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

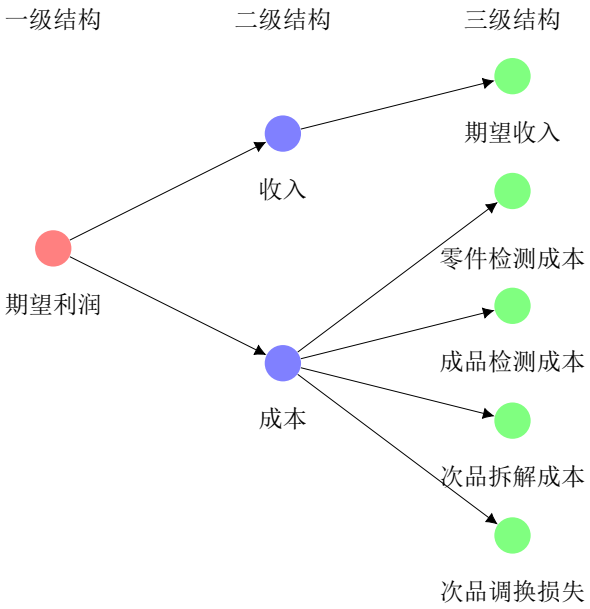

图 1　利润结构图

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

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

六、 问题二的模型的建立和求解

七、 问题三的模型的建立和求解

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。、

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)
```

```
        components_p12_n = components_p12[-1]  / (1 - (1 -
components_p12[-1])**3)
        components_p13_n = components_p13[-1]  / (1 - (1 -
components_p13[-1])**3)
        components_p14_n = components_p14[-1]  / (1 - (1 -
components_p14[-1])**3)
        components_p15_n = components_p15[-1]  / (1 - (1 -
components_p15[-1])**3)
        components_p16_n = components_p16[-1]  / (1 - (1 -
components_p16[-1])**3)
        components_p17_n = components_p17[-1] / (1 - (1 -
components_p17[-1])**2)
        components_p18_n = components_p18[-1] / (1 - (1 -
components_p18[-1])**2)
        components_p11.append(components_p11_n)
        components_p12.append(components_p12_n)
        components_p13.append(components_p13_n)
        components_p14.append(components_p14_n)
        components_p15.append(components_p15_n)
        components_p16.append(components_p16_n)
        components_p17.append(components_p17_n)
        components_p18.append(components_p18_n)

        # 计算组件合格率
        components_q11_n = 1 - components_p11_n if ls1[0]==0
else 1
        components_q12_n = 1 - components_p12_n if ls1[1]==0
else 1
        components_q13_n = 1 - components_p13_n if ls1[2]==0
else 1
        components_q14_n = 1 - components_p14_n if ls1[3]==0
else 1
        components_q15_n = 1 - components_p15_n if ls1[4]==0
else 1
        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
165         theta_n = z1 * (1 - qf[-1])*k2_prev + z1*(1 -
    components_qf_n)*components_k2_n * beta_n
166         qf.append(qf_n)  # 保存当前迭代的qf值
167
168         k2_prev = k2_n
169         theta_list.append(theta_n)
170         beta_list.append(beta_n)
171
172         multiply_theta = math.prod(theta_list)
173         multiply_beta = math.prod(beta_list)
174
175         # 半成品相关利润（每次迭代结果存入half_pai）
176         half_revenue_n = k2_n * qf_n * data["成品"]["市场售价"
    ]
177         half_cost_n = (sum([x * data[f'半成品{i}']['检查成本']
     for i, x in zip(range(1, 4), ls2)]) +
178                        z21*(1 - q21_n)*data['半成品1']['拆解费
    用'] +
179                        z22*(1 - q22_n)*data['半成品2']['拆解费
    用'] +
180                        z23*(1 - q23_n)*data['半成品3']['拆解费
    用'] +
181                        k2_n*(data['成品']['装配费用'] + x3 *
    data['成品']['检查成本'] +
182                              (1 - qf_n) * z1 * data['成品']['拆
    解费用'] +
183                              (1 - x3) * (1 - qf_n) * data['成品
    ']['调换损失']))
184         current_half_pai = multiply_theta * (half_revenue_n -
    half_cost_n)
185         half_pai.append(current_half_pai)
186
187         # 零配件相关利润
188         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,
}
```

```python
def calculate_qf(ls1, ls2, data):
    """计算成品合格率"""
    q11 = 1 - data["零配件1"]["次品率"] if ls1[0] == 0 else 1
    q12 = 1 - data["零配件2"]["次品率"] if ls1[1] == 0 else 1
    q13 = 1 - data["零配件3"]["次品率"] if ls1[2] == 0 else 1
    q14 = 1 - data["零配件4"]["次品率"] if ls1[3] == 0 else 1
    q15 = 1 - data["零配件5"]["次品率"] if ls1[4] == 0 else 1
    q16 = 1 - data["零配件6"]["次品率"] if ls1[5] == 0 else 1
    q17 = 1 - data["零配件7"]["次品率"] if ls1[6] == 0 else 1
    q18 = 1 - data["零配件8"]["次品率"] if ls1[7] == 0 else 1
    q21 = q11 * q12 * q13 * (1 - data["半成品1"]["次品率"] if ls2[0] == 0 else 1)
    q22 = q14 * q15 * q16 * (1 - data["半成品2"]["次品率"] if ls2[1] == 0 else 1)
    q23 = q17 * q18 * (1 - data["半成品3"]["次品率"] if ls2[2] == 0 else 1)
    return q21 * q22 * q23 * (1 - data["成品"]["次品率"])


# 设置遗传算法参数
POP_SIZE = 100      # 种群大小
N_GEN = 10          # 迭代代数
CX_PB = 0.5         # 交叉概率
MUT_PB = 0.2        # 变异概率

# 创建适应度和个体类
creator.create("FitnessMax", base.Fitness, weights=(1.0,))  # 最大化问题
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=16)
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))
```