

## 生产企业原材料订购与运输的评价与规划模型

### 摘要

本文针对现有的生产企业原材料订购与运输方案的制定问题，通过历史订货和供货数据，针对供货商建立反映了其重要性的多指标评价模型，对模型筛选得出的重要供货商，建立原材料订购与运输方案的多目标规划模型，分别尝试以保障生产稳定性、原材料订购结构优化和产能提升为优化目标，通过遗传算法对模型进行求解，得到基于三种优化目标最佳的订购方案与运输方案。

本题的数据较多，在建立模型前首先对数据进行预处理，发现附件数据不存在明显的错误和缺失。在比较附件 1 的历史订货数据与供货数据的偏差后，发现存在供货质量极低的 33 家供货商，将这类明显的低质量供货商剔除，不纳入后续的研究与分析。

研究供货商的历史数据，量化分析供货商的供货特征，首先通过对附件数据进行分析，建立了 6 个能有效反映供货商供货能力和供货风险的指标。这 6 个指标分别为供货次数、平均供货量、单次最大供货量、供货稳定性、供货一致性和合理供货比例。通过 TOPSIS 法，结合 6 个指标，建立多指标评价模型，对每个供货商的重要性程度进行评价。基于模型的评价结果，本文筛选出了 50 家最重要的供货商，其中得分最高三家的供货商分别为 S140, S229, S361，这三家供货商的得分分别为 0.9435, 0.8755 和 0.8401。

以保障生产的稳定性为目标，建立最经济的订购方案与损耗最小的运输方案。针对问题一模型所得的 50 家最重要供货商，以供货商数目最小和供货能力能满足最大产能为目标，建立基于多目标规划的筛选模型，通过遗传算法求解，发现至少需要 39 家固定的供货商供货，才能恰好满足最大产能需求。针对这 39 家供货商，以最小化成本为目标，生产稳定性为约束建立动态规划模型，通过遗传算法进行求解，得到了最经济的订购方案。针对每周的订购方案，以运输损耗为优化目标，建立运输方案的规划模型，通过程序模拟迭代对模型进行求解，得到了损耗最小的转运方案。通过检验库存的变化和估计损耗率，对订购方案和转运方案进行评估，发现本文所得结果是合理且有效的。

通过优化原材料订购结构，对成本进一步压缩。对问题二中的订购方案模型进行分析，发现由于各项成本的数量级差异，导致模型对原材料订购结构并不敏感，进而使得原材料订购结构无法达到最优。以问题二模型结果为起始点，添加对原材料订购结构的强约束，通过遗传算法对模型进行求解，得到新订购方案与转运方案。相比于问题二中的订购方案，新的订购方案优化了原材料的订购结构，使每周的订购成本平均缩减 0.956%，最大可缩减 4.847%。

考虑生产企业产能提升的潜力，结合供货商和转运商的实际情况，重新制定订购方案与运输方案。供货商和转运商的能力决定了企业产能提升的潜力，在问题二模型的基础上，基于生产背景对模型进行修改，以转运商的转运能力为企业产能提升的上限，放低对供货商供货质量的要求。通过遗传算法对模型进行求解，得到了新的订购方案和转运方案，发现在现有的供货链中，该生产企业的产能还可提高 1.58 万立方米左右。

**关键词：**TOPSIS，评价模型，遗传算法，多目标规划模型

## 一、问题重述

某生产企业每年按 48 周安排生产，需提前制定 24 周的原材料订货和转运计划。计划的制定需要根据产能要求确定每周的原材料供货商和相应每周的原材料订货数量，并确定第三方物流公司，委托其将原材料转运至企业仓库。该生产企业所用的原材料分为 A、B 和 C 三种类型，其采购单价不同，均为独立用于生产产品，生产一立方米产品所需消耗的量不同，三种原材料的运输和储存的单位费用则是相同的。该企业每周产能为 2.82 万立方米，且要求尽可能保持不少于满足两周生产需求的原材料库存量。供货商实际供货量可能多于或少于订货量，企业对供货商实际供货的原材料总是全部收购。

每家转运商的运输能力为 6000 立方米/周。在原材料的实际转运过程中会有一定的损耗，不同的转运商产生的损耗程度不同。一家供货商每周的供货的原材料应尽量由一家转运商运输。

附件 1 给出了该企业近 5 年 402 家原材料供货商每周的订货量和供货量数据；附件 2 给出了 8 家转运商近 5 年每周的运输损耗率数据。现完成如下问题：

1. 量化分析附件 1 中 402 家供货商的供货特征，建立反映保障企业生产重要性的数学模型，从而确定 50 家最重要的供货商。
2. 参照问题一所得的供货商重要性，制定未来 24 周每周最经济的原材料订购方案及损耗最少的转运方案，并对方案的实施效果进行分析。
3. 考虑每周所订货的三种原材料的组成结构，压缩生产成本，制定新的订购方案，对新方案的实施效果进行分析，并与原方案对比。
4. 企业具备提高产能的潜力，根据现有供货商和转运商的实际情况，分析其规律，确定该企业每周产能可提高多少，制定新的订购方案，对新方案的实施效果进行分析。

## 二、问题分析

生产企业原材料的订货与运输问题给出了 402 家供货商近 5 年的订货与供货的数据，题目要求据此制定未来 24 周的订货与转运方案。供货商供货的不稳定性，使得生产企业的供货链上游不可避免的产生风险。为了有效地管理风险，采购商需要与主要供货商建立更紧密的关系，这些供货商被期望提供解决方案，增强采购公司的核心竞争力。<sup>[1]</sup>因此，面对良莠不齐的 402 家供货商，考虑到优先保障企业生产以及库存的稳定性，有必要对供货商的供货能力进行筛选，挑选出那些供货能力强，供货稳定性高的供货商建立长期稳定的合作关系，以确保供货链的上游既高效又稳定。

### 2.1 量化分析供货商供货特征衡量供货商重要性

问题一要求基于供货商的历史订货与供货数据进行量化分析，探寻供货商的供货特征，并建立反映保障企业生产重要性的数学模型，筛选出 50 家最重要的供货商。首先，对附件 1 的数据进行预处理，确保预处理后的数据不存在错

误数据。接着针对预处理后的数据，进行探索性数据分析，观察 402 家供货商在供货能力、供货稳定性上是否存在明显的差异化特征。基于数据分析的结果，构建能够有效反映供货商供货能力与供货稳定性的指标，对指标进行分析和筛选。结合筛选过后的指标，建立多指标评价模型，对 402 家供货商进行重要性程度评价，基于评价结果，选择最重要的 50 家供货商。

## 2.2 考虑生产稳定性的订购方案与运输方案的制定

问题二要求基于问题一，筛选出至少多少家供货商供货能满足每周的最大产能需求，并针对筛选所得的供货商，制定未来 24 周最经济的订货与损耗最少的转运计划。供货商的选择是结合了供货能力和供货稳定性的综合决策，因此本文通过在建立筛选模型时，需要在满足最大产能原材料需求的前提下，考虑供货不确定性以及运输损耗带来的风险，给出所需供货商数目最少的组合。针对筛选得出的供货商组合，将订货一批原材料的成本作为衡量订货方案经济程度的指标，成本需要将运输成本、储存成本和购买成本都考虑在内。挖掘供货商订货与供货数据的周期性，基于此建立各个供货商供货能力的周期性模型。以成本最小化为目标，动态的库存以及供货商供货能力的周期性变化为约束，建立每周的多变量优化模型。逐周的求解优化模型，即可得到未来 24 周最经济的订购方案。转运方案的制定，需要考虑到原材料的不同，结合原材料价格以及运输成本，可知不同原材料生产一立方米产品的成本（简称“产品生产成本”）不同，因此转运商的选取应当以损耗率低的转运商运送产品生产成本较高的原材料为原则，据此建立规划模型，结合题干信息建立约束，求解规划模型即可得到每周损耗最小的转运方案。

## 2.3 考虑原材料订货结构优化的订购方案和转运方案制定

对于问题三，本文考虑基于问题二所构建的模型，通过添加对原材料的偏好，尝试对模型进行修改。检验原材料订购结构是否得到优化，通过对成本进行量化，对比问题二和问题三的两种方案的成本差异。

## 2.4 考虑产能提升的订购方案与转运方案制定

对于问题四，同样沿用问题二的思路和模型，对目标供货商与约束条件进行修改。由于企业已具备了提高产能的潜力，为满足企业产能提升的需求，考虑将供货商选取范围进一步扩大。同时由于产能提升潜力的存在，原产能上限已不再成为限制，库存积压导致的成本问题能有效被解决，同时限制企业每周产能的因素转变为转运商每周的总转运量。据此问题二的订购方案中的目标函数，重新求解符合问题四的订购方案。

# 三、基本假设与符号说明

## 3.1 基本假设

- (1) 该生产企业再过去五年内的生产规模相对稳定，未出现较大变化；
- (2) 该生产企业再过去 5 年的订货数据都是基于对供货商供货能力的充分考虑；
- (3) 该生产企业的仓库有足以满足两周以最大产能生产消耗的原材料库存。

## 3.2 符号说明

符号	含义	单位
$n_i$	供货商 <i>i</i> 的供货次数	\
$m_i$	供货商 <i>i</i> 的平均供货量	$m^3$
$x_i^{max}$	供货商 <i>i</i> 的单次最大供货量	$m^3$
$\delta_i$	供货商 <i>i</i> 的供货稳定性	\
$\Delta_i$	供货商 <i>i</i> 的供货连续性	\
$\gamma_i$	供货商 <i>i</i> 的合理供货比例	\
$s_i$	供货商 <i>i</i> 的重要性评分	\
$z_{i,t}$	第 <i>t</i> 周企业对供货商 <i>i</i> 订单量	$m^3$
$x_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 实际供货量	$m^3$
$\epsilon_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 实际供货偏差量	$m^3$
$\hat{x}_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 预测供货量	$m^3$
$V_{A,t}$	第 <i>t</i> 周原材料 <i>A</i> 的库存	$m^3$
$E_{A,t}$	第 <i>t</i> 周原材料 <i>A</i> 的消耗量	$m^3$
$C_{trans}$	单位原材料运输费用	$元/m^3$
$C_{purchase,A}$	单位原材料 <i>A</i> 采购费用	$元/m^3$
$C_{store}$	单位原材料储存费用	$元/m^3$
$O_t$	在第 <i>t</i> 周的目标产能	$m^3$

#### 四、数据预处理

本题附件 1 提供了 402 家供货商近 5 年内 240 周的订单数据与供货数据。由于数据量较大，需要对所给的数据进行一定的预处理，以防止其中存在错误的数据，使得计算时对结果造成影响导致出现误差。

借助 Excel 软件对附件 1 中数据进行整理，得到企业对于各供货商的订单总数量，各供货商对企业的供货总单数，供货总量以及各供货商接收到的订货量与实际供货量的差值。如下表所示：

表 1：对附件 1 数据的统计性分析表

供货商 ID	供货次数	供货总量	平均供货量	平均差值
S001	25	49	2.0	2.0
S002	71	273	3.9	0.4
S003	191	13138	68.8	5.7
S004	33	64	1.9	6.3
S005	107	6912	64.6	-3.3
S006	13	30	2.3	7.9
...	...	...	...	...

由上表可知，94%的供货商接收到的订货量与实际供货量的差值在 100 立方米以内，对于个别存在差值较大的供货商需进行进一步分析，检验与处理。分别统计各供货商供货量与订单量差值大于 100 立方米及 1000 立方米的次数。所

得结果如表二所示：

表 2：供货商的缺货情况统计

供货商 ID	缺货大于 100	缺货大于 1000
S161	17	17
S160	15	15
S201	14	13
S246	13	13
S074	20	12
...	...	...

基于假设 2，该生产企业再过去 5 年的订货数据都是基于对供货商供货能力的充分考虑，可推知该企业过去 5 年内的所有订单量均无误。结合如上表格，认定供货总单数较少且供货量与订单量差值大于 100 次数较多的供货商为极低质量供货商。因此为防止其作为极值影响后续计算结果，将 S034, S047, S118, S137 等 33 家极低质量供货商从数据集中剔除，附录 2 中给出了剔除的 33 家供货商的详细数据。

至此完成数据的整理与剔除，后文的建模和求解均建立在此数据预处理的基础上进行。

## 五、问题一的模型建立与求解

本问要求基于供货商的历史订货与供货数据对供货商的供货特征进行量化分析，建立模型反映保障企业生产重要性的模型，并在此基础上给出 50 家最重要的供货商。企业在选择供货商时，需要综合考虑供货商的运营情况、成本控制、生产运作、技术开发等信息。<sup>[2]</sup>在本问题中，本文将从所给的历史数据提取一定的指标来衡量供货商的供货特征，指标的选取基于附件 1402 家供货商近 5 年的订货与供货数据，基于所得指标，建立多指标评价模型，给出 402 家供货商的重要性程度。

### 5.1 建模前准备

根据参考文献可知，评价指标应当刻画的是被评价对象的特征或属性。每一项所选取的评判指标均为从不同方面反映评价对象所具有某种特征大小的一个度量。而在多个评价指标构成指标体系时，构建评价指标体系的原则在于：系统性、科学性、可比性、可测性和独立性。

依据此原则，分析供货商的订货与供货数据，构建能够反映供货商供货特征的指标。本文从供货商供货能力与供货稳定性两个角度出发，构建供货次数，平均供货量和单次最大供货量用于反映供货商供货能力；供货稳定性，供货一致性和合理供货比例用于反映供货商供货风险。以这 6 个指标，建立多指标评价模型，以模型的输出结果作为供货商重要性的体现。

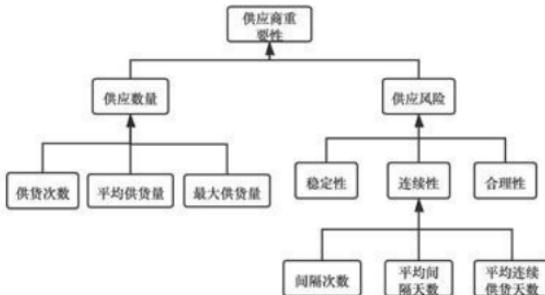


图 1：问题一多指标评价模型

**a. 供货次数**

供货次数间接反映供货商接到订货的次数，生产企业在过去的5年向该供货商的订货次数，能够直接反映生产企业对该供货商的依赖程度。由假设2可知，生产企业对该供货商的依赖程度能够有效反映生产企业对该供货商供货能力的认可度，即供货次数越多的供货商，具有更强的供货能力。

**b. 平均供货量**

基于假设2，平均供货量在一定程度上反映该供货商的供货能力。同时结合供货总量也可以得到各供货商对于企业的供货次数。可以间接反映出该供货商与企业的业务联系，进而反映出该生产企业与该供货商是否存在长久合作关系。若供货商的供货次数越多，则说明供货商与企业和合作关系越紧密，越可能成为高质量供货商。

**c. 单次最大供货量**

由于该企业对供货商实际提供的原材料总是全部收购，供货商为实现利益最大化，一定会保持最大生产效率。因此单次最大供货量直接反映了供货商供货能力的上限，同样是衡量供货商供货能力的一个重要指标。

**d. 供货稳定性**

对于企业而言，供货量过多会导致成本增加，供货量过少影响产能释放，因此供货量需要与订单量保持较高的一致性。即供货商的供货量需要保持在企业给予其订单量的一定范围内波动。波动值越小则说明供货商的供货能力越稳定，越受企业的重视。

**e. 供货量连续性**

供货同种原材料的供货商，能够持续提供原材料的供货商要比间歇性提供原材料的供货商实力更强，同时企业为了节约决策成本，也更愿意选择能够持续供货的供货商。因此供货商的供货连续性越高，则越有可能成为企业的高质量供货商。

**f. 合理供货比例**

理想供货商的供货量应当与企业的订货量相当，能够充分满足企业需求量的供货商可以为企业减少许多不必要的成本。即供货商的合理供货比例越高，企业越有可能与供货商达成长期合作。

## 5.2 模型的建立

### 5.2.1 指标计算

#### a. 供货次数

供货总量的计算，基于附件 1 的供货商供货数据。通过计算各个供货商在过去 5 年参与供货的次数得到：

$$n_i = \sum_{t=1}^{240} y_{i,t}$$

$$y_{i,t} \in \{0, 1\}, i = 1, 2, \dots, 402$$

其中， $n_i$  即为供货商  $i$  在近五年供货次数， $y_{i,t}$  为逻辑变量，其含义为在第  $t$  周供货商  $i$  是否参与了供货，“1”表示参与供货，“0”表示未参与供货。

#### b. 平均供货量

平均供货量的计算首先需要针对原材料进行换算，将其统一为可生产产品的数目后，剔除在 240 周内未参与供货的数据，求取平均值：

$$m_i = \frac{1}{n_i} \sum_{t=1}^n \frac{x_{i,t}}{p_i}, i = 1, 2, \dots, 402$$

$$p_i = 0.6, i \in A$$

$$p_i = 0.66, i \in B$$

$$p_i = 0.72, i \in C$$

其中， $m_i$  即为供货商  $i$  的平均供货量， $p_i$  为生产一立方米产品所需的原材料消耗， $x_{i,t}$  为第  $t$  周供货商  $i$  的供货量，集合  $A, B, C$  分别表示各类原材料的供货商集合。

#### c. 单次最大供货量

通过附件 1 所给的供货商历史供货数据，首先对原材料进行换算，将其统一为可生产产品数目，再寻找每家供货商的历史供货数据的最大值：

$$x_i^{\max} = \max\{x_{i,t}, t = 1, 2, \dots, 240\}, i = 1, 2, \dots, 402$$

其中， $x_i^{\max}$  即为供货商  $i$  的单次最大供货量。

#### d. 供货稳定性

通过比较各供货商每次供货量与订单量的差值，可以直观的得到各供货商的供货稳定性，其计算公式如下：

$$\delta_i = \frac{1}{n_i} \sum_{t=1}^n (x_{i,t} - z_{i,t})^2, i = 1, 2, \dots, 402$$

其中， $\delta_i$  即表示供货商  $i$  的供货量与订单量的均方误差， $z_{i,t}$  为第  $t$  周供货商  $i$  的订货量。

#### e. 供货量连续性

由于供货连续性无法由附件 1 的数据直接得出，故引入时间间隔这一概念，即企业对于各供货商两次订单中间间隔的周数，以及供货商连续为企业提供原材料的周数。构造间隔次数，平均间隔周数，平均连续供货周数三个指标来衡量企业供货连续性。通过熵权法计算三个指标的系数，以三者的综合得分作为各供货商的供货连续性指标。

对于间隔天数，分析后发现其为成本型指标；对于平均间隔天数和平均连续供货天数，分析后发现其为效益型指标。对不同类型的指标进行不同的归一化处理，处理公式如下：

① 成本型指标归一化

$$x_{Scale} = \frac{x_{max} - x}{x_{max} - x_{min}}$$

② 效益型指标归一化

$$x_{Scale} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

结合针对归一化处理后的指标，供货连续性评分模型建立如下：

$$\Delta_i = w_1 r_{i,1} + w_2 r_{i,2} + w_3 r_{i,3} \quad (1)$$

其中  $w_j$  即为第  $j$  个指标的权重， $r_{i,j}$  即为供货商  $i$  第  $j$  个指标的值。通过熵权法求解权重，代入(1)式，即可得供货商  $i$  供货连续性  $\Delta_i$ 。

### f. 合理供货比例

理想供货商的供货量应该在企业所给订单量的一定范围内上下浮动。此处设置浮动范围为 20%。即  $x_{i,t} \in (80\%z_{i,t}, 120\%z_{i,t})$ 。若供货事件符合条件，则记为 1，否则记为 0，则有：

$$Count_i = \begin{cases} 1, & x_{i,t} \in (0.8z_{i,t}, 1.2z_{i,t}) \\ 0, & x_{i,t} \notin (0.8z_{i,t}, 1.2z_{i,t}) \end{cases}$$

下一步计算符合条件的供货次数占供货商总供货次数的比值，即可得到供货商  $i$  的合理供货比例，即：

$$\gamma_i = \frac{Count_i}{n_i}$$

### 5.2.2 模型汇总

确定了能有效描述供货商的供货特征的指标后，构建供货重要性评价模型，通过熵权法方法求解出各个指标的权重，建立基于 TOPSIS 的多指标评价模型，该得分即反映出供货商重要性，值越大，则重要性程度越高。

熵权法(Entropy Weight Method, EWM)的基本思路是根据指标变异性的大小来确定客观权重。一般来说，若某个指标的信息熵越小，表明指标值得变异程度越大，提供的信息量越多，在综合评价中所能起到的作用也越大，其权重也

就越大。相反，某个指标的信息熵越大，表明指标值得变异程度越小，提供的信息量也越少，在综合评价中所起到的作用也越小，其权重也就越小。

TOPSIS(Technique for Order Preference by Similarity to Ideal Solution)法是一种常用的组内综合评价方法，能充分利用原始数据的信息，其结果能精确地反映各评价方案之间的差距。基本过程为基于归一化后的原始数据矩阵，采用余弦法找出有限方案中的最优方案和最劣方案，然后分别计算各评价对象与最优方案和最劣方案间的距离，获得各评价对象与最优方案的相对接近程度，以此作为评价优劣的依据。

### 5.3 基于 TOPSIS 的多指标评价模型的求解

#### 5.3.1 算法求解

(1) 熵权法求解流程如下：

##### 熵权法求解流程

**Step 1** 将数据导入 Python，对指标进行标准化处理

**Step 2** 计算标准化后指标的熵值

**Step 3** 计算各指标的差异系数

**Step 4** 计算各指标的权重

**Step 5** 分别用权重乘以归一化后的数据

(2) TOPSIS 算法求解流程如下：

##### TOPSIS 算法求解流程

**Step 1** 将数据导入 Python，根据熵权法得到供货次数，平均供货量，单词供货最大值，供货稳定性，供货连续性和合理供货比例共六个指标的权重

**Step 2** 对六个指标数据进行指标属性同向化处理并构造归一化初始矩阵

**Step 3** 确定最优方案和最劣方案

**Step 4** 计算各评价对象与最优方案、最劣方案的接近程度

**Step 5** 计算各评价对象与最优方案的贴近程度

**Step 6** 根据贴近程度大小进行排序，给出评价结果

**Output:** 各供货商 TOPSIS 评价结果

#### 5.3.2 求解结果展示

表 3：供货连续性评价体系指标权重

连续性评价指标	间隔次数	平均间隔周数	平均连续供货周数
权重	0.4383	0.3560	0.2057

如上表即为供货连续性评价体现的指标权重，即：

$$\Delta_i = 0.4383r_{i,1} + 0.3560r_{i,2} + 0.2057r_{i,3}$$

从权重的值不难看出，供货连续性主要受到间隔次数和平均间隔周数的影响，其中间隔次数是反映供货连续性的最终要的一个指标。

表 4：TOPSIS 多指标评价模型指标权重

指标	供货总量	平均供货量	单词供货最大值	供货稳定性	供货连续性	合理供货比例
权重	0.2617	0.2272	0.1622	0.3143	0.0254	0.0093

上表即为 TOPSIS 多属性评价体现的指标权重，从权重的值来看供货的稳定性是影响供货商重要性最重要的指标，其次为供货的总量，这两点分别对应了供货能力和供货风险。

#### 5.4五十家重要性供货商结果展示

表 5: 50 家最重要的供货商

供货商 ID	重要性程度	供货商 ID	重要性程度	供货商 ID	重要性程度
S140	0.9435	S268	0.4676	S244	0.2627
S229	0.8755	S306	0.4565	S080	0.2611
S361	0.8401	S307	0.4498	S218	0.2528
S348	0.8150	S194	0.4425	S003	0.2481
S151	0.8070	S352	0.4257	S189	0.2469
S108	0.7432	S143	0.3990	S086	0.2460
S395	0.5700	S247	0.3860	S210	0.2441
S374	0.5631	S037	0.3654	S114	0.2320
S139	0.5623	S284	0.3643	S074	0.2238
S340	0.5459	S365	0.3612	S007	0.2194
S282	0.5450	S031	0.3495	S123	0.2189
S330	0.5409	S040	0.3069	S273	0.2169
S308	0.5269	S364	0.3003	S291	0.2167
S275	0.5230	S346	0.2981	S292	0.2023
S329	0.5145	S367	0.2908		
S126	0.5111	S294	0.2782		
S131	0.4878	S055	0.2763		
S356	0.4840	S338	0.2755		

#### 六、问题二的模型建立与求解

在问题一的求解中，本文通过对供货商的供货特征的分析，综合考虑供货商供货能力和供货风险，建立了反映保障企业生产重要性的数学模型，对供货商的重要性进行了评价，并筛选出了 50 家最重要的供货商。问题二在问题一结论的基础上，通过建立供货商筛选的多目标规划模型，以确定该生产企业在考虑满足原材料需求的情况下，至少需要选择多少家供货商供货原材料才得以满足生产的需求。针对模型筛选所得的供货商组合，以保障生产的稳定性为前提，考虑供货不稳定性和运输损耗对库存的影响，建立动态规划模型，基于遗传算法，逐周求解最优订购方案与转运方案，制定未来 24 周最经济的订购方案和转运方案。

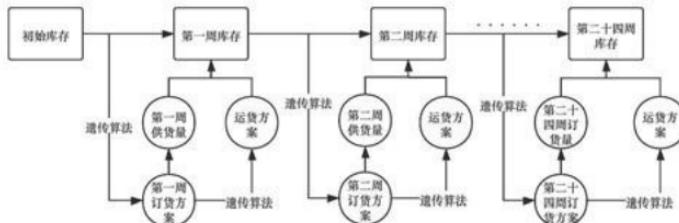


图 2: 问题二的多目标规划模型

### 6.1 供货商筛选模型的建立

在问题一的模型输出结果的基础上，本问需要进一步缩小目标订货供货商的集合，寻找供货商数目最小的供货商集合。以供货商数目最小为目标的前提下，以每周供货量可满足最大产能需求为约束，建立一个多目标规划模型。如果仅仅要求满足最大产能需求，那就意味着模型只考虑了供货商的供货能力，而无法将供货风险考虑再模型中。因此本文将问题一模型输出的供货商重要性也纳入模型中，将供货商重要性之和最大作为第二个优化目标，尝试构建一个多目标规划模型。

#### 6.1.1 目标函数的确定

本模型为一个多目标的优化模型，存在两个优化目标，第一个优化目标为每周订货的供货商数目最少，其数学表达式为：

$$\min_y \sum_{i=1}^n y_i$$

其中， $y_i$ 为一个逻辑变量，取值只有“0”或“1”，其含义为供货商*i*是否参与本周的供货，0表示不参与供货，1表示参与供货。如上公式即为参与每周供货的供货商数目。

第二个优化目标为参与每周供货的供货商的重要性程度之和最大，公式表达为：

$$\max \sum_{i=1}^n y_i \cdot s_i$$

其中， $s_i$ 为问题一模型所得的重要性程度指标，其取值范围为[0,1]。该优化目标的设置是考虑了供货商对生产企业的重要性程度，将重要性程度之和设置为最大化目标，可确保在优化问题的求解中，使得重要性程度高的企业优先被考虑参与供货。

对两个优化目标进行进一步化简，将多目标优化转化为单目标优化，所得新式如下：

$$\min_y \frac{\sum_{i=1}^n y_i}{\sum_{i=1}^n y_i \cdot s_i}$$

### 6.1.2 约束条件的确定

首先，在假设 1 的前提条件下，有如下推论，该生产企业 5 年内规模维持稳定，从企业的供求关系看，该生产企业的产能设置必然会经过仔细的考虑，以避免应产能过剩，而导致增加固定成本，因此可以推论在该假设前提下，该生产企业在日常运营中其产量与最大产能持平。在此推论的基础上，可知该生产企业每周的进货量必须能够满足每周的生产消耗，即最大产能 2.82 万立方米。对每周的供货量，有如下约束：

$$\begin{aligned} \sum_{i=1}^n \frac{y_i \hat{x}_{i,t} (1 - \alpha_{i,t})}{p_i} &\geq 2.82 \times 10^4, \quad t = 1, 2, \dots, 24 \\ \hat{x}_{i,t} &= z_{i,t} + \hat{\epsilon}_{i,t} \\ p_i &= 0.6, \quad i \in A \\ p_i &= 0.66, \quad i \in B \\ p_i &= 0.72, \quad i \in C \\ y_i &\in \{0, 1\} \end{aligned}$$

其中， $p_i$  为生产一立方米产品所需消耗的原材料， $\hat{x}_{i,t}$  为第  $t$  周供货商  $i$  的供货量的估计值， $\alpha_{i,t}$  为第  $t$  周供货商  $i$  所供货的原材料转运损失率， $z_{i,t}$  为第  $t$  周供货商  $i$  所接到的订单数， $\hat{\epsilon}_{i,t}$  为第  $t$  周供货商  $i$  的供货偏差。

### 6.1.3 优化模型汇总

式 (2) 即为供货商筛选优化模型，最终汇总如下所示：

$$\begin{aligned} \min_y \quad & \frac{\sum_{i=1}^n y_i}{\sum_{i=1}^n y_i \cdot s_i} \\ \text{s.t.} \quad & \sum_{i=1}^n \frac{y_i \cdot \hat{x}_{i,t} (1 - \alpha_{i,t})}{p_i} \geq 2.82 \times 10^4, t = 1, 2, \dots, 24 \\ & \hat{x}_{i,j} = z_{i,j} + \hat{\epsilon}_{i,j} \\ & p_i = 0.6, \quad i \in A \\ & p_i = 0.66, \quad i \in B \\ & p_i = 0.72, \quad i \in C \\ & y_i \in \{0, 1\} \end{aligned} \tag{2}$$

### 6.1.4 模型求解

在对供货商筛选模型进行求解时，可进一步化简，使得模型求解过程更加方便，而不影响结果的准确性。在模型的约束中，本文考虑了原材料在运输过程中所产生的损失，以及供货商的供货偏差，这两个变量属于随机变量。

其中损失率  $\alpha_{i,t}$ ，以附件 2 的历史数据作为参考，损失率的取值范围大概在  $[0, 0.05]$  内，其取值较小，因此本文选取附件 2 的历史数据求取损失率的平均值作为损失率的估计值：

$$\alpha = \frac{1}{240} \sum_{t=1}^{240} \frac{1}{n_t} \sum_{j=1}^m \alpha_{t,j}$$

$\alpha_{t,j}$  为过去五年中第  $t$  周转运商  $j$  的损失率,  $n_t$  为第  $t$  周参与转运的转运商数目, 经计算得损失率的估计值为 1.34%

对于供货量  $x_{i,t}$ , 在约束中的含义用于体现供货商的供货能力, 而不指代供货商  $i$  每周的供货量。因此本文在求解时是使用附件 1 所提供的供货商历史供货数据的平均值作为对供货能力的估计值:

$$x_i = \frac{1}{m_i} \sum_{t=1}^{m_i} x_{i,t}$$

其中,  $m_i$  为供货商  $i$  在过去 5 年中供货的次数,  $x_{i,t}$  为过去第  $t$  周供货商  $i$  供货的数量。

在进行如上的简化后, 通过 Python 编写遗传算法对模型进行求解。遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型, 是一种通过模拟自然进化过程搜索最优解的方法。

算法求解本模型的迭代过程如下图所示:

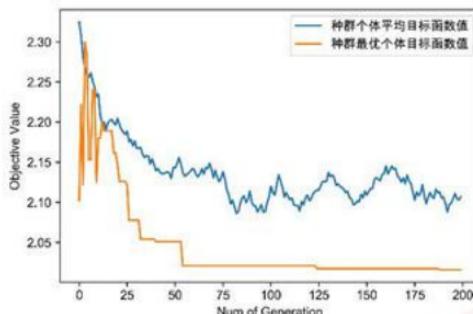


图 3: 问题二订货方案模型的遗传算法求解过程

求解后所得结果如下表所示, 至少需要向表 6 所示的 39 个供货商订货才可满足每周的生产需求。

表 6: 筛选模型所得的 39 家重要供货商

供货商 ID	材料分类						
S003	C	S131	B	S273	A	S340	B
S031	B	S139	B	S275	A	S346	B
S037	C	S140	B	S282	A	S348	A
S055	B	S143	A	S284	C	S356	C
S074	C	S151	C	S292	A	S361	C
S080	C	S189	A	S294	C	S364	B

S086	C	S194	C	S306	C	S367	B
S108	B	S210	C	S329	A	S374	C
S123	A	S229	A	S330	B	S395	A
S126	C	S268	C	S338	B		

## 6.2 订购方案模型

6.1 中通过建立供货商筛选模型，对问题一所选择的 50 家重要供货商进行筛选，进一步缩小了原材料供货商的选择范围，再此基础上建立订购方案模型。在本问题中，订购方案的制定，以保持企业生产的稳定以及库存的稳定为原则，以订货成本最小化为目标。基于假设 1，本文有如下推论：对于一个规模稳定的生产企业，其产能是以市场的需求为参考设计的，即生产企业再日常运营中，很少会出现因为市场需求不足，而导致产能过剩的情况。基于此推论，本文在制定每周订货方案时，将以每周产能即为最大产能考虑。

再考虑供货商的供货能力时，问题二的供货商筛选模型中，本文以供货商的平均供货数量作为供货商供货能力的体现。在制定订货方案时，本文认为不应该直接以该值作为每周供货能力的估计值。主要原因有两点：首先，模型的目标是制定未来 24 周订货方案，如果直接以平均供货数量作为每周的供货量估计值，那么模型的输出结果中每周的订货方案必然是较为一致的，而从供货商的历史订货和供货数据来看，最直观的发现就是，供货商的订单数目并不是每周一致的。其次，考虑到原材料为木制纤维和其他植物素纤维，作为经济作物，其生产是存在一定的周期性的<sup>[1]</sup>，即极有可能供货商的供货能力是存在一定的周期性的。因此，本文假设供货商的供货存在一定的周期性，再模型建立和求解时，考虑对供货商筛选模型所输出的 39 家供货商的历史数据以 24 周为一个周期进行周期性规律探索和建模。

### 6.2.1 目标函数的确定：订货成本的计算

订货成本可分为原材料购买成本、原材料运输成本和原材料储存成本，成计算的公式如下：

$$\begin{aligned} Cost_t = & C_{Trans} \sum_{i \in S} x_{i,t} + \\ & C_{Purchase,A} \sum_{i \in A \cap S} x_{i,t} + C_{Purchase,B} \sum_{i \in B \cap S} x_{i,t} + C_{Purchase,C} \sum_{i \in C \cap S} x_{i,t} + (3) \\ & C_{Store}(V_{A,t} + V_{B,t} + V_{C,t}) \end{aligned}$$

其中  $C_{Trans}$  为原材料每立方米运输成本， $x_{i,t}$  为第  $t$  周供货商  $i$  的供货量， $C_{Purchase,A}$  为每立方米原材料 A 的采购价格， $C_{Store}$  为每立方米原材料的储存成本， $V_{A,t}$  为第  $t$  原材料 A 的库存量， $S$  为供货商集合。其中，供货量与订货量有关，其计算公式为：

$$x_{i,t} = z_{i,t} + \epsilon_{i,t}$$

$z_{i,t}$  为第  $t$  周供货商  $i$  的供货量， $\epsilon_{i,t}$  第  $t$  周供货商  $i$  的供货误差。

A、B、C 三种原材料均可用于生产产品，不同的原材料生产一立方米产品

的消耗不同，不同原材料的采购单价也不同。三种原材料的价格有如下关系：

$$C_{Purchase,A} = 1.2 C_{Purchase,C}; \quad C_{Purchase,B} = 1.1 C_{Purchase,C}$$

为进一步简化表达式，不妨设原材料 C 的每立方米采购价格为  $C_{Purchase}$ ，结合上述价格关系，式(3)可简化为：

$$\begin{aligned} Cost_t &= C_{Trans} \sum_{i \in S} x_{i,t} + \\ &C_{Purchase} \left( \sum_{i \in A \cap S} 1.2x_{i,t} + \sum_{i \in B \cap S} 1.1x_{i,t} + \sum_{i \in C \cap S} x_{i,t} \right) + \\ &C_{Store}(V_{A,t} + V_{B,t} + V_{C,t}) \end{aligned}$$

如上公式所计算订货成本  $Cost_t$ ，即为第 t 周订购方案模型的目标函数。

### 6.2.2 模型约束的确定

#### a. 库存更新

在企业的生产与订货中，为了满足企业的正常的生产需求，要求企业尽可能保持仓库中有两周生产所需的原材料库存。由于原材料的特殊性，导致供货商供货的不稳定性，使得生产企业的库存每周都有可能会因为供货的差异而导致发生改变。因此，在计划每周的订货数量时，必须考虑库存情况，若库存原材料少于两周生产所需时，则需要增加下周的订货量以填充库存，若库存原材料多于两周生产所需时，则需要减少下周的订货量以缩减库存。综上所述，仓库库存必须作为模型的约束，以确保订货方案的合理性，防止出现库存积压或不足。

在 24 周内，第 t 周仓库中不同原材料的库存可表示为如下公式：

$$\begin{aligned} V_{A,t-1} - E_{A,t} + \sum_{i \in A \cap S} x_{i,t} (1 - \alpha_{i,t}) &= V_{A,t} \\ V_{B,t-1} - E_{B,t} + \sum_{i \in B \cap S} x_{i,t} (1 - \alpha_{i,t}) &= V_{B,t} \\ V_{C,t-1} - E_{C,t} + \sum_{i \in C \cap S} x_{i,t} (1 - \alpha_{i,t}) &= V_{C,t} \end{aligned}$$

其中， $E_{A,t}$  表示第 t 周原材料 A 的消耗量， $V_{A,t}$  为第 t 周原材料 A 的库存量， $x_{i,t}$  为第 t 周供货商 i 的供货量， $\alpha_{i,t}$  为第 t 周供货商 i 所供货物在运输产生的损耗率，S 为目标供货商集合。

#### b. 库存维持

如上分析，生产企业必须保持企业仓库每周的库存量维持在两周所需的原材料的需求。就生产稳定性而言，对于生产企业来说，仓库库存的作用是防止出现原材料不足，导致出现生产无法正常进行，故维持两周生产所需的原材料是仓库原材料库存量的下限，因此要求第 t 周的累计库存符合以下条件：

$$\frac{V_{A,t-1}}{0.6} + \frac{V_{B,t-1}}{0.66} + \frac{V_{C,t-1}}{0.72} \geq 5.64 \times 10^4$$

其中,  $V_{A,t}$  为第  $t$  周原材料 A 的库存量, 公式右侧为仓库所存储原材料可生产产品需要大于该生产企业两周的最大产能, 即 5.64 万立方米。

#### c. 每周的产品生产

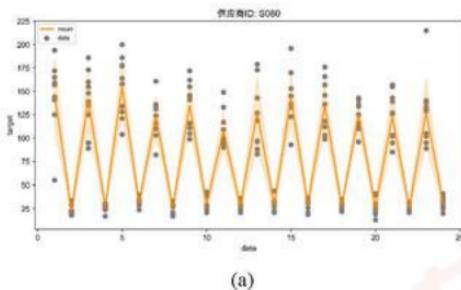
该生产企业的规模有限, 每周的最大产能为 2.82 万立方米。在建立该模型时, 考虑到附件中并未给出市场需求以及市场的供求关系。在本文中, 基于假设 1, 本文认为该企业的供求关系相对稳定。因此, 不存在生产产能过剩的情况。故在订货模型中, 每周的产品生产有如下约束:

$$\frac{E_{A,t}}{0.6} + \frac{E_{A,t}}{0.66} + \frac{E_{A,t}}{0.72} = 2.82 \times 10^4$$

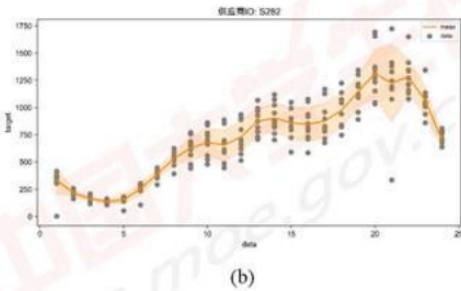
其中,  $E_{A,t}$  表示第  $t$  周原材料 A 的消耗量, 公式右侧为企业的每周最大产能, 即 2.82 万立方米。

#### d. 供货商 $i$ 的供货能力

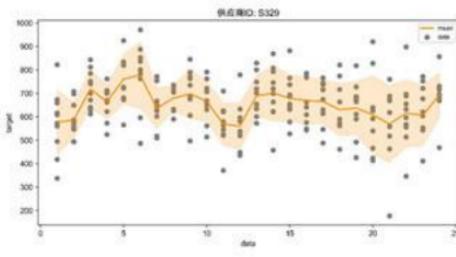
如 6.2 所述, 针对 39 家供货商的历史供货数据, 本文以 24 周为一个周期长度, 将 240 周划分为 10 个周期, 对历史供货数据进行了绘图, 发现部分供货商供货存在较明显的规律, 如下图所示:



(a)



(b)



(c)

图 4：以 24 周为周期的历史供货情况 (S080 为(a)、S282 为(b)、S329 为(c))

上图所示结果为三个供货商再过去 240 周，共 10 个周期的供货量数据，从图 4 中可发现供货商的供货情况是存在供货规律的。因此以 24 周作为一个供货周期，将每一个供货商过去 5 年，10 个周期内的供货数据作为对供货商供货能力范围的估计，确定供货商在第  $t$  周的供货能力范围，即：

$$z_{i,t}^{\min} \leq z_{i,t} \leq z_{i,t}^{\max}$$

其中， $z_{i,t}^{\min}$  表示历史 10 个周期中第  $t$  周供货商  $i$  平均最低供货量， $z_{i,t}^{\max}$  表示历史 10 个周期中第  $t$  周供货商  $i$  平均最高供货量的 1.5 倍。

#### e. 供货偏差

由于原材料的特殊性，供货商无法保证供货的稳定性，历史数据显示每个供货商在每次供货时都存在一定程度的供货偏差。因此，在建立每周的订购方案模型时需要考虑供货偏差的影响，可表示为如下公式：

$$\epsilon_{i,t} = G_i(z_{i,t})$$

其中， $G_i(t)$  表示供货商  $i$  的偏差函数，偏差函数通过高斯过程回归对供货商 10 个周期的订货与供货数据拟合得到，能够有效反映供货商的供货特点：

$$G_i(z_{i,t}) \sim N(\mu_i(z_{i,t}), \sigma_i^2(z_{i,t})) \quad (4)$$

图 5 为对两家供货商进行高斯过程回归拟合的结果。

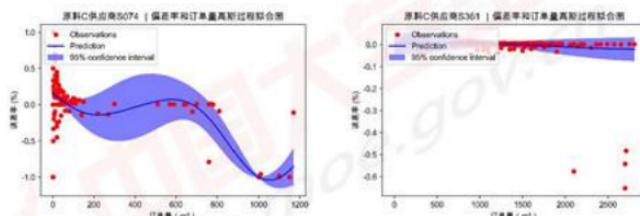


图 5:S074、S361 的供货数据拟合情况

#### 6.2.3 优化模型汇总

综上所述，每周订购方案的多目标优化模型汇总如下所示：

$$\begin{aligned}
 & \min_x C_{Transport} \sum_{i \in S} x_{i,t} + \\
 & C_{Purchase} \left( \sum_{i \in A \cap S} 1.2x_{i,t} + \sum_{i \in B \cap S} 1.1x_{i,t} + \sum_{i \in C \cap S} x_{i,t} \right) + \\
 & C_{Store} (V_{A,t} + V_{B,t} + V_{C,t}) \\
 \text{s.t.} \quad & x_{i,t} = z_{i,t} + \epsilon_{i,t} \\
 & \frac{V_{A,t-1}}{0.6} + \frac{V_{B,t-1}}{0.66} + \frac{V_{C,t-1}}{0.72} \geq 5.64 \times 10^4 \\
 & V_{A,t-1} - E_{A,t} + \sum_{i \in A \cap S} x_{i,t} (1 - \alpha_{i,t}) = V_{A,t} \\
 & V_{B,t-1} - E_{B,t} + \sum_{i \in B \cap S} x_{i,t} (1 - \alpha_{i,t}) = V_{B,t} \\
 & V_{C,t-1} - E_{C,t} + \sum_{i \in C \cap S} x_{i,t} (1 - \alpha_{i,t}) = V_{C,t} \\
 & \frac{E_{A,t}}{0.6} + \frac{E_{B,t}}{0.66} + \frac{E_{C,t}}{0.72} = 2.82 \times 10^4 \\
 & z_{i,t}^{\min} \leq z_{i,t} \leq z_{i,t}^{\max} \quad i \in S \\
 & \epsilon_{i,t} = G_i(t) z_{i,t} \quad i \in S \\
 & x_{i,t} \geq 0 \quad i \in S
 \end{aligned} \tag{5}$$

### 6.3 转运分配模型

在为各个供货商选择转运商时，考虑损耗率带来的影响，将由转运损耗带来的原材料损失换算为可生产产品的价值，以最小化该价值作为优化目标。对于每一个转运商，其损耗率的历史数据反映了该转运商的运输能力，本文尝试对转运商  $j$  的历史损耗率数据进行统计分布的拟合，以此方式对转运商的运输能力进行建模。通过 Python 的 `fitter` 库实现遍历 80 种分布对数据进行拟合，以拟合优度作为指标，选用单样本 K-S 检验的结果作为拟合优度的度量。对分布进行筛选，选择拟合优度最高的分布作为拟合结果输出，该库能有效的保证对转运商运输能力建模的准确性。转运商  $j$  的概率密度函数  $\varphi_j$  如下所示：

$$\varphi_j = P(\theta_j | \alpha_{j,1}, \alpha_{j,2}, \dots, \alpha_{j,t}) \tag{6}$$

其中， $\theta_j$  为分布函数的参数。

在制定第  $t$  周的转运方案时，基于式 (6) 拟合所得的分布函数，生成满足该分布的随机数，作为第  $t$  周转运商  $j$  的损耗率估计值。在考虑分配转运商时，本文的模型将基于如下几条原则：

- ① 优先选择损失率估计值低的转运商；
- ② 尽可能使得一家供货商的原材料全部由一家转运商运输
- ③ 在②的基础上，转运商  $j$ ，预留出部分转运能力，以应对供货商的供货不稳

定性：

- ④ 优先分配原材料 A 的供货商；
- ⑤ 在④的基础上，优先分配供货量大的供货商。

据此，以损耗原材料可生产产品数量作为优化目标，第  $t$  周有如下转运分配方案的优化模型：

$$\begin{aligned} \min_w & \sum_{i \in A \cap S} \left( \sum_{j=1}^8 \frac{x_{i,t}}{0.6} \varphi_j(t) w_{ij} \right) + \sum_{i \in B \cap S} \left( \sum_{j=1}^8 \frac{x_{i,t}}{0.66} \varphi_j(t) w_{ij} \right) + \\ & \sum_{i \in C \cap S} \left( \sum_{j=1}^8 \frac{x_{i,t}}{0.72} \varphi_j(t) w_{ij} \right) \\ \text{s.t. } & \sum_{i \in S} x_{i,t} w_{ij} \leq 6000(1 - \xi_{j,t}) \quad j = 1, 2, \dots, 8 \\ & \sum_{j=1}^8 w_{ij} = 1 \quad \forall i \in S \\ & w_{ij} = \{0, 1\} \quad \forall i \in S \\ & 0 < \xi_{j,t} < 0.1 \quad j = 1, 2, \dots, 8 \end{aligned}$$

其中， $w_{ij}$  为逻辑变量，其含义为供货商  $i$  所分配的转运商是否为转运商  $j$ ，“0”表示为否，“1”表示为是。 $\varphi_j(t)$  表示为第  $t$  周转运商  $j$  的运输损耗率估计值， $\xi_{j,t}$  基于原则③设置，是对供货商供货不稳定性的预期，其取值范围为  $(0, 0.1)$ 。

#### 6.4 模型的求解

订购方案模型由 24 个多变量规划模型组成，每周的规划模型其约束受到前一周订货方案的影响，主要体现在每周接收量的差异以及原材料消耗量的差异，导致的库存变化，即每周的订货方案安排是受到上一周生产消耗情况的影响的。而转运模型同样由 24 个多变量规划模型组成，其求解的基础为本周的订购方案。因此在求解订货与转运方案时，逐周对式（5）模型进行求解，先求出第  $t-1$  周的订购方案，在基于订购方案得到本周的转运方案，更新第  $t-1$  周的库存情况，作为第  $t$  周的求解基础。

在求解订购方案时，基于题目信息设定初始条件。基于假设 3 可知，第一周时仓库的库存是充足的，不失一般性的设定仓库中三种原材料的库存存量可生产的产品数量是一致的，即每种原材料均可生产 1.88 万立方米产品。设该生产企业每周的生产消耗的原材料为等比例的，即每周分别使用每种原材料生产 0.94 万立方米产品。在订购方案的模型中，考虑了第  $t$  周供货商  $i$  供货时的转运损耗率  $\alpha_{i,t}$ ，就损耗率来说，其变化的范围较小，在附件 2 的历史数据中范围大概在  $[0, 0.05]$ ，因此取历史损耗率的均值作为每周供货商供货时转运损耗率的估计值，取  $\alpha_{i,t} = 1.34\%$ 。

基于上述对模型的简化，本文通过遗传算法对每周的订购方案和转运方案模型进行逐周求解，所得结果如附件 A 所示，遗传算法求解过程如下：

#### 遗传算法求解过程

**Step 1** 设定初始状态，参数以及编码等，导入数据

**Step 2** 初始化，开始迭代进化

**Step 3** 对可行解进行选择、重组、变异和合并

**Step 4** 依据目标函数值的大小分配适应度值

**Step 5** 计算出当代最优个体的序号**Step 6** 重复 3、4、5 三个步骤，直至进化到符合最优或达到最大遗传代数**Step 7** 结果解码输出

在求解转运方案模型时，本文通过 Python 程序对该分配模型进行模拟迭代求解，直至损耗成本的变化达到终止条件时，即将结果作为最优转运方案输出。在进行模拟前，本文对参数  $\xi_j$  进行了如下设定：当转运商转运的原材料总量分配已经接近 5400 立方米时，如果下一家需转运的供货商供货量超过了 300 立方米，则不由该转运商转运该笔订单，且在后续的分配中，不再考虑该转运商；如果不超过 300 立方米，则任然由该公司转运，在后续的分配中，不再考虑该转运商。

两个模型的求解代码以及结果参见附录 3，及附件 A、B。

## 6.5 订购方案及转运方案实施效果分析

对于实施效果，本文从宏观和微观两个角度对实施结果进行展示及分析。在宏观角度下，展示和分析未来 24 周订购方案和转运方案实施时，每周的库存和产量的动态变化，以及成本和损耗的情况。在微观角度下，展示和分析第 3 周和第 13 周的订购方案和转运方案实施效果，检验本周的方案是否合理。

### 6.5.1 宏观角度：未来 24 周方案实施的效果分析

在评估订购方案与转运方案的实施效果时，本文构建了三个指标进行分析，分别为每周库存原材料可生产产品（简称“库存产量”）、每周原材料占比情况和每周转运损耗情况。其中库存产量和每周转运损耗情况均换算为可生产产品的数量，以便比较和分析。库存产量指标的构建方式如下所示：

$$V_t = X_{Receive,t} + V_{t-1} - E_t \quad (6)$$

其中， $V_t$  即为第  $t$  周库存产量， $X_{Receive,t}$  为第  $t$  周的接收量， $E_t$  为每周产能，式 (6) 所有变量的值均经过换算，其单位均为一立方米产品。其中  $E_t$  的值为定值 2.82 万立方米产品，初始库存产量  $V_0 = 5.64$  万立方米产品。第  $t$  周接收量的计算考虑了可能存在的供货商供货不稳定性和转运商的运输损耗，其计算方式如下：

$$X_{Receive,t} = (X_{Order,t} + \epsilon(t)) \cdot (1 - \alpha(t))$$

其中， $\epsilon(t)$  为供货不稳定性风险函数， $\alpha(t)$  为运输损失风险函数。这两个函数是通过高斯过程对 5 年 10 个周期（24 周为一个周期）内的历史数据所得的，与任一供货商或转运商无关，是在宏观上对原材料不稳定性和转运损耗的模拟，基于式 (4) 拟合所得的分布函数，随机生成对供货不稳定性和转运损耗这两个宏观层面的随机变量的估计值。

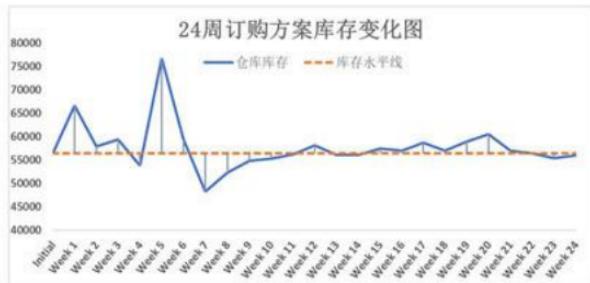


图 6: 24 周订购方案库存变化图

上图即为库存产量在该订购方案和转运方案下 24 周的变化情况。如图可知，库存产量总是在基准水平的上下浮动。当库存超过 5.64 万立方米时，下一周的实际接收量总是会更少，使得库存回归到基准水平。类似地，在库存产量低于基准水平时，下一周的实际接收量总是会更多，以使得库存回归到基准水平。就每周库存产量的变化来看，其中与基准水平差异较小的情况可以解释为由于供货商供货不稳定以及运输损耗带来的小幅度差异，属于正常水平；尽管其中出现了库存产量与基准水平差异较大的情况，如第 5 周和第 7 周，但值得注意的是，在下一周库存产量马上回归到了基准水平周围。此差异较大现象可能是由于供货商的供货不稳定性造成，而库存产量再下一周迅速回归正常水平，这一现象表明了本文所提出的模型对库存成本的高敏感性，说明了模型，使得能更加合理的制定每周订购方案。

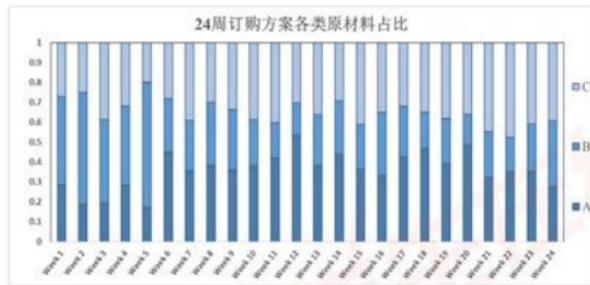


图 7: 24 周订购方案各类原材料占比

上图即为 24 周订购方案中，订货量中各类原材料的占比情况。如上图所示，三种原材料尽管在生产成本上存在一定差异，而本文所建立的模型，虽然考虑了各产品单价，以及生产消耗，但遗传算法求解所得的最优解中，并未出现对某种原材料的偏向性。

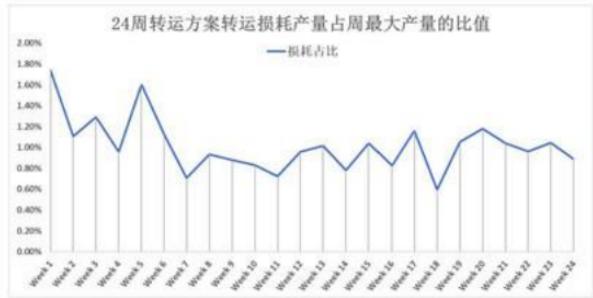


图 8：24 周转运方案转运损耗产量占周最大产量的比值

上图即为每周转运损耗情况，由损耗产量估计值与周最大产量的比值度量。损耗产量估计值考虑损耗存在的一般情况，将原材料损耗换算为可生产产品，以估计损耗产量，其计算方式如下：

$$x_{loss,t} = \frac{x_{Supply,t} \cdot \alpha_t}{28400 \times P_i} \times 100\%$$

其中  $\alpha_t$  为转运商  $j$  的历史损耗率的均值， $P_i$  为原材料与产品的换算系数，其取值取决于原材料种类。再 24 周的转运方案实施过程中，损耗产量占最大产能的比例平均值为 1.02%。

### 6.5.2 微观角度：第 3 周和第 13 周的方案实施效果合理性分析

#### (1) 第 3 周订购方案及转运方案

第 3 周的订货方案中，估计的接收量换算为可生产产品，为 31130.71 立方米，该值偏离 2.82 万立方米的原因是由于供货商供货的不稳定性以及运输损耗。为分析第 3 周订购方案的合理性，我们结合的第 2 周库存情况进行考虑，在第 2 周估计的库存产量为 57860.64 万立方米，第 2 周的库存是满足生产企业的稳定性需要的，即尽可能使得库存产量满足两周需求，再此情况下，第 2 周的订购方案在考虑到供货不稳定性和运输损耗时，其订货总量应当与最大产量的差别不大，从第 3 周的结果上看，显然是这样的。因此，第 3 周的订购方案在生产企业稳定性角度看，是合理的。



图 9：第 3 周各原材料供货商供货占比

有关第 3 周的订货数据如上图表所示，本文展示了三种原材料的供货商供货分布。从各原材料的供货商供货分布来看，原材料 A 和原材料 C 的供货商分布较为均衡，不存在过度依赖某一供货商的情况；原材料 B 的供货商分布较为

特别，S139 供货商供货的比例达到了 55%，其实际供货原材料 B 的数目为 4731 立方米，换算为可生产产品数目为 7168 立方米左右，占到了一周产能的 25%。S139 对原材料 B 的供货商在第三周的供货规律如下所示：

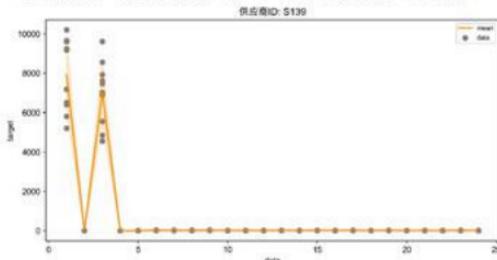


图 10：供货商 S139 的历史供货数据 (以 24 周为一周期)

由于本模型构建基于供货商供货能力的周期性规律，即通过过去 10 个周期的供货数据的最大值和最小值作为供货能力的区间边界，第 3 周出现如上订购方案的原因，是因为 S139 在过去 10 个周期内的供货历史数据是遵循历史规律的，因此导致出现了供货能力急剧提升的情况。对此，本文认为 S139 应当是一家通过种植生产原材料的供货商，由于经济作物的生长规律限制，以至于其供货往往是半年两次左右，且单次可供货数量较多。

## (2) 第 13 周订购方案及转运方案

第 13 周的订货方案中，估计的接收量换算为可生产产品，为 27849 立方米，该值偏离 2.82 万立方米的原因是由于供货商供货的不稳定性以及运输损耗。为分析第 13 周订购方案的合理性，我们结合的第 12 周库存情况进行考虑，在第 12 周估计的库存产量为 58080.48 万立方米，第 12 周的库存是满足生产企业的稳定性需要的，即尽可能使得库存产量满足两周需求，再此情况下，第 13 周的订购方案在考虑到供货不稳定性和运输损耗时，其订货总量应当与最大产量的差别不大，从第 13 周的结果上看，显然是这样的。因此，第 13 周的订购方案在生产企业稳定性角度看，是合理的。

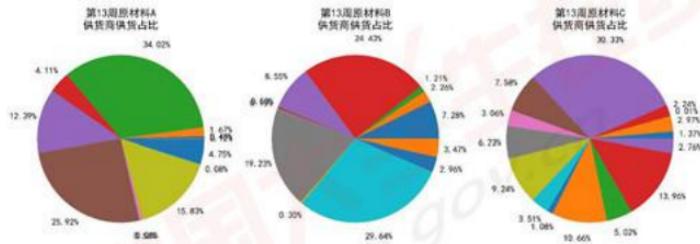


图 11：第 13 周各原材料供货商供货比例

有关第 13 周的订货数据如上图表所示，本文展示了三种原材料的供货商供货结构。从各原材料的供货商供货分布来看，三种原材料的供货商分布较为均衡，不存在过度依赖某一供货商的情况。

## 七、问题三的模型建立与求解

企业希望进一步加强供货链成本管理，本小问主要探究如何通过对原材料选购偏好，在未来的 24 周内分配对于原材料的更优的订单量，从而进一步降低转运和仓储成本。

通过资料得知，供货链企业选择原材料时需要衡量采购成本、产能成本这两大方面。采购成本是购买材料的现金流开销，影响公司显性收益。虽然问题二的模型就此以“订单成本”为优化目标，得到采购成本最经济的方案。然而我们并未发现问题二所得结果中存在对某种原材料的偏向性。这是因为木质纤维材料的采购成本远大于材料的产能成本，而在问题二模型的求解过程中采购成本的下降往往掩盖原材料的产能成本仍然虚高的现象。原材料的偏向性与其产能成本相关。经市场调研数据显示<sup>[4]</sup>，产能成本是一个公司或组织为提供或提高其大规模经营能力而发生的费用，衡量为企业实现单位产能需要支出的费用。由题干信息可知，该企业生产相同单位产品所消耗的原材料 A、原材料 B、原材料 C 的比值为 1:1.1:1.2；而相同单位原材料 A、原材料 B、原材料 C 的比值的价格比值为 1.2:1.11。已知比例关系，进一步挖掘出不同材料的产能成本特征：

$$Cost_A = 0.72C_{Purchase} + 0.6(C_{Trans} + C_{Store})$$

$$Cost_B = 0.726C_{Purchase} + 0.66(C_{Trans} + C_{Store})$$

$$Cost_C = 0.72C_{Purchase} + 0.72(C_{Trans} + C_{Store})$$

由上述公式可知，原材料 A、C 的采购费用并不区分两者的产能成本，均为  $0.72C_{Purchase}$ ；而对于生产单位产品所需转运、仓储的费用，单位原材料 A 的开销比单位原材料 C 更大。因此，用原材料 A 进行生产的成本更低。

企业为了压缩单位产能的开销，需要尽量多地采购 A 类和尽量少地采购 C 类原材料。因此，本小问将在问题二模型结果的基础上改进原有订购方案模型，在控制原方案下采购材料 A、C 等效的产能保持不变的情况下，尽可能减少原材料 A、C 订单所涉及的转运、仓储的费用。

### 7.1 模型的更新

#### 7.1.1 订购方案目标函数的更新

由于问题三要求尽量多地采购 A 类和尽量少地采购 C 类原材料，并未提及 B 类原材料。因此考虑在保持 B 类原材料的供货商以及订货量不变的情况下对 A 类和 C 类原材料的供货商选择以及订货量进行进一步优化。

$$\min C_{Trans} \left( \sum_{i \in A \cap S} x'_{i,t} + \sum_{i \in C \cap S} x'_{i,t} \right) + C_{Store}(V'_{A,t} + V'_{C,t})$$

#### 7.1.2 订购方案约束条件的更新

为保持每周计划产能不变，需要保证调整 A 类和 C 类原材料前后可生产的产品总量保持不变，即：

$$\frac{X'_{A,t-1}}{0.6} + \frac{X'_{C,t-1}}{0.72} = \frac{X_{A,t-1}}{0.6} + \frac{X_{C,t-1}}{0.72}$$

同时还需保证库存内可生产的产品总量保持不变，即：

$$\frac{E'_{A,t-1}}{0.6} + \frac{E'_{C,t-1}}{0.72} = \frac{E_{A,t-1}}{0.6} + \frac{E_{C,t-1}}{0.72}$$

为了保证优化结果为尽量多地采购 A 类和尽量少地采购 C 类原材料，对 C 类原材料的订货量区间进行限制；对 A 类原材料的订货量区间进行放宽。

### 7.1.3 订购方案模型汇总

如下即为基于问题二的模型，添加对订单结构约束的模型，其初始的输入方案即为问题二所得的最优订购方案。

$$\begin{aligned} \min \quad & C_{Trans} \left( \sum_{i \in A \cap S} x'_{i,t} + \sum_{i \in C \cap S} x'_{i,t} \right) + C_{Store} (V'_{A,t} + V'_{C,t}) \\ \text{s.t.} \quad & \frac{V'_{A,t-1}}{0.6} + \frac{V'_{C,t-1}}{0.72} = \frac{V_{A,t-1}}{0.6} + \frac{V_{C,t-1}}{0.72} \\ & \frac{X'_{A,t-1}}{0.6} + \frac{X'_{C,t-1}}{0.72} = \frac{X_{A,t-1}}{0.6} + \frac{X_{C,t-1}}{0.72} \\ & z_{i,t}^{\min} \leq z_{i,t} \leq z_{i,t}^{\max} \quad i \in A \\ & z_{i,t}^{\min} \leq z_{i,t} \leq z_{i,t}^{\max} \quad i \in C \\ & x_{i,t} = z_{i,t} + \epsilon_{i,t} \quad i \in A \cup C \\ & x_{i,t} \geq 0 \quad i \in S \end{aligned}$$

### 7.1.4 转运方案模型说明

在问题三的背景之下，转运方案模型不需要作出修改更新。问题二转运模型的输入为本周的订购方案，该转运模型总是以损耗最小产能为优化目标，求解出最优的转运分配方案。由此可知，企业生产情况的改变，对转运方案的模型是不存在任何影响的，因此，转运方案模型不需要做出更新。

### 7.2 模型的求解

订购方案和转运方案模型虽然存在一定的更新，但是从模型结构的角度看，模型并未产生结构上的变化，因此同样可以考虑使用遗传算法对模型进行求解，遗传算法间接以及求解过程，参照上文所述，求解结果如附件 A、B 所示。

### 7.3 新方案实施效果分析

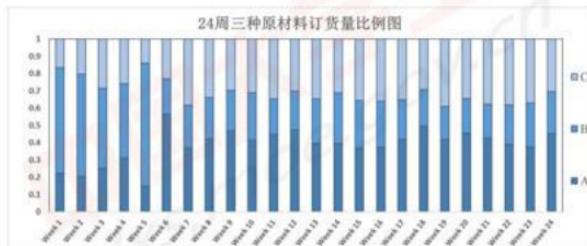


图 12: 24 周三种原材料订货量比例图

如上图所示，即为新的订购方案中，每周订购的原材料构成比例，通过原材料换算，统一换算为可生产产品的数量。相较于问题二模型所得结果不难看出，新的订购方案中，原材料A的订货比例明显再大多数情况下是多于C的。

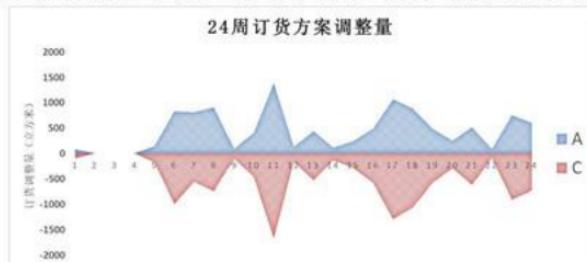


图 13: 24 周订货方案调整量

如上图所示，模型再经过修正后，新的订购方案显然对原材料订购比例进行了调整，且调整结果较为显著，本文认为该结果达到了原材料的最优订购结构。

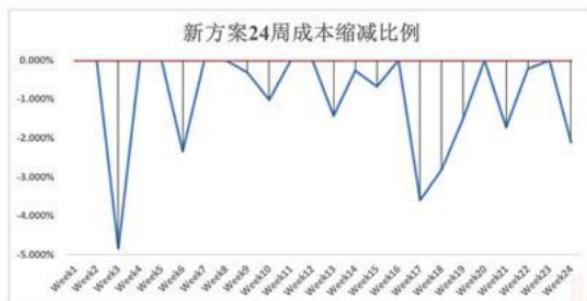


图 14: 新方案 24 周成本缩减比例

对于新的订购方案，在优化了原材料订购结构的情况下，理论上生产成本可以得到进一步的压缩。本文针对问题二和问题三的订购方案，对成本进行量化比较分析。如下图所示，通过对订购方案的成本进行量化估计，计算除了新方案相较于问题二方案的成本变化比率。其中成本最高减幅达到了 5% 左右，综合来看，新方案的平均成本缩减比例为 0.956% 左右。由此可见，新方案虽然对成本起到了压缩的效果，但效果并不显著，对此，本文认为，是由于可选供货商过少，供货商的供货能力也只是恰好满足每周的最大生产要求，因此该模型订购结构的可行域本身并不大，故而差异并不是非常显著。

## 八、问题四的模型建立与求解

问题四题干可知，该企业通过技术改造已具备了提高产能的潜力，可认为该企业的生产力可以满足一切产能需求。即该企业每周的产能已不再成为限制其订货总量的因素。同时，企业也不再存在存储问题，即每周消耗原材料等于各个供货商在当周的供货量。因此，本文考虑该公司产能提升的潜力，基于问

题二的订购方案与转运方案模型，对模型进行一定的修正，以得到新的模型。

### 8.1 模型的更新

#### 8.1.1 供货商的重新选择

由于企业已具备提高产能的潜力，为取得充足的原材料供货量，供货商的选取不应该再局限于问题二中的 39 家企业，而应当放宽到所有具有良好供货能力的供货商。该生产企业的产能提升潜力，确保该生产企业有足够的能力应对供货商的不稳定性，即不会出现库存的积压导致成本急剧上升的情况；再者，由于供货商的增加，生产企业的常规最大产能，即 2.82 万立方米，是更加容易满足的。基于如上分析，本文在问题四的模型中将数据预处理后得到的 369 家良好供货商均作为备选供货商考虑。

#### 8.1.2 订购方案模型目标函数的更新

首先，库存情况将不在作为约束被考虑在模型当中，因为该生产企业具备了在生产层面上应对供货商供货不稳定性的能力，即超出原有最大产能的原材料部分，不再需要被放置在仓库中，而是可以直接被用于生产。因此，仓库的原材料库存产能将恒定维持在 5.64 万立方米，而仓库成本也成了一个固定成本，因此考虑将仓储成本从订购方案模型的目标函数中删除。

#### 8.1.3 订购方案模型约束的更新

如上文所述，仓库成本成为了一个固定成本，在问题四的模型中，将不再需要考虑仓库的情况，因此，考虑将订购方案模型的约束中，仓储有关部分删除。

考虑将原有最大产能设置为该生产企业每周产能的下限，将转运商的总转运能力作为每周产能限制的上限。本文认为，在产能具备提升潜力的情况下，可以认为生产企业总是能把每周接收的原材料全部用于本周的生产消耗。而在不考虑调用库存的情况下，生产企业每周的生产上限即为每周的接收量总数，由于转运商数目有限，且其转运能力有限，可推知，该生产企业每周的订单数量的上限，即为所有转运商每周的最大转运能力。即每周最大订单量为 4.8 万立方米。即：

$$\sum_{t \in S} z_{i,t} \leq 4.8 \times 10^4$$

由于新模型考虑拓宽供货商渠道，以增加订货总量，对于这家具备产能提升潜力的生产企业，考虑将其原最大产能设置为每周产能的下线，即规定每周生产量不得少于原产能，即：

$$\frac{\sum_{t \in A \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.6} + \frac{\sum_{t \in B \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.66} + \frac{\sum_{t \in C \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.72} \geq 2.82 \times 10^4$$

#### 8.1.4 更新后的订购方案模型

如下即为考虑产能提升潜力情况，在问题二的订购方案模型的基础上更新得到的产能提升订购方案模型：

$$\begin{aligned} \min_s & C_{Trans} \sum_s x_{i,t} + C_{Purchase} \left( \sum_{i \in A \cap S} 1.2x_{i,t} + \sum_{i \in B \cap S} 1.x_{i,t} + \sum_{i \in C \cap S} x_{i,t} \right) \\ \text{s.t.} & \sum_{i \in S} z_{i,t} \leq 4.8 \times 10^4 \\ & z_{i,t}^{\min} \leq z_{i,t} \leq z_{i,t}^{\max} \quad i \in S \\ & \frac{\sum_{i \in A \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.6} + \frac{\sum_{i \in B \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.66} + \frac{\sum_{i \in C \cap S} x_{i,t} (1 - \alpha_{i,t})}{0.72} \geq 2.82 \times 10^4 \\ & x_{i,t} = z_{i,t} + \epsilon_{i,t} \quad i \in S \end{aligned}$$

### 8.1.5 转运方案模型的说明

在问题四的背景之下，转运方案模型不需要作出修改更新。问题二转运模型的输入为本周的订购方案，该转运模型总是以损耗最小产能为优化目标，求解出最优的转运分配方案。由此可知，企业生产情况的改变，对转运方案的模型是不存在任何影响的，因此，转运方案模型不需要做出更新。

### 8.2 模型的求解与结果展示

订购方案和转运方案模型虽然存在一定的更新，但是从模型结构的角度看，模型并未产生结构上的变化，因此同样可以考虑使用遗传算法对模型进行求解，遗传算法间接以及求解过程，参照上文所述，求解结果如附件 A、B 所示。

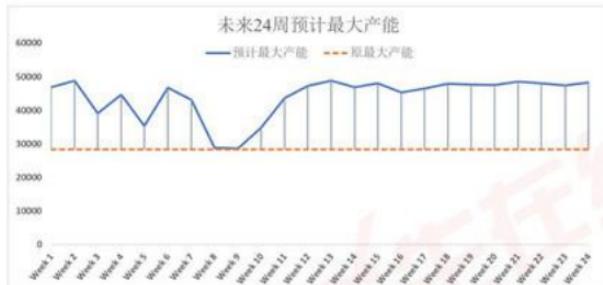


图 15：未来 24 周预计最大产能

对于问题四的产能提升订购方案模型，其产能的提升效果如下图所示。结合预计最大产能和原最大产能对比来看，不难看出，在考虑了产能提升潜力后的订购方案模型，其输出的新的订购方案，显著的提升了产能。其中产能提升最大值为 20490 立方米左右，最小值为 298 立方米左右，平均每周提升产能为 15790 立方米左右，可见产能具备提升潜力时，该生产企业的产能可得到显著提升。

## 九、模型的评价与改进方向

## 9.1 模型的优点

- (1) 问题一的模型利用 TOPSIS 法，对六个重要指标赋予权重，得到各供货商质量评分，指标的选取全面地考虑了已知信息，例如对度量供货连续性指标时通过熵权法构建评价体系，使得模型更加可靠；
- (2) 问题二的对订货与运输方案构造从问题一中的高质量供货商出发，以保障企业生产的稳定性为前提，对订货与运输方案的制定尽可能全面地考虑其影响因素，且尽量避免主观因素的影响，对随机变量也通过使用如高斯过程回归拟合以及统计分布拟合进行预测估计，使得模型更加客观且真实；
- (3) 问题三的订货与运输方案以问题二的模型为基础，考虑到实际情况，并对事件进行仿真模拟，使得模型结果可信度更高；
- (4) 问题四的订货与运输方案同样以问题二的模型为基础，进一步放宽约束条件，使企业产能得到充分释放，模型结果表现优秀。

## 9.2 模型的缺点

- (1) 对小部分数据进行删除，忽略了这部分数据带来的影响，不够全面；
- (2) 再建模过程中，对部分约束的研究还不够深入，未能分析其影响结果的逻辑。

## 9.3 模型的改进方向

- (1) 问题一中对供货商质量分析采用的是 TOPSIS 法，可以考虑采用其余的模型来分析附件 1 中的定价规律；
- (2) 问题二中构建的订货与运输方案模型可以考虑更多约束条件，使模型更加符合现实。

## 十、参考文献

- [1]Larry C. Giunipero,Reham Aly Eltantawy. Securing the upstream supply chain: a risk management approach[J]. International Journal of Physical Distribution & Logistics Management,2004,34(9);
- [2]梁樱.供货链管理模式下企业采购管理方法[N].中国会计报,2021-08-13(007).
- [3]唐林彬,梁樑,浦徐进.供货链中供货存在周期性波动情况下的合作模型[J].系统工程,2004(08):24-27.
- [4]南通市建设工程造价管理处(通建价[2020]37号)

## 十一、附录

附录 1：支撑材料文件列表

附录 2：补充表格、图片和公式推导

附录 3：Python 代码

## 附录 1：支撑材料文件列表

### 文件名列表

附件 A 订购方案数据结果.xlsx  
附件 B 转运方案数据结果.xlsx  
删除数据.xlsx  
数据处理.py  
问题一代码.py  
南通市建设工程造价管理处（通建价 [2020]37 号）.pdf

### 问题二相关代码

供货货能力（高斯过程拟合）.py  
问题二供货商筛选模型.py  
问题二订购方案模型.py  
问题二转运模型.py

### 问题三相关代码

问题三订购方案模型.py  
问题三转运模型.py

### 问题四相关代码

问题四订购方案模型.py  
问题四转运模型.py

## 附录 2：补充表格、图片和公式推导

### 数据预处理后删除的 33 家极低质量供货商

供货商 ID	订单计数	供货计数	供货总量	缺货大于 100	缺货大于 1000	间隔个数	平均间隔天数	平均连续供货天数
S015	54	15	28	7	7	14	16.07	1.15
S034	77	14	30	8	0	10	17.40	1.40
S047	89	28	66	4	4	19	9.53	1.47
S096	54	28	73	7	7	20	10.60	1.47
S097	74	30	63	18	8	19	11.05	1.67
S118	70	28	66	9	0	20	9.05	1.40
S137	84	19	33	7	0	13	14.08	1.46
S153	56	13	175	12	0	8	22.88	1.63
S158	51	20	37	7	7	16	13.75	1.33
S160	68	20	45	15	15	18	12.00	1.11
S161	72	20	47	17	17	16	13.50	1.25
S162	89	14	28	7	0	12	14.42	1.17
S164	78	22	72	7	0	17	11.00	1.29
S173	39	16	372	7	0	9	11.11	1.78
S183	82	15	28	11	0	10	19.40	1.50

S200	73	30	65	8	0	17	10.53	1.76
S201	49	28	81989	14	13	6	35.33	4.67
S215	76	14	50	10	0	12	14.58	1.17
S222	39	15	32	7	0	12	14.33	1.25
S228	57	15	151	9	0	9	19.56	1.50
S231	78	16	31	10	0	12	13.83	1.33
S246	72	18	44	13	13	9	22.22	1.80
S277	82	16	30	8	0	13	13.15	1.23
S285	79	19	35	18	0	15	14.73	1.36
S293	67	8	192	8	0	7	33.14	1.33
S302	89	25	59	7	0	20	10.75	1.32
S334	72	10	148	12	0	10	22.40	1.00
S335	52	18	50	7	7	15	12.27	1.20
S343	70	11	28	7	0	8	22.00	1.38
S354	64	23	57	9	0	15	14.47	1.64
S369	67	26	146	10	0	12	12.83	2.17
S380	50	25	85	2	1	17	10.82	1.47
S396	91	25	38	8	0	21	10.24	1.25

TOPSIS 算法流程

---

### TOPSIS 算法流程

---

#### Input:

供营商数据集  $X = \{x_1, x_2, \dots, x_n\}, x_i \in R^m \forall i = 1, \dots, n$

各指标权重  $w = (w_1, w_2, \dots, w_m), w_i \in R^n \forall i = 1, \dots, m$

#### Process:

0. 对供营商数据集中的指标属性同向化  $X'$

1. 构造向量归一化后的标准化矩阵  $Z = \{z_1, z_2, \dots, z_n\}$

2. 对  $Z$  每一列  $z_i$

    2.1 最劣方案  $Z^-$  的第  $i$  维度  $\leftarrow z_i$  元素最小值

    2.2 最优方案  $Z^+$  的第  $i$  维度  $\leftarrow z_i$  元素最大值

3. 对  $Z$  每一列  $z_i$

    3.1  $z_i$  与最优方案的接近程度  $D_i^+ \leftarrow$  式(7.1)

    3.2  $z_i$  与最劣方案的接近程度  $D_i^- \leftarrow$  式(7.2)

    3.3  $z_i$  与最优方案的接近程度  $C_i \leftarrow$  式(8)

4. 根据  $C_i$  大小进行排序

---

#### Output: 各供营商 TOPSIS 评价结果

---

完整符号表

符号	含义	单位
$n_i$	供货商 <i>i</i> 的供货次数	\
$m_i$	供货商 <i>i</i> 的平均供货量	$m^3$
$x_i^{max}$	供货商 <i>i</i> 的单次最大供货量	\
$\delta_i$	供货商 <i>i</i> 的供货稳定性	\
$\Delta_i$	供货商 <i>i</i> 的供货连续性	$m^3$
$\gamma_i$	供货商 <i>i</i> 的合理供货比例	\
$s_i$	供货商 <i>i</i> 的重要性评分	\
$n$	供货商总数	\
$S$	选择供货商	\
$y_i$	是否选择原材料供货商 <i>i</i>	\
$z_{i,t}$	第 <i>t</i> 周企业对供货商 <i>i</i> 订单量	$m^3$
$x_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 实际供货量	$m^3$
$\epsilon_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 实际供货偏差量	$m^3$
$\hat{x}_{i,t}$	第 <i>t</i> 周供货商 <i>i</i> 预测供货量	$m^3$
$V_{A,t}$	第 <i>t</i> 周原材料A的库存	$m^3$
$V_{B,t}$	第 <i>t</i> 周原材料B的库存	$m^3$
$V_{C,t}$	第 <i>t</i> 周原材料C的库存	$m^3$
$E_{A,t}$	第 <i>t</i> 周原材料A的消耗量	$m^3$
$E_{B,t}$	第 <i>t</i> 周原材料B的消耗量	$m^3$
$E_{C,t}$	第 <i>t</i> 周原材料C的消耗量	$m^3$
$C_{Trans}$	单位原材料运输费用	$元/m^3$
$C_{Purchase,A}$	单位原材料A采购费用	$元/m^3$
$C_{Purchase,B}$	单位原材料B采购费用	$元/m^3$
$C_{Purchase,C}$	单位原材料C采购费用	$元/m^3$
$C_{Store}$	单位原材料储存费用	$元/m^3$
$O_t$	在第 <i>t</i> 周的目标产能	$m^3$

## 附录 2 Python 代码

### 数据预处理代码

```
import numpy as np
import pandas as pd
import math

data=pd.DataFrame(pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx',sheet_name=0))
data1=pd.DataFrame(pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx',sheet_name=1))

data_a = data.loc[data['材料分类'] == 'A'].reset_index(drop=True)
data_b = data.loc[data['材料分类'] == 'B'].reset_index(drop=True)
data_c = data.loc[data['材料分类'] == 'C'].reset_index(drop=True)

data1_a = data1.loc[data1['材料分类'] == 'A'].reset_index(drop=True)
data1_b = data1.loc[data1['材料分类'] == 'B'].reset_index(drop=True)
data1_c = data1.loc[data1['材料分类'] == 'C'].reset_index(drop=True)

data_a.to_excel('订货A.xlsx')
data_b.to_excel('订货B.xlsx')
data_c.to_excel('订货C.xlsx')

data1_a.to_excel('供货A.xlsx')
data1_b.to_excel('供货B.xlsx')
data1_c.to_excel('供货C.xlsx')

num_a=(data_a == 0).astype(int).sum(axis=1)
num_b=(data_b == 0).astype(int).sum(axis=1)
num_c=(data_c == 0).astype(int).sum(axis=1)

num_a = (240-np.array(num_a)).tolist()
num_b = (240-np.array(num_b)).tolist()
num_c = (240-np.array(num_c)).tolist()

total_a=data1_a.sum(axis=1).to_list()
total_b=data1_b.sum(axis=1).to_list()
total_c=data1_c.sum(axis=1).to_list()

#平均供货量
a=[]
b=[]
c=[]
for i in range(len(total_a)):
    a.append(total_a[i]/num_a[i])
for i in range(len(total_b)):
    b.append(total_b[i]/num_b[i])
for i in range(len(total_c)):
    c.append(total_c[i]/num_c[i])

data_a = pd.DataFrame(pd.read_excel('A.xlsx',sheet_name=1))
data_b = pd.DataFrame(pd.read_excel('B.xlsx',sheet_name=1))
data_c = pd.DataFrame(pd.read_excel('C.xlsx',sheet_name=1))

a=np.array(data_a)
a=np.delete(a, [0,1], axis=1)
b=np.array(data_b)
b=np.delete(b, [0,1], axis=1)
```

```
c=np.array(data_c)
c=np.delete(c, [0,1], axis=1)

a1=a.tolist()
b1=b.tolist()
c1=c.tolist()

def count(a1):
    tem1=[]
    for j in range(len(a1)):
        tem=[]
        z=[]
        for i in range(len(a1[j])):
            if a1[j][i]!=0:
                tem.append(z)
                z=[]
            else:
                z.append(a1[j][i])
        list1 = [x for x in tem if x != []]
        tem1.append(list1)
    return tem1

def out2(tem1):
    tem = []
    for i in range(len(tem1)):
        if len(tem1[i]) == 0 :
            tem.append(0)
        else:
            a=0
            for j in range(len(tem1[i])):
                a=a+len(tem1[i][j])
            tem.append(a/len(tem1[i]))
    return tem

def out1(tem1):
    tem = []
    for i in range(len(tem1)):
        if len(tem1[i]) == 0 :
            tem.append(0)
        else:
            tem.append(len(tem1[i]))
    return tem

tem1 = count(a1)
tem = out1(tem1)
print(tem)
tem = out2(tem1)
print(tem)

tem1 = count(b1)
tem = out1(tem1)
print(tem)
tem = out2(tem1)
print(tem)

tem1 = count(c1)
tem = out1(tem1)
print(tem)
tem = out2(tem1)
print(tem)
```

## 问题一代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn import preprocessing

#熵权法
def cale(data):
    s = 0
    for i in data:
        if i == 0:
            s = s+0
        else:
            s = s+i*math.log(i)
    return s/(-math.log(len(data)))

def get_beta(data,a=402):
    name=data.columns.to_list()
    del name[0]
    beta=[]
    for i in name:
        t=np.array(data[i]).reshape(a,1)
        min_max_scaler = preprocessing.MinMaxScaler()
        X_minMax = min_max_scaler.fit_transform(t)
        beta.append(cale(X_minMax.reshape(1,a).reshape(a,)))
    return beta

tdata = pd.DataFrame(pd.read_excel('表格1.xlsx'))
beta = get_beta(tdata,a=369)

con=[]
for i in range(3):
    a=(1-beta[4+5])/(3-(beta[5]+beta[6]+beta[7]))
    con.append(a)
a=np.array(tdata[['间隔个数']]*con[0]+np.array(tdata[['平均间隔天数']]*con[1]+np.array(tdata[['平均连续供货天数']])*con[2])
print(con) #熵权法确定的供货连续性系数

#topsis
def topsis(data1, weight=None,a=402):
    # 归一化
    t = np.array(data1[['供货总量','平均供货量(供货总量/供货计数)','稳定性(累加(供货量-订单量)^2)','供货极差','满足比例(在20%误差内)','连续性']]).reshape(a,6)
    min_max_scaler = preprocessing.MinMaxScaler()
    data = pd.DataFrame(min_max_scaler.fit_transform(t))

    # 最优最劣方案
    Z = pd.DataFrame([data.min(), data.max()], index=['负理想解', '正理想解'])

    # 距离
    weight = entropyWeight(data1) if weight is None else np.array(weight)
    Result = data.copy()
    Result['正理想解'] = np.sqrt(((data - Z.loc['正理想解'])) ** 2 * weight).sum(axis=1)
    Result['负理想解'] = np.sqrt(((data - Z.loc['负理想解'])) ** 2 * weight).sum(axis=1)

    # 综合得分指数
```

```

Result['综合得分指数'] = Result['负理想解'] / (Result['负理想解'] + Result['正理想解'])
Result['排序'] = Result.rank(ascending=False)[['综合得分指数']]

return Result, Z, weight

def entropyWeight(data):
    data = np.array(data[['供货总量', '平均供货量(供货总量/供货计数)', '稳定性(累加(供货量-订单量)
^2)', '供货极差', '满足比例(在20%误差内)', '连续性']])
    # 归一化
    P = data / data.sum(axis=0)

    # 计算熵值
    E = np.nansum(-P * np.log(P) / np.log(len(data)), axis=0)
    # 计算权系数
    return (1 - E) / (1 - E).sum()

tdata = pd.DataFrame(pd.read_excel('表格2.xlsx'))
Result, Z, weight = topsis(tdata, weight=None, a=369)
Result.to_excel('结果2.xlsx')

```

## 问题二代码

#### 供应货能力 (高斯过程拟合)

```

'S080',
'S218',
'S189',
'S086',
'S210',
'S074',
'S007',
'S273',
'S292']

#####
# 导入数据 #####
file0 = pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx', sheet_name = '企业的订货量 (m³)')
file1 = pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx', sheet_name = '供应商的供货量 (m³)')
#####

for supplier in select_suppliers:
    plt.figure(figsize=(10, 5), dpi = 400)
    lw = 2
    X = np.tile(np.arange(1,25),(1,10)).T
    X_plot = np.linspace(1, 24, 24)
    y = np.array(file1[file1['供应商ID'] == supplier].iloc[:,2:]).ravel()
    descrip = pd.DataFrame(np.array(file1[file1['供应商ID'] == supplier].iloc[:,2:]).reshape(-1,24)).describe()
    y_mean = descrip.loc['mean',:]
    y_std = descrip.loc['std',:]
    plt.scatter(X, y, c='grey', label='data')
    plt.plot(X_plot, y_mean, color='darkorange', lw=lw, alpha = 0.9,
              label='mean')

    plt.fill_between(X_plot, y_mean - 1.* y_std, y_mean + 1. * y_std, color='darkorange',
                     alpha=0.2)
    plt.xlabel('data')
    plt.ylabel('target')
    plt.title(f'供应商ID: {supplier}')
    plt.legend(loc="best", scatterpoints=1, prop={'size': 8})
    plt.savefig(f'./img/供应商ID_{supplier}.png')
    plt.show()

```

## 问题二订货方案模型

```

'S139',
'S340',
'S282',
'S308',
'S275',
'S329',
'S126',
'S131',
'S356',
'S268',
'S306',
'S307',
'S352',
'S247',
'S284',
'S365',
'S031',
'S040',
'S364',
'S346',
'S294',
'S055',
'S338',
'S080',
'S218',
'S189',
'S086',
'S210',
'S074',
'S007',
'S273',
'S292']

gp_dict = {
    'A': [],
    'B': [],
    'C': []
}

#####
# 导入数据 #####
select_info = pd.read_excel('P2-1优化结果.xlsx')
file0 = pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx', sheet_name = '企业的订货量 (m³)')
file1 = pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx', sheet_name = '供应商的供货量 (m³)')
loss_ratio_df = pd.read_excel('附件2 近5年8家转运商的相关数据.xlsx')

x_A_pred_list = pd.read_excel('P2-2A结果.xlsx', index = False).to_numpy()
x_B_pred_list = pd.read_excel('P2-2B结果.xlsx', index = False).to_numpy()
x_C_pred_list = pd.read_excel('P2-2C结果.xlsx', index = False).to_numpy()

#####
# 处理导入数据 #####
order_volume = file0[file0['供应商ID'].isin(select_suppliers)].iloc[:,2:]
supply_volume = file1[file1['供应商ID'].isin(select_suppliers)].iloc[:,2:]
error_df = (supply_volume - order_volume)/order_volume

y_mean_list, y_std_list = [], []
#### 在24周期内，计算8家转运商的平均转运损耗率 #####
for i in range(loss_ratio_df.shape[0]):
    plt.figure(figsize=(10, 5), dpi = 400)
    lw = 2
    X = np.tile(np.arange(1,25),(1,10)).T

```

```

X_plot = np.linspace(1, 24, 24)
y = np.array(loss_ratio_df[loss_ratio_df['转运商ID'] == 'T'+str(i+1)].iloc[:,1:]).ravel()
sub_df = pd.DataFrame(np.array(loss_ratio_df[loss_ratio_df['转运商ID'] == 'T'+str(i+1)].iloc[:,1:]).reshape(-1,24))
y_mean = sub_df.mean(axis = 0)
y_std = sub_df.std(axis = 0)
y_mean_list.append(y_mean)
y_std_list.append(y_std)
plt.scatter(X, y, c='grey', label='data')
plt.plot(X_plot, y_mean, color='darkorange', lw=lw, alpha = 0.9,
         label='mean')

plt.fill_between(X_plot, y_mean - 1.* y_std, y_mean + 1. * y_std, color='darkorange',
                  alpha=0.2)
plt.xlabel('周期')
plt.ylabel('转运损耗率(%)')
plt.title(f'转运商ID: T{str(i+1)})')
plt.legend(loc="best", scatterpoints=1, prop={'size': 8})
plt.show()

alpha = np.array(y_mean_list).mean(axis = 0)

select_A_ID = select_info[select_info['材料分类'] == 'A']['供应商ID']
select_B_ID = select_info[select_info['材料分类'] == 'B']['供应商ID']
select_C_ID = select_info[select_info['材料分类'] == 'C']['供应商ID']
select_suppliers = pd.unique(select_info['供应商ID'])

num_of_a = select_A_ID.shape[0]
num_of_b = select_B_ID.shape[0]
num_of_c = select_C_ID.shape[0]

# ##### 训练高斯过程回归模型，预测供应商供应误差率 #####
def GP_reg(i):
    type_ = file0[file0['供应商ID'].isin(select_suppliers)].iloc[i, 1]
    np.random.seed(i)

    y = error_df.iloc[i, :]
    X = order_volume.iloc[i, :]
    X = np.array(X[(y < 1000)]).reshape(-1, 1)
    y = np.array(y[(y < 1000)]).ravel()
    k = GPy.kern.RBF(1)
    m = GPy.models.GPRegression(X, y.reshape(-1, 1), k, normalizer=True)
    m.optimize()
    print(m)

    dy = np.sqrt(list(m['Gaussian_noise.variance'])[@]) * np.random.random(y.shape)

    # Instantiate a Gaussian Process model
    kernel = C(list(m['rbf.variance'])[@], (1e-3, 1e3)) * RBF(list(m['rbf.lengthscale'])[@],
    (1e-5, 1e5))
    # kernel = C(0.985, (1e-3, 1e3)) * RBF(0.377, (1e-5, 1e3))
    gp = GaussianProcessRegressor(kernel=kernel, alpha=dy ** 2,
                                  n_restarts_optimizer=10)

    # Fit to data using Maximum Likelihood Estimation of the parameters
    gp.fit(X, y)
    gp_dict[type_].append(gp)

    # Make the prediction on the meshed x-axis (ask for MSE as well)
    x = np.atleast_2d(np.linspace(0, max(X)[@], 1000)).T

```

```

y_pred, sigma = gp.predict(x, return_std=True)

# Plot the function, the prediction and the 95% confidence interval based on
# the MSE
plt.figure(dpi=400)
# plt.errorbar(X.ravel(), y, dy, fmt='r.', markersize=10, label='Observations')
plt.plot(x, y, 'r.', markersize=10, label='Observations')
plt.plot(x, y_pred, 'b-', label='Prediction')
plt.fill(np.concatenate([x, x[::-1]]),
         np.concatenate([y_pred - 1.9600 * sigma,
                        (y_pred + 1.9600 * sigma)[::-1]]),
         alpha=.5, fc='b', ec='None', label='95% confidence interval')
plt.xlabel('订单量 (m³)')
plt.ylabel('误差率 (%)')
plt.legend(loc='upper left')
supplier = file0[file0['供应商ID'].isin(select_suppliers)].iloc[i, 0]

plt.title(f'原料{type_}供应商{supplier} | 偏差率和订单量高斯过程拟合图')
plt.show()

```

##### 构建问题2-2模型 #####

```

def get_range_list(type_, week_i = 1):
    df = file1[file1['材料分类'] == type_].iloc[:, 2:]
    min_ = df.iloc[:, np.arange(week_i - 1, 240, 24)].min(axis = 1)
    max_ = df.iloc[:, np.arange(week_i - 1, 240, 24)].max(axis = 1)
    range_ = list(zip(min_, max_*1.5))
    range_ = [[i[0], i[1]+1] for i in range_]
    return range_

```

##### 问题2-2模型目标函数 #####

```

Output_list = []
def aimfunc(Phen, V_a, V_b, V_c, CV, NIND):
    global V_a_list, V_b_list, V_c_list
    global V_tol_list, _prepare_list
    global x_a_pred_list, x_b_pred_list, x_c_pred_list
    global df_A_all, df_B_all, df_C_all
    global Output_list

    C_trans = 1,
    C_purchase = 100,
    C_store = 1

    def error_pred(z, type_):
        a = np.array([gp_dict[type_][i].predict(z[:, [i]], return_std=True) for i in
range(z.shape[1])])
        err_pred = a[:, 0, :]
        err_std = a[:, 1, :]
        np.random.seed(1)
        nor_err = np.random.normal(err_pred, err_std)
        err = np.transpose(nor_err) * 1e-2
        err *= z
        #           err = -0.0001*z
        return err

    def convert(x, type_):
        if type_ == 'a': x_ = x * 1.2
        if type_ == 'b': x_ = x * 1.1
        if type_ == 'c': x_ = x

```

```

    return x_

z_a = Phen[:, 0:num_of_a]
z_b = Phen[:, num_of_a: num_of_a + num_of_b]
z_c = Phen[:, num_of_a + num_of_b:]

errorA = error_pred(z_a, 'A')
errorB = error_pred(z_b, 'B')
errorC = error_pred(z_c, 'C')

x_a0, x_b0, x_c0 = z_a + errorA, z_b + errorB, z_c + errorC
x = np.hstack([x_a0, x_b0, x_c0])
x_a = convert(x_a0, 'a')
x_b = convert(x_b0, 'b')
x_c = convert(x_c0, 'c')

# constraint 5
# E_a/0.6 = E_b/0.66 = E_c/0.72 = 0.96 = 2.82/3
#     e_a = max(np.ones_like(V_a)*0.8, 2.82*1e4*V_a/(V_a *0.6 + V_b *0.66 + V_c * 0.72))
#     e_b = max(np.ones_like(V_b)*0.8, 2.82*1e4*V_b/(V_a *0.6 + V_b *0.66 + V_c * 0.72))
#     e_c = max(np.ones_like(V_c)*0.8, 2.82*1e4*V_c/(V_a *0.6 + V_b *0.66 + V_c * 0.72))
e_a = 0.9
e_b = 0.9
e_c = 0.9

#     # constraint 2 ~ 4
V_a2 = V_a * (1 - e_a) + x_a.sum(axis=1)
V_b2 = V_b * (1 - e_b) + x_b.sum(axis=1)
V_c2 = V_c * (1 - e_c) + x_c.sum(axis=1)

Output = (V_a * e_a * 0.6) + (V_b * e_b * 0.66) + (V_c * e_c * 0.72)
Output_list.append(Output)

f = C_trans * x.sum(axis=1) + C_purchase * (x_a.sum(axis=1) + x_b.sum(axis=1) +
x_c.sum(axis=1))
f = f.reshape(-1, 1)

# constraint 1
CV1 = (2.82 * 1e4 * 2 - x_a.sum(axis=1) * (1 - alpha[week_i - 1]) / 0.6 - x_b.sum(axis=1) *
(
    1 - alpha[week_i - 1]) / 0.66 - x_c.sum(axis=1) * (1 - alpha[week_i - 1]) /
0.72)
#     print('【进化内部】CV1', CV1.mean())
CV1 = CV1.reshape(-1, 1)

CV2 = (z_a.sum(axis=1) + z_b.sum(axis=1) + z_c.sum(axis=1) - 4.8 * 1e4).reshape(-1, 1)
CV1 = np.hstack([CV1, CV2])

# update 库存
#     print('【进化内部】库存总量: ', (V_a2+V_b2+V_c2).mean())
return [f, CV1, V_a2, V_b2, V_c2, x_a0, x_b0, x_c0]

##### 绘制图形 #####
def plot_pie(week_i, variable):
    global V_a_list, V_b_list, V_c_list
    global V_tol_list, O_prepare_list
    global x_a_pred_list, x_b_pred_list, x_c_pred_list
    global df_A_all, df_B_all, df_C_all
    global x_A_all, x_B_all, x_C_all

    select_A_ID = file1[file1['材料分类'] == 'A']['供应商ID']

```

```

select_B_ID = file1[file1['材料分类'] == 'B']['供应商ID']
select_C_ID = file1[file1['材料分类'] == 'C']['供应商ID']

num_of_a = select_A_ID.shape[0]
num_of_b = select_B_ID.shape[0]
num_of_c = select_C_ID.shape[0]

tol_v = [variable[0, : num_of_a].sum(), \
          variable[0, num_of_a:num_of_a + num_of_b].sum(), \
          variable[0, num_of_a + num_of_b:].sum()]
print(f'原材料A在第{week_i}周订单量总额{tol_v[0]}\n'
原材料B在第{week_i}周订单量总额{tol_v[1]}\n'
原材料C在第{week_i}周订单量总额{tol_v[2]}\n'
\n'
原材料第{week_i}周订单量总额:{sum(tol_v)})'

##### plot A #####
fig = plt.figure(dpi = 400)
# fig = plt.figure(figsize = (7,7))
explode = (0.04,) * num_of_a
labels = select_A_ID
plt.pie(variable[0, : num_of_a], explode = explode, labels = labels, autopct='%.1f%%')
plt.title(f'原材料A在第{week_i}周订单量比例')
# plt.savefig(f'./img/P2-2-Aweek{week_i}.png')
plt.show()

##### plot B #####
fig = plt.figure(dpi = 400)
# fig = plt.figure(figsize = (7,7))
explode = (0.04,) * num_of_b
labels = select_B_ID
plt.pie(variable[0, num_of_a:num_of_a+num_of_b], explode = explode, labels = labels,
autopct='%.1f%%')
plt.title(f'原材料B在第{week_i}周订单量比例')
# plt.savefig(f'./img/P2-2-Bweek{week_i}.png')
plt.show()

##### plot C #####
fig = plt.figure(dpi = 400)
# fig = plt.figure(figsize = (7,7))
explode = (0.04,) * num_of_c
labels = select_C_ID
plt.pie(variable[0, num_of_a+num_of_b:], explode = explode, labels = labels,
autopct='%.1f%%')
plt.title(f'原材料C在第{week_i}周订单量比例')
plt.savefig(f'./img/P2-2-Cweek{week_i}.png')
plt.show()

df_A = pd.DataFrame(dict(zip(select_A_ID, variable[0, : num_of_a])), index=[f'Week{week_i}'])
df_B = pd.DataFrame(dict(zip(select_B_ID, variable[0, num_of_a:num_of_a + num_of_b])), index=[f'Week{week_i}'])
df_C = pd.DataFrame(dict(zip(select_C_ID, variable[0, num_of_a + num_of_b:])), index=[f'Week{week_i}'])

df_A_all = pd.concat([df_A_all, df_A])
df_B_all = pd.concat([df_B_all, df_B])
df_C_all = pd.concat([df_C_all, df_C])

def run_algorithm(week_i):

```

```

global V_a_list, V_b_list, V_c_list
global V_tol_list, O_prepare_list
global x_a_pred_list, x_b_pred_list, x_c_pred_list
global df_A_all, df_B_all, df_C_all
global x_A_all, x_B_all, x_C_all
global output_list

"""=====变量设置=====
tol_num = file1.shape[0]

z_a = get_range_list('A', week_i) # 第一个决策变量范围
z_b = get_range_list('B', week_i) # 第一个决策变量范围
z_c = get_range_list('C', week_i) # 第一个决策变量范围

B = [[1, 1]] * tol_num # 第一个决策变量边界, 1表示包含范围的边界, 0表示不包含
D = [1, ] * tol_num
ranges = np.vstack([z_a, z_b, z_c]).T # 生成自变量的范围矩阵, 使得第一行为所有决策变量的下界, 第二行为上界
borders = np.vstack(B).T # 生成自变量的边界矩阵
varTypes = np.array(D) # 决策变量的类型, 0表示连续, 1表示离散
"""=====染色体编码设置=====
Encoding = 'BG' # 'BG'表示采用二进制/格雷编码
codes = [0, ] * tol_num # 决策变量的编码方式, 设置两个0表示两个决策变量均使用二进制编码
precisions = [4, ] * tol_num # 决策变量的编码精度, 表示二进制编码串解码后能表示的决策变量的精度可达到小数点后6位
scales = [0, ] * tol_num # 0表示采用算术刻度, 1表示采用对数刻度
FieldD = ea.crtfld(Encoding, varTypes, ranges, borders, precisions, codes, scales) # 调用函数创建译码矩阵

"""=====遗传算法参数设置=====
# NIND = 1000; # 种群个体数目
MAXGEN = 100; # 最大遗传代数
maxormins = [1] # 列表元素为1则表示对应的目标函数是最小化, 元素为-1则表示对应的目标函数是最大化
maxormins = np.array(maxormins) # 转化为Numpy array向量
selectStyle = 'rws' # 采用轮盘赌选择
recStyle = 'xovdp' # 采用两点交叉
mutStyle = 'mutbin' # 采用二进制染色体的变异算子
Lind = int(np.sum(FieldD[0, :])) # 计算染色体长度
pc = 0.8 # 交叉概率
pm = 1 / Lind # 变异概率
obj_trace = np.zeros((MAXGEN, 2)) # 定义目标函数值记录器
var_trace = np.zeros((MAXGEN, Lind)) # 染色体记录器, 记录历代最优个体的染色体
"""=====开始遗传算法进化=====
start_time = time.time() # 开始计时
Chrom = ea.crtpc(Encoding, NIND, FieldD) # 生成种群染色体矩阵
variable = ea.bs2ri(Chrom, FieldD) # 对初始种群进行解码
CV = np.zeros((NIND, 1)) # 初始化一个CV矩阵 (此时因为未确定个体是否满足约束条件, 因此初始化元素为0, 假认为所有个体是可行解个体)

V_a = V_a_list[-1]
V_b = V_b_list[-1]
V_c = V_c_list[-1]
ObjV, CV, V_a_new, V_b_new, V_c_new, x_a, x_b, x_c = aimfunc(variable, V_a, V_b, V_c, CV, NIND) # 计算初始种群个体的目标函数值

FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
# 开始进化
for gen in tqdm(range(MAXGEN), leave=False):
    SelCh = Chrom[ea.selecting(selectStyle, FitnV, NIND - 1), :] # 选择
    SelCh = ea.recombin(recStyle, SelCh, pc) # 重组
    SelCh = ea.mutate(mutStyle, Encoding, SelCh, pm) # 变异

```

```

# 把父代精英个体与子代的染色体进行合并，得到新一代种群
Chrom = np.vstack([Chrom[best_ind, :], SelCh])
Phen = ea.bs2ri(Chrom, FieldD) # 对种群进行解码(二进制转十进制)
ObjV, CV, V_a_new, V_b_new, V_c_new, x_a, x_b, x_c = aimfunc(Phen, V_a, V_b, V_c, CV,
NIND) # 求种群个体的目标函数值
#         if len(CV[CV>0])/len(CV) > 0.2:
#             print('CV > 0')
#             V_a_new, V_b_new, V_c_new = V_a, V_b, V_c
#             break
FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
# 记录
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
obj_trace[gen, 0] = np.sum(ObjV) / ObjV.shape[0] # 记录当代种群的目标函数均值
obj_trace[gen, 1] = ObjV[best_ind] # 记录当代种群最优个体目标函数值
var_trace[gen, :] = Chrom[best_ind, :] # 记录当代种群最优个体的染色体
# 进化完成
end_time = time.time() # 结束计时
ea.trcplot(obj_trace, [[‘种群个体平均目标函数值’, ‘种群最优个体目标函数值’]]) # 绘制图像

"""
=====输出结果=====
"""

best_gen = np.argmax(obj_trace[:, [1]])
print(‘最优解的目标函数值：’, obj_trace[best_gen, 1])
variable = ea.bs2ri(var_trace[[best_gen], :], FieldD) # 解码得到表现型（即对应的决策变量值）
#     print(‘最优解的决策变量值：’)
#     for i in range(variable.shape[1]):
#         print(‘z’+str(i)+‘=’, variable[0, i])
V_tol = (V_a_new + V_b_new + V_c_new)[best_gen];
V_tol_list.append(V_tol)
O_prepare = (V_a_new / 0.6 + V_b_new / 0.66 + V_c_new / 0.72)[best_gen];
O_prepare_list.append(O_prepare)
x_a_pred = x_a[best_gen]
x_b_pred = x_b[best_gen]
x_c_pred = x_c[best_gen]
x_a_pred_list.append(x_a_pred)
x_b_pred_list.append(x_b_pred)
x_c_pred_list.append(x_c_pred)
print(‘库存总量：’, V_tol)
print(‘预备产能：’, O_prepare)

print(‘该周产能：’, Output_list[-1][best_gen])
output_list.append(Output_list[-1][best_gen])
print(‘用时：’, end_time - start_time, ‘秒’)

x_A = pd.DataFrame(dict(zip(select_A_ID, x_a_pred)), index=[f‘Week{week_i}’])
x_B = pd.DataFrame(dict(zip(select_B_ID, x_b_pred)), index=[f‘Week{week_i}’])
x_C = pd.DataFrame(dict(zip(select_C_ID, x_c_pred)), index=[f‘Week{week_i}’])

x_A_all = pd.concat([x_A_all, x_A])
x_B_all = pd.concat([x_B_all, x_B])
x_C_all = pd.concat([x_C_all, x_C])

# update V_list
V_a_list.append(V_a_new)
V_b_list.append(V_b_new)
V_c_list.append(V_c_new)

return variable

#####
运用遗传算法求解 #####
def pred_Nweeks(N=24):

```

```

global V_a_list, V_b_list, V_c_list
global V_tol_list, O_prepare_list
global x_a_pred_list, x_b_pred_list, x_c_pred_list
global df_A_all, df_B_all, df_C_all
global x_A_all, x_B_all, x_C_all

NIND = 1000; # 种群个体数目
V_a_init = np.tile(np.array(0.94 * 1e4), NIND) * 2
V_b_init = np.tile(np.array(0.94 * 1e4), NIND) * 2
V_c_init = np.tile(np.array(0.94 * 1e4), NIND) * 2
V_a_list, V_b_list, V_c_list = [V_a_init], [V_b_init], [V_c_init]
df_A_all = pd.DataFrame([[]] * len(select_A_ID), index=select_A_ID).T
df_B_all = pd.DataFrame([[]] * len(select_B_ID), index=select_B_ID).T
df_C_all = pd.DataFrame([[]] * len(select_C_ID), index=select_C_ID).T

x_A_all = pd.DataFrame([[]] * len(select_A_ID), index=select_A_ID).T
x_B_all = pd.DataFrame([[]] * len(select_B_ID), index=select_B_ID).T
x_C_all = pd.DataFrame([[]] * len(select_C_ID), index=select_C_ID).T

V_tol_list, O_prepare_list = [], []
x_a_pred_list, x_b_pred_list, x_c_pred_list = [], [], []

for week_i in range(1, N + 1):
    print(f'***** Week {week_i} *****')
    variable = run_algorithm(week_i)
    plot_pie(week_i, variable)

return [df_A_all, df_B_all, df_C_all, x_a_pred_list, x_b_pred_list, x_c_pred_list,
        V_tol_list, O_prepare_list]

df_A_all, df_B_all, df_C_all, x_a_pred_list, x_b_pred_list, x_c_pred_list, V_tol_list,
O_prepare_list = pred_Nweeks(N=24)

```

## 问题二供货商筛选模型

```

import numpy as np
import geatpy as ea # 导入geatpy库
import time
import pandas as pd
import matplotlib.pyplot as plt

cost_dict = {
    'A': 0.6,
    'B': 0.66,
    'C': 0.72,
}

#####
# 导入数据 #####
data = pd.read_excel('第一问结果.xlsx')
cost_trans = pd.read_excel('附件2 近5年8家转运商的相关数据.xlsx')
supplier_id = data['供应商ID']
score = data['综合得分指数'].to_numpy()
pred_volume = data['平均供货量(供货总量/供货计数)'].to_numpy()

output_volume_list = []
for i in range(len(pred_volume)):
    output_volume = pred_volume[i] / cost_dict[data['材料分类'][i]]
    output_volume_list.append(output_volume)

```

```

output_volume_array = np.array(output_volume_list)
file1 = pd.read_excel('附件1 近5年402家供应商的相关数据.xlsx', sheet_name = '供应商的供货量 (m³)')
##### 构建问题2-1模型 #####
def aimfunc(Y, CV, NIND):
    coef = 1
    cost = (cost_trans.mean()/100).mean()
    f = np.divide(Y.sum(axis = 1).reshape(-1,1),coef)*(Y * np.tile(score, (NIND,1))).sum(axis = 1).reshape(-1,1)
    CV = -(Y * np.tile(output_volume_array * (1-cost), (NIND,1))).sum(axis = 1) +
    (np.ones((NIND))*2.82*1e4)
    CV = CV.reshape(-1,1)
    return [f, CV]

#####
# 使用遗传算法求解 #
#####

##### 变量设置 #####
X = [[0, 1]]*50 # 决策变量范围
B = [[1, 1]]*50 # 决策变量边界, 1表示包含范围的边界, 0表示不包含
D = [1,] * 50
ranges=np.vstack(X).T # 生成自变量的范围矩阵, 使得第一行为所有决策变量的下界, 第二行为上界
borders=np.vstack(B).T # 生成自变量的边界矩阵
varTypes = np.array(D) # 决策变量的类型, 0表示连续, 1表示离散
#####
# 染色体编码设置 #
Encoding = 'BG' # 表示采用二进制/格雷编码
codes = [0,]*50 # 决策变量的编码方式, 设置两个0表示两个决策变量均使用二进制编码
precisions = [4,]*50 # 决策变量的编码精度, 表示二进制编码串解码后能表示的决策变量的精度可达到小数点后6位
scales = [0,]*50 # 0表示采用算术刻度, 1表示采用对数刻度
FieldD = ea.crtfld(Encoding, varTypes, ranges, borders, precisions, codes, scales) # 调用函数创建译码矩阵
#####
# 遗传算法参数设置 #
NIND = 100; # 种群个体数目
MAXGEN = 200; # 最大遗传代数
maxormins = [1] # 列表元素为1则表示对应的目标函数是最小化, 元素为-1则表示对应的目标函数是最大化
maxormins = np.array(maxormins) # 转化为Numpy array行向量
selectStyle = 'rws' # 采用轮盘赌选择
recStyle = 'xovdp' # 采用两点交叉
mutStyle = 'mutbin' # 采用二进制染色体的变异算子
Lind = int(np.sum(FieldD[0, :])) # 计算染色体长度
pc = 0.7 # 交叉概率
pm = 1/Lind # 变异概率
obj_trace = np.zeros((MAXGEN, 2)) # 定义目标函数值记录器
var_trace = np.zeros((MAXGEN, Lind)) # 染色体记录器, 记录历代最优个体的染色体
#####
# 开始遗传算法进化 #
start_time = time.time() # 开始计时
Chrom = ea.crtpc(Encoding, NIND, FieldD) # 生成种群染色体矩阵
variable = ea.bs2ri(Chrom, FieldD) # 对初始种群进行解码
CV = np.zeros((NIND, 1)) # 初始化一个CV矩阵(此时因为未确定个体是否满足约束条件, 因此初始化元素为0, 假认为所有个体是可行解个体)
ObjV, CV = aimfunc(variable, CV, NIND) # 计算初始种群个体的目标函数值
FitnV = ea.rankling(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
#####
# 开始进化 #
for gen in range(MAXGEN):
    SelCh = Chrom[ea.selecting(selectStyle, FitnV, NIND-1),:] # 选择
    SelCh = ea.recombin(recStyle, SelCh, pc) # 重组
    SelCh = ea.mutate(mutStyle, Encoding, SelCh, pm) # 变异
    # 把父代精英个体与子代的染色体进行合并, 得到新一代种群
    Chrom = np.vstack([Chrom[best_ind, :], SelCh])
    Phen = ea.bs2ri(Chrom, FieldD) # 对种群进行解码(二进制转十进制)
    ObjV, CV = aimfunc(Phen, CV, NIND) # 求种群个体的目标函数值

```

```

FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
# 记录
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
obj_trace[gen,0]=np.sum(ObjV)/ObjV.shape[0] #记录当代种群的目标函数均值
obj_trace[gen,1]=ObjV[best_ind] #记录当代种群最优个体目标函数值
var_trace[gen,:]=Chrom[best_ind,:] #记录当代种群最优个体的染色体
#####
进化完成 #####
end_time = time.time() # 结束计时
fig = plt.figure(figsize = (10,20), dpi = 400)
ea.trcplot(obj_trace, [['种群个体平均目标函数值', '种群最优个体目标函数值']]) # 绘制图像
plt.show()
#####
输出结果 #####
best_gen = np.argmax(obj_trace[:, [1]])
print('最优解的目标函数值: ', obj_trace[best_gen, 1])
variable = ea.bs2ri(var_trace[[best_gen], :], FieldID) # 解码得到表现型（即对应的决策变量值）
print('最优解的决策变量值为: ')
material_dict = {'A':0, 'B':0, 'C':0}
for i in range(variable.shape[1]):
    if variable[0, i] == 1:
        print('x'+str(i)+ '=' ,variable[0, i], '原材料类别:',data['材料分类'][i])
        material_dict[data['材料分类'][i]] += 1
print('共选择个数: '+str(variable[0,:].sum()))
print('用时: ', end_time - start_time, '秒')
print(material_dict)
#%%
#####
模型2-1 结果: 选择39家贸易供应商 #####
select_idx = [i for i in range(variable.shape[1]) if variable[0, i] == 1];
select_suppliers = supplier_id[select_idx]

df_out = file1[file1['供应商ID'].isin(select_suppliers)].reset_index(drop = True)
df_out.to_excel('P2-1优化结果.xlsx', index = False)
#####

```

## 问题二转运模型

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import math
from scipy.stats import exponnorm
from fitter import Fitter

#对各转运商的损失率进行预估
data=pd.DataFrame(pd.read_excel('2.xlsx'))
a=np.array(data)
a=np.delete(a, 0, axis=1)
loss = []

for j in range(8):
    f = Fitter(np.array(a[j].tolist()), distributions='exponnorm')
    f.fit()
    k=1/(f.fitted_param['exponnorm'][0]*f.fitted_param['exponnorm'][2])
    count = 0
    tem = []
    while count < 24:
        r = exponnorm.rvs(k, size=1)
        if r > 0 and r<5:
            tem.append(float(r))

```

```
        count = count+1
    else:
        pass
    loss.append(item)

nloss=np.array(loss).T.tolist()

#求解转运方案

rdata=pd.DataFrame(pd.read_excel('P2-2B.xlsx'))
sa = rdata.columns.tolist()

for j in range(24):

    d = {'序号':[1,2,3,4,5,6,7,8],'损失':nloss[j]}
    tem=pd.DataFrame(d).sort_values(by='损失')
    d1 = pd.DataFrame({'商家':sa[0:12],'订单量':np.array(rdata)[j][0:12]}).sort_values(by='订单量',ascending = [False])
    d2 = pd.DataFrame({'商家':sa[12:24],'订单量':np.array(rdata)[j][12:24]}).sort_values(by='订单量',ascending = [False])
    d3 = pd.DataFrame({'商家':sa[24:39],'订单量':np.array(rdata)[j][24:39]}).sort_values(by='订单量',ascending = [False])
    new = d1.append(d2)
    new = new.append(d3).reset_index(drop=True)
    count1 = 0
    tran = 0
    arr = []
    for i in range(39):
        if new['订单量'][i] == 0:
            arr.append(0)
        else:
            if new['订单量'][i] > 6000:
                re = new['订单量'][i]-6000
                count1 = count1 +1
                tran = 0
                arr.append(item['序号'].tolist()[count1])
            if re >6000:
                re = re -6000
                count1 = count1 +1
                if re >6000:
                    count1 = count1 +1
            else:
                tran = tran + new['订单量'][i]
                if tran < 5500:
                    arr.append(item['序号'].tolist()[count1])

                elif tran > 6000:
                    count1 = count1 +1
                    tran = new['订单量'][i]

                    arr.append(item['序号'].tolist()[count1])

                else:
                    tran = 0

                    arr.append(item['序号'].tolist()[count1])
                    count1 = count1 +1
    new['转运']=arr
    if j ==0 :
        result = new.copy()
```

```
    else:  
        result = pd.merge(result,new,on='商家')  
print(result)
```

### 问题三订货方案模型

```
import numpy as np  
import geatpy as ea # 导入geatpy库  
import time  
import pandas as pd  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
  
pre_A = []  
pre_C = []  
now_A = []  
now_C = []  
cost_pre = []  
cost_now = []  
  
x_A_pred_list = pd.read_excel('P2-2A结果.xlsx', index=False).to_numpy()  
x_C_pred_list = pd.read_excel('P2-2C结果.xlsx', index=False).to_numpy()  
select_info = pd.read_excel('P2-1优化结果.xlsx')  
select_A_ID = select_info[select_info['材料分类'] == 'A']['供应商ID']  
select_B_ID = select_info[select_info['材料分类'] == 'B']['供应商ID']  
select_C_ID = select_info[select_info['材料分类'] == 'C']['供应商ID']  
  
num_of_a = select_A_ID.shape[0]  
num_of_b = select_B_ID.shape[0]  
num_of_c = select_C_ID.shape[0]  
  
select_suppliers = pd.unique(select_info['供应商ID'])  
  
file0 = pd.read_excel('附件1.xlsx', sheet_name = '企业的订货量 (m³)')  
file1 = pd.read_excel('附件1.xlsx', sheet_name = '供应商的供货量 (m³)')  
order_volume = file0[file0['供应商ID'].isin(select_suppliers)].iloc[:,2:]  
supply_volume = file1[file1['供应商ID'].isin(select_suppliers)].iloc[:,2:]  
error_df = (supply_volume - order_volume)/order_volume  
  
def get_range_list(type_, week_i=1):  
    df = select_info[select_info['材料分类'] == type_].iloc[:, 2:]  
    min_ = df.iloc[:, np.arange(week_i - 1, 240, 24)].min(axis=1)  
    max_ = df.iloc[:, np.arange(week_i - 1, 240, 24)].max(axis=1)  
  
    if type_ == 'A':  
        range_ = list(zip(min_, max_* 1.5))  
    elif type_ == 'C':  
        range_ = list(zip(min_, max_* 0.9))  
    range_ = [[i[0], i[1] + 1] for i in range_]  
  
    return range_  
  
def aimfunc(Phen, V_a, V_c, CV, NIND):  
    global week_i, x_A_pred_list, x_C_pred_list
```

```

global V_a2_previous, V_c2_previous
global C_trans, C_store

def error_pred(z, type_):
    err = - 1e-4
    err *= z
    return err

z_a = Phen[:, :num_of_a]
z_c = Phen[:, num_of_a:]

errorA = error_pred(z_a, 'A')
errorC = error_pred(z_c, 'C')
x_a = (z_a + errorA) # /0.6
x_c = (z_c + errorC) # /0.72
x = np.hstack([x_a, x_c])
# constraint 4
E_a, E_c = (2.82 * 1e4 * 1 / 8), (2.82 * 1e4 * 1 / 8)

# constraint 2 ~ 3
V_a2_ = V_a - E_a / 0.6 + x_a.sum(axis=1)
V_c2_ = V_c - E_c / 0.72 + x_c.sum(axis=1)

V_a2 = x_a.sum(axis=1)
V_c2 = x_c.sum(axis=1)

f = C_trans * x.sum(axis=1) + C_store * (V_a2 + V_c2)
f = f.reshape(-1, 1)

# constraint 1
CV1 = np.abs((V_a2 / 0.6 + V_c2 / 0.72) - (V_a2_previous / 0.6 + V_c2_previous / 0.72))
CV1 = CV1.reshape(-1, 1)

return [f, CV1, V_a2_, V_c2_]

```

```

def run_algorithm(V_a_list, V_c_list, week_i):
    global V_a2_previous, V_c2_previous
    global V_a2_previous_list, V_c2_previous_list
    global C_trans, C_store

    V_a2_previous, V_c2_previous = V_a2_previous_list[week_i - 1], V_c2_previous_list[week_i - 1]
    """*****变量设置*****"""
    tol_num = num_of_a + num_of_c

    z_a = get_range_list('A', week_i) # 第一个决策变量范围
    z_c = get_range_list('C', week_i) # 第一个决策变量范围

    B = [[1, 1]] * tol_num # 第一个决策变量边界，1表示包含范围的边界，0表示不包含
    D = [1, ] * tol_num
    ranges = np.vstack([z_a, z_c]).T # 生成自变量的范围矩阵，使得第一行为所有决策变量的下界，第二行为上界
    borders = np.vstack(B).T # 生成自变量的边界矩阵
    varTypes = np.array(D) # 决策变量的类型，0表示连续，1表示离散
    """*****染色体编码设置*****"""
    Encoding = 'BG' # 'BG'表示采用二进制/格雷编码
    codes = [0, ] * tol_num # 决策变量的编码方式，设置两个0表示两个决策变量均使用二进制编码
    precisions = [5, ] * tol_num # 决策变量的编码精度，表示二进制编码串解码后能表示的决策变量的精度可达到小数点后6位
    scales = [0, ] * tol_num # 0表示采用算术刻度，1表示采用对数刻度

```

```

#     print(len(varTypes),len(ranges),len(borders),len(precisions),len(codes),len(scales))
FieldD = ea.crtfld(Encoding, varTypes, ranges, borders, precisions, codes, scales) # 调用函数创建译码矩阵
"""
=====遗传算法参数设置=====
# NIND      = 1000; # 种群个体数目
MAXGEN = 300; # 最大遗传代数
maxormins = [1] # 列表元素为1则表示对应的目標函数是最小化，元素为-1则表示对应的目標函数是最大化
maxormins = np.array(maxormins) # 转化为Numpy array行向量
selectStyle = 'rws' # 采用轮盘赌选择
recStyle = 'xovdp' # 采用两点交叉
mutStyle = 'mutbin' # 采用二进制染色体的变异算子
Lind = int(np.sum(FieldD[0, :])) # 计算染色体长度
pc = 0.5 # 交叉概率
pm = 1 / Lind # 变异概率
obj_trace = np.zeros((MAXGEN, 2)) # 定义目标函数数值记录器
var_trace = np.zeros((MAXGEN, Lind)) # 染色体记录器，记录历代最优个体的染色体
"""
=====开始遗传算法进化=====
start_time = time.time() # 开始计时
Chrom = ea.crtpc(Encoding, NIND, FieldD) # 生成种群染色体矩阵
variable = ea.bs2ri(Chrom, FieldD) # 对初始种群进行解码
CV = np.zeros((NIND, 1)) # 初始化一个CV矩阵（此时因为未确定个体是否满足约束条件，因此初始化元素为0，暂认为所有个体是可行解个体）

V_a = V_a_list[-1]
V_c = V_c_list[-1]
# print(V_a.shape[0], V_c.shape[0])
ObjV, CV, V_a_new, V_c_new = aimfunc(variable, V_a, V_c, CV, NIND) # 计算初始种群个体的目标函数值

FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
# 开始进化
for gen in tqdm(range(MAXGEN)):
    SelCh = Chrom[ea.selecting(selectStyle, FitnV, NIND - 1), :] # 选择
    SelCh = ea.recombin(recStyle, SelCh, pc) # 重组
    SelCh = ea.mutate(mutStyle, Encoding, SelCh, pm) # 变异
    # 把父代精英个体与子代的染色体进行合并，得到新一代种群
    Chrom = np.vstack([chrom[best_ind, :], SelCh])
    Phen = ea.bs2ri(Chrom, FieldD) # 对种群进行解码(二进制转十进制)
    ObjV, CV, V_a_new, V_c_new = aimfunc(Phen, V_a, V_c, CV, NIND) # 求种群个体的目标函数值
    # if np.mean(CV) > 0:
    #     print('CV > 0')
    #     V_a_new, V_c_new = V_a,V_c
    #     break
    FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
    # 记录
    best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
    obj_trace[gen, 0] = np.sum(ObjV) / ObjV.shape[0] # 记录当代种群的目标函数均值
    obj_trace[gen, 1] = ObjV[best_ind] # 记录当代种群最优个体目标函数值
    var_trace[gen, :] = Chrom[best_ind, :] # 记录当代种群最优个体的染色体
# 进化完成
end_time = time.time() # 结束计时
ea.trcplot(obj_trace, [['种群个体平均目标函数值', '种群最优个体目标函数值']]) # 绘制图像
"""
=====输出结果=====
best_gen = np.argmax(obj_trace[:, [1]])
x_sum = x_A_pred_list[week_i - 1, :].sum() + x_C_pred_list[week_i - 1, :].sum()
cost_previous = C_trans * x_sum + C_store * (V_a2_previous + V_c2_previous)[best_gen]
print('调整前转运：', cost_previous)
print('调整后转运、仓储成本：', obj_trace[best_gen, 1])
cost_pre.append(cost_previous)

```

```

cost_now.append(obj_trace[best_gen, 1])

variable = ea.bs2ri(var_trace[[best_gen], :, FieldD]) # 解码得到表现型（即对应的决策变量值）
V_a_new = variable[:, : num_of_a].copy()
V_c_new = variable[:, num_of_a: ].copy()

V_tol = (V_a_new.sum() + V_c_new.sum());
V_tol_list.append(V_tol)
O_prepare = (V_a_new.sum() / 0.6 + V_c_new.sum() / 0.72);
O_prepare_list.append(O_prepare)

print('库存总量: ', V_tol)
print('原A,C总预备产能: ', (V_a2_previous / 0.6 + V_c2_previous / 0.72)[best_gen])
print('预备产能: ', O_prepare)
print('用时: ', end_time - start_time, '秒')

# update V_list
V_a_list.append(V_a_new_)
V_c_list.append(V_c_new_)
return [variable, V_a_list, V_c_list]

def plot_pie(week_i, variable, df_A_all, df_C_all):
    global pre_A, pre_C, now_A, now_C

    tol_v = [variable[:, : num_of_a].sum(), \
              variable[:, num_of_a: ].sum()]
    print(
        f' 调整前-原材料A在第{week_i}周订单量总额{x_A_pred_list.sum(axis=1)[week_i - 1]}\n调整前-原
材料C在第{week_i}周订单量总额{x_C_pred_list.sum(axis=1)[week_i - 1]}\n调整前-原材料第{week_i}周订
单量总额:{x_A_pred_list.sum(axis=1)[week_i - 1] + x_C_pred_list.sum(axis=1)[week_i - 1]}')

    print()

    print(f'原材料A在第{week_i}周订单量总额{tol_v[0]}\n 原材料C在第{week_i}周订单量总额{tol_v[1]}\n
原材料第{week_i}周订单量总额:(sum(tol_v))')

    pre_A.append(x_A_pred_list.sum(axis=1)[week_i - 1])
    pre_C.append(x_C_pred_list.sum(axis=1)[week_i - 1])

    now_A.append(tol_v[0])
    now_C.append(tol_v[1])

##### plot A #####
fig = plt.figure(dpi = 400)
explode = (0.04,) * num_of_a
labels = select_A_ID
plt.pie(variable[:, : num_of_a], explode = explode, labels = labels, autopct='%.1f%%')
plt.title(f'原材料A在第{week_i}周订单量比例')
# plt.savefig(f'./img/P2-2-Aweek{week_i}.png')
plt.show()

##### plot C #####
fig = plt.figure(dpi = 400)
explode = (0.04,) * num_of_c
labels = select_C_ID
plt.pie(variable[:, num_of_a:], explode = explode, labels = labels, autopct='%.1f%%')
plt.title(f'原材料C在第{week_i}周订单量比例')
# plt.savefig(f'./img/P2-2-Cweek{week_i}.png')
plt.show()

```

```

df_A = pd.DataFrame(dict(zip(select_A_ID, variable[0, : num_of_a])), index=[f'Week{week_i}'])
df_C = pd.DataFrame(dict(zip(select_C_ID, variable[0, num_of_a:])), index=[f'Week{week_i}'])

df_A_all = pd.concat([df_A_all, df_A])
df_C_all = pd.concat([df_C_all, df_C])

return df_A_all, df_C_all

NIND = 1000; # 种群个体数目
V_a_init = np.tile(np.array(1.88 * 1e4), NIND) / 0.6
V_c_init = np.tile(np.array(1.88 * 1e4), NIND) / 0.72
V_a2_previous_list, V_c2_previous_list = [V_a_init], [V_c_init]

for i in range(24):
    V_a2_ = np.tile(np.sum(x_A_pred_list[i, :]), NIND)
    V_c2_ = np.tile(np.sum(x_C_pred_list[i, :]), NIND)
    V_a2_previous_list.append(V_a2_)
    V_c2_previous_list.append(V_c2_)

V_a2_previous_list = np.array(V_a2_previous_list)[1:]
V_c2_previous_list = np.array(V_c2_previous_list)[1:]
df_A_all = pd.DataFrame([[[]] * len(select_A_ID)], index=select_A_ID).T
df_C_all = pd.DataFrame([[[]] * len(select_C_ID)], index=select_C_ID).T

for week_i in range(1,25):
    variable, V_a_list, V_c_list = run_algorithm(V_a_list, V_c_list, week_i)
    df_A_all, df_C_all = plot_pie(week_i, variable, df_A_all, df_C_all)

df_A_all.to_excel('P3-1A结果2.xlsx', index = False)
df_C_all.to_excel('P3-1C结果2.xlsx', index = False)
result = pd.DataFrame({'A前订单量(立方米)': np.array(pre_A),
                       'A后订单量(立方米)': np.array(now_A),
                       'C前订单量(立方米)': pre_C,
                       'C后订单量(立方米)': now_C,
                       '调整前转运、仓储成本(元)': np.array(cost_pre)*1e3,
                       '调整后转运、仓储成本(元)': np.array(cost_now)*1e3},
                      index = df_A_all.index)
result.to_excel('P3-1附加信息.xlsx')

df_all = pd.concat([df_A_all, df_C_all], axis = 1).T
df_all = df_all.sort_index()
output_df = pd.DataFrame([[[]]*df_A_all.shape[0]], index = df_A_all.index).T
x_B_pred_list = pd.read_excel('P2-2B结果.xlsx', index = False).T
x_B_pred_list.columns = output_df.columns
df_all = pd.concat([df_A_all, df_C_all], axis = 1).T
df_all = df_all.sort_index()
output_df = pd.DataFrame([[[]]*df_A_all.shape[0]], index = df_A_all.index).T
x_B_pred_list = pd.read_excel('P2-2B结果.xlsx', index = False).T
x_B_pred_list.columns = output_df.columns

for i in range(1,403):
    index = 'S'+str(i)*(3-len(str(i)))+str(i)
    if index in df_all.index:
        output_df = pd.concat([output_df,pd.DataFrame(df_all.loc[index,:]).T])
    elif index in x_B_pred_list.index:
        output_df = pd.concat([output_df, pd.DataFrame(x_B_pred_list.loc[index,:]).T])
    else:

```

```

output_df = pd.concat([output_df,
                      pd.DataFrame([[0]]*df_A_all.shape[0],
                                   index = df_A_all.index,
                                   columns = [index]).T])

output_df.to_excel('P3-1结果总和.xlsx')

```

### 问题三转运模型

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import math
from scipy.stats import exponnorm
from fitter import Fitter

#对各转运营的损失率进行预估
data=pd.DataFrame(pd.read_excel('2.xlsx'))
a=np.array(data)
a=np.delete(a, 0, axis=1)
loss = []

for j in range(8):
    f = Fitter(np.array(a[j].tolist()), distributions='exponnorm')
    f.fit()
    k=1/(f.fitted_param['exponnorm'][0]*f.fitted_param['exponnorm'][2])
    count = 0
    tem = []
    while count < 24:
        r = exponnorm.rvs(k, size=1)
        if r > 0 and r<5:
            tem.append(float(r))
            count = count+1
        else:
            pass
    loss.append(tem)

nloss=np.array(loss).T.tolist()

#求解转运方案
rdataa=pd.DataFrame(pd.read_excel('P3-1A结果2.xlsx'))
rdatab=pd.DataFrame(pd.read_excel('P2-2B.xlsx'))
rdatac=pd.DataFrame(pd.read_excel('P3-1C结果2.xlsx'))
sa = rdataa.columns.tolist()
sb = rdatab.columns.tolist()
sc = rdatac.columns.tolist()
for j in range(24):

    d = {'序号':[1,2,3,4,5,6,7,8],'损失':nloss[j]}
    tem=pd.DataFrame(d).sort_values(by='损失')
    d1 = pd.DataFrame({'商家':sa,'订单量':np.array(rdataa)[j]}).sort_values(by='订单量',ascending=[False])
    d2 = pd.DataFrame({'商家':sb[12:24],'订单量':np.array(rdatab)[j][12:24]}).sort_values(by='订单量',ascending=[False])
    d3 = pd.DataFrame({'商家':sc,'订单量':np.array(rdatac)[j]}).sort_values(by='订单量',ascending=[False])
    new = d1.append(d2)
    new = new.append(d3).reset_index(drop=True)
    count1 = 0

```

```

tran = 0
arr = []
for i in range(39):
    if new['订单量'][i] == 0:
        arr.append(0)
    else:
        if new['订单量'][i] > 6000:
            re = new['订单量'][i]-6000
            count1 = count1 +1
            tran = 0
            arr.append(temp['序号'].tolist()[count1])
            if re > 6000:
                re = re -6000
                count1 = count1 +1
                if re > 6000:
                    count1 = count1 +1
            else:
                tran = tran + new['订单量'][i]
                if tran < 5500:
                    arr.append(temp['序号'].tolist()[count1])

                elif tran > 6000:
                    count1 = count1 +1
                    tran = new['订单量'][i]

                    arr.append(temp['序号'].tolist()[count1])

                else:
                    tran = 0

                    arr.append(temp['序号'].tolist()[count1])
                    count1 = count1 +1
new['转运']=arr
if j ==0 :
    result = new.copy()
else:
    result = pd.merge(result,new,on='商家')
print(result)

```

#### 问题四订货方案模型

```

#!/usr/bin/env python
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

select_info = pd.read_excel('P2-1优化结果.xlsx')
select_A_ID = select_info[select_info['材料分类'] == 'A']['供应商ID']
select_B_ID = select_info[select_info['材料分类'] == 'B']['供应商ID']
select_C_ID = select_info[select_info['材料分类'] == 'C']['供应商ID']
select_suppliers = pd.unique(select_info['供应商ID'])

num_of_a = select_A_ID.shape[0]
num_of_b = select_B_ID.shape[0]
num_of_c = select_C_ID.shape[0]

Output_list, output_list = [], []

```

```

def aimfunc(Phen, V_a, V_b, V_c, CV, NIND):
    global V_a_list, V_b_list, V_c_list
    global V_tol_list, _prepare_list
    global x_a_pred_list, x_b_pred_list, x_c_pred_list
    global df_A_all, df_B_all, df_C_all
    global Output_list

    C_trans = 1,
    C_purchase = 100,
    C_store = 1

    def error_pred(z, type_):
        a = np.array([gp_dict[type_][i].predict(z[:, [i]], return_std=True) for i in
range(z.shape[1])])
        err_pred = a[:, 0, :]
        err_std = a[:, 1, :]
        np.random.seed(1)
        nor_err = np.random.normal(err_pred, err_std)
        err = np.transpose(nor_err) * 1e-2
        err *= z
        return err

    def convert(x, type_):
        if type_ == 'a': x_ = x * 1.2
        if type_ == 'b': x_ = x * 1.1
        if type_ == 'c': x_ = x

        return x_

    z_a = Phen[:, 0:num_of_a]
    z_b = Phen[:, num_of_a: num_of_a + num_of_b]
    z_c = Phen[:, num_of_a + num_of_b:]

    errorA = error_pred(z_a, 'A')
    errorB = error_pred(z_b, 'B')
    errorC = error_pred(z_c, 'C')

    x_a0, x_b0, x_c0 = z_a + errorA, z_b + errorB, z_c + errorC
    x = np.hstack([x_a0, x_b0, x_c0])
    x_a = convert(x_a0, 'a')
    x_b = convert(x_b0, 'b')
    x_c = convert(x_c0, 'c')

    # constraint 5
    e_a = max(np.ones_like(V_a)*0.9, 2.82*1e4*V_a/(V_a * 0.6 + V_b * 0.66 + V_c * 0.72))
    e_b = max(np.ones_like(V_b)*0.9, 2.82*1e4*V_b/(V_a * 0.6 + V_b * 0.66 + V_c * 0.72))
    e_c = max(np.ones_like(V_c)*0.9, 2.82*1e4*V_c/(V_a * 0.6 + V_b * 0.66 + V_c * 0.72))

    # constraint 2 ~ 4
    V_a2 = V_a * (1 - e_a) + x_a.sum(axis=1)
    V_b2 = V_b * (1 - e_b) + x_b.sum(axis=1)
    V_c2 = V_c * (1 - e_c) + x_c.sum(axis=1)

    Output = (V_a * e_a * 0.6) + (V_b * e_b * 0.66) + (V_c * e_c * 0.72)
    Output_list.append(Output)

    f = C_trans * x.sum(axis=1) + C_purchase * (x_a.sum(axis=1) + x_b.sum(axis=1) +
x_c.sum(axis=1))
    f = f.reshape(-1, 1)

```

```

# constraint 1
CV1 = (2.82 * 1e4 * 2 - x_a.sum(axis=1) * (1 - alpha[week_i - 1]) / 0.6 - x_b.sum(axis=1) *
(
    1 - alpha[week_i - 1]) / 0.66 - x_c.sum(axis=1) * (1 - alpha[week_i - 1]) /
0.72)
CV1 = CV1.reshape(-1, 1)

CV2 = (z_a.sum(axis=1) + z_b.sum(axis=1) + z_c.sum(axis=1) - 4.8 * 1e4).reshape(-1, 1)
CV1 = np.hstack([CV1, CV2])
return [f, CV1, V_a2, V_b2, V_c2, x_a0, x_b0, x_c0]

def run_algorithm2(week_i):
    global x_a_pred_list, x_b_pred_list, x_c_pred_list

    x_a_pred_list, x_b_pred_list, x_c_pred_list = x_A_pred_list[week_i - 1, :],
    x_B_pred_list[week_i - 1,
                  :],
    x_C_pred_list[week_i - 1, :]

    """=====
    变量设置
    ====="""
    tol_num_A = 8 * num_of_a
    tol_num_B = 8 * num_of_b
    tol_num_C = 8 * num_of_c
    tol_num2 = tol_num_A + tol_num_B + tol_num_C
    z_a = [[0, 1]] * tol_num_A
    z_b = [[0, 1]] * tol_num_B
    z_c = [[0, 1]] * tol_num_C

    B = [[1, 1]] * tol_num2 # 第一个决策变量边界，1表示包含范围的边界，0表示不包含
    D = [0, ] * tol_num2
    ranges = np.vstack([z_a, z_b, z_c]).T # 生成自变量的范围矩阵，使得第一行为所有决策变量的下界，第二行为上界
    borders = np.vstack(B).T # 生成自变量的边界矩阵
    varTypes = np.array(D) # 决策变量的类型，0表示连续，1表示离散
    """=====
    染色体编码设置
    ====="""
    Encoding = 'BG' # 'BG'表示采用二进制/格雷编码
    codes = [0, ] * tol_num2 # 决策变量的编码方式，设置两个0表示两个决策变量均使用二进制编码
    precisions = [4, ] * tol_num2 # 决策变量的编码精度，表示二进制编码串解码后能表示的决策变量的精度可达到小数点后6位
    scales = [0, ] * tol_num2 # 0表示采用算术刻度，1表示采用对数刻度
    FieldD = ea.crtfld(Encoding, varTypes, ranges, borders, precisions, codes, scales) # 调用函数创建译码矩阵
    """=====
    遗传算法参数设置
    ====="""
    NIND = 1000; # 种群个体数目
    MAXGEN = 300; # 最大遗传代数
    maxormins = [1] # 列表元素为1则表示对应的目标函数是最小化，元素为-1则表示对应的目标函数是最大化
    maxormins = np.array(maxormins) # 转化为Numpy array行向量
    selectStyle = 'rws' # 采用轮盘赌选择
    recStyle = 'xovdp' # 采用两点交叉
    mutStyle = 'mutbin' # 采用二进制染色体的变异算子
    Lind = int(np.sum(FieldD[0, :])) # 计算染色体长度
    pc = 0.8 # 交叉概率
    pm = 1 / Lind # 变异概率
    obj_trace = np.zeros((MAXGEN, 2)) # 定义目标函数值记录器
    var_trace = np.zeros((MAXGEN, Lind)) # 染色体记录器，记录历代最优个体的染色体
    """=====
    开始遗传算法进化
    ====="""
    start_time = time.time() # 开始计时
    Chrom = ea.crtpc(Encoding, NIND, FieldD) # 生成种群染色体矩阵

```

```

variable = ea.bs2ri(Chrom, FieldD) # 对初始种群进行解码
# CV1 = np.zeros((NIND, 1)) # 初始化一个CV矩阵（此时因为未确定个体是否满足约束条件，因此初始化元素为0。暂认为所有个体是可行解个体）
CV = np.zeros((NIND, 615))
ObjV, CV = aimfunc2(variable, CV, NIND)
FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
# 开始进化
for gen in tqdm(range(MAXGEN), leave=False):
    SelCh = Chrom[ea.selecting(selectStyle, FitnV, NIND - 1), :] # 选择
    SelCh = ea.recombin(recStyle, SelCh, pc) # 重组
    SelCh = ea.mutate(mutStyle, Encoding, SelCh, pm) # 变异
    # 把父代精英个体与子代的染色体进行合并，得到新一代种群
    Chrom = np.vstack([Chrom[best_ind, :], SelCh])
    Phen = ea.bs2ri(Chrom, FieldD) # 对种群进行解码(二进制转十进制)
    ObjV, CV = aimfunc2(Phen, CV, NIND) # 求种群个体的目标函数值
    FitnV = ea.ranking(ObjV, CV, maxormins) # 根据目标函数大小分配适应度值
    # 记录
    best_ind = np.argmax(FitnV) # 计算当代最优个体的序号
    obj_trace[gen, 0] = np.sum(ObjV) / ObjV.shape[0] # 记录当代种群的目标函数均值
    obj_trace[gen, 1] = ObjV[best_ind] # 记录当代种群最优个体目标函数值
    var_trace[gen, :] = Chrom[best_ind, :] # 记录当代种群最优个体的染色体
# 进化完成
end_time = time.time() # 结束时间
ea.trcpplot(obj_trace, [['种群个体平均目标函数值', '种群最优个体目标函数值']]) # 绘制图像
print('用时：', end_time - start_time, '秒')

"""
-----输出结果-----
best_gen = np.argmax(obj_trace[:, [1]])
print('最优解的目标函数值：', obj_trace[best_gen, 1])
variable = ea.bs2ri(var_trace[[best_gen], :], FieldD) # 解码得到表现型（即对应的决策变量值）

W = variable
W_a, W_b, W_c = W[:, :tol_num_A], W[:, tol_num_A:tol_num_A + tol_num_B], W[:, tol_num_A + tol_num_B:]
W_a = W_a.reshape((f_transA.shape[1], 8))
W_b = W_b.reshape((f_transB.shape[1], 8))
W_c = W_c.reshape((f_transC.shape[1], 8))

assign_A = dict(zip(select_A_ID, [np.argmax(W_a[i, :]) + 1 for i in range(f_transA.shape[1])]))
assign_B = dict(zip(select_B_ID, [np.argmax(W_b[i, :]) + 1 for i in range(f_transB.shape[1])]))
assign_C = dict(zip(select_C_ID, [np.argmax(W_c[i, :]) + 1 for i in range(f_transC.shape[1])]))
print('原材料A供应商对应的转运商ID:\n', assign_A)
print('原材料B供应商对应的转运商ID:\n', assign_B)
print('原材料C供应商对应的转运商ID:\n', assign_C)

return [assign_A, assign_B, assign_C]

pd.DataFrame(np.array(output_list)*1.4, index = [f'Week{i}' for i in range(1, 25)], columns = ['产能']).plot.bar(alpha = 0.7, figsize = (12,5))
plt.axhline(2.82*1e4, linestyle='dashed', c = 'black', label = '期望产能')
plt.title('扩大产能后:预期24周产能')
plt.legend()
plt.ylabel('立方米')
plt.show()

output_df.to_excel('P4-2分配结果.xlsx')

```

#### 问题四转运模型

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import math
from scipy.stats import exponnorm
from fitter import Fitter

#对各转运商的损失率进行预估
data=pd.DataFrame(pd.read_excel('2.xlsx'))
a=np.array(data)
a=np.delete(a, 0, axis=1)
loss = []

for j in range(8):
    f = Fitter(np.array(a[j].tolist()),distributions='exponnorm')
    f.fit()
    k=1/(f.fitted_param['exponnorm'][0]*f.fitted_param['exponnorm'][2])
    count = 0
    tem = []
    while count < 24:
        r = exponnorm.rvs(k, size=1)
        if r > 0 and r<5:
            tem.append(float(r))
            count = count+1
        else:
            pass
    loss.append(tem)

nloss=np.array(loss).T.tolist()

#求解转运方案
rdata=pd.DataFrame(pd.read_excel('/Users/yongjuhao/Desktop/第四问结果1.xlsx'))
sa = rdata.columns.tolist()
for j in range(24):
    d = {'序号':[1,2,3,4,5,6,7,8],'损失':nloss[j]}
    tem=pd.DataFrame(d).sort_values(by='损失')
    d1 = pd.DataFrame({'商家':sa[0:136],'订单量':np.array(rdata)[j][0:136]}).sort_values(by='订单量',ascending = [False])
    d2 = pd.DataFrame({'商家':sa[136:262],'订单量':np.array(rdata)[j][136:262]}).sort_values(by='订单量',ascending = [False])
    d3 = pd.DataFrame({'商家':sa[262:369],'订单量':np.array(rdata)[j][262:369]}).sort_values(by='订单量',ascending = [False])
    new = d1.append(d2)
    new = new.append(d3).reset_index(drop=True)
    count1 = 0
    tran = 0
    arr = []
    for i in range(369):
        if new['订单量'][i] == 0:
            arr.append(0)
        else:
            if new['订单量'][i] > 6000:
                re = new['订单量'][i]-6000
                count1 = count1 + 1
                tran = 0
                arr.append(tem['序号'].tolist()[count1])
                if re >6000:
                    re = re -6000
            else:
```

```
        count1 = count1 +1
        if re >6000:
            count1 = count1 +1
    else:
        tran = tran + new['订单量'][i]
        if tran < 5500:
            if count1 +1 >= 8:
                arr.append(temp['序号'].tolist()[?])
            else:
                arr.append(temp['序号'].tolist()[count1])
        elif tran > 6000:
            count1 = count1 +1
            tran = new['订单量'][i]
            if count1 +1 >= 8:
                arr.append(temp['序号'].tolist()[?])
            else:
                arr.append(temp['序号'].tolist()[count1])
        else:
            tran = 0
            if count1 +1 >= 8:
                arr.append(temp['序号'].tolist()[?])
            else:
                arr.append(temp['序号'].tolist()[count1])
            count1 = count1 +1
new['转运']=arr
if j ==0 :
    result = new.copy()
else:
    result = pd.merge(result,new,on='商家')
```