基于利润最大化的多阶段决策模型

摘要

在《中国制造 2025》的战略指引下，电子制造企业通过技术创新优化生产流程，降低成本已成为必然趋势。本论文通过借助数学模型，设计抽样方案，建立决策模型，为企业生产过程中各个阶段提供决策建议（包括是否检测，是否拆解等），助力企业降本增效，提高市场竞争力。

针对问题一，企业在质量检验过程中面临的零配件次品率检测问题，论文提出了一种基于**截尾贯序最优检验的抽样方案**，该方案通过动态调整抽样策略，旨在有效减少检测成本。利用**动态规划**技术，我们对方案进行了优化求解。具体地，本方案以一批假设的 50 件零配件为研究对象，通过模拟分析得出：在 95% 的置信水平下，最优抽样方案推荐抽取 29 件零配件进行检测；而在 90% 的置信水平下，推荐抽样数量为 22 件。图示分析进一步验证了本方案在提高检测效率、降低成本方面的显著优势。

针对问题二，电子产品生产过程中的最优决策问题，论文建立了一个决策模型，以最大化企业的期望利润。模型涵盖了零配件检测、成品检测、成品装配以及不合格品的拆解和退换处理等关键环节，并考虑了检测、装配、拆解和调换损失的成本。通过引入 0-1 决策变量，模型计算出在不同检测策略下一批零件的期望利润，期望利润最高对应的策略即为最优生产决策方案。结果表明，不同的生产环境和次品率下，最优策略应根据具体情况灵活调整，以达到最大化利润的效果。

针对问题三，构建了一个**多阶段的决策模型**，旨在最大化企业的利润。我们综合考虑了零件购买成本、检测成本、装配成本、拆解成本以及调换损失，制定了在各个生产阶段是否进行检测和拆解的最优决策方案。对于含有两道工序和八个零配件的生产流程，为求解这一问题我们采用了**模拟退火算法**，通过状态空间的定义与能量函数（总成本）的优化，最终得到最优决策。具体而言，结果显示在所有零配件、半成品和成品均进行检测并对不合格品进行拆解的情况下，利润最大化。设定每种零件数量为 100，则最终求得最大利润为 3559.40 元。该模型为生产企业在实际操作中的多阶段决策问题提供了有效参考。

针对问题四，我们采用**蒙特卡洛模拟方法**估计出企业生产过程中的零配件、半成品和成品的次品率，并通过并行计算提高模拟的效率。利用抽样检测得到的次品率估计结果 0.0999，我们对问题二和问题三进行重新求解。具体而言，结果显示在所有零配件、半成品和成品均进行检测并对不合格品进行拆解的情况下，利润最大化。设定每种零件数量为 100，则最终求得最大利润为 3560.94 元。

最后，我们对模型进行了优缺点分析和推广。

关键字：截尾序贯最优检验　动态规划　模拟退火　多阶段决策　蒙特卡洛模拟

一、 问题重述

1.1 背景资料

随着中国电子制造业的蓬勃发展，国家对提升产品质量、促进产业转型升级提出了明确要求。近年来，随着《中国制造 2025》战略的实施，国家鼓励企业通过技术创新和流程优化来提高产品质量和生产效率。这不仅要求企业通过引进先进生产设备和技术，提升自动化水平，还需要企业在管理模式上不断创新，运用数学模型、数据分析等工具对生产流程进行优化，降低生产成本、减少次品率，提高市场竞争力。

电子产品通常由多个零配件组装而成，任何一个零配件的质量问题都会导致整个产品的不合格，因此，确保每个零配件的质量至关重要。然而，全面检测所有零配件不仅耗费时间和资源，还会显著增加企业的成本，因此设计高效的抽样检测方案成为必然选择。与此同时，成品组装后的检测、不合格成品的处理以及客户退货的应对，都是影响企业质量控制和运营成本的关键决策。通过科学的检测与决策方法，企业能够有效降低次品率，减少因次品造成的经济损失与品牌信誉损失，提升市场竞争力，并在复杂的供应链中提高生产效率，从而为企业在竞争激烈的市场中赢得长远发展提供决策支持。

1.2 问题要求

某企业生产一款畅销电子产品，需购买两种零配件进行装配。产品质量受零配件质量影响，但即使零配件合格，成品仍可能不合格。企业可选择对不合格成品进行报废或拆解。拆解后，零配件可重新使用，但需支付拆解费用。问题要求通过建立数学模型，探讨电子产品生产各环节的质量控制与决策问题。

（1）企业希望通过抽样检测确定零配件的质量是否符合供应商所说的标称标准。要求设计一个检测方案，以最少的抽样次数在不同信度下决定是否接受零配件。具体情形包括在 95% 的置信度下拒收次品率超过标称值（10%）的零配件，和在 90% 的置信度下接收次品率不超过标称值（10%）的零配件。

（2）在生产过程中，企业需决定是否对零配件、成品进行检测，以及是否对不合格成品进行拆解。决策涉及成本、次品率及市场售价等因素，目标是通过合理决策获取最大利润。问题探讨如何优化零配件检测、成品检测及不合格成品处理的决策。

（3）在多道工序和多种零配件的复杂生产流程中，需进一步扩展问题二中的决策方案，适应更多零配件和半成品的情况，分析各阶段的检测与拆解决策。

（4）假设零配件、半成品和成品的次品率均通过抽样检测方法获得，重新评估并制定问题二与问题三中的生产决策方案。

二、 问题分析

2.1 问题一分析

企业在采购零配件时，需要对供应商提供的零配件进行质量检验，确保其次品率不超过供应商声称的标称值。在此背景下，企业面临的核心问题是如何通过抽样检测方案，以尽量少的检测次数来确定是否接收这批零配件，并控制犯第一类错误（接受错误的次品率）和第二类错误（拒绝合格次品率）的概率，确保决策的科学性。

基于问题的背景和要求，适合使用序贯概率比检验（SPRT）方法来设计抽样方案。SPRT 方法具有高效性，灵活性，控制错误率等特点，可以通过动态调整抽样过程。在本问题中，进一步使用截断的 SPRT 方法，设定一个最大样本量，以避免因样本量无限增长而导致的成本过高或时间延误。这种截断方式能在控制成本的同时，保证检验的效率和准确性。在此基础上，通过动态规划法，我们可以求解最优决策策略，最小化期望样本量，最终获得在保证质量控制精度的前提下，最大限度减少检测成本的最优方案。

2.2 问题二分析

问题二要求为企业生产过程中的各个阶段制定优化决策方案。决策的目标是最大化企业的期望利润。具体问题包括：零配件检测决策；成品检测决策；不合格成品拆解决策；调换决策。这些决策将影响生产过程的最终利润。

由此，我们建立了成品次品率的计算模型，该公式综合考虑了零配件的检测情况及成品是否拆解的影响。通过计算成品装配数量及成品次品率得到期望利润，由成品收入与各项成本之差得到，其中包括零配件采购成本、检测成本、装配成本、拆解成本和调换损失。接着利用计算机优化工具和遍历算法对目标函数进行优化，对不同策略组合进行评估。

2.3 问题三分析

问题三要求针对生产过程中涉及的 n 道工序和 m 个零配件，结合已知的零配件、半成品和成品的次品率，构建一个生产过程的决策方案，并重复解决问题二中的情况。该问题可以视为一个复杂的多阶段决策问题，目的是最大化企业的利润。由于该问题涉及的决策变量较多，我们采用了模拟退火算法来进行求解。模拟退火算法通过不断调整决策变量，逐渐接近最优解。每次迭代时，会随机选择一个决策变量并改变其值（如是否检测零配件或是否拆解次品），并计算新的总成本。如果新解的成本更低，则更新当前解，从而逐步逼近最优解。

2.4 问题四分析

我们使用蒙特卡洛模拟方法估计零配件、半成品和成品抽样检测获得次品率，并重新评估并制定问题二与问题三中的生产决策方案。可以考虑到生产过程中的随机性和不确定性，提供一个较为可靠的次品率估计。

三、 符号说明

符号	意义
P_i	零配件 i 的次品率
P_f	成品的次品率
L	似然比（假设在样本下的相对支持程度）
$V(i,k)$	在状态 (i,k) 下的期望样本量
K	装配为成品的数量
C_i	零配件 i 的购买单价
C_a	单个装配成本
C_{ti}	零配件 i 的单个检测成本
C_t	成品的单个检测成本
S_m	成品的单个市场售价
L_r	不合格品的单个调换损失
C_d	不合格成品的单个拆解费用
P_m	将正品零配件装配后的产品次品率
E	期望利润
S	一批零配件每种的数量

表 **1**　符号说明表格

四、 问题一模型的建立及求解

4.1 基于截尾序贯最优检验的抽样方案建立

4.1.1 截尾序贯最优检验的抽样方法简介

1.SPRT 方法简介

序贯概率比抽样检验方案（SPRT）是一种基于概率比检验的序贯分析方法，它克服了传统固定样本量检验的局限性，适用于各种领域和问题的假设检验。与传统检验不同，SPRT 每次只抽取一个或多个样本，利用似然比统计量判断原假设和备择假设的概率大小，从而做出接受、拒绝或继续抽样的决策。

序贯概率比抽样检验方案（SPRT）首先由 Wald 于 1945 年提出，检验方案是由两条平行的直线构成[1]。在满足限制条件的所有检验方案中，SPRT 具有最少的平均试验次数的最优特性。[2] 由于 SPRT 具有结构清晰、易于理解、计算简便和操作方便的特点，它广泛应用于多个领域，如传感器故障检测、无线电通信系统、信号检测、甚至神经生物分析等。例如，Gao 等[3] 提出了一种双通道序贯概率比测试故障检测算法, 可以有效检测传感器缓慢增长的故障和突然的故障。Kira[4] 等从神经生物学角度，完成了基于 Wald 序贯概率比检验的神经实现。在认知无线电网络中，多假设序贯概率比检验也有不少应用，如顺序典型相关技术（S-CCT）方法，可以快速准确地估计活性 PU 的数量。[5]

2. 截尾的 SPRT 方法

截尾的序贯概率比检验（Truncated SPRT）是对传统 SPRT 方法的一种扩展，它在保留了 SPRT 方法高效性和灵活性的基础上，通过设定一个预先定义的最大样本量，来避免样本量无限增大的风险，使试验时间得到有效控制。

在 SPRT 的基础上，设定一个最大样本量 N，当样本量达到 N 时，无论检验结果如何，试验都将停止。截断 SPRT 仍然需要控制犯第一类和第二类错误的概率，这可以通过调整边界值或采用其他方法实现。

4.2 截尾的 SPRT 方案假设

- 独立性假设：假设每次抽取的零配件之间相互独立，即前一次的抽样结果不影响后一次的抽样结果。
- 无替换抽样：在抽取样本后，被抽样的零配件不会被放回，即每次抽样都是从剩余的未被抽样的零配件中进行。
- 同分布假设：假设零配件 **1** 和零配件 **2** 具有相同的次品率 P。
- 完全信息：在每次抽样后，我们能够立即且准确地获得该样本是否为次品的信息。

4.2.1 截尾的 SPRT 方案的具体描述

在截尾的 SPRT 方案中，我们的目标是确定一批零配件的次品率是否超过了预定的标称值 P_0。我们希望通过最少的抽样来做出决策，以减少检验成本。

1. 假设检验

我们设定零配件次品率 P，其标称值为 $P_0 = 10\%$。为检验供应商提供的零配件次品率是否符合标称值，建立如下假设：

$$H_0 : P \leq P_0 = 0.1 \tag{1}$$

$$H_1 : P > P_0 = 0.1 \tag{2}$$

（1）式表示零假设 H_0：假设零配件的次品率 P 符合供应商的声明，即次品率 P 小于 10%。（2）式表示备择假设 H_1：假设零配件的次品率 P 超过标称的 10%，即 P 大于 10%。

该假设检验用于判断是否接受或拒绝这批零配件。

2. 抽样方案

从一批零配件中逐一抽取样本，并对每个样本进行检测，记录每个样本是否为次品。每次抽样后，根据检测结果来计算似然比，以判断是否有足够证据拒绝或接受供应商所声称的次品率。

3. 停止规则

在本方案中，使用截尾贯序最优检验来决定是否继续抽样，截尾值和样本量相同。在截尾的 SPRT 方案中，停止规则是基于似然比函数来决定是否继续抽样。似然比函数 L 表示两种假设 H_0 和 H_1 在当前样本下的相对支持程度，公式为：

$$L(P_1, P_0) = \frac{{P_1}^d (1 - P_1)^{n-d}}{{P_0}^d (1 - P_0)^{n-d}} \tag{3}$$

其中，P_0 是供应商声明的次品率（在本方案中为 10%），P_1 是假设次品率超过标称值的情况下的检验标准。d 是检测到的不合格零件数量，n 是总检测样本量。该似然比用于计算两种假设下的观测结果，并根据其值来决定是否停止检测。

停止规则如下：

$$\begin{cases} L_i > A, & \text{试验停止，拒绝原假设} \\ L_i < B, & \text{试验停止，接受原假设} \\ B \leq L_i \leq A, & \text{继续下一次观测} \end{cases}$$

A，B 为边界值，可近似由 α 和 β 设定，计算公式为：

$$\begin{cases} A = \frac{1-\beta}{\alpha} \\ B = \frac{\beta}{1-\alpha} \end{cases} \tag{4}$$

A 是拒绝零假设（即认为次品率超标）的临界值，B 是接受零假设（即次品率不超标）的临界值。

4.3 基于截断的 SPRT 方案的模型求解

为了实现截断的 SPRT 方案，我们采用动态规划方法进行求解。

4.3.1 动态规划方程的建立

1. 定义问题

假设企业希望在最小化检测成本的前提下，判断一批零配件的次品率是否符合供应商声明的标称值 $P_0 = 10\%$。我们通过序贯概率比检验 (Sequential Probability Ratio Test, SPRT) 来动态决定是否继续抽样，还是在某一时刻接受或拒绝供应商的零配件。

2. 动态规划的相关定义

在动态规划中，通过对每个状态（检测到的样本数和观察到的次品数）进行递推计算，我们可以找到最优的决策路径。动态规划中的关键定义如下：

- **状态** (i, k): 表示已经抽取了 i 个样本，且观察到 k 个次品的状态。
- **决策** d: 每次抽样后可以做出的决策有三个：
  - 接受零假设 H_0 （认为次品率不超过标称值）。
  - 拒绝零假设 H_1 （认为次品率超过标称值）。
  - 继续抽样，直到有足够证据接受或拒绝某一假设。

3. 动态规划方程

为了在最小样本量下做出准确决策，我们引入已抽 i 个样本，并观测到 k 个次品时的最小期望样本量 $V(i, k)$ 的动态规划方程。该方程表示在状态 (i, k) 下，抽取更多样本的期望样本量。具体的递推关系如下：

$$V(i, k) = \min(R(i, k), S(i, k), A(i, k)) \tag{5}$$

其中：

- $R(i, k)$: 如果在状态 (i, k) 下拒绝零假设 H_0 的期望样本量为 i。
- $S(i, k)$: 如果在状态 (i, k) 下接受零假设 H_0 的期望样本量为 i。
- $A(i, k)$: 如果在状态 (i, k) 下选择继续抽样的期望样本量为当前样本量加上继续抽样的期望样本量。公式如下：

$$A(i, k) = 1 + \sum_{k_1=0}^{1} P(k_1 \mid i, k) \cdot V(i + 1, k_1) \tag{6}$$

其中 $P(k_1 \mid i,k)$ 是在已抽取 i 个样本且观察到 k 个次品的情况下，下一次抽样中出现次品的概率。

4. 边界条件设定

若 i 达到截尾值 n_0，必须做出决策, 此时 $V(n_0,k)$ 取到 n_0。

当 k 达到某阈值，使累计概率超过 α 或 β, 此时必须做出决策。

4.3.2 期望样本量（$V(i,k)$）的计算

期望样本量 $V(i,k)$ 是衡量序贯抽样效率的一个关键指标。它代表在不同的次品率下，企业进行检测时平均需要抽取的样本数量。基于前述的动态规划方程，可以计算得出 $V(i,k)$ 的值。通常情况下，序贯抽样的期望样本量显著小于固定样本量的抽样方案，因此在检测成本上有较大优势。

若供应商提供的产品总共有 50 件，声称次品率 $P_0 = 10\%$。通过动态规划法及计算，企业在不同的置信度下，分别需要抽取以下期望样本量：

- 在 95% 的置信度下认定零配件次品率超过标称值时：通过截断的 SPRT 方案，计算得出在 50 件总零配件中，需抽取 29 件进行检测。此时，若检测到的次品数量使得似然比 L>A，则可以在 95% 的信度下认为次品率超过标称值，并拒收该批零配件。企业可根据不同信度设定检测方案。
- 在 90% 的置信度下认定零配件次品率不超过标称值时：在相对较低的信度水平下，计算结果表明企业只需抽取 22 件零配件进行检测。若检测结果使得似然比 L<B，则可以在 90% 的信度下认为次品率不超过标称值，并接收该批零配件。

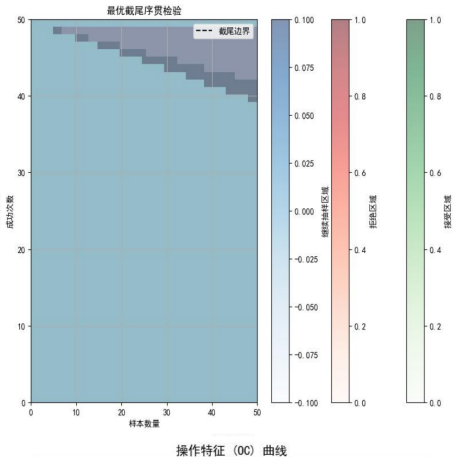

图 **1**　最优截尾序贯检验

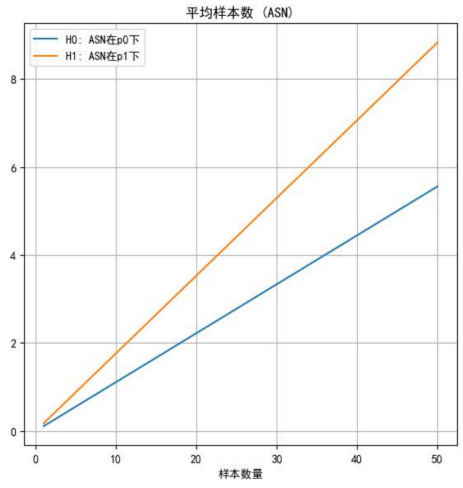

图 **2**　平均样本数

图 1 展示了最优截尾序贯检验方案，其中包含了接受域、拒绝域和继续抽样的区间。图 2 图展示了在不同假设条件下，检验方案所需的平均样本数量（ASN）。两条

曲线分别代表在原假设（H0）和备择假设（H1）下，随着样本数量的增加，ASN 的变化情况。可以看出，ASN 随着样本数量的增加而增加。

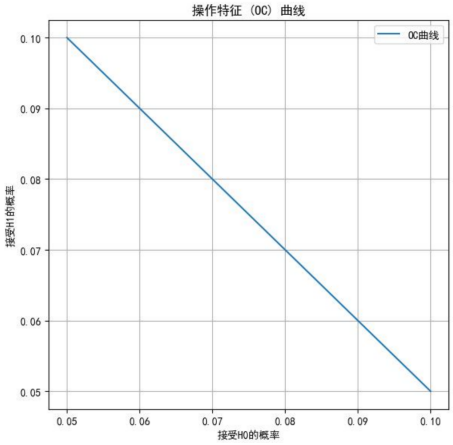

图 **3**　操作特征曲线

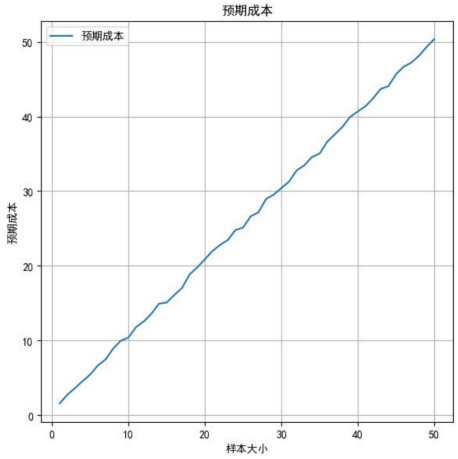

图 **4**　预期成本

图 3 中显示检验方案在不同假设条件下的犯错误的概率。曲线展示了从完全接受原假设到完全接受备择假设的概率变化，可以用来评估检验方案的性能。图 4 展示了不同样本大小下的预期成本，其中横轴表示样本大小，纵轴表示预期成本。

五、 问题二模型的建立与求解

5.1 模型假设

- **假设企业检测零配件按照批次进行，每批零件数量固定且不同零件数量相同。**例如设定一批次的零配件数量为 200，则代表一批零配件中的零配件 1 和零配件 2 数量均为 S=100，包括前一批未装配的零配件、前一批拆解回收的零配件。
- **假设顾客收到的所有不合格品都会被退回。**企业对于收到不合格品的顾客提供无条件调换，模型中假设所有的不合格品都会被退回。
- **假设检测操作是完美无误的，即检测结果总是准确的。**该假设保证了次品率不会受到检测结果影响。

5.2 模型的建立

5.2.1 决策模型的分析

在生产过程中，每批生产的零配件和成品中存在一定的次品率，而企业需要对是否检测以及是否拆解做出决策，以最大化每批期望利润。

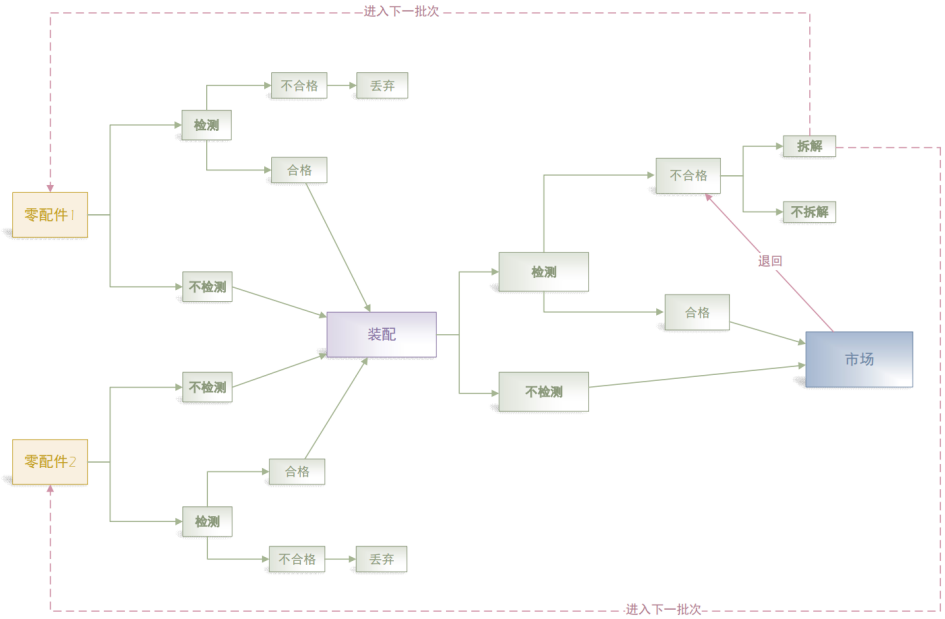

图 **5**　第二问流程图

从图中可以看出，整个生产流程包括零配件检测、装配、成品检测以及退货处理环节。具体而言：

- **零配件检测**：选择检测，则将检测出的不合格零配件丢弃，保留合格的零配件进入装配环节。如果不检测，所有零配件无论是否合格，都会直接进入装配环节。
- **装配过程检测**：若不检测，组装完成的产品将直接投放市场；若执行检测，则只有通过检测、质量达标的产品才进入市场，不合格品选择是否拆解。
- **不合格产品拆解**：若不进行拆解，不合格产品将直接被丢弃；若进行拆解，则拆解得到的零部件将加入下一批待检测零部件。
- **购买商品替换**：企业承担顾客退换产生的相应调换损失（例如物流费用、品牌信誉损失等），对于退回的不合格商品并入检验不合格进行处理。

5.2.2 决策变量设定

引入 0-1 变量 X_1, X_2, X_f, X_d:

$$X_i = \begin{cases} 1, & \text{检测零配件} \\ 0, & \text{不检测零配件} \end{cases} \qquad i = 1, 2$$

$$X_f = \begin{cases} 1, & \text{检测成品} \\ 0, & \text{不检测成品} \end{cases}$$

$$X_d = \begin{cases} 1, & \text{拆解不合格成品} \\ 0, & \text{不拆解不合格成品} \end{cases}$$

5.2.3 决策方程的建立

在第二问中，我们要为企业生产过程中的各个环节（包括零配件是否检测、成品是否检测、不合格品是否拆解等）作出决策，目标是企业每一批零件的期望利润最大化。为此，方程的建立需要从以下几个步骤展开：

1. 成品次品率的计算

成品的次品率 P_f 受两种零配件的次品率及是否检测的影响。因此，成品次品率 P_f 可以表示为：

$$P_f = 1 - (1 - P_1 \cdot (1 - X_1)) \cdot (1 - P_2 \cdot (1 - X_2)) \cdot (1 - P_m) \tag{7}$$

（7）中，P_f 表示成品的次品率,$P_{1,2}$ 表示零配件 1 或 2 的次品率，P_m 表示正品装配后的次品率。将所有步骤合格的概率相乘，最后用得出整体次品率。

2. 成品装配数量的计算

对于一批零配件 S，经过检测或未检测的零配件会进入装配环节，装配为成品的数量 K 取决于的数量较少的零配件，剩余的将被回收至下一批次。因此，装配数量 K 可以表示为：

$$K = \min\left\{S \cdot (1 - X_1 \cdot P_1), S \cdot (1 - X_2 \cdot P_2)\right\} \tag{8}$$

3. 期望利润的计算

期望利润 E 由收入减去总成本得到：

$$E = \text{Revenue} - \text{Cost} \tag{9}$$

– 收入由最终合格成品的数量和售价决定：

$$\text{Revenue} = K \cdot (1 - P_f) \cdot S_m \tag{10}$$

公式（10）中，K 为装配为成品的数量，S_m 为每件合格成品的市场售价。企业的总收入是由合格的成品数量乘以其售价得到的。

– 在模型中，成本被分为五个部分，分别是零配件的购买成本 (W_0)、检测成本 (W_1)、装配成本 (W_2)、拆解成本 (W_3) 和调换损失 (W_4)。

$$\begin{cases} W_0 = C_1 \cdot S \cdot P_1 + C_2 \cdot S \cdot P_2 + (C_1 + C_2) \cdot (K - X_d \cdot K \cdot P_f) \\ W_1 = S \cdot C_{t1} \cdot X_1 + S \cdot C_{t2} \cdot X_2 + C_t \cdot X_f \\ W_2 = K \cdot C_a \\ W_3 = K \cdot P_f \cdot X_d \cdot C_d \\ W_4 = (1 - X_f) \cdot P_f \cdot L_r \end{cases} \tag{11}$$

W_0 表示采购零配件的总成本。企业购买零配件 1 和零配件 2，并拆解了的不合格成品（即 $X_d = 1$）回收零配件至下一批次，不计入购买成本。W_1 表示对零

配件和成品进行检测所产生的成本，是否进行检测由决策变量 X_1、X_2、X_f 决定。W_2 为装配零配件为成品的总成本。W_3 为拆解不合格成品的成本，拆解的成品数量由成品次品率 P_f 和拆解决策变量 X_d 决定。W_4 为因未检测成品而导致的调换损失。

5.2.4 约束条件

$$\begin{cases} S, K > 0\text{且为整数} \\ P_f > 0 \\ X_i, X_f, X_d = 0, 1 \\ X_d \leq X_f \end{cases}$$

5.3 模型的求解及结果展示

代入题目中提供的数据，通过 Python 遍历目标函数进行求解，得到结果如图 6 所示。（表格中列的表头 1，2，3... 分别对应题目中的情况 1，2，3...）

检测零配件	检测成品	是否拆解	1	2	3	4	5	6
1, 2	是	是	1179	436	1179	**816**	468	1413.5
1, 2	是	否	1026	164	1026	544	332	1499
1, 2	否	是	1395	580	1179	496	548	1651
1, 2	否	否	1242	308	1026	224	412	1736.5
1	是	是	1343.1	596.8	1343.1	776.8	517.2	1469.58
1	是	否	1052.4	107.2	1052.4	287.2	88.8	1636.3
1	否	是	1510.5	**664**	1100.1	72.8	445.2	1661.95
1	否	否	1219.8	174.4	809.4	-416.8	16.8	1828.68
2	是	是	1103.1	216.8	1103.1	496.8	1027.2	1299.58
2	是	否	812.4	-272.8	812.4	7.2	768.8	1466.3
2	否	是	1270.5	284	860.1	-207.2	1035.2	1491.95
2	否	否	979.8	-205.6	569.4	-696.8	776.8	1658.68
不检测	是	是	1443.1	596.8	**1443.1**	696.8	**1227.2**	1444.57
不检测	是	否	982.4	-232.8	982.4	-132.8	628.8	1701.3
不检测	否	是	**1580.5**	604	930.1	-567.2	1075.2	1601.95
不检测	否	否	1119.8	-225.6	469.4	1396.8	476.8	**1858.67**

图 **6**　不同检测策略下的各项数值

从图 6 可以看出最佳的决策方案, 各情况相应的生产策略选择：

情况一：最高值为 1580.5，对应的策略是“不检测零配件 1 和 2，不检测成品，成品拆解”；情况二：最高值为 664，对应的策略是“仅检测零配件 1，成品不检测，成品拆解”; 情况三：最高值为 1443.1，对应的策略是“所有配件均不检测，成品检测，成品拆解”; 情况四：最高值为 816，对应的策略是“检测零配件 1 和 2，检测成品，

成品拆解”; 情况五：最高值为 1227.2，对应的策略是“所有配件均不检测，成品检测，成品拆解”; 情况六：最高值为 1858.67，对应的策略是“所有配件均不检测，成品不检测，成品不拆解”。

六、 问题三模型的建立与求解

6.1 m 道工序 n 个零配件的决策方案

在问题三中，针对 m 道工序和 n 个零配件，已知零配件、半成品以及成品的次品率的生产过程决策方案，我们构建了一个多阶段决策模型，以获得最大利润。

$$\begin{cases} E = \text{Revenue} - \text{Cost} \\ \text{Revenue} = \text{成品中所有正品的销售收入} \end{cases} \tag{12}$$

$$\text{Cost} = W_0 + W_1 + W_2 + W_3 + W_4 \tag{13}$$

$$\begin{cases} W_0 = \sum_{i=1}^{n} \text{零件 i 检测不合格的购买成本} + \sum_{j=1}^{m-1}\sum_{b=1}^{t} \text{半成品 b 检测不合格不拆除的购买成本+ 成本除拆除外的购买成本} \\ W_1 = \sum_{i=1}^{n} \text{零件 i 检测成本} + \sum_{j=1}^{m-1}\sum_{b=1}^{t} \text{半成品检测费 + 成品检测费} \\ W_2 = \sum_{j=1}^{m-1}\sum_{b=1}^{t} \text{半成品 b 的装配成本} \\ W_3 = \sum_{j=1}^{m-1}\sum_{b=1}^{t} \text{半成品 b 的拆解成本 + 成品拆解成本} \\ W_4 = \text{不检测成品中所有次品的调换损失} \end{cases} \tag{14}$$

公式 (14) 中 t 表示第 j 项工序下半成品的数量。该模型旨在帮助企业在生产过程中做出决策，平衡零配件、成品检测不合格品拆解与调换损失的成本，最终达到利润最大化。

6.2 2 道工序、8 个零配件

问题三要求对于 2 道工序、8 个零配件的情况，为企业生产过程中各个阶段的质量控制进行建模，特别是在不同阶段的检测、拆解等环节进行优化决策。基于零配件、半成品以及成品的次品率，达到企业的期望利润最大化目标。

6.2.1 决策变量设定

引入 0-1 变量 X_i, Y_j, Z, α:

$$X_i = \begin{cases} 1, & \text{检测零配件} \\ 0, & \text{不检测零配件} \end{cases} \qquad i = 1, 2, 3, \ldots, 8$$

$$Y_j = \begin{cases} 1, & \text{检测半成品} \\ 0, & \text{不检测半成品} \end{cases} \quad i = 1, 2, 3$$

$$Z = \begin{cases} 1, & \text{检测成品} \\ 0, & \text{不检测成品} \end{cases}$$

$$\alpha_j = \begin{cases} 1, & \text{拆解半成品} \\ 0, & \text{不拆解半成品} \end{cases}$$

$$\alpha = \begin{cases} 1, & \text{拆解成品} \\ 0, & \text{不拆解成品} \end{cases}$$

6.2.2 2 道工序、8 个零配件的模型建立

1. 期望利润公式

$$E = \text{Revenue} - \text{Cost} \tag{15}$$

$$\text{Revenue} = K \cdot (1 - P_f) \cdot S_m \tag{16}$$

其中，收入 Revenue 来自销售合格成品，成本 Cost 则由多项构成，包括零件的购买成本、检测成本、装配成本、拆解成本和调换损失。

2. 成本构成

企业的成本仍由五部分组成：零件的购买成本 、检测成本、装配成本、拆解成本以及调换损失。

– 零件购买成本

$$\begin{cases} W_0 = G_1 + G_2 + G_3 + G_4 + G_5 \\ G_1 = \sum_{i=1}^{8} SP_i C_i X_i \\ G_2 = \left(\sum_{i=1}^{8} C_i \right) (K_f - K_f P_f \alpha) \\ G_3 = \left(\sum_{i=1}^{3} C_i \right) K_1 P_{b1} Y_1 (1 - \alpha_1) \\ G_4 = \left(\sum_{i=4}^{6} C_i \right) K_2 P_{b2} Y_2 (1 - \alpha_2) \\ G_5 = \left(\sum_{i=7}^{8} C_i \right) K_3 P_{b3} Y_3 (1 - \alpha_3) \end{cases} \tag{17}$$

零件的购买成本 W_0 由三部分组成：第一部分 G_1 是检测不合格的零件的购买成本。这里，P_i 是第 i 个零件的次品率，C_i 是其单价，X_i 表示是否检测，S 是每种零件的数量。第二部分 G_2 表示成品除拆解外产品的购买成本。第三部分 G_3-G_5 是半成品不合格不拆解的购买成本，其中 K_j 是半成品 j 的数量，P_bj 是其次品率，Y_j 是检测与否的决策，α_j 表示是否拆解。

– 其他成本

$$\begin{cases} W_1 = \sum_{i=1}^{8} S \cdot C_{ti} \cdot X_i + \sum_{j=1}^{3} (K_j + C_{bj} + Y_j) + K_f \cdot C_f \cdot Z \\ W_2 = \sum_{j=1}^{3} K_j \cdot C_{a_j} + K_f \cdot C_{a_f} \\ W_3 = K_f \cdot P_f \cdot C_d \cdot \alpha + \sum_{j=1}^{3} C_{d_j} \cdot K_j \cdot P_{b_j} \cdot \alpha_j \\ W_4 = (1 - Z) \cdot K_f \cdot P_f \cdot L_r \end{cases} \tag{18}$$

总检测成本是零件的检测成本、半成品检测的成本和成品检测的成本的加总。C_{ti} 是第 i 个零件的检测费用,X_i 表示是否检测,K_j 是半成品 j 的数量,C_bj 是其检测费用,Y_j 表示是否检测,C_f 是成品的检测成本,Z 表示是否检测成品; 装配成本包括半成品和成品的装配费用，C_{a_j} 是半成品 j 的装配费用，C_{a_f} 是成品的装配费用；C_d 是成品拆解费用，C_{d_j} 是半成品 j 的拆解费用，P_{b_j} 是半成品 j 的次品率；L_r 是每件不合格产品的调换损失。

3. 不同阶段的数量和次品率计算

– 零件、半成品和成品的数量计算

对于半成品 j，根据各零配件的检测结果和次品率，我们可以计算装配进入半成品 j 的零件数量：

$$\begin{cases} K_1 = \min\{S \cdot (1 - X_1 \cdot P_1), S \cdot (1 - X_2 \cdot P_2), S \cdot (1 - X_3 \cdot P_3)\} \\ K_2 = \min\{S \cdot (1 - X_4 \cdot P_4), S \cdot (1 - X_5 \cdot P_5), S \cdot (1 - X_6 \cdot P_6)\} \\ K_3 = \min\{S \cdot (1 - X_7 \cdot P_7), S \cdot (1 - X_8 \cdot P_8)\} \end{cases} \tag{19}$$

最终装配成成品的数量 K_f 与各个半成品的是否检测及其次品率有关。

$$K_f = \min\{S \cdot (1 - Y_1 \cdot P_{b_1}), S \cdot (1 - Y_2 \cdot P_{b_2}), S \cdot (1 - Y_3 \cdot P_{b_3})\} \tag{20}$$

其中，S 是每种零件的数量，P_{b_j} 是半成品 j 的次品率。

– 半成品和成品的次品率计算

半成品 P_b 的次品率是受所装配的各个零配件的次品率以及是否检测的影响。因此，各项次品率可以表示为：

$$\begin{cases} P_{b_1} = 1 - [1 - P_1 \cdot (1 - X_1)] \cdot [1 - P_2 \cdot (1 - X_2)] \cdot [1 - P_3 \cdot (1 - X_3)] \cdot (1 - P_{m_1}) \\ P_{b_2} = 1 - [(1 - P_4 \cdot (1 - X_4)) \cdot (1 - P_5 \cdot (1 - X_5)) \cdot (1 - P_6 \cdot (1 - X_6))] \cdot (1 - P_{m_2}) \\ P_{b_3} = 1 - [(1 - P_7 \cdot (1 - X_7)) \cdot (1 - P_8 \cdot (1 - X_8))] \cdot (1 - P_m) \\ P_f = 1 - [(1 - P_{b_1} \cdot (1 - Y_1)) \cdot (1 - P_{b_2} \cdot (1 - Y_2)) \cdot (1 - P_{b_3} \cdot (1 - Y_3))] \cdot (1 - P_m) \end{cases} \tag{21}$$

其中，P_i 分别为零配件 i 的次品率，P_{m_j} 是正品装配后半成品 j 的次品率 P_m 是装配工艺的次品率，P_f 是最终成品的次品率，P_{b_j} 分别为半成品 j 的次品率，P_m 是正品装配后成品的次品率。

6.3 约束条件

$$\begin{cases} S, K_j, K_f > 0 \text{且为整数} \\ P_f > 0, P_{bj} > 0 \\ X_i, Y_j, Z, \alpha_j, \alpha = 0, 1 \\ \alpha_j \leq Y_j \\ \alpha \leq Z \end{cases}$$

6.4 模型的求解

对于这一存在大量的决策变量（零件、半成品和成品的检测与拆解），并且优化的目标函数具有复杂的非线性关系的问题，采用模拟退火算法求解。模拟退火算法的原理如流程图所示：

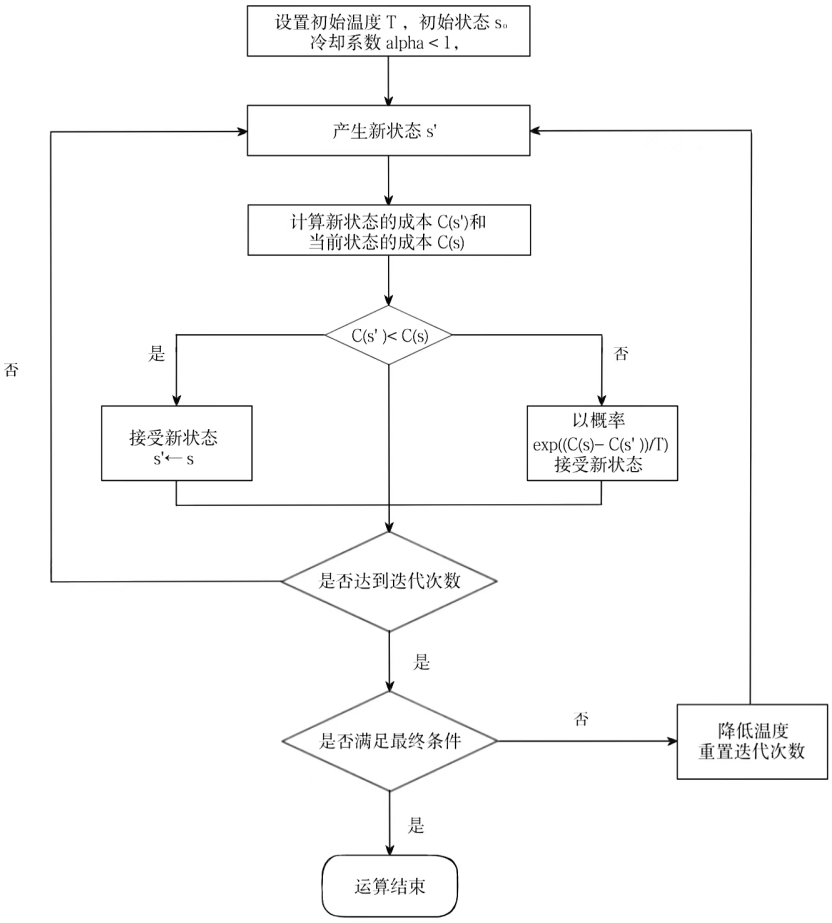

图 **7**　模拟退火算法原理

在解决这一问题时，我们将状态空间定义为一组决策，例如是否检测零配件、是否

检测成品、是否拆解不合格成品等，能量函数定义为总成本，包括购买成本、检测成本、装配成本、市场售价、调换损失和拆解费用。在每次迭代中，随机选择一个决策点并改变其决策（例如，从不检测改为检测），接着计算新解的能量（总成本）。如果新解的能量低于当前解，或者根据 Metropolis 准则接受新解，则更新当前解。由于模拟退火算法的全局最优解为能量值最低的决策[6]，我们在实际计算中对目标函数进行取负值处理。

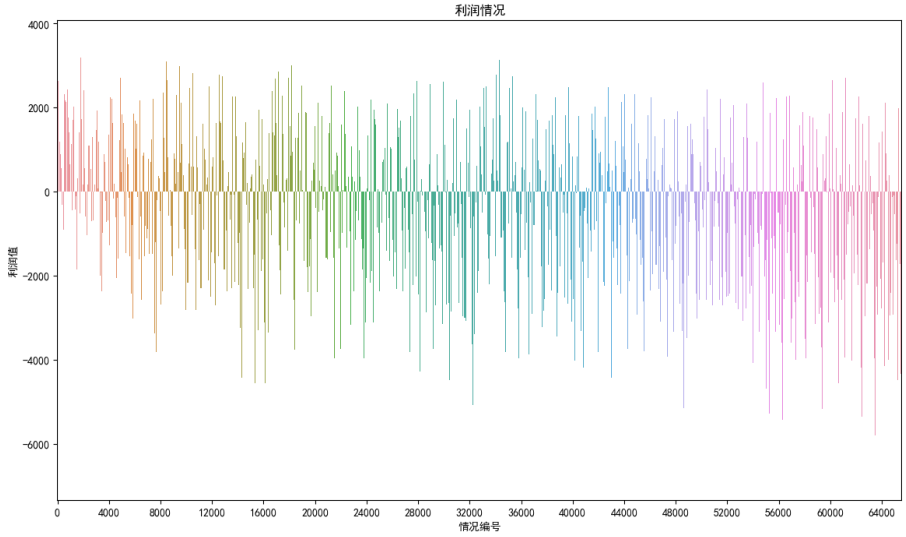

图 **8**　问题三结果可视化

图 8 为不同决策方案下 6 万多种情况的利润值对比图。横坐标为根据决策方案排列组合出的情况编号，纵坐标为利润值。其中，我们求解出来的最优决策为 8 个零配件全部检测，3 个半成品全部检测并拆解，成品检测并拆解，我们假设一批零件中每种零件的个数为 100 个的情况下，利润最大为 3559.40 元。

七、 问题四模型的建立与求解

在第四问中，我们采用蒙特卡洛模拟方法来估计次品率及其置信区间，并通过并行计算提高模拟的效率。

7.1 蒙特卡洛模拟

7.1.1 理论支撑

1. 大数定律: 随着样本数量 N 的增加，样本均值收敛到总体均值。对任意整数 ε:

$$\lim_{N\to\infty} P(|\bar{X} - \mu| > \varepsilon) = 0 \tag{22}$$

2. 中心极限定理：即使 X 不是正态分布，样本均值的分布随着 N 的增加趋近于正

态分布。

$$\sqrt{N}(\bar{X}-\mu) \xrightarrow{d} N(0,\sigma^2), \text{其中 } \sigma^2 \text{ 是 } X \text{ 的方差} \tag{23}$$

通过上述理论支持，我们采用蒙特卡洛模拟方法，生成大量的样本次品率，估计总体次品率的均值及置信区间。

7.1.2 次品率估计

期望的估计：随机变量 X 的期望 $E[X]$ 可以通过样本的平均值来估计：

$$E[X] \approx \frac{1}{N}\sum_{i=1}^{N} X_i \tag{24}$$

次品率置信区间：SE 为标准误差

$$\text{Lower Bound=p-z*SE} \tag{25}$$

$$\text{Upper Bound=p+z*SE} \tag{26}$$

7.1.3 并行化

蒙特卡洛模拟可以通过并行计算来加速，因为每个样本的生成和处理是独立的。这使得拟可以在多核处理器或分布式计算环境中高效执行。

7.2 取值计算结果

假设每个样本中次品的数量遵循二项分布，即 X Binomial(n,p)，其中 n 是样本大小，p 是次品率。对于每个样本，我们计算次品的比例，即。取每次模拟样本大小为 1000，模拟次数为 10000 次取 α=0.05，在 95% 的置信水平下计算置信区间的上下界。得到 p=0.0999，置信区间 =[0.0997，0.1001]。

情况		零配件 1			零配件 2			成品				不合格成品	
		次品率	购买单价	检测成本	次品率	购买单价	检测成本	次品率	装配成本	检测成本	市场售价	调换损失	拆解费用
A	1	9.99%	4	2	9.99%	18	3	9.99%	6	3	56	6	5
	2	9.99%	4	2	9.99%	18	3	9.99%	6	3	56	6	5
B	3	9.99%	4	2	9.99%	18	3	9.99%	6	3	56	30	5
C	4	9.99%	4	1	9.99%	18	1	9.99%	6	2	56	30	5
D	5	9.99%	4	8	9.99%	18	1	9.99%	6	2	56	10	5
E	6	9.99%	4	2	9.99%	18	3	9.99%	6	3	56	10	40

图 **9**　第四问对第 **2** 问问题重述

检测零配件	检测成品	拆解不合格成品	A	B	C	D	E
1, 2	是	是	1179. 78	1179. 78	**1569. 79**	869. 79	865. 06
1, 2	是	否	1026. 92	1026. 92	1416. 93	716. 93	1026. 92
1, 2	否	否	1243	1027. 19	1327. 19	807. 03	**1207. 03**
1	是	是	1343. 95	1343. 95	1533. 96	833. 96	745. 95
1	是	否	1053. 49	1053. 49	1243. 5	543. 5	1053. 49
1	否	否	1221. 01	810. 95	910. 95	552. 66	1152. 66
2	是	是	1104. 09	1104. 09	1394. 1	1394. 1	506. 09
2	是	否	813. 63	813. 63	1103. 64	1103. 64	813. 63
2	否	否	981. 15	571. 09	771. 09	1112. 8	912. 8
不检测	是	是	**1444. 05**	**1444. 05**	1544. 05	**1544. 05**	496. 4
不检测	是	否	983. 76	983. 76	1083. 76	1083. 76	983. 76
不检测	否	否	1121. 31	471. 49	471. 49	1013	1013

图 **10**　第四问对第 **2** 问的重解

图 9 中展示了对第二问问题的重述，更改零配件 1、2 和成品的次品率为蒙特卡洛模拟的估计值后，原情况 1 与 2 相同，合并为情况 A，后续依次对应情况 B、C、D 和 E。图 10 为对问题 2 重解的结果，情况 A、B、D 的最优决策均为不检测零配件 1、2，检测并拆解不合格成品；情况 C 的最优决策为检测零配件 1、2，检测并拆解不合格成品；情况 E 的最优决策为检测零配件 1、2，不检测成品。

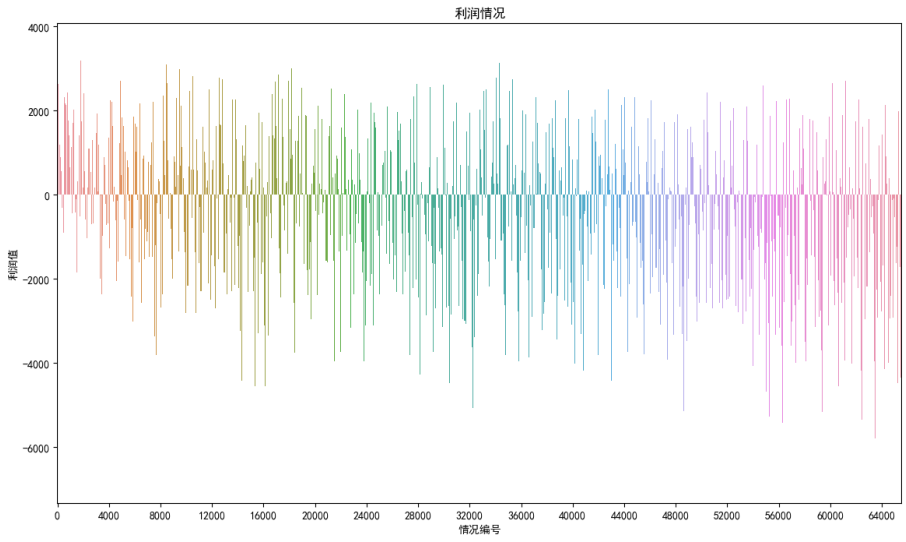

图 **11**　问题四对问题三的重解

图 11 为将所有零配件、半成品、成品的次品率改为蒙特卡洛模拟出的预估值下的不同决策方案下 6 万多种情况的利润值对比图。横坐标为根据决策方案排列组合出的情况编号，纵坐标为利润值。其中，我们求解出来的最优决策为 8 个零配件全部检测，3 个半成品全部检测并拆解，成品检测并拆解，我们假设一批零件中每种零件的个数为 100 个的情况下，利润最大为 3560.94 元。

八、 模型的评价

8.1 模型的亮点

1. **动态调整与灵活性：**论文提出的基于截尾序贯最优检验的抽样方案，能够动态调整抽样过程，根据实时数据调整检测策略，有效减少检测成本并提高决策效率。这种灵活性使得模型能够适应不同生产环境和次品率的变化。

2. **多阶段决策优化：**对于多工序、多零配件的生产流程，模型构建了多阶段决策模型，综合考虑了零配件、半成品和成品的次品率，制定了最优的检测和拆解策略。这种全面的优化方法有助于最大化企业的利润。

3. **算法多样性：**论文采用了动态规划、模拟退火算法等先进算法来求解复杂的非线性优化问题。此外，通过蒙特卡洛模拟方法估计次品率，并结合并行计算提高模拟效率，使得模型能够更准确地反映实际生产情况，并快速给出优化方案。

8.2 模型可改进之处

1. **模型假设的局限性：**模型在构建过程中基于一些简化的假设，如次品率的分布、检测成本的固定性等，可能与实际生产情况存在偏差，影响模型的准确性和适用性。

2. **人为因素：**模型忽略了人为因素对生产过程和质量控制的影响，如操作人员的技能水平、工作态度等。这些因素在实际生产中可能对次品率和检测效率产生重要影响。

3. **风险与不确定性：**尽管模型考虑了次品率和检测成本的不确定性，但其他潜在的风险和不确定性因素（如供应链中断、市场需求变化等）可能未被充分考虑。

九、 模型的推广

1. 供应链管理：模型有助于改进供应链中的质量管理，通过抽样检测减少供应链中的次品率，确保供应链的高效和可靠性。

2. 库存管理：在库存管理中，模型可以帮助企业决定何时进行库存检查，以及如何平衡库存成本和缺货风险。确定最优库存水平，减少库存成本，同时避免缺货或过剩。

3. 生产决策优化：在生产过程中，模型可以帮助企业在成本和质量之间做出最佳决策，如是否进行额外的质量检测，或者如何处理不合格产品。

4. 生产计划和调度：优化生产流程，合理安排生产计划，减少机器空闲时间和提高生产效率。

5. 服务运营优化：在服务行业，如餐饮、酒店和物流，优化服务流程，提高服务质量和效率。

这些推广方向都涉及到决策制定、成本效益分析和风险管理，模型的核心思想是利用统计和概率方法来优化决策过程，减少成本和提高效率。

参考文献

[1] Wald A. Sequential tests of statistical hypotheses. Ann Math Statist, 1945, 16: 117–186

[2] Wald A, Wolfowitz J. Optimum character of the sequential probability ratio test. Ann Math Statist, 1947, 19: 326–339

[3] Gao, G.L., Zhong, Y.M., Gao, S.S., Gao, B.B. Double-Channel Sequential Probability Ratio Test for Failure Detection in Multisensor Integrated Systems. Ieee Transactions on Instrumentation and Measurement,2021: 70.

[4] Kira, S., Yang, T.M., Shadlen, M.N. A Neural Implementation of Wald's Sequential Probability Ratio Test. Neuron, 2015,85(4), 861-873.

[5] Han, R., Du, L.P., Chen, Y.Y. Performance Analysis of Sequential Detection of Primary User Number Based on Multihypothesis Sequential Probability Ratio Test. Ieee Communications Letters,2018, 22(5), 1034-1037.

[6] Drexl, A., Nikulin, Y. Multicriteria airport gate assignment and Pareto simulated annealing. Iie Transactions, 2008,40(4), 385-397.

附录 A　代码

```
import numpy as np
from scipy.stats import binom

# 参数设置
p0 = 0.1 # 标称次品率
n0 = 50 # 最大样本量
alpha = 0.05 # 第一类错误概率
beta = 0.1 # 第二类错误概率

# 计算累积分布函数值
def binomcdf(k, n, p):
return binom.cdf(k, n, p)

# 计算SPRT边界
A = (1 - beta) / alpha
B = alpha / (1 - beta)

# 动态规划求解
def dynamic_programming(n0, p0, A, B):
# 初始化动态规划矩阵
V = np.full((n0 + 1, n0 + 1), np.inf) # 使用无穷大初始化
decision = np.zeros((n0 + 1, n0 + 1), dtype=int)

# 设置终止状态
for k in range(n0 + 1):
if binomcdf(k, n0, p0) <= beta:
decision[n0, k] = 1 # 拒绝H0
elif binomcdf(k, n0, p0) >= 1 - alpha:
decision[n0, k] = 0 # 接受H0

# 反向求解
for i in range(n0 - 1, -1, -1):
for k in range(i + 1):
for j in range(max(0, k - 5), min(i + 2, k + 6)):
next_k = k + j
if next_k > n0:
continue
cdf = binomcdf(next_k, i + 1, p0)
if cdf <= beta:
expected_samples = 1 + V[i + 1, next_k]
else:
expected_samples = np.inf
V[i, k] = min(V[i, k], expected_samples)
decision[i, k] = 1 if binomcdf(k, i, p0) <= beta else 0
```

```
return V, decision

# 计算最优策略
V, decision = dynamic_programming(n0, p0, A, B)

# 打印结果
print("Optimal expected number of samples matrix (V):")
print(V)
print("Decision matrix:")
print(decision)

# 根据信度95%和90%计算最优抽样次数
def calculate_samples(n0, decision, confidence_level):
p_confidence = 1 - confidence_level
for i in range(n0 + 1):
for k in range(i + 1):
if binomcdf(k, i, p0) <= p_confidence:
return i # 返回满足信度要求的最小抽样次数
return n0 # 如果没有满足条件的抽样次数，则返回最大抽样次数

samples_95 = calculate_samples(n0, decision, 0.95)
samples_90 = calculate_samples(n0, decision, 0.90)

print(f"Minimum number of samples for 95% confidence: {samples_95}")
print(f"Minimum number of samples for 90% confidence: {samples_90}")
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import matplotlib

matplotlib.rcParams['font.sans-serif'] = ['SimHei'] # 'SimHei'是黑体的字体名
matplotlib.rcParams['font.family']='sans-serif'
matplotlib.rcParams['axes.unicode_minus'] = False

np.random.seed(0)

# 假设的参数
p0 = 0.1 # 原假设下的成功概率
p1 = 0.15 # 备择假设下的成功概率
alpha = 0.05 # 第一类错误的概率
beta = 0.1 # 第二类错误的概率
sample_size_limit = 50 # 截尾值

# 计算截尾序贯检验的边界
Z_alpha = norm.ppf(1-alpha)
Z_beta = norm.ppf(beta)
```

```
# 似然比的计算
def likelihood_ratio(n, k, p0, p1):
if k == 0 or k == n:
return np.inf # 避免除以零
return (k / n) * np.log(p1 / p0) + ((1 - k / n) * np.log((1 - p1) / (1 - p0)))

# 计算ASN
def asn(p, n):
return n * (p / (1 - p))

# 绘制截尾序贯检验方案图
plt.figure(figsize=(12, 12))

plt.subplot(221)
n_samples = np.arange(0, sample_size_limit + 1, 1)
k_samples = np.arange(0, n_samples.max() + 1, 1)
lr = np.array([[likelihood_ratio(n, k, p0, p1) for k in k_samples] for n in n_samples])
accept_region = lr < Z_alpha
reject_region = lr > Z_beta

plt.imshow(accept_region, extent=(0, sample_size_limit, 0, sample_size_limit), aspect='auto',
    cmap='Greens', alpha=0.5)
plt.imshow(reject_region, extent=(0, sample_size_limit, 0, sample_size_limit), aspect='auto',
    cmap='Reds', alpha=0.5)
plt.plot(Z_alpha, sample_size_limit, 'ko', label='接受边界')
plt.plot(Z_beta, sample_size_limit, 'ko', label='拒绝边界')
plt.plot([sample_size_limit, sample_size_limit], [0, sample_size_limit], 'k--',
    label='截尾边界')
plt.xlabel('样本数量')
plt.ylabel('成功次数')
plt.title('最优截尾序贯检验')
plt.legend()
plt.grid(True)

# 绘制ASN图
plt.subplot(222)
asn_values_p0 = [asn(p0, i) for i in range(1, sample_size_limit + 1)]
asn_values_p1 = [asn(p1, i) for i in range(1, sample_size_limit + 1)]
plt.plot(range(1, sample_size_limit + 1), asn_values_p0, label='H0: ASN在p0下')
plt.plot(range(1, sample_size_limit + 1), asn_values_p1, label='H1: ASN在p1下')
plt.xlabel('样本数量')
plt.ylabel('平均样本数 (ASN)')
plt.title('平均样本数 (ASN)')
plt.legend()
plt.grid(True)
```

```
# 绘制OC曲线
plt.subplot(223)
beta_values = np.linspace(beta, alpha, 100)
alpha_values = np.linspace(alpha, beta, 100)
plt.plot(beta_values, alpha_values, label='OC曲线')
plt.xlabel('接受H0的概率')
plt.ylabel('接受H1的概率')
plt.title('操作特征 (OC) 曲线')
plt.legend()
plt.grid(True)

# 绘制预期成本图
plt.subplot(224)
costs = [i + np.random.rand() for i in range(1, sample_size_limit + 1)] # 假设成本数据
plt.plot(range(1, sample_size_limit + 1), costs, label='预期成本')
plt.xlabel('样本大小')
plt.ylabel('预期成本')
plt.title('预期成本')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

def calculate_expected_value(m, p1, p2, x1, x2, pm, sm, c1, c2, ct1, ct2, ca, ct, xf,xd, cd,
    lr):
# 计算装配为成品的数量 K
K = min(m * (1 - x1 * p1), m * (1 - x2 * p2))

# 计算成品次品率
pf = 1 - (1 - p1 * (1 - x1)) * (1 - p2 * (1 - x2)) * (1 - pm)

# 计算收入
Revenue = K * (1 - pf) * sm

# 计算零件的购买成本 W0
W0 = c1 * m * p1 * x1 + c2 * m * p2 * x2 +(c1+c2)*(K-xd*K*pf)

# 计算检测成本 W1
W1 = m * ct1 * x1 + m * ct2 * x2 + K * ct * xf

# 计算装配成本 W2
W2 = K * ca

# 计算拆解成本 W3
W3 = K * pf * xd * cd
```

```
# 计算调换损失 W4
W4 = (1 - xf) * K * pf * lr

# 计算总成本
Cost = W0 + W1 + W2 + W3 + W4

# 计算期望利润
Expected_Profit = Revenue - Cost

return Expected_Profit

# 给定的参数
m = 100 # 一批零件的数量
p1 = 0.1 # 零配件1的次品率
c1 = 4  # 零配件1的购买单价
ct1 = 2 # 零配件1的检测成本
p2 = 0.1 # 零配件2的次品率
c2 = 18 # 零配件2的购买单价
ct2 = 3 # 零配件2的检测成本
pm = 0.1 # 将正品零配件装配后的产品次品率
ca = 6  # 装配成本
ct = 3  # 成品的检测成本
sm = 56 # 成品的市场售价
lr = 6  # 调换损失
cd = 5  # 拆解成本

# 遍历所有可能的决策组合
scenarios = [
# (检测零配件1, 检测零配件2, 检测成品, 拆解不合格成品)
(True, True, True, True),
(True, True, True, False),
(True, True, False, False),
(True, False, True, True),
(True, False, True, False),
(True, False, False, False),
(False, True, True, True),
(False, True, True, False),
(False, True, False, False),
(False, False, True, True),
(False, False, True, False),
(False, False, False, False),
]

# 输出每种方案的期望值
for i, (test_part1, test_part2, test_final, disassemble_returned) in enumerate(scenarios):
expected_value = calculate_expected_value(
m, p1, p2, test_part1, test_part2, pm, sm, c1, c2, ct1, ct2, ca, ct, test_final,
```

```
    disassemble_returned, cd, lr
)
print(f"Scenario {i+1}: Expected Value = {expected_value:.2f}")
import random
import math
import matplotlib.pyplot as plt
from itertools import product

costs = {
   'Ci': [2, 8, 12, 2, 8, 12, 8, 12], # 每个零件的购买单价
   'Cti': [1, 1, 2, 1, 1, 2, 1, 2], # 每个零件的检测成本
   'Cbj': [4, 4, 4], # 每个半成品的检测成本
   'Cf': 6, # 成品检测成本
   'Caf': 8, # 成品装配成本
   'Cd': 10, # 成品拆解成本
   'Cdj': [6, 6, 6], # 半成品拆解成本
   'Lr': 40, # 不合格成品的调换损失
   'Caj':[8, 8, 8] #半成品的装配成本
}

defective_rates = {
   'Pi': [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1], # 每个零件的次品率
   'Pmj': [0.1, 0.1, 0.1], # 每个半成品的次品率
   'Pm': 0.1 # 成品次品率
}

# 生成所有可能的状态组合
all_states = [
{
   'x': {i: xi[i] for i in range(8)},
   'y': {i: yj[i] for i in range(3)},
   'a': {i: ak[i] if yj[i] else False for i in range(3)},
   'z': z,
   'a_final': a if z else False
}
for xi in product([True, False], repeat=8)
for yj in product([True, False], repeat=3)
for ak in product([True, False], repeat=3)
for z in [True, False]
for a in [True, False]
]

# 定义成本函数
def cost_function(state, costs, defective_rates, S, Sm):
"""
:param state: 每个组件是否检测、每个半成品是否检测、成品是否检测
:param costs: 检测成本、装配成本、拆解费用和调换损失
```

```
:param defective_rates: 缺陷率
:param S: 一批零件的数量
:param Sm: 成品的市场售价
:return: 成本减收入
"""

# 计算各阶段的次品率
Pb1 = 1 - (1 - defective_rates['Pi'][0] * (1 - state['x'][0])) * \
(1 - defective_rates['Pi'][1] * (1 - state['x'][1])) * \
(1 - defective_rates['Pi'][2] * (1 - state['x'][2])) * (1 - defective_rates['Pmj'][0])
Pb2 = 1 - (1 - defective_rates['Pi'][3] * (1 - state['x'][3])) * \
(1 - defective_rates['Pi'][4] * (1 - state['x'][4])) * \
(1 - defective_rates['Pi'][5] * (1 - state['x'][5])) * (1 - defective_rates['Pmj'][1])
Pb3 = 1 - (1 - defective_rates['Pi'][6] * (1 - state['x'][6])) * \
(1 - defective_rates['Pi'][7] * (1 - state['x'][7])) * (1 - defective_rates['Pmj'][2])
Pb={'Pbj':[Pb1,Pb2,Pb3]}

Pf = 1 - (1 - Pb1 * (1 - state['y'][0])) * (1 - Pb2 * (1 - state['y'][1])) * (1 - Pb3 * (1 -
    state['y'][2])) * (1 - defective_rates['Pm'])

# 计算有效数量
K1 = min(S * (1 - state['x'][0] * defective_rates['Pi'][0]), S * (1 - state['x'][1] *
    defective_rates['Pi'][1]), S * (1 - state['x'][2] * defective_rates['Pi'][2]))
K2 = min(S * (1 - state['x'][3] * defective_rates['Pi'][3]), S * (1 - state['x'][4] *
    defective_rates['Pi'][4]), S * (1 - state['x'][5] * defective_rates['Pi'][5]))
K3 = min(S * (1 - state['x'][6] * defective_rates['Pi'][6]), S * (1 - state['x'][7] *
    defective_rates['Pi'][7]))
K={'Kj':[K1,K2,K3]}

Kf = min(K1 * (1 - state['y'][0] * Pb1), K2 * (1 - state['y'][1] * Pb2), K3 * (1 -
    state['y'][2] * Pb3))

# 零件的购买成本
W0 = sum([costs['Ci'][i] * S * state['x'][i]* defective_rates['Pi'][i] for i in range(8)])+
    sum([costs['Ci'][i] for i in range(8)])*(1-Pf*state['a_final'])*Kf + \
sum([costs['Ci'][i] for i in range(3)])*K1*Pb1*state['y'][0]*(1-state['a'][0]) +
    sum([costs['Ci'][i] for i in range(3,6)])*K2*Pb2*state['y'][1]*(1-state['a'][1])+ \
sum([costs['Ci'][i] for i in range(6,7)])*K1*Pb1*state['y'][2]*(1-state['a'][2])

W1 = state['z'] * costs['Cf'] * Kf + sum([costs['Cti'][i] * S * state['x'][i] for i in
    range(8)]) + sum([costs['Cbj'][j] * K['Kj'][j] * state['y'][j] for j in range(3)])

# 成品装配成本
W2 = Kf * costs['Caf'] + sum([K['Kj'][j] * costs['Caj'][j] for j in range(3)])

#成品拆解成本
W3 = Kf * Pf * costs['Cd'] * state['a_final'] + sum([costs['Cdj'][j] * K['Kj'][j] *
```

```
    Pb['Pbj'][j] * state['a'][j] for j in range(3)])

W4 = (1-state['z']) * costs['Lr'] * Kf * Pf

# 总成本
total_cost = W0 + W1 + W2 + W3 + W4

# 总收入
total_revenue = Kf * Sm * (1 - Pf)

# 成本减收入
return total_cost - total_revenue

class SA:
def __init__(self, cost_func, states, costs, defective_rates, S, Sm, iter=100, T0=100,
    Tf=0.01, alpha=0.99):
self.cost_func = cost_func
self.states = states
self.iter = iter
self.alpha = alpha
self.T0 = T0
self.Tf = Tf
self.T = T0
self.current_state = random.choice(states)
self.history = {'f': [], 'T': []}
self.costs = costs
self.defective_rates = defective_rates
self.S = S
self.Sm = Sm

def generate_new(self, current_state):
"""生成一个新的邻近状态。"""
new_state = {
   'x': current_state['x'].copy(),
   'y': current_state['y'].copy(),
   'a': current_state['a'].copy(),
   'z': current_state['z'],
   'a_final': current_state['a_final']
}
# 随机选择一个要翻转的键
keys = list(new_state.keys())
key_to_flip = random.choice(keys)
# 如果是列表类型的键，随机选择一个索引来翻转
if isinstance(new_state[key_to_flip], dict):
index_to_flip = random.choice(list(new_state[key_to_flip].keys()))
new_state[key_to_flip][index_to_flip] = not new_state[key_to_flip][index_to_flip]
else:
```

```
new_state[key_to_flip] = not new_state[key_to_flip]
return new_state

def Metropolis(self, f, f_new):
"""Metropolis准则。"""
if f_new <= f:
return True
else:
p = math.exp((f - f_new) / self.T)
return random.random() < p

def best(self):
""" 获取最优目标函数值。"""
f_list = []
for state in self.states:
f = self.cost_func(state, self.costs, self.defective_rates, self.S, self.Sm)
f_list.append(f)
f_best = min(f_list)
idx = f_list.index(f_best)
return f_best, idx

def run(self):
count = 0
# 外循环迭代，当前温度小于终止温度的阈值
while self.T > self.Tf:
# 内循环迭代100次
for _ in range(self.iter):
f = self.cost_func(self.current_state, self.costs, self.defective_rates, self.S, self.Sm)
new_state = self.generate_new(self.current_state)
f_new = self.cost_func(new_state, self.costs, self.defective_rates, self.S, self.Sm)
if self.Metropolis(f, f_new):
self.current_state = new_state

# 迭代L次记录在该温度下最优解
ft, _ = self.best()
self.history['f'].append(ft)
self.history['T'].append(self.T)
# 温度按照一定的比例下降（冷却）
self.T = self.T * self.alpha
count += 1

# 得到最优解
f_best, idx = self.best()
print(f"F={f_best}, State={self.states[idx]}")

S = 100 # 一批零件的数量
Sm = 200 # 成品的市场售价
```

```
sa = SA(cost_function, all_states, costs, defective_rates, S, Sm)
sa.run()

plt.plot(sa.history['T'], sa.history['f'])
plt.title('Simulated Annealing Optimization')
plt.xlabel('Temperature')
plt.ylabel('Cost - Revenue')
plt.gca().invert_xaxis()
plt.show()
import pandas as pd

data = []

# 遍历所有状态
for state in all_states:
# 计算每种状态下的成本减收入
cost_revenue = cost_function(state, costs, defective_rates, S, Sm)
# 将状态和成本减收入作为元组添加到数据列表中
data.append((state, cost_revenue))

# 创建一个 DataFrame
df = pd.DataFrame(data, columns=['State', 'Total_Cost_Revenue'])

# 将状态转换为可读的字符串形式
df['State'] = df['State'].apply(lambda x: str(x))

# 保存到 Excel 文件
excel_path = r'C:\Users\何雨杨\Desktop\数模\cost_revenue.xlsx'
df.to_excel(excel_path, index=False)

print(f"数据已导出到 {excel_path}")
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['font.family'] = 'sans-serif'
matplotlib.rcParams['axes.unicode_minus'] = False

# 加载Excel文件
file_path = r'C:\Users\何雨杨\Desktop\利润.xlsx'
sheet_name = 'Sheet1'

# 使用pandas读取Excel文件
df = pd.read_excel(file_path, sheet_name=sheet_name)
```

```
# 查看数据的前几行以理解其结构
print(df.head())

# 绘制数据
plt.figure(figsize=(14, 8))

# 假设第一列是情况编号，第二列是利润值
sns.barplot(x=df.index, y=df.iloc[:, 1], data=df)

# 设置图表标题和轴标签
plt.title('利润情况')
plt.xlabel('情况编号')
plt.ylabel('利润值')

# 获取x轴的原始刻度位置
original_ticks = df.index

# 计算新的刻度位置，每隔2000个标签显示一次
new_ticks = original_ticks[::4000]

# 生成新的标签列表
new_labels = [str(tick) if tick in new_ticks else '' for tick in original_ticks]

# 将new_ticks转换为Python列表以便索引
new_ticks_list = new_ticks.tolist()

# 设置x轴的刻度位置和标签
plt.xticks(new_ticks_list, [new_labels[tick] for tick in new_ticks_list])

# 显示图表
plt.show()
import numpy as np
from scipy.stats import norm
from joblib import Parallel, delayed
import os

def estimate_confidence_interval(sample_proportions, alpha=0.05):
mean = np.mean(sample_proportions)
std_error = np.std(sample_proportions, ddof=1) / np.sqrt(len(sample_proportions))
z_score = norm.ppf(1 - alpha / 2)
lower_bound = mean - z_score * std_error
upper_bound = mean + z_score * std_error
return lower_bound, upper_bound

def parallel_monte_carlo_sampling(p_hat, n, trials=10000, n_jobs=-1):
# 确保至少有一个任务
```

```
if n_jobs <= 0:
n_jobs = min(os.cpu_count(), trials)

# 分配任务
chunk_size, remainder = divmod(trials, n_jobs)
chunks = [chunk_size + 1 if i < remainder else chunk_size for i in range(n_jobs)]

print(f"Total chunks: {len(chunks)}")

# 使用 joblib 的 Parallel 函数并行执行任务
all_sample_proportions = Parallel(n_jobs=n_jobs, backend="threading")( # 使用线程后端
delayed(generate_sample_proportion)(n, p_hat, chunk) for chunk in chunks
)

# 检查是否有结果
if not all_sample_proportions:
raise ValueError("No results were returned from the parallel tasks.")

print(f"Results count: {len(all_sample_proportions)}")

sample_proportions = np.concatenate(all_sample_proportions)
corrected_defect_rate = np.mean(sample_proportions)

lower_bound, upper_bound = estimate_confidence_interval(sample_proportions)

return corrected_defect_rate, (lower_bound, upper_bound)

def generate_sample_proportion(n, p_hat, chunk):
# 生成二项分布的随机数
defective_counts = np.random.binomial(n, p_hat, chunk)
# 计算样本比例
proportions = defective_counts.astype(float) / n
return proportions

# 假设初始次品率估计为10%，样本大小为1000，模拟10000次
initial_defect_rate = 0.10
sample_size = 1000
simulations = 10000

# 进行模拟
corrected_defect_rate, confidence_interval = parallel_monte_carlo_sampling(
initial_defect_rate, sample_size, simulations
)

print(f"修正后的次品率为: {corrected_defect_rate:.4f}")
print(f"置信区间为: {confidence_interval}")
```