

基于抽样检测与优化决策的企业生产过程质量控制模型研究

摘要

本文主要研究企业生产过程中原料质量监测、商品生产销售过程中涉及的决策优化问题，建立抽样检测模型来辅助企业做好原料质量把控，并就生产过程的复杂情况提出决策方案。

针对问题一，本文将企业遇到的原料总体供应量划为大样本和小样本两种情况，分析得到在小样本情况下，零件的次品数量服从二项分布，在大样本情况下，次品数量均值近似服从正态分布，通过建立统计检验量和显著性水平所对应临界值的大小关系，采用迭代遍历的方法，得到了 I. 在小样本下，两种情况的抽样样本数与次品界值对应表。（详见正文）。II. 在大样本下，求解出在 95% 的信度和 90% 的信度下认定零配件次品率是否合格的最小样本数分别为 138 和 97，对应抽取的次品界值分别为 20 和 13。

针对问题二，在具体生产过程中，选取四种决策关键点：零件 1 是否检测，零件 2 是否检测，成品是否检测，次品是否拆解，通过排列组合出 16 种情况，以利润最大化为决策目标，穷举遍历 16 种情况，对于这四种决策关键点，假定 1 表示执行，0 表示不执行，则可以得到在问题二的六种情况下，最优策略分别为

$$\{0, 0, 0, 1\}, \{1, 1, 0, 1\}, \{0, 0, 1, 1\}, \{1, 1, 1, 1\}, \{0, 0, 1, 1\}, \{0, 0, 0, 0\}$$

针对问题三，对于多个零件，多道工序的综合分析，我们采取从特殊到一般的解决思路，首先依然利用穷举法解决 2 道工序，8 种零配件的特殊情况，其后推广到 m 道工序、 n 个零部件的一般情况，采用模拟退火算法寻找全局近似最优解，得出解决 2 道工序，8 个零配件生产过程的最优决策为对所有零件和半成品进行检测，对所有半成品进行拆解，对成品不检测但进行拆解。

针对问题四，我们在问题三的基础上提出了一种模拟策略来估计实际次品率。具体步骤如下：I. 样本生成与次品率模拟：使用理论次品率生成样本，并进行抽样检测以统计次品率。通过加入随机扰动来模拟实际检测中的波动，并计算标准误差和置信区间，从而得到模拟的实际次品率。II. 应用于问题二和问题三：将模拟得到的次品率应用于问题二和问题三的决策优化中。结果表明，在问题二中，最初确定的六种最优方案在存在次品率波动的情况下，仍然有较高概率成为最优方案。在问题三中，尽管存在维度灾难等挑战，选择前百种盈利策略后，初始的最优方案之一仍然保持了最优性。

通过这些模拟策略，我们能够更准确地反映次品率的实际情况，并对生产决策进行有效优化。

关键字： 抽样检测，决策问题，遍历求解，模拟退火

一、问题重述

1.1 问题背景

本问题反映了现代生产环境中普遍存在的挑战，特别是在电子产品市场竞争日益激烈的背景下。企业在确保产品质量和成本控制之间需要找到一个平衡点，这涉及到复杂的决策分析和风险管理。

在本文中，我们针对某企业生产流程中的关键决策进行研究。企业生产的是一种市场畅销的电子产品，该产品的生产需要购买两种关键的零配件（零配件 1 和零配件 2）。在生产中，任何一个零配件的不合格都会导致成品不合格；即便两个零配件均合格，最终的成品也可能不合格。对于不合格的成品，企业可以选择报废或者拆解。拆解过程虽然不会对零配件造成损害，但会产生额外的拆解费用。

企业面临的主要挑战包括如何有效地通过抽样检测来确定是否接受供应商提供的零配件，以及如何在生产过程中针对零配件和成品的质量控制作出最优的决策。这些决策不仅影响产品质量，还直接关联到成本控制和市场供应的稳定性。

通过建立数学模型来模拟和优化这些生产决策，企业能够更有效地管理风险，提高生产效率，从而在保证产品质量的同时控制成本，保持市场竞争力。这种模型的建立涉及多学科知识的融合，包括运筹学、统计学、经济学和工程技术，是现代工业工程和管理领域中的一项关键任务。

1.2 问题要求

问题 1：零配件质量抽样检测方案设计

- 目标：设计一种抽样检测方案，用以决定是否接受供应商提供的零配件。要求方案能在尽可能少的检测次数下，判断零配件的次品率是否超出供应商声明的标准。
- 具体要求：
 1. 如果供应商声称的次品率标准为 10%，设计一个方案，在 95% 的信度下如果检测得到的次品率超过此值，则决定拒收零配件。
 2. 在 90% 的信度下如果检测得到的次品率不超过标准值，则决定接收零配件。

问题 2：生产过程中的质量控制决策

- 目标：为企业的生产过程中的质量控制提供决策支持，以优化成品质量并控制相关成本。
- 具体要求：

1. 决定是否对每种零配件进行质量检测，未检测的直接进入装配环节。
2. 决定是否对装配好的成品进行检测，未检测的成品直接销售。
3. 决定是否对不合格成品进行拆解，拆解后的零配件可重新进入生产流程。
4. 设计处理顾客退回不合格产品的策略，包括调换损失的考量。

问题 3：复杂生产线的决策模型

- 目标：为包含多个工序和多种零配件的复杂生产线设计质量控制和生产决策模型。
- 具体要求：
 1. 给定各种零配件、半成品和成品的次品率，设计一个决策方案来优化生产过程。
 2. 考虑到各个组件的购买成本、检测成本、装配成本和市场售价，提出成本效益最优的生产策略。

问题 4：基于实际检测数据的生产策略调整

- 目标：在实际得到的次品率数据基础上，重新评估和优化之前设计的生产和质量控制策略。
- 具体要求：
 1. 假设问题 2 和问题 3 中的次品率数据是通过抽样检测获得的，使用这些数据重新完成相关的生产和质量控制决策。

二、问题分析

2.1 对问题一的分析

假设各个零部件的合格情况为 $\{X_1, X_2, X_3, \dots\}$ ，当从一批零部件中抽取 N 个样本时，次品数量 X 服从二项分布。由于这些零件属于同一批次，我们假设它们的合格情况相互独立且服从相同分布（即独立同分布）。在大样本条件下，即当样本容量 $N > 30$ 时，中心极限定理能够较好成立，二项分布可以较好地近似为正态分布。根据相关论文 [1] 指出，正态分布完全适用于样本容量 $N > 30$ ，此时期望和方差几乎对适用样本量几乎没有影响，而在小样本情况下，中心极限定理不适用，不能近似，因而使用原始的二项分布。基于题目中给出的置信度要求，我们构建了一个在大样本条件基于正态分布的假设检验模型和在小样本条件下基于二项分布的假设检验模型。

为了使模型适用于更广泛的情境，我们还考虑了小样本情况。在小样本条件下，由于中心极限定理不再适用，但样本的合格情况仍满足二项分布。在这种情况下，我们采用二项分布的抽样概率表达式进行度量分析。

本问题的关键在于确定不合格率是否超过标称值，由于这一问题具有明确的方向性预测，我们可以采用单尾检验。在单尾检验的框架下，次品的概率分布可以近似为正态分布，进而可以推导出样本数量 N 与标准正态分布临界值 Z 之间的约束关系。通过迭代方法求解，我们能够得到满足置信度要求的最小样本量 N 。

2.2 对问题二的分析

问题二是一个典型的多过程决策优化问题，从过程分析可知，影响决策的因素主要有四个，分别是零件 1 的检查情况，零件 2 的检查情况，成品检查情况，不合格成品的拆卸情况，决策点不同，会导致产生的利润，成本花费不同，根据排列组合理论，可以得到企业面临的每一种情况都有 16 种策略可供选择，将其视为以利润最大化的单目标优化问题，利用固定的零件合格率以及模型假设中的生产过程中的生产策略一致性原则并对特殊情况可能引起成品合格率波动较大的情况进行具体分析，逐步计算每一步决策所带来的利润，通过比较，得到最终企业遇到的每一种情况的最优方案。

2.3 对问题三的分析

问题三是一个多阶段、多工序的复杂生产决策问题，涉及零配件、半成品和成品的次品率及其检测与拆解的组合决策。核心问题在于企业如何在生产过程中权衡不同环节的检测和拆解成本，最大化总体利润。该问题的复杂性在于每个决策点都与检测或拆解有关，决策点的不同组合会直接影响生产成本与最终成品的合格率，穷举所有决策组合的计算复杂度非常高。因此，为了在可接受的时间内找到接近最优的解，我们引入了模拟退火算法（SA）。通过逐步收敛的方法，模拟退火能够在复杂的决策空间中，平衡各个环节的成本与收益，最终找到最优的生产决策方案，达到利润最大化的目标。

2.4 对问题四的分析

在问题四的新情境中，所有零件、半成品以及成品的次品率都是通过抽样检测方法得到的，次品率并非完全客观的准确值，抽样检测本身给模型带来了不确定性，这在一定程度上会影响我们的决策，因此我们可以用一个模拟策略来估计实际次品率，用理论次品率生成样本，并进行抽样检测以统计实际次品率。通过加入随机扰动模拟检测波动，计算标准误差和置信区间，得到模拟的实际次品率。并对模拟的实际次品率进行验证，理论下应符合正态分布。验证成功后将模拟的次品率应用于问题二和问题三的决策优化中。

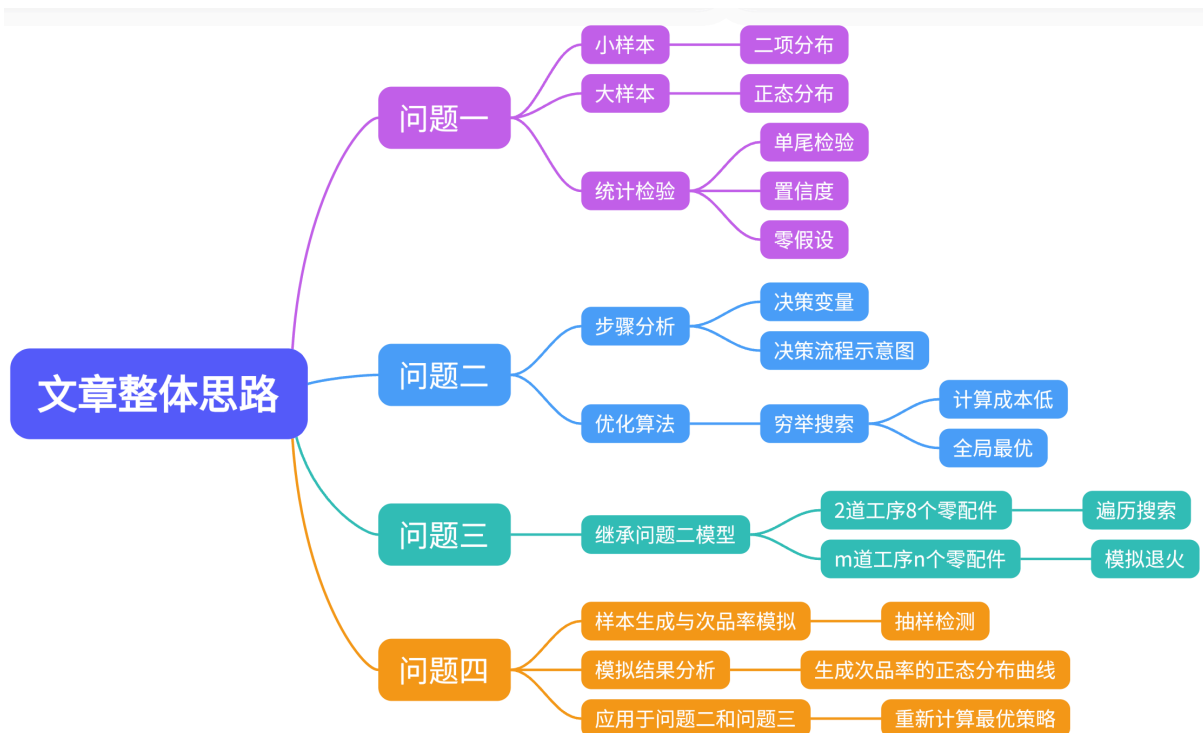


图 1 文章整体思路图

三、模型假设

1. 统一采购量假设：

在问题二和问题三中，假设每种零件的采购数量均为 1000 个。这一数量的设定是为了便于计算，并避免在数量较小时可能出现的取整误差。由于该问题本质上是概率值计算，因此初始数量对模型的结果不会产生实质性影响。

2. 重复检测假设：

为了简化模型，当经过拆解后的零件再次经过步骤一时，我们假设不改变初次检测的决策。重复的检测过程仅会增加成本而不会带来收益。因此，在此模型中，拆解后的零件直接进入步骤二进行处理，不再进行步骤一的检测。

3. 成品装配与次品率的独立性假设：

假设每个零配件的次品率相互独立，且零配件次品率对成品次品率的影响是独立的。

4. 检测过程的准确性假设：

假设每次检测的准确率是 100%，即不会出现漏检或误判的情况。在实际操作中，这一假设帮助简化模型，使我们无需考虑检测误差对决策的影响。

5. 成品不合格的处理成本假设：

假设在成品检测出不合格并经过拆解后，处理这些不合格品的成本与检测出的零配件次品的处理成本是独立的，且不考虑潜在的返工损失。

四、符号说明

符号	说明	单位
d_i	问题二四个决策变量	无单位
N_p	装配成品数量	件
N_1, N_2	零件 1 和零件 2 库存数量	件
Q_p	成品合格率	无单位
Q_1, Q_2	零件 1 和零件 2 合格率	无单位
d_p	成品次品率	无单位
N_q	合格成品数量	件
N_d	不合格成品数量	件
R	总收入	元
C	总成本	元
c_1, c_2	零件 1 和零件 2 采购成本	元/件
k_1, k_2	零件 1 和零件 2 检测成本	元/件
k_p	装配费用	元/件
C_{return}	退换不合格成品的损失	元
N_r	退换成品数量	件
L_r	调换费用	元/件
c_{dis}	不合格产品的拆解成本	元/件
Π	总利润	元
$N_{\text{semi},i}$	第 i 个半成品的库存数量	件
$N_{\text{semi},j}$	第 j 个半成品的装配数量	件
Q_{semi}	半成品的合格率	无单位
d_{semi}	半成品装配过程的次品率	无单位
C_{purchase}	零件采购成本	元
C_{inspect}	零件或成品的检测成本	元
C_{assemble}	装配成本	元
$C_{\text{disassemble}}$	拆解不合格成品的成本	元
P	成品的市场价格	元/件

五、模型建立与求解

5.1 问题一模型的建立

问题一要求对零配件的次品率进行抽样检测，在标称值次品率为 10% 的情况下，设计一个在两种不同的情形下对于企业来说检测次数尽可能少的抽样检测方案。对于抽取的样品本身而言，数据服从二项分布，在小样本情况下，可以较容易处理，但是在大量样本情况下，我们不考虑直接选取二项分布而是选择正态分布来进行建模，详细的理由如下：

- 中心极限定理：二项分布在样本量较大时，根据中心极限定理，其分布的形状会趋近于正态分布。当样本量大于 30，并且事件发生的概率 p 不接近 0 或 1 时，使用正态分布来近似二项分布是合适的。这使得计算更为简便，特别是在进行假设检验和构建置信区间时。
- 计算简便：正态分布的数学性质比二项分布更简单，尤其在概率计算和统计推断时。正态分布有两个参数（均值和标准差），这使得从理论上分析和应用各种统计技术（如置信区间和假设检验）变得更容易。
- 样本比率的分布：在对比例或比率（例如，合格品率或发生某事件的比率）进行统计分析时，即使原始数据遵循二项分布，其比率的抽样分布在样本量足够大时也可以近似为正态分布。

为了要计算得到抽取样品中的临界的不合格品的数量，我们需要先确定标准正态分布的分位数。

正态分布表（ Z 表）中包含标准正态分布的累计分布函数值，表示某个 Z 值对应的累计概率，对于给定的显著性水平 α ，我们可以通过查表或者使用计算工具来找到相对应的 Z 值。而在统计检验中，常见的检验方法包括双尾检验和单尾检验。两种检验方法适用的情况不同，具体如下：

- 单尾检验用于当研究假设具有明确方向性时。例如，当研究者预测一个特定的处理或条件会导致结果变得更高或更低时使用。在单尾检验中，整个显著性水平（比如 5%）都集中在分布的一端。因此，如果假设检验是为了检测一个值是否显著大于或小于另一个值（而不是不等于），就会使用单尾检验。
- 双尾检验则不预设方向，用于当研究假设只预测两个变量不相等时。在双尾检验中，显著性水平被分成两半，分布在正态分布的两端。例如，如果显著性水平是 5%，则数据分布的两端各占 2.5%。

在企业抽检样品的问题中，很显然我们仅仅需要考虑抽取样品的实际次品率是否超过了标称值，而没有必要考虑另一侧的情况，因此很明显应该采用单尾检验，而对于单尾检验而言，已知显著性水平 α ，即可得到正态分位数为 Z_{α} 。

在本文中，我们给出原假设 H_0 形式如下：

$$H_0 : p \leq p_0$$

其中 p 表示实际次品率， p_0 表示标价值下的次品率。

给出备择假设 H_1 形式如下：

$$H_1 : p > p_0$$

根据题目要求，可以知道一共有两个显著性水平，分别是 $\alpha = 0.05$ 和 $\alpha = 0.1$ ，题目中第一个小问要求有 95% 的把握下能正确认为零配件的次品率超过标价值，也就是说企业能够接受第一类错误限度为 0.05，第二个小问要求有 90% 的把握正确认为零配件的次品率不超过标价值，在这种情况下企业能够接受第二类错误限度为 0.1。引入检验统计量 Z ，结合假设检验统计理论，可以得到检验判别标准如下：

- 当 $Z > Z_\alpha$ 时，应当拒绝原假设
- 当 $Z < Z_\alpha$ 时，应当接受原假设

5.1.1 基于小样本情况下的二项分布假设检验模型

当样本数量较小时，中心极限定理不适用，考虑样本检测结果为次品服从二项分布，即 $X \sim \text{Binomial}(n, p)$ ，由于要确定最小抽样次数，样本的均值，标准差都未知，但由于二项分布的在 n 次实验下的概率 P 是已知，所以我们选取利用 P 和 α 的大小关系进行比较。其中：

$$P(X \geq k) = \sum_{x=k}^n \binom{n}{x} p^x (1-p)^{n-x}$$

当 P 值大于显著性水平 α 时，认为有超过 $1 - \alpha$ 的把握认定其次品率超过标定值

5.1.2 基于大样本情况下的近似正态分布假设检验模型

在处理二项分布假设检验的情况下，当样本量 n 较大时，我们可以应用正态分布的近似来简化计算。这种方法基于中心极限定理，将二项分布近似为正态分布，从而进行单侧假设检验。根据中心极限定理，无论原样本如何分布，随着样本增大，独立同分布的随机变量的样本均值分布会趋向于正态分布 [2]，故而对于二项分布 $X \sim \text{Binomial}(n, p)$ ，其样本均值的分布可以近似为正态分布：

$$\bar{X} \sim \mathcal{N}(np, np(1-p))$$

即次品数量 X 的期望为 np ，方差为 $np(1-p)$ 。于是样本比例 $\hat{p} = \frac{X}{n}$ 也服从正态分布：

$$\hat{p} \sim \mathcal{N}\left(p, \frac{p(1-p)}{n}\right)$$

通过标准化, 可以将该分布转化为标准正态分布。为了进行右尾检验, 我们定义统计量 Z 为:

$$Z = \frac{\hat{p} - p}{\sqrt{\frac{p(1-p)}{n}}}$$

其中:

- \hat{p} 是从样本中计算得到的次品率;
- p 是原假设中的次品率;
- n 是样本量。

在假设 H_0 成立的情况下, 统计量 Z 服从标准正态分布, 即 $Z \sim \mathcal{N}(0, 1)$ 。

为了进行右尾检验, 我们需要计算在给定显著性水平 α 下的临界值 Z_α 。右尾检验意味着我们关心的是 Z 是否超出了这个临界值。由于我们不关注合格率的精确值, 所以允许存在一定的误差, 设误差为 E , 则有:

$$Z_{\alpha/2} = \frac{E}{\sqrt{np(1-p)}}$$

从这个公式中可以解出样本量 n :

$$n = \frac{Z_{\alpha/2}^2 \cdot p(1-p)}{E^2}$$

5.2 问题一模型的求解

在该问题中, 由于我们不知道具体的抽样数量, 而每一个不同的抽样数量的值对应于不同的临界值 k , 因此我们考虑使用计算机程序来求解该问题。

在程序中, 根据不同的模型使用条件, 在小样本情况下, 分别从样本量为 1 到样本量为 29, 逐步增加样本量, 以尝试找到满足误差容忍度要求的最小样本量 n 和对应的临界值 k 。对每一个 n , 遍历所有可能的 k 值 (从 0 到 n), 计算实际的错误率, 并与目标错误率进行比较, 如果某个 k 的实际错误率与目标错误率的差距小于或者等于误差容忍度, 这意味着找到了一对合适的 n 和 k , 程序将返回这些值, 在大样本情况下, 利用推导的式子带入计算得到 n , 并遍历后得到 k 。

5.2.1 小样本模型求解

根据概率论相关文献 [3] 中指出, 对于单边假设检验问题:

$$H_0 : p \leq p_0 \quad \text{vs} \quad H_1 : p > p_0$$

很明显, 一个解决方法是确定拒绝域, X 取整数值, K 为正数, 对于一般情况, 仅需有

$$\sum_{k=c_0}^n \binom{n}{i} p_0^i (1-p_0)^{n-i} > \alpha > \sum_{k=c_0+1}^n \binom{n}{i} p_0^i (1-p_0)^{n-i}.$$

于是在临界值为 k 的情况下，设对应的右尾概率为 p_k ：

$$p_k = 1 - P(X \leq k)$$

利用计算机程序进行遍历，设定初始最小样本量 n 为 1，寻找最小的 k ，逐步增大 n 的值，寻找不同的 k 值，结果如图¹²：

表 1 问题一情形一对应抽取样本量 n 、临界值 k 和对应右尾概率 p_k 值

抽取样本量 n	2	3	4	5	6	7
临界值 k	2	2	3	3	3	3
对应右尾概率 p_k 值	0.010000	0.028000	0.003700	0.008560	0.015850	0.025691
抽取样本量 n	8	9	10	11	12	13
临界值 k	3	4	4	4	4	4
对应右尾概率 p_k 值	0.038092	0.008331	0.012795	0.018535	0.025637	0.034161
抽取样本量 n	14	15	16	17	18	19
临界值 k	4	5	5	5	5	5
对应右尾概率 p_k 值	0.044133	0.012720	0.017004	0.022144	0.028194	0.035194
抽取样本量 n	20	21	22	23	24	25
临界值 k	5	6	6	6	6	6
对应右尾概率 p_k 值	0.043174	0.014445	0.018216	0.022608	0.027658	0.033400
抽取样本量 n	26	27	28	29		
临界值 k	6	6	7	7		
对应右尾概率 p_k 值	0.039859	0.047057	0.017907	0.021617		

对问题一第一种情形结果的分析：当样本量小于 30，将 n 从 1 到 29 逐渐增大，1 找不到符合情况，所以不显示。图中列举出样本量以及对应抽到不合格样品数量的匹配值，当抽出 n 个样本时，只有当 k 值大于表中对应的 k 时，其才能有超过 95% 的概率认为真实的次品率超过标价值。对于企业生产而言，由于只抽一次存在过多偶然性，所以最小抽取次数为 1 意义不大，应当舍弃，在真实次品概率认为较小情况下，企业应当选取 n 和 k 有着较大差异的组合进行 n 值的合理选择。

表 2 问题一第二种情形抽取样本量 n 、临界值 k 和对应右尾概率 p_k 值

抽取样本量 n	1	2	3	4	5	6
临界值 k	1	2	2	2	2	3
对应右尾概率 p_k	0.100000	0.010000	0.028000	0.052300	0.081460	0.015850
抽取样本量 n	7	8	9	10	11	12
临界值 k	3	3	3	3	3	4
对应右尾概率 p_k	0.025691	0.038092	0.052972	0.070191	0.089562	0.025637
抽取样本量 n	13	14	15	16	17	18
临界值 k	4	4	4	4	4	4
对应右尾概率 p_k	0.034161	0.044133	0.055556	0.068406	0.082641	0.098197
抽取样本量 n	19	20	21	22	23	24
临界值 k	5	5	5	5	5	5
对应右尾概率 p_k	0.035194	0.043174	0.052152	0.062134	0.073113	0.085075
抽取样本量 n	25	26	27	28	29	
临界值 k	5	6	6	6	6	
对应右尾概率 p_k	0.097994	0.039859	0.047057	0.055007	0.063717	

对问题一第 2 小问结果分析：表中显示了抽取的样本量和其对应最小抽取次品数量的匹配情况，当实际抽取的 k 值小于表中 k 值时，原假设才被接受，才能有 90% 的概率认为真实次品率小于标价值，接受该批次零件。

5.2.2 大样本正态分布近似模型求解

在正态分布近似条件下的单尾检验中，选取 Z 统计量作为检验统计量，通过查表有³：

表3 置信度 α 与 Z 值对应表

置信度 α	Z 值
0.9	1.282
0.95	1.645

5.2.3 情形一：在 95% 的信度下认定零配件次品率超过标称值，则拒收这批零配件

通过查表可得：

$$Z_{\alpha} \approx 1.645$$

由于 $X \sim B(n, p)$ 可以近似为 $X \sim N(np, np(1-p))$ ，所以：

$$P\left(\frac{X - np}{\sqrt{np(1-p)}} \geq \frac{k - np}{\sqrt{np(1-p)}}\right) = 0.05$$

根据前面推导过程，可以直接使用我们得到的表达式：

$$n = \frac{Z_{\alpha/2} \cdot p(1-p)}{E^2}$$

取 $E=0.05$ ，带入数据求解得到 n 的最小值为：138，利用计算机迭代得到的对应 k 值为 20

我们由此可以得到结论：最小值 n 为 138，当在这个样本量中抽到的次品数目多于 20 时，认为有 95% 的把握认为超过标价值

5.2.4 情形二：在 90% 的信度下认定零配件次品率不超过标称值，则接收这批零配件

- 确定阈值：

$$Z(0.90) \approx 1.282$$

- 计算临界值 k ：

$$P(X \leq k \mid p = 0.10) = 0.90$$

正态近似：

$$P\left(\frac{X - np}{\sqrt{np(1-p)}} \leq \frac{k - np}{\sqrt{np(1-p)}}\right) = 0.90$$

- 根据前面推导过程，可以直接使用我们得到的表达式：

$$n = \frac{Z^2 \cdot p(1-p)}{E^2}$$

取 $E=0.05$ 时，我们找到的满足条件的最小 n 值为：97， k 值为：13

在 n 最小为 97 的情况，只要抽到的次品数量不高于 13，则认为有 90% 的把握认为低于标价值，零件可以接受。

5.3 问题二模型的建立

5.3.1 模型建立前部分观察与分析

在模型建立过程前，我们首先观察给出的表 1，对六种不同情境下的 12 个参数进行对比分析。通过对照六种情况的参数设置，我们得到以下信息²：

情况	零配件 1			零配件 2			成品				不合格成品	
	次品率	购买单价	检测成本	次品率	购买单价	检测成本	次品率	装配成本	检测成本	市场售价	调换损失	拆解费用
1	10%	4	2	10%	18	3	10%	6	3	56	6	5
2	20%	4	2	20%	18	3	20%	6	3	56	6	5
3	10%	4	2	10%	18	3	10%	6	3	56	30	5
4	20%	4	1	20%	18	1	20%	6	2	56	30	5
5	10%	4	8	20%	18	1	10%	6	2	56	10	5
6	5%	4	2	5%	18	3	5%	6	3	56	10	40

图 2 六种情况示意图

通过不同情况对照来看：

- **情况 1 与情况 2 对比：**情况 2 中零配件 1、零配件 2 以及成品的次品率均较情况 1 更高，而其他参数保持不变。
- **情况 1 与情况 3 对比：**在情况 3 中，调换损失显著增加，但其他参数与情况 1 相同。
- **情况 2 与情况 4 对比：**情况 4 的检测成本相比情况 2 有所下降，但调换损失大幅增加，次品率保持一致。
- **情况 3 与情况 4 对比：**情况 4 中的零配件和成品次品率均较情况 3 更高，然而检测费用有所降低。

从决策流程的整体来看：

- **次品率差异：**情况 2 和情况 4 的零配件与成品次品率较高，而情况 6 的次品率显著较低。
- **调换损失：**情况 3 和情况 4 的调换损失较高，均为 30 元，远高于其他情况。
- **检测成本差异：**情况 5 中零配件 1 的检测成本（8 元）远高于其他情况的检测成本。
- **拆解费用差异：**情况 6 中的拆解费用高达 40 元，远高于其他情况。

这些观察为我们后续模型求解提供了便利，可以据此对模型结果进行大致预估，或对模型结果进行检验。例如：我们可以大胆猜测，在情况 2，4 下，应尽可能做更多检测（因为次品率高）；在情况 3，4 中，应尽可能减少不合格品流入市场（因为调换损失很高）。

5.3.2 模型建立

问题二要求给出生产过程的最优化决策方案，根据题目中说明的生产流程，我们可以做出如下生产示意图³：

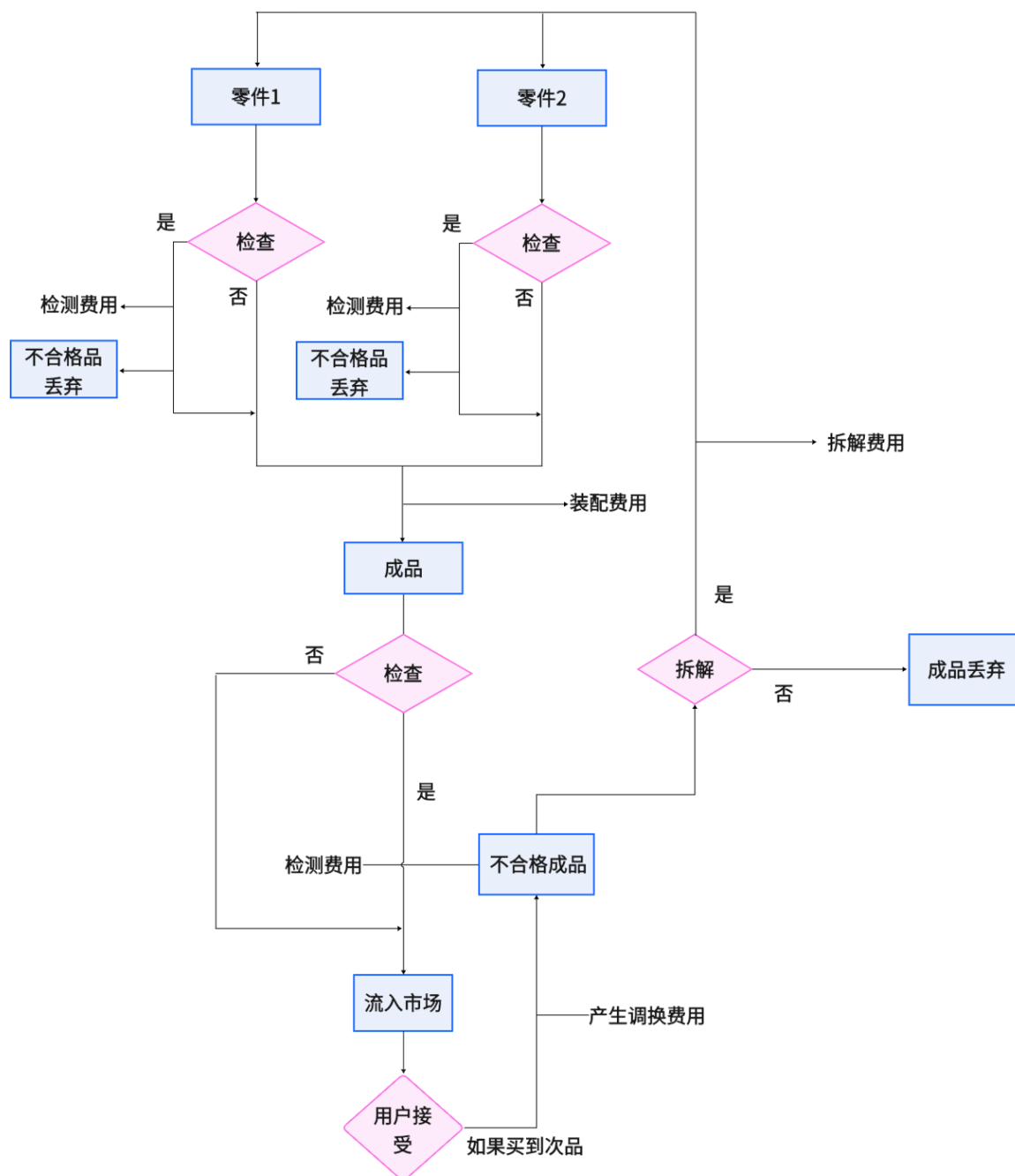


图3 问题二决策流程示意图

下面对流程进行大致解释：流程可以分为以下几个步骤：

1. 零件1与零件2的检测：

如果选择检测零件，系统将对零件进行检查，产生检测费用。检测后如果零件是不

合格的，零件会被丢弃。如果选择不检测，则零件直接进入下一个阶段。

2. 零件装配：

零件 1 和零件 2 通过检测后（或选择不检测），它们将进入装配阶段。这个阶段产生装配费用，装配完成后生成成品。

3. 成品检验（如果不检验，进入步骤 5）：

装配后的成品进入成品检测流程：如果成品检测合格，则可以进入市场流通阶段。如果成品不合格，将进行下一步的处理。

4. 成品拆解与丢弃：

如果成品检测不合格，系统会判断是否对成品进行拆解。如果选择拆解，成品将被拆解为零件，拆解会产生额外的拆解费用。拆解后不合格的零件将会进行生产调整并且可以重新进入流程。如果选择不拆解，成品将被直接丢弃。

5. 流入市场：

通过成品检测的合格成品会流入市场。市场流通的产品还需要进行用户反馈，即用户接受环节，确保最终产品符合用户需求。

6. 用户接受：

如果用户接受了产品，则流程完成。如果用户不接受产品（遇到次品），则需要采取进一步的调换，调换得到的产品进入步骤四。

在多个步骤中都存在检测与不检测的选择，不同选择将影响生产成本和不合格品处理策略。不合格品有两种处理方式：丢弃或者拆解，具体选择将影响后续流程。流程中各个阶段会产生额外的费用，如检测费用、装配费用、拆解费用等。这是一个典型的质量控制流程，企业通过检测和拆解等手段来控制产品的质量，并在生产过程中不断做出决策，以达到最佳的成本效益。通过以上对问题二的细致分析，我们可以发现该决策问题具体包含了以下四个决策变量来决定⁴：

表 4 决策变量表

决策变量的含义	变量	说明
零配件 1 是否检测	d_1	F：不检测，T：检测
零配件 2 是否检测	d_2	F：不检测，T：检测
成品是否检测	d_3	F：不检测，T：检测
不合格品是否拆解	d_4	F：不拆解，T：拆解

从表格中我们可以分析得出，该决策问题实际上是一个遍历求解问题，通过对四个不同的布尔型的决策变量进行排列组合，我们可以设计出 16 种不同的决策方案，对于六种不同的情况而言，我们可以使用计算机程序对每一种决策方案的盈利大小进行计

算，从而比较在每一种情况下，具体是哪一种决策方案可以**获得最大的利润**，即为最优的决策方案（决策目标），从而给企业提供重要的参考。

以下为每一部分具体的公式推导：

1. **成品数量计算**考虑到实际生产过程中的木桶效应，如果零件一和零件二的库存数量不同（指经过检验后，数量可能不同），我们所组装出的成品只能根据两种不同的零件中，数量较少的那个零件来计算：

$$N_p = \min(N_1, N_2)$$

2. 成品合格率计算

成品的合格率取决于零件 1 和零件 2 的合格率以及成品的次品率。其合格率的计算公式为：

$$Q_p = Q_1 \times Q_2 \times (1 - d_p)$$

其中零件一的合格率 Q_1 和零件二的合格率 Q_2 并非固定不变，而是要视在零件检测阶段，有没有对零件一和零件二进行检测来具体判断

3. 合格成品数量计算

合格成品的数量通过成品数量乘以合格率得出：

$$N_q = N_p \times Q_p$$

4. 不合格成品数量计算

不合格成品数量为装配完成的成品数量减去合格成品数量，公式为：

$$N_d = N_p - N_q$$

5. 总收入计算

总收入为销售的合格成品数量乘以市场价格，计算公式为：

$$R = N_q \times P$$

6. 总成本计算

总成本由零件采购成本、检测成本、装配成本、拆解成本和退换损失组成：

$$C = N_1 \times c_1 + N_2 \times c_2 + N_1 \times k_1 + N_2 \times k_2 + N_p \times k_p$$

如果有退换产品，还需要加上调换损失：

$$C_{\text{return}} = N_r \times L_r$$

若选择拆解不合格产品，拆解成本为：

$$C_{\text{dis}} = N_d \times c_{\text{dis}}$$

7. 总利润计算

总利润为总收入减去总成本，计算公式为：

$$\Pi = R - C$$

至此，基于穷举法解决问题二的决策问题的模型建立完毕。

5.4 问题二模型的求解

在程序的设计中，我们采用循环来实现模拟次品被拆解然后再被重新检测，但是考虑到企业的实际生产情况，首次被检为次品的成品和被用户调换回来的次品被拆解之后重新组装为成品，这样的过程不会反复出现，也即：企业在实际生产的过程中不会反复对一个成品进行拆解和组装，所以我们在程序中设置循环数为 2，使得模型更接近于实际情况。

5.4.1 模型求解的结果

在模型求解的过程中，我们默认零件一和零件二采购的数量相同，且均为 1000 件（原因于模型假设处说明），对六种情况，分别遍历 16 种决策，并计算所得的利润。

16 种决策方案所对应的具体的各个决策变量的明细如下图所示（ T 表示是， F 表示否）⁵：

表 5 决策方案明细表

决策方案	零件一是否检测	零件二是否检测	成品是否检测	次品是否拆解
1	T	T	T	T
2	T	T	T	F
3	T	T	F	T
4	T	T	F	F
5	T	F	T	T
6	T	F	T	F
7	T	F	F	T
8	T	F	F	F
9	F	T	T	T
10	F	T	T	F
11	F	T	F	T
12	F	T	F	F
13	F	F	T	T
14	F	F	T	F
15	F	F	F	T
16	F	F	F	F

以下为分别对六种情况而言，16 种不同的决策方案的利润统计图⁴：

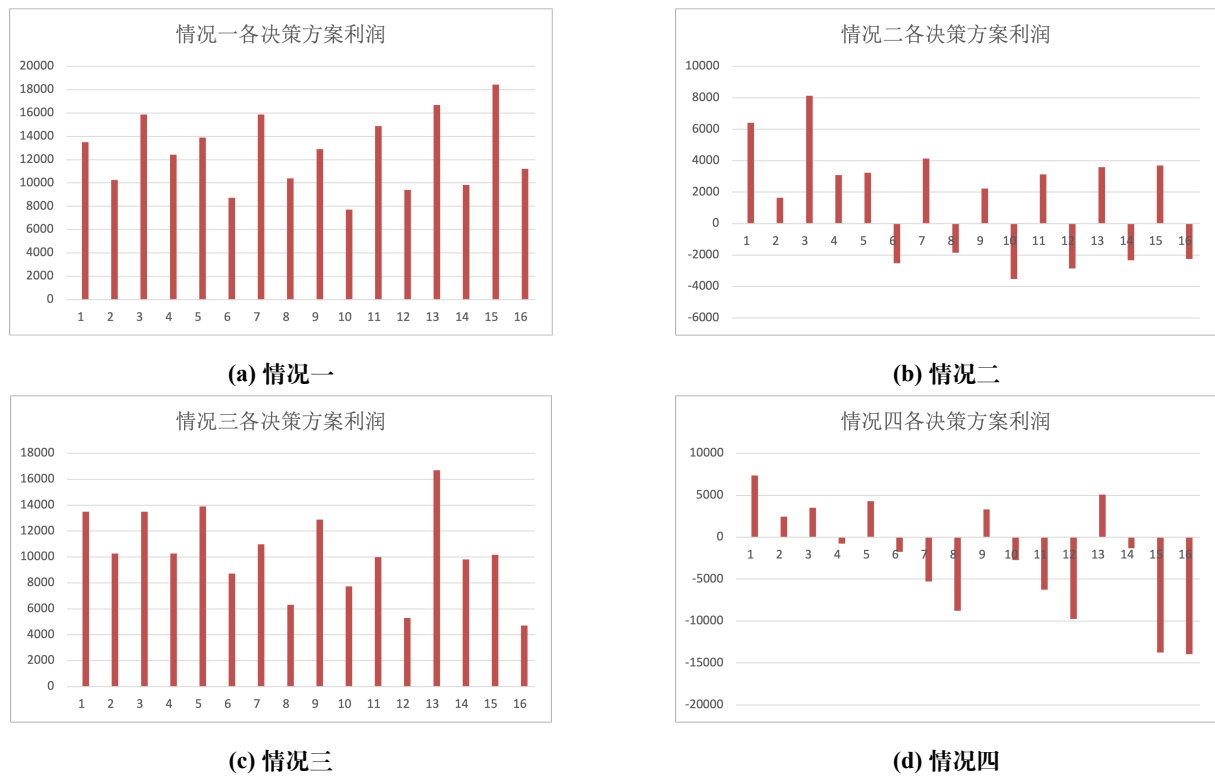
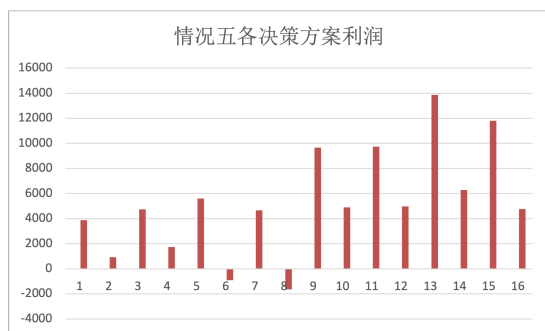
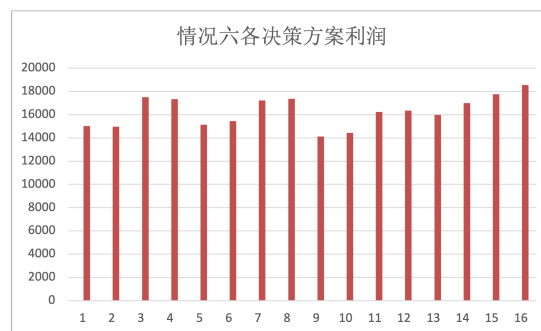


图 4 六种情况下不同决策方案的利润统计图 (第 1 页)



(a) 情况五



(b) 情况六

图 5 六种情况下不同决策方案的利润统计图 (第 2 页)

根据图片中所展示的结果，我们可以得到在假设 1000 个初始零件，六种不同的情况下，我们应该具体采取的决策方案以及最大利润（其余策略与其具体利润见附件）：

- **情况 1:** 零件一二均不检测、成品不检测、次品拆解，最大利润为 18435。
- **情况 2:** 零件一二均检测、成品不检测、次品拆解，最大利润为 8136。
- **情况 3:** 零件一二均不检测、成品检测、次品拆解，最大利润为 16692。
- **情况 4:** 零件一二均检测、成品检测、次品拆解，最大利润为 7368。
- **情况 5:** 零件一二均不检测、成品检测、次品拆解，最大利润为 13860。
- **情况 6:** 零件一二均不检测、成品不检测、次品不拆解，最大利润为 18562。

综上所述，通过对不同决策方案的比较分析，我们可以为每种具体情况推荐最优的生产决策。分析结果表明，最佳策略取决于具体的生产条件和成本结构。在某些情况下，减少检测和拆解可以显著提高利润，而在其他情况下，适当的检测和拆解则是维持质量和优化利润的关键。

例如，结合之前我们对不同情况的特征的分析，我们可以很清楚地看到，情况 2 和 4 相较于其他情况的很突出的特征就是零件的次品率很高，而这往往预示着我们在生产流程中需要对零件一二进行检测，而我们计算得到的结果也符合这个预期；此外，我们还可以发现，情况 6 的拆解费用相较于别的情况来说特别高，因为这也指向不对成品进行拆解这一决策。

这些发现为企业在面对生产过程中的复杂决策提供了有力的支持。

5.5 问题三模型的建立

5.5.1 模型建立前部分观察与分析

在问题三中，产品生产过程涉及多个零配件和工序的组合，但其核心思路与问题二相似。我们需要对不同的检测和拆解方案进行决策分析。为解决该问题，我们采用由特殊到一般的解决思路，首先考虑两道工序、八种零配件的情况⁶，然后扩展至 m 道工序、

n 个零配件的情况。

零配件	次品率	购买单价	检测成本	半成品	次品率	装配成本	检测成本	拆解费用
1	10%	2	1	1	10%	8	4	6
2	10%	8	1	2	10%	8	4	6
3	10%	12	2	3	10%	8	4	6
4	10%	2	1					
5	10%	8	1	成品	10%	8	6	10
6	10%	12	2					
7	10%	8	1		市场售价	调换损失		
8	10%	12	2	成品	200	40		

图 6 各部件明细表

通过观察并分析表 2，我们可以得到以下信息：

1. 零配件 1 和 4 的次品率、购买单价和检测成本完全一致，零配件 2、5 和 7 亦相同，零配件 3、6 和 8 同样一致。
2. 三种半成品的次品率、装配成本、检测成本和拆解费用等完全一致，但半成品 1 和 2 由三个零配件组成，而半成品 3 则由两个零配件组成。

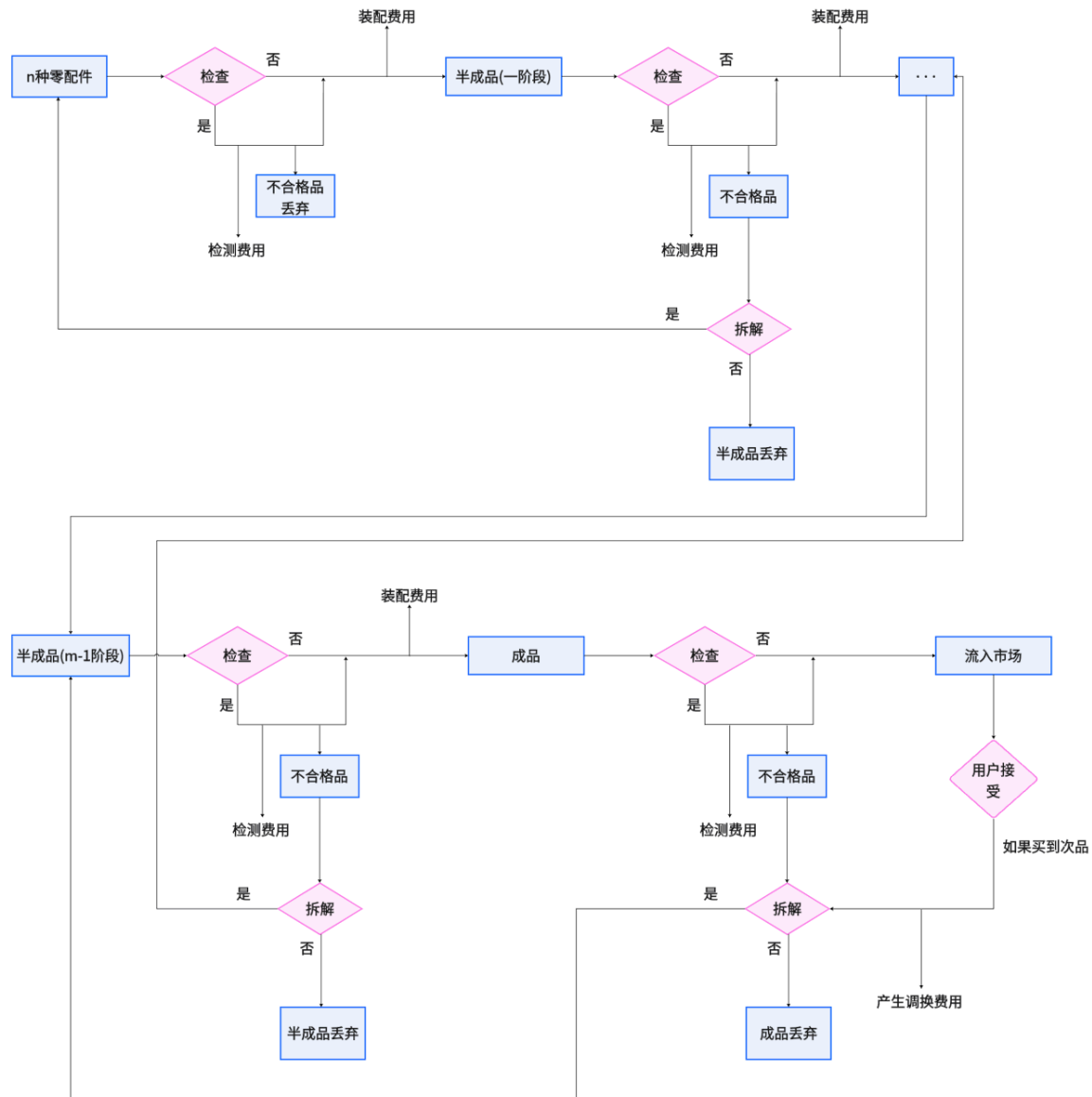
因此，从理论分析的角度来看，我们可以将零件 1 与零件 4 视为完全相同的零件，零件 2 与零件 5、7 视为相同的零件，零件 3 与零件 6、8 同样相同。此外，半成品 1 与半成品 2 也可视为完全相同。这一分析为我们在之后的模型建立与模型验证提供了重要的简化依据。

5.5.2 模型建立

首先，我们考虑八个零配件、两道工序的情况，在这种情况下，生产过程可分为两个阶段。第一道工序是将 8 个零配件组装成 3 个半成品。由于各个零配件具有一定的次品率，因此在此阶段我们需要决定是否对每个零配件进行检验，这涉及 8 个决策。第二道工序是将 3 个半成品组装成最终成品，各半成品也存在一定的次品率（此处的半成品次品率指的是由不合格零件和组装过程共同导致的实际次品率），因此我们需要考虑是否对每个半成品进行检验，这涉及 3 个决策。此外，成品的次品率同样受到不合格半成品和组装过程的影响，因此我们需要决定是否对成品进行检验，这进一步增加了 1 个决策。

此外，对于不合格的半成品和成品（包括通过检验发现的不合格品以及用户退换的成品），我们需要考虑是否对这些不合格产品进行拆解，以回收零配件，这涉及 4 个决

策。因此，在整个生产过程中，我们共需进行 16 个决策，对应 2 的 16 次方，即 65536 种决策。



由于穷举法的计算复杂度呈指数级增长，我们考虑采用模拟退火算法 (*Simulated Annealing, SA*) [4] 来寻找近似最优解。*SA* 算法的核心思想是通过引入“温度”参数控制解的搜索过程，逐步缩小搜索范围，最终收敛到近似最优解。其步骤如下：

或多个决策进行小范围的随机扰动来生成，例如随机改变某个零件是否检测或某个半成品是否拆解的决策。

3. **接受新解**：比较新解与当前解的利润值。

- 如果新解的利润更高，则直接接受该新解，更新当前解。
- 如果新解的利润较低，则以一定概率接受该新解，避免陷入局部最优。接受概率由以下公式给出：

$$P = \exp\left(\frac{\Delta f}{T}\right)$$

其中 Δf 表示新解与当前解的利润差值， T 表示当前温度。当温度较高时，算法更容易接受较差解；随着温度逐渐降低，算法趋向于只接受较优的解。

4. **降低温度**：按照预定的降温策略逐渐降低温度。我们采用指数衰减法：

$$T_{k+1} = \alpha T_k$$

其中 $\alpha \in (0, 1)$ 为降温系数， T_k 表示第 k 次迭代后的温度。

5. **终止条件**：重复生成新解和接受新解的步骤，直到满足终止条件。终止条件可以为温度降至某一较低值，或达到预设的最大迭代次数。最终输出当前的最优解作为生产决策方案。

通过以上步骤，模拟退火算法能够在复杂的 m 道工序、 n 个零件的生产环境中，有效找到接近最优的生产决策方案。该算法具有全局搜索能力，能够避免陷入局部最优，同时通过降温过程逐渐收敛到最优解。

而作为企业，**决策目标（最优解）毫无疑问为：总利润最高**，即尽可能增加收入，减少成本。

在具体生产过程中，部分步骤与公式说明如下：

1. **成品数量计算**

考虑到实际生产过程中的木桶效应，如果各个半成品的库存数量不同（经过检验后数量不同），我们所组装出的成品只能根据数量最少的那个半成品来计算：

$$N_p = \min(N_{\text{semi},1}, N_{\text{semi},2}, \dots, N_{\text{semi},k})$$

2. **半成品数量计算**

半成品由多个零件装配而成，装配数量受限于组成零件的最少库存数量（经过检验后数量不同）：

$$N_{\text{semi},j} = \min(N_1, N_2, \dots, N_m)$$

3. **半成品合格率计算**

假设半成品由若干个零件组装而成，其实际合格率为所有组成零件的合格率乘积，再乘以半成品装配的次品率：

$$Q_{\text{semi}} = \prod_{i \in \text{components}} (1 - d_i) \times (1 - d_{\text{semi}})$$

4. 成品合格率计算

成品的合格率由各个半成品的合格率以及成品装配过程中产生的次品率决定：

$$Q_p = \prod_{i \in \text{semi-products}} Q_{\text{semi},i} \times (1 - d_p)$$

5. 合格成品数量计算

成品的实际合格数量取决于成品的装配数量和成品的合格率：

$$N_q = N_p \times Q_p$$

6. 不合格成品数量计算

不合格成品数量为总成品数量减去合格成品数量：

$$N_d = N_p - N_q$$

7. 总收入计算

总收入由合格成品的销售收入决定：

$$R = N_q \times P$$

8. 总成本计算

总成本包括以下部分：

$$C = C_{\text{purchase}} + C_{\text{inspect}} + C_{\text{assemble}} + C_{\text{disassemble}} + C_{\text{return}}$$

具体展开为：

$$C_{\text{purchase}} = \sum_{i=1}^n (N_0 \times c_i)$$

$$C_{\text{inspect}} = \sum_{i=1}^n (N_0 \times k_i \times I_i)$$

$$C_{\text{assemble}} = N_p \times k_p$$

$$C_{\text{disassemble}} = N_d \times c_{\text{dis}}$$

$$C_{\text{return}} = N_r \times L_r$$

其中：

- C_{purchase} : 采购总成本, 单位为元;
- C_{inspect} : 检测总成本, 单位为元;
- C_{assemble} : 装配总成本, 单位为元;
- $C_{\text{disassemble}}$: 拆解不合格产品的总成本, 单位为元;
- C_{return} : 退换不合格品的总损失, 单位为元;

9. 总利润计算

总利润为总收入减去总成本:

$$\Pi = R - C$$

至此, 基于穷举法解决特殊情况, 模拟退火解决一般情况的模型建立完成。

5.6 问题三模型的求解

按照模型建立部分说明的生产过程, 我们首先采用穷举搜索的办法来解决两道工序, 八个零件的情况, 即对 65536 中决策依次进行遍历, 分别得到其对应的总利润。

每个决策由 16 个布尔值构成, 分别对应于不同的生产决策, 其中:

- 前 8 个布尔值对应零件 1-8 的是否进行检测的决策;
- 第 9-11 个布尔值对应半成品 1-3 是否进行检测的决策;
- 第 12-14 个布尔值对应半成品 1-3 是否进行拆解的决策;
- 第 15 个布尔值对应成品是否进行检测的决策;
- 第 16 个布尔值对应成品是否进行拆解的决策。

下表展示了利润最高的前 10 种决策组合与其对应的利润 (全部决策及其对应利润于附件中给出)⁶:

表 6 利润排名前 10 的决策组合与对应的利润

决策组合 (T/F)	利润 (元)
1.(T, T, T, T, T, T, T, T, T, T, T, T, T, T, F, T)	39232
2.(T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, T)	37422
3.(T, T, T, T, T, T, T, T, T, T, T, T, T, T, F, F)	37050
4.(T, T, F, T, T, F, T, F, T, T, T, T, T, T, F, T)	35680
5.(T, T, T, T, T, T, T, T, T, T, T, T, T, T, T, F)	35304
6.(T, T, F, T, T, F, F, T, T, T, T, T, T, T, F, T)	34680
7.(T, T, F, T, F, T, T, F, T, T, T, T, T, T, F, T)	34680
8.(T, T, F, F, T, T, T, F, T, T, T, T, T, T, F, T)	34680
9.(T, F, T, T, T, F, T, F, T, T, T, T, T, T, F, T)	34680
10.(F, T, T, T, T, F, T, F, T, T, T, T, T, T, F, T)	34680

前五种情况依次为

- 对所有零件和半成品进行检测，对所有半成品进行拆解，对成品不检测但进行拆解。
- 对所有零件和半成品进行检测，并对所有半成品和成品进行检测和拆解。
- 对所有零件和半成品进行检测，对所有半成品进行拆解，对成品既不检测也不拆解。
- 检测零件 3, 6, 8, 对所有半成品检测 and 拆解，不检测成品但进行拆解
- 对所有零件和半成品进行检测，对所有半成品进行拆解，对成品检测但不进行拆解。

剩下五种情况均为检测部分零件，对所有半成品检测 and 拆解，不检测成品但进行拆解。

对于 m 道工序、 n 个零件的生产环境，因为不确定几个零件合成一个一阶段半成品，几个一阶段半成品合成二阶段半成品等等，我们没有办法在给定 m, n 的情况下确定一个最优解，因此我们尝试对给定的决两道工序，八个零件的情况进行验证，结果发现，最优情况为 1. 对所有零件和半成品进行检测，对所有半成品进行拆解，对成品不检测但进行拆解。或 2. 对所有零件和半成品进行检测，并对所有半成品和成品进行检测 and 拆解。这验证了我们上述穷举的正确性，同时也说明了模拟退火算法的可行性。

总结：通过对问题三的分析与求解，我们构建了一个较为复杂的生产决策模型，该模型涵盖了多道工序和多零配件的决策过程。我们首先采用穷举法对两道工序、八个零配件的情况进行了全面的决策搜索，并得出了最佳的决策组合。在这种情况下，**最优方案为对所有零件和半成品进行检测与拆解，而对于成品，则只需拆解而无需检测。**

当我们扩展到 m 道工序、 n 个零件的更大规模问题时，由于问题复杂度的显著提升，穷举法难以在合理的时间内找到最优解。因此，我们引入了模拟退火算法 (SA) 作为求解手段。通过对生产流程的模拟和模型的测试，证明了该方法的有效性和可行性。本模型通过对检测和拆解过程的合理决策，最大化了生产过程中的利润，同时通过穷举和模拟退火两种算法的结合，为复杂生产环境下的决策提供了科学的依据。

5.7 问题四模型的建立

在问题四中，我们面临的挑战是如何基于已知的理论次品率，通过模拟抽样检测得到实际次品率，并在此基础上优化企业的生产决策。

在问题 2 和问题 3 中，次品率是一个给定的理论值，代表供应商提供的零配件或生产过程中零件和成品的理论缺陷率。然而，在实际生产环境中，通过抽样检测得到的次品率通常会与理论值有所偏差，概率往往在一定范围波动 [5]。为模拟这种差异，我们通过抽样检测方法从给定的理论次品率中生成实际的次品率，用以进一步指导生产决策。

5.7.1 样品生成和次品率模拟：

1. **使用理论次品率生成样本：**首先，我们根据已知的理论次品率，模拟生产过程中实际可能检测到的次品数量。这个过程模拟了抽样检测，即在总产品数量中抽取一个固定数量的样本。
2. **计算样本中的次品率：**对于每个被抽取的样本，检测其是否为次品。通过对检测结果进行统计，计算样本中的次品率。这个样本次品率不仅依赖于理论次品率，同时考虑到了抽样过程中可能产生的随机波动。
3. **加入随机性以模拟实际检测波动：**为了使模拟更贴近实际，在计算样本次品率时，我们加入了一个小的随机扰动 [6]。这一随机扰动模拟了检测过程中可能存在的误差或环境影响，使得实际检测到的次品率不会完全等于理论次品率。
4. **计算标准误差：**根据抽样所得的样本次品率，我们计算出标准误差 [7]。标准误差反映了由于样本量有限导致的次品率估计的不确定性。标准误差越大，表示估计的不确定性越强。它也体现了检测过程中，由于样本数量的不同，对次品率估计结果的可靠性差异。
5. **置信区间的计算：**基于标准误差，进一步计算 95% 的置信区间。该置信区间代表了在一定置信水平下，实际次品率可能位于的范围。这个置信区间为后续生产决策提供了一个可靠的次品率参考范围。
6. **输出模拟的实际次品率：**最终，函数生成一个模拟的实际次品率，并给出置信区间作为参考。这一结果模拟了现实中通过抽样检测可能获得的次品率，从而更好地反映了生产中的实际情况。

5.7.2 模拟结果分析

问题四的模型是在问题二和问题三模型的基础上增加抽样检测模块得到的，下图是我们使用真实次品率模拟函数模块，并大量取样之后得到的样本次品率（百分之十）的分布图⁸：

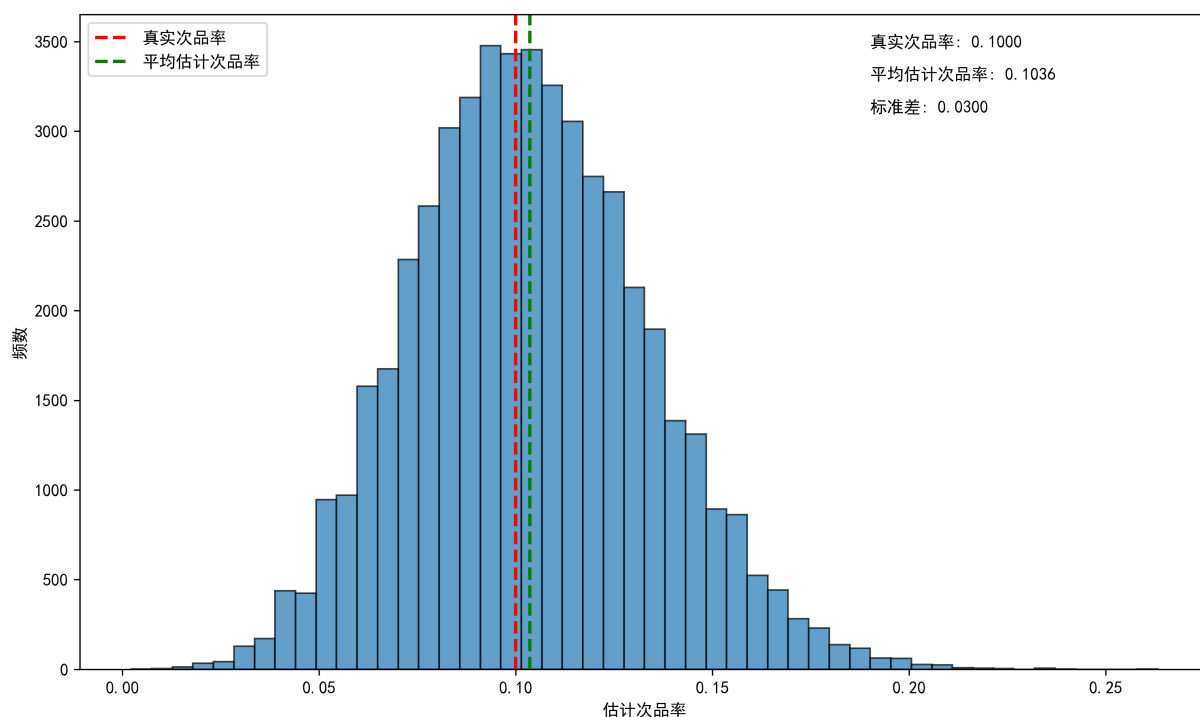


图8 样本次品率分布图

我们可以看到随着样本数量的增加，样本次品率的分布逐渐呈现出正态分布。在对样本进行抽样检测的过程中，随着抽样数量的增加，我们计算得到的样本次品率会更加接近于真实次品率。这也与企业生产当中的实际情况相符，证明了我们真实次品率模拟过程的合理性。

我们将根据这一模拟函数重新计算问题二和问题三，对零件，半成品，成品的次品率均作处理，生成类似的实际次品率，来考察先前的最优决策方案在次品率有波动的情况下是否仍然能保持高效。

5.8 问题四模型的求解

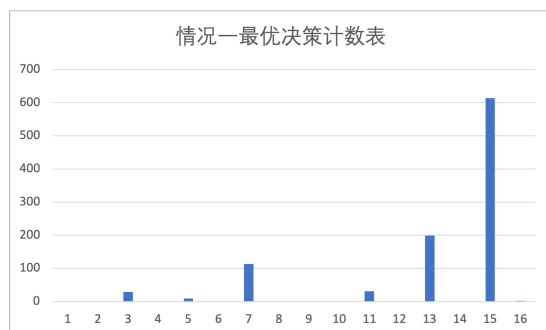
5.8.1 问题二决策方案的修正

在对各次品率做模拟真实次品率后，并使用该次品率生成大量样本后，重新计算问题二可以得到各情况的结果（下面仅对情况1结果⁷进行说明，其他情况以图表形式给出）¹⁰：

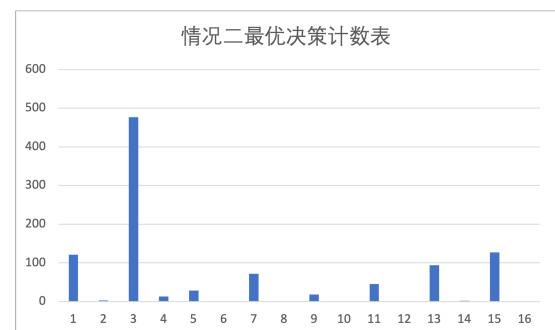
表 7 1000 个样品，最优决策方案的分布

决策组合 (T/F)	利润 (元)
(True, True, True, True)	0
(True, True, True, False)	0
(True, True, False, True)	31
(True, True, False, False)	2
(True, False, True, True)	7
(True, False, True, False)	0
(True, False, False, True)	87
(True, False, False, False)	1
(False, True, True, True)	3
(False, True, True, False)	0
(False, True, False, True)	42
(False, True, False, False)	0
(False, False, True, True)	167
(False, False, True, False)	1
(False, False, False, True)	654
(False, False, False, False)	5

观察表格，我们可以发现，在 1000 个样品情况下（具有真实次品率分布），最常见的最佳决策组合为不检测零件一，不检测零件二，不检测成品，对次品进行拆解。这与我们在问题二中得到的最优决策组合相一致。另外我们可以发现，也有其他决策组合出现，例如不检测零件一，不检测零件二，检测成品，对次品进行拆解。这可能对应实际次品率偏高的情况，需要进行更多检测。



(a) 情况一



(b) 情况二

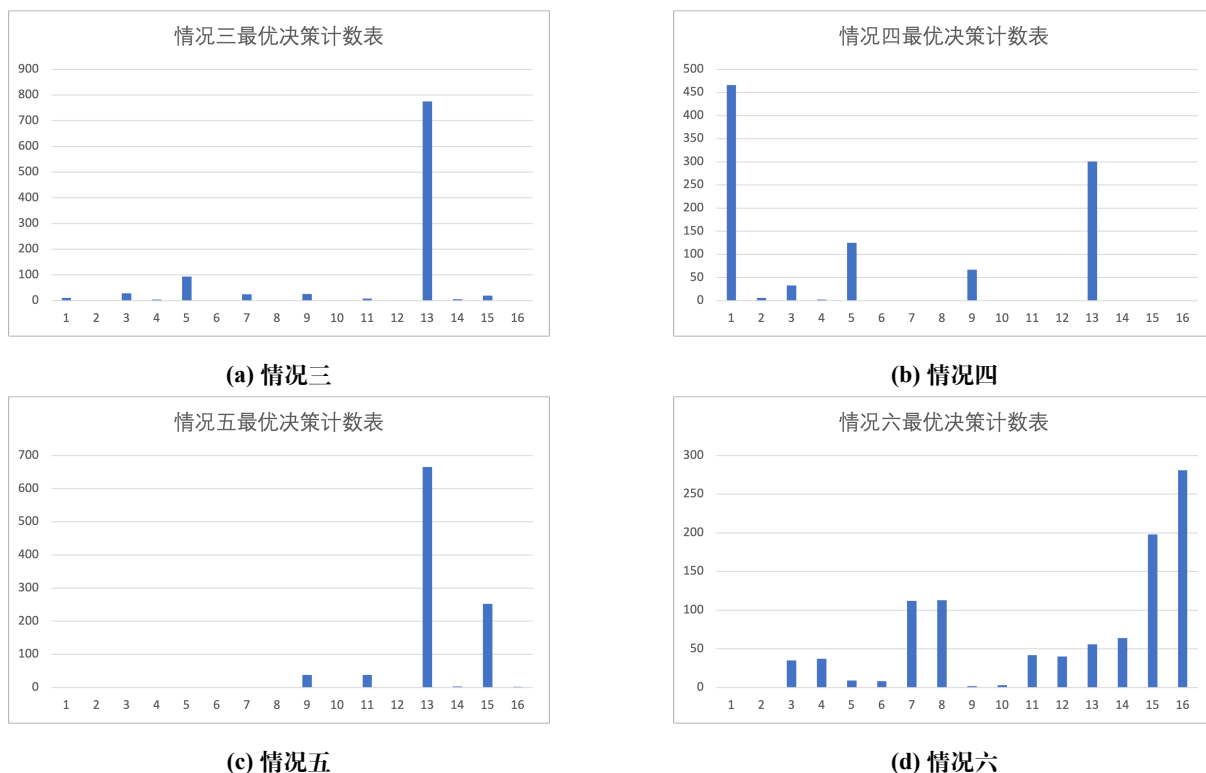


图 10 修改后问题二六种情况下各决策方案为最优方案的计数统计图

分析以上结果发现，在问题二中我们在六种情况下得到的最优决策方案，在次品率存在波动的情况下，依然能够以较大概率成为最优的决策方案⁸，这充分证明了我们得出的最优决策方案具有很好的鲁棒性。

表 8 修改后六种不同情况下较优化决策方案表

情况	最优方案	次优方案
1	零件一二不检测、成品不检测、次品拆解	无
2	零件一二检测、成品不检测、次品拆解	无
3	零件一二不检测、成品检测、次品拆解	无
4	零件一二检测、成品检测、次品拆解	零件一二不检测、成品检测、次品拆解
5	零件一二不检测、成品检测、次品拆解	无
6	零件一二不检测、成品不检测、次品不拆解	零件一二不检测、成品不检测、次品拆解

5.8.2 问题三决策方案的修正

在重新计算问题三的过程中，我们需要对模型进行适当的修改，问题三与问题二不同的点在于，问题二仅仅包括 4 个决策变量，而问题三包含了多达 16 个决策变量，如果简单迁移模型，毫无疑问会遇到维度灾难，模型的计算成本过于庞大。

因此，我们考虑到当样本量较大的时候，样本的次品率会很接近于真实次品率，从而我们不妨认为次品率的波动是较为轻微的，因此我们考虑将模型建立在问题三求解的结论之上，也即：不再遍历所有可能的决策方案，而是选取在问题三计算的结论中，盈利排名前 100 的一百种不同的决策方案，使得计算成本大大降低。

我们计算得到如下结果¹¹：



图 11 问题三各个排名决策方案为最优策略计数统计图

从图中我们可以看到，原问题三当中计算得到的最优决策方案（6 号）依然为最优的决策方案，但考虑到图中的数据相对而言较为分散，因此 3 号、4 号和 8 号也是可以接受的次优决策方案⁹。

表 9 修改后问题三较优化决策方案表

决策方案	具体决策细节
6	零件均检测、半成品均检测、半成品均拆解、成品不检测、成品拆解
3	零件均检测、半成品均检测、半成品均拆解、成品不检测、成品不拆解
4	零件均检测、半成品均检测、半成品均拆解、成品检测、成品拆解
8	零件均检测、半成品均检测、半成品均拆解、成品检测、成品不拆解

5.8.3 总结

通过这种抽样检测模拟，我们从给定的理论次品率出发，生成了一个更接近实际检测过程的次品率。这一过程模拟了在生产环境中，抽样检测所带来的结果波动。利用这

一方法，我们可以在问题 2 和问题 3 的基础上，更加贴近真实情况地完成零件、半成品和成品的次品率估计，并为生产过程中是否检测、是否拆解等决策提供可靠依据。

六、模型的分析 and 检验

6.1 灵敏度分析

灵敏度分析是用于评估模型输出对输入变量变化的敏感程度的技术，主要考察模型对输入变量的不确定性如何反应。

而在问题四中，我们将确定的次品率替换成为了抽样检测获得的次品率，并通过随机取样得到了正态分布的样品次品率曲线。我们将这些样品次品率带入模型进行一一计算，从而得出最优策略，并与问题二、问题三中所得的计算结果进行比较，发现在问题二、问题三中的最优决策方案在问题四的模型中依然全部都是最优的决策方案，反映出模型具有较好的稳定性。

6.2 误差分析

在问题二、问题三的模型计算当中，我们计算半成品和成品的良品率，采用的是子部件的所有良品率和当前物件的良品率的乘积的方法。这一方法在第一次循环的时候符合事实情况，但在第二轮循环的时候，考虑许多正品已经在第一轮的时候被售出，实际上被拆解的部件中次品的比例大幅度上升，这时候我们再采用原先的方法计算次品率，会比实际情况低一些。

但考虑到完全修正每一个循环当中各部件的次品率会大幅度增加模型的复杂程度，并且我们所建立的模型只进行两次循环，所以这一误差对模型的计算结果影响较小，因此我们依然选择在这一点上对模型进行简化。

七、模型的评价与改进

7.1 模型的优点

1. 灵活性和适应性

该模型通过抽样检测模拟生产环境中的次品率波动，能够更接近实际情况。这种方法提高了质量控制决策的精准度，使得企业能够根据实际次品率做出更加合理的生产决策。

2. 简化计算以便实际应用

模型通过简化计算，例如将生产周期限制为两次，降低了计算复杂度，同时保持了预测的准确性。这种简化使得模型在工业应用中更加实用，而不会产生过高的计算成本。

3. 全面的决策支持

模型综合考虑了零件、半成品、成品的检测、拆解等决策，为企业提供了全方位的质量控制优化方案，确保生产流程的高效和利润最大化。

7.2 模型的缺点

1. 假设过于理想化

模型中某些假设（例如固定的采购量和一致的生产流程）可能与实际生产情况不符，尤其是在更复杂的生产线或多变的市场条件下。

2. 模型复杂度

尽管模型使用了简化方法，但在问题三中扩展到多道工序和多个零件时，计算复杂度仍然较高。如果进一步增加变量或决策点，可能会导致计算时间和资源的显著增加。

7.3 模型的改进与推广

1. 复杂工序的优化

当生产流程涉及多个工序、多种零配件时，模型的复杂度显著提升，可能导致计算效率低下，特别是在面对更多决策变量组合时，如成百上千的可能组合。可以采用启发式算法（如遗传算法、粒子群算法等）或机器学习中的强化学习方法，来有效减少模型的计算复杂度，快速找到最优或近似最优解。

2. 处理次品率的非线性波动

模型假设次品率的波动较轻微，且根据抽样检测得到的次品率来进行决策。然而在实际生产中，次品率可能会由于多种因素（如供应商质量、生产工艺等）发生较大的波动。可以采用更复杂的统计模型，如蒙特卡洛模拟或不确定性量化技术，来对次品率的波动进行建模。

3. 改进对返工和拆解的决策

模型目前对拆解后的零件简单的跳过零件检测步骤，但在实际过程中，可能考虑在第二次循环的时候对先前没有检测过的零件进行检测，而不是始终保持决策不变。

参考文献

- [1] 王维, 向瀚淋, 龚雯丽, 等. 常见分布中心极限定理适用样本量研究 [J]. 高师理科学刊, 2021, 41(07): 20-25.
- [2] 王军虎. 基于假设检验的区间估计必要样本容量确定 [J]. 统计与决策, 2023, 39(21): 29-33.

- [3] . 茆诗松, 程依明, 濮晓龙. 概率论与数理统计 [M]. 2 版. 北京: 高等教育出版社, 2011: 136.
- [4] 王浩博, 刘建涛. 基于改进模拟退火算法的防空武器集群打击目标动态分配 [J]. 计算机测量与控制, 2024, 32(07): 196-202.
- [5] 李建新, 胡刚. 基于模糊概率的股价波动分析模型 [J]. 系统工程理论与实践, 2006, (09): 33-42.
- [6] 孙喜滨, 杨国辉, 吴兆仁. 基于过程模拟与优化的化工工艺改进方法研究 [J]. 辽宁化工, 2024, 53 (07): 1126-1129.
- [7] [1] 宁敏东. 标准误差的整体认识及运用 [J]. 国防科技大学学报, 1994, (03): 131-134.

附录 A 支撑材料文件清单

1.1 数据文件清单

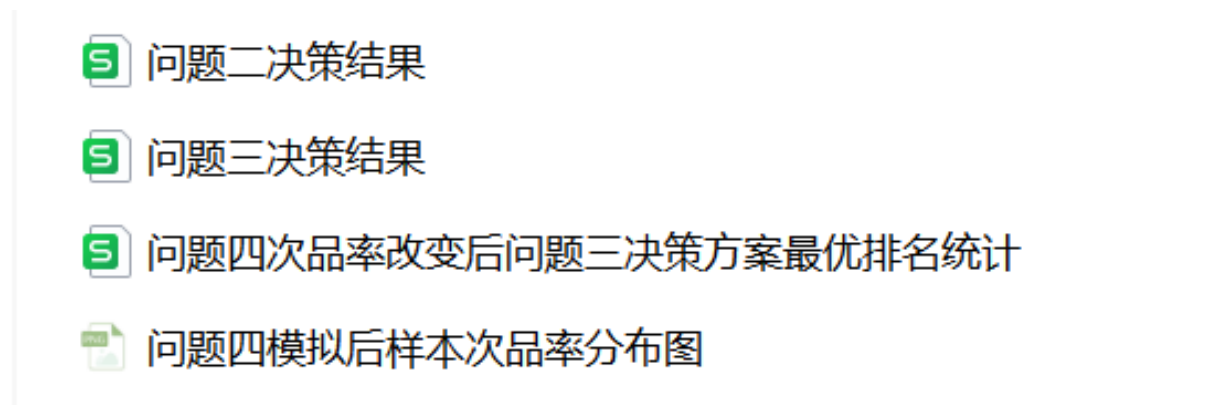




- 
-  问题二决策结果
 -  问题三决策结果
 -  问题四次品率改变后问题三决策方案最优排名统计
 -  问题四模拟后样本次品率分布图

图 12 图片说明

1.2 代码文件清单

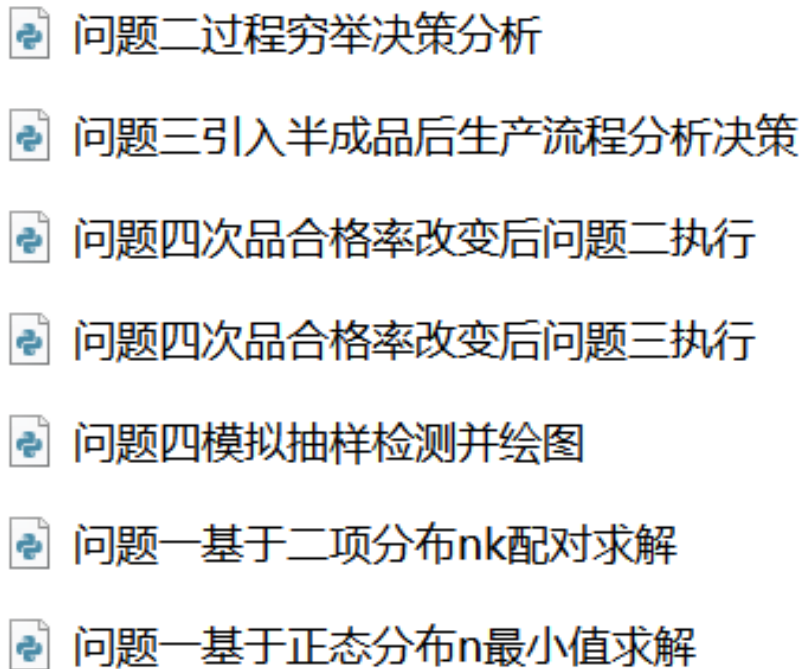


图 13 图片说明

附录 B 代码

2.1 问题一代码

2.1.1 二项分布模型求样本量

```
1 import scipy.stats as stats
2
3 # 定义标称次品率 p0 和显著性水平 alpha
4 p0 = 0.1
5 alpha = 0.05
6
7
8 # 计算从 n = 1 到 n = 29 的样本数及临界值
9 def find_critical_values_binomial(p0, alpha, max_n=29):
10     results = []
11     for n in range(1, max_n + 1):
12         for k in range(n + 1):
13             # 计算二项分布的右尾概率
14             p_value = 1 - stats.binom.cdf(k - 1, n, p0)
15             if p_value < alpha:
16                 results.append((n, k, p_value)) # 存储 n, k 和 p 值
17                 break # 找到第一个符合条件的 k 就跳出循环
```

```

18     return results
19
20
21 # 计算 n 从 1 到 29 的临界值 k 和 p 值
22 results = find_critical_values_binomial(p0, alpha)
23
24 # 输出结果
25 for n, k, p_value in results:
26     print(f"样本量 n = {n}, 临界值 k = {k}, 对应 p 值 = {p_value:.6f}")

```

2.1.2 正态分布模型求样本量

```

1  import math
2  from scipy import stats
3
4
5  def calculate_sample_size(p, confidence, margin_of_error):
6      z = stats.norm.ppf((1 + confidence) / 2)
7      return math.ceil((z**2 * p * (1-p)) / (margin_of_error**2))
8
9
10 def calculate_rejection_threshold(n, p, confidence):
11     return math.ceil(stats.binom.ppf(confidence, n, p))
12
13
14 def calculate_acceptance_threshold(n, p, confidence):
15     return math.floor(stats.binom.isf(1 - confidence, n, p))
16
17
18 def find_optimal_sample_size(p, confidence):
19     n = calculate_sample_size(p, confidence, 0.05) # 使用5%的误差范围作为起始点
20     threshold = calculate_rejection_threshold(n, p, confidence)
21     return n, threshold
22
23
24 # 参数设置
25 p = 0.10 # 标称次品率
26
27 # 情况1: 95%的信度下认定零配件次品率超过标称值
28 print("情况1: 95%的信度下认定零配件次品率超过标称值")
29 n1, threshold1 = find_optimal_sample_size(p, 0.95)
30 print(f"抽样数量: {n1}")
31 print(f"拒收阈值: 如果不合格品数量 >= {threshold1}, 则拒收")
32 alpha1 = 1 - stats.binom.cdf(threshold1 - 1, n1, p)
33 print(f"第一类错误概率 (误拒概率): {alpha1:.4f}")

```

```

34
35 print("\n" + "="*50 + "\n")
36
37 # 情况2: 90%的信度下认定零配件次品率不超过标称值
38 print("情况2: 90%的信度下认定零配件次品率不超过标称值")
39 n2, threshold2 = find_optimal_sample_size(p, 0.90)
40 acceptance_threshold2 = calculate_acceptance_threshold(n2, p, 0.90)
41 print(f"抽样数量: {n2}")
42 print(f"接收阈值: 如果不合格品数量 <= {acceptance_threshold2}, 则接收")
43 beta2 = stats.binom.cdf(acceptance_threshold2, n2, p)
44 print(f"第二类错误概率 (误收概率): {beta2:.4f}")

```

2.2 问题二代码

```

1 import itertools
2 import pandas as pd
3
4
5 # 改进的生产决策过程函数
6 def improved_production_decision_process(
7     initial_quantity,
8     defect_rate_1, purchase_cost_1, inspection_cost_1,
9     defect_rate_2, purchase_cost_2, inspection_cost_2,
10    defect_rate_product, assembly_cost, inspection_cost_product,
11    market_price, return_loss, disassembly_cost,
12    inspect_component_1=True, inspect_component_2=True, inspect_product=True,
13    disassemble_defective=True, max_cycles=2
14 ):
15     total_revenue = 0
16     total_cost = 0
17
18     # 初始购买成本
19     total_cost += initial_quantity * (purchase_cost_1 + purchase_cost_2)
20
21     # 初始零件检测 - 只在开始时进行一次
22     if inspect_component_1:
23         inventory_1 = int(initial_quantity * (1 - defect_rate_1)) # 检测零件1, 使用合格品
24         total_cost += initial_quantity * inspection_cost_1
25     else:
26         inventory_1 = initial_quantity # 不检测, 使用全部零件1, 包括不合格品
27
28     if inspect_component_2:
29         inventory_2 = int(initial_quantity * (1 - defect_rate_2)) # 检测零件2, 使用合格品
30         total_cost += initial_quantity * inspection_cost_2
31     else:

```

```

32     inventory_2 = initial_quantity # 不检测, 使用全部零件2, 包括不合格品
33
34 for cycle in range(max_cycles):
35     # 计算当前批次的零件合格率
36     part1_quality = 1 - defect_rate_1 if not inspect_component_1 else 1.0
37     part2_quality = 1 - defect_rate_2 if not inspect_component_2 else 1.0
38
39     # 检查库存是否足够装配
40     if inventory_1 == 0 or inventory_2 == 0:
41         break # 退出循环
42
43     # 组装过程, 成品合格率 = 零件1合格率 * 零件2合格率 * (1 - 成品次品率)
44     assembled_products = min(inventory_1, inventory_2)
45
46     # 装配成品
47     inventory_1 -= assembled_products
48     inventory_2 -= assembled_products
49     total_cost += assembled_products * assembly_cost
50
51     # 成品合格率的计算
52     product_quality = part1_quality * part2_quality * (1 - defect_rate_product)
53
54     # 成品检测
55     if inspect_product:
56         qualified_products = int(assembled_products * product_quality)
57         defective_products = assembled_products - qualified_products
58         total_cost += assembled_products * inspection_cost_product
59         returned_products = 0 # 无退换
60     else:
61         qualified_products = int(assembled_products * product_quality)
62         defective_products = assembled_products - qualified_products
63         returned_products = defective_products
64         # 销售的次品导致退货.
65         total_cost += returned_products * return_loss # 调换损失
66
67     # 销售收入
68     total_revenue += qualified_products * market_price
69
70     # 处理不合格品
71     total_defective = defective_products
72     if total_defective > 0:
73         if disassemble_defective:
74             total_cost += total_defective * disassembly_cost
75             inventory_1 += total_defective # 拆解后的次品重新进入生产流程
76             inventory_2 += total_defective
77         else:
78             # 如果不拆解, 次品直接丢弃

```

```

79         total_defective = 0
80
81     # 返回总利润
82     profit = total_revenue - total_cost
83     return profit, total_revenue, total_cost
84
85
86 # 优化决策函数，并将结果保存为DataFrame
87 def optimize_decisions(params):
88     decision_profits = []
89     best_profit = float('-inf')
90     best_decisions = None
91
92     for decisions in itertools.product([True, False], repeat=4):
93         inspect_1, inspect_2, inspect_prod, disassemble = decisions
94         profit, revenue, cost = improved_production_decision_process(
95             inspect_component_1=inspect_1,
96             inspect_component_2=inspect_2,
97             inspect_product=inspect_prod,
98             disassemble_defective=disassemble,
99             **params
100         )
101         decision_profits.append({
102             "Inspect Component 1": inspect_1,
103             "Inspect Component 2": inspect_2,
104             "Inspect Product": inspect_prod,
105             "Disassemble Defective": disassemble,
106             "Profit": profit,
107             "Revenue": revenue,
108             "Cost": cost
109         })
110
111     # 找到利润最高的决策
112     if profit > best_profit:
113         best_profit = profit
114         best_decisions = decisions
115
116     return pd.DataFrame(decision_profits), best_profit, best_decisions
117
118
119 # 主函数，处理多个案例并将结果保存到Excel
120 def main():
121     # 基础参数
122     params = {
123         'initial_quantity': 1000,
124         'assembly_cost': 6, 'market_price': 56
125     }

```

```

126
127 # 分析表格中的所有情况
128 table_cases = [
129     {'defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
130      'defect_rate_2': 0.10, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
131      'defect_rate_product': 0.10, 'return_loss': 6, 'disassembly_cost': 5,
132      'inspection_cost_product': 3},
133
134     {'defect_rate_1': 0.20, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
135      'defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
136      'defect_rate_product': 0.20, 'return_loss': 6, 'disassembly_cost': 5,
137      'inspection_cost_product': 3},
138
139     {'defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
140      'defect_rate_2': 0.10, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
141      'defect_rate_product': 0.10, 'return_loss': 30, 'disassembly_cost': 5,
142      'inspection_cost_product': 3},
143
144     {'defect_rate_1': 0.20, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
145      'defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
146      'defect_rate_product': 0.20, 'return_loss': 30, 'disassembly_cost': 5,
147      'inspection_cost_product': 2},
148
149     {'defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 8,
150      'defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
151      'defect_rate_product': 0.10, 'return_loss': 10, 'disassembly_cost': 5,
152      'inspection_cost_product': 2},
153
154     {'defect_rate_1': 0.05, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
155      'defect_rate_2': 0.05, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
156      'defect_rate_product': 0.05, 'return_loss': 10, 'disassembly_cost': 40,
157      'inspection_cost_product': 3}
158 ]
159
160 writer = pd.ExcelWriter("decision_results.xlsx", engine='xlsxwriter')
161
162 for i, case in enumerate(table_cases, 1):
163     case_params = params.copy()
164     case_params.update(case)
165     print(f"\nCase {i}:")
166     decision_df, best_profit, best_decisions = optimize_decisions(case_params)
167     print(f"Best profit: {best_profit:.2f}")
168     print(f"Best decisions: Inspect Component 1: {best_decisions[0]}, "
169           f"Inspect Component 2: {best_decisions[1]}, "
170           f"Inspect Product: {best_decisions[2]}, "
171           f"Disassemble Defective: {best_decisions[3]}")

```

```

167     # 将每个案例的结果保存到Excel中
168     decision_df.to_excel(writer, sheet_name=f"Case_{i}")
169
170     writer._save()
171     print("Results saved to 'decision_results.xlsx'.")
172
173
174 if __name__ == "__main__":
175     main()

```

2.3 问题三代码

```

1  import itertools
2  import pandas as pd
3
4
5  def multi_stage_production_decision_process(
6      initial_quantity,
7      component_params,
8      semi_product_params,
9      final_product_params,
10     max_cycles=3
11 ):
12     total_revenue = 0
13     total_cost = 0
14
15     # 零件初始购买成本
16     for comp in component_params:
17         total_cost += initial_quantity * comp['purchase_cost']
18
19     # 零件检测和初始库存计算
20     inventories = []
21     for comp in component_params:
22         if comp['inspect']:
23             inventory = int(initial_quantity * (1 - comp['defect_rate']))
24             total_cost += initial_quantity * comp['inspection_cost']
25         else:
26             inventory = initial_quantity
27         inventories.append(inventory)
28
29     # 半成品库存
30     semi_product_inventories = [0] * len(semi_product_params)
31
32     for cycle in range(max_cycles):
33         # 半成品的装配与检测

```



```

34 semi_products = []
35 for idx, semi_prod in enumerate(semi_product_params):
36     assembled = min([inventories[i-1] for i in semi_prod['components']])
37     assembled += semi_product_inventories[idx] # 加上之前的半成品库存
38     semi_product_inventories[idx] = 0 # 清空半成品库存
39
40     for i in semi_prod['components']:
41         inventories[i-1] -= min(assembled, inventories[i-1])
42     total_cost += assembled * semi_prod['assembly_cost']
43
44     # 计算半成品的实际合格率
45     semi_product_quality = 1.0
46     for i in semi_prod['components']:
47         if not component_params[i-1]['inspect']:
48             semi_product_quality *= (1 - component_params[i-1]['defect_rate'])
49     semi_product_quality *= (1 - semi_prod['defect_rate']) # 考虑装配过程的次品率
50
51     actual_qualified = int(assembled * semi_product_quality)
52     if semi_prod['inspect']:
53         qualified = actual_qualified
54         defective = assembled - qualified
55         total_cost += assembled * semi_prod['inspection_cost']
56         if semi_prod['disassemble']:
57             total_cost += defective * semi_prod['disassembly_cost']
58         for i in semi_prod['components']:
59             inventories[i-1] += defective
60     else:
61         qualified = assembled # 不检验时, 所有产品都进入下一阶段, 包括不合格品
62
63     semi_products.append((qualified, actual_qualified))
64
65     # 成品的装配与检测
66     final_assembled = min([sp[0] for sp in semi_products])
67     total_cost += final_assembled * final_product_params['assembly_cost']
68
69     # 计算成品的实际合格率
70     final_quality = 1.0
71
72     # 考虑每个半成品的实际合格率, 而不是仅考虑是否检测
73     for idx, sp in enumerate(semi_products):
74         qualified, actual_qualified = sp
75         semi_product_quality = actual_qualified / qualified if qualified > 0 else 0
76         final_quality *= semi_product_quality
77
78     # 再考虑成品装配过程的次品率
79     final_quality *= (1 - final_product_params['defect_rate'])
80

```

```

81     actual_qualified_products = int(final_assembled * final_quality)
82
83     if final_product_params['inspect']:
84         qualified_products = actual_qualified_products
85         defective_products = final_assembled - qualified_products
86         total_cost += final_assembled * final_product_params['inspection_cost']
87         returned_products = 0
88     else:
89         qualified_products = actual_qualified_products
90         defective_products = final_assembled - qualified_products
91         returned_products = defective_products# 实际不合格品最终会被退回
92         total_cost += returned_products * final_product_params['return_loss']
93
94     # 销售收入
95     total_revenue += qualified_products * final_product_params['market_price']
96
97     # 处理不合格品
98     total_defective = defective_products
99     if total_defective > 0 and final_product_params['disassemble']:
100         total_cost += total_defective * final_product_params['disassembly_cost']
101         # 将拆解的成品均匀分配到各个半成品库存中
102         for i in range(len(semi_product_params)):
103             semi_product_inventories[i] += total_defective // len(semi_product_params)
104
105     # 返回总利润
106     profit = total_revenue - total_cost
107     return profit, total_revenue, total_cost
108
109 def optimize_multi_stage_decisions(params):
110     decision_data = []
111
112     for decisions in itertools.product([True, False], repeat=16): # 8个零件 + 3个半成品检测 +
113         # 3个半成品拆解 + 1个成品检测 + 1个拆解决策
114         for i, decision in enumerate(decisions[:8]):
115             params['component_params'][i]['inspect'] = decision
116         for i, decision in enumerate(decisions[8:11]):
117             params['semi_product_params'][i]['inspect'] = decision
118         for i, decision in enumerate(decisions[11:14]):
119             params['semi_product_params'][i]['disassemble'] = decision
120         params['final_product_params']['inspect'] = decisions[14]
121         params['final_product_params']['disassemble'] = decisions[15]
122
123         profit, revenue, cost = multi_stage_production_decision_process(**params)
124
125         # 保存决策与结果
126         decision_data.append({
127             'Decisions': decisions,

```

```

127         'Profit': profit,
128         'Revenue': revenue,
129         'Cost': cost
130     })
131
132     return decision_data
133
134 def save_decision_data_to_excel(decision_data, file_name):
135     # 将决策数据转换为DataFrame
136     df = pd.DataFrame(decision_data)
137
138     # 将数据保存到Excel文件
139     df.to_excel(file_name, index=False)
140
141 def main():
142     params = {
143         'initial_quantity': 1000,
144         'component_params': [
145             {'defect_rate': 0.10, 'purchase_cost': 2, 'inspection_cost': 1, 'inspect': False},
146             {'defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect': False},
147             {'defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect': False},
148             {'defect_rate': 0.10, 'purchase_cost': 2, 'inspection_cost': 1, 'inspect': False},
149             {'defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect': False},
150             {'defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect': False},
151             {'defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect': False},
152             {'defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect': False}
153         ],
154         'semi_product_params': [
155             {'defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4, 'disassembly_cost':
156                 6, 'components': [1, 2, 3], 'inspect': False, 'disassemble': False},
157             {'defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4, 'disassembly_cost':
158                 6, 'components': [4, 5, 6], 'inspect': False, 'disassemble': False},
159             {'defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4, 'disassembly_cost':
160                 6, 'components': [7, 8], 'inspect': False, 'disassemble': False}
161         ],
162         'final_product_params': {
163             'defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 6, 'market_price': 200,
164             'disassembly_cost': 10, 'return_loss': 40, 'inspect': False, 'disassemble': False
165         }
166     }
167
168     decision_data = optimize_multi_stage_decisions(params)
169
170     # 保存到Excel文件
171     save_decision_data_to_excel(decision_data, 'multi_stage_production_results.xlsx')
172     print("所有决策的利润数据已保存到 multi_stage_production_results.xlsx 文件中。")

```

```

171 # 找出最佳决策
172 best_decision = max(decision_data, key=lambda x: x['Profit'])
173 print("\n最佳决策:")
174 print(f"决策: {best_decision['Decisions']}")
175 print(f"利润: {best_decision['Profit']:.2f}")
176 print(f"收入: {best_decision['Revenue']:.2f}")
177 print(f"成本: {best_decision['Cost']:.2f}")
178
179 if __name__ == "__main__":
180     main()

```

2.4 问题四代码

2.4.1 样品次品率分布

```

1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib import font_manager
5
6 # 设置中文字体
7 plt.rcParams['font.sans-serif'] = ['SimHei'] # 用黑体显示中文
8 plt.rcParams['axes.unicode_minus'] = False # 正常显示负号
9
10
11 def sample_inspection(population_size, sample_size, true_defect_rate):
12     defects = sum(random.random() < true_defect_rate for _ in range(sample_size))
13     # 添加小的随机扰动以创造连续分布
14     sample_defect_rate = (defects + random.uniform(0, 1)) / (sample_size + 1)
15     std_error = (sample_defect_rate * (1 - sample_defect_rate) / sample_size) ** 0.5
16     ci_lower = max(0, sample_defect_rate - 1.96 * std_error)
17     ci_upper = min(1, sample_defect_rate + 1.96 * std_error)
18     return sample_defect_rate, (ci_lower, ci_upper)
19
20 def run_multiple inspections(population_size, sample_size, true_defect_rate, num_simulations):
21     return [sample_inspection(population_size, sample_size, true_defect_rate)[0] for _ in
22             range(num_simulations)]
23
24 def analyze_results(results, true_defect_rate):
25     mean = np.mean(results)
26     std_dev = np.std(results)
27     median = np.median(results)
28
29     print(f"真实次品率: {true_defect_rate:.4f}")
30     print(f"平均估计次品率: {mean:.4f}")

```

```

30 print(f"中位数估计次品率: {median:.4f}")
31 print(f"标准差: {std_dev:.4f}")
32
33 sem = std_dev / np.sqrt(len(results))
34 ci_lower = mean - 1.96 * sem
35 ci_upper = mean + 1.96 * sem
36 print(f"95%置信区间: ({ci_lower:.4f}, {ci_upper:.4f})")
37
38 # 添加偏差和均方根误差
39 bias = mean - true_defect_rate
40 rmse = np.sqrt(np.mean((np.array(results) - true_defect_rate)**2))
41 print(f"偏差: {bias:.4f}")
42 print(f"均方根误差: {rmse:.4f}")
43
44 def plot_results(results, true_defect_rate):
45     plt.figure(figsize=(12, 7))
46     n, bins, patches = plt.hist(results, bins=50, edgecolor='black', alpha=0.7)
47     plt.axvline(true_defect_rate, color='r', linestyle='dashed', linewidth=2, label='真实次品率')
48     plt.axvline(np.mean(results), color='g', linestyle='dashed', linewidth=2,
49                 label='平均估计次品率')
50
51     plt.xlabel('估计次品率')
52     plt.ylabel('频数')
53     plt.title('次品率估计分布')
54     plt.legend()
55
56     # 添加注释
57     plt.annotate(f'真实次品率: {true_defect_rate:.4f}', xy=(0.7, 0.95), xycoords='axes fraction')
58     plt.annotate(f'平均估计次品率: {np.mean(results):.4f}', xy=(0.7, 0.90), xycoords='axes
59                 fraction')
60     plt.annotate(f'标准差: {np.std(results):.4f}', xy=(0.7, 0.85), xycoords='axes fraction')
61
62     plt.savefig('defect_rate_distribution.png', dpi=300, bbox_inches='tight')
63     plt.close()
64
65 def main():
66     population_size = 10000
67     sample_size = 100
68     true_defect_rate = 0.10
69     num_simulations = 50000 # 增加模拟次数
70
71     results = run_multiple_inspections(population_size, sample_size, true_defect_rate,
72                                       num_simulations)
73
74     analyze_results(results, true_defect_rate)
75     plot_results(results, true_defect_rate)

```

```

74     print("分布图已保存为 'defect_rate_distribution.png'")
75
76 if __name__ == "__main__":
77     main()

```

2.4.2 修改后问题二的模型

```

1  import random
2  import itertools
3  from scipy import stats
4
5
6  def sample_inspection(population_size, sample_size, true_defect_rate):
7      """
8      Perform a sample inspection and estimate the defect rate.
9      """
10     defects = sum(random.random() < true_defect_rate for _ in range(sample_size))
11     sample_defect_rate = defects / sample_size
12
13     # Calculate standard error
14     std_error = (sample_defect_rate * (1 - sample_defect_rate) / sample_size) ** 0.5
15
16     # Calculate confidence interval
17     ci_lower = max(0, sample_defect_rate - 1.96 * std_error)
18     ci_upper = min(1, sample_defect_rate + 1.96 * std_error)
19
20     return sample_defect_rate, (ci_lower, ci_upper)
21
22
23 def improved_production_decision_process(
24     initial_quantity,
25     true_defect_rate_1, purchase_cost_1, inspection_cost_1,
26     true_defect_rate_2, purchase_cost_2, inspection_cost_2,
27     true_defect_rate_product, assembly_cost, inspection_cost_product,
28     market_price, return_loss, disassembly_cost,
29     inspect_component_1=True, inspect_component_2=True, inspect_product=True,
30     disassemble_defective=True, max_cycles=2,
31     sample_size_1=100, sample_size_2=100, sample_size_product=100
32 ):
33     total_revenue = 0
34     total_cost = 0
35
36     # Initial purchase cost
37     total_cost += initial_quantity * (purchase_cost_1 + purchase_cost_2)
38

```

```

39 # Sample inspection of components
40 defect_rate_1, _ = sample_inspection(initial_quantity, sample_size_1, true_defect_rate_1)
41 defect_rate_2, _ = sample_inspection(initial_quantity, sample_size_2, true_defect_rate_2)
42
43 if inspect_component_1:
44     inventory_1 = int(initial_quantity * (1 - defect_rate_1))
45     total_cost += initial_quantity * inspection_cost_1
46 else:
47     inventory_1 = initial_quantity
48
49 if inspect_component_2:
50     inventory_2 = int(initial_quantity * (1 - defect_rate_2))
51     total_cost += initial_quantity * inspection_cost_2
52 else:
53     inventory_2 = initial_quantity
54
55 for cycle in range(max_cycles):
56     # Calculate current batch quality rate
57     part1_quality = 1 - defect_rate_1 if not inspect_component_1 else 1.0
58     part2_quality = 1 - defect_rate_2 if not inspect_component_2 else 1.0
59
60     if inventory_1 == 0 or inventory_2 == 0:
61         break
62
63     assembled_products = min(inventory_1, inventory_2)
64
65     inventory_1 -= assembled_products
66     inventory_2 -= assembled_products
67     total_cost += assembled_products * assembly_cost
68
69     # Sample inspection of products
70     defect_rate_product, _ = sample_inspection(assembled_products, sample_size_product,
71                                                true_defect_rate_product)
72     product_quality = part1_quality * part2_quality * (1 - defect_rate_product)
73
74     if inspect_product:
75         qualified_products = int(assembled_products * product_quality)
76         defective_products = assembled_products - qualified_products
77         total_cost += assembled_products * inspection_cost_product
78         returned_products = 0
79     else:
80         qualified_products = int(assembled_products * product_quality)
81         defective_products = assembled_products - qualified_products
82         returned_products = defective_products
83         total_cost += returned_products * return_loss
84
85     total_revenue += qualified_products * market_price

```

```

85
86     total_defective = defective_products
87
88     if total_defective > 0:
89         if disassemble_defective:
90             total_cost += total_defective * disassembly_cost
91             inventory_1 += total_defective
92             inventory_2 += total_defective
93         else:
94             total_defective = 0
95
96     profit = total_revenue - total_cost
97     return profit, total_revenue, total_cost
98
99
100 def optimize_decisions(params):
101     best_profit = float('-inf')
102     best_decisions = None
103
104     for decisions in itertools.product([True, False], repeat=4):
105         inspect_1, inspect_2, inspect_prod, disassemble = decisions
106         profit, revenue, cost = improved_production_decision_process(
107             inspect_component_1=inspect_1,
108             inspect_component_2=inspect_2,
109             inspect_product=inspect_prod,
110             disassemble_defective=disassemble,
111             **params
112         )
113         if profit > best_profit:
114             best_profit = profit
115             best_decisions = decisions
116
117     return best_profit, best_decisions
118
119
120 def main():
121     params = {
122         'initial_quantity': 1000,
123         'assembly_cost': 6, 'market_price': 56
124     }
125
126     table_cases = [
127         {'true_defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
128          'true_defect_rate_2': 0.10, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
129          'true_defect_rate_product': 0.10, 'return_loss': 6, 'disassembly_cost': 5,
130          'inspection_cost_product': 3},

```



```

131     {'true_defect_rate_1': 0.20, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
132      'true_defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
133      'true_defect_rate_product': 0.20, 'return_loss': 6, 'disassembly_cost': 5,
134      'inspection_cost_product': 3},
135
136     {'true_defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
137      'true_defect_rate_2': 0.10, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
138      'true_defect_rate_product': 0.10, 'return_loss': 30, 'disassembly_cost': 5,
139      'inspection_cost_product': 3},
140
141     {'true_defect_rate_1': 0.20, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
142      'true_defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
143      'true_defect_rate_product': 0.20, 'return_loss': 30, 'disassembly_cost': 5,
144      'inspection_cost_product': 2},
145
146     {'true_defect_rate_1': 0.10, 'purchase_cost_1': 4, 'inspection_cost_1': 8,
147      'true_defect_rate_2': 0.20, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
148      'true_defect_rate_product': 0.10, 'return_loss': 10, 'disassembly_cost': 5,
149      'inspection_cost_product': 2},
150
151     {'true_defect_rate_1': 0.05, 'purchase_cost_1': 4, 'inspection_cost_1': 2,
152      'true_defect_rate_2': 0.05, 'purchase_cost_2': 18, 'inspection_cost_2': 3,
153      'true_defect_rate_product': 0.05, 'return_loss': 10, 'disassembly_cost': 40,
154      'inspection_cost_product': 3}
155 ]
156
157 num_simulations = 1000 # Number of simulations for each case
158
159 for i, case in enumerate(table_cases, 1):
160     case_params = params.copy()
161     case_params.update(case)
162
163     total_profit = 0
164     best_decisions_count = {(True, True, True, True): 0, (True, True, True, False): 0,
165                             (True, True, False, True): 0, (True, True, False, False): 0,
166                             (True, False, True, True): 0, (True, False, True, False): 0,
167                             (True, False, False, True): 0, (True, False, False, False): 0,
168                             (False, True, True, True): 0, (False, True, True, False): 0,
169                             (False, True, False, True): 0, (False, True, False, False): 0,
170                             (False, False, True, True): 0, (False, False, True, False): 0,
171                             (False, False, False, True): 0, (False, False, False, False): 0}
172
173     for _ in range(num_simulations):
174         best_profit, best_decisions = optimize_decisions(case_params)
175         total_profit += best_profit
176         best_decisions_count[best_decisions] += 1

```

```

173     avg_profit = total_profit / num_simulations
174     most_common_decision = max(best_decisions_count, key=best_decisions_count.get)
175
176     print(f"\nCase {i}:")
177     print(f"Parameters: {case}")
178     print(f"Average best profit: {avg_profit:.2f}")
179     print(f"Most common best decision: Inspect Component 1: {most_common_decision[0]}, "
180           f"Inspect Component 2: {most_common_decision[1]}, "
181           f"Inspect Product: {most_common_decision[2]}, "
182           f"Disassemble Defective: {most_common_decision[3]}")
183     print(f"Decision distribution: {best_decisions_count}")
184
185
186 if __name__ == "__main__":
187     main()

```

2.4.3 修改后问题三的模型

```

1  import itertools
2  import pandas as pd
3  import random
4  from collections import Counter
5
6
7  def sample_inspection(population_size, sample_size, true_defect_rate):
8      if sample_size == 0:
9          return 0
10     defects = sum(random.random() < true_defect_rate for _ in range(sample_size))
11     return defects / sample_size
12
13
14  def multi_stage_production_decision_process(
15     initial_quantity,
16     component_params,
17     semi_product_params,
18     final_product_params,
19     max_cycles=2
20 ):
21     total_revenue = 0
22     total_cost = 0
23
24     for comp in component_params:
25         total_cost += initial_quantity * comp['purchase_cost']
26
27     inventories = []

```

```

28 for comp in component_params:
29     if comp['inspect']:
30         estimated_defect_rate = sample_inspection(initial_quantity, min(100,
31             initial_quantity), comp['true_defect_rate'])
32         inventory = int(initial_quantity * (1 - estimated_defect_rate))
33         total_cost += initial_quantity * comp['inspection_cost']
34     else:
35         inventory = initial_quantity
36         inventories.append(inventory)
37
38 semi_product_inventories = [0] * len(semi_product_params)
39
40 for cycle in range(max_cycles):
41     semi_products = []
42     for idx, semi_prod in enumerate(semi_product_params):
43         assembled = min([inventories[i-1] for i in semi_prod['components']])
44         assembled += semi_product_inventories[idx]
45         semi_product_inventories[idx] = 0
46
47         for i in semi_prod['components']:
48             inventories[i-1] -= min(assembled, inventories[i-1])
49             total_cost += assembled * semi_prod['assembly_cost']
50
51     semi_product_quality = 1.0
52     for i in semi_prod['components']:
53         if not component_params[i-1]['inspect']:
54             semi_product_quality *= (1 - component_params[i-1]['true_defect_rate'])
55             semi_product_quality *= (1 - semi_prod['true_defect_rate'])
56
57     actual_qualified = int(assembled * semi_product_quality)
58     if semi_prod['inspect'] and assembled > 0:
59         estimated_defect_rate = sample_inspection(assembled, min(100, assembled), 1 -
60             semi_product_quality)
61         qualified = int(assembled * (1 - estimated_defect_rate))
62         defective = assembled - qualified
63         total_cost += assembled * semi_prod['inspection_cost']
64         if semi_prod['disassemble']:
65             total_cost += defective * semi_prod['disassembly_cost']
66             for i in semi_prod['components']:
67                 inventories[i-1] += defective
68         else:
69             qualified = assembled
70
71     semi_products.append((qualified, actual_qualified))
72
73 final_assembled = min([sp[0] for sp in semi_products])
74 total_cost += final_assembled * final_product_params['assembly_cost']

```

```

73
74     final_quality = 1.0
75     for idx, sp in enumerate(semi_products):
76         qualified, actual_qualified = sp
77         semi_product_quality = actual_qualified / qualified if qualified > 0 else 0
78         final_quality *= semi_product_quality
79     final_quality *= (1 - final_product_params['true_defect_rate'])
80
81     actual_qualified_products = int(final_assembled * final_quality)
82
83     if final_product_params['inspect'] and final_assembled > 0:
84         estimated_defect_rate = sample_inspection(final_assembled, min(100,
85             final_assembled), 1 - final_quality)
86         qualified_products = int(final_assembled * (1 - estimated_defect_rate))
87         defective_products = final_assembled - qualified_products
88         total_cost += final_assembled * final_product_params['inspection_cost']
89         returned_products = 0
90     else:
91         qualified_products = actual_qualified_products
92         defective_products = final_assembled - qualified_products
93         returned_products = defective_products
94         total_cost += returned_products * final_product_params['return_loss']
95
96     total_revenue += qualified_products * final_product_params['market_price']
97
98     total_defective = defective_products
99     if total_defective > 0 and final_product_params['disassemble']:
100         total_cost += total_defective * final_product_params['disassembly_cost']
101         for i in range(len(semi_product_params)):
102             semi_product_inventories[i] += total_defective // len(semi_product_params)
103
104     profit = total_revenue - total_cost
105     return profit, total_revenue, total_cost
106
107 def optimize_multi_stage_decisions(params, decision_combinations):
108     best_profit = float('-inf')
109     best_decision = None
110
111     for decisions in decision_combinations:
112         for i, decision in enumerate(decisions[:8]):
113             params['component_params'][i]['inspect'] = decision
114         for i, decision in enumerate(decisions[8:11]):
115             params['semi_product_params'][i]['inspect'] = decision
116         for i, decision in enumerate(decisions[11:14]):
117             params['semi_product_params'][i]['disassemble'] = decision
118         params['final_product_params']['inspect'] = decisions[14]
119         params['final_product_params']['disassemble'] = decisions[15]

```

```

119     profit, revenue, cost = multi_stage_production_decision_process(**params)
120
121
122     if profit > best_profit:
123         best_profit = profit
124         best_decision = decisions
125
126     return best_decision, best_profit
127
128
129 def run_multiple_simulations(params, decision_combinations, num_simulations=1000):
130     best_decisions = []
131     for _ in range(num_simulations):
132         best_decision, best_profit = optimize_multi_stage_decisions(params,
133                             decision_combinations)
134         best_decisions.append(best_decision)
135
136     decision_counts = Counter(tuple(decision) for decision in best_decisions)
137     return decision_counts
138
139 def main():
140     params = {
141         'initial_quantity': 1000,
142         'component_params': [
143             {'true_defect_rate': 0.10, 'purchase_cost': 2, 'inspection_cost': 1, 'inspect':
144                 False},
145             {'true_defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect':
146                 False},
147             {'true_defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect':
148                 False},
149             {'true_defect_rate': 0.10, 'purchase_cost': 2, 'inspection_cost': 1, 'inspect':
150                 False},
151             {'true_defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect':
152                 False},
153             {'true_defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect':
154                 False},
155             {'true_defect_rate': 0.10, 'purchase_cost': 8, 'inspection_cost': 1, 'inspect':
156                 False},
157             {'true_defect_rate': 0.10, 'purchase_cost': 12, 'inspection_cost': 2, 'inspect':
158                 False}
159         ],
160         'semi_product_params': [
161             {'true_defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4,
162                 'disassembly_cost': 6, 'components': [1, 2, 3], 'inspect': False, 'disassemble':
163                 False},
164             {'true_defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4,

```

```

        'disassembly_cost': 6, 'components': [4, 5, 6], 'inspect': False, 'disassemble':
        False},
155     {'true_defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 4,
        'disassembly_cost': 6, 'components': [7, 8], 'inspect': False, 'disassemble':
        False}
156 ],
157     'final_product_params': {
158         'true_defect_rate': 0.10, 'assembly_cost': 8, 'inspection_cost': 6, 'market_price':
        200,
159         'disassembly_cost': 10, 'return_loss': 40, 'inspect': False, 'disassemble': False
160     }
161 }
162
163 # 从文件中读取决策组合
164 with open('decision_combinations.txt', 'r') as f:
165     decision_combinations = [eval(line.strip()) for line in f]
166
167 decision_counts = run_multiple_simulations(params, decision_combinations)
168
169 print("最优策略统计结果:")
170 for decision, count in decision_counts.most_common():
171     print(f"策略: {decision}, 出现次数: {count}")
172
173 df = pd.DataFrame([
174     {"Strategy": str(decision), "Count": count}
175     for decision, count in decision_counts.items()
176 ])
177 df.to_excel("optimal_strategies_distribution.xlsx", index=False)
178 print("结果已保存到 optimal_strategies_distribution.xlsx 文件中。")
179
180
181 if __name__ == "__main__":
182     main()

```

附录 C 附录文件

3.1 问题二文件

表 10 情况一决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	13491	49896	36405
TRUE	TRUE	TRUE	FALSE	10260	45360	35100
TRUE	TRUE	FALSE	TRUE	15867	49896	34029
TRUE	TRUE	FALSE	FALSE	12420	45360	32940
TRUE	FALSE	TRUE	TRUE	13893	48552	34659
TRUE	FALSE	TRUE	FALSE	8724	40824	32100
TRUE	FALSE	FALSE	TRUE	15882	48552	32670
TRUE	FALSE	FALSE	FALSE	10398	40824	30426
FALSE	TRUE	TRUE	TRUE	12893	48552	35659
FALSE	TRUE	TRUE	FALSE	7724	40824	33100
FALSE	TRUE	FALSE	TRUE	14882	48552	33670
FALSE	TRUE	FALSE	FALSE	9398	40824	31426
FALSE	FALSE	TRUE	TRUE	16692	51856	35164
FALSE	FALSE	TRUE	FALSE	9824	40824	31000
FALSE	FALSE	FALSE	TRUE	18435	51856	33421
FALSE	FALSE	FALSE	FALSE	11198	40824	29626

表 11 情况二决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	6408	43008	36600
TRUE	TRUE	TRUE	FALSE	1640	35840	34200
TRUE	TRUE	FALSE	TRUE	8136	43008	34872
TRUE	TRUE	FALSE	FALSE	3080	35840	32760
TRUE	FALSE	TRUE	TRUE	3224	38976	35752
TRUE	FALSE	TRUE	FALSE	-2528	28672	31200
TRUE	FALSE	FALSE	TRUE	4136	38976	34840
TRUE	FALSE	FALSE	FALSE	-1856	28672	30528
FALSE	TRUE	TRUE	TRUE	2224	38976	36752
FALSE	TRUE	TRUE	FALSE	-3528	28672	32200
FALSE	TRUE	FALSE	TRUE	3136	38976	35840
FALSE	TRUE	FALSE	FALSE	-2856	28672	31528
FALSE	FALSE	TRUE	TRUE	3589	42616	39027
FALSE	FALSE	TRUE	FALSE	-2328	28672	31000
FALSE	FALSE	FALSE	TRUE	3691	42616	38925
FALSE	FALSE	FALSE	FALSE	-2256	28672	30928

表 12 情况三决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	13491	49896	36405
TRUE	TRUE	TRUE	FALSE	10260	45360	35100
TRUE	TRUE	FALSE	TRUE	13491	49896	36405
TRUE	TRUE	FALSE	FALSE	10260	45360	35100
TRUE	FALSE	TRUE	TRUE	13893	48552	34659
TRUE	FALSE	TRUE	FALSE	8724	40824	32100
TRUE	FALSE	FALSE	TRUE	10986	48552	37566
TRUE	FALSE	FALSE	FALSE	6294	40824	34530
FALSE	TRUE	TRUE	TRUE	12893	48552	35659
FALSE	TRUE	TRUE	FALSE	7724	40824	33100
FALSE	TRUE	FALSE	TRUE	9986	48552	38566
FALSE	TRUE	FALSE	FALSE	5294	40824	35530
FALSE	FALSE	TRUE	TRUE	16692	51856	35164
FALSE	FALSE	TRUE	FALSE	9824	40824	31000
FALSE	FALSE	FALSE	TRUE	10155	51856	41701
FALSE	FALSE	FALSE	FALSE	4694	40824	36130

表 13 情况四决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	7368	43008	35640
TRUE	TRUE	TRUE	FALSE	2440	35840	33400
TRUE	TRUE	FALSE	TRUE	3528	43008	39480
TRUE	TRUE	FALSE	FALSE	-760	35840	36600
TRUE	FALSE	TRUE	TRUE	4312	38976	34664
TRUE	FALSE	TRUE	FALSE	-1728	28672	30400
TRUE	FALSE	FALSE	TRUE	-5272	38976	44248
TRUE	FALSE	FALSE	FALSE	-8768	28672	37440
FALSE	TRUE	TRUE	TRUE	3312	38976	35664
FALSE	TRUE	TRUE	FALSE	-2728	28672	31400
FALSE	TRUE	FALSE	TRUE	-6272	38976	45248
FALSE	TRUE	FALSE	FALSE	-9768	28672	38440
FALSE	FALSE	TRUE	TRUE	5077	42616	37539
FALSE	FALSE	TRUE	FALSE	-1328	28672	30000
FALSE	FALSE	FALSE	TRUE	-13757	42616	56373
FALSE	FALSE	FALSE	FALSE	-13968	28672	42640

表 14 情况五决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	3872	44352	40480
TRUE	TRUE	TRUE	FALSE	920	40320	39400
TRUE	TRUE	FALSE	TRUE	4752	44352	39600
TRUE	TRUE	FALSE	FALSE	1720	40320	38600
TRUE	FALSE	TRUE	TRUE	5593	46424	40831
TRUE	FALSE	TRUE	FALSE	-912	36288	37200
TRUE	FALSE	FALSE	TRUE	4667	46424	41757
TRUE	FALSE	FALSE	FALSE	-1632	36288	37920
FALSE	TRUE	TRUE	TRUE	9655	43176	33521
FALSE	TRUE	TRUE	FALSE	4888	36288	31400
FALSE	TRUE	FALSE	TRUE	9749	43176	33427
FALSE	TRUE	FALSE	FALSE	4968	36288	31320
FALSE	FALSE	TRUE	TRUE	13860	49056	35196
FALSE	FALSE	TRUE	FALSE	6288	36288	30000
FALSE	FALSE	FALSE	TRUE	11804	49056	37252
FALSE	FALSE	FALSE	FALSE	4768	36288	31520

表 15 情况六决策方案表

Inspect Component 1	Inspect Component 2	Inspect Product	Disassemble Defective	Profit	Revenue	Cost
TRUE	TRUE	TRUE	TRUE	15010	53032	38022
TRUE	TRUE	TRUE	FALSE	14962	50512	35550
TRUE	TRUE	FALSE	TRUE	17494	53032	35538
TRUE	TRUE	FALSE	FALSE	17332	50512	33180
TRUE	FALSE	TRUE	TRUE	15133	52640	37507
TRUE	FALSE	TRUE	FALSE	15442	47992	32550
TRUE	FALSE	FALSE	TRUE	17232	52640	35408
TRUE	FALSE	FALSE	FALSE	17362	47992	30630
FALSE	TRUE	TRUE	TRUE	14133	52640	38507
FALSE	TRUE	TRUE	FALSE	14442	47992	33550
FALSE	TRUE	FALSE	TRUE	16232	52640	36408
FALSE	TRUE	FALSE	FALSE	16362	47992	31630
FALSE	FALSE	TRUE	TRUE	15977	54824	38847
FALSE	FALSE	TRUE	FALSE	16992	47992	31000
FALSE	FALSE	FALSE	TRUE	17766	54824	37058
FALSE	FALSE	FALSE	FALSE	18562	47992	29430