基于决策优化模型的企业生产决策分析与应用

摘要

研究企业在生产过程中的决策对企业的良性发展至关重要。本文建立决策优化模型，运用假设检验、决策树、模拟退火、单因素分析法等方法，解决了企业在生产过程中的一系列决策问题，对缩减企业成本，提升企业效益有一定作用。

对于问题一, 首先确定抽样检测方法为简单随机抽样，对于样本量 n，使用正态近似模拟其条件分布。然后通过单侧假设检验确定拒收或接收条件，推导出样本量计算公式。最后需要设定拒收阈值（最小偏差 + 标称值）解出最小样本量。在情况一中，将**偏差定为 0.05** 时，得到**样本量为 97**，将**偏差定为 0.1 时**，得到**样本量为 24**；在情况二中，将**偏差定为 0.05 时**，得到**样本量为 59**，将**偏差定为 0.1 时**，得到**样本量为 15;**

对于问题二, 以最大化企业利润为目标，根据不同情况下的次品率、检测成本、装配成本、调换损失、市场售价等一系列参数，确定决策变量、构建目标函数，从而建立决策树模型。对于每种情况，以每一步决策所带来的成本或收益为依据，最优利润期望与对应成本为指标 [1]，遍历其所有可能的决策方案，得到最优决策。具体决策方案与指标结果详见问题二求解结果。

对于问题三, 需要在问题二的基础上同时考虑两道工序，多个零配件和半成品，其中涉及多种方案决策，分析这些决策对期望利润和成本的影响，从而得到最优决策。首先根据题目信息对决策变量进行简单降维处理，将具有相同特征的零部件和半成品用同一个决策变量进行控制，然后在此基础上利用模拟退火算法得到利润最大化的最优决策方案，**此时利润最大为 69.4 元，成本为 110.6 元**，具体决策方案见问题三求解结果。

对于问题四, 在零部件在次品率是由抽样检测得出的情况下，考虑抽样检测的误差，重新解决问题二和问题三。首先对决策进行独热编码，并基于统计学的思想改进问题二、三的模型，利用**单因素分析法**建立了**零部件次品率——决策位频率变化模型**。综合得到最优决策，具体方案见问题四求解结果。

关键字：企业生产　决策优化　假设检验　决策树模型　模拟退火　单因素分析法

一、 问题重述

1.1 问题背景

某企业生产畅销的电子产品，需要采购两种零配件（零配件 1 和零配件 2），并通过装配为成品进行销售。成品质量受零配件的合格性影响，若任何一个零配件不合格，则成品必然不合格；即使两个零配件均合格，装配出的成品仍可能出现不合格的情况。对于不合格成品，企业可以选择报废或者拆解，拆解后零配件不受损坏，但需要支付拆解费用。

企业在生产过程中面临以下几个重要决策：如何设计最优的抽样检测决定是否接收零部件，如何在各个生产阶段做出最优检测与处理决策，以及如何应对多道工序和多个零配件的生产情景。为解决这些问题，需建立合适的数学模型，分析和制定出合理的策略，以最小化成本并保障生产效率。

1.2 问题要求

问题 1：供应商声称一批零配件的次品率不会超过某个标称值，企业准备采用抽样检测方法决定是否接收该批次零配件。检测次数应尽可能少，且满足以下要求：

1. 当 95% 的信度下认定零配件次品率超过标称值时，拒收该批零配件；
2. 当 90% 的信度下认定零配件次品率不超过标称值时，接收该批零配件。

问题 2：已知零配件和成品的次品率，企业需对生产过程各阶段作出以下决策：

1. 是否对零配件（零配件 1 和/或零配件 2）进行检测，不检测的零配件直接进入装配环节；
2. 是否对成品进行检测，不检测的成品直接进入市场；
3. 是否对不合格成品进行拆解，拆解后的零配件重新进入检测和装配流程；
4. 对用户退回的不合格成品，企业需无条件调换，并承担相应损失（如物流成本、企业信誉损失等），退回的不合格成品进入拆解流程。

问题 3：针对多个零配件和多道工序的生产情况，企业需重复问题 2 中的决策。并给出相应的决策方案和指标结果。

问题 4：假设问题 2 和问题 3 中的零配件、半成品和成品的次品率通过抽样检测方法得到，重新完成问题 2 和问题 3 中的生产决策。

二、 问题分析

2.1 问题一分析

在问题一中，我们需设计一种抽样检测方案，以评估供应商提供的零配件是否满足标称次品率要求。目标是在给定的信度水平下，采用最少的检测次数，合理地做出接收或拒收决策。为简化建模过程，我们忽略产品批次和编号等次要因素，选择**简单随机抽样方法**进行样品抽取。因此，问题的核心在于**合理确定抽样量和检测标准**，以在有效控制次品率风险的同时，降低检测成本，提高企业效益。

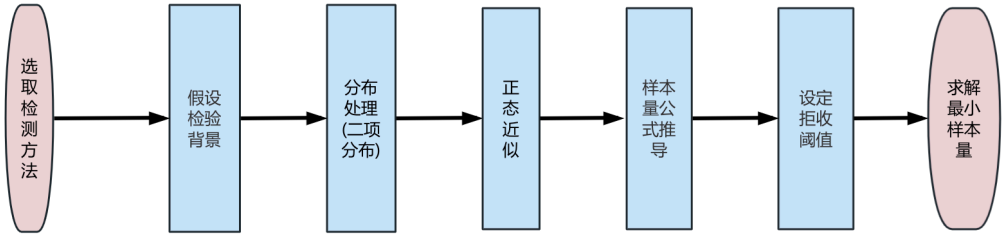


图 1　分析过程

在抽样量 n 的计算中，我们基于**假设检验及二项分布的正态近似**，通过假设检验确定拒收或接收条件，并推导样本量计算公式。针对检测标准，由于两问中我们均只需认定零配件次品率超过或者不超过标称值，因此符合**单侧检验**的要求。并依据假设检验结果，设定拒收阈值（即最小偏差 [2]+ 标称值）。若检测出的次品率超过该阈值，则拒收该批次零配件，否则予以接收。

2.2 问题二分析

针对问题二，我们需综合考虑不同情境下的次品率、检测成本、装配成本、替换损失、市场售价等多项关键参数，构建一个优化利润的决策模型。通过对每种情况的不同决策进行组合，我们需要评估是否对零配件以及成品进行检测，以及是否对不合格成品进行拆解，以获得最大化利润。

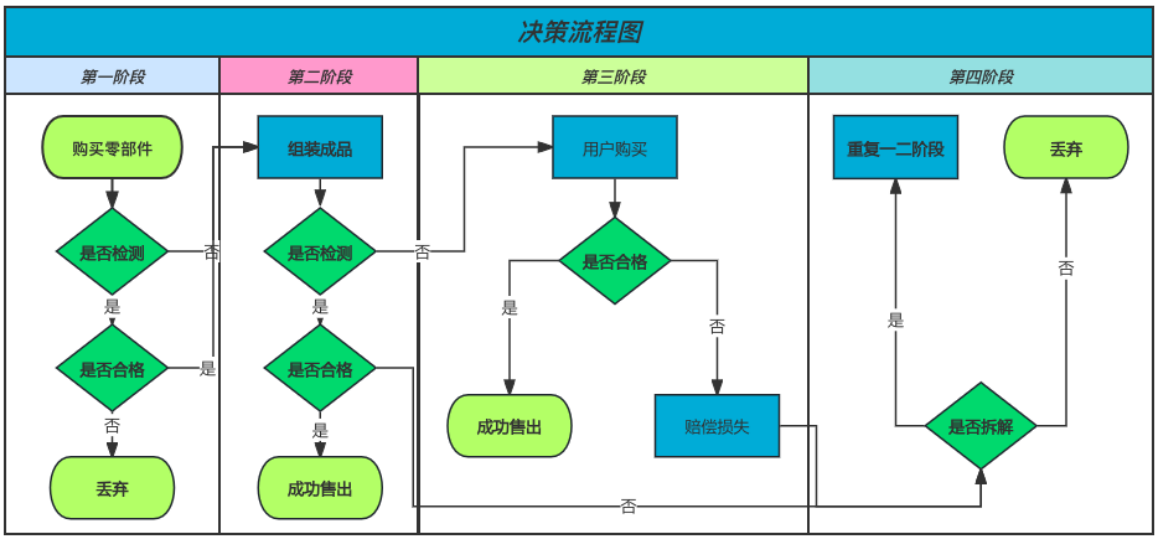


图 2　决策分析过程

问题二的主要目标是通过优化决策方案来最大化利润。我们基于**决策树**模型，遍历每种可能的决策方案，并对比各叶子结点的利润期望，从而选择最优方案。

2.3 问题三分析

对于问题三，面对一个包含 8 个零部件、3 个半成品和最终成品的复杂生产关系，我们需优化每一阶段的检测和拆解决策，目标是控制控制成本和次品率的前提下最大化企业利润。为此我们根据题目的特性进行了简化处理和降维分析。

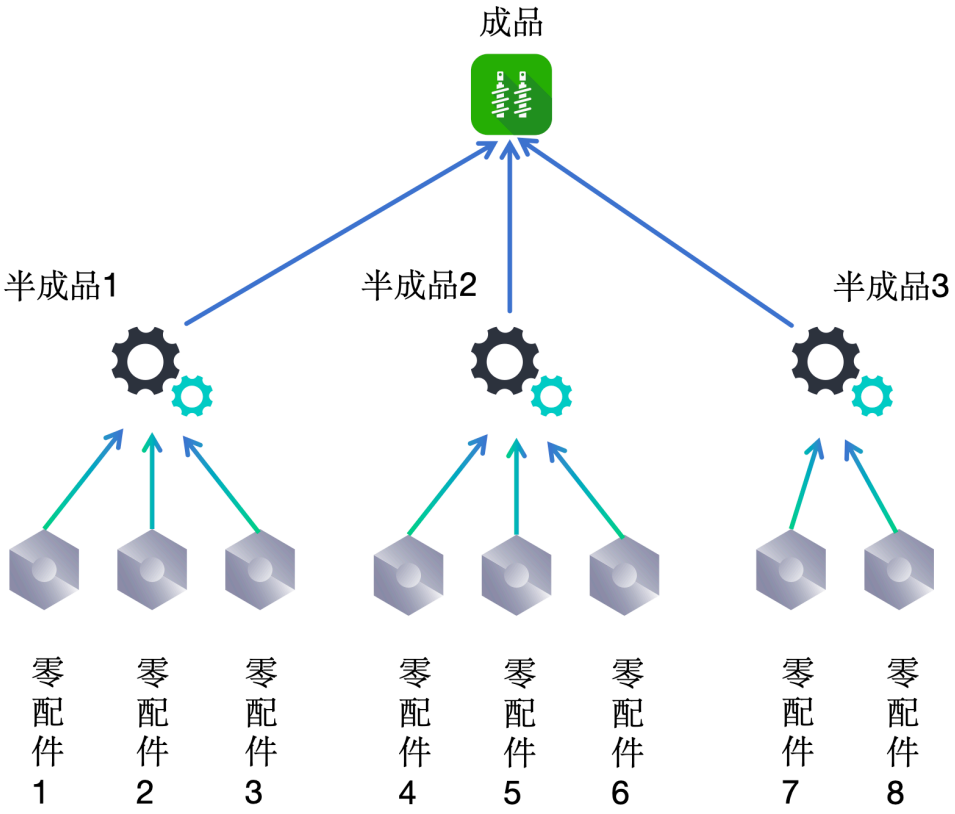


图 3　两道工序示意图

由工序图可以看出半成品 1 由零配件 1、2、3 组成；半成品 2 由零配件 4、5、6 组成；半成品 3 由零件 7、8 组成。根据题目所给零配件、半成品的详细信息，发现：

- 从次品率、购买单价、检测成本这三个指标来看，零件 1 与零件 4 相同，零件 2 与零件 5 和零件 7 相同，零件 3 与零件 6 和零件 8 相同。
- 从配件组成、次品率、装配成本和检测成本来看，半成品 1 和半成品 2 可被视作等价对象。

基于以上分析，我们将决策变量降维，使用单个决策变量同步控制具有相同特征的零部件和半成品：

1. **零部件降维：**通过分析，零件 1、4，零件 2、5、7，零件 3、6、8 被分别合并为三个代表零件，记为零件 A,B,C。
2. **半成品降维：**在工程指标下，半成品 1 和半成品 2 可以视为完全相同，合并为一个半成品，半成品 3 独立存在。因此，降维后我们只需考虑 2 个半成品、5 个零部件的检测和拆解决策。

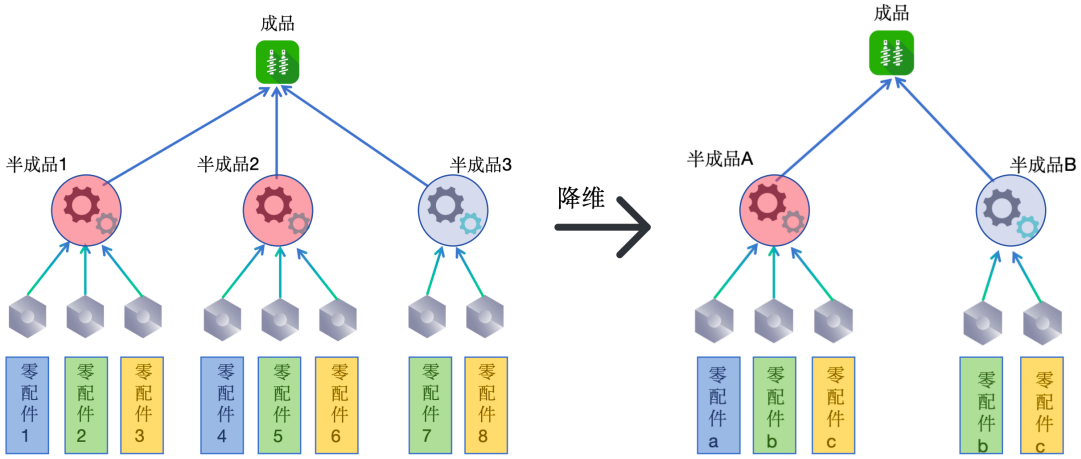


图 4　决策变量降维

在问题二中，决策树模型能有效处理较少的决策变量，找到最优策略。然而，在问题三中，如果使用决策树算法，决策树的节点数量将呈指数级增长，空间复杂度高，并且容易陷入局部最优解。在问题三中，**模拟退火算法比决策树模型更具优势**，模拟退火算法具备强大的全局搜索能力，能够通过控制温度参数，在初始阶段接受较大的解空间探索，从而避免陷入局部最优解，并逐渐逼近全局最优解。此外，模拟退火算法能够灵活处理多个决策变量，通过不断的迭代和解的更新，寻找最优的检测和拆解策略。其高效性和可控性使得算法可以在合理的计算时间内找到接近全局最优解 [3]。

我们只需从随机初始化的检测、拆解决策组合开始，逐步更新领域解，并在每次决策后分别计算零件、半成品、成品的检测成本、装配成本、拆解收益和调换损失，并最终确定成品的总成本与次品率，从而计算利润。在算法的降温过程中，若其利润较高，新的决策方案会被接受，**或者以一定概率接受较差的解，从而避免陷入局部最优**。算法在温度降到一定程度或者迭代次数达到设定上限时停止，并输出使利润最大化的最优决策方案。

2.4 问题四分析

对于问题四的分析，我们需在给定抽样检测结果的次品率基础上，重新设计问题 2 和问题 3 中的决策方案。为简化模型分析，我们假定问题 2 和问题 3 中所给的次品率为其**数学期望**。

接下来，我们设定真实次品率的条件分布的最大方差 σ^2，根据该方差及显著水平 α（通常为 95%），计算出置信区间，以确定次品率的上限和下限。在该区间内，以步长 Δp 生成次品率值，并对每个次品率执行问题对应的模型，求解最优决策。

在描述问题之前，我们定义两个概念：**标准件**和**决策位**。

1. **标准件：**零部件、半成品和成品的统称。例如问题二中，零部件 1，2，成品可分别记作标准件 1，标准件 2 和标准件 3。
2. **决策位：**对决策方案进行独热编码后，每一位编码即可表示一种决策，这一位叫作决策位。例如问题二中，决策位 1 可以表示“零部件 1 是否生产”，取 1 表示“检测”，取 0 表示“不检测”。

为避免指数级增长的决策复杂度，同时也为了观测单一变量对整体决策的影响，我们采用**单因素分析法**：当某一零件的次品率变化时，其余零件的次品率保持不变，计算该零件的最优决策 [4]。接着使用**独热编码**记录各个决策位结果。对于问题二，**统计每种情形下**，在各个部件的次品率变化时，每个决策位取值为 1 的频率。在问题三中，不需要再区分情形，只需要类似地统计每个零部件的次品率变化时，其各个决策位等于 1 的频率，按照相同标准判断是否采取相关策略。若某一决策位为 1 的频率明显高于 0.5，则策略为“是”；若明显低于 0.5，则策略为“否”。对于临界情况，可以结合企业具体条件做进一步调整。

三、 模型假设

为简化问题，本文做出以下假设：

- **假设 1：**每个零配件的质量状态（合格或次品）相互独立，即零配件 1 的次品率与零配件 2 的次品率不相关。
- **假设 2：**零配件和成品的检测是完全准确的，不需要考虑检测的错误率或检测设备性能，即不会出现漏检或误检的情况。
- **假设 3：**假设企业能够实时对零配件和成品进行检测与拆解，不需要考虑库存积压或生产延迟的影响。
- **假设 4：**假设零配件的供应是充足的，即在检测出次品后，能够立刻从供应商处购入合格的零配件，不受供应链限制。
- **假设 5：**假设企业的目标是最大化利润，不需要考虑其他因素如环境影响、生产稳定性等。企业能更加聚焦于收益和检测策略的优化，不过多在意前期付出的成本。

四、 符号说明

符号	说明	单位
p_0	标称次品率	%
p_1	检测结果中的次品率	%
δ	实际次品率与标称次品率的偏差	%
n	样本量	-
X	检测到的次品数	件
α	显著性水平	-
Z_α	标准正态分布的分位数	-
$\bar{x}$	实际次品率的数学期望	-
σ	次品率的标准差	-
CI	置信区间	-
$Defective\_rate\_sequence$	次品率序列	-
$Decision\_position\_frequency$	决策位频率	-
T	模拟退火温度	$^\circ C$
β	降温系数	-

表 1　符号说明

注：未在表中给出的符号均在第一次使用时定义。

五、 问题一的模型的建立和求解

5.1 模型建立与求解

Step1：假设检验背景

根据假设检验原则，我们设置两个假设：

1. **零假设（H_0）：**实际次品率小于等于标称值 $p_0(p_0 = 0.10)$。
2. **备择假设（H_1）：**实际次品率大于标称值。

为了能够在给定的置信水平下做出判断，我们还需要确定抽样数量 n，以便能够有足够的证据拒绝或者接收这些批次的零部件。

Step2：次品数的分布处理

对于单个零部件，其是正品或次品的事件满足二项分布，即：

$$X \sim B(n, p)$$

其中 X 为抽样检测中发现的次品数，n 是样本数，p 为实际次品率。当 n 足够大时，根据**中心极限定理**，二项分布 $B(n, p)$ 可以用正态分布近似：

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

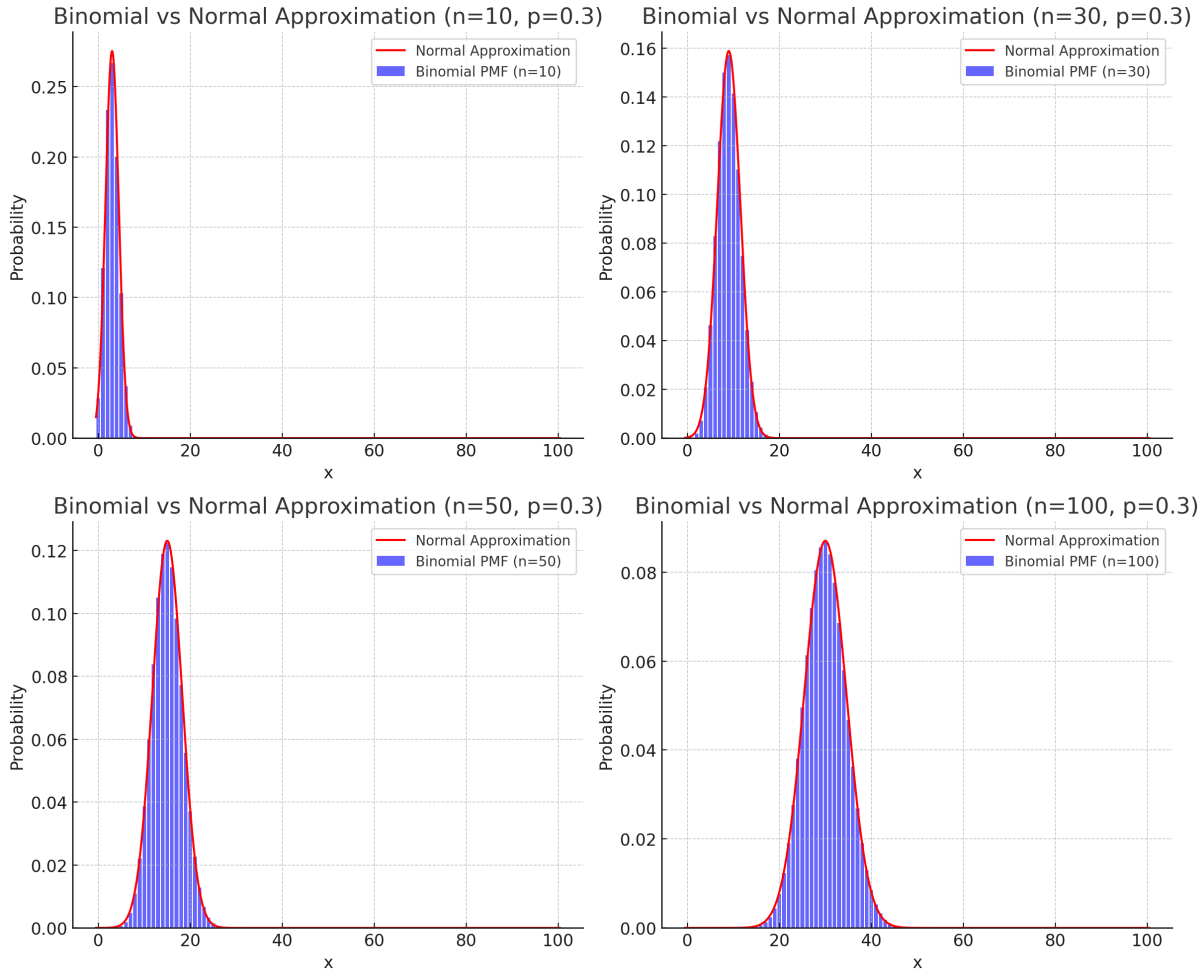


图 5　二项分布的正态近似

Step3：样本量公式推导

为了决定拒收或接收一批零配件，我们通过以下两种情况计算样本量：

（1）拒收批次（在 95% 置信水平下拒收零配件）

我们希望在 95% 置信水平下，如果次品率高于标称值 p_0，我们能拒收该批次。假设我们要检测到次品率偏离标称值 p_0 的最小偏差 δ，那么根据假设检验的原理，样本量 n 满足：

$$P(\frac{X - np_0}{\sqrt{np_0(1-p_0)}} > z_\alpha) = \alpha \tag{2}$$

其中 α 为显著性水平（0.05），Z_α 是显著性水平下的标准正态分布临界值（95% 置信水平对应 $Z_\alpha \approx 1.645$）用于假设检验。

设检测结果次品率为 p_1，根据**大数定理**，可以用 p_1 代替真实次品率，则 $\delta = p_1 - p_0$。因此检测发现的次品数 $X = np_1$，代入式 (2) 得：

$$P(\frac{np_1 - np_0}{\sqrt{np_0(1-p_0)}} > z_\alpha) = \alpha \tag{3}$$

又因为 $p_1 = p_0 + \delta$，因此有：

$$P(\frac{n\delta}{\sqrt{np_0(1-p_0)}} > z_\alpha) = \alpha \tag{4}$$

为了在 95% 的置信水平下拒绝零假设（即认为零配件次品率超过标称值），则当 $\frac{n\delta}{\sqrt{np_0(1-p_0)}} = z_\alpha$ 时，其最小偏差 δ 已经取得了最大值，此时的样本数即为最小样本数 n_{min}。如果抽检样本小于这个最小样本数，则偏差 δ 过大，计算结果不符合实际。化简上式，得样本数公式：

$$n = \frac{z_\alpha^2 p_0(1-p_0)}{\delta^2} \tag{5}$$

（2）接收批次（在 90% 置信水平下接收零配件）

类似的，在 90% 置信水平下接收零配件的条件也遵循相同的公式，与（1）的求解区别在于假设检验背景和置信水平不同，此时的假设检验背景为：

1. **零假设（H_0）：**实际次品率大于标称值 $p_0(p_0 = 0.10)$。
2. **备择假设（H_1）：**实际次品率小于等于标称值。

置信水平为 90%，采用单侧检验时 $Z_\beta = 1.28$。类似于拒收批次的情形，此时检验的是次品率的差异。设允许的最大偏差 $\delta = p_0 - p_1$，此时用检测到的次品率替代实际次品率 p，则对应的检验条件是：

$$P(\frac{n(p_0 - p_1)}{\sqrt{np_0(1-p_0)}} \leq z_\beta) = 1 - \beta \tag{6}$$

通过代入 $p_1 = p_0 - \delta$，整理得到：

$$P(\frac{n\delta}{\sqrt{np_0(1-p_0)}} \leq z_\beta) = 1 - \beta \tag{7}$$

为了保证在 90% 的置信水平下接收该批次，我们要求当 $\frac{n\delta}{\sqrt{np_0(1-p_0}} = z_\beta$ 时样本量达到最小，因此样本量计算公式为：

$$n = \frac{z_\beta^2 p_0(1-p_0)}{\delta^2} \tag{8}$$

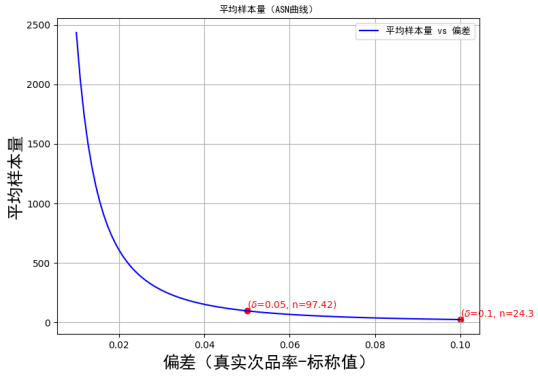
5.2 求解结果与分析

对于最小偏差 δ 的取值，其取决于企业对于零配件质量检测的容忍度，具体来说：如果企业比较看重零配件质量，则 δ 应适当小一些，增加检测次数（样本数）；如果企

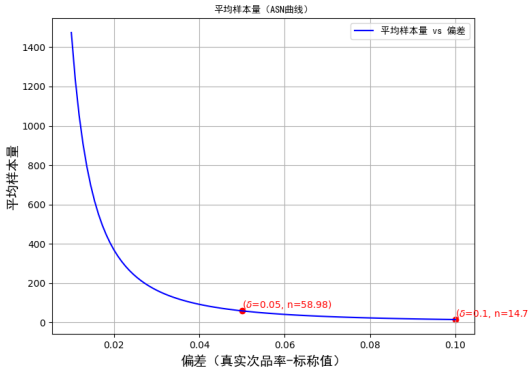
业比较希望降低成本，则 δ 应适当大一些，减少检测次数。例如，当 $\delta \in [1\%, 10\%]$ 时，对应的检测次数（样本数）如图6。

对于情形（1），当企业比较看重零配件质量，将最大偏差定为 0.05 时，得到样本量约为 97；但当企业比较看重利润，将最大偏差定为 0.1 时，得到样本量仅为 24。对于情形（2），将偏差定为 0.05 时，得到样本量约为 59；将偏差允许为 0.1 时，得到样本量仅为 15。

图 6　问题一求解结果



(a) 情形（1）平均样本曲线



(b) 情形（2）平均样本曲线

六、 问题二的模型的建立和求解

6.1 模型建立与求解

Step1：确定变量和参量

我们定义了多个参量来刻画不同的成本和收益，包括：

- P_{parts1}：零配件 1 的次品率；
- P_{parts2}：零配件 2 的次品率；
- P_{final}：成品的次品率；
- $final\_defect$：实际的成品次品率；
- C_{parts1}：零配件 1 的购买成本；
- C_{parts2}：零配件 2 的购买成本；
- C_{assemble}：装配成本；
- D_{parts1}：零配件 1 的检测成本；
- D_{parts2}：零配件 2 的检测成本；
- D_{final}：成品的检测成本；
- L_{replace}：调换损失；
- $D_{\text{disassemble}}$：拆解费用；

- Market_price：市场售价。

同时定义了以下变量代表我们在每一阶段的决策：

- $xp1, xp2$：表示是否对零配件 1 或零配件 2 进行检测，取值为 0 或 1；
- xf：表示是否对成品进行检测，取值为 0 或 1；
- xd：表示是否对不合格成品进行拆解，取值为 0 或 1。

Step2：构建目标函数

为了衡量决策的功效性和合理性，我们构建了利润目标函数对我们的决策进行评判。其计算公式为：

$$Profit = TotalIncome - TotalCost \tag{9}$$

其中总收入（Total Income）为:

$$TotalIncome = Market\_Price \times (1 - P_{final}) \tag{10}$$

总成本（Total Cost）包括以下几部分：

1. **零配件购买成本：** $C_{parts1} + C_{parts2}$。无论后续是否检测零配件，其基本购买成本固定。
2. **零配件检测成本：** 如果对零配件不进行检测，则检测成本为 0；如果对零配件进行检测，则检测成本为零配件检测费用与**更换损失** $Lose$ 之和。对于更换损失，我们以数学期望的形式给出其定义与计算公式：

   · **更换损失 (**$Lose$**)：** 进行检测的零配件会存在一定数量的次品，我们需要将其丢弃，同时购进新的零配件用以组装，在这个过程中构建新的零配件所产生的费用的期望即为更换损失，计算公式为：

$$Lose = P_{parts[i]} \times C_{parts[i]}, i = 1, 2 \tag{11}$$

3. **装配成本：** 固定为 $C_{assemble}$，每件成品必须通过装配得到。
4. **平均调换损失（**$\bar{L_{replace}}$**）：** 当未检测的成品流入市场时，质量不合格的产品会被退回，此时企业必须给客户调换并承担损失，因此，调换损失产生的重要前提是成品未被检测，由此给出其计算公式：

$$\bar{L_{replace}} = P_{final} \times L_{replace}, xf = 0 \tag{12}$$

此时需要注意的是：P_{final} 并不直接等同于题目中提供的成品次品率，因为题目规定的成品次品率是指**将正品零配件装配后的产品次品率**，这是一个条件概率，事件发生的前提为“零配件均为正品”。如果将未进行检测的零部件直接用于装配，会导致实际的成品次品率提高。因此，在这里我们根据零部件的检测状态给出修正后的成品次品率。

一个成品为次品包括两种情况：(1) 装配所用的零部件至少有一个为次品；（2）装配所用的零部件均为正品，但装配结果是次品。对于情况（1），设事件 Z：“装配所用的零部件至少有一个是次品”，则其对立事件 $\bar{Z}$：“装配所用的零部件均为正品”，显然有：

$$P(\bar{Z}) = (1 - P_{parts1})(1 - P_{parts2}) \tag{13}$$

$$P(Z) = 1 - P(\bar{Z}) = 1 - (1 - P_{parts1})(1 - P_{parts2}) \tag{14}$$

对于情况（2），事件“装配所用的零部件均为正品”发生的概率即是 $P(\bar{Z})$，不难得到此时的次品率 $P_{checked}$ 为：

$$P_{checked} = P(\bar{Z}) \times P_{final} \tag{15}$$

因此，实际次品率 $final\_defect$ 为两种情况下的次品率之和，即：

$$final\_defect = P(Z) + P_{checked} \tag{16}$$

将式12中的 P_{final} 替换为 $final\_defect$ 才是实际的调换损失期望值。

5. **拆解成本（$D_{disassemble}$）：**对于检测得到的不合格品，或是用户退回的不合格品，企业可以选择是否对其进行拆解。如果不进行拆解，其拆解成本为 0；如果进行拆解，拆解成本期望 $\bar{D_{disassemble}}$ 为：

$$\bar{D_{disassemble}} = D_{disassemble} \times finnal\_defect \tag{17}$$

是否进行拆解，取决于对不合格品拆解是否会对企业生产成本有利，也就是拆解成本期望和拆解得到的**零部件 i 的价值期望**之间的大小关系，所谓零部件 i 的价值期望，指的是拆解得到的正品零部件 i 的平均成本，其计算也分两种情形：
情形（1）：零部件 i 通过检测，此时其必为正品，零部件平均成本即为其购买成本 $C_{parts[i]}$。
情形（2）：零部件 i 未被检测，则此时其平均成本是在零部件 i 为正品的情况下的购买成本，这是一个条件概率问题：在成品为次品时零部件 i 为正品的概率。设事件 A：“成品为次品”，事件 B：“零部件 i 为正品”，则事件“在成品为次品时零部件 i 为正品”的概率即为 $P(B|A)$，由条件概率计算公式有：

$$P(B|A) = \frac{P(AB)}{P(A)} \tag{18}$$

对于事件“零部件 i 为正品且成品为次品”的概率（P(AB)），可由下列样本空间图表示：

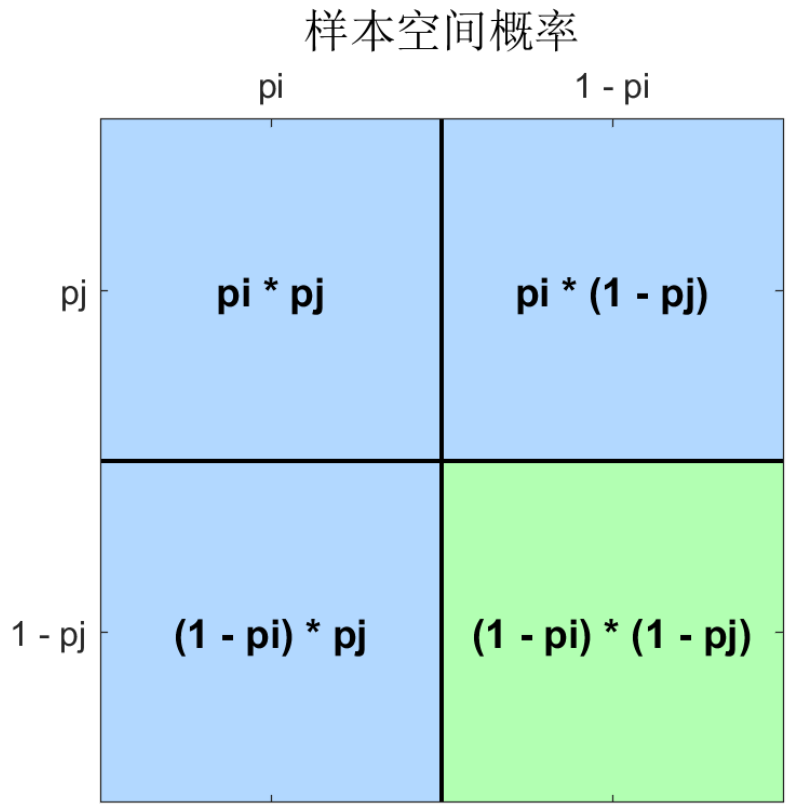


图 7　样本空间概率

只要任一零部件不合格，则成品必然不合格，即图中灰色部分；如果两个零部件均合格，其也不一定合格，不合格的占比为 P_final，因此得到：

$$P(AB) = p_i(1-p_j) + P_{final}(1-p_i)(1-p_j), i = 1,2, j = 1,2, i \neq j \tag{19}$$

$$P(A) = final\_defect \tag{20}$$

经过对以上两种情形的分析，即可得到零部件 i 的价值期望 $disassemble\_profit[i]$ 的计算公式：

$$disassemble\_profit[i] = \begin{cases} C_{parts[i]} & \text{if } xp1 = 1 \\ P(B|A) \times C_{parts[i]} & \text{if } xp1 = 0 \end{cases} \tag{21}$$

因此，是否对不合格成品进行拆解的决策取值 xd 可以直接通过计算给出：

$$xd = \begin{cases} 1 & \text{if } disassemble\_profit \geq D_{disassemble} \\ 0 & \text{else} \end{cases} \tag{22}$$

Step3：求解决策树的最优期望

通过对 $xp1, xp2, xf, xd$ 取不同的值（实际上 xd 是动态计算得到，无需遍历），并在每个阶段的决策后计算相应成本，再根据公式 (6) 将计算得到的利润期望以及相应决策存储在每个叶子结点中，最后根据**利润期望的最大值**确定整个流程的决策。

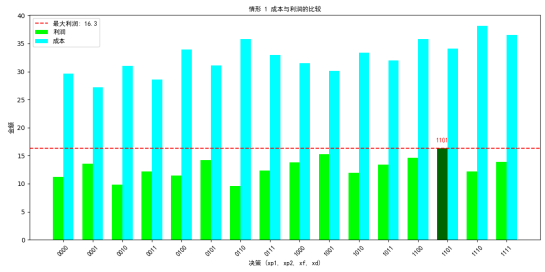
6.2 求解结果与分析

我们选取最大利润期望和相应情况下的成本为指标，对应 6 种情形下的具体决策方案如表2所示，其中 1 表示进行检测，0 表示不检测；最大利润与对应最小成本均是期望。

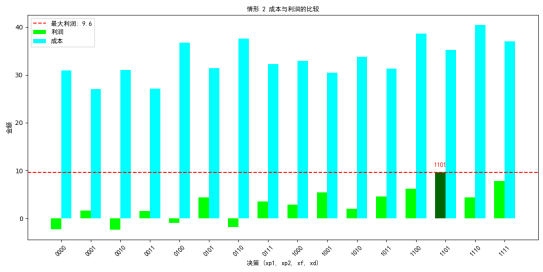
表 2　对应情形下的决策方案

情形	零配件 1 检测	零配件 2 检测	成品检测	不合格成品拆解	最大利润 (元)	对应最小成本 (元)
1	1	1	0	1	16.30	34.10
2	1	1	0	1	9.60	35.20
3	1	1	0	1	13.90	36.50
4	1	1	1	1	11.80	33.30
5	0	1	0	1	13.69	31.67
6	1	0	0	0	19.36	31.18

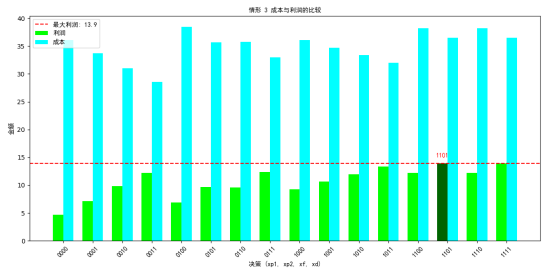
使用**穷举法**对每种情形下的不同决策进行同样的指标求解，可以验证通过我们的模型得到的决策是正确的，每种情形下的 16 种情况得到的利润如下所示：



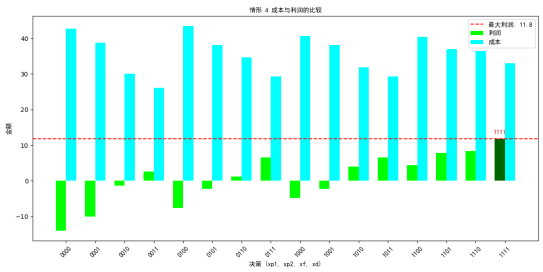
(a) 情形 1 穷举结果



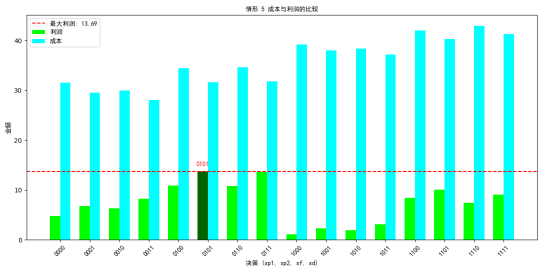
(b) 情形 2 穷举结果



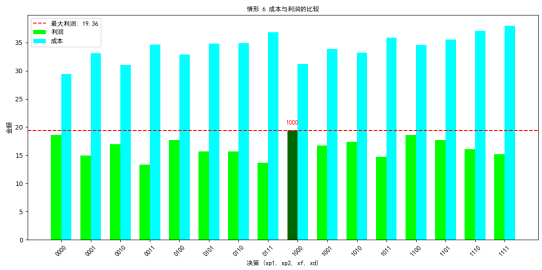
(a) 情形 3 穷举结果



(b) 情形 4 穷举结果



(a) 情形 5 穷举结果



(b) 情形 6 穷举结果

根据模型的决策结果可以得出，对于题目中提供的零配件和成品数据，大多数情况下进行零配件的检测是必要的。尤其当零配件的次品率较高时，它会显著影响成品的质量，导致成品不合格，进而造成装配费用的浪费，甚至增加售出后的调换成本。同时，如果零配件已经经过检测，而成品的次品率在修正前较低，或调换损失较小，则成品检测往往不必进行，以避免额外的成本开销，例如在情形 1、2 和 3 中。然而，在情形 4 中，调换费用过高，且零配件和成品的次品率均非常高，因此对零配件和成品进行检测都是必要的。在情形 5 中，由于零配件 1 的检测成本较高，且其次品率对成品次品率的影响不显著，因此选择不对其进行检测以获得较高利润。而在情形 6 中，由于零配件 2 的检测成本和更换损失之和较高，因此仅对零配件 2 进行了检测。

通常情况下，对于检测出的不合格成品，建议进行拆解，因为零配件的次品率通常较低，从中拆出的零配件可以再次用于生产，相比重新购买零配件更具经济性。然而，在情形 6 中，拆解费用过高，因此最终选择不进行拆解。

七、 问题三的模型的建立和求解

7.1 模型建立与求解

Step1：设计决策变量

在对问题进行降维处理后，我们需建立决策模型，其涉及的决策变量包括：

1. **零件的检验决策** x_{p_i}：针对 5 个零件分别决定是否进行检验。
2. **半成品的检验决策** x_{h_i}：针对两个半成品分别决定是否进行检验。
3. **半成品的拆解决策** x_{hd_i}：针对两个半成品分别决定是否进行拆解。
4. **成品的检验决策** xf：针对成品决定是否进行检验。
5. **成品的拆解决策** xd：针对成品决定是否进行拆解。

Step2：决策模型建立

由于问题三的求解目标也是求解最大利润期望，因此其目标函数与问题二相同，均为式9。其所包含的各部分成本和计算公式如下：

1. **零件成本** C_{parts}

$$C_{\text{parts}} = \sum (C_{p_i} + x_{p_i} \cdot D_{p_i} + x_{p_i} \cdot C_{p_i} \cdot P_{p_i}) \tag{23}$$

其中，C_{p_i} 为零件的购买成本，x_{p_i} 为检测决策，D_{p_i} 为检测成本，P_{p_i} 为次品率。

2. **半成品成本** C_{half}

$$C_{\text{half}} = \sum (C_{h_i} + x_{h_i} \cdot D_{h_i} + (x_{hd_i} \cdot D_{hd_i} - P_{hd_i}) \cdot P_{h_i}) \tag{24}$$

其中，C_{h_i} 为半成品的装配成本，x_{h_i} 为半成品检测决策，D_{h_i} 为检测成本，x_{hd_i} 为半成品拆解决策，D_{hd_i} 为拆解成本，P_{hd_i} 为拆解收益。

3. **成品装配成本** C_{assemble}

$$C_{\text{assemble}} = C_a + x_f \cdot D_f \tag{25}$$

其中，C_a 为成品装配成本，x_f 为成品检测决策，D_f 为成品检测成本。

4. **成品调换损失** C_{replace}

$$C_{\text{replace}} = (1 - x_f) \cdot L_r \cdot P_f \tag{26}$$

其中，x_f 为成品检测决策，L_r 为调换损失，P_f 为成品的次品率。

5. **成品拆解成本** $C_{\text{disassemble}}$

$$C_{\text{disassemble}} = x_d \cdot (D_d - P_d) \cdot P_f \tag{27}$$

其中，x_d 为成品拆解决策，D_d 为成品拆解成本，P_d 为拆解收益。

6. **总成本** C_{total}

$$C_{\text{total}} = C_{\text{parts}} + C_{\text{half}} + C_{\text{assemble}} + C_{\text{replace}} + C_{\text{disassemble}} \tag{28}$$

Step3：模拟退火求解最优利润期望

设定初始温度为 $1000°C$，最低温度为 $1°C$，降温系数 β 为 0.9。随后进行以下步骤：

1. **生成初始解：**随机生成所有决策变量，进行第一次求解，并将计算得到的利润记为最优利润。
2. **邻域搜索与目标函数计算：**每次从当前解的领域中随机生成一个新解，通过随机改变某个决策变量得到新的利润期望，并根据新解的利润与当前解的利润进行比较，决定是否接受新解。
3. **模拟退火：**在模拟退火过程中，使用温度 T 控制解的接受概率。若新解的利润 new_profit 高于当前解利润 $best\_profit$，则直接接受；否则当 $e^{\frac{new\_profit - best\_profit}{T}} > random$ 时才接受新解，其中 random 为一个随机数。这可以避免陷入局部最优。并且随着 T 降至阈值或满足迭代次数，**算法会逐步收敛到全局最优解**，得到最优检验与拆解决策，并计算出对应的最大利润和最小成本。

7.2 求解结果与分析

通过上述求解过程，我们得到了四种决策方案如表3，其目标函数求解得到的**利润期望均为 69.40，对应成本为 110.60**。

表 3　四种最优决策表

决策项	决策 1	决策 2	决策 3	决策 4
零件 1，4 检验	0	0	0	0
零件 2，5 检验	0	0	0	0
零件 3，6 检验	0	0	0	0
零件 7 检验	0	0	0	0
零件 8 检验	0	0	0	0
半成品 1，2 检验	1	1	1	1
半成品 1，2 拆解	0	0	1	1
半成品 3 检验	1	1	1	1
半成品 3 拆解	0	1	0	1
成品检验	0	0	0	0
成品拆解	1	1	1	1

这四种方案的共同点在于，在所有决策中，零件 1 至 8 均未进行检测，而半成品 1、半成品 2 和半成品 3 均进行了检测。成品未进行检测，但在检测出不合格时选择了拆解。方案之间的差异体现在半成品 1、2 和半成品 3 的拆解策略上。由于这四种决策的核心思想一致，即**通过控制成品次品率和选择合适的拆解时机**来最大化利润，因此当成品的次品率较高时，通过拆解半成品回收的零配件成本可能会与拆解成本相抵消。因此，即使半成品 1、2 和半成品 3 的拆解策略有所不同，仍能获得相同的利润期望和成本期望。

八、 问题四的模型的建立和求解

8.1 模型建立与求解

Step1：置信区间计算

我们认为问题二、三给出的对应的标准件的次品率即为实际次品率的数学期望 $\bar{x}$，为了尽可能得到精确的结果，经过查阅资料，我们设定其方差 $\sigma^2 = 0.05$，则标准差 $\sigma = 0.2236$，样本量 $n = 1000$，$Z_{\frac{\alpha}{2}} = 1.96$（双边 95% 置信水平），则置信区间可由下列

计算式得到：

$$CI = \left[\bar{x} - Z_{\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}},\ \bar{x} + Z_{\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{n}}\right] = \begin{cases} [0.03837, 0.06163], & \text{当 } \bar{x} = 0.05 \\ [0.08837, 0.11163], & \text{当 } \bar{x} = 0.1 \\ [0.18837, 0.21163], & \text{当 } \bar{x} = 0.2 \end{cases} \tag{29}$$

Step2：步长设定

由于数学期望为 10 分位大小，为了在保证模型时间复杂度不会过高，同时又具有一定的精度，我们采用万分之一为步长生成次品率值。步长 $\Delta p = 0.0001$, 对应的次品率序列为：

$$Defective\_rate\_sequence = [p_{min}, p_min + \Delta p, ..., p_{max}] \tag{30}$$

其中 p_{min} 和 p_{max} 分别为置信区间的下限和上限。

Step3：单因素分析法求解

为简化模型求解，采用单因素分析法：只改变某一个标准件的次品率，其他标准件的次品率保持不变。对于每个次品率值，使用问题 2 和问题 3 中的模型，计算出该标准件的最优决策。通过这种方式可以避免在多个标准件同时变化时导致的决策复杂度。

1. **独热编码与决策位频数统计：**对于每个标准件的次品率值，使用对应模型求解最优策略，并通过独热编码记录此时的决策方案，分别统计每一个决策位取 1 的次数为决策位频数，记作 $decision\_count_i$。
2. **计算决策位频率** $Decision\_position\_frequency_i$**：**对于决策位 i，其决策位频率：

$$Decision\_position\_frequency_i = decision\_count_i \times \Delta p \tag{31}$$

3. **决策位分析与决策调整：**如果某一决策位取 1 的频率明显大于 0.5，则该标准件的策略为“是”；若明显低于 0.5，则策略为“否”。对于某些接近临界值的情形（即决策位取 1 的频率接近 0.5），说明该决策位取 1 或 0 对企业利润的影响很小，此时可结合企业实际情况进一步调整决策。例如成本、市场需求等，确定是否执行该决策。

8.2 求解结果与分析

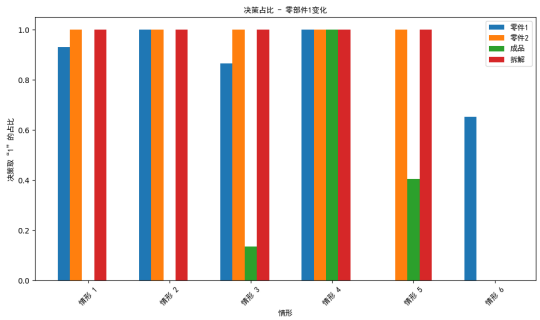
对于问题二，其 4 个决策位的对应指标为：

决策位	指标
1	零部件 1 是否检测
2	零部件 2 是否检测
3	成品是否检测
4	成品是否拆解

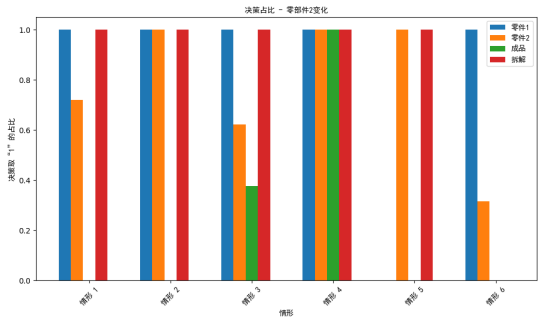
表 4　问题 2 决策位与对应指标

在此情况下，我们分别统计了在零部件 1，零部件 2，成品的次品率变化时，6 种情形下的决策位频率，具体如下图所示：

图 11　问题 2 零部件次品率变化-决策位频率



(a) 问题 2 零部件 1 次品率变化-决策位频率



(b) 问题 2 零部件 2 次品率变化-决策位频率

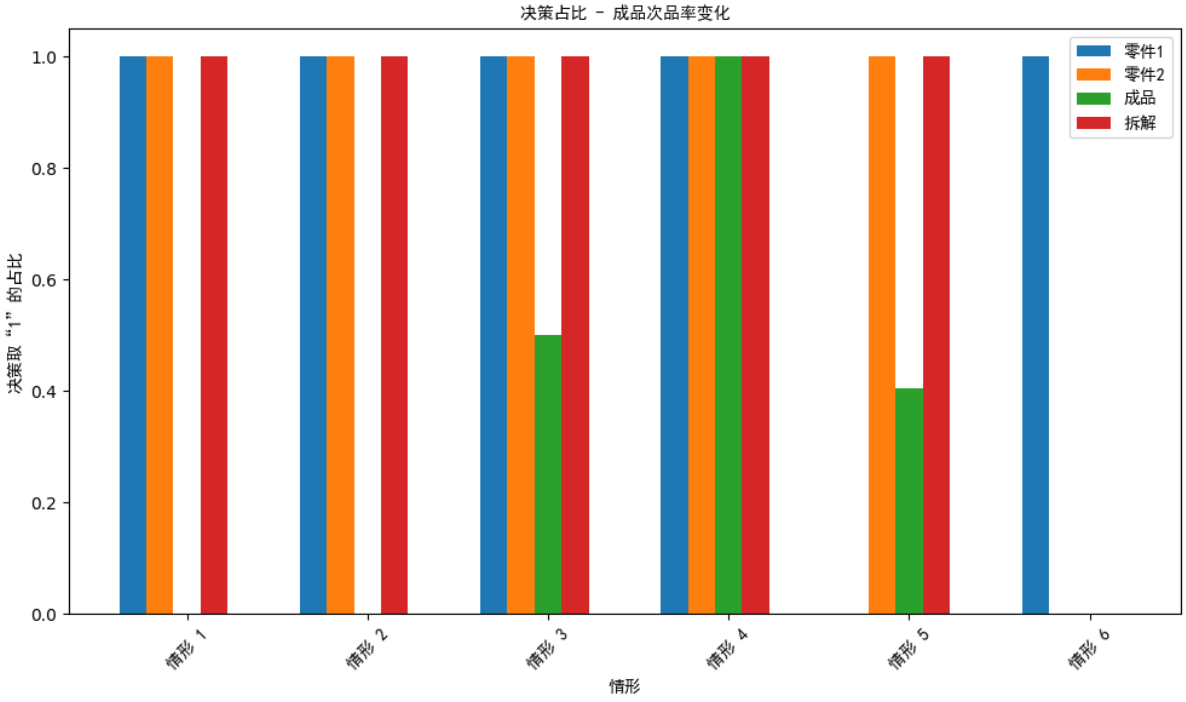


图 12　问题 2 成品次品率变化-决策位频率

我们需要给出的是每种情形下的最优决策，因此需要根据情形对标准件变化时决策位的频率进行统计，得到热力图如13所示。根据决策分析，可以很明显的得到各种情形下的相应决策，如表5所示：

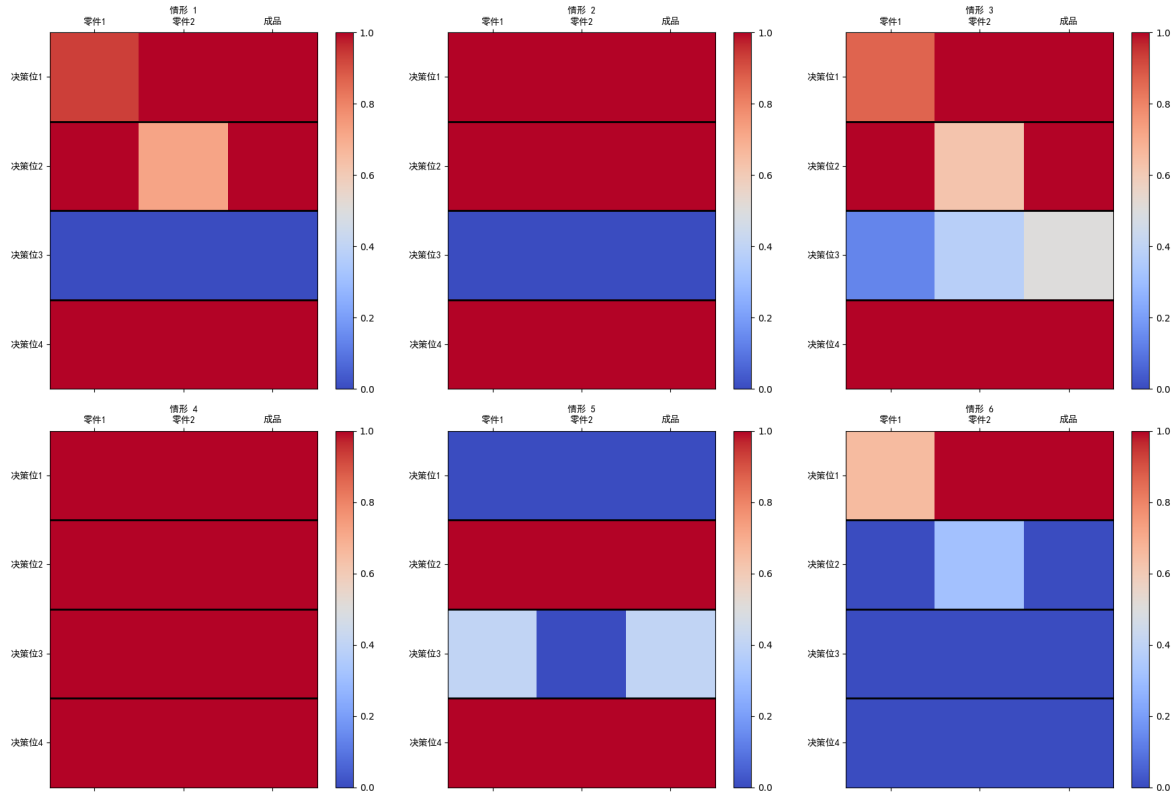


图 13　不同情形下的决策位频率热力图

表 5　问题 2 决策方案

情形	零配件 1 检测	零配件 2 检测	成品检测	不合格成品拆解
1	1	1	0	1
2	1	1	0	1
3	1	1	0	1
4	1	1	1	1
5	0	1	0	1
6	1	0	0	0

对于问题三，其 16 个决策位对应的指标如表6所示。

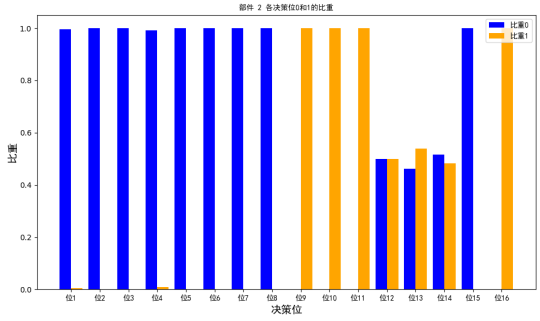
决策位	指标
1	零件 1 检测
2	零件 2 检测
3	零件 3 检测
4	零件 4 检测
5	零件 5 检测
6	零件 6 检测
7	零件 7 检测
8	零件 8 检测

决策位	指标
9	半成品 1 检测
10	半成品 2 检测
11	半成品 3 检测
12	半成品 1 拆解
13	半成品 2 拆解
14	半成品 3 拆解
15	成品检测
16	成品拆解

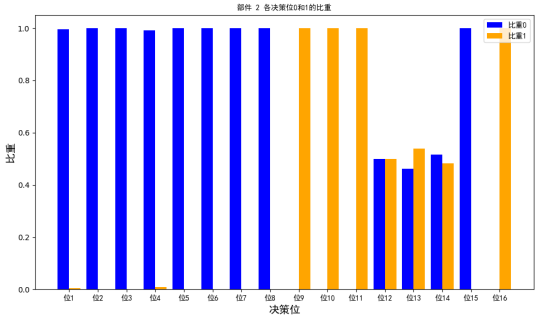
表 6　问题 3 决策位与对应指标

我们统计零部件 1-8，半成品 1-3，成品共计 12 个标准件的次品率分别变化时的决策位频率，得到零部件 1，2 的次品率分别变化时对应的决策为频率如下图所示：

图 14　问题 3 零部件次品率变化-决策位频率



(a) 问题 3 零部件 1 次品率变化-决策位频率



(b) 问题 3 零部件 2 次品率变化-决策位频率

为了给出其在任意标准件的次品率变化时的最优策略，同样做出其热力图，如图15所示。

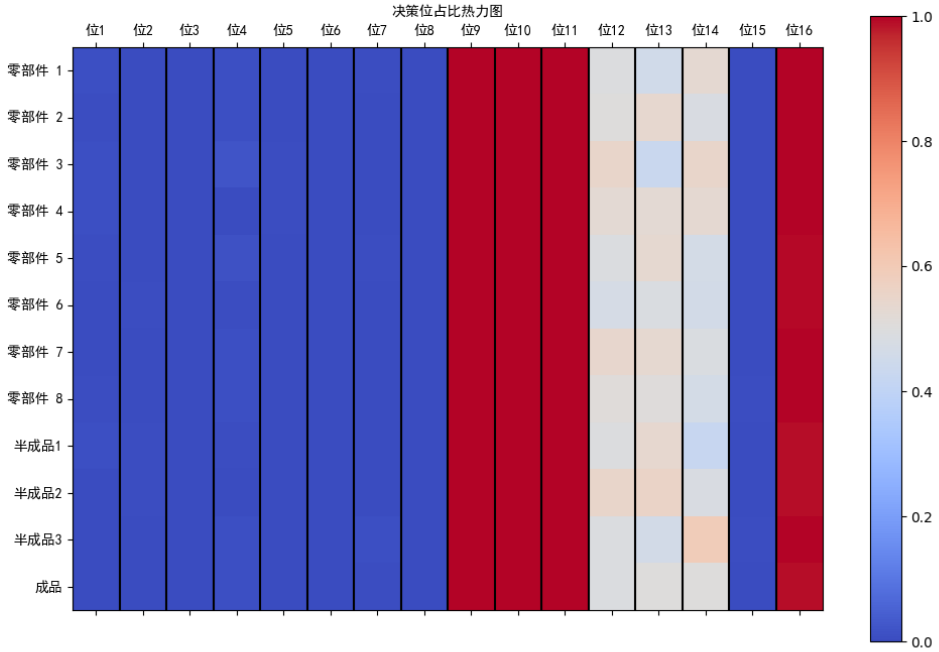


图 15　多标准件次品率变化-决策位频率热力图

对应的，我们给出问题 3 的最优决策：

表 7　问题 3 决策方案

决策项	决策
零件 1 检验	0
零件 2 检验	0
零件 3 检验	0
零件 4 检验	0
零件 5 检验	0
零件 6 检验	0
零件 7 检验	0
零件 8 检验	0

决策项	决策
半成品 1 检验	1
半成品 2 检验	1
半成品 3 检验	1
半成品 1 拆解	0/1
半成品 2 拆解	0/1
半成品 3 拆解	0/1
成品检验	0
成品拆解	1

我们发现，对于半成品 1、2、3 的拆解，其决策方案不是固定的，通过大量的计算发现，其在理想情况下：当事件发生次数足够多时，其决策的结果对于企业利润的影响非常的小，因此企业可以根据自身情况动态调整决策方案 [5]。

九、 模型的分析与检验

9.1 灵敏度分析

我们对问题三中模拟退火算法的降温系数 β 进行分析，得到对应情况下的最大利润期望和相应成本如下图所示：

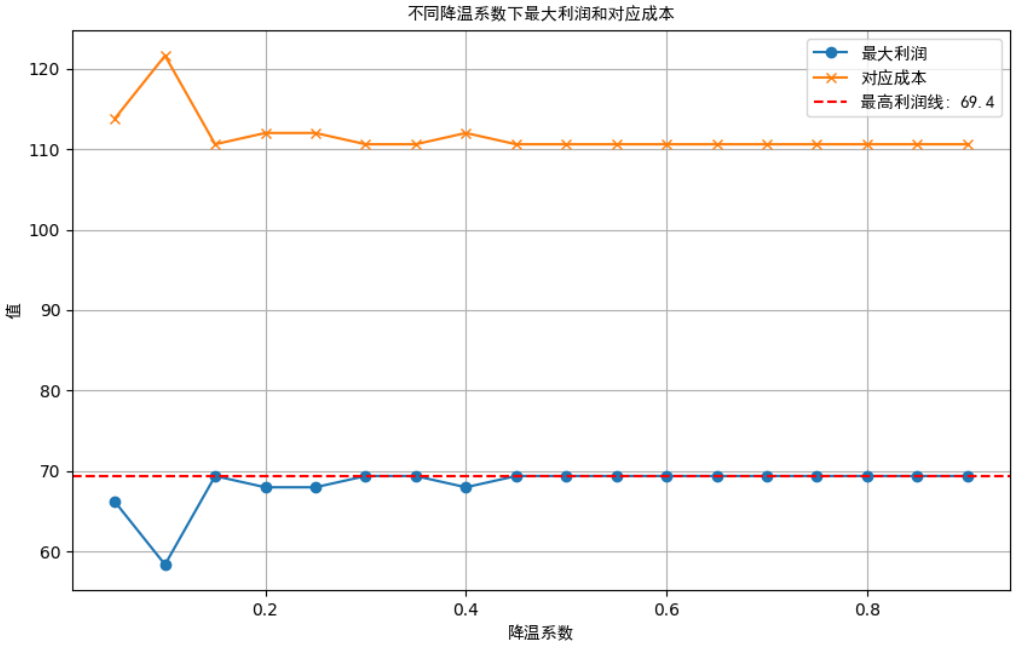


图 16　降温参数灵敏度分析

由折线图可以看出，当降温参数 $\beta \geq 0.45$ 时，最大期望利润对降温系数并不敏感，但是当 $\beta \leq 0.45$ 时，最大期望利润会随着降温系数的波动而改变。因此，为了尽可能地保证取得最优决策，我们应保证降温系数不低于 0.45。

9.2 误差分析

1. **拆解半成品/成品带来的收益：**在问题二、三中，我们考虑到对半成品/成品拆解后得到的标准件的价值，并将其看作是一部分收益，这在单次工序中是合理的。但忽略了该标准件可能多次被拆解得到，从而将其带来的收益进行了重复计算，导致成本偏低，利润偏高。
2. **单因素分析方法的系统误差：**在问题四中，当因素之间互不干扰时，采用单因素分析法不存在系统误差；但是在本题中，零配件的次品率和其装配而成的半成品或成品的次品率实际上是相互关联的，采用单因素分析法有可能导致装配而成的半成品或成品的次品率偏高或者偏低，从而影响决策。

十、 模型的评价

10.1 模型的优点

• 优点 1：降维思路简洁有效。在问题 2 中，通过对零部件和半成品的相似性进行观察，成功地将 8 个零部件降维为 5 个零部件，并简化了问题。这减少了决策变量的数量，大大降低了计算复杂度，使得模型的计算更加高效和可行。

• 优点 2：模拟退火算法的应用。使用模拟退火算法来优化问题，能够避免陷入局部最优。对于复杂的非线性问题，模拟退火能够提供较为可靠的最优解。

10.2 模型的缺点

• 缺点 1：多目标优化的缺失。当前模型的目标是最大化利润，未考虑到其他可能的重要因素，如生产效率、市场需求变化等。未来可以引入多目标优化，将多个目标函数综合考虑，以提升模型的多样性和适应性。

参考文献

[1] WANG Z, ZHANG Y. Optimization techniques in decision-making[M]. [S.l.]: Springer, 2014.

[2] 国家统计局企调总队. 专项调查知识（二）：专项调查方法简介[M/OL]. 温州市统计局, 2014. https://wztjj.wenzhou.gov.cn/art/2014/8/27/art_1243866_2907863.html.

[3] 张海龙, 陈淮莉. 在线订单拆分合并的多仓库打包决策[J/OL]. 上海海事大学学报, 2023, 44(03):100-106. DOI: 10.13340/j.jsmu.2023.03.015.

[4] BOX G E, HUNTER J S, HUNTER W G. Statistics for experimenters: Design, innovation, and discovery[M]. [S.l.]: Wiley, 2005.

[5] SIEGEL A F. Practical business statistics[M]. [S.l.]: Academic Press, 2016.

[6] MONTGOMERY D C. Introduction to statistical quality control[M]. [S.l.]: Wiley, 2013.

[7] ROSS S M. Introduction to probability models[M]. [S.l.]: Academic Press, 2009.

[8] AARTS E, KORST J. Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing[M]. [S.l.]: Wiley, 1989.

附录 A　文件列表

文件名	功能描述
q1_draw_n.py	问题一程序结果图代码
q2.py	问题二程序代码
q2 穷举可视化.py	问题二结果验证可视化代码
q2 穷举验证.py	问题二结果验证代码
q2 样本空间.py	绘制样本空间图
q3.py	问题三程序代码
q4_1.py	问题四重新解决第二问时零件 1 次品率为单变量
q4_2.py	问题四重新解决第二问时零件 2 次品率为单变量
q4_3.py	问题四重新解决第二问时成品次品率为单变量
q4_question3.py	问题四重新解决第三问代码
q4 问题 2 表的可视化.py	问题 2 表的可视化代码
q4 问题 2 图的可视化.py	问题 2 图的可视化代码
q4 问题 3 表的可视化.py	问题 3 表的可视化代码
q4 问题 3 图的可视化.py	问题 3 图的可视化代码
降温系数灵敏度检验.py	降温系数灵敏度检验代码
正态近似.py	问题一进行正态近似代码

附录 B　代码

q1_draw_n.py

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

# Define constants
z_alpha1 = 1.645  # 情形1
z_alpha2 = 1.28   # 情形2
p_0 = 0.10

```

```
# Define delta values
delta_values = np.linspace(0.01, 0.1, 100)

# Calculate n for each delta for both z_alpha1 and z_alpha2
n_values_1 = (z_alpha1**2 * p_0 * (1 - p_0)) / delta_values**2
n_values_2 = (z_alpha2**2 * p_0 * (1 - p_0)) / delta_values**2

# Recalculate n for delta = 0.05 and delta = 0.1
delta_05 = 0.05
delta_10 = 0.1
n_05_1 = (z_alpha1**2 * p_0 * (1 - p_0)) / delta_05**2
n_10_1 = (z_alpha1**2 * p_0 * (1 - p_0)) / delta_10**2
n_05_2 = (z_alpha2**2 * p_0 * (1 - p_0)) / delta_05**2
n_10_2 = (z_alpha2**2 * p_0 * (1 - p_0)) / delta_10**2

# Load SimHei font
myfont = FontProperties(fname='/Users/panlongxiang/Documents/
    数模/2024_B题/SimHei.ttf')

# Plotting delta vs n
plt.figure(figsize=(8, 6))

# Plot for z_alpha1
plt.plot(delta_values, n_values_1, label=r'情形（1）平均样本量
    曲线', color='b')
# Marking delta = 0.05 and delta = 0.1 for z_alpha1
plt.scatter([delta_05, delta_10], [n_05_1, n_10_1], color='b')
     # points
plt.text(delta_05, n_05_1 + 500, f'n={n_05_1:.2f}', fontsize
    =12, ha='left', va='bottom', color='b')
plt.text(delta_10, n_10_1 + 500, f'n={n_10_1:.2f}', fontsize
    =12, ha='left', va='bottom', color='b')

# Plot for z_alpha2
plt.plot(delta_values, n_values_2, label=r'情形（2）平均样本量
```

```
    曲线', color='r')
# Marking delta = 0.05 and delta = 0.1 for z_alpha2
plt.scatter([delta_05, delta_10], [n_05_2, n_10_2], color='r')
    # points
plt.text(delta_05, n_05_2 - 500, f'n={n_05_2:.2f}', fontsize
    =12, ha='left', va='top', color='r')
plt.text(delta_10, n_10_2 - 500, f'n={n_10_2:.2f}', fontsize
    =12, ha='left', va='top', color='r')

# Plot configurations with SimHei font
plt.xlabel(r'偏差（真实次品率-标称值）', fontsize=14,
    fontproperties=myfont)
plt.ylabel('平均样本量', fontsize=14, fontproperties=myfont)
plt.title(r'平均样本量（ASN曲线）', fontsize=16,
    fontproperties=myfont)
plt.grid(True)
plt.legend(prop=myfont)
plt.tight_layout()
plt.show()
```

q2.py

```
import numpy as np

# 定义参数
P_parts_defect1 = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]  # 零配件 1
    的次品率
P_parts_defect2 = [0.1, 0.2, 0.1, 0.2, 0.2, 0.05]  # 零配件 2
    的次品率
P_final_defect = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]   # 成品的次
    品率

C_parts1 = 4  # 零配件 1 的购买成本
C_parts2 = 18  # 零配件 2 的购买成本
D_parts1 = [2, 2, 2, 1, 8, 2]  # 零配件 1 的检测成本
D_parts2 = [3, 3, 3, 1, 1, 3]  # 零配件 2 的检测成本
```

```
C_assemble = 6  # 装配成本
D_final = [3, 3, 3, 2, 2, 3]  # 成品的检测成本
Market_price = 56  # 市场售价
L_replace = [6, 6, 30, 30, 10, 10]  # 调换损失
D_disassemble = [5, 5, 5, 5, 5, 40]  # 拆解费用

# 初始化最优结果
optimal_decision = np.zeros((6, 4), dtype=int)
max_profit = np.full(6, -np.inf)  # 用于存储最大利润
min_cost = np.full(6, np.inf)     # 用于存储最小成本（对应最大利润的方案）
xd = 0
# 成本和利润计算函数
def calculate_cost_and_profit(solution, i):
    xp1, xp2, xf, tmp1 = solution
    global xd
    # 零配件购买成本始终存在，无论是否销售
    cost_parts = C_parts1 + C_parts2

    # 如果对零配件进行检测，加入检测成本
    if xp1 == 1:
        cost_parts += D_parts1[i] + C_parts1 * P_parts_defect1[i]
        p1 = 0  # 检测后无次品
    else:
        p1 = P_parts_defect1[i]

    if xp2 == 1:
        cost_parts += D_parts2[i] + C_parts2 * P_parts_defect2[i]
        p2 = 0  # 检测后无次品
    else:
        p2 = P_parts_defect2[i]

    # 成品次品率计算
```

```
    if xp1 == 1 and xp2 == 1:
        final_defect = P_final_defect[i]
    elif xp1 == 0 and xp2 == 0:
        P_not_checked = 1 - (1 - p1) * (1 - p2)
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]
    else:
        P_not_checked = p1 if xp1 == 0 else p2
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]

    # 装配成本
    cost_assemble = C_assemble

    # 成品检测成本
    if xf == 1:
        cost_final = D_final[i]
        cost_replace = 0  # 成品检测后不需要调换
    else:
        cost_final = 0  # 未检测时没有检测成本
        cost_replace = L_replace[i] * final_defect  # 未检测成品的调换损失

    # 拆解与否只关注带来的效益，同时要乘以次品率，因为只有次品才需要拆解,条件概率问题
    dis_profit = 0
    if xp1:
        dis_profit += C_parts1
    else:
        dis_profit += (p2 * (1-p1) + P_final_defect[i] * (1-p1) * (1-p2)) / final_defect * C_parts1

    if xp2:
        dis_profit += C_parts2
    else:
```

```
        dis_profit += (p1 * (1-p2) + P_final_defect[i] * (1-p1) * (1-p2))/ final_defect * C_parts2

    if dis_profit > D_disassemble[i]:
        cost_disassemble = (D_disassemble[i] - dis_profit) * final_defect
        xd = 1
    else:
        cost_disassemble = 0
        xd = 0

    # 计算总成本（固定成本 + 可变成本）
    total_cost = cost_parts + cost_assemble + cost_final + cost_replace + cost_disassemble

    # 计算利润：总收入减去总成本
    total_income = Market_price * (1 - final_defect)  # 售卖的合格成品的收入
    profit = total_income - total_cost  # 利润

    return total_cost, profit

# 遍历每种情形，寻找最大利润方案
for i in range(6):
    dp = np.full((2, 2, 2, 2), -np.inf)  # 更新为利润最大化

    # 初始化每个状态的利润
    for xp1 in range(2):
        for xp2 in range(2):
            for xf in range(2):
                total_cost, profit = calculate_cost_and_profit([xp1, xp2, xf, xd], i)
                dp[xp1, xp2, xf, xd] = profit

                # 更新最大利润并记录对应的成本
```

```
                if profit > max_profit[i]:
                    max_profit[i] = profit
                    min_cost[i] = total_cost  # 对应最大利润的成本

    # 寻找最大利润对应的决策方案
    optimal_index = np.unravel_index(np.argmax(dp), dp.shape)
    optimal_decision[i, :] = np.array(optimal_index)

# 输出结果
for i in range(6):
    print(f'情形 {i+1} 的最优决策方案:')
    print(f'零配件1检测: {optimal_decision[i, 0]}')
    print(f'零配件2检测: {optimal_decision[i, 1]}')
    print(f'成品检测: {optimal_decision[i, 2]}')
    print(f'不合格成品拆解: {optimal_decision[i, 3]}')
    print(f'最大利润: {max_profit[i]:.2f} 元')
    print(f'对应的最小成本: {min_cost[i]:.2f} 元\n')
```

q3.py

```
import numpy as np
import random
import math

# 定义参数
P_parts_defect_1 = 0.1  # 零件1的次品率
P_parts_defect_4 = 0.1  # 零件4的次品率
P_parts_defect_2 = 0.1  # 零件2的次品率
P_parts_defect_5 = 0.1  # 零件5的次品率
P_parts_defect_3 = 0.1  # 零件3的次品率
P_parts_defect_6 = 0.1  # 零件6的次品率
P_parts_defect_7 = 0.1  # 零件7的次品率
P_parts_defect_8 = 0.1  # 零件8的次品率

P_half_defect_1 = 0.1   # 半成品1的次品率
```

```
P_half_defect_2 = 0.1    # 半成品2的次品率
P_half_defect_3 = 0.1    # 半成品3的次品率
P_final_defect = 0.1     # 成品的次品率

# 购买成本和检测成本
C_parts = [2, 8, 12, 2, 8, 12, 8, 12]  # 零件成本
D_parts = [1, 1, 2, 1, 1, 2, 1, 2]     # 零件检测成本

# 半成品的装配成本、检测成本和拆解费用
C_half = [8, 8, 8]  # 半成品装配成本
D_half = [4, 4, 4]  # 半成品检测成本
D_disassemble_half = [6, 6, 6]  # 半成品拆解费用

# 成品的装配成本、检测成本、市场售价、调换损失、拆解费用
C_assemble = 8        # 成品装配成本
D_final = 6           # 成品检测成本
Market_price = 200    # 市场售价
L_replace = 40        # 调换损失
D_disassemble_final = 10  # 成品拆解费用

# 拆解收益函数1（用于处理三个零件的拆解：半成品1和2）
def calculate_disassemble_profit_three(xp1, xp2, xp3, p1, p2,
    p3, final_defect, C_parts1, C_parts2, C_parts3,
    P_final_defect):
    # 判断 final_defect 是否为 0，若为 0 则将其设为 1，避免除
    零错误
    if final_defect == 0:
        final_defect = 1

    dis_profit = 0
    # 计算零件1的拆解收益
    if xp1:  # 零件1检测过，直接拆解
        dis_profit += C_parts1
    else:  # 未检测，使用条件概率计算拆解收益
        dis_profit += (1 - p1) * (p2 + p3 - p2 * p3 + (1 - p2)
```

```
        * (1 - p3) * 0.1) / final_defect * C_parts1
48
49      # 计算零件2的拆解收益
50      if xp2:  # 零件2检测过，直接拆解
51          dis_profit += C_parts2
52      else:  # 未检测，使用条件概率计算拆解收益
53          dis_profit += (1 - p2) * (p1 + p3 - p1 * p3 + (1 - p1)
        * (1 - p3) * 0.1) / final_defect * C_parts2
54
55      # 计算零件3的拆解收益
56      if xp3:  # 零件3检测过，直接拆解
57          dis_profit += C_parts3
58      else:  # 未检测，使用条件概率计算拆解收益
59          dis_profit += (1 - p3) * (p1 + p2 - p1 * p2 + (1 - p1)
        * (1 - p2) * 0.1) / final_defect * C_parts3
60
61      return dis_profit
62
63  def calculate_disassemble_profit_two(xp1, xp2, p1, p2,
       final_defect, C_parts1, C_parts2, P_final_defect):
64      # 判断 final_defect 是否为 0，若为 0 则将其设为 1，避免除
        零错误
65      if final_defect == 0:
66          final_defect = 1
67
68      dis_profit = 0
69      # 计算零件1的拆解收益
70      if xp1:  # 零件1检测过，直接拆解
71          dis_profit += C_parts1
72      else:  # 未检测，使用条件概率计算拆解收益
73          dis_profit += (1 - p1) * p2 / final_defect * C_parts1
74
75      # 计算零件2的拆解收益
76      if xp2:  # 零件2检测过，直接拆解
77          dis_profit += C_parts2
```

```
78      else:  # 未检测，使用条件概率计算拆解收益
79          dis_profit += (1 - p2) * p1 / final_defect * C_parts2
80
81      return dis_profit
82
83  # 计算成品的拆解收益，将成品视为由三个半成品组成
84  def calculate_final_disassemble_profit(xh1, xh2, xh3, p1, p2,
    p3, final_defect, C_half1, C_half2, C_half3, P_final_defect)
    :
85      # 计算成品的拆解收益，使用半成品的成本（包括零件成本和装配
    成本）及次品率
86      return calculate_disassemble_profit_three(xh1, xh2, xh3,
    p1, p2, p3, final_defect, C_half1, C_half2, C_half3,
    P_final_defect)
87
88  # 成本和利润计算函数
89  def calculate_cost_and_profit(solution):
90      # 解的结构：零件检测（5个独立决策，分别对应1=4，2=5，3=6，
    7，8），半成品检测+拆解（2个同步决策控制半成品1=2，半成品3），
    成品检测+拆解
91      xp = solution[:5]  # 零件检测决策（1=4，2=5，3=6，7，8）
92      xh_detect = solution[5:7]  # 半成品检测决策（1个同步决策控
    制半成品1=2，半成品3）
93      xh_disassemble = solution[7:9]  # 半成品拆解决策（1个同步
    决策控制半成品1=2，半成品3）
94      xf_detect = solution[9]    # 成品检测决策
95      xf_disassemble = solution[10]  # 成品拆解决策
96
97      # 零件次品率处理：如果检测了，次品率设为0；否则保持原有次
    品率
98      p_parts_defect_1 = 0 if xp[0] == 1 else P_parts_defect_1
    # 零件1
99      p_parts_defect_4 = 0 if xp[0] == 1 else P_parts_defect_4
    # 零件4（同步决策）
100     p_parts_defect_2 = 0 if xp[1] == 1 else P_parts_defect_2
```

```
    # 零件2
101  p_parts_defect_5 = 0 if xp[1] == 1 else P_parts_defect_5
    # 零件5（同步决策）
102  p_parts_defect_3 = 0 if xp[2] == 1 else P_parts_defect_3
    # 零件3
103  p_parts_defect_6 = 0 if xp[2] == 1 else P_parts_defect_6
    # 零件6（同步决策）
104  p_parts_defect_7 = 0 if xp[3] == 1 else P_parts_defect_7
    # 零件7（独立）
105  p_parts_defect_8 = 0 if xp[4] == 1 else P_parts_defect_8
    # 零件8（独立）
106
107  # 半成品次品率处理：半成品1和2的决策相同
108  p_half_defect_1 = (1 - (1 - p_parts_defect_1) * (1 -
    p_parts_defect_2) * (1 - p_parts_defect_3)) + \
109                    (1 - p_parts_defect_1) * (1 -
    p_parts_defect_2) * (1 - p_parts_defect_3) * P_half_defect_1
110
111  p_half_defect_2 = (1 - (1 - p_parts_defect_1) * (1 -
    p_parts_defect_2) * (1 - p_parts_defect_3)) + \
112                    (1 - p_parts_defect_1) * (1 -
    p_parts_defect_2) * (1 - p_parts_defect_3) * P_half_defect_2
113
114  p_half_defect_3 = (1 - (1 - p_parts_defect_7) * (1 -
    p_parts_defect_8)) + \
115                    (1 - p_parts_defect_7) * (1 -
    p_parts_defect_8) * P_half_defect_3
116
117  # 根据检测决策更新半成品次品率
118  p_half_defect_1 = 0 if xh_detect[0] == 1 else
    p_half_defect_1
119  p_half_defect_2 = 0 if xh_detect[0] == 1 else
    p_half_defect_2  # 决策同步
120  p_half_defect_3 = 0 if xh_detect[1] == 1 else
    p_half_defect_3
```

```
121
122     # 成品次品率处理：如果检测了，次品率设为0；否则根据半成品次品率计算
123     p_final_defect = (1 - (1 - p_half_defect_1) * (1 - p_half_defect_2) * (1 - p_half_defect_3)) + \
124                      (1 - p_half_defect_1) * (1 - p_half_defect_2) * (1 - p_half_defect_3) * P_final_defect
125
126     # 计算零件的购买和检测成本
127     cost_parts = (C_parts[0] + xp[0] * D_parts[0] + xp[0] * C_parts[0] * 0.1) * 2 + \
128                  (C_parts[1] + xp[1] * D_parts[1] + xp[1] * C_parts[1] * 0.1) * 2 + \
129                  (C_parts[2] + xp[2] * D_parts[2] + xp[2] * C_parts[2] * 0.1) * 2 + \
130                  (C_parts[6] + xp[3] * D_parts[6] + xp[3] * C_parts[6] * 0.1) + \
131                  (C_parts[7] + xp[4] * D_parts[7] + xp[4] * C_parts[7] * 0.1)
132
133
134     # 计算拆解收益（半成品1=2使用三个零件，半成品3使用两个零件）
135     dis_profit_three = calculate_disassemble_profit_three(xp[0], xp[1], xp[2], p_half_defect_1, p_half_defect_2, p_half_defect_3, p_final_defect, C_parts[0], C_parts[1], C_parts[2], P_final_defect)
136     dis_profit_two = calculate_disassemble_profit_two(xp[3], xp[4], p_half_defect_3, p_half_defect_3, p_final_defect, C_parts[6], C_parts[7], P_final_defect)
137
138     # 半成品的装配、检测和（拆解成本-拆解收益）
139     cost_half = sum(
140         [C_half[0] + xh_detect[0] * D_half[0] + (xh_disassemble[0] * D_disassemble_half[0] - dis_profit_three
```

```
    ) * p_half_defect_1,  # 半成品1=2
141         C_half[1] + xh_detect[0] * D_half[1] +(xh_disassemble
    [0] * D_disassemble_half[1] - dis_profit_three) *
    p_half_defect_1,  # 半成品2
142         C_half[2] + xh_detect[1] * D_half[2] + (
    xh_disassemble[1] * D_disassemble_half[2] - dis_profit_two)
    * p_half_defect_3])  # 半成品3
143
144     # 计算成品的拆解收益
145     final_disassemble_profit =
    calculate_final_disassemble_profit(xh_detect[0], xh_detect
    [0], xh_detect[1], p_half_defect_1, p_half_defect_2,
    p_half_defect_3, p_final_defect, C_half[0], C_half[1],
    C_half[2], P_final_defect)
146
147     # 判断成品是否拆解
148     if xf_disassemble:
149         cost_disassemble_final = (D_disassemble_final -
    final_disassemble_profit) * p_final_defect
150     else:
151         cost_disassemble_final = 0
152
153     # 成品的装配、检测和拆解成本
154     cost_assemble = C_assemble + xf_detect * D_final
155     cost_replace = 0 if xf_detect == 1 else L_replace *
    p_final_defect  # 调换损失
156
157     # 计算总成本
158     total_cost = cost_parts + cost_half + cost_assemble +
    cost_replace + cost_disassemble_final
159     # 计算售价
160     total_income = Market_price * (1 - p_final_defect)
161
162     profit = total_income - total_cost
163     return total_cost, profit
```

```
164
165
166 # 模拟退火算法实现
167 def simulated_annealing():
168     # 初始化解（随机决策，11个决策变量：5个零件检测决策，半成品1=2检测+拆解决策，半成品3检测+拆解决策，成品检测+拆解决策）
169     current_solution = np.random.randint(2, size=11)  # 初始化包含零件、半成品和成品的决策
170     best_solution = current_solution.copy()
171     best_profit = calculate_cost_and_profit(current_solution)[1]
172     best_cost = calculate_cost_and_profit(current_solution)[0]
173     T = 100 # 初始温度
174     T_min = 1  # 最低温度
175     beta =  0.9 # 降温系数
176
177     while T > T_min:
178         for _ in range(50):
179             # 生成邻域解（随机改变一个决策变量）
180             new_solution = current_solution.copy()
181             idx = random.randint(0, 10)  # 随机改变1个决策变量
182             new_solution[idx] = 1 - new_solution[idx]  # 改变该决策变量
183
184             # 计算新解的利润
185             new_cost, new_profit = calculate_cost_and_profit(new_solution)
186
187             # 接受新解的概率
188             if new_profit > best_profit or random.random() < math.exp((new_profit - best_profit) / T):
189                 current_solution = new_solution.copy()
190                 if new_profit > best_profit:
191                     best_solution = new_solution.copy()
```

```
                    best_profit = new_profit
                    best_cost = new_cost

        T *= beta  # 降低温度

    return best_solution, best_profit, best_cost


# 运行模拟退火算法
best_solution, best_profit, best_cost = simulated_annealing()

# 输出结果，包括零件、半成品和成品的决策
print(f"最优决策方案: 零件检测决策1=4={best_solution[0]}, 2=5={best_solution[1]}, 3=6={best_solution[2]}, 零件7={best_solution[3]}, 零件8={best_solution[4]}")
print(f"半成品检测决策1=2检测={best_solution[5]}, 拆解={best_solution[7]}, 半成品3检测={best_solution[6]}, 拆解={best_solution[8]}")
print(f"成品检测决策={best_solution[9]}, 成品拆解决策={best_solution[10]}")
print(f"最大利润: {best_profit:.2f} 元")
print(f"对应成本: {best_cost:.2f} 元")
```

q4_1.py

```
import numpy as np
import csv

# 定义零件1的次品率的95%置信区间
confidence_intervals_parts1 = [
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.03837, 0.06163)   # 对应5%次品率
```

```
]
# 定义参数
P_parts_defect2 = [0.1, 0.2, 0.1, 0.2, 0.2, 0.05]  # 零配件2的次品率（不变）
P_final_defect = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]  # 成品的次品率
C_parts1 = 4  # 零配件1的购买成本
C_parts2 = 18  # 零配件2的购买成本
D_parts1 = [2, 2, 2, 1, 8, 2]  # 零配件1的检测成本
D_parts2 = [3, 3, 3, 1, 1, 3]  # 零配件2的检测成本
C_assemble = 6  # 装配成本
D_final = [3, 3, 3, 2, 2, 3]  # 成品的检测成本
Market_price = 56  # 市场售价
L_replace = [6, 6, 30, 30, 10, 10]  # 调换损失
D_disassemble = [5, 5, 5, 5, 5, 40]  # 拆解费用


# 成本和利润计算函数，包含成品拆解决策
def calculate_cost_and_profit(solution, i, P_parts_defect1):
    xp1, xp2, xf, xd = solution  # 包含拆解决策（xd）
    global disassemble_profit
    # 零配件购买成本始终存在，无论是否销售
    cost_parts = C_parts1 + C_parts2

    # 如果对零配件1进行检测，加入检测成本
    if xp1 == 1:
        cost_parts += D_parts1[i] + C_parts1 * P_parts_defect1
        p1 = 0  # 检测后无次品
    else:
        p1 = P_parts_defect1

    # 如果对零配件2进行检测，加入检测成本
    if xp2 == 1:
        cost_parts += D_parts2[i] + C_parts2 * P_parts_defect2
    [i]
```

```
        p2 = 0  # 检测后无次品
    else:
        p2 = P_parts_defect2[i]

    # 成品次品率计算
    if xp1 == 1 and xp2 == 1:
        final_defect = P_final_defect[i]
    elif xp1 == 0 and xp2 == 0:
        P_not_checked = 1 - (1 - p1) * (1 - p2)
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]
    else:
        P_not_checked = p1 if xp1 == 0 else p2
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]

    # 装配成本
    cost_assemble = C_assemble

    # 成品检测成本
    if xf == 1:
        cost_final = D_final[i]
        cost_replace = 0  # 成品检测后不需要调换
    else:
        cost_final = 0  # 未检测时没有检测成本
        cost_replace = L_replace[i] * final_defect  # 未检测成品的调换损失

    # 计算拆解收益
    disassemble_profit = 0
    if xp1:
        disassemble_profit += C_parts1
    else:
        disassemble_profit += (p2 * (1 - p1) + P_final_defect[i] * (1 - p1) * (1 - p2)) / final_defect * C_parts1
```

```
75
76      if xp2:
77          disassemble_profit += C_parts2
78      else:
79          disassemble_profit += (p1 * (1 - p2) + P_final_defect[
    i] * (1 - p1) * (1 - p2)) / final_defect * C_parts2
80
81      # 判断是否拆解成品
82      if xd == 1 and disassemble_profit > D_disassemble[i]:
83          cost_disassemble = (D_disassemble[i] -
    disassemble_profit) * final_defect
84      else:
85          cost_disassemble = 0
86
87      # 计算总成本（固定成本 + 可变成本）
88      total_cost = cost_parts + cost_assemble + cost_final +
    cost_replace + cost_disassemble
89
90      # 计算利润：总收入减去总成本
91      total_income = Market_price * (1 - final_defect)  # 售卖的
    合格成品的收入
92      profit = total_income - total_cost  # 利润
93
94      return total_cost, profit
95
96  delta = 0.1
97  while delta >= 0.0000001:
98      # 遍历每种情形，将零件1的置信区间等分为400份，并使用区间中
    点作为次品率
99      detection_results = []
100     for i, (lower_bound, upper_bound) in enumerate(
    confidence_intervals_parts1):
101         step_size = delta
102         samples = np.array([lower_bound + j * step_size for j
    in range(int((upper_bound - lower_bound) / delta))])
```

```
103
104         # 初始化检测次数统计
105         detection_count_1 = 0  # 零件1
106         detection_count_2 = 0  # 零件2
107         detection_count_f = 0  # 成品
108         detection_count_d = 0  # 成品拆解
109
110         # 遍历所有可能的次品率
111         for sample in samples:
112             # 枚举所有检测和拆解组合，并选择利润最大的一种
113             max_profit = -np.inf
114             best_solution = None
115
116             for xp1 in range(2):  # 零件1是否检测
117                 for xp2 in range(2):  # 零件2是否检测
118                     for xf in range(2):  # 成品是否检测
119                         for xd in range(2):  # 成品是否拆解
120                             total_cost, profit =
    calculate_cost_and_profit([xp1, xp2, xf, xd], i, sample)
121                             if profit > max_profit:
122                                 max_profit = profit
123                                 best_solution = [xp1, xp2, xf,
     xd]
124
125             # 统计检测和拆解次数
126             if best_solution[0] == 1:
127                 detection_count_1 += 1
128             if best_solution[1] == 1:
129                 detection_count_2 += 1
130             if best_solution[2] == 1:
131                 detection_count_f += 1
132             if best_solution[3] == 1:
133                 detection_count_d += 1
134
135         # 将统计结果添加到结果列表中
```

```
        detection_results.append([detection_count_1, detection_count_2, detection_count_f, detection_count_d])

    # 将检测结果写入CSV文件
    csv_file = f"data_parts1_{delta}.csv"
    with open(csv_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['circumstance', 'part1', 'part2', 'final_product', 'disassemble'])
        for i, result in enumerate(detection_results):
            writer.writerow([f'circumstance {i + 1}', result[0], result[1], result[2], result[3]])

    # 输出完成信息
    print(f"检测结果已保存到 {csv_file}")
    delta /= 10
```

q4_2.py

```
import numpy as np
import csv

# 定义零件2的次品率的95%置信区间
confidence_intervals_parts2 = [
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.03837, 0.06163)   # 对应5%次品率
]

# 定义参数
P_parts_defect1 = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]  # 零配件1的次品率（不变）
P_final_defect = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]  # 成品的次品
```

```
    率
C_parts1 = 4  # 零配件1的购买成本
C_parts2 = 18  # 零配件2的购买成本
D_parts1 = [2, 2, 2, 1, 8, 2]  # 零配件1的检测成本
D_parts2 = [3, 3, 3, 1, 1, 3]  # 零配件2的检测成本
C_assemble = 6  # 装配成本
D_final = [3, 3, 3, 2, 2, 3]  # 成品的检测成本
Market_price = 56  # 市场售价
L_replace = [6, 6, 30, 30, 10, 10]  # 调换损失
D_disassemble = [5, 5, 5, 5, 5, 40]  # 拆解费用


# 成本和利润计算函数，包含成品拆解决策
def calculate_cost_and_profit(solution, i, P_parts_defect2):
    xp1, xp2, xf, xd = solution  # 包含拆解决策（xd）
    global disassemble_profit
    # 零配件购买成本始终存在，无论是否销售
    cost_parts = C_parts1 + C_parts2

    # 如果对零配件1进行检测，加入检测成本
    if xp1 == 1:
        cost_parts += D_parts1[i] + C_parts1 * P_parts_defect1
    [i]
        p1 = 0  # 检测后无次品
    else:
        p1 = P_parts_defect1[i]

    # 如果对零配件2进行检测，加入检测成本
    if xp2 == 1:
        cost_parts += D_parts2[i] + C_parts2 * P_parts_defect2
        p2 = 0  # 检测后无次品
    else:
        p2 = P_parts_defect2

    # 成品次品率计算
```

```
    if xp1 == 1 and xp2 == 1:
        final_defect = P_final_defect[i]
    elif xp1 == 0 and xp2 == 0:
        P_not_checked = 1 - (1 - p1) * (1 - p2)
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]
    else:
        P_not_checked = p1 if xp1 == 0 else p2
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect[i]

    # 装配成本
    cost_assemble = C_assemble

    # 成品检测成本
    if xf == 1:
        cost_final = D_final[i]
        cost_replace = 0  # 成品检测后不需要调换
    else:
        cost_final = 0  # 未检测时没有检测成本
        cost_replace = L_replace[i] * final_defect  # 未检测成品的调换损失

    # 计算拆解收益
    disassemble_profit = 0
    if xp1:
        disassemble_profit += C_parts1
    else:
        disassemble_profit += (p2 * (1 - p1) + P_final_defect[i] * (1 - p1) * (1 - p2)) / final_defect * C_parts1

    if xp2:
        disassemble_profit += C_parts2
    else:
        disassemble_profit += (p1 * (1 - p2) + P_final_defect[
```

```
    i] * (1 - p1) * (1 - p2)) / final_defect * C_parts2
81
82      # 判断是否拆解成品
83      if xd == 1 and disassemble_profit > D_disassemble[i]:
84          cost_disassemble = (D_disassemble[i] -
    disassemble_profit) * final_defect
85      else:
86          cost_disassemble = 0
87
88      # 计算总成本（固定成本 + 可变成本）
89      total_cost = cost_parts + cost_assemble + cost_final +
    cost_replace + cost_disassemble
90
91      # 计算利润：总收入减去总成本
92      total_income = Market_price * (1 - final_defect)  # 售卖的
    合格成品的收入
93      profit = total_income - total_cost  # 利润
94
95      return total_cost, profit
96
97
98  # 遍历每种情形，将零件2的置信区间等分为400份，并使用区间中点作
    为次品率
99  delta = 0.1
100 while delta >= 0.000001:
101     detection_results = []
102     for i, (lower_bound, upper_bound) in enumerate(
    confidence_intervals_parts2):
103         # 将置信区间等分成400份，并使用每个区间的中点
104         step_size = (upper_bound - lower_bound) / 400
105         samples = np.array([lower_bound + (j + 0.5) *
    step_size for j in range(400)])
106
107         # 初始化检测次数统计
108         detection_count_1 = 0  # 零件1
```

```
        detection_count_2 = 0  # 零件2
        detection_count_f = 0  # 成品
        detection_count_d = 0  # 成品拆解

        # 遍历所有可能的次品率
        for sample in samples:
            # 枚举所有检测和拆解组合，并选择利润最大的一种
            max_profit = -np.inf
            best_solution = None

            for xp1 in range(2):  # 零件1是否检测
                for xp2 in range(2):  # 零件2是否检测
                    for xf in range(2):  # 成品是否检测
                        for xd in range(2):  # 成品是否拆解
                            total_cost, profit =
    calculate_cost_and_profit([xp1, xp2, xf, xd], i, sample)
                            if profit > max_profit:
                                max_profit = profit
                                best_solution = [xp1, xp2, xf,
     xd]

            # 统计检测和拆解次数
            if best_solution[0] == 1:
                detection_count_1 += 1
            if best_solution[1] == 1:
                detection_count_2 += 1
            if best_solution[2] == 1:
                detection_count_f += 1
            if best_solution[3] == 1:
                detection_count_d += 1

        # 将统计结果添加到结果列表中
        detection_results.append([detection_count_1,
    detection_count_2, detection_count_f, detection_count_d])

```

```
    # 将检测结果写入CSV文件
    csv_file = f"data_parts2_{delta}.csv"
    with open(csv_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['circumstance', 'part1', 'part2', '
  final_product', 'disassemble'])
        for i, result in enumerate(detection_results):
            writer.writerow([f'circumstance {i + 1}', result
  [0], result[1], result[2], result[3]])

    # 输出完成信息
    print(f"检测结果已保存到 {csv_file}")
    delta /= 10
```

q4_3.py

```
import numpy as np
import csv

# 定义成品次品率的95%置信区间
confidence_intervals_parts2 = [
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.18837, 0.21163),  # 对应20%次品率
    (0.08837,0.11163),  # 对应10%次品率
    (0.03837, 0.06163)   # 对应5%次品率
]

# 定义参数
P_parts_defect1 = [0.1, 0.2, 0.1, 0.2, 0.1, 0.05]  # 零配件1的
  次品率
P_parts_defect2 = [0.1, 0.2, 0.1, 0.2, 0.2, 0.05]  # 零配件2的
  次品率
C_parts1 = 4  # 零配件1的购买成本
C_parts2 = 18  # 零配件2的购买成本
```

```
D_parts1 = [2, 2, 2, 1, 8, 2]  # 零配件1的检测成本
D_parts2 = [3, 3, 3, 1, 1, 3]  # 零配件2的检测成本
C_assemble = 6  # 装配成本
D_final = [3, 3, 3, 2, 2, 3]  # 成品的检测成本
Market_price = 56  # 市场售价
L_replace = [6, 6, 30, 30, 10, 10]  # 调换损失
D_disassemble = [5, 5, 5, 5, 5, 40]  # 拆解费用


# 成本和利润计算函数，包含成品拆解决策
def calculate_cost_and_profit(solution, i, P_final_defect):
    xp1, xp2, xf, xd = solution  # 包含拆解决策（xd）
    global disassemble_profit
    # 零配件购买成本始终存在，无论是否销售
    cost_parts = C_parts1 + C_parts2

    # 如果对零配件1进行检测，加入检测成本
    if xp1 == 1:
        cost_parts += D_parts1[i] + C_parts1 * P_parts_defect1[i]
        p1 = 0  # 检测后无次品
    else:
        p1 = P_parts_defect1[i]

    # 如果对零配件2进行检测，加入检测成本
    if xp2 == 1:
        cost_parts += D_parts2[i] + C_parts2 * P_parts_defect2[i]
        p2 = 0  # 检测后无次品
    else:
        p2 = P_parts_defect2[i]

    # 成品次品率计算
    if xp1 == 1 and xp2 == 1:
        final_defect = P_final_defect
```

```
    elif xp1 == 0 and xp2 == 0:
        P_not_checked = 1 - (1 - p1) * (1 - p2)
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect
    else:
        P_not_checked = p1 if xp1 == 0 else p2
        final_defect = P_not_checked + (1 - P_not_checked) * P_final_defect

    # 装配成本
    cost_assemble = C_assemble

    # 成品检测成本
    if xf == 1:
        cost_final = D_final[i]
        cost_replace = 0  # 成品检测后不需要调换
    else:
        cost_final = 0  # 未检测时没有检测成本
        cost_replace = L_replace[i] * final_defect  # 未检测成品的调换损失

    # 计算拆解收益
    disassemble_profit = 0
    if xp1:
        disassemble_profit += C_parts1
    else:
        disassemble_profit += (p2 * (1 - p1) + P_final_defect * (1 - p1) * (1 - p2)) / final_defect * C_parts1

    if xp2:
        disassemble_profit += C_parts2
    else:
        disassemble_profit += (p1 * (1 - p2) + P_final_defect * (1 - p1) * (1 - p2)) / final_defect * C_parts2

```

```
    # 判断是否拆解成品
    if xd == 1 and disassemble_profit > D_disassemble[i]:
        cost_disassemble = (D_disassemble[i] - disassemble_profit) * final_defect
    else:
        cost_disassemble = 0

    # 计算总成本（固定成本 + 可变成本）
    total_cost = cost_parts + cost_assemble + cost_final + cost_replace + cost_disassemble

    # 计算利润：总收入减去总成本
    total_income = Market_price * (1 - final_defect)  # 售卖的合格成品的收入
    profit = total_income - total_cost  # 利润

    return total_cost, profit


# 遍历每种情形，将成品次品率的置信区间等分为400份，并使用区间中点作为次品率
detection_results = []
for i, (lower_bound, upper_bound) in enumerate(confidence_intervals_final):
    # 将置信区间等分成400份，并使用每个区间的中点
    step_size = (upper_bound - lower_bound) / 400
    samples = np.array([lower_bound + (j + 0.5) * step_size for j in range(400)])

    # 初始化检测次数统计
    detection_count_1 = 0  # 零件1
    detection_count_2 = 0  # 零件2
    detection_count_f = 0  # 成品
    detection_count_d = 0  # 成品拆解

```

```
    # 遍历所有可能的次品率
    for sample in samples:
        # 枚举所有检测和拆解组合，并选择利润最大的一种
        max_profit = -np.inf
        best_solution = None

        for xp1 in range(2):  # 零件1是否检测
            for xp2 in range(2):  # 零件2是否检测
                for xf in range(2):  # 成品是否检测
                    for xd in range(2):  # 成品是否拆解
                        total_cost, profit =
    calculate_cost_and_profit([xp1, xp2, xf, xd], i, sample)
                        if profit > max_profit:
                            max_profit = profit
                            best_solution = [xp1, xp2, xf, xd]

        # 统计检测和拆解次数
        if best_solution[0] == 1:
            detection_count_1 += 1
        if best_solution[1] == 1:
            detection_count_2 += 1
        if best_solution[2] == 1:
            detection_count_f += 1
        if best_solution[3] == 1:
            detection_count_d += 1

    # 将统计结果添加到结果列表中
    detection_results.append([detection_count_1,
    detection_count_2, detection_count_f, detection_count_d])

# 将检测结果写入CSV文件
csv_file = "data_final_product.csv"
with open(csv_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['circumstance', 'part1', 'part2', '
```

```
    final_product', 'disassemble'])
144     for i, result in enumerate(detection_results):
145         writer.writerow([f'circumstance {i + 1}', result[0],
    result[1], result[2], result[3]])
146
147 # 输出完成信息
148 print(f"检测结果已保存到 {csv_file}")
```

q4_question3.py

```
1  import numpy as np
2  import random
3  import math
4
5  # 定义参数
6  confidence_interval = (0.08837, 0.11163) #抽样概率的置信区间
7  # 定义参数
8  confidence_interval = (0.08837, 0.11163)  # 抽样概率的置信区间
9  P_parts_defect = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,
       0.1, 0.1, 0.1]  # 各零件初始次品率
10
11 # 购买成本和检测成本
12 C_parts = [2, 8, 12, 2, 8, 12, 8, 12]  # 零件成本
13 D_parts = [1, 1, 2, 1, 1, 2, 1, 2]    # 零件检测成本
14
15 # 半成品的装配成本、检测成本和拆解费用
16 C_half = [8, 8, 8]  # 半成品装配成本
17 D_half = [4, 4, 4]  # 半成品检测成本
18 D_disassemble_half = [6, 6, 6]  # 半成品拆解费用
19
20 # 成品的装配成本、检测成本、市场售价、调换损失、拆解费用
21 C_assemble = 8        # 成品装配成本
22 D_final = 6          # 成品检测成本
23 Market_price = 200   # 市场售价
24 L_replace = 40       # 调换损失
25 D_disassemble_final = 10  # 成品拆解费用
```

```
26
27  # 拆解收益函数1（用于处理三个零件的拆解：半成品1和2）
28  def calculate_disassemble_profit_three(xp1, xp2, xp3, p1, p2,
        p3, final_defect, C_parts1, C_parts2, C_parts3,
        P_final_defect):
29      # 判断 final_defect 是否为 0，若为 0 则将其设为 1，避免除
        零错误
30      if final_defect == 0:
31          final_defect = 1
32
33      dis_profit = 0
34      # 计算零件1的拆解收益
35      if xp1:  # 零件1检测过，直接拆解
36          dis_profit += C_parts1
37      else:  # 未检测，使用条件概率计算拆解收益
38          dis_profit += (1 - p1) * (p2 + p3 - p2 * p3 + (1 - p2)
        * (1 - p3) * 0.1) / final_defect * C_parts1
39
40      # 计算零件2的拆解收益
41      if xp2:  # 零件2检测过，直接拆解
42          dis_profit += C_parts2
43      else:  # 未检测，使用条件概率计算拆解收益
44          dis_profit += (1 - p2) * (p1 + p3 - p1 * p3 + (1 - p1)
        * (1 - p3) * 0.1) / final_defect * C_parts2
45
46      # 计算零件3的拆解收益
47      if xp3:  # 零件3检测过，直接拆解
48          dis_profit += C_parts3
49      else:  # 未检测，使用条件概率计算拆解收益
50          dis_profit += (1 - p3) * (p1 + p2 - p1 * p2 + (1 - p1)
        * (1 - p2) * 0.1) / final_defect * C_parts3
51
52      return dis_profit
53
54  def calculate_disassemble_profit_two(xp1, xp2, p1, p2,
```

```
    final_defect, C_parts1, C_parts2, P_final_defect):
55      # 判断 final_defect 是否为 0，若为 0 则将其设为 1，避免除
    零错误
56      if final_defect == 0:
57          final_defect = 1
58
59      dis_profit = 0
60      # 计算零件1的拆解收益
61      if xp1:  # 零件1检测过，直接拆解
62          dis_profit += C_parts1
63      else:  # 未检测，使用条件概率计算拆解收益
64          dis_profit += (1 - p1) * p2 / final_defect * C_parts1
65
66      # 计算零件2的拆解收益
67      if xp2:  # 零件2检测过，直接拆解
68          dis_profit += C_parts2
69      else:  # 未检测，使用条件概率计算拆解收益
70          dis_profit += (1 - p2) * p1 / final_defect * C_parts2
71
72      return dis_profit
73
74  # 计算成品的拆解收益，将成品视为由三个半成品组成
75  def calculate_final_disassemble_profit(xh1, xh2, xh3, p1, p2,
    p3, final_defect, C_half1, C_half2, C_half3, P_final_defect)
    :
76      # 计算成品的拆解收益，使用半成品的成本（包括零件成本和装配
    成本）及次品率
77      return calculate_disassemble_profit_three(xh1, xh2, xh3,
    p1, p2, p3, final_defect, C_half1, C_half2, C_half3,
    P_final_defect)
78
79  # 成本和利润计算函数
80  def calculate_cost_and_profit(solution):
81      # 解的结构：零件检测（8个独立决策，分别对应1，2，3，4，5，
    6，7，8），半成品检测+拆解（半成品1，2,3），成品检测+拆解
```

```
    xp = solution[:8]  # 零件检测决策 (1=4, 2=5, 3=6, 7, 8)
    xh_detect = solution[8:11]  # 半成品检测决策 (1个同步决策控制半成品1=2, 半成品3)
    xh_disassemble = solution[11:14]  # 半成品拆解决策 (1个同步决策控制半成品1=2, 半成品3)
    xf_detect = solution[14]    # 成品检测决策
    xf_disassemble = solution[15]  # 成品拆解决策

    # 零件次品率处理: 如果检测了, 次品率设为0; 否则保持原有次品率
    p_parts_defect_1 = 0 if xp[0] == 1 else P_parts_defect[0]  # 零件1
    p_parts_defect_4 = 0 if xp[1] == 1 else P_parts_defect[1]  # 零件4 (同步决策)
    p_parts_defect_2 = 0 if xp[2] == 1 else P_parts_defect[2]  # 零件2
    p_parts_defect_5 = 0 if xp[3] == 1 else P_parts_defect[3]  # 零件5 (同步决策)
    p_parts_defect_3 = 0 if xp[4] == 1 else P_parts_defect[4]  # 零件3
    p_parts_defect_6 = 0 if xp[5] == 1 else P_parts_defect[5]  # 零件6 (同步决策)
    p_parts_defect_7 = 0 if xp[6] == 1 else P_parts_defect[6]  # 零件7 (独立)
    p_parts_defect_8 = 0 if xp[7] == 1 else P_parts_defect[7]  # 零件8 (独立)

    # 半成品次品率处理: 半成品1和2的决策相同
    p_half_defect_1 = (1 - (1 - p_parts_defect_1) * (1 - p_parts_defect_2) * (1 - p_parts_defect_3)) + \
                      (1 - p_parts_defect_1) * (1 - p_parts_defect_2) * (1 - p_parts_defect_3) * P_parts_defect[8]

    p_half_defect_2 = (1 - (1 - p_parts_defect_4) * (1 -
```

```
    p_parts_defect_5) * (1 - p_parts_defect_6)) + \
103                     (1 - p_parts_defect_4) * (1 -
    p_parts_defect_5) * (1 - p_parts_defect_6) * P_parts_defect
    [9]
104
105  p_half_defect_3 = (1 - (1 - p_parts_defect_7) * (1 -
    p_parts_defect_8)) + \
106                     (1 - p_parts_defect_7) * (1 -
    p_parts_defect_8) * P_parts_defect[10]
107
108  # 根据检测决策更新半成品次品率
109  p_half_defect_1 = 0 if xh_detect[0] == 1 else
    p_half_defect_1
110  p_half_defect_2 = 0 if xh_detect[1] == 1 else
    p_half_defect_2
111  p_half_defect_3 = 0 if xh_detect[2] == 1 else
    p_half_defect_3
112
113  # 成品次品率处理：如果检测了，次品率设为0；否则根据半成品
    次品率计算
114  p_final_defect = (1 - (1 - p_half_defect_1) * (1 -
    p_half_defect_2) * (1 - p_half_defect_3)) + \
115                   (1 - p_half_defect_1) * (1 -
    p_half_defect_2) * (1 - p_half_defect_3) * P_parts_defect
    [11]
116
117  # 计算零件的购买和检测成本
118  cost_parts = (C_parts[0] + xp[0] * D_parts[0] + xp[0] *
    C_parts[0] * P_parts_defect[0]) + \
119               (C_parts[1] + xp[1] * D_parts[1] + xp[1] *
    C_parts[1] * P_parts_defect[1]) + \
120               (C_parts[2] + xp[2] * D_parts[2] + xp[2] *
    C_parts[2] * P_parts_defect[2]) + \
121               (C_parts[3] + xp[3] * D_parts[3] + xp[3] *
    C_parts[3] * P_parts_defect[3]) + \
```

```
122                 (C_parts[4] + xp[4] * D_parts[4] + xp[4] *
    C_parts[4] * P_parts_defect[4]) + \
123                 (C_parts[5] + xp[5] * D_parts[5] + xp[5] *
    C_parts[5] * P_parts_defect[5]) + \
124                 (C_parts[6] + xp[6] * D_parts[6] + xp[6] *
    C_parts[6] * P_parts_defect[6]) + \
125                 (C_parts[7] + xp[7] * D_parts[7] + xp[7] *
    C_parts[7] * P_parts_defect[7])
126
127
128     # 计算拆解收益（半成品1=2使用三个零件，半成品3使用两个零
    件）
129     dis_profit_three_1 = calculate_disassemble_profit_three(xp
    [0], xp[1], xp[2], p_parts_defect_1, p_parts_defect_2,
    p_parts_defect_3, p_half_defect_1, C_parts[0], C_parts[1],
    C_parts[2], P_parts_defect[11])
130     dis_profit_three_2 = calculate_disassemble_profit_three(xp
    [3], xp[4], xp[5], p_parts_defect_3, p_parts_defect_4,
    p_parts_defect_5, p_half_defect_2, C_parts[3], C_parts[4],
    C_parts[5], P_parts_defect[11])
131     dis_profit_two = calculate_disassemble_profit_two(xp[6],
    xp[7], p_parts_defect_6, p_parts_defect_7, p_half_defect_3,
    C_parts[6], C_parts[7], P_parts_defect[11])
132
133     # 半成品的装配、检测和（拆解成本-拆解收益）
134     cost_half = sum(
135         [C_half[0] + xh_detect[0] * D_half[0] + (
    xh_disassemble[0] * D_disassemble_half[0] -
    dis_profit_three_1) * p_half_defect_1,  # 半成品1=2
136          C_half[1] + xh_detect[1] * D_half[1] +(xh_disassemble
    [1] * D_disassemble_half[1] - dis_profit_three_2) *
    p_half_defect_2,  # 半成品2
137          C_half[2] + xh_detect[2] * D_half[2] + (
    xh_disassemble[2] * D_disassemble_half[2] - dis_profit_two)
    * p_half_defect_3])  # 半成品3
```

```
138
139     # 计算成品的拆解收益
140     final_disassemble_profit =
    calculate_final_disassemble_profit(xh_detect[0], xh_detect
    [1], xh_detect[2], p_half_defect_1, p_half_defect_2,
    p_half_defect_3, p_final_defect, C_half[0], C_half[1],
    C_half[2], P_parts_defect[11])
141
142     # 判断成品是否拆
143     if xf_disassemble:
144         cost_disassemble_final = (D_disassemble_final -
    final_disassemble_profit) * p_final_defect
145     else:
146         cost_disassemble_final = 0
147
148     # 成品的装配、检测和拆解成本
149     cost_assemble = C_assemble + xf_detect * D_final
150     cost_replace = 0 if xf_detect == 1 else L_replace *
    p_final_defect  # 调换损失
151
152     # 计算总成本
153     total_cost = cost_parts + cost_half + cost_assemble +
    cost_replace + cost_disassemble_final
154     # 计算售价
155     total_income = Market_price * (1 - p_final_defect)
156
157     profit = total_income - total_cost
158     return total_cost, profit
159
160
161 # 模拟退火算法实现
162 def simulated_annealing():
163     # 初始化解（随机决策，16个决策变量：8个零件检测决策，半成
    品1检测+拆解决策，半成品2检测+拆解决策，半成品3检测+拆解决策
    ，成品检测+拆解决策）
```

```
    current_solution = np.random.randint(2, size=16)  # 初始化包含零件、半成品和成品的决策
    best_solution = current_solution.copy()
    best_profit = calculate_cost_and_profit(current_solution)[1]
    best_cost = calculate_cost_and_profit(current_solution)[0]
    T = 1000  # 初始温度
    T_min = 1  # 最低温度
    alpha = 0.9  # 降温系数

    while T > T_min:
        for _ in range(100):
            # 生成邻域解（随机改变一个决策变量）
            new_solution = current_solution.copy()
            idx = random.randint(0, 15)  # 随机改变1个决策变量
            new_solution[idx] = 1 - new_solution[idx]  # 改变该决策变量

            # 计算新解的利润
            new_cost, new_profit = calculate_cost_and_profit(new_solution)

            # 接受新解的概率
            if new_profit > best_profit or random.random() < math.exp((new_profit - best_profit) / T):
                current_solution = new_solution.copy()
                if new_profit > best_profit:
                    best_solution = new_solution.copy()
                    best_profit = new_profit
                    best_cost = new_cost

        T *= alpha  # 降低温度

    return best_solution, best_profit, best_cost

```

```
194
195 # 函数：将决策转换为文字说明
196 def get_decision_description(solution):
197     part_desc = [
198         f"零件1检测：{'是' if solution[0] == 1 else '否'}",
199         f"零件2检测：{'是' if solution[1] == 1 else '否'}",
200         f"零件3检测：{'是' if solution[2] == 1 else '否'}",
201         f"零件4检测：{'是' if solution[3] == 1 else '否'}",
202         f"零件5检测：{'是' if solution[4] == 1 else '否'}",
203         f"零件6检测：{'是' if solution[5] == 1 else '否'}",
204         f"零件7检测：{'是' if solution[6] == 1 else '否'}",
205         f"零件8检测：{'是' if solution[7] == 1 else '否'}"
206     ]
207     half_part_desc = [
208         f"半成品1检测：{'是' if solution[8] == 1 else '否'}",
209         f"半成品2检测：{'是' if solution[10] == 1 else '否'}",
210         f"半成品3检测：{'是' if solution[12] == 1 else '否'}",
211         f"半成品1拆解：{'是' if solution[9] == 1 else '否'}",
212         f"半成品2拆解：{'是' if solution[11] == 1 else '否'}",
213         f"半成品3拆解：{'是' if solution[13] == 1 else '否'}",
214         f"成品检测：{'是' if solution[14] == 1 else '否'}",
215         f"成品拆解：{'是' if solution[15] == 1 else '否'}"
216     ]
217     return part_desc + half_part_desc
218
219 # 函数：枚举当前部件的次品率，并保持其他部件的次品率不变
220 def explore_defect_rate_for_part(part_index,
    confidence_interval, resolution=0.0001):
221     """
222     对某个部件的次品率进行枚举，固定其他部件次品率，调用模拟退
    火算法寻找最优决策。
223     :param part_index: 当前部件的索引。
224     :param confidence_interval: 当前部件次品率的置信区间。
225     :param resolution: 次品率变化步长，默认为万分位。
226     """
```

```
    lower_bound, upper_bound = confidence_interval
    defect_rates = np.arange(lower_bound, upper_bound + resolution, resolution)

    # 保存每个次品率下的最优决策结果
    decision_results = []
    global P_parts_defect
    # 遍历次品率
    for defect_rate in defect_rates:
        # 设置当前部件的次品率
        P_parts_defect[part_index] = defect_rate

        # 运行模拟退火算法，计算最优解
        best_solution, best_profit, best_cost = simulated_annealing()

        # 保存结果
        decision_results.append({
            'defect_rate': defect_rate,
            'best_solution': best_solution,
            'best_profit': best_profit,
            'best_cost': best_cost
        })

        # 以16位二进制格式输出最佳方案
        binary_solution = ''.join(map(str, best_solution))
        decision_desc = get_decision_description(best_solution)

    return decision_results

# 函数：按决策方案对结果排序
def sort_by_decision(decision_results):
    """
    按决策方案排序，统计出现频率最多的决策组合。
```

```
    :param decision_results: 决策结果列表。
    """
    decision_counts = {}

    for result in decision_results:
        decision_tuple = tuple(result['best_solution'])
        if decision_tuple in decision_counts:
            decision_counts[decision_tuple] += 1
        else:
            decision_counts[decision_tuple] = 1

    # 按频率排序
    sorted_decisions = sorted(decision_counts.items(), key=lambda x: x[1], reverse=True)

    return sorted_decisions

# 主函数：依次对每个部件次品率进行探索
for part_idx in range(12):
    print(f"\n==== 探索部件 {part_idx + 1} 的次品率 ====")
    part_results = explore_defect_rate_for_part(part_idx, confidence_interval)

    # 对决策方案进行排序
    sorted_decisions = sort_by_decision(part_results)

    print(f"\n部件 {part_idx + 1} 的次品率下，不同决策方案的出现频率排序：")
    for decision, count in sorted_decisions:
        # 将决策以16位二进制数显示
        binary_decision = ''.join(map(str, decision))
        print(f"决策 {binary_decision} 出现次数：{count}")
```

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

```python
from matplotlib.font_manager import FontProperties

# 加载SimHei字体
myfont = FontProperties(fname='/Users/panlongxiang/Documents/数模/2024_B题/SimHei.ttf')

# 决策数据统计结果，包含所有部件的决策数据（根据您的数据添加完整内容）
results = {
    "部件 1": [
        "0000000011100001", "0000000011110101", "0000000011111101",
        "0000000011100101", "0000000011101101", "0000000011101001",
        "0000000011110001", "0000000011111001", "1000000011111101",
        "0001000011100001", "1000000011110101", "0001001011110101",
        "0000000011111100"
    ],
    "部件 2": [
        "0000000011101001", "0000000011111101", "0000000011110001",
        "0000000011101101", "0000000011100001", "0000000011110101",
        "0000000011111001", "0000000011100101", "0001000011111101",
        "1000000011101101", "0001000011101101"
    ],
    "部件 3": [
        "0000000011110101", "0000000011111101", "0000000011100001",
        "0000000011110001", "0000000011100101", "0000000011101101",
        "0000000011111001", "0000000011101001", "
```

```
    0001000011100101",
27          "1000000011100001", "0000100011100101", "
    0001000011101101",
28          "0001000011111001"
29      ],
30      # 添加部件 4 到部件 12 的决策数据...
31  }
32
33  # 出现次数，按部件分别列出
34  frequency = {
35      "部件 1": [36, 35, 29, 29, 27, 25, 24, 24, 1, 1, 1, 1],
36      "部件 2": [36, 34, 33, 31, 30, 27, 22, 18, 1, 1, 1],
37      "部件 3": [36, 36, 32, 32, 27, 24, 23, 16, 3, 2, 1, 1, 1],
38      # 添加部件 4 到部件 12 的出现次数...
39  }
40
41  # 计算每个位的0和1的比重
42  def calculate_bit_ratios(results, frequencies):
43      ratios = np.zeros((16, 2))  # 每个位的0和1的比例
44      total_count = sum(frequencies)  # 总的决策次数
45
46      for decision, freq in zip(results, frequencies):
47          for i, bit in enumerate(decision):
48              if bit == "0":
49                  ratios[i, 0] += freq  # 统计0的次数
50              else:
51                  ratios[i, 1] += freq  # 统计1的次数
52
53      # 计算比例
54      ratios = ratios / total_count
55      return ratios
56
57  # 绘制柱状图函数
58  def plot_bit_ratios(part_name, ratios):
59      fig, ax = plt.subplots(figsize=(10, 6))
```

```
    indices = np.arange(1, 17)  # 决策位从1到16

    # 绘制每个位的0和1比重
    ax.bar(indices - 0.2, ratios[:, 0], width=0.4, label='比重0', color='blue')
    ax.bar(indices + 0.2, ratios[:, 1], width=0.4, label='比重1', color='orange')

    # 设置标题和标签，并加载中文字体
    ax.set_xlabel("决策位", fontsize=14, fontproperties=myfont)
    ax.set_ylabel("比重", fontsize=14, fontproperties=myfont)
    ax.set_title(f"{part_name} 各决策位0和1的比重", fontsize=16, fontproperties=myfont)

    # 添加图例
    ax.legend(prop=myfont)

    # 设置x轴刻度
    ax.set_xticks(indices)
    ax.set_xticklabels([f"位{i}" for i in range(1, 17)], fontproperties=myfont)

    # 显示图表
    plt.tight_layout()
    plt.savefig(f"part{part_name}变化决策位占比图.png")
    plt.show()

# 遍历每个部件，生成其决策比重的柱状图
for part_name, decisions in results.items():
    frequencies = frequency[part_name]  # 对应的出现次数
    ratios = calculate_bit_ratios(decisions, frequencies)  # 计算每个位的比重
    plot_bit_ratios(part_name, ratios)  # 绘制柱状图
```

注：未附注的代码均在支撑文件中给出。