多工序生产过程下的企业最优决策

摘要

本文将企业生产过程中的各项决策定义为多个 0-1 变量，将多工序生产过程的决策问题转化为求变量最优解的优化问题。在假设成品售价不变的前提下，最小化出售单位成品的实际成本，从而使企业利润达到最大化，此时的决策方案即为理性企业的最优方案。对于总体次品率非固定的情况，先基于样本次品率，用正态分布对总体次品率进行拟合，再运用蒙特卡洛方法模拟试验，得到最优的决策方案。

对于问题一，我们为零配件的购买环节设计了一种抽样检测方案，在最小化检测成本的前提下判断是否接收这批零件。该方案在保证一定置信水平的前提下，通过假设检验确定最小样本容量，从而为企业提供科学的决策依据。这一方案的优势在于，企业可以通过实际生产时对误差的包容限度来选取样本容量，从而在保证置信度的同时最大限度压低检测成本。例如，若企业最多接受 3% 的误差，则在情形 (1) 和 (2) 中分别需要检测至少 271 和 164 个样本。

对于问题二，我们不采取分类讨论，而是将企业生产过程中需要做出的各项决策分别定义为 0-1 变量，用多个决策变量来表达出售单位成品需要的实际成本。在计算时，我们将实际成本分为三部分——成品的制造费用、对成品进行检测后不合格品带来的损失和不对成品进行检测时调换不合格品带来的损失。得到实际成本的表达式后，我们将问题转化成最小化实际成本的多变量优化问题。在问题二的较小规模下，使用枚举法分别得到了题目给出的 6 种情况的最优决策方案。

对于问题三，我们在问题二的基础上，将整个装配流程拆解为许多个类似于问题二的流程单元，并对于每个元件（零配件、半成品以及成品）定义三个特征量——获得成本、损坏率和拆解返还费用。我们按从前往后的顺序依次计算每个元件的特征量，并在最后一个流程单元中根据成品的特征量计算出售单位成品的实际成本。与第二问同理，将问题转换为最小化实际成本的优化问题。在通过枚举法给出题中两道工序、8 个零配件情形的最优决策后，我们将问题扩大到 m 道工序、n 个零配件的情形，引入大规模数据集进行比较分析，找到了最优决策算法——禁忌退火算法。然后，我们对该算法进行了实验检测、相关性分析和敏感度分析，以证明模型的解释力。

对于问题四，我们基于问题一确定了抽样调查基础上元件实际损坏率的分布函数。然后基于蒙特卡洛方法，运用问题三中的优化模型，通过大量随机模拟来消除抽样调查带来的元件实际损坏率不确定的影响。对于规模较小的问题二，我们对其每一可能决策进行蒙特卡洛模拟，得到成品实际成本的期望，再选取期望最小的决策为最优决策；而对于规模更大、复杂度更高的问题三，我们不再枚举所有决策，改为对整个问题进行蒙特卡洛模拟，并对得到的大量单次模拟最优决策进行统计处理，从而得到问题三的最优决策。对于 m 道工序、n 个零配件的一般情形，也可以使用同样的方法计算最优决策。

关键词： 显著性检验　0-1 规划　禁忌退火算法　蒙特卡洛方法　转化与化归思想

一、 问题重述

1.1 问题背景

企业在生产过程中会面临多个环节的决策问题，如是否对零件进行检测、是否对装配好的成品进行检测、对检测出的不合格品是否进行拆解等。这些问题的决策直接影响到企业的生产成本和产品质量，进而影响企业的市场竞争力。因此，如何通过科学的方法进行生产决策，实现效益最大化，是企业亟需解决的问题。

在理性人假设前提下，厂商会以利润最大化为目标进行生产决策，在售价一定时，也就等价于成本最小化。因此，厂商需要建立成本与各环节生产决策之间的联系，寻找最优的决策方案，从而实现成本最小化和利润最大化。

1.2 需要解决的问题

本题要求结合企业生产的实际情况建立数学模型，解决以下问题：

问题 1 供应商声称一批零配件的次品率不会超过某个标称值，企业决定采用抽样检测方法决定是否接收这批零配件。为最小化企业的检测费用，设计一个检测次数尽可能少的抽样检测方案。并在标称值为 10% 的情况下，针对下列两种情形，分别给出具体结果。

(1) 在 95% 的置信度下认为零配件次品率超过标称值，则拒收该批零配件；

(2) 在 90% 的置信度下认为零配件次品率不超过标称值，则接收该批零配件。

问题 2 装配成品需要零配件 1 和零配件 2。对于用户购买的不合格品，企业予以无条件调换，并产生一定的调换损失（物流成本、企业信誉等）。已知两种零配件和成品的次品率，为企业生产过程的各个阶段做出最优决策，并给出决策依据以及相应的指标结果。

(1) 是否对零配件进行检测；

(2) 是否对装配好的成品进行检测；

(3) 对于检测出的不合格品和用户退回的不合格品是否进行拆解。

问题 3 将问题 2 中 1 道工序、两个零配件的情形拓展为 m 道工序、n 个零配件的情形，已知各元件（零配件、半成品和成品）的次品率，给出生产过程中的最优决策方案。特别地，针对题目中给出的两道工序、8 个零配件的情形，给出具体的决策方案、决策依据及相应指标结果。

问题 4 假设问题 2 和问题 3 中各元件的次品率并非直接给定，而是通过抽样检测方法得到，重新完成问题 2 和问题 3。

二、 问题分析

2.1 问题一的分析

该问题要求为企业提供一种零配件的抽样检测方案，在满足一定置信水平的前提下尽可能减少检测次数，以降低企业的检测成本。为得到特定显著性水平之下样本容量 n 的最小值，需要对题目中给出的两种情况分别进行假设检验。

首先需要构造假设检验的统计量。不妨假设同一总体中每个零件是否为次品是独立同分布的，则抽样检测的样本中次品数量和次品率均服从二项分布。假定样本容量充足，那么可以用正态分布来近似样本次品率的分布，并根据样本容量和总体的实际次品率来确定均值和方差，进而构造检验统计量。

对于两种情况，分别使用上述构造出的统计量进行假设检验，得到对应的拒绝域，并反解出抽样检测的样本容量 n。样本容量 n 由两个因素决定，总体次品率的标称值 F_0 和包容误差 $f - F_0$。标称值给定时，包容误差越大，代表可接受的抽样检测误差越大，所需的样本容量 n 就越小。综合考虑误差和检测成本，选定包容误差为 0.03，分别计算出两种情形所需的最小样本容量 n，进而得出假设检验的具体结果。

2.2 问题二的分析

该问题考虑最简单的两配件一成品的情形，要求根据给定的次品率、成品的拆解费用等指标，为企业做出最优生产决策，也就是最小化出售单位成品的实际成本。目标转化为计算实际成本 c_0 的表达式。

分析可知，每一步不同的决策会产生不同的费用，进而导致不同的实际成本 c_0。我们将每一步的决策定义为 0-1 变量，并由此计算出最终出售单位成品的实际成本表达式，从而构成一个 0-1 整数规划问题。在本题中决策情况相对较少，因此我们仅需采取枚举的方式即可得到最优决策方案和对应的最小实际成本。

2.3 问题三的分析

该问题是问题二的推广，将一道工序、两个零配件扩大为 m 道工序、n 个零配件，生产过程中的决策步骤大大增多。因此，我们采取了模块化的分析方法，将完整的生产流程分割为若干流程单元，对每个流程单元采用类似问题二的思路，考虑获得每一个半成品的成本与各个决策变量的关系，最终逐步组合为一个规模较大的 0-1 规划问题。对于每个流程单元，兼顾获得成本较低和损坏率较低的双重目标；而对于最后一个包含成品的流程单元，则仍然以最小化出售单位成品的实际成本为优化目标。

当 m、n 较小时，如题目中所给的 $m = 2, n = 8$ 的情形，可以与问题二一样，采取枚举法计算所有决策情况下的实际成本，选取最小化实际成本的决策作为最优方案。但如果 m、n 较大，则需要采用动态规划、启发式算法等算法进行优化。我们建立了一个

$m = 5, n = 21$ 的测试集，将多种算法分别应用于题目所给情形和测试集进行测试，从中选取了效果较好的模拟退火-禁忌混合算法，并进一步调试参数，最终得到了一个对于不同初值、不同规模的问题都能较好收敛的模型。

为进一步证明模型的可解释性，我们分析了不同工序数 m 的最优决策集当中决策变量与对应元件获得成本、损坏率等变量的相关性，得到了与现实相符的结论。同时，我们还对模型进行了灵敏度分析，以指导企业更好地改变元件参数，从而降低成本。

2.4 问题四的分析

该问题要求改变题设，重做问题二和问题三。新的问题二、问题三与之前的唯一不同之处在于，次品率 p_i 没有固定数值，而是随机变量 f 的一个取值。将问题一中的结果代入，可以得到随机变量 f 服从的正态分布。为描述实际情况的不确定性，我们采用蒙特卡洛模拟，根据概率分布生成 p_i 的随机值，经过多次试验后综合得到优化结果。

问题二为一道工序、两个零配件的情形，规模较小，可以采用枚举法，对于每种决策方案进行 1000 次蒙特卡洛模拟，分别对得到的实际成本取平均值。将所有决策方案的平均实际成本进行比较，选择平均实际成本最小的作为最优方案。

问题三为 m 道工序、n 个零配件的复杂情形，规模较大，若采用枚举法计算量过于庞大。以题目中两道工序、8 个零配件的情形为例，我们直接进行 1000 次蒙特卡洛模拟，对得到的 1000 组次品率运用前述禁忌退火算法进行优化，得到对应的 1000 个最优决策。然后对 1000 个最优决策进行统计分析，得到最终的最优决策方案。

三、 模型假设与符号说明

3.1 模型假设

- 企业在进行生产决策时是理性的，仅考虑利润最大化，不考虑退换货对产品声誉的潜在影响；
- 假设成品售价不变，忽略销售费用等额外费用，则可以通过最小化实际成本来最大化企业利润；
- 同一批样品中每个零配件是否为次品相互独立且概率分布相同，每一批零配件样品数量足够多；
- 生产过程中的各个环节（如零配件的采购、装配、检测等）以及生产条件、市场需求、成本和价格等因素相互独立且保持稳定；
- 同种元件的单位检测成本固定，即同种元件总的检测成本与抽样检测的样本容量呈正相关，且与其他因素均无关；
- 检测结果是 100% 可靠的，不会出现任何检测错误；
- 运输成本和废弃零件的成本可以忽略不计。

3.2 符号说明

符号	意义
F	零配件次品率的真实值
F_0	零配件次品率的标称值
f	抽样检测得到的样本次品率
n_0	抽样检测得到的样本中次品的数量
n	样本容量
c_i	零配件 i 的购买单价
$\tilde{c}_i$	零配件 i 进入装配环节的费用
p_i	零配件 i 或成品（$i=0$）的次品率
t_i	零配件 i 或成品（$i=0$）的检测成本
q_i	零配件 i 损坏的概率
$\tilde{p}_0$	装配后成品的不合格率
z	成品的装配成本
h	成品的调换损失
c_{tear}	已检出次品的拆解费用
t_i	是否对零配件 i 或成品（$i=0$）进行检测
I_{tear}	是否对成品进行拆解
c_0	问题二中出售单位成品的实际成本
C_i	元件 i 的获得成本
P_i	元件 i 的损坏率
B_i	元件 i 的拆解返还费用
I_{s_i}	是否对元件 i 进行拆解
s_i	元件 i 的拆解费用
$\tilde{C}_0$	问题三中出售单位成品的实际成本

四、 问题一模型的建立与求解

4.1 假设检验统计量的构造

设零配件次品率（总体次品率）的真实值为 F，标称值为 F_0。构造统计量 f（样本次品率）作为 F 的估计值，即 $f = \frac{n_0}{n}$，其中 n 为样本容量，n_0 为样本中次品的数量。

假设总体足够大，可知 n_0 服从二项分布 $B(n, F)$。只要选取的 n 足够大，使得 $nF > 5$，便可以用正态分布对 n_0 进行近似，即：

$$n_0 \sim N(nF, nF(1-F)) \tag{1}$$

进而也可以用正态分布对 f 进行近似，如下：

$$f \sim N(F, \frac{F(1-F)}{n}) \tag{2}$$

进一步将 f 标准化，可以构造假设检验的统计量 Z, 服从标准正态分布：

$$Z = \frac{f - F}{\sqrt{\frac{F(1-F)}{n}}} \tag{3}$$

4.2 样本容量 n 的确定

样本容量 n 的确定与总体次品率的标称值 F_0 和可接受误差决定。本题中 $F_0 = 10\%$，下面根据可接受误差的大小，对样本容量进行讨论。

4.2.1 (1) 中样本容量 n 的确定

对统计量 Z 进行显著性检验。原假设 H_0 为 $F < F_0$，H_1 为 $F \geq F_0$，显著性水平 $\alpha = 0.05$。拒绝域应为 $Z > Z_{0.05}$，解得样本容量 n 应满足：

$$n > (\frac{Z_{0.05}\sqrt{F_0(1-F_0)}}{F_0 - f})^2 \tag{4}$$

由上式可知，在标称值 F_0 给定时，所需样本容量 n 的大小取决于可接受的误差量，在本题中具体表现为 $f - F_0$。例如，如果企业认为只要这批零配件次品率的真实值与标称值的差不大于三个百分点就可以接受，称包容误差为 0.03，代入数据 $f = 0.13$，计算得到最小的整数 $n = 271$；如果企业认为这批零配件次品率的真实值与标称值的差异在 5 个百分点内就能够接受，则代入 $f = 0.15$，计算得 $n = 98$。这与我们的生活经验一致，检测的要求越苛刻，需要的样本就越多。如果对误差的包容度较高，则只需要少量的样本即可。

同时，本题中 $nF > 5$ 的条件充分满足，说明我们进行正态近似的操作是合理的。

4.2.2 (2) 中样本容量 n 的确定

(2) 与 (1) 类似，同样对统计量 Z 进行显著性检验。原假设 H_0 为 $F > F_0$，H_1 为 $F \leq F_0$，显著性水平 $\alpha = 0.1$。此时，拒绝域应为 $Z < Z_{0.1}$，解得样本容量 n 应满足：

$$n > (\frac{Z_{0.1}\sqrt{F_0(1-F_0)}}{F_0 - f})^2 \tag{5}$$

同样地，分别代入 $f = 0.13$ 和 $f = 0.15$，计算得到最小的样本容量 n 分别为 164 和 59。(1) 和 (2) 中更详细的“包容误差-最小样本容量”的计算结果如下图所示：

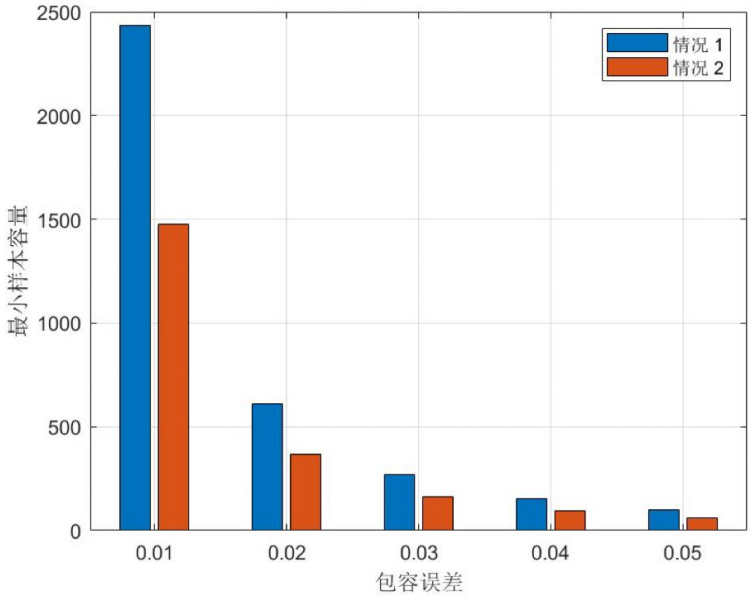


图 **1**　不同包容误差下的最小样本容量

4.3 抽样检测结果判定

根据上述计算结果，当包容误差取 0.03 时，可以在较低的误差下选取较小的样本容量，从而在控制误差与降低检测成本之间达到平衡。此时 (1) 和 (2) 中的最小样本容量分别为 271 和 164，下面分别对假设检验的具体结果进行说明。

4.3.1 (1) 中的抽样检测结果判定

选取最小样本容量 $n = 271$，假设检验的拒绝域为 $Z > Z_{0.05}$，即：

$$\frac{f - F_0}{\sqrt{\frac{F_0(1-F_0)}{n}}} > Z_{0.05} \tag{6}$$

整理得：

$$f > F_0 + Z_{0.05}\sqrt{\frac{F_0(1-F_0)}{n}} \tag{7}$$

将 f 代入，得到关于 n_0 的拒绝域：

$$n_0 > nF_0 + Z_{0.05}\sqrt{nF_0(1-F_0)} \tag{8}$$

代入数据解得，最小的整数 n_0 为 36，即至少检测出 36 个次品时，可拒绝 H_0，在 95% 的置信度下认定零配件次品率超过标称值，拒收这批零配件。

4.3.2 (2) 中的抽样检测结果判定

同理，对于 (2)，选取最小样本容量 $n = 164$，假设检验的拒绝域为 $Z < Z_{0.1}$，整理得：

$$n_0 < nF_0 + Z_{0.1}\sqrt{nF_0(1-F_0)} \tag{9}$$

代入数据解得，最大的整数 n_0 为 21，即至多检测出 21 个次品时，可拒绝 H_0，在 90% 的置信度下认定零配件次品率不超过标称值，接收这批零配件。

五、 问题二模型的建立与求解

5.1 零配件 i 进入装配环节的费用 $\tilde{c}_i$

将零配件 i 的购买单价记为 c_i。在企业生产过程的第一阶段，我们将零配件 i 进入装配环节的费用设为 $\tilde{c}_i$，其取值与是否对该零配件进行检测相关。用 I_{t_i} 表示是否对零配件 i 或成品（$i = 0$）进行检测，取值如下：

$$I_{t_i} = \begin{cases} 1, & \text{对零配件 } i \text{ 进行检测;} \\ 0, & \text{不对零配件 } i \text{ 进行检测} \end{cases} \tag{10}$$

若不对零配件 i 进行检测，则其直接进入装配环节，过程中产生的费用即为购买单价：

$$\tilde{c}_i = c_i \quad (I_{t_i} = 0) \tag{11}$$

若对零配件进行检测，将零配件 i 或成品（$i = 0$）的次品率记为 p_i，检测成本记为 t_i。根据次品率，每检测出一个合格的零配件 i，都需要消耗 $\frac{1}{1-p_i}$ 个未检测的零配件 i，同时检测过程中每个零配件 i 还会产生 t_i 的检测成本，则有：

$$\tilde{c}_i = \frac{1}{1-p_i}(c_i + t_i) \quad (I_{t_i} = 1) \tag{12}$$

利用变量 I_{t_i} 将上述两种情况**统一**起来，得到零配件 i 进入装配环节的费用：

$$\tilde{c}_i = \frac{1}{1 - I_{t_i} p_i}(c_i + I_{t_i} t_i) \tag{13}$$

5.2 装配后成品的不合格率 $\tilde{p}_0$

用 q_i 表示零配件 i 损坏的概率，与是否对该零配件进行检测有关。若进行检测，则损坏概率为 0；若不进行检测，则损坏概率即为该零配件的次品率。如下所示：

$$q_i = \begin{cases} 0, & I_{t_i} = 1; \\ p_i, & I_{t_i} = 0 \end{cases} \tag{14}$$

整理得：

$$q_i = (1 - I_{t_i}) p_i \tag{15}$$

装配后，成品不合格有两种情况。一种是零配件本身损坏，对应概率为 $1 - \prod_{i=1}^{2}(1 - q_i)$；另一种是零配件没有损坏，但装配出的成品为次品，对应概率为 $p_0 \prod_{i=1}^{2}(1 - q_i)$。设装配后成品的不合格率为 $\tilde{p}_0$，如下所示：

$$\begin{aligned} \tilde{p}_0 &= \left[1 - \prod_{i=1}^{2}(1 - q_i)\right] + \left[p_0 \prod_{i=1}^{2}(1 - q_i)\right] \\ &= 1 - (1 - p_0) \prod_{i=1}^{2}(1 - q_i) \end{aligned} \tag{16}$$

将 q_i 代入得：

$$\tilde{p}_0 = 1 - (1 - p_0) \prod_{i=1}^{2}[1 - p_i(1 - I_{t_i})] \tag{17}$$

5.3 出售单位成品的实际成本 c_0

定义 c_0 为**出售单位成品的实际成本**，由三部分组成：成品的制造费用、对成品进行检测后不合格品带来的损失、不对成品进行检测时调换不合格品带来的损失，分别记为 c_{01}、c_{02} 和 c_{03}。

成品的制造费用 c_{01} 与零配件 i 进入装配环节的费用 $\tilde{c}_i$ 同理，与是否对成品进行检测有关。将成品的装配成本记为 z，则制造费用如下所示：

$$c_{01} = \frac{1}{1 - I_{t_0} \tilde{p}_0}(\sum_{i=1}^{2} \tilde{c}_i + z + I_{t_0} t_0) \tag{18}$$

若对成品进行检测，得到一个合格的成品需要消耗 $\frac{1}{1-\tilde{p}_0}$ 个未检测的成品。对于剩余 $\frac{\tilde{p}_0}{1-\tilde{p}_0}$ 个已检出的不合格品，需要选择是否对其进行拆解，记为 I_{tear}，取值如下：

$$I_{tear}=\begin{cases}1, & \text{对已检出的不合格品进行拆解;}\\ 0, & \text{不对已检出的不合格品进行拆解}\end{cases} \tag{19}$$

将已检出不合格品的拆解费用记为 c_{tear}。检测后不合格品带来的损失与是否进行拆解有关。如果进行拆解，会损失拆解费用 c_{tear}，但可以收回零部件的成本 $\sum_{i=1}^{2} c_i$。最终的损失 c_{02} 如下：

$$c_{02}=I_{t_0}\frac{\tilde{p}_0}{1-\tilde{p}_0}I_{tear}(c_{tear}-\sum_{i=1}^{2}c_i) \tag{20}$$

如果不对成品进行检测，企业将对售出的不合格品进行调换，将产生的信誉等调换损失记为 h，除此之外还会损失调换品 c_0 的实际成本。对于调换回的不合格品，企业仍需选择是否对其进行拆解，同样可能产生拆解损失。所以，不对成品进行检测时调换不合格品带来的损失如下所示：

$$c_{03}=(1-I_{t_0})\tilde{p}_0\left[h+c_0+I_{tear}(c_{tear}-\sum_{i=1}^{2}c_i)\right] \tag{21}$$

将为 c_{01}、c_{02} 和 c_{03} 相加，即可得到出售单位成品的实际成本 c_0，如下所示：

$$\begin{aligned}c_0&=c_{01}+c_{02}+c_{03}\\&=\frac{1}{1-I_{t_0}\tilde{p}_0}(\sum_{i=1}^{2}\tilde{c}_i+z+I_{t_0}t_0)\\&+I_{t_0}\frac{\tilde{p}_0}{1-\tilde{p}_0}I_{tear}(c_{tear}-\sum_{i=1}^{2}c_i)\\&+(1-I_{t_0})\tilde{p}_0\left[h+c_0+I_{tear}(c_{tear}-\sum_{i=1}^{2}c_i)\right]\end{aligned} \tag{22}$$

整理得：

$$\begin{aligned}c_0=&\frac{1}{1-(1-I_{t_0})\tilde{p}_0}\left[\frac{1}{1-I_{t_0}\tilde{p}_0}(\sum_{i=1}^{2}\tilde{c}_i+z+I_{t_0}t_0)\right.\\&\left.+(1-I_{t_0})\tilde{p}_0h+\tilde{p}_0I_{tear}(1-I_{t_0}+\frac{I_{t_0}}{1-\tilde{p}_0})(c_{tear}-\sum_{i=1}^{2}c_i)\right]\end{aligned} \tag{23}$$

c_0 为出售单位成品的实际成本。本题中模型的优化目标为**最小化**单位实际成本，即最小化 c_0。

5.4 基于模型的最优决策方案

根据上述模型，以最小化单位实际成本 c_0 为目标，分别对情况 1-6 进行计算。对于每种情况，枚举出所有 16 种决策，计算每种决策的单位实际成本 c_0，选取 c_0 最小的作为最优方案。得到的最优决策方案和对应的最低单位实际成本如下表所示：

情况	是否检测			是否拆解	实际成本
	零配件1	零配件2	成品		
1	0	0	0	1	34.32
2	0	1	0	1	42.56
3	0	0	1	1	36.20
4	0	1	1	1	40.44
5	0	1	0	1	36.72
6	0	0	0	0	34.32

图 **2**　问题二的最优决策方案

六、 问题三模型的建立与求解

6.1 优化模型的建立

在问题二的基础上，我们运用**转化与化归思想**，将生产流程分割为若干个单元，分别进行考虑。其中一个流程单元由 n 个元件共同合成一个元件 0，如下图中黑色方框所示：

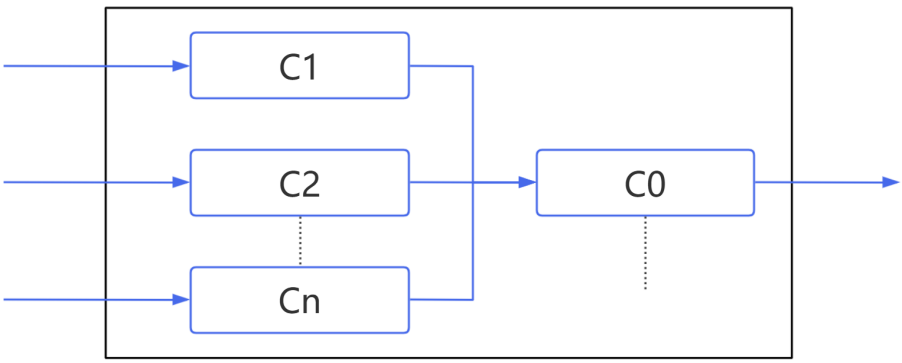


图 **3**　一个流程单元

对于每一个**元件（零配件、半成品、成品）**，均有三个属性随着生产决策的调整而改变：

- **获得成本** C_i：获得一个元件 i 的费用，包括先前的检测成本等。
- **损坏率** P_i：元件 i 损坏的概率，可以为 0。

- **拆解返还费用** B_i：对损坏的元件 i 进行拆解后，返还的零部件的成本。

特别地，对于零配件，定义 $C_i = c_i$、$P_i = p_i$、$B_i = 0$。下面先针对一个流程单元单独进行考虑。

6.1.1 元件 i 进入装配环节的费用 $\tilde{c}_i$

设元件 i 的拆解费用为 s_i。是否进行拆解记为 I_{s_i}，取值如下：

$$I_{s_i} = \begin{cases} 1, & \text{对元件 } i \text{ 进行拆解;} \\ 0, & \text{不对元件 } i \text{ 进行拆解} \end{cases} \tag{24}$$

零配件无法进行拆解，所以 $I_{s_i} = 0$。元件进入装配环节的费用分为两部分：一部分是元件的成本和可能的检测费用，另一部分则是检测后的拆解费用。对于元件 i，若进行检测，得到一个合格的元件 i 共需要 $\frac{1}{1-P_i}$ 个待测元件，检测出的 $\frac{P_i}{1-P_i}$ 个次品需要选择是否进行拆解。若拆解，则会花费 s_i 的拆解费用，但同时会得到 B_i 的拆解返还费用。所以，元件 i 进入装配环节的费用 $\tilde{c}_i$ 如下：

$$\tilde{c}_i = \frac{1}{1 - I_{t_i} P_i}(C_i + I_{t_i} t_i) + I_{t_i} \frac{P_i}{1 - P_i} I_{s_i}(s_i - B_i) \tag{25}$$

6.1.2 装配后元件 0 的损坏率 P_0

与 **5.2** 中式 (16) 同理，元件 0 的损坏分为组成元件 $i(i = 1, 2, \ldots\ldots, n)$ 损坏和装配过程中损坏两种情况。装配后元件 0 的损坏率 P_0 如下所示：

$$P_0 = 1 - (1 - p_0) \prod_{i=1}^{n} [1 - P_i(1 - I_{t_i})] \tag{26}$$

定义元件 0 的装配费用为 z_0，则元件 0 的获得成本为：

$$C_0 = \sum_{i=1}^{n} \tilde{c}_i + z_0 \tag{27}$$

同时，元件 0 的拆解返还费用为：

$$B_0 = \sum_{i=1}^{n} C_i \tag{28}$$

6.1.3 出售单位成品的实际成本 $\tilde{C}_0$

按从前往后（零配件、半成品、成品）的顺序，对每个图3所示的流程单元依次进行计算，可得每个元件的对应变量 C_i、P_i 和 B_i。在包含成品的最终流程单元中，我们可以根据成品的 C_0、P_0、B_0，计算得到**出售单位成品的实际成本** $\tilde{C}_0$。

与 **5.3** 中式 (22) 同理，出售单位成品的实际成本分为三部分：配件与组装成本，检测成品后处理次品的费用，和不检测成品时处理售后的费用。出售单位产品的实际成本为：

$$\begin{aligned}\tilde{C}_0 = &\frac{1}{1-I_{t_0}P_0}(C_0 + I_{t_0}t_0) \\ &+ I_{t_0}\frac{P_0}{1-P_0}I_{s_0}(s_0 - B_0) \\ &+ (1-I_{t_0})P_0\left[h + \tilde{C}_0 + I_{s_0}(s_0 - B_0)\right]\end{aligned} \tag{29}$$

整理得：

$$\begin{aligned}\tilde{C}_0 = &\frac{1}{1-(1-I_{t_0})P_0}\left[\frac{1}{1-I_{t_0}P_0}(C_0 + I_{t_0}t_0)\right. \\ &\left.+(1-I_{t_0})P_0h + P_0I_{s_0}(1 - I_{t_0} + \frac{I_{t_0}}{1-P_0})(s_0 - B_0)\right]\end{aligned} \tag{30}$$

根据上述模型，与问题二同理，优化目标即为最小化单位成品的实际成本 $\tilde{C}_0$。

6.2 两道工序、8 个零配件情形的决策方案

根据优化模型，原题被抽象为决定 n 个 0-1 变量使得目标函数最小的问题。[1] 当工序和零配件数量较少时，与问题二类似，可以采用**枚举法**进行计算，根据实际成本 $\tilde{C}_0$ 最小的优化目标得到最优决策。本题中两道工序、8 个零配件情形的最优决策方案如下：

元件	零配件								半成品			成品
	1	2	3	4	5	6	7	8	1	2	3	0
是否检测	0	0	0	0	0	0	0	1	0	0	1	0
是否拆解	/	/	/	/	/	/	/	/	0	0	1	1
实际成本	113.11											

图 **4**　两道工序、**8** 个零配件情形的最优决策方案

6.3 m 道工序、n 个零配件情形的优化算法

一般地，考虑 m 道工序、n 个零配件的情形。当工序和零配件数量较多时，枚举法所需时间成指数级爆炸增长。因此，我们迫切地需要找到一个简单、快速的优化算法来确定最佳的 0-1 向量。

6.3.1 优化算法的选择

在前面的分析过程中，我们将整个拼装流程看作许多个图3所示流程单元的组合。对于每个元件，无论是零件、半成品还是成品，都有三个随先前决策变化的变量——获

得成本、损坏率和拆解返还费用。每个流程单元中各流程零件的三个变量均由前置流程的决策决定。如此，我们首先尝试将问题近似为一个**马尔可夫过程**，可以尝试利用**贪婪算法**、**动态规划**或者以一定条件**贪心剪枝的动态规划算法**来求最优解。

我们的贪心策略是，从前往后（即从零配件到成品）求解，对于每一个小的拼装子流程，定义一个子目标函数 $f=\frac{C_0}{1-P_0}$，通过动态规划取到使该函数最小的决策。我们的剪枝动态规划的剪枝策略是，若某节点的子目标函数大于历史最小值的某个倍数，则舍去之后的分支。

此外，由于该问题具有规模大，结构较为复杂的特点，我们也可以采取**启发式算法**[2]来在尽可能短的情况下得到最优解。利用 **6.2** 中两道工序、8 个零配件的情形对不同算法进行测试，其中启发式算法经过多次计算取平均值，得到结果如下：

算法	运行时间（秒）	实际成本（元/件）	超出最小实际成本的比例
枚举	28.2292	113.11	/
逐层贪婪	0.0007	122.99	8.73%
动态规划	3.6531	113.11	/
剪枝动态规划	1.9044	113.11	/
遗传	0.0306	113.35	0.22%
蜂群	0.2427	113.11	/
粒子群	0.1325	113.11	/
退火	0.1671	113.11	/
禁忌	0.0608	113.11	/

图 **5**　两道工序、**8** 个零配件情形的不同算法测试结果

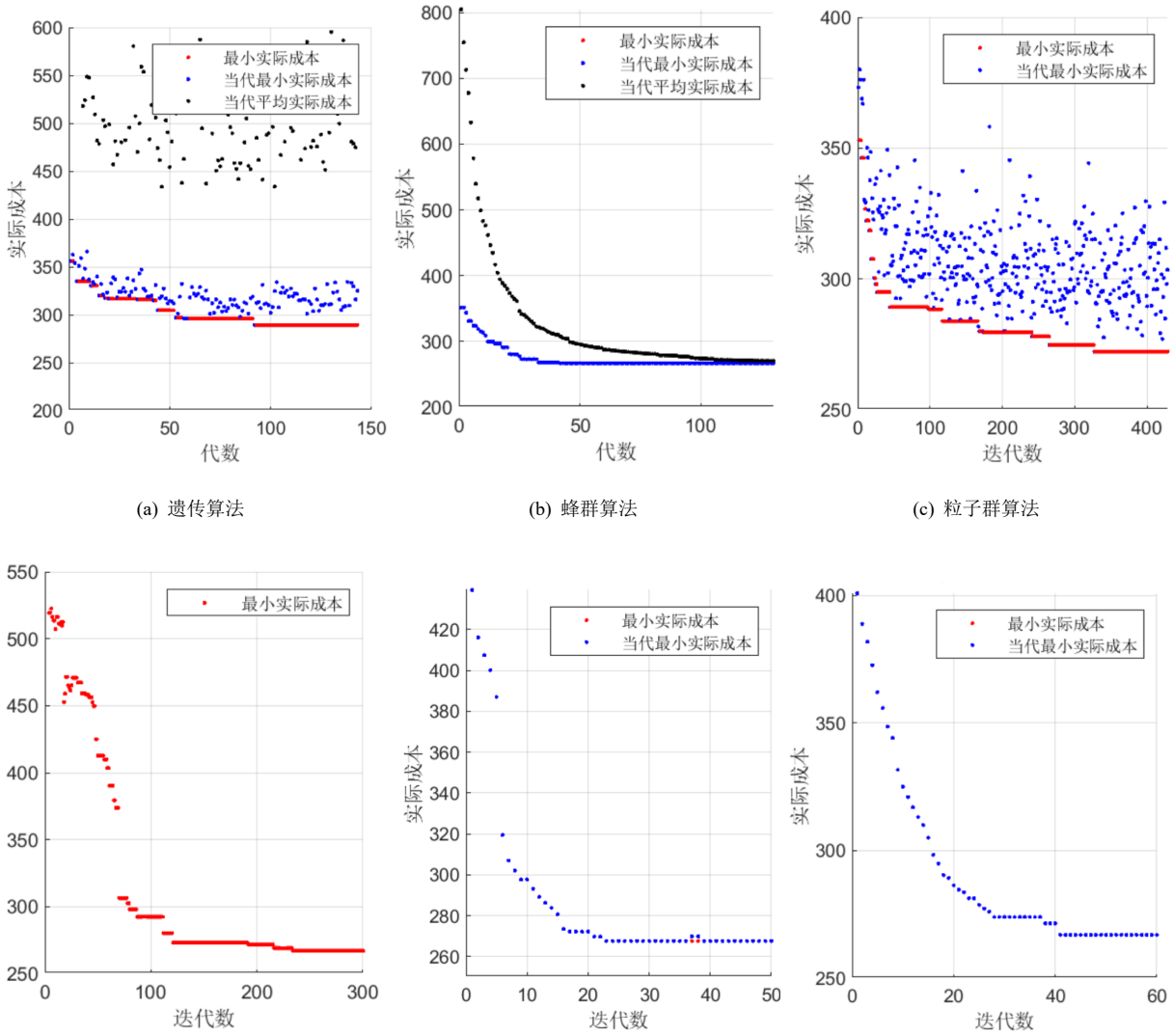
可以看到，在当前规模下，除逐层贪婪算法和遗传算法之外，其他算法均有比较好的结果。考虑到题目要求情形的大规模性，我们令 $m=5$，$n=21$，自行生成了一个具有 5 道工序、21 个零配件的测试集（见附件）。利用该测试集对上述算法进行测试，其中剪枝动态规划采用不同的剪枝策略、启发式算法均多次运行取平均值，得到结果如下：

算法	运行时间（秒）	实际成本（元/件）	超出最小实际成本的比例
枚举	600+	/	/
逐层贪婪	0.0012	541.65	102.87%
动态规划	600+	/	/
剪枝动态规划1	31.9754	427.76	60.22%
剪枝动态规划2	409.99	344.92	29.18%
遗传	0.689	289.16	8.30%
蜂群	0.6157	267.15	0.05%
粒子群	1.2126	272.24	1.97%
退火	0.3613	266.99	/
禁忌	0.2726	266.99	/
禁忌退火	0.3125	266.99	/

图 **6**　**5** 道工序、**21** 个零配件情形的不同算法测试结果

如上表所示，逐层贪婪、动态规划及两种剪枝动态规划算法运行时间明显更长，收

敛结果不如启发式算法。可见处理大规模的规划问题，启发式算法在全局搜索能力和适应性方面的优势是难以替代的。相较而言，蜂群算法和粒子群算法精度尚可，但运行速度较慢；同时由图7可以看出，粒子群算法得到最优解的随机性较大，蜂群算法则比较容易陷入局部最优解，二者均存在缺陷。**退火算法**和**禁忌算法**表现较好，精度高的同时运行速度也较快，值得采用。



(a) 遗传算法　(b) 蜂群算法　(c) 粒子群算法

(d) 退火算法　(e) 禁忌算法　(f) 退火禁忌算法

图 7　5 道工序、21 个零配件情形的不同算法迭代情况

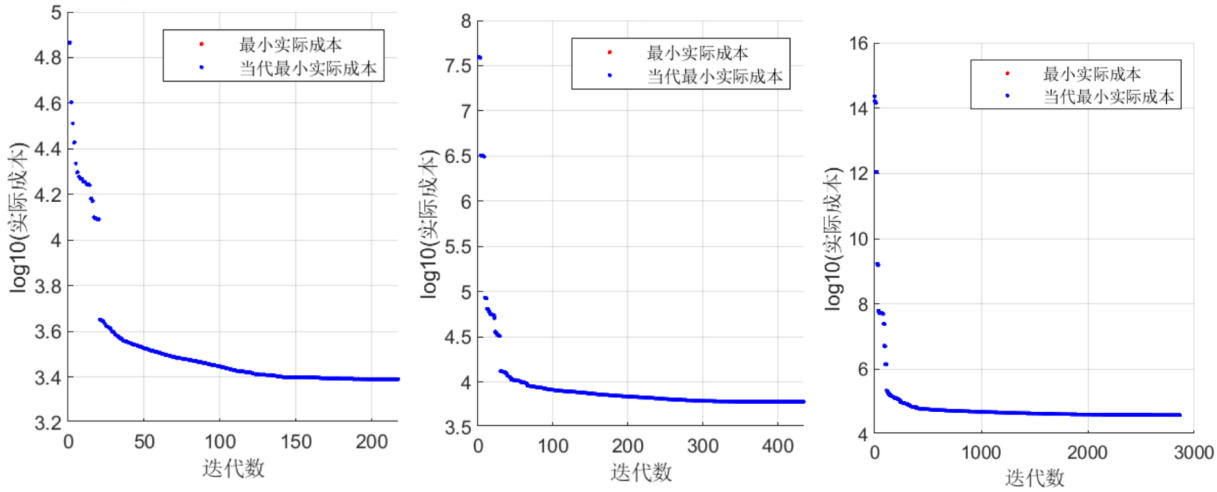
需要注意的是，退火算法和禁忌算法有着不同的特点，禁忌算法从邻域中选取更优解，且通过禁忌表的方式来避免重复搜索，收敛速度较快；退火算法通过一定概率接受劣解的方式来跳出局部最优，有一定的全局搜索能力，但收敛速度会偏慢。

因此，我们可以将两者结合起来，用禁忌搜索来解决结构化的局部问题，而用模拟退火来进一步优化解决方案，增强解的多样性。[3] 具体表现为在禁忌搜索的每一次邻域搜索引入退火，使得邻域解的选择上更具多样性和记忆功能，提升搜索效率和质量；

通过禁忌表来防止退火在高温时反复搜索无效解，并且在温度降低时对局部最优进行深度挖掘。禁忌退火算法应用于 5 道工序、21 个零配件测试集上的结果也展示在图6和图7中，效果很好。

6.3.2 禁忌退火算法的测试

为检验禁忌退火算法的普适性，我们在 5 道工序、21 个零配件情形的基础上，进一步增大 m 和 n 的取值，在更大的流程规模下对禁忌退火算法进行测试，得到的迭代图像如下：



(a) $m=5$，$n=165$ 情形（6.14 秒）　(b) $m=6$，$n=384$ 情形（12.32 秒）　(c) $m=7$，$n=2448$ 情形（1340.64 秒）

图 **8**　退火算法测试迭代情况

如图8所示，当问题规模扩大时，禁忌退火算法仍能保持较好的准确性和较快的收敛速度。所以，在下面的优化问题中，我们均优先采用禁忌退火算法来计算最优决策方案。

七、 问题四模型的建立与求解

在问题四的假设下，题中所给各元件的次品率不再是总体的真实次品率，而是通过抽样检测得到的样本次品率。为了给出此前提下的最优决策，我们需要使用**蒙特卡洛方法**，通过足够多次模拟实验来消除真实次品率的不确定性带来的影响。为此，我们首先要根据题目中已知的样本次品率来得到真实次品率的分布函数。

与问题一同理，在抽样调查的样本容量确定的情况下，样本次品率是以真实次品率为期望的正态分布，如图9中 (a) 所示。由对称性可知，已知真实次品率为 x 时样本次品率为 y 的概率，与已知样本次品率为 y 时真实次品率为 x 的概率相同，如图9中 (b) 所

示。因此，在已知样本次品率时，真实次品率服从以样本次品率为期望的正态分布，且方差与已知真实次品率时样本次品率的正态分布相同，如图9中 (c) 所示。

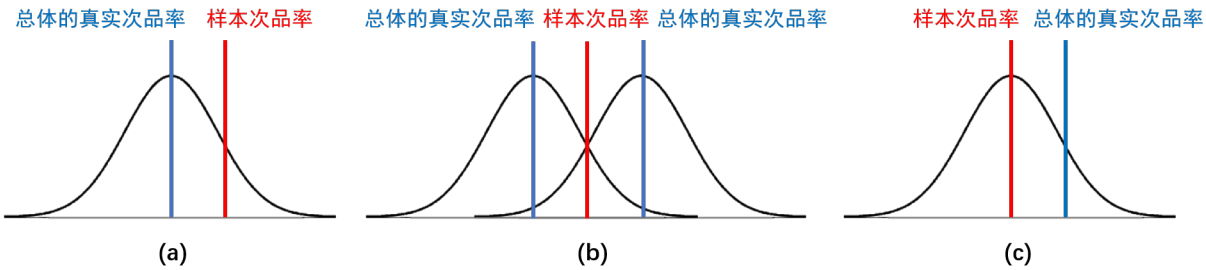


图 **9**　正态分布

由此，只要给定抽样检测时的样本容量 n_i，我们就能得到真实次品率 F_i 关于样本次品率 f_i（即题目中给出的次品率）的概率分布，如下所示：

$$F_i \sim N(f_i, \frac{f_i \times (1-f_i)}{n_i}) \tag{31}$$

考虑到第一题的抽样调查方法，与 **4.3.2** 类似，我们取显著性水平 $\alpha = 10\%$，包容误差为 3%。将题目中样本次品率分别为 5%，10%，20% 的元件的真实次品率分别记为 F_1, F_2, F_3，那么结合公式5，检测所需的最小样本容量分别为 87，164，292。代入数据可得 F_1，F_2，F_3 的分布，如下所示：

$$\begin{aligned} F_1 &\sim N(0.05, \frac{0.05 \times (1-0.05)}{87}) \\ F_2 &\sim N(0.1, \frac{0.1 \times (1-0.1)}{164}) \\ F_3 &\sim N(0.2, \frac{0.2 \times (1-0.2)}{292}) \end{aligned} \tag{32}$$

此时，新的问题与前述问题二、问题三的唯一区别是，次品率 F_i（即问题二、问题三公式中的 p_i）没有固定数值，而是随机变量 F_1，F_2 或 F_3 的一个取值。为描述实际情况的不确定性，我们采用蒙特卡洛模拟[4]，生成 p_i 的随机值，经过多次试验后综合得到优化结果。

7.1 次品率 p_i 非固定下的问题二

问题二是一道工序、两个零配件的基础情形，规模较小，可以采用枚举法。对于每种情况的四个 0-1 变量进行枚举，共有 $2^4 = 16$ 种决策方案。对于每种决策方案，分别进行 1000 次蒙特卡洛模拟，对得到的实际成本取平均值。将 16 种决策方案的平均实际成本进行比较，选择平均实际成本最小的作为最优方案。6 种情况的最优方案和对应的最小平均实际成本如下表所示：

情况	是否检测 零配件1	零配件2	成品	是否拆解	平均实际成本
1	0	0	0	1	34.3674
2	0	1	0	1	42.6063
3	0	0	1	1	36.2479
4	0	1	1	1	40.4744
5	0	1	0	1	36.7626
6	0	1	0	0	34.3625

图 **10**　次品率 p_i 非固定下的问题二最优决策

7.2 次品率 p_i 非固定下的问题三

问题三中的情形为两道工序、8 个零配件，规模较大，若采用枚举法需要进行 $2^{16} \times 1000$ 次蒙特卡洛模拟，运算量过于庞大。所以，我们对蒙特卡洛方法进行调整，重复 N 次下列操作：

(1) 按照式 (32) 所示的概率分布进行蒙特卡洛模拟，分别生成各元件（零配件、半成品、成品）的次品率 p_i；

(2) 根据生成的次品率 p_i，运用 **6.3** 中的禁忌退火算法，计算出对应的最优决策，记为向量 J_i。

当 N 足够大时（本题中取 $N = 1000$），根据蒙特卡洛原理，最终的最优决策即为 J_i 的期望。但由于决策分量只能取 0 或 1，需要对 J_i 进行统计处理，得到次品率变动之下的最优决策。统计处理的操作步骤如下：

(1) 仿照前题，对于 J_i 中的每个决策分量，记 1 为检测、拆解，0 为不检测、不拆解，由此 J_i 成为一个 16 维的向量；

(2) 记 J^* 为最终的最优决策。对 J_i 的各分量分别计算均值，将结果记为 J_0。若 J_0 的第 k 个分量 $< l$，则令 J^* 的第 k 个分量为 0；若 J_0 的第 k 个分量 $> u$，则令 J^* 的第 k 个分量为 1；

(3) 对于 J_0 所有在 l 和 u 之间的分量，进行相关度分析，计算两两之间的相关系数；

(4) 对于 J_0 的第 1 个在 l 和 u 之间的分量，以 0.5 为分界取 1 或 0。若该分量 ≥ 0.5，其在 J^* 中的对应分量取 1，否则取 0；

(5) 对于 J_0 之后的每一个在 l 和 u 之间的分量，依次判定其与已确定分量的相关系数。若相关系数的绝对值不小于 $corr_{max}$，则根据正相关或负相关取对应值；若所有相关系数的绝对值均小于 $corr_{max}$，则与 (4) 同理，以 0.5 为分界取 1 或 0。

在本题中，取 $l = 0.4$，$u = 0.6$，$corr_{max} = 0.3$，由此得到最终的最优决策 J^*，各决策分量的取值如下表所示：

元件	零配件								半成品			成品
	1	2	3	4	5	6	7	8	1	2	3	0
是否检测	0	0	0	0	0	0	0	1	0	0	1	0
是否拆解	/	/	/	/	/	/	/	/	1	1	1	1

图 **11**　次品率 p_i 非固定下的问题三最优决策

对于问题三中 m 道工序、n 个零配件的情形，也可采取上述方法解决，同样需要对统计参数 l、u 和 $corr_{max}$ 进行恰当取值，这里不再赘述。

八、 模型分析

8.1 模型的相关性分析

为了说明问题三中通用模型的可解释性，我们对同一元件（零配件、半成品、成品）的各变量进行相关性分析，探究检测、拆解决策的影响因素。我们按前述方式生成了多个工序数 m 不同的流程，对于每个流程，分别使用优化算法求得最优决策方案，然后求出两个决策变量与各变量之间的相关系数。将求得的相关系数按工序数 m 进行分类，计算出平均值，如下表所示：

	是否检测I_{ti}				是否拆解I_{si}			
工序数m	3	4	5	6	3	4	5	6
获得成本C_i	0.13	0.23	0.15	0.05	0.63	0.38	0.28	0.24
损坏率P_i	0.69	0.29	0.04	0.05	0.25	0.46	0.54	0.59
检测成本t_i	-0.52	-0.16	-0.17	0.01	-0.01	0.15	-0.14	-0.03
P_i*C_i/t_i	0.62	0.51	0.32	0.02	0.51	0.42	0.24	0.12
拆解费用s_i	0.13	0.28	0.06	0.12	-0.55	-0.38	-0.32	-0.26
拆解返还费用B_i	0.36	0.6	0.5	0.06	0.9	0.44	0.26	0.25
B_i/s_i	0.41	0.6	0.15	0.06	0.68	0.69	0.59	0.02

图 **12**　按工序数分类的决策变量与各变量之间的平均相关系数

记 $P_i\frac{C_i}{t_i}$ 为比值 1，$\frac{B_i}{s_i}$ 为比值 2。对上表进行分析，可得到如下结论：

(1) 是否检测与获得成本、损坏率、比值 1 呈正相关，与检测成本呈负相关，且相关性均均随工序数的增加而减小；

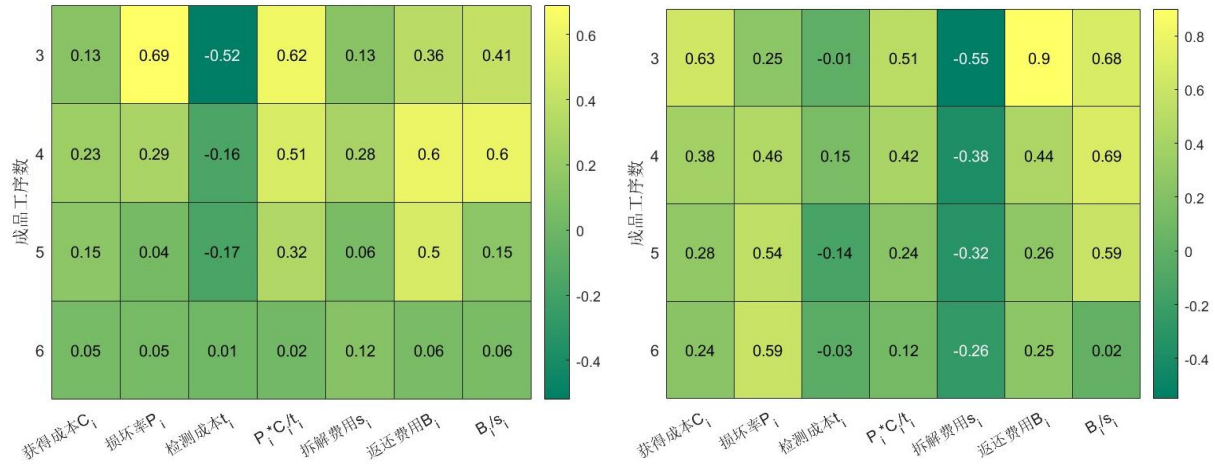
(2) 是否检测与拆解费用、拆解返还费用、比值 2 的相关性较弱；

(3) 是否拆解与获得成本、拆解返还费用、比值 1 呈正相关，与拆解费用呈负相关，且相关性均随工序数的增加而减小；

(4) 是否拆解与损坏率、比值 2 始终保持较强的正相关性；

(5) 是否拆解与检测成本之间的相关性较弱。

为了更直观地展示检测决策和拆解决策与各变量之间的相关关系，我们分别作出热力图，进行相关因子分析：



(a) 检测决策的相关因子分析　　(b) 拆解决策的相关因子分析

图 **13**　检测决策和拆解决策的相关因子分析

由图13可知，随着工序数 m 的增加，各流程单元之间的相互作用增强，导致检测决策和拆解决策与绝大部分变量之间的相关性逐渐下降。但拆解决策与损坏率之间的相关性则随着工序数 m 的增加而上升，是因为工序数较多时，拆解可以挽回更多的损失。

上述结论也与生活经验相符合。检测决策与元件的获得成本、检测成本和损坏率相关。当元件的检测成本与获得成本相比较低时，选择进行检测的花费相对较小，但可以避免较大的损失，企业会倾向于选择对元件进行检测。当元件的损坏率较高时，进行检测有助于避免可能产生的装配损失，企业也会倾向于选择检测。

拆解决策与拆解费用和拆解返还费用相关。当元件的拆解费用较低、拆解返还费用较高时，对不合格元件进行拆解可以挽回更多的损失，企业会倾向于选择对元件进行拆解。拆解返还费用与拆解费用的比值对企业选择拆解所能获得收益的影响较为直接，所以相关性随着工序数的增加没有明显的降低趋势。是否拆解与检测成本之间的相关性较弱，则说明了拆解决策与检测决策之间相对独立。

8.2 模型的敏感度分析

为了进一步指导商家调整元件（零配件、半成品、成品）的各变量，从而更好地降低成本，我们对问题三中得到的通用模型进行敏感度分析，计算各元件参数 X 与出售单位成品的实际成本 $\tilde{C}_0$ 之间的弹性 E_X，如下：

$$E_X = \frac{\Delta \tilde{C}_0 / \tilde{C}_0}{\Delta X / X} \tag{33}$$

以问题三中给定的数据为例，将元件变量 X 整体乘一个系数 α，得到一组新的数据，并根据模型计算得到新的最小实际成本。用新数据与原数据计算对应变量的弹性，得到结果如下表所示：

X	购买单价	次品率	检测成本	装配成本	拆解费用	调换损失
E_X	0.568	0.084	0.122	0.294	0.016	0.039

图 **14**　两道工序、**8** 个零配件情形的敏感度分析

可以看出，购买单价和装配成本等直接变量与实际成本的弹性较大。企业在生产过程中，为降低实际成本，可以优先考虑提高议价能力，从供应商处获得较低的购买单价。此外，还可以考虑研发改进技术，从而降低装配成本。

九、 模型评价

9.1 模型的优点

1. 选取检测方案时，企业可以通过实际生产时对误差的接受限度来选取样本容量，实现在保证检测效果的同时最大限度压低检测成本。
2. 考虑决策的整体性将问题抽象为 0-1 规划问题，并且使用启发式算法，保证了模型强大的适应性；对于不同规模的问题比较多种启发式算法结果，为选择禁忌-退火算法的优越性提供依据。
3. 禁忌算法通过禁忌表来避免重复搜索，能够较快收敛；退火算法引入“温度”来随机改变结果，有利于跳出局部最优。二者的结合保证了结果收敛的效率和稳定性。
4. 引入蒙特卡洛方法处理次品率的不确定性，并且根据不同的问题规模采用适合的模拟方案，保证了结果的鲁棒性和适用性。

9.2 模型的缺点

1. 问题规模较大时，启发式算法刚需对计算机的使用，对算力有较高要求；且可解释性较差，不利于决策过程的分析与灵活调整。
2. 为提高模型的普适性和用户友好性，还需进一步研究如何对模型的算法参数进行系统的选择与调整。

十、 模型推广

我们认为，本模型以其不凡的拟合能力和良好的泛用性，在多个行业领域中都可以展现出一定的的应用前景，特别是在那些流程复杂、步骤繁多且对容错性有一定要求的决策与规划问题中，其适用性尤为显著。例如，在汽车制造或电子产品组装领域，模型能够针对流水线的规划提供优化方案，通过精确计算和模拟，确保生产流程的高效运转和产品质量的持续提升。此外，模型在库存管理和物流路径规划方面的应用，能够显著降低企业的运营成本，提高供应链的响应速度和整体灵活性。

在服务行业，模型的应用不仅限于流程优化，还能提升服务质量。在医疗、教育或金融服务等领域，模型能够通过精细化管理，提高服务流程的效率和质量，从而增强用户体验和满意度。随着全球化的不断深入，模型在跨国生产和分销决策中的应用，能为国际企业在全球范围内的运营提供决策支持，优化了全球供应链网络，提高企业的国际竞争力。

在公共政策领域，该模型同样展现出应用潜力。城市规划、交通管理等政策制定过程中，模型能够提供数据驱动的决策支持，帮助政策制定者更科学地进行规划和管理。同时，在技术创新与产品开发方面，模型的引入能够指导研发资源的合理分配，加速新产品的研发进程，缩短产品从设计到上市的周期。

综上所述，本模型由其广泛的适用性和灵活性从而可以成为解决更广泛社会和经济问题的工具，具有一定的影响力和实用价值。随着技术的不断进步和应用场景的不断拓展，模型的潜力将进一步得到挖掘和发挥，为各行各业的发展贡献力量。

参考文献

[1] 姚成成. 基于 0-1 规划下企业原材料订购与运输的量化关系研究. 中国市场, (22):166–168+193, 2022.

[2] 董德威, 颜云辉, 张尧, and 李骏. 矩形件优化排样的自适应遗传模拟退火算法. 中国机械工程, 24(18):2499–2504, 2013.

[3] 闫磊 and 董辉. 模拟退火与禁忌搜索算法在协同配送中的应用. 宜春学院学报, 39(9):39–42, 2017.

[4] 石文辉, 别朝红, and 王锡凡. 大型电力系统可靠性评估中的马尔可夫链蒙特卡洛方法. 中国电机工程学报, (4):9–15, 2008.

附录 A　代码

```
%============================================================
% 采用枚举做B2
%============================================================
%% 输入
% 成本
cost=[4,18,0];
% 次品率
p_fail=[.05,.05,.05];
% 检测成本
cost_test=[2,3,3];
% 装配成本
cost_set=[0,0,6];
% 拆解费用
cost_tear=[0,0,40];
% 下级合成材料
next_level=[3,3,0];
% 调换损失
cost_change=10;

%% 过程量
% 拆解返还费用
cost_back=zeros(1,length(next_level));
% 形式成本
new_cost=cost;
% 形式次品率
new_p_fail=p_fail;

%% 确定量
% 是否检测
if_test=[];
% 是否拆解
if_tear=[];

%% 遍历
min=inf;
answer=[];
bar=waitbar(0);
for i_try=0:(2^(2*length(next_level))-1)
temp_ans=bitget(i_try,2*length(next_level):-1:1);
if_test=temp_ans(1:end/2);
if_tear=temp_ans(1+end/2:end);

%% 计算
% 计算最终层
```

```
k=length(next_level);
% 寻找k的前置
kids=[];
for i=1:length(next_level)
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
% k的新损坏率
temp=1-p_fail(k);
for j=1:n
temp=temp*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-temp;

new_cost(k)=new_cost(k)+cost_set(k);
new_cost(k)=(new_cost(k)+if_test(k)*cost_test(k)) / (1-if_test(k)*new_p_fail(k))...
+if_test(k)*new_p_fail(k)/(1-new_p_fail(k))*if_tear(k)*(cost_tear(k)-cost_back(k))...
+(1-if_test(k))*new_p_fail(k)*(cost_change+if_tear(k)*(cost_tear(k)-cost_back(k)));
new_cost(k)=new_cost(k)/(1-(1-if_test(k))*new_p_fail(k));

if new_cost(k)<min
min=new_cost(k);
answer=temp_ans;
end
waitbar(i_try/(2^(2*length(next_level))-1),bar,'waiting...')
end

disp(min);
disp(answer([1:3,6]));
```

```
%=============================================================
% 递归算多工序问题中售卖一份成品需要的实际成本
%=============================================================
decision=[0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1];
[r_cost,r_p,r_back]=RealCost(test_1,cost_change_1,decision,size(test_1,2));

function fun_back=RealCost(test,cost_change,decision,k)
% 计算节点k的前置层
% 寻找k的前置
kids=[];
costs=[];
ps=[];
backs=[];
for iii=1:k-1
if test(1,iii)==k
kids=[kids iii];
temp=RealCost(test,cost_change,decision,iii);
costs=[costs temp(1)];
ps=[ps temp(2)];
backs=[backs temp(3)];
end
end
num_kids=length(kids);
if num_kids>0
% 临时量
temp_cost=zeros(1,num_kids);
r_back=0;
r_cost=0;

% 更新前置的形式成本
for iii=1:num_kids
temp_cost(iii)=(costs(iii)+decision(kids(iii))*test(4,kids(iii))) /
    (1-decision(kids(iii))*ps(iii))...
+decision(kids(iii))*ps(iii)/(1-ps(iii))*decision(end/2+kids(iii))
*(test(6,kids(iii))-backs(iii));
% k的返还费用
r_back=r_back+costs(iii);
% k的新费用
r_cost=r_cost+temp_cost(iii);
end
r_cost=r_cost+test(5,k);
% k的新损坏率
r_p=1-test(3,k);
for j=1:num_kids
r_p=r_p*(1-ps(iii)*(1-decision(kids(iii))));
end
```

```
r_p=1-r_p;
else
r_cost=test(2,k);
r_p=test(3,k);
r_back=0;
end

if k==size(test,2)
r_cost=(r_cost+decision(k)*test(4,k)) / (1-decision(k)*r_p)...
+decision(k)*r_p/(1-r_p)*decision(end/2+k)*(test(6,k)-r_back)...
+(1-decision(k))*r_p*(cost_change+decision(end/2+k)*(test(6,k)-r_back));
r_cost=r_cost/(1-(1-decision(k))*r_p);
end
fun_back=[r_cost,r_p,r_back];
end
```

```
%============================================================
% 穷举最优化B3前半问题
%============================================================
%% 采用穷举做b-3
% 28.229226 秒
% 113.1056
% 是否检测：0   0    0    0    0    0    0    1    0    0    1    0
% 是否拆解：0   0    0    0    0    0    0    0    0    0    1    1

tic;
%% 输入
% 成本
cost=[2,8,12,2,8,12,8,12,0,0,0,0];
% 次品率
p_fail=[.1,.1,.1,.1,.1,.1,.1,.1,.1,.1,.1,.1];
% 检测成本
cost_test=[1,1,2,1,1,2,1,2,4,4,4,6];
% 装配成本
cost_set=[0,0,0,0,0,0,0,0,8,8,8,8];
% 拆解费用
cost_tear=[0,0,0,0,0,0,0,0,6,6,6,10];
% 下级合成材料
next_level=[9,9,9,10,10,10,11,11,12,12,12,0];
% 调换损失
cost_change=40;

%$ 过程量
% 拆解返还费用
cost_back=[0,0,0,0,0,0,0,0,0,0,0,0];
% 形式成本
```

```
new_cost=cost;
% 形式次品率
new_p_fail=p_fail;

%$ 确定量
% 是否检测
if_test=[];
% 是否拆解
if_tear=[];

%% 尝试穷举
% 前8个必然不拆解，只需穷举12+4个选项
min=inf;
answer=[];
bar=waitbar(0);
upper=12+4;
for i_try=0:2^upper-1
temp_ans=bitget(i_try,upper:-1:1);
if_test=temp_ans(1:12);
if_tear=[zeros(1,8),temp_ans(13:16)];

%% 计算
% 是否完成计算
if_complete=zeros(1,length(next_level));
for ii=1:length(next_level)-1
k=next_level(ii);
if if_complete(k)==0 && k~=length(next_level)
% 计算节点k的前置层
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
```

```
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
new_cost(k)=new_cost(k)+cost_set(k);
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);
end
if_complete(k)=1;
end
% 计算最终层
k=length(next_level);
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);

new_cost(k)=new_cost(k)+cost_set(k);
```

```
new_cost(k)=(new_cost(k)+if_test(k)*cost_test(k)) / (1-if_test(k)*new_p_fail(k))...
+if_test(k)*new_p_fail(k)/(1-new_p_fail(k))*if_tear(k)*(cost_tear(k)-cost_back(k))...
+(1-if_test(k))*new_p_fail(k)*(cost_change+if_tear(k)*(cost_tear(k)-cost_back(k)));
new_cost(k)=new_cost(k)/(1-(1-if_test(k))*new_p_fail(k));
% 更新最优解
if new_cost(k)<min
min=new_cost(k);
answer=[if_test;if_tear];
end
waitbar(i_try/(2^upper),bar,'waiting...');
end

toc;
disp(min);
disp(answer);
```

```
%============================================================
% 剪枝动态规划最优化B3前半问题
%============================================================
%% 输入
test=test_B3;
cost_change=cost_change_B3;
goal=1;

main(test,cost_change,goal);

function main(test,cost_change,goal)
%% 初始化
num=size(test,2);
num_low=length(find(test(5,:)==0));
min=inf;
answer=[];
bar=waitbar(0);
upper=num*2-num_low;

%% 贪婪
global best_node;
best_node=ones(1,num)*inf;

%% 穷举
tic;
for i_try=0:2^upper-1
temp_ans=bitget(i_try,upper:-1:1);
if_test=temp_ans(1:num);
if_tear=[zeros(1,num_low),temp_ans(num+1:end)];
decision=[if_test,if_tear];
```

```
current_result=RealCost(test,cost_change,decision,num);

if current_result(4)==1 && current_result(1)<min
min=current_result(1);
answer=[if_test;if_tear];
end
waitbar(i_try/(2^upper),bar,'waiting...');
end

toc;
disp(min);
disp(answer);

%% CO实际成本计算函数
function fun_back=RealCost(test,cost_change,decision,k)
flag=1;

% 计算节点k的前置层
% 寻找k的前置
kids=[];
costs=[];
ps=[];
backs=[];
for iii=1:k-1
if test(1,iii)==k
kids=[kids iii];
temp=RealCost(test,cost_change,decision,iii);
if flag==0
fun_back=[-1,-1,-1,0];
return;
end
costs=[costs temp(1)];
ps=[ps temp(2)];
backs=[backs temp(3)];
end
end
num_kids=length(kids);
if num_kids>0
% 临时量
temp_cost=zeros(1,num_kids);
r_back=0;
r_cost=0;

% 更新前置的形式成本
for iii=1:num_kids
temp_cost(iii)=(costs(iii)+decision(kids(iii))*test(4,kids(iii))) /
```

```
    (1-decision(kids(iii))*ps(iii))...
+decision(kids(iii))*ps(iii)/(1-ps(iii))
*decision(end/2+kids(iii))*(test(6,kids(iii))-backs(iii));
% k的返还费用
r_back=r_back+costs(iii);
% k的新费用
r_cost=r_cost+temp_cost(iii);
end
r_cost=r_cost+test(5,k);
% k的新损坏率
r_p=1-test(3,k);
for j=1:num_kids
r_p=r_p*(1-ps(iii)*(1-decision(kids(iii))));
end
r_p=1-r_p;
else
r_cost=test(2,k);
r_p=test(3,k);
r_back=0;
end

if k==size(test,2)
r_cost=(r_cost+decision(k)*test(4,k)) / (1-decision(k)*r_p)...
+decision(k)*r_p/(1-r_p)*decision(end/2+k)*(test(6,k)-r_back)...
+(1-decision(k))*r_p*(cost_change+decision(end/2+k)*(test(6,k)-r_back));
r_cost=r_cost/(1-(1-decision(k))*r_p);
end

if r_cost<best_node(k)
best_node(k)=r_cost;
elseif r_cost>best_node(k)*(1+goal)
flag=0;
end

fun_back=[r_cost,r_p,r_back,flag];
end
end
```

```
%============================================================
% 分层贪婪最小化B3前半问题
%============================================================
%% 采用分层贪婪做b-3
% 0.357134 秒
% 122.9904
% 是否检测: 0   0    0    0    0    0    0    0    0    0    1    0
% 是否拆解: 0   0    0    0    0    0    0    0    0    0    1    0
```

```
tic;
%% 输入
% 成本
cost=[2,8,12,2,8,12,8,12,0,0,0,0];
% 次品率
p_fail=[.1,.1,.1,.1,.1,.1,.1,.1,.1,.1,.1,.1];
% 检测成本
cost_test=[1,1,2,1,1,2,1,2,4,4,4,6];
% 装配成本
cost_set=[0,0,0,0,0,0,0,0,8,8,8,8];
% 拆解费用
cost_tear=[0,0,0,0,0,0,0,0,6,6,6,10];
% 下级合成材料
next_level=[9,9,9,10,10,10,11,11,12,12,12,0];
% 调换损失
cost_change=40;

%% 过程量
% 拆解返还费用
cost_back=[0,0,0,0,0,0,0,0,0,0,0,0];
% 形式成本
new_cost=cost;
% 形式次品率
new_p_fail=p_fail;

% 确定量
% 是否检测
if_test=[];
% 是否拆解
if_tear=[];


%% 尝试分层贪婪
%% 计算前置层
if_complete=zeros(1,length(next_level));
for ii=1:length(next_level)-1
k=next_level(ii);
if if_complete(k)==0 && k~=length(next_level) % 找到一个中间层
% 计算节点k的前置层
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
```

```
n=length(kids);

% 开始贪婪,需要决定n个是否拆解，n个是否检测
min=inf;
answer=[];
answer_k=[];
% 遍历
for i_try=0:2^(2*n)-1
temp_ans=bitget(i_try,2*n:-1:1);
for iii=1:n
if_test(kids(iii))=temp_ans(iii);
if_tear(kids(iii))=temp_ans(n+iii);
end

% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;

% 计算
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
new_cost(k)=new_cost(k)+cost_set(k);
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);

% 目标函数
if new_cost(k)/(1-new_p_fail(k))<min
% new_cost(k)*(1-new_p_fail(k))+new_p_fail(k)*(cost_tear(kids(i))-cost_back(kids(i)))
min=new_cost(k)*(1-new_p_fail(k));
answer=[if_test;if_tear];
answer_k=[new_cost(k),new_p_fail(k),cost_back(k)];
end
end
```

```
% 赋以最小值
if_test=answer(1,:);
if_tear=answer(2,:);
new_cost(k)=answer_k(1);
new_p_fail(k)=answer_k(2);
cost_back(k)=answer_k(3);

if_complete(k)=1;
end
end

%% 最终层贪婪
k=length(next_level);
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 贪婪,需要决定n个是否拆解, n个是否检测
min=inf;
answer=[];
answer_k=[];
for i_try=0:2^(2*n)-1
temp_ans=bitget(i_try,2*n:-1:1);
for iii=1:n
if_test(kids(iii))=temp_ans(iii);
if_tear(kids(iii))=temp_ans(n+iii);
end

% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;

% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
```

```
end
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);

new_cost(k)=new_cost(k)+cost_set(k);
new_cost(k)=(new_cost(k)+if_test(k)*cost_test(k)) / (1-if_test(k)*new_p_fail(k))...
+if_test(k)*new_p_fail(k)/(1-new_p_fail(k))*if_tear(k)*(cost_tear(k)-cost_back(k))...
+(1-if_test(k))*new_p_fail(k)*(cost_change+if_tear(k)*(cost_tear(k)-cost_back(k)));
new_cost(k)=new_cost(k)/(1-(1-if_test(k))*new_p_fail(k));
% 更新
if new_cost(k)<min
min=new_cost(k);
answer=[if_test;if_tear];
answer_k=[new_cost(k),new_p_fail(k),cost_back(k)];
end
end
% 赋以最小值
if_test=answer(1,:);
if_tear=answer(2,:);
new_cost(k)=answer_k(1);
new_p_fail(k)=answer_k(2);
cost_back(k)=answer_k(3);
toc;
disp(min);
disp(answer);
```

```
%============================================================
% 禁忌退火算法
%============================================================
layers=3;
test=test_B3;
cost_change=cost_change_B3;
[best_cost,best_solution]=hybrid_tabu_sa_optimization(test,cost_change,layers);
disp(best_cost);

%% 优化函数
function [best_cost,best_solution]=hybrid_tabu_sa_optimization(test,cost_change,layers)
% 参数设置
num_variables = size(test,2)*2-length(find(test(5,:)==0)); % 变量个数（二进制变量）
max_iterations = 1000;  % 最大迭代次数
max_del_iter=50;        % 找不到新的极值一定世界后直接结束
tabu_list_length = 100;  % 禁忌表长度
```

```
initial_temp = 100;    % 模拟退火初始温度
cooling_rate = 0.995;     % 模拟退火降温速率
neighborhood_size = 20;  % 邻域大小

% 初始化解（随机生成一个二进制向量）
current_solution = randi([0, 1], 1, num_variables);
best_solution = current_solution;
current_cost = objective_function(current_solution,test,cost_change);
best_cost = current_cost;
best_iter=1;

% 初始化禁忌表
tabu_list = zeros(tabu_list_length, num_variables);
tabu_index = 1;

% 初始化模拟退火温度
temperature = initial_temp;

% 迭代禁忌搜索与模拟退火
clf;hold on;grid on;tic;
for iteration = 1:max_iterations
% 生成邻域解并选择最优解（禁忌表外的解）
neighborhood = generate_neighborhood(current_solution, neighborhood_size);
best_neighbor = [];
best_neighbor_cost = inf;

for i = 1:neighborhood_size
neighbor = neighborhood(i, :);
if ~is_in_tabu_list(neighbor, tabu_list)
neighbor_cost = objective_function(neighbor,test,cost_change);
if neighbor_cost < best_neighbor_cost
best_neighbor = neighbor;
best_neighbor_cost = neighbor_cost;
end
end
end

% 如果找到了更优解，更新当前解
if ~isempty(best_neighbor)
% 模拟退火机制 - 以一定概率接受劣解
delta = best_neighbor_cost - current_cost;
if delta < 0 || rand() < exp(-delta / temperature)
current_solution = best_neighbor;
current_cost = best_neighbor_cost;

% 更新禁忌表
tabu_list(tabu_index, :) = current_solution;
```

```
    tabu_index = mod(tabu_index, tabu_list_length) + 1;
    end
    end

    % 更新全局最优解
    if current_cost < best_cost
    best_solution = current_solution;
    best_cost = current_cost;
    best_iter=iteration;
    end
    plot(iteration,best_cost,'r.');
    plot(iteration,current_cost,'b.');

    % 降低模拟退火温度
    temperature = temperature * cooling_rate;

    % 输出当前迭代的最优值
    fprintf('Iteration: %d, Best Cost: %.4f, Current Temperature: %.2f\n', iteration,
        best_cost, temperature);

    % 收敛后直接结束
    if iteration-best_iter>=max_del_iter
    break;
    end
    end
    toc;
    title(['退火禁忌-' num2str(layers) '工序-' num2str(length(find(test(5,:)==0))) '零配件'])
    xlabel("迭代数")
    ylabel("实际成本")
    legend('最小实际成本','当代最小实际成本')

    end

    %% 目标函数
    function cost = objective_function(solution,test,cost_change)
    %% 输入
    % 成本
    cost=test(2,:);
    % 次品率
    p_fail=test(3,:);
    % 检测成本
    cost_test=test(4,:);
    % 装配成本
    cost_set=test(5,:);
    % 拆解费用
    cost_tear=test(6,:);
    % 下级合成材料
```

```
next_level=test(1,:);
% 调换损失
% cost_change=40;

%% 过程量
% 拆解返还费用
cost_back=zeros(1,size(test,2));
% 形式成本
new_cost=cost;
% 形式次品率
new_p_fail=p_fail;

%% 确定量
% 是否检测
if_test=[];
% 是否拆解
if_tear=[];

%% 计算
temp_ans=solution;
if_test=temp_ans(1:size(test,2));
if_tear=[zeros(1,length(find(test(5,:)==0))),temp_ans(size(test,2)+1:end)];
% 是否完成计算
if_complete=zeros(1,length(next_level));
for ii=1:length(next_level)-1
k=next_level(ii);
if if_complete(k)==0 && k~=length(next_level)
% 计算节点k的前置层
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
```

```
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
new_cost(k)=new_cost(k)+cost_set(k);
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);
end
if_complete(k)=1;
end

% 计算最终层
k=length(next_level);
% 寻找k的前置
kids=[];
for i=1:k-1
if next_level(i)==k
kids=[kids i];
end
end
n=length(kids);
% 临时量
temp_cost=zeros(1,n);
cost_back(k)=0;
new_cost(k)=0;
% 更新前置的形式成本
for i=1:n
temp_cost(i)=(new_cost(kids(i))+if_test(kids(i))*cost_test(kids(i))) /
    (1-if_test(kids(i))*new_p_fail(kids(i)))...
+if_test(kids(i))*new_p_fail(kids(i))/(1-new_p_fail(kids(i)))
*if_tear(kids(i))*(cost_tear(kids(i))-cost_back(kids(i)));
% k的返还费用
cost_back(k)=cost_back(k)+new_cost(kids(i));
% k的新费用
new_cost(k)=new_cost(k)+temp_cost(i);
end
% k的新损坏率
new_p_fail(k)=1-p_fail(k);
for j=1:n
new_p_fail(k)=new_p_fail(k)*(1-new_p_fail(kids(i))*(1-if_test(kids(i))));
end
new_p_fail(k)=1-new_p_fail(k);
new_cost(k)=new_cost(k)+cost_set(k);
```

```
new_cost(k)=(new_cost(k)+if_test(k)*cost_test(k)) / (1-if_test(k)*new_p_fail(k))...
+if_test(k)*new_p_fail(k)/(1-new_p_fail(k))*if_tear(k)*(cost_tear(k)-cost_back(k))...
+(1-if_test(k))*new_p_fail(k)*(cost_change+if_tear(k)*(cost_tear(k)-cost_back(k)));
new_cost(k)=new_cost(k)/(1-(1-if_test(k))*new_p_fail(k));
cost=new_cost(k);
end

% 生成当前解的邻域解 (通过随机翻转指定个数的位)
function neighborhood = generate_neighborhood(solution, size)
num_variables = length(solution);
neighborhood = zeros(size, num_variables);

for i = 1:size
new_solution = solution;
flip_index = randi([1 num_variables]); % 随机选择一位翻转
new_solution(flip_index) = 1 - new_solution(flip_index); % 翻转位
neighborhood(i, :) = new_solution;
end
end

% 检查解是否在禁忌表中
function is_in_tabu = is_in_tabu_list(solution, tabu_list)
is_in_tabu = any(ismember(tabu_list, solution, 'rows'));
end
```

```
%============================================================
% B4: 使用蒙特卡洛算法重算B3
%============================================================
%% 输入第三题数据到test中
% 略

%% 蒙特卡洛
   num=1000;
solution_list=[];
bar=waitbar(0);
for mm=1:num
% 随机概率
test2=test;
for i=1:size(test,2)
if abs(test2(3,i)-0.05)<0.01
nnn=87;
elseif abs(test2(3,i)-0.1)<0.01
nnn=164;
else
nnn=292;
end
```

```
sigma=sqrt(test2(3,i)*(1-test2(3,i))/nnn);
test2(3,i)=normrnd(test2(3,i),sigma);
end

[best_cost,best_solution]=hybrid_tabu_sa_optimization(test2);
solution_list=[solution_list;num2str(best_solution)];
waitbar(mm/num,bar,'waiting...');
end

solution_list=str2num(solution_list);
maybe_answer=sum(solution_list)/num;

u=0.6;l=0.4;
pick=[];
for i=1:size(maybe_answer,2)
if maybe_answer(i)>l && maybe_answer(i)<u
pick=[pick,i];
elseif maybe_answer(i)<l
maybe_answer(i)=0;
else
maybe_answer(i)=1;
end
end

% 对pick进行相关度分析
solution_pick=solution_list(:,pick);
corr(solution_pick);

maybe_answer(pick(1))=(sign(maybe_answer(pick(1))-0.5)+1)/2;
for i=2:1:length(pick)
for j=1:i-1
if corr(j,i)>0.3
maybe_answer(pick(i))=maybe_answer(pick(j));
break;
elseif corr(j,i)<-0.3
maybe_answer(pick(i))=1-maybe_answer(pick(j));
break;
end
maybe_answer(pick(i))=(sign(maybe_answer(pick(i))-0.5)+1)/2;
end
end

%% 禁忌退火优化函数
function [best_cost,best_solution]=hybrid_tabu_sa_optimization(test)
% 略
end
```

```
% 目标函数定义
function cost = objective_function(solution,test)
% 略
end

% 生成当前解的邻域解 (通过随机翻转指定个数的位)
function neighborhood = generate_neighborhood(solution, size)
% 略
end

% 检查解是否在禁忌表中
function is_in_tabu = is_in_tabu_list(solution, tabu_list)
% 略
end
```