基于截尾序贯抽样检测模型的企业生产过程决策方案

摘要

某企业作为供应链中的一环，需要做好对上游供应商所提供零配件的次品检测及验收工作，同时须关注自身环节中成品组装以及流向市场的成品售后问题。本文基于题中所给相关数据信息，基于**截尾序贯抽样检测法**建立数学模型进行分析，为该企业制定合理的生产过程决策方案，实现更高收益。

对于问题一：要求设计检测次数尽可能少的抽样检测方案，确认配件的次品率。本文采用**截尾序贯抽样检测法**对此批零配件进行抽样，以降低**中心位置**、**分散度**等因素的影响，计算并绘制出本文抽样检测模型的 **OC 曲线**衡量模型的稳定性与可信度，**设置零配件序列**，推断抽样结果分布状态，设置截尾，从而进行序贯抽样。再设置零假设、备择假设用以判断是否接受零配件。最终求解出在 **95% 信度水平下**认定零配件次品率超过标称值，拒收该批零配件的最小检测次数为 **30 次**，**在 90% 信度水平下**认定零配件次品率不超过标称值，接收该批零配件的最小检测次数为 **36 次**，结果见表 1。

对于问题二：要求为企业生产过程的各个阶段做出决策。本文以生产整体利润最大化，成本最小化为目标。首先通过**动态规划**，将本题目标拆分，拆解为每种零配件的检测成本，成品的装配、检测、调换成本，不合格成品的拆解、丢弃成本等子目标，从而在企业生产过程的各个阶段进行决策，并设立决策指标记录决策。接着对模型进行优化，降低**计算复杂度**。最终对问题二表格中的 6 种情况给出具体的决策方案、决策依据、指标结果见表 3。

对于问题三：要求拓展模型的普适性，对于更多道工序和更多零配件的生产过程能够给出最佳的决策规划。本文基于问题二基础上建立模型，依旧采用最大利润为原目标，同时考虑了半成品、成品由非正品零配件（或半成品）装配时导致的次品率影响，重新计算了必要参数，并讨论了问题三中拆解费用的两种可能情况，最后得到了两种最佳决策方案、决策依据、指标结果见表 4。

对于问题四：要求建立抽样检测模型，通过抽样检测所得次品率作为问题二、三表格中的次品率，重新求解问题二、三。本文基于问题一的截尾序贯抽样检测模型，将问题二、三表格中的次品率为标称值，得出检测次数尽量少的抽样方案，计算新次品率，此外，由于问题二、三模型中曾考虑非正品零配件装配后对给定半成品或成品次品率的影响，而问题四中通过问题一模型求解出的产品、半成品、零配件截尾序贯抽样检测的次品率亦考虑了上述影响，故本文在问题四模型构建中调整了问题二、三的模型，除去了其中重新计算成品或半成品次品率的步骤。最终，通过截尾序贯抽样检测得出的次品率，重新规划了问题二、三的决策方案、依据与决策指标结果，结果见表 6和表 7。

关键字： 序贯假设检验　截尾序贯抽样　多工序制造过程　质量检验　动态规划

一、 问题重述

1.1 问题背景

某企业在生产某畅销电子产品，采购该产品的两种零配件用以装配成成品，并销售给用户。

在该产品的生产过程中，只要其中一个零配件不合格，则成品必然不合格。企业可以决定检测该批零部件的数量，控制检测费用的支出。由于装配等因素，装配出的成品也有一定的次品率，因此企业可以决定是否对成品进行检测。在出现次品时，企业可以选择报废或者拆解，拆解不会损坏零配件，但存在拆解费用。企业需要综合考虑多方面因素，平衡好购买成本、检测成本、次品拆解成本以及用户购买到不合格品时的调换损失与商品利润关系。在维护一定企业信誉的同时能保证较多盈利。具体生产过程如图 1。

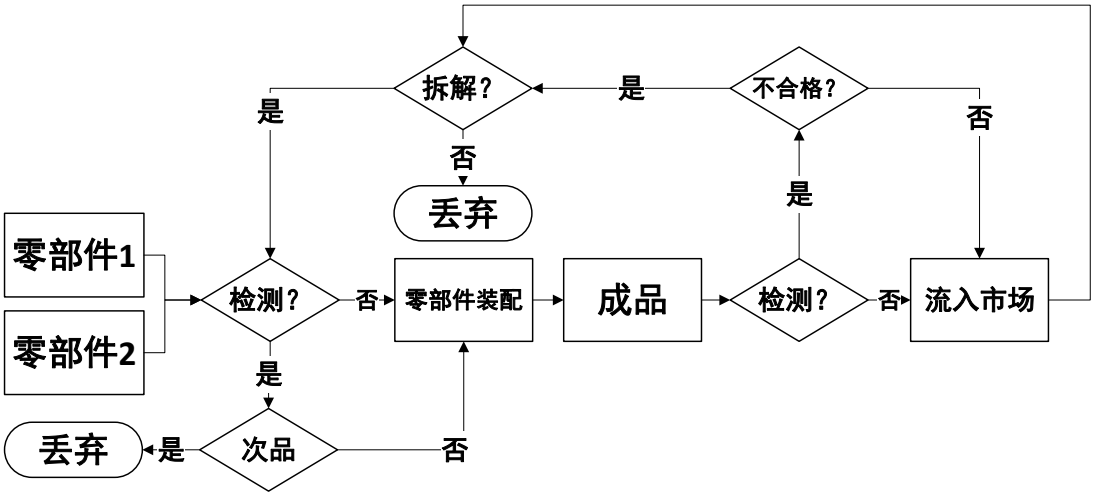

图 **1**　生产过程示意图

1.2 问题要求

问题 1：为企业设计检测次数尽可能少的抽样检测方案，检测零配件的次品率是否超过标称值。

问题 2：针对企业生产过程的各个阶段给出具体的决策方案，并给出决策的依据及相应的指标结果。

问题 3：在问题 2 基础上，探究在更复杂情况下，有 m 道工序、n 个零配件的生产过程的决策方案。

问题 4：探究若问题 2 及问题 3 中零配件、半成品和成品的次品率均是通过问题 1 中使用的方法得到的，重新给出决策以及决策方案

二、 问题分析

2.1 问题一分析

对于问题一，要求设计检测次数尽可能少的抽样检测方案，确认零配件的次品率是否超过标称值。由于需检测样本是无序的排列，且一批零配件是一个孤立批，若采用折半查找，会存在中心位置、分散度和偏离目标值等因素的影响，为减少各种影响因素造成的误差，本文通过截尾序贯抽样检测的方法对一批零件进行抽样，判断抽样结果分布情况并对不同信度下是否接收该批零配件进行假设检验 [1]，最终得出在 95% 的信度下拒收某批零配件时的检测次数与在 90% 的信度下接收某批零配件时的检测次数。此外，本文绘制截尾序贯抽样检测方案的 OC 曲线来衡量其表现效果与鉴别能力，证明了本文所设计的抽样检测方案具有较好的稳定性。

2.2 问题二分析

对于问题二，要求为企业生产过程的各个阶段做出决策，本文以最终的整体利润最大为目标，通过动态规划，将原目标拆分为：每种零配件的检测成本，成品的装配、检测、调换成本，不合格成品的丢弃、拆解成本等子目标函数，而后设置决策指标：企业零配件 i 是否检测，若检测验，$p_i = 0$，若不检测，$p_i = 1$；成品是否检测，若检测验，$P_i = 0$，若不检测，$P_i = 1$，不合格品否拆解，若拆解，则 $u = 0$，若不拆解，则 $u = 1$。问题二共有 $2^4 = 16$ 种决策方案，可通过状态转移方程，代入某种情况的参数，寻找到原目标的最优解。

在参数的运用中，本文注意到表 1 中成品的次品率是将正品零配件装配后的产品次品率，对于实际决策中，存在次品零配件装配为成品的情况，为提高模型精度，本文通过计算，将成品的次品率更新为任何零配件装配后的次品率，此外，为减少计算的复杂度，本文假设在首轮流水线作业后，后续重复流水线作业的决策方案不再改变，但已检测过的正品零配件不再进行重复检验。将表 1 所示的各种情况分别代入基于动态规划算法构建的决策模型，最终分别求出具体的决策方案、决策依据、指标结果。

2.3 问题三分析

对于问题三，要求给出 m 道工序、n 个零配件在生产过程中的决策方案，本文主要在模型二的基础上进行更复杂的决策规划，为提高模型的普适性，本文将问题三中零配件装配半成品、半成品装配成品两道工序分别视作视作问题二中零配件装配成品，调整

模型中的子目标为：计算每种零配件的检测成本 U_{ji}，每种成品（半成品）检测成本 F_{jv}，每种成品（半成品）的装配、拆解成本 E_{jki} 以及最终成品的总调换损失 β_0，由于问题三的决策方案有 $2^{8+3+2}=8192$，为避免局部最优解扰乱整体最优解，本文依旧采用最大利润作为原目标，通过整体最大利润得出最优的决策方案。

同时，本文考虑了半成品、成品由非正品零配件（或半成品）装配时导致的次品数这一影响，对表 2 中半成品与成品的次品率进行更新，得到更为准确的次品率，以提高模型精度。此外，本文亦讨论了问题三表格中成品的拆解费用是指将成品拆解为上一道工序的半成品费用，还是将成品完全拆解成零配件的费用，在两种可能情况下，依照工序顺序对零配件、半成品、成品进行生产，计算各种决策方案的总成本、总利润选择各自最大利润的决策方案。

2.4 问题四分析

问题四中假设问题二、三中产品、半成品、零配件的次品率均是通过抽样检测的方法得到，即需要建立一个合适的抽样检测模型，对问题二、三表格中产品、半成品、零配件的次品率进行更新，而后重新为企业的生产过程设立新的决策方案。本文主要基于问题一所建立的截尾序贯抽样检测方案，将问题二、三表格中产品、半成品、零配件各自的次品率作为其序贯抽样检测的标称值，在保证抽样检测可信度的情况下，求出检测次数尽量少的抽样方案，计算所得抽样方案在经过截尾序贯抽样检测后的抽样检测次品率，将新的次品率代入问题二、三所建立模型中，重新规划新的决策方案。

需要注意的是，本文问题二、三所建立的模型中，考虑了问题二、三所给表格中半成品（成品）的次品率是将正品零配件（半成品）装配后的次品率，建立了次品率的优化模型调整半成品（成品）的次品率为任意零配件（半成品）装配后的次品率。在问题四中，由问题一中模型求解得出的产品、半成品、零配件截尾序贯抽样检测的次品率已经表示任意零配件（半成品）装配后的次品率，无需再进行优化调整。

三、 模型假设

为简化问题，本文做出以下假设：

1. 企业购置每种零配件的数量一致，且合格的零配件在装配、拆解等生产过程中不会损坏。

2. 售出的不合格成品一定会被退回，产生调换损失。

3. 在首轮流水线作业后，后续重复流水线作业中，决策方案沿用首轮所作决策，不再改变。

4. 通过 $m \geq 2$ 道工序所得到的成品，成品的拆解费用为将成品直接拆解成所有的零配件的拆解费用。

四、 符号说明

符号	含义	单位
n	零配件的种类数	/
m	工序数	/
d	所抽样本中次品数量	个
q	所抽样本中正品数量	个
z	抽样的总样本量	个
Z	抽样次数	次
r	标称值	/
$\hat{r}$	抽样检测的次品率	/
S_Z	截尾序贯检测统计量	个
W	总利润	元
x	零配件的数量	个
a	零配件的次品率	/
b	零配件的购买单价	元
c	零配件的检测成本	元
y	零配件的正品率	/
A	成品的部分次品率	/
A'	成品的整体次品率	/
B	成品的装配成本	元
C	成品的检测成本	元
D	成品的拆解费用	元
α	成品的市场售价	元
β	成品的调换损失	元
$min\{x_1, x_2, ..., x_i\}$	成品的数量	个
Q	合格的成品数量	个
p, P, u	边界条件	/
θ	失效概率	/
ω	接收概率	/

五、 问题一的模型的建立和求解

5.1 模型简介

本文在问题一中，采用序贯抽样的抽样检测方法，在对抽样结果分布进行判别后，进行假设检验，以判断企业在不同信度下是否接收某批零件零配件，具体流程如下：

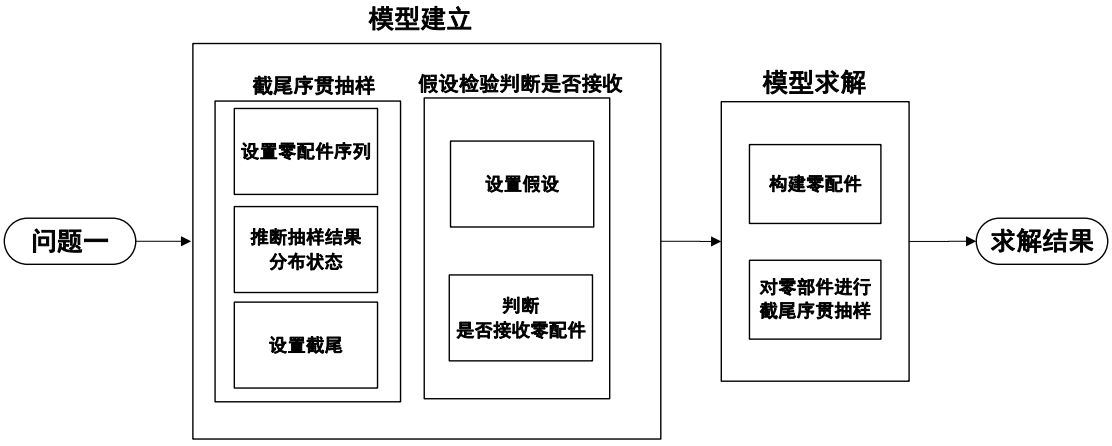

图 **2**　问题一流程图

5.2 模型建立

由于一批零配件中次品的分布完全随机，存在中心位置、分散度和偏离目标值等影响因素，本文通过序贯抽样检测的方法降低中心位置、分散度等因素的影响，从而提高模型的稳定性，同时，因为一批零配件是一个孤立批，属于 Dodge 和 Roming 于 1944 年提出的两类抽样中的 A 类抽样 [2]，本文通过计算并绘制出本文抽样检测模型的 OC 曲线衡量模型的稳定性与可信度。模型建立步骤如下：

5.2.1 截尾序贯抽样

Step 1：设置零配件序列

由于题设中并未明确指出零配件的数量与品质等指标，本文通过设置一行 0、1 的随机数列表示一个批次的零配件，其中 1 代表不合格零配件，0 代表合格的零配件，通过标称值 r, 可得 1 在随机数列中的占比为 r，0 的占比为 $1-r$。

Step 2：推断抽样结果分布状态

本文在零配件随机序列的设置中，次品率为标称值，在序贯抽样的初始阶段，样本的合格品和不合格品的数量可以视为二项分布。假设共进行了 Z 次抽样，总样本量为

z，次品率为 r，则其中次品数量 X 服从二项分布如式 (1):

$$X \sim Binomial(z, r) \tag{1}$$

由于本文选取截尾序贯抽样，具有停止规则，故抽样检测出 X 个次品所需的抽样的次数 Z 服从负二项分布如式 (2):

$$Z \sim NegativeBinomial(X, r) \tag{2}$$

当抽样的样本量较大时，根据中心极限定理，二项分布可以近似为正态分布。在本文次品率为 10% 的情况下，可以判定抽样结果符合近似正态分布如式 (3):

$$X \sim Z(zr, zr(1-r)) \tag{3}$$

其中，zr 是抽出次品数的期望值，$zr(1-r)$ 是标准差。

Step 3：设置截尾（抽样停止规则）

截尾是指在序贯抽样过程中，判断何时停止抽样的一种停止规则，截尾的设置与抽样结果的分布状态有关，若抽样结果服从二项分布，可以通过计算在 r 二项分布的累积分布函数（CDF）来确定截尾的上下边界，设定上边界 μ，下边界为 μ'，信度为 t 使得：

$$\begin{cases} \psi(X \leq \mu | z, r) \geq 1 - t \\ \psi(X \geq \mu' | z, r) \geq 1 - t \end{cases} \tag{4}$$

当样本量 z 较大时，抽样结果服从近似正态分布，可以设定上边界 UCL、下边界 LCL 为：

$$\begin{cases} \mathrm{UCL} = \bar{\mathrm{x}} + \mathrm{z_t} \cdot \frac{\mathrm{s}}{\sqrt{\mathrm{z}}} \\ \mathrm{LCL} = \bar{\mathrm{x}} - \mathrm{z_t} \cdot \frac{\mathrm{s}}{\sqrt{\mathrm{z}}} \end{cases} \tag{5}$$

其中 $\bar{x}$ 表示样本均值，s 表示标准差，z_t 是标准正态分布的临界值。

Step 4：序贯抽样

序贯抽样检测相较于其他种类抽样方案，在每次抽样后，会根据当前的样本结果符合要求与否决定是否继续抽样。对于所设立的随机数列，本文通过对总序列进行 Z 轮随机抽样，每轮随机抽样 μ 个零配件进行检测，最终总样本量为 $z = Z\mu$，其会随着每次抽样而变化，本文在此设定一个样本量的最大值 $z_{max} = 1000$，以避免无限抽样。

5.2.2 假设检验判断是否接收

对于截尾序贯抽样检测的结果的次品率 $\hat{r}$，为判定其是否达到预定的标称值，需要进行假设检验，具体步骤如下：

Step 1：设置假设

本文设置零假设 H_0：$\hat{r}$=r_0，设置备择假设 H_1：$\hat{r}$=r_1，$(r_0 > r_1)$。

其中，r_0 表示产品的接收质量限(AQL)，为第一类错误的发生概率 $r_0 = \psi\{X > c|\hat{r} \leq r\}$，$r_1$ 表示企业的风险质量，为第二类错误发生概率 $r_1 = \psi\{X \leq c|\hat{r} > r\}$。

Step 2：判断是否接收零配件

记 X_{hi} 为第 h 轮抽样中第 i 个抽样结果，且 $X_{hi} = 1$ 表示抽到次品，$X_{hi} = 0$ 表示抽到正品，$S_Z = \sum_{h=1}^{Z} X_h$ 为序贯检验统计量，表示前 Z 次抽样中累计抽到次品的数量，对截尾序贯抽样检测，可以设 $M = \inf\{Z|S_Z \geq U_Z or S_Z \leq L_Z, Z = 1, 2, ...Z_{max}\}$，当 $S_M \leq L_M$ 时，停止抽样并拒绝零假设 H_0，拒收零配件，当 $S_M \geq U_M$ 时，停止抽样并接受零假设 H_0，即接收该批零配件，若 $L_M \leq S_M \leq U_M$ 时，不能判定，需进行下一轮的抽样。其中，$\{L_1, L_2, ..., L_{Z_{max}}\}$ 与 $[\{U_1, U_2, ...U_{Z_{max}}\}$ 为两个单调递增的整数列并满足式 (6)：

$$\begin{cases} L_i + 2 \leq U_i \quad i = 1, 2, ..., Z_{max} \\ L_{Z_{max}} + 1 = U_{Z_{max}} \end{cases} \tag{6}$$

这是截尾的上下边界，即停止规则。

5.3 模型求解

Step 1：构建一批零配件进行截尾序贯抽样检测

本文设置一批零配件的总数为 1000，包含 100 个次品零配件，900 个正品零配件，从而符合次品率的标称值 $r = 10\%$。由于本文所用抽样方法含有停止条件，则抽样检测结果服从负二项分布式（负二项，在抽样样本量较大（通常大于 30）时可以视为大样本，抽样检测结果服从近似正态分布式（正态分布）。由对不同分布的截尾序贯抽样检测结果动态设置不同的阈值 (4) 和 (5)，得到在 95% 信度水平下认定零配件次品率超过标称值，拒收该批零配件的检测轮数 $Z_{0.95} = 3$，每轮检测最优样本量为 $\mu_{0.95} = 16$，在 90% 信度水平下认定零配件次品率不超过标称值，接收该批零配件的检测轮数 $Z_{0.90} = 3$，每轮检测最优样本量为 $\mu_{0.90} = 12$。

Step 2：假设检验

设置零假设 H_0：$\hat{r}$=r_0 与备择假设 H_1：$\hat{r}$=r_1，$(r_0 > r_1)$，

通过 $S_M \geq U_M$ 判定 95% 信度水平下该批零配件时的次品率拒绝零假设 H_0，此时，截尾序贯抽样检测的次品率 $\hat{r}_{0.95} = 12.50\%$。通过 $S_M \leq L_M$ 判定 90% 信度水平下该批零配件时的次品率接收零假设 H_0，此时，截尾序贯抽样检测的次品率 $\hat{r}_{0.90} = 8.33\%$。

5.4 求解结果

最终求解出对一批零配件，在 95% 信度水平下认定零配件次品率超过标称值，拒收该批零配件的最小检测次数为 30 次，在 90% 信度水平下认定零配件次品率不超过标称值，接收该批零配件的最小检测次数为 36 次，具体结果如表 1：

表 **1**　检测次数尽可能少的抽样检测方案

情形	次品率 $\hat{r}$	每轮最优样本量 μ	检测轮数 Z	最小检测次数 z
在 95% 信度下拒收	12.50%	16	3	48
在 90% 信度下接受	8.33%	12	3	36

分析表 1中数据可以得出，在标称值为 10% 时，若信度为 95%，企业通过截尾序贯抽样检测抽取 3 轮，每轮抽取 16 个零配件，最终零配件截尾序贯抽样检测的次品率超过标称值，即拒收这批零配件。若信度为 90% 企业通过截尾序贯抽样检测抽取 3 轮，每轮抽取 12 个零配件，最终零配件截尾序贯抽样检测的次品率不超过标称值，即接收这批零配件。

六、 问题二的模型的建立和求解

6.1 模型简介

本文先设置了最终目标为成品成功售出的总利润最大，基于动态规划，定义决策变量有每种零配件的检测成本，成品的装配、检测、调换成本，不合格成品的丢弃、拆解成本等，计算 16 种决策方案各自的总利润，得出总利润最大的决策方案，输出这种决策方案的总利润、总成本等指标结果，具体流程图如图 3：

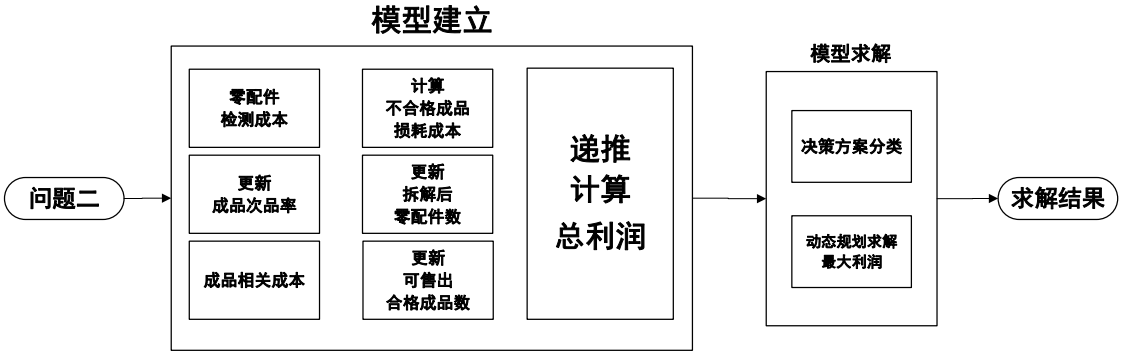

图 **3**　问题二流程图

6.2 模型建立

由于总利润受各种成本，售卖收益的影响，可以通过动态规划将一个目标分解为几个更简单的子目标，基于解决子目标得出的结果，来构建出原目标的解。本文中，企业看重的因素有净收益、企业信誉、成本等因素，故本文设置原目标为总利润最大 $\max W$，

则子目标有零配件检测、购买等相关成本 U_{ki}，成品装配、检测、调换等相关成本 F，不合格成品的拆解、丢弃等相关成本 E，总收益 $Q\cdot\alpha$ 等子目标，其中各个目标的状态转移方程与计算步骤如下：

Step1：零配件检测成本

零配件相关成本的计算主要在于“是否检测零配件”这两种决策下各自的相关成本，具体计算公式如下：

$$U_{ki}=\begin{cases}c_i x_{ki} & p_i=0\\ 0 & p_i=1\end{cases} \tag{7}$$

其中，$p_i=0$ 表示对零配件 i 进行检测，$p_i=1$ 表示不对零配件 i 进行检测，两者为边界条件，U_{ki} 表示零配件 i 在第 k 轮流水线作业的相关成本，c_i 表示零配件 i 的检测成本，x_{ki} 表示第 k 轮流水线作业零配件 i 的数量，故 $c_i x_{ki}$ 表示第 k 轮流水线作业中零配件 i 的检测成本。

Step 2：更新成品的次品率

在成品装配后，由于表 1 中成品的次品率是将正品零配件装配后的产品次品率，本文在此设变量 y_i 表示零配件 i 的正品率，其计算公式如式(8)：

$$y_i=\begin{cases}p_i(1-a_i)+1 & p_i=0\\ p_i(1-a_i) & p_i=1\end{cases} \tag{8}$$

而后，通过计算，更新成品次品率为任意状况下的次品率，以提高模型的精度，为简化问题，本文假设后续拆解后得到的零配件重复曾进行的决策进行生产，从而确保成品次品率只需在首轮流水线作业中更新一次，后续不再改变。由任意状况零配件装配的成品次品率具体计算公式如式(9)：

$$A_0{}'=A_0\prod_{i=1}^{n}y_i+1-\prod_{i=1}^{n}y_i \tag{9}$$

其中，A_0' 表示更新后的成品次品率，n 表示零配件种类数，A_0 表示问题二表格中的成品次品率。

Step 3：成品相关成本

成品相关成本的计算主要在于“是否对成品进行检测”两种决策下各自的相关成本，若检测，则成品相关成本中就包含检测成本、装配成本，若不检测，则成品相关成本中包含次品调换成本、装配成本，具体计算公式如式(10)：

$$F_0=\begin{cases}B_0\min\{x_{k1},x_{k2},...,x_{ki}\}+C_0\min\{x_{k1},x_{k2},...,x_{ki}\} & P_0=0\\ B_0\min\{x_{k1},x_{k2},...,x_{ki}\}+\beta A_0{}'\min\{x_{k1},x_{k2},...,x_{ki}\} & P_0=1\end{cases} \tag{10}$$

其中，$P_0=0$ 表示对成品进行检测，$P_0=1$ 表示不对成品进行检测，两者为边界条件 F_0 表示成品相关成本，$\min\{x_{k1}x_{k2}...x_{ki}\}$ 表示成品数，B_0 表示成品的装配成本，C_0

表示成品的检测成本，β 表示调换损失。

Step 4：不合格成品的损耗成本

对于不检测成品时最终调换回来的不合格成品和检测成品时得到的不合格成品，两种决策中的不合格成品数是一样的，为 $A_0'\min\{x_{k1}x_{k2}...x_{ki}\}$，其拆解成本的差异主要在于“是否拆解不合格成品”这两种决策，若拆解，需考虑拆解费用这一成本，若不拆解，则损耗成本为 0，具体计算公式如式(11):

$$E_0=\begin{cases}A_0{}'\min\{x_{k1},x_{k2},...,x_{ki}\}\cdot D_0 & u_0=0\\ 0 & u_0=1\end{cases} \tag{11}$$

其中，$u_0=0$ 表示拆解不合格成品，$u_0=1$ 表示不拆解不合格成品，两者为边界条件，E_0 表示拆解成本，D_0 表示成品的拆解费用。

Step 5：更新拆解后各零配件数量与可售出合格成品数

对于拆解不合格成品的这一决策，会增加剩余各种零配件的数量，因此在进行拆解与否的决策后，需要更新零配件 i 的数量 x_{ki}，计算公式如(12)：

$$x_{(k+1)i}=x_{ki}-\min\{x_{k1}x_{k2}...x_{ki}\}+(1-u)A_0'\min\{x_{k1}x_{k2}...x_{ki}\} \tag{12}$$

对于更新后的零配件数量，若所有零配件数量均大于 0（$\forall x_{(k+1)i}>0$）则可以再进行一轮检测与装配，则流水线的轮数 $\lambda=\lambda+1$ （λ 的初始值为 1），即重复所选决策，再进行一轮检测、装配、拆解等步骤，若任意一种零配件数量小于等于 0（$\exists x_{(k+1)i}\le 0$）则无法再进行新一轮成品的检测与装配。对于最终的合格成品数 Q，定义初始值 $Q=0$，在每一轮流水线作业后更新其数量，计算公式如式(13)：

$$Q=\sum_{k=1}^{\lambda}\left(1-A_0{}'\right)\min\{x_{k1},x_{k2},...,x_{ki}\} \tag{13}$$

由于新一轮流水线作业中的决策方案不变，首轮更新后的成品次品率不变。

Step 6：递推计算总利润

建立子问题之间的关系，通常通过一个递推公式来表示，通过上述各种子目标的计算，构建状态转移方程，通过递推公式来计算利润，本文所构建的递推公式如式(14)：

$$W=\left[\sum_{k=1}^{\lambda}\left(Q\cdot\alpha-\sum_{i=1}^{n}b_ix_{1i}-F-E\right)-\sum_{i=1}^{n}\left(\sum_{k=1}^{\lambda}U_{ki}\right)\right] \tag{14}$$

其中，W 表示总利润，α 表示产品的市场售价，b_ix_{1i} 表示零配件 i 的总购买成本，a_i 表示零配件 i 的次品率，b_i 表示零配件 i 的购买单价，$[\,]$ 为取整符号，通过式(14)可求出问题二中某种情况下利润最大的决策方案，即求出目标函数 $\max W$。

6.2.1 模型优化

由于文本采取多轮检测、装配等流水线作业，以达到再无可装配为成品的冗余零配件的目的，由于本文假设合格的零配件在装配、拆解等生产过程中不会损坏，故如果某种零配件曾检测过，就无需再次检测，在这种假设下，可对状态转移方程进行优化，以降低计算的复杂度，优化后的递推公式如式(15)：

$$W = \left[\sum_{k=1}^{\lambda}\left(Q\cdot\alpha - \sum_{i=1}^{n} b_i x_{1i} - \sum_{i=1}^{n} U_{1i} - F - E\right)\right] \tag{15}$$

优化基于决策指标 p_i，若零配件 i 在首轮装配中选择检测，有 $p_i = 0$，以避免重复检测零配件的情况出现，若零配件 i 在首轮装配中不选择检测，有 $p_i = 1$，选择不检验零配件。

6.3 模型求解

对于企业生产过程种各个阶段的决策，取决于利润最大目标函数，初始零配件数量的差异也会对模型求解结果产生影响，为简化问题，本文中所有零配件最初购进的数量相同，均为 100 个。

6.3.1 决策方案分类

在问题二中，仅有 1 道工序，2 个零配件，故在生产过程中存在 $2^4 = 16$ 种决策方案，详情如表 2。

6.3.2 动态规划求解最大利润

通过本节构建的模型，代入问题二表格中所给定的参数，计算出每种情况下的 16 种决策方案的总利润如下：

观察图 4可以发现，在 6 种情况下所有决策方案各自的总利润这一折线图中，可能出现某种情况下几种不同决策方案总利润近似，导致难以直接从折线图分辨出总利润最大的决策方案，故本文分别绘制了每种情况下 16 种决策方案的总利润直方图。本文以情况 1 与情况 3 为例，分别绘图如下：

由图 5可以发现，情况 1 中决策方案 3 的总利润最高，为 3748.0 元，即在情况 1 下，企业在生产过程中应选择检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品的决策方案。

由图 6可以发现，情况 3 中决策方案 1 和决策方案 3 的总利润最高，均为 3512.0 元，此时需计算两种决策方案各自的总成本，选择总成本较低的一种决策方案。计算得到，此时决策方案 1 和决策方案 3 的总成本相同，即在情况 3 下，企业在生产过程中可以选

表 **2**　问题二的 **16** 种决策方案

决策方案	零配件 1 是否检测	零配件 2 是否检测	成品是否检测	不合格成品是否拆解
1	是	是	是	是
2	是	是	是	否
3	是	是	否	是
4	是	是	否	否
5	是	否	是	是
6	是	否	是	否
7	是	否	否	是
8	是	否	否	否
9	否	是	是	是
10	否	是	是	否
11	否	是	否	是
12	否	是	否	否
13	否	否	是	是
14	否	否	是	否
15	否	否	否	是
16	否	否	否	否

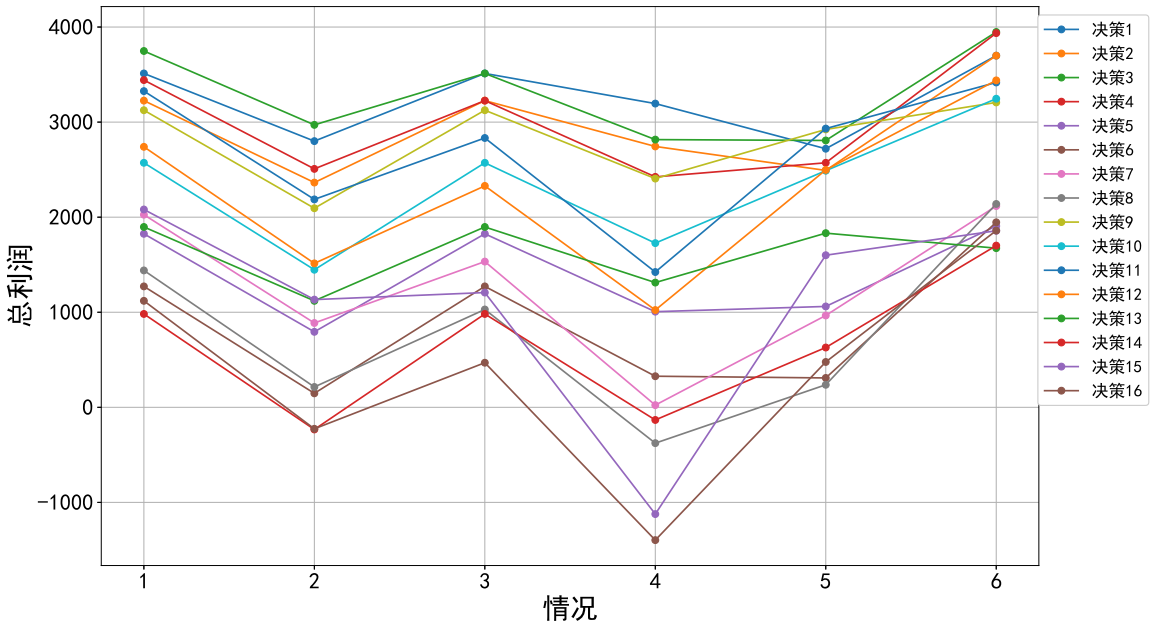

图 **4**　**6** 种情况下所有决策方案各自的总利润

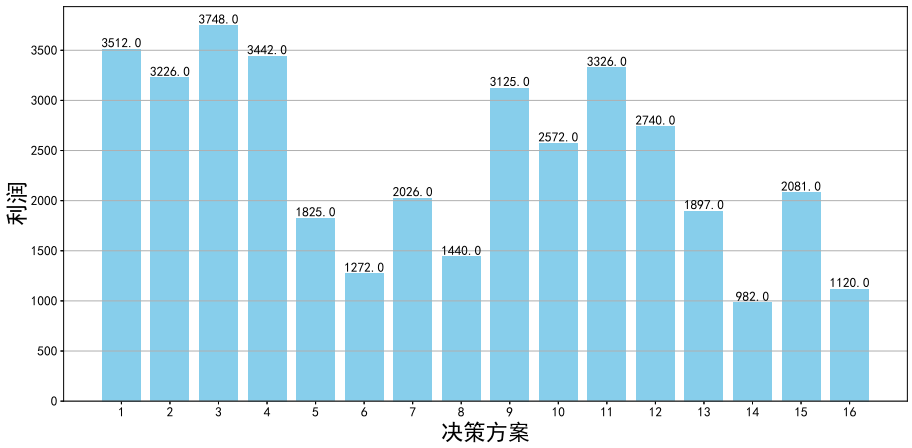

图 **5**　不同决策方案在情况 **1** 下的总利润直方图

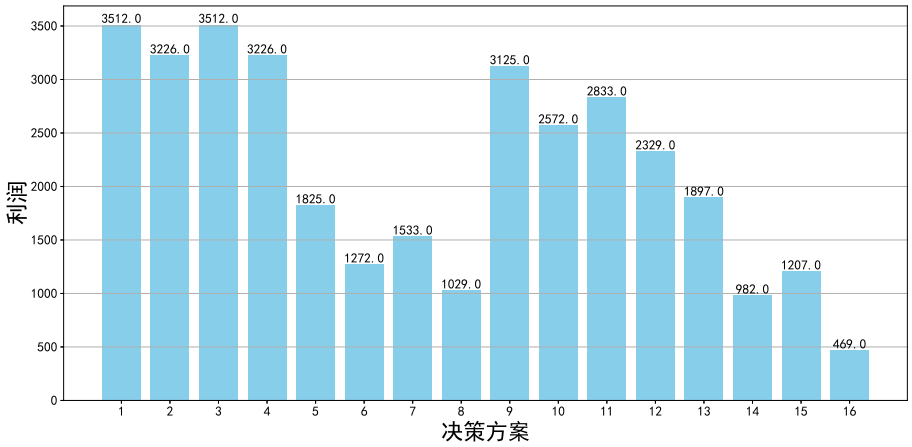

图 **6**　不同决策方案在情况 **3** 下的总利润

择检测零配件 1，检测零配件 2，拆解不合格成品，检测成品与否皆可的决策方案。两种决策方案的区别主要在于是否检测成品，若不检测成品，会有不合格的成品售予顾客，虽然调换损失和检测成本两者在成本上的影响相同，但产生调换损失的同时会影响企业信誉，在实际生活中可能会对成品的销售额产生影响，因此本文在情况 3 下选择决策方案 1：检测零配件 1，检测零配件 2，拆解不合格成品，检测成品作为生产过程中的决策方案。

6.4 求解结果

对于问题二中的 6 种情况，通过动态规划模型计算，寻找出的最佳决策方案与决策指标结果如下：表（2）6 种情况下各自最佳的决策方案与决策指标

由表 3可知情况 1、2、6 应选择决策方案 3（检测零配件 1，检测零配件 2，不检测

表 **3**　**6** 种情况下各自最佳的决策方案与决策指标

情况	决策方案	零配件 1 是否检测	零配件 2 是否检测	成品 是否检测	成品 是否拆解	总成本 (元)	总利润 (元)
1	3	是	是	否	是	1139	3748
2	3	是	是	否	是	1156	2971
3	1	是	是	是	是	1355	3512
4	1	是	是	是	是	920	3195
5	11	否	是	否	是	1208	2932
6	3	是	是	否	是	1307.5	3948

成品，拆解不合格成品）进行生产，情况 3、4 应选择决策方案 1（检测零配件 1，检测零配件 2，检测成品，拆解不合格成品）进行生产，情况 5 应选择决策方案 11（不检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品）进行生产，以达到每种情况下的总利润最大。决策依据是总利润最大，若总利润相同时，选择总成本较小的决策方案。

七、 问题三的模型的建立和求解

7.1 模型简介

本文在问题三中主要优化了问题二所建立的动态规划模型，调整了模型中的子目标为：E_{jki}、F_{jki}、E_{jki}、β_0，通过计算各种决策方案的整体总利润这一决策指标，寻找其中的最大总利润并选择其对应的决策方案进行企业生产。具体流程图如下：

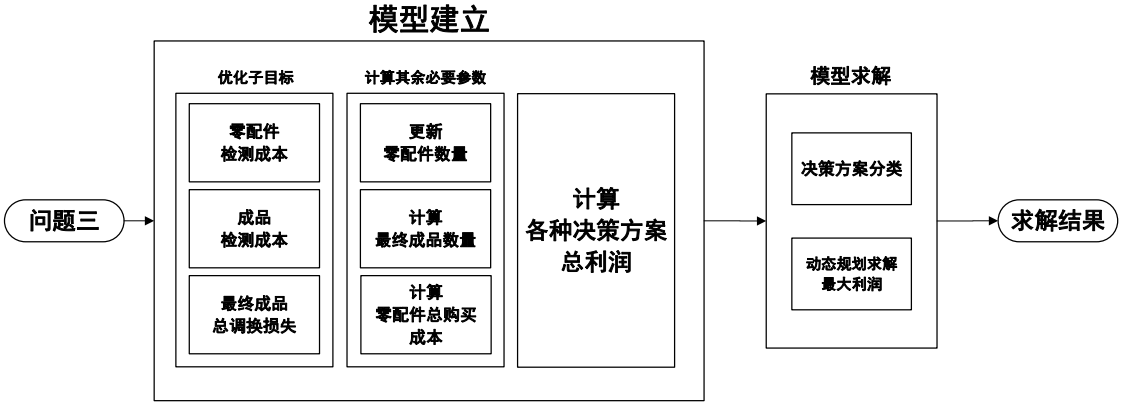

图 **7**　问题三流程图

7.2 模型建立

7.2.1 优化子目标

对于 m 道工序、n 个零配件在生产过程中的决策方案，生产流程如图所示图 8：

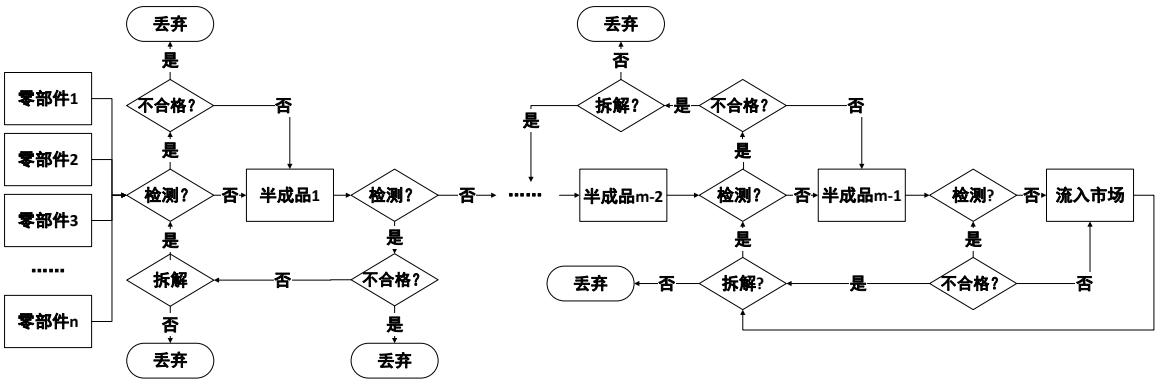

图 **8**　问题三生产流程示意图

可将半成品装配成品视作新的零配件装配成品工序，设第 j 道工序中”零配件“（初始零配件或视作零配件的半成品）的种类数为 n_j，第 j 道工序中“成品”（最终成品或视作成品的半成品）种类数为 N_j，第 j 道工序中装配成品（最终成品或视作成品的半成品）v 所需零配件（初始零配件或视作零配件的半成品）种类数为 S_jv，则有 $n_j = \sum_{v=1}^{N_j} S_{jv}$，优化后的子目标如下：

“零配件”i 的检测成本 U_{jki}

$$U_{ji} = \begin{cases} c_{1i}x_{11i} & p_{ji} = 0, P_{(j-1)v} = 0 \\ 0 & p_{ji} = 1, P_{(j-1)v} = 1 \end{cases} \tag{16}$$

其中，$p_{ji} = 0$ 表示在第 j 道工序中对零配件 i 进行检测，$p_{ji} = 1$ 表示在第 j 道工序中不对零配件 i 进行检测，$P_{(j-1)v=0}$ 表示 $j-1$ 道工序中的成品已检验，$P_{(j-1)v=0}$ 表示 $j-1$ 道工序中的成品未检验，它们均为边界条件，U_{ji} 表示在第 j 道工序中零配件 i 的总检测成本，c_{ji} 表示在第 j 道工序中每个零配件 i 的检测成本，x_{jki} 表示在第 j 道工序中第 k 轮流水线作业中零配件 i 的个数，为降低检测成本，在第 k 轮流水线作业中，若某种零配件曾检测过，就无需再次检测，则零配件的检测成本仅与首轮流水线作业的零配件数量 x_{j1i} 有关，在第 j 道工序中便无需检验，若未检验，保持原决策。

首先计算“成品”v 的检测成本 F_{jki}：

$$F_{jv} = \begin{cases} \sum_{k=1}^{\lambda} C_{jv} \min\{x_{jk1}, x_{jk2}, ..., x_{jki}\} & P_{jv} = 0 \\ 0 & P_{jv} = 1 \end{cases} \tag{17}$$

其中，$P_{jv} = 0$ 表示在第 j 道工序中对成品进行检测，$P_{jv} = 1$ 表示在第 j 道工序中不对成品进行检测，两者为边界条件，F_{jv} 表示第 j 道工序中成品 v 的检测成本，

$\min\{x_{jk1}, x_{jk2}, ..., x_{jki}\}$ 表示在第 j 道工序中第 k 轮流水线作业的成品数，C_{jv} 表示在第 j 道工序中成品 v 的检测成本

接着计算成品 v 的装配、拆解成本 E_{jki}：

$$E_{jv} = \begin{cases} B_{jv} \min\{x_{jk1}, x_{jk2}, ...x_{jki}\} + A'_{jv} \min\{x_{jk1}, x_{jk2}, ..., x_{jki}\} \cdot D_{jv} & u_{jv} = 0 \\ B_{jv} \min\{x_{jk1}, x_{jk2}, ...x_{jki}\} & u_{jv} = 1 \end{cases} \tag{18}$$

其中，$u_{jv} = 0$ 表示在第 j 道工序中拆解不合格的成品 v，$u_0 = 1$ 表示在第 j 道工序中不拆解不合格的成品 v，两者为边界条件，E_{jv} 表示成品 v 的拆解成本，B_{jv} 表示在第 j 道工序中成品 v 的装配成本，D_{jv} 表示在第 j 道工序中成品 v 的拆解费用，A'_{jv} 表示在第 j 道工序中更新后成品 v 的次品率。对于 A'_{jv}，主要受装配成品 v 的几种零配件次品率的影响，为简化问题，本文假设后续拆解后得到的零配件重复曾进行的决策进行生产，从而确保成品次品率仅在首轮流水线作业中更新一次，后续不再改变，具体计算步骤如下：

$$y_{ji} = \begin{cases} 1 & p_{ji} = 0 \\ 1 - a_{ji} & p_{ji} = 1 \end{cases} \tag{19}$$

其中，y_{ji} 表示在第 j 道工序中零配件 i 的正品率，a_{ji} 表示在第 j 道工序中零配件 i 的次品率。

$$A'_{jv} = A_{jv} \prod_{i=1}^{n_j} y_{ji} + 1 - \prod_{i=1}^{n_j} y_{ji} \tag{20}$$

其中，A_{jv} 表示在第 j 道工序中成品 v 在问题三表格中的原始次品率。最终成品的总调换损失 β_0 为

$$\beta_0 = \sum_{k=1}^{\lambda} \sum_{i=i}^{S_{mv}} \beta_{mv} A'_{mv} \min\{x_{mk1}, x_{mk2}, ..., x_{mki}\} \tag{21}$$

其中，由于共有 m 道工序，则 S_{mv} 表示最后一道工序中装配最终成品 v 所需要的零配件种类数，即直接装配出最终成品的半成品种类数，$\min\{x_{mk1}, x_{mk2}, ..., x_{mki}\}$ 表示最终成品的数量，A'_{mv} 表示更新后的最终成品次品率，β_{mv} 表示最终成品的调换损失。

7.2.2 计算其余必要参数

Step 1：更新配件数量

由于拆解不合格成品的决策会增加剩余各种零配件的数量，因此在进行拆解与否的决策后，需要更新第 j 道工序中零配件 i 在第 $k+1$ 轮流水线作业中的数量 x_{jki}，计算公式如下

$$x_{j(k+1)i} = x_{jki} - \min\{x_{jk1}, x_{jk2}, ..., x_{jki}\} + (1 - u_{jv}) A'_{jv} \min\{x_{jk1}, x_{jk2}, ..., x_{jki}\} \tag{22}$$

对于更新后的零配件数量，若所有零配件数量均大于 0（$\forall x_{j(k+1)i} > 0$）则可以再进行一轮检测与装配，流水线的轮数 $\lambda = \lambda + 1$（λ 的初始值为 1），即重复所选决策，再

进行一轮检测、装配、拆解等步骤，若任意一种零配件数量小于等于 0（$\exists x_{j(k+1)i} \leq 0$）则无法再进行新一轮成品的检测与装配。

Step 2：计算最终成品数量

对于合格的最终成品数量 Q，定义初始值 $Q=0$，在每一轮流水线作业后更新其数量，计算公式如下

$$Q=\sum_{k=1}^{\lambda}\sum_{i=i}^{S_{mv}}(1-A'_{mv})\min\{x_{mk1},x_{mk2},...,x_{mki}\} \tag{23}$$

Step 3：计算零配件总购买成本

在生产过程中，企业首先需要购买零配件才能进行生产，无论零配件是正品还是次品，其总购买成本仅仅与零配件的购买数量 x_{1ki} 相关，零配件总购买成本 R 的计算公式如下

$$R=\sum_{i=1}^{n_1}(x_{11i}b_{1i}) \tag{24}$$

7.2.3 计算各种决策方案的总利润

本文计算各种决策方案的总利润是基于问题二所建立模型基础上，优化子目标，改进递推公式得到的，具体计算过程如下：

$$W=Q\cdot\alpha-R-\beta_0-\sum_{j=1}^{m}\sum_{v=1}^{N_j}\sum_{i=1}^{S_{jv}}U_{ji}-\sum_{j=1}^{m}\sum_{v=1}^{N_j}\sum_{k=1}^{\lambda}E_{jv}-\sum_{j=1}^{m}\sum_{v=1}^{N_j}F_{jv} \tag{25}$$

根据式(25)计算各种决策方案的总利润可以得出总利润最高的决策方案。

7.3 模型求解

由于企业生产过程种各个阶段的决策，取决于利润最大目标函数，初始零配件数量的差异也会对模型求解结果产生影响，为简化问题，假设所有零配件最初购进的数量相同，均为 100 个，在此基础上进行问题三的求解。此外，问题三在 $m=2$，$n=8$ 的情况下，共有 $2^{14}=16384$ 种决策方案，从而决定是否检测“零配件”i、“成品”v，是否拆解所有不合格半成品以及是否拆解不合格成品，所得 16384 种决策方案见附录 A。

Step1：通过动态规划求出各种决策方案的总利润

本文基于问题二优化出普适性更好的动态规划模型，可以计算出 m 道工序、n 种零配件的生产过程中各种决策方案的总利润，由问题三可得 $m=2$，$n=8$，$n_1=8$，$n_2=3$，$N_1=3$，$N_2=1$，$S_{11}=3$，$S_{12}=3$，$S_{13}=2$，$S_{21}=3$，根据所得的各种参数，计算 2 道工序，8 种零配件的所有决策方案的总利润，结果如图 9：

观察图 9，较难明确看出哪种决策方案的总利润最大，本文通过绘制部分总利润较高决策方案的总利润直方图，以便于选取最大利润的决策方案。

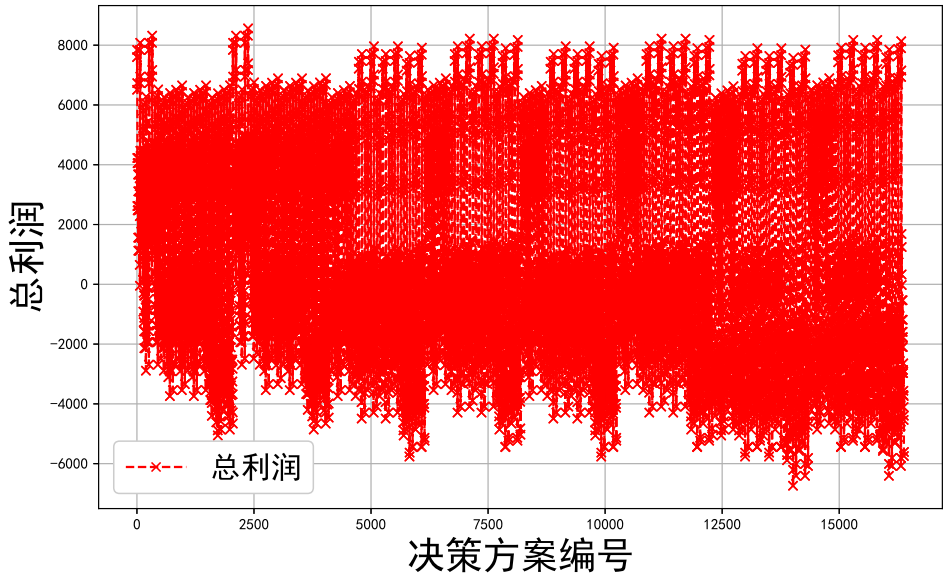

图 **9**　各个决策方案的总利润

Step2：选取最大利润的决策方案

由于问题三的决策方案众多，可能出现某些决策方案的总利润相同的情况，本文对所有决策组合的总利润倒序排列，选取前 10 个绘制其利润直方图如图 10。

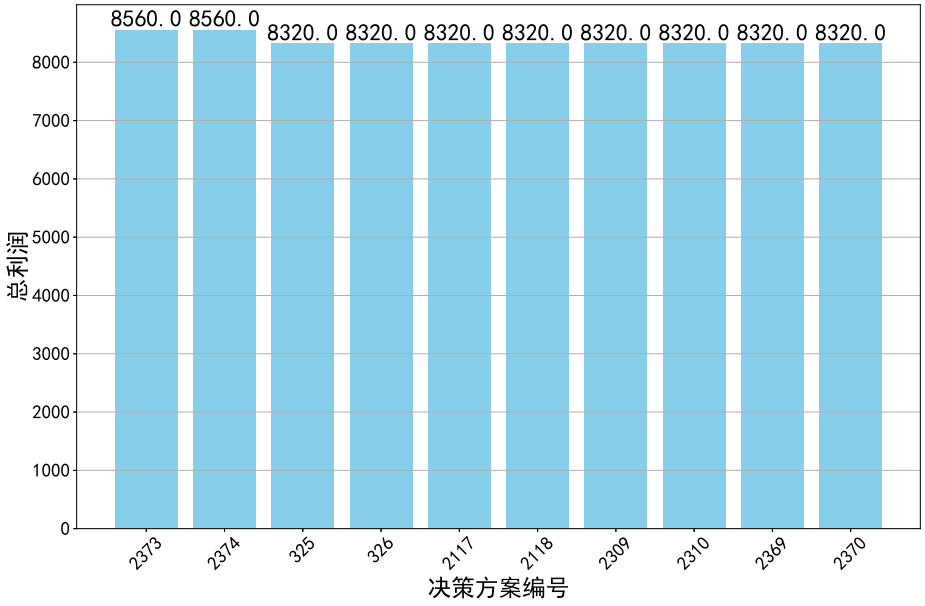

图 **10**　总利润最大的前 **10** 个决策方案

分析图 10可得，决策方案 2373 与 2374 的总利润相同且最高，需比较两种决策方案的成本，绘制图 10中 10 种决策方案的总成本直方图如图 11。

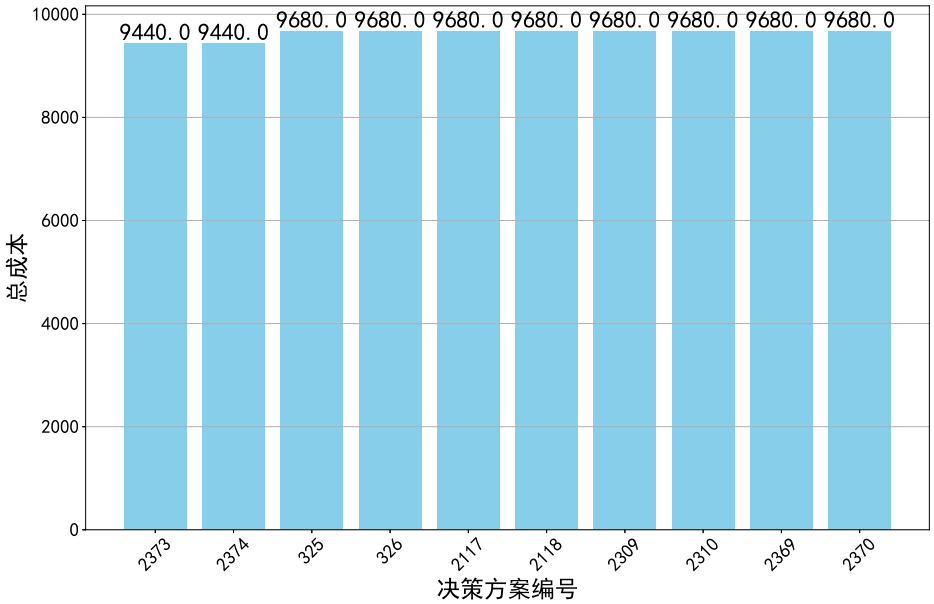

图 **11**　总成本最大的前 **10** 个决策方案的总成本

分析图 11可得，决策方案 2373、2374 的总成本相同且较低，所以 2373、2374 两种决策方案均可以作为企业在生产过程中的最优决策方案。

7.4 求解结果

由图表 4可知，决策方案 2373 与 2374 的总利润较高且总成本较低，属于最佳的两种决策方案，故企业在对 2 道工序，8 种零配件进行组装的生产过程中，需要检测零配件 1、2、4、5、7，无需检测零配件 3、6，零配件 8 检测与否皆可，需要检测所有半成品，需拆解所有不合格的半成品，无需检测成品，不合格的成品拆解与否解皆可。

表 **4**　问题三

决策方案	零配件是否检测								半成品是否检测			半成品	成品是否		总成本	总利润
	1	2	3	4	5	6	7	8	1	2	3	是否拆解	检测	拆解		
2373	是	是	否	是	是	否	是	否	是	是	是	是	否	是	9440	8560
2374	是	是	否	是	是	否	是	是	是	是	是	是	否	否	9440	8560

八、 问题四的模型的建立和求解

8.1 模型简介

本文在问题四中基于问题一截尾序贯抽样方案，将问题二、三表格中产品、半成品、零配件次品率作为标称值，在保证可信度下，求出检测次数最少的抽样方案，计算抽样检测次品率后，代入问题二、三模型，重新规划决策方案，具体流程如图 12：

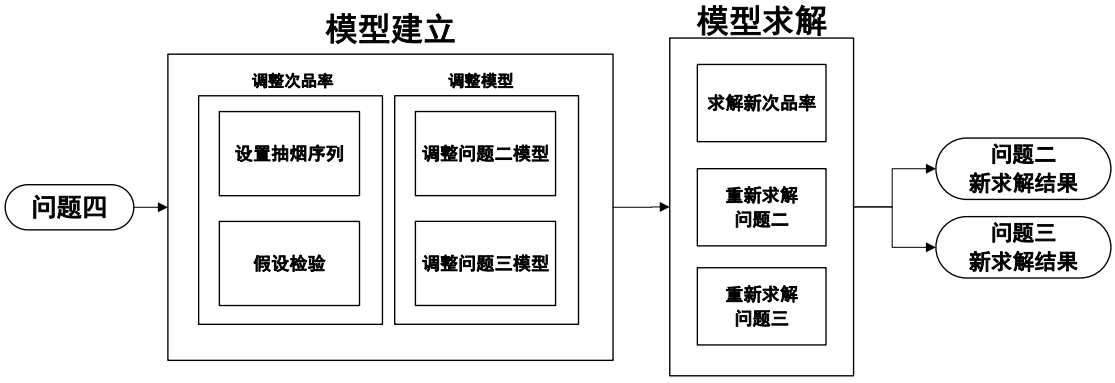

图 **12**　问题四流程图

8.2 模型建立

8.2.1 调整次品率

次品率需通过问题一中所建立的截尾序贯抽样检测的方法求出，具体步骤如下：

Step 1：设置抽样序列

对于问题二表格中的 2 种零配件、1 种成品与问题三表格中的 8 种零配件，3 种成品，1 种成品共 30 种不同物品，将其视作问题一数学模型中的不同批零配件，即需设置 30 种随机序列，对其进行截尾序贯抽样检测得到以得到各自截尾序贯抽样检测的次品率。

Step 2：假设检验

对随机序列进行截尾序贯抽样检测的抽样检测结果进行假设检验，本文在此设立的信度为 95%，设置同问题一所建模型相同的假设进行假设检验，求出零配件、半成品、成品共计 30 种各自在 95% 的信度下次品率不超过标称值，即接收某批零配件（或半成品、成品）时的检测次数尽量少的检测方案, 并得出相应的次品率。

表 **5**　企业在生产中遇到的情况（问题 **2**，新）

情况	次品率		
	零配件 1	零配件 2	成品
1	9.60%	10.65%	9.24%
2	16.67%	19.62%	22.30%
3	8.73%	11.20%	12.42%
4	21.43%	21.71%	19.43%
5	9.72%	17.36%	10.42%
6	5.30%	6.20%	3.27%

8.2.2 调整问题二及问题三模型

在问题二、三建立的模型中，本文考虑了问题二、三所给表格中半成品（成品）的次品率是将正品零配件（半成品）装配后的次品率，从而构建数学公式如 (10) 和 (20)，将表格中半成品（成品）次品率优化为任意零配件（半成品）装配后的次品率。问题四中，基于问题一模型求解出的次品率已代表任意零配件（半成品）装配后的次品率，无需再优化。故在去除问题二、三所建模型中优化次品率的步骤后，代入全新的截尾序贯抽样检测结果的次品率，计算总利润，选择总利润最大的决策方案作为企业生产过程中的决策方案。

8.3 模型求解

8.3.1 调整次品率

基于截尾序贯抽样检测得出问题二表格中 2 种零配件和成品的次品率，调整后的企业在生产中遇到的情况表格如表 5。

次品率的调整会影响最终决策方案，需重新对调整后的问题二模型进行求解，得出对表 5中的情形给出具体的决策方案，决策的依据及相应的指标结果。

8.3.2 问题二模型重新求解

Step1：决策方案分类

同问题二，仍为 1 道工序，2 个零配件，仍如表 2有 16 种决策方案。

Step2：动态规划求解最大利润

再次计算出每种情况下的 16 种决策方案的总利润如图 13：

观察图 13可以发现，依旧难以直接从折线图分辨出每种情况总利润最大的决策方案，以情况 1 与情况 3 为例，分别绘制每种情况下 16 种决策方案的总利润直方图如图 14：

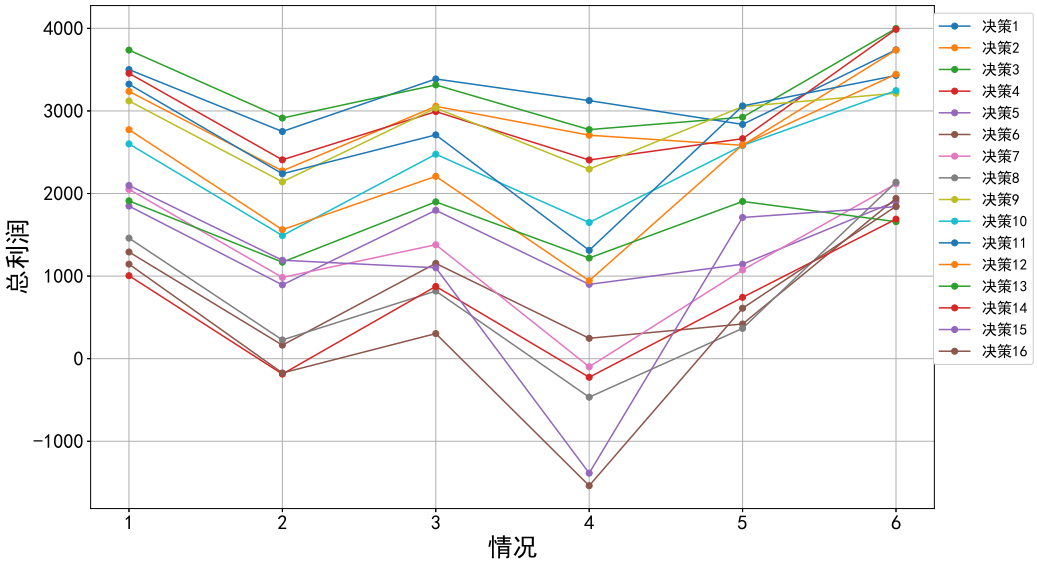

图 **13**　**6** 种情况下所有决策方案各自的总利润（新）

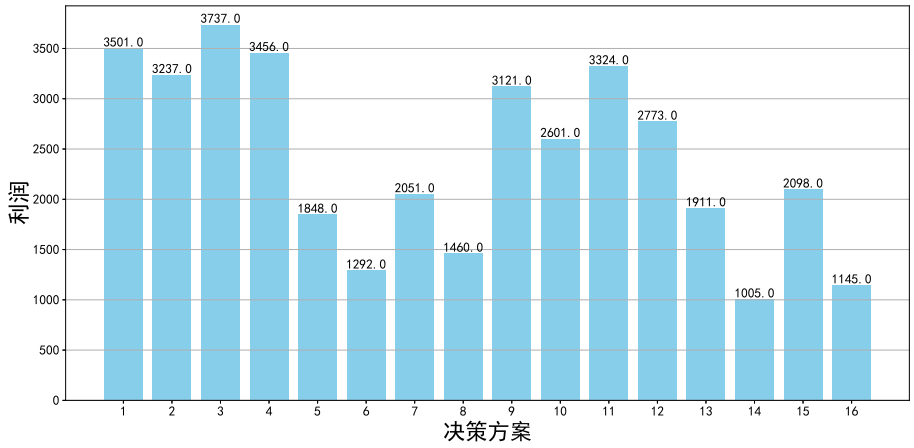

图 **14**　不同决策方案在情况 **1** 下的总利润直方图（新）

由图 14可以发现，在次品率更改后，情况 1 中选择决策方案 3 的总利润最高，为 3737.0 元，即在情况 1 下，企业在生产过程中应选择检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品的决策方案。由图 15可以发现，在次品率更改后，情况 3 中选择决策方案 1 的总利润最高，为 3387.0 元，即在情况 3 下，企业在生产过程中应选择检测零配件 1，检测零配件 2，拆解不合格成品，检测成品作为生产过程中的决策方案。

8.3.3 问题三模型重新求解

与问题三相同，零配件数目仍为 100 个，决策方案仍为 16384 种，仅次品率发生变化，在此基础上重新进行问题三的求解。

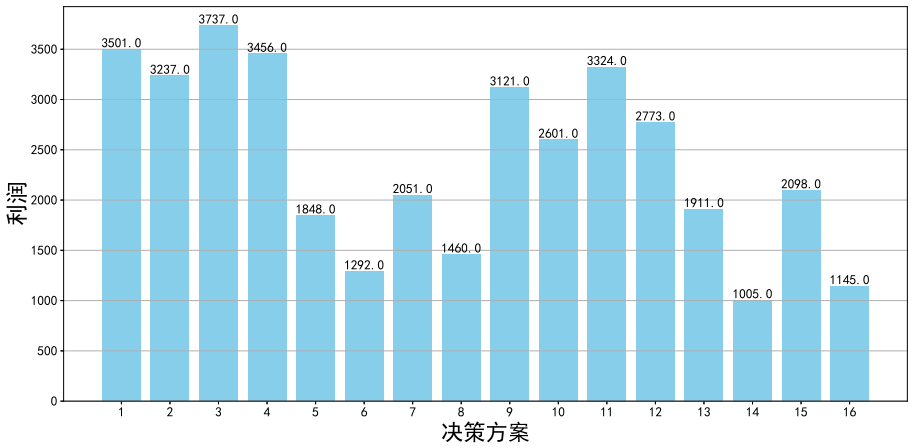

图 **15**　不同决策方案在情况 **3** 下的总利润直方图（新）

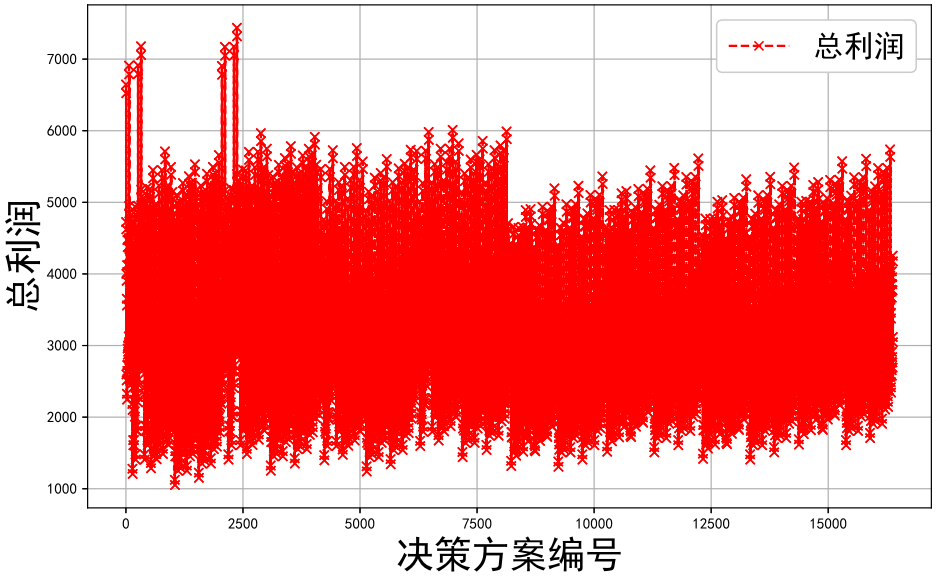

图 **16**　**2** 道工序，**8** 种零配件各种决策方案的总利润（新）

Step1：重新通过动态规划求出各种决策方案的总利润代入问题三所建立模型，重新计算出 $m=2$ 道工序、$n=8$ 种零配件的生产过程中各种决策方案的总利润，计算各种决策方案的总利润，结果图如图 16：

观察图 16，依旧较难明确看出哪种决策方案的总利润最大，仍需绘制部分总利润较高决策方案的总利润直方图，以便于选取最大利润的决策方案。

Step2：选取最大利润的决策方案

由于问题三的决策方案众多，可能出现某些决策方案的总利润相同的情况，本文对所有决策组合的总利润倒序排列，选取前 10 个绘制其利润直方图如图 17：

分析图 17可得，决策方案 2369 与 2370 的总利润相同且最高，需比较两种决策方案

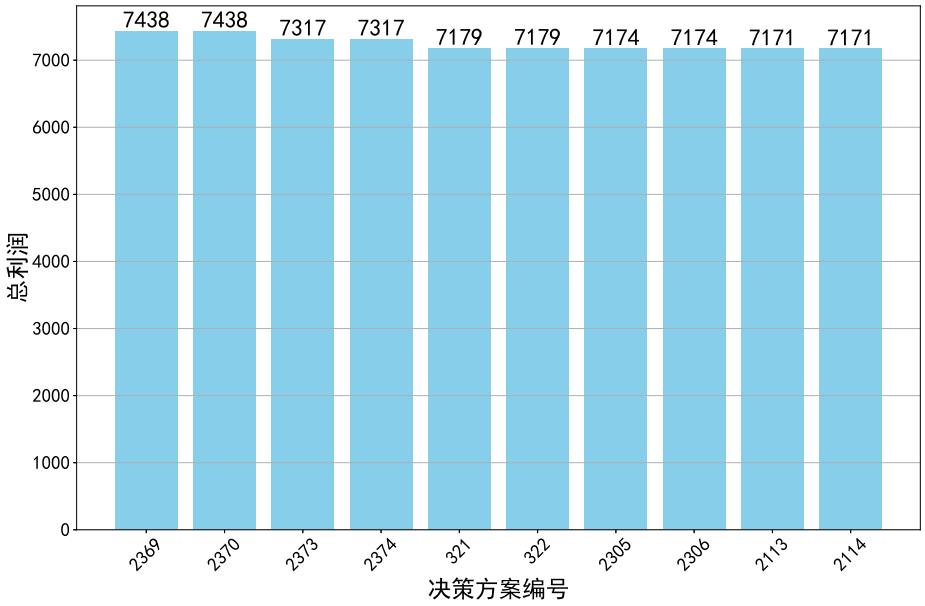

图 **17**　总利润最大的前十个决策方案（新）

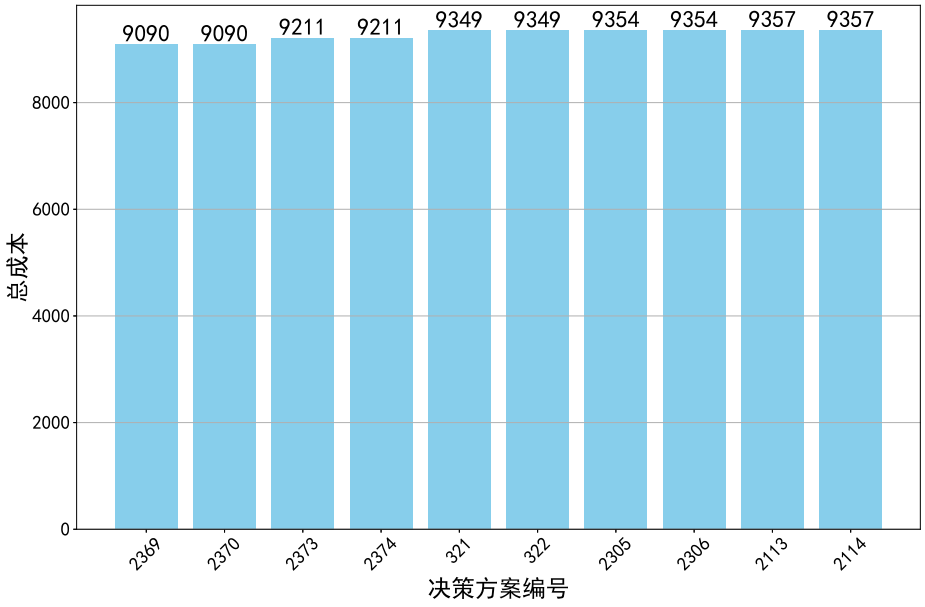

图 **18**　总利润最大的前十个高决策方案的总成本（新）

的成本，绘制图 17中 10 种决策方案的总成本直方图如图 18。

分析图 18可得，决策方案 2369、2370 的总成本相同且较低，故 2369、2370 两种决策方案均可以作为企业在生产过程中的最优决策方案。

8.4 求解结果

8.4.1 重新求解问题二结果

表 6　6 种情况下各自最佳的决策方案与决策指标（新）

情况	决策方案	零配件 1 是否检测	零配件 2 是否检测	成品 是否检测	成品 是否拆解	总成本 (元)	总利润 (元)
1	3	是	是	否	是	1139	3737
2	3	是	是	否	是	1156	2914
3	1	是	是	是	是	1355	3387
4	1	是	是	是	是	920	3125
5	11	否	是	否	是	1208	3061
6	3	是	是	否	是	1307.5	3999

分析表 6可知，情况 1、2、6 应选择决策方案 3（检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品）进行生产，情况 3、4 应选择决策方案 1（检测零配件 1，检测零配件 2，检测成品，拆解不合格成品）进行生产，情况 5 应选择决策方案 11（不检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品）进行生产，以达到每种情况下的总利润最大。决策依据是总利润最大，若总利润相同时，选择总成本较小的决策方案。所做决策方案与问题二求解结果相同，但决策指标不同。

8.4.2 重新求解问题三结果

表 7　2 道工序，8 个零配件中最佳决策方案与决策指标（新）

决策方案	零配件是否检测								半成品是否检测			半成品是否拆解	成品		总成本	总利润
	1	2	3	4	5	6	7	8	1	2	3		是否检测	是否拆解		
2369	是	是	否	是	是	否	是	否	是	是	是	是	是	是	9090	7438
2370	是	是	否	是	是	否	是	是	是	是	是	是	是	否	9090	7438

分析表 7可知，决策方案 2369、2370 的总利润相同且最大，总成本相同较低，属于最佳的两种决策方案，故企业在对 2 道工序，8 种零配件进行组装的生产过程中，需要检测零配件 1、2、4、5、7，无需检测零配件 3、6，零配件 8 检测与否皆可，需要检测所有半成品，需拆解所有不合格的半成品，需检测成品，不合格的成品拆解与否解皆可。

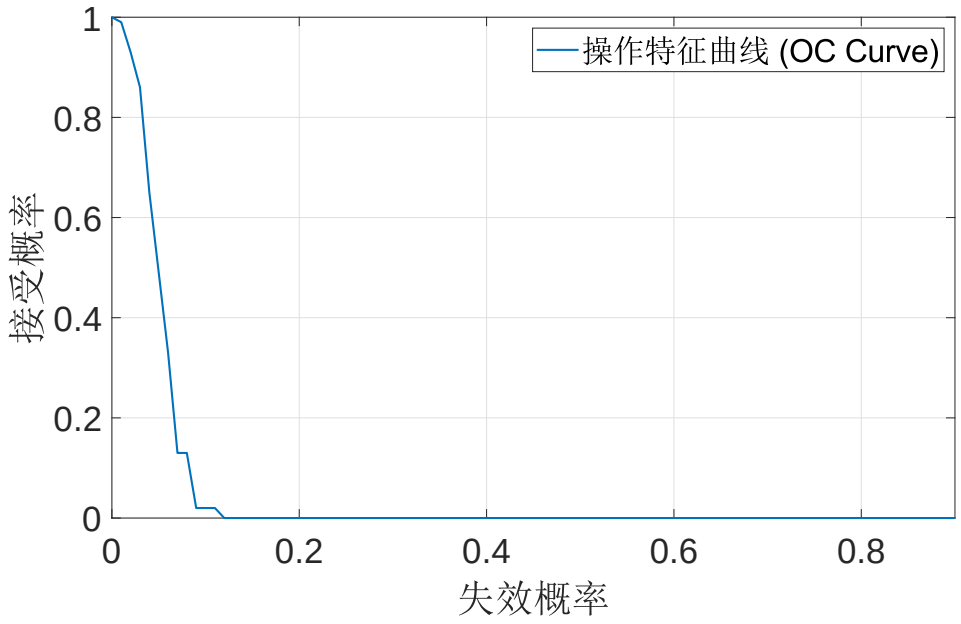

图 **19**　截尾序贯抽样检测方案的 **OC** 曲线图

九、 模型的检验

对于所建立的截尾序贯抽样检测方案，为检验其可行性与鉴别力，本文通过绘制 OC 曲线 [3] 进行判别，如果 OC 曲线平滑且连续，说明某种抽样检测模型在不同质量水平下的表现稳定，此外，如果 OC 曲线越陡峭，表示在较小的质量水平变化下，接受概率变化较大，说明某种抽样检测模型的鉴别力越强。在本文中，由于对一批零配件的抽样检测属于 A 类抽样，可以设 OC 曲线的横轴为该批零配件截尾序贯抽样检测的失效概率 θ，纵轴为接收概率 ω。OC 曲线曲线是一条通过（0,1）和（1,0）两点的对批次品率 $\hat{r}$ 的一条严格单调下降的连续函数曲线，其计算公式如 (26)。

$$\begin{cases} \theta(X=d) = \dbinom{z}{d} \hat{r}^d (1-\hat{r})^{z-d} \\ \omega(\theta) = \sum_{d=0}^{q} \dbinom{z}{d} \hat{r}^d (1-\hat{r})^{z-d} \end{cases} \tag{26}$$

其中，d 为抽样检测所抽样本中的次品数，q 为正品数，z 为样本总数。由此计算可得 OC 曲线图如图 19：

分析 OC 曲线图可以发现，曲线连续且具有较好的平滑性，说明模型具有较好的稳定性，同时，OC 曲线图在临界次品率附近，接受概率迅速下降，即模型的鉴别力较强，能有效区分某一批零配件中的次品与正品。

十、 模型的评价

10.1 模型的优点

- **实时决策**

截尾序贯抽样模型能够在每次抽样后立即做出决策，允许快速识别问题，从而在生产过程中进行实时调整，提升响应速度，可以有效应对零配件质量波动的问题。

- **检测效率高**

通过截尾序贯抽样模型逐步检测，企业可以在达到足够的置信度之前停止抽样，减少了不必要的检测次数和成本，提高了总体效率。

- **灵活性**

截尾序贯抽样模型可以根据实际情况灵活调整决策标准和参数 (如接受或拒绝的阈值)，适应不同的生产环境和质量要求。

10.2 模型的缺点

- **模型参数依赖性**

截尾序贯抽样模型的效果高度依赖于参数的准确设定，如假设的次品率和接受的阈值等。如果这些设定不准确，可能导致错误的决策。

- **可能的决策不确定性**

截尾序贯抽样模型的效果高度依赖于参数的准确设定，如假设的次品率和接受的阈值等。如果这些设定不准确，可能导致错误的决策。

10.3 模型展望

本文所构建的截尾序贯抽样检测模型可以进行一定的**参数优化**，探索动态调整的参数设置。根据实际抽样结果和生产反馈，实时优化样本大小和决策阈值，提高决策质量。此外，问题三中所构建的数学模型可以求解出任意生产过程为 m 道工序，n 种零配件下的决策方案，只需具有较高的普适性。

参考文献

[1] 胡思贵. 截尾序贯最优检验方法研究及其在机械产品质量检验中的应用[D]. 贵州大学, 2019.

[2] DODGE H F, ROMIG H G. Single sampling and double sampling inspection tables[J]. The Bell System Technical Journal, 1941, 20(1): 1-61.

[3] 刘芳宇. 质量三等级产品计数抽样理论与方法研究[D]. 北京理工大学, 2015.

[4] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[5] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.

[6] 戴敏. 多工序制造过程质量分析方法与信息集成技术研究[D]. 东南大学, 2006.

[7] 陈斌. 出口自行车零件抽样检查方案的研讨[J]. 中国自行车, 1993(01): 16-19.

附录 A　文件列表

文件名	功能描述
pro1.m	问题一程序代码
pro2.py	问题二程序代码
pro3.py	问题三程序代码
pro4-1.py	问题四程序代码 1
pro4-2.py	问题四程序代码 2
pro4-3.m	问题四程序代码 3
pro2-result.txt	问题二决策方案在各情况下的结果
pro3-result1.xlsx	问题三决策方案结果
pro3-result2.xlsx	问题三决策方案中成本最高的 10 个
pro4-result1.xlsx	问题四决策方案结果
pro4-result2.xlsx	问题四决策方案中成本最高的 10 个
pro4-result3.txt	问题四决策方案在各情况下的结果

附录 B　代码

pro1.m

```
1
2  %% 截尾序贯抽样检测
3
4  % 设置随机种子确保结果可重复
5  rng(4);
6  N = 10000;                % 总样本大小
7  p0 = 0.05;                % 标称次品率
8  data = rand(1, N) < p0;
9  %%
10 % 设置随机种子确保结果可重复
11 clc;
12
13
14 results_95 = [];        % 存储95%信度下的次品数量
15 results_90 = [];        % 存储90%信度下的次品数量
16 % N = 100;               % 总样本大小
```

```
p0 = 0.1;                % 标称次品率
% data = rand(1, N) < p0; % 创建随机0-1序列

% 定义参数
max_rounds = 100;        % 最多检测轮数
Z_alpha_95 = norminv(0.975);  % 95%信度的Z值
Z_alpha_90 = norminv(0.975);   % 90%信度的Z值

% (1) 在95%信度下拒收的情况下，寻找最小检测次数
min_detection_count_95 = inf; % 初始化最小检测次数
optimal_sample_size_95 = 0;    % 最优样本量

for sample_size = 10:100 % 从10到100的样本量
    results_95 = [];
    n_total_95 = 0;  % 95%信度下的次品总数
    accepted_95 = false; % 95%信度下是否接受

    for round = 1:max_rounds
        if (round - 1) * sample_size + sample_size > N
            break; % 确保不会超出总样本数
        end
        indices = randperm(N, sample_size); % 生成随机索引
        sample = data(indices); % 根据随机索引抽样
        % sample = data((round - 1) * sample_size + 1: round * sample_size); % 抽样
        n = sum(sample);  % 次品数量
        n_total_95 = n_total_95 + n;  % 统计95%信度下的次品总数

        % 更新均值和标准差
        mean_95 = n_total_95 / round;
        std_dev_95 = std([results_95, n]);  % 次品数量的标准差

        % 计算接受范围
        L_95 = mean_95 - Z_alpha_95 * (std_dev_95 / sqrt(round
```

```
    ));
50          U_95 = mean_95 + Z_alpha_95 * (std_dev_95 / sqrt(round
    ));
51
52          % 检测接受与否
53          if n > U_95
54              accepted_95 = false;  % 拒绝这批零配件
55              break;  % 停止抽样
56          elseif n < L_95
57              accepted_95 = true;  % 接受这批零配件
58
59              break;  % 停止抽样
60
61          else
62
63          results_95(end+1) = n;  % 继续抽样
64          end
65      end
66
67      % 计算检测次数
68      detection_count = round * sample_size;
69
70      if ~accepted_95 && detection_count <
    min_detection_count_95
71          min_detection_count_95 = detection_count; % 更新最小检
    测次数
72          optimal_sample_size_95 = sample_size;     % 更新最优样
    本量
73          round_95 = round;
74          n_total_95_min = n_total_95;
75          % 计算95%信度下的次品率
76          final_defect_rate_95 = n_total_95_min / (round_95  *
    sample_size); % 95%信度下的次品率
77      end
78  end
```

```
79
80
81
82  % 输出95%信度下的结果
83  fprintf('在95%%信度下拒收的情况下，次品率：%.2f%%，\n最小检测
        次数：%d，最优样本量：%d，检测轮数：%d\n', ...
84          final_defect_rate_95 *100 , min_detection_count_95,
        optimal_sample_size_95, round_95);
85
86
87  %
88  % % 输出95%信度下的结果
89  % fprintf('95%%信度下的检测轮数：%d\n', round);
90  % fprintf('95%%信度下的最终次品数量：%d\n', n_total_95);
91  % fprintf('95%%信度下的次品率：%.2f%%\n', final_defect_rate_95
        / 100);
92
93  % (2) 在90%信度下接受的情况下，寻找最小检测次数
94  min_detection_count_90 = inf; % 初始化最小检测次数
95  optimal_sample_size_90 = 0;    % 最优样本量
96
97  for sample_size = 10:100 % 从10到100的样本量
98      results_90 = [];
99      n_total_90 = 0;  % 90%信度下的次品总数
100     accepted_90 = false; % 90%信度下是否接受
101
102     for round = 1:max_rounds
103         if (round - 1) * sample_size + sample_size > N
104             break; % 确保不会超出总样本数
105         end
106
107         indices = randperm(N, sample_size); % 生成随机索引
108         sample = data(indices); % 根据随机索引抽样
109         % sample = data((round - 1) * sample_size + 1: round *
        sample_size); % 抽样
```

```
        n = sum(sample);  % 次品数量
        n_total_90 = n_total_90 + n;  % 统计90%信度下的次品总数

        % 更新均值和标准差
        mean_90 = n_total_90 / round;

        results_90(end+1) = n;  % 继续抽样
        std_dev_90 = std(results_90);  % 次品数量的标准差

        % 计算接受范围
        L_90 = mean_90 - Z_alpha_90 * (std_dev_90 / sqrt(round));
        U_90 = mean_90 + Z_alpha_90 * (std_dev_90 / sqrt(round));

        % 检测接受与否
        if n < L_90
            accepted_90 = true;  % 接受这批零配件


            break;  % 停止抽样
        elseif n > U_90
            accepted_90 = false;  % 拒绝这批零配件

            break;  % 停止抽样


        end
    end

    % 计算检测次数
    detection_count = round * sample_size;

    if accepted_90 && detection_count < min_detection_count_90
```

```
142         min_detection_count_90 = detection_count; % 更新最小检测次数
143         optimal_sample_size_90 = sample_size;     % 更新最优样本量
144         round_90 = round;
145         n_total_90_min = n_total_90;
146         % 计算95%信度下的次品率
147         final_defect_rate_90 = n_total_90_min / (round_90  * optimal_sample_size_90); % 95%信度下的次品率
148     end
149 end
150
151
152
153 % 输出95%信度下的结果
154 fprintf('在90%%信度下接受的情况下，次品率：%.2f%%，\n最小检测次数：%d，最优样本量：%d，检测轮数：%d\n', ...
155         final_defect_rate_90 * 100, min_detection_count_90, optimal_sample_size_90, round_90);
156
157
158 %%
159 % 初始化参数
160 N = 10000;                % 总样本大小
161 num_samples = 100;        % 每个比例的样本数量
162 p_defect_ratios = 0:0.1:1;  % 次品比例从0到1
163 acceptance_probabilities = zeros(size(p_defect_ratios)); % 接受概率
164
165 % 对于每个次品比例进行抽样
166 for i = 1:length(p_defect_ratios)
167     p0 = p_defect_ratios(i); % 当前次品比例
168     data = rand(1, N) < p0;  % 根据比例生成次品数据
169
170     accepted_count = 0; % 被接受的次数
```

```
171
172     for j = 1:num_samples
173         sample_size = 100; % 每次抽样的样本量
174         indices = randperm(N, sample_size); % 随机抽样
175         sample = data(indices); % 抽样数据
176         n = sum(sample);  % 次品数量
177
178         % 决策逻辑（假设的范围，可以根据您的模型调整）
179         if n < sample_size * 0.5 % 设定的接受条件
180             accepted_count = accepted_count + 1; % 接受
181         end
182     end
183
184     % 计算接受概率
185     acceptance_probabilities(i) = accepted_count / num_samples
    ; % 接受概率
186 end
187
188 % 绘制 OC 曲线
189 figure;
190 plot(p_defect_ratios, acceptance_probabilities, '-o','
    LineWidth',1.5);
191 xlabel('次品率');
192 ylabel('接受概率');
193 legend('操作特征曲线 (OC Curve)')
194 % title('操作特征曲线 (OC Curve)');
195 grid on;
196
197 % 显示额外信息
198 xlim([0 1]);
199 ylim([0 1]);
200 set(gca,'FontSize',25)
201
202 % 保存图形为 .eps 文件
203 print('oc_curve', '-depsc'); % 'oc_curve.eps' 将被创建
```

pro2.py

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

# 设置中文字体以支持中文标签
rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体显示中文
rcParams['axes.unicode_minus'] = False  # 正常显示负号

# 定义不同检测情况下的参数，包括次品率、检测成本、市场售价等
cases = [
    {'零配件1次品率': 0.1, '零配件2次品率': 0.1, '成品次品率': 0.1, '零配件1检测成本': 2, '零配件2检测成本': 3,
     '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用': 5},
    {'零配件1次品率': 0.2, '零配件2次品率': 0.2, '成品次品率': 0.2, '零配件1检测成本': 2, '零配件2检测成本': 3,
     '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用': 5},
    {'零配件1次品率': 0.1, '零配件2次品率': 0.1, '成品次品率': 0.1, '零配件1检测成本': 2, '零配件2检测成本': 3,
     '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用': 5},
    {'零配件1次品率': 0.2, '零配件2次品率': 0.2, '成品次品率': 0.2, '零配件1检测成本': 1, '零配件2检测成本': 1,
     '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用': 5},
    {'零配件1次品率': 0.1, '零配件2次品率': 0.2, '成品次品率': 0.1, '零配件1检测成本': 8, '零配件2检测成本': 1,
     '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解
```

```
    费用': 5},
21      {'零配件1次品率': 0.05, '零配件2次品率': 0.05, '成品次品率
    ': 0.05, '零配件1检测成本': 2, '零配件2检测成本': 3,
22        '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成
    本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解
    费用': 40}
23  ]
24
25
26  # 计算总利润
27  def total_profit_cal(case, n, BUY, detect_parts1,
    detect_parts2, detect_final, dismantle):
28      """
29      计算总成本和总利润
30      参数:
31      case (dict): 包含当前情况的各种参数。
32      detect_parts1 (bool): 是否检测零配件1。
33      detect_parts2 (bool): 是否检测零配件2。
34      detect_final (bool): 是否检测成品。
35      dismantle (bool): 是否拆解不合格成品。
36      返回:
37      tuple: 包含总成本和缺陷率。
38      """
39      n_parts1 = n  # 假设零配件1数量
40      n_parts2 = n  # 假设零配件2数量
41      # 零配件检测成本
42      cost_parts1 = n_parts1 * (case['零配件1检测成本'] if
    detect_parts1 else 0 + case['零配件1购买单价']) if BUY else
    0
43      cost_parts2 = n_parts2 * (case['零配件2检测成本'] if
    detect_parts2 else 0 + case['零配件2购买单价']) if BUY else
    0
44
45      # 计算未检测零配件情况下的损失
46      # loss_parts1 = (n_parts1 * case['零配件1次品率']) * (case
```

```
['装配成本']) if not detect_parts1 else 0
47      # loss_parts2 = (n_parts2 * case['零配件2次品率']) * (case
['装配成本']) if not detect_parts2 else 0
48
49      # 装配前的零件数
50      before_final_n_parts1 = n_parts1 * ((1 - case['零配件1次品
率']) if detect_parts1 else 1)
51      before_final_n_parts2 = n_parts2 * ((1 - case['零配件2次品
率']) if detect_parts2 else 1)
52
53      # 成品数量
54      n_final_products = min(before_final_n_parts1,
before_final_n_parts2)
55
56      # 计算装配成本
57      cost_assembly = n_final_products * case['装配成本']
58
59      # 计算检测成品成本
60      cost_final = n_final_products * case['成品检测成本'] if
detect_final else 0
61
62      # 零件加工前的正品率
63      standard_rate1 = 1 if detect_parts1 else (1 - case['零配件
1次品率'])
64      standard_rate2 = 1 if detect_parts2 else (1 - case['零配件
2次品率'])
65
66      # 更新后的成品次品率
67      updated_defective_rate = 1 + (case['成品次品率'] - 1)*(
standard_rate1 * standard_rate2)
68
69      # 不合格的成品
70      defective_final = n_final_products *
updated_defective_rate
71
```

```
    # 计算不检测成品的调换损失
    loss_final = defective_final * case['调换损失'] if not
detect_final else 0

    # 拆解成本
    dismantle_cost = defective_final * case['拆解费用'] if
dismantle else 0

    # 拆解后能挽回的损失
    if dismantle and (defective_final > 2):
        a, b, dismantle_revenue = total_profit_cal(case,
defective_final, False, detect_parts1, detect_parts2,
detect_final, dismantle)
    else:
        dismantle_revenue = 0


    #卖出去的件数
    n_sell = n_final_products * (1 - updated_defective_rate)

    # 收入
    total_gain = case['市场售价'] * n_sell + dismantle_revenue
     # (if detect_final else 1)

    # 计算总成本
    total_cost = (cost_parts1 + cost_parts2 + cost_assembly +
                  cost_final + loss_final + dismantle_cost)

    # 利润
    total_profit = round(total_gain - total_cost)



    return round(total_cost), round(total_gain), total_profit

```

```python

# 定义所有检测策略的组合
strategies = []
for detect_parts1 in [True, False]:
    for detect_parts2 in [True, False]:
        for detect_final in [True, False]:
            for dismantle in [True, False]:
                # strategies;
                strategies.append({
                    "detect_parts1": detect_parts1,
                    "detect_parts2": detect_parts2,
                    "detect_final": detect_final,
                    "dismantle": dismantle
                })

# 生成策略说明
strategy_explanations = []
for i, strategy in enumerate(strategies):
    strategy_explanations.append(
        f"策略 {i + 1}: 检测零配件 1 {'是' if strategy['detect_parts1'] else '否'}, "
        f"检测零配件 2 {'是' if strategy['detect_parts2'] else '否'}, "
        f"检测成品 {'是' if strategy['detect_final'] else '否'}, "
        f"拆解不合格成品 {'是' if strategy['dismantle'] else '否'}"
    )

    # 存储每种情况的成本和缺陷率
costs = []
profits = []

# 遍历每种情况并计算成本和缺陷率
for i, case in enumerate(cases):
```

```
    print(f"\n情况 {i + 1}:")
    case_costs = []
    case_total_profit = []
    for j, strategy in enumerate(strategies):
        total_cost, total_gain, total_profit = total_profit_cal(case, 100, True, **strategy)
        case_costs.append(total_cost)
        case_total_profit.append(total_profit)
        print(f"{strategy_explanations[j]}: 总成本 = {total_cost:.1f}, 总收入 = {total_gain:.1f},总利润 = {total_profit:.1f}")
    costs.append(case_costs)
    profits.append(case_total_profit)

    # 将利润转换为NumPy数组以便于处理
costs = np.array(costs)
profits = np.array(profits)


# # 绘制不同情况和策略下的总成本比较图
# plt.figure(figsize=(10, 6))
# for i in range(costs.shape[1]):
#     plt.plot(range(1, costs.shape[0] + 1), costs[:, i], marker='o', label=f"策略 {i + 1} - 总成本")
# plt.xlabel("情况")
# plt.ylabel("总成本")
# plt.title("不同情况和策略下的总成本比较")
# plt.legend()
# plt.grid(True)
# plt.show()
#%%
plt.figure(figsize=(15, 9))
for i in range(profits.shape[1]):
    plt.plot(range(1, profits.shape[0] + 1), profits[:, i], marker='o', label=f"决策{i + 1}")
```

```
plt.xlabel("情况", fontsize=25)
plt.ylabel("总利润", fontsize=25)
# plt.title("不同情况和策略下的总利润比较")
# 设置坐标轴刻度字体大小
plt.tick_params(axis='both', labelsize=20)  # 设置 x 和 y 轴刻度字体大小
plt.legend(loc='upper left', bbox_to_anchor=(0.99, 1), fontsize=15)
plt.grid(True)
# 保存图形为 .eps 文件
plt.savefig("pics/问题二：不同决策方案在不同情况下的总利润曲线.eps", format='eps')
plt.savefig("pics/问题二：不同决策方案在不同情况下的总利润曲线.png", format='png')
plt.show()

# 找到每一行的最大值及其索引
max_values = np.max(profits, axis=1)  # 每一行的最大值
max_indices = np.argmax(profits, axis=1)  # 每一行最大值的列索引

# 输出结果
for row in range(profits.shape[0]):
    col = max_indices[row]
    print(f"情况 {row + 1}: 使用决策方案 {col + 1}，所需总成本{costs[row,col]:.1f}元，可获得最大利润{max_values[row]:.1f}元")


#%%
    # 提取第一行数据
    first_row_data = profits[0, :]  # 取第一行的前6个策略数据

    # 策略标签
    strategies = [f"{i + 1}" for i in range(16)]
```

```
191
192     # 绘制直方图
193     plt.figure(figsize=(17, 8))
194
195     bars = plt.bar(strategies, first_row_data, color='skyblue'
    )
196
197     # 在每个柱子上添加数据标签
198     for bar in bars:
199         yval = bar.get_height()  # 获取柱子的高度
200         plt.text(bar.get_x() + bar.get_width() / 2, yval, f'{
    yval:.1f}',
201                  ha='center', va='bottom', fontsize=15)  # 在
    柱子上方添加文本
202
203     # 设置坐标轴标签和标题
204     plt.xlabel("决策方案", fontsize=25)
205     plt.ylabel("利润", fontsize=25)
206     # plt.title("策略利润直方图", fontsize=20)
207
208     # 设置坐标轴刻度字体大小
209     plt.tick_params(axis='both', labelsize=15)
210
211     # 显示网格
212     plt.grid(axis='y')
213
214     plt.savefig("pics/问题二：不同决策方案在情况1下的总利润直
    方图.eps", format='eps')
215     plt.savefig("pics/问题二：不同决策方案在情况1下的总利润直
    方图.png", format='png')
216     # 显示图形
217     plt.show()
218
219 #%%
220     # 提取第一行数据
```

```
    first_row_data = profits[2, :]  # 取第一行的前6个策略数据

    # 策略标签
    strategies = [f"{i + 1}" for i in range(16)]

    # 绘制直方图
    plt.figure(figsize=(17, 8))

    bars = plt.bar(strategies, first_row_data, color='skyblue')

    # 在每个柱子上添加数据标签
    for bar in bars:
        yval = bar.get_height()  # 获取柱子的高度
        plt.text(bar.get_x() + bar.get_width() / 2, yval, f'{yval:.1f}',
                 ha='center', va='bottom', fontsize=15)  # 在柱子上方添加文本

    # 设置坐标轴标签和标题
    plt.xlabel("决策方案", fontsize=25)
    plt.ylabel("利润", fontsize=25)
    # plt.title("策略利润直方图", fontsize=20)

    # 设置坐标轴刻度字体大小
    plt.tick_params(axis='both', labelsize=15)

    # 显示网格
    plt.grid(axis='y')

    plt.savefig("pics/问题二：不同决策方案在情况3下的总利润直方图.eps", format='eps')
    plt.savefig("pics/问题二：不同决策方案在情况3下的总利润直方图.png", format='png')
    # 显示图形
```

```
    plt.show()
```

pro3.py

```
#%%
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
import pandas as pd
from matplotlib import rcParams

# 设置中文字体以便于图表显示
rcParams['font.sans-serif'] = ['SimHei']
rcParams['axes.unicode_minus'] = False

# 定义零配件的属性，包括次品率、购买单价和检测成本
components = {
    '零配件1': {'次品率': 0.1, '购买单价': 2, '检测成本': 1},
    '零配件2': {'次品率': 0.1, '购买单价': 8, '检测成本': 1},
    '零配件3': {'次品率': 0.1, '购买单价': 12, '检测成本': 2},
    '零配件4': {'次品率': 0.1, '购买单价': 2, '检测成本': 1},
    '零配件5': {'次品率': 0.1, '购买单价': 8, '检测成本': 1},
    '零配件6': {'次品率': 0.1, '购买单价': 12, '检测成本': 2},
    '零配件7': {'次品率': 0.1, '购买单价': 8, '检测成本': 1},
    '零配件8': {'次品率': 0.1, '购买单价': 12, '检测成本': 2},
}

# 定义半成品的属性，包括次品率、装配成本、检测成本和拆解费用
semi_products = {
    '半成品1': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品2': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品3': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
```

```
}

# 定义成品的属性，包括次品率、装配成本、检测成本、拆解费用和市场售价
final_product = {
    '成品': {'次品率': 0.1, '装配成本': 8, '检测成本': 6, '拆解费用': 10, '售价': 200, '调换损失': 40}
}

def total_profit_cal(n, BUY,  components, semi_products, final_product, detect_components,
                             detect_semi, semi_dismantle, detect_final, dismantle):
    """
    计算预期总成本和缺陷率。
    参数:
    components (dict): 零配件信息。
    semi_products (dict): 半成品信息。
    final_product (dict): 成品信息。
    detect_components (list): 是否检测各零配件的布尔列表。
    detect_semi (list): 是否检测各半成品的布尔列表。
    detect_final (bool): 是否检测成品的布尔值。
    dismantle (bool): 是否拆解不合格成品的布尔值。
    返回:
    tuple: 总成本和缺陷率。
    """
    n_comp = np.zeros(8)  # 变成半成品之前的零件个数
    n_semi = np.zeros(3)  # 变成成品之前的半成品个数
    n_semi_dismantle = np.zeros(3)
    # 零件加工前的正品率
    standard_rate = np.zeros(8)

    total_cost = 0  # 初始化总成本
    # total_defective_rate = 1  # 初始化总缺陷率

```

```
    # 计算零配件成本和总缺陷率
    for i, (comp_name, comp_data) in enumerate(components.items()):

        if i < 3:
            n_separate = n[0]  # 第一个零件组
        elif i < 6:
            n_separate = n[1]   # 第二个零件组
        else:
            n_separate = n[2]   # 第三个零件组
        cost = n_separate * comp_data['购买单价'] if BUY else 0  # 零配件的购买成本
        if detect_components[i]:  # 如果检测该零配件
            cost += n_separate * comp_data['检测成本']  # 加上检测成本
            standard_rate[i] = 1
        else:
            standard_rate[i] = 1 - comp_data['次品率']
        n_comp[i] = standard_rate[i] * n_separate

        total_cost += cost  # 累加到总成本

    updated_defective_semi_rate = np.zeros(3)  # 创建一个包含3个零的数组

    # 计算乘积并存储在 n_semi 中
    updated_defective_semi_rate[0] = 1 + (0.1 - 1) * np.prod(standard_rate[0:2])  # 第1到第3个元素的乘积
    updated_defective_semi_rate[1] = 1 + (0.1 - 1) * np.prod(standard_rate[3:5])  # 第4到第6个元素的乘积
    updated_defective_semi_rate[2] = 1 + (0.1 - 1) * np.prod(standard_rate[6:7])  # 第7到第8个元素的乘积

    # 组装成的半成品个数
    n_semi[0] = np.min(n_comp[0:2])  # 第1-3个值的最小值
```

```
    n_semi[1] = np.min(n_comp[3:5])  # 第4-6个值的最小值
    n_semi[2] = np.min(n_comp[6:7])  # 第7-8个值的最小值

    defective_semi = np.zeros(3)

    # 半成品的正频率
    semi_standard_rate = np.zeros(3)

    # 计算半成品成本和总缺陷率
    for i, (semi_name, semi_data) in enumerate(semi_products.items()):
        cost = n_semi[i] * semi_data['装配成本']  # 半成品的装配成本

        if detect_semi[i]:  # 如果检测该半成品
            defective_semi[i] = n_semi[i] * updated_defective_semi_rate[i]
            cost += 4 * n_semi[i]  # 加上检测成本
            semi_standard_rate[i] = 1
        else:
            semi_standard_rate[i] = 1 - updated_defective_semi_rate[i]
        # n_semi_dismantle[i] = (1 - semi_standard_rate[i]) * n_semi[i]
        n_semi[i] = semi_standard_rate[i] * n_semi[i]
        total_cost += cost  # 累加到总成本

    if np.all(detect_semi) and semi_dismantle and (np.min(defective_semi) > 2):  # 如果所有半成品均已检测且需要拆解

        cost = np.sum(defective_semi) * 6  # 加上拆解费用

        # 拆解后，重新计算相关零件费用
        dismantled_comp_cost, dismantled_gain, dismantled_profit = total_profit_cal(
```

```
            defective_semi, False, components, semi_products, final_product,
            detect_components, detect_semi, semi_dismantle,
            detect_final, dismantle  # 为了避免重复拆解
        )
        total_cost += cost - dismantled_profit

    updated_defective_final_rate = 1 + (0.1 - 1) * np.prod(semi_standard_rate[:])

    # 最终的成品数量
    n_final = np.min(n_semi)
    # 卖出去的成品只有正品
    n_sell = n_final * (1 - updated_defective_final_rate)
    # 不合格的成品
    defective_final = n_final * updated_defective_final_rate

    # 计算成品的成本
    # final_defective_rate = final_product['成品']['次品率'] * (0.5 if detect_final else 1)  # 成品的次品率
    final_cost = n_final * final_product['成品']['装配成本']  # 成品的装配成本

    if detect_final:  # 如果检测成品
        final_cost += n_final * final_product['成品']['检测成本']  # 加上检测成本

    else:               # 如果不检测成品
        final_cost += defective_final * final_product['成品']['调换损失']

    total_cost += final_cost  # 累加到总成本

    if dismantle and (defective_final > 2):  # 如果拆解不合格成品
```

```
145         total_cost += defective_final * final_product['成品'][
    '拆解费用']  # 加上拆解费用
146         # 拆解后，重新计算相关零件费用
147         dismantled_final_cost, dismantled_final_gain,
    dismantled_final_profit = total_profit_cal(
148             defective_final * np.ones(3), False, components,
    semi_products, final_product,
149             detect_components, detect_semi, semi_dismantle,
150             detect_final, dismantle  # 为了避免重复拆解
151         )
152         total_cost += - dismantled_final_profit
153     # 收入
154     total_gain = final_product['成品']['售价'] * n_sell  #*
    ((1 - updated_defective_rate) if detect_final else 1) +
    dismantle_revenue
155
156     # 利润
157     total_profit = round(total_gain - total_cost)
158
159     # 返回总成本和利润
160     return round(total_cost), round(total_gain), total_profit
161
162     # 生成所有可能的检测组合
163
164
165 component_combinations = list(product([True, False], repeat=8)
    )  # 零配件检测组合
166 semi_combinations = list(product([True, False], repeat=3))  #
    半成品检测组合
167 semi_dismantle_combinations = list(product([True, False],
    repeat=1))  # 半成品拆解组合
168 final_combinations = list(product([True, False], repeat=2))  #
     成品检测和拆解组合
169
170 strategy_descriptions = []  # 策略描述列表
```

```python
results = []  # 结果列表


def generate_strategy_description(detect_components, detect_semi, semi_dismantle, detect_final, dismantle):
    """
    生成策略描述。

    参数:
    detect_components (list): 零配件检测布尔列表。
    detect_semi (list): 半成品检测布尔列表。
    detect_final (bool): 成品检测布尔值。
    dismantle (bool): 拆解布尔值。

    返回:
    str: 策略描述字符串。
    """
    description = ""
    for i, detect in enumerate(detect_components):
        description += f"检测零配件 {i + 1}, " if detect else f"不检测零配件 {i + 1}, "
    for i, detect in enumerate(detect_semi):
        description += f"检测半成品 {i + 1}, " if detect else f"不检测半成品 {i + 1}, "
    for i, dismantle in enumerate(semi_dismantle):
        description += f"拆解半成品, " if semi_dismantle else f"不拆解半成品, "

    description += "检测成品, " if detect_final else "不检测成品, "
    description += "拆解不合格成品" if dismantle else "不拆解不合格成品"
    return description


```

```
# 策略编号初始化
strategy_number = 1
# 遍历所有组合，计算成本和缺陷率
for comp_comb in component_combinations:
    for semi_comb in semi_combinations:
        for final_comb in final_combinations:
            for semi_dismantle_comb in semi_dismantle_combinations:
                semi_dismantle = semi_dismantle_comb  # 当前半成品拆解组合
                detect_components = comp_comb  # 当前零配件检测组合
                detect_semi = semi_comb  # 当前半成品检测组合
                detect_final = final_comb[0]  # 当前成品检测状态
                dismantle = final_comb[1]  # 当前拆解状态

                # 计算当前策略的总成本和利润
                total_cost, total_gain, total_profit = total_profit_cal([100, 100, 100], True, components, semi_products, final_product, detect_components,

 detect_semi, semi_dismantle, detect_final, dismantle)

                # 生成当前策略的描述
                strategy_description = generate_strategy_description(detect_components, detect_semi, semi_dismantle, detect_final,

       dismantle)
                results.append([strategy_number, strategy_description, total_cost, total_gain, total_profit])  # 将结果添加到列表
                strategy_number += 1  # 策略编号递增

```

```
# 将结果转换为DataFrame并输出
df = pd.DataFrame(results, columns=['决策方案编号', '决策方案描述', '总成本', '总收入', '总利润'])
print(df.head())  # 打印前五条策略结果

# 将结果保存到Excel文件
df.to_excel('问题三决策方案结果.xlsx', index=False)
print("所有策略已经保存到 '决策方案结果.xlsx' 文件中。")


#%%
# 绘制总成本比较图
plt.figure(figsize=(10, 6))
plt.plot(df['决策方案编号'], df['总成本'], marker='o', linestyle='-', color='blue', label="总成本")
plt.xlabel("决策方案编号", fontsize=25)
plt.ylabel("总成本", fontsize=25)
# plt.title("不同决策方案下的总成本比较", fontsize=25)
plt.grid(True)
plt.legend(fontsize=20)
plt.savefig("pics/问题三：不同决策方案的总成本折线图.eps", format='eps')
plt.savefig("pics/问题三：不同决策方案的总成本折线图.png", format='png')
plt.show()

# 绘制总利润比较图
plt.figure(figsize=(10, 6))
plt.plot(df['决策方案编号'], df['总利润'], marker='x', linestyle='--', color='red', label="总利润")
plt.xlabel("决策方案编号", fontsize=25)
plt.ylabel("总利润", fontsize=25)
# plt.title("不同决策方案下的总利润比较")
plt.grid(True)
plt.legend(fontsize=20)
```

```
plt.savefig("pics/问题三：不同决策方案的总利润折线图.eps", format='eps')
plt.savefig("pics/问题三：不同决策方案的总利润折线图.png", format='png')
plt.show()

#%%


# 找出总利润最大的10行
top_10_profit = df.nlargest(10, '总利润')

# 输出决策方案描述及其 '总成本', '总收入', '总利润'
#print(top_10_profit[['决策方案描述']].to_string(index=False), "所得的总利润为：", top_10_profit[['总利润']].to_string(index=False))

# 绘制直方图
plt.figure(figsize=(16, 10))
bars = plt.bar(range(len(top_10_profit)), top_10_profit['总利润'], color='skyblue')

# 在每个条形上标出数据
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom', fontsize=25)

# 设置图表标题和标签
#plt.title('总利润最大的10个决策方案')
plt.xlabel('决策方案编号', fontsize=25)
plt.ylabel('总利润', fontsize=25)

# 使用字符标注横坐标
plt.xticks(range(len(top_10_profit)), top_10_profit['决策方案
```

```
    编号'].astype(str), rotation=45)
282
283 # plt.tight_layout()  # 自动调整布局
284 # 设置坐标轴刻度字体大小
285 plt.tick_params(axis='both', labelsize=20)
286 plt.grid(axis='y')
287 plt.savefig("pics/问题三：总利润最大的10个决策方案直方图.eps",
        format='eps')
288 plt.savefig("pics/问题三：总利润最大的10个决策方案直方图.png",
        format='png')
289 # 显示图表
290 plt.show()
291 top_10_profit.to_excel('问题三：决策方案中利润最高的10个.xlsx'
    , index=False)
292
293 #%%
294 # 找出总利润最大的10行
295 top_10_profit = df.nlargest(10, '总利润')
296
297 # 输出决策方案描述及其 '总成本', '总收入', '总利润'
298 #print(top_10_profit[['决策方案描述']].to_string(index=False),
        "所得的总利润为：", top_10_profit[['总利润']].to_string(
    index=False))
299
300 # 绘制直方图
301 plt.figure(figsize=(16, 10))
302 bars = plt.bar(range(len(top_10_profit)), top_10_profit['总成
    本'], color='skyblue')
303
304 # 在每个条形上标出数据
305 for bar in bars:
306     yval = bar.get_height()
307     plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval
    , 2), ha='center', va='bottom', fontsize=25)
308
```

```
# 设置图表标题和标签
#plt.title('总成本最大的10个决策方案')
plt.xlabel('决策方案编号', fontsize=25)
plt.ylabel('总成本', fontsize=25)

# 使用字符标注横坐标
plt.xticks(range(len(top_10_profit)), top_10_profit['决策方案编号'].astype(str), rotation=45)

# plt.tight_layout()  # 自动调整布局
# 设置坐标轴刻度字体大小
plt.tick_params(axis='both', labelsize=20)
plt.grid(axis='y')
plt.savefig("pics/问题三：总成本最大的10个决策方案直方图.eps", format='eps')
plt.savefig("pics/问题三：总成本最大的10个决策方案直方图.png", format='png')
# 显示图表
plt.show()
top_10_profit.to_excel('问题三：决策方案中成本最高的10个.xlsx', index=False)
```

pro4-1.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams

# 设置中文字体以支持中文标签
rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体显示中文
rcParams['axes.unicode_minus'] = False  # 正常显示负号

# 定义不同检测情况下的参数，包括次品率、检测成本、市场售价等
cases = [
    {'零配件1次品率': 0.096, '零配件2次品率': 0.1065, '成品次品率': 0.0924, '零配件1检测成本': 2, '零配件2检测成本': 3,
```

```
12      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用': 5},
13      {'零配件1次品率': 0.1667, '零配件2次品率': 0.1962, '成品次品率': 0.223, '零配件1检测成本': 2, '零配件2检测成本': 3,
14      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用': 5},
15      {'零配件1次品率': 0.0873, '零配件2次品率': 0.112, '成品次品率': 0.1242, '零配件1检测成本': 2, '零配件2检测成本': 3,
16      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用': 5},
17      {'零配件1次品率': 0.2143, '零配件2次品率': 0.2171, '成品次品率': 0.1943, '零配件1检测成本': 1, '零配件2检测成本': 1,
18      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用': 5},
19      {'零配件1次品率': 0.0972, '零配件2次品率': 0.1736, '成品次品率': 0.1042, '零配件1检测成本': 8, '零配件2检测成本': 1,
20      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解费用': 5},
21      {'零配件1次品率': 0.0573, '零配件2次品率': 0.0620, '成品次品率': 0.0327, '零配件1检测成本': 2, '零配件2检测成本': 3,
22      '零配件1购买单价': 4, '零配件2购买单价': 18, '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解费用': 40}
23  ]
24
25
26  # 计算总利润
27  def total_profit_cal(case, n, BUY, detect_parts1, detect_parts2, detect_final, dismantle):
28      """
```

```
    计算总成本和总利润
    参数:
    case (dict): 包含当前情况的各种参数。
    detect_parts1 (bool): 是否检测零配件1。
    detect_parts2 (bool): 是否检测零配件2。
    detect_final (bool): 是否检测成品。
    dismantle (bool): 是否拆解不合格成品。
    返回:
    tuple: 包含总成本和缺陷率。
    """
    n_parts1 = n  # 假设零配件1数量
    n_parts2 = n  # 假设零配件2数量
    # 零配件检测成本
    cost_parts1 = n_parts1 * (case['零配件1检测成本'] if detect_parts1 else 0 + case['零配件1购买单价']) if BUY else 0
    cost_parts2 = n_parts2 * (case['零配件2检测成本'] if detect_parts2 else 0 + case['零配件2购买单价']) if BUY else 0

    # 计算未检测零配件情况下的损失
    # loss_parts1 = (n_parts1 * case['零配件1次品率']) * (case['装配成本']) if not detect_parts1 else 0
    # loss_parts2 = (n_parts2 * case['零配件2次品率']) * (case['装配成本']) if not detect_parts2 else 0

    # 装配前的零件数
    before_final_n_parts1 = n_parts1 * ((1 - case['零配件1次品率']) if detect_parts1 else 1)
    before_final_n_parts2 = n_parts2 * ((1 - case['零配件2次品率']) if detect_parts2 else 1)

    # 成品数量
    n_final_products = min(before_final_n_parts1, before_final_n_parts2)
```

```
55
56      # 计算装配成本
57      cost_assembly = n_final_products * case['装配成本']
58
59      # 计算检测成品成本
60      cost_final = n_final_products * case['成品检测成本'] if
    detect_final else 0
61
62      # 零件加工前的正品率
63      standard_rate1 = 1 if detect_parts1 else (1 - case['零配件
    1次品率'])
64      standard_rate2 = 1 if detect_parts2 else (1 - case['零配件
    2次品率'])
65
66      # 更新后的成品次品率
67      updated_defective_rate = 1 + (case['成品次品率'] - 1)*(
    standard_rate1 * standard_rate2)
68
69      # 不合格的成品
70      defective_final = n_final_products *
    updated_defective_rate
71
72      # 计算不检测成品的调换损失
73      loss_final = defective_final * case['调换损失'] if not
    detect_final else 0
74
75      # 拆解成本
76      dismantle_cost = defective_final * case['拆解费用'] if
    dismantle else 0
77
78      # 拆解后能挽回的损失
79      if dismantle and (defective_final > 2):
80          a, b, dismantle_revenue = total_profit_cal(case,
    defective_final, False, detect_parts1, detect_parts2,
    detect_final, dismantle)
```

```
    else:
         dismantle_revenue = 0


    #卖出去的件数
    n_sell = n_final_products * (1 - updated_defective_rate)

    # 收入
    total_gain = case['市场售价'] * n_sell + dismantle_revenue
     # (if detect_final else 1)

    # 计算总成本
    total_cost = (cost_parts1 + cost_parts2 + cost_assembly +
                  cost_final + loss_final + dismantle_cost)

    # 利润
    total_profit = round(total_gain - total_cost)



    return total_cost, total_gain, total_profit


# 定义所有检测策略的组合
strategies = []
for detect_parts1 in [True, False]:
    for detect_parts2 in [True, False]:
        for detect_final in [True, False]:
            for dismantle in [True, False]:
                # strategies;
                strategies.append({
                    "detect_parts1": detect_parts1,
                    "detect_parts2": detect_parts2,
                    "detect_final": detect_final,
                    "dismantle": dismantle
```

```
                })

# 生成策略说明
strategy_explanations = []
for i, strategy in enumerate(strategies):
    strategy_explanations.append(
        f"策略 {i + 1}: 检测零配件 1 {'是' if strategy['detect_parts1'] else '否'}, "
        f"检测零配件 2 {'是' if strategy['detect_parts2'] else '否'}, "
        f"检测成品 {'是' if strategy['detect_final'] else '否'}, "
        f"拆解不合格成品 {'是' if strategy['dismantle'] else '否'}"
    )

    # 存储每种情况的成本和缺陷率
costs = []
profits = []

# 遍历每种情况并计算成本和缺陷率
for i, case in enumerate(cases):
    print(f"\n情况 {i + 1}:")
    case_costs = []
    case_total_profit = []
    for j, strategy in enumerate(strategies):
        total_cost, total_gain, total_profit = total_profit_cal(case, 100, True, **strategy)
        case_costs.append(total_cost)
        case_total_profit.append(total_profit)
        print(f"{strategy_explanations[j]}: 总成本 = {total_cost:.1f}, 总收入 = {total_gain:.1f},总利润 = {total_profit:.1f}")
    costs.append(case_costs)
    profits.append(case_total_profit)
```

```
143
144     # 将利润转换为NumPy数组以便于处理
145 costs = np.array(costs)
146 profits = np.array(profits)
147
148
149 # # 绘制不同情况和策略下的总成本比较图
150 # plt.figure(figsize=(10, 6))
151 # for i in range(costs.shape[1]):
152 #     plt.plot(range(1, costs.shape[0] + 1), costs[:, i],
    marker='o', label=f"策略 {i + 1} - 总成本")
153 # plt.xlabel("情况")
154 # plt.ylabel("总成本")
155 # plt.title("不同情况和策略下的总成本比较")
156 # plt.legend()
157 # plt.grid(True)
158 # plt.show()
159 #%%
160 plt.figure(figsize=(15, 9))
161 for i in range(profits.shape[1]):
162     plt.plot(range(1, profits.shape[0] + 1), profits[:, i],
    marker='o', label=f"决策{i + 1}")
163 plt.xlabel("情况", fontsize=25)
164 plt.ylabel("总利润", fontsize=25)
165 # plt.title("不同情况和策略下的总利润比较")
166 # 设置坐标轴刻度字体大小
167 plt.tick_params(axis='both', labelsize=20)  # 设置 x 和 y 轴刻
    度字体大小
168 plt.legend(loc='upper left', bbox_to_anchor=(0.99, 1),
    fontsize=15)
169 plt.grid(True)
170 # 保存图形为 .eps 文件
171 plt.savefig("pics/问题四：不同决策方案在不同情况下的总利润曲线
    .eps", format='eps')
172 plt.savefig("pics/问题四：不同决策方案在不同情况下的总利润曲线
```

```
    .png", format='png')
plt.show()

# 找到每一行的最大值及其索引
max_values = np.max(profits, axis=1)  # 每一行的最大值
max_indices = np.argmax(profits, axis=1)  # 每一行最大值的列索引

# 输出结果
for row in range(profits.shape[0]):
    col = max_indices[row]
    print(f"情况 {row + 1}: 使用决策方案 {col + 1}，所需总成本{costs[row,col]:.1f}元，可获得最大利润{max_values[row]:.1f}元")


#%%
    # 提取第一行数据
    first_row_data = profits[0, :]  # 取第一行的前6个策略数据

    # 策略标签
    strategies = [f"{i + 1}" for i in range(16)]

    # 绘制直方图
    plt.figure(figsize=(17, 8))

    bars = plt.bar(strategies, first_row_data, color='skyblue')

    # 在每个柱子上添加数据标签
    for bar in bars:
        yval = bar.get_height()  # 获取柱子的高度
        plt.text(bar.get_x() + bar.get_width() / 2, yval, f'{yval:.1f}',
                 ha='center', va='bottom', fontsize=15)  # 在
```

```
    柱子上方添加文本
202
203     # 设置坐标轴标签和标题
204     plt.xlabel("决策方案", fontsize=25)
205     plt.ylabel("利润", fontsize=25)
206     # plt.title("策略利润直方图", fontsize=20)
207
208     # 设置坐标轴刻度字体大小
209     plt.tick_params(axis='both', labelsize=15)
210
211     # 显示网格
212     plt.grid(axis='y')
213
214     plt.savefig("pics/问题四：不同决策方案在情况1下的总利润直
    方图.eps", format='eps')
215     plt.savefig("pics/问题四：不同决策方案在情况1下的总利润直
    方图.png", format='png')
216     # 显示图形
217     plt.show()
218
219 #%%
220     # 提取第一行数据
221     first_row_data = profits[2, :]  # 取第一行的前6个策略数据
222
223     # 策略标签
224     strategies = [f"{i + 1}" for i in range(16)]
225
226     # 绘制直方图
227     plt.figure(figsize=(17, 8))
228
229     bars = plt.bar(strategies, first_row_data, color='skyblue'
    )
230
231     # 在每个柱子上添加数据标签
232     for bar in bars:
```

```
        yval = bar.get_height()  # 获取柱子的高度
        plt.text(bar.get_x() + bar.get_width() / 2, yval, f'{yval:.1f}',
                 ha='center', va='bottom', fontsize=15)  # 在柱子上方添加文本

    # 设置坐标轴标签和标题
    plt.xlabel("决策方案", fontsize=25)
    plt.ylabel("利润", fontsize=25)
    # plt.title("策略利润直方图", fontsize=20)

    # 设置坐标轴刻度字体大小
    plt.tick_params(axis='both', labelsize=15)

    # 显示网格
    plt.grid(axis='y')

    plt.savefig("pics/问题四：不同决策方案在情况3下的总利润直方图.eps", format='eps')
    plt.savefig("pics/问题四：不同决策方案在情况3下的总利润直方图.png", format='png')
    # 显示图形
    plt.show()
```

pro4-2.py

```

#%%
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
import pandas as pd
from matplotlib import rcParams

# 设置中文字体以便于图表显示
rcParams['font.sans-serif'] = ['SimHei']
```

```
rcParams['axes.unicode_minus'] = False

# 定义零配件的属性，包括次品率、购买单价和检测成本
components = {
    '零配件1': {'次品率': 0.1296, '购买单价': 2, '检测成本': 1},
    '零配件2': {'次品率': 0.0741, '购买单价': 8, '检测成本': 1},
    '零配件3': {'次品率': 0.0926, '购买单价': 12, '检测成本': 2},
    '零配件4': {'次品率': 0.1065, '购买单价': 2, '检测成本': 1},
    '零配件5': {'次品率': 0.0873, '购买单价': 8, '检测成本': 1},
    '零配件6': {'次品率': 0.0825, '购买单价': 12, '检测成本': 2},
    '零配件7': {'次品率': 0.1133, '购买单价': 8, '检测成本': 1},
    '零配件8': {'次品率': 0.1120, '购买单价': 12, '检测成本': 2},
}

# 定义半成品的属性，包括次品率、装配成本、检测成本和拆解费用
semi_products = {
    '半成品1': {'次品率': 0.0972, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品2': {'次品率': 0.1389, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品3': {'次品率': 0.1211, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
}

# 定义成品的属性，包括次品率、装配成本、检测成本、拆解费用和市场售价
final_product = {
```

```
    '成品': {'次品率': 0.1736, '装配成本': 8, '检测成本': 6, '拆解费用': 10, '售价': 200, '调换损失': 40}
}

def total_profit_cal(n, BUY,  components, semi_products, final_product, detect_components,
                             detect_semi, semi_dismantle, detect_final, dismantle):
    """
    计算预期总成本和缺陷率。
    参数:
    components (dict): 零配件信息。
    semi_products (dict): 半成品信息。
    final_product (dict): 成品信息。
    detect_components (list): 是否检测各零配件的布尔列表。
    detect_semi (list): 是否检测各半成品的布尔列表。
    detect_final (bool): 是否检测成品的布尔值。
    dismantle (bool): 是否拆解不合格成品的布尔值。
    返回:
    tuple: 总成本和缺陷率。
    """
    n_comp = np.zeros(8)  # 变成半成品之前的零件个数
    n_semi = np.zeros(3)  # 变成成品之前的半成品个数
    n_semi_dismantle = np.zeros(3)
    # 零件加工前的正品率
    standard_rate = np.zeros(8)

    total_cost = 0  # 初始化总成本
    # total_defective_rate = 1  # 初始化总缺陷率

    # 计算零配件成本和总缺陷率
    for i, (comp_name, comp_data) in enumerate(components.items()):

        if i < 3:
```

```
            n_separate = n[0]  # 第一个零件组
        elif i < 6:
            n_separate = n[1]   # 第二个零件组
        else:
            n_separate = n[2]   # 第三个零件组
        cost = n_separate * comp_data['购买单价'] if BUY else 0  # 零配件的购买成本
        if detect_components[i]:  # 如果检测该零配件
            cost += n_separate * comp_data['检测成本']  # 加上检测成本
            standard_rate[i] = 1
        else:
            standard_rate[i] = 1 - comp_data['次品率']
        n_comp[i] = standard_rate[i] * n_separate

        total_cost += cost  # 累加到总成本

    updated_defective_semi_rate = np.zeros(3)  # 创建一个包含3个零的数组

    # 计算乘积并存储在 n_semi 中
    updated_defective_semi_rate[0] = semi_products['半成品1']['次品率']
    updated_defective_semi_rate[1] = semi_products['半成品2']['次品率']
    updated_defective_semi_rate[2] = semi_products['半成品3']['次品率']

    # 组装成的半成品个数
    n_semi[0] = np.min(n_comp[0:2])  # 第1-3个值的最小值
    n_semi[1] = np.min(n_comp[3:5])  # 第4-6个值的最小值
    n_semi[2] = np.min(n_comp[6:7])  # 第7-8个值的最小值

    defective_semi = np.zeros(3)

```

```
94      # 半成品的正频率
95      semi_standard_rate = np.zeros(3)
96
97      # 计算半成品成本和总缺陷率
98      for i, (semi_name, semi_data) in enumerate(semi_products.
    items()):
99          cost = n_semi[i] * semi_data['装配成本']  # 半成品的装
    配成本
100
101         if detect_semi[i]:  # 如果检测该半成品
102             defective_semi[i] = n_semi[i] *
    updated_defective_semi_rate[i]
103             cost += 4 * n_semi[i]  # 加上检测成本
104             semi_standard_rate[i] = 1
105         else:
106             semi_standard_rate[i] = 1 -
    updated_defective_semi_rate[i]
107         # n_semi_dismantle[i] = (1 - semi_standard_rate[i]) *
    n_semi[i]
108         n_semi[i] = semi_standard_rate[i] * n_semi[i]
109         total_cost += cost  # 累加到总成本
110
111     if np.all(detect_semi) and semi_dismantle and (np.min(
    defective_semi) > 2):  # 如果所有半成品均已检测且需要拆解
112
113         cost = np.sum(defective_semi) * 6  # 加上拆解费用
114
115         # 拆解后，重新计算相关零件费用
116         dismantled_comp_cost, dismantled_gain,
    dismantled_profit = total_profit_cal(
117             defective_semi, False, components, semi_products,
    final_product,
118             detect_components, detect_semi, semi_dismantle,
119             detect_final, dismantle  # 为了避免重复拆解
120         )
```

```
121         total_cost += cost - dismantled_profit
122
123     updated_defective_final_rate = final_product['成品']['次品率']
124
125     # 最终的成品数量
126     n_final = np.min(n_semi)
127     # 卖出去的成品只有正品
128     n_sell = n_final * (1 - updated_defective_final_rate)
129     # 不合格的成品
130     defective_final = n_final * updated_defective_final_rate
131
132     # 计算成品的成本
133     # final_defective_rate = final_product['成品']['次品率'] * (0.5 if detect_final else 1)  # 成品的次品率
134     final_cost = n_final * final_product['成品']['装配成本']  # 成品的装配成本
135
136     if detect_final:  # 如果检测成品
137         final_cost += n_final * final_product['成品']['检测成本']  # 加上检测成本
138
139     else:               # 如果不检测成品
140         final_cost += defective_final * final_product['成品']['调换损失']
141
142     total_cost += final_cost  # 累加到总成本
143
144     if dismantle and (defective_final > 2):  # 如果拆解不合格成品
145         total_cost += defective_final * final_product['成品']['拆解费用']  # 加上拆解费用
146         # 拆解后，重新计算相关零件费用
147         dismantled_final_cost, dismantled_final_gain, dismantled_final_profit = total_profit_cal(
```

```
148             defective_final * np.ones(3), False, components,
    semi_products, final_product,
149             detect_components, detect_semi, semi_dismantle,
150             detect_final, dismantle  # 为了避免重复拆解
151         )
152         total_cost += - dismantled_final_profit
153     # 收入
154     total_gain = final_product['成品']['售价'] * n_sell  #*
    ((1 - updated_defective_rate) if detect_final else 1) +
    dismantle_revenue
155
156     # 利润
157     total_profit = round(total_gain - total_cost)
158
159     # 返回总成本和利润
160     return round(total_cost), round(total_gain), total_profit
161
162     # 生成所有可能的检测组合
163
164
165 component_combinations = list(product([True, False], repeat=8)
    )  # 零配件检测组合
166 semi_combinations = list(product([True, False], repeat=3))  #
    半成品检测组合
167 semi_dismantle_combinations = list(product([True, False],
    repeat=1))  # 半成品拆解组合
168 final_combinations = list(product([True, False], repeat=2))  #
     成品检测和拆解组合
169
170 strategy_descriptions = []  # 策略描述列表
171 results = []  # 结果列表
172
173
174 def generate_strategy_description(detect_components,
    detect_semi, semi_dismantle, detect_final, dismantle):
```

```
    """
    生成策略描述。

    参数:
    detect_components (list): 零配件检测布尔列表。
    detect_semi (list): 半成品检测布尔列表。
    detect_final (bool): 成品检测布尔值。
    dismantle (bool): 拆解布尔值。

    返回:
    str: 策略描述字符串。
    """
    description = ""
    for i, detect in enumerate(detect_components):
        description += f"检测零配件 {i + 1}, " if detect else f"不检测零配件 {i + 1}, "
    for i, detect in enumerate(detect_semi):
        description += f"检测半成品 {i + 1}, " if detect else f"不检测半成品 {i + 1}, "
    for i, dismantle in enumerate(semi_dismantle):
        description += f"拆解半成品, " if semi_dismantle else f"不拆解半成品, "

    description += "检测成品, " if detect_final else "不检测成品, "
    description += "拆解不合格成品" if dismantle else "不拆解不合格成品"
    return description


# 策略编号初始化
strategy_number = 1
# 遍历所有组合, 计算成本和缺陷率
for comp_comb in component_combinations:
    for semi_comb in semi_combinations:
```

```
205         for final_comb in final_combinations:
206             for semi_dismantle_comb in
    semi_dismantle_combinations:
207                 semi_dismantle = semi_dismantle_comb  # 当前半
    成品拆解组合
208                 detect_components = comp_comb  # 当前零配件检
    测组合
209                 detect_semi = semi_comb  # 当前半成品检测组合
210                 detect_final = final_comb[0]  # 当前成品检测状
    态
211                 dismantle = final_comb[1]  # 当前拆解状态
212
213                 # 计算当前策略的总成本和利润
214                 total_cost, total_gain, total_profit =
    total_profit_cal([100, 100, 100], True, components,
    semi_products, final_product, detect_components,
215
     detect_semi, semi_dismantle, detect_final, dismantle)
216
217                 # 生成当前策略的描述
218                 strategy_description =
    generate_strategy_description(detect_components, detect_semi
    , semi_dismantle, detect_final,
219
           dismantle)
220                 results.append([strategy_number,
    strategy_description, total_cost, total_gain, total_profit])
      # 将结果添加到列表
221                 strategy_number += 1  # 策略编号递增
222
223 # 将结果转换为DataFrame并输出
224 df = pd.DataFrame(results, columns=['决策方案编号', '决策方案
    描述', '总成本', '总收入', '总利润'])
225 print(df.head())  # 打印前五条策略结果
226
```

```
# 将结果保存到Excel文件
df.to_excel('问题四决策方案结果.xlsx', index=False)
print("所有策略已经保存到 '决策方案结果.xlsx' 文件中。")


#%%
# 绘制总成本比较图
plt.figure(figsize=(10, 6))
plt.plot(df['决策方案编号'], df['总成本'], marker='o',
    linestyle='-', color='blue', label="总成本")
plt.xlabel("决策方案编号", fontsize=25)
plt.ylabel("总成本", fontsize=25)
# plt.title("不同决策方案下的总成本比较", fontsize=25)
plt.grid(True)
plt.legend(fontsize=20)
plt.savefig("pics/问题四：不同决策方案的总成本折线图.eps",
    format='eps')
plt.savefig("pics/问题四：不同决策方案的总成本折线图.png",
    format='png')
plt.show()

# 绘制总利润比较图
plt.figure(figsize=(10, 6))
plt.plot(df['决策方案编号'], df['总利润'], marker='x',
    linestyle='--', color='red', label="总利润")
plt.xlabel("决策方案编号", fontsize=25)
plt.ylabel("总利润", fontsize=25)
# plt.title("不同决策方案下的总利润比较")
plt.grid(True)
plt.legend(fontsize=20)
plt.savefig("pics/问题四：不同决策方案的总利润折线图.eps",
    format='eps')
plt.savefig("pics/问题四：不同决策方案的总利润折线图.png",
    format='png')
plt.show()
```

```
256
257 #%%
258
259
260 # 找出总利润最大的10行
261 top_10_profit = df.nlargest(10, '总利润')
262
263 # 输出决策方案描述及其 '总成本', '总收入', '总利润'
264 #print(top_10_profit[['决策方案描述']].to_string(index=False),
      "所得的总利润为: ", top_10_profit[['总利润']].to_string(
    index=False))
265
266 # 绘制直方图
267 plt.figure(figsize=(16, 10))
268 bars = plt.bar(range(len(top_10_profit)), top_10_profit['总利
    润'], color='skyblue')
269
270 # 在每个条形上标出数据
271 for bar in bars:
272     yval = bar.get_height()
273     plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval
    , 2), ha='center', va='bottom', fontsize=25)
274
275 # 设置图表标题和标签
276 #plt.title('总利润最大的10个决策方案')
277 plt.xlabel('决策方案编号', fontsize=25)
278 plt.ylabel('总利润', fontsize=25)
279
280 # 使用字符标注横坐标
281 plt.xticks(range(len(top_10_profit)), top_10_profit['决策方案
    编号'].astype(str), rotation=45)
282
283 # plt.tight_layout()  # 自动调整布局
284 # 设置坐标轴刻度字体大小
285 plt.tick_params(axis='both', labelsize=20)
```

```
plt.grid(axis='y')
plt.savefig("pics/问题四：总利润最大的10个决策方案直方图.eps", format='eps')
plt.savefig("pics/问题四：总利润最大的10个决策方案直方图.png", format='png')
# 显示图表
plt.show()
top_10_profit.to_excel('问题四：决策方案中利润最高的10个.xlsx', index=False)

#%%
# 找出总利润最大的10行
top_10_profit = df.nlargest(10, '总利润')

# 输出决策方案描述及其 '总成本', '总收入', '总利润'
#print(top_10_profit[['决策方案描述']].to_string(index=False), "所得的总利润为：", top_10_profit[['总利润']].to_string(index=False))

# 绘制直方图
plt.figure(figsize=(16, 10))
bars = plt.bar(range(len(top_10_profit)), top_10_profit['总成本'], color='skyblue')

# 在每个条形上标出数据
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom', fontsize=25)

# 设置图表标题和标签
#plt.title('总成本最大的10个决策方案')
plt.xlabel('决策方案编号', fontsize=25)
plt.ylabel('总成本', fontsize=25)

```

```python
# 使用字符标注横坐标
plt.xticks(range(len(top_10_profit)), top_10_profit['决策方案编号'].astype(str), rotation=45)

# plt.tight_layout()  # 自动调整布局
# 设置坐标轴刻度字体大小
plt.tick_params(axis='both', labelsize=20)
plt.grid(axis='y')
plt.savefig("pics/问题四：总成本最大的10个决策方案直方图.eps", format='eps')
plt.savefig("pics/问题四：总成本最大的10个决策方案直方图.png", format='png')
# 显示图表
plt.show()
top_10_profit.to_excel('问题四：决策方案中成本最高的10个.xlsx', index=False)
```

pro4-3.m

```
%% 截尾序贯抽样检测

% 设置随机种子确保结果可重复
rng(42);
N = 10000;                % 总样本大小
p0 = 0.1;                 % 标称次品率
data = rand(1, N) < p0;
%%
% 设置随机种子确保结果可重复
clc;


% N = 100;                % 总样本大小
% p0 = 0.2;                % 标称次品率
% data = rand(1, N) < p0; % 创建随机0-1序列

```

```
% 定义参数
max_rounds = 4;       % 最多检测轮数
Z_alpha_95 = norminv(0.975);  % 95%信度的Z值
Z_alpha_90 = norminv(0.975);   % 90%信度的Z值



sample_size = 18;  % 从10到100的样本量
    results_90 = [];
    n_total_90 = 0;  % 90%信度下的次品总数
    accepted_90 = false; % 90%信度下是否接受

    for round = 1:max_rounds
        if (round - 1) * sample_size + sample_size > N
            break; % 确保不会超出总样本数
        end

        indices = randperm(N, sample_size); % 生成随机索引
        sample = data(indices); % 根据随机索引抽样
        % sample = data((round - 1) * sample_size + 1: round * sample_size); % 抽样
        n = sum(sample);  % 次品数量
        n_total_90 = n_total_90 + n;  % 统计90%信度下的次品总数

        % 更新均值和标准差
        mean_90 = n_total_90 / round;

        results_90(end+1) = n;  % 继续抽样
        std_dev_90 = std(results_90);  % 次品数量的标准差

        % 计算接受范围
        L_90 = mean_90 - Z_alpha_90 * (std_dev_90 / sqrt(round));
        U_90 = mean_90 + Z_alpha_90 * (std_dev_90 / sqrt(round
```

```
    ));
50
51          % 检测接受与否
52          if n < L_90
53              accepted_90 = true;  % 接受这批零配件
54
55
56              break;  % 停止抽样
57          elseif n > U_90
58              accepted_90 = false;  % 拒绝这批零配件
59
60              break;  % 停止抽样
61
62
63          end
64      end
65
66      % 计算检测次数
67      detection_count = round * sample_size;
68
69          final_defect_rate_90 = n_total_90 / (round  *
    sample_size ); % 95%信度下的次品率
70
71
72
73  % 输出95%信度下的结果
74  fprintf('在95%%信度下接受的情况下，次品率：%.2f%%，\n最小检测
    次数：%d，最优样本量：%d，检测轮数：%d\n', ...
75          final_defect_rate_90 * 100, detection_count,
    sample_size, round);
```