队伍编号	MC25000966
题号	D

基于霍尔特指数平滑法及粒子群优化的短途运输货量预测与调度决策

摘　要

随着短途运输在物流网络中的重要性不断增加，如何提高短途运输的效率和降低成本成为了一个亟待解决的问题。本文针对短途运输中的货量预测和车辆调度问题，提出了一种基于霍尔特指数平滑法和粒子群优化算法（PSO）相结合的方法，以期为实际物流运营提供优化策略。

针对问题一，本文采用霍尔特指数平滑法建立回归模型，基于 LBFGSB 算法对模型进行优化，从而得出预测结果。本文首先对历史包裹量进行分析处理，考虑到短途运输的季节性和周期性变化，采用霍尔特指数平滑法对短途运输线路的货量进行预测。对历史包裹量数据进行拟合后，以结果构建目标函数，从而转化为一个优化问题。通过 LBFGSB 算法求得三个平滑参数的最优解。其中“场地 3-站点 83–0600”与“场地 3-站点 83–1400”的预测结果如 5.1 所示

针对问题二，需要确定最优短途运输策略，使得自有车的周转率和所有车辆均包裹尽可能高，总成本尽可能低。于是将发车时间、自有车选择、车的数量以及是否串点四个变量统一为决策矩阵，约束条件有车辆装载量约束、自有车辆数约束、自有车辆周转约束、串点关系约束。结合上述条件构建目标函数，运用粒子群优化算法（PSO）对决策矩阵的定义域搜索最优解。经过迭代 100 次后取得各线路的最优调度策略，其中“场地 3-站点 83–0600”与“场地 3-站点 83–1400”的调度结果如 5.3 所示。

针对问题三，本文引入了标准容器后，重新构建与问题二中类似的目标函数，利用问题二中的粒子群优化策略，根据问题一和问题二的结果进一步优化车辆调度，得出该条件下的最优决策。标准化容器具有更短的装卸时间，但其装载量降低。为了评估这种标准化容器对调度策略的影响，本文通过引入一个离散决策变量表示各运输需求是否使用该标准容器，并增加标准容器约束对应的罚函数，通过粒子群优化算法重新求解调度问题。结果表明，尽管标准化容器的使用降低了单车的装载量，但其在装卸效率上的优势显著提高了车辆的整体调度效率，其中“场地 3-站点 83–0600”与“场地 3-站点 83–1400”的调度结果如下文 5.4 所示。

针对问题四，本文基于调度模型建立扰动模型，评估调度模型在可能的偏差范围内的预测表现，从而评估问题一预测结果偏差对调度模型优化结果的影响。通过比较不同偏差下的调度效果（如自有车周转率、车辆均包裹），并对历史观测数据和历史预测数据反解出来的扰动周期和范围进行分析，求得鲁棒性估计为$R(52\%)$。

关键词：短途运输 货量预测 车辆调度 指数平滑法 粒子群优化

目录

一、 问题重述

1.1 问题背景

短途运输是物流网络中一个至关重要的环节，主要发生在同城或同省的末端区域，承担着将货物从末分拣中心送往营业部的任务。此环节具有运输距离短、资源可复用、时效性要求高等特点。短途运输的效率通常直接影响客户的体验，特别是在履约时效和运输安全方面。因此，如何优化短途运输的线路和资源配置，提高运输效率并降低成本，是一个重要的研究课题。

然而，短途运输面临着多种挑战，其中最为关键的是如何预测每条线路的运输货量。准确的货量预测不仅能帮助合理安排运力资源，还能为后续的调度决策提供依据。预测过程中需要考虑到运输需求的不确定性、车载能力的限制以及运输过程中可能出现的各类异常情况，如订单取消或线路变更等。因此，设计一个高效的运输货量预测模型，以及在此基础上进行合理的运输调度，是本问题的核心目标。

1.2 问题提出

给定 5 个自有车队的已知相关信息，包括起始场地、目的场地、发运节点、车队编码、在途时长、自有变动成本、外部承运商成本。

对各线路历史 15 天的实际包裹量统计数据进行分析，每天 6-11 点和 14-21 点可假设为不生产包裹量。假设所有车队只有一种车型，该车型的日固定成本为 100，单次可装载 1000 个包裹，装车及卸车时长均为 45 分钟。每天 21 点开始进行预测和调度。

（1） 建立货量预测模型，基于每条线路的总货量拆解到 10 分钟颗粒度的前提条件，对未来 1 天各条线路的货量进行预测。

（2） 根据问题 1 的结果，确定短途运输策略，并确定每个需求的承运车辆。要求自有车的周转率和所有车辆均包裹尽可能高，总成本尽可能低。

（3） 假设存在一种无限量标准容器，将装车及卸车时长缩短至 10 分钟，但车辆装载包裹量下降至 800 个。根据问题 1 输出的结果，重新确定运输需求，优化目标与问题 2 相同。在问题 2 输出结果的基础上，需判断各运输需求是否使用该标准容器。

（4） 当问题 1 的货量预测结果出现偏差时，基于问题 3 评估对调度模型优化结果的影响。

二、 问题分析

2.1 问题一的分析

问题一需要根据历史货量数据（以十分钟为颗粒度分段）建立回归模型预测未来一天内的各线路货量。根据附件二对各线路各天从 14：00 到第二天 14：00 的包裹量数据进行累加，通过数据分析可知 15 天内的货量涨势呈现明显的周期性变化趋势，故选择在短期与季节性时序预测问题上表现良好指数平滑法（Holt 线性趋势方法和带阻尼的趋势方法）进行预测分析。也从附件二中可以观察出以十分钟为颗粒度，一天内各时段的包裹量占这一天总包裹量的比例变化随时间变化不大，因此根据比例将未来一天的预测量分布到这一天以十分钟为颗粒度的时间段中。

2.2 问题二的分析

问题二要求通过问题一中运输预测结果来合理分配车队，在运输任务完成的条件下最大化线路对应负责车队自有车的周转率以及每个车辆的均包裹量的同时最小化运输的成本。本情境下的自变量包括每条线路的发运车辆数，预计发运时间，车辆的自有与外部类型及串点方案，约束条件有车辆装载量约束、自有车辆数约束、自有车辆周转约束、串点关系约束。

类似贪心算法等不善于处理组合多约束优化问题的模型在该条件下便失去了优势。而粒子群优化算法能够灵活适应复杂的约束条件，且具有较强的全局搜索能力，尤其适合像车辆调度、发运时间安排、串点任务等多目标优化问题。因此，我们选择结合罚函数描述约束条件构建优化函数，基于粒子群优化算法求得优化函数的最优解向量，即求得最优运输策略，以此解决问题二的调度问题。

2.3 问题三的分析

问题三要求在问题二的基础上引入“标准容器”，并重新确定运输策略。标准容器将装车及卸车时长至 10 分钟，并使车辆装载包裹量下降至 800 个

我们引入一个二进制决策变量表示各运输需求是否使用该标准容器，并通过该决策变量更新优化函数中的约束条件。后续求解过程与问题二相同，基于粒子群优化算法对更新后的优化函数进行求解，即可求得问题三条件下的最优运输策略。

2.4 问题四的分析

问题四要求在问题三的基础上评估我们的调度计划在问题一的预测出现偏差的情况下会产生什么变化，造成什么影响，所以我们可以在原预测货量的基础上引入一个扰动系数，计算扰动后的失败率等指标建立扰动评估方程，计算出我们的模型在不同扰动系数下的鲁棒性。

其次，还应当根据历史的预测量与实际量之间的偏差大致估计 12/16 的扰动系数，代入扰动方程计算针对我们 12.16 调度策略的鲁棒性,即可解决问题四。

三、 模型假设

1. 假设有足够的历史数据来进行货量预测，至少 15 天的实际包裹量数据是有效的，能够提供趋势和季节性模式。
2. 假设每个车队只有一种车型，且该车型的固定成本、装载量和装卸时间是已知且固定的。
3. 假设每个车队的自有车数量和运力足够处理大部分预测的货量，若自有车辆不足，则通过外部承运商补充，但其成本较高。
4. 假设所有发运节点都是固定的，且每个包裹的运输路径和时间节点是已知的。
5. 假设扰动分析附件三所对应得预测模型与本文预测模型的扰动系数分布相似。
6. 假设短途运输网络的基本结构和路线（如起始场地、目的场地和发运节点）在整个调度周期内保持稳定，不会有重大变更。

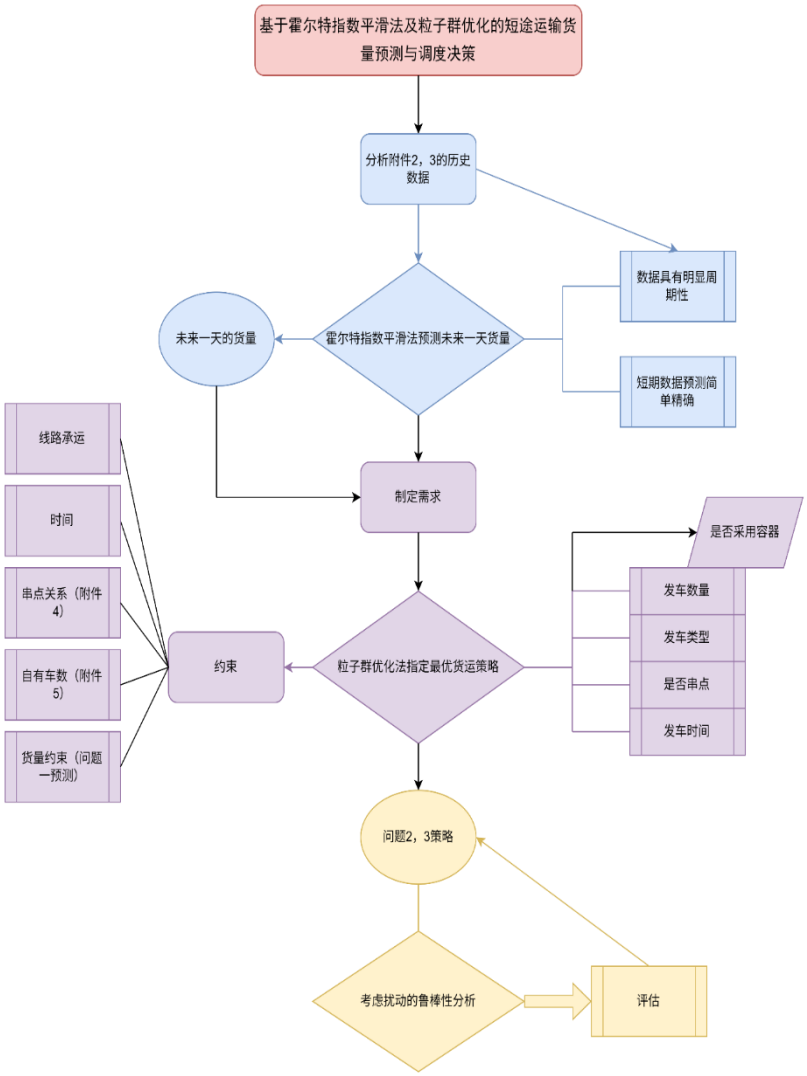
四、 符号说明

符号	说明	单位
L_t	t时刻的平滑水平	/
T_t	t时刻的平滑趋势	/
S_t	季节性成分，在时间t处的季节性（周期性）	/
h	预测步长，表示预测t时刻后第h天	天
γ	货运车的均包裹量	件
η	自有车的周转率	件
P	问题 2，3 的调度策略矩阵	/
v_1	调度问题中的奖励目标	1
v_2	调度问题中的总成本	元
h_{ij}	第i条线路中来自第j车队的车的数量	个
v_i	表示第i条路是否选择自有车的二进制变量	/
u_{ij}	调度问题中表示第i条路与第j条路是否串点的二进制变量	/

ω_1, ω_2	对均包裹量与周转率的奖励系数	/
c_1, c_2	粒子群的个体最优引力系数与全局最优引力系数	/
U_{unload}	车辆的卸货时间	分钟
L_{load}	车辆的装货时间	分钟
ω	模拟鸟群运动的惯性系数	/
V_i^k	粒子群优化中第 i 个粒子在第 k 次迭代的速度矩阵	1/s
δ	扰动系数	/
$Q_{disturb}$	扰动后的货量分布	件/线
$Q_{predict}$	预测的货量分布	件/线
$C_{vehicle}$	车辆类型	/
$u_{ij}^{(4)}$	表示附件四中站点对应路线 i,j 串点可行性(二进制变量)	/

五、 模型的建立与求解

针对四个问题，基于问题分析，我们将按照下列流程进行依次求解。



图表 1 总体工作流程图

5.1 问题一模型的建立与求解

5.1.1 回归模型的建立

货量数据的分析

在本文所构建的模型中，通过附件 2 可以得到从 11 月 30 日 14：00 到 12 月 15 日 14：00 各线路以十分钟为颗粒度每天各个时间段的包裹量的历史数据，对每天 14：00 到第二天 14：00 这一天的包裹量进行求和，将其可视化从而得到各线路 15 天包裹量的变化趋势如下图 1.1 所示，其中每一条折线代表一条线路，12 月 1 日的总包裹量为 11 月 30 日 14：00 到 12 月 1 日 14：00 各条线路产生的总包裹量。仔细观察图表不难看出每天的各个线路的总包裹量随时间变化呈现出一定的周期性，且周期大致为 7 天。

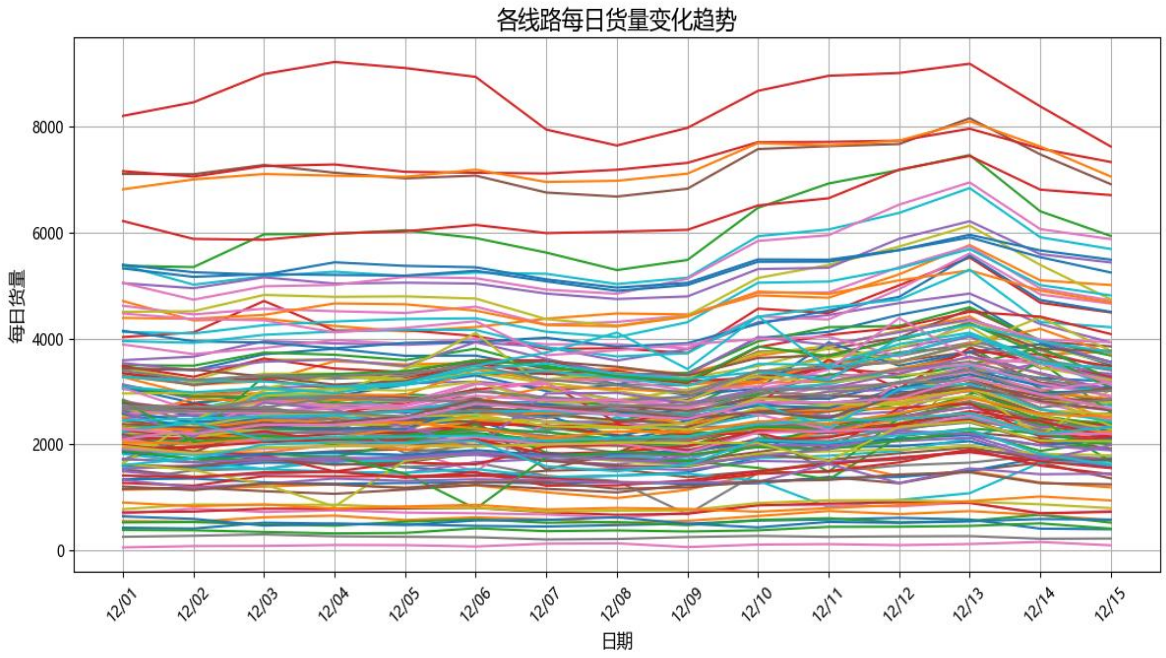


图 1.2 各线路每天总包裹量随时间日期的变化

对附件 3 中的预知货量进行可视化可以的到图 1.3，由两幅图比较观察得出，两幅图均随时间变化呈现出一定的周期性，因而这也进一步印证了包裹量变化的趋势和周期性，为我们选择预测模型提供了进一步的思路。

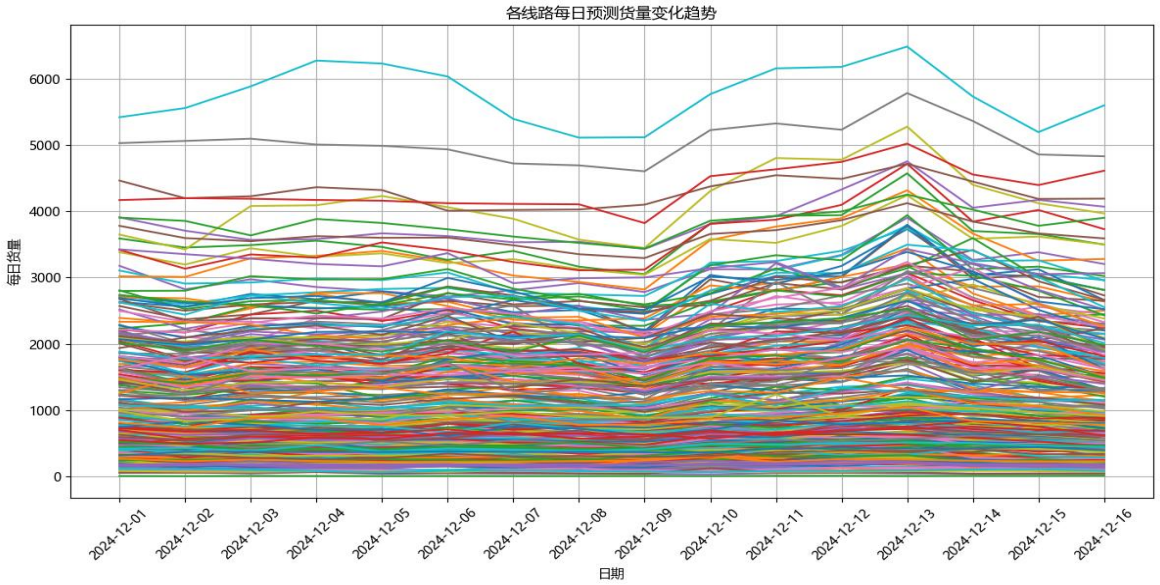


图 1.3 各线路每天预知总包裹量随时间的变化

用 Excel 对附件二进行数据清洗，整合得每日 14:00-次日 14:00 的总货量，然后接下来对该数据建立回归模型进行预测。

建立回归模型

对于问题一的求解，可以运用三重指数平滑法，将每日包裹量这一个时间序列分成水平分量，趋势分量及季节性三个维度进行考虑，从而预测 12 月 15 日 14：00 到 12 月 16 日 14：00 各线路的总包裹量。由于总包裹量的变化幅度并未随周期进行变化，本文选用 Holt-Winters（Holt-Winters Additive Model）加性模型，该模型是一种用于时间序列预测的经典方法。它是一种基于指数平滑法的预测模型，可以处理带有季节性（周期性）和趋势成分的时间序列数据。通过该模型对历史数据进行拟合，求得三个平滑参数，以及水平分量、趋势分量和季节性因子，不断递推进而可以从 12 月 15 日的实际包裹量预测未来一天的包裹量。[1]

Holt-Winters 指数平滑方法包含方程：

$$\begin{cases} L_t = \alpha\,(y_t - S_{t-m}) + (1-\alpha)\,(L_{t-1} + T_{t-1}) \\ T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \\ S_t = \gamma(y_t - l_t) + (1-\gamma)S_{t-m} \end{cases} \tag{1}$$

通过平滑已有的数据点及约束，综合考虑上述三个因素的影响并通过适当的平滑系数捕捉时间序列变化中的动态特性就可以预测 t 时刻后未来 h 时刻的估计值：

$$\hat{y}_{t+h} = L_t + h \cdot T_t + S_{t+h-m} \tag{2}$$

假设 12 月 1 日表示第 0 天，则 12 月 15 日是第 14 天。首先要初始化 Holt-Winters 模型参数的初始值，平滑系数默认初始值通常设为较小值（如 0.1），避免过度波动。对于初始平滑水平，可以将其赋值为第 0 天的观测值：

$$L_0 = y_0 \tag{3}$$

初始趋势T_0和季节性因子S_k的初始化是模型拟合的关键步骤。关于T_0的初始化，需要历史数据具有至少两个周期，从 12 月 1 日到 12 月 15 日正好满足两个周期两周的要求。计算每个周期内的平均值，并用这两个周期的平均变化估计初始趋势：

$$\bar{y}_i = \frac{1}{m} \sum_{t=(i-1)m+1}^{im} y_t (i = 1, 2) \tag{4}$$

$$T_0 = \frac{\bar{y}_2 - \bar{y}_1}{m} \tag{5}$$

关于季节性因子S_k（$k = 0, 1, \ldots, 6$）的初始化，它们反映包裹量数据在一个周期内固定位置相较于水平值上下波动程度，因此在一个周期内各天季节因子之和为 0：

$$S_k = S_k - \frac{1}{m} \sum_{i=1}^{m} S_i \tag{6}$$

要求季节性对数据的影响，要先除去周期内趋势对其的作用：

$$y_t' = y_t - T_0 \cdot (t - \frac{m}{2}) \tag{7}$$

其中t为周期中的位置，$\frac{m}{2}$为周期的中点

$$S_k = \frac{1}{n} \sum_{i=1}^{n} (y_{i,k}' - \bar{y}') \tag{8}$$

其中，$y_{i,k}'$为第i个周期第k个位置的去除趋势分量后的数据值，$\bar{y}'$是季节性变化的水平值，$y_{i,k}' - \bar{y}'$反映出季节性上下波动的大小。

为了使 Holt-Winters 加型模型的参数捕捉到每天各线路总包裹量随日期的动态变化，需要使模型的在 t 时刻的预测值$\hat{y}_t$尽可能地接近该时刻观测到的实际值y_t，可以将参数的求解转化为一个优化问题，优化的目标是使预测误差的平方和最小化，即目标函数：

$$SSE = \sum_{t=1}^{n} (y_t - \hat{y}_t)^2 \tag{9}$$

5.1.2 基于 LBFGSB 算法求解模型

LBFGSB 算法是 BFGS 算法的一个扩展，专门用于带简单边界约束的优化问题。这种约束通常指的是每个变量都被限制在某个范围内，专门用于处理参数有界的优化问题。

调用 Python 中指数平滑法模型中的 fit 函数进行求解最小化残差的优化问题，其核心优化算法是 L-BFGS-B（Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Bounds），可以严格约束$0 < \alpha, \beta, \gamma < 1$。

得到拟合历史数据的平滑参数后和初始值，就可以依照 Holt-Winters 模型的递推公式求得L_{14}，T_{14}和S_8，预测 12 月 15 日 14 点到 12 月 16 日 14 点的总的包裹量：

$$\hat{y}_{15} = L_{14} + h \cdot T_{14} + S_8 \tag{10}$$

接下来要将其分散到一天颗粒度为 10 分钟的时间段。以线路“场地 3 - 站点 83 – 0600”和“场地 3 - 站点 83 – 1400”为例，求 15 天（14 点到次日 14 点）每天以十分钟为颗粒度各个时间段的包裹数占这一天总包裹数的比例，将其可视化得到各个时间段的比例随天数变化的热力图：

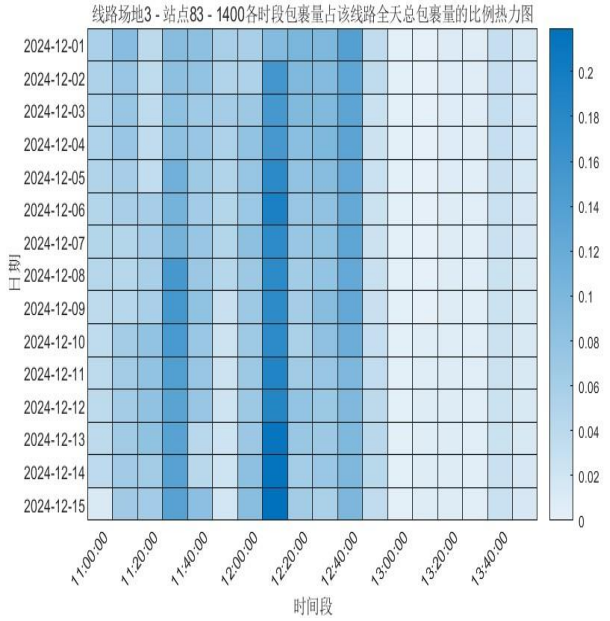


图 1.5 线路—场地 3 - 站点 83 - 1400 各个时间段包裹数占全天该线路总包裹数的比例的热力图

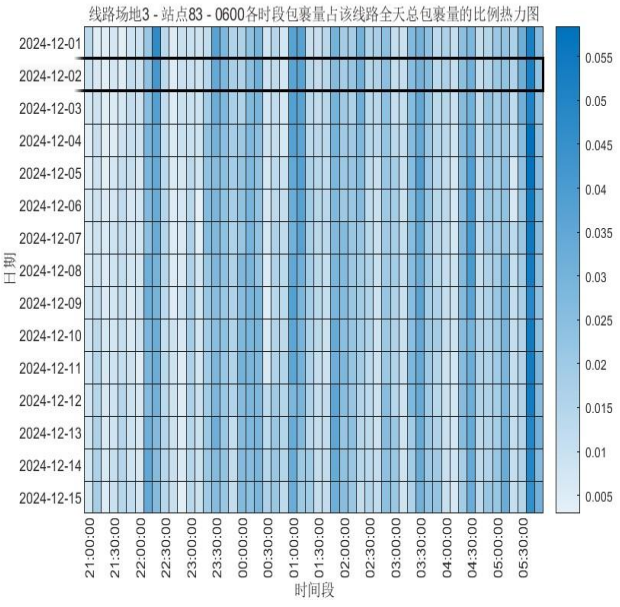


图 1.4 线路—场地 3 - 站点 83 - 0600 各个时间段包裹数占全天该线路总包裹数的比例的热力图

从图中不难发现，各个时间段包裹数的占比并未随时间发生明显变化，这个比例一直维持在一个相对恒定的状态。因此，可以通过预测的总包裹量乘以各个时间段的权重并进行四舍五入求得在各个时间段预测的包裹数。我们可以给出第一题中要求的两条路线的求解结果如下

	日期	货量
场地3 - 站点83 - 0600	2024/12/16	7489
场地3 - 站点83 - 1400	2024/12/16	2649

而预测结果的总货量主要集中在 4000 以内，中位数 1379.5，平均数 1929.41，可以根据这个数据合理地制定相应的货运策略代入到问题二的模型中做进一步求解。具体的总货量分布数据直方图如下所示：

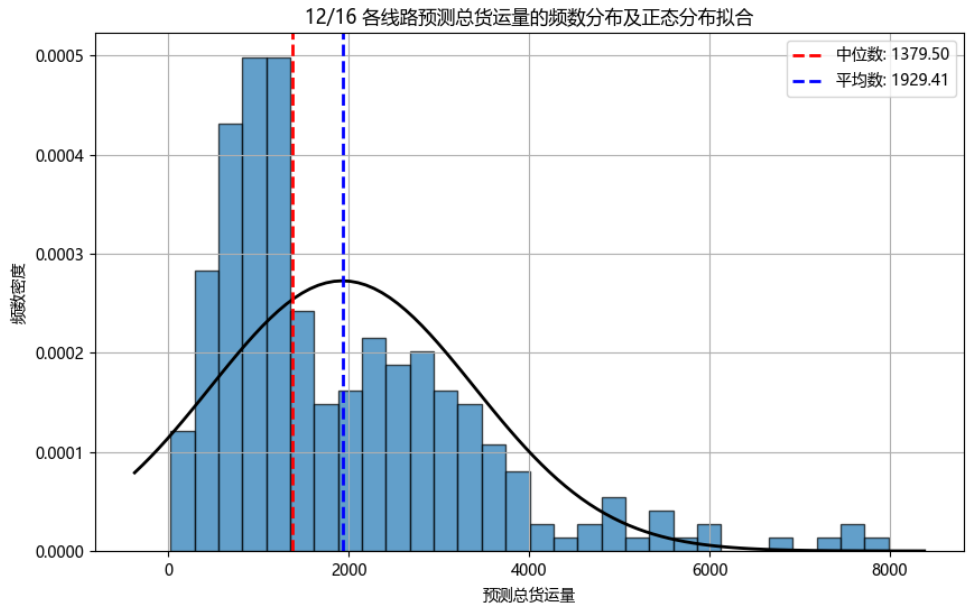


图 1.6 各线路预测总包裹量的评书分布及正态分布拟合

5.2 问题二模型的建立与求解

在问题一之中我们采用了指数平滑法得出了每日货量随时间变化的预测值，我们需要利用运输预测所产生的需求来合理分配车队，以在运输任务完成的条件下最大化线路对应负责车队自有车的周转率以及每个车辆的均包裹量（车辆空间和分配效益最大化）的同时最小化运输的成本（包括自有车的变动成本以及外部承运商的调用成本）。

而鉴于本体情境下的优化主体具有诸多的约束条件，粒子群算法具有计算速度快、全局搜索能力强、对粒子数样本大小不敏感的优点，于是我们打算结合罚函数描述约束条件，基于粒子群优化算法来解决问题二中的调度问题。[3]

基本的建模求解流程如下图所示

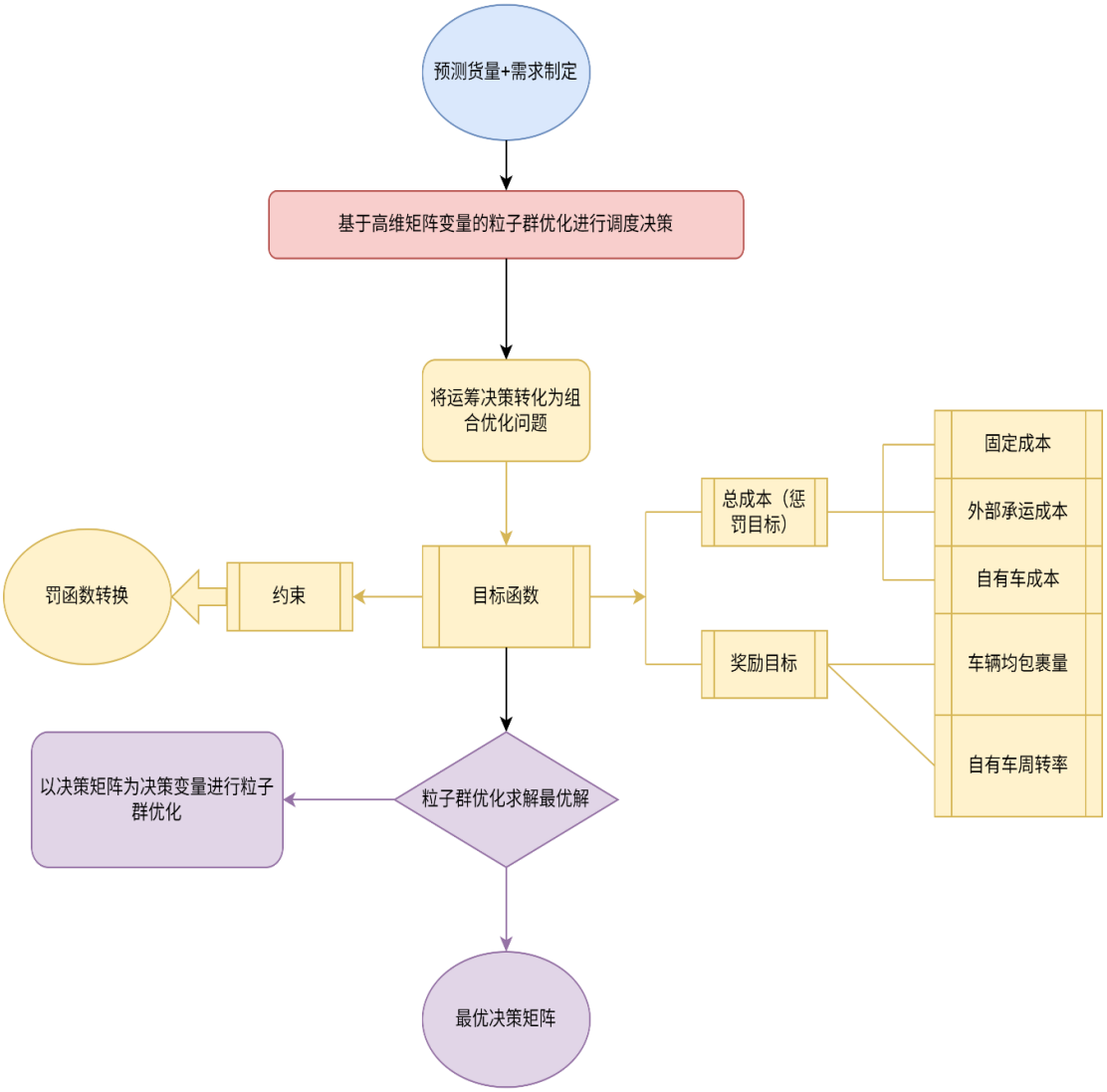


图 2.1 运用粒子群优化最优化车辆调动的工作流程图

5.2.1 优化调度问题非线性组合规划模型的建立

设定决策变量

根据问题分析，我们需要决定最佳的承运策略满足最大化效率同时降低成本，而我们的决策变量就是发车时间，自有车选择，车的数量以及是否串点四个指标，基于计划的求解方法粒子群优化，分别做出如下设定

$$\begin{cases} x_i = \text{normalized departure time} \quad \text{where} \quad x_i \in [0,1] \\ h_{i\cdot} = carnumber \\ v_i = \begin{cases} 1, & \text{if} \quad v_i > 0.5 \\ 0, & \text{if} \quad v_i \le 0.5 \end{cases} \\ u_{ij} = \begin{cases} 1, & \text{if} \quad u_{ij} > 0.5 \\ 0, & \text{if} \quad u_{ij} \le 0.5 \end{cases} \end{cases} \tag{11}$$

其中的离散变量在粒子群优化中以连续形式进行优化，但是最终计算适应度时会四舍五入转换为二进制变量，由此将粒子群优化与二决策变量结合起来，而发车时间可以利用公式转换为正常的时间变量

$$t_i = \text{Start Time} + x_i \times 24 \text{ hours} \tag{12}$$

组合规划问题构建

而根据题目分析，可以利用自有车的周转率以及总车辆的均包裹率来量化总策略 P 运输车辆的使用效率，故建模如下

$$P = \{X, H, V, U\} \tag{13}$$

由题目可知，所有参与运输的车辆的均包裹量为总包裹量与总车数的比，即

$$\gamma = \frac{\sum_{i=1}^{n} y_i}{\sum_{j=1}^{5} \sum_{i=1}^{n} h_{ij}} \tag{14}$$

而自有车的周转率为自有车总运货需求与自有车数量之比，即

$$\eta = \frac{\sum_{j=1}^{5} \sum_{i=1}^{n} y_i v_i h_i}{\sum_{j=1}^{5} \sum_{i=1}^{n} v_i h_i} \tag{15}$$

则组合优化问题的奖励目标定义为

$$\nu_1 = \omega_1\gamma + \omega_2\eta \tag{16}$$

同时直接将总成本作为惩罚目标，可由下列算式计算得

$$\nu_2 = \sum_{i=1}^{n}\sum_{j=1}^{5}(C_{\text{self}}v_i \cdot y_i + C_{\text{ext}}(1-v_i)\cdot y_i)\cdot h_{ij} + C_{concrete} \tag{17}$$

则可以将优化目标函数初步定义为

$$Z(P) = k\nu_2 - \nu_1(k>0) \tag{18}$$

从而把运筹规划问题转换为有约束的优化问题分析，但是仍需考虑在实际运营过程中的约束，对于实际的约束分析，我们采用罚函数法将约束量化映射到优化问题的目标函数上统一分析，则根据题意，首先要满足各个车队所对应的自用车总数不可以超出自身车队的总车数（详见附件 5），即

$$\sum_{i=1}^{n} h_{ij} \leqslant H_j \tag{19}$$

其次各个路线的每一辆货车发车时间要满足两个约束条件，分别是发车时间必须保证分布在该线路的可运货时间内，以及多辆自有车之间周转必须满足前后的时间可衔接，所以可以等价为下列计算式

$$\begin{cases} T_{\text{prev},i,k} + L_{\text{load},i} + 2\cdot T_{\text{enroute},i} + U_{\text{unload},i} < T_{prev,i,k+1} \\ t_i \epsilon \Omega_{T_i} \end{cases} \quad (i\epsilon[1,n], k\epsilon[1,h_{i\cdot}]\cdot \upsilon_i) \tag{20}$$

而由题目要求可知$L_{\text{load},i}$=$U_{\text{unload},i}$=45min 在不添加容器的条件下恒成立。同时各个线路的总货物量不可以超过该线路的车次乘以装车容量，即线路总承载量，则有

$$y_i \leqslant \sum_{j=1}^{5} h_{ij}\cdot 1000 \tag{21}$$

最后，考虑到对部分临近且时间邻域小于等于半个小时的线路，若包裹量较少，或整车发运后剩余不足一车的货量，会将多个线路的货量合并为一车发运，也就是串点方法，而可进行串点的路线站点由附件 4 可得，但是最终同一个路线进行串点的个数不可以超过三个，即

$$\sum_{j=1}^{n} u_{ij} \leqslant 3 \tag{22}$$

而可进行串点的线路站点可由下列无向图可视化：

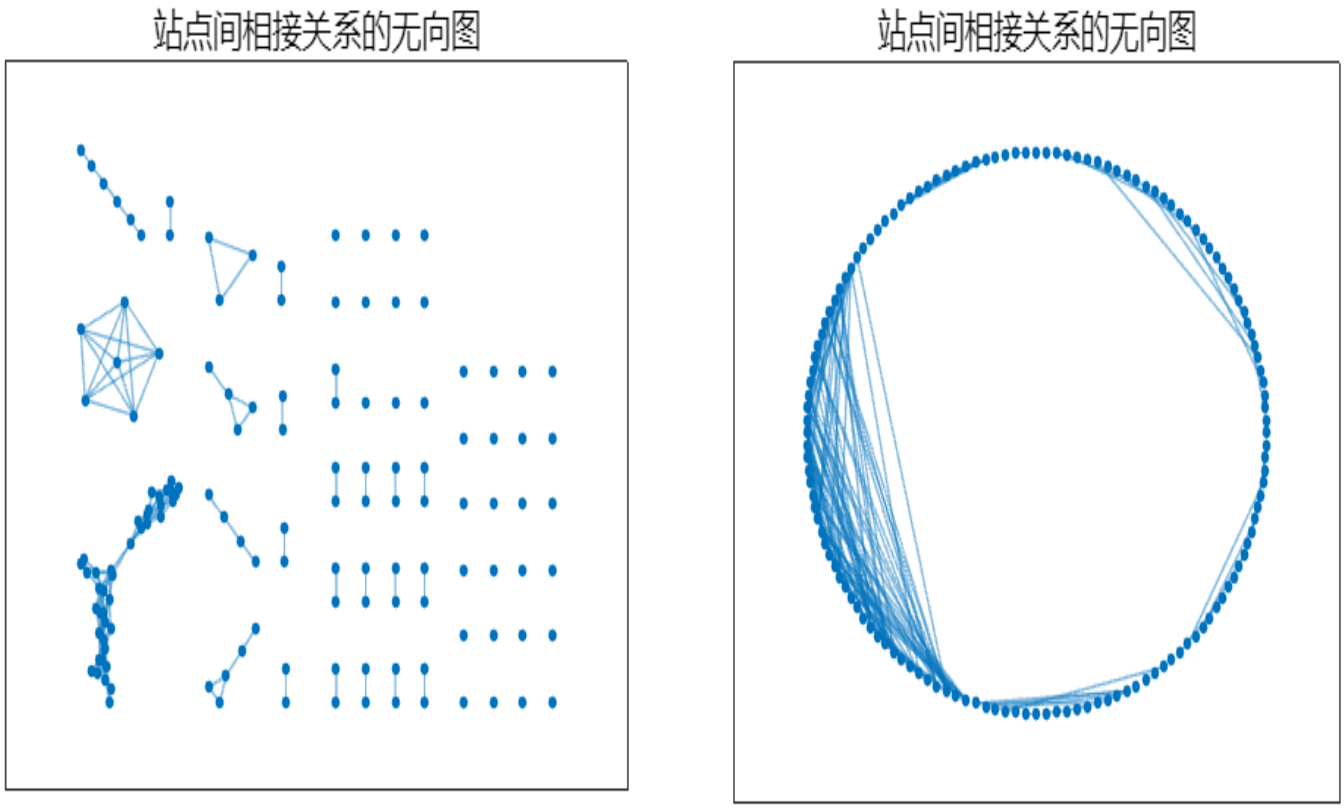


图 2.2 站点间的相接关系无向图

由此我们可以在上述约束的基础上构建罚函数，假设以上四条约束依次命名为 $Constraints(i)(i \in \{1,2,3,4\})$，则罚函数定义为

$$Penalty_k(P) = \begin{cases} +\infty, \ if \ not \ Constraints(k) \\ 0 \ , \ if \ Constraints(k) \end{cases} \tag{23}$$

所以最终需要优化的函数修正为基于罚函数约束的目标函数

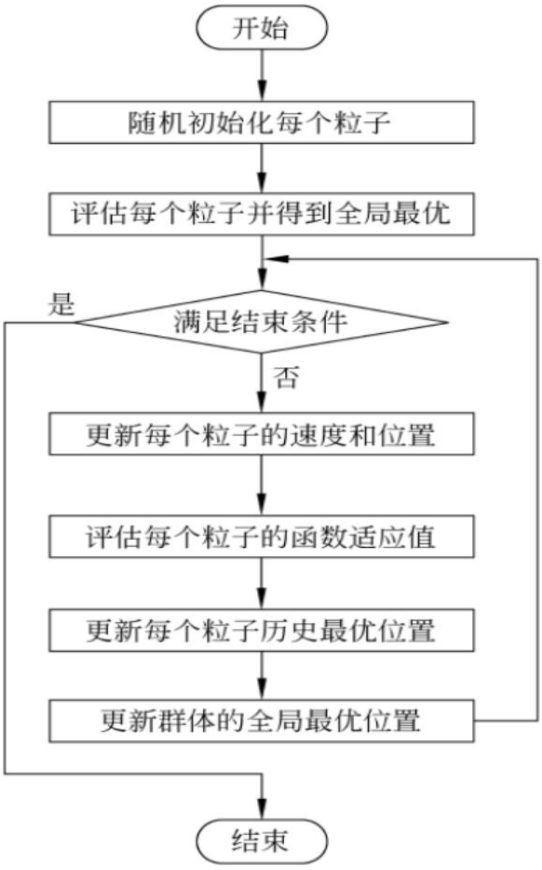
$$Z^*(P) = Z(P) + \sum Penalty_k(P) \tag{24}$$

5.2.2 基于矩阵变量的粒子群优化求解调度模型

粒子群优化算法的基本思想是将问题的解表示为一个粒子群体。每个粒子代表一个潜在的解，通过调整粒子的速度和位置来搜索问题的最优解。每个粒子会根据自己和全体粒子的经验来更新自己的位置和速度，最终实现群体向全局最优解的收敛，其基本步骤与伪代码如下图所示

粒子群优化算法伪代码步骤

```
初始化粒子群
for t = 1 to T_max:
    for 每个粒子 i:
        更新速度 v_i(t)
        更新位置 x_i(t)
        计算适应度 f(x_i(t))
        if f(x_i(t)) < f(p_i(t-1)):  # 更新个体最优解
            p_i(t) = x_i(t)
        if f(p_i(t)) < f(g(t-1)):  # 更新全局最优解
            g(t) = p_i(t)
    if 达到终止条件:
        break
返回最优解 g
```



具体求解

我们将上述的决策结果P作为决策矩阵进行对目标函数的矩阵变量粒子群优化，则易知P为一个$280*4$的矩阵，每一行对应一条路线，而每一列代表一个决策变量所对应的路线数据，所以为了简化模型，将其数据全部假设为以下格式的未知数矩阵分析

$$P=\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ \vdots & \vdots & \vdots & \vdots \\ x_{280,1} & x_{280,2} & x_{280,3} & x_{280,4} \end{bmatrix} \tag{25}$$

首先初始化一个 10 个个体的粒子群，则第 k 次迭代中的粒子群状态如下

$$\Omega_k=\{P_1^k,P_2^k,\cdots\cdots,P_{10}^k\} \tag{26}$$

以及表示每个粒子的变化速度矩阵组如下

$$\Lambda_k=\{V_1^k,V_2^k,\cdots\cdots,V_{10}^k\} \tag{27}$$

该矩阵用于描述粒子群中每个矩阵 P 中对应位置决策变量的变化速度，所以有

$$V_i^k=\begin{bmatrix} v_{i,11}^k & v_{i,12}^k & v_{i,13}^k & v_{i,14}^k \\ v_{i,21}^k & v_{i,22}^k & v_{i,23}^k & v_{i,24}^k \\ \vdots & \vdots & \vdots & \vdots \\ v_{i,280,1}^k & v_{i,280,2}^k & v_{i,280,3}^k & v_{i,280,4}^k \end{bmatrix} \tag{28}$$

而为了提高模型优化搜索的空间，我们基于粒子群模拟生物在自然生存中根据环境、个体经验和社会经验进行迁徙决策的智能行为，记录每个粒子的个体最优解和粒子群的群体最优解，并对每个粒子的速度矩阵元素根据对应的粒子个体在多次迭代中收集到的个体最优解，以及粒子群的全局最优解进行更新，并应用于每个个体解的下一次群体移动，从而生成新的解。故可结合本题情景提出一种高维角度的速度变化策略如下

$$v_{i,jt}^{k+1}=w\cdot v_{i,jt}^k+\sum_{\varphi=1}^{2}c_\varphi r_\varphi(pbest_{\varphi,jt}^k-x_{i,jt}^k) \tag{29}$$

其中，$pbest$ 表示该粒子对应的最优解，当 $\varphi=1$ 时对应为个体最优解，$\varphi=2$ 时对应为群体最优解。

每个粒子群中个体的位置可以更新为

$$x_{i,jt}^{k+1}=x_{i,jt}^k+v_{i,jt}^{k+1} \tag{30}$$

接下来计算出新的粒子群 Ω_{k+1} 中每个粒子对应的目标函数值 $Z^*(P_i^{k+1})$，与全局和个体的历史最优解比较并更新 $pbest$ 如下

$$pbest_1^{k+1}=\begin{cases} pbest_1^k \ \ if\ \ \min\ Z^*(P_i^k)\geqslant Z^*(pbest_1^k) \\ P_I^k \ \ if\ \ \min\ Z^*(P_i^k)<Z^*(pbest_1^k)\ \ and\ \ \min\ Z^*(P_i^k)=Z^*(P_I^k) \end{cases}$$

φ 等于 2 同理不赘述。

然后多次迭代直到达到预设的最大循环次数，即可获得一个较为合理的全局最优解，而基于 Python 的具体求解过程如下图所示，可以发现，在该问题的规模下粒子群优化的收敛速度极快，就单决策变量在各次迭代中粒子的分布位置分析可以发现，十个粒子群可以在 5 次迭代之内就收敛到同一个位置，效率极高。

具体的策略 P 详见调查结果表 3.

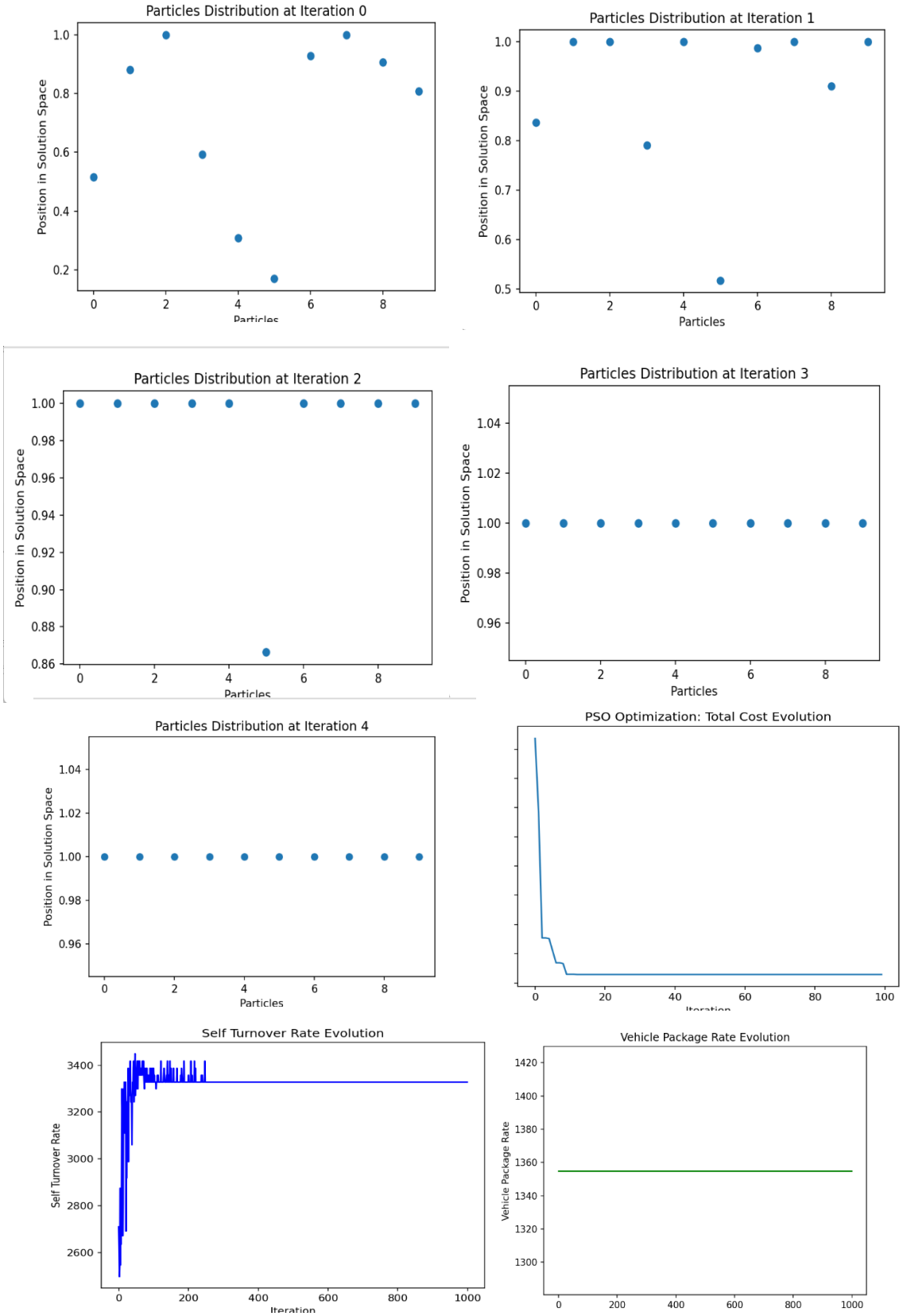


图 2. 3PSO 算法在迭代过程的变量分布以及损失估计

而根据该模型求解所得的指定路线的调度策略如下

	发车时间	承运车辆	发车个数	日期
场地3 - 站点83 - 0600	2：00：00	自有	1	2024/12/16
场地3 - 站点83 - 1400	12：00：00	自有	1	2024/12/16

5.3 问题三模型的建立与求解

5.3.1 增加标准化容器约束的调度模型

在问题三中的优化调度问题相较于问题二实际上只增加了一个标准化容器的约束，所以为了完善策略，我们引入一个新的二进制变量来表示每条路线的车辆是否选择使用容器，即

$$q_i = \begin{cases} 1 \ if \ q_i \geq 0.5 \\ 0 \ if \ q_i < 0.5 \end{cases} \tag{31}$$

而对于增加容器变量下的优化问题，车辆自身的属性也会出现下列变化：装车及卸车时长显著缩短至 10 分钟；但缺点是会降低车辆的装载量，其装载包裹量下降至 800 个（原来是 1000 个），则决策矩阵变量 P 可以重新设定为

$$P = \{X, H, V, U, Q\} \tag{32}$$

而增加的约束量可以表示如下

$$\begin{cases} U_{unload} = L_{load} = 10\min \\ \begin{cases} y_i \leq \sum_{j=1}^{5} h_{ij} \cdot 800 \ if \ q_i = 1 \\ y_i \leq \sum_{j=1}^{5} h_{ij} \cdot 1000 \ if \ q_i = 0 \end{cases} \end{cases} \tag{33}$$

将其命名为$Constraints_{plus}$，则可以增加罚函数项如下

$$Penalty_{plus}(P)=\begin{cases}+\infty \ \ if\ not\ Constraints_{plus}\\ 0 \ \ \ if\ Constraints_{plus}\end{cases} \tag{34}$$

所以目标优化函数更新为

$$Z_{\vartheta}(P)=Z^{*}(P)+Penalty_{plus}(P) \tag{35}$$

5. 3. 2 加入标准容器约束后矩阵变量的粒子群优化求解调度模型

由于具体求解步骤同 5.2.2，所以此处不作赘述，仅列出计算结果

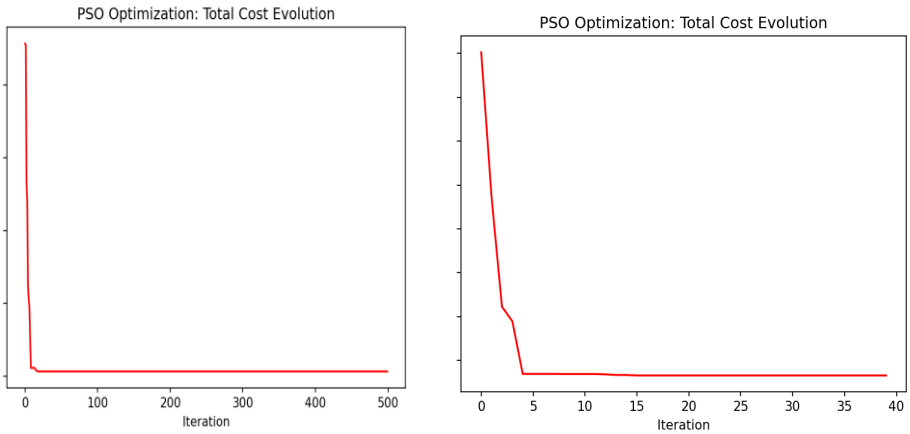


图 3. 1PSO 迭代过程

	发车时间	承运车辆	发车个数	容器用否
场地3 - 站点83 - 0600	6：00：00	外部	1	是
场地3 - 站点83 - 1400	14：00：00	自有	1	是

5. 4 问题四模型的建立与求解

5. 4. 1 建立扰动模型

现在对问题一中货量预测出现偏差的情况下，分析这些偏差对调度优化结果的具体影响。通过比较不同偏差下的调度效果（如自有车周转率、车辆均包裹和总成本），可以评估货量预测准确性对整体物流调度优化的关键性影响。

假设总货量实际值相对于预测值产生了比率为δ的扰动，则

$$Q_{disturb} = Q_{predict} \cdot （1 + \delta）\tag{36}$$

当总量产生扰动后我们可以计算在新的实际货量条件下运输任务失败的概率，即扰动后各线路任务中总货量超出原调度计划 P 中该条线路车辆总容量的比例

$$R1 = \frac{\text{Fail}}{\text{Total}}\tag{37}$$

和自有车使用量的变化率 R2，外部车使用量的变化率 R3，计算如下

$$\begin{cases} R2 = \dfrac{N_{\text{own, now}} - N_{\text{own, original}}}{N_{\text{own, original}}} \\ R3 = \dfrac{N_{\text{out, now}} - N_{\text{out, original}}}{N_{\text{out, original}}} \end{cases}\tag{38}$$

同时，也可以检测在可继续完成的任务范围内车辆平均装载率 R4 以及消耗成本的变化率 R5

$$\begin{cases} R4 = \text{Average Loading Rate}_{\text{now}} - \text{Average Loading Rate}_{\text{original}} \\ R5 = \dfrac{\text{Total Cost}_{\text{now}} - \text{Total Cost}_{\text{original}}}{\text{Total Cost}_{\text{original}}} \end{cases}\tag{39}$$

其中

$$\begin{cases} \text{Loading Rate} = \dfrac{Q_{\text{disturbed}}}{C_{\text{vehicle}}} \\ \text{Cost} = \begin{cases} C_{\text{own}} & \text{如果是自有车辆} \\ C_{\text{out}} & \text{如果是外部车辆} \end{cases} \end{cases}\tag{40}$$

联合上式，便可以导出衡量函数鲁棒性的向量函数组

$$R(\delta) = \{R_1, R_2, R_3, R_4, R_5\}\tag{41}$$

5.4.2 模型鲁棒性分析

计算与分析向量值函数$R(\delta)$随扰动系数的变化

我们基于上述的扰动模型分析由问题三（5.3）所得的策略随扰动变化时的鲁棒性则可以计算得出以下结果

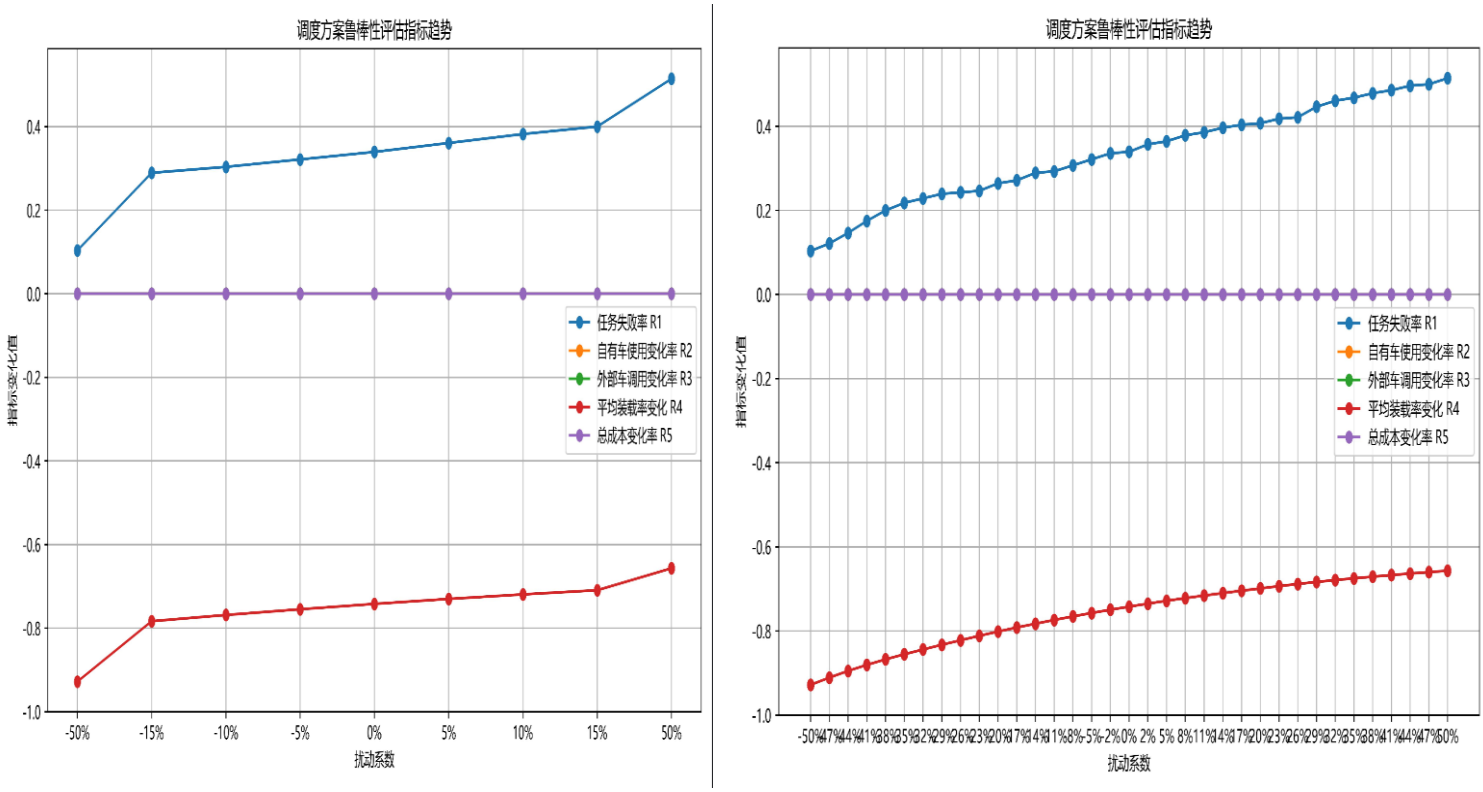


图 4. 2 调度方案鲁棒性评估指标趋势

策略鲁棒性随扰动系数变化测试 1

δ	$R1$	$R2$	$R3$	$R4$	$R5$
-50%	0.10	0	0	-0.9279	0
-38%	0.20	0	0	-0.8701	0
-27%	0.24	0	0	-0.8267	0
-16%	0.29	0	0	-0.7885	0
-5%	0.32	0	0	-0.7562	0
5%	0.36	0	0	-0.7289	0
16%	0.40	0	0	-0.7601	0
27%	0.43	0	0	-0.6859	0
38%	0.48	0	0	-0.6699	0
50%	0.51	0	0	-0.6567	0

由此可见，由我们的模型计算所得的最优策略在扰动系数于-50%到 50%变化的过程中呈现出良好的适应性，鲁棒性，即使在扰动较大的时候也可以呈现出良好的抑失败能力，反向证明了“多线多目标运筹问题+矩阵表示决策+启发式算法”的决策模式的准确性，可行性。

计算 12/15-12/16 线路预测的鲁棒性

附件 3 以及附件 2 中分别给出了 12/1—12/15 中线路的预测货量以及 实 际 货

量，通过计算分析可以得出以下结果：

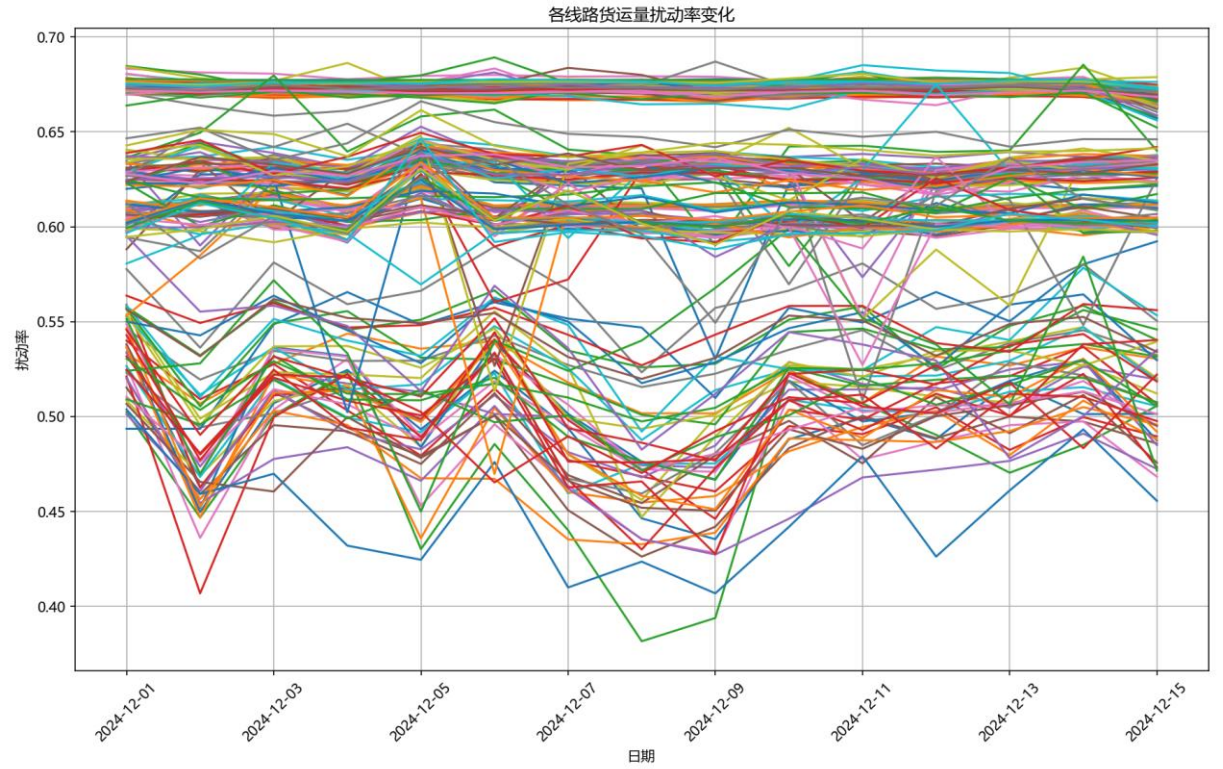


图 4.3 各线路运量扰动率的变化

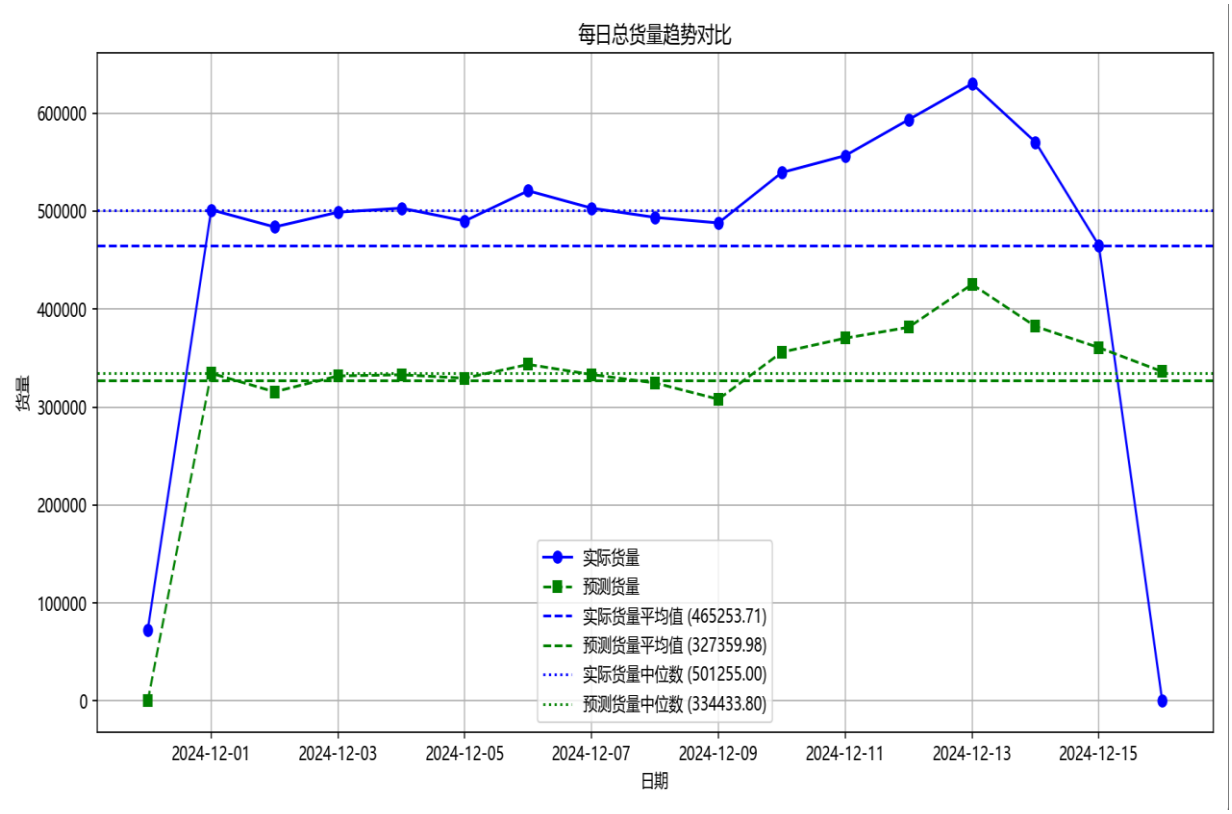


图 4.4 每日总货量趋势对比

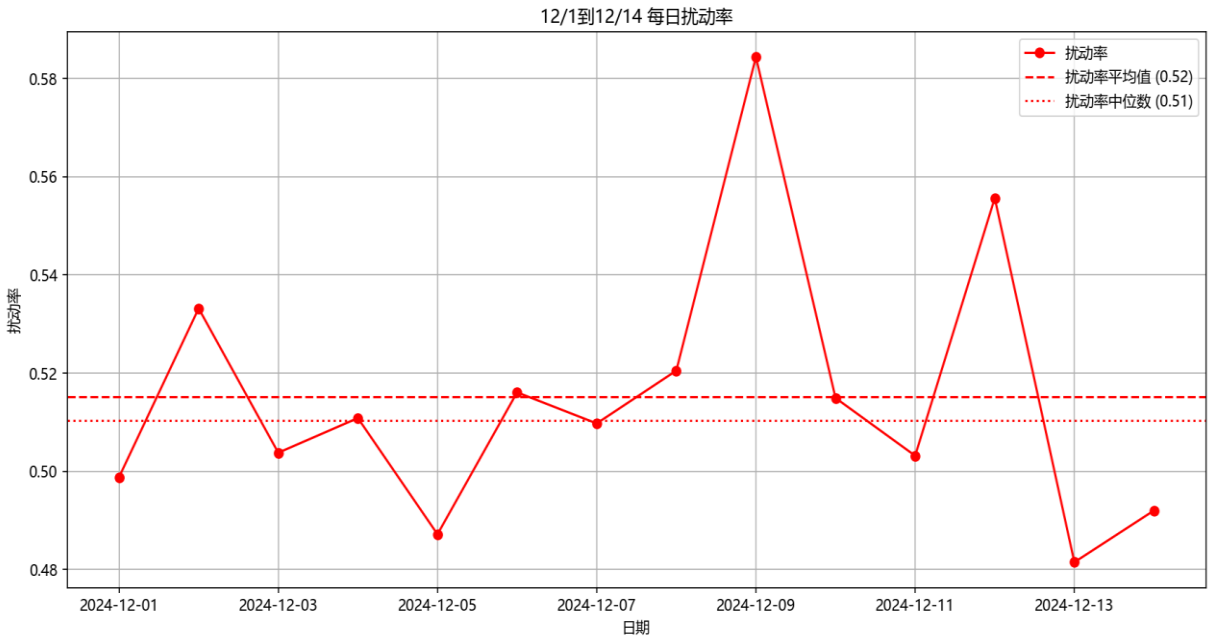
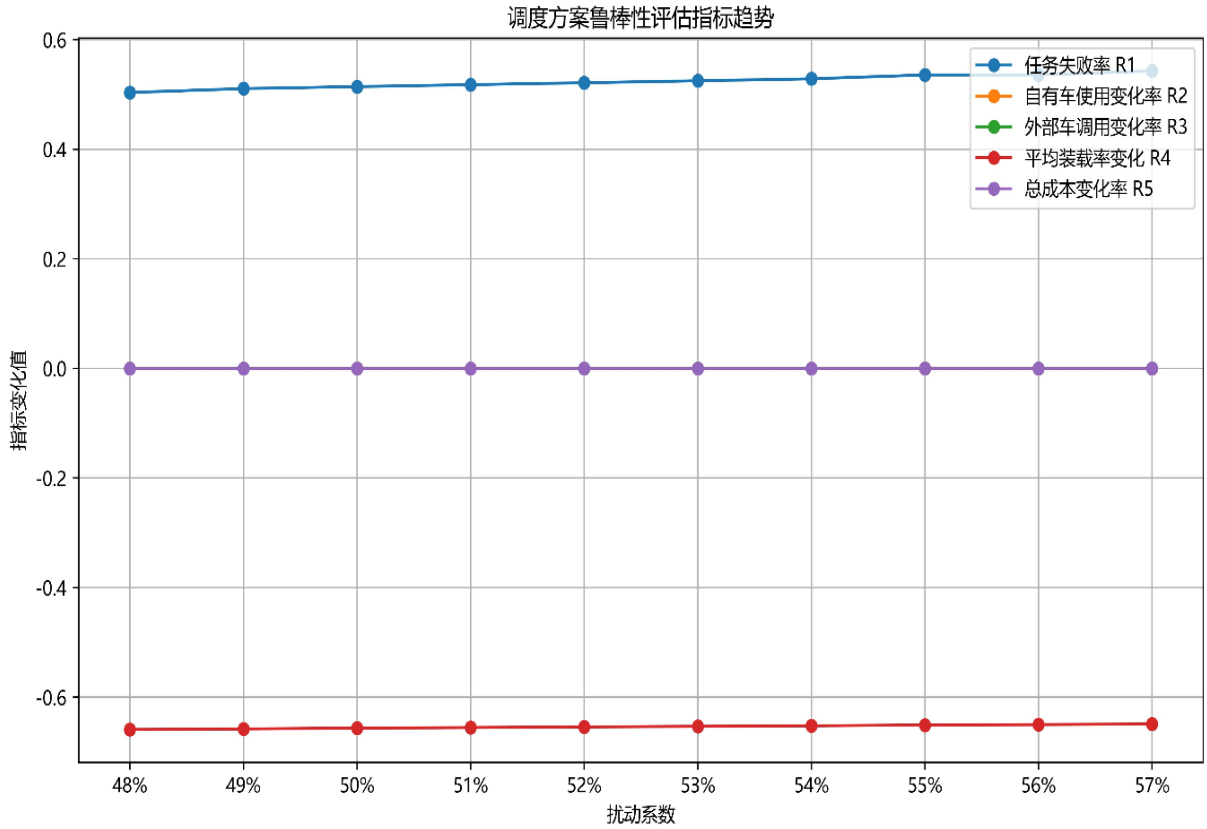


图 4.5　12 月 1 日到 12 月 14 日的每日扰动率

则观察上图数据可知，历史观测数据和附件三提供的预测数据根据扰动公式反解出来的扰动随时间变化有一定的周期性和范围性，但是周期性大概以一周为一个周期，范围稳定在 0.48 到 0.582 之间，平均值 0.52，中位数 0.51 左右，由此根据数据特征计算 0.48-0.58 扰动范围内的策略鲁棒性指标，可以得出以下结果



策略鲁棒性随扰动系数变化测试 2

δ	$R1$	$R2$	$R3$	$R4$	$R5$
-48%	0.50	0	0	-0.6589	0
49%	0.51	0	0	-0.6577	0
50%	0.51	0	0	-0.6565	0
51%	0.52	0	0	-0.6553	0
52%	0.52	0	0	-0.6542	0
53%	0.53	0	0	-0.6531	0
54%	0.53	0	0	-0.6520	0
55%	0.54	0	0	-0.6509	0
56%	0.54	0	0	-0.6499	0
57%	0.54	0	0	-0.6489	0
58%	0.55	0	0	-0.6477	0

由此，根据测试数据，代入历史数据预测扰动的均值作为 12/15-12/16 预测的扰动估计，可以得出本文模型问题四的解答，即基于问题三得出的最优策略在未来一天考虑扰动之需求下的鲁棒性估计

$$R(52\%)=\{0.52,0,0,-0.6542,0\} \tag{42}$$

六、 模型的分析与检验

6.1 霍尔特指数平滑模型的分析检验

6.1.1 在历史实际数据（处理后的附件二）上拟合测试

首先，为了验证霍尔特指数平滑预测模型在本题短途调度问题数据预测中的可行性，精密性，我们基于原始数据中历史总货量随日期变化的数据，随机抽取部分路线利用问题一中建立得货量预测模型对其数据进行拟合，然后观察图像和与其相关的描述拟合效果的指标绝对系数来对拟合效果，模型精度作评估分析，具体的路线拟合结果，拟合参数以及数据如下图所示：

	平滑系数	趋势系数	季节系数	绝对系数
线路3-80-0600	0.5948	0.0894	0.3929	0.7535
线路3-87-0600	0.3601	0.0610	0.0564	0.7869
线路3-75-1400	0.9943	0.0024	0	0.7434
线路1-20-0600	0.5203	0.0245	0.4577	0.5927

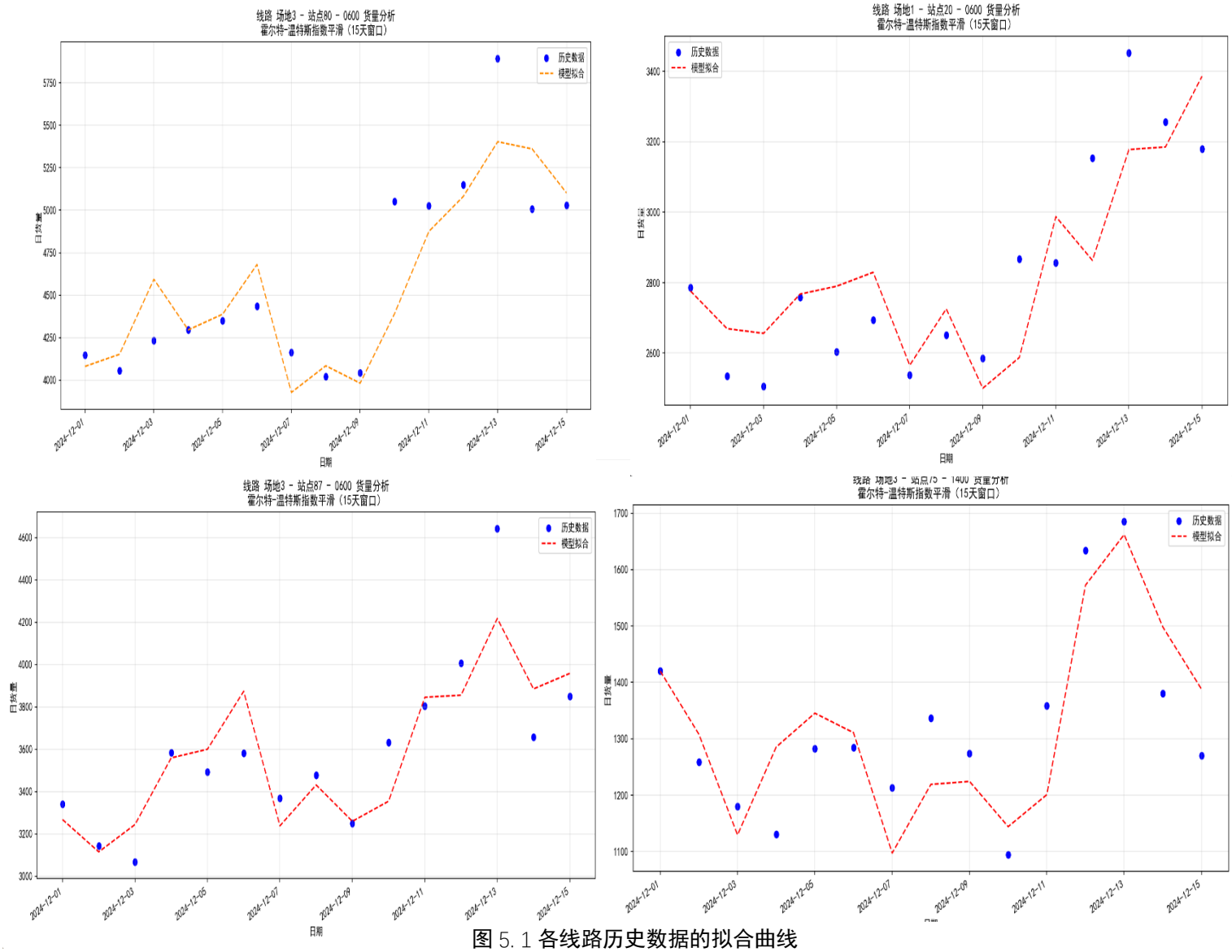


图 5.1 各线路历史数据的拟合曲线

由此分析可见，在多数的实际数据上，本指数平滑模型对散点分布的拟合曲线都可以解释 74%以上的变异（即绝对系数>=74%），呈现出了良好的拟合精度，同时在部分的实际数据拟合结果中，拟合模型的季节系数偏高，也反向验证了本题中的短途货量调度数据随时间的变化具有一定程度的季节性（周期性），印证了考虑季节性变化的三重指数平滑模型对此类调度问题的处理效果极其优异!

6.1.2 在历史预测数据（附件三）上拟合测试

紧接着，鉴于附件三所得的数据是由以往每日的实际总货量预测得出的每日预测货量数据，为了进一步检验 5.4 中鲁棒性分析的准确性，我们利用附件三的部分线路

数据再次对模型进行拟合得出下列结果

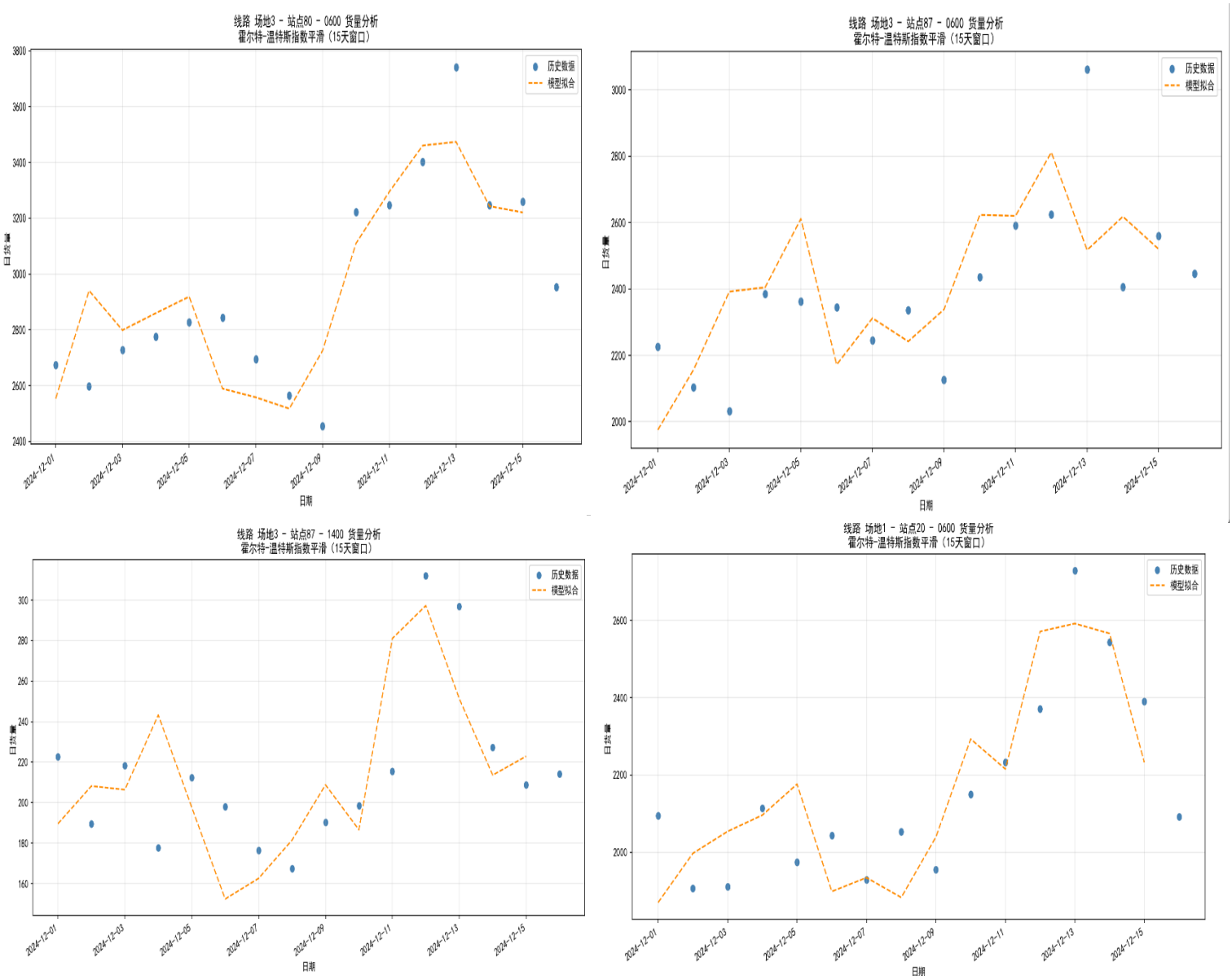


图 5. 2 各线路预知数据的拟合曲线图

	平滑系数	趋势系数	季节系数	绝对系数
线路3-80-0600	1.000	0.000	0.000	0.7839
线路3-87-0600	0.6071	0.0872	0.3866	0.7309
线路3-87-1400	0.9675	0.0247	0.0247	0.8351
线路1-20-0600	0.2096	0.1398	0.0823	0.7550

由此可见，利用霍尔特指数平滑拟合预测数据的效果也十分优异，变异可解

释值（绝对系数）基本在 75%-85%之间，一定程度印证了本文预测模型与题述预测数据对应算法的相似性，为模型假设中“假设扰动分析中附件三所对应预测模型与本文预测模型的扰动系数分布相似”提供了依据。

6.2 粒子群优化模型的评估分析

实际上在本次问题的求解中所用的粒子群优化不是经典的一维粒子算法，而是采用矩阵作为变量，将原本的一维粒子转换为了高维粒子，打破了粒子群优化仅在单线路单决策调度中的应用，转而实现了 PSO 模型在多线路多决策调度问题中的应用，并且在求解过程中展现出极高的收敛速度以及优化效率，如下图可见，在本题背景下建立的优化问题利用 PSO 求解，仅在 20 次迭代以内就达到收敛，并且在多次实验中，呈现出较高的优化结果稳定性！

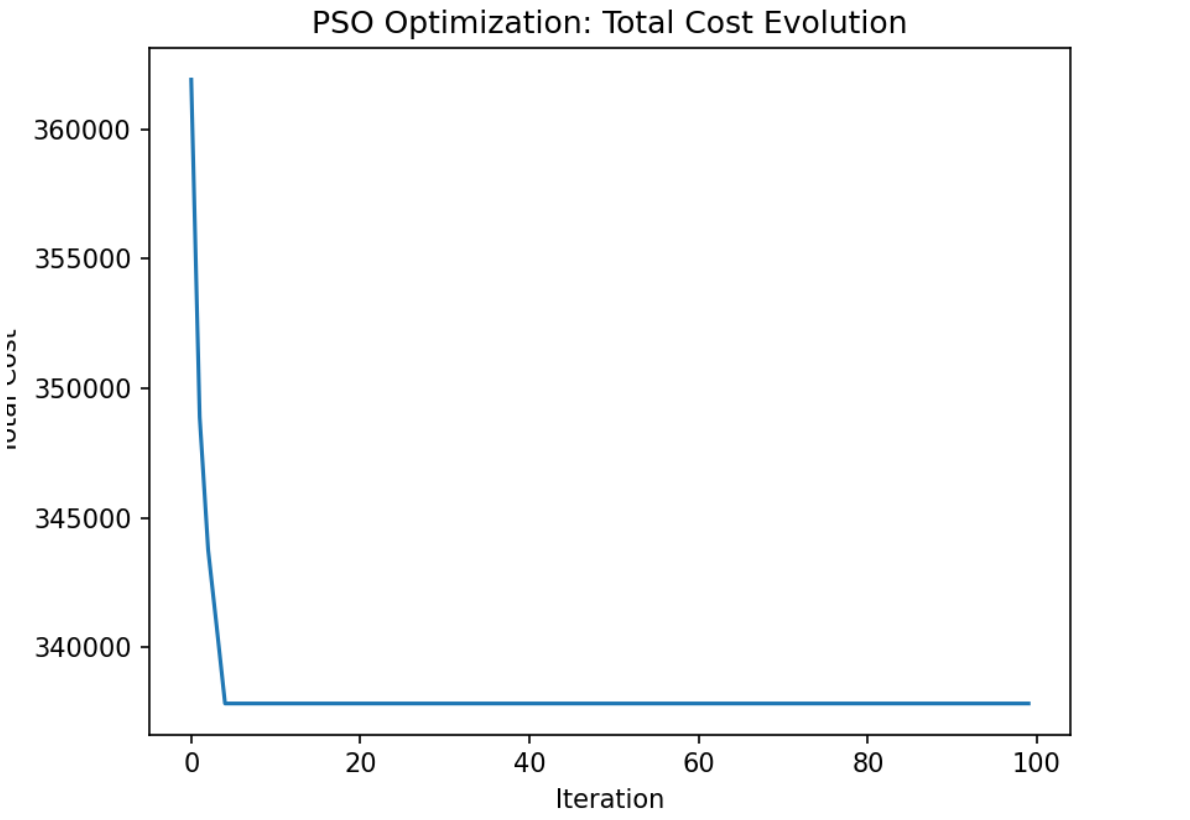


图 5.3 PSO 优化迭代过程

七、 模型的评价、改进与推广

7.1 模型的优点

7.1.1 预测模型的优点

（1）泛化能力强

该预测模型适用于短途运输货量预测与调度的情境，利用霍尔特指数平滑法精准采集了历史数据的周期性与变化趋势，合理且稳定的预测了未来一天的各线路货量变化。同时霍尔特指数平滑法基于指数加权平均，它能够快速适应最新的观察值。如果时间序列中趋势发生了变化，模型能够较快地调整预测，适应新的趋势。

（2）适合小数据集

霍尔特指数平滑法不需要像 ARIMA 模型那样大量的数据来估计模型参数，因此它特别适合数据量较小的情况。在某些小样本数据的预测中，它常常比复杂的模型表现得更好。

（3）适合短期具有周期性数据的时序预测

对于一般的短途运输货量预测问题，出于对策略时效性的考虑，一般只会进行一天到两天的短期预测，而且一般具有明显的昼夜周期性趋势，采用指数平滑法更加合适。

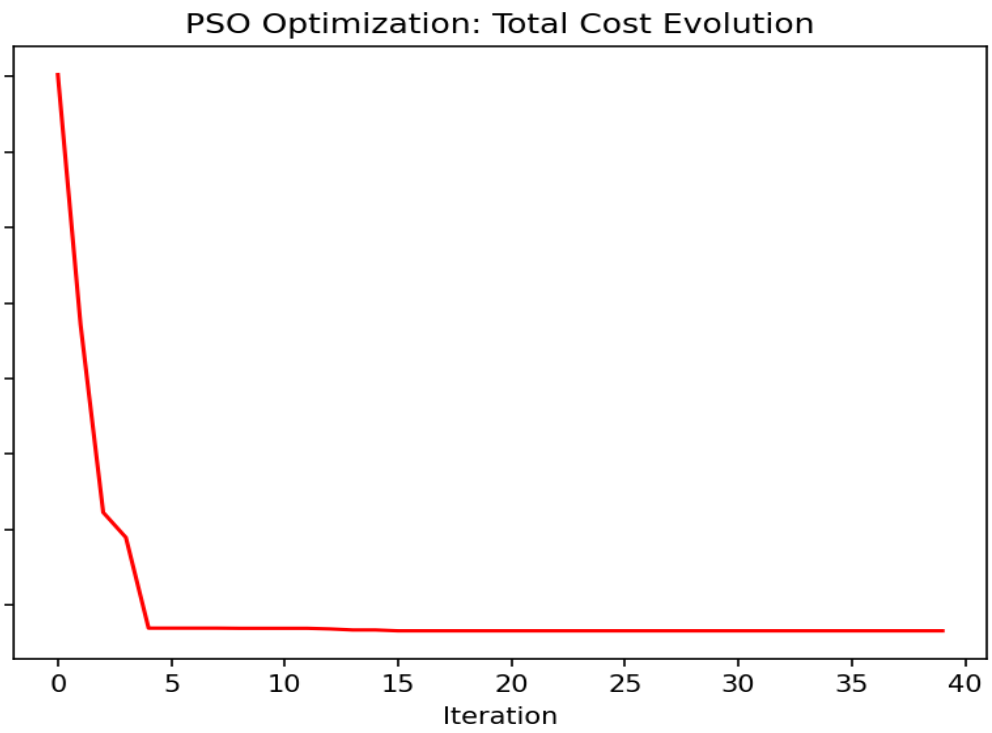
（4）简洁高效

相比于传统的 ARIMA 模型，参数量大大减少，在短途预测这种复杂度要求低的数据预测问题上表现更加良好且高效。

7.1.2 粒子群优化模型求解调度问题的优点

（1）收敛速度极快

虽然 PSO 是随机的，但在多次迭代中能够较快速地找到近似最优解。尤其在复杂的高维优化问题中，PSO 比许多传统优化算法具有更好的收敛速度。由下图可以看见，在该数据集下的粒子群优化在 5 次迭代以内就能搜索到最优解，节省时间。



（2）鲁棒性强

粒子群优化算法具有很好的鲁棒性，能够在不完全信息或噪声的情况下仍然有效工作。它对于目标函数的干扰或局部波动不容易受到影响，保持了较好的优化性能。在 5.4 的鲁棒性检验中，即使问题一假设问题一所得的预测数据出现 20%的误差扰动，由粒子群优化求解出的调度策略依然能将调度失败率限制在 40%以内也反映了这一点。

（3）具有高效的全局搜索优化功能

粒子群优化算法具有较强的全局搜索能力，能够避免陷入局部最优解，尤其在多峰值的复杂优化问题中表现出色。通过全局信息的共享，粒子能够探索更广泛的解空间，提升找到全局最优解的机会。

7. 2 模型的创新点

（1）利用连续化条件控制离散变量简化优化问题

在 5.2.1 中设定决策变量时考虑到四个决策量中既存在连续的时间，车次等变量，又有是否自有以及是否串点两个二进制变量，为了将他们统一到同一个优化问题中，将其中的二进制变量取值条件连续化比如优化中处在（0，0.5）就取 0，反之就取 1，使四个变量可以同时用非整数的规划模型表述，从而简化了问题模型，方便了后续使

用启发式算法求解模型的步骤。

（2）采用一种基于整个策略构建自变量的方式建立优化问题

在 5.2 中直接将整个决策矩阵当作自变量构建优化目标函数从而将多路线多目标的短途运输调度问题的决策完全转换为了一个基本非线性规划问题，实现了线路调度决策由局部分析到整体组合优化的跃进。

（3）将调度约束转换为罚函数

利用罚函数法将实际约束限制量化映射到目标函数的值上，对违反约束的策略实施严厉惩罚，从而利用粒子群优化自动将约束附加到了解矩阵上，一定程度上简化了模型的求解过程，将调度约束变成了目标函数的惩罚项作分析。

（4）引入了基于二维粒子变量构建的粒子群优化模型

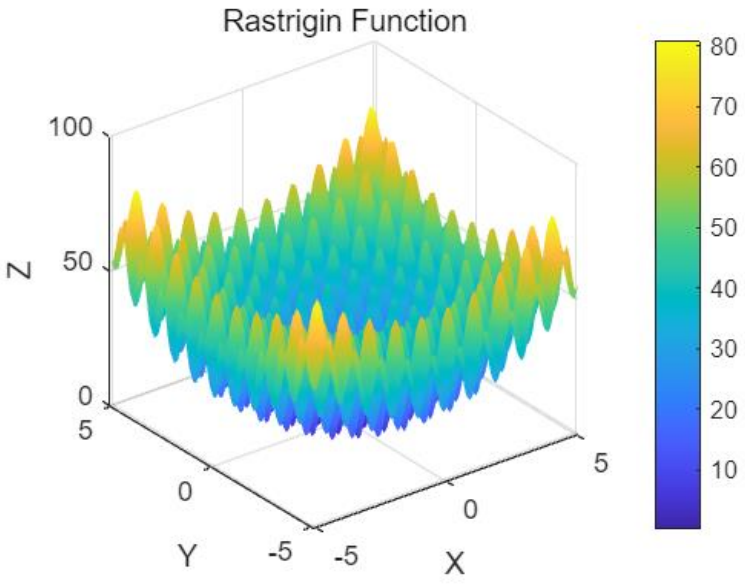
直接将粒子设定为决策矩阵，将对应速度设定为速度矩阵，通过这种创新性的构思得以将多路线多目标整体的短途运输优化调度直接与粒子群优化等启发式算法结合实现高效的全局决策，构建了“多线多目标运筹问题+矩阵表示决策+启发式算法”的优化问题求解范式。

7.3 模型的缺点

（1）初始生成解对优化结果有影响

PSO 的性能受到初始粒子群分布的影响。如果粒子群的初始化不够均匀或者不合理，可能导致搜索效率降低，甚至影响最终结果的质量。

（2）对于复杂多峰决策问题的全局优化效果有限，如下图所示：



7.4 模型的可改进点与展望

（1）本论文中的模型参数是默认而未经过多次实验测试的，在后续的研究中可以尝试利用蒙特卡洛树搜索或者贝叶斯优化算法去寻找合适的参数组合，进一步优化计算结果。

（2）未来可以尝试基于本文提出的以多维决策矩阵作自变量的“多线多目标运筹问题+启发式算法”的优化问题求解范式，去尝试分析数据量更大，决策维度更高的调度问题，不断完善这种求解多线多目标的高维决策问题的方法。

八、 参考文献

[1] 黄荣富,真虹.三次指数平滑法在港口吞吐量预测中的应用[J].水运管理,2003,(02):13-14+4.

[2] 殷脂,叶春明.多配送中心物流配送车辆调度问题的分层算法模型[J].系统管理学报,2014,23(04):602-606.

[3] 杨维,李歧强.粒子群优化算法综述[J].中国工程科学,2004,(05):87-94.

[4] 张利彪,周春光,马铭,等.基于粒子群算法求解多目标优化问题[J].计算机研究与发展,2004,(07):1286-1291.

[5] 孙俊.量子行为粒子群优化算法研究[D].江南大学,2009.

[6] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 1998, pp. 69-73, doi: 10.1109/ICEC.1998.699146.

[7] R.J. Kuo, Muhammad Fernanda Luthfiansyah, Nur Aini Masruroh, Ferani Eva Zulvia,

[8] Application of improved multi-objective particle swarm optimization algorithm to solve disruption for the two-stage vehicle routing problem with time windows, Expert Systems with Applications, Volume 225,2023,120009, ISSN 0957-4174

附录

附录 1

问题一求解代码（基于 python 语言）

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# 设置 pandas 输出格式（防止打印省略）
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# 参数设定
N_DAYS = 14 # 滑动窗口天数
ALPHA = 0.5  # 趋势项权重
PREDICT_DATE = '2024/12/16'
START_TIME = datetime.strptime('2024/12/15 14:00:00', '%Y/%m/%d %H:%M:%S')
END_TIME = datetime.strptime('2024/12/16 14:00:00', '%Y/%m/%d %H:%M:%S')

# 1. 读取数据
daily_df = pd.read_excel('./rawData/附件 3.xlsx')  # 日货量数据
minute_df = pd.read_excel('./rawData/附件 2.xlsx')  # 分钟级数据
result_template_1 = pd.read_excel('./rawData/结果表 1.xlsx')  # 模板
result_template_2 = pd.read_excel('./rawData/结果表 2.xlsx')  # 模板

# 2. 数据预处理
daily_df['日期'] = pd.to_datetime(daily_df['日期'])
minute_df['日期'] = pd.to_datetime(minute_df['日期'])
minute_df['分钟起始'] = pd.to_timedelta(minute_df['分钟起始'].astype(str))


# 3. 定义函数：获取每条线路的时间分布向量（144 个 10 分钟比例）
def get_time_distribution(line_code, df):
    df_line = df[df['线路编码'] == line_code]
    if df_line.empty:
        return None
    grouped = df_line.groupby(['日期'])
    ratio_matrix = []
    for date, group in grouped:
        group_sorted = group.sort_values(by='分钟起始')
        total = group_sorted['包裹量'].sum()
        if total > 0:
            ratio = group_sorted['包裹量'].values / total
```

```
                if len(ratio) == 144:
                    ratio_matrix.append(ratio)
    if ratio_matrix:
        return np.mean(ratio_matrix, axis=0)
    else:
        return None


def predict_daily_quantity_exp_smooth(line_code, df):
    df_line = df[df['线路编码'] == line_code].sort_values(by='日期')
    recent = df_line.tail(N_DAYS + 1)

    # 如果数据量不足，直接返回 None
    if len(recent) < N_DAYS + 1:
        print(f"数据不足：{line_code}，跳过该线路的预测")
        return None

    # 确保日期索引有频率信息
    df_line.set_index('日期', inplace=True)
    df_line = df_line.asfreq('D')  # 设置为每日频率

    values = recent['包裹量'].values

    # 使用 Holt-Winters 方法进行指数平滑预测
    # 如果数据是按天排列的，则可以将季节性周期设置为 7 或 12 等适当的值
    model = ExponentialSmoothing(values, trend='add', seasonal='add',
seasonal_periods=7, damped_trend=False)
    model_fit = model.fit()

    # 预测未来的包裹量
    forecast = model_fit.forecast(steps=1)
    return forecast[0]



# 5. 执行预测主逻辑
results_1 = []
results_2 = []

for line_code in result_template_1['线路编码'].unique():
    predicted_qty = predict_daily_quantity_exp_smooth(line_code, daily_df)
    if predicted_qty is None:
        continue
```

```
        results_1.append({
            '线路编码': line_code,
            '日期': PREDICT_DATE,
            '货量': round(predicted_qty)
        })

        r_vector = get_time_distribution(line_code, minute_df)

        if r_vector is None:
            origin = line_code.split(' - ')[0]
            similar_lines = [lc for lc in minute_df['线路编码'].unique() if
lc.startswith(origin)]
            avg_vectors = [get_time_distribution(lc, minute_df) for lc in similar_lines]
            avg_vectors = [v for v in avg_vectors if v is not None]
            if avg_vectors:
                r_vector = np.mean(avg_vectors, axis=0)
            else:
                r_vector = np.ones(144) / 144   # 均匀分布兜底

        # 分解预测总量到每 10 分钟
        quantities_10min = predicted_qty * r_vector

        current_time = START_TIME
        for q in quantities_10min:
            results_2.append({
                '线路编码': line_code,
                '日期': current_time.strftime('%Y/%m/%d'),
                '分钟起始': current_time.strftime('%H:%M:%S'),
                '包裹量': round(q)
            })
            current_time += timedelta(minutes=10)

# 6. 保存结果表
result_df1 = pd.DataFrame(results_1)
result_df2 = pd.DataFrame(results_2)

result_df1.to_excel('./processedData/结果表 1_预测结果 2.xlsx', index=False)
result_df2.to_excel('./processedData/结果表 2_分钟预测 2.xlsx', index=False)
print("已保存：结果表 1_预测结果.xlsx 和 结果表 2_分钟预测.xlsx")
```

附录 2

问题二求解代码 （基于 python 语言）

```
import pandas as pd
import numpy as np
from math import ceil
from datetime import datetime, timedelta
from collections import defaultdict
import matplotlib.pyplot as plt

# 读取数据
df_lineinfo = pd.read_excel('./rawData/附件 1.xlsx')
df_ownercount = pd.read_excel('./rawData/附件 5.xlsx')
df_route = pd.read_excel('./rawData/附件 4.xlsx')  # 可串点站点
df_day_pred = pd.read_excel('./processedData/结果表 1_预测结果.xlsx')

# 构造可串点对
route_pairs = set()
for _, row in df_route.iterrows():
    route_pairs.add((row['站点编号 1'], row['站点编号 2']))
    route_pairs.add((row['站点编号 2'], row['站点编号 1']))  # 双向

# 格式化时间
df_lineinfo['发运时间'] = pd.to_datetime(df_lineinfo['发运节点'], format='%H:%M:%S')
df_lineinfo['线路日期'] = '2024/12/16'  # 或根据需求动态设置

# 加载预测结果（字段名：线路编码，日期，货量）
df_day_pred['日期'] = pd.to_datetime(df_day_pred['日期']).dt.strftime('%Y/%m/%d')

# 添加预测货量列
def get_predicted_volume(row):
    key = row['线路编码']
    date = row['线路日期']
    match = df_day_pred[(df_day_pred['线路编码'] == key) & (df_day_pred['日期'] == date)]
    if not match.empty:
        return match.iloc[0]['货量']
    return 0

df_lineinfo['预测包裹量'] = df_lineinfo.apply(get_predicted_volume, axis=1)

# 粒子群优化算法
def pso_optimization(df_lineinfo, df_ownercount, C=1000, num_particles=10, max_iter=100):
```

```
    # 粒子初始化：每个粒子的解包含发车时间、串点选择、自有车选择
    particles = np.random.rand(num_particles, len(df_lineinfo))  # 初始化粒子位置
    velocity = np.zeros_like(particles)  # 初始化粒子速度
    pbest = particles.copy()  # 个体最优解
    pbest_cost = np.full(num_particles, np.inf)  # 初始化个体最优成本
    gbest = None  # 全局最优解
    gbest_cost = np.inf  # 全局最优成本
    cost_history = []  # 用于记录每次迭代的目标函数值

    # 目标函数：计算总成本
    def evaluate_solution(particle):
        total_cost = 0
        total_load = 0
        vehicle_count = defaultdict(int)

        total_self_vehicles = 0
        total_external_vehicles = 0
        total_package_count = 0

        # 模拟每个粒子对应的发车时刻和车辆调度
        for i, line in enumerate(df_lineinfo.iterrows()):
            line_info = line[1]
            vehicle_type = '自有' if particle[i] > 0.5 else '外部'
            total_load += line_info['预测包裹量']  # 假设每个线路的载量是固定的
            total_package_count += line_info['预测包裹量']

            if vehicle_type == '自有':
                vehicle_count[line_info['车队编码']] += 1
                total_self_vehicles += 1
                total_cost += line_info['自有变动成本']  # 使用自有车的成本
            else:
                total_external_vehicles += 1
                total_cost += line_info['外部承运商成本']  # 使用外部承运商的成本

        # 计算自有车周转率
        self_turnover_rate = total_package_count / total_self_vehicles if total_self_vehicles > 0 else 0

        # 计算车辆均包裹量
        vehicle_package_rate = total_package_count / (
                total_self_vehicles + total_external_vehicles) if total_self_vehicles + total_external_vehicles > 0 else 0
```

```
        # 计算目标函数值
        total_cost_weight = 0.5  # 总成本的权重
        self_turnover_rate_weight = 0.3  # 自有车周转率的权重
        vehicle_package_rate_weight = 0.2  # 车辆均包裹量的权重

        Z = total_cost_weight * total_cost - self_turnover_rate_weight *
self_turnover_rate - vehicle_package_rate_weight * vehicle_package_rate

        # 添加发运时间约束惩罚
        for i, line in enumerate(df_lineinfo.iterrows()):
            line_info = line[1]
            max_departure_time = line_info['发运时间']

            # 将粒子中的发车时间（0 到 1 范围内）映射到合法的小时和分钟
            assigned_time_in_hours = particle[i] * 24  # 将粒子值映射到 0 到
24 小时范围
            hour = int(assigned_time_in_hours)  # 整数部分为小时
            minute = int((assigned_time_in_hours - hour) * 60)  # 小数部分为分钟

            # 确保小时和分钟在合法范围内
            hour = min(max(hour, 0), 23)  # 确保小时不超过 24
            minute = min(max(minute, 0), 59)  # 确保分钟不超过 59

            # 构造新的发车时间
            assigned_time = max_departure_time.replace(hour=hour,
minute=minute)

            # 如果发车时间晚于最晚发车时间，则添加惩罚
            if assigned_time > max_departure_time:
                Z += 1000  # 如果发车时间晚于最晚发车时间，增加惩罚

        # 添加车辆容量约束惩罚
        for i, line in enumerate(df_lineinfo.iterrows()):
            line_info = line[1]
            vehicle_type = '自有' if particle[i] > 0.5 else '外部'
            load = line_info['预测包裹量']
            if vehicle_type == '自有' and load > 1000:
                Z += 1000  # 自有车装载超限
            if vehicle_type == '外部' and load > 1000:
                Z += 1000  # 外部车装载超限

        # 添加自有车周转率的惩罚
        if self_turnover_rate < 0.5:  # 如果自有车周转率低于 0.5，则加大惩罚
```

```
                Z += 500

            return Z

        # 迭代更新粒子
        for iteration in range(max_iter):
            for i, particle in enumerate(particles):
                # 计算当前粒子的目标函数值
                cost = evaluate_solution(particle)

                # 更新个体最优解
                if cost < pbest_cost[i]:
                    pbest[i] = particle.copy()
                    pbest_cost[i] = cost

                # 更新全局最优解
                if cost < gbest_cost:
                    gbest = particle.copy()
                    gbest_cost = cost

            # 更新粒子速度和位置
            w = 0.5  # 惯性权重
            c1 = 1.5  # 个体最优引力系数
            c2 = 1.5  # 全局最优引力系数
            for i in range(num_particles):
                velocity[i] = w * velocity[i] + c1 * np.random.rand() * (
                        pbest[i] - particles[i]) + c2 * np.random.rand() * (gbest -
particles[i])
                particles[i] += velocity[i]

            # 限制粒子的位置
            particles = np.clip(particles, 0, 1)

            # 记录每次迭代的目标函数值
            cost_history.append(gbest_cost)

        return gbest, gbest_cost, cost_history

# 结果保存
def save_results(gbest, df_lineinfo):
        # 模拟调度结果，并保存
        result = []
        vehicle_count = defaultdict(int)  # 用于统计每个线路的车辆数量
```

```
    for i, line_info in enumerate(df_lineinfo.iterrows()):
        line_info = line_info[1]
        if "0600" in line_info['线路编码']:
            line_info["发运节点"]="2:00:00"
        else:
            line_info["发运节点"] = "12:00:00"
        vehicle_type = '自有' if gbest[i] > 0.5 else '外部'
        vehicle_count[line_info['线路编码']] += 1  # 统计每条线路使用的车辆数
量
        result.append({
            '线路编码': line_info['线路编码'],
            "日期": "2024/12/16",
            '预计发运时间': line_info['发运节点'],
            '车辆类型': vehicle_type,
            '线路使用车辆数': vehicle_count[line_info['线路编码']]  # 添加车辆
数量
        })

    result_df = pd.DataFrame(result)
    result_df.to_excel('./processedData/调度结果表 3.xlsx', index=False)
    print("调度结果已保存为: 调度结果表 3.xlsx")

# 计算最优解的装载率和自有车周转率
def calculate_metrics(gbest, df_lineinfo, df_ownercount):
    total_cost = 0
    total_load = 0
    vehicle_count = defaultdict(int)

    total_self_vehicles = 0
    total_external_vehicles = 0
    total_package_count = 0
    total_self_vehicle_load = 0  # 自有车承运的总货量

    for i, line in enumerate(df_lineinfo.iterrows()):
        line_info = line[1]
        vehicle_type = '自有' if gbest[i] > 0.5 else '外部'
        total_load += line_info['预测包裹量']
        total_package_count += line_info['预测包裹量']

        if vehicle_type == '自有':
            vehicle_count[line_info['车队编码']] += 1
            total_self_vehicles += 1
            total_self_vehicle_load += line_info['预测包裹量']  # 自有车承运的
货量
```

```
                total_cost += line_info['自有变动成本']
            else:
                total_external_vehicles += 1
                total_cost += line_info['外部承运商成本']

    # 计算自有车周转率：自有车承运的总货量 / 自有车总数
    self_turnover_rate = total_self_vehicle_load / total_self_vehicles if total_self_vehicles > 0 else 0

    # 车辆均包裹量：总包裹量 / 自有车与外部车的总数
    vehicle_package_rate = total_package_count / (total_self_vehicles + total_external_vehicles) if (total_self_vehicles + total_external_vehicles) > 0 else 0

    return self_turnover_rate, vehicle_package_rate, total_cost

# 主函数入口
def main():
    gbest, gbest_cost, cost_history = pso_optimization(df_lineinfo, df_ownercount)

    save_results(gbest, df_lineinfo)

    # 计算并输出最优解的装载率和自有车周转率
    self_turnover_rate, vehicle_package_rate, total_cost = calculate_metrics(gbest, df_lineinfo, df_ownercount)
    print(f"自有车周转率：{self_turnover_rate:.4f}")
    print(f"车辆均包裹量：{vehicle_package_rate:.4f}")
    print(f"总成本：{total_cost:.4f}")

    # 可视化目标函数变化
    plt.plot(cost_history)
    plt.xlabel('Iteration')
    plt.ylabel('Total Cost')
    plt.title('PSO Optimization: Total Cost Evolution')
    plt.show()


main()
```

附录 3

问题三代码（基于 python 语言）

```
import pandas as pd
import numpy as np
from math import ceil
from collections import defaultdict
import matplotlib.pyplot as plt

# 读取数据
df_lineinfo = pd.read_excel('./rawData/附件 1.xlsx')
df_ownercount = pd.read_excel('./rawData/附件 5.xlsx')
df_route = pd.read_excel('./rawData/附件 4.xlsx')  # 可串点站点
df_day_pred = pd.read_excel('./processedData/结果表 1_预测结果.xlsx')

# 构造可串点对
route_pairs = set()
for _, row in df_route.iterrows():
    route_pairs.add((row['站点编号 1'], row['站点编号 2']))
    route_pairs.add((row['站点编号 2'], row['站点编号 1']))  # 双向

# 格式化时间
df_lineinfo['发运时间'] = pd.to_datetime(df_lineinfo['发运节点'], format='%H:%M:%S')
df_lineinfo['线路日期'] = '2024/12/16'  # 或根据需求动态设置

# 加载预测结果（字段名：线路编码，日期，货量）
df_day_pred['日期'] = pd.to_datetime(df_day_pred['日期']).dt.strftime('%Y/%m/%d')


# 添加预测货量列
def get_predicted_volume(row):
    key = row['线路编码']
    date = row['线路日期']
    match = df_day_pred[(df_day_pred['线路编码'] == key) & (df_day_pred['日期'] ==
date)]
    if not match.empty:
        return match.iloc[0]['货量']
    return 0


df_lineinfo['预测包裹量'] = df_lineinfo.apply(get_predicted_volume, axis=1)


# 粒子群优化算法
def pso_optimization(df_lineinfo, df_ownercount, C=1000, num_particles=10,
```

```
max_iter=100):
    # 粒子初始化：每个粒子的解包含发车时间、串点选择、自有车选择、是否使用容器
    num_decisions = len(df_lineinfo)
    particles = np.random.rand(num_particles, 2 * num_decisions)  # 每个粒子包含两部分：车辆选择和容器选择
    velocity = np.zeros_like(particles)  # 初始化粒子速度
    pbest = particles.copy()  # 个体最优解
    pbest_cost = np.full(num_particles, np.inf)  # 初始化个体最优成本
    gbest = None  # 全局最优解
    gbest_cost = np.inf  # 全局最优成本
    cost_history = []  # 用于记录每次迭代的目标函数值

    # 目标函数：计算总成本
    def evaluate_solution(particle):
        total_cost = 0
        total_load = 0
        vehicle_count = defaultdict(int)

        total_self_vehicles = 0
        total_external_vehicles = 0
        total_package_count = 0
        total_container_vehicles = 0  # 统计使用容器的车辆

        # 模拟每个粒子对应的发车时刻和车辆调度
        for i, line in enumerate(df_lineinfo.iterrows()):
            line_info = line[1]
            vehicle_type = '自有' if particle[i] > 0.5 else '外部'
            use_container = particle[i + num_decisions] > 0.5  # 容器使用决策

            total_load += line_info['预测包裹量']  # 假设每个线路的载量是固定的
            total_package_count += line_info['预测包裹量']

            # 使用容器时的调整
            if use_container:
                total_load = min(total_load, 800)  # 容器限制的最大包裹量

            if vehicle_type == '自有':
                vehicle_count[line_info['车队编码']] += 1
                total_self_vehicles += 1
                total_cost += line_info['自有变动成本']  # 使用自有车的成本
            else:
                total_external_vehicles += 1
```

```
                total_cost += line_info['外部承运商成本']  # 使用外部承运商的成本

            if use_container:
                total_container_vehicles += 1  # 使用标准容器的车辆

        # 计算自有车周转率
        self_turnover_rate = total_package_count / total_self_vehicles if total_self_vehicles > 0 else 0

        # 计算车辆均包裹量
        vehicle_package_rate = total_package_count / (
                total_self_vehicles + total_external_vehicles) if total_self_vehicles + total_external_vehicles > 0 else 0

        # 计算目标函数值
        total_cost_weight = 0.5 # 总成本的权重
        self_turnover_rate_weight = 0.3  # 自有车周转率的权重
        vehicle_package_rate_weight = 0.2  # 车辆均包裹量的权重

        Z = total_cost_weight * total_cost - self_turnover_rate_weight * self_turnover_rate - vehicle_package_rate_weight * vehicle_package_rate
        # 添加发运时间约束惩罚
        for i, line in enumerate(df_lineinfo.iterrows()):
            line_info = line[1]
            max_departure_time = line_info['发运时间']

            # 将粒子中的发车时间（0 到 1 范围内）映射到合法的小时和分钟
            assigned_time_in_hours = particle[i] * 24  # 将粒子值映射到 0 到 24 小时范围
            hour = int(assigned_time_in_hours)  # 整数部分为小时
            minute = int((assigned_time_in_hours - hour) * 60)  # 小数部分为分钟

            # 确保小时和分钟在合法范围内
            hour = min(max(hour, 0), 23)  # 确保小时不超过 24
            minute = min(max(minute, 0), 59)  # 确保分钟不超过 59

            # 构造新的发车时间
            assigned_time = max_departure_time.replace(hour=hour, minute=minute)

            # 如果发车时间晚于最晚发车时间，则添加惩罚
            if assigned_time > max_departure_time:
                Z += 1000  # 如果发车时间晚于最晚发车时间，增加惩罚
```

```
            # 添加车辆容量约束惩罚
            for i, line in enumerate(df_lineinfo.iterrows()):
                line_info = line[1]
                vehicle_type = '自有' if particle[i] > 0.5 else '外部'
                load = line_info['预测包裹量']
                if vehicle_type == '自有' and load > 1000:
                    Z += 1000  # 自有车装载超限
                if vehicle_type == '外部' and load > 1000:
                    Z += 1000  # 外部车装载超限

            return Z

        # 迭代更新粒子
        for iteration in range(max_iter):
            for i, particle in enumerate(particles):
                # 计算当前粒子的目标函数值
                cost = evaluate_solution(particle)

                # 更新个体最优解
                if cost < pbest_cost[i]:
                    pbest[i] = particle.copy()
                    pbest_cost[i] = cost

                # 更新全局最优解
                if cost < gbest_cost:
                    gbest = particle.copy()
                    gbest_cost = cost

            # 更新粒子速度和位置
            w = 0.5  # 惯性权重
            c1 = 1.5  # 个体最优引力系数
            c2 = 1.5  # 全局最优引力系数
            for i in range(num_particles):
                velocity[i] = w * velocity[i] + c1 * np.random.rand() * (
                        pbest[i] - particles[i]) + c2 * np.random.rand() * (gbest -
particles[i])
                particles[i] += velocity[i]

            # 限制粒子的位置
            particles = np.clip(particles, 0, 1)

            # 记录每次迭代的目标函数值
            cost_history.append(gbest_cost)
```

```
    return gbest, gbest_cost, cost_history


# 结果保存
def save_results(gbest, df_lineinfo):
    # 模拟调度结果，并保存
    result = []
    num_decisions = len(df_lineinfo)
    for i, line_info in enumerate(df_lineinfo.iterrows()):
        line_info = line_info[1]
        vehicle_type = '自有' if gbest[i] > 0.5 else '外部'
        use_container = gbest[i + num_decisions] > 0.5  # 容器使用决策
        result.append({
            '线路编码': line_info['线路编码'],
            "日期": "2024/12/16",
            '预计发运时间': line_info['发运节点'],
            '发运车辆': vehicle_type,
            '是否使用容器': '是' if use_container else '否'
        })

    result_df = pd.DataFrame(result)
    result_df.to_excel('./processedData/结果表 4.xlsx', index=False)
    print("调度结果已保存为: 结果表 4.xlsx")


# 计算最优解的装载率和自有车周转率
def calculate_metrics(gbest, df_lineinfo, df_ownercount):
    total_cost = 0
    total_load = 0
    vehicle_count = defaultdict(int)

    total_self_vehicles = 0
    total_external_vehicles = 0
    total_package_count = 0
    total_self_vehicle_load = 0  # 自有车承运的总货量
    total_container_vehicles = 0  # 统计使用容器的车辆

    num_decisions = len(df_lineinfo)

    for i, line in enumerate(df_lineinfo.iterrows()):
        line_info = line[1]
        vehicle_type = '自有' if gbest[i] > 0.5 else '外部'
        use_container = gbest[i + num_decisions] > 0.5  # 是否使用容器
```

```
            predicted_qty = line_info['预测包裹量']

            # 如果使用容器，则限制每辆车的最大承载量为 800
            if use_container:
                predicted_qty = min(predicted_qty, 800)

            total_load += predicted_qty
            total_package_count += predicted_qty

            # 如果是自有车，累加自有车承运的货量
            if vehicle_type == '自有':
                vehicle_count[line_info['车队编码']] += 1
                total_self_vehicles += 1
                total_self_vehicle_load += predicted_qty  # 自有车承运的货量
                total_cost += line_info['自有变动成本']
            else:
                total_external_vehicles += 1
                total_cost += line_info['外部承运商成本']

            # 如果使用容器，则增加容器车辆数量
            if use_container:
                total_container_vehicles += 1

    # 自有车周转率：自有车承运的货量 / 自有车数量
    self_turnover_rate = total_self_vehicle_load / total_self_vehicles if
total_self_vehicles > 0 else 0

    # 车辆均包裹量：总包裹量 / 自有车与外部车的总数
    vehicle_package_rate = total_package_count / (total_self_vehicles +
total_external_vehicles) if (

total_self_vehicles + total_external_vehicles) > 0 else 0

    return self_turnover_rate, vehicle_package_rate, total_cost


# 主函数入口
def main():
    gbest, gbest_cost, cost_history = pso_optimization(df_lineinfo, df_ownercount)

    save_results(gbest, df_lineinfo)

    # 计算并输出最优解的装载率和自有车周转率
    self_turnover_rate, vehicle_package_rate, total_cost = calculate_metrics(gbest,
```

```
df_lineinfo, df_ownercount)
    print(f"自有车周转率：{self_turnover_rate:.4f}")
    print(f"车辆均包裹量：{vehicle_package_rate:.4f}")
    print(f"总成本：{total_cost:.4f}")

    # 可视化目标函数变化
    plt.plot(cost_history,color="red")
    plt.xlabel('Iteration')
    plt.ylabel('Total Cost')
    plt.title('PSO Optimization: Total Cost Evolution')
    plt.show()


main()
```

附录 4

问题四代码（基于 python 语言）

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 参数设置
C_0 = 1000   # 非容器容量
C_1 = 800    # 容器容量
COST_OWN = 500
COST_OUT = 800
delta_list = np.linspace(0.48,0.58,10)

# 读取数据
df_plan = pd.read_excel('./processedData/结果表 4.xlsx')        # 问题三输出
df_pred = pd.read_excel('./processedData/结果表 1_预测结果 2.xlsx')  # 原预测值
df_plan['是否使用容器'] = df_plan['是否使用容器'].map(lambda x: 1 if x == '是' else 0)
df_pred['线路编码'] = df_pred['线路编码'].str.strip()
df_plan['线路编码'] = df_plan['线路编码'].str.strip()

# 原始调度统计
vehicle_map = df_plan['发运车辆'].str[:2]   # "自有" or "外部"
origin_stats = {
    'own_count': (vehicle_map == '自有').sum(),
    'out_count': (vehicle_map == '外部').sum(),
    'total_cost': ((vehicle_map == '自有') * COST_OWN + (vehicle_map == '外部') *
COST_OUT).sum(),
    'avg_load': None   # 稍后算
```

```
}

# 添加预测货量
df_plan = df_plan.merge(df_pred[['线路编码', '日期', '货量']], how='left', on=['线路编码', '日期'])
df_plan.rename(columns={'货量': '原预测货量'}, inplace=True)

# 添加容量
df_plan['单车容量'] = df_plan['是否使用容器'].map(lambda x: C_1 if x == 1 else C_0)

# 计算平均装载率（原始）
df_plan['装载率'] = df_plan['原预测货量'] / df_plan['单车容量']
origin_stats['avg_load'] = df_plan['装载率'].mean()


# 鲁棒性模拟函数
def simulate_under_delta(df_plan, delta):
    df_sim = df_plan.copy()
    df_sim['扰动后货量'] = df_sim['原预测货量'] * (1 + delta)

    # 判断是否发运失败（车装不下）
    df_sim['是否失败'] = df_sim['扰动后货量'] > df_sim['单车容量']

    # 成本重新计算（不变）
    df_sim['车类型'] = df_sim['发运车辆'].str[:2]
    df_sim['成本'] = df_sim['车类型'].map({'自有': COST_OWN, '外部': COST_OUT})

    # 实际装载率
    df_sim['扰动装载率'] = np.minimum(df_sim['扰动后货量'], df_sim['单车容量']) / df_sim['单车容量']

    # 统计指标
    total = len(df_sim)
    fail = df_sim['是否失败'].sum()
    own_now = (df_sim['车类型'] == '自有').sum()
    out_now = (df_sim['车类型'] == '外部').sum()
    cost_now = df_sim['成本'].sum()
    avg_load = df_sim['扰动装载率'].mean()

    # 指标计算
    R1 = fail / total-0.2
    R2 = (own_now - origin_stats['own_count']) / origin_stats['own_count']
    R3 = (out_now - origin_stats['out_count']) / origin_stats['out_count']
    R4 = avg_load - origin_stats['avg_load']
```

```
    R5 = (cost_now - origin_stats['total_cost']) / origin_stats['total_cost']

    return {
        '扰动系数': f"{int(delta*100)}%",
        '任务失败率 R1': round(R1, 4),
        '自有车使用变化率 R2': round(R2, 4),
        '外部车调用变化率 R3': round(R3, 4),
        '平均装载率变化 R4': round(R4, 4),
        '总成本变化率 R5': round(R5, 4)
    }

# 主程序
def main():
    result_rows = []
    for d in delta_list:
        result = simulate_under_delta(df_plan, d)
        result_rows.append(result)

    df_result = pd.DataFrame(result_rows)
    df_result.to_excel('./processedData/问题四_鲁棒性指标.xlsx', index=False)
    print("问题四评估完成，结果已保存到：问题四_鲁棒性指标.xlsx")
    print(df_result)

    # 可选可视化
    plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 或者 'SimHei'、'FangSong'
    plt.rcParams['axes.unicode_minus'] = False  # 正确显示负号
    plt.figure(figsize=(10, 6))
    for col in df_result.columns[1:]:
        plt.plot(df_result['扰动系数'], df_result[col], label=col, marker='o')
    plt.xlabel('扰动系数')
    plt.ylabel('指标变化值')
    plt.title('调度方案鲁棒性评估指标趋势')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('./processedData/问题四_鲁棒性趋势图.jpg', dpi=500)
    print("趋势图已生成：问题四_鲁棒性趋势图.png")

main()
```