分拣中心货量和人员排班预测模型

摘要

对于电商平台来说，分拣中心的分拣速度对物流网络中的运输时效和运作成本影响巨大。本文基于历史数据建立数学模型对各分拣中心的货量进行预测，并在各种情境下尝试给出合理的人力资源安排，对物流网络的管理决策和效能提升具有一定的参考价值。

对于问题一, 我们首先进行数据预处理，采用 **SVR 插值**对过去 30 天每小时的历史货量数据进行**缺失值**填补，并通过**白噪音检测**剔除过去 4 个月每天货量和过去 30 天小时货量中的**异常值**。随后建立**双向卷积神经网络**模型和**基于贝叶斯优化的随机森林**模型，使用题目提供的历史数据对模型进行训练，尝试得到合适的模型参数。随后，我们将历史货量导入训练后的模型进行预测，得到 57 个分拣中心未来 30 天每天及每小时的货量预测结果，记录在结果表 1 和结构表 2 中。比较两个模型的预测结果，发现其拟合效果和误差差别不大，但是随机森林模型出现了较为严重的末端提前收敛，因此我们选择基于双向神经网络的货量预测模型作为问题一的预测模型。

对于问题二, 首先根据图论中**有向图**的相关概念，把运输线路网抽象成有向图，两分拣中心之间的运输线路即连接两顶点的单向边，分拣中心之间的货量即为有向图边的**权重**，把问题二转化为已知有向图的边和顶点，求每条边对应权重的问题。采用**出度**和**入度**边平均权重的差值计算新边的权重，建立问题二的货量预测模型，将附件 1 和 2 的历史数据导入后得到预测结果，记录在结果表 3 和 4 中。

对于问题三, 将其当作**双目标整数线性规划问题**来建模求解，把题目要求的尽可能多使用正式工和安排人天数尽可能少当作目标函数的两个指标，把保证配置能完成每天货量处理的人力、正式工名额为 60 人、每天的实际小时人效尽可能均衡当作**约束条件**，把每个班次中的正式工和临时工人数作为决策变量，即可建立基于双目标整数线性规划的排班预测模型。再通过多线程方法对模型进行求解，将预测排班结果记录着结果表 5 中。

对于问题四, 将其看成一个**混合整数规划问题**，在问题三的基础上，同样把每个班次中的正式工和临时工人数作为决策变量，把新增的目标函数的指标尽可能少地使用临时工和约束条件正式工出勤率不能高于 85%、正式工不能连续出勤超过 7 天、正式工出勤率尽量均衡加入到原有的排班预测模型中，即可建立基于混合整数规划的排班预测模型。按照类似思路进行求解，并将排班的预测结果记录在结果表 6 中。

关键字： BiTCN-LSTM 模型　随机森林算法　有向网络图　整数规划模型

一、 问题重述

1.1 问题背景

近年来，随着电商平台的兴起，物流网络也不断完善，运输效率不断提高。在物流网络中，分拣中心起到关键作用，其管理效率和分拣速度极大地影响了运输时效和运作成本。因此，基于历史数据对各分拣中心的货量进行预测，合理安排人力和运力资源，降低成本，对物流网络的管理决策和效能提升具有重要意义。

1.2 问题要求

问题 1 要求依据附件 1 中过去 4 个月每个分拣中心每天的货量和附件 2 中过去 30 天每小时的货量 1. 建立货量预测模型;2. 对每个分拣中心未来 30 天每天的货量进行预测，将预测结果写入结果表 1;3. 对每个分拣中心未来 30 天每个小时的货量进行预测，将预测结果写入结果表 2。

问题 2 附件 3 为过去 90 天各分拣中心之间的运输路线和运量，附件 4 为未来 30 天各分拣中心之间的运输路线。1. 要求依据附件 3 来预测附件 4 中未来 30 天各运输路线上的货量;2. 并结合附件 1 和 2 对未来 30 天每个分拣中心的每天和每小时的货量进行预测，将预测结果写入结果表 3 和 4 中。

问题 3 要求基于问题二的预测结果，假设在每个分拣中心有 60 名正式工，人员安排优先使用正式工的情况下，要求安排的人天数尽可能少，每天的实际小时人效均衡，1. 建立预测模型;2. 确定未来 30 天每个分拣中心每个班次的出勤人数，将预测结果写入结果表 5 中。

问题 4 要求对 SC60 进行预测，在问题三的基础上，假设 SC60 有 200 名正式工，要求正式工出勤率不高于 85%，连续出勤不能超过 7 天，且正式工出勤率也要尽量均衡，1. 建立预测模型;2. 确定未来 30 天 SC60 每名正式工和临时工的班次出勤计划。

二、 问题分析

2.1 问题一分析

对于问题一，要求根据历史货量建立货量预测模型，我们只需要首先进行数据预处理，采用 SVR 插值来填补缺失值，并通过白噪音检测剔除异常值；随后建立机器学习模型，并使用题目提供的历史数据对模型进行训练，即可得到合适的模型参数和预测数据。

2.2 问题二分析

对于问题二，要求在运输路线变更之后重新根据历史数据建立货量预测模型进行预测。实际上我们只需要把运输线路抽象成有向图，把每两个分拣中心之间的货量抽象成有向图边的权重，把问题转化为已知历史状态的有向图，在新状态下求边和顶点都确定的有向图权重的问题。依据图论相关概念建立预测模型，将附件 1 和 2 的历史数据导入后求解即可。

2.3 问题三分析

对于问题三，要求在给定正式工名额、优先使用正式工的条件下，建立安排人天数最少，且每天实际小时人效最均衡的排班预测模型。对此，我们只要把问题三看成一个双目标整数线性规划问题，把尽可能多使用正式工和安排人天数尽可能少两个题目要求当作目标函数的两个关键量，把保证配置能完成每天货量处理的人力、正式工名额为 60 人、每天的实际小时人效尽可能均衡当作约束条件，把每个班次中的正式工和临时工人数作为决策变量，即可建立双目标整数线性规划模型。再把题给数据导入，通过多线程方法对模型进行求解，预测排班结果即可。

2.4 问题四分析

对于问题四，将问题改变为研究特定的分拣中心 SC60，并在问题三的基础上增加了目标尽可能少地使用临时工和约束条件正式工出勤率不能高于 85%、正式工不能连续出勤超过 7 天、正式工出勤率尽量均衡。对此，我们只要把问题四看成一个混合整数规划问题，同样把每个班次中的正式工和临时工人数作为决策变量，把新增的目标函数关键量和约束条件加入到原有的问题三的模型中，即可建立混合整数规划模型。再把题给数据导入，按照类似的思路进行模型求解，得出排班的预测结果即可。

三、 模型假设

为简化问题，本文做出以下假设：

- 假设 1 假设供应链运输和人员排版均为正常时间段，即不存在如突发事件、节假日、集中促销等特殊情况。
- 假设 2 假设各分拣中心可以立即找到足够的临时工，并且能随时清退。
- 假设 3 假设正式工和临时工的小时能效指标均以最高计。

四、 符号说明

符号	说明	单位
ω	节点新边权重	/
ω_i	节点入度边权重	/
ω_k	节点出度边权重	/
i	节点入度	/
k	节点出度	/
x_i	每班次正式工人数	人
y_i	每班次临时工人数	人
A	目标 1：尽可能多使用正式工	/
B	目标 2：尽可能少使用临时工	/
C	目标 3：安排的人天数尽可能少	/
M	预测货量	/
U	实际小时人效	包裹/小时
$G1$	旧路径有向图	/
$G2$	新路径无权重有向图	/
$G3$	新路径有权重有向图	/
n	正式工出勤总天数	天
n_{con}	正式工连续出勤天数	天
l	目标函数	/
Δn_{min}	约束 1：正式工出勤率尽量均衡	/
ΔU_{min}	约束 2：实际小时人效尽量均衡	/

五、 问题一的模型的建立和求解

5.1 问题一模型的建立

我们先后构建 BiTCN-LSTM（双向卷积神经网络）模型和随机森林模型对问题一进行建模求解，比较两个模型的拟合度和预测图像选择更加合适的模型作为问题一的货量

预测模型。

5.1.1 构建 BiTCN-LSTM 模型

5.1.2 异常值处理

我们采用白噪声检测对历史货量数据中的异常值进行处理。
首先计算数据的自相关系数，随后根据置信水平选择合适的 z 分数，检查该数据点是否为白噪声。

5.1.3 构建 BiTCN-LSTM 模型-小时预测模型

5.1.4 缺失值填充

由于原始的小时历史货量数据存在缺失值，我们采用 SVR 对缺失值进行替换。
首先，我们通过 $fit_transformer$ 将数据标准化，随后训练 SVR 模型来预测缺失值并替换。

5.1.5 模型构建

首先添加多个时间卷积层和双向 LSTM 层进行模型训练，随后在测试集上进行预测以评估模型性能。比较反缩放预测结果和计算测试集上的误差，并绘制模型对原始数据拟合效果的折线图

5.1.6 构建 BiTCN-LSTM 模型-天数预测模型

与上述小时预测模型的建立过程一致。

5.1.7 缺失值填充

与神经网络模型类似，采用 SVR 进行缺失填充。

5.1.8 构建基于贝叶斯优化的随机森林模型

首先定义随机森林回归模型，采用历史数据进行训练，得到贝叶斯优化随机森林模型参数，并使用最佳参数训练模型。

随后，我们在测试集上进行预测以评估模型性能，比较反缩放预测结果并计算测试集上的误差，绘制模型对原始数据拟合效果的折线图。

5.2 问题一模型的求解

5.2.1 数据准备与预处理

Step1：读取数据

使用 pandas 库读取 CSV 文件中的数据，确保日期时间格式正确。对于日数据，将”日期” 列转换为 pandas 的 datetime 类型。对于小时数据，创建一个新的 datetime 列，该列由” 日期” 和” 小时” 拼接而成。

Step2：数据分组与排序

使用 groupby 方法按” 分拣中心” 对数据进行分组，并使用 $sortvalues$ 方法确保每组数据按时间顺序排序。

Step3：数据清洗与预处理

(1). 异常值处理：对于天数据，使用自相关函数 ACF 检验数据是否为白噪音，如果不是，则使用支持向量回归 SVR 模型来预测并替换异常值。

(2). 缺失值填充：对于小时数据，使用支持向量回归 SVR 模型预测缺失的货量值。

(3). 归一化：使用最小-最大归一化（$MinMaxScaler$）将货量数据缩放到 0 到 1 之间，以便于模型训练。

d. 创建监督学习问题：将时间序列数据转换为监督学习问题所需的格式，即特征集（X）和目标变量（Y）。

5.2.2 构建评估 BiTCN-LSTM 模型

Step1：建立并训练 BiTCN-LSTM 模型

(1). 使用 Sequential 模型来堆叠多个层。

(2). 添加多个时间卷积层 ($Conv1D$): 在时间序列数据上使用一维卷积层来提取局部特征。这些层可以帮助模型捕获数据中的短期依赖性和周期性模式。

(3). 添加双向 LSTM 层: 通过双向 LSTM 层，模型能够同时学习时间序列的正向和反向依赖，从而捕获长期依赖性。

(4). 添加一个全连接层 (Dense) 作为输出层。

(5). 使用 adam 优化器和 $meansquarederror$ 作为损失函数来编译模型。

(6). 数据集划分: 使用 $traintestsplit$ 方法将处理后的数据划分为训练集和测试集，测试集占总数据的 20%。

(7). 训练模型

使用训练集数据训练 BiTCN-LSTM 模型，通过多次迭代（epochs）和批量大小

（$batchsize$）来调整模型权重。同时在训练过程中，模型通过反向传播算法和梯度下降来优化权重，以最小化损失函数。

Step2：模型评估

(1). 在测试集上进行预测，以评估模型的性能。

(2). 使用均方误差（MSE）和平均绝对误差（MAE）作为评估指标。

(3). 绘制测试集上的实际值与预测值的对比图，以直观展示模型的拟合效果。

Step3：预测与结果保存

(1). 多步预测

使用 $multistepforecast$ 函数进行未来多步预测未来 30 天的日数据和 720 小时的小时数据），并在预测过程中，将上一步的预测结果作为下一步的输入，循环进行预测。

(2). 结果整理与保存

将预测结果整理成 DataFrame 表格并保存到 CSV 文件中，以便于后续分析和使用。

Step4：可视化数据

使用 matplotlib 库绘制原始数据的时间序列图、模型拟合效果的折线图和预测结果的时间序列图。这些图表有助于直观地理解数据的变化趋势和模型的预测能力。

5.2.3 构建基于贝叶斯优化的随机森林模型

Step1：定义随机森林模型

使用 $RandomForestRegressor$ 类来定义随机森林回归模型。此时，模型尚未被训练，只是定义了其结构。

Step2：贝叶斯优化随机森林模型参数

为了找到随机森林模型的最佳参数，代码使用了贝叶斯优化方法，通过 BayesSearchCV 类实现。

贝叶斯优化是一种在给定目标函数（在本例中是模型的负均方误差）的情况下，通过概率模型来寻找最优参数的技术。

(1). 定义参数分布：通过 $paramdistributions$ 定义了随机森林模型的各个参数的分布范围或选项。这些参数包括树的数量（$nestimators$）、树的最大深度（$maxdepth$）、分割内部节点所需的最小样本数（$minsamplessplit$）、叶节点所需的最小样本数（$minsamplesleaf$）以及考虑用于分割的最大特征数（$maxfeatures$）。

(2). 执行参数搜索：BayesSearchCV 通过交叉验证（通过 cv 参数指定）在训练集上迭代地搜索最佳参数组合。$niter$ 参数指定了搜索的迭代次数。

(3). 获取最佳参数：搜索完成后，$bestparams$ 属性包含了找到的最佳参数组合。

Step3：使用最佳参数训练模型

使用贝叶斯优化找到的最佳参数重新训练随机森林模型。这一步确保了模型是使用经过优化的参数进行训练的。

Step4：模型评估与预测

(1). 模型评估：在测试集上评估训练好的模型性能，计算并输出均方误差（MSE）和平均绝对误差（MAE）。

(2). 绘制拟合效果：绘制测试集上的实际值与预测值的对比图，以直观展示模型的拟合效果。

(3). 多步预测：使用 $multistepforecast$ 函数进行多步预测，即预测未来多个时间步的货量。该函数通过迭代地使用模型的预测结果作为下一个时间步的输入来实现多步预测。

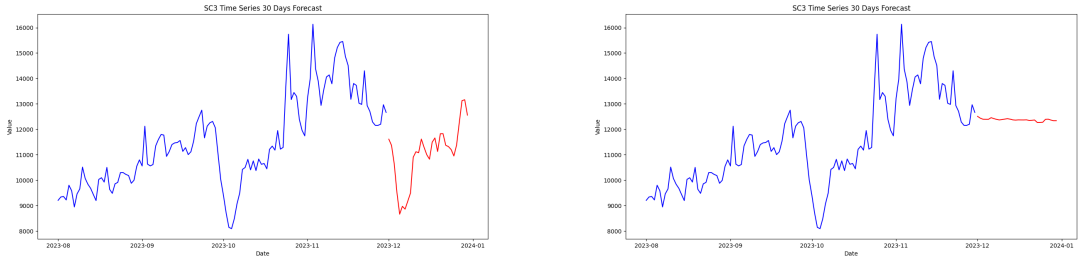
Step5：结果保存与可视化

(1). 保存预测结果：将预测结果保存到 CSV 文件中，以便后续分析。

(2). 可视化预测结果：绘制原始数据的时间序列图和预测数据的时间序列图，以便直观比较。

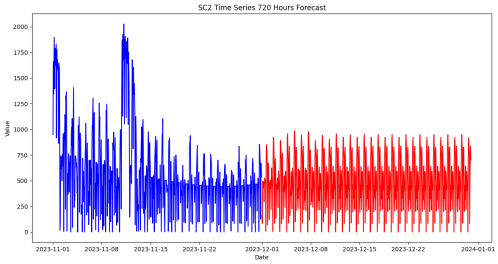
5.3 问题一求解结果

5.3.1 神经网络和随机森林的求解结果比较

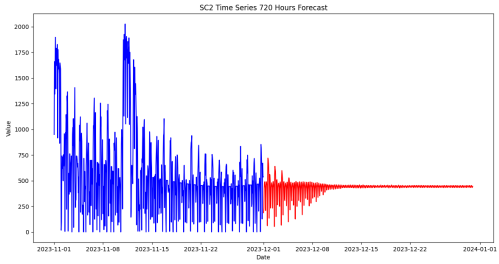


(a) BiTCN-LSTM 日模型　　**(b) RandomForest 日模型**

图 1　Day prediction



(a) BiTCN-LSTM 小时模型



(b) RandomForest 小时模型

图 2　Hour prediction

表 1　小时预测模型 MAE

分拣中心	SC1	SC10	SC12	SC14	SC15	SC16
神经网络日预测 MAE	3831.70	2053.31	601.76	136.29	430.04	473.54
随机森林日预测 MAE	3058.39	1715.33	472.99	149.62	417.09	330.13

表 2　小时预测模型 MAE

分拣中心	SC1	SC10	SC12	SC14	SC15	SC16
神经网络小时预测 MAE	397.27	431.03	128.80	26.62	140.38	32.38
随机森林小时预测 MAE	386.20	347.01	100.01	21.74	103.84	28.48

对比采用神经网络模型和随机森林模型得到的结果，我们可以直观看出，虽然随机森林模型预测的平均绝对误差低于神经网络模型预测的平均绝对误差，预测准确度随机

森林模型高于神经网络模型，但是随机森林模型预测的末端收敛过早，不符合常理。因此，我们选择神经网络模型作为问题一最后的模型和求解结果。

六、 问题二的模型的建立和求解

在本题的建模过程中使用了图论中有向图的知识，因为货物的运输是单程的，具有单向性，且在分拣过程中不计损耗，完全符合有向图的概念。因此，我们将货物分拣中心的货量转化为每个顶点出度和入度的差值，每条边的权重来表示，从而简化问题的解决过程。

6.1 问题二模型的建立

6.1.1 有向图的概念

首先，我们可以把有向图简单理解为每条边都带有方向的一些顶点构成的图结构。随后，为了解决本题，我们还需要明确出度和入度的概念。

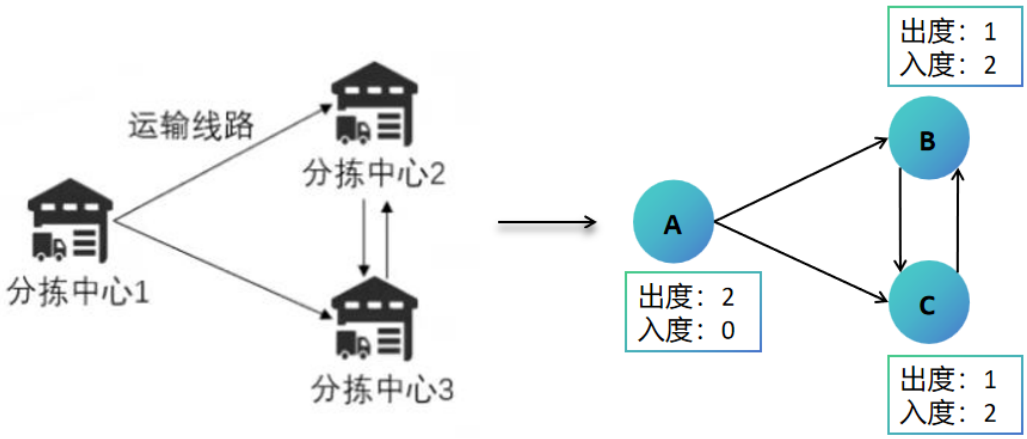


图 3　物流网络运输线路示意图

对于一个有向图中的顶点来说，从该顶点出发的边的数目就是出度，指向该顶点的边的数目则是入度。我们以题目给出的物流网络示意图为例，分别令运输线路上的分拣中心 1,2,3 为 A,B,C 三个点，就将其抽象为了一个有向图。根据上述概念可知，对 A 点来说，有两条边从它出发，但没有边指向它，因此 A 点的出度为 2，入度为 0。同理易知 B 点的出度为 1，入度为 2；C 点的出度为 1，入度为 2。

特别的，对于有向图，均有所有顶点的入度之和等于所有顶点的出度之和。

为便于模型建立，我们还把连接两个分拣中心的运输线路的平均货量抽象为连接两个顶点的边上的权重。

6.1.2 问题二货量预测模型

现在，我们只需要把运输线路中的各个量抽象为有向图中的各个结构。

我们把始发分拣中心看成起点，到达分拣中心看成终点，把两点之间存在的路径看成是一条有向边，把每两个分拣中心之间的运输货量看成这条边的权重。由此构建出图网络。

由于所有运达分拣中心的货物都会被最终送到营业部，因此，所有顶点的入度之和与出度之和相等这一定理成立。

首先，我们将附件 3 中给出的“过去 90 天各分拣中心之间的各运输线路平均货量”绘制成有向图 G1。并将附件 4 中给出的“未来 30 天变化后的运输线路”绘制成一个新的有向图 G2，其中权重未知，但顶点和边均已给出，因此每个顶点的出度和入度也是确定的。

为方便建立数学模型，我们复制一份有向图 G1 到 G3，使得 G3 和 G1 拥有相同的原始状态下的边和权重。对比 G3 和 G2，我们删除 G3 中不在 G2 中的边，并添加 G2 中存在但在 G1 中不存在的新边到 G3，此时 G3 中的边变成了未来 30 天的新状态。因此，要求出未来 30 天的货量预测，我们实际上只要求出有向图 G3 中新边的未知权重即可。

使用 G1 中的原始节点和权重来进行计算。如果节点的平均出权重或入权重为零，则新边的权重为两者之和；否则，为新边的权重分配两者平均值。

以图 4为例，左图表示线路变化前的局部分拣中心之间的线路和权重。则在右图的新状态中，由于 A 的出度边只有 1 和 5，容易得出新路线 5 的权重和原来路线 1 的相等；同理，由于 B 的出度边只有 2 和 6，容易得出边 6 的权重即等于边 2。

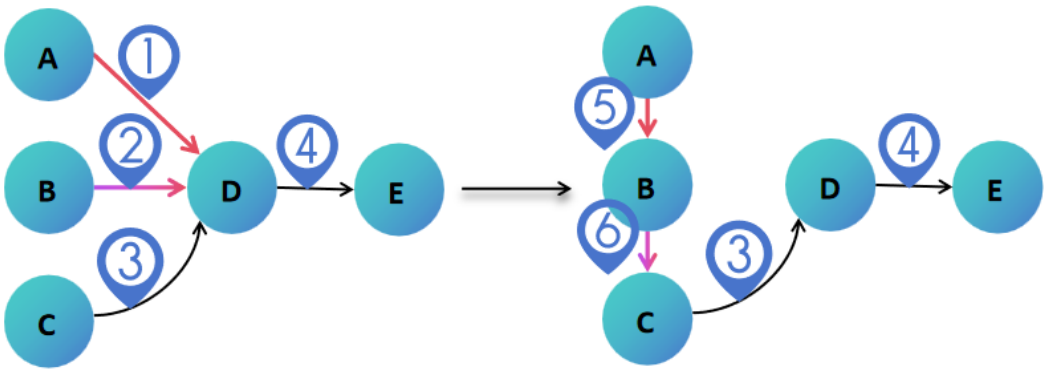


图 4　权重分配示意图

我们把对于每个顶点的入度记为 i，出度记为 k，把每条入度边对应的权重用 ω_i 表示，每条出度的边用 ω_k 表示, 可以得到以下新边的权重计算式子：

$$\omega = \frac{\sum_{i=0}\omega_i}{2i} - \frac{\sum_{k=0}\omega_k}{2k} \tag{1}$$

如果出度和入度任意一个权重之和为 0，则不用权重计算式为：

$$\omega = \frac{\sum_{i=0}\omega_i}{2i} - \frac{\sum_{k=0}\omega_k}{k} \tag{2}$$

如果计算得到 $\omega < 0$，则将新路线的权重 ω 记为 0; 故得到**问题二的货量预测模型**：

$$\begin{cases} \omega = \frac{\sum_{i=0}\omega_i}{2i} - \frac{\sum_{k=0}\omega_k}{2k} \\ \omega = \frac{\sum_{i=0}\omega_i}{2i} - \frac{\sum_{k=0}\omega_k}{k} (\sum_{i=0}\omega_i = 0 \; or \sum_{k=0}\omega_k = 0) \\ \omega = 0 \end{cases} \tag{3}$$

将问题一中未来 30 天每天和每小时的预测货量作为权重依次放入 G1，即可得到对应的 G3 的权重，即运输路线变化后的每天和每小时的预测货量。

6.2 问题二模型的求解

6.2.1 读取数据

首先，从问题一的两个结果表 CSV 文件中读取货量数据，以及从两个附件中读取新旧两个运输线路数据。

6.2.2 构建运输线路的有向网络图

Step1：初始化图

创建三个有向图 G1、G2 和 G3。

向 G1 添加原始边和权重（$edges_{origin}$）。

向 G2 添加新的边（edges）。

向 G3 添加与 G1 相同的原始边和权重。

Step2：删除 G3 中不在 G2 中的边

遍历 G3 的所有边，并移除那些不在 G2 中的边。这一步确保 G3 只包含 G2 也有的边。

Step3：添加 G2 中存在但在 G1 中不存在的新边到 G3

遍历 G2 的所有边，并找出那些不在 G1 中的新边。

对于这些新边，检查 G1 中是否存在相应的节点。

如果存在节点，计算新边的权重。如果节点的平均出权重或入权重为零，则新边的权重为两者之和；否则，为新边的权重分配两者平均值。将新边及其权重添加到 G3 中。

Step4: 更新 G3 中已存在边的权重

遍历 G3 的所有边，并更新它们的权重。

a. 如果 G1 中也有这条边，则直接从 G1 中获取权重。

b. 如果 G1 中没有这条边，则使用与添加新边时相同的逻辑计算权重，并将其分配给 G3 中的边。

step5: 删除没有边的点

遍历 G3 的所有节点，并移除那些没有连接任何边的孤立节点。确保 G3 中只包含至少有一条边的节点。

6.2.3 图的存储与可视化

将 G3 的节点和边信息存储到字典 graph 中，并保存到 CSV 文件。

使用 matplotlib 和 networkx 绘制 G3 的有向图，并根据边的权重调整边的宽度。

6.2.4 货量更新

遍历两个结果表中的每个分拣中心，计算每个分拣中心在 G1 中的入边和出边权重之和的差值，以此来更新每个分拣中心的货量。

如果更新后的货量出现负数，则将其设置为 0，表示货量不能为负。

更新后的货量数据被保存到新的 CSV 文件中。

6.3 问题二求解结果

采用 Python 编程求解，将未来 30 天 57 个分拣中心每天的货量预测数据记录在结果表 3 中，每小时的货量预测数据记录在结果表 4 中，并将变化后的路线预测货量加权后的结果可视化为有向图，如图 5所示。

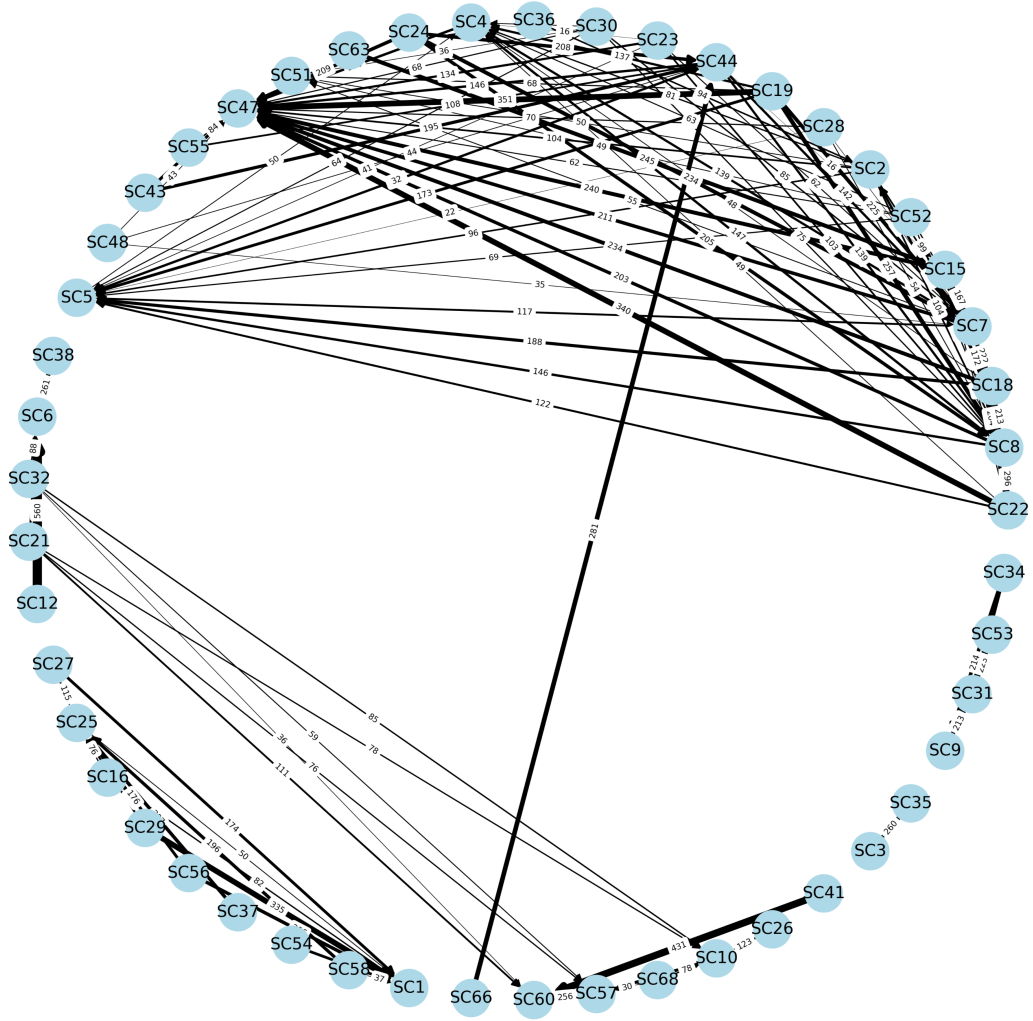


图 5　Q2 未来 30 天的新路线货量预测图

七、 问题三的模型的建立和求解

7.1 问题三模型的建立

在问题三中，我们采用双目标整数规划模型来解决问题。在满足分拣中心货量处理需求的同时，最小化临时工的使用。

在模型建立过程中，我们把每小时的总处理能力需满足货量处理需求、正式工总数不超过最大限制、临时工使用受到正式工总数的约束、每天实际小时人效均衡作为约束条件，通过线性规划方法计算出每个班次应使用的正式工和临时工人数。

7.1.1 双目标整数线性规划模型

双目标整数线性规划模型 (2-OILP) 涉及两个目标函数同时最大化或者最小化，并且所有变量必须是整数。它在满足约束条件的同时可以平衡多个性能指标。

决策变量： 1. 正式工在每个班次中的人数，将班次用 i 表示，则该决策变量记为 x_i; 2. 临时工在每个班次中的人数, 同理将该决策变量记为 y_i。

约束条件： 1. 约束条件一　配置的人力要保证能完成每天的货量处理。

由题设提供的小时人效指标可知，正式工的最高小时能效为 25 个包裹/小时，临时工的最高人效为 20 个包裹/小时，因此可以得到人力配置上的约束条件：

$$25x_i + 20y_i \geq M \tag{4}$$

其中 M 是预测货量。

2. 约束条件二　按题设，每个分拣中心有 60 名正式工：

$$0 \leq x_i \leq 60 \tag{5}$$

3. 约束条件三　题目要求每天的实际小时人效尽量均衡：

因为有

$$\text{实际小时人效} = \frac{\text{实际完成的工作量}}{\text{实际工作时间}} \tag{6}$$

且按照题设，一个班次工作时间为 6 小时，故将实际小时人效记为 U，得到式子：

$$\begin{cases} U = \frac{25x_i+20y_i}{6} \\ \Delta U_{min} \end{cases} \tag{7}$$

目标函数： 建立最小化临时工的使用的双目标函数 1. 尽可能多使用正式工，记为 A

2. 安排的人天数尽可能少，记为 C

$$l = \alpha A + \gamma C \tag{8}$$

基于上述，我们可以把**问题三的双目标整数线性规划模型**表达为

$$\begin{cases} l = \alpha A + \gamma C \\ 25x_i + 20y_i \geq M \\ 0 \leq x_i \leq 60 \\ U = \frac{25x_i+20y_i}{6} \\ \Delta U_{min} \end{cases} \tag{9}$$

7.2 问题三模型的求解

Step1：问题定义

(1). 目标：最小化临时工的使用数量，同时限制正式工的使用不超过一定数量。

(2). 约束：每个时间段的货量需求必须得到满足，且正式工的总数不能超过设定的上限。

Step2：数据准备

(1). 从 CSV 文件中读取货量数据，确保数据按分拣中心和日期分组。

(2). 对货量数据进行四舍五入处理，以适应整数规划的要求。

(3). 定义班次模式，表示每个班次在不同时间段的工作状态（1 表示工作，0 表示不工作）。

Step3：模型构建 (1). 变量定义：

a.$xvars$：正式工的班次安排，每个班次一个变量，类型为整数。

b.$yvars$：临时工的班次安排，同样每个班次一个变量，类型为整数。

c.$totalregularworkers$：正式工的总数，用于限制正式工的使用量。

(2). 目标函数：

目标是最小化临时工的使用数量，同时对正式工的班次安排给予较小的权重（这里用 0.01 表示），以体现双目标优化的意图。

(3). 约束条件：

a. 货量需求约束：每个时间段的货量需求必须被满足，通过正式工和临时工的工作量累加来覆盖。这里考虑了不同班次在不同时间段的工作状态以及不同类型工人的工作效率。

b. 正式工总数约束：正式工的总数不能超过设定的上限。

c. 临时工使用约束：通过一个大数 M 来引入条件约束，当正式工数量未达到上限时，允许使用临时工。

Step4：求解过程

(1). 使用 pulp 库的 LpProblem 类创建问题实例，并调用 CBC 求解器进行求解。

(2). 设置求解器的消息输出和时间限制（300 秒）。

(3). 调用 solve 方法求解问题，并获取变量值。

Step5：结果处理

(1). 初始化结果字典，用于存储每个分拣中心、每个日期、每个班次的正式工和临时工人数。

(2). 使用 multiprocessing.Pool 进行多线程求解模型。

(3). 将求解结果添加到结果字典中，并保存到 CSV 文件。

7.3 问题三求解结果

采用 Python 编程求解，将未来 30 天每个分拣中心每个班次出勤人数的预测数据记录着结果表 5 中。

八、 问题四的模型的建立和求解

8.1 问题四模型的建立

8.1.1 混合整数规划模型

混合整数规划是一种结合了线性规划和整数规划特点的优化问题，能够很好地处理离散决策变量，精确描述优化条件，在物流领域的库存管理和货物配送方面使用广泛。

决策变量： 1. 正式工在每个班次中的人数，将班次用 i 表示，则该决策变量记为 x_i; 2. 临时工在每个班次中的人数, 同理将该决策变量记为 y_i。

约束条件： 1. 约束条件一　配置的人力要保证能完成每天的货量处理。
由题设提供的小时人效指标可知，正式工的最高小时人效为 25 个包裹/小时，临时工的最高人效为 20 个包裹/小时，因此可以得到人力配置上的约束条件：

$$25x_i + 20y_i \geq M \tag{10}$$

其中预测货量记为 M。

2. 约束条件二　按题设，分拣中心 SC60 有 200 名正式工：

$$0 \leq x_i \leq 200 \tag{11}$$

3. 约束条件三　题目要求每名正式工的出勤率不能高于 85%。将正式工的出勤天数记为 n, 得到

$$\frac{n}{30} \leq 85\% \tag{12}$$

4. 约束条件四　题目要求每名正式员工的连续出勤天数不能超过七天:

$$n_{con} \leq 7 \tag{13}$$

5. 约束条件五　题目要求每天的实际小时人效尽量均衡：因为有

$$\text{实际小时人效} = \frac{\text{实际完成的工作量}}{\text{实际工作时间}} \tag{14}$$

且按照题设，一个班次工作时间为 6 小时，故将实际小时人效记为 U，得到式子：

$$\begin{cases} U = \frac{25x_i + 20y_i}{6} \\ \Delta U_{min} \end{cases} \tag{15}$$

6. 约束条件六　题目要求正式工出勤率尽量均衡：

$$\Delta n_{min} \tag{16}$$

目标函数：建立最小化临时工的使用的多目标函数

1. 尽可能多使用正式工，记为 A

2. 尽可能少使用临时工，记为 B

3. 安排的人天数尽可能少，记为 C

$$l = \alpha A + \beta B + \gamma C \tag{17}$$

基于上述，我们可以把**问题四的混合整数规划模型**表达为

$$\begin{cases} l = \alpha A + \beta B + \gamma C \\ 25x_i + 20y_i \geq M \\ 0 \leq x_i \leq 200 \\ \frac{n}{30} \leq 85\% \\ n_{con} \leq 7 \\ U = \frac{25x_i + 20y_i}{6} \\ \Delta U_{min} \Delta n_{min} \end{cases} \tag{18}$$

8.2 问题四模型的求解

Step1：变量定义

(1). $xvars$：正式工的班次变量，表示每个正式工在每个班次的工作情况。

(2). $yvars$：临时工的班次变量，表示每个临时工在每个班次的工作情况。

(3). $regulardays$：正式工的出勤天数变量，表示每个正式工的总出勤天数。

(4). $regularshifts$：正式工的班次分配变量，表示每个正式工在每个班次是否被分配工作。

(5). $regularshiftcounts$：正式工的班次数变量，表示每个正式工的总班次数。

Step2：模型构建

(1). 目标函数：

目标是最大化正式工的使用，最小化临时工的使用，并最小化所有员工的总工作天数。这通过组合这些变量的线性表达式来实现。

(2). 约束条件：

a. 每个班次的员工能量需求必须满足货量需求。

b. 正式工和临时工的总班次数有限制。

c. 正式工的连续出勤天数有限制。

d. 每个班次至少有正式工参与。

e. 临时工的使用不能超过正式工。

f. 正式工的班次分配需要尽量平均。

Step3：模型求解

(1). 调用 prob.solve() 方法求解问题。

(2). 提取变量的解，即每个班次的正式工和临时工的数量。

Step4：结果处理

(1). 处理求解结果，将每个班次的员工分配情况记录到结果字典中。

(2). 多线程处理：由于需要对多个日期进行处理，使用 multiprocessing.Pool 来并行处理每个日期的排班问题，以提高计算效率。

(3). 将结果字典转换为 DataFrame，并保存到 CSV 文件中。

8.3 问题四求解结果

采用 Python 编程求解，将未来 30 天分拣中心 SC60 每天六个班次的排班结果记录在结果表 6 中。

九、 模型的评价

9.1 模型的优点

- 优点 1：全面的数据处理能力

各模型在数据处理阶段均表现出色，包括数据清洗（异常值处理、缺失值填充）、特征工程（如使用 CNN 进行特征提取）等，确保输入数据的高质量，为模型性能打下坚实基础。

- 优点 2：高效的计算加速技术

利用 GPU 加速和多线程处理技术，显著提高了模型训练和数据处理的速度，减少了时间成本。

- 优点 3: 细致的误差评估与可视化

对预测结果进行了详细的误差评估（如 MSE、MAE 等），并通过图表形式直观展示模型性能，便于理解和改进模型。

9.2 模型的缺点

- 缺点 1: 模型复杂度与资源需求

部分模型（如 BiTCN-LSTM）结构复杂，训练过程中需要较高的计算资源和存储空间，可能不适用于资源受限的环境。

- 缺点 2: 超参数调优困难

多模型包含大量超参数，调优过程复杂且耗时，需要丰富的经验和实验来确定最佳参数组合。

- 缺点 3: 业务约束处理的复杂性

在处理涉及多目标和复杂约束的优化问题时（如排班优化），模型设计需要考虑多种业务规则，增加了模型的复杂度和求解难度。同时，对业务理解的要求也较高，需要深入理解业务逻辑和规则。

参考文献

[1] 罗子健. 迭代学习策略在网络系统及供应链库存中的理论研究与应用[J]. 西南财经大学, 2023.

[2] 龚其国黄文辉. 供应链管理中集中库存研究综述与展望[J]. 管理评论, 2017.

附录 A　文件列表

文件名	功能描述
Q1 随机森林.py	问题一程序代码
Q1BiTCN-LSTM.py	问题一程序代码
Q2 有向图.py	问题二程序代码
Q3 整数规划模型.py	问题三程序代码
Q4 混合规划模型.py	问题四程序代码

附录 B　代码

Q1 随机森林.py

```python
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
from skopt import BayesSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,
   mean_absolute_error
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from statsmodels.tsa.stattools import acf


"""
    : 使用贝叶斯优化随机森林模型进行时间序列预测
"""


dayforecast={"分拣中心":[], "日期":[], "货量":[]}
hourforecast={"分拣中心":[], "日期":[], "小时":[], "货量":[]}
de={"name":[],"mse":[],"mae":[]}
he={"name":[],"mse":[],"mae":[]}
```

```
23
24  # 白噪音检验和异常值替换
25  def detect_and_replace_outliers(data):
26      # 计算自相关系数
27      acf_values = acf(data, nlags=len(data) // 2)
28      # 根据置信水平选择合适的 z 分数
29      confidence_level = 0.90
30      z_score = stats.norm.ppf((1 + confidence_level) / 2)
31      # 检查是否为白噪声
32      is_white_noise = all(abs(value) < z_score / np.sqrt(len(
        data)) for value in acf_values[1:])
33
34      if not is_white_noise:
35          # 使用SVR进行异常值检测与替换
36          svr = SVR(kernel='rbf', C=1e3, gamma=0.1)
37          # 使用除当前点外的所有点来预测当前点
38          X = np.arange(len(data)).reshape(-1, 1)
39          y = data
40
41          svr.fit(X[:-1], y[:-1])
42          predictions = svr.predict(X[1:])
43
44          # 替换异常值
45          for i in range(1, len(data)):
46              if abs(y[i] - predictions[i - 1]) > 2 * np.std(y):
47                  y[i] = predictions[i - 1]
48
49      return y
50
51  #SVR替换缺失值
52  def fill_missing_data_with_svr(df):
53      # 创建完整的日期时间索引
54      date_rng = pd.date_range(start=f"{df['datetime'].min().
        floor('D')} 00:00", end=f"{df['datetime'].max().floor('D')}
        23:00", freq='H')
```

```
    full_index = pd.DataFrame(index=date_rng)
    df = df.set_index('datetime').reindex(full_index.index).reset_index().rename(columns={'index': 'datetime'})

    # 仅选择需要填充的列
    df = df[['datetime', '货量']]

    # 检查是否存在缺失值
    if df['货量'].isnull().any():
        # 使用SVR填充缺失值
        not_null_indices = df['货量'].notnull()
        X = df.loc[not_null_indices, 'datetime'].values.reshape(-1, 1)
        y = df.loc[not_null_indices, '货量'].values

        # 数据标准化
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X.astype(float))

        # 训练SVR模型
        svr = SVR(kernel='rbf', C=1e3, gamma=0.1)
        svr.fit(X_scaled, y)

        # 预测缺失值
        missing_indices = ~df['货量'].notnull()
        X_missing = df.loc[missing_indices, 'datetime'].values.reshape(-1, 1)
        X_missing_scaled = scaler.transform(X_missing.astype(float))
        y_missing_pred = svr.predict(X_missing_scaled)

        # 替换缺失值
        df.loc[missing_indices, '货量'] = y_missing_pred
    else:
        # 如果没有缺失值，则直接返回原始数据集
```

```
        pass

    return df

# 创建时间序列数据集
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# 多步预测函数
def multi_step_forecast(scaler, model, scaled_data,
   forecast_steps, look_back):
    forecast_result = []
    last_window = scaled_data[-look_back:]
    for _ in range(forecast_steps):
        # 预测下一个时间步
        next_step = model.predict(last_window.reshape(1, -1))
        # 将预测结果添加到结果列表
        forecast_result.append(next_step[0])
        # 更新窗口
        last_window = np.roll(last_window, -1)
        last_window[-1] = next_step
    return scaler.inverse_transform(np.array(forecast_result).
   reshape(-1, 1))

# ----------------定义随机森林预测模型----------------
def RandomForest_model_D(df, dfname, forecast_steps):
    # 缩放数据
    scaler = MinMaxScaler(feature_range=(0, 1))

    # 异常值处理
```

```
    cleaned_data = detect_and_replace_outliers(df["货量"].values)
    scaled_data = scaler.fit_transform(cleaned_data.reshape(-1, 1))

    # 数据预处理
    look_back = 30
    X, Y = create_dataset(scaled_data, look_back)

    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

    # 定义随机森林模型
    rf = RandomForestRegressor(random_state=42)

    # 定义参数分布
    param_distributions = {
        'n_estimators': (100, 300),
        'max_depth': (10, 50),
        'min_samples_split': (2, 10),
        'min_samples_leaf': (1, 10),
        'max_features': ['auto', 'sqrt']
    }

    # 使用 BayesSearchCV 执行参数搜索
    bayes_search = BayesSearchCV(
        estimator=rf,
        search_spaces=param_distributions,
        n_iter=40,  # 搜索迭代次数
        cv=5,       # 交叉验证折数
        scoring='neg_mean_squared_error',  # 优化的目标
        n_jobs=1,  # 并行任务数量
        verbose=2  # 输出详细信息
    )
```

```
    bayes_search.fit(X_train, y_train)

    # 获取最佳参数
    best_params = bayes_search.best_params_
    print("Best parameters:", best_params)

    # 使用最佳参数重新训练模型
    best_rf = RandomForestRegressor(**best_params,
random_state=42)
    best_rf.fit(X_train, y_train)

    # 在测试集上进行预测以评估模型性能
    y_test_pred = best_rf.predict(X_test)

    # 反缩放预测结果
    y_test_pred_unscaled = scaler.inverse_transform(
y_test_pred.reshape(-1, 1))
    y_test_unscaled = scaler.inverse_transform(y_test.reshape
(-1, 1))

    # 计算测试集上的误差
    mse_train = mean_squared_error(y_test_unscaled,
y_test_pred_unscaled)
    mae_train = mean_absolute_error(y_test_unscaled,
y_test_pred_unscaled)
    de["name"].append(dfname)
    de["mse"].append(mse_train)
    de["mae"].append(mae_train)
    print(f"Training MSE: {mse_train}")
    print(f"Training MAE: {mae_train}")

    # 绘制模型对原始数据拟合效果的折线图
    plt.figure(figsize=(14, 7))
    plt.plot(y_test_unscaled, label='Actual Test Data', color=
'blue')
```

```
    plt.plot(y_test_pred_unscaled, label='Predicted Test Data', color='green')
    plt.title(f'{dfname} Model Fit on Test Data')
    plt.xlabel('Time Step')
    plt.ylabel('Value')
    plt.legend()
    plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q113\\Q12 {dfname} Model Fit on Test Data.png")
    plt.show()

    # 预测未来30天的数据
    forecast_result_original = multi_step_forecast(scaler, best_rf, scaled_data, forecast_steps, look_back)
    forecast_result_o = [forecast_result_original[i][0] for i in range(len(forecast_result_original))]
    for i in range(len(forecast_result_o)):
        if forecast_result_o[i] < 0:
            forecast_result_o[i] = 0
    dayforecast["货量"].extend(forecast_result_o)
    dayforecast["分拣中心"].extend([dfname]*len(forecast_result_o))

    # 绘制原始数据的时间序列图
    plt.figure(figsize=(14, 7))
    plt.plot(df["日期"], df["货量"], label='Original Data', color='blue')

    # 将预测数据转换为DataFrame并添加到原始DataFrame中
    forecast_df = pd.DataFrame(forecast_result_original, columns=['Forecast'])
    forecast_df['datetime'] = pd.date_range(start=df["日期"].iloc[-1] + pd.Timedelta(days=1), periods=len(forecast_df), freq='1D')
    dayforecast["日期"].extend(list(pd.to_datetime(forecast_df['datetime']).dt.date.replace("-", "/")))
```

```
205
206     # 绘制预测数据的时间序列图
207     plt.plot(forecast_df['datetime'], forecast_df['Forecast'],
        label='Forecast', color='red')
208
209     # 设置图表标题和图例
210     plt.title(f'{dfname} Time Series 30 Days Forecast')
211     plt.xlabel('Date')
212     plt.ylabel('Value')
213     plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q113\\Q11 {
      dfname} Time Series 30 Days Forecast.png")
214     plt.legend()
215
216     # 显示图表
217     plt.show()
218
219
220 def RandomForest_model_H(df, dfname, forecast_steps):
221     # 填充缺失值
222     df = fill_missing_data_with_svr(df)
223
224     # 缩放数据
225     scaler = MinMaxScaler(feature_range=(0, 1))
226     scaled_data = scaler.fit_transform(df["货量"].values.
      reshape(-1, 1))
227
228     # 数据预处理
229     look_back = 30
230     X, Y = create_dataset(scaled_data, look_back)
231
232     # 划分训练集和测试集
233     X_train, X_test, y_train, y_test = train_test_split(X, Y,
      test_size=0.2, random_state=42)
234
235     # 定义随机森林模型
```

```
    rf = RandomForestRegressor(random_state=42)

    # 定义参数分布
    param_distributions = {
        'n_estimators': (100, 300),
        'max_depth': (10, 50),
        'min_samples_split': (2, 10),
        'min_samples_leaf': (1, 6),
        'max_features': ['auto', 'sqrt']
    }

    # 使用 BayesSearchCV 执行参数搜索
    bayes_search = BayesSearchCV(
        estimator=rf,
        search_spaces=param_distributions,
        n_iter=40,  # 搜索迭代次数
        cv=5,       # 交叉验证折数
        scoring='neg_mean_squared_error',  # 优化的目标
        n_jobs=1,  # 并行任务数量
        verbose=2  # 输出详细信息
    )
    bayes_search.fit(X_train, y_train)

    # 获取最佳参数
    best_params = bayes_search.best_params_
    print("Best parameters:", best_params)

    # 使用最佳参数重新训练模型
    best_rf = RandomForestRegressor(**best_params,
random_state=42)
    best_rf.fit(X_train, y_train)

    # 在测试集上进行预测以评估模型性能
    y_test_pred = best_rf.predict(X_test)

```

```
    # 反缩放预测结果
    y_test_pred_unscaled = scaler.inverse_transform(y_test_pred.reshape(-1, 1))
    y_test_unscaled = scaler.inverse_transform(y_test.reshape(-1, 1))

    # 计算测试集上的误差
    mse_train = mean_squared_error(y_test_unscaled, y_test_pred_unscaled)
    mae_train = mean_absolute_error(y_test_unscaled, y_test_pred_unscaled)
    he["name"].append(dfname)
    he["mse"].append(mse_train)
    he["mae"].append(mae_train)
    print(f"Training MSE: {mse_train}")
    print(f"Training MAE: {mae_train}")

    # 绘制模型对原始数据拟合效果的折线图
    plt.figure(figsize=(14, 7))
    plt.plot(y_test_unscaled, label='Actual Test Data', color='blue')
    plt.plot(y_test_pred_unscaled, label='Predicted Test Data', color='green')
    plt.title(f'{dfname} Model Fit on Test Data')
    plt.xlabel('Time Step')
    plt.ylabel('Value')
    plt.legend()
    plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q114\\Q12 {dfname} Model Fit on Test Data.png")
    plt.show()

    # 预测未来30天每小时的数据
    forecast_result_original = multi_step_forecast(scaler, best_rf , scaled_data, forecast_steps, look_back)
    # 整理数据，小于0的变为0
```

```
    for i in range(len(forecast_result_original)):
        if forecast_result_original[i][0]<0:
            forecast_result_original[i][0]=0

    forecast_result_o=[forecast_result_original[i][0] for i in range(len(forecast_result_original))]
    hourforecast["货量"].extend(forecast_result_o)
    hourforecast["分拣中心"].extend([dfname] * len(forecast_result_o))

    # 绘制原始数据的时间序列图
    plt.figure(figsize=(14, 7))
    plt.plot(df['datetime'], df["货量"], label='Original Data', color='blue')

    # 将预测数据转换为DataFrame并添加到原始DataFrame中
    forecast_df = pd.DataFrame(forecast_result_original, columns=['Forecast'])
    forecast_df ['datetime'] = pd.date_range(start=df['datetime'].iloc[-1] + pd.Timedelta(hours=1), periods=len(forecast_df), freq='1H')
    hourforecast["日期"].extend(list(pd.to_datetime(forecast_df['datetime']).dt.date.replace("-", "/")))
    hourforecast["小时"].extend(list(pd.to_datetime(forecast_df['datetime']).dt.hour))

    # 绘制预测数据的时间序列图
    plt.plot(forecast_df['datetime'], forecast_df['Forecast'], label='Forecast', color='red')

    # 设置图表标题和图例
    plt.title(f'{dfname} Time Series 720 Hours Forecast')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q114\\Q12 {
```

```
    dfname} Time Series 720 Hours Forecast.png")
323     plt.legend()
324
325     # 显示图表
326     plt.show()
327
328 # ----------------预测30天数据----------------
329 df_day=pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附件
    1.csv", encoding='GBK')
330 df_day["日期"] = pd.to_datetime(df_day["日期"])
331 df_day=df_day.groupby("分拣中心").apply(lambda x: x.
    sort_values('日期')).reset_index(drop=True)
332 groupnamed=list(df_day.groupby("分拣中心").groups.keys())
333 # 预测每个分拣中心30天数据
334 for i in groupnamed:
335     df_group=df_day[df_day["分拣中心"] == i]
336     RandomForest_model_D(df_group, i, 30)
337
338 # 保存预测30天数据
339 pd.DataFrame(dayforecast).to_csv("F:\\Python数模\\暑期培训
    \\2406\\结果\\结果表1.csv", index=False, encoding='GBK')
340 pd.DataFrame(de).to_csv("Q11day误差.csv", index=False)
341
342
343 # ----------------预测30天每小时数据----------------
344 df_hour=pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附
    件2.csv", encoding='GBK')
345 df_hour['datetime'] = df_hour.apply(lambda row: pd.to_datetime
    (f"{row['日期']} {row['小时']}:00"), axis=1)
346 df_hour=df_hour.groupby("分拣中心").apply(lambda x: x.
    sort_values('datetime')).reset_index(drop=True)
347 groupnameh=list(df_hour.groupby("分拣中心").groups.keys())
348 # 预测每个分拣中心30天每小时数据
349 for i in groupnameh:
350     df1_group=df_hour[df_hour["分拣中心"] == i]
```

```
    RandomForest_model_H(df1_group, i, 720)

# 保存预测30天每小时数据
pd.DataFrame(hourforecast).to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表2.csv", index=False, encoding='GBK')
pd.DataFrame(he).to_csv("Q11hour误差.csv", index=False)
```

Q1BiTCN-LSTM.py

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.layers import LSTM, Dense, Bidirectional, Conv1D
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.svm import SVR
from statsmodels.tsa.stattools import acf
from sklearn.metrics import mean_squared_error, mean_absolute_error
import scipy.stats as stats
import matplotlib.pyplot as plt
import tensorflow as tf
import os


"""
    : 使用BiTCN-LSTM双向时间卷积神经网络结合长短期记忆神经网络模型进行时间序列预测
"""


# --------------使用GPU加速---------------
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
```

```
    try:
        tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
    except RuntimeError as e:
        print(e)
else:
    print("No GPU available.")

# ----------------设置TensorFlow的线程数---------------
os.environ['TF_NUM_INTRAOP_THREADS'] = '24'

dayforecast={"分拣中心":[], "日期":[], "货量":[]}
hourforecast={"分拣中心":[], "日期":[], "小时":[], "货量":[]}
de={"name":[],"mse":[],"mae":[]}
he={"name":[],"mse":[],"mae":[]}


# 创建监督学习问题
def create_dataset(dataset, look_back=1):
    X, Y = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

# 白噪音检验和异常值替换
def detect_and_replace_outliers(data):
    # 计算自相关系数
    acf_values = acf(data, nlags=len(data) // 2)
    # 根据置信水平选择合适的 z 分数
```

```
    confidence_level = 0.90
    z_score = stats.norm.ppf((1 + confidence_level) / 2)
    # 检查是否为白噪声
    is_white_noise = all(abs(value) < z_score / np.sqrt(len(data)) for value in acf_values[1:])

    if not is_white_noise:
        # 使用SVR进行异常值检测与替换
        svr = SVR(kernel='rbf', C=1e3, gamma=0.1)
        # 使用除当前点外的所有点来预测当前点
        X = np.arange(len(data)).reshape(-1, 1)
        y = data

        svr.fit(X[:-1], y[:-1])
        predictions = svr.predict(X[1:])

        # 替换异常值
        for i in range(1, len(data)):
            if abs(y[i] - predictions[i - 1]) > 2 * np.std(y):
                y[i] = predictions[i - 1]

    return y

#SVR替换缺失值
def fill_missing_data_with_svr(df):
    # 创建完整的日期时间索引
    date_rng = pd.date_range(start=f"{df['datetime'].min().floor('D')} 00:00", end=f"{df['datetime'].max().floor('D')} 23:00", freq='H')
    full_index = pd.DataFrame(index=date_rng)
    df = df.set_index('datetime').reindex(full_index.index).reset_index().rename(columns={'index': 'datetime'})

    # 仅选择需要填充的列
    df = df[['datetime', '货量']]
```

```
88
89      # 检查是否存在缺失值
90      if df['货量'].isnull().any():
91          # 使用SVR填充缺失值
92          not_null_indices = df['货量'].notnull()
93          X = df.loc[not_null_indices, 'datetime'].values.
    reshape(-1, 1)
94          y = df.loc[not_null_indices, '货量'].values
95
96          # 数据标准化
97          scaler = StandardScaler()
98          X_scaled = scaler.fit_transform(X.astype(float))
99
100         # 训练SVR模型
101         svr = SVR(kernel='rbf', C=1e3, gamma=0.1)
102         svr.fit(X_scaled, y)
103
104         # 预测缺失值
105         missing_indices = ~df['货量'].notnull()
106         X_missing = df.loc[missing_indices, 'datetime'].values
    .reshape(-1, 1)
107         X_missing_scaled = scaler.transform(X_missing.astype(
    float))
108         y_missing_pred = svr.predict(X_missing_scaled)
109
110         # 替换缺失值
111         df.loc[missing_indices, '货量'] = y_missing_pred
112     else:
113         # 如果没有缺失值，则直接返回原始数据集
114         pass
115
116     return df
117
118
119 # 多步预测函数，预测未来30天每小时的数据
```

```
120 def multi_step_forecast(scaler, model, scaled_data, steps,
    look_back):
121     current_data = scaled_data[-look_back:]
122     forecast_data = []
123     for _ in range(steps):
124         # 预测下一个值
125         predicted = model.predict(current_data.reshape(1, 1,
    look_back))
126         forecast_data.append(predicted[0][0])
127         current_data = np.roll(current_data, -1)
128         current_data[-1] = predicted[0][0]
129     return scaler.inverse_transform(np.array(forecast_data).
    reshape(-1, 1))
130
131
132 # ----------------定义BiTCN-LSTM天模型----------------
133 def BiTCN_LSTM_model_D(df, dfname, forecast_steps):
134     # 缩放数据
135     scaler = MinMaxScaler(feature_range=(0, 1))
136
137     # 异常值处理
138     cleaned_data = detect_and_replace_outliers(df["货量"].
    values)
139     scaled_data = scaler.fit_transform(cleaned_data.reshape
    (-1, 1))
140
141     # 数据预处理
142     look_back = 30
143     X, Y = create_dataset(scaled_data, look_back)
144
145     # 划分训练集和测试集
146     X_train, X_test, y_train, y_test = train_test_split(X, Y,
    test_size=0.2, random_state=42)
147
148     # 重塑输入数据为[samples, time steps, features]
```

```
149     X_train = np.reshape(X_train, (X_train.shape[0], 1,
    X_train.shape[1]))
150     X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.
    shape[1]))
151
152     # 构建BiTCN-LSTM模型
153     model = Sequential()
154
155     # 添加多个时间卷积层
156     model.add(Conv1D(filters=32, kernel_size=3, activation='
    relu', padding='same', input_shape=(1, look_back)))
157     model.add(Conv1D(filters=64, kernel_size=5, activation='
    relu', padding='same'))
158     model.add(Conv1D(filters=128, kernel_size=7, activation='
    relu', padding='same'))
159     model.add(Conv1D(filters=256, kernel_size=9, activation='
    relu', padding='same'))
160     model.add(Conv1D(filters=512, kernel_size=11, activation='
    relu', padding='same'))
161
162     # 添加双向LSTM层
163     model.add(Bidirectional(LSTM(50, input_shape=(1, look_back
    ), return_sequences=True)))
164     model.add(Bidirectional(LSTM(50)))
165     model.add(Dense(1))
166     model.compile(loss='mean_squared_error', optimizer='adam')
167
168     # 训练模型
169     model.fit(X_train, y_train, epochs=200, batch_size=32,
    verbose=2)
170     model.save(f"F:\\Python数模\\暑期培训\\2406\\Q111m\\Q11{
    dfname}.h5")
171     # model = load_model(f"F:\\Python数模\\暑期培训\\2406\\
    Q111m\\Q11{dfname}.h5")
172
```

```
    # 在测试集上进行预测以评估模型性能
    y_test_pred = model.predict(X_test)

    # 反缩放预测结果
    y_test_pred_unscaled = scaler.inverse_transform(y_test_pred)
    y_test_unscaled = scaler.inverse_transform(y_test.reshape(-1, 1))

    # 计算测试集上的误差
    mse_train = mean_squared_error(y_test_unscaled, y_test_pred_unscaled)
    mae_train = mean_absolute_error(y_test_unscaled, y_test_pred_unscaled)
    de["name"].append(dfname)
    de["mse"].append(mse_train)
    de["mae"].append(mae_train)
    print(f"Training MSE: {mse_train}")
    print(f"Training MAE: {mae_train}")

    # 绘制模型对原始数据拟合效果的折线图
    plt.figure(figsize=(14, 7))
    plt.plot(y_test_unscaled, label='Actual Test Data', color='blue')
    plt.plot(y_test_pred_unscaled, label='Predicted Test Data', color='green')
    plt.title(f'{dfname} Model Fit on Test Data')
    plt.xlabel('Time Step')
    plt.ylabel('Value')
    plt.legend()
    plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q111\\Q12 {dfname} Model Fit on Test Data.png")
    #plt.show()

    # 预测未来30天的数据
```

```
    forecast_result_original = multi_step_forecast(scaler, model, scaled_data, forecast_steps, look_back)
    forecast_result_o=[forecast_result_original[i][0] for i in range(len(forecast_result_original))]
    for i in range(len(forecast_result_o)):
        if forecast_result_o[i]<0:
            forecast_result_o[i]=0
    dayforecast["货量"].extend(forecast_result_o)
    dayforecast["分拣中心"].extend([dfname]*len(forecast_result_o))

    # 绘制原始数据的时间序列图
    plt.figure(figsize=(14, 7))
    plt.plot(df["日期"], df["货量"], label='Original Data', color='blue')

    # 将预测数据转换为DataFrame并添加到原始DataFrame中
    forecast_df = pd.DataFrame(forecast_result_original, columns=['Forecast'])
    forecast_df['datetime'] = pd.date_range(start=df["日期"].iloc[-1] + pd.Timedelta(days=1), periods=len(forecast_df), freq='1D')
    dayforecast["日期"].extend(list(pd.to_datetime(forecast_df['datetime']).dt.date.replace("-", "/")))

    # 绘制预测数据的时间序列图
    plt.plot(forecast_df['datetime'], forecast_df['Forecast'], label='Forecast', color='red')

    # 设置图表标题和图例
    plt.title(f'{dfname} Time Series 30 Days Forecast')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q111\\Q11 {dfname} Time Series 30 Days Forecast.png")
```

```
    plt.legend()

    # 显示图表
    #plt.show()


# ----------------定义BiTCN-LSTM小时模型----------------
def BiTCN_LSTM_model_H(df, dfname, forecast_steps):
    # 填充缺失值
    df = fill_missing_data_with_svr(df)

    # 缩放数据
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(df["货量"].values.reshape(-1, 1))

    # 数据预处理
    look_back = 24
    X, Y = create_dataset(scaled_data, look_back)

    # 划分训练集和测试集
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

    # 重塑输入数据为[samples, time steps, features]
    X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
    X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

    # 构建BiTCN-LSTM模型
    model = Sequential()

    # 添加多个时间卷积层
    model.add(Conv1D(filters=32, kernel_size=3, activation='
```

```
relu', padding='same', input_shape=(1, look_back)))
257  model.add(Conv1D(filters=64, kernel_size=5, activation='
relu', padding='same'))
258  model.add(Conv1D(filters=128, kernel_size=7, activation='
relu', padding='same'))
259  model.add(Conv1D(filters=256, kernel_size=9, activation='
relu', padding='same'))
260  model.add(Conv1D(filters=512, kernel_size=11, activation='
relu', padding='same'))
261
262  # 添加双向LSTM层
263  model.add(Bidirectional(LSTM(50, input_shape=(1, look_back
), return_sequences=True)))
264  model.add(Bidirectional(LSTM(50)))
265  model.add(Dense(1))
266  model.compile(loss='mean_squared_error', optimizer='adam')
267
268  # 训练模型
269  model.fit(X_train, y_train, epochs=200, batch_size=32,
verbose=2)
270  model.save(f"F:\\Python数模\\暑期培训\\2406\\Q112m\\Q11{
dfname}.h5")
271  # model = load_model(f"F:\\Python数模\\暑期培训\\2406\\
Q111m\\Q11{dfname}.h5")
272
273  # 在测试集上进行预测以评估模型性能
274  y_test_pred = model.predict(X_test)
275
276  # 反缩放预测结果
277  y_test_pred_unscaled = scaler.inverse_transform(
y_test_pred)
278  y_test_unscaled = scaler.inverse_transform(y_test.reshape
(-1, 1))
279
280  # 计算测试集上的误差
```

```
281     mse_train = mean_squared_error(y_test_unscaled,
    y_test_pred_unscaled)
282     mae_train = mean_absolute_error(y_test_unscaled,
    y_test_pred_unscaled)
283     he["name"].append(dfname)
284     he["mse"].append(mse_train)
285     he["mae"].append(mae_train)
286     print(f"Training MSE: {mse_train}")
287     print(f"Training MAE: {mae_train}")
288 
289     # 绘制模型对原始数据拟合效果的折线图
290     plt.figure(figsize=(14, 7))
291     plt.plot(y_test_unscaled, label='Actual Test Data', color=
    'blue')
292     plt.plot(y_test_pred_unscaled, label='Predicted Test Data'
    , color='green')
293     plt.title(f'{dfname} Model Fit on Test Data')
294     plt.xlabel('Time Step')
295     plt.ylabel('Value')
296     plt.legend()
297     plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q112\\Q12 {
    dfname} Model Fit on Test Data.png")
298     #plt.show()
299 
300     # 预测未来30天每小时的数据
301     forecast_result_original = multi_step_forecast(scaler,
    model, scaled_data, forecast_steps, look_back)
302     # 整理数据，小于0的变为0
303     for i in range(len(forecast_result_original)):
304         if forecast_result_original[i][0]<0:
305             forecast_result_original[i][0]=0
306 
307     forecast_result_o=[forecast_result_original[i][0] for i in
     range(len(forecast_result_original))]
308     hourforecast["货量"].extend(forecast_result_o)
```

```
309     hourforecast["分拣中心"].extend([dfname] * len(
        forecast_result_o))
310
311     # 绘制原始数据的时间序列图
312     plt.figure(figsize=(14, 7))
313     plt.plot(df['datetime'], df["货量"], label='Original Data'
        , color='blue')
314
315     # 将预测数据转换为DataFrame并添加到原始DataFrame中
316     forecast_df = pd.DataFrame(forecast_result_original,
        columns=['Forecast'])
317     forecast_df ['datetime'] = pd.date_range(start=df['
        datetime'].iloc[-1] + pd.Timedelta(hours=1), periods=len(
        forecast_df), freq='1H')
318     hourforecast["日期"].extend(list(pd.to_datetime(
        forecast_df['datetime']).dt.date.replace("-", "/")))
319     hourforecast["小时"].extend(list(pd.to_datetime(
        forecast_df['datetime']).dt.hour))
320
321     # 绘制预测数据的时间序列图
322     plt.plot(forecast_df['datetime'], forecast_df['Forecast'],
         label='Forecast', color='red')
323
324     # 设置图表标题和图例
325     plt.title(f'{dfname} Time Series 720 Hours Forecast')
326     plt.xlabel('Date')
327     plt.ylabel('Value')
328     plt.savefig(f"F:\\Python数模\\暑期培训\\2406\\Q112\\Q12 {
        dfname} Time Series 720 Hours Forecast.png")
329     plt.legend()
330
331     # 显示图表
332     #plt.show()
333
334 # ----------------预测30天数据----------------
```

```
df_day=pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附件1.csv", encoding='GBK')
df_day["日期"] = pd.to_datetime(df_day["日期"])
df_day=df_day.groupby("分拣中心").apply(lambda x: x.sort_values('日期')).reset_index(drop=True)
groupnamed=list(df_day.groupby("分拣中心").groups.keys())
# 预测每个分拣中心30天数据
for i in groupnamed:
    df_group=df_day[df_day["分拣中心"] == i]
    BiTCN_LSTM_model_D(df_group, i, 30)

# 保存预测30天数据
pd.DataFrame(dayforecast).to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表1(2).csv", index=False, encoding='GBK')
pd.DataFrame(de).to_csv("Q1day误差.csv", index=False)

# ----------------预测30天每小时数据----------------
df_hour=pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附件2.csv", encoding='GBK')
df_hour['datetime'] = df_hour.apply(lambda row: pd.to_datetime(f"{row['日期']} {row['小时']}:00"), axis=1)
df_hour=df_hour.groupby("分拣中心").apply(lambda x: x.sort_values('datetime')).reset_index(drop=True)
groupnameh=list(df_hour.groupby("分拣中心").groups.keys())
# 预测每个分拣中心30天每小时数据
for i in groupnameh:
    df1_group=df_hour[df_hour["分拣中心"] == i]
    BiTCN_LSTM_model_H(df1_group, i, 720)

# 保存预测30天每小时数据
pd.DataFrame(hourforecast).to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表2(2).csv", index=False, encoding='GBK')
pd.DataFrame(he).to_csv("Q1hour误差.csv", index=False)
```

Q2 有向图.py

```python
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# 计算出边的总权重
def sum_outgoing_weight(G, node):
    if node not in G:
        return 0
    outgoing_weights = [G[node][v]['weight'] for v in G.successors(node)]
    return sum(outgoing_weights) if outgoing_weights else 0

# 计算入边的总权重
def sum_incoming_weight(G, node):
    if node not in G:
        return 0
    incoming_weights = [G[u][node]['weight'] for u in G.predecessors(node)]
    return sum(incoming_weights) if incoming_weights else 0

# 计算出边的平均权重
def average_outgoing_weight(G, node):
    if node not in G:
        return 0
    outgoing_weights = [G[node][v]['weight'] for v in G.successors(node)]
    return sum(outgoing_weights) / len(outgoing_weights) if outgoing_weights else 0

# 计算入边的平均权重
def average_incoming_weight(G, node):
    if node not in G:
        return 0
    incoming_weights = [G[u][node]['weight'] for u in G.predecessors(node)]
```

```
    return sum(incoming_weights) / len(incoming_weights) if incoming_weights else 0

if __name__ == '__main__':
    df_day = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表1.csv", encoding='GBK')
    df_hour = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表2.csv", encoding='GBK')
    df_route_origin = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附件3.csv", encoding='GBK')
    df_route_future = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\附件\\附件4.csv", encoding='GBK')
    graph={"start":[],"end":[],"weight":[]}

    # 构建边列表
    edges_origin = [(row['始发分拣中心'], row['到达分拣中心'], {'weight': row['货量']}) for _, row in df_route_origin.iterrows()]
    edges = [(row['始发分拣中心'], row['到达分拣中心']) for _, row in df_route_future.iterrows()]

    # 创建有向图
    G1 = nx.DiGraph()

    # 添加边和权重
    G1.add_edges_from(edges_origin)

    # 创建有向图
    G2 = nx.DiGraph()

    # 添加边和权重
    G2.add_edges_from(edges)

    # 更新图 G1 成为 G2 的形式
    # 删除 G1 中不在 G2 中的边
```

```
    edges_to_remove = [(u, v) for u, v in G1.edges() if (u, v) not in edges]
    G1.remove_edges_from(edges_to_remove)

    # 添加 G2 中存在但在 G1 中不存在的新边
    new_edges = [edge for edge in edges if edge not in G1.edges()]
    for u, v in new_edges:
        weight = (average_outgoing_weight(G1, u) + average_incoming_weight(G1, v)) / 2
        G1.add_edge(u, v, weight=weight)

    # 更新已存在的边的权重
    for u, v in G1.edges():
        weight = (average_outgoing_weight(G1, u) + average_incoming_weight(G1, v)) / 2
        G1[u][v]['weight'] = weight

    # 删除没有边的点
    isolated_nodes = [n for n in G1.nodes() if G1.degree(n) == 0]
    G1.remove_nodes_from(isolated_nodes)

    # 输出 G1 的节点和边
    print("Nodes in G1:", G1.nodes())
    print("Edges in G1:", G1.edges(data=True))

    # 存储节点和边的信息
    for i in G1.edges(data=True):
        graph["start"].append(i[0])
        graph["end"].append(i[1])
        graph["weight"].append(i[2]["weight"])

    pd.DataFrame(graph).to_csv("graph.csv", encoding='GBK')

```

```
    # 绘制有向图
    plt.figure(figsize=(20, 20))
    pos = nx.circular_layout(G1)  # 使用圆形布局
    node_colors = ['lightblue' for _ in G1.nodes()]  # 节点颜色
    edge_colors = ['black' for _ in G1.edges()]  # 边的颜色
    edge_widths = [G1[u][v]['weight'] * 0.01 for u, v in G1.edges()]  # 根据权重调整边的宽度

    nx.draw(G1, pos, with_labels=True, node_color=node_colors, node_size=500, font_size=10,
            arrows=True, arrowstyle='-|>', arrowsize=20, edge_color=edge_colors, width=edge_widths)
    edge_labels = {(u, v): f"{G1[u][v]['weight']:.2f}" for u, v in G1.edges()}  # 保留两位有效数字
    nx.draw_networkx_edge_labels(G1, pos, edge_labels=edge_labels, font_size=6, label_pos=0.5)
    plt.title('Circular Layout of Directed Graph G1')
    plt.savefig("Q2graph.png", dpi=500)
    plt.show()

    # 计算每个分拣中心的出度和入度
    groupnamed=list(df_day.groupby("分拣中心").groups.keys())
    groupnameh=list(df_hour.groupby("分拣中心").groups.keys())

    for i in groupnamed:
        df_day.loc[df_day["分拣中心"] == i, "货量"] += sum_incoming_weight(G1,i) - sum_outgoing_weight(G1,i)

    df_day.to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表3.csv", encoding='GBK', index=False)

    for i in groupnameh:
        df_hour.loc[df_hour["分拣中心"] == i,"货量"] += sum_incoming_weight(G1,i) - sum_outgoing_weight(G1,i)
```

```
114
115     df_hour.to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表4.csv", encoding='GBK', index=False)
```

Q3 整数规划模型.py

```
1  import pulp
2  import pandas as pd
3  from multiprocessing import Pool
4
5
6  # 双目标整数规划模型
7  def dual_objective_integer_programming_model(df):
8      max_regular = 60
9      worker_energy = {'Regular': 25, 'Temp': 20}
10     package_demands = df["货量"].values.tolist()
11
12     # 班次
13     flag = [
14         [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
15         [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
16         [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
17         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
18         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
19         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
20     ]
21
22     # 创建问题实例
23     prob = pulp.LpProblem("Package_Processing_Scheduling", pulp.LpMinimize)
```

```
24
25      # 正式工
26      x_vars = [pulp.LpVariable(f"正式工班次{i}", lowBound=0,
        cat='Integer') for i in range(1, 7)]
27      """
28      # 正式工
29      x1 = pulp.LpVariable("正式工班次1", lowBound=0, cat='
        Integer')
30      x2 = pulp.LpVariable("正式工班次2", lowBound=0, cat='
        Integer')
31      x3 = pulp.LpVariable("正式工班次3", lowBound=0, cat='
        Integer')
32      x4 = pulp.LpVariable("正式工班次4", lowBound=0, cat='
        Integer')
33      x5 = pulp.LpVariable("正式工班次5", lowBound=0, cat='
        Integer')
34      x6 = pulp.LpVariable("正式工班次6", lowBound=0, cat='
        Integer')
35      x_vars = [x1, x2, x3, x4, x5, x6]
36      """
37      # 临时工
38      y_vars = [pulp.LpVariable(f"临时工班次{i}", lowBound=0,
        cat='Integer') for i in range(1, 7)]
39      """
40      # 临时工
41      y1 = pulp.LpVariable("临时工班次1", lowBound=0, cat='
        Integer')
42      y2 = pulp.LpVariable("临时工班次2", lowBound=0, cat='
        Integer')
43      y3 = pulp.LpVariable("临时工班次3", lowBound=0, cat='
        Integer')
44      y4 = pulp.LpVariable("临时工班次4", lowBound=0, cat='
        Integer')
45      y5 = pulp.LpVariable("临时工班次5", lowBound=0, cat='
        Integer')
```

```
    y6 = pulp.LpVariable("临时工班次6", lowBound=0, cat='Integer')

    # 将它们组合回列表，如果需要的话
    y_vars = [y1, y2, y3, y4, y5, y6]
    """
    # 正式工总数
    total_regular_workers = pulp.LpVariable("TotalRegularWorkers", lowBound=0, cat='Integer')

    # 最小化临时工的使用
    prob += pulp.lpSum([0.01 * x for x in x_vars])+pulp.lpSum(y_vars)

    # 约束条件
    for i in range(24):
        prob += (
            (worker_energy["Regular"] * sum(x * f[i] for x, f in zip(x_vars, flag)) +
             worker_energy["Temp"] * sum(y * f[i] for y, f in zip(y_vars, flag))) >=
            package_demands[i]
        )

    """
    for i in range(24):
    prob += (
        (worker_energy["Regular"] * (x1 * flag[0][i] + x2 * flag[1][i] + x3 * flag[2][i] + x4 * flag[3][i] + x5 * flag[4][i] + x6 * flag[5][i]) +
         worker_energy["Temp"] * (y1 * flag[0][i] + y2 * flag[1][i] + y3 * flag[2][i] + y4 * flag[3][i] + y5 * flag[4][i] + y6 * flag[5][i])) >=
        package_demands[i]
    )
```

```
72      """
73
74      # 限制正式工的使用数量
75      prob += total_regular_workers == pulp.lpSum(x_vars)
76      prob += total_regular_workers <= max_regular
77
78      # 对每个临时工的使用引入额外约束
79      M = 1e6  # 选择一个合适的足够大的M值
80      for i in range(1, 7):
81          # 当正式工总数未达上限时，允许使用临时工
82          prob += y_vars[i-1] <= max_regular -
    total_regular_workers + M
83
84      solver = pulp.PULP_CBC_CMD(msg=True, timeLimit=300)  # 限
    制求解时间为300秒
85      prob.solve(solver=solver)
86
87      # 返回结果
88      return [var.varValue for var in x_vars + y_vars]
89
90
91  # 读取随机生成的数据
92  df_hour = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\结果\\
    结果表4.csv", encoding='GBK')
93  df_hour["货量"]=df_hour["货量"].round()
94
95  # 分组
96  groupnameh = list(df_hour.groupby("分拣中心").groups.keys())
97  during = ["00:00-08:00", "05:00-13:00", "08:00-16:00", "
    12:00-20:00", "14:00-22:00", "16:00-24:00"]
98
99  # 初始化结果字典
100 results = {"分拣中心": [], "日期": [], "班次": [], "正式工人数
    ": [], "临时工人数": []}
101
```

```python
if __name__ == '__main__':
    # 多线程处理
    with Pool() as pool:
        for i in groupnameh:
            groupnamed = list(df_hour.groupby("日期").groups.keys())
            for j in groupnamed:
                df_hour_ij = df_hour.query("分拣中心 == @i and 日期 == @j")
                print(df_hour_ij)
                # 使用多线程处理
                results_list = pool.map(dual_objective_integer_programming_model, [df_hour_ij] * 6)
                for k in range(6):
                    results["分拣中心"].append(i)
                    results["日期"].append(j)
                    results["班次"].append(during[k])
                    results["正式工人数"].append(results_list[k][k])
                    results["临时工人数"].append(results_list[k][k + 6])

    # 保存结果
    pd.DataFrame(results).to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表5.csv", index=False, encoding='GBK')
```

Q4 混合规划模型.py

```python
import pulp
import pandas as pd
from multiprocessing import Pool


# 混合整数模型
def model_day(df, regular_workers, max_regular_shifts=6, max_consecutive_days=7):
```

```
    worker_energy = {'Regular': 25, 'Temp': 20}
    package_demands = df["货量"].values.tolist()
    flag = [
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
    ]

    # 创建问题实例
    prob = pulp.LpProblem("Package_Processing_Scheduling", pulp.LpMinimize)

    # 定义变量
    x_vars = [pulp.LpVariable(f"正式工班次{i}", lowBound=0, cat='Integer') for i in range(1, 7)]
    y_vars = [pulp.LpVariable(f"临时工班次{i}", lowBound=0, cat='Integer') for i in range(1, 7)]
    regular_days = [pulp.LpVariable(f"正式工出勤天数{i}", lowBound=22, upBound=25, cat='Integer') for i in range(regular_workers)]  # 出勤天数限制
    regular_shifts = [[pulp.LpVariable(f"正式工{i}_班次{j}", lowBound=0, upBound=1, cat='Binary') for j in range(1, 7)] for i in range(regular_workers)]
    regular_shift_counts = [pulp.LpVariable(f"正式工{i}_班次数", lowBound=0, upBound=max_regular_shifts, cat='Integer') for i in range(regular_workers)]
```

```
28
29      # 计算每小时的人效
30      hour_efficiency = [worker_energy["Regular"] * sum(sum(x *
        f[i] for x, f in zip(worker_shifts, flag)) for worker_shifts
        in regular_shifts) + worker_energy["Temp"] * sum(y * f[i]
        for y, f in zip(y_vars, flag)) for i in range(24)]
31      total_efficiency = pulp.lpSum(worker_energy["Regular"] *
        sum(sum(x * f[i] for x, f in zip(worker_shifts, flag)) for
        worker_shifts in regular_shifts) for i in range(24))
32      avg_efficiency = pulp.lpSum(hour_efficiency) / 24
33
34      # 效率差异
35      efficiency_diff_abs = pulp.LpVariable.dicts("
        EfficiencyDiffAbs", range(24), lowBound=0, cat='Continuous')
36      M = 1000
37
38      # 效率差异约束
39      for i in range(24):
40          diff = hour_efficiency[i] - avg_efficiency
41          abs_diff_var = pulp.LpVariable(f"AbsDiff_{i}",
        lowBound=0, cat='Continuous')
42          binary_var = pulp.LpVariable(f"Binary_{i}", lowBound
        =0, upBound=1, cat='Binary')
43          prob += abs_diff_var >= diff - M * (1 - binary_var)
44          prob += abs_diff_var <= diff + M * binary_var
45          prob += efficiency_diff_abs[i] == abs_diff_var
46
47      # 工作天数差异
48      avg_days = pulp.lpSum(regular_days) / regular_workers
49      days_diff_abs = pulp.LpVariable.dicts("DaysDiffAbs", range
        (regular_workers), lowBound=0, cat='Continuous')
50      for i in range(regular_workers):
51          diff = regular_days[i] - avg_days
52          abs_diff_var = pulp.LpVariable(f"AbsDiffDays_{i}",
        lowBound=0, cat='Continuous')
```

```
        binary_var = pulp.LpVariable(f"BinaryDays_{i}", lowBound=0, upBound=1, cat='Binary')
        prob += abs_diff_var >= diff - M * (1 - binary_var)
        prob += abs_diff_var <= diff + M * binary_var
        prob += days_diff_abs[i] == abs_diff_var

    # 总出勤天数
    total_days = pulp.lpSum(regular_days)

    # 添加正式工出勤总人数的约束
    prob += (total_days >= 4000)
    prob += (total_days <= 5000)

    # 修改目标函数：最大化正式工的数量，最小化临时工的数量，并最小化正式工和临时工的总工作天数
    prob += (-100000 * pulp.lpSum(x_vars) + 10000 * pulp.lpSum(y_vars) + 1000 * pulp.lpSum(regular_days) + 1000 * pulp.lpSum(y_vars))

    # 确保所有正式工都出勤
    for i in range(regular_workers):
        prob += (regular_days[i] >= 22)  # 正式工出勤天数下限
        prob += (regular_days[i] <= 25)  # 正式工出勤天数上限

    # 其他约束条件
    for i in range(24):
        prob += ( (worker_energy["Regular"] * sum(sum(x * f[i] for x, f in zip(worker_shifts, flag)) for worker_shifts in regular_shifts) + worker_energy["Temp"] * sum(y * f[i] for y, f in zip(y_vars, flag))) >= package_demands[i])

    prob += (pulp.lpSum(x_vars) <= max_regular_shifts)

    # **确保正式工优先调度**：通过约束确保正式工尽可能被使用
    for i in range(regular_workers):
```

```
81          prob += (regular_days[i] <= max_consecutive_days)
82          prob += (sum(regular_shifts[i]) <= 1)
83          prob += (regular_shift_counts[i] == sum([
    regular_shifts[i][j - 1] for j in range(1, 7)]))
84          prob += (regular_shift_counts[i] <= max_regular_shifts
    )
85
86      # **确保每个班次至少有正式工参与**
87      for j in range(6):
88          prob += (x_vars[j] >= 1)
89          prob += (x_vars[j] == sum([regular_shifts[i][j] for i
    in range(regular_workers)]))
90
91      for i in range(regular_workers):
92          prob += (regular_days[i] == sum([regular_shifts[i][j -
     1] for j in range(1, 7)]))
93
94      # **确保首先使用正式员工**
95      for j in range(6):
96          prob += (y_vars[j] <= x_vars[j])
97
98      # 计算平均每个班次需要的正式工人次
99      average_shifts_per_worker = max_regular_shifts /
    regular_workers
100     average_shifts_per_day_per_slot =
    average_shifts_per_worker / 6
101
102     # 为每个班次添加平均分配约束
103     for j in range(6):
104         actual_shifts_in_slot = pulp.lpSum([regular_shifts[i][
    j] for i in range(regular_workers)])  # 实际在第j个班次工作
    的正式工人数
105         # 确保实际分配人数与平均值之间的差距在可接受范围内
106         prob += (actual_shifts_in_slot >=
    average_shifts_per_day_per_slot - M * 0.1)  # 下限约束
```

```
107         prob += (actual_shifts_in_slot <=
    average_shifts_per_day_per_slot + M * 0.1)  # 上限约束
108
109     # 解决问题
110     prob.solve()
111
112     return [var.varValue for var in x_vars], [var.varValue for
    var in y_vars]
113
114
115 # 判断连续出勤天数
116 def continue_time(stat,day):
117     flag=0
118     if day>7:
119         for i in range(1,8):
120             if stat[day-i-1]==0:
121                 flag=1
122                 break
123     else:
124         flag=1
125
126     return flag
127
128
129 if __name__ == '__main__':
130     # 读取预测结果数据
131     df_hour = pd.read_csv("F:\\Python数模\\暑期培训\\2406\\结
    果\\结果表4.csv", encoding='GBK')
132     df_hour["货量"]=df_hour["货量"].round()
133     groupnamed = list(df_hour[df_hour['分拣中心'] == 'SC60'].
    groupby("日期").groups.keys())
134     during = ["00:00-08:00", "05:00-13:00", "08:00-16:00", "
    12:00-20:00", "14:00-22:00", "16:00-24:00"]
135
136     # 参数设置
```

```
    regular_workers = 200  # 正式工数量
    max_regular_shifts = int(0.85 * 30)  # 正式工最大班次数
    max_consecutive_days = 7  # 正式工连续出勤的最大天数

    # 初始化结果
    results = {"分拣中心": [], "日期": [], "班次": [], "出勤员工": []}
    employee_counter={"临时工":1}
    r_employee_counter={}   # 正式工出勤总天数
    regular_d=[[0]*30]*200  # 正式工连续出勤
    for i in range(1,201):
        r_employee_counter[f"正式工{i}"]=0

    if __name__ == '__main__':
        # 多线程处理
        with Pool() as pool:
            mday, t=0, 0  # 记录日期和出勤的正式工
            # 遍历每个日期
            for j in groupnamed:
                # 为当前分拣中心和日期创建 DataFrame
                df_hour_ij = df_hour.query("分拣中心 == 'SC60' and 日期 == @j")
                mday+=1

                # 使用多线程处理
                results_list = pool.starmap(model_day, [(df_hour_ij, regular_workers, max_regular_shifts, max_consecutive_days)])

                # 处理结果
                for regular_shifts, temp_shifts in results_list:
                    coun = []  # 记录已分配的工作人员
                    for k in range(6):
                        # 对于正式工
```

```
                        for l in range(int(regular_shifts[k])):
                            # 找到未被安排工作的正式工
                            if continue_time(regular_d[t], mday) and r_employee_counter[f"正式工{t+1}"] < max_regular_shifts and (f"正式工{t+1}" not in coun):
                                coun.append(f"正式工{t+1}")
                                results["分拣中心"].append('SC60')
                                results["日期"].append(j)
                                results["班次"].append(during[k])
                                results["出勤员工"].append(f"正式工{t+1}")
                                regular_d[t][mday-1]=1
                                # 更新正式工的总出勤天数
                                r_employee_counter[f"正式工{t+1}"] += 1
                                t=(t+1)%200

                        # 对于临时工
                        employee_counter['临时工'] = 1
                        for l in range(int(temp_shifts[k])):
                            results["分拣中心"].append('SC60')
                            results["日期"].append(j)
                            results["班次"].append(during[k])
                            results["出勤员工"].append(f"临时工{employee_counter['临时工']}")
                            employee_counter['临时工'] += 1

        # 保存结果
        pd.DataFrame(results).to_csv("F:\\Python数模\\暑期培训\\2406\\结果\\结果表6.csv", index=False, encoding='GBK')
```