生产过程中的决策问题

摘要

本文聚焦企业电子产品生产流程中的核心决策问题，通过构建数学模型与量化分析，系统解决了抽样检测方案设计、单工序生产决策优化及多工序多零配件场景下的综合决策难题，为企业平衡成本与风险、提升运营效益提供了系统性解决方案。

对于问题一，针对在 10% 标称次品率下设计 95% 与 90% 信度下检测次数最少的抽样方案这一需求，我们基于统计假设检验理论构建模型。首先设定零假设（样本比例等于标称值）与备择假设（样本比例偏离标称值），利用中心极限定理将次品数量的分布近似为正态分布，推导样本容量计算方法。

置信水平	临界值 $Z_{\alpha/2}$	容忍区间 d	标称次品率 p_0	最小样本量 n
95%	1.960	0.02	0.10	865
90%	1.645	0.02	0.10	609

对于问题二，基于六种零配件与成品参数组合，构建包含资源回收机制的多轮迭代利润模型，以最大化总利润为目标优化生产决策。模型采用 0-1 **规划**求解，考虑成品销售收益、检测成本、装配成本、调换损失及拆解后的零件循环利用价值。通过枚举法对 16 种可能策略组合进行计算。

对于问题三，将模型推广至多道工序、多个零配件的通用场景，针对给定的 2 道工序、8 个零配件流程，构建包含半成品与零件双循环回收的利润模型。模型新增半成品检测与拆解决策，引入**遗传算法**解决多变量带来的计算复杂性。求解结果显示，最优策略为“全零配件检测 + 全半成品检测 + 不检测成品 + 成品拆解 + 半成品 2 和 3 拆解”，此时单位利润最大（74.8344 元）。该策略通过早期检测降低次品流转风险，同时利用拆解回收高价值资源，实现了多环节成本与收益的动态平衡。

最后，本文分析了模型的优缺点：经验公式与 0-1 规划模型简洁高效，但存在对次品独立性假设的局限及变量增多时计算复杂度激增的问题，并提出结合成本整合优化与智能算法改进的方向，模型可推广至供应链质检、可靠性测试等多领域。

关键字： 抽样检测方案；遗传算法；0-1 规划；OC 曲线；多工序生产

一、 问题重述

1.1 问题背景

为高效利用有效资源、减少生产成本、提高运营收益，优化生产决策是每个企业经营管理中的核心环节。某企业生产电子产品需采购两种零配件，并将其装配后形成成品，且成品合格性受零配件质量影响，因此企业需通过抽样检测控制零配件次品率，并对不合格成品选择报废或拆解。生产过程中需决策是否检测零配件或成品、是否拆解不合格品，并考虑调换市场退回品的损失。对此我们需要设计抽样方案、优化生产阶段决策，并推广到多工序多零配件场景，以通过成本与风险平衡提升整体效益。

1.2 题设数据

表一给出了两种零配件和成品的次品率，以及购买单价、检测成本、市场售价等参数信息。

表二给出了八个零配件在两道工序中的次品率、购买单价、检测成本、拆解费用等参数信息。

图一给出了给出了 2 道工序、8 个零配件的基本流程

1.3 需要解决问题

问题一：在 10% 的标称度下，求出在 95% 与 90% 的信度下使检测次数尽可能少的抽样检测方案。

问题二：基于给定的六种情况下零配件与成品的参数，给出使利润最大化的最优生产决策方案。

问题三：在 m 道工序、n 个零配件生产系统中，基于各环节的次品率、成本和售价等参数，制定最优的检测、拆解和调换决策方案，并对给定的 2 道工序、8 个零配件给出具体决策依据和量化结果。

二、 模型假设

假设一：零配件的次品率相互独立，且与成品的次品率相互独立。

假设二：厂家每轮生产策略相同，即递归过程中零配件的次品率不变。

假设三：合格成品的市场售价与不合格成品的调换损失为定值，不随市场需求变化。

假设四：单位零配件所产生的期望利润相同。

三、 符号说明

符号	说明	单位
n	样本数	-
Z_α	临界值	-
α	置信度	-
p_0	标称值	-
H_0	零假设	-
H_1	备择假设	-
X	次品数量	个
p	次品率	-
d	容忍区间	-
Π	总利润期望	元
q	合格率	-
c_j	单位零件成本	元
s	市场售价	元
x_{ij}	第 i 道工序中第 j 个部件检测状态	-
d_{ij}	第 i 道工序中第 j 个部件检测成本	个
a_j	单位成品装配成本	元
t	拆解成本	元
l	调换损失	元
θ	回收比例	-

(其余符号详见正文)

四、 问题分析

4.1 对问题一的分析

题目要求我们在标称值确定的情况下，设计出一种合理的抽样检测方案，使得在给定的两种不同情形下，抽样的次数最小。要使抽样的次数最小，即使得抽样的样本容量最小即可。基于此，我们根据经典检验样本容量公式，结合题目的标称值得到理想样本比例，同时根据不同情形下的置信度确定临界值，最终计算出抽样次数。

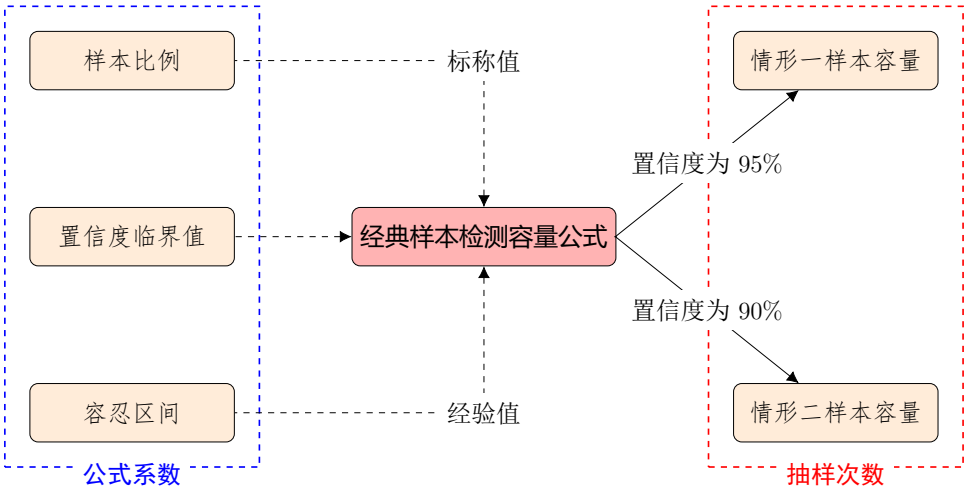


图 1　第一问流程图

4.2 对问题二的分析

题目要求我们设计出一种最优的生产决策方案，使得在给定的六种情况下，利润最大化。我们首先对各工序之间的关系，绘制了如下流程图:

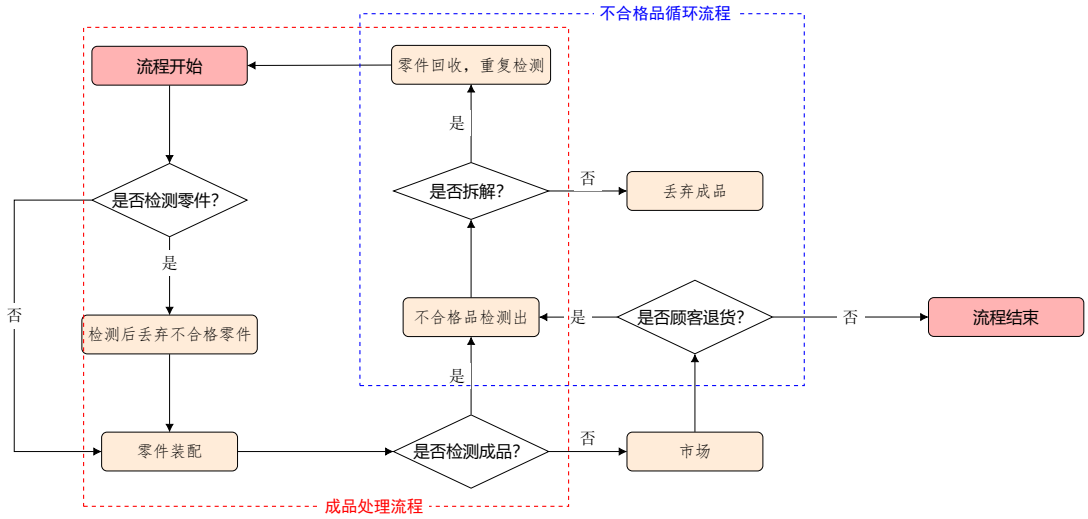


图 2　各工序关系流程图

为定量刻画检测与拆解策略在多轮装配过程中的利润演化情况，本文建立了一个包含资源回收机制的多轮迭代模型。假设在初始轮次中系统仅使用原始投入的零部件，装配所得成品经由检测决定是否进入市场，不合格成品可选择拆解，回收所得零部件将进入下一轮生产。模型在每一轮中动态更新原材料结构，重构装配过程中的次品率与资源使用情况，并据此计算当轮利润。

在利润计算方面，每一轮均考虑成品销售收益、检测成本、装配成本、调换成本与拆解成本。成品利润受检测策略影响，若跳过检测，次品流入市场将引致一定的损失；若实施检测，则可避免售后损失但需承担检测成本。同时，拆解策略决定回收零部件的数量和质量，进而影响下一轮的装配结构与合格率，利润也由此产生间接影响。

最终，系统的长期收益由所有轮次的利润叠加构成，其中每一轮利润会乘以回收带来的资源贡献程度。通过控制迭代次数，直到收益收敛，我们能够更全面地评估不同检测与拆解策略的长期经济效益，为企业在资源循环利用背景下的决策优化提供可靠依据，得到利润最大化的生产决策方案。

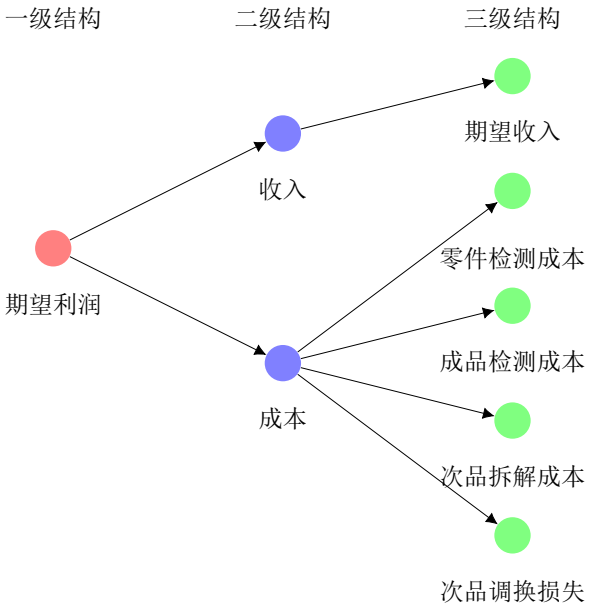


图 3　利润结构图

4.3 对问题三的分析

题目要求我们设计出一种最优的检测、拆解和调换决策方案，并对给定的 2 道工序、8 个零配件给出具体决策依据和量化结果。我们基于第二问的模型，对利润结构进行适当的重构，引入半成品工序所导致一系列成本。同时由于半成品工序的存在，我们需要对循环利润进行适当调整，即在零件回收利润的基础上，引入了半成品回收利润。最后将得到目标函数利用遗传算法进行优化，得到利润最大化的生产决策方案。

五、 问题一的模型的建立和求解

5.1 模型建立

5.1.1 假设框架

根据题设条件，我们构建了如下假设框架:

- 零假设 H_0：样本比例 p 等于标称值 p_0
- 备择假设 H_1：对于拒收情形，样本比例 p 大于标称值 p_0；对于接受情形，样本比例 p 小于标称值 p_0。

5.1.2 样本容量公式

假设从总体中抽取容量为 n 的样本，记其中的次品个数为随机变量 X，显然该变量服从二项分布 $X \sim Bin(n,p)$。其中 p 为样本比例。根据中心极限定理，当 n 足够大时，X 可以近似服从正态分布，即

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

进一步的，我们可确定样本中的次品率的统计量 $\hat{p}$ 近似服从正态分布，即

$$\hat{p} \sim N(p, \frac{p(1-p)}{n}) \tag{2}$$

由上式我们可推导出经典样本检测容量公式

$$n = \frac{Z_\alpha^2 p(1-p)}{d^2} \tag{3}$$

其中 Z_α 为置信区间在标准正态分布中的临界值，α 为置信度，d 为容忍区间，一般取经验值。

5.2 模型求解

将题设条件代入上式，同时，我们取容忍区间为 0.02，最终求得结果如下表

表 1　不同置信水平下的样本量计算（$d = 0.02, p_0 = 0.10$）

置信水平	临界值 $Z_{\alpha/2}$	容忍区间 d	标称次品率 p_0	最小样本量 n
95%	1.960	0.02	0.10	865
90%	1.645	0.02	0.10	609

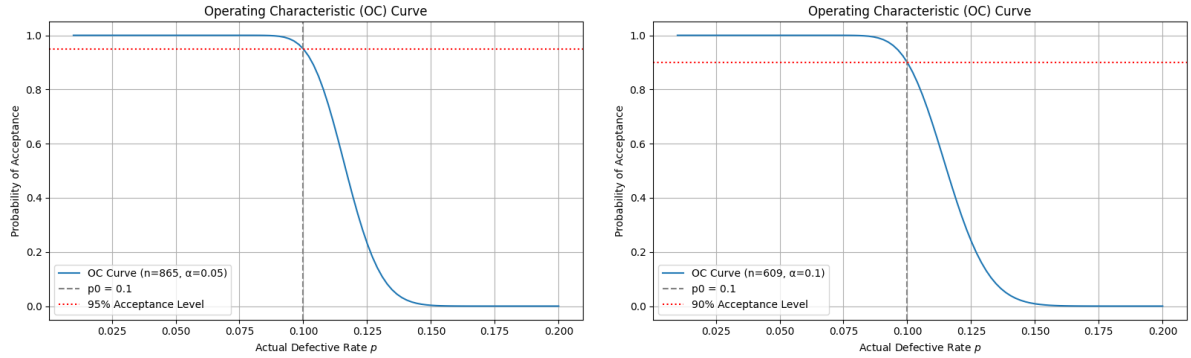
由表可知，在 95% 置信水平下，临界值 $Z_{\alpha/2} = 1.960$，容忍区间 $d = 0.02$，标称次品率 $p_0 = 0.10$，最小样本量 $n = 865$。在 90% 置信水平下，临界值 $Z_{\alpha/2} = 1.645$，容忍区间 $d = 0.02$，标称次品率 $p_0 = 0.10$，最小样本量 $n = 609$。

5.3 结果分析

上述结果表明，若零件总数远大于最小样本量时，无论样本容量为何值时，样本容量取结果值时能取到满足题设条件与统计约束上的最优值。

为进一步验证得到的样本容量的合理性，本文拟采用 OC 曲线对结果进行可视化分析，OC 曲线反映了在不同实际不合格率 p 下，产品被接受的概率 P_{accept}。

通过计算，并绘制了如下的 OC 曲线：



(a) 样本空间为 865 的 OC 曲线　　(b) 样本空间为 609 的 OC 曲线

图 4　双图

利用 OC 曲线可以直观地观察在不同实际不合格率下当前抽样计划的接受概率。从图4a中不难发现，当实际不合格率接近标称值 $p_0 = 0.10$ 时，接受概率约为 95%，与设定置信水平一致；特别地，我们注意到，随着实际不合格率上升，接受概率陡然下降，说明该检验方案对不合格产品的敏感性较高，即对不合格产品的识别能力较强。

同时，对于质量较好的批次，该方案也能保持较高的接受概率，体现了良好的灵敏度和平衡性。我们从图4b也能得到类似的结论。

此外，根据 OC 曲线我们还能得出，该种抽样方案下对不合格品的最大容忍数 $n_{\max}$，即若发现样本容量中不合格数大于该数值，则拒收，其算式为

$$n_{\max} = \frac{Z_{\alpha/2}^2 p_0(1-p_0)}{d^2} \tag{4}$$

对于情形一而言，$n_{\max} = 103$，即若抽样容量中不合格数大于 103，则拒收。对于情形二而言，$n_{\max} = 74$，即若抽样容量大于 74，则拒收。

六、 问题二的模型的建立和求解

6.1 模型建立

6.1.1 决策变量设立

针对本问题中装配企业对零配件与成品的检测及拆解决策，我们从全局最优的角度出发，希望最大化企业的期望利润。由于每道工序（如零件检测、成品检测、成品拆解）均为可选动作，决策结果具有离散性，故本问题可建模为一个 0-1 整数规划问题。

故我们引入如下决策变量:

- x_{ij}：第 i 道工序中第 j 个零件的**检测状态**，$x_{ij}=1$ 表示检测通过，$x_{ij}=0$ 表示检测不通过。特别地，x_f 代表成品检测状态。
- d_{ij}：第 i 道工序中第 j 个零件的**检测成本**，d_{ij} 为非负实数，特别地，d_f 代表成品检测成本。
- a_{ij}：第 j 个零件的单位成品**固定成本**，即固定成本为零件装配成本或购买成本。a_j 为非负实数。特别地，a_f 代表成品固定成本。
- t：**总拆解成本**，t 为非负实数。
- l：**总调换损失**，l 为非负实数。
- θ_{ij}：第 i 道工序中第 j 个零件**回收比例**，θ 为实数。
- z_{ij}：从成品往后开始的第 i 道工序的第 j 个成品（或半成品）的**拆解状态**，$z_i=1$ 表示该零件已拆解，$z_i=0$ 表示该零件未拆解。
- p_{ij}: 第 i 道工序中第 j 个零件的**次品率**，p_{ij} 为实数。
- q_{ij}: 第 i 道工序中第 j 个零件的**合格率**，$q_{ij}=1-p_{ij}$。
- k_{ij}：第 i 道工序中第 j 个零件的**实际比例**，k_{ij} 为非负实数。特别地，k_f 代表成品实际比例。
- s：**市场售价**，s 为非负实数。

其中每一组决策 $(x_{11},x_{12},...,x_f,...,z_{21},z_1)$ 对应一种检测-拆解策略。

6.1.2 目标函数构建

由于不合格成品可被拆解，拆解出的零件可循环使用，投入到下一轮生产中时会产生对应的利润，本文称其为:“利润积累”效应。我们定义 π_0 为第一轮循环的单位利润，θ 为每轮可回收再利用的零件比例，显然的，对于第 i 轮的利润，它等于第本轮收益乘上回收比例 θ，即

$$\pi_i=(P_i-C_i)\theta^n \tag{5}$$

对于总利润而言，它等于每一轮的利润求和，则总利润的期望为：

$$\pi_{total} = \sum_{n=0}^{\infty} \pi_i = \pi_0 + \sum_{n=1}^{\infty} (P_i - C_i)\theta^n, \quad 0 < \theta < 1 \tag{6}$$

因此，我们最后的目标函数即为：

$$\max\{\pi_{total}\} \tag{7}$$

6.1.3 第一轮利润的构建

第一轮利润 π_0 可视为初始收益减去初始成本，即

$$\pi_0 = P_0 - C_0 \tag{8}$$

对于初始收益而言，其数值为市场售价、成品合格率、成品实际比例三者之积，即

$$P_0 = sq_f k_f \tag{9}$$

而对于初始成本，其数值为固定成本、检测成本、拆解成本、调换损失之和，对于本问，考虑仅有一道工序、两个零件的情况，我们由如下公式计算：

$$C_0 = \sum_{j=1}^{2} (a_{1j} + x_{1j} d_{1j}) + k_f (a_f + x_f d_f + z_1 t(1 - q_f) + (1 - x_f)(1 - q_f) l) \tag{10}$$

需要注意的是，q_f 由如下算式确定：

$$q_f = (1 - p_f)(1 - (1 - x_{11})p_1)(1 - (1 - x_{12})p_2) \tag{11}$$

而 k_f 由如下算式确定：

$$k_f = \frac{(1 - x_{11}p_1) + (1 - x_{12}p_2)}{2} \tag{12}$$

6.1.4 迭代利润的构建

以某一轮的利润 π_m 为例，其构成与第一轮类似，区别在于零件的成本与次品率的计算，以及回收系数的引入。

首先对于零件的成本而言，由于零件的来源为前一轮成品的拆解，故我们可以忽略其购买成本，即第一道工序中的固定成本 a_{1j}。

对于零件的次品率，由于零件的来源为前一轮成品的拆解，由于拆解的成品必为次品，次品成品中至少含有一个次品零件，故该轮中的次品率实际为，在已知两个零件中至少含有一个次品零件与次品率（实际为上一轮的次品率）的情况下，任选一个是次品的概率，即为条件概率，易得其算式为：

$$p_{ij}^{(n)} = \frac{p_{ij}^{(n-1)}}{1 - (1 - p_{ij}^{(n-1)})^2} \tag{13}$$

于是我们便有第 m 轮的收益，即

$$P_m = sq_f^{(m)}k_f^{(m)} \tag{14}$$

而对应的成本则为：

$$C_m = \sum_{j=1}^{2} x_{1j}d_{1j} + k_f^{(m)}(a_f + x_f d_f + z_1 t(1-q_f^{(m)}) + (1-x_f)(1-q_f^{(m)})l) \tag{15}$$

现在我们来确定回收系数，回收系数即从第 $m-1$ 轮的成品中回收零件的比例，它等于第 $m-1$ 轮的次品率、第 $m-1$ 轮的成品实际比例、第 $m-1$ 轮的回收系数之积，同时他还受拆解状态制约，由此我们最终有：

$$\theta_m = z_1(1-q_f^{(m-1)})k_f^{(m-1)}\theta_{m-1} \tag{16}$$

于是，第 m 轮的利润可由如下公式计算：

$$\pi_m = (P_m - C_m)\theta_m \tag{17}$$

最终迭代的总利润为：

$$\pi_{total} = \pi_0 + \sum_{m=1}^{\infty}\pi_m \tag{18}$$

6.1.5 模型求解

考虑问题二中，0-1 变量仅有四个，故总情况为 $2^4 = 16$ 种，即使考虑有六种不同的情形，最可能数也仅为 96 种，故可以采用枚举法求解。

代入表中数据，最终得到如下最优解，见表2

情况编号	利润/元	零件 1 检测	零件 2 检测	成品检测	拆解
1	13.4047	是	是	否	是
2	4.1750	是	是	否	是
3	11.1390	是	是	否	是
4	6.9998	是	是	是	是
5	11.2487	否	是	是	是
6	18.5867	否	否	否	否

表 2　各情形最优策略及对应单位利润

6.2 结果分析

通过对比可以发现，最大利润出现在情形 6（18.5867 元），该情况下次品率整体最低，企业选择完全不检测也不拆解，以节省各项成本，并凭借较高成品合格率实现最大化收益。相反，利润最小值出现在情形 2（4.175 元），该参数设定下次品率较高，且尽管执行全检测策略并启用拆解，仍难以弥补由质量波动带来的成本损耗，导致整体收益受限。

值得注意的是，情形 3 与情形 4 仅在“成品检测”上有所差异，前者选择跳过成品检测以降低成本，而后者引入成品检测以规避售后损失，二者单位利润分别为 11.1390 元与 6.9998 元，提示当拆解费用较低但调换损失较大时，是否检测成品对利润影响较显著。情形 5 的策略表明，在零件 1 质量稳定、但检测成本偏高时，放弃该零件检测反而有利于提升整体收益

为更直观展示各情形利润水平与策略分布关系，绘制如下柱状图

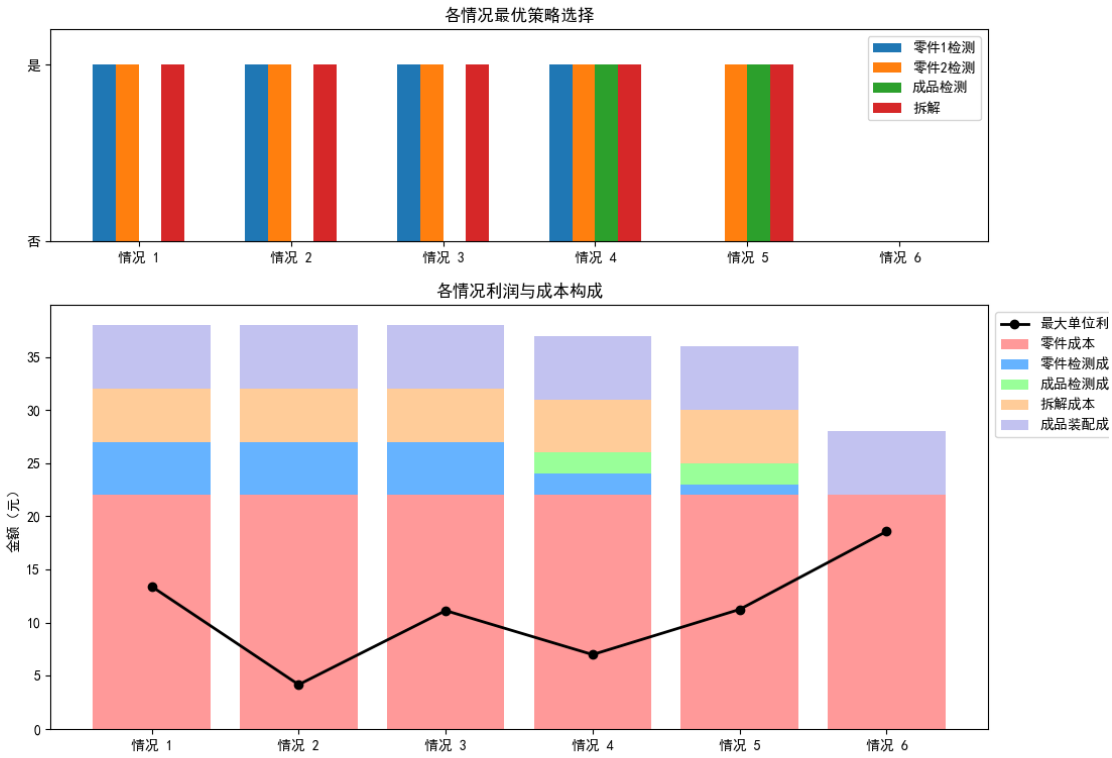


图 5　各情形利润与策略分布

明显的，由于情形 6 的成本较低，故其即使全部不检测也可获得较高收益。

七、 问题三的模型的建立和求解

7.1 模型建立

参照问题二的模型建立，问题三的模型建立与问题二相同，主要区别在于每轮循环的收益由半成品收益与零件收益两部分组成，即：

$$\pi_i = \pi_{half_i} + \pi_{componet_i} \tag{19}$$

其中，π_{half_i} 为第 i 轮半成品收益，$\pi_{componet_i}$ 为第 i 轮零件收益。

当然，我们在此还是先给出第一轮的收益计算公式：

$$\pi_0 = P_0 - C_0 \tag{20}$$

对于初始收益而言，其数值为市场售价、成品合格率、成品实际比例三者之积，即

$$P_0 = sq_fk_f \tag{21}$$

而对于初始成本，其数值为固定成本、检测成本、拆解成本、调换损失之和，对于本问，考虑仅有一道工序、两个零件的情况，我们由如下公式计算：

$$C_0 = \sum_{j=1}^{8}(a_{1j}+x_{1j}d_{1j})+\sum_{j=1}^{3}k_{1j}(a_{2j}+x_{2j}d_{2j}+z_{2j}(1-q_{2j})t_{2j})+k_f(z_1t_1(1-q_f)+(1-x_f)(1-q_f)l) \tag{22}$$

需要注意的是，q_f 由如下算式确定：

$$q_f = q_{21}q_{22}q_{23}(1-p_f) \tag{23}$$

对于任意的下级 q_{ij} 都有类似确定方式，在此不再赘述。而 k_f 由如下算式确定：

$$k_f = \frac{(1-x_{21}p_{21})k_{11}q_{11}q_{12}q_{13} + (1-x_{22}p_{22})k_{12}q_{14}q_{15}q_{16} + (1-x_{23}p_{23})k_{13}q_{17}q_{18}}{3} \tag{24}$$

对于任意的下级 k_{ij} 都有类似确定方式，在此不再赘述。

接下来我们对某一轮的收益进行计算，假设第 m 轮的收益为：

$$\pi_m = \pi_{half_m} + \pi_{componet_m} \tag{25}$$

其中，π_{half_m} 为第 m 轮半成品收益，$\pi_{componet_m}$ 为第 m 轮零件收益。

对于半成品收益而言，其数值为：

$$P_{half_m} = sq_f^{(m)}k_f^{(m)} \tag{26}$$

半成品回收后不存在装配成本，故其成本可写作

$$C_{half}^{(m)} = \sum_{j=1}^{3}(a_{2j}+x_{2j}d_{2j}+z_{2j}(1-q_{2j}^{(m)})t_{2j})+k_f^{(m)}(z_1t_1(1-q_f^{(m)})+(1-x_f)(1-q_f^{(m)})l) \tag{27}$$

对于零件收益而言，其数值为：

$$P_{component_m} = sq_{fc}^{(m)} k_{fc}^{(m)} \tag{28}$$

零件回收后不存在购买成本，故其成本可写作

$$C_{component}^{(m)} = \sum_{j=1}^{8}(x_{1j}d_{1j}) + \sum_{j=1}^{3} k_{1j}(a_{2j} + x_{2j}d_{2j} + z_{2j}(1-q_{c2j}^{(m)})t_{2j}) + k_f^{(m)}(z_1 t_1(1-q_{fc}^{(m)}) + (1-x_f)(1-q_{fc}^{(m)})l) \tag{29}$$

其中 q_{fc}^m 与 k_{fc}^m 及其相关衍生公式的确定方法同问题二，不再赘述。

最后，来确定回收系数，由于存在两种不同的回收利润，故回收系数也分为两种，分别为 θ_{half} 与 $\theta_{component}$，其计算公式如下：

$$\theta_{half}^{(m)} = z_1(1 - q_f^{(m-1)}) * k_f^{(m-1)} + z_1(1 - q_{fc}^{(m-1)}) * k_{fc}^{(m-1)} \theta_{beta}^{(m-1)} \theta_{half}^{(m-1)} \tag{30}$$

$$\theta_{component}^{(m)} = \frac{3z_{21}(1 - q_{21}^{m-1}) + 3z_{22}(1 - q_{22}^{m-1}) + 2z_{23}(1 - q_{23}^{m-1})}{8} \tag{31}$$

最后我们可以确定第 m 轮的收益，即：

$$\pi_m = (P_{half_m} - C_{half}^{(m)})\theta_{half}^{(m)} + (P_{component_m} - C_{component}^{(m)})\theta_{component}^{(m)} \tag{32}$$

总收益对上式进行求和即可

7.2 模型求解

由于决策变量变为了 16 个，模型求解情况指数级增长，求解起来较为困难，故我们采用遗传算法，在每轮迭代种，得出最稳健的解，并将其作为下一轮种群的初始种群。最终输出我们需要的决策集合，即使得利润最大的策略。

将题中数据代入模型，最终求得最优策略如下：

表 3　最佳检测与拆解策略下的各项决策取值（单位利润最大值：74.8344）

决策项	取值
零配件 1	是
零配件 2	是
零配件 3	是
零配件 4	是
零配件 5	是

续表：最佳检测与拆解策略下的各项决策取值

决策项	取值
零配件 6	是
零配件 7	是
零配件 8	是
半成品 1	是
半成品 2	是
半成品 3	是
成品	否
成品拆解	是
半成品 1 拆解	否
半成品 2 拆解	是
半成品 3 拆解	是

见表可得，最大利润为 74.8344 元。

7.3 结果分析

本研究建立了一个多级装配系统的检测与拆解策略优化模型，涵盖零配件、半成品及成品的检测和拆解决策，目标是最大化单位产品的期望利润。在充分考虑各类次品率、检测成本、装配费用、成品售价、拆解费用与调换损失等因素的基础上，我们穷举了所有 2^{16} 种可能策略组合，并计算每种方案对应的利润水平。结果表明，最优策略是在所有零配件与半成品阶段均进行检测，放弃成品检测，但保留对成品及部分半成品的拆解选项，最终单位利润达到最大值 74.8344 元。

从策略组合分析来看，检测确实带来成本增加，但在装配早期就发现并隔离次品，有效提高了最终成品的合格率，减少了调换损失，使得整体收益上升。同时，成品检测成本较高，且不能回收资源，放弃成品检测、改用拆解方式回收半成品和零配件，不仅节省了成本，也提高了资源利用率。最终策略正是检测与拆解之间实现平衡的最优解。

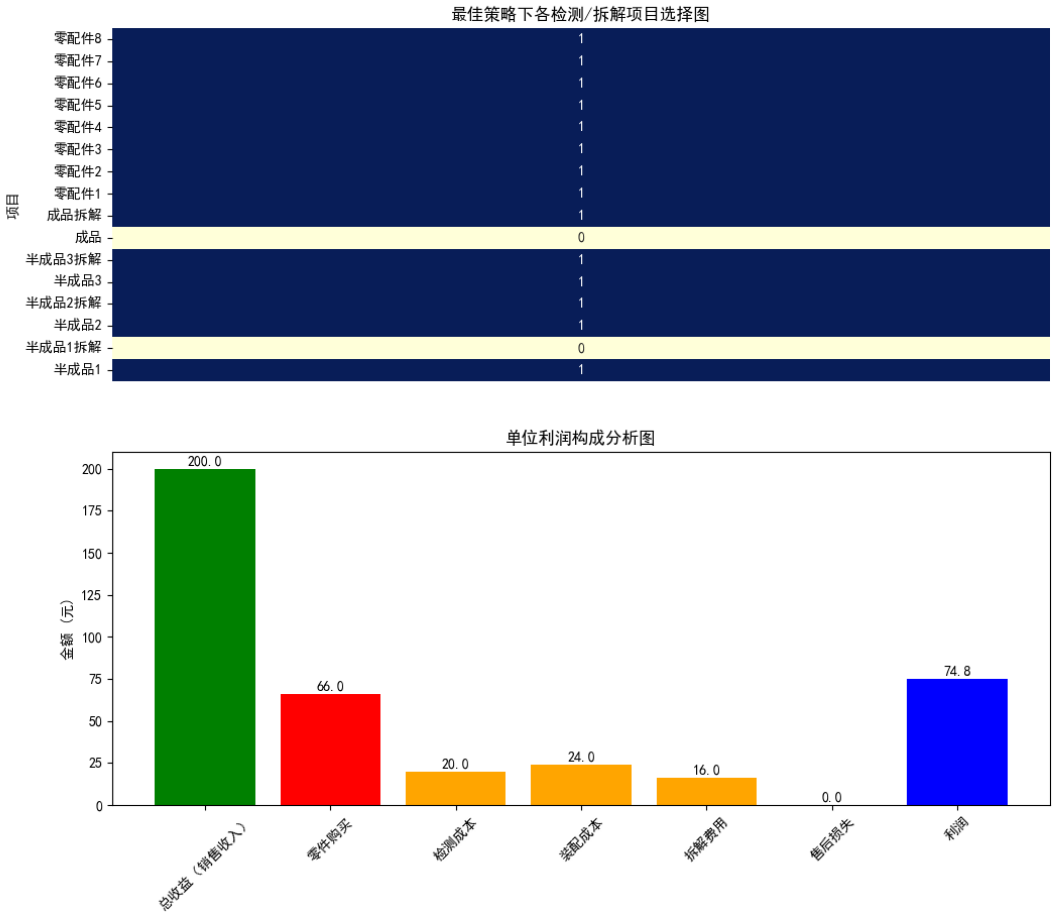


图 6　各情形利润与策略分布

从结果图上可以看出，不同策略下的合格率、成本和利润三者之间呈现显著的权衡关系。最佳策略在检测投入和资源回收之间取得了协调，从而最大程度地提升了系统整体的经济效益，为复杂制造流程中的策略制定提供了量化依据和优化路径。

八、 模型的评价

8.1 模型的优点

- 优点 1：经验样本容量公式计算简便，无需复杂统计工具，并且很好的控制了两类错误，即生产方风险和使用方风险。在信度要求变化时，公式能灵活调整抽样方案，实现了动态调整。
- 优点 2：0-1 规划整合复杂逻辑约束，简要精准的描述了“是否”类二元决策问题，简化了模型，适用于多阶段决策。

8.2 模型的缺点

- 缺点 1：问题一假设样本中的次品出现是独立且概率恒定的，但实际生产中，次品可能集中在某些批次，此时抽样结果会低估真实风险。
- 缺点 2：0-1 规划的计算复杂度高，变量增多时求解时间指数级增长，增大计算难度。

九、 模型的改进与推广

9.1 改进

- 改进 1：问题一可进行成本整合优化：将公式嵌入决策树或线性规划，同时优化样本量、检测成本、拆解费用等。
- 改进 2：

9.2 推广

- 推广 1：经验样本容量公式可推广到供应商来料检验、电子产品可靠性测试与寿命评估、市场质量反馈与售后分析等实际问题。
- 推广 2：0-1 规划可推广到投资组合优化、电路设计、广告投放、网络与路径优化等所有二元决策类问题。

参考文献

[1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[2] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.

[3] VÁZQUEZ J I H, GONZÁLEZ S H, VÁZQUEZ J O H, et al. Production planning through lean manufacturing and mixed integer linear programming[J]. Leather and Footwear Journal, 2021, 21(1):47-62.

[4] LEE A H, KANG H Y. A mixed 0-1 integer programming for inventory model: a case study of tft-lcd manufacturing company in taiwan[J]. Kybernetes, 2008, 37(1):66-82.

[5] CLAASSEN G. Mixed integer (0–1) fractional programming for decision support in paper production industry[J]. Omega, 2014, 43:21-29.

附录 A　文件列表

文件名	功能描述
exercise_1.py	问题一程序代码
exercise_2.py	问题二程序代码
exercise_3.py	问题三程序代码

附录 B　代码

exercise_1.py

```
import math
from scipy.stats import norm

# 定义参数
p0 = 0.10  # 标称次品率
alpha_95 = 0.05  # 95%置信水平
alpha_90 = 0.10  # 90%置信水平
z_95 = norm.ppf(1-alpha_95/2) # 95%的临界值
z_90 = norm.ppf(1-alpha_90/2 )  # 90%的临界值

# 计算样本量
def calculate_sample_size(z_alpha, p0, delta):
    n=(z_alpha/delta)**2*p0*(1-p0)
    return math.ceil(n)
# 假设检测误差 delta 为 5%
delta = 0.02
n_95=calculate_sample_size(z_95,p0,delta)
n_90=calculate_sample_size(z_90,p0,delta)
print(n_95,n_90)
```

exercise_2.py

```
import itertools
import pandas as pd
```

```python
from tabulate import tabulate
import math

# 表1数据（6种场景）
scenarios = [
    {"case":1, "p1":0.10,"c1":4,"d1":2, "p2":0.10,"c2":18,"d2":3, "pf":0.10,"cf":6,"df":3, "s":56,"L":6,"D":5},
    {"case":2, "p1":0.20,"c1":4,"d1":2, "p2":0.20,"c2":18,"d2":3, "pf":0.20,"cf":6,"df":3, "s":56,"L":6,"D":5},
    {"case":3, "p1":0.10,"c1":4,"d1":2, "p2":0.10,"c2":18,"d2":3, "pf":0.10,"cf":6,"df":3,"s":56 ,"L":30,"D":5},
    {"case":4, "p1":0.20,"c1":4,"d1":1, "p2":0.20,"c2":18,"d2":1, "pf":0.20,"cf":6,"df":2, "s":56,"L":30,"D":5},
    {"case":5, "p1":0.10,"c1":4,"d1":6, "p2":0.20,"c2":18,"d2":1, "pf":0.10,"cf":6,"df":2, "s":56,"L":10,"D":5},
    {"case":6, "p1":0.05,"c1":4,"d1":2, "p2":0.05,"c2":18,"d2":3, "pf":0.05,"cf":6,"df":3, "s":56,"L":10,"D":30},
]

# 计算利润函数，返回所有关键参数列表
def calculate_profit(x1, x2, x3, x4, params, iterations=100):
    # 初始化参数
    p1 = [params["p1"]]  # 零件1初始次品率
    p2 = [params["p2"]]  # 零件2初始次品率

    # 初始合格率计算
    q1 = [1 if x1 == 1 else (1 - p1[0])]  # 零件1合格率
    q2 = [1 if x2 == 1 else (1 - p2[0])]  # 零件2合格率
    qf = [q1[0] * q2[0] * (1 - params["pf"])]  # 成品有效合格率
    qz = [q1[0] * q2[0] * (1 - params["pf"])]  # 理论生产合格率
    k = [((1 if x1 == 0 else (1 - p1[0])) + (1 if x2 == 0 else (1 - p2[0]))) / 2]  # 装配比例

```

```
    # 初始利润计算（第0次迭代）
    revenue_0 = k[0] * qf[0] * params["s"]
    costs_0 = (params["c1"] + params["c2"] +
               x1 * params["d1"] +
               x2 * params["d2"] +
               k[0] * (params["cf"] +
                       x3 * params["df"] +
                       params["D"] * (1 - qz[0]) * x4 +
                       (1 - x3) * (1 - qf[0]) * params["L"]))
    pai_0 = revenue_0 - costs_0  # 初始利润

    # 初始化迭代参数列表
    pai = []  # 利润列表
    theta_list = []  # theta值列表
    qz_list = []  # 理论合格率列表
    qf_list = []  # 有效合格率列表
    p1_list = []  # 零件1次品率列表
    p2_list = []  # 零件2次品率列表
    k_prev = k[0]
    for n in range(1, iterations + 1):
        # 更新零件次品率（若不检测则次品率上升）
        p1_n = p1[-1] / (1 - (1 - p1[-1])**2)
        p2_n = p2[-1] / (1 - (1 - p2[-1])**2)
        p1.append(p1_n)
        p2.append(p2_n)
        p1_list.append(p1_n)
        p2_list.append(p2_n)

        # 更新合格率
        q1_n = 1 if x1 == 1 else (1 - p1_n)  # 零件1合格率
        q2_n = 1 if x2 == 1 else (1 - p2_n)  # 零件2合格率
        qf_n = q1_n * q2_n * (1 - params["pf"])  # 有效合格率（受检测影响）
        qz_n = q1_n * q2_n * (1 - params["pf"])  # 理论合格率（不受检测影响）
```

```
        k_n = ((1 if x1 == 0 else (1 - p1_n)) + (1 if x2 == 0 else (1 - p2_n))) / 2  # 装配比例
         # 计算theta值（与拆解决策相关）

        # 保存当前迭代的合格率
        qz_list.append(qz_n)


        # 累积theta乘积
        theta_n = x4 * (1 - qf[-1]) * k_prev
        theta_list.append(theta_n)
        k_prev = k_n
        qf_list.append(qf_n)
        # 计算theta乘积的积
        multiply_theta = math.prod(theta_list)
        # 计算当前迭代的利润
        revenue_n = qf_n * k_n * params["s"]
        cost_n = (x1 * params["d1"] +
                  x2 * params["d2"] +
                  k_n * (params["cf"] +
                         x3 * params["df"] +
                         params["D"] * (1 - qz_n) * x4 +
                         (1 - x3) * (1 - qf_n) * params["L"]))
        pai_n = multiply_theta * (revenue_n - cost_n)
        pai.append(pai_n)

    total_profit = sum(pai) + pai_0  # 总利润（包括初始利润）
    # 返回所有计算结果
    return (total_profit, theta_list, pai,
            qz_list, qf_list, p1_list, p2_list,
            pai_0, qf[0], qz[0])

# 汇总结果存储
summary_results = []
detailed_records = []
```

```python
best_details = []  # 存储最优策略的详细信息

for sc in scenarios:
    case_id = sc["case"]
    max_profit = -float('inf')
    best_decision = None
    # 初始化最优策略的详细参数
    best_theta = None
    best_pai = None
    best_qz = None
    best_qf = None
    best_p1 = None
    best_p2 = None
    best_pai0 = None
    best_qf0 = None
    best_qz0 = None

    # 遍历所有可能的决策组合（x1, x2, x3, x4均为0或1）
    for x1, x2, x3, x4 in itertools.product([0, 1], repeat=4):
        (profit, theta_list, pai_list,
         qz_list, qf_list, p1_list, p2_list,
         pai_0, qf_0, qz_0) = calculate_profit(x1, x2, x3, x4,
    sc)

        # 记录详细策略结果
        detailed_records.append({
            "情况": case_id,
            "x1_零件1检测": x1,
            "x2_零件2检测": x2,
            "x3_成品检测": x3,
            "x4_拆解": x4,
            "单位利润": round(profit, 4)
        })

        # 更新最大利润组合
```

```
        if profit > max_profit:
            max_profit = profit
            best_decision = (x1, x2, x3, x4)
            best_theta = theta_list
            best_pai = pai_list
            best_qz = qz_list
            best_qf = qf_list
            best_p1 = p1_list
            best_p2 = p2_list
            best_pai0 = pai_0
            best_qf0 = qf_0
            best_qz0 = qz_0

    # 保存最优策略到汇总结果
    x1, x2, x3, x4 = best_decision
    summary_results.append({
        "情况": case_id,
        "最大单位利润": round(max_profit, 4),
        "零配件1检测": "是" if x1 == 1 else "否",
        "零配件2检测": "是" if x2 == 1 else "否",
        "成品检测": "是" if x3 == 1 else "否",
        "不合格成品拆解": "是" if x4 == 1 else "否"
    })

    # 保存最优策略的详细信息
    best_details.append({
        "case": case_id,
        "decision": best_decision,
        "total_profit": max_profit,
        "theta_list": best_theta,
        "pai_list": best_pai,
        "qz_list": best_qz,
        "qf_list": best_qf,  # 有效合格率列表
        "p1_list": best_p1,
        "p2_list": best_p2,
```

```
        "pai_0": best_pai0,
        "qf_0": best_qf0,  # 初始有效合格率
        "qz_0": best_qz0
    })

# 输出最优策略汇总表
df_summary = pd.DataFrame(summary_results)
print("\n【最优策略汇总】")
print(tabulate(df_summary, headers="keys", tablefmt="grid",
   showindex=False))

# 输出每种情形最优策略的qf（有效合格率）信息
for detail in best_details:
    case_id = detail["case"]
    decision = detail["decision"]
    qf_0 = detail["qf_0"]  # 初始有效合格率
    qf_list = detail["qf_list"]  # 迭代过程中的有效合格率

    print(f"\n\n【情况 {case_id} 最优策略的有效合格率(qf)】")
    print(f"最优决策: x1={decision[0]}（零件1检测: {'是' if
   decision[0]==1 else '否'}）, "
          f"x2={decision[1]}（零件2检测: {'是' if decision
   [1]==1 else '否'}）, "
          f"x3={decision[2]}（成品检测: {'是' if decision
   [2]==1 else '否'}）, "
          f"x4={decision[3]}（拆解: {'是' if decision[3]==1
   else '否'}）")
```

exercise_3.py

```
import itertools
import pandas as pd
import numpy as np
import math
from tqdm import tqdm
from deap import base, creator, tools, algorithms
```

```python
import random

# 参数数据
data = {
    "零配件1": {"次品率": 0.1, "购买单价": 2, "检测成本": 1},
    "零配件2": {"次品率": 0.1, "购买单价": 8, "检测成本": 1},
    "零配件3": {"次品率": 0.1, "购买单价": 12, "检测成本": 2},
    "零配件4": {"次品率": 0.1, "购买单价": 2, "检测成本": 1},
    "零配件5": {"次品率": 0.1, "购买单价": 8, "检测成本": 1},
    "零配件6": {"次品率": 0.1, "购买单价": 12, "检测成本": 2},
    "零配件7": {"次品率": 0.1, "购买单价": 8, "检测成本": 1},
    "零配件8": {"次品率": 0.1, "购买单价": 12, "检测成本": 2},
    "半成品1": {"次品率": 0.1, "装配费用": 8, "检查成本": 4, "拆解费用": 6},
    "半成品2": {"次品率": 0.1, "装配费用": 8, "检查成本": 4, "拆解费用": 6},
    "半成品3": {"次品率": 0.1, "装配费用": 8, "检查成本": 4, "拆解费用": 6},
    "成品": {"次品率": 0.1, "装配费用": 8, "检查成本": 6, "拆解费用": 10, "市场售价": 200, "调换损失": 30}
}


def calculate_profit(x, data, iterations=1000):
    x11,x12,x13,x14,x15,x16,x17,x18,x21,x22,x23,x3,z1,z21,z22,z23 = x
    ls1 = [x11,x12,x13,x14,x15,x16,x17,x18]
    ls2 = [x21,x22,x23]

    # 基础参数初始化
    p11 =[data["零配件1"]["次品率"]]
    p12 =[data["零配件2"]["次品率"]]
    p13 =[data["零配件3"]["次品率"]]
    p14 =[data["零配件4"]["次品率"]]
    p15 =[data["零配件5"]["次品率"]]
```

```
    p16 =[data["零配件6"]["次品率"]]
    p17 =[data["零配件7"]["次品率"]]
    p18 =[data["零配件8"]["次品率"]]
    p21 =[data["半成品1"]["次品率"]]
    p22 =[data["半成品2"]["次品率"]]
    p23 =[data["半成品3"]["次品率"]]
    components_p11 =[data["零配件1"]["次品率"]]
    components_p12 =[data["零配件2"]["次品率"]]
    components_p13 =[data["零配件3"]["次品率"]]
    components_p14 =[data["零配件4"]["次品率"]]
    components_p15 =[data["零配件5"]["次品率"]]
    components_p16 =[data["零配件6"]["次品率"]]
    components_p17 =[data["零配件7"]["次品率"]]
    components_p18 =[data["零配件8"]["次品率"]]
    pf = data["成品"]["次品率"]

    # 合格率计算
    q11=[1 - p11[0] if ls1[0]==0 else 1]
    q12=[1 - p12[0] if ls1[1]==0 else 1]
    q13=[1 - p13[0] if ls1[2]==0 else 1]
    q14=[1 - p14[0] if ls1[3]==0 else 1]
    q15=[1 - p15[0] if ls1[4]==0 else 1]
    q16=[1 - p16[0] if ls1[5]==0 else 1]
    q17=[1 - p17[0] if ls1[6]==0 else 1]
    q18=[1 - p18[0] if ls1[7]==0 else 1]
    q21 = [q11[0] * q12[0] * q13[0] * (1 - p21[0] if ls2[0]==0
    else 1)]
    q22 = [q14[0] * q15[0] * q16[0] * (1 - p22[0] if ls2[1]==0
    else 1)]
    q23 = [q17[0] * q18[0] * (1 - p23[0] if ls2[2]==0 else 1)]
    qf_initial = q21[0] * q22[0] * q23[0] * (1 - pf)
    qf = [qf_initial]  # 存储所有迭代的qf值，初始值为第0次

    # 装配比例计算
    k11=[((1 - p11[0] if ls1[0]==1 else 1)+(1 - p12[0] if ls1
```

```
[1]==1 else 1)+(1 - p13[0] if ls1[2]==1 else 1))/3]
 k12=[((1 - p14[0] if ls1[3]==1 else 1)+(1 - p15[0] if ls1
[4]==1 else 1)+(1 - p16[0] if ls1[5]==1 else 1))/3]
 k13=[((1 - p17[0] if ls1[6]==1 else 1)+(1 - p18[0] if ls1
[7]==1 else 1))/2]
 k2=[(k11[0]*q11[0] * q12[0] * q13[0] * (1 - p21[0] if ls2
[0]==1 else 1)+
      k12[0]*q14[0] * q15[0] * q16[0] * (1 - p22[0] if ls2
[1]==1 else 1)+
      k13[0]*q17[0] * q18[0] * (1 - p23[0] if ls2[2]==1
else 1))/3]

 # 初始利润构成（第0次迭代）
 revenue_0 = qf[0] * data["成品"]["市场售价"]  # 初始收入
 S1 = sum([data[f'零配件{i}']['购买单价'] + x * data[f'零配
件{i}']['检测成本']
          for i, x in zip(range(1, 9), ls1)])  # 零配件总成
本
 S2 = sum([k*data[f'半成品{i}']['装配费用'] + k*x * data[f'
半成品{i}']['检查成本']
          for i, x, k in zip(range(1, 4), ls2,[k11[0],k12
[0],k13[0]])])  # 半成品总成本
 S3 = k2[0]*(data['成品']['装配费用'] + x3 * data['成品']['
检查成本'])  # 成品总成本
 A = (k2[0] * z1 * (1-qf[0]) * data['成品']['拆解费用'] +
      sum([z * (1 - q2) * data[f'半成品{idx+1}']['拆解费用'
]
          for idx, (z, q2) in enumerate(zip([z21, z22, z23
], [q21[0], q22[0], q23[0]]))]))  # 拆解总成本
 re = k2[0]*(1 - x3) * (1 - qf[0]) * data['成品']['调换损失
']  # 调换损失成本
 cost_0 = S1 + S2 + S3 + A + re  # 初始总成本
 pai_0 = revenue_0 - cost_0  # 初始利润

 # 迭代利润累积
```

```
    half_pai = []  # 半成品相关利润（保留迭代过程）
    components_pai = []  # 零配件相关利润
    theta_list = []
    beta_list = []
    first_iteration_details = None  # 存储第一次迭代的详细信息
    theta_first_breakdown = None  # 存储theta第一项的构成
    k2_prev = k2[0]  # 存储上一次迭代的k2值

    for n in range(1, iterations + 1):
        # 更新次品率与合格率
        p21_n = p21[-1]  / (1 - (1 - p21[-1])**3)
        p22_n = p22[-1] / (1 - (1 - p22[-1])**3)
        p23_n = p23[-1] / (1 - (1 - p23[-1])**3)
        p21.append(p21_n)
        p22.append(p22_n)
        p23.append(p23_n)

        q21_n = q11[0]*q12[0]*q13[0] *(1 - p21_n if ls2[0]==0
else 1)
        q22_n = q14[0]*q15[0]*q16[0] *(1 - p22_n if ls2[1]==0
else 1)
        q23_n = q17[0]*q18[0] *(1 - p23_n if ls2[2]==0 else 1)
        qf_n = q21_n * q22_n * q23_n * (1 - pf)


        k2_n = (q11[0]*q12[0]*q13[0] *(1 - p21_n if ls2[0]==1
else 1) +
                q14[0]*q15[0]*q16[0]*(1 - p22_n if ls2[1]==0
else 1) +
                q17[0]*q18[0]*(1 - p23_n if ls2[2]==0 else 1))
 / 3

        # 更新零配件次品率
        components_p11_n = components_p11[-1]  / (1 - (1 -
components_p11[-1])**3)
```

```
120         components_p12_n = components_p12[-1]  / (1 - (1 -
    components_p12[-1])**3)
121         components_p13_n = components_p13[-1]  / (1 - (1 -
    components_p13[-1])**3)
122         components_p14_n = components_p14[-1]  / (1 - (1 -
    components_p14[-1])**3)
123         components_p15_n = components_p15[-1]  / (1 - (1 -
    components_p15[-1])**3)
124         components_p16_n = components_p16[-1]  / (1 - (1 -
    components_p16[-1])**3)
125         components_p17_n = components_p17[-1] / (1 - (1 -
    components_p17[-1])**2)
126         components_p18_n = components_p18[-1] / (1 - (1 -
    components_p18[-1])**2)
127         components_p11.append(components_p11_n)
128         components_p12.append(components_p12_n)
129         components_p13.append(components_p13_n)
130         components_p14.append(components_p14_n)
131         components_p15.append(components_p15_n)
132         components_p16.append(components_p16_n)
133         components_p17.append(components_p17_n)
134         components_p18.append(components_p18_n)
135
136         # 计算组件合格率
137         components_q11_n = 1 - components_p11_n if ls1[0]==0
    else 1
138         components_q12_n = 1 - components_p12_n if ls1[1]==0
    else 1
139         components_q13_n = 1 - components_p13_n if ls1[2]==0
    else 1
140         components_q14_n = 1 - components_p14_n if ls1[3]==0
    else 1
141         components_q15_n = 1 - components_p15_n if ls1[4]==0
    else 1
142         components_q16_n = 1 - components_p16_n if ls1[5]==0
```

```
    else 1
143         components_q17_n = 1 - components_p17_n if ls1[6]==0
    else 1
144         components_q18_n = 1 - components_p18_n if ls1[7]==0
    else 1
145
146         # 计算半成品组件合格率
147         components_q21_n = components_q11_n * components_q12_n
     * components_q13_n * (1 - p21_n if ls2[0]==1 else 1)
148         components_q22_n = components_q14_n * components_q15_n
     * components_q16_n * (1 - p22_n if ls2[1]==1 else 1)
149         components_q23_n = components_q17_n * components_q18_n
     * (1 - p23_n if ls2[2]==1 else 1)
150
151         # 计算迭代中的利润组件
152         components_qf_n = components_q21_n * components_q22_n
    * components_q23_n * (1 - pf )
153
154         components_k2_n = (
155             ((1 - p11[-1] if ls1[0]==1 else 1)+(1 - p12[-1] if
     ls1[1]==1 else 1)+(1 - p13[-1] if ls1[2]==1 else 1))/3 *
156             components_q11_n * components_q12_n *
    components_q13_n * (1 - p21_n if ls2[0]==1 else 1) +
157             ((1 - p14[-1] if ls1[3]==1 else 1)+(1 - p15[-1] if
     ls1[4]==1 else 1)+(1 - p16[-1] if ls1[5]==1 else 1))/3 *
158             components_q14_n * components_q15_n *
    components_q16_n * (1 - p22_n if ls2[1]==1 else 1) +
159             ((1 - p17[-1] if ls1[6]==1 else 1)+(1 - p18[-1] if
     ls1[7]==1 else 1))/2 *
160             components_q17_n * components_q18_n * (1 - p23_n
    if ls2[2]==1 else 1)
161         ) / 3
162
163         # 累积参数与利润
164         beta_n = (3*z21*(1 - q21_n) + 3*z22*(1 - q22_n) + 2*
```

```
    z23*(1 - q23_n))/8
        theta_n = z1 * (1 - qf[-1])*k2_prev + z1*(1 - components_qf_n)*components_k2_n * beta_n
        qf.append(qf_n)  # 保存当前迭代的qf值

        k2_prev = k2_n
        theta_list.append(theta_n)
        beta_list.append(beta_n)

        multiply_theta = math.prod(theta_list)
        multiply_beta = math.prod(beta_list)

        # 半成品相关利润（每次迭代结果存入half_pai）
        half_revenue_n = k2_n * qf_n * data["成品"]["市场售价"]
        half_cost_n = (sum([x * data[f'半成品{i}']['检查成本'] for i, x in zip(range(1, 4), ls2)]) +
                       z21*(1 - q21_n)*data['半成品1']['拆解费用'] +
                       z22*(1 - q22_n)*data['半成品2']['拆解费用'] +
                       z23*(1 - q23_n)*data['半成品3']['拆解费用'] +
                       k2_n*(data['成品']['装配费用'] + x3 * data['成品']['检查成本'] +
                             (1 - qf_n) * z1 * data['成品']['拆解费用'] +
                             (1 - x3) * (1 - qf_n) * data['成品']['调换损失']))
        current_half_pai = multiply_theta * (half_revenue_n - half_cost_n)
        half_pai.append(current_half_pai)

        # 零配件相关利润
        components_revnue_n = components_k2_n *
```

```
    components_qf_n * data["成品"]["市场售价"]
        components_S1_n = sum([x * data[f'零配件{i}']['检测成本'] for i, x in zip(range(1, 9), ls1)])
        components_S2_n = sum([k * data[f'半成品{i}']['装配费用'] + k * x * data[f'半成品{i}']['检查成本']
                               for i, x, k in zip(range(1, 4), ls2, [k11[0], k12[0], k13[0]])])
        components_S3_n = components_k2_n * (data['成品']['装配费用'] + x3 * data['成品']['检查成本'])
        components_A_n = (components_k2_n * z1 * (1 - components_qf_n) * data['成品']['拆解费用'] +
                          sum([z * (1 - components_q21_n if idx==0 else components_q22_n if idx==1 else components_q23_n) *
                               data[f'半成品{idx+1}']['拆解费用']
                               for idx, z in enumerate([z21, z22, z23])]))
        components_re = components_k2_n * (1 - x3) * (1 - components_qf_n) * 30
        components_cost_n = components_S1_n + components_S2_n + components_S3_n + components_A_n + components_re
        components_pai.append(multiply_theta * multiply_beta * (components_revnue_n - components_cost_n))

    # 总利润及构成汇总
    total_iteration_profit = sum(half_pai) + sum(components_pai)  # 迭代累积利润
    total_profit = total_iteration_profit + pai_0  # 总利润

    # 返回利润构成及相关详细信息
    return {
        "总利润": total_profit,
    }


```

```
211 def calculate_qf(ls1, ls2, data):
212     """计算成品合格率"""
213     q11 = 1 - data["零配件1"]["次品率"] if ls1[0] == 0 else 1
214     q12 = 1 - data["零配件2"]["次品率"] if ls1[1] == 0 else 1
215     q13 = 1 - data["零配件3"]["次品率"] if ls1[2] == 0 else 1
216     q14 = 1 - data["零配件4"]["次品率"] if ls1[3] == 0 else 1
217     q15 = 1 - data["零配件5"]["次品率"] if ls1[4] == 0 else 1
218     q16 = 1 - data["零配件6"]["次品率"] if ls1[5] == 0 else 1
219     q17 = 1 - data["零配件7"]["次品率"] if ls1[6] == 0 else 1
220     q18 = 1 - data["零配件8"]["次品率"] if ls1[7] == 0 else 1
221     q21 = q11 * q12 * q13 * (1 - data["半成品1"]["次品率"] if
    ls2[0] == 0 else 1)
222     q22 = q14 * q15 * q16 * (1 - data["半成品2"]["次品率"] if
    ls2[1] == 0 else 1)
223     q23 = q17 * q18 * (1 - data["半成品3"]["次品率"] if ls2[2]
     == 0 else 1)
224     return q21 * q22 * q23 * (1 - data["成品"]["次品率"])
225
226
227 # 设置遗传算法参数
228 POP_SIZE = 100      # 种群大小
229 N_GEN = 10          # 迭代代数
230 CX_PB = 0.5         # 交叉概率
231 MUT_PB = 0.2        # 变异概率
232
233 # 创建适应度和个体类
234 creator.create("FitnessMax", base.Fitness, weights=(1.0,))  #
    最大化问题
235 creator.create("Individual", list, fitness=creator.FitnessMax)
236
237 toolbox = base.Toolbox()
238 toolbox.register("attr_bool", random.randint, 0, 1)
239 toolbox.register("individual", tools.initRepeat, creator.
    Individual, toolbox.attr_bool, n=16)
240 toolbox.register("population", tools.initRepeat, list, toolbox
```

```
    .individual)

# 评估函数（带约束判断）
def eval_profit(ind):
    x11,x12,x13,x14,x15,x16,x17,x18,x21,x22,x23,x3,z1,z21,z22,z23 = ind
    if z21 > x21 or z22 > x22 or z23 > x23:
        return -1e6,  # 非可行解惩罚
    result = calculate_profit(ind, data)
    return result["总利润"],

toolbox.register("evaluate", eval_profit)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 初始化种群
population = toolbox.population(n=POP_SIZE)

# 进化过程
print("开始遗传算法优化...")
for gen in tqdm(range(N_GEN), desc="进化中", ncols=80):
    offspring = algorithms.varAnd(population, toolbox, cxpb=CX_PB, mutpb=MUT_PB)
    fits = list(map(toolbox.evaluate, offspring))
    for ind, fit in zip(offspring, fits):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))
print("优化完成。")

# 选出最优个体
best_ind = tools.selBest(population, k=1)[0]
best_profit_result = calculate_profit(best_ind, data)

max_profit = best_profit_result["总利润"]
```

```
best_decision = best_ind
profit_components = best_profit_result

# 输出最佳决策
labels = ["零配件1","零配件2","零配件3","零配件4","零配件5","零配件6","零配件7","零配件8",
          "半成品1","半成品2","半成品3","成品","成品拆解","半成品1拆解","半成品2拆解","半成品3拆解"]
choices = ["是" if i else "否" for i in best_decision]
results_df = pd.DataFrame(zip(["最大单位利润（总）"] + labels,
    [round(max_profit, 4)] + choices),
                          columns=["决策项", "取值"])
print("\n最佳决策方案: ")
print(results_df.to_string(index=False))
```