基于最优决策模型的工厂生产利润优化研究

摘　要

本文基于某电子产品生产企业的实际生产流程，结合概率论和最优决策模型，解决了在生产环节中零配件是否检测、不合格成品处理等问题。通过假设检验方法优化抽样检测，评估不同检测策略的成本与收益，最终构建了最大化企业利润的生产决策模型。

问题一解决了企业如何通过抽样检测判断供应商零配件的次品率是否达标的问题。应用了基于假设检验的模型，结合**二项分布**和**正态分布**近似方法，计算了在 95%和 90%置信水平下的样本量和次品数临界值。在 95%置信水平下，需抽样 **139** 件，若检测出次品数目超过 **20** 件则拒收；在 90%置信水平下，需抽样 **98** 件，若检测出次品数目低于 **14** 件则接收。这种方法减少了检测成本，提高了企业对供应商零配件质量判断的置信度和统计准确性。

问题二解决了企业在生产过程中如何决策检测零配件和成品、处理不合格品的策略问题。具体包括是否检测零配件、成品，是否拆解不合格成品，以及如何处理退回的不合格品等四个阶段的决策。结合 **Bayes** 定理，本文遍历评估了 **16 种决策组合**，计算出每种组合的总成本和收益，以确定最优生产策略。模型还通过收益直方图分析了不同情况下检测的必要性，指出在高次品率和高调换损失时加强检测的重要性，而在低次品率或高检测成本时减少检测的经济效益。结果显示，通过采用最优策略，在**情况 1 到 6** 中，本文得到的最优收益分别为 **18.50，14.00，16.10，12.80，15.77 和 19.70** 元，同时给出每种情况下的最优方案。

问题三涉及为一个包含多道工序、零配件和半成品的生产流程设计最优决策方案。以最大化总利润为目标，决策依据包括零配件的次品率、检测成本、半成品和成品的装配与拆解成本，以及市场售价。基于**模拟退火算法**，综合考虑企业利益和产品合格率，建立了一个与概率论相结合的**多阶段决策最优模型**。经过多次迭代优化，最优策略为：**所有零配件选择“不检测”，半成品选择“检测”和“拆解”，成品选择“不检测”和“拆解”。** 通过减少不必要的检测环节，将质量控制放在半成品这一中间阶段，能有效提高产品合格率和总利润。

在问题四中，企业的电子产品次品率通过抽样检测得到。基于问题一中采用的抽样检测方法，对零配件、半成品和成品的次品率进行了动态更新，重新评估问题二和问题三所建立的模型。这样可以更准确地反映生产过程中次品率的波动，并基于这些更新的数据进行决策优化。与固定次品率的原始模型相比，问题四的动态更新策略提高了决策的灵活性，更好地适应了生产中不确定性。

关键词：抽样检测　多阶段决策　Bayes 定理　模拟退火模型　二项分布

一、 问题重述

1.1 背景分析

随着电子产品制造业的快速发展，生产企业面临着两大挑战：一是最优化成本，提高收益。二是，提高产品合格率，保持良好的企业信誉。为了在两者中找到合适的平衡点，企业会选择在生产过程中对各零配件进行抽样检测，同时对检测不合格的产品会予以合理处理（丢弃或拆解回收）。为了控制成本和保证产品质量，企业需要决定在什么情况下对零配件进行检测，从而使得收益最高的同时产品质量得到保证，已经成为现代制造业的重要研究课题。

1.2 问题提出

问题一：某供应商声称一批零配件（零配件 1 或零配件 2）的次品率不会超过某个标称值。为了决定是否接收供应商的这批零配件，企业计划通过抽样检测进行评估，且检测费用由企业承担。要求为企业设计一个抽样检测方案，使得检测次数尽可能少。

设标称次品率为 10%，根据设计方案，请分别回答以下两种情况：

(1) 在 95%的置信水平下，认定零配件的次品率超过标称值，则拒收该批次零配件。
(2) 在 90%的置信水平下，认定零配件的次品率不超过标称值，则接收该批次零配件。

问题二：假设已知两种零配件和成品的次品率，企业需要在生产过程的各个阶段作出如下决策：

(1) 是否对零配件（零配件 1 和/或零配件 2）进行检测？如果不检测，则该零配件直接进入装配环节；如果检测，则将不合格的零配件丢弃。
(2) 是否对每件装配好的成品进行检测？如果不检测，则成品直接进入市场；如果检测，只有检测合格的成品才能进入市场。
(3) 是否对检测出的不合格成品进行拆解？如果不拆解，则直接丢弃不合格成品；如果拆解，则将拆解出的零配件重新用于生产，并重复步骤(1)和步骤(2)。
(4) 对于用户购买的有缺陷产品，企业需无条件更换，并产生一定的更换损失（如物流成本、企业信誉等）。对于退回的不合格品，企业应重复步骤(3)。

请根据上述决策，对表 1 中所提供的情形给出具体的决策方案，并提供决策依据及相应的关键指标。

问题三：假设生产过程包括 m 道工序和 n 个零配件，已知每个零配件、半成品和成品的次品率，要求类似于问题 2，设计出生产过程的决策方案。图 1 展示了 2 道工序和 8 个零配件的情形，具体数据由表 2 给出。

请根据表 2 中的数据，给出详细的决策方案，并提供决策依据及相应的关键指标。

问题四：假设问题 2 和问题 3 中提到的零配件、半成品和成品的次品率均是通过抽样检测方法（例如，问题 1 中的方法）得到的，请重新解决问题 2 和问题 3，提出新的决策方案。

二、 问题分析

2.1 问题一的分析

在问题一中，企业需要通过抽样检测方法评估供应商提供的零配件是否符合其声称的次品率上限（10%）。企业希望在尽可能少的抽样次数下，根据不同的置信水平，决定是否接收或拒绝该批次的零配件。

考虑到题目要求，既要提高检测的效率（样本量尽量少），又要保证了判断的准确性（高置信水平下的拒收与接收标准）。在统计分析中，利用正态分布中的 Z 值公式，结合标称次品率和允许误差（5%），计算出不同置信水平下的样本量。对于 95%置信水平，计算得出最小样本量，确保能够拒收超过 10%次品率的批次；而对于 90%置信水平，计算样本量。通过二项分布的累积概率函数，逐步增加可能的次品数量，找到分别对应 95%和 90%置信水平下的最大容忍次品数。这一步骤的目标是确定接收或拒收批次时能容忍的次品数量上限。在 95%置信水平下，确定拒收的次品数阈值，而在 90%置信水平下，计算接收的次品数阈值。

2.2 问题二的分析

在问题二的求解中，企业面临四个关键决策：是否检测零配件 1、零配件 2、成品，以及是否拆解不合格的成品。每个决策有两种选择（是或否），形成 16 种可能的决策组合。决策流程从零配件检测开始，若检测需支付检测成本；若不检测，次品率保持不变，直接参与装配，可能影响后续成品质量。成品检测则决定是否剔除装配后的不合格品，以减少售后损失，若不检测成品，则可能导致次品进入市场，增加调换成本和影响品牌声誉等，最后根据成品合格率和收益函数选择拆解或丢弃不合格品。

遍历 16 种决策组合，计算每种组合的总成本与收益。模型分析表明，尽管检测可以降低次品率和售后风险，但并非所有情况下都需要检测或拆解。不同的生产条件下，企业应权衡成本与收益，选择最优的生产决策策略。

2.3 问题三的分析

问题三的核心是为复杂的生产流程设计最优决策方案，针对多个零配件、半成品和成品进行检测、拆解等操作。基于组合优化问题，采用模拟退火算法求解最优解。在这个问题中，模型以最大化利润为目标函数，包含零配件、半成品和成品的成功率、检测、装配和拆解等成本。在模型求解中，通过设定初始解并定义各个阶段的决策变量，逐步迭代得到最优解。通过模拟退火优化，该方案实现了成本与收益的平衡，降低了生产中次品造成的损失，提高了整体利润。

2.4 问题四的分析

针对问题四，假设问题二和问题三中涉及的零配件、半成品和成品的次品率是通过抽样检测得出的。为了更好地模拟生产中的不确定性，本文采用了二项分布模拟方法，对次品率进行动态更新。具体而言，抽样样本大小设为 139，基于此样本对零件和成品的次品率进行重新估计。相比直接使用固定次品率的数据进行决策，问题四的动态更新策略使决策更具灵活性，可以适应次品率的波动。

通过引入动态更新的次品率，问题四重新评估了问题二和问题三的模型，并进行了优化求解。结果表明，虽然抽样得到的次品率与原始次品率存在一定波动，但整体

差异较小，说明抽样方法能较好地反映生产中的实际次品情况。模型求解结果显示，在问题二和问题三中，动态更新次品率对检测和拆解策略的优化影响较小，且收益变化不显著。

三、 模型假设

1. 假设拆解过程不会对零配件造成损坏，拆解后零件可以再次使用，拆解后的零配件如果检测合格，可以重新用于生产以节省购买或制造的成本。
2. 假设库存中已存在一定数量的成品，以满足市场需求。当销售的不合格成品出现退换情况时，企业只需支付固定的调换费用，而无需考虑库存影响或额外的生产成本。
3. 假设所有零配件的供应充足，能够满足生产需求，不会因供应不足而影响生产过程或决策。
4. 假设检测过程准确无误，检测结果（合格/不合格）与实际次品率一致，不存在假阳性或假阴性情况。
5. 假设半成品或成品的次品率仅与装配的过程失误导致，不会对原先合格的零配件造成损坏。

四、 符号说明

符号	说明	单位
p	次品率	%
α	情境 1 的显著性水平	%
β	情境 2 的显著性水平	%
n	样本量	件
Z	标准正态分布的临界值	无
E	误差限度	无
k	次品数	件
$P_{\text{defectiveFinal}}$	成品总次品率	%
S	成品市场售价	元
C_{total}	所有成本之和	元
R	售出成品利润	元
p_1	零配件 1 不合格的概率	%
p_2	零配件 2 不合格的概率	%
p_3	成品装配过程不合格的概率	%
E_1	零配件 1 不合格	无
E_2	零配件 2 不合格	无
E_3	成品最后不合格	无
S	成品市场售价	元

C_{total}	所有成本之和	元
R	售出成品利润	元
$\hat{p}$	估计次品率	%

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 求解思路

针对问题一中给出的两种情境，本文根据置信区间，结合 z 值公式，计算得出最小样本量 n，并根据给定的标称值估计次品数目阈值。

首先，根据不同置信水平选择显著性水平。对于情境 1，进行右侧检验，判断次品率是否超过 10%。对于情境 2，进行左侧检验，判断次品率是否符合标称值不超过 10%。通过查询标准正态分布比对表，确定出 z 值，并根据公式计算，针对不同置信水平，得出相应的样本量 n。

最后，利用二项分布公式计算出每个假设检验的临界值，即次品数 k。对于拒收情境，计算出的次品数超过某一临界值时，判定批次次品率大于 10%，则拒收；对于接收情境，次品数低于某一临界值，则判定次品率不超过 10%，接收批次。

5.1.2 模型的建立与求解

步骤 1：设定零假设和备择假设

抽样检测目标是为了在尽可能少的检测次数（样本量）下，提供足够的统计依据来判断供应商的零配件次品率是否符合企业要求，针对题目要求，存在以下两种情境。

情境 1：在 95% 的置信水平下，如果零配件次品率超过标称值，则拒收零配件。
情境 2：在 90% 的置信水平下，如果零配件次品率不超过标称值， 则接收零配件。

因此，我们需要进行两个假设检验：

(1) 零假设： $p \le 10\%$

(2) 备择假设： $p > 10\%$

步骤 2：选择显著性水平和检验类型

(1) 对于情境 1（拒收）：

我们要在 95% 的置信水平下，确定次品率超过 10%，即设置显著性水平 $\alpha = 5\%$ 这是一个右侧检验，即我们检验是否有足够证据证明 $p > 10\%$

(2) 对于情境 2（接收）：

我们要在 90% 的置信水平下，认定次品率不超过 10%，即设置显著性水平 $\beta = 10\%$ 这是一个左侧检验，即我们检验是否有足够证据证明 $p \le 10\%$

步骤 3：确定样本量 n

为了确保检验具有足够的置信水平和显著性，我们可以使用样本量计算公式，通

过控制两类错误概率（I 类错误和 II 类错误）来确定样本量。

I 类错误：当 $p \leq 10\%$ 时，我们拒绝了正确的假设。

II 类错误：当 $p > 10\%$ 时，我们接受了错误的假设。

我们需要根据显著性水平和效能要求，确定样本量 n。通常样本量计算可以基于以下公式近似得到：

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

其中：

- Z 是标准正态分布的临界值（95%和 90%的置信水平对应的 Z 分别为 1.96，1.645）
- $p = 0.1$ 是假设的次品率
- E 是误差限度（本文中定义 $E = 0.05$ ）

步骤 4：计算临界值

零配件样本中的次品的数量可以视为二项分布问题，如下所示：

$$X \sim \text{Binomial}(n, p) \tag{2}$$

在 n 个样本中，有 k 个次品的概率为：

$$P(X = k) = C_n^k p^k (1-p)^{n-k} \tag{3}$$

其中：

- 总检测样本数为 n；
- 样本中次品的数量为 k；
- 每个样本的次品概率为 p。

利用上一个步骤求出的样本量 n 以及上文的概率公式，我们可以计算出对于每种假设的**临界值**，即在抽样检测中，多少个次品数 k 表明应该拒收或接收该批次零配件。

对于右侧检验（拒收的情境），我们计算样本中满足条件的临界值次品数 k：

$$P(X \geq k) \leqslant 5\% \tag{4}$$

对于左侧检验（接收的情境），我们计算样本中满足条件的临界值次品数 k：

$$P(X \leq k) \geq 90\% \tag{5}$$

步骤 5：综合决策

根据以上步骤和求解思路，求得在两种情境下的样本量和临界值

表 1：在两种情境下计算得出的样本量和临界值

计算结果	样本量	临界值
拒绝批次（95%置信度）	139	20
接受批次（90%置信度）	98	14

(1) 对于情境 1 的抽样检测方案：

在 95%的信度下，对这一批零配件采样 139 件，如果得到的次品数大于 20 件，则拒收这批零配件

(2) 对于情境 2 的抽样检测方案：

在 90%的信度下，对这一批零配件采样 98 件，如果得到的次品数小于 14 件，则接受这批零配件

5.2 问题二模型的建立与求解

5.2.1 求解思路

在问题二中，我们需要对每个生产阶段的决策进行分析，具体涉及以下四个决策变量：零配件 1 是否检测，零配件 2 是否检测，成品是否检测，不合格品是否拆解

每个决策都有两个选择：是或否，因此每个情形下可能的决策组合为$2^4=16$种。具体选择顺序如下：

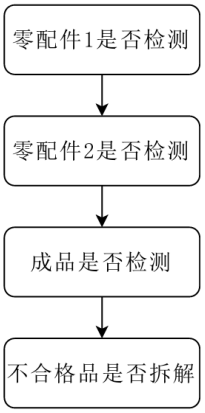


图 1：每一个决策选择顺序流程图

(1) 第 1 层：零配件 1 是否检测

如果检测零配件 1，则可以剔除次品，零配件 1 的次品率降为 0，但产生检测成本。如果不检测零配件 1，则该次品率保持不变，直接参与装配。

(2) 第 2 层：零配件 2 是否检测

同样，决定是否检测零配件 2。如果检测，次品率为 0，产生检测成本；不检测则保留原次品率，直接装配。

(3) 第 3 层：成品是否检测

装配成品的次品率由零配件 1 和零配件 2 的质量和成品自身的次品率共同决定。如果选择对成品进行检测，会产生检测成本。如果不检测，次品率为组合次品率，可能导致不合格品进入市场。

(4) 第 4 层：不合格成品是否拆解

如果成品检测为不合格，可以选择拆解回收零件，零配件被回收使用，需支付拆解费用，同时会产生回收收益。如果不拆解，则直接丢弃不合格成品。如果不合格成品进入市场，则需支付调换损失。

5.2.2 模型的建立

本文将结合每一层决策，按照嵌套循环的方式遍历所有 16 种决策组合。对于每个组合，计算其总成本和总收益，比较最优解。以下为具体的实施步骤：

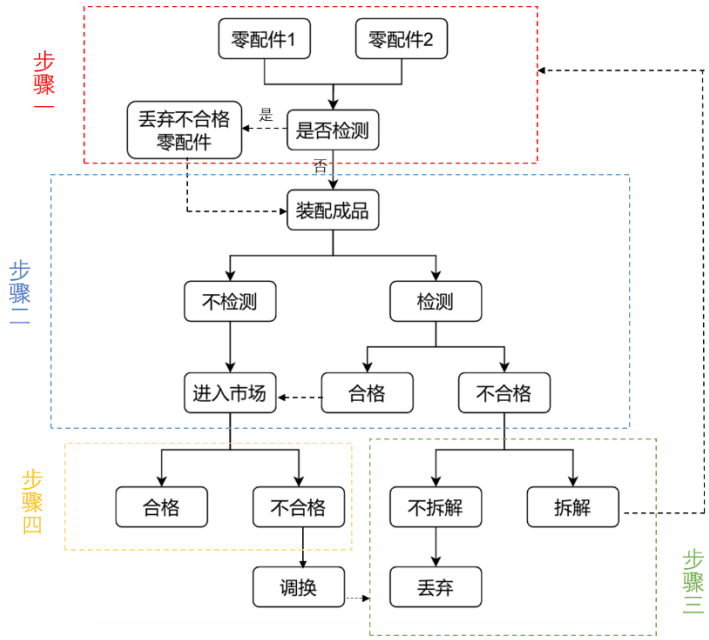


图 2：具体实施步骤流程图

步骤一：处理零配件 1 和零配件 2：

(1) 零配件 1：

- 若检测零配件 1，成本为：

$$C_{\text{part}1} = C_{\text{detect}1} + C_{\text{price}1} \tag{6}$$

其中：$C_{\text{part}1}$，$C_{\text{detect}1}$ 和 $C_{\text{price}1}$，分别为零配件 1 的总成本，检测成本和购买单价

- 若不检测，次品率保持不变，为 P_{part1}，零配件直接进入装配环节，成本为：

$$C_{\text{part}1} = C_{\text{price}1} \tag{7}$$

(2) 零配件 2：

零配件 2 同理，获得 P_{part2} 和 $C_{\text{part}2}$

步骤二：处理装配好的成品：

(1) 若检测成品，成品成本为：

$$C_{\text{final}} = C_{\text{detectFinal}} + C_{\text{assembly}} \tag{8}$$

其中：C_{final}，$C_{\text{detectFinal}}$ 和 C_{assembly} 分别为：成品的总成本，检测成本和装配成本

(2) 若不检测成品，成品次品率为：

$$P_{\text{defectiveFinal}} = 1-(1-P_{\text{part1}})\ (1-P_{\text{part2}})\ (1-P_{\text{final}}) \tag{9}$$

其中：$P_{\text{defectiveFinal}}$ 和 P_{final} 分别为：成品最终的次品率，成品装配过程的次品率
装配成品成本为：

$$C_{\text{final}} = C_{\text{assembly}} \tag{10}$$

步骤三：处理检测不合格的成品：

在企业的生产决策中，针对不合格的成品，企业有两个处理选择：拆解或者直接报废。根据题目说明，拆解过程不会对零配件造成损坏，但会产生额外的拆解费用。因此，如果零件之前已经检测，则不需要进行检测；若之前零配件未进行检测，拆解后选择为这些零配件补充检测成本进行检测，并将检测合格的零配件的购买单价之和计算为收益$G_{recycle}$，最后将所有的成本加起来减去收益$G_{recycle}$，得到最终的成本。

以上的过程可以被总结为如下的流程图：

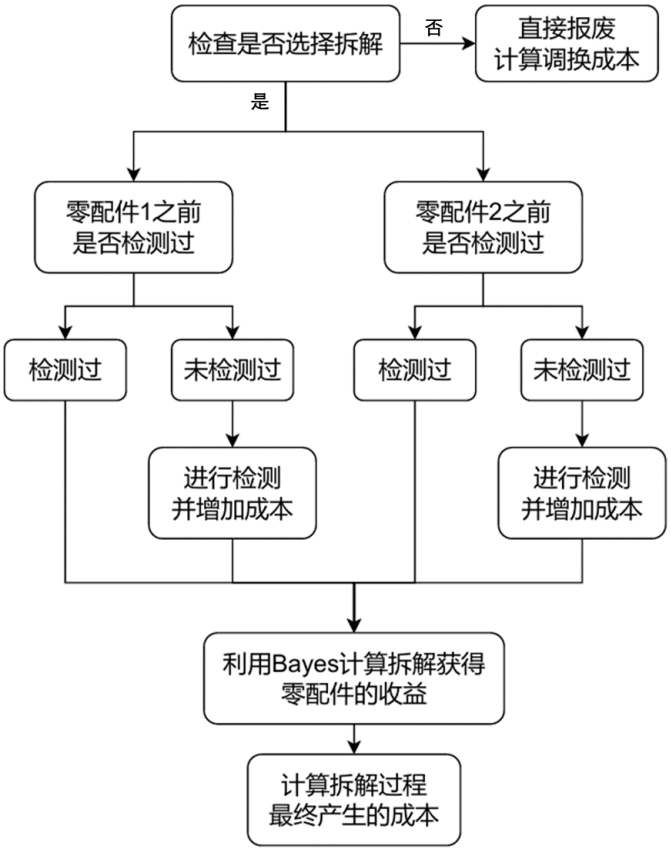


图 3：处理检测不合格品的思路流程图

为计算$G_{recycle}$，利用以下假设：拆解过程不会对零配件造成损坏，结合假设，同时利用 Bayes 公式进行分析，得到以下四种情况：

注：

- p_1：零配件 1 不合格的概率
- p_2：零配件 2 不合格的概率
- p_3：成品装配过程不合格的概率
- E_1：零配件 1 不合格
- E_2：零配件 2 不合格
- E_3：成品最后不合格

(1) 情况 1：零配件 1 和零配件 2 都未检测过。因此，在成品最后不合格的前提下，零配件 1 和 2 不合格的概率分别为：

$$P(E_1|E_3)=\frac{p_1}{1-(1-p_1)(1-p_2)(1-p_3)} \tag{11}$$

$$P(E_2|E_3)=\frac{p_2}{1-(1-p_1)(1-p_2)(1-p_3)} \tag{12}$$

(2) 情况 2：零配件 1 检测过，零配件 2 未检测过。因此，在成品最后不合格的前提下，零配件 1 和 2 不合格的概率分别为：

$$P(E_1|E_3)=\frac{0}{1-(1-p_2)(1-p_3)}=0 \tag{13}$$

$$P(E_2|E_3)=\frac{p_2}{1-(1-p_2)(1-p_3)} \tag{14}$$

(3) 情况 3：零配件 2 检测过，零配件 1 未检测。因此，在成品最后不合格的前提下，零配件 1 和 2 不合格的概率分别为：

$$P(E_1|E_3)=\frac{p_1}{1-(1-p_1)(1-p_3)} \tag{15}$$

$$P(E_2|E_3)=\frac{0}{1-(1-p_1)(1-p_3)}=0 \tag{16}$$

(4) 情况 4：零配件 1 和零配件 2 都检测过。因此，在成品最后不合格的前提下，零配件 1 和 2 不合格的概率分别为：

$$P(E_1|E_3)=0 \tag{17}$$

$$P(E_2|E_3)=0 \tag{18}$$

分别对以上四种情况进行分析，得到检测合格零配件的购买单价之和，即收益$G_{recycle}$

$$\mathrm{part\,1\,recovery}=(1-P(\mathrm{E_1}|\mathrm{E_3})\ )\cdot C_{\mathrm{price1}} \tag{19}$$

$$\text{part}\,2\text{recovery} = (1 - P(\text{E}_2|\text{E}_3)\ ) \cdot C_{\text{price}2} \tag{20}$$

$$G_{\text{recycle}} = \text{part}\,1\text{recovery} + \text{part}\,2\text{recovery} \tag{21}$$

注：

- $\text{part}\,1\text{recovery}$：拆解零配件 1 所获收益
- $\text{part}\,2\text{recovery}$：拆解零配件 2 所获收益

拆解过程详细的成本计算公式如下所示。

(1) 若拆解不合格成品，可以回收部分零配件，拆解过程产生的成本如下：

$$C_{\text{repair}} = P_{\text{defectiveFinal}} \times (C_{\text{replacement}} + C_{\text{disassemble}} + C_{\text{detectPart}\,1} + C_{\text{detectPart}\,2} - G_{recycle}) \tag{22}$$

其中：C_{repair}，$P_{\text{defectiveFinal}}$，$C_{\text{replacement}}$，$C_{\text{disassemble}}$，$C_{\text{detectPart}\,1}$，$C_{\text{detectPart}\,2}$，$G_{recycle}$ 分别为拆解不合格品所带来的费用，成品总次品率，调换损失，拆解费用，拆解后零配件 1 的检测成本，拆解后零配件 2 的检测成本，检测合格零配件的购买单价之和。

(2) 若不拆解，直接丢弃不合格品，不拆解过程产生的成本如下：

$$C_{\text{repair}} = P_{\text{defectiveFinal}} \times C_{\text{replacement}} \tag{23}$$

步骤四：处理进入市场的成品：

对于不合格的成品，重复步骤三操作，对于合格的成品计算售出成品利润，公式如下所示：

$$R = (1 - P_{\text{defectiveFinal}}) \times S - C_{total} \tag{24}$$

其中：$P_{\text{defectiveFinal}}$，$S$，$C_{total}$，$R$ 分别为成品总次品率，成品市场售价，所有成本之和，售出成品利润。

5.2.3 模型的求解

利用以上建立的模型，使用 MATLAB 求解得出以下结果：

为了直观了解同一情况下不同策略所带来的收益，绘制出在 16 种不同的策略所产生收益的直方图。

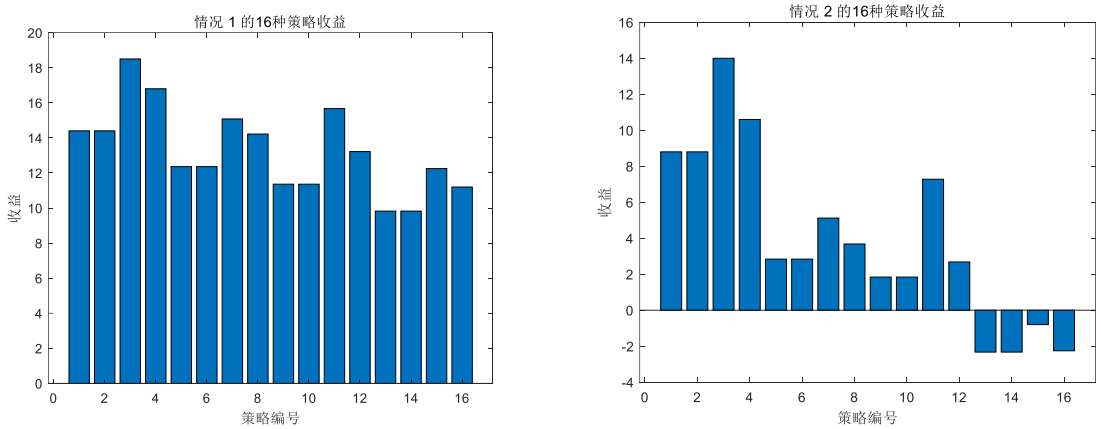


图 4：情况 1 和情况 2 的采用 16 种不同的策略所产生的收益的直方图

注：由于图表过多，正文部分只展示情况 1 和情况 2 的采用 16 种不同的策略所产生的收益的直方图，其余部分见附录。

观察直方图结果发现：

在情况 1 中，我们观察到收益范围从 9.82 到 18.50，这表明策略选择对最终结果有着显著的影响。通过实施零配件和成品的检测策略，我们能够显著提升收益。最优策略的关键在于平衡检测与拆解的程度，这样可以避免因直接报废零部件或完全不进行检测而造成的潜在高损失。

在情况 2 中，收益范围从-2.33 到 14.00，这揭示了某些策略可能导致负收益，特别是那些不检测零件和成品，直接丢弃或不进行拆解的组合。这些策略可能会带来较大的财务风险。然而，通过实施合理的检测和拆解策略，我们不仅可以避免这些风险，还能显著提高收益，特别是通过避免不检测带来的潜在损失。

对于六种情况，分析采用 16 种不同的策略所产生的最大收益和最小成本以及所采用的具体策略，得到以下图表：

表 2：针对六种情况的最终求解结果

情况	最小成本	最大收益	零配件 1	零配件 2	成品	拆解与否
情况 1	28.57	18.50	检测	检测	不检测	拆解
情况 2	28.56	14.00	检测	检测	不检测	拆解
情况 3	31.00	16.10	检测	检测	不检测	拆解
情况 4	30.00	12.80	检测	检测	检测	拆解
情况 5	29.59	15.77	不检测	检测	不检测	拆解
情况 6	29.43	19.70	检测	检测	不检测	不拆解

(1) 针对情况 1，最小成本为 28.57，最大收益为 18.50。最优策略为对零配件 1 和零配件 2 进行检测，对成品不进行检测，并对不合格成品选择拆解处理。

(2) 针对情况 2，最小成本为 28.56，最大收益为 14.00。最佳决策为对零配件 1 和零配件 2 进行检测，对成品不进行检测，并选择拆解处理不合格成品。

(3) 针对情况 3，最小成本为 31.00，最大收益为 16.10。推荐的决策是对零配件 1 和零配件 2 进行检测，对成品不进行检测，并对不合格成品进行拆解。

(4) 针对情况 4，最小成本为 30.00，最大收益为 12.80。建议对零配件 1 和零配件 2 进行检测，对成品也进行检测，并对不合格成品选择拆解处理。

(5) 针对情况 5，最小成本为 29.59，最大收益为 15.77。最优策略是对零配件 1 不进行检测，对零配件 2 进行检测，对成品不进行检测，并对不合格成品选择拆解处理。

(6) 针对情况 6，最小成本为 29.43，最大收益为 19.70。最佳决策是对零配件 1 和零配件 2 进行检测，对成品不进行检测，并选择不拆解不合格成品。

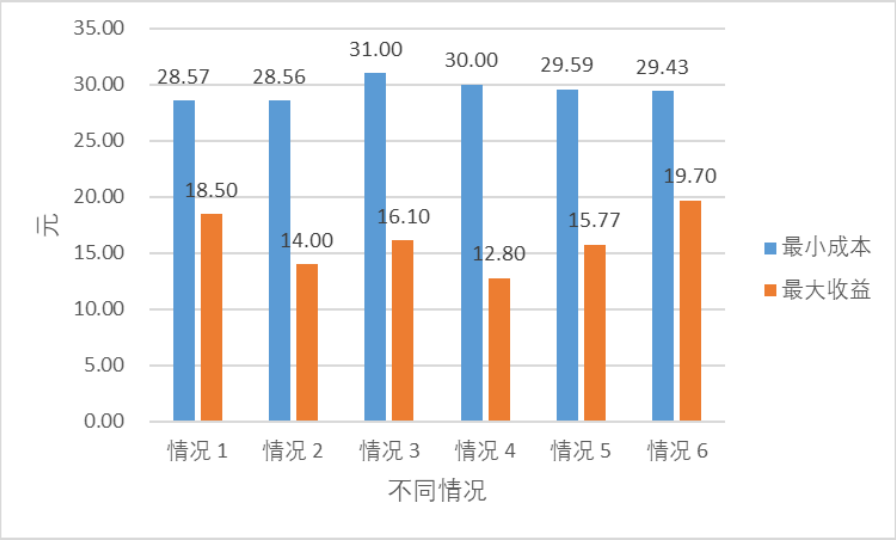


图 5：对不同情况选择最优策略产生的最小成本和最大收益对比图

分析以上图表，得到如下结果：

(1) 成本与收益的平衡：

最低成本出现在情况 2（28.56 元），而最大收益则出现在情况 6（19.70 元）。成本最低的情况并不一定带来最高收益，表明需要综合考虑检测和拆解策略，不能单纯依据成本作出决策。

(2) 检测策略的重要性：

大多数情况下，对零件 1 和零件 2 进行检测是常见的最佳决策，表明检测有助于减少整体次品率，并降低返工和客户投诉的风险。只有在情况 5 中，对零件 1 采取了不检测的策略，这是因为零件 1 的检测成本较高（8 元），与检测可能带来的收益不成正比，因此不检测更具经济性。

(3) 成品检测的决策：

在大多数情况下，成品不进行检测是最佳决策，尤其是次品率较低的情况。这说明对于次品率较低的情况下，不检测可以降低成本，而次品率高的情况下（如情况 4），检测成品成为必要措施，以避免高昂的调换损失。

(4) 拆解策略的选择：

情况 6 是唯一一个不选择拆解的情形，因为虽然拆解费用在大多数情况下较低（5 元），但在情况 6 中，拆解费用显著增加到 40 元。这导致拆解成本高于不拆解带来的收益，因此不拆解是更为经济的选择。

(5) 调换损失的影响：

调换损失较大的情况（如情况 3 和 4）对成品的检测决策有较大影响。当调换损

失增加时，企业更倾向于对成品进行检测以避免未来更高的售后成本。

根据以上六种情况，总结出以下经验：

(1) 高次品率和高调换损失时，检测零配件和成品。
(2) 低次品率和低调换损失时，可以减少或不进行检测。
(3) 高购买单价和高装配成本时，加强检测以避免不必要的浪费。
(4) 低检测成本时，检测是有效提高成品质量的手段，而高检测成本时应谨慎考虑检测环节。

5.3 问题三模型的建立与求解

5.3.1 求解思路

问题三的求解要求我们为一个复杂的生产流程设计最优决策方案，其中涉及多个工序、零配件和半成品。此问题可以归纳为一个组合优化问题，并通过**模拟退火算法**求解最优方案。

模拟退火算法是一种基于随机搜索和局部搜索的全局优化算法，主要用于求解复杂的组合优化问题。它源于物理退火过程，借鉴了固体退火时的物理原理。通过在高温时允许“坏解”，并随着温度的降低逐渐减少“坏解”的接受概率，模拟退火能够跳出局部最优，最终找到全局最优解。

模拟退火的核心思想是通过引入一个概率机制，即使当前解较差，也能以一定的概率接受这个解，这种机制允许算法跳出局部最优解，避免陷入局部最优。

退火过程的步骤：

(1) **初始化温度和解**：设置初始温度 $T_0 = 1000$ 和初始解 x_0。
(2) **生成邻域解**：对当前解 x_t 进行扰动，生成新的解 x_{new}。
(3) **计算目标函数值**：分别计算当前解 $f(x_t)$ 和新解 $f(x_{new})$ 的目标函数值。
(4) **判断接受准则**：

- 若 $f(x_{new})$ 优于 $f(x_t)$，则接受新解；
- 若 $f(x_{new})$ 劣于 $f(x_t)$，则以一定概率 $P = \exp\left(\frac{f(x_{new}) - f(x_t)}{T}\right)$ 接受该解，概率依赖当前温度 T。

(5) **温度衰减**：每次迭代后降低温度，使用**几何退火方式**：$T_{new} = \alpha T_{old}$，其中衰减因子 $\alpha = 0.95$。

(6) **终止条件**：当温度降到阈值 $T_{\min} = 1$ 时，算法终止，输出最优解。

5.3.2 模型的建立

以下是该问题的详细求解步骤：

步骤一：定义初始解

设定初始解 x_0，该解包含所有决策变量：

- 8 个零部件的检测决策；
- 3 个半成品的检测决策；

- 成品的检测和拆解决策。

令当前解为 $x=(x_1,x_2,\ldots,x_{16})$，其中 x_i 为 0 或 1，表示是否对某个零部件或产品进行检测或拆解。

步骤二：定义目标函数

目标函数是总利润，包括以下几部分：

$$R=P_{\text{market}}\cdot P_{\text{success}}-C_{\text{total}}-C_{\text{replace}}\cdot(1-P_{\text{success}})-C_{\text{disassemble}} \tag{25}$$

其中：

- P_{market}：市场价格
- P_{success}：产品的成功率（成功率由零部件、半成品、成品的合格率计算得到）
- C_{total}：总成本（包括零部件、半成品和成品的检测、组装成本）
- C_{replace}：置换成本
- $C_{\text{disassemble}}$：拆解成本

1. **成本计算：**

(1) 零部件总成本 C_{comp}：

$$C_{\text{comp}}=\sum_{i=1}^{8}(x_i\cdot(C_{\text{buy},i}+C_{\text{inspect},i})+(1-x_i)\cdot C_{\text{buy},i}) \tag{26}$$

其中：

- x_i：第 i 个零部件的检测决策变量，1 表示检测，0 表示不检测
- $C_{\text{buy},i}$：第 i 个零部件的购买成本
- $C_{\text{inspect},i}$：第 i 个零部件的检测成本

(2) 半成品总成本 C_{sub}：

$$C_{\text{sub}}=\sum_{j=1}^{3}(y_j\cdot(C_{\text{assemble},j}+C_{\text{inspect},j})+(1-y_j)\cdot C_{\text{assemble},j}) \tag{27}$$

其中：

- y_j：第 j 个半成品的检测决策变量，1 表示检测，0 表示不检测
- $C_{\text{assemble},j}$：第 j 个半成品的组装成本
- $C_{\text{inspect},j}$：第 j 个半成品的检测成本

(3) 成品总成本 C_{final}：

$$C_{\text{final}}=z\cdot(C_{\text{assemble,final}}+C_{\text{inspect,final}})+(1-z)\cdot C_{\text{assemble,final}} \tag{28}$$

其中：

- z：成品的检测决策变量，1 表示检测，0 表示不检测
- $C_{\text{assemble,final}}$：成品的组装成本

- $C_{\text{inspect,final}}$：成品的检测成本

(4) 总成本（包括零部件、半成品和成品的检测、组装成本）C_{total}：

$$C_{\text{total}} = C_{\text{comp}} + C_{\text{sub}} + C_{\text{final}} \tag{29}$$

(5) 拆解成本$C_{\text{disassemble}}$：

$$C_{\text{disassemble}} = \sum_{k=1}^{3} d_k \cdot C_{\text{disassemble},k} + d_{\text{final}} \cdot C_{\text{disassemble,final}} \tag{30}$$

其中：

- d_k：第 k 半成品的拆解决策变量，1 表示拆解，0 表示不拆解
- $C_{\text{disassemble},k}$：第 k 个半成品的拆解成本
- d_{final}：成品的拆解决策变量
- $C_{\text{disassemble,final}}$：成品的拆解成本

i. 考虑到对不合格的半成品拆解后，有可能获得合格的零配件，因此第 k 个半成品的拆解成本$C_{\text{disassemble},k}$的计算方式，如下图所示：

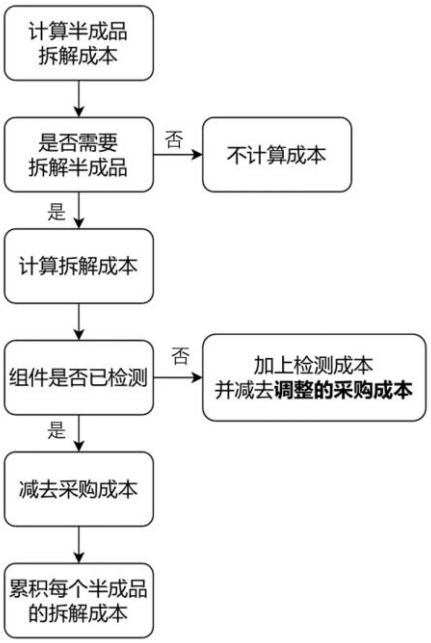


图 6：半成品拆解成本的计算流程图

其中：**调整的采购成本**的计算方式为：组件的缺陷概率对采购成本进行调整

ii. 考虑到对不合格的成品拆解后，有可能获得合格的零配件，因此第 k 个半成品的拆解成本$C_{\text{disassemble},k}$的计算方式，如下图所示：

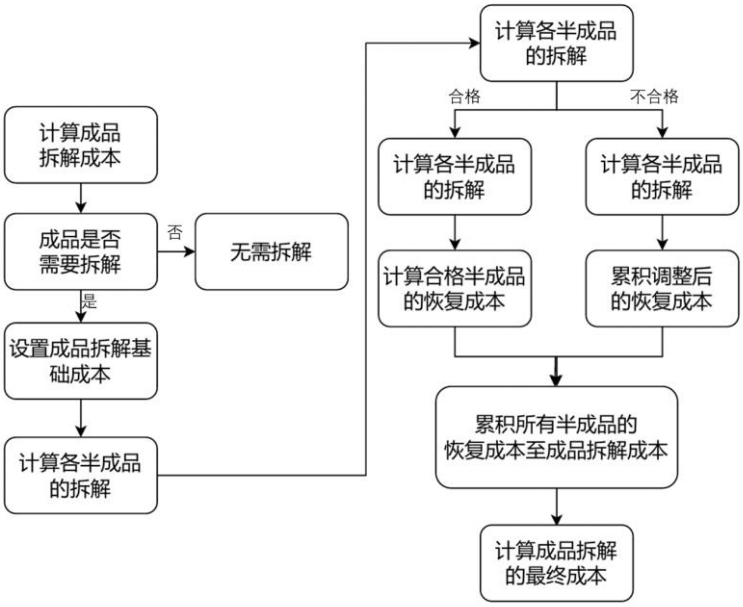


图 7：成品拆解成本的计算流程图

其中：成本的计算方式为：

- 合格的半成品：如果该半成品是合格的，计算其恢复成本（包括组件的采购成本和组装成本），并将这些成本从成品拆解成本中扣除。
- 不合格的半成品：如果该半成品不合格，首先计算半成品通过的概率，然后基于概率调整恢复成本，最后从成品拆解成本中扣除调整后的恢复成本。

2. **合格率计算**：

零部件、半成品和成品的合格率均与检测决策有关，结合概率论的知识，通过回溯已经进行过的决策进行动态更新。

以下为每个步骤的合格率计算公式。组成第 j 个半成品，零部件的合格率 $P_{\text{comp,j}}$：

$$P_{\text{comp},j} = \prod_{i=ns_j}^{ne_j} (1 - p_{\text{defect},i})^{1-x_i} \tag{31}$$

其中

- ns_j：组成第 j 个半成品，所需零配件的起始标号
- ne_j：组成第 j 个半成品，所需零配件的结束标号
- $p_{\text{defect},i}$：第 i 个零部件的次品率

(1) 半成品的合格率 P_{sub}：

$$P_{\text{sub},j} = (1 - p_{\text{defect,sub},j}) \cdot P_{\text{comp},j} \tag{32}$$

其中：

- $p_{\text{defect,sub},j}$：第 j 个半成品的次品率
- $P_{\text{comp},j}$：构成第 j 个半成品的零部件的合格率（由相应零部件的成功率计算）

(2) 成品的合格率 P_{final}：

$$P_{\text{final}} = (1 - p_{\text{defect,final}}) \cdot \prod_{j=1}^{3} P_{\text{sub},j} \tag{33}$$

其中：

- $p_{\text{defect,final}}$：成品的次品率
- $P_{\text{sub},j}$：第 j 个半成品的合格率

步骤三：生成邻域解

在每次迭代中，通过对当前解 x_t 进行扰动生成新的解 x_{new}。具体操作是随机选择一个或多个决策变量 x_i，将其值翻转，即 $x_i \leftarrow 1 - x_i$，并引入了检测机制，避免出现重复的解。

步骤四：接受准则

对比新解和旧解的利润，若新解的利润更大，则直接接受新解；若新解的利润较差，则以概率 $P = \exp\left(\frac{\Delta f}{T}\right)$，接受新解，其中 $\Delta f = f(x_{new}) - f(x_t)$，T 为当前温度。

步骤五：温度衰减

每次迭代后，更新温度 T，使用几何衰减方式：

$$T = \alpha T \tag{34}$$

其中衰减率 α 取 0.99。

步骤六：终止条件

当温度降到阈值 $T_{\min}$ 时，算法停止，返回当前最优解。

5.3.3 模型的求解

根据以上模型的建立过程，使用 MATLAB 进行求解，收益随着迭代过程的变化如下图所示：

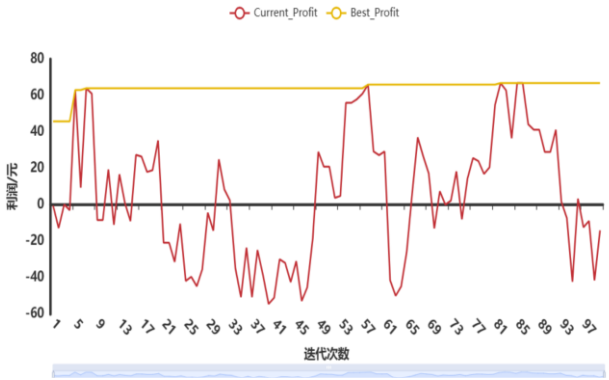


图 8：当前利润和最优利润随着迭代次数增加的变化曲线图

表 3：问题三的最终决策结果

决策阶段	决策结果	决策阶段	决策结果
零配件 1	不检测	半成品 1	检测
零配件 2	不检测	半成品 2	检测
零配件 3	不检测	半成品 3	检测
零配件 4	不检测	半成品 1	拆解
零配件 5	不检测	半成品 2	拆解
零配件 6	不检测	半成品 3	拆解
零配件 7	不检测	成品	不检测
零配件 8	不检测	成品	拆解

分析以上结果，得出以下结论：

(1) 零配件决策分析：

- 决策结果：所有零配件 1-8 均选择了“不检测”。
- 原因分析：

零配件的次品率统一为 10%，检测成本相对于零配件的购买单价较低，但次品率并不高。检测成本如零配件 3、6、8 的 2 元，对于 10%的次品率来说较高，因此检测这些零配件并不划算。

由于不检测可以避免检测成本的支出，尽量减少不必要的检测环节，将检测和质量把关放在后续的半成品和成品阶段更为合适。

(2) 半成品决策分析

- 决策结果：半成品 1、2、3 选择了“检测”和“拆解”。
- 原因分析：

半成品的次品率同样为 10%，但装配成本（8 元）和拆解成本（6 元）较高，因此在装配后对半成品进行检测显得非常重要，可以有效减少不合格半成品进入成品阶段，避免后续更大的损失。

拆解决策：如果检测出半成品不合格，则立即进行拆解，通过拆解来回收部分零配件，减少直接报废造成的损失。因为拆解成本相对较低且可以回收利用不合格品中的组件。

(3) 成品决策分析

- 决策结果：成品选择了“不检测”和“拆解”。
- 原因分析：

成品的市场售价为 200 元，虽然次品率也是 10%，但在半成品阶段已经进行过检测和拆解，因此成品阶段的次品率已经被有效控制，不需要再进行检测来增加成本。拆解决策：一旦发现成品不合格，选择直接进行拆解。这是因为成品的拆解成本较高（10 元），但是相较于调换损失（40 元），拆解仍然是较为经济的选择，尤其在成品市场售价较高时，通过拆解回收零配件来挽回部分损失能最大限度降低风险。

总结：

(1) 由于半成品处于三道工序的中间位置，起到了承上启下的关键作用。如果不对半成品进行检测，即使所有零配件都是合格的，半成品仍会有 10%的次品率。这样，即使在最理想的情况下，成品的合格率也只有 0.9 * 0.9 * 0.9 * 0.9 = 0.6561，这显然不利于企业的信誉和利益。

(2) 从合格率的角度来看，确实对所有零配件都进行检测能最大程度确保合格，但这将大幅增加检测成本。鉴于零配件的不合格率并不高，只需通过检测半成品是否合格即可进行拆解处理。在拆解过程中，我引入了更贴近实际的零配件合格率估计，结合单位购买单价，这种方式更好地体现了拆解回收的收益预期。

(3) 此外，当半成品通过检测后，组装成成品，其合格率可以达到 90%，从而降低成品次品率，减少了隐形的置换成本，提升了收益，并改善了企业信誉。更为重要的是，由于半成品已经通过检测，即使最终组装的成品因 10%的次品率被退换，我们对其进行拆解后，仍然可以得到合格的半成品，避免了资源浪费，提高了生产效率。

5.4 问题四模型的建立与求解

5.4.1 求解思路

在问题四的分析中，我们假设问题二和问题三中提到的零配件、半成品和成品的次品率是基于抽样检测得出的。为了更准确地模拟实际生产中的不确定性，我们采用了二项分布模拟方法对次品率进行了更新。具体来说，我们以样本大小为 139 进行了抽样，并据此计算出每个零件及成品的新次品率。这种方法能够更好地反映生产过程中次品率的波动，提高了模拟的准确性。

与问题二和问题三直接使用固定数据进行决策不同，问题四在抽样部分对次品率进行了动态更新，并基于这些更新后的数据进行决策优化。这种策略使得问题四的解决方案更加灵活，能够适应次品率的不同波动情况，实现更合理的优化。

5.4.2 模型的建立

在问题四中，引入了二项分布模拟来对零件及成品的次品率进行抽样更新。二项分布的概率均值均与题目表格中的次品率标称值相同，具体而言，采用了二项分布的方式，通过使用 MATLAB 对产品的次品率进行模拟和更新。以下是详细求解估计次品率的过程：

步骤一：二项分布简介：

在生产过程中，假设每个产品的次品率是已知的，且每个产品的好坏是独立的，那么可以用二项分布来模拟生产批次中的次品数量。

二项分布的定义是：对于每次独立试验，成功的概率为 p，在 n 次试验中，发生 k 次成功的概率为：

$$P(X=k)=C_n^k p^k(1-p)^{n-k} \tag{35}$$

其中：

- n：抽样的次数，本文为 139。
- p：每个产品真实的次品率。

- k：次品的数量。

步骤二：使用二项分布模拟次品率

在问题四中，基于给定的样本大小 n=139 （即抽样的样本量），我们可以通过二项分布来抽样模拟次品数量。

假设某个零件的次品率为 p，那么生产 n 个该零件时，使用二项分布模拟产生的次品数量为 X，可以使用 MATLAB 生成。

$$X = \text{binornd}(n, p) \tag{36}$$

其中 X 是二项分布的随机变量，表示这批产品中次品的个数。

步骤三：计算新的次品率

在生产过程中，次品率可能会波动。我们通过抽样获得的次品数量 X，可以用于计算每批次更新的次品率 $\hat{p}$。假设在一次生产中，生成了 n 个产品，其中次品的数量是 X，则新的次品率 $\hat{p}$ 为：

$$\hat{p} = \frac{X}{n} \tag{37}$$

5.4.3 模型的求解

基于以上步骤求解得到的次品率 $\hat{p}$，重新对问题二和问题三进行求解，得到如下结果：

(1) 对于问题二，我们对比了原始方法和问题四中改进后的方法所得到的结果：

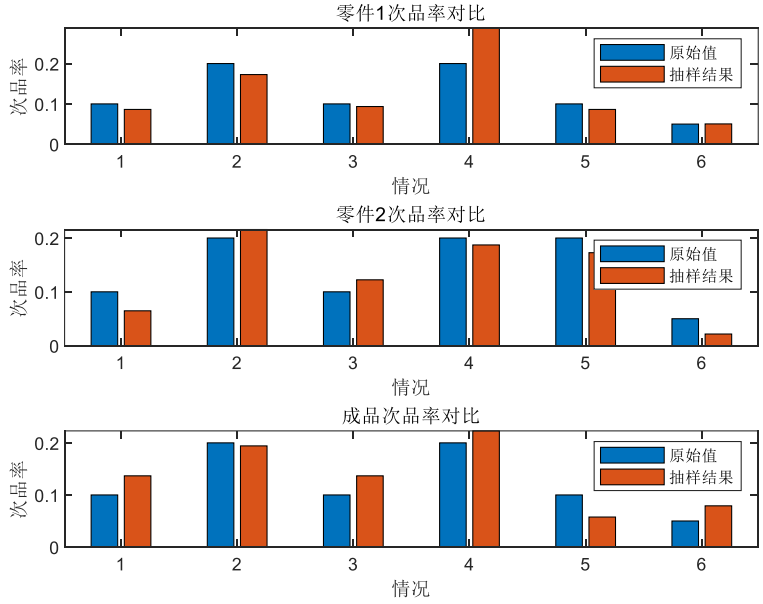


图 9：问题二中原始值和抽样得到的次品率的条形对比图

上图展示了问题二中原始次品率与抽样得到的次品率的条形图对比。通过直观比较，我们发现抽样得到的次品率与原始次品率相差无几，均在原始值附近波动，这表明抽样方法能够较好地预测原始次品率。

表 4：问题二更新后的次品率

零配件 1	零配件 2	成品
次品率	次品率	次品率
7.91%	11.51%	5.04%
20.14%	23.74%	17.27%
7.91%	10.07%	8.63%
25.90%	27.34%	20.14%
8.63%	20.86%	8.63%
5.76%	2.16%	3.60%

表 5：使用抽样得到的次品率求解问题二得到的结果

情况	最小成本	最大收益	零配件 1	零配件 2	成品检测	拆解与否
情况 1	28.77	17.50	检测	检测	不检测	拆解
情况 2	28.62	14.58	检测	检测	不检测	拆解
情况 3	31.00	15.06	检测	检测	不检测	拆解
情况 4	30.00	11.91	检测	检测	检测	拆解
情况 5	29.36	19.24	不检测	检测	不检测	拆解
情况 6	29.24	21.10	检测	检测	不检测	不拆解

通过对比分析问题二和抽样检测的结果发现：在六种情况下，最小成本和最大收益的整体差异较小，说明抽样方法能较好地反映生产中的实际次品情况。

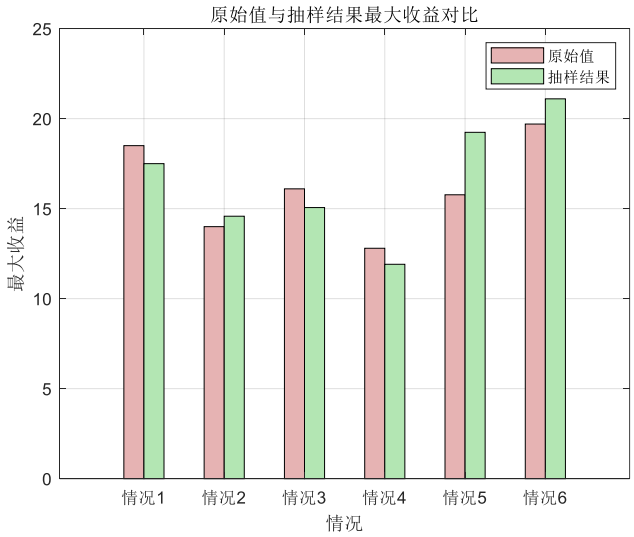


图 10：问题二中使用原始值和抽样次品率的结果条形对比图

上图展示了问题二中原始次品率与抽样得到的次品率的条形图对比。通过直观比较，我们发现抽样得到的次品率与原始次品率相差无几，均在原始值附近波动，这表明抽样方法能够较好地预测原始次品率。

(2) 对于问题三，我们同样比较了原始方法和问题四中改进后的方法所得到的结果：

表 6：问题三更新后的次品率

标号	次品率	标号	次品率
零配件 1	8.63%	零配件 7	8.63%
零配件 2	8.63%	零配件 8	13.67%
零配件 3	8.63%	半成品 1	9.35%
零配件 4	10.07%	半成品 2	10.07%
零配件 5	4.32%	半成品 3	6.47%
零配件 6	9.35%	成品	7.91%

表 7：使用抽样次品率得到最优策略结果

决策阶段	决策结果	决策阶段	决策结果
零配件 1	不检测	半成品 1	检测
零配件 2	不检测	半成品 2	检测
零配件 3	不检测	半成品 3	检测
零配件 4	不检测	半成品 1	拆解
零配件 5	不检测	半成品 2	不拆解
零配件 6	不检测	半成品 3	拆解
零配件 7	不检测	成品	不检测
零配件 8	不检测	成品	拆解

通过对比改进前后的结果，我们发现原始方法的最大收益为 75.80，而改进方法的最大收益为 75.68。两者之间的收益差异非常小，这表明抽样检测得到的次品率对拆解费用的优化并没有显著影响整体收益。原始方法通过检测和拆解半成品及成品来控制次品，而改进方法则通过减少部分拆解费用来平衡次品率带来的收益损失。

六、 模型的评价、改进与推广

6.1 模型的优点

通过合理运用概率论知识，我们将其与实际工厂生产中的抽样检测紧密结合，构建了一个详尽的利益计算函数。该函数充分考虑了生产过程中的各个环节。例如，拆解并非必需的步骤，只有在检测不合格或成品被退换时才会做出此决策。我们通过概率计算对这一过程进行量化，从而在利益函数中合理模拟工厂的生产流程。

从工厂管理者的角度出发，我们充分考虑了产品合格率对企业的重要性。这不仅影响潜在的置换成本，还对客户与企业的合作意愿产生深远影响。通过构建有效的最大收益目标函数，成品的合格率得以提升，无形中为企业积累了财富。

模型采用了二项分布模拟次品率，二项分布适用于具有固定次数独立试验的情况，每次试验都只有两个可能的结果（合格或次品）。这一点与实际生产中的次品情况相符，使得模型能较为准确地反映实际生产中可能遇到的次品情况。

模型整合了生产过程中的关键决策变量，包括零配件检测、半成品处理和成品拆解等。充分考虑到拆解后带来的成本节省。这些决策变量是生产过程中的核心环节，

通过详细模拟这些环节，模型能够全面覆盖生产过程的主要操作，确保优化结果具有较高的现实参考价值。

模拟退火算法能够在复杂的解空间中找到接近最优解。其优点在于能有效处理非线性和复杂约束问题，具有较强的全局搜索能力。

6.2 模型的缺点

模型在成本和收益计算中进行了简化，可能无法完全反映复杂生产过程中的所有实际情况。

模拟退火算法的计算效率受参数设置影响，可能需要较长时间才能找到最优解，尤其在解空间较大时。

6.3 模型的改进

问题一中可以考虑将模型扩展为动态模型，根据实时数据更新模型参数，以适应实际情况中的变化。

当前模型可能主要关注生产的某些阶段，如零配件检测和成品拆解。可以将模型扩展到其他生产阶段，如原材料采购、生产计划等，以提供更全面的决策支持。

6.4 模型的推广

通过调整模型参数和决策变量，可以将模型应用于不同产品类别的生产线。例如，对于电子产品生产线和机械部件生产线，模型需要根据各自的生产特性（如次品率的分布模式）进行适当调整。

将模型推广到服务业，例如维修服务的资源分配。类似的决策过程涉及服务的检测、处理和替换。

扩展模型至供应链优化中，考虑供应商选择、库存管理等因素。模型可以调整为在供应链管理中模拟不同的决策变量，如供应商质量检查、库存成本等。

七、 参考文献

[1]陈应飞,彭正超,胡晓兵,等.基于改进模拟退火-遗传算法的 FMS 生产排程优化分析[J].机械,2021,48(02):7-16.

[2]张士军,刘志国.基于贝叶斯理论的小批量产品抽样检验方法[J].统计与决策,2017,(11):24-27.DOI:10.13546/j.cnki.tjyjc.2017.11.006.

[3]秦心静,章平,张新杨.基于位置子市场划分的房价贝叶斯概率模型[J].重庆工商大学学报(自然科学版),2023,40(05):81-88.DOI:10.16055/j.issn.1672-058X.2023.0005.011.

附录

附录 1
介绍：支撑材料的文件列表
问题一的材料：ques1.m 问题二的材料：ques2.m, graphics.m, 问题二结果 问题三的材料：ques3.m, result3.xlsx 问题四的材料：ques4_2.m, ques4_3.m, wenti4_2.xlsx, q24_comp.m

问题一 matlab 求解代码

```
clc; clear;
% 给定参数
p_max = 0.10;      % 供应商声称的缺陷率 (10%)
E = 0.05;          % 允许的误差幅度 (5%)


% 不同置信水平的 Z 值
Z_95 = 1.96;       % 95% 置信水平的 Z 值（单侧检验），对应于标准正态分布的上分位点
Z_90 = 1.645;      % 90% 置信水平的 Z 值（单侧检验），对应于标准正态分布的上分位点


% 计算 95% 置信水平下的样本大小（拒绝批次）
% 使用样本大小公式: n = (Z^2 * p * (1 - p)) / E^2
n_95 = (Z_95^2 * p_max * (1 - p_max)) / E^2;
n_95 = ceil(n_95); % 向上取整确保样本大小足够


% 计算 90% 置信水平下的样本大小（接受批次）
% 使用相同的样本大小公式，但 Z 值不同
n_90 = (Z_90^2 * p_max * (1 - p_max)) / E^2;
n_90 = ceil(n_90); % 向上取整确保样本大小足够


% 显示样本大小结果
fprintf('95%% 置信水平下的样本大小（拒绝批次）: %.2f\n', n_95);
fprintf('90%% 置信水平下的样本大小（接受批次）: %.2f\n', n_90);


% 基于 95% 置信水平确定拒绝标准（使用二项分布近似）
% 计算每个可能的缺陷数量对应的累计概率，找到拒绝标准
% 查找可以容忍的最大缺陷数量
for k = 0:n_95
    p_reject = binocdf(k, n_95, p_max);  % 计算小于或等于 k 个缺陷的累计概率
    if p_reject >= 0.95
        reject_threshold = k;  % 找到累计概率超过 95% 的最大缺陷数量
        break;
    end
end

% 显示拒绝标准
```

```
fprintf('拒绝标准（95%% 置信水平）：%d 个缺陷（总样本数：%d）\n', reject_threshold,
n_95);


% 基于 90% 置信水平确定接受标准（使用二项分布近似）
% 计算每个可能的缺陷数量对应的累计概率，找到接受标准
% 查找可以容忍的最大缺陷数量
for k = 0:n_90
    p_accept = binocdf(k, n_90, p_max);  % 计算小于或等于 k 个缺陷的累计概率
    if p_accept >= 0.90
        accept_threshold = k;  % 找到累计概率超过 90% 的最大缺陷数量
        break;
    end
end


% 显示接受标准
fprintf('接受标准（90%% 置信水平）：%d 个缺陷（总样本数：%d）\n', accept_threshold,
n_90);
```

问题二 matlab 求解代码

```
close all;clc;clear;
% 定义参数（次品率，购买单价，检测成本，装配成本，市场售价，调换损失，拆解费用）
data = [
    0.10, 4, 2, 0.10, 18, 3, 0.10, 6, 3, 56, 6, 5;
    0.20, 4, 2, 0.20, 18, 3, 0.20, 6, 3, 56, 6, 5;
    0.10, 4, 2, 0.10, 18, 3, 0.10, 6, 3, 56, 30, 5;
    0.20, 4, 1, 0.20, 18, 1, 0.20, 6, 2, 56, 30, 5;
    0.10, 4, 8, 0.20, 18, 1, 0.10, 6, 2, 56, 10, 5;
    0.05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40;
];
% 结果保存变量
results = zeros(size(data,1), 1); % 成本
profits = zeros(size(data,1), 1); % 最终收益
decisions = strings(size(data,1), 4); % 用于记录每一步的检测和拆解决策


% 遍历所有情况
for i = 1:size(data,1)
    % 定义不同的决策组合，四层嵌套循环分别代表：零件 1 检测，零件 2 检测，成品检测，成品是否拆解
    min_cost = Inf;
    max_profit = -Inf;
    best_decision = '';

    for detect_part1 = [true, false]
        for detect_part2 = [true, false]
            for detect_final = [true, false]
                for disassemble = [true, false]
                    % 计算当前决策组合的总成本和收益
```

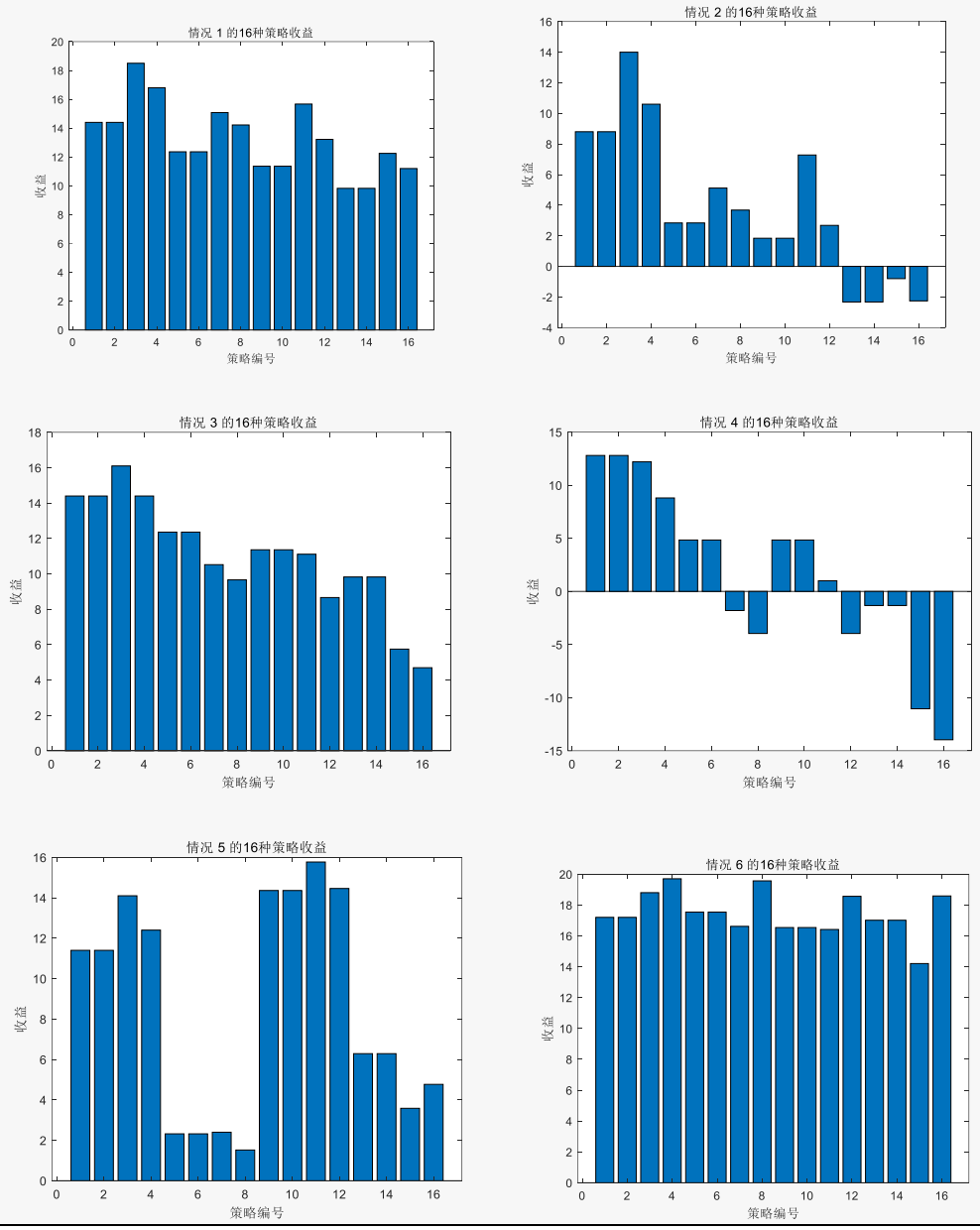
```
                    [current_cost, current_profit] =
evaluate_cost_and_profit(detect_part1, detect_part2, detect_final, disassemble,
i, data);

                    % 判断是否为最优决策
                    if current_cost < min_cost
                        min_cost = current_cost;
                    end
                    if current_profit > max_profit
                        max_profit = current_profit;
                        best_decision = sprintf('Part1: %s, Part2: %s, Final: %s,
Disassemble: %s', ...
                                                bool_to_str(detect_part1),
bool_to_str(detect_part2), bool_to_str(detect_final),
whether_disassemble(disassemble));
                    end
                end
            end
        end
    end

    % 保存最优决策
    results(i) = min_cost;
    profits(i) = max_profit;
    decisions(i) = best_decision;
end


% 输出结果
disp('最佳决策、成本与收益：');
for i = 1:size(data,1)
    fprintf('情况 %d: 最小成本 = %.2f, 最大收益 = %.2f, 决策 = %s\n', i, results(i),
profits(i), decisions(i));
end


% 定义递归函数，动态评估每一步的成本和收益
function [total_cost, total_profit] = evaluate_cost_and_profit(part1_detected,
part2_detected, final_detected, disassemble, i, data)
    % 读取数据
    defect_part1 = data(i,1);  % 零配件 1 次品率
    price_part1 = data(i,2);   % 零配件 1 购买单价
    detect_cost_part1 = data(i,3);  % 零配件 1 检测成本

    defect_part2 = data(i,4);  % 零配件 2 次品率
    price_part2 = data(i,5);   % 零配件 2 购买单价
    detect_cost_part2 = data(i,6);  % 零配件 2 检测成本

    defect_final = data(i,7);  % 成品次品率
    assembly_cost = data(i,8); % 成品装配成本
    detect_cost_final = data(i,9); % 成品检测成本

    market_price = data(i,10); % 市场售价
```

```
    replacement_cost = data(i,11); % 调换损失
    disassemble_cost = data(i,12); % 拆解费用

    % 计算不同决策的成本

    % 1. 零件 1 检测与否的成本
    if part1_detected
        part1_cost = detect_cost_part1 + price_part1; % 检测并丢弃不合格品
        defect_part1_prob = 0; % 检测后次品率为 0
    else
        part1_cost = price_part1; % 未检测则直接使用
        defect_part1_prob = defect_part1;
    end

    % 2. 零件 2 检测与否的成本
    if part2_detected
        part2_cost = detect_cost_part2 + price_part2; % 检测并丢弃不合格品
        defect_part2_prob = 0; % 检测后次品率为 0
    else
        part2_cost = price_part2; % 未检测则直接使用
        defect_part2_prob = defect_part2;
    end

    % 3. 成品检测与否的成本
    combined_defect_prob = 1 - (1 - defect_part1_prob) * (1 - defect_part2_prob)
* (1 - defect_final); % 成品不合格概率
    if final_detected
        final_cost = detect_cost_final + assembly_cost; % 检测成品
        final_defect_prob = 0; % 检测后的成品次品率为 0
    else
        final_cost = assembly_cost; % 不检测成品
        final_defect_prob = combined_defect_prob;
    end

    % 4. 退换与拆解的成本
    if disassemble
        % 计算条件概率：基于成品不合格的前提，分别计算零件 1 和零件 2 不合格的概率
        if ~part1_detected && ~part2_detected
            % 情况：1 和 2 都没有检测
            P_defect_part1_given_final = defect_part1 / (1 - (1 - defect_part1) *
(1 - defect_part2) * (1 - defect_final));
            P_defect_part2_given_final = defect_part2 / (1 - (1 - defect_part1) *
(1 - defect_part2) * (1 - defect_final));
        elseif part1_detected && ~part2_detected
            % 情况：1 检测过，2 没有检测
            P_defect_part1_given_final = 0; % 零件 1 已经检测合格
            P_defect_part2_given_final = defect_part2 / (1 - (1 - defect_part2) *
(1 - defect_final));
        elseif ~part1_detected && part2_detected
            % 情况：1 没有检测，2 检测过
            P_defect_part1_given_final = defect_part1 / (1 - (1 - defect_part1) *
(1 - defect_final));
            P_defect_part2_given_final = 0; % 零件 2 已经检测合格
```

```
        else
            % 情况：1 和 2 都检测过
            P_defect_part1_given_final = 0;
            P_defect_part2_given_final = 0;
        end

        % 根据条件概率计算合格零件的回收收益
        part1_recovery = (1 - P_defect_part1_given_final) * price_part1;
        part2_recovery = (1 - P_defect_part2_given_final) * price_part2;
        disassemble_benefit = part1_recovery + part2_recovery;

        % 检测成本只在未检测时计算
        part1_detect_cost = 0;
        part2_detect_cost = 0;

        if ~part1_detected
            part1_detect_cost = detect_cost_part1;
        end

        if ~part2_detected
            part2_detect_cost = detect_cost_part2;
        end

        % 计算修理成本，扣除合格零件回收收益
        repair_cost = final_defect_prob * (replacement_cost + disassemble_cost +
part1_detect_cost + part2_detect_cost - disassemble_benefit);
    else
        % 不拆解，直接报废，修理成本为调换损失
        repair_cost = final_defect_prob * replacement_cost;
    end



    % 总成本
    total_cost = part1_cost + part2_cost + final_cost + repair_cost;

    % 计算收益
    successful_final_prob = (1 - defect_part1_prob) * (1 - defect_part2_prob) *
(1 - defect_final);
    total_profit = successful_final_prob * market_price - total_cost; % 成品销售收
益 - 成本
end


% 布尔值转字符串函数
function str = bool_to_str(val)
    if val
        str = '检测';
    else
        str = '不检测';
    end
end
function str = whether_disassemble(val)
```

```
    if val
        str = '拆解';
    else
        str = '不拆解';
    end
     end
```

问题二中六种情况下所有策略收益条形图



问题三模拟退火算法 matlab 代码

```
% 主脚本：执行模拟退火优化过程并调用相关函数
    clc;
clear;
close all;
```

```
% 调用模拟退火函数
[best_solution, best_profit, iter_results] = simulated_annealing();


% 输出结果
disp('优化后的解:');
disp(best_solution);
disp('优化后的最大收益:');
disp(best_profit);


% 保存到 Excel
writetable(iter_results, 'wenti3jieguo.xlsx');


% 模拟退火函数
function [best_solution, best_profit, iter_results] = simulated_annealing()
    % 参数设置
    num_components = 8; % 零配件的数量
    num_subproducts = 3; % 半成品的数量
    num_decisions = num_components + num_subproducts + 5; % 决策变量的总数，包括零配
件、半成品和额外的 5 个决策变量

    num_iterations = 10; % 每个温度下的迭代次数

    % 初始化数据
    % 零配件数据 [缺陷概率, 采购成本, 检测成本]
    component_data = [
        0.10, 2, 1;
        0.10, 8, 1;
        0.10, 12, 2;
        0.10, 2, 1;
        0.10, 8, 1;
        0.10, 12, 2;
        0.10, 8, 1;
        0.10, 12, 2
    ];

    % 半成品数据 [缺陷概率, 组装成本, 检测成本, 拆解成本]
    subproduct_data = [
        0.10, 8, 4, 6;
        0.10, 8, 4, 6;
        0.10, 8, 4, 6
    ];

    % 成品数据 [缺陷概率, 组装成本, 检测成本, 拆解成本, 市场价格, 置换成本]
    final_product_data = [
        0.10, 8, 6, 10, 200, 40
    ];

    % 初始解
    % 随机生成初始解（0 或 1）
```

```
    current_solution = randi([0, 1], num_decisions, 1);
    % 计算当前解的利润
    current_profit = profit_function(current_solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);
    % 初始化最佳解和最佳利润
    best_solution = current_solution;
    best_profit = current_profit;

    % 模拟退火参数
    T = 1000; % 初始温度
    T_min = 1; % 终止温度
    alpha = 0.99; % 温度衰减率

    % 初始化输出表格
    iter_results = table('Size', [0, 6], 'VariableTypes', {'double', 'double',
'double', 'cell', 'cell', 'double'}, ...
                         'VariableNames', {'Iteration', 'Current_Profit',
'Best_Profit', 'Current_Solution', 'Best_Solution',
'Final_Product_Success_Rate'});


    % 保存迭代结果
    iteration_count = 0;
    while T > T_min
        for i = 1:num_iterations
            % 生成邻域解
            new_solution = neighbor_solution(current_solution);
            % 计算新解的利润
            new_profit = profit_function(new_solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);

            % 计算接受新解的概率
            if new_profit > current_profit
                accept = true; % 新解更好，则接受新解
            else
                % 否则，根据概率决定是否接受新解
                accept = rand < exp((new_profit - current_profit) / T);
            end

            if accept
                % 更新当前解和当前利润
                current_solution = new_solution;
                current_profit = new_profit;
                % 更新最佳解和最佳利润
                if current_profit > best_profit
                    best_solution = current_solution;
                    best_profit = current_profit;
                end
            end

            % 计算最终产品的合格率
```

```
            prob_success = calculate_success_probability(current_solution,
num_components, num_subproducts, component_data, subproduct_data,
final_product_data);

            % 保存迭代结果
            iteration_count = iteration_count + 1;
            new_row = table(iteration_count, current_profit, best_profit,
{mat2str(current_solution')}, {mat2str(best_solution')}, prob_success, ...
                        'VariableNames', iter_results.Properties.VariableNames);
            iter_results = [iter_results; new_row];
        end

        % 降低温度
        T = T * alpha;
    end
end


% 生成邻域解的函数
function new_solution = neighbor_solution(solution)
    % 随机选择一个或多个位置进行翻转
    num_decisions = length(solution);
    num_changes = randi([1, min(3, num_decisions)]); % 随机选择 1 到 3 个位置进行翻转
    % 生成一个随机的翻转位置索引
    indices_to_flip = randperm(num_decisions, num_changes);
    % 生成邻域解
    new_solution = solution;
    new_solution(indices_to_flip) = 1 - new_solution(indices_to_flip); % 翻转选择
的位置
end


% 计算利润的函数
function total_profit = profit_function(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data)
    % 提取数据
    defect_prob_component = component_data(:, 1); % 零配件缺陷概率
    purchase_cost_component = component_data(:, 2); % 零配件采购成本
    detect_cost_component = component_data(:, 3); % 零配件检测成本

    defect_prob_subproduct = subproduct_data(:, 1); % 半成品缺陷概率
    assembly_cost_subproduct = subproduct_data(:, 2); % 半成品组装成本
    detect_cost_subproduct = subproduct_data(:, 3); % 半成品检测成本
    disassemble_cost_subproduct = subproduct_data(:, 4); % 半成品拆解成本

    defect_prob_final = final_product_data(1); % 成品缺陷概率
    assembly_cost_final = final_product_data(2); % 成品组装成本
    detect_cost_final = final_product_data(3); % 成品检测成本
    disassemble_cost_final = final_product_data(4); % 成品拆解成本
    market_price_final = final_product_data(5); % 成品市场价格
    replacement_cost_final = final_product_data(6); % 成品置换成本

    % 计算零配件成本
```

```
    part_detected = solution(1:num_components); % 零配件是否被检测
    cost_component = sum(part_detected .* (purchase_cost_component +
detect_cost_component)) + ...
                    sum(~part_detected .* purchase_cost_component);

    % 计算半成品成本
    subproduct_detected = solution(num_components + (1:num_subproducts)); % 半成
品是否被检测
    cost_subproduct = 0;
    for i = 1:num_subproducts
        if subproduct_detected(i)
            cost_subproduct = cost_subproduct + detect_cost_subproduct(i) +
assembly_cost_subproduct(i);
        else
            cost_subproduct = cost_subproduct + assembly_cost_subproduct(i);
        end
    end

    % 计算半成品拆解成本
    disassemble_subproduct = solution(num_components + num_subproducts +
1:num_components + num_subproducts + 3); % 半成品拆解决策
    disassemble_cost = zeros(num_subproducts, 1);
    prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data);

    for i = 1:num_subproducts
        if disassemble_subproduct(i)
            disassemble_cost(i) = disassemble_cost_subproduct(i);
            component_indices = get_component_indices(i);
            for idx = component_indices
                if part_detected(idx)
                    disassemble_cost(i) = disassemble_cost(i) -
purchase_cost_component(idx);
                else
                    disassemble_cost(i) = disassemble_cost(i) +
detect_cost_component(idx) - (1 - defect_prob_component(idx)) *
purchase_cost_component(idx);
                end
            end
        end
    end
    % 计算半成品的拆解成本，并不一定拆解，乘上概率
    disassemble_subproduct_cost = 0;
    for i = 1:num_subproducts
        disassemble_subproduct_cost = disassemble_cost(i) * (1-
prob_subproduct(i)) + disassemble_subproduct_cost;
    end

    % 计算成品成本
    final_detected = solution(num_components + num_subproducts + 4); % 成品是否被
检测
    if final_detected
        cost_final = detect_cost_final + assembly_cost_final;
```

```
    else
        cost_final = assembly_cost_final;
    end

    % 计算成品拆解
    disassemble_final = solution(end); % 成品拆解决策
    cost_disassemble_final = 0;
    if disassemble_final
        cost_disassemble_final = disassemble_cost_final;
        for i = 1:num_subproducts
            if solution(num_components + i)
                component_indices = get_component_indices(i);
                recovery_cost = sum(purchase_cost_component(component_indices)) +
assembly_cost_subproduct(i);
                cost_disassemble_final = cost_disassemble_final - recovery_cost;
            else
                prob_subproduct_pass = prob_subproduct(i);
                component_indices = get_component_indices(i);
                recovery_cost = (sum(purchase_cost_component(component_indices)) +
assembly_cost_subproduct(i)) * prob_subproduct_pass;
                cost_disassemble_final = cost_disassemble_final - recovery_cost;
            end
        end
    end


    % 不包含置换的总费用
    total_cost = cost_component + cost_subproduct + cost_final;

    % 计算成功概率
    prob_success = calculate_success_probability(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);

    % 计算调换费用的概率
    prob_defective_final = 1 - prob_success;

    % 计算总利润
    total_profit = prob_success * market_price_final - total_cost -
(replacement_cost_final + cost_disassemble_final) * prob_defective_final -
disassemble_subproduct_cost;
end


% 计算半成品合格概率的函数
function prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data)
    defect_prob_component = component_data(:, 1); % 零配件缺陷概率
    defect_prob_subproduct = subproduct_data(:, 1); % 半成品缺陷概率

    detected_components = solution(1:num_components); % 零配件是否被检测
    prob_component_pass = zeros(num_components, 1);
    for i = 1:num_components
        if detected_components(i)
```

```
            prob_component_pass(i) = 1; % 如果零配件被检测，其合格概率为 1
        else
            prob_component_pass(i) = 1 - defect_prob_component(i); % 否则，按照缺陷
概率计算合格概率
        end
    end


    subproduct_detected = solution(num_components + (1:num_subproducts)); % 半成
品是否被检测
    prob_subproduct = zeros(num_subproducts, 1);
    for i = 1:num_subproducts
        if subproduct_detected(i)
            prob_subproduct(i) = 1; % 如果半成品被检测，其合格概率为 1
        else
            component_indices = get_component_indices(i); % 获取该半成品的组件索引
            prob_subproduct(i) = (1 - defect_prob_subproduct(i)) *
prod(prob_component_pass(component_indices)); % 半成品的合格概率
        end
    end
end


% 计算最终产品成功概率的函数
function prob_success = calculate_success_probability(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data)
    % 计算半成品合格概率
    prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data);

    % 计算最终产品的成功概率
    defect_prob_final = final_product_data(1); % 成品缺陷概率
    prob_final = (1 - defect_prob_final) * prod(prob_subproduct); % 最终产品的成功
概率
    prob_success = prob_final;
end


% 获取组件索引的函数
function indices = get_component_indices(subproduct_index)
    % 根据半成品索引返回组件索引
    % 假设每个半成品涉及的组件索引为 [1:3, 4:6, 7:8]，实际情况需根据数据修改
    if subproduct_index == 1
        indices = 1:3;
    elseif subproduct_index == 2
        indices = 4:6;
    elseif subproduct_index == 3
        indices = 7:8;
    else
        indices = [];
    end
end
```

问题四重解问题二 matlab 代码

```
% 清理工作区和图形窗口
close all; % 关闭所有图形窗口
clc; % 清空命令行
clear; % 清除工作区中的所有变量
% 设置随机种子
rng('shuffle');
% 定义抽样次数
n = 139; % 每个样本的抽样次数

% 定义参数，包括次品率、购买单价、检测成本、装配成本、市场售价、调换损失、拆解费用
% 每一行代表一种参数配置
data = [
    0.10, 4, 2, 0.10, 18, 3, 0.10, 6, 3, 56, 6, 5;
    0.20, 4, 2, 0.20, 18, 3, 0.20, 6, 3, 56, 6, 5;
    0.10, 4, 2, 0.10, 18, 3, 0.10, 6, 3, 56, 30, 5;
    0.20, 4, 1, 0.20, 18, 1, 0.20, 6, 2, 56, 30, 5;
    0.10, 4, 8, 0.20, 18, 1, 0.10, 6, 2, 56, 10, 5;
    0.05, 4, 2, 0.05, 18, 3, 0.05, 6, 3, 56, 10, 40;
];


% 获取数据矩阵的行数
num_rows = size(data, 1);


%根据题目所给的每个标称值来进行二项分布抽样
% 循环遍历每一行数据
for i = 1:num_rows
    % 从数据中提取次品率
    p_fail_1 = data(i, 1); % 零件 1 的次品率
    p_fail_2 = data(i, 4); % 零件 2 的次品率
    p_fail_product = data(i, 7); % 成品的次品率

    % 计算零件和成品的合格概率
    p_1 = 1 - p_fail_1; % 零件 1 的合格概率
    p_2 = 1 - p_fail_2; % 零件 2 的合格概率
    p_product = 1 - p_fail_product; % 成品的合格概率

    % 使用二项分布生成样本数据
    num_failures_1 = binornd(n, 1 - p_1); % 零件 1 的次品数量
    num_failures_2 = binornd(n, 1 - p_2); % 零件 2 的次品数量
    num_failures_product = binornd(n, 1 - p_product); % 成品的次品数量

    % 计算更新后的次品率（以百分比表示）
    new_defect_rate_1 = num_failures_1 / n * 100;
    new_defect_rate_2 = num_failures_2 / n * 100;
    new_defect_rate_product = num_failures_product / n * 100;

    % 更新数据矩阵中的次品率（将百分比转化为比例）
    data(i, 1) = new_defect_rate_1 / 100; % 零件 1 的次品率
    data(i, 4) = new_defect_rate_2 / 100; % 零件 2 的次品率
```

```
    data(i, 7) = new_defect_rate_product / 100; % 成品的次品率

    % 输出模拟结果
    fprintf('第 %d 行 - 零件 1 的新次品率为: %.2f%%\n', i, new_defect_rate_1);
    fprintf('第 %d 行 - 零件 2 的新次品率为: %.2f%%\n', i, new_defect_rate_2);
    fprintf('第 %d 行 - 成品的新次品率为: %.2f%%\n', i, new_defect_rate_product);
end


% 显示更新后的数据矩阵
disp('更新后的次品率矩阵:');
disp('   零件 1       零件 2       成品');
disp([data(:,1) data(:,4) data(:,7)]);


% 结果保存变量初始化
results = zeros(size(data,1), 1); % 成本
profits = zeros(size(data,1), 1); % 最终收益
decisions = strings(size(data,1), 4); % 记录每一步的检测和拆解决策


% 遍历所有数据行
for i = 1:size(data,1)
    % 定义不同的决策组合，四层嵌套循环分别代表：零件 1 检测、零件 2 检测、成品检测、成品是否拆解
    min_cost = Inf; % 初始化最小成本为无穷大
    max_profit = -Inf; % 初始化最大收益为负无穷大
    best_decision = ''; % 初始化最佳决策为一个空字符串

    for detect_part1 = [true, false]
        for detect_part2 = [true, false]
            for detect_final = [true, false]
                for disassemble = [true, false]
                    % 计算当前决策组合的总成本和收益
                    [current_cost, current_profit] =
evaluate_cost_and_profit(detect_part1, detect_part2, detect_final, disassemble,
i, data);

                    % 判断是否为最优决策
                    if current_cost < min_cost
                        min_cost = current_cost;
                    end
                    if current_profit > max_profit
                        max_profit = current_profit;
                        best_decision = sprintf('Part1: %s, Part2: %s, Final: %s,
Disassemble: %s', ...
                                                bool_to_str(detect_part1),
bool_to_str(detect_part2), bool_to_str(detect_final),
whether_disassemble(disassemble));
                    end
                end
            end
        end
    end
```

```
    % 保存最优决策
    results(i) = min_cost;
    profits(i) = max_profit;
    decisions(i) = best_decision;
end


% 输出结果，打印决策及其对应的成本，利益
disp('最佳决策、成本与收益：');
for i = 1:size(data,1)
    fprintf('情况 %d: 最小成本 = %.2f, 最大收益 = %.2f, 决策 = %s\n', i, results(i),
profits(i), decisions(i));
end


% 定义递归函数，动态评估每一步的成本和收益
function [total_cost, total_profit] = evaluate_cost_and_profit(part1_detected,
part2_detected, final_detected, disassemble, i, data)
    % 读取数据
    defect_part1 = data(i,1);  % 零配件 1 次品率
    price_part1 = data(i,2);   % 零配件 1 购买单价
    detect_cost_part1 = data(i,3);  % 零配件 1 检测成本

    defect_part2 = data(i,4);  % 零配件 2 次品率
    price_part2 = data(i,5);   % 零配件 2 购买单价
    detect_cost_part2 = data(i,6);  % 零配件 2 检测成本

    defect_final = data(i,7);  % 成品次品率
    assembly_cost = data(i,8); % 成品装配成本
    detect_cost_final = data(i,9); % 成品检测成本

    market_price = data(i,10); % 市场售价
    replacement_cost = data(i,11); % 调换损失
    disassemble_cost = data(i,12); % 拆解费用

    % 计算不同决策的成本

    % 1. 零件 1 检测与否的成本
    if part1_detected
        part1_cost = detect_cost_part1 + price_part1; % 检测并丢弃不合格品
        defect_part1_prob = 0; % 检测后次品率为 0
    else
        part1_cost = price_part1; % 未检测则直接使用
        defect_part1_prob = defect_part1;
    end

    % 2. 零件 2 检测与否的成本
    if part2_detected
        part2_cost = detect_cost_part2 + price_part2; % 检测并丢弃不合格品
        defect_part2_prob = 0; % 检测后次品率为 0
    else
        part2_cost = price_part2; % 未检测则直接使用
        defect_part2_prob = defect_part2;
```

```
    end

    % 3. 成品检测与否的成本
    combined_defect_prob = 1 - (1 - defect_part1_prob) * (1 - defect_part2_prob)
* (1 - defect_final); % 成品不合格概率
    if final_detected
        final_cost = detect_cost_final + assembly_cost; % 检测成品
        final_defect_prob = 0; % 检测后的成品次品率为 0
    else
        final_cost = assembly_cost; % 不检测成品
        final_defect_prob = combined_defect_prob;
    end

    % 4. 退换与拆解的成本
    if disassemble
        % 计算条件概率：基于成品不合格的前提，分别计算零件 1 和零件 2 不合格的概率
        if ~part1_detected && ~part2_detected
            % 情况：1 和 2 都没有检测
            P_defect_part1_given_final = defect_part1 / (1 - (1 - defect_part1) *
(1 - defect_part2) * (1 - defect_final));
            P_defect_part2_given_final = defect_part2 / (1 - (1 - defect_part1) *
(1 - defect_part2) * (1 - defect_final));
        elseif part1_detected && ~part2_detected
            % 情况：1 检测过，2 没有检测
            P_defect_part1_given_final = 0; % 零件 1 已经检测合格
            P_defect_part2_given_final = defect_part2 / (1 - (1 - defect_part2) *
(1 - defect_final));
        elseif ~part1_detected && part2_detected
            % 情况：1 没有检测，2 检测过
            P_defect_part1_given_final = defect_part1 / (1 - (1 - defect_part1) *
(1 - defect_final));
            P_defect_part2_given_final = 0; % 零件 2 已经检测合格
        else
            % 情况：1 和 2 都检测过
            P_defect_part1_given_final = 0;
            P_defect_part2_given_final = 0;
        end

        % 根据条件概率计算合格零件的回收收益
        part1_recovery = (1 - P_defect_part1_given_final) * price_part1;
        part2_recovery = (1 - P_defect_part2_given_final) * price_part2;
        disassemble_benefit = part1_recovery + part2_recovery;

        % 检测成本只在未检测时计算
        part1_detect_cost = 0;
        part2_detect_cost = 0;

        if ~part1_detected
            part1_detect_cost = detect_cost_part1;
        end

        if ~part2_detected
            part2_detect_cost = detect_cost_part2;
```

```
        end

        % 计算修理成本，扣除合格零件回收收益
        repair_cost = final_defect_prob * (replacement_cost + disassemble_cost +
part1_detect_cost + part2_detect_cost - disassemble_benefit);
    else
        % 不拆解，直接报废，修理成本为调换损失
        repair_cost = final_defect_prob * replacement_cost;
    end


    % 计算总成本
    total_cost = part1_cost + part2_cost + final_cost + repair_cost;

    % 计算收益
    successful_final_prob = (1 - defect_part1_prob) * (1 - defect_part2_prob) *
(1 - defect_final);
    total_profit = successful_final_prob * market_price - total_cost; % 成品销售收
益 - 成本
end


% 布尔值转字符串函数
function str = bool_to_str(val)
    % 将布尔值转换为'检测'或'不检测'
    if val
        str = '检测';
    else
        str = '不检测';
    end
end


% 判断是否拆解的函数
function str = whether_disassemble(val)
    % 将布尔值转换为'拆解'或'不拆解'
    if val
        str = '拆解';
    else
        str = '不拆解';
    end
end
```

问题四重解问题三 matlab 代码

```
% 主脚本：执行模拟退火优化过程并调用相关函数
clc;
clear;
close all;


% 设置随机种子
rng('shuffle');
```

```
% 定义抽样次数和标称概率
n = 139; % 抽样次数
p = 0.9; % 标称概率为 0.9


%模拟抽样检测过程
% 计算每个样本的次品率
for i = 1:12
    defected_num = binornd(n, 1 - p); % 生成一个次品数量的随机数
    defected_rate(i) = defected_num / n; % 计算次品率
end




%抽样结果显示
% 显示不同类别的次品率
disp('零件次品率');
disp(defected_rate(1:8)');
disp('半成品次品率');
disp(defected_rate(9:11)');
disp('成品次品率');
disp(defected_rate(12)');


% 调用模拟退火函数进行优化，传入次品率的数据替换
[best_solution, best_profit, iter_results] =
simulated_annealing(defected_rate);


% 输出优化后的结果
disp('优化后的解:');
disp(best_solution);
disp('优化后的最大收益:');
disp(best_profit);


% 保存结果到 Excel 文件
writetable(iter_results, 'wenti4_2.xlsx');


% 模拟退火函数


function [best_solution, best_profit, iter_results] =
simulated_annealing(defected_rate)
    % 设置模拟退火算法的参数
    num_components = 8; % 零配件数量
    num_subproducts = 3; % 半成品数量
    num_decisions = num_components + num_subproducts + 5; % 总决策变量数，包括半成品
拆解、成品检测、成品拆解
```

```
    num_iterations = 10; % 每个温度下的迭代次数

    % 初始化数据
    % 零配件数据：每一行表示一个配件的次品率、购买单价和检测成本
    component_data = [
        defected_rate(1), 2, 1;
        defected_rate(2), 8, 1;
        defected_rate(3), 12, 2;
        defected_rate(4), 2, 1;
        defected_rate(5), 8, 1;
        defected_rate(6), 12, 2;
        defected_rate(7), 8, 1;
        defected_rate(8), 12, 2
    ];

    % 半成品数据：每一行表示一个半成品的次品率、装配成本、检测成本和拆解费用
    subproduct_data = [
        defected_rate(9), 8, 4, 6;
        defected_rate(10), 8, 4, 6;
        defected_rate(11), 8, 4, 6
    ];

    % 成品数据：包括次品率、装配成本、检测成本、拆解费用、市场售价和调换损失
    final_product_data = [
        defected_rate(12), 8, 6, 10, 200, 40
    ];

    % 初始化当前解
    % 随机生成一个初始解
    current_solution = randi([0, 1], num_decisions, 1);
    % 计算当前解的利润
    current_profit = profit_function(current_solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);
    % 将当前解设置为最佳解
    best_solution = current_solution;
    best_profit = current_profit;

    % 模拟退火算法参数
    T = 1000; % 初始温度
    T_min = 1; % 终止温度
    alpha = 0.99; % 温度衰减率

    % 初始化迭代结果表格
    iter_results = table('Size', [0, 6], 'VariableTypes', {'double', 'double',
'double', 'cell', 'cell', 'double'}, ...
                         'VariableNames', {'Iteration', 'Current_Profit',
'Best_Profit', 'Current_Solution', 'Best_Solution',
'Final_Product_Success_Rate'});


    % 记录迭代次数
    iteration_count = 0;
```

```
    % 模拟退火过程
    while T > T_min
        for i = 1:num_iterations
            % 生成邻域解
            new_solution = neighbor_solution(current_solution);
            % 计算邻域解的利润
            new_profit = profit_function(new_solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);

            % 根据接受概率决定是否接受新解
            if new_profit > current_profit
                accept = true; % 如果新解的利润更高，则接受新解
            else
                % 根据概率决定是否接受新解
                accept = rand < exp((new_profit - current_profit) / T);
            end

            if accept
                % 更新当前解和利润
                current_solution = new_solution;
                current_profit = new_profit;
                % 更新最佳解和最佳利润
                if current_profit > best_profit
                    best_solution = current_solution;
                    best_profit = current_profit;
                end
            end

            % 计算最终产品的成功概率
            prob_success = calculate_success_probability(current_solution,
num_components, num_subproducts, component_data, subproduct_data,
final_product_data);

            % 保存当前迭代的结果
            iteration_count = iteration_count + 1;
            new_row = table(iteration_count, current_profit, best_profit,
{mat2str(current_solution')}, {mat2str(best_solution')}, prob_success, ...
                        'VariableNames', iter_results.Properties.VariableNames);
            iter_results = [iter_results; new_row]; % 将新结果添加到结果表格中
        end

        % 降低温度
        T = T * alpha; % 根据衰减率更新温度
    end
end


% 生成邻域解的函数
function new_solution = neighbor_solution(solution)
    % 随机选择一个或多个位置进行翻转
    num_decisions = length(solution);
    num_changes = randi([1, min(3, num_decisions)]); % 随机选择 1 到 3 个位置进行翻转
    % 生成一个随机的翻转位置索引
```

```
    indices_to_flip = randperm(num_decisions, num_changes);
    % 生成邻域解
    new_solution = solution;
    new_solution(indices_to_flip) = 1 - new_solution(indices_to_flip); % 翻转选择
的位置
end


% 计算总利润的函数
function total_profit = profit_function(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data)
    % 从数据中提取零配件、半成品和成品的信息
    defect_prob_component = component_data(:, 1); % 零配件的次品率
    purchase_cost_component = component_data(:, 2); % 零配件的购买单价
    detect_cost_component = component_data(:, 3); % 零配件的检测成本

    defect_prob_subproduct = subproduct_data(:, 1); % 半成品的次品率
    assembly_cost_subproduct = subproduct_data(:, 2); % 半成品的装配成本
    detect_cost_subproduct = subproduct_data(:, 3); % 半成品的检测成本
    disassemble_cost_subproduct = subproduct_data(:, 4); % 半成品的拆解费用

    defect_prob_final = final_product_data(1); % 成品的次品率
    assembly_cost_final = final_product_data(2); % 成品的装配成本
    detect_cost_final = final_product_data(3); % 成品的检测成本
    disassemble_cost_final = final_product_data(4); % 成品的拆解费用
    market_price_final = final_product_data(5); % 成品的市场售价
    replacement_cost_final = final_product_data(6); % 成品的调换损失

    % 计算零配件的总成本
    part_detected = solution(1:num_components); % 零配件的检测状态
    cost_component = sum(part_detected .* (purchase_cost_component +
detect_cost_component)) + ...
                     sum(~part_detected .* purchase_cost_component); % 计算零配件总成
本

    % 计算半成品的总成本
    subproduct_detected = solution(num_components + (1:num_subproducts)); % 半成
品的检测状态
    cost_subproduct = 0; % 初始化半成品的总成本
    for i = 1:num_subproducts
        if subproduct_detected(i)
            cost_subproduct = cost_subproduct + detect_cost_subproduct(i) +
assembly_cost_subproduct(i); % 检测并装配
        else
            cost_subproduct = cost_subproduct + assembly_cost_subproduct(i); % 仅
装配
        end
    end

    % 计算半成品拆解的成本
    disassemble_subproduct = solution(num_components + num_subproducts +
1:num_components + num_subproducts + 3); % 半成品的拆解状态
    disassemble_cost = zeros(num_subproducts, 1); % 初始化拆解成本
```

```
    prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data); % 计算半成品合
格概率

    for i = 1:num_subproducts  %遍历三个半成品
        if disassemble_subproduct(i)
            disassemble_cost(i) = disassemble_cost_subproduct(i); % 如果拆解，直接添
加拆解费用
            component_indices = get_component_indices(i); % 获取涉及的组件索引
            for idx = component_indices
                if part_detected(idx)
                    disassemble_cost(i) = disassemble_cost(i) -
purchase_cost_component(idx); % 已检测的组件减少拆解成本
                else
                    disassemble_cost(i) = disassemble_cost(i) +
detect_cost_component(idx) - (1 - defect_prob_component(idx)) *
purchase_cost_component(idx); % 未检测到的组件增加拆解成本
                end
            end
        end
    end
    % 计算半成品拆解的总成本（乘以拆解概率）
    disassemble_subproduct_cost = 0;
    for i = 1:num_subproducts
        disassemble_subproduct_cost = disassemble_cost(i) * (1-
prob_subproduct(i)) + disassemble_subproduct_cost;
    end

    % 计算成品的总成本
    final_detected = solution(num_components + num_subproducts + 4); % 成品的检测
状态
    if final_detected
        cost_final = detect_cost_final + assembly_cost_final; % 检测并装配的总成本
    else
        cost_final = assembly_cost_final; % 仅装配的总成本
    end

    % 计算成品拆解的成本
    disassemble_final = solution(end); % 成品拆解状态
    cost_disassemble_final = 0; % 初始化成品拆解成本
    if disassemble_final
        cost_disassemble_final = disassemble_cost_final; % 如果拆解，直接添加拆解费用
        for i = 1:num_subproducts
            if solution(num_components + i)
                component_indices = get_component_indices(i); % 获取涉及的组件索引
                recovery_cost = sum(purchase_cost_component(component_indices)) +
assembly_cost_subproduct(i); % 计算恢复成本
                cost_disassemble_final = cost_disassemble_final - recovery_cost; %
减去恢复成本
            else
                prob_subproduct_pass = prob_subproduct(i); % 半成品合格的概率
                component_indices = get_component_indices(i); % 获取涉及的组件索引
```

```
            recovery_cost = (sum(purchase_cost_component(component_indices)) +
assembly_cost_subproduct(i)) * prob_subproduct_pass; % 计算恢复成本
            cost_disassemble_final = cost_disassemble_final - recovery_cost; %
减去恢复成本
        end
    end
end


% 计算总费用（不包含置换费用）
total_cost = cost_component + cost_subproduct + cost_final;

% 计算最终产品成功的概率
prob_success = calculate_success_probability(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data);

% 计算成品调换费用的概率
prob_defective_final = 1 - prob_success;

% 计算总利润
total_profit = prob_success * market_price_final - total_cost -
(replacement_cost_final + cost_disassemble_final) * prob_defective_final -
disassemble_subproduct_cost;
end


% 计算半成品合格概率的函数
function prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data)
% 从数据中提取组件和半成品的次品率
defect_prob_component = component_data(:, 1);
defect_prob_subproduct = subproduct_data(:, 1);

% 计算每个组件的通过概率
detected_components = solution(1:num_components);
prob_component_pass = zeros(num_components, 1);
for i = 1:num_components
    if detected_components(i)
        prob_component_pass(i) = 1; % 如果组件被检测，则通过概率为 1
    else
        prob_component_pass(i) = 1 - defect_prob_component(i); % 未检测组件的通
过概率
    end
end


% 计算每个半成品的合格概率
subproduct_detected = solution(num_components + (1:num_subproducts));
prob_subproduct = zeros(num_subproducts, 1);
for i = 1:num_subproducts
    if subproduct_detected(i)
        prob_subproduct(i) = 1; % 如果半成品被检测，则合格概率为 1
    else
        component_indices = get_component_indices(i); % 获取涉及的组件索引
```

```
            prob_subproduct(i) = (1 - defect_prob_subproduct(i)) *
prod(prob_component_pass(component_indices)); % 半成品的合格概率
        end
    end
end


% 计算最终产品成功概率的函数
function prob_success = calculate_success_probability(solution, num_components,
num_subproducts, component_data, subproduct_data, final_product_data)
    % 计算半成品的合格概率
    prob_subproduct = calculate_subproduct_probabilities(solution,
num_components, num_subproducts, component_data, subproduct_data);

    % 计算最终产品的成功概率
    defect_prob_final = final_product_data(1);
    prob_final = (1 - defect_prob_final) * prod(prob_subproduct); % 成品的成功概率
    prob_success = prob_final; % 返回成功概率
end


% 获取组件索引的函数
function indices = get_component_indices(subproduct_index)
    % 根据半成品的索引返回涉及的组件的索引
    % 根据实际情况填写每个半成品涉及的组件
    if subproduct_index == 1
        indices = 1:3; % 半成品 1 涉及组件 1, 2, 3
    elseif subproduct_index == 2
        indices = 4:6; % 半成品 2 涉及组件 4, 5, 6
    elseif subproduct_index == 3
        indices = 7:8; % 半成品 3 涉及组件 7, 8
    else
        indices = []; % 无效索引返回空
    end
end
```