生产过程中的决策问题

摘要

本文针对现有的电子产品生产检测过程，通过**假设检验**理论确定抽样检测方案。针对企业各阶段生产过程，由于数据量较小，生产过程固定，本文通过建立**深度优先搜索模型**遍历所有可能的决策路径，针对每一条决策路径计算总利润的数学期望，以此找到最优的决策方案。

对于问题1，我们首先为简化抽样过程，假设进行简单随机抽样，并且采用统计学中的假设检验方法，在不同信度条件下，我们将使用二项比例检验来决定是否接受这批零部件。基于具体的假设检验理论，可使用**假设检验样本量计算公式**来计算最小样本量，在给定的显著性水平和检验力下做出准确的决策。最终计算可得，最少抽取223个样本，其中次品数大于29个时，拒绝零假设，接受备选假设，即在95%的信度下认定零配件次品率超过标称值10%，拒绝这批零件；抽取162个样本，其中次品数小于12个，拒绝零假设，接受备选假设，即在90%的信度下认定零配件次品率不超过标称值10%，接受这批零件。

对于问题2，已知两种零配件的购买单价和成品次市场售价和次品率以及整个生产过程中可能存在的各种成本，需要为企业生产过程的各个阶段作出决策，为此我们引入多个**0-1变量**来表示决策过程，不同的变量代表不同的决策分支，分别计算各个阶段在不同决策情况下的成本和可能的损耗，建立关于总利润的数学模型，以最大化产品利润为目标函数，遍历所有决策路径。最终结果可见表1。

对于问题3，问题3与问题2相似，但在问题2的基础上增加了多个工序和多个零配件，零配件与成品之间存在产品的中间状态，生产过程产生多个决策分支。本质上，一个半成品的生产与问题2成品的生产并无根本性的不同，所以这是一个**自相似的分形结构**，可以利用**动态规划**的思想将大问题转化为小问题。具体而言，题目中给出2道工序、8个零配件的具体数据，要求给出具体决策方案，为此我们依旧引入0-1变量，在问题2的模型基础上加以改进，以最大化产品利润为目标函数，根据题目中的具体情况给出决策。所有结果可见表2。

对于问题4，由于问题2、3中零配件、半成品和成品的次品

对于问题1，本文将假设检验理论应用于实际生产生活中，通过准确计算帮助企业作出决策，方式简洁，可行度高；对于问题2，3，本文主要通过引入0-1变量来表示各个决策阶段，通过深度优先搜索遍历所有决策路径，过程简洁，方法明了，对帮助企业在生产过程中作出决策有着实际意义，最终本文给出优缺点以及需要改进的地方，便于模型的进一步推广。

关键词：生产决策 假设检验 深度优先搜索 0-1变量 动态规划

# 问题重述

## 问题背景

性能优秀的电子产品能够为人们提供便利、提升工作效率并且提供大量娱乐和学习的机会。某企业需要分别购买两种零配件：零配件1和零配件2用以生产某种畅销的电子产品，零配件的合格与否将会影响成品的合格与否，而对于不合格的成品，企业将进行报废或拆解。

建立数学模型，求解以下问题。

## 问题重述

问题1：某供应商承诺一批零配件的次品率不会超过某个标称值，企业将采用抽样检测的方法决定是否从供应商购买零配件，费用自行承担。在标称值为10％的情况下，请分别针对以下两种情形，为企业设计检测次数尽可能少的抽样检测方案并给出具体结果。

1. 在95％的信度下认定零配件次品率超过标称值，则拒收这批零配件；
2. 在90％的信度下认定零配件次品率不超过标称值，则接受这批零配件。

问题2：已知零配件1、零配件2和成品的次品率，为企业生产该零配件的各个阶段作出决策：

（1）对零配件1和零配件2是否进行检测，如果对某种零配件不检测，则该零配件直接进入装配环节；否则若检测出不合格零配件，则将给零配件丢弃。

（2）对装配好的每一件成品是否进行检测，如果对某件成品不检测，则该成品直接进入市场；否则只有检测合格的成品能够进入到市场；

（3）对检测出来的不合格品是否进行拆解，如果不拆解该不合格品，则直接丢弃；否则对其进行拆解，将拆解后的零配件进行步骤（1）和（2）.

（4）企业无条件对用户购买的不合格品进行调换，且将产生一定的调换损失，如物流成本、企业信誉等。对于退回的不合格品，将重复步骤（3）。

根据所做决策，对题中表1的情形给出决策方案，同时给出决策的依据和相应的指标结果。

问题3：对于m道工序与n个零配件，且已知零配件、半成品和成品的次品率，重复问题2，并同样给出生产过程中的决策方案。题中图1与表2分别给出2道工序、8个零配件的情况和具体数值。针对上述情形，给出具体的决策方案、决策的依据和相应指标。

问题4：假设问题2与问题3中零配件、半成品和承成品的次品率均是通过问题1使用的抽样检测方法得出的，请重新完成问题2与问题3。

# 问题分析

## 问题总分析

企业生产过程中通常面对产品质量检验问题与生产决策问题。针对抽样检测问题可能涉及到统计学中已有的假设检验定理公式。

针对生产决策问题，企业在采购零配件、装配零配件以及出售成品中有着多个决策分支，而由于这里的决策基本上是二分选择，即是或者否，则整个决策过程可以形式化为一棵二叉树，而找到最优决策方案的问题本质上是要将决策过程数学建模成为一个路径搜索遍历的问题，或者说是树的遍历问题，二者在形式上是一致的，我们的最终目的是找到最大利润。总体需要通过引入0-1变量表示树的分支节点、最后在叶子节点计算成本费用和最终利润来确定具体的决策方案。具体问题分析如下：

### 问题1的分析

将供应商提供的所有零件看成一个大样本，假设该供应商提供的零配件样本容量足够大，当样本量足够大时，二项分布可以近似为正态分布，为设计合适的抽样检测方案，本文假设问题1采用简单随机抽样，并且使用统计学中的假设检验方法，并通过二项比例检验来决定是否接收这批零件。在给定的显著性水平和我们选定的标准检验力下，基于具体的假设检验理论，使用假设检验样本量计算公式，计算得出最少需要的样本量。得到样本量后，再根据检验统计量反推出抽样检测所抽出的小样本的次品率的范围，已知样本量与次品率，则自然可推出抽样的样本中次品数的范围。

### 问题2的分析

企业在采购零配件、装配零配件以及出售成品中有着多个决策分支，而由于这里的决策基本上是二分选择，即是或者否，则整个决策过程可以形式化为一棵二叉树，而找到最优决策方案的问题本质上是要将决策过程数学建模成为一个路径搜索遍历的问题，或者说是树的遍历问题，二者在形式上是一致的，我们的最终目的是找到最大利润。从采购零配件到出售成品，需要经过多个决策阶段，不同的决策将会导致各个阶段成本的变化，题中首先给出了两种零配件的购买单价、次品率和检测成本，在装配阶段，题中给出成品的次品率、装配成本与检测成本，流入市场阶段，题中给出对不合格成品的调换损失与拆解费用。通过引入0-1变量表示树的分支节点、最后在叶子节点计算成本费用和最终利润来确定具体的决策方案。

### 问题3的分析

问题3与问题2相似，但在问题2的基础上增加多个零配件以及工序，零配件与成品两种状态之间出现半成品的状态，由此决策分支也有所增加。但本质上，一个半成品的生产与问题2成品的生产并无根本性的不同，所以这是一个自相似的分形结构，可以利用动态规划的思想将大问题转化为小问题。根据题中给出的2道工序、8个零配件的情况，我们同样引入0-1变量，继续沿用问题2中的模型，但需要增加多个环节的费用计算，即对问题2中的数学模型进行适当改进。依旧以最大化利润为目标函数，得出相应决策方案。

### 问题4的分析

# 模型假设与约定

1. 假设抽样检测中样品数量足够大。
2. 假设每次抽取相互独立。
3. 假设最终成品的次品率是由合格零件装配抽样得出。
4. 假设进入市场的次品全部被客户退回。
5. 不考虑生产过程中除丢弃或拆解造成的零件损毁情况。
6. 不考虑一定时间范围内市场价格的波动变化。

# 符号说明

|  |  |  |
| --- | --- | --- |
| 符号 | 说明 | 单位 |
|  | 标称次品率 |  |
|  | 需要的最少样本数 |  |
|  | 置信度的值 |  |
|  | 检验力的值 |  |
|  | 次品率超出标称值的幅度 |  |
|  | 零配件过程中的成本 | 元 |
|  | 装配过程中的成本 | 元 |
|  | 进入市场过程中的成本 | 元 |
|  | 成品总销售额 | 元 |
|  | 最终利润 | 元 |

# 模型的建立与求解

## 问题1的模型

本文将使用统计学中的**假设检验方法**来决定是否接收这批零件。假设检验的目标是通过样本数据来判断是否有足够的证据拒绝零假设，并支持备选假设。在抽样检测方案设计时，样本量对假设检验有着至关重要的影响：样本太少：可能没有足够的证据拒绝零假设，即无法识别次品率是否超过10%。；**样本太多**：虽然增加样本量能提高检验的精度，但会浪费资源，带来不必要的成本。因此，我们希望找到**最小的样本量**，在给定的**显著性水平**和选定的一般**检验力**下做出准确的决策。根据假设检验的**样本量计算公式**算出**最小的样本量后，再根据假设检验，确定次品个数界限。**

首先我们假设总体次品率为，并从中抽取个样本，统计其中的次品数，在样本量足够大时，根据中心极限定理，二项分布可近似为正态分布，即：



其中样本次品数量的均值是，方差是。

其次，为进行假设检验，将样本次品率进行标准化，标准化公式为：



其中，为样本次品率，为标准误差。

### 假设检验的样本量计算公式

在假设检验中，通常使用以下公式计算样本量：



其中为需要的样本量；为显著性水平（置信度）的值；为检验力的值，即检验力表示在零假设为假时，正确拒绝零假设的概率，通常情况下，我们希望检验力为80%，取；为次品率，即供应商承诺的次品率；为希望检测的差异，即次品率超过的幅度，如过小，则需大量样本来检测出差异，设定一个合理的值可以帮助我们找到具有实际意义且可以接受的差异，本文假设合理的。

### 置信度时的样本量及抽样检测标准

针对第一种情形，设定如下假设：

Ⅰ零假设：次品率，即次品率不超过标称值。

Ⅱ备选假设：次品率，即次品率超过标称值。

对于置信度，采用单尾检验，，带入参数计算可得：



即在的信度下认定零配件次品率超过标称值，在显著性水平为的条件下，以的检验力来检验次品率是否超过标称值，并检测到至少的次品率差异，作出拒绝决策，至少需要223个样本。

然后根据假设检验，，当时，此时，即抽取223个样本，其中次品数大于29个时，拒绝零假设，接受备选假设，即在的信度下认定零配件次品率超过标称值，拒绝这批零件。

### 置信度时的样本量及抽样检测标准

针对第二种情形，适当修改上述两个假设：

Ⅰ零假设：次品率，即次品率超过标称值。

Ⅱ备选假设：次品率，即次品率不超过标称值。

同样采用单尾检验，，带入参数计算可得：

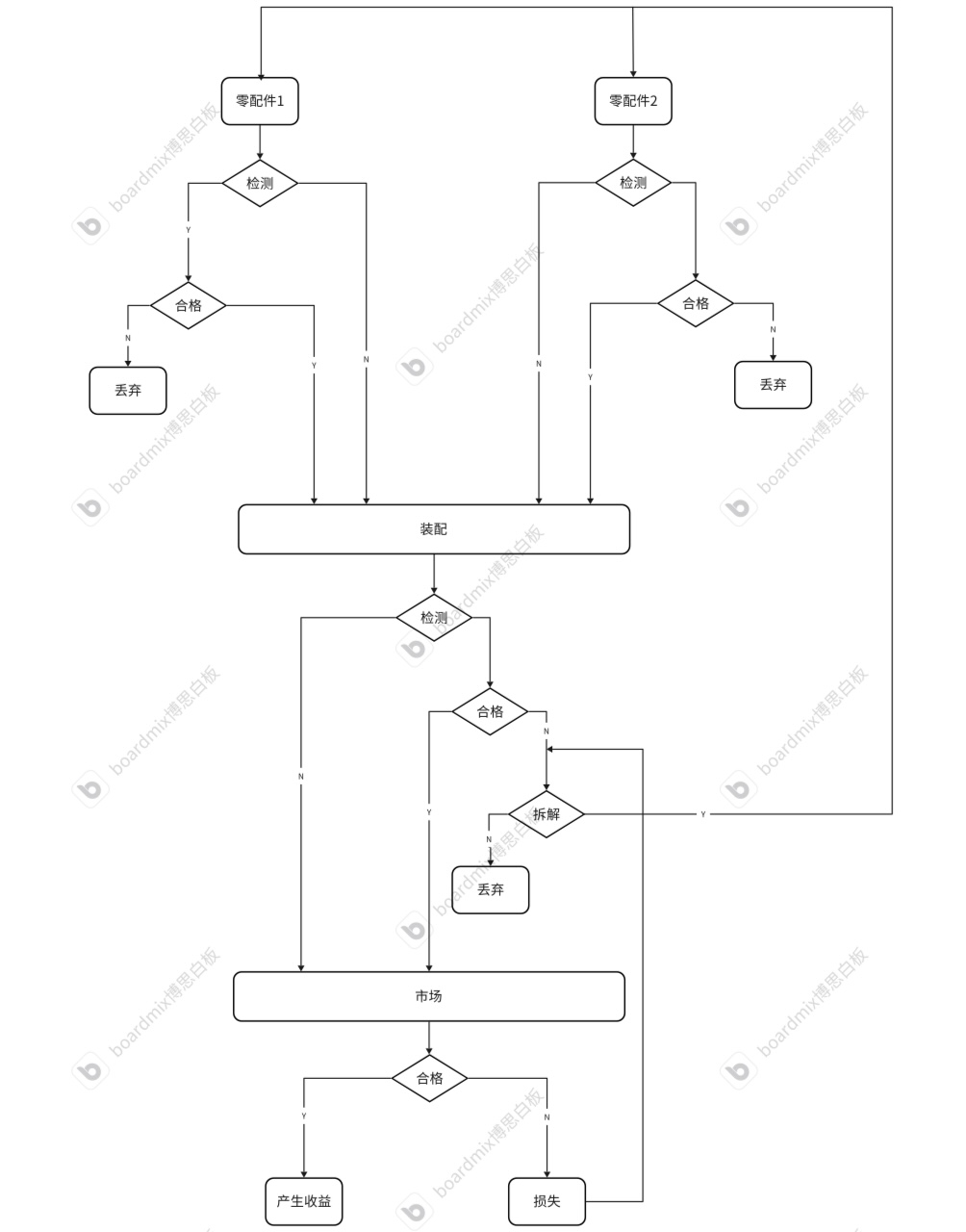


即在的信度下认定零配件次品率超过标称值，在显著性水平为的条件下，以的检验力来检验次品率是否超过标称值，并检测到至少的次品率差异，作出接收决策，至少需要162个样本。

然后根据假设检验，，当时，此时，即抽取162个样本，其中次品数小于12个，拒绝零假设，接受备选假设，即在的信度下认定零配件次品率不超过标称值，接受这批零件。

## 问题2的模型

已知两种零配件和成品次品率以及生产过程中涉及到的不同费用，对题中表1的情形给出具体的决策方案，这是一个决策问题，整个过程涉及多个分支（见流程图图1），首先我们引入0-1变量，建立统一的数学模型。



图表 1 生产流程图

### 模型准备

1. 参数假设

假设购买零配件1、零配件2的个数为、，购买单价为、，次品率为、，检测成本为、；装配而成的成品个数为，装配成本为，次品率为，检测成本为，市场售价为；对于不合格成品的调换损失为，拆解费用为。除次品率外参数单位均为元/件。

假设最终成品的销售总额为，最终利润为，单位均为元。

（2）0-1变量

在步骤（1）引入0-1变量，，即

，

在装配环节的步骤（2）、步骤（3）引入0-1变量、，即

，

最终在流入市场环节中的步骤（4），企业无条件对用户购买的不合格品进行调换，在处理被退回的不合格品环节，引入0-1变量，即



### 不同环节的费用计算

1. 零配件环节

企业从供应商中购买零配件1和零配件2，此时产生零件购买成本与，即：



若企业决定对零配件1与零配件2进行检测，则产生检测成本与，并且将检测出的不合格零配件丢弃，此时零配件1与零配件2的个数变为；否则，分别含有与次品的零配件1和2直接进入装配环节，其中

，

综上，在该环节中，涉及到的成本费用为



1. 装配环节

企业将两个零配件装配成成品，由于装配一件成品一定需要零配件1与零配件2，故对于上一环节中的两种零配件个数, 我们取为装配而成的成品数量，假设此时装配成本为。若对装配好的每一件成品进行检测，则产生检测成本，最终只有检测合格的成品进入到市场；若对装配好的每一件成品不进行检测，则含有次品的成品直接进入到市场。其中，

，

而对装配好的每一件成品进行检测后，对于不合格的产品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节，为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的成品仍需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的成品个数为



1. 市场环节

成品最终进入市场以单价进行售卖，获得总销售额，若在装配环节中对成品未进行检测，则有部分不合格产品流入市场，对于用户购买的不合格品，企业将无条件予以调换，此时产生一定的调换损失成本，如物流成本和企业信誉等；而对于被退回的不合格品，企业将选择将其拆解或直接丢弃，若选择拆解，则产生拆解成本。同样，为简化模型，拆解后的零件需进行检测，且装配而成的成品也需进行检测。其中，

，，

综上，在该环节中，涉及到的成本费用为



最终根据利润公式：利润=总销售额-总成本，建立数学模型，其中我们对于的零件量设置初值（如）。

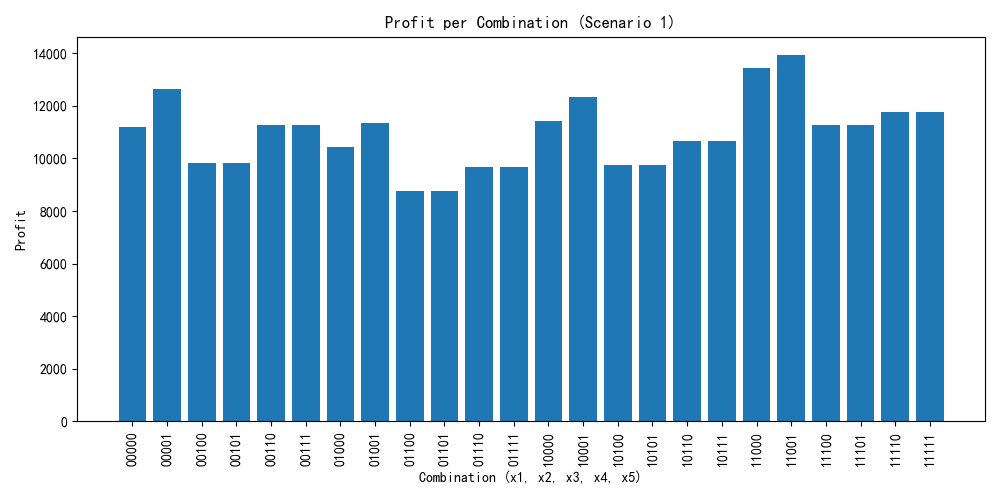
### 模型的求解

根据题目中表1所给出不同情况数据代入，最终得出具体决策方案以及相应指标结果（程序与全部图片见附件1），见下表1。

表1不同情况下的决策方案和指标结果

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **情况/决策** | **是否检测零配件** | | **是否检测成品** | **不合格品是否进行拆解** | **退回不合格品是否进行拆解** | **总利润/元** |
| 1 | 2 |
| 1 | 是 | 是 | 否 | 否 | 是 | 13925.0 |
| 2 | 是 | 是 | 否 | 否 | 否 | 3408.0 |
| 3 | 是 | 是 | 否 | 否 | 是 | 11765.0 |
| 4 | 是 | 是 | 是 | 是 | 否 | 6632.0 |
| 5 | 否 | 是 | 否 | 否 | 否 | 7552.0 |
| 6 | 是 | 否 | 否 | 否 | 否 | 18562.0 |

以情况1为例，给出同一情况下不同决策与利润的关系图，见图2。 其他图片见附件image\_q2。



图表 2情况1不同决策与利润间的关系

### 模型可视化

为观察各决策路径产生利润差异的原因，我们以情况1为例，绘制箱型图和小提琴图。

图表, 箱线图

描述已自动生成

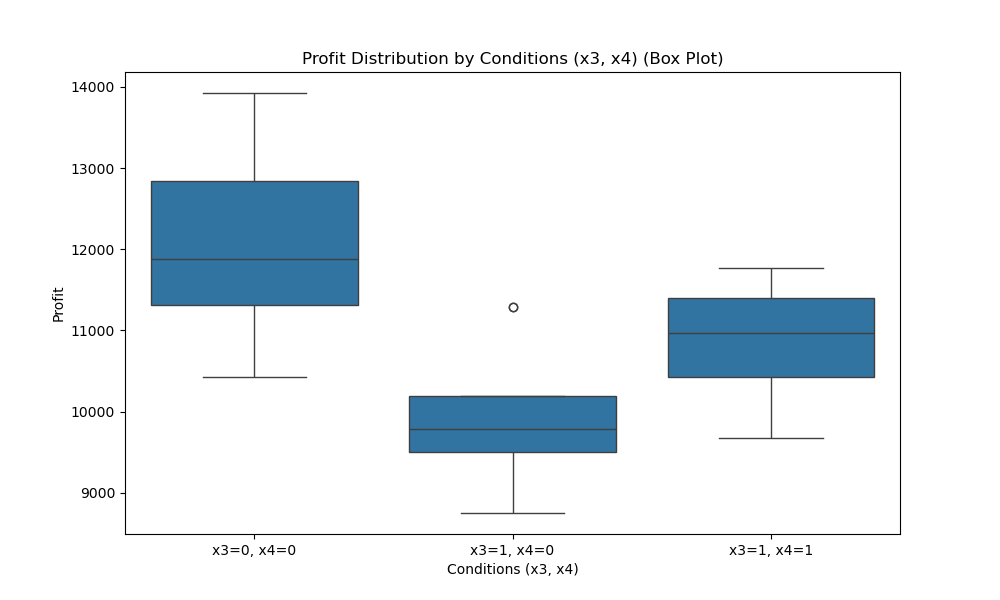
图表 3 是否进行零件检测对总利润的贡献箱型图

形状

描述已自动生成

图表 4是否进行零件检测对总利润的贡献小提琴图

由以上两图可看出，对零件1和零件2都进行检测整体利润会更高。



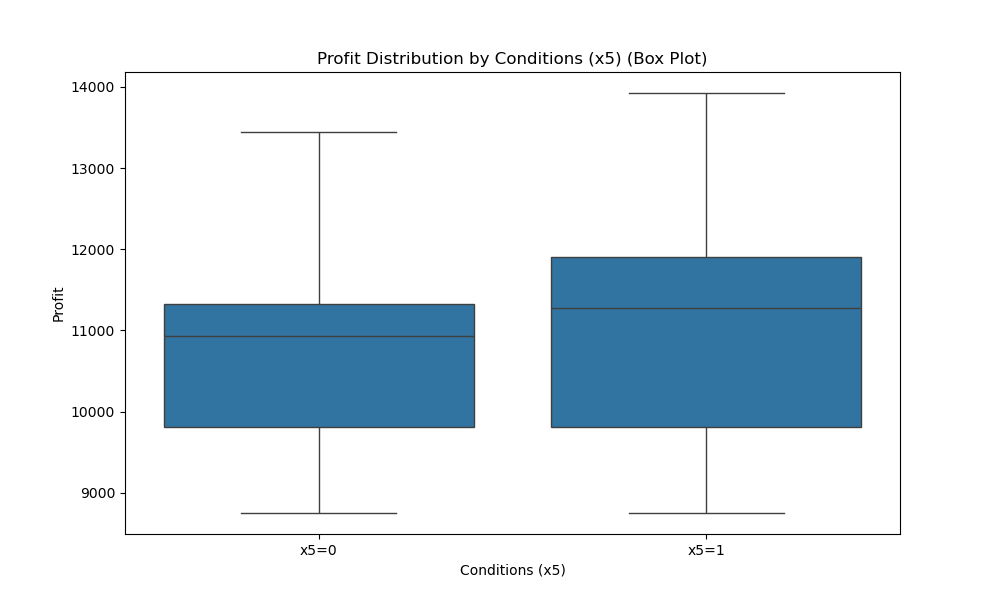
图表 5是否对成品进行检测和拆解对总利润的贡献箱型图

图表

描述已自动生成

图表 6是否对成品进行检测和拆解对总利润的贡献小提琴图

由于必须先进行检测才能进行拆解，所以只有x3为1时，x4才有意义。由图可以看出，不对成品检测会有更高的利润上限，但整体数据较分散，而进行检测后利润区间则相对集中，这说明检测保证了产品质量从而稳定了利润，而不检测则会产生一定的随机性。



图表 7调换的不合格产品是否拆解对总利润的贡献箱型图

图表, 雷达图

描述已自动生成

图表 8调换的不合格产品是否拆解对总利润的贡献小提琴图

从图中可以看出，对于调换的不合格产品是否拆解产生的总利润相差不多，但是明显不拆解更加稳定，因为不合格产品中存在次品零件的概率更高，对退回的不合格产品进行拆解需要面对拆解后无法回本的风险。

## 问题3的模型

对于问题3，题目在问题2的基础上增加了多个零配件与工序，且相比于问题2，生产过程中不同参数和决策分支均有所增加，为表示生产过程中的决策方案，即对每个零配件和半成品的检测与否、对成品的检测和拆解与否进行决策，我们依旧引入0-1变量，建立统一数学模型。

### 模型准备

1. 参数假设

假设购买零配件的个数为，购买单价为，次品率为，检测成本为（）；装配而成的半成品的个数为，次品率为，装配成本为，检测成本为，拆解费用为（）；装配而成的成品个数为，装配成本为，次品率为，检测成本为，市场售价为；对不合格成品的调换损失为，拆解费用为。

由于零配件，半成品与成品的次品率均为，为简化过程，统一用字母表示。

假设最终成品的销售总额为，最终利润为，单位均为元。

（2）0-1变量

在零配件环节引入0-1变量代表是否对零配件进行检测，即



在半成品的装配环节引入0-1变量代表是否对半成品进行检测，即



且引入0-1变量代表是否对检测出的不合格半成品进行拆解，即



在成品的装配环节引入0-1变量、分别代表是否对成品进行检测和是否对检测出的不合格品进行拆解，即

，

最终在流入市场环节中的环节引入0-1变量代表是否对退回的不合格品进行拆解，即



### 不同环节的费用计算

本题中共有四个环节，即零配件环节、半成品装配环节、成品装配环节与流入市场环节。

（1）零配件环节

企业依旧从供应商中购买零配件，零配件从问题2的2个增加至8个，此时产生零件购买成本，即：



若企业决定对零配件进行检测，则产生检测成本，并且将检测出的不合格零配件丢弃，由此零配件的个数变为；否则，含有次品的零配件直接进入半成品装配环节，其中

，

综上，在该环节中，涉及到的成本费用为



1. 半成品装配环节

企业将零配件1，2，3装配成半成品1，将零部件4，5，6装配成半成品2，将零部件7，8装配成半成品3，由于零配件齐全才能装配半成品，故对于上一环节中的零配件个数，与问题2同理，我们需要取数量所含最小的必须零配件，即



假设此时装配成本为。若对装配好的每一件半成品进行检测，则产生检测成本，最终只有检测合格的半成品进入成品的装配环节；若对装配好的每一件半成品不进行检测，则含有次品的半成品直接进入成品的装配环节。其中，

，

而对装配好的每一件半成品进行检测后，对于不合格的半成品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节，为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的半成品和成品仍需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的半成品个数为



（3）成品装配环节

企业将半成品1，2，3装配成成品，由于装配一件成品一定共同需要半成品1，2，3，故对于上一环节中的三种半成品个数，我们取为装配而成的成品数量，假设此时装配成本为。若对装配好的每一件成品进行检测，则产生检测成本，最终只有检测合格的成品进入到市场；若对装配好的每一件成品不进行检测，则含有次品的成品直接进入到市场。其中，

，

而对装配好的每一件成品进行检测后，对于不合格的产品，企业若需对其进行拆解，则产生拆卸成本，拆卸后的零件将重新进入零配件环节，为简化模型，不妨假设拆卸后的零件都需要进行检测，由拆卸零件做成的半成品与成品需进行检测；若企业对不合格成品不进行拆卸，则直接将不合格成品丢弃。其中，



综上，在该环节中，涉及到的成本费用为



最终进入市场的成品个数为



（4）流入市场环节

经过两个环节最终装配而成成品最终进入市场以单价进行售卖，获得总销售额，若在上一成品装配环节中对成品未进行检测，则有部分不合格产品流入市场，对于用户购买的不合格品，企业将无条件予以调换，此时产生一定的调换损失成本，如物流成本和企业信誉等；而对于被退回的不合格品，企业将选择将其拆解或直接丢弃，若选择拆解，则产生拆解成本。同样，为简化模型，拆解后的零件需进行检测，且装配而成的半成品和成品也需进行检测。其中，

，，

综上，在该环节中，涉及到的成本费用为



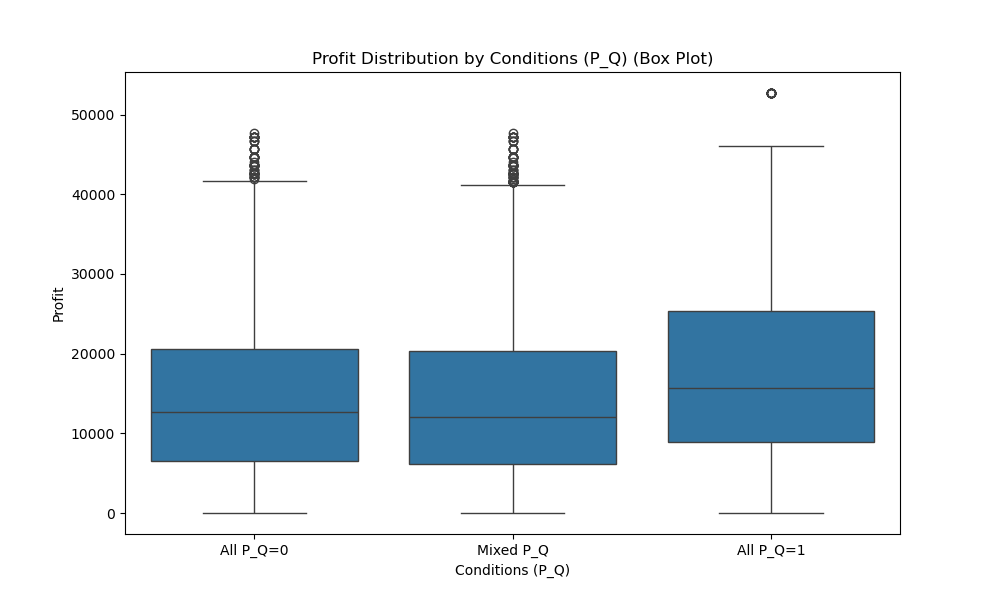
最终根据利润公式：利润=总销售额-总成本，建立数学模型。 这里默认初始的各种零件数为

### 两道工序、8个零件条件下的具体决策方案与指标结果

表2具体决策方案与指标结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **是否对零配件进行检测** | | | | | | | |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **是否对半成品进行检测** | | | | | | | |
| **1** | | **2** | | | **3** | | |
| **0** | | **0** | | | **0** | | |
| **是否对半成品中的不合格品进行拆解** | | | | | | | |
| **1** | | **2** | | | **3** | | |
| **0** | | **0** | | | **0** | | |
| **是否对成品进行检测** | | | | | | | |
| **1** | | | | | | | |
| **是否对成品中的不合格品进行拆解** | | | | | | | |
| **1** | | | | | | | |
| **是否最被退回的不合格品进行拆解** | | | | | | | |
| **0** | | | | | | | |
| **最终利润** | | | | **52717** | | | |

### 模型可视化



图表 9是否对成品进行检测拆解对利润的贡献箱型图

由图表9可知，同时进行检测和拆解利润最大，并且有一个明显高的离群值。整体离群值较多，且离群值较为集中。

图表

描述已自动生成

图表 10是否对成品进行检测拆解对利润的贡献小提琴图

图表, 箱线图

描述已自动生成

图表 11零件检测对于利润的贡献箱型图

由图表10可以看出，整体而言，都检测会有更高的利润，但是完全不检测而在后期检测会出现非常高的离群值。表明成品检测效果好于零件检测。

图表, 箱线图

描述已自动生成

图表 12 半成品是否检测对利润的贡献箱型图

由图11可以看出，对于半成品，不检测的利润会更高，这是由于半成品的检测不可替代性较差，前有零件检测，后有成品检测。

## 问题4的模型

# 模型的检验与分析

灵敏度分析和误差分析

# 模型评价、改进与推广

## 模型的优点

1. 直接性与直观性：对于不同的生产情况，不用在最初就考虑过于复杂的情况，能直接根据问题本身和给出的数据较为清晰地得出总的生产成本以及生产利润作出直观的生产流程图，进行程序化实现后，与实际情况进行比对，作出合理解释，从而为不同情况制定不同的生产方案。
2. 计算简便性：可以依据流程图每步进行成本和损失的计算从而较系统地评估每个决策步骤。
3. 动态规划性：考虑到在工序当中的检测、拆解和重组等工序中可能存在零件丢弃的情况从而使总的零件数目有所改变，分别对应设置初始值，使得总零件数量随着流程而变化进行循环，通过策略迭代进行优化，进一步地，可在此基础上对问题规模进行扩展。
4. 在后续循环的工作周期里合理地省略步骤，从而降低问题的复杂性且避免了死循环。比如对要召回拆解重组的零件统一进行检测，用拆卸的零件做的成品也进行检测。

## 模型的缺点

1. 为简化模型，采用随机抽样的抽样方式，可能造成效率低下。
2. 为简化模型，假设拆卸后的零配件和这些零配件组成的成品、半成品都需要进行检验，可能会增加检验成本。
3. 拆解或丢掉的零件对初始零配件的数量值的影响较为模糊，结果可能存在误差。

## 模型的改进与推广

1. 可进一步探究零配件拆解以及丢弃对初始零件值的准确影响。
2. 可考虑生产过程中的零件损耗率，以使得该模型更加具有实用性。
3. 由于该模型本质思想为引入0-1变量代表决策过程，计算各环节费用以及最大化利润，除考虑生产电子产品过程中的决策方案外，该模型可以应用至不同企业的不同产品生产过程或不同项目决策过程。

# 参考文献

【1】陈希孺 高等数理统计学 中国科学技术大学出版社 2009年08月

作者，论文名，杂志名，卷期号，起止页码，出版年

附录（相关代码与结果图片）

# 附录

## 其他图片

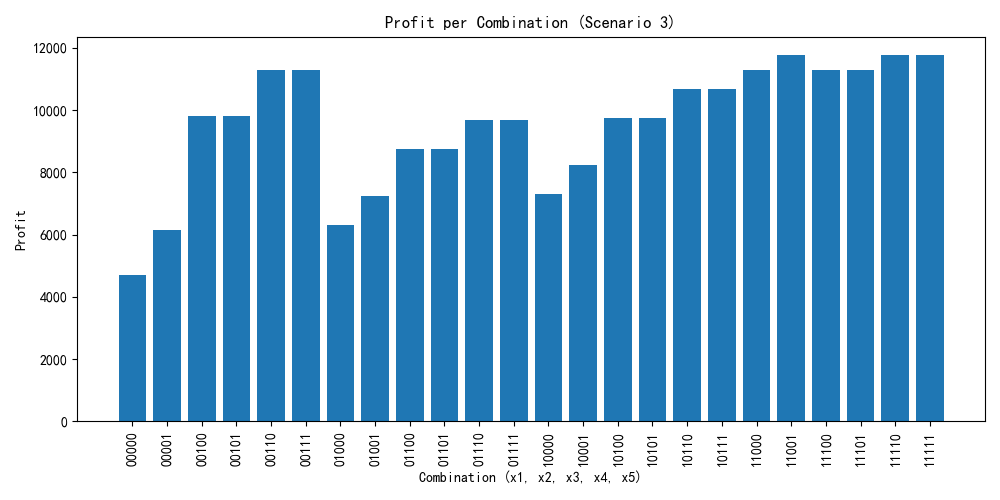
图表, 雷达图

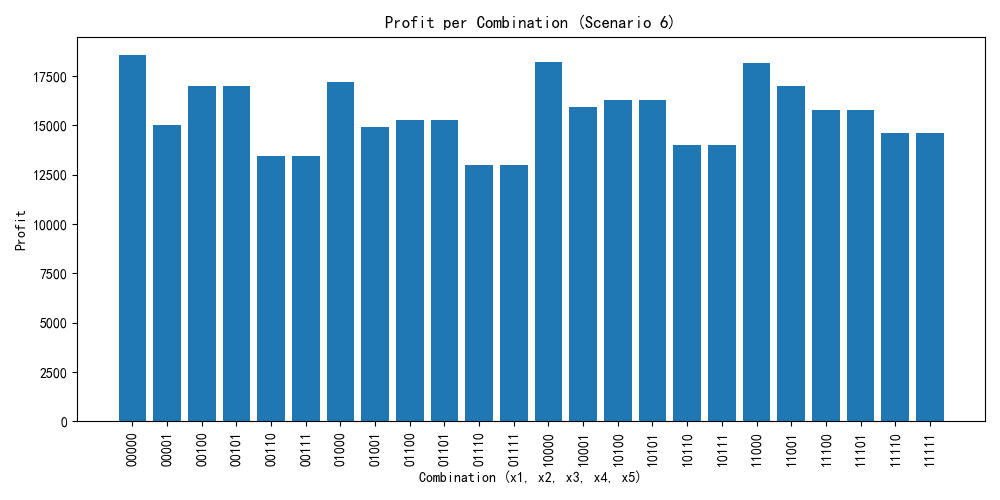
描述已自动生成图表, 雷达图

描述已自动生成图表, 雷达图

描述已自动生成图表, 雷达图

描述已自动生成图表, 条形图

描述已自动生成图表, 条形图

描述已自动生成

## 代码

Q2.py

import pandas as pd

import numpy as np

from scipy.stats import t

import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus']=False #解决负数坐标显示问题

# 定义零配件1的购买及检测成本

# 两个参数，x1为0-1变量，表示是否检测，n表示购买数量，m表示第m种情况

def cost\_spare1(x1,n,m):

# cost = data['零配件 1 购买单价'][m] \* n+ x1\*(data['零配件 1 检测成本'][m] \* n+data['零配件 1 次品率'][m] \* n \*data['零配件 1 购买单价'][m])

cost = data['零配件 1 购买单价'][m] \* n+ x1\*(data['零配件 1 检测成本'][m] \* n)

num\_of\_spare1 = n-x1\*int(n\*data['零配件 1 次品率'][m])

return cost,num\_of\_spare1

# 定义零配件2的购买及检测成本

def cost\_spare2(x2,n,m):

cost = data['零配件 2 购买单价'][m] \* n+ x2\*(data['零配件 2 检测成本'][m] \* n)

num\_of\_spare2 = n-x2\*n\*data['零配件 2 次品率'][m]

return cost,num\_of\_spare2

def Num\_of\_reject(x1,x2,num\_of\_finished\_products,m):

# 如果x1=1,x2=1,则不合格的只由成品组装导致

if x1==1 and x2==1:

num\_of\_reject\_final=int(num\_of\_finished\_products\*data['成品次品率'][m])# 次品个数

# 如果x1=1,x2=0,则需要先确认零配件2的次品个数，再在剩下的正品零件中再减去装配产生的次品

if x1==1 and x2==0 :

num\_of\_reject\_2 = int(num\_of\_finished\_products\*data['零配件 2 次品率'][m])

num\_of\_reject\_final = int((num\_of\_finished\_products-num\_of\_reject\_2)\*data['成品次品率'][m])+num\_of\_reject\_2

# 如果x1=0,x2=1,则需要先确认零配件1的次品个数，再在剩下的正品零件中再减去装配产生的次品

if x1==0 and x2==1:

num\_of\_reject\_1 = int(num\_of\_finished\_products\*data['零配件 1 次品率'][m])

num\_of\_reject\_final = int((num\_of\_finished\_products-num\_of\_reject\_1)\*data['成品次品率'][m])+num\_of\_reject\_1

# 如果x1=0,x2=0,由于两个都会产生次品，只有(1-零件1次品率)\*(1-零件2次品率)\*num\_of\_finished\_products的零件组是正品,其余都是次品。在正品零件组中减去装配产生的次品

if x1==0 and x2==0:

num\_of\_qualify = int((1-data['零配件 1 次品率'][m])\*(1-data['零配件 2 次品率'][m])\*(1-data['成品次品率'][m])\*num\_of\_finished\_products)# 正品数量

num\_of\_reject\_final = int(num\_of\_finished\_products-num\_of\_qualify)

return num\_of\_reject\_final

# 定义成品装配及检测拆解的成本,n1，n2分别表示零配件1和零配件2进入装配环节的数量,x4只有当x3=1时才有意义

def cost\_finished\_products(n1,n2,m,x1,x2,x3,x4):

num\_of\_finished\_products = min(n1,n2) # 可以产生成品的零件组数量

cost\_assemble = num\_of\_finished\_products\*data['成品装配成本'][m] # 装配成本，成本4

cost\_test = num\_of\_finished\_products\*data['成品检测成本'][m] # 检测成本，成本5

num\_of\_reject = Num\_of\_reject(x1,x2,num\_of\_finished\_products,m) # 次品个数

cost\_chaijie = num\_of\_reject\*data['不合格成品拆解成本'][m] # 拆解成本，成本7

cost\_of\_all = cost\_assemble+x3\*(cost\_test+x4\*cost\_chaijie) # 总成本

num\_of\_finished\_products = num\_of\_finished\_products - x3\*num\_of\_reject # 实际产生的成品数量

return cost\_of\_all,num\_of\_finished\_products,x4\*num\_of\_reject

# 定义不合格成品调换损失，以及召回的不合格成品数量,n为成品数量，m为第m种情况

def cost\_loss(x1,x2,x3,num\_of\_finished\_products,m):

# 如果经过成品检测到市场，不合格成品的数量为0

if x3==1:

return 0,0

else:

num\_of\_loss = Num\_of\_reject(x1,x2,num\_of\_finished\_products,m) # 次品个数

loss = num\_of\_loss\*data['不合格成品调换损失'][m] # 总调换损失

return loss,num\_of\_loss

# 定义总利润函数

#x1,x2,x3,x4分别表示零配件1是否检测，零配件2是否检测，成品是否检测，不合格成品是否拆解,召回的成品是否拆解

def profit(x1,x2,x3,x4,x5,n1,n2,m):

m-=1

# 第一轮

# 零配件的购买及检测成本

cost1,num\_of\_spare1 = cost\_spare1(x1,n1,m)

cost2,num\_of\_spare2 = cost\_spare2(x2,n2,m)

# 零配件使用后剩余的零件数量，由于可能有些零配件检测出次品数不同，可能出现两种配件数量不匹配的情况

n1-=min(num\_of\_spare1,num\_of\_spare2)

n2-=min(num\_of\_spare1,num\_of\_spare2)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3,num\_of\_finished\_products,num\_of\_chaijie = cost\_finished\_products(num\_of\_spare1,num\_of\_spare2,m,x1,x2,x3,x4)

# 不合格成品调换损失，以及召回的不合格成品数量

cost4,num\_of\_loss = cost\_loss(x1,x2,x3,num\_of\_finished\_products,m)

total\_cost = cost1+cost2+cost3+cost4+x5\*num\_of\_loss\*data['不合格成品拆解成本'][m]

total\_profit = data['市场售价'][m]\*(num\_of\_finished\_products-num\_of\_loss)-total\_cost

# 拆解所得零件和召回的零件进行第二轮，默认全部检测且不再进行拆解

n1 += num\_of\_chaijie+x5\*num\_of\_loss

n2 += num\_of\_chaijie+x5\*num\_of\_loss

if n1>0 and n2>0:

cost1,num\_of\_spare1 = cost\_spare1(1,n1,m)

cost2,num\_of\_spare2 = cost\_spare2(1,n2,m)

cost3,num\_of\_finished\_products,num\_of\_chaijie = cost\_finished\_products(num\_of\_spare1,num\_of\_spare2,m,1,1,1,0)

cost4,num\_of\_loss = cost\_loss(1,1,1,num\_of\_finished\_products,m)

total\_cost = cost1+cost2+cost3+cost4

total\_profit += data['市场售价'][m]\*(num\_of\_finished\_products-num\_of\_loss)-total\_cost

return total\_profit

# 主程序

# 导入itertools用于生成组合

import itertools

# 载入数据

data = pd.read\_csv('data/Q2.csv')

# 将相关列转换为浮点数，确保可以进行数学运算

# 假设数据列 '零配件 1 次品率' 和其他次品率列可能以百分比或字符串形式存在

data['零配件 1 次品率'] = pd.to\_numeric(data['零配件 1 次品率'].str.rstrip('%'), errors='coerce') / 100

data['零配件 2 次品率'] = pd.to\_numeric(data['零配件 2 次品率'].str.rstrip('%'), errors='coerce') / 100

data['成品次品率'] = pd.to\_numeric(data['成品次品率'].str.rstrip('%'), errors='coerce') / 100

# 确保所有其他涉及到数值运算的列都是数值类型

data['零配件 1 购买单价'] = pd.to\_numeric(data['零配件 1 购买单价'], errors='coerce')

data['零配件 2 购买单价'] = pd.to\_numeric(data['零配件 2 购买单价'], errors='coerce')

data['市场售价'] = pd.to\_numeric(data['市场售价'], errors='coerce')

print(data)

# 生成所有有效的组合

combinations = list(itertools.product([0, 1], repeat=5))

valid\_combinations = [combo for combo in combinations if combo[3] == 0 or combo[2] == 1]

# 循环m取值1到6

for m in range(1, 7):

# 存储每种m值对应的组合及利润

results = []

for combo in valid\_combinations:

p = profit(\*combo, n1=1000, n2=1000, m=m)

results.append({'x1': combo[0], 'x2': combo[1], 'x3': combo[2], 'x4': combo[3], 'x5': combo[4], 'profit': p})

# 创建 DataFrame，并将 x1, x2, x3, x4, x5 分开

df\_results = pd.DataFrame(results)

# 将结果保存到CSV文件，文件名根据m值不同

csv\_filename = f'result/result\_q2/result\_q2\_m{m}.csv'

df\_results.to\_csv(csv\_filename, index=False, encoding='utf-8-sig')

# 找到最大利润

max\_profit = df\_results['profit'].max()

print(f"第 {m} 种情况的最大利润为: {max\_profit}")

print('此时，x1, x2, x3, x4, x5的取值为：')

print(df\_results.loc[df\_results['profit'].idxmax()])

# 为绘图准备横坐标的组合字符串

df\_results['combination'] = df\_results[['x1', 'x2', 'x3', 'x4', 'x5']].astype(str).agg(''.join, axis=1)

# 绘制利润图，横坐标为组合字符串

plt.figure(figsize=(10, 5))

plt.bar(df\_results['combination'], df\_results['profit'])

plt.title(f"Profit per Combination (Scenario {m})")

plt.xlabel("Combination (x1, x2, x3, x4, x5)")

plt.ylabel("Profit")

plt.xticks(rotation=90)

plt.tight\_layout()

# 保存图片，文件名根据m值不同

pdf\_filename = f'image/image\_q2/Q2\_scenario\_{m}.png'

plt.savefig(pdf\_filename, format='png')

Q3.py

import pandas as pd

import numpy as np

from scipy.stats import t

import matplotlib.pyplot as plt

import itertools

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode\_minus']=False #解决负数坐标显示问题

scenarios = {

# 零配件数据

"part1\_defect\_rate": 0.10, "part1\_cost": 2, "part1\_test\_cost": 1,

"part2\_defect\_rate": 0.10, "part2\_cost": 8, "part2\_test\_cost": 1,

"part3\_defect\_rate": 0.10, "part3\_cost": 12, "part3\_test\_cost": 2,

"part4\_defect\_rate": 0.10, "part4\_cost": 2, "part4\_test\_cost": 1,

"part5\_defect\_rate": 0.10, "part5\_cost": 8, "part5\_test\_cost": 1,

"part6\_defect\_rate": 0.10, "part6\_cost": 12, "part6\_test\_cost": 2,

"part7\_defect\_rate": 0.10, "part7\_cost": 8, "part7\_test\_cost": 1,

"part8\_defect\_rate": 0.10, "part8\_cost": 12, "part8\_test\_cost": 2,

# 半成品数据

"semi\_product1\_defect\_rate": 0.10, "semi\_product1\_assembly\_cost": 8, "semi\_product1\_test\_cost": 4, "semi\_product1\_disassembly\_cost": 6,

"semi\_product2\_defect\_rate": 0.10, "semi\_product2\_assembly\_cost": 8, "semi\_product2\_test\_cost": 4, "semi\_product2\_disassembly\_cost": 6,

"semi\_product3\_defect\_rate": 0.10, "semi\_product3\_assembly\_cost": 8, "semi\_product3\_test\_cost": 4, "semi\_product3\_disassembly\_cost": 6,

# 成品数据

"finished\_product\_defect\_rate": 0.10,"": 8, "finished\_product\_test\_cost": 6, "finished\_product\_disassembly\_cost": 10,

"market\_price": 200, "replacement\_loss": 40,

"num\_parts1": 100, "num\_parts2": 100

}

# 零件是否检测的0-1变量列表x\_values，零件的数量列表n\_parts，半成品的编号semi\_product\_number，半成品是否检测的0-1变量y\_value，半成品是否拆解的0-1变量z\_value/

# 成品是否检测的0-1变量p\_value，成品是否拆解的0-1变量q\_value

# 零部件的购买检测费用和剩余数量

def calculate\_cost\_and\_spare(part\_number, x, n, scenarios):

# 构造零件信息的键

part\_info\_key = f"{part\_number}\_defect\_rate"

# 获取零件的次品率

defect\_rate = scenarios[part\_info\_key]

# 获取零件的购买单价

purchase\_price = scenarios[f"{part\_number}\_cost"]

# 获取零件的检测成本

test\_cost = scenarios[f"{part\_number}\_test\_cost"]

# print(f"零件{part\_number}的购买单价为{purchase\_price}，次品率为{defect\_rate}，检测成本为{test\_cost},购买数量为{n}")

# 总成本包括购买成本和检测成本

total\_cost = purchase\_price \* n + x \* (test\_cost \* n)

# 可用的零件数量为订购数量检测出的次品数量

usable\_quantity = n - x \* int(n \* defect\_rate)

return total\_cost, usable\_quantity

# 半成品次品数量

def num\_of\_reject\_semi\_product(semi\_product\_number, x\_values, num\_of\_semi\_products, scenarios):

"""

计算指定半成品组装后的次品数量。

参数:

semi\_product\_number (int): 半成品的编号，用于确定使用的零件。

x\_values (list of int): 对应零件是否检测的0-1变量列表。

num\_of\_semi\_products (int): 半成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

int: 次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 获取当前半成品的次品率

defect\_rate\_semi\_product = scenarios[f"semi\_product{semi\_product\_number}\_defect\_rate"]

# 初始化合格组合的概率

qualified\_combination\_rate = 1

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 计算每个零件的有效次品率，并更新合格组合的概率

for i, part\_number in enumerate(part\_numbers):

defect\_rate = scenarios[f"{part\_number}\_defect\_rate"]

# 如果零件进行了检测，则其次品率为0

effective\_defect\_rate = 0 if x\_values[i] == 1 else defect\_rate

qualified\_combination\_rate \*= (1 - effective\_defect\_rate)

# 计算合格组合数量

num\_of\_qualified\_combinations = int(qualified\_combination\_rate \* num\_of\_semi\_products)

# 计算合格零件组合中由于装配过程产生的次品数量

num\_of\_unqualified\_combinations = int(num\_of\_qualified\_combinations \* defect\_rate\_semi\_product)

# 计算由不合格零件产生的次品数

num\_of\_unqualified\_parts = num\_of\_semi\_products - num\_of\_qualified\_combinations

# 计算总次品数量

num\_of\_reject\_final = num\_of\_unqualified\_combinations + num\_of\_unqualified\_parts

return num\_of\_reject\_final

# 半成品的装配、检测和拆解成本

def cost\_semi\_product(n\_parts, semi\_product\_number, x\_values, y\_value, z\_value, scenarios):

"""

计算半成品装配、检测和拆解的成本。

参数:

n\_parts (list of int): 每个零件的数量列表，例如 [n1, n2, n3]。

semi\_product\_number (int): 半成品的编号，用于确定使用的零件。

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_value (int): 半成品是否检测的0-1变量。

z\_value (int): 半成品是否拆解的0-1变量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含三个元素的元组，分别为总成本、实际产生的半成品数量,拆解的不合格成品数量,半成品次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 可以产生成品的零件组数量

num\_of\_semi\_products = min(n\_parts)

# 计算半成品的装配成本

cost\_assemble = num\_of\_semi\_products \* scenarios[f'semi\_product{semi\_product\_number}\_assembly\_cost']

# 计算半成品的检测成本

cost\_test = num\_of\_semi\_products \* scenarios[f'semi\_product{semi\_product\_number}\_test\_cost'] if y\_value == 1 else 0

# 计算半成品中的次品数量

num\_of\_reject = num\_of\_reject\_semi\_product(semi\_product\_number, x\_values, num\_of\_semi\_products, scenarios)

# 计算不合格半成品的拆解成本

cost\_chaijie = num\_of\_reject \* scenarios[f'semi\_product{semi\_product\_number}\_disassembly\_cost'] if y\_value == 1 and z\_value == 1 else 0

# 总成本

cost\_of\_all = cost\_assemble + cost\_test + cost\_chaijie

# 实际产生的半成品数量

num\_of\_semi\_products\_actual = num\_of\_semi\_products - num\_of\_reject if y\_value == 1 else num\_of\_semi\_products

# 拆解的不合格半成品数量

num\_of\_chaijie = num\_of\_reject if y\_value == 1 and z\_value == 1 else 0

return cost\_of\_all, num\_of\_semi\_products\_actual, num\_of\_chaijie, num\_of\_reject

# 成品次品数量

def num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios):

"""

计算成品的次品数量。

参数:

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

num\_of\_finished\_products (int): 成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

int: 次品数量。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 固定的半成品编号列表

semi\_product\_numbers = [1, 2, 3]

# 初始化合格组合的概率

qualified\_combination\_rate = 1

# 计算每个半成品及其零件的有效次品率，并更新合格组合的概率

for i, semi\_product\_number in enumerate(semi\_product\_numbers):

# 获取当前半成品对应的零件编号列表

part\_numbers = semi\_product\_parts.get(semi\_product\_number, [])

# 计算每个零件的有效次品率，并更新合格组合的概率

part\_effective\_defect\_rate = 1

for j, part\_number in enumerate(part\_numbers):

defect\_rate\_part = scenarios[f"{part\_number}\_defect\_rate"]

# 如果零件进行了检测，则其次品率为0

effective\_defect\_rate = 0 if x\_values[j] == 1 else defect\_rate\_part

part\_effective\_defect\_rate \*= (1 - effective\_defect\_rate)

# 获取当前半成品的次品率

defect\_rate\_semi\_product = scenarios[f"semi\_product{semi\_product\_number}\_defect\_rate"]

# 如果半成品进行了检测，则其次品率为0

effective\_defect\_rate\_semi\_product = 0 if y\_values[i] == 1 else defect\_rate\_semi\_product

# 更新合格组合的概率

qualified\_combination\_rate \*= (1 - part\_effective\_defect\_rate \* effective\_defect\_rate\_semi\_product)

# 计算有效组合数量

num\_of\_qualified\_combinations = int(qualified\_combination\_rate \* num\_of\_finished\_products)

# 计算合格零件组合中由于装配过程产生的次品数量

num\_of\_unqualified\_combinations = int(num\_of\_qualified\_combinations \* scenarios['finished\_product\_defect\_rate'])

# 计算成品中的次品数量

num\_of\_reject\_final = num\_of\_unqualified\_combinations + num\_of\_finished\_products - num\_of\_qualified\_combinations

return num\_of\_reject\_final

# 成品的装配、检测和拆解成本

def cost\_finished\_product(n\_semi\_products, x\_values, y\_values, p\_value, q\_value, scenarios):

"""

计算成品装配、检测和拆解的成本。

参数:

n\_semi\_products (list of int): 每个半成品的数量列表，例如 [n1, n2, n3]。

semi\_product\_numbers (list of int): 半成品的编号列表，例如 [1, 2, 3]。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

q\_value (int): 成品是否拆解的0-1变量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含四个元素的元组，分别为总成本、实际产生的成品数量、拆解的不合格成品数量、成品次品数。

"""

# 可以产生成品的半成品组数量

num\_of\_finished\_products = min(n\_semi\_products)

# 计算成品的装配成本

cost\_assemble = num\_of\_finished\_products \* scenarios['']

# 计算成品的检测成本

cost\_test = num\_of\_finished\_products \* scenarios['finished\_product\_test\_cost'] if p\_value == 1 else 0

# 计算成品中的次品数量

num\_of\_reject = num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios)

# 计算不合格成品的拆解成本

cost\_chaijie = num\_of\_reject \* scenarios['finished\_product\_disassembly\_cost'] if p\_value == 1 and q\_value == 1 else 0

# 实际产生的成品数量

num\_of\_finished\_products\_actual = num\_of\_finished\_products - num\_of\_reject if p\_value == 1 else num\_of\_finished\_products

# 总成本

cost\_of\_all = cost\_assemble + cost\_test + cost\_chaijie

# 拆解的不合格成品数量

num\_of\_chaijie = num\_of\_reject if p\_value == 1 and q\_value == 1 else 0

return cost\_of\_all, num\_of\_finished\_products\_actual, num\_of\_chaijie, num\_of\_reject

def cost\_loss(x\_values, y\_values, p\_value, num\_of\_finished\_products, scenarios):

"""

计算不合格成品的调换损失和召回的不合格成品数量。

参数:

x\_values (list of int): 对应零件是否检测的0-1变量列表。

y\_values (list of int): 对应半成品是否检测的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

num\_of\_finished\_products (int): 成品的数量。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

tuple: 包含两个元素的元组，分别为总调换损失和召回的不合格成品数量。

"""

# 如果经过成品检测到市场，不合格成品的数量为0

if p\_value == 1:

return 0, 0

else:

# 计算成品的次品数量

num\_of\_reject = num\_of\_reject\_finished\_product(x\_values, y\_values, num\_of\_finished\_products, scenarios)

# 计算总调换损失

# 使用 scenarios 中的 replacement\_loss 字段

loss = num\_of\_reject \* scenarios['replacement\_loss']

return loss, num\_of\_reject

def profit(x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, n\_parts, scenarios):

"""

计算总利润，并记录每种零件被拆解得到的数量。

参数:

x\_values (list of int): 八个零件是否检测的0-1变量列表。

y\_values (list of int): 三个半成品是否检测的0-1变量列表。

z\_values (list of int): 三个半成品是否拆解的0-1变量列表。

p\_value (int): 成品是否检测的0-1变量。

q\_value (int): 成品是否拆解的0-1变量。

r\_value (int): 市场退回的不合格成品是否拆解的0-1变量。

n\_parts (list of int): 八个零件的数量列表。

scenarios (dict): 包含所有零件和半成品数据的字典。

返回:

float: 总利润。

"""

# 定义每个半成品对应的零件编号

semi\_product\_parts = {

1: ['part1', 'part2', 'part3'],

2: ['part4', 'part5', 'part6'],

3: ['part7', 'part8']

}

# 初始化拆解得到的零件数量

disassembled\_parts = {f'part{i + 1}': 0 for i in range(len(n\_parts))}

# 第一轮

# 零配件的购买及检测成本

costs = []

num\_of\_spares = []

for i in range(len(n\_parts)):

part\_key = f'part{i + 1}'

cost, num\_of\_spare = calculate\_cost\_and\_spare(part\_key, x\_values[i], n\_parts[i], scenarios)

# print(f'零配件{i + 1}的购买及检测成本为{cost}，剩余数量为{num\_of\_spare}')

costs.append(cost)

num\_of\_spares.append(num\_of\_spare)

# 零配件使用后剩余的零件数量

num\_of\_semi\_products = min(num\_of\_spares)

for i in range(len(n\_parts)):

n\_parts[i] -= num\_of\_semi\_products

# 半成品装配及检测拆解的成本,以及实际产生的半成品数量，拆解的不合格半成品数量

semi\_product\_costs = []

semi\_product\_nums = []

for semi\_product\_num in range(1, 4):

cost\_semi, num\_of\_semi\_products\_actual, num\_of\_chaijie, \_ = cost\_semi\_product(

[num\_of\_semi\_products] \* 3, semi\_product\_num, x\_values, y\_values[semi\_product\_num - 1], z\_values[semi\_product\_num - 1], scenarios)

# 更新拆解得到的零件数量

parts\_to\_add = semi\_product\_parts[semi\_product\_num]

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

semi\_product\_costs.append(cost\_semi)

semi\_product\_nums.append(num\_of\_semi\_products\_actual)

# 最少的半成品数量

num\_of\_semi\_products\_actual = min(semi\_product\_nums)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3, num\_of\_finished\_products, num\_of\_chaijie, disassembled\_parts\_ = cost\_finished\_product(

[num\_of\_semi\_products\_actual] \* 3, x\_values, y\_values, p\_value, q\_value, scenarios)

# 更新拆解得到的零件数量

parts\_to\_add= ['part1', 'part2', 'part3', 'part4', 'part5', 'part6', 'part7', 'part8']

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

# 不合格成品调换损失，以及召回的不合格成品数量

cost4, num\_of\_loss = cost\_loss(x\_values, y\_values, p\_value, num\_of\_finished\_products, scenarios)

# 总成本

total\_cost = sum(costs) + sum(semi\_product\_costs) + cost3 + cost4 + r\_value \* num\_of\_loss \* scenarios['finished\_product\_disassembly\_cost']

# 总利润

total\_profit = scenarios['market\_price'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

# 拆解所得零件和召回的零件进行第二轮，默认全部检测且不再进行拆解

for i in range(len(n\_parts)):

n\_parts[i] += disassembled\_parts.get(f'part{i + 1}', 0) + r\_value \* num\_of\_loss

if all(n > 0 for n in n\_parts):

costs = []

num\_of\_spares = []

for i in range(len(n\_parts)):

part\_key = f'part{i + 1}'

cost, num\_of\_spare = calculate\_cost\_and\_spare(part\_key, 1, n\_parts[i], scenarios)

costs.append(cost)

num\_of\_spares.append(num\_of\_spare)

num\_of\_semi\_products = min(num\_of\_spares)

for i in range(len(n\_parts)):

n\_parts[i] -= num\_of\_semi\_products

# 半成品装配及检测拆解的成本,以及实际产生的半成品数量，拆解的不合格半成品数量

semi\_product\_costs = []

semi\_product\_nums = []

for semi\_product\_num in range(1, 4):

cost\_semi, num\_of\_semi\_products\_actual, num\_of\_chaijie, disassembled\_parts\_ = cost\_semi\_product(

[num\_of\_semi\_products] \* 3, semi\_product\_num, [1] \* 8, [1] \* 3, 0, scenarios)

# 更新拆解得到的零件数量

parts\_to\_add = semi\_product\_parts[semi\_product\_num]

for part in parts\_to\_add:

disassembled\_parts[part] += num\_of\_chaijie

semi\_product\_costs.append(cost\_semi)

semi\_product\_nums.append(num\_of\_semi\_products\_actual)

# 最少的半成品数量

num\_of\_semi\_products\_actual = min(semi\_product\_nums)

# 成品装配及检测拆解的成本,以及实际产生的成品数量，拆解的不合格成品数量

cost3, num\_of\_finished\_products, num\_of\_chaijie, disassembled\_parts\_ = cost\_finished\_product(

[num\_of\_semi\_products\_actual] \* 3, [1] \* 8, [1] \* 3, 1, 0, scenarios)

# 不合格成品调换损失，以及召回的不合格成品数量

cost4, num\_of\_loss = cost\_loss([1] \* 8, [1] \* 3, 1, num\_of\_finished\_products, scenarios)

# 总成本

total\_cost = sum(costs) + sum(semi\_product\_costs) + cost3 + cost4

# 更新总利润

total\_profit += scenarios['market\_price'] \* (num\_of\_finished\_products - num\_of\_loss) - total\_cost

return total\_profit

def main():

max\_profit = float('-inf')

best\_strategy = None

# 假设的零件数量

n\_parts = [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]

# 生成所有可能的参数组合

x\_values\_combinations = list(itertools.product([0, 1], repeat=8))

y\_values\_combinations = list(itertools.product([0, 1], repeat=3))

z\_values\_combinations = list(itertools.product([0, 1], repeat=3))

p\_value\_combinations = [0, 1]

q\_value\_combinations = [0, 1]

r\_value\_combinations = [0, 1]

all\_strategies = []

# 计算所有策略的利润

for x\_values in x\_values\_combinations:

for y\_values in y\_values\_combinations:

for z\_values in z\_values\_combinations:

for p\_value in p\_value\_combinations:

for q\_value in q\_value\_combinations:

for r\_value in r\_value\_combinations:

# 零件数量重置

n\_parts = [1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000]

current\_profit = profit(x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, n\_parts, scenarios)

all\_strategies.append((x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, current\_profit))

if current\_profit > max\_profit:

max\_profit = current\_profit

best\_strategy = (x\_values, y\_values, z\_values, p\_value, q\_value, r\_value)

#保存所有策略及其利润

with open('result/Q3.txt', 'w') as f:

for strategy in all\_strategies:

x\_values, y\_values, z\_values, p\_value, q\_value, r\_value, current\_profit = strategy

f.write(f"strategy: X={x\_values}, Y={y\_values}, Z={z\_values}, P={p\_value}, Q={q\_value}, R={r\_value}, profit: {current\_profit}\n")

f.write(f"\nmax\_profit: {max\_profit}\n")

f.write(f"best\_strategy: X={best\_strategy[0]}, Y={best\_strategy[1]}, Z={best\_strategy[2]}, P={best\_strategy[3]}, Q={best\_strategy[4]}, R={best\_strategy[5]}")

if \_\_name\_\_ == "\_\_main\_\_":

main()