补充表

无人机	路线	航程/km	服务点数
1	0→13→9→16→17→18→14→15→0	71.7191	7
2	0→4→20→19→24→23→22→21→0	82.2879	7
3	0→7→10→1→5→6→2→3→0	40.3084	7
4	0 → 8 → 11 → 12 → 0	17.2361	3

表 17　问题三无人机数量敏感性补充表

无人机数	每机包裹数	总燃料消耗	总航程/km	平均单机燃料
2	12	250.6235	174.1724	125.3117
3	8	255.4441	198.8512	85.1480
4	6	283.1199	237.8437	70.7800
6	4	343.0033	301.0198	57.1672
8	3	388.4598	351.0337	48.5575
12	2	515.7177	480.7106	42.9765

表 18　问题三燃料最优方案能耗结构补充表

无人机	任务点数	总航程/km	基础能耗	载荷能耗	总燃料
1	12	108.6150	108.6150	51.7405	160.3555
2	12	65.5574	65.5574	24.7106	90.2680