基于多目标粒子算法的企业原材料订购与运输

摘　要

生产企业与供应商的合作一直伴随着企业的日常生产，传统供应商关系存在供应商数量多而缺乏长期紧密合作、供应批量小、合作性质单一等弊端，从而影响企业日常生产。本文依据四个问题进行求解，旨在助企业转变为供应链合作伙伴型供应商关系，选择稳定供货、供应批量大、质量可控、少而精的供应商。

对于问题一：在量化分析供应商的供货特征时，从 5 年来每周的详细数据以及每个周期(24 周)的整体数据这两个方面进行分析。从各供应商的描述性统计结果来看，供应商间的供应规模存在较大差异，大部分供应商的不满足率高于超额率。本文将供应商重要性的一级评价指标定为：企业合作、供给稳定性、供货能力。再结合附件一中的数据将指标细分为 8 个二级指标，使用**熵权法**的 **Topsis** 法对供应商的重要性进行评分，8 个指标和权重如下表所示：

指标名称（周期）	最大值	最小值	平均金额	方差	增长量	频数	超额率	不满足率
权重	0.295	0.319	0.295	0.0004	0.0009	0.064	0.0063	0.017

对于问题二：检验 402 家供应商的整体平均偏差，得到近似服从**高斯分布**，因此供应商每次实际供应量受偏差率因子 p 的影响，取一个标准差的偏差作为供应量的最大偏差，为选择最少的供应商，如果所选供应商都为最低供应时，仍能满足企业的周产能，则取重要性得分最高的一组中的供应商作为所选供应商：

最少供应商数量(个)	最低提供产能总和(m^3)	重要性得分总和
18	28252.0	0.169775

以每周采购成本最小为目标函数，以至少满足一周生产的产能和每个供货商供货量小于 6000 为约束条件建立单目标规划模型，使用**惩罚函数法的模拟退火算法**对最经济的采购方案进行求解。为转运损失最小，以每个供应商运输量不多于 6000，每个供应商只能选择一个转运商为条件建立转运优化模型，使用**蒙特卡洛**算法模拟 100 万次求解出最小损失量下转运方案。

对于问题三：以每周供应商提供的产能与企业产能差值的最小值、CA 比例最小值为目标函数，建立多目标规划模型，以供货量大于周产能和每个供应商供货量小于 6000 为约束条件，使用**多目标粒子群算法**进行求解，得到 24 周的订购计划，并使用问题二中的转运优化模型对最优方案进行求解。前 6 周的多余产能和 C 与 A 比例结果如下：

周数	1	2	3	4	5	6
多余产能	84.39	276.92	390.03	0.14	0.87	5.59
比例(C/A)	0.0674	0.0411	0.0279	0	0	0

对于问题四：企业每周的产能受到供应量的限制，本文以供应商最大供应量与转运商最大转运量二者中的最小值作为企业一周收到的最大原材料量。当企业数达 25 家时，转运商转运量到达上限。以采购金额最小为目标函数，以供应原料的产能大于企业产能和企业产能为仓储量的平方为约束条件，每次计算都将约束条件中的产能定为仓储的一半，建立动态单目标规划模型，计算出每周的进货金额和产能，使用 Matlab 对数据点进行**多项式拟合**，最终最大产能维持在 44892m^3。

关键词：　TOPSIS　模拟退火算法　蒙特卡罗模拟　多目标规划　粒子群算法

一、 问题重述

1.1 问题背景

对于需要原材料的企业来说，选择实力强、信誉良好的合作伙伴以及合理的原材料采购和转运方案对企业的正常生产有着重要意义，本文将结合实际情况建立数学模型解决企业对供应商的选择问题以及制定合理的原材料采购和转运方案。

1.2 问题一的重述

依据该企业近 5 年原材料供应商的订货量和供货量数据，提取并量化分析供货商的供货特征，进而建立数学模型来反映保障企业生产重要性的因子，通过对比得到 50 家最重要的供应商。

1.3 问题二的重述

在问题一的基础之上，为了满足企业正常生产的需要，求解出至少选择多少家供应商;在未来的 24 周里制定每周的最优的原材料订购方案以及损耗最少的转运方案，最后对于分析实施效果。

1.4 问题三的重述

为了实现生产成本最优化以及转运商的损耗率最少，在转运和仓储中，A 类原材料比 C 类原材料更经济，所以尽量多地采购 A 类，从而需要制定新的原材料订购方案和转运方案，并对方案的实施效果进行分析。

1.5 问题四的重述

依据现有的供应商和转运商的实际情况，在保证正常生产的情况下，需确定该企业每周的产能最高可提高多少；以及新的产能之下，确定未来 24 周的订购和转运方案。

二、 问题分析

2.1 问题一的分析

对题意进行分析后，可将问题一分为两个小问。

第一小问：对 402 家供应商的供货特征进行量化的分析，粗略观察附件一中的数据，多数供应商的供应量的波动较大，猜测与企业是否订购该家供应商的原料有关，企业以一个订购周期(24 周)订购供应商的原材料，因此本文先从 5 年的整体数据对 402 家供应商的供货特征进行分析，再从每个订购周期的局部数据对 402 家供应商的供货特征进行分析。结合实际生活，考虑到供应商供应量没有达到订购量与超额完成订购量对企业带来的影响不同，因此将偏差率分为不满足率和超额率分别进行计算。

第二小问：这是一个评价类的问题，首先确定供应商的评价指标，在查阅大量文献，以及结合实际生活后，本文将反应供应商重要性的一级评价指标定为企业合作、供给稳定性和供货能力，再进一步的将三个一级指标细分为 8 个二级指标，在 Excel 中计算出各指标的数据，使用熵权法改进的 Topsis 对各个供应商的重要性进行

打分，将打分结果在按降序排序，找出前 50 家供应商的详细数据与其他的供应商进行对比。

2.2 问题二的分析

对题意进行分析后，可将问题二分为三小问。

第一小问：分析后得知，这是一个 0-1 规划的问题，企业至少要选择多少家供应商才能满足正常的生产需求。企业库存至少应保持在满足两周产能的量，所有当所选择的供应商都提供最小供应量时应正好满足一周的产量，这样才能保证企业能够维持正常的生产和补充库存。供应商的实际供应量通常与订单量存在偏差，将 402 家企业 5 年的平均偏差进行统计，发现结果近似满足高斯分布，将各供应商的偏差率在对应的高斯分布中随机取值。以偏差一个标准差的范围定义供应商的最大最小供应量，使用 Excel 数据透视表计算出满足条件的供应商的组合结果，在结果中挑选出供应商数量最少的结果，在这些结果中选择重要性分数之和最高的一组作为挑选的对象。

第二小问：分析后得知，这是一个单目标的规划问题，针对上一小问得出的供应商，制定 24 周的订购计划，使得每周订购所花的成本最少，但同时也要满足企业生产的最低需求，因此可以以每周采购所需的成本为目标函数，以至少满足一周生产的产能和每个供货商供货量小于 6000 为约束条件，建立单目标规划模型，使用惩罚函数法的模拟退火算法对模型进行求解。

第三小问：分析后得知，这是一个非线规划的问题，一个供应商只能在 8 家转运商中选择一家对原料进行转运，而转运商的每周转运量也不能超过 6000。为简化模型，本文将转运商的耗损率看作服从高斯分布，每次计算时都从高斯分布中选取损失率。以损失量最小为目标函数，以每家转运商转运量小于 6000，每个供应商只能选择一家转运商为约束条件，建立转运优化模型，使用蒙特卡洛算法对最优的转运方案进行求解。

2.3 问题三的分析

对题意进行分析，可将问题三分为两个小问。

第一小问：分析后得知，这是一个多目标的规划问题。题目要求在压缩仓储成本的同时增大 A 的采购而减小 C 的采购。压缩仓储成本，也就是尽量使每次采购的原材料都尽可能地消耗完，也即是每周采购原料所能提供的产能尽量接近企业的周产能。因此将采购原料所能提供的产能与企业每周产能之差最小作为第一个目标函数，将 C 比 A 的比值最小作为第二个目标函数。以所提供的产能大于企业每周产能和每个供应商的供应量不超过 6000 作为约束条件，建立多目标规划模型，使用多目标粒子群算法对模型进行求解，得出每周的订购方案。

第二小问：将得出的订购方案的数据代入问题二中的转运规划模型对 24 周的转运方案进行求解。做出变化曲线分析方案的实施效果。

2.4 问题四的分析

对题意进行分析，企业的最大产能主要受到每周供应量大小的影响，因此首先用各个供应商每周供应的最大值计算出转运商至多能接受多少家供应商的转运要求。为确保企业的正常生产，再利用供应商每周供应的最小值计算出企业产能的极限，当所有企业都提供最小值时，如果企业接收到的原材料依然能够满足一周的生产量，那么就认为此时的状态是最安全的。以每周的采购金额最小目标函数，各供应商供应量小于其最大值，大于其最小值，实际供应的原料所带来的产能大于本周的产能，为三个约束条件建立单目标规划模型，使用模拟退火算法进行求解。在求解下一次的订购量时首先对产能进行更新，将约束条件中的产能更新为上周仓库剩余量的一半。模拟 24

周的订购方案即可。在使用问题二中的转运优化模型对转运方案进行求解。

三、 模型假设

1. 假设每个供应商都会尽力完成订单的要求。
2. 假设供应商所提供的原料均为可以正常投入生产。
3. 假设供应商和转运商均不会出现突发事故导致停业。
4. 假设转运商都能每周按时将原材料送至企业。
5. 假设各转运商的损耗率服从高斯分布。

四、 符号说明

符号	说明
v_a	超额率
v_b	不满足率
z	增长量
P	偏差率
G	供应量
D	订购量
p	偏差因子
O	企业周产能
E	采购成本
K	库存量

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 供货特征的量化分析

首先分析题意，题目要求根据附件一中的信息对 402 家供应商的供货特征进行定量的分析，附件一中的内容为企业 5 年来每周对供应商的订货量，以及 5 年来 402 家供应商每周为企业提供的原材料量。在分析完题干后得知，此题为数据挖掘以及数据预处理的问题，只需找出能够反映供应商供货特征的指标，并计算其具体数值即可对特征进行量化分析。为了能更好的反应供应商供货的特征，本文从 5 年整体数据，以及每一次订购计划（半年）两个方面入手，分别对供应商的供货特征进行分析。

(1) 五年整体数据处理

首先将 402 家供应商的供应数据按 5 年整体进行初步的描述性统计。将附件一中的 402 家供应商数据导入 SPSS 工具，按时间与供应商 ID 两个变量对数据集进行转置，对 402 家供应商的供应数据进行描述性统计，供应商 ID S1~S10 的结果如下结果（402 家供应商结果详见附录 2）：

表 1 整体描述性统计表

ID	最小值	最大值	总计供应	平均值	标准差	峰度
S001	0	43.00	231.00	0.9625	4.5787	72.791
S002	0	60.00	309.00	1.2875	5.7642	89.917

S003	0	440.00	14279.00	59.4958	87.8094	4.084
S004	0	100.00	713.00	2.9708	15.2179	36.039
S005	0	140.00	6538.00	27.2417	34.9510	-0.916
S006	0	100.00	462.00	1.9250	12.8076	55.971
S007	8	200.00	6716.00	27.9833	36.3036	8.033
S008	0	10.00	94.00	0.3917	1.2495	43.379
S009	0	320.00	715.00	2.9792	28.2271	117.825
S010	0	100.00	576.00	2.4000	10.2715	47.549

根据表一中的数据，可对这 S1~S10 的供应商 5 年整体供应特征进行定量分析，只有 S07 供应商在 5 年中供应的最小值不为 0 其余均为 0，说明 S07 在 5 年中一直保持着原材料的供应，最大值间接能反映出供应商的规模大小，S03 最大值为这 10 家供应商中最大的，说明它能在单周内提供的原材料最多。S03 在 5 年内总计给企业提供了 14279 立方米的原材料，平均值为 59.48，反应出 S03 的供货规模最大。

(2) 每半年（1 个订购周期）的数据处理

题干中，企业每次提前制定 24 周的原材料订购和转运计划，因此本文以 24 周为一个订购周期，因此依照每个订购周期对供应商的供应特征进行分析，更能反应出供应商的真实供应特征。在查阅相关资料后，本文确立了供应商供货特征的指标，分别为，周期总值最大值，周期总值最小值，每周期平均供应量，周期方差，每周期增长量平均值，总供货频率，每周期偏差平均值，超额完成率和不满足率周期。

方差计算公式如下所示:

$$S^2 = \frac{\sum_{i=1}^{n}\left(\bar{x} - x_i\right)^2}{n} \tag{1}$$

公式 1 中 x 为每周期内供应量的总和

由于供应商超额完成与未完成对企业的影响大小不同，未完成的订单对企业生产的影响较大，因此本文对两个数分别进行计算，超额率v_a以及不满足率v_b计算公式如下所示：

$$v_a = \begin{cases} \dfrac{G-D}{D}\,(G>D) \\ 0\ \ (G<D) \end{cases} \tag{2}$$

$$v_b = \begin{cases} \dfrac{D-G}{D}\,(D>G) \\ 0\ \ (D<G) \end{cases} \tag{3}$$

公式 2,3 中 G 对周期内供应商实际供应量之和，D 为周期内企业向供应商订购的原材料量之和。

使用 Excel 的数据透视表工具求出以上指标的具体数值，S1~S10 所得指标结果如下表所示（402 家供应商结果详见附录 3）：

表 2 周期描述性统计表

ID	最大值	最小值	平均值	方差	增长量	频率	超额率	不满足率

S001	9	1	4.9	9.29	0.22	25	0.091	0.676
S002	71	1	27.3	370.01	-3.22	71	0.218	0.260
S003	2371	189	1313.8	720511.56	128.11	191	0.055	0.155
S004	23	1	6.4	44.64	-0.44	33	0.055	0.720
S005	1913	29	691.2	479155.76	94.77	107	0.096	0.110
S006	7	1	3	5.8	-0.22	13	0.096	0.718
S007	754	582	694.8	1894.16	16.11	240	0.151	0.086
S008	18	1	4.1	34.49	-0.22	15	0.051	0.671
S009	6	1	3.1	5.89	-0.55	19	0.019	0.839
S010	64	8	17	404.8	-2.22	32	0.030	0.666

对表 2 的结果进行分析，首先是对最大值的分析，S03 的周期最大值是最大的，为 2371，可以反映出 S03 的供应规模相较其他的大。最小值为 S07 最大，说明企业与 S07 一直保持着密切的商业关系。平均值依然为 S03 最大，而 S03 的方差相较其他的供应商有明显的变化，说明虽然 S03 的供应规模较大，但与企业间的合作波动也最大，而 S07 一直保持在一个中等的区间。S03 和 S07 的供应频率领先其他供应商较多，占据这十家供应商总频率的 57.7%，说明这两家供应商与企业的合作最为密切。超额率 S02 最大，表示 S02 在供应量大于订购量时，平均每次都供应超 21.8%。S03 与 S07 供应的不满足率分别为 0.155 和 0.086，远小于其他供应商，说明这两家企业每次都会尽可能的满足企业订购的需求量。

为更加直观的看出各企业间的供应特征差异，根据各企业的超额率与满足率数据可做出下图：

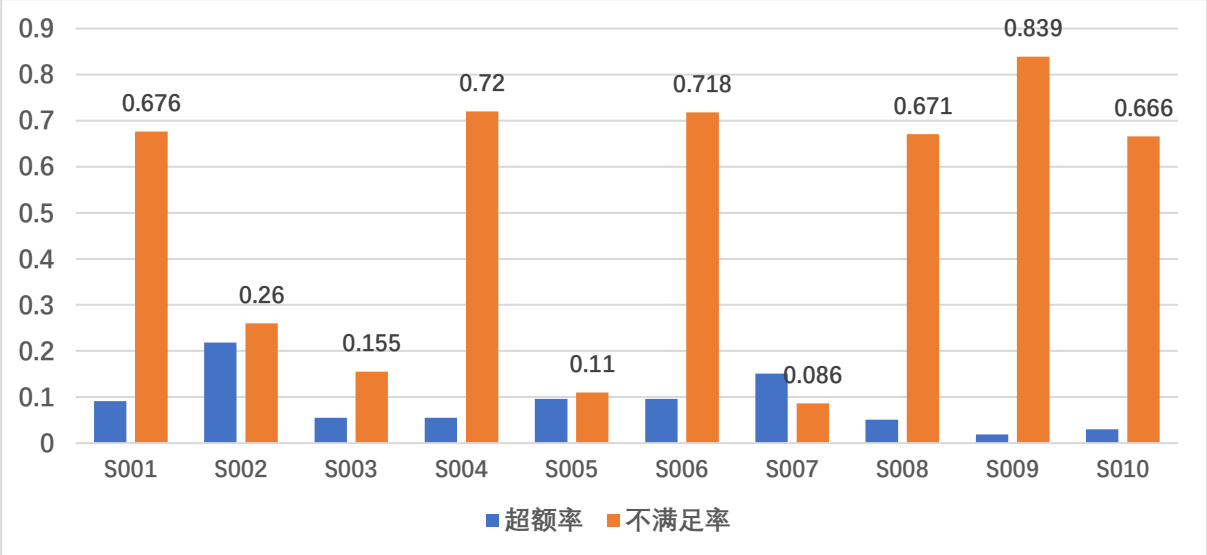

图 1 企业失约情况图

图 1 中蓝色线条为供应商超额率，红色为不满足率。从图 1 中可以直观地看出，S003 和 S007 的失约率要远小于其他各个供应商，这也与上面分析的结果相一致。图中 90%供应商的不满足率都大于超额率，说明大部分供应商都难以保持每次供应量都满足企业订购的需求。供应商提供的原材料量不能满足企业的需求，对企业来说无疑会影响产品的生产，进而导致企业的损失。因此可以通过观察企业与这些供应商的交易频数来判断企业对这些供应商的态度。取非零的订单数作为企业与供应商交易数，企业与供应商的交易频数图如下所示：

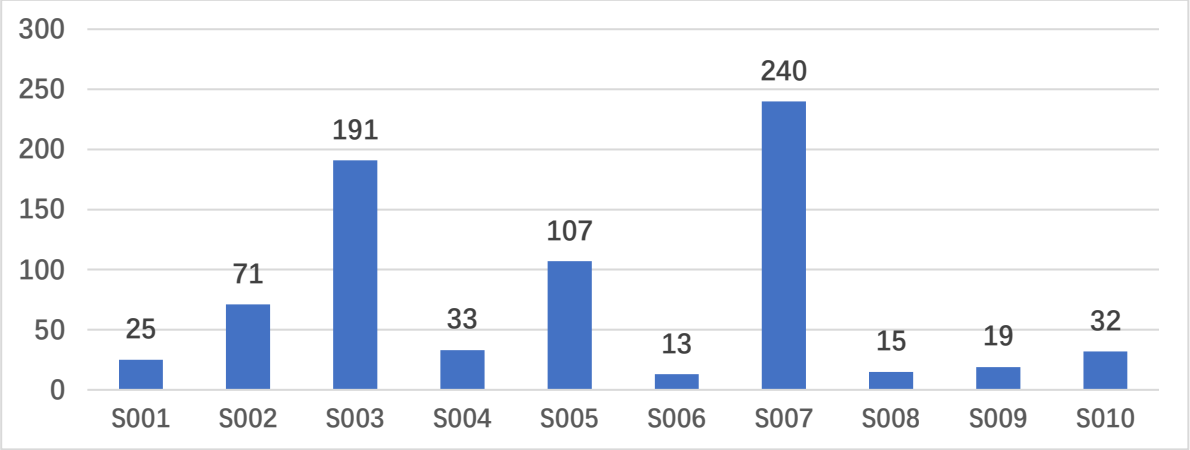

图 2 交易频数图

从图 2 中可以直观的看出，企业与 S03 和 S07 供应商的交易频数较多占据总交易频数的 57.7%，说明企业对这两个供应商更加信赖。这也与企业的失约情况正好相反，供应商的失约率越高，企业与之交易的次数也就越少，这也与上面推出的结果相一致。这些指标的结果与保障企业的生产有着重要意义，接下来本文将建立供应商重要性模型，对每个供应商的重要性进行打分。

5.1.2 评价指标的确定

由上面的量化分析初步判断，企业在供应商的订货量与供应商的规模有关，企业与供应商的交易频数与企业的失信率负相关。首先依照现实情况分析供应商对企业的重要性，供应商的失信行为无疑会对供应商的信誉与企业的生产产生影响。与企业交流密切，供给量大的供应商对企业的影响更大，因此选择实力强、信誉高的供应商作为合作伙伴对保障企业的生产起着重要作用。

为了对企业的实力和信誉进行打分，首先需要找到与之相关的评价指标，在查阅大量相关文献后，本文选择了三篇与问题相关度较高的文献并对他们的评价指标进行总结，依照范琛[1]针对供应链管理中的供应商选择的研究一文中，以准时交货、原料价格、运输成本、服务质量四个指标作为选择供应商的评价指标。在雷星晖[2]的层次分析法对供应商绩效评价一文中，以合作、服务、准时性、成本为四个一级指标对供应商进行评价。在黄瀚[3]的精益生产方式下供应商模糊综合评价一文中，以供应商的质量水平、管理能力、供货能力、竞争力、协助能力这五个指标用于评价供应商的价值。对所有供应商的评价指标进行总结后，可初步得到评价供应商重要性的一级指标，指标如下图所示：

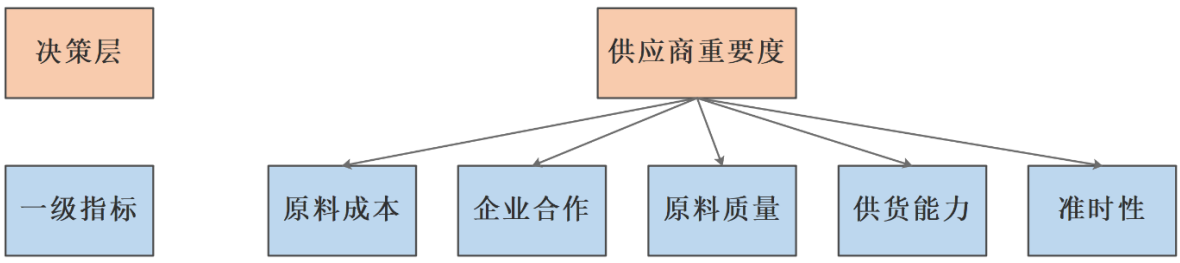

图 3 一级指标分析图

图 3 中，经过初步筛选得到的一级指标为：原料成本、企业合作、供给稳定性、供货能力、准时性。共得到五个一级指标，将这些指标与本题所给出的供应商数据相

结合，进而剔除掉一些与本题无关的指标，同时推得能用数据量化表示的二级指标来反映供应商的重要程度，二级评价指标如下图所示：

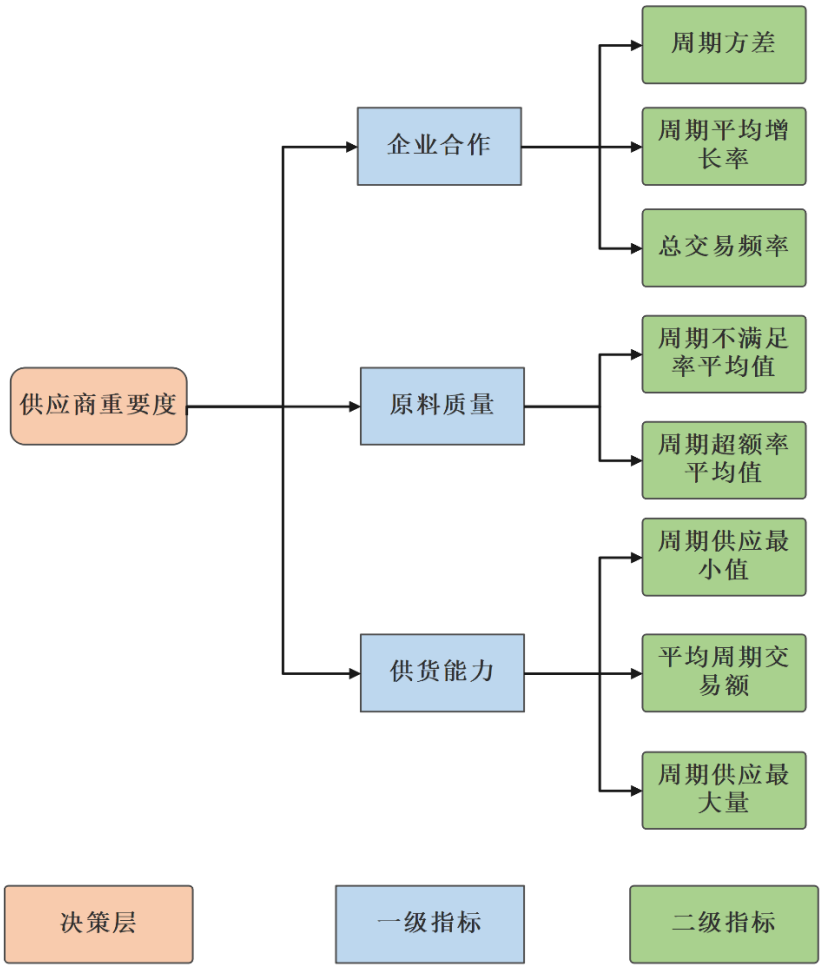

图 4 指标分析图

在图 4 中得到各指标间的关系，本文将实际情况与问题相结合，在数据中挖掘三大方面，八大指标来评价对于该企业，每家供应商的重要性。

(1) 供应商与企业间的合作情况

为了反应供应商与企业间的合作情况得到以下指标

每周期的方差:

$$S^2 = \frac{\sum_{i=1}^{24}\left(\bar{x} - x_i\right)^2}{24} \tag{4}$$

平均增长量:

$$\bar{z} = \frac{\sum_{i}^{n} z_{i+1} - z_i}{n} \tag{5}$$

(2) 原材料供给的稳定性

以周期不满足率的平均值和周期超额完成率的平均值来反映供应商提供原料与

企业订购数量的偏差，进而体现出原材料供给的稳定性。

每周期不满足率的平均值

$$(\text{当}D\text{大于}G\text{时})\ \overline{v_b}=\frac{\sum_{i=1}^{n}(D_{\mathrm{i}}-G_{\mathrm{i}})}{\mathrm{n}} \tag{6}$$

每周期超额完成率的平均值

$$(\text{当}G\text{大于}D\text{时})\ \overline{v_a}=\frac{\sum_{i=1}^{n}(G_{\mathrm{i}}-D_{\mathrm{i}})}{\mathrm{n}} \tag{7}$$

(3)供应商的供应规模和供货能力

以周期供应的最小值，平均每周期交易额和周期供应量的最大值反映一家供应商的供应规模和供货能力。

每周期平均订单金额：

$$W=\frac{\sum_{i=1}^{n}G_i q_x}{n} \tag{8}$$

$$\text{其中}q_a=1.2,\ q_b=1.1,\ q_c=1$$

公式 4 中,q 为各个类型的原材料相较 c 类型原材料的价格，根据各个供应商供应的原材料类型的不同，q 的大小也相应改变。

每周期供应量的最大值：

$$MAX\{x_1,x_2,.....,x_{10}\} \tag{9}$$

每周期供应的最小值

$$MIN\{x_1,x_2,.....,x_{10}\} \tag{10}$$

5.1.3 供应商重要性模型的建立

(1)Topsis 评价模型的建立

在数据预处理完成后，所有评价指标(共八个)都已经被量化表示，由于数据量庞大且决策因素较多的缘故，其他评价方法无法客观且量化的反映出各供应商对企业的重要程度，因此本文采用 Topsis 法建立重要性评价模型对所有供应商对企业的重要程度进行打分。Topsis 法的简略流程图如下图所示：

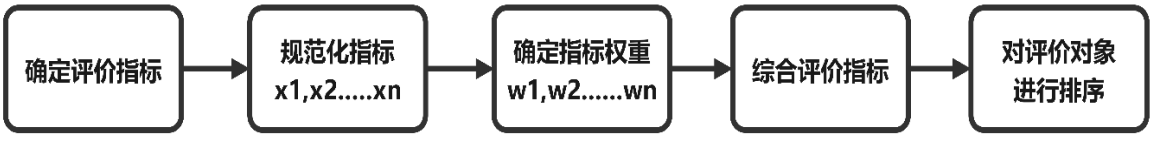

图 5 Topsis 流程图

第一步：建立评价矩阵

现有 402 家待评价的供应商，共 8 各评价指标，因此可建立如下的评价矩阵 X：

$$X=\begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{18} \\ x_{21} & x_{22} & \cdots & \cdots & x_{28} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & \cdots & x_{n8} \end{bmatrix} \tag{11}$$

矩阵共有 402 行，8 列，402 行代表 402 家供应商，8 列分别为 8 个评价指标。

第二步：正向化处理

首先分析指标的类型，对于供应商供货能力的指标中，周期供应最大值，周期最小值，平均周期交易金额，反应供应商的供应能力以及供应规模，因此都为为极大型。对于供应商供给稳定性的指标中，平均周期不满足率，平均周期超额完成率都反应了供应商供货与企业订购间的差异，因此二者都为极小型。对于企业与供应商合作的指标中，周期平均增长量，总交易频数都反应出企业与供应商间的合作密切程度，因此为极大型，周期供应量方差反应合作的稳定性因此为极小型。根据上述分析可做出下表：

表 3 指标类型表

一级指标	企业合作			供给稳定性		供货能力		
二级指标	方差	增长量	总频数	不满足率	超额率	最小值	最大值	交易额
指标类型	极小	极大	极大	极小	极小	极大	极大	极大

表 3 中，共 8 个二级评价指标，其中有 5 个已经为极大型指标，只需将剩下的三个极小型的指标转化为极大型的指标即可完成指标正向化处理，转化公式为 $x_i=x_{\max}-x_i$ ，其中 x 表示极小值的数值，$x_{\max}$ 为该列中极小值的最大值。

第三步：矩阵标准化

对已经正向化处理后的矩阵进行标准化处理，目的是为了消除指标间不同量纲对评价结果所产生的影响。将标准化后的矩阵记为 Z，那么对于 Z 中的每一个元素 z 都有如下公式：

$$z_{ij}=\frac{x_{ij}}{\sqrt{\sum_{i=1}^{n} x_{ij}^2}} \tag{12}$$

通过公式 6 可以求得 Z 中的每一个元素，进而可以得到标准化后的矩阵 Z：

$$Z=\begin{bmatrix} z_{11} & z_{12} & \cdots & z_{18} \\ z_{21} & z_{22} & \cdots & z_{28} \\ \cdots & \cdots & \cdots & \cdots \\ z_{n1} & z_{n2} & \cdots & z_{n8} \end{bmatrix}$$

第四步：熵权法确定指标权重

首先计算概率矩阵，这里记为 P，对于 P 中的每一个元素 p 都有如下公式：

$$p_{ij}=\frac{\widetilde{z}_{ij}}{\sum_{i=1}^{n} \widetilde{z}_{ij}} \tag{13}$$

在计算出 p 后，进而计算出每个指标的信息熵 e，对于第 j 个指标而言，其信息

熵的计算公式如下所示：

$$e_j = \frac{1}{\ln n} \sum_{i=1}^{n} p_{ij} \ln(p_{ij}) \ (j = 1, 2, 3, 4, 5) \tag{14}$$

通过公式 8 计算出各指标的信息熵后，利用信息熵可以计算信息效用值 $d_j = 1 - e_j$ 计算出每个指标的权重 H，计算公式如下所示：

$$H_j = \frac{d_j}{\sum_{i=1}^{n} d_j} \ (j = 1, 2, 3, 4, 5) \tag{15}$$

第五步：计算得分并归一化处理

定义第 i 各评价对象与最大值的距离为 D_i^+，与最小值的距离为 D_i^-，那么就可以计算出第 i 各评价对象未归一化的得分记为 S,其计算公式如下所示：

$$S_i = \frac{D_i^-}{D_i^+ + D_i^-} \tag{16}$$

最后对得分进行归一化处理得到最终的得分 F，归一化公式如下所示：

$$F = \frac{S_i}{\sum_{j=1}^{n} S_j} \tag{17}$$

到此就可以得出各个供应商的重要度分数，分值越高的供应商对企业的重要程度越大，对得分进行排序取前 50 家供应商即可。

(2)Topsis 评价模型的求解与结果

将已经在 excel 中预处理好的数据导入 Matlab 中，使用编写的 Topsis 算法(代码详见附录 4)对模型进行求解。解得 8 个评价指标的权重如下表所示：

表 4 指标权重表

指标名称	最大值	最小值	平均金额	方差	增长量	频数	超额率	不满足率
权重	0.295	0.319	0.295	0.0004	0.0009	0.064	0.0063	0.017

表 4 中指标均以订购周期为单位，分析表 4 可知，由熵权法得出的权重中，最大值、最小值、平均金额三个权重之和占据总权重的 90.9%，说明供货能力强的供应商对企业的重要性更大。

使用该权重结果对各个供应商的重要性分数进行求解，并对结果进行归一化处理，可得出最终的分数，前 50 家供应商的结果如下表所示（402 家供应商的结果详见附录 5）：

表 5 排名前 50 供应商得分

1~10		11~20		21~30		31~40		41~50	
ID	得分	ID	得分	ID	得分	ID	得分	ID	得分
S229	0.0156	S139	0.0089	S143	0.0061	S367	0.0048	S114	0.0044
S361	0.0144	S268	0.0077	S247	0.0055	S346	0.0048	S395	0.0044
S140	0.0125	S306	0.0075	S201	0.0053	S080	0.0047	S338	0.0044
S108	0.0118	S131	0.0075	S126	0.0052	S294	0.0047	S150	0.0043

S151	0.0114	S374	0.0073	S284	0.0052	S055	0.0047	S037	0.0043
S282	0.0091	S330	0.0072	S307	0.0052	S218	0.0047	S314	0.0043
S275	0.0091	S308	0.0071	S031	0.0052	S244	0.0047	S291	0.0042
S348	0.0090	S356	0.0071	S365	0.0051	S266	0.0045	S086	0.0041
S340	0.0090	S194	0.0068	S040	0.0049	S123	0.0045	S098	0.0040
S329	0.0089	S352	0.0062	S364	0.0049	S007	0.0045	S003	0.0040

本文针对这 50 家供应商得分排名的前 10%和后 10%的供应数据进行分析，首先在 excel 中提取这 10 家供应商的详细数据，提取结果如下表所示：

表 6 前 5 后 5 详细数据表

ID	最大值	最小值	平均金额	方差	频数	超额率	不满足率
S229	43485	27127	42586.44	21357929.41	240	0.0029	0.0153
S361	37700	28134	32808	8329918.8	240	0.0028	0.0186
S140	40063	15816	33225.17	40633772.41	219	0.0007	0.2870
S108	26370	21555	26504.5	1890838.8	240	0.0038	0.0999
S151	62204	11455	19449.8	206822871.8	240	0.0052	0.1866
S314	191	143	5068.6	252.2	235	0.44	0.2863
S291	1018	534	167	16469.44	219	22.44	0.2993
S086	2227	736	958.08	169523.09	206	165.66	0.3735
S098	54	35	1794.9	39.21	206	-0.88	0.0884
S003	2371	189	45.43	720511.56	191	-128.11	0.1551

首先对表 6 中的数据进行量化分析，从表中可知，排名靠前的供应商最大值、最小值、平均交易金额数据都远高于排名靠后供应商。这也与指标的权重大小相对应，虽然前 5 个供应商的供货方差较大，但因为方差所占的比例仅有 0.0004，因此方差对排名的影响微乎其微。排名靠前的供应商与排名靠后的供应商在交易频数上的差别不明显，但排名靠前的供应商对供货量的偏差普遍低于排名靠后的企业，从表中不能直观的看出各供应商之间供货偏差的差异，因此根据各企业的超额率和不满足率可以做出如下图像：

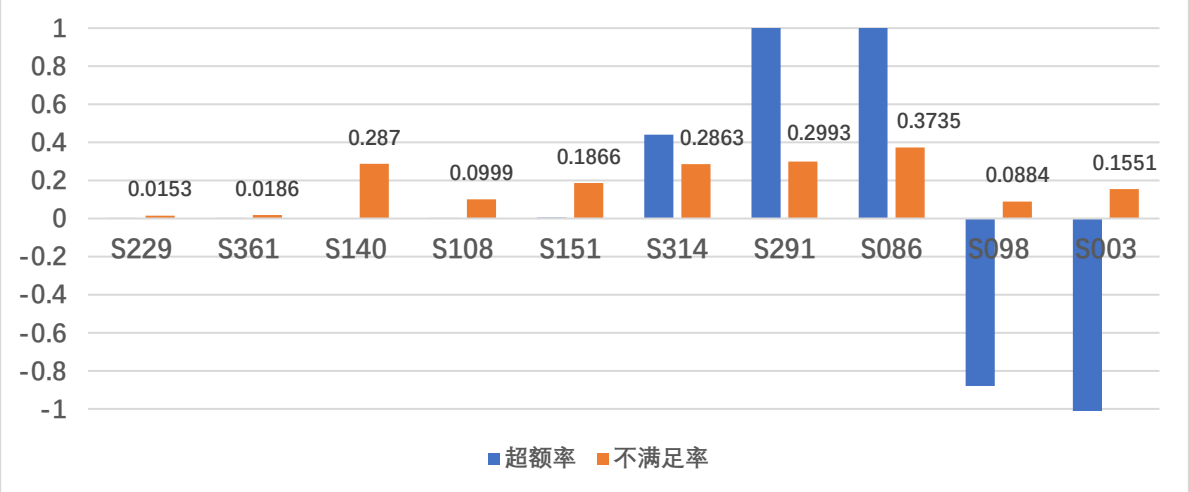

图 6 供应商偏差率情况

图 6 中蓝色线段为供应商的超额完成率，红色线段为供应商的不满足率，从图中可以非常直观的看出，排名前 5 的供应商在偏差率上相较后 5 家供应商有着明显差

异，排名靠后的供应商无法保证供货的偏差在一个合理的区间之内，这对企业的生产来说是不允许的。而排名前2的供应商（S229，S361）的不满足率仅有0.0153和0.0186，无论是供应的规模还是供应量的偏差情况都远优于其他供应商，因此前两家供应商应作为企业的优质长期合作伙伴。

5.2 问题二模型的建立与求解

5.2.1 企业选择模型的建立与求解

首先对题目进行分析，每家供应商每次的实际供应量可能多于或少于企业订购的实际值，为了保证企业的正常生产，企业需一直保持至少两周生产量的库存，且企业对于供应商实际供应的原材料总是全部收购。每个供应商只会提供一种类型的原材料，且每种类型的材料能生产的产品量还有材料的价格也不同，将C的价格设为单位‘1’，根据附件1中给出的各企业的原材料信息可做出下表：

表7 原材料信息表

材料类型	供应商数量（个）	每m^3产品所需量	与C材料价格比例
A	146	$0.6m^3$	1.2
B	134	$0.66m^3$	1.1
C	122	$0.72m^3$	1

分析表7中的数据，三种原材料的供应商数量近似，A类型材料的价格最贵但单位体积的A能够生产的产品量也最多。而B类型材料生产单位体积的产品所耗费的金额最高。

(1)供应商供应的最大值

要使供货的供货商最少，同时也要满足企业正常运转的需求，因此供货商每周的最低供货量应正好维持一周的产能（$28200m^3$），否则就会出现原材料短缺。根据分析，该问题为0-1规划问题，能维持最低需求的供货量应满足以下公式：

$$\sum_{i=1}^{n} \frac{G_{i\,\min}}{p_x} \geq 28200 \tag{18}$$

$$\text{其中} q_a = 1.2, q_b = 1.1, q_c = 1$$

公式12中，n为供应商的数量，$G_{i\,\min}$为各供应商每周供应量的最小值，p为每单位体积产品所需的原材料的体积。根据公式12，只需知道各供应商的最小供应量即可解得满足条件的供应商的数量和代号。

为了找出各个供应商的每周最小供应量，首先分析供应商在5年中每周的供应量，根据附件一中的数据做得重要性得分最高的供应商S229的供应曲线

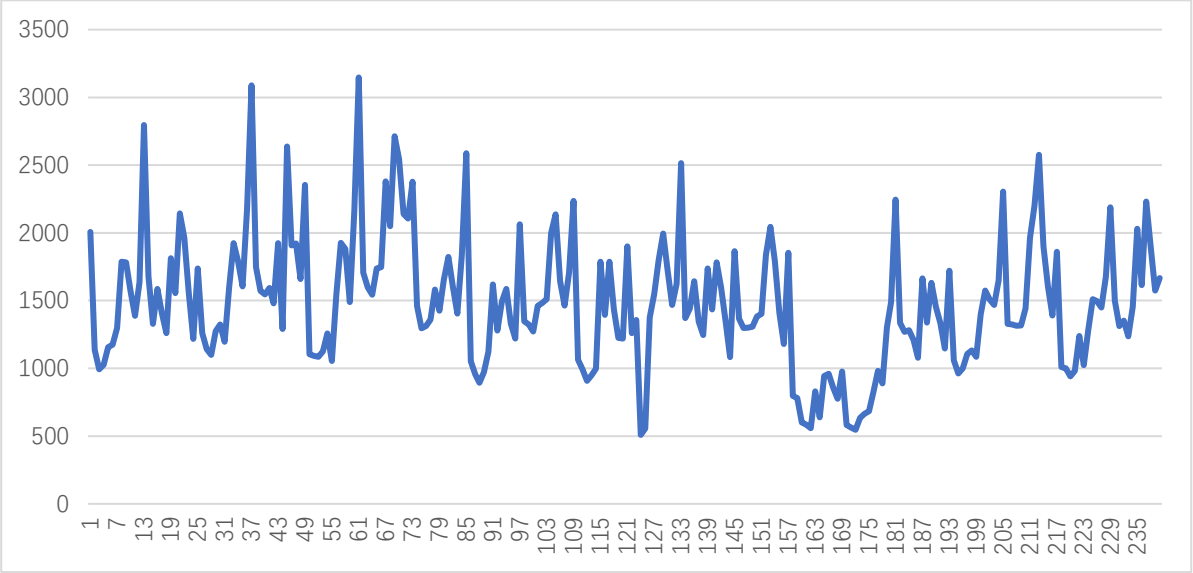

图 7 S229 五年供应情况

观察图 7，横坐标为周数，纵坐标为供应量，S229 五年的供应曲线每周都上下浮动，难以发现其内在规律，但在每个订购周期（24 周）内波动的规律非常近似，为了简化模型，本文假设供应商总会尽力的满足订单的要求，因此本文以每个供应商供应量最大的一个周期作为供应商的周期供给量$G_{\max}$，以这个周期(半年)每周的平均值G_{i}作为供应商一周的正常供应量。即：

$$G_i = \frac{G_{\max}}{24} \tag{19}$$

(2)实际供货量模型的建立

计算各个企业的平均偏差率：

供应商的供货量并不是每次都能按照企业的订购量提供，通常情况下供应商的供货量都在企业订购量上下浮动。为了观察各个供应商的偏差量，计算出每周的实际的偏差率，偏差率计算公式如下所示：

$$P = |v_a - v_b| \tag{20}$$

公式 13 中，P 为实际偏差率，v_a，v_b 分别为超额率和不满足率，在计算出各个供应商每周的实际偏差率后，再求出 5 年内每周偏差率的平均值。偏差率公式计算公式如下所示：

$$\overline{P} = \frac{\sum_{i=1}^{n} P}{n} \tag{21}$$

部分供应商的 5 年每周偏差平均数如下表所示（所有供应商数据详见附录 6）：

表 8 平均偏差表

ID	偏差平均数	ID	偏差平均数
S001	0.58525	S006	0.62197
S002	0.04207	S007	0.06557
S003	0.10013	S008	0.62012
S004	0.66466	S009	0.82001
S005	0.01358	S010	0.63662

高斯分布的检验：

将得到的所有供应商的偏差率分为正偏差率和负偏差率，共得到 804 个数据点，统计 804 个偏差出现的频率，并作出如下图像：

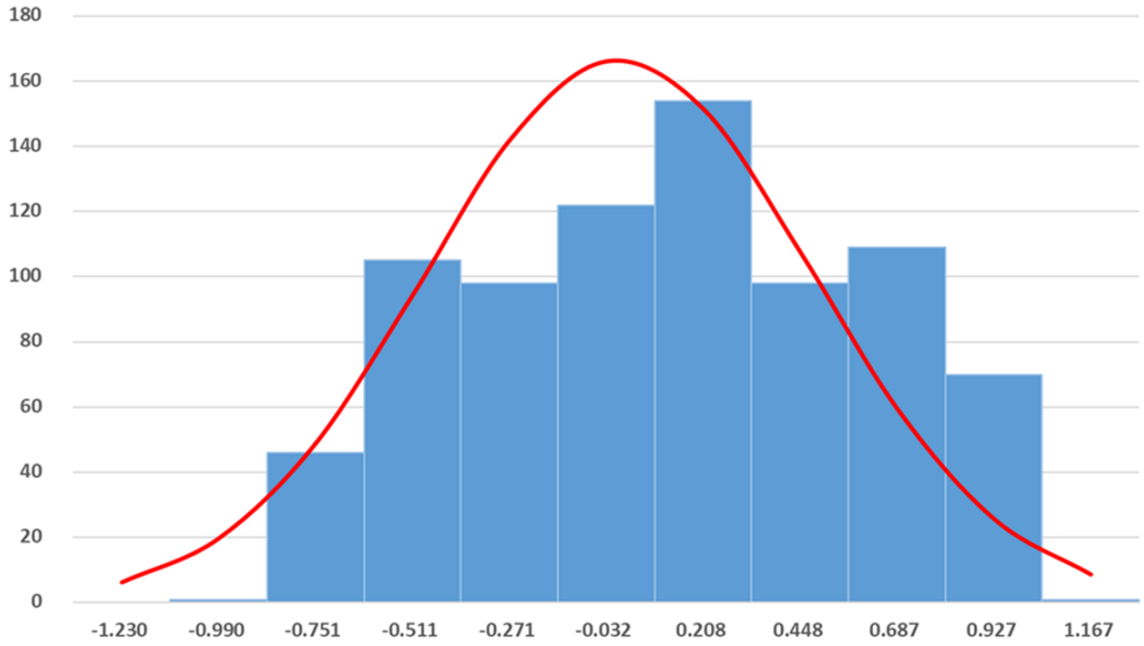

图 8 偏差直方图

图 8 中，横坐标为偏差率(每隔 0.24 为一个区间)，纵坐标为偏差率数据点出现的次数。对图 8 进行分析，402 家供应商的偏差率集中在[-0.032,0.208]之间，因此可以确定 402 家供应商的偏差率在[-0.032,0.208]的概率最大，观察图像，402 家供应商的偏差概率近似服从高斯分布，为了检验数据点对高斯分布的拟合性，对数据点的偏度与峰度进行计算，偏度和峰度的计算公式如下所示：

偏度计算公式：

$$S_k = \frac{\mu_3}{\sigma^3} \tag{22}$$

公式 xx 中，S_k 为偏度，μ_3 为三阶中心矩，σ 为标准差

峰度计算公式：

$$\Upsilon = \frac{\mu_4}{\sigma^4} - 3 \tag{23}$$

公式 xx 中，γ 为峰度，μ_4 为四阶中心矩。

将数据点导入 Spss 中对偏度和峰度进行计算得到如下结果：

表 9 平均偏差的偏度和峰度

偏度	峰度	平均值	标准差
0	-1.002	0	0.4739

由表 9 中的信息可知，因为正负偏差率绝对对称偏度为 0，因此 402 家供应商的偏差率近似服从峰度为-1 标准差为 0.4739 的高斯分布，假设每一家供应商的偏差率都满足高斯分布。每家供应商的平均数 $\bar{p}$ 和标准差σ 计算公式如下所示：

$$\bar{p}=\frac{\sum_{i=1}^{n} p_i}{n}\quad,\quad \sigma=\sqrt{\frac{\sum_{i=1}^{n}\left(p-\bar{p}\right)^2}{n}} \tag{24}$$

(3) 供应商供应的最小值的求解

本文以供应商损失一个标准差的供应量，作为供应商的最低供应量，因为 402 家供应商同时都小于一个标准差的概率只有 0.16 的 402 次方，几乎为不可能事件。因此只要满足此时的供应能够满足产能，那么就能保证企业一直正常生产。每家供应商的最小供应量公式如下所示：

$$G_{i\ \min}=G_i\times(1-\sigma_i) \tag{25}$$

使用公式 22 可以解得各个企业的最小供应量$G_{i\ \min}$

供应商选择结果的求解：

使用 Excel 的数据透视表工具，解出所有符合公式 xx 的供应商组合，选择供应商数量最少的组别，经过统计供应商数量的最小值为 18 家企业，有多个组别的供应商都符合要求，因此再对每个组别的重要性得分进行求和处理，以重要性得分最高的一组作为最佳组别，最佳组别中的供应商即为所求。18 家供应商详细数据如下表所示（其它 n 值最小的结果详见附录 7）：

表 10 18 家供应商详细信息

编号	类别	最低产能	重要性得分
S229	A	2951.043	0.015645
S361	C	2117.532	0.014353
S140	B	1905.329	0.012496
S108	B	1453.266	0.011759
S151	C	2713.601	0.011415
S282	A	1413.796	0.009106
S275	A	1259.456	0.00909
S348	A	2675.623	0.009007
S340	B	1231.622	0.008956
S329	A	1275.585	0.008936
S139	B	1032.365	0.008858
S268	C	849.8918	0.007706
S131	B	995.1139	0.007523
S374	C	2423.916	0.007345
S330	B	1055.986	0.007178
S308	B	949.5342	0.007102
S356	C	989.9125	0.007054
S352	A	958.5212	0.006245
产能总和		重要性得分总和	
28252.0		0.169775	

分析表 10 中的数据，提供原材料类别为 A 的供应商共 6 家，类别为 B 的供应商共 7 家，类别为 C 的供应商共 5 家。这 18 家供应商最低所能给企业提供的产能之和为 28152，这 18 个供应商供应量均低于最低产能的概率仅为 0.16 的 18 次方，基本为不可能事件，满足题意。

5.2.2 原材料订购方案的模型建立与求解

(1)单目标规划问题的建立

分析题意，针对以上求得的 18 家供应商，为该企业定制未来 24 周每周从这些供应商进货的量，使得每周进货花费最少。这是一个单目标规划问题，目标函数应为每周进货所花的钱最少，约束条件应为保证每周 28200 立方米产品的产能。

目标函数的确定

经过上述的分析，已经确定了目标函数为每周进货花费最少，设 A 原料供应商本周的进货量为 a，设 B 原料供应商本周的进货量为 b，设 C 原料供应商本周的进货量为 c，设本周供应商服从高斯分布的供货偏差为 p，18 家企业在供应 A 的有 6 家，供应 B 的有 7 家，供应 C 的有 5 家,现要是每周进货的花费最少，价格以 c 的单价表示，公式如下所示：

$$\min E=\sum_{i=1}^{6}1.2a_i+\sum_{j=1}^{7}1.1b_j+\sum_{k=1}^{5}c_k \tag{26}$$

约束条件的确定

企业需要时刻保存两周或两周以上产能的库存，因此只需每周进货原料所带来的产能大于每周生产的产能即可。第一个约束条件为：

$$\sum_{i=1}^{6}\frac{a_i}{0.6}(1+p_i)+\sum_{j=1}^{7}\frac{b_j}{0.66}(1+p_j)+\sum_{k=1}^{5}\frac{c_k}{0.72}(1+p_k)\geq 28200 \tag{27}$$

由于一个供应商只能选择一家转运商进行转运，而一家转运商的最大载货量为 6000，每家供货商的供货量不能大于 6000。第二个约束条件为：

$$0\leq a_i\ ,\ b_j\ ,\ c_k\ \leq 6000 \tag{28}$$

综上所述，每周进货最经济的单目标规划模型为：

$$\min E=\sum_{i=1}^{6}1.2a_i+\sum_{j=1}^{7}1.1b_j+\sum_{k=1}^{5}c_k$$

$$s.t.\begin{cases}\sum_{i=1}^{6}\frac{a_i}{0.6}(1+p_i)+\sum_{j=1}^{7}\frac{b_j}{0.66}(1+p_j)+\sum_{k=1}^{5}\frac{c_k}{0.72}(1+p_k)\geq 28200\\ a_i\ ,\ b_j\ ,\ c_k\ \geq 0\\ a_i\ ,\ b_j\ ,\ c_k\ \leq 6000\end{cases} \tag{29}$$

(2)单目标规划模型的求解

模拟退火算法的概述

模拟退火算法(Simulated Annealing Algorithm,简记为 SSA 或 SA)，它是受物理中固体物质的退火机理与优化问题存在相似性的启发，而建立的一种优化方法，在概率层面，利用随机搜索技术找到目标函数的全局最优解。

带有惩罚函数法的模拟退火算法

因为存在约束条件，所以使用惩罚函数法来实现约束条件对目标函数的作用。当不满足惩罚函数时，会让目标函数的值变差。

惩罚函数法将约束条件为：max:$F(X)$ 其转化为 $\min(0,f(X))^2$，最后添加上

在原有的目标函数，为无约束问题的目标函数。

算法步骤

利用 Python 编程语言(源代码见附录 8)编写算法，其中模拟退火算法的核心为搜索最优解的过程可视为随机过程中的马尔科夫过程，根据状态转移概率P^*，由一个候选解转移到另一个候选解。当温度T降为 0 时，x_i的分布为：

$$P_i^* = \begin{cases} \dfrac{1}{|S_{\min}|}, & x_i \in S_{\min} \\ 0, & \text{其他} \end{cases} \qquad (30)$$

$$\sum_{x_i \in S_{\min}} P_i^* = 1$$

算法具体流程图如下图所示：

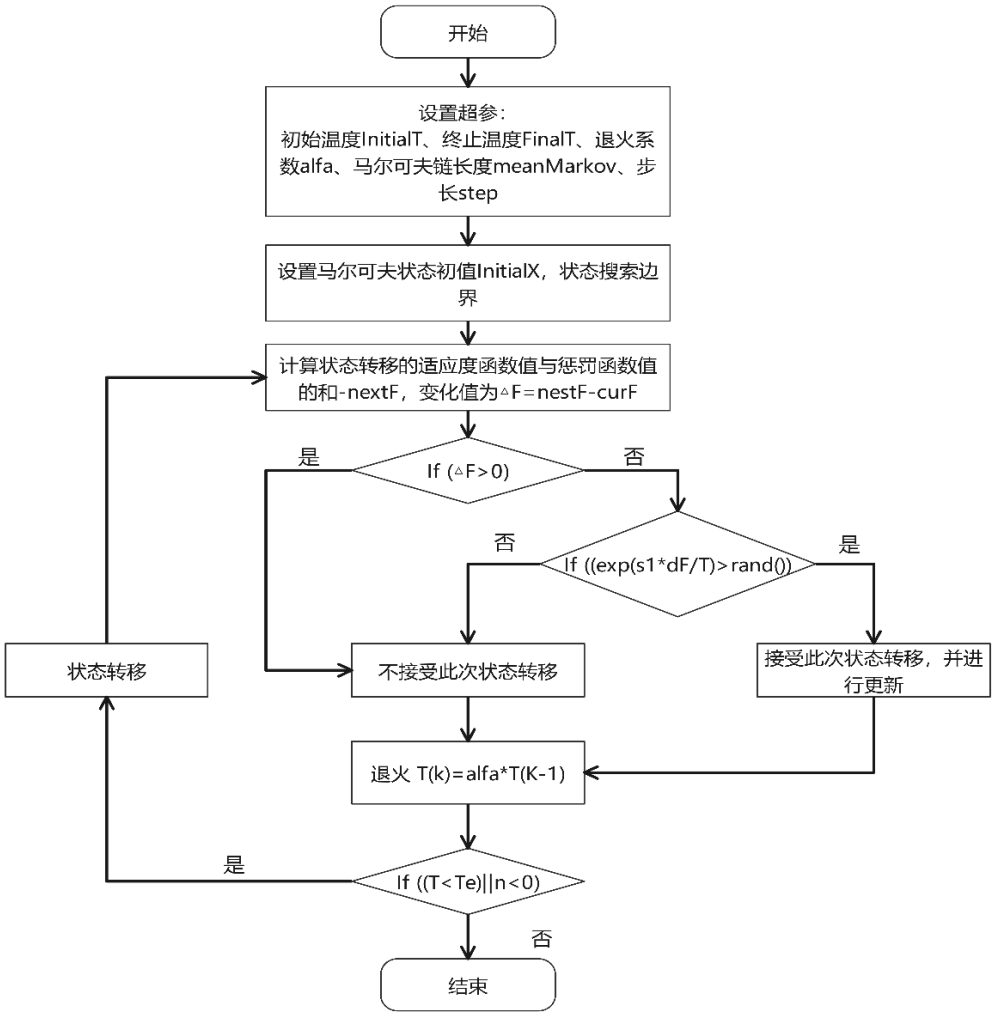

图 9 模拟退火流程图

(3)单目标规划模型结果

利用模拟退火算法对公式 19 的单目标规划模型进行求解，为模拟现实场景，偏差率因子 p 为各个供应商服从高斯分布的偏差率，每计算出一周的结果，下一周的 p 值就会发生改变。第一周的求解结果如下表所示（24 周结果详见附录 9）：

表 11 第一周采购计划表

供应商 ID	原料类别	供应量	各类供应总量

S229	A	2101.249	7643.98
S282	A	2953.494	
S275	A	339.3119	
S348	A	198.9135	
S329	A	2029.91	
S352	A	21.10182	
S140	B	112.9069	3769.293
S108	B	2461.781	
S340	B	135.4224	
S139	B	569.5618	
S131	B	178.9964	
S330	B	263.6932	
S308	B	46.9315	
S361	C	200.9461	7755.666
S151	C	2652.743	
S268	C	1216.781	
S374	C	3393.391	
S356	C	291.8056	

对表 11 中的数据进行分析第一周采购原材料 B 的量最少，与前面提到的 B 生产每单位的产品所耗费的金额最大结果相一致。A 类型供应商中的 S352 采购的量最小，可能是收到偏差率因子 p 的影响，导致该供应商的不满足率较大，为节省订购成本因此在这家供应商的订购量要尽可能小。

根据 24 周内每周采购的总金额，画出一个采购周期的每周采购总金额的变化曲线，变化图像如下图所示：

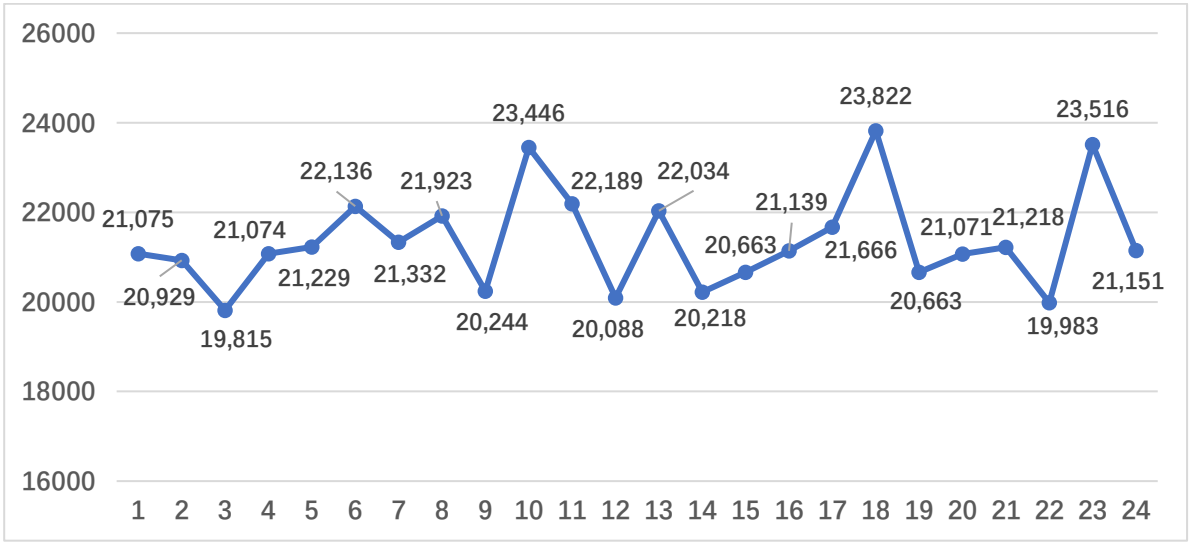

图 10 周期采购金额变化曲线

图 10 中，横坐标为一个采购周期内的周数，纵坐标为每次分析图 10 中的信息，一个采购周期内的总金额在[19814,23822]上下浮动，平均数为 21359，平均每周采购量能达到的产能为 29665，超企业正常产能 1465，可能是受到了偏差率因子 p 的影响。

在解决采购问题后，接下来本文建立转运优化模型对损耗最小的转运方案进行求解。

5.2.3 转运优化方案模型的建立与求解

在得到每周最经济的采购方案后，针对这 24 周的采购方案设计出损耗最小的转运方案，分析题干后得知，每个转运商的最多只能转运 $6000m^3$，而每次转运途中都会出现损耗，根据上述分析可以做出如下转运流程图：

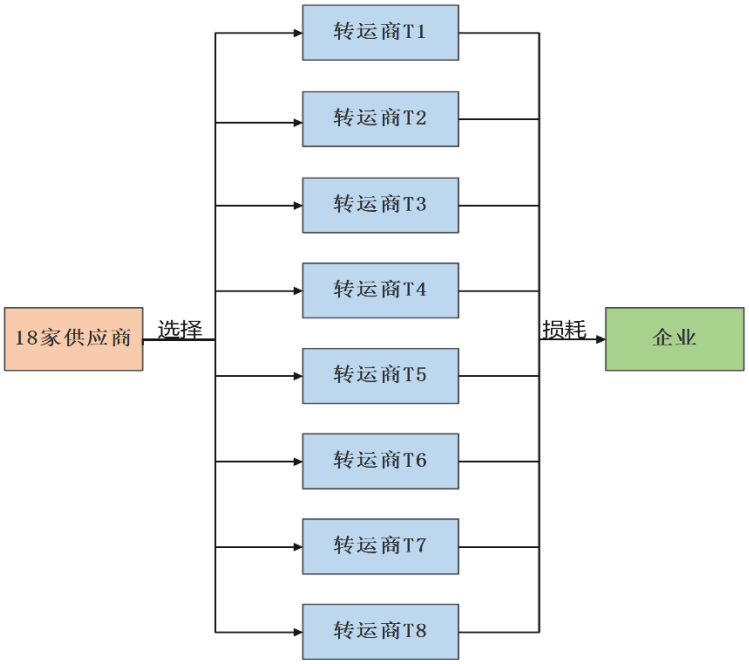

图 11 转运流程图

(1) 模型的推导

为解决最小转运损耗量的问题，首先建立供应商与转运商的选择矩阵 C，矩阵如下所示：

$$C=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} \\ x_{21} & x_{22} & \cdots & x_{2j} \\ \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & \cdots & x_{ij} \end{bmatrix} \qquad \text{其中}\begin{cases} x=0,1 \\ i=18 \\ j=8 \\ \sum_{1}^{j} x_{ij} \le 1 \end{cases}$$

C 矩阵中 18 行表示 18 家供应商，8 列表示 8 家转运商，其中 x 表示是否选择，是为 1 否为 0，由于一个供应商只能指定一家转运商，因此矩阵的每一行之和必须小于等于 1。

建立供应商供应量矩阵 H 和转运商转运损耗率矩阵 T，矩阵如下所示：

$$H=\begin{bmatrix} h_{11} \\ h_{21} \\ \cdots \\ h_{i1} \end{bmatrix} \qquad T=[t_{11} \quad t_{12} \quad \cdots \quad t_{1j}]$$

矩阵 H 中的元素 h 表示 18 家供应商的供应量，矩阵 T 中的元素 t 表示 8 个企业的损耗率，附件二中给出了 8 家转运商 5 年内每周的损耗率。为简化模型且使模型更贴合实际情况，本文近似的将每家转运商转运的损耗率的分布情况看作高斯分布，计算出每个转运商的平均转运损失率 $\bar{p}$ 和标准差 σ 即可求出对于转运商损失率的分布曲线，计算公式如下所示：

$$\bar{p}=\frac{\sum_{i=1}^{n} p_i}{n} \quad , \quad \sigma=\sqrt{\frac{\sum_{i=1}^{n}\left(p-\bar{p}\right)^2}{n}} \tag{31}$$

在得出一个转运商的损耗率分布后，转运商的损耗率即为分布上对于曲线上的一个点，也即是对应 t 的取值。将选择矩阵 C 与供应商供应量矩阵 H 进行点乘，得到一个新的矩阵 CH。具体公式如下所示：

$$CH=C\cdot H=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} \\ x_{21} & x_{22} & \cdots & x_{2j} \\ \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & \cdots & x_{ij} \end{bmatrix}\cdot\begin{bmatrix} h_{11} \\ h_{21} \\ \cdots \\ h_{i1} \end{bmatrix} \tag{32}$$

矩阵 CH 为每家供应商在对应选择的转运商中的转运量，将 CH 矩阵逐行求和，得到每个转运商的转运总量矩阵记为 V，由于 1 家转运商最大运算量为 6000 因此矩阵 V 中的每个元素 v 均不能大于 6000，矩阵 V 与损耗率矩阵 T 进行点乘，即可解得每个转运商的损失量 L，具体公式如下所示：

$$L=V\cdot T=[\,v_{11}\ \ v_{12}\ \ \ldots\ v_{1j}]\cdot[\,t_{11}\ \ t_{12}\ \ \ldots\ t_{1j}]\ \ (v<6000) \tag{33}$$

矩阵 L 为每个转运商转运的损失量，将 L 中的所有元素进行求和即为本次转运计划的总耗损量。

得到最终优化函数：

$$\min l=l_{11}+l_{12}+\cdots+l_{1j} \tag{34}$$

公式 34 中 l 为矩阵 L 中的元素，元素 l 的含义是每个转运商的转运损耗量，将矩阵 L 中的所有元素进行求和，即可得到这次转运计划的总损耗量。

接下来将基于蒙特卡洛算法使用 python 代码对优化函数的最小值进行求解。

(2) 优化模型的求解与结果

使用编写好的 python 算法(代码详见附录 10)，对选择矩阵进行 100 万次模拟，计算每次的损耗量 l，最终求解出最小 l 所对应的转运方案，部分供应商的前 12 周转运方案结果如下表所示（所有供应商的方案详见附录 9）：

表 12 部分供应商 12 周转运计划表

周数	S229	S361	S140	S108	……	S330	S308	S356	S352
1	T5	T3	T3	T7	……	T7	T7	T7	T4
2	T2	T6	T8	T1	……	T2	T8	T1	T2
3	T5	T1	T3	T1	……	T3	T4	T4	T4
4	T3	T4	T8	T8	……	T8	T5	T4	T3
5	T3	T5	T3	T5	……	T3	T7	T7	T2
6	T4	T6	T6	T6	……	T8	T6	T5	T6
7	T6	T2	T4	T8	……	T3	T4	T4	T3
8	T4	T7	T4	T4	……	T5	T7	T7	T5
9	T1	T1	T4	T3	……	T7	T4	T6	T4

10	T1	T3	T8	T2	……	T3	T8	T8	T4
11	T2	T1	T2	T3	……	T2	T6	T8	T8
12	T6	T3	T8	T2	……	T8	T1	T8	T5

由于偏差率因子的影响，每周选择的转运商都在变化，难以从表中发现规律。因此将每周的采购金额与每周转运时的损耗损耗量进行对比，从而分析方案的实施效果，对比结果如下图所示：

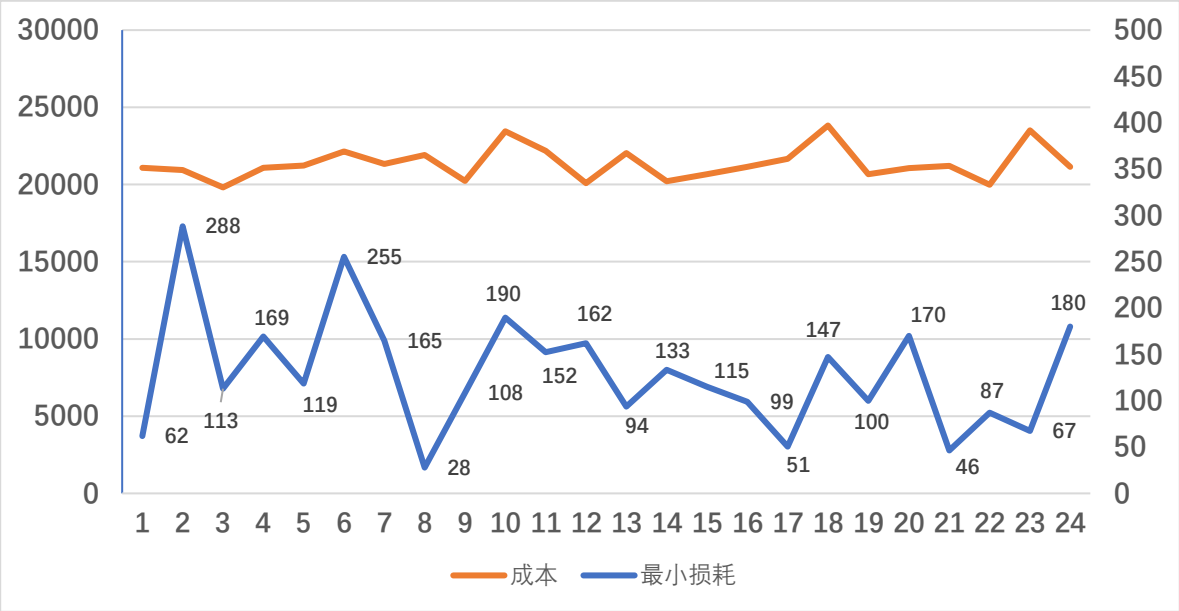

图 12 采购成本与最小耗损对比图

图 12 中，蓝色线段代表最小耗损对应的右边的刻度，红色线段为采购最小采购成本对应的左边的刻度。分析图 12 中的信息，耗损的量与采购成本的量对变化趋势基本相同，因为受到偏差率因子的影响，最小损耗会在一个区间内浮动，这一个采购周期（24 周）内损耗量的浮动范围为[28，288]。按此方案进行采购和转运，每周采购维持在 23000 上下浮动，每周的转运耗损维持在 150 上下浮动。

5. 3 问题三模型的建立与求解

5. 3. 1 原材料订购方案的模型建立与求解

(1) 多目标规划模型的建立

首先对题目进行分析，本题与问题二的区别在于 A 类型与 C 类型采购的量不同，问题三要求企业尽可能多采购 A 类型的原材料以减少转运以及仓储的成本，当一周的实际供货量与产能相等时，能保证仓储成本的最小。因此本问为多目标规划问题，根据上述分析，建立目标函数。

第一个目标函数：

为尽可能压缩仓储成本

$$\min N = \left(\sum_{i=1}^{6} \frac{a_i}{0.6}(1+p_i) + \sum_{j=1}^{7} \frac{b_j}{0.66}(1+p_j) + \sum_{k=1}^{5} \frac{c_k}{0.72}(1+p_k) \right) - 28200 \tag{35}$$

公式 26 中，N 为每周多余产能，a_i 为 A 类型供应商的订购量，b_j 为 B 类型供应

商的订购量，c_k 为 C 类型供应商的订购量，p 为服从高斯分布的偏差率因子。

第二个目标函数：

为使 A 原材料订购量尽量大于 C 原材料订购量

$$\min b = \frac{\sum_{k=1}^{5} c_k}{\sum_{i=1}^{7} a_i} \tag{36}$$

公式 27 中，b 为 C 类型订购量与 A 类型订购量的比例。

约束条件的确定：

由于转运商的限制，每个供货商的供货量不能大于 6000，也不能小于 0。

$$0 \leq a_i \ , \ b_j \ , \ c_k \ \leq 6000 \tag{37}$$

综上所述，保证仓储成本最小且 A 类型与 C 类型订购量差值尽可能大的双目标规划模型为：

$$\begin{aligned} \min N = & \left(\sum_{i=1}^{6} \frac{a_i}{0.6}(1+p_i) + \sum_{j=1}^{7} \frac{b_j}{0.66}(1+p_j) + \sum_{k=1}^{5} \frac{c_k}{0.72}(1+p_k) \right) - 28200 \\ & \min b = \frac{\sum_{k=1}^{5} c_k}{\sum_{i=1}^{7} a_i} \\ & s.t. \begin{cases} a_i, b_i, c_i \geqslant 0 \\ a_i, b_i, c_i \leqslant 6000 \end{cases} \end{aligned} \tag{38}$$

(2) 多目标规划模型的求解

为什么选择多目标粒子群算法?

模拟退火作为一种单目标优化算法，求解过程具有连续性，在非线性目标中易陷入局部最优解；对于超参的敏感度高，不同的选取会导致结果差异较大。

所以对于多目标规划问题，在这种不同函数之间的博弈过程，为求解结果，本文选择了作为群体智能算法中的一种，多目标粒子群算法(MOPSO)。它在已经在很多优化问题上得到成功应用，同时不要被优化函数具有可微、可导、连续等性质，并具有收敛速度快、控制参数少、计算速度快等特点[]。

粒子群算法概述：

粒子群算法，也称为粒子群优化算法(particle swarm optimization，PSO)，是 Eberhart 等[6]在 1995 年提出的新仿算法。该算法通过模拟鸟群觅食行为在求解空间进行目标函数的最优解的过程。在粒子群算法中，每个优化问题的可能被假定为 D 维搜索空间中的一个多边形的顶点，称为“粒子”[7]。所有粒子具有目标函数确定的适应度值，粒子按照确定飞行方向和飞行距离的速度，跟随搜索空间中当前的最佳粒子进行搜索[8]。粒子经初始化后，确定一个随机解，然后通过不断更新个体极值和全局极值的迭代方式来寻优[9]。

基于经典的粒子群算法，加入惯性权重，同时限制粒子的多个适应值，实现多目标粒子群算法。得到核心公式如下：

$$v_i^d = wv_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)$$

其中c_1:个体加速因子 c_2:社会加速因子 w:惯性权重

v_i^d:第d次迭代时第i个粒子的速度

x_i^d:第d次迭代时第i个粒子的位置 (39)

$pbest_i^d$:到第d次迭代为止，第i个粒子经过的最好的位置

$gbest^d$:到第d次迭代为止，所有粒子经过的最好的位置

r_1，r_2是$[0,1]$上的随机数

算法步骤：

使用 Python 实现多目标粒子群算法(代码详见附录 11 和 12)，算法的具体流程图如下所示：

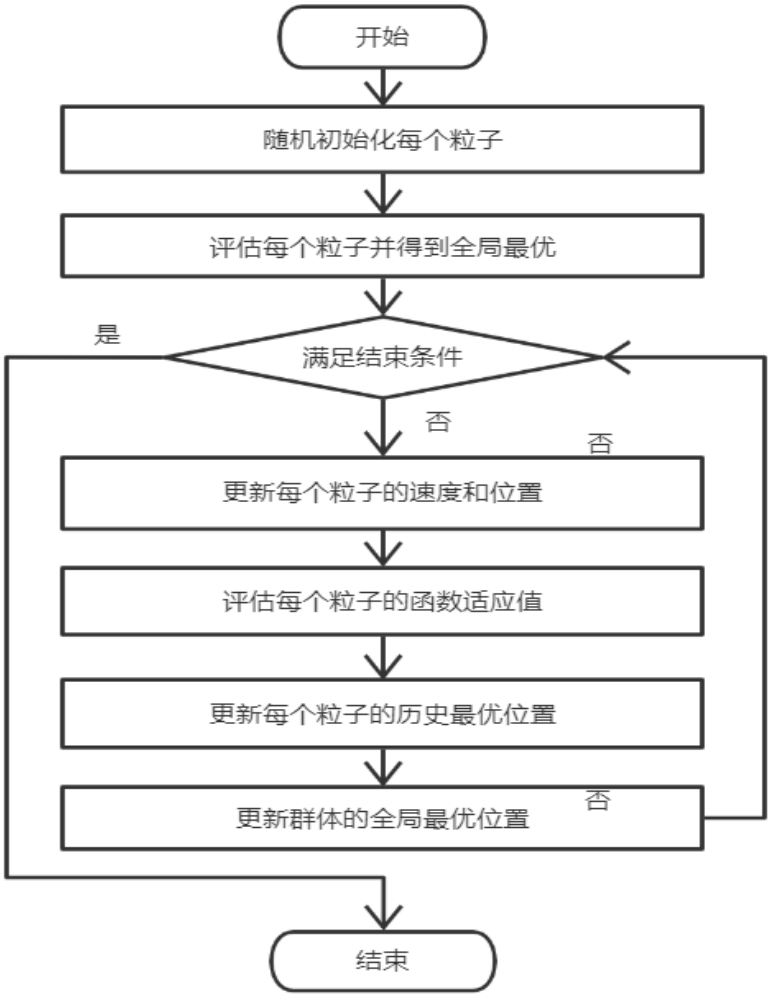

图 13 粒子群算法流程图

(3) 多目标规划模型的结果：

利用粒子群算法对多个目标函数进行求解，为模拟真实情况，每计算一周的计划后，各供应商的偏差率因子 p 就会发生改变，共计算出 24 周所有的采购计划，前 6 周的两个目标函数求解结果如下表所示：

表 13 前 6 周结果表

周数	1	2	3	4	5	6
多余产能 N	84.39	276.92	390.03	0.14	0.87	5.59

比例 b	0.0674	0.0411	0.0279	0	0	0

前六周中，每周的多余产能在[0.14，390.03]间浮动，满足减少仓储成本的要求。C 采购量与 A 采购量的比值最大仅有 0.0674，满足尽可能多采购 A 少采购 C 的要求。经过整理可得到每一周的详细方案，第一周的详细方案如下表所示：

表 14 第一周采购方案

供应商 ID	原料类别	供应量	各原料总供应量
S229	A	2382.476	18540
S282	A	3426.835	
S275	A	3808.958	
S348	A	2231.117	
S329	A	3827.245	
S352	A	2863.526	
S140	B	3875.461	16981
S108	B	2381.477	
S340	B	1413.174	
S139	B	2972.337	
S131	B	2211.238	
S330	B	1843.058	
S308	B	2284.874	
S361	C	350.4482	1250
S151	C	521.504	
S268	C	71.45799	
S374	C	95.48551	
S356	C	211.3103	

分析表 14 中的数据，将 A 类型的供应商订购量相加得到 A 类型的总订购量为 11124.09，同理 B 类型的总订购量为 11207.87，C 类型的总订购量为 900.15，由于随机因子的影响会导致供应商实际提供的量不满足订购量，因此订购的原材料理论商提供的产能会大于实际提供的产能。

从 24 周整体进行分析，将得到的每周的多余产能 N 和比例 b 导入 excel，可得到如下图像：

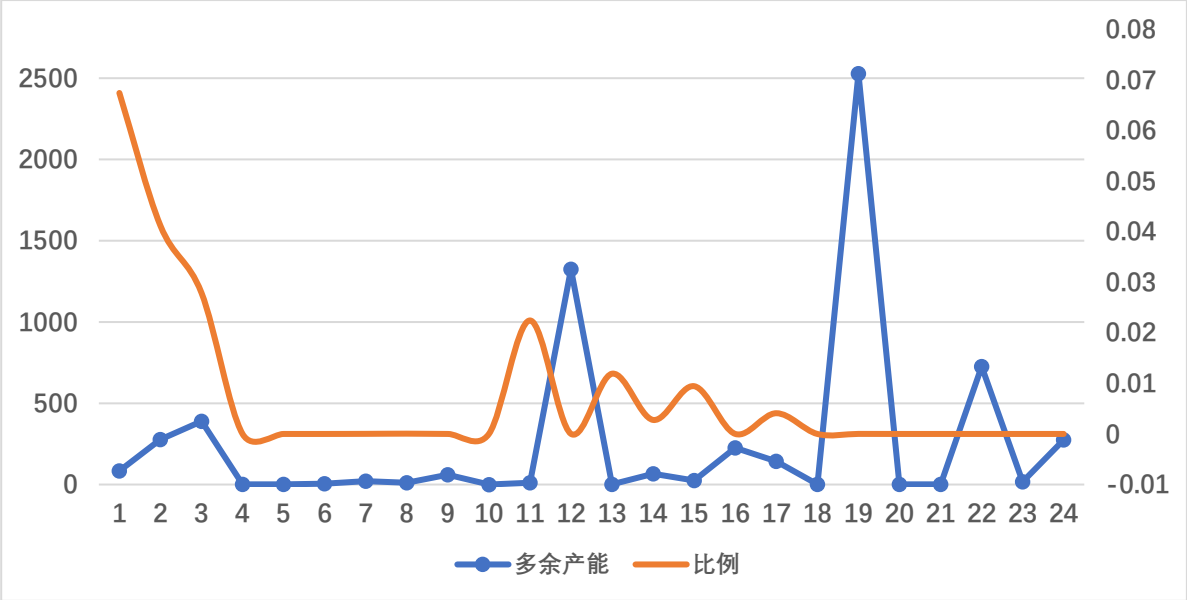

图 14 每周多余产能与比例图

图 14 中蓝色曲线代表每周多余产能，橙色曲线代表每周采购 A 与 C 数量的比例。对图 14 进行分析，19 周的多余产能突然到达了 2500 以上，但比例仍保持稳定，可能是因为 19 周供应商的偏差率因子与标准值偏差较大，进而导致实际供应量与理论订购量的偏差也较大。观察比例与多余产能的变化趋势，多数情况下，在比例下降时多余产能上升，可以推测比例与多余产能呈现反比的关系。

5. 3. 2 转运方案模型的建立与求解

在多目标规划问题的求解中已经求出了各周的详细订购方案，只需将每个供应商订货量数据带入问题二已经建立好的转运优化模型中，即可解得各周的转运计划，部分供应商的前 12 周转运方案结果如下表所示：

表 15 部分供应商 12 周转运计划表

周数	S229	S361	S140	S108	……	S330	S308	S356	S352
1	T2	T2	T3	T8	……	T1	T4	T2	T4
2	T5	T2	T4	T3	……	T2	T6	T5	T6
3	T6	T6	T4	T2	……	T8	T4	T7	T6
4	T7	T1	T6	T7	……	T8	T1	T3	T8
5	T6	T1	T4	T1	……	T1	T4	T5	T1
6	T8	T4	T2	T7	……	T7	T6	T8	T7
7	T8	T5	T4	T6	……	T3	T3	T8	T8
8	T2	T8	T4	T5	……	T3	T3	T7	T8
9	T6	T5	T1	T3	……	T6	T5	T4	T4
10	T6	T6	T8	T2	……	T1	T4	T3	T1
11	T2	T2	T1	T6	……	T8	T7	T4	T4
12	T1	T6	T4	T4	……	T1	T8	T2	T5

分析表 15 中的数据，由于偏差率因子的影响每周的计划也会有所改变，每周选择的供应商数量大多为损耗率低的 5 家。从表中无法直观的看出耗损、多余产能和比例间的联系，将三者放入一个图像中对方案的实施效果进行分析，如下图所示：

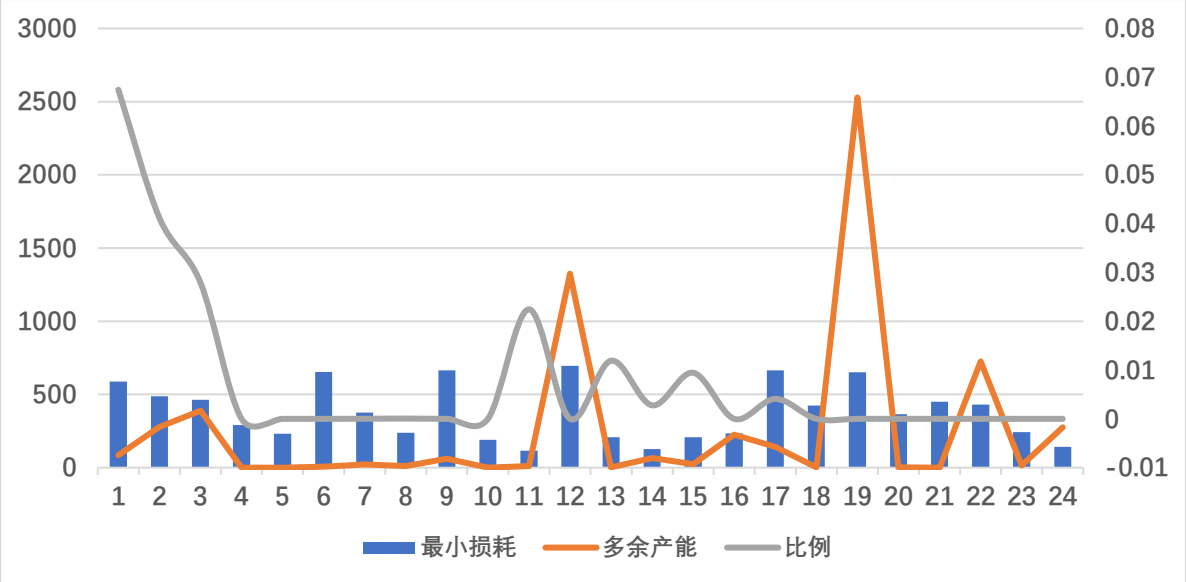

图 15 每周计划结果表

图 15 中，横坐标为周数共 1 各采购周期(24 周)，蓝色柱体表示转运方案每周的最小耗损，对应左边的纵轴，红色线段表示每周多余产能，对应左边的纵轴，灰色曲线为每周采购 C 与 A 的比例。对图中的信息进行分析，多数情况下转运的最小耗损会随着比例的变化而产生相同趋势的变化，二者呈正相关，而多余产能与比例呈现负相关的关系，因此可以推测，多采购 A 而少采购 C 的方案不仅能减少转运时的损耗，还能减少采购的花费。

5. 4 问题四模型的建立与求解

5. 4. 1 动态模拟模型的建立

首先分析题意，通过技术改造企业的产能获得了提升，但产能的提升上限收到供应商供应量以及转运商转运量的限制，当供应商的供应量到达上限时，或者转运商的转运量到达上限时，在保证企业正常生产的情况下，企业的产能就无法继续提升了。因此需要找到供应量和转运量最大值中的最小值，作为一周的最大供给量。

供应商供应的最大值：

在问题二中，本文以供应商供应平均值与 1 减标准差相乘的结果作为供应商供应量的最小值即：$G_{i\ \min}=G_i\times(1-\sigma_i)$。本题中，本文以供应商平均供应量与 1 加标准差相乘的结果作为供应商供应量的最大值即：$G_{i\ \max}=G_i\times(1+\sigma_i)$，经上述可得到供应商与转运商最大值的不等式：

$$\sum_{i=1}^{n} G_{i\ \max}\geq 48000 \tag{40}$$

公式 32 中，48000 表示 8 家转运商都装满时的最大转运量，将企业按照最大供应量降序排列，对不等式进行求解，解得 n=25 时供应商的供应量之和为 47507.68，当第 26 企业加入时供应量超过了 48000，因此选择最大供应量前 25 的供应商进行供货。

企业产能的最大值：

供应商的供货量不可能时刻保持在最大值，所有供应商都到达最大值以上的概率仅为 0.16 的 25 次方，基本可以忽略。因此本文以这 25 家企业的供应最小值之和作为

企业的最小采购量，当企业的产能在最小采购量时也能正常(供应量的产能大于等于企业每周产能)且安全(库存大于等于两周的产量)的生产时，这时就是企业产能的最大值。具体公式如下所示：

$$\sum_{i=1}^{25}\frac{G_{i\ \min}}{q}\geq O_{\max}\begin{cases}q=0.6\\q=0.66\\q=0.72\end{cases}\tag{41}$$

$$K\geq 2O_{\max}$$

公式 33 中，q 为单位体积的产品所需的原材料，O 为企业产能，K 为企业库存。

每周产能增加量的目标函数：

根据上述的分析，在尽可能减少仓储成本的情况下，参考问题二的目标函数，25 家供货商中，A 类型供货商共有 8 家，B 类型供货商共有 7 家，C 类型供货商共有 10 家，设计企业每周的采购成本目标函数：

$$\min E=\sum_{i=1}^{8}1.2a_i+\sum_{j=1}^{7}1.1b_j+\sum_{k=1}^{10}c_k\tag{42}$$

约束条件：

需保证每周进货量所带来的产能至少能够供应一周的产能。

$$\sum_{i=1}^{8}\frac{a_i}{0.6}(1+p_i)+\sum_{j=1}^{7}\frac{b_j}{0.66}(1+p_j)+\sum_{k=1}^{10}\frac{c_k}{0.72}(1+p_k)-O>0\tag{43}$$

记公式 35 得到的多余产能为 N，将多余产能的原材料存入仓库，产能就能提高至当前库存的一半，每计算一周后，下一周的产能会要改变，产能改变公式如下所示：

$$O_{next}=\frac{K+N}{2}\tag{44}$$

在计算下一周的目标函数时就需要将O_{next}作为 O 带入方程 35 中，作为新的约束条件。

5.4.2 动态模拟模型的求解与结果

使用问题二中模拟退火算法，将产能的初值 28200 带入约束条件中，每次计算后更新产能为最新值，计算出 24 周的产能数据，前 6 周数据如下表所示：

表 16 前六周企业产能情况

周数	1	2	3	4	5	6
采购金额	30012.20	30012.20	31276.87	30540.12	32802.96	31696.79
产能差值	15756.63	14077.85	-781.09	5919.17	2580.12	-738.92
本周产能	28200	36078	43118	42726	45685	46975

分析表 16 中的数据，前两周的产能差值均在 10000 以上，第三周以后产能差值就开始波动，推测可能在第三周产能就接近了最大值。

使用 Matlab 拟合工具箱拟合(拟合方法选择 Polynomial)出 24 周的产能变化情况曲线，直观的对产能的变化进行分析，5 阶拟合图像如下所示：

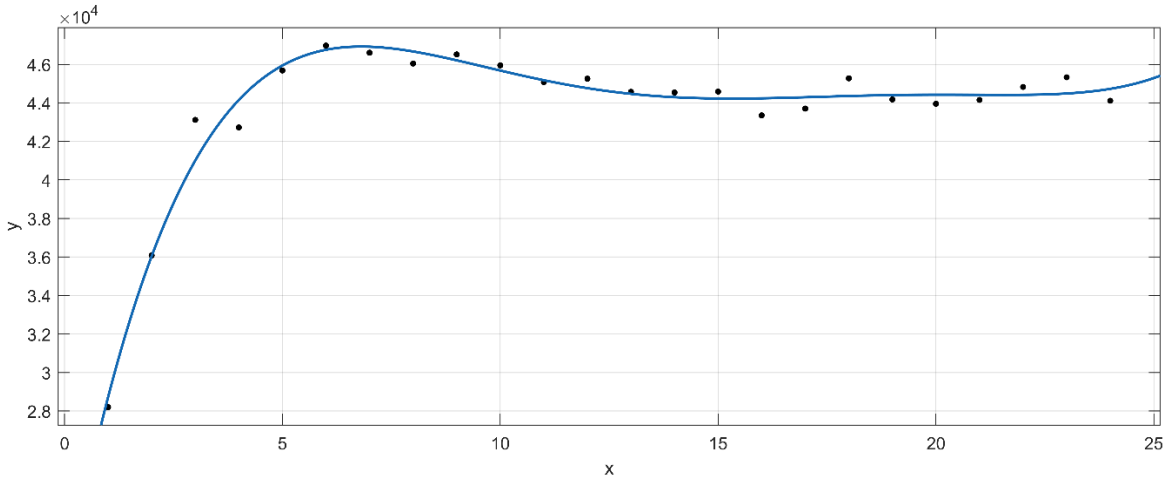

图 16 产能变化拟合图

图 16 中，横坐标为周数，纵坐标为周产能。对图 16 进行分析，在提升产能后，企业的产能在 6 周时到达最大。

在约束条件下，为求解企业可以到极限的产能，结合图像以及求解结果，计算第 7 周到 24 周的产能平均值为 44892。因此在技术改进后，企业每周的产能最大能够提升 56%。

转运方案模型的求解与结果

根据解得的每周最经济的采购信息，制定出未来 24 周的采购计划，再使用问题二中的转运优化模型求解出 24 周的转运计划，第一周的采购计划如下表所示：

供应商 ID	原料类别	供应量	各类供应总量
S229	A	2951.043	
S282	A	1413.796	
S275	A	1259.456	
S348	A	2675.623	12720.72
S329	A	1275.585	
S352	A	958.5212	
S143	A	608.4315	
S201	A	1578.263	
S140	B	1905.329	
S108	B	1453.266	
S340	B	1231.622	
S139	B	1032.365	8623.217
S131	B	995.1139	
S330	B	1055.986	
S308	B	949.5342	
S361	C	2117.532	
S151	C	2713.601	
S268	C	849.8918	13119.98
S306	C	835.1867	
S374	C	2748.289	
S356	C	1005.676	

S194	C	685.4278
S247	C	380.9493
S126	C	1365.788
S284	C	417.6396

六、 模型的评价、改进与推广

6.1 模型的优点

第一问：结合实际情况，考虑到企业对供应商的供货短缺和超量供货的接受度不同，将偏差率分为不满足率和超额率分开进行计算，是模型更具客观性。

第二问：第三问的模型中，在对供应商的实际供货量，和转运商的损耗率的计算中引入浮动因子，使模型更符合实际情形。

第四问：每周的进货量会影响下周的产能，而产能又会影响进货量，因此产能的增长是一个动态变化的过程，使模型更具普适性。

6.2 模型的缺点

Topsis 结合熵权法确定权重，考虑了数据的信息，但是没有结合实际情况，可能导致权重并没有良好的解释性

规划模型的求解大多采用启发式算法，求解结果存在上下波动，对于结果可复现性的条件比较苛刻。

6.3 模型的改进

在确定各指标的权重时，结合专家建议的定性权重，实现定量及定性的综合考虑。

6.4 模型的推广

传统型供应商关系存在着供应商评价标准单一、供货不稳定、合同性质单一、供应商数量大等问题，可应用本文模型，该关系转变为供应链合作伙伴型供应商关系，达到供应商评价多标准并行考虑、长期稳定紧密合作、供应商数量少而精，供货批量大且稳定。

七、 参考文献

[1] 雷星晖,尤筱玥.基于层次分析法支持决策的外包服务供应商绩效评价[J].同济大学学报(自然科学版),2014,42(11):1770-1775.
[1] 范琛,王效俐.一种综合的供应商评价方法[J].同济大学学报(自然科学版),2012,40(12):1899-1904.
[1] 黄瀚,吴晓,时磊.精益生产方式下供应商模糊综合评价[J].科技管理研究,2012,32(02):71-74.
[1] 崔智泉.浅谈高斯分布的原理和应用[J].中国校外教育,2018(16):63-64.
[1] 王宇嘉. 多目标粒子群优化算法的全局搜索策略研究[D].上海交通大学,2008.
[1] 李灿,张凤荣,朱泰峰,奉婷,安萍莉.基于熵权 TOPSIS 模型的土地利用绩效评价及关联分析[J].农业工程学报,2013,29(05):217-227.

附录

附录 1

介绍：支撑材料的文件列表

附录 2

介绍：408 家供应商前 50 家的描述性统计（具体 408 家供应商见支撑材料）

ID	最小值	最大值	总计供应	平均值	标准差	峰度
S001	0	43	231	0.9625	4.578768	72.79086
S002	0	60	309	1.2875	5.764254	89.91744
S003	0	440	14279	59.49583	87.80946	4.084237
S004	0	100	713	2.970833	15.21791	36.0389
S005	0	140	6538	27.24167	34.95101	-0.91648
S006	0	100	462	1.925	12.80766	55.97148
S007	8	200	6716	27.98333	36.30369	8.032902
S008	0	10	94	0.391667	1.249575	43.37906
S009	0	320	715	2.979167	28.22718	117.8249
S010	0	100	576	2.4	10.27195	47.54945
S011	0	10	84	0.35	1.128927	36.78616
S012	0	50	308	1.283333	6.682499	44.58699
S013	0	8	44	0.183333	0.708388	70.66083
S014	0	2	32	0.133333	0.397341	9.559265
S015	0	4210	27883	116.1792	671.6032	30.41772
S016	0	6	38	0.158333	0.659603	42.9336
S017	0	20	194	0.808333	2.824862	34.81384
S018	0	15	113	0.470833	1.293362	72.45158
S019	0	50	277	1.154167	6.105521	50.96712
S020	0	15	183	0.7625	2.009874	32.52766
S021	0	240	324	1.35	15.48553	238.9793
S022	0	50	314	1.308333	6.441622	44.86983
S023	0	400	7466	31.10833	90.97637	9.61447
S024	0	35	235	0.979167	3.772226	60.23492
S025	0	7	309	1.2875	1.567401	0.62892
S026	0	4	23	0.095833	0.413151	44.96585
S027	0	7	83	0.345833	0.933558	18.92738
S028	0	20	146	0.608333	2.293509	45.30772
S029	0	450	976	4.066667	38.74717	119.0097
S030	0	20	126	0.525	2.484976	47.56695
S031	4	250	40554	168.975	32.24539	3.381549
S032	0	7	97	0.404167	0.914312	19.06279
S033	0	5	28	0.116667	0.504098	52.13768

S034	0	200	1706	7.108333	35.91524	25.52167
S035	0	20	226	0.941667	2.531131	30.14576
S036	0	8	100	0.416667	1.023435	18.40791
S037	0	5390	67677	281.9875	846.3891	18.33522
S038	0	4	94	0.391667	0.6442	6.637294
S039	0	200	945	3.9375	24.05503	54.78993
S040	50	430	32360	134.8333	82.8012	5.531518
S041	0	55	203	0.845833	4.848886	109.0511
S042	0	50	302	1.258333	6.013925	49.06879
S043	0	100	349	1.454167	9.972241	78.7307
S044	0	3	27	0.1125	0.447973	20.48597
S045	0	20	147	0.6125	2.271308	65.43316
S046	0	4	222	0.925	0.897812	1.527016
S047	0	2300	8840	36.83333	279.8165	56.58286
S048	0	15	93	0.3875	1.292014	72.8741
S049	0	3	53	0.220833	0.538461	10.88348
S050	0	7	51	0.2125	0.703139	40.21567

附录 3

介绍： 408 家供应商前 50 家的周期描述性统计表

企业代号	最大值	最小值	平均值	方差	增长量	频率	超额率	不满足率
S001	9	1	4.9	9.29	0.222222	25	0.091332	0.676583
S002	71	1	27.3	370.01	-3.22222	71	0.218914	0.260985
S003	2371	189	1313.8	720511.6	-128.111	191	0.055022	0.15515
S004	23	1	6.4	44.64	-0.44444	33	0.055418	0.720082
S005	1913	29	691.2	479155.8	94.77778	107	0.096858	0.110442
S006	7	1	3	5.8	-0.22222	13	0.09625	0.71822
S007	754	582	694.8	1894.16	16.11111	240	0.151838	0.086268
S008	18	1	4.1	34.49	-0.22222	15	0.051786	0.671905
S009	6	1	3.1	5.89	-0.55556	19	0.019643	0.839654
S010	64	8	17	404.8	-2.22222	32	0.030082	0.666702
S011	20	4	8.5	42.45	-2.11111	32	0.16855	0.349563
S012	11	2	2.9	15.89	-0.33333	12	0.067675	0.877996
S013	9	2	4.4	11.24	-0.11111	20	0.204921	0.280397
S014	6	1	2.8	4.56	-0.66667	16	0.168333	0.423333
S015	10	1	2.8	13.96	-0.88889	15	0.075556	0.74127
S016	11	1	3.7	12.21	-0.55556	17	0.078175	0.12619

S017	65	7	13.8	371.56	-1.66667	25	0.108692	0.57331
S018	25	1	6.6	47.44	-0.77778	31	0.115606	0.603939
S019	89	1	10.9	685.89	-0.66667	12	0.053032	0.826777
S020	24	1	7.2	49.76	-1.11111	33	0.041975	0.662836
S021	25	2	8	47.8	-0.88889	42	0.183911	0.450703
S022	79	1	14	688.2	-8.77778	16	0.048077	0.791035
S023	750	32	163.1	60596.49	2.111111	133	0.049514	0.644711
S024	34	4	12.4	123.64	-1.66667	45	0.110714	0.546154
S025	51	28	37.5	45.85	0.888889	126	0.278189	0.061161
S026	6	2	2.8	5.76	-0.66667	14	0.365	0.158333
S027	20	3	8.1	41.29	-0.88889	30	0.166861	0.292074
S028	19	4	10.6	18.64	0.111111	19	0.172979	0.363624
S029	15	1	6.8	24.16	-1	29	0.039228	0.723107
S030	68	8	16.2	472.16	-1.44444	28	0.205207	0.1
S031	4894	3872	4120.7	145511.2	107.2222	240	0.026293	0.010353
S032	24	1	7.1	45.49	-0.88889	30	0.12066	0.517005
S033	12	1	3	14.4	-0.22222	17	0.063333	0.093333
S034	19	2	3	32	-0.22222	14	0.014286	0.840233
S035	23	7	14.4	15.04	0.666667	55	0.063908	0.374514
S036	27	3	12.4	68.04	-1.22222	49	0.31412	0.157094
S037	14872	132	5068.6	24868450	-243.222	157	0.022084	0.252101
S038	9	1	3.6	8.64	-0.22222	22	0.077321	0.707132
S039	195	7	38.5	3685.65	-21.6667	45	0.101163	0.43604
S040	3618	2729	3190.5	43207.85	57.66667	240	0.031271	0.039125
S041	23	2	7.1	38.49	-0.77778	28	0.112743	0.581078
S042	65	6	13.8	388.96	-1.55556	25	0.080909	0.631021
S043	11	1	3.9	11.69	-0.44444	18	0.065795	0.714865
S044	12	1	3	20.4	-1.33333	14	0.086797	0.218182
S045	11	1	3.1	10.49	-0.55556	20	0.043269	0.806464
S046	25	15	19.7	10.21	-0.44444	140	0.138774	0.235127
S047	25	2	6.6	46.04	-1	28	0.078175	0.66744
S048	25	1	6.5	46.65	-0.77778	26	0.169766	0.505967
S049	10	1	3.8	10.16	0	19	0.114683	0.528849
S050	17	2	6.6	22.84	-0.55556	29	0.393182	0.14

附录 4

介绍：matlab 编写的计算 Topsis 得分代码

```
clear
load('data02.mat')
%第 4，7，8 列需要处理
%正向化指标类型 1，1，1
%%   step1:执行 Forward（正向化）操作
[n,m] = size(X);
```

```
%n 为样本数，m 为指标数
disp(['问题一共有' num2str(n) '个供应商, ' num2str(m) '个供货特征指标'])
%accurate 来判断是否进行以下操作
accurate = input(['这' num2str(m) '个供货指标需不需要进行正向化处理,1 需要，0 不需要 ']);

if accurate == 1
    location = input('希望正向化指标所在的列，比如第 1,2,3 列需要处理，那么你需要输入[1,2,3]： ');
    %location 表示需要正向化的列
    disp('请输入需要正向化指标类型（1{极小型}， 2{中间型}， 3{区间型} ')
    difference = input('如果，需要正向化的指标为极小型那么就[1]： ');
    %difference 表示指标类型
    for i = 1 : size(location,2)  %这里需要对这些列分别处理，因此我们需要知道一共要处理的次数，即循环的次数
        %forward 进行正向化函数
        X(:,location(i)) =forward(X(:,location(i)),difference(i),location(i));
    end
end
%% step2:标准化操作
Z = X ./ repmat(sum(X.*X) .^ 0.5, n, 1);

%% step3:增加权重
% disp("增加权重请输 1，不需要请输 0")
accurate = input('确定是否增加权重，1 增加，0 不增加 ');
if accurate == 1
    accurate = input('如果需要使用熵权法，请输 1，否则请输 0： ');
    if accurate == 1
        if sum(sum(Z<0)) >0    % 如果标准化后的 Z 矩阵中存在负数，则重新对 X 进行标准化
            disp('标准化矩阵存在负数，需要重新确定标准化矩阵')
            for i = 1:n
                for j = 1:m
                    Z(i,j) = [X(i,j) - min(X(:,j))] / [max(X(:,j)) - min(X(:,j))];
                end
            end
%               disp('X 重新进行标准化得到的标准化矩阵 Z 为:  ')
%               disp(Z)
        end
        weight = entropy(Z);
    else
        disp(['请输入你确定的指标,比如三个指标输入[1,2,3]']);
        weight = input(['请以行向量的形式输入这' num2str(m) '个权重: ']);
        issafe = 0;  % 用来判断用户的输入格式是否正确
        while issafe == 0
            if abs(sum(weight) -1)<0.000001 && size(weight,1) == 1 && size(weight,2) == m  % 注意，Matlab 中浮点数的比较要小心
                issafe =1;
            else
                weight = input('有误，请重新输入权重行向量: ');
            end
        end
    end
else
    weight = ones(1,m) ./ m ; %       如果没有选择自己设置权重，则默认设置每个指标权重为
```

```
1/m
end

%% step4:优劣阶解距离法，确定得分
max_p = sum([(Z - repmat(max(Z),n,1)) .^ 2 ] .* repmat(weight,n,1) ,2) .^ 0.5;     % 结果 与最大值的距离向量
min_p = sum([(Z - repmat(min(Z),n,1)) .^ 2 ] .* repmat(weight,n,1) ,2) .^ 0.5;     % 结果与最小值的距离向量
S = min_p ./ (max_p+min_p);       % 没有标准化的结果
disp('最后的得分为：')
score = S / sum(S)
disp('------------------------------------------------------------------------')
```

附录 5

介绍：408 家供应商前 50 家得分表（具体 408 家供应商得分表见支撑材料）

企业代号	得分	企业代号	得分
S001	0.001489	S026	0.002
S002	0.002274	S027	0.001965
S003	0.003973	S028	0.001781
S004	0.001583	S029	0.001593
S005	0.003009	S030	0.002262
S006	0.001376	S031	0.00517
S007	0.004506	S032	0.001668
S008	0.001545	S033	0.002401
S009	0.001507	S034	0.001503
S010	0.001689	S035	0.002154
S011	0.001873	S036	0.002195
S012	0.001324	S037	0.004299
S013	0.001895	S038	0.001474
S014	0.001671	S039	0.001928
S015	0.001413	S040	0.004935
S016	0.002324	S041	0.00158
S017	0.001582	S042	0.001568
S018	0.001563	S043	0.001476
S019	0.001396	S044	0.002147
S020	0.001671	S045	0.001467
S021	0.001752	S046	0.003148
S022	0.001448	S047	0.001549
S023	0.002825	S048	0.001574
S024	0.001757	S049	0.001606
S025	0.003101	S050	0.002083

附录 6

介绍：50 家供应商的平均偏差表（408 家供应商的平均偏差表见支撑材料）

ID	平均偏差数	ID	平均偏差数
S001	-0.58525	S026	0.206667
S002	-0.04207	S027	-0.12521
S003	-0.10013	S028	-0.19065
S004	-0.66466	S029	-0.68388
S005	-0.01358	S030	0.105207
S006	-0.62197	S031	0.01594
S007	0.06557	S032	-0.39634
S008	-0.62012	S033	-0.03
S009	-0.82001	S034	-0.82595
S010	-0.63662	S035	-0.31061
S011	-0.18101	S036	0.157026
S012	-0.81032	S037	-0.23002
S013	-0.07548	S038	-0.62981
S014	-0.255	S039	-0.33488
S015	-0.66571	S040	-0.00785
S016	-0.04802	S041	-0.46833
S017	-0.46462	S042	-0.55011
S018	-0.48833	S043	-0.64907
S019	-0.77375	S044	-0.13139
S020	-0.62086	S045	-0.7632
S021	-0.26679	S046	-0.09635
S022	-0.74296	S047	-0.58926
S023	-0.5952	S048	-0.3362
S024	-0.43544	S049	-0.41417
S025	0.217028	S050	0.253182

附录 7

介绍：50 家供应商详细信息（408 家企业的具体见支撑材料）

编号	类别	最低产能	重要性得分
S229	A	2951.043	0.015645
S151	C	2713.601	0.011415
S348	A	2675.623	0.009007

S374	C	2423.916	0.007345
S361	C	2117.532	0.014353
S140	B	1905.329	0.012496
S201	A	1578.263	0.005326
S108	B	1453.266	0.011759
S282	A	1413.796	0.009106
S329	A	1275.585	0.008936
S275	A	1259.456	0.00909
S340	B	1231.622	0.008956
S330	B	1055.986	0.007178
S139	B	1032.365	0.008858
S131	B	995.1139	0.007523
S126	C	990.5089	0.005206
S356	C	989.9125	0.007054
S352	A	958.5212	0.006245
S308	B	949.5342	0.007102
S268	C	849.8918	0.007706
S395	A	839.2301	0.004412
S306	C	831.6046	0.007538
S194	C	680.9611	0.00677
S143	A	608.4315	0.006143
S307	A	550.4352	0.005191
S037	C	538.8911	0.004299
S284	C	417.293	0.005199
S247	C	377.8187	0.005483
S031	B	305.4005	0.00517
S365	C	268.4706	0.005145
S074	C	214.1677	0.003586
S040	B	212.769	0.004935
S367	B	193.6306	0.004843
S364	B	187.5968	0.004886
S338	B	187.443	0.004365
S346	B	154.0087	0.004783
S078	A	137.6654	0.002978
S210	C	136.2942	0.002918
S208	A	134.3823	0.002422
S055	B	129.9048	0.004718
S294	C	123.0658	0.00474
S080	C	121.482	0.004743
S154	A	111.3693	0.002205
S273	A	105.4795	0.002599
S003	C	102.3053	0.003973
S086	C	102.2853	0.004079
S218	C	100.2898	0.004687

S244	C	96.54678	0.004651
S189	A	95.11809	0.003467
S005	A	87.15534	0.003009

附录 8

介绍：用 python 编写的模拟退火算法

```
#   -*- coding: utf-8 -*-
#   python3.8.6
# 模拟退火算法   -- 因为智能算法的特点，求解结果可能有浮动 遵循 badAccept 尽量小的原则
# 惩罚函数法求解---单目标函数线性规划问题
import math              # 导入 math 库
import random            # 导入 random 库
import numpy as np   # 导入 numpy 库
#-------------------------------------------------------------
# 目标函数
def fuction1(X):
      a = X[0:6]
      b = X[6:13]
      c = X[13:18]

      fa = [1.2*i for i in a]
      fb = [1.1*i for i in b]
      fc = [i for i in c]
      fx = (sum(fa)+sum(fb)+sum(fc))
      return fx

#-----------------------------------------------------------------------------
def constraint(ai,bi,ci,obj):   #ai——每个 A 类企业 bi——每个 B 类企业 ci————每个 C 类企业
      # A:          S229          S282                S275                S348                S329
S352
      mua = [-0.012352808,   -0.286290701, -0.796383525, -0.096072404, 0.001367907,  -
0.004750679]
      sigmaa = [0.022766078, 0.246676241,  0.322169134,  0.127048152,  0.004907213,
      0.017950498]
      wa = [( ai[i]/0.6*(1+random.normalvariate(mua[i],sigmaa[i])) ) for i in range(len(ai))]
      # B:          S140              S108              S340              S139
S131              S330              S308
      mub = [-0.365593952,   0.245040572,  0.004985737,  0.002597354,      -0.1538234,      -
0.016037504,  -0.473645748]
      sigmab = [0.443062048, 0.100150966,  0.003052963,  0.002850013,  0.192837887,
      0.031840497,  0.490003479]
      wb = [( bi[i]/0.66*(1+random.normalvariate(mub[i],sigmab[i])) ) for i in range(len(bi))]
      #C:          S361          S151              S268              S374              S356
```

```
    muc = [-0.18142276,     -0.01575503,  -0.004192796, -0.107749196, -0.011274986]
    sigmac = [0.246173395, 0.029417595,  0.016292483,  0.161788595,  0.02659263]
    wc = [( ci[i]/0.72*(1+random.normalvariate(muc[i],sigmac[i])) ) for i in range(len(ci))]
    w = (min(0, sum(wa)+sum(wb)+sum(wc)-obj))**2
    return w
#----------------模拟退火-------------------------------------------------------------------
# 子程序：定义优化问题的目标函数
def my_procedure(X, nVar, mk):  # m(k)：惩罚因子，随迭代次数 k 逐渐增大
    #A: X[0] X[1] X[2] X[3] X[4] X[5]   B:X[6] X[7] X[8] X[9] X[10] X[11] X[12]   C:X[13] X[14]
X[15] X[16] X[17]
    fx = fuction1(X)
    a = X[0:6]
    b = X[6:13]
    c = X[13:18]
    w1 = constraint(a,b,c,28200)
    #修正值 25797   25040 22860 21141 21918 21523
    #20708 19623   18760
    return fx+mk*w1
#    参数设置
def ParameterSetting():
    cName = "funcOpt"
    nVar = 18                           # 18 家企业的进货量
    xMin = [0,0,0,0,0,0,
            0,0,0,0,0,0,0,
            0,0,0,0,0]
            # 每个进货量的下限
    # My_xMax = [3057.094572,3253.32477,4182.704389,1824.711446,1280.121515,980.6785272,
    #
6560.540944,2033.628269,1411.054601,1259.774953,1509.512772,1044.324912,2862.096351,
    #            4252.848459,2216.085917,1257.269667,1364.3358,1028.422183]
    xMax = [6000,6000,6000,6000,6000,6000
            ,6000,6000,6000,6000,6000,6000
            ,6000,6000,6000,6000,6000,6000]
             # 每个进货量的上限
    InitialT = 2000.0         # 设置初始温度    大 易找到全局最优解，注意计算量
    FinalT   = 0.1              # 设置终止温度   小 越精细
    alfa     = 0.96             # 设置降温系数   一般 0.5 到 0.99，越大放慢温度衰减的速度，提
高迭代次数。
    meanMarkov = 1000          # Markov 链长度，也即内循环运行次数
    step     = 0.3             # 设置步长，可以设为固定值或逐渐缩小
    return cName, nVar, xMin, xMax, InitialT, FinalT, alfa, meanMarkov, step
# 模拟退火算法
def SSA(nVar,xMin,xMax,InitialT,FinalT,alfa,meanMarkov,step):
    randseed = random.randint(1, 100)
    random.seed(randseed)  # 定义种子
    # ====== 随机产生初始解 =====
    xInitial = np.zeros((nVar))
    for v in range(nVar):
        xInitial[v] = random.uniform(xMin[v], xMax[v])
    fxInitial = my_procedure(xInitial, nVar, 1)
    # ====== 整体算法初始化 ======
    xNew = np.zeros((nVar))
    xNow = np.zeros((nVar))
    xBest = np.zeros((nVar))
    xNow[:]   = xInitial[:]             # 当前情况下的初始解设置为当前解
```

```
    xBest[:] = xInitial[:]              # 当前情况下的解设置为最优解
    fxNow  = fxInitial                  # 设置为当前值
    fxBest = fxInitial                  # 设置为最优值
    print('x_Initial:{:.6f},{:.6f},\tf(x_Initial):{:.6f}'.format(xInitial[0], xInitial[1], fxInitial))

    recordIter = []                     # Initialization 外循环次数
    recordFxNow = []                    # Initialization 当前解的目标函数值
    recordFxBest = []                   # Initialization 最佳解的目标函数值
    recordPBad = []                     # Initialization 劣质解的接受概率
    kIter = 0                           # 外循环迭代次数，温度状态数
    totalMar = 0                        # 总计 Markov 链长度
    totalImprove = 0                    # fxBest 改善次数
    nMarkov = meanMarkov                # 固定长度 Markov 链
    # ====== 优化 ======
    # 进行外循环，当前温度降低到终止温度
    tNow = InitialT
    while tNow >= FinalT:
        # 在当前温度下，进行状态转移直到热平衡
        kBetter,kBadAccept,kBadRefuse = 0,0,0

        # 进行内循环
        for k in range(nMarkov):
            totalMar += 1

            # 产生新解的过程
            xNew[:] = xNow[:]
            v = random.randint(0, nVar-1)    # 产生范围之间的随机数
            xNew[v] = xNow[v] + step * (xMax[v]-xMin[v]) * random.normalvariate(0, 1) #正太
分布
            xNew[v] = max(min(xNew[v], xMax[v]), xMin[v])    #保证解的上下值
            # ---计算函数值
            fxNew = my_procedure(xNew, nVar, kIter)
            deltaE = fxNew - fxNow
            if fxNew < fxNow:   # 求解更新更优解：
                accept = True
                kBetter += 1
            else:
                pAccept = math.exp(-deltaE / tNow)
                if pAccept > random.random():
                    accept = True
                    kBadAccept += 1
                else:
                    accept = False
                    kBadRefuse += 1
            # 冻结新解
            if accept == True:   # 如果接受，跟新当前解
                xNow[:] = xNew[:]
                fxNow = fxNew
                if fxNew < fxBest:   # 如果函数值好于最优解，跟新最优解
                    fxBest = fxNew
                    xBest[:] = xNew[:]
                    totalImprove += 1
                    step = step*0.99   # 可变搜索步长步，逐减小搜索范围，提高搜索精度
        # 保存并输出
```

```
            pBadAccept = kBadAccept / (kBadAccept + kBadRefuse)
            recordIter.append(kIter)
            recordFxNow.append(round(fxNow, 4))
            recordFxBest.append(round(fxBest, 4))
            recordPBad.append(round(pBadAccept, 4))
            if kIter%10 == 0:
                print('i:{},t(i):{:.2f}, badAccept:{:.6f}, f(x)_best:{:.6f}'.\
                      format(kIter, tNow, pBadAccept, fxBest))

            # 定义我的降温曲线：T(k)=alfa*T(k-1)
            tNow = tNow * alfa
            kIter = kIter + 1
            fxBest = my_procedure(xBest, nVar, kIter)   # 惩罚因子回逐渐扩大，增加
            # ====== 结束 ======
        print('improve:{:d}'.format(totalImprove))
        return kIter,xBest,fxBest,fxNow,recordIter,recordFxNow,recordFxBest,recordPBad

# 结果校验与输出
def ResultOutput(cName,nVar,xBest,fxBest,kIter,recordFxNow,recordFxBest,recordPBad,recordIter):
        fxCheck = my_procedure(xBest, nVar, kIter)
        print("\nOptimization by simulated annealing algorithm:")
        print()
        for i in range(nVar):
            # print('\tx[{}] = {:.6f}'.format(i,xBest[i]))
            print(xBest[i],end=" ")
        print()
        print('\n\tf(x):{:.6f}'.format(my_procedure(xBest,nVar,0)))
        return
# main 函数   准备完毕
def main():
        [cName, nVar, xMin, xMax, InitialT, FinalT, alfa, meanMarkov, step] = ParameterSetting()
        [kIter,xBest,fxBest,fxNow,recordIter,recordFxNow,recordFxBest,recordPBad]                    =
SSA(nVar,xMin,xMax,InitialT,FinalT,alfa,meanMarkov,step)
        ResultOutput(cName,
nVar,xBest,fxBest,kIter,recordFxNow,recordFxBest,recordPBad,recordIter)
if __name__ == '__main__':
        main()
```

附录 9
介绍：问题 2 前八周采购计划表（具体见支撑材料）

供应商	第一周	第二周	第三周	第四周	第五周	第六周	第七周	第八周
S229	2101.249	2335.862	488.2107	2403.567	123.5124	134.6155	11.07748	964.3423
S361	200.9461	563.4926	202.6354	2932.232	1511.124	245.3755	1671.33	2216.478
S140	112.9069	2140.962	4881.097	595.3657	2117.651	962.6475	276.8636	1942.812
S108	2461.781	1337.984	1475.622	1934.591	859.3511	701.1895	574.5162	885.1233
S151	2652.743	129.8032	180.3069	1328.877	727.5366	1666.038	377.8425	1313.68
S282	2953.494	129.8032	1194.527	1338.025	40.185	317.3342	4029.647	1471.505
S275	339.3119	650.8016	681.4856	357.142	231.0266	1192.248	501.5972	222.5678
S348	198.9135	1270.621	178.8553	1093.024	106.2425	1110.985	819.7484	189.3288
S340	135.4224	520.3828	2084.647	1770.245	2300.729	509.1976	164.8139	1148.601
S329	2029.91	1159.138	219.8633	369.4707	766.8289	387.5141	4733.691	140.663
S139	569.5618	2350.069	323.4304	161.7494	2860.887	2335.517	58.53619	1946.231
S268	1216.781	1009.586	164.9763	406.2438	633.7458	1062.619	326.5109	1345.765
S131	178.9964	35.96003	54.47749	1191.233	409.9807	476.586	2474.549	116.3793
S374	3393.391	1554.694	2019.382	1626.886	14.13799	4253.455	55.21654	410.1323
S330	263.6932	1461.228	509.7916	5.561382	141.2401	2488.444	695.6245	490.4937
S308	46.9315	431.9603	1184.889	393.6219	1421.524	266.4418	28.41557	267.8513
S356	291.8056	913.9478	374.3841	1106.499	3496.752	1588.325	1540.828	2974.012
S352	21.10182	420.4966	1660.284	284.8176	1835.103	862.5421	454.1862	2165.97

附录 10

介绍：问题 2、3 用 python 编写的蒙特卡洛算法

```
import random
def Attrition_rate(supply):
    #1.计算损耗率
    Transshipment = []   # 8 家转运公司每一周的损耗率
    muc = [1.904769167, 0.921370417, 0.552594017, 1.677088235, 2.906491566, 0.863002778, 2.119170711, 1.042585222]
    sigmac = [0.655931509, 0.479449258, 0.495973992, 1.872677366, 1.952098818, 1.264809417, 1.421022552, 1.112991683]
    for i in range(8):
        speed = random.normalvariate(muc[i], sigmac[i])
        # 满足正数的正态分布
        if (speed < 0):
            Transshipment.append(0.00000001)   # !!!!!!!!!!!!!!0??????????????????
        else:
            Transshipment.append(speed)
    print("每一个转运商的损耗率",Transshipment)

    last_consume=[0 for i in range(8)]   #最后每个转运商的存放货物多少
    last_loss = 100000000   # 最终的最小损耗，默认初值为 1 亿
    last_location = [0 for i in range(18)]   # 最后 18 家企业选择的转运商

    # 运用蒙特卡洛模拟，计算好多次总损耗，每一次比较上一次的结果，如果小了则覆盖，并保存选择
```

```
    total_numer =100000   #总的运行次数
    total_loss = [0 for i in range(total_numer)]              # 总损耗
    location = [[0] * 18 for i in range(total_numer)]   # total_numer*18 的矩阵，18 家企业选择的转运商
    consume = [[0] * 8 for i in range(total_numer)]   #每次转运商运输的货物重量
    for j in range(total_numer):
        for i in range(18):   # 18 家供应商从第一个开始
            choose = random.randint(0, 7)   # 这第 i 家供应商选择的转运商  (int)
            total_loss[j] = total_loss[j] + supply[i] * Transshipment[choose] * 0.01
            consume[j][(choose)] = consume[j][choose] + supply[i]
            location[j][i] = choose

    index = 0
    for hang in consume:
        for j in range(8):
            if (hang[j] > 6000):
                total_loss[index] = 100000000000
        index = index + 1

    for i in range(total_numer):
        if (total_loss[i] < last_loss):
            last_loss = total_loss[i]
            last_location = location[i]
            last_consume = consume[i]
            # print(last_loss)
            # print(last_location)
            # print(last_consume)
    return (last_loss,"18 家企业选择的转运商", last_location,"每次转运商运输的货物重量",
last_consume)
```

附录

介绍：问题 2 惩罚函数法的模拟退火算法(Python)

(1) main.py

```
#encoding: utf-8
#python 3.8.6
"""
############多目标函数规划的粒子群算法#################
也就是两个函数，相同的输入，不同的函数形式，找到函数最优输出值   最优输入值
需要多次测试，取稳定值
求解结果，可能会用浮动，遵循目标函数最优，来选择进货量
"""
import numpy as np
from mopso import *
import os
def main():
    #设置超参
    w = 0.9   #惯性权重因子   initialization-0.8
```

```
    c1 = 1.49 #个体学习因子   initialization-0.1
    c2 = 1.49 #社会学习因子   initialization 0.1
    particals = 500 #granule 数目  initialization-100
    cycle_ = 2000   #整体迭代次数              initialization0-800
    mesh_div = 100 #网格等分数量   initialization-100
    thresh = 300 #外部存档阀值       initialization-300
    min_ = np.array([0,0,0,0,0,0,    0,0,0,0,0,0,     0,0.23,0,0,0,0])    #18 个自变量的下界
    max_  =  np.array([6000,6000,6000,6000,6000,6000,     6000,6000,6000,6000,6000,6000,6000,
6000,6000,6000,6000,6000])   #18 个自变量的上界
    mopso_ = Mopso(particals,w,c1,c2,max_,min_,thresh,mesh_div)   #granule 的 instantiation
    pareto_in,pareto_fitness = mopso_.done(cycle_)                    #经过 cycle_轮迭代后，pareto
边界 granule
    print(pareto_in)           #最优值
    print(pareto_fitness)   #get-目标函数的最优值
    if os.path.exists('./img_txt') == False:
        os.makedirs('./img_txt')
        print('创建文件夹 img_txt:保存 granule 群每一次迭代的图片' )
    np.savetxt("./img_txt/pareto_in.txt",pareto_in)                  #保存得到最优解
    np.savetxt("./img_txt/pareto_fitness.txt",pareto_fitness) #print 的目标函数的最优值
    print("\n","pareto 边界的解保存于：/img_txt/pareto_in.txt")
    print("pareto 边界的适应值保存于：/img_txt/pareto_fitness.txt")
    print("\n","迭代结束,over")
if __name__ == "__main__":
    main()
```

(2) mopso.py

```
#encoding: utf-8
#python 3.8.6
import init                                  #导入 init
import update                          #导入 update
from fitness_funs import *          #导入 fitness_funs

#-----------------------------------------------------------------------------------------
import numpy as np                 #导入 numpy
class Mopso:     #粒子群算法
    def __init__(self,particals,w,c1,c2,max_,min_,thresh,mesh_div=10):
        self.w,self.c1,self.c2 = w,c1,c2
        self.mesh_div = mesh_div
        self.particals = particals
        self.thresh = thresh
        self.max_ = max_
        self.min_ = min_
        self.max_v = (max_-min_)*0.05   #下界
        self.min_v = (max_-min_)*0.05*(-1) #上界
```

```
#-------------------------------------------------------------------------------
    #计算值
    def evaluation_fitness(self): #gogo
        fitness_curr = []
        for i in range((self.in_).shape[0]):
            fitness_curr.append(fitness_(self.in_[i]))
        self.fitness_ = np.array(fitness_curr) #值
#-------------------------------------------------------------------------------
    def initialize(self):    #初始化
        #initializationgranule 解
        self.in_ = init.init_designparams(self.particals,self.min_,self.max_)
        #initializationgranule 速度
        self.v_ = init.init_v(self.particals,self.min_v,self.max_v)
        #计算值
        self.evaluation_fitness()
        #initialization 个体最优
        self.in_p,self.fitness_p = init.init_pbest(self.in_,self.fitness_)
        #initialization 外部存档
        self.archive_in,self.archive_fitness = init.init_archive(self.in_,self.fitness_)
        #initialization 社会最优
        self.in_g,self.fitness_g                                                    =
init.init_gbest(self.archive_in,self.archive_fitness,self.mesh_div,self.min_,self.max_,self.particals)
#-------------------------------------------------------------------------------------------------------------------
----------------
    def update_(self):    #跟新
        #更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
        self.v_                                                                     =
update.update_v(self.v_,self.min_v,self.max_v,self.in_,self.in_p,self.in_g,self.w,self.c1,self.c2)
        self.in_ = update.update_in(self.in_,self.v_,self.min_,self.max_)
        self.evaluation_fitness()
        self.in_p,self.fitness_p = update.update_pbest(self.in_,self.fitness_,self.in_p,self.fitness_p)
        self.archive_in,self.archive_fitness                                        =
update.update_archive(self.in_,self.fitness_,self.archive_in,self.archive_fitness,self.thresh,self.mesh_d
iv,self.min_,self.max_,self.particals)
        self.in_g,self.fitness_g                                                    =
update.update_gbest(self.archive_in,self.archive_fitness,self.mesh_div,self.min_,self.max_,self.partica
ls)
#-------------------------------------------------------------------------------------------------------------------
---------------------------
    def done(self,cycle_):    #完成
        self.initialize()
        for i in range(cycle_):   #循环中 i
            self.update_()
        return self.archive_in,self.archive_fitness
```

(3) pareto.py

```
#encoding: utf-8
#python 3.8.6
import numpy as np   #导入 numpy
#-----------------------------------------------------
def compare_ (fitness_curr,fitness_ref):
#判断 fitness_curr 是否可以被 fitness_ref 完全支配
    for i in range(len(fitness_curr)):
        if fitness_curr[i] < fitness_ref[i]: #  当前  和  跟新
            return True
    return False
#-----------------------------------------------------
#  进行判断
def judge_(fitness_curr,fitness_data,cursor):
#当前 granule 的值 fitness_curr 与 dataset fitness_data 进行比较，判断是否为非劣解
    for i in range(len(fitness_data)):
        if i == cursor:       #如果相等
            continue
        #如果 dataset  中存在一个 granule 可以完全支配当前解，说明当前解为劣解，返回 False
        if compare_(fitness_curr,fitness_data[i]) == False:   #  如果为 False
            return False
    return True
#-----------------------------------------------------
#更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
# granule
class Pareto_:
    def __init__(self,in_data,fitness_data): #全局变量
        self.in_data = in_data   #granule 群解信息
        self.fitness_data = fitness_data #granule 群值信息
        self.cursor = -1 #initialization 游标位置
        self.len_ = in_data.shape[0] #granule 群的数量
        self.bad_num = 0 #非解数
    def next(self):
        #将下表的数值前移一步，并返回所在下表位的 granule 解、granule 值
        self.cursor = self.cursor+1
        return self.in_data[self.cursor],self.fitness_data[self.cursor]
    def hasNext(self):
        #if 来确定是否所有 granule 都检查更好
        return self.len_ > self.cursor + 1 + self.bad_num
    def remove(self):
        #将非优解从 dataset  删除，防止反复进行 2 比较。
        self.fitness_data = np.delete(self.fitness_data,self.cursor,axis=0)
        self.in_data = np.delete(self.in_data,self.cursor,axis=0)
```

```
            #下表回退一步
            self.cursor = self.cursor-1
            #加 1
            self.bad_num = self.bad_num + 1
        def pareto(self):
            while(self.hasNext()):
                #获取当前位置的 granule 信息
                in_curr,fitness_curr = self.next()
                #判断当前 granule 是否 pareto 最优
                if judge_(fitness_curr,self.fitness_data,self.cursor) == False : #如果为 False
                    self.remove()
            return self.in_data,self.fitness_data

#更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
#-------------------------------------------------------------------------------------------
```

附录

介绍：问题 2 惩罚函数法的模拟退火算法(Python)

```
#   -*- coding: utf-8 -*-
#   python3.8.6
# 模拟退火算法   -- 因为智能算法的特点，求解结果可能有浮动 遵循 badAccept 尽量小的原则
# 惩罚函数法求解---单目标函数线性规划问题
import math              # 导入 math 库
import random            # 导入 random 库
import numpy as np   # 导入 numpy 库
#------------------------------------------------------------
# 目标函数
def fuction1(X):
    a = X[0:6]
    b = X[6:13]
    c = X[13:18]

    fa = [1.2*i for i in a]
    fb = [1.1*i for i in b]
    fc = [i for i in c]
    fx = (sum(fa)+sum(fb)+sum(fc))
    return fx

#-----------------------------------------------------------------------------
def constraint(ai,bi,ci,obj):   #ai——每个 A 类企业 bi——每个 B 类企业 ci————每个 C 类企业
    # A:          S229          S282              S275              S348              S329          S352
```

```
    mua = [-0.012352808,   -0.286290701, -0.796383525, -0.096072404, 0.001367907,  -
0.004750679]
    sigmaa = [0.022766078, 0.246676241,  0.322169134,  0.127048152,  0.004907213,
    0.017950498]
    wa = [( ai[i]/0.6*(1+random.normalvariate(mua[i],sigmaa[i])) ) for i in range(len(ai))]
    # B:          S140                S108              S340              S139
S131              S330              S308
    mub = [-0.365593952,   0.245040572,  0.004985737,  0.002597354,      -0.1538234,     -
0.016037504,  -0.473645748]
    sigmab = [0.443062048, 0.100150966,  0.003052963,  0.002850013,  0.192837887,
    0.031840497,  0.490003479]
    wb = [( bi[i]/0.66*(1+random.normalvariate(mub[i],sigmab[i])) ) for i in range(len(bi))]
    #C:          S361          S151              S268              S374              S356
    muc = [-0.18142276,     -0.01575503,   -0.004192796, -0.107749196, -0.011274986]
    sigmac = [0.246173395, 0.029417595,  0.016292483,  0.161788595,  0.02659263]
    wc = [( ci[i]/0.72*(1+random.normalvariate(muc[i],sigmac[i])) ) for i in range(len(ci))]
    w = (min(0, sum(wa)+sum(wb)+sum(wc)-obj))**2
    return w
#-----------------模拟退火-------------------------------------------------------------------
# 子程序：定义优化问题的目标函数
def my_procedure(X, nVar, mk):   # m(k)：惩罚因子，随迭代次数 k 逐渐增大
    #A: X[0] X[1] X[2] X[3] X[4] X[5]   B:X[6] X[7] X[8] X[9] X[10] X[11] X[12]   C:X[13] X[14]
X[15] X[16] X[17]
    fx = fuction1(X)
    a = X[0:6]
    b = X[6:13]
    c = X[13:18]
    w1 = constraint(a,b,c,28200)
    #修正值  25797   25040 22860 21141 21918 21523
    #20708 19623   18760
    return fx+mk*w1
#    参数设置
def ParameterSetting():
    cName = "funcOpt"
    nVar = 18                          # 18 家企业的进货量
    xMin = [0,0,0,0,0,0,
            0,0,0,0,0,0,0,
            0,0,0,0,0]
            # 每个进货量的下限
    # My_xMax = [3057.094572,3253.32477,4182.704389,1824.711446,1280.121515,980.6785272,
    #
6560.540944,2033.628269,1411.054601,1259.774953,1509.512772,1044.324912,2862.096351,
    #           4252.848459,2216.085917,1257.269667,1364.3358,1028.422183]
    xMax = [6000,6000,6000,6000,6000,6000
```

```
                ,6000,6000,6000,6000,6000,6000
                ,6000,6000,6000,6000,6000,6000]
                 # 每个进货量的上限
    InitialT = 2000.0          # 设置初始温度     大 易找到全局最优解，注意计算量
    FinalT   = 0.1               # 设置终止温度    小 越精细
    alfa     = 0.96              # 设置降温系数    一般 0.5 到 0.99，越大放慢温度衰减的速度，提高迭代次数。
    meanMarkov = 1000            # Markov 链长度，也即内循环运行次数
    step     = 0.3               # 设置步长，可以设为固定值或逐渐缩小
    return cName, nVar, xMin, xMax, InitialT, FinalT, alfa, meanMarkov, step
# 模拟退火算法
def SSA(nVar,xMin,xMax,InitialT,FinalT,alfa,meanMarkov,step):
    randseed = random.randint(1, 100)
    random.seed(randseed)   # 定义种子
    # ====== 随机产生初始解 =====
    xInitial = np.zeros((nVar))
    for v in range(nVar):
        xInitial[v] = random.uniform(xMin[v], xMax[v])
    fxInitial = my_procedure(xInitial, nVar, 1)
    # ====== 整体算法初始化 ======
    xNew = np.zeros((nVar))
    xNow = np.zeros((nVar))
    xBest = np.zeros((nVar))
    xNow[:]   = xInitial[:]              # 当前情况下的初始解设置为当前解
    xBest[:] = xInitial[:]              # 当前情况下的解设置为最优解
    fxNow    = fxInitial                  # 设置为当前值
    fxBest = fxInitial                   # 设置为最优值
    print('x_Initial:{:.6f},{:.6f},\tf(x_Initial):{:.6f}'.format(xInitial[0], xInitial[1], fxInitial))

    recordIter = []                      # Initialization 外循环次数
    recordFxNow = []                     # Initialization 当前解的目标函数值
    recordFxBest = []                    # Initialization 最佳解的目标函数值
    recordPBad = []                      # Initialization 劣质解的接受概率
    kIter = 0                            # 外循环迭代次数，温度状态数
    totalMar = 0                         # 总计 Markov 链长度
    totalImprove = 0                     # fxBest 改善次数
    nMarkov = meanMarkov                 # 固定长度 Markov 链
    # ====== 优化 ======
    # 进行外循环，当前温度降低到终止温度
    tNow = InitialT
    while tNow >= FinalT:
        # 在当前温度下，进行状态转移直到热平衡
        kBetter,kBadAccept,kBadRefuse = 0,0,0
```

```
        # 进行内循环
        for k in range(nMarkov):
            totalMar += 1

            # 产生新解的过程
            xNew[:] = xNow[:]
            v = random.randint(0, nVar-1)     # 产生范围之间的随机数
            xNew[v] = xNow[v] + step * (xMax[v]-xMin[v]) * random.normalvariate(0, 1) #正太
分布
            xNew[v] = max(min(xNew[v], xMax[v]), xMin[v])     #保证解的上下值
            # ---计算函数值
            fxNew = my_procedure(xNew, nVar, kIter)
            deltaE = fxNew - fxNow
            if fxNew < fxNow:   # 求解更新更优解：
                accept = True
                kBetter += 1
            else:
                pAccept = math.exp(-deltaE / tNow)
                if pAccept > random.random():
                    accept = True
                    kBadAccept += 1
                else:
                    accept = False
                    kBadRefuse += 1
            # 冻结新解
            if accept == True:   # 如果接受，跟新当前解
                xNow[:] = xNew[:]
                fxNow = fxNew
                if fxNew < fxBest:   # 如果函数值好于最优解，跟新最优解
                    fxBest = fxNew
                    xBest[:] = xNew[:]
                    totalImprove += 1
                    step = step*0.99   # 可变搜索步长步，逐减小搜索范围，提高搜索精度
        # 保存并输出
        pBadAccept = kBadAccept / (kBadAccept + kBadRefuse)
        recordIter.append(kIter)
        recordFxNow.append(round(fxNow, 4))
        recordFxBest.append(round(fxBest, 4))
        recordPBad.append(round(pBadAccept, 4))
        if kIter%10 == 0:
            print('i:{},t(i):{:.2f}, badAccept:{:.6f}, f(x)_best:{:.6f}'.\
                format(kIter, tNow, pBadAccept, fxBest))

        # 定义我的降温曲线：T(k)=alfa*T(k-1)
```

```
            tNow = tNow * alfa
            kIter = kIter + 1
            fxBest = my_procedure(xBest, nVar, kIter)   # 惩罚因子回逐渐扩大，增加
            # ====== 结束 ======
        print('improve:{:d}'.format(totalImprove))
        return kIter,xBest,fxBest,fxNow,recordIter,recordFxNow,recordFxBest,recordPBad

# 结果校验与输出
def ResultOutput(cName,nVar,xBest,fxBest,kIter,recordFxNow,recordFxBest,recordPBad,recordIter):
        fxCheck = my_procedure(xBest, nVar, kIter)
        print("\nOptimization by simulated annealing algorithm:")
        print()
        for i in range(nVar):
            # print('\tx[{}] = {:.6f}'.format(i,xBest[i]))
            print(xBest[i],end=" ")
        print()
        print('\n\tf(x):{:.6f}'.format(my_procedure(xBest,nVar,0)))
        return
# main 函数   准备完毕
def main():
        [cName, nVar, xMin, xMax, InitialT, FinalT, alfa, meanMarkov, step] = ParameterSetting()
        [kIter,xBest,fxBest,fxNow,recordIter,recordFxNow,recordFxBest,recordPBad]                    =
SSA(nVar,xMin,xMax,InitialT,FinalT,alfa,meanMarkov,step)
        ResultOutput(cName,
nVar,xBest,fxBest,kIter,recordFxNow,recordFxBest,recordPBad,recordIter)
if __name__ == '__main__':
        main()
```

附录 11

介绍：问题 3 多目标粒子群算法的目标函数(Python)

```
# encoding：UTF-8
# python 3.8.6
#------------------------------目标函数---------------------------------
import random
def Purchase(X,obj):   #进货量目标函数
        ai = X[0:6]
        bi = X[6:13]
        ci = X[13:18]

        # A:            S229          S282                S275              S348              S329
S352
        mua = [-0.012352808,    -0.286290701, -0.796383525, -0.096072404, 0.001367907,  -
0.004750679]
```

```
    sigmaa = [0.022766078, 0.246676241,  0.322169134,  0.127048152,  0.004907213,
    0.017950498]
    wa = [( ai[i]/0.6*(1+random.normalvariate(mua[i],sigmaa[i])) ) for i in range(len(ai))]

    # B:          S140                  S108              S340              S139
S131              S330              S308
    mub = [-0.365593952,   0.245040572,  0.004985737,  0.002597354,      -0.1538234,     -
0.016037504,  -0.473645748]
    sigmab = [0.443062048, 0.100150966,  0.003052963,  0.002850013,  0.192837887,
    0.031840497,  0.490003479]
    wb = [( bi[i]/0.66*(1+random.normalvariate(mub[i],sigmab[i])) ) for i in range(len(bi))]
    #C:          S361          S151              S268              S374              S356
    muc = [-0.18142276,     -0.01575503,  -0.004192796, -0.107749196, -0.011274986]
    sigmac = [0.246173395, 0.029417595,  0.016292483,  0.161788595,  0.02659263]
    wc = [( ci[i]/0.72*(1+random.normalvariate(muc[i],sigmac[i])) ) for i in range(len(ci))]
    w = (sum(wa)+sum(wb)+sum(wc)-obj)**2
    return w

def minC(X):    #A 类尽量多  C 类尽量的少  比例尽量小
    ai = X[0:6]
    bi = X[6:13]
    ci = X[13:18]
    #w = (sum(ai)-sum(ci))**2
    w = sum(ci)/sum(ai)
    return w
```

附录 12

介绍：问题 3 主要的多目标粒子群算法代码(完整代码请参考支撑材料)

(4) main.py

```
#encoding: utf-8
#python 3.8.6
"""
###########多目标函数规划的粒子群算法################
也就是两个函数，相同的输入，不同的函数形式，找到函数最优输出值   最优输入值
需要多次测试，取稳定值
求解结果，可能会用浮动，遵循目标函数最优，来选择进货量
"""
import numpy as np
from mopso import *
import os
def main():
    #设置超参
    w = 0.9   #惯性权重因子   initialization-0.8
```

```
    c1 = 1.49 #个体学习因子   initialization-0.1
    c2 = 1.49 #社会学习因子   initialization 0.1
    particals = 500 #granule 数目  initialization-100
    cycle_ = 2000   #整体迭代次数              initialization0-800
    mesh_div = 100 #网格等分数量   initialization-100
    thresh = 300 #外部存档阀值       initialization-300
    min_ = np.array([0,0,0,0,0,0,    0,0,0,0,0,0,     0,0.23,0,0,0,0])    #18 个自变量的下界
    max_  =  np.array([6000,6000,6000,6000,6000,6000,     6000,6000,6000,6000,6000,6000,6000,
6000,6000,6000,6000,6000])   #18 个自变量的上界
    mopso_ = Mopso(particals,w,c1,c2,max_,min_,thresh,mesh_div)   #granule 的 instantiation
    pareto_in,pareto_fitness = mopso_.done(cycle_)                    #经过 cycle_轮迭代后，pareto
边界 granule
    print(pareto_in)           #最优值
    print(pareto_fitness)   #get-目标函数的最优值
    if os.path.exists('./img_txt') == False:
        os.makedirs('./img_txt')
        print('创建文件夹 img_txt:保存 granule 群每一次迭代的图片' )
    np.savetxt("./img_txt/pareto_in.txt",pareto_in)                  #保存得到最优解
    np.savetxt("./img_txt/pareto_fitness.txt",pareto_fitness) #print 的目标函数的最优值
    print("\n","pareto 边界的解保存于：/img_txt/pareto_in.txt")
    print("pareto 边界的适应值保存于：/img_txt/pareto_fitness.txt")
    print("\n","迭代结束,over")
if __name__ == "__main__":
    main()
```

(5) mopso.py

```
#encoding: utf-8
#python 3.8.6
import init                                  #导入 init
import update                          #导入 update
from fitness_funs import *          #导入 fitness_funs

#-----------------------------------------------------------------------------------------
import numpy as np                 #导入 numpy
class Mopso:     #粒子群算法
    def __init__(self,particals,w,c1,c2,max_,min_,thresh,mesh_div=10):
        self.w,self.c1,self.c2 = w,c1,c2
        self.mesh_div = mesh_div
        self.particals = particals
        self.thresh = thresh
        self.max_ = max_
        self.min_ = min_
        self.max_v = (max_-min_)*0.05   #下界
        self.min_v = (max_-min_)*0.05*(-1) #上界
```

```
#-------------------------------------------------------------------------------
    #计算值
    def evaluation_fitness(self): #gogo
        fitness_curr = []
        for i in range((self.in_).shape[0]):
            fitness_curr.append(fitness_(self.in_[i]))
        self.fitness_ = np.array(fitness_curr) #值
#-------------------------------------------------------------------------------
    def initialize(self):    #初始化
        #initializationgranule 解
        self.in_ = init.init_designparams(self.particals,self.min_,self.max_)
        #initializationgranule 速度
        self.v_ = init.init_v(self.particals,self.min_v,self.max_v)
        #计算值
        self.evaluation_fitness()
        #initialization 个体最优
        self.in_p,self.fitness_p = init.init_pbest(self.in_,self.fitness_)
        #initialization 外部存档
        self.archive_in,self.archive_fitness = init.init_archive(self.in_,self.fitness_)
        #initialization 社会最优
        self.in_g,self.fitness_g                                                    =
init.init_gbest(self.archive_in,self.archive_fitness,self.mesh_div,self.min_,self.max_,self.particals)
#-------------------------------------------------------------------------------------------------------------------
----------------
    def update_(self):    #跟新
        #更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
        self.v_                                                                     =
update.update_v(self.v_,self.min_v,self.max_v,self.in_,self.in_p,self.in_g,self.w,self.c1,self.c2)
        self.in_ = update.update_in(self.in_,self.v_,self.min_,self.max_)
        self.evaluation_fitness()
        self.in_p,self.fitness_p = update.update_pbest(self.in_,self.fitness_,self.in_p,self.fitness_p)
        self.archive_in,self.archive_fitness                                        =
update.update_archive(self.in_,self.fitness_,self.archive_in,self.archive_fitness,self.thresh,self.mesh_d
iv,self.min_,self.max_,self.particals)
        self.in_g,self.fitness_g                                                    =
update.update_gbest(self.archive_in,self.archive_fitness,self.mesh_div,self.min_,self.max_,self.partica
ls)
#-------------------------------------------------------------------------------------------------------------------
---------------------------
    def done(self,cycle_):    #完成
        self.initialize()
        for i in range(cycle_):   #循环中 i
            self.update_()
        return self.archive_in,self.archive_fitness
```

(6) pareto.py

```
#encoding: utf-8
#python 3.8.6
import numpy as np   #导入 numpy
#-----------------------------------------------------
def compare_ (fitness_curr,fitness_ref):
#判断 fitness_curr 是否可以被 fitness_ref 完全支配
    for i in range(len(fitness_curr)):
        if fitness_curr[i] < fitness_ref[i]: #  当前  和  跟新
            return True
    return False
#-----------------------------------------------------
#  进行判断
def judge_(fitness_curr,fitness_data,cursor):
#当前 granule 的值 fitness_curr 与 dataset fitness_data 进行比较，判断是否为非劣解
    for i in range(len(fitness_data)):
        if i == cursor:       #如果相等
            continue
        #如果 dataset  中存在一个 granule 可以完全支配当前解，说明当前解为劣解，返回 False
        if compare_(fitness_curr,fitness_data[i]) == False:   #  如果为 False
            return False
    return True
#-----------------------------------------------------
#更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
# granule
class Pareto_:
    def __init__(self,in_data,fitness_data): #全局变量
        self.in_data = in_data   #granule 群解信息
        self.fitness_data = fitness_data #granule 群值信息
        self.cursor = -1 #initialization 游标位置
        self.len_ = in_data.shape[0] #granule 群的数量
        self.bad_num = 0 #非解数
    def next(self):
        #将下表的数值前移一步，并返回所在下表位的 granule 解、granule 值
        self.cursor = self.cursor+1
        return self.in_data[self.cursor],self.fitness_data[self.cursor]
    def hasNext(self):
        #if 来确定是否所有 granule 都检查更好
        return self.len_ > self.cursor + 1 + self.bad_num
    def remove(self):
        #将非优解从 dataset  删除，防止反复进行 2 比较。
        self.fitness_data = np.delete(self.fitness_data,self.cursor,axis=0)
        self.in_data = np.delete(self.in_data,self.cursor,axis=0)
```

```
            #下表回退一步
            self.cursor = self.cursor-1
            #加 1
            self.bad_num = self.bad_num + 1
        def pareto(self):
            while(self.hasNext()):
                #获取当前位置的 granule 信息
                in_curr,fitness_curr = self.next()
                #判断当前 granule 是否 pareto 最优
                if judge_(fitness_curr,self.fitness_data,self.cursor) == False : #如果为 False
                    self.remove()
            return self.in_data,self.fitness_data

#更新 granule 解、granule 速度、值、个体学习最优、外部存储、社会最优
#-------------------------------------------------------------------------------------------
```