基于遗传算法和 NSGA-II 的企业生产决策优化研究

摘要

对于商家而言，从产品生产到回收的整个过程中，会面临一系列决策。这些决策的正确与否直接影响到收益的增减。鉴于此，本文利用遗传算法和 NSGA-II 方法，为企业提供了一套优化决策的方案模板，以帮助提升收益。

针对问题一，零件的次品数服从二项分布。利用**中心极限定理**可知，在大样本条件下，该分布近可视为正态分布。并且在题目给出的置信水平下构建次品率的检验问题，并基于检验统计量的目标函数推导出抽样次数 n 的表达式。通过**蒙特卡罗方法**计算，在 95.0% 和 90.0% 的置信水平下，所需的抽样次数分别为 139 和 98。

针对问题二，我们以检测成本、装配成本、成品检验成本、拆解成本和调换成本构成的总成本为自变量，以利润为因变量构建了线性回归模型。通过构建 6 个回归模型，并利用**遗传算法**求解，分别得到了 6 种情况下的最大利润值，分别为 5480、5860、5240、5820、5400 和 5490。在选择相同的决策方案（即不检测零配件 1、不检测零配件 2、不检测成品，且对不合格成品进行拆解）时，次品率较高的情境下能够获得更高的收益。各项指标中，虽然调换损失有所不同，但收益相近，这表明在一定程度上可以忽略调换损失的影响。

针对问题三，相较于问题二，问题三中涉及到八种零部件和三道工序，导致决策数量显著增加，使得在问题二中使用的遗传算法难以有效解决。为此，本文引入了 NSGA-II 算法，以求解最佳决策方案。通过该算法，得出的策略为：不检测零配件 1，检测零配件 2，不检测零配件 3、4、5、6、7、8，不检测半成品 1、2、3，不检测成品，并且不拆解不合格成品。在此策略下，获得的最大利润为 19872 元。所建立的模型具有广泛的适用性，可以扩展应用于包含 N 个零配件和 M 道工序的实际问题中。

针对问题四，首先，根据第一问中的不等式关系确定检测的个数，然后利用**蒙特卡罗**方法对次品率进行求解。接着，将求得的次品率代入问题二和问题三中所建立的算法模型，进行决策选择并计算最大利润。最后得到的决策和第二问和第三问中决策一致。

从长远的公司发展角度来看，虽然在某些情况下，次品率较高时如果选择不检测可能会带来较高的利润，但这种策略并不可持续。因此，企业在制定决策时，应该综合考虑成本、质量、市场和可持续性等多方面因素，以实现长期的利润增长和企业的健康发展。

关键词：蒙特卡罗法 遗传算法 NSGA-II

一、问题重述

1.1 问题背景

在企业生产过程中，如何科学地针对零配件和成品的次品率进行决策，是企业必须面对的重要问题。正确的决策不仅可以降低成本，还能提高企业的运行效率。本文对此问题进行了深入分析，为企业提供了一种解决此类问题的参考模板。在抽样检测过程中，准确确定次品率并明确拒绝域，合理设定抽检数量，可以有效减少成本，提高企业效益。

1.2 问题的提出

在题目中给出了供第一问使用的次品率和两种情况下的置信区间，给出了供第二问第三问使用的各零件的次品率，购买单价，检测成本等指标。

问题一：提供了次品率和两种置信区间。根据这些信息和分布规律，推导出一个包含抽样数量 n 的表达式，并通过计算确定最优的 n 值。

问题二：基于表一中提供的六种情况，给出了不同的次品率、购买单价、检测成本等指标。根据这些信息，制定出对应的决策方案，并计算出每种方案下的最小成本。

问题三：涉及 8 个零配件和 3 个半成品。题目提供了次品率、购买单价、检测成本、装配成本、市场售价、调换损失和拆解费用等指标。根据这些指标，计算各个零部件的关键参数，得出最优决策，并最终将问题推广至含有 n 个零配件和 m 道工序的复杂生产环境。

问题四：需要设计一个抽样检测模型，用以计算次品率，并将计算得到的次品率代入问题二和问题三中，重新求解利润，以确保决策的优化和利润的最大化。

二、问题分析

2.1 问题一的分析

问题一要求确定在给定两种情况和两种置信区间下的抽样数量。基于正态分布的规律，可以推导出一个包含抽样数量 n 的表达式。然而，由于该等式较为复杂，无法直接求解，因此本文引入了蒙特卡罗方法来估算 n 的值。最终，通过计算得到了在两种置信区间下的最优抽样数量。

2.2 问题二的分析

问题二要求在六种不同的次品率、购买单价、检测成本、装配成本、市场售价、调换损失和拆解费用等条件下，制定出应对四种不同情境的最优决策方案，并以最小成本作为指标。由于该问题具有极强的可扩展性，并且每种决策会影响

后续的决策，因此本文引入了遗传算法来进行求解。

2.3 问题三的分析

问题三提供了与问题二类似的次品率、购买单价、检测成本、装配成本、市场售价、调换损失和拆解费用。然而，问题三涉及 8 个零配件和 3 个半成品，且对应的策略数量众多，传统的遗传算法难以处理如此复杂的决策空间。因此，本文引入了 NSGA-II 算法进行决策规划，并通过总成本度量模型进行评估。本文先解决 8 个零配件和 3 个半成品的情境，再将模型推广至包含 n 个零配件和 m 道工序的复杂生产问题。

2.2 问题四的分析

在问题一中，我们已经推导出检测数量和次品率之间的不等关系式。为了重新设计抽样检测模型，首先需要假定一个检测数量的值。随后，利用问题一中的蒙特卡罗方法来求解次品率。在得到次品率后，将其代入问题二和问题三的模型中，选择最优策略并计算最大利润值。

三、模型假设

1. 问题一中供检测的零件数量 n 足够大。
2. 假设问题二和问题三中成品的销量是一百。
3. 没有外部元素对实验结果产生影响如设备故障或市场需求波动影响。
4. 次品率是稳定的不会受到因时间波动而产生的影响
5. 假设第二文中零件一和零件二的数量为 100
6.

四、符号说明

表 1　符号说明表

符号	说明
n	抽样的零配件数量
p	零配件的次品率
α	显著性水平
x	次品的数量
C_d	零件的检测成本
$C_{\text{调换}}$	成品调换的损失
p_z	最终的成品次品率
p_1	零件一的次品率

p_2	零件二的次品率
种群	初始定义解集
个体	每个解
适应度	个体在解空间的优劣

五、模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 问题一的流程图

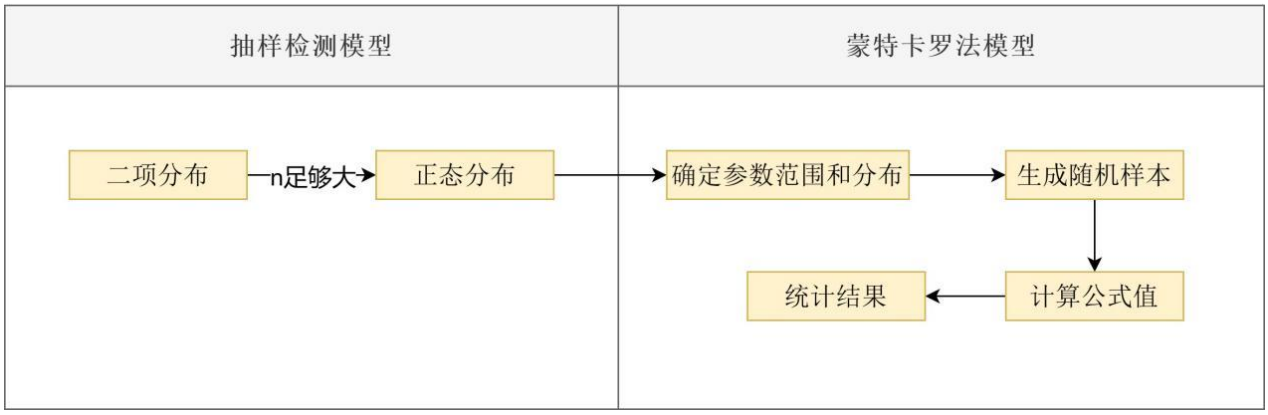


图 1 问题一流程图

对于第一问，由于检测的零件只有两种状态：次品或非次品，并且样本量 n 足够大，因此可以将这些零件的分布视为二项分布。然而，根据中心极限定理，这些零件的分布函数会趋近于正态分布。设零假设 h_0 为零件次品率不超过标称值，备择假设为零件的次品率超过标称值。通过分布规律和分布函数推导出 n 的不等式，以建立抽样检测模型。接下来，利用蒙特卡罗方法对 n 的值进行求解。首先，将参数带入蒙特卡罗模型中，生成随机样本，然后将这些随机样本代入建立的抽样检测模型中，获取 n 的结果。将所有结果收集后，根据这些解集选择最优的 n 值。

5.1.2 抽样检测模型的建立

对于问题一 $p > p_0$ 来说，针对每个零件来说都存在两种情况即该零件不为次品，反之该零件为次品。这符合二项分布的定义。同时因为 n 的数量足够大，通过中心极限定理可以将零件的分布视为正态分布。同时可以提炼出本模型的两个检验假设：（1）原假设 $p \le p_0$（2）备择假设，据推导得到如下的分布规律：

$$x \sim N(np, np(1-p)) \qquad (1)$$

通过标准化后可以得到如下的公式：

$$Z=\frac{x-np_0}{\sqrt{np_0(1-\mathrm{p}_0)}} \tag{2}$$

5.1.3 蒙特卡罗法模型的建立

当 $Z>Z_{0.05}$ 时则拒收，结合样本量的计算公式可以得到如下的等式:

$$n>\frac{(Z_\alpha\sqrt{np_0(1-p_0)}+np_0)^2}{x} \tag{3}$$

为了对这个等式进行求解，我们引入蒙特卡罗法进行求解这种方法是一种通过生成合适的随机数来解决问题的方法。[1]首先在本文中我们通过题意进行参数的初始化设定 $Z_{0.05}$， p_0 和阙值的值。接着对不同 n 值进行多次随机采样，计算不等式右面的值。最后得到最小的 n 值，在 95.0% 信度下的样本大小:139，在 90.0% 信度下的样本大小:98。下面是使用蒙特卡罗方法的求解过程：

（1）确定参数范围和分布

（2）生成随机样本

（3）计算公式值

（4）统计结果

（5）重复实验并计算结果

5.2 问题二模型的建立与求解

5.2.1 问题二的流程图

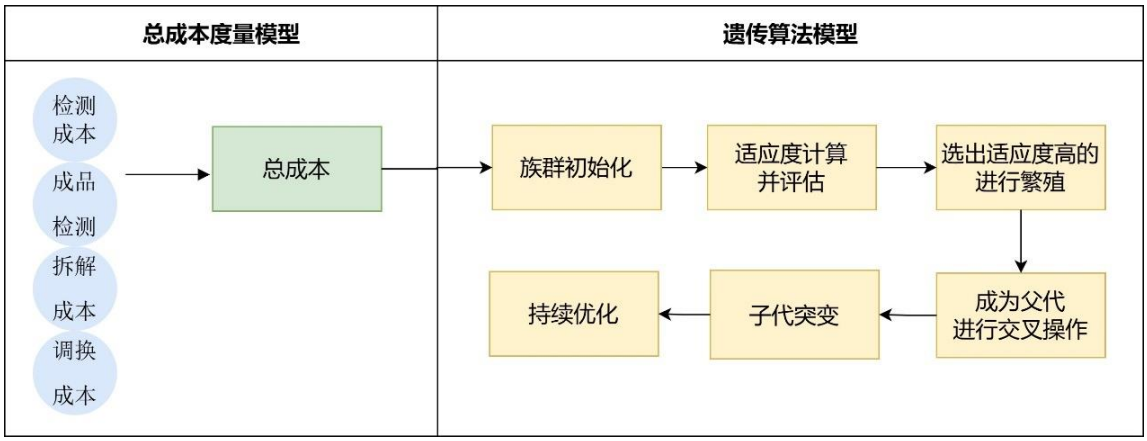


图 2 问题二的思维导图

对于第二问，首先引入检测成本、装配成本、成品检验成本、拆解成本和调换成本，构建主成本度量模型。利用题干中提供的参数，列出各项成本的计算公式，并汇总得到总成本度量模型。随后，利用遗传算法进行决策，并以总利润作为评价指标。

5.2.2 总成本度量模型的建立

在建立总成本度量函数模型的过程中通常我们需要考量检测成本，装配成本，成品检验成本，拆解成本和调换成本，据此本文建立了如下总成本度量函数模型：

检测成本：

$$C_{零件检测} = x_1 \bullet C_{d1} \bullet n_1 + x_2 \bullet C_{d2} \bullet n_2 \tag{4}$$

其中 x_1 为是否检验零配件一（1 表示检测，0 表示不检测），x_2 为是否检验零配件二（1 表示检测，0 表示不检测），n_1 和 n_2 分别表示采购的零配件 1 和 2 的数量，C_{d1} 和 C_{d2} 分别表示检测成本。

成品检验成本：

$$C_{成品检测} = x_f \bullet C_d \bullet n_f \tag{5}$$

其中 x_f 为是否检验成品，C_d 为成品检验成本，n_f 为装配成品的数量。

装配成本：

$$C_{装配} = C_a \bullet n_f \tag{6}$$

其中 C_a 表示装配成本。

拆解成本：

$$C_{拆解} = x_t \bullet C_t \bullet p_f \bullet n_f \tag{7}$$

其中 p_f 为不合格成品率，n_f 为成品数量。

总销售额：

$$C_{总销售} = C_m \bullet n_f \tag{8}$$

其中 C_m 为销售单价

调换成本：

$$C_{调换成本} = (1 - x_t) \bullet C_r \bullet p_f \bullet n_f \tag{9}$$

成品次品率：

$$p_z = p_1(1 - x_1) + p_2(1 - x_2) + (1 - p_1)(1 - p_2)\, p_1 p_2 \tag{10}$$

将以上方程进行整合可得到总成本度量函数模型如下所示：

$$C_{总成本} = C_{零配件检测} + C_{装配} + C_{成品检测} + C_{拆解} + C_{调换} \tag{11}$$

总成本公式：

$$C_{总成本} = x_1 \bullet C_{d1} \bullet n_1 + x_2 \bullet C_{d2} \bullet n_2 + x_f \bullet C_d \bullet n_f + C_a \bullet n_f + x_t \bullet C_t \bullet p_f \bullet n_f + (1-x_t) \bullet C_r \bullet p_f \bullet n_f \quad 12$$

成品次品率：

$$p_z = 1-(1-[p_1(1-x_1)+p_2(1-x_2)+(1-p_1)(1-p_2)p_1p_2])^2 \qquad (13)$$

5.2.3 遗传检测算法模型的建立

在问题二中存在多个阶段的生产决策，如零件的检测方式，是否对不合格成品进行拆解。在生产过程中每个阶段的决策都可以看成是一个二元选择问题，同时在模型建立的过程中每个阶段的决策都会影响最终的结果。而遗传算法正是一种仿生优化算法，该算法可以在模拟生物在进化过程中的复杂的选择，基因交叉和变异的过程来寻找最优解，其广泛适用于组合优化问题的求解，善于应对多阶段，多变量的复杂性,具有较好的适应度函数灵活性，因此本文决定引入该算法进行求解。遗传算法主要步骤如下：

（1）对种群进行初始化

（2）对种群适应度进行计算并评估

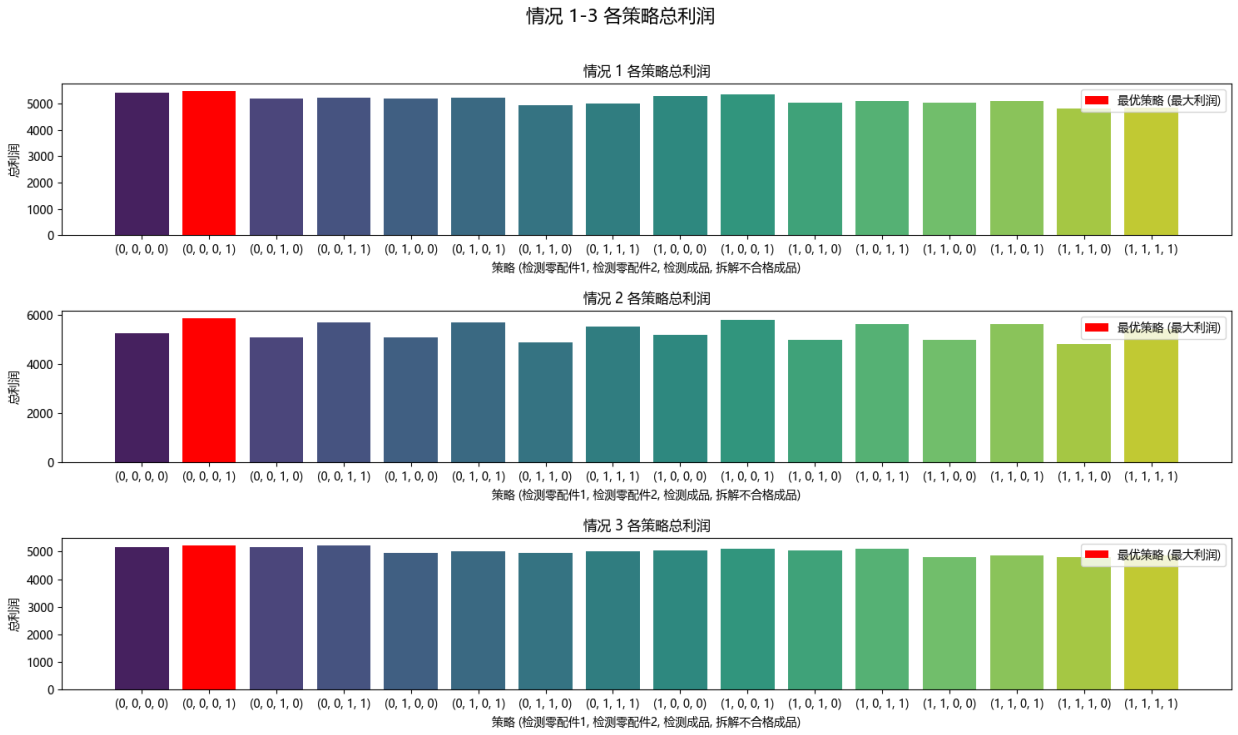
（3）选出适应度高的个体使其参与繁殖

（4）将选出的个体作为父代来产生新子代

（5）对生成的子代进行突变

（6）持续更新种群

经过遗传算法进行求解得到结果如下（图中红色的条形图为最优策略）：



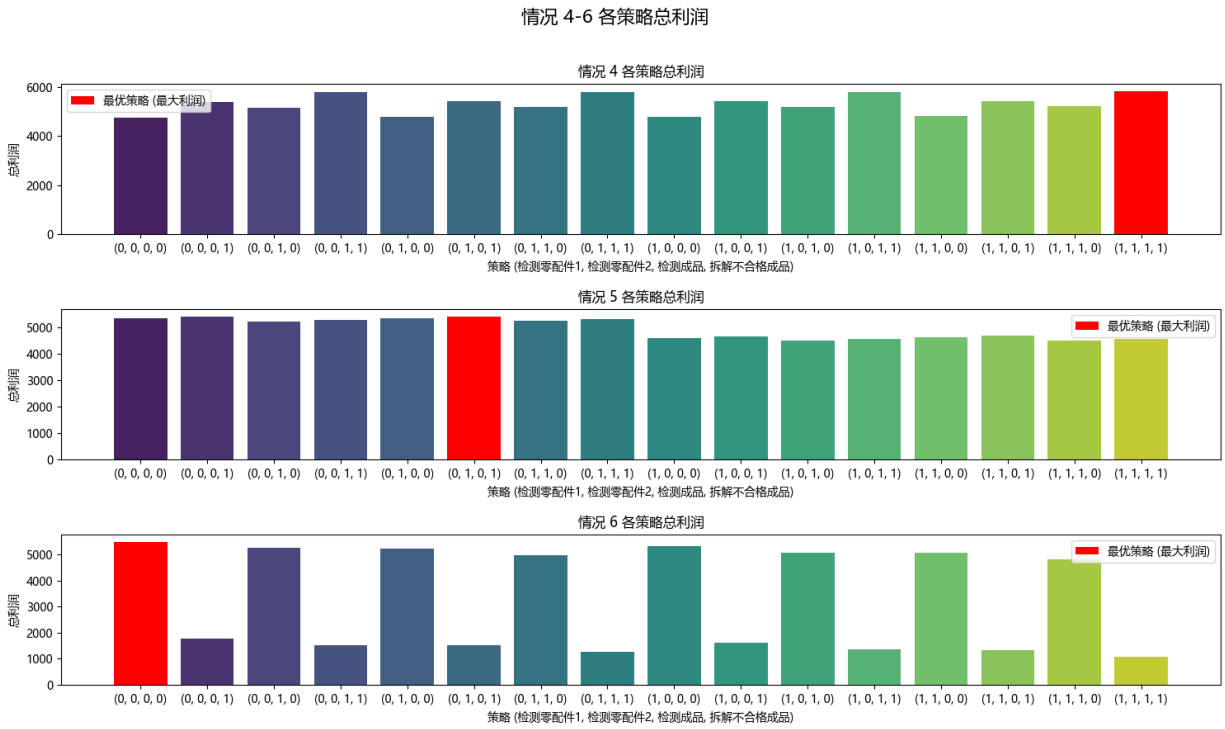


图 3 遗传算法结果

据图片可以看出对于第一种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5480 元。对于第二种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5860。对于第三种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5240。对于第四种情况来说最优方案为检测零配件 1，检测零配件 2，检测成品，拆解不合格成品，最大利润为 5820。第五种情况来说最优方案为不检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5400。第六种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，不拆解不合格成品，最大利润为 5490。

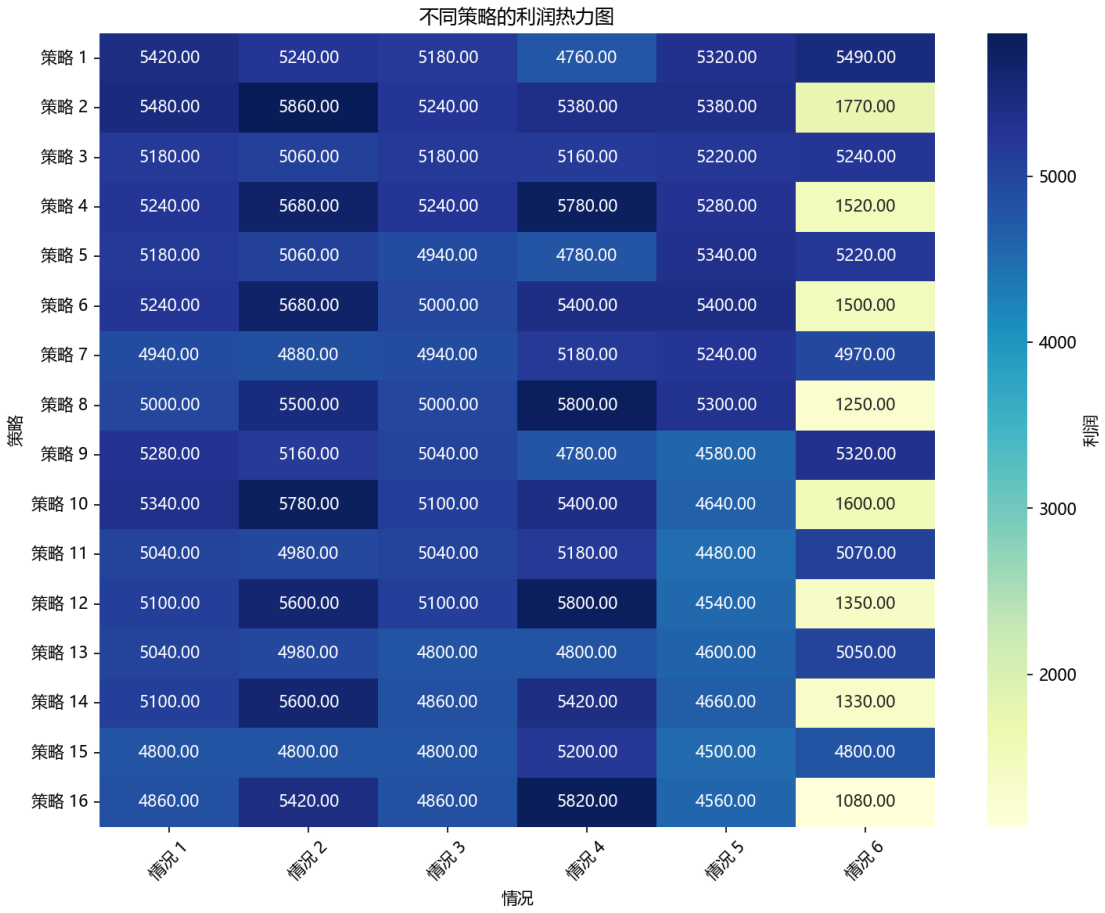


图 4 不同情况利润汇总图

通过分析结果可以看出，在不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品的策略下会获得最大的利润，这说明在企业进行生产和销售时，可以尽量在次品率可控的范围内尽可能大同时要对不合格的产品进行不合格产品的回收，降低成本。

通过方案一和方案二的对比可以看出当次品率，购买单价，检测成本，装配成本，检测成本，市场售价，调换损失，拆解费用各项指标中只有次品率不同时，在选择相同的决策方案即

不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品时次品率高的获得更高的收益。

通过方案一和方案三的对比可以看出当次品率，购买单价，检测成本，装配成本，检测成本，市场售价，调换损失，拆解费用各项指标中只有调换损失不同时但收益相似说明可以在一定程度上忽视调换损失的影响。

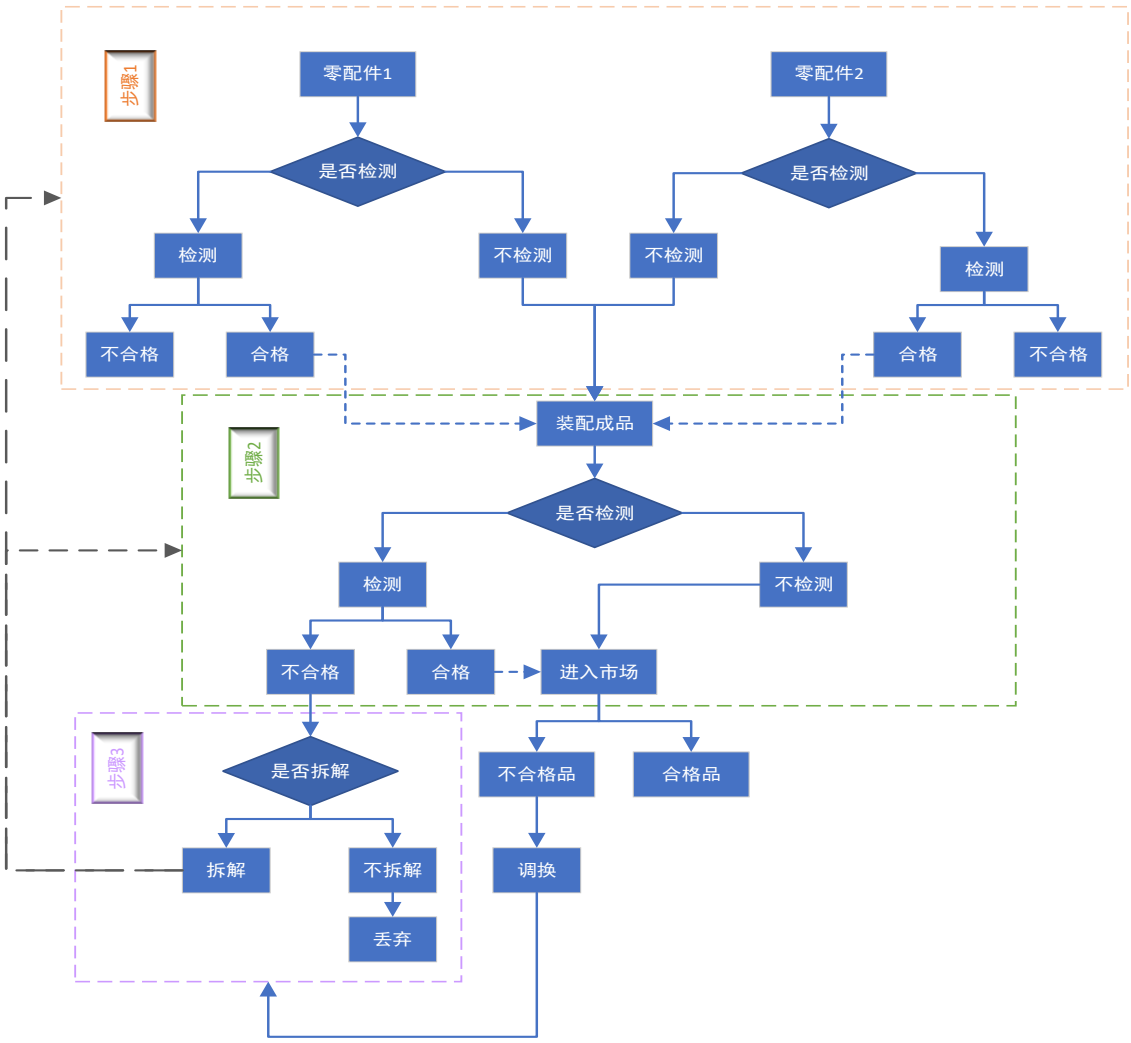


图 5 生产策略图

表 2 策略方案对照图

策略 1	检测零配件 1:否，检测零配件 2:否 检测成品:否， 拆解不合格成品:否
策略 2	检测零配件 1:否，检测零配件 2:否 检测成品:否， 拆解不合格成品:是
策略 3	检测零配件 1:否，检测零配件 2:否 检测成品:是， 拆解不合格成品:否
策略 4	检测零配件 1:否，检测零配件 2:否 检测成品:是， 拆解不合格成品:是
策略 5	检测零配件 1:否，检测零配件 2:是 检测成品:否， 拆解不合格成品:否
策略 6	检测零配件 1:否，检测零配件 2:是

	检测成品:否， 拆解不合格成品:是
策略 7	检测零配件 1:否，检测零配件 2:是 检测成品:是， 拆解不合格成品:否
策略 8	检测零配件 1:否，检测零配件 2:是 检测成品:是， 拆解不合格成品:是
策略 9	检测零配件 1:是，检测零配件 2:否 检测成品:否， 拆解不合格成品:否
策略 10	检测零配件 1:是，检测零配件 2:否 检测成品:否， 拆解不合格成品:是
策略 11	检测零配件 1:是，检测零配件 2:否 检测成品:是， 拆解不合格成品:否
策略 12	检测零配件 1:是，检测零配件 2:否 检测成品:是， 拆解不合格成品:是
策略 13	检测零配件 1:是，检测零配件 2:是 检测成品:否， 拆解不合格成品:否
策略 14	检测零配件 1:是，检测零配件 2:是 检测成品:否， 拆解不合格成品:是
策略 15	检测零配件 1:是，检测零配件 2:是 检测成品:是， 拆解不合格成品:否
策略 16	检测零配件 1:是，检测零配件 2:是 检测成品:是， 拆解不合格成品:是

5.2.4 模型的检验

在本文中为了验证模型的精度，所以在计算出总成本的基础上引入蒙特卡洛模拟法进行求解得到的图像如下：

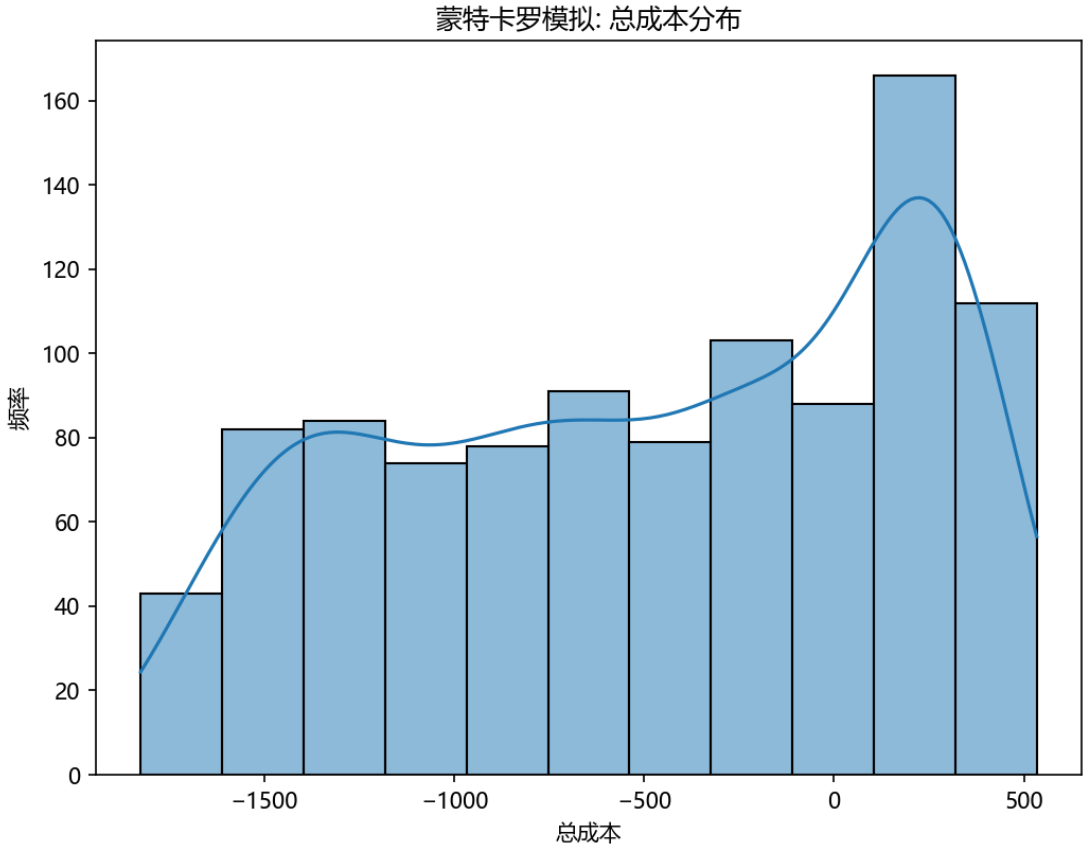


图 6　蒙特卡洛模拟法

可以看到成本分布接近于钟形，说明大部分结果集中于某个平均值由此证明模型具有较强的稳定性。此外，模型能够模拟出从显著亏损到盈利的各种成本情况，显示了模型在处理不同经济情景时的灵活性和适应性。

5.3 问题三模型的建立与求解

5.3.1 总成本度量模型的建立

在建立总成本度量函数模型的过程中通常我们需要考量检测成本，产品次品率，拆解成本和因次品造成的损失，据此本文建立了如下总成本度量函数模型：

检测成本：

$$C_{零件检测}=\sum_{i=1}^{n}x_i \bullet C_{di} \bullet n_i \qquad (14)$$

其中 x_i 为是否检验零配件 i（1 表示检测，0 表示不检测），n_i 为采购的零配件数量，C_{di} 表示检测成本。

成品检验成本：

$$C_{成品检测}=\sum_{j=1}^{m}y_j \bullet C_{dj} \bullet n_{fj} \qquad (15)$$

其中 y_j 为是否检验成品，C_{dj} 为成品检验成本，n_{fj} 为装配成品的数量。

装配成本：

$$C_{装配}=\sum_{j=1}^{m}C_{aj}\bullet n_{fj} \tag{16}$$

其中 n_{fj} 表示工序 j 生产的成品数量。

拆解成本：

$$C_{拆解}=\sum_{j=1}^{m}z_j\bullet C_t\bullet p_{fj}\bullet n_{fj} \tag{17}$$

其中 p_{fj} 为不合格成品率，n_{fj} 为成品数量。

调换成本：

$$C_{调换成本}=(1-z_j)\bullet C_r\bullet p_{fj}\bullet n_{fj} \tag{18}$$

成品次品率：

$$p_{fj}=1-\prod(1-p_{零部件}) \tag{19}$$

将以上方程进行整合可得到总成本度量函数模型如下所示：

$$C_{总成本}=C_{零配件检测}+C_{装配}+C_{成品检测}+C_{拆解}+C_{调换} \tag{20}$$

5.3.2 总成本度量模型的建立

为了对应用 NSGA-II 这一改进的遗传算法后得到的策略进行评价得到的指标在本文中引入总成本度量模型进行求解，为了构建总成本度量模型我们需要引入检测成本，产品次品率，拆解成本和因次品造成的损失进行计算，这几个数值的变量如下：

5.3.3NSGA-II 算法模型的建立

在处理问题二的情境中，即当生产工序数量 m=1 且零配件数量 n=2 时，我们构建了一个总成本度量函数模型来优化生产过程。由与在第三问中存在 8 个零部件和 3 个半成品，因此决策过程过于复杂情况过于多，因此在第二问中所使用的传统遗传算法不能解决问题三因此 NSGA-II 这一改进的遗传算法进行求解。NSGA-II 是一种广泛适用的多目标优化算法，在面对较为复杂的决策问题时枚举所有的支配关系是不现实的，而 NSGA-II 这一改进的遗传算法即保存了遗传算法交叉变异淘汰的思想又引入更高效的非支配排序算法。其工作的流程图如下：

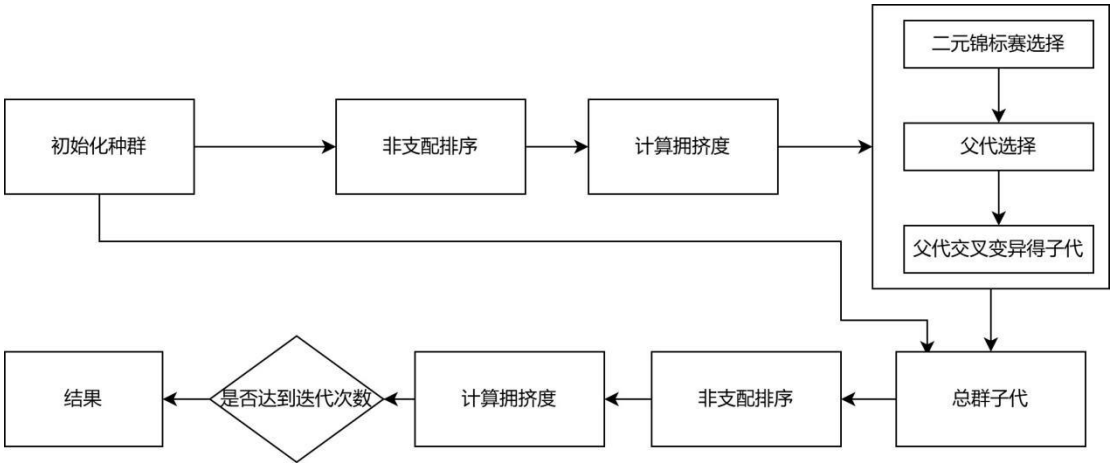


图 7　NSGA-II 算法运行流程图

具体步骤如下：

（1）初始化：生成种群即全部解集。

（2）非支配排序：选取对于目标函数具有相同优解的决策并将其分类。

（3）计算拥挤度：计算各类解集间的距离

（4）选择:利用二元基锦标法进行求解，及每次选择两个解，据拥挤度和非支配等级进行比较选择拥挤度小或非支配等级高的元素。

（5）交叉和变异:模拟生物进化过程中的进化和变异

（6）更新种群:将新生成的解与当前种群进行合并最后重新计算新种群的

（7）终止

5.3.4 结果分析

在经过 81 代的训练后得到训练过程图示如下，下图中展示了随代数增加最佳花费和最佳适应度以及最佳决策程度的迭代变化图：

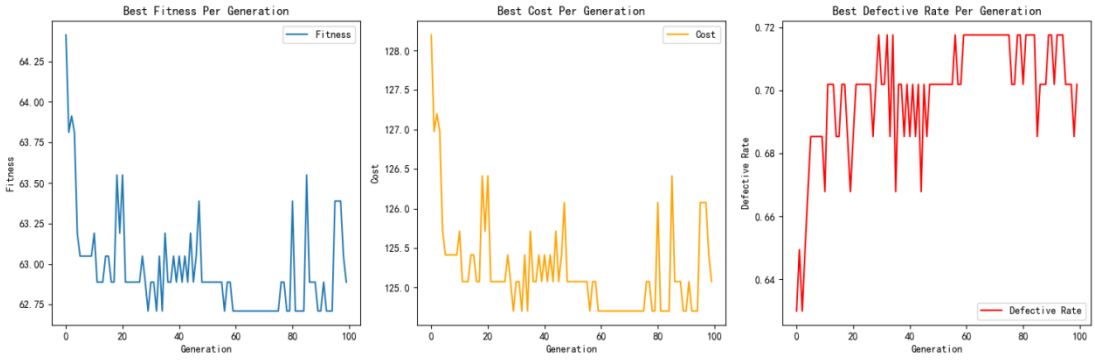


图 5　训练代数图

最后得到的最优策略为不检测零配件 1，检测零配件 2，不检测零配件 3，不检测零配件 4，不检测零配件 5，不检测零配件 6，不检测零配件 7，不检测零配件 8，不检测半成品 1，不检测半成品 2，不检测半成品 3，不检测成品，不拆解不合格成品。该策略是在训练第 81 代后得到的结果。在该策略下得到的

总利润为 19872 元。

5.4 问题四模型的建立与求解

5.4.1 抽样检测模型的建立

问题四要求我们重新审视问题二和问题三的决策方案，但这一次假设次品率是通过抽样检测得到的。在第一问中本文已经求出了 n 与 x（次品数量）的关系，由于次品率 $p=\frac{x}{n}$，我们进一步推导得出：

$$p>\frac{(Z_{\alpha}\sqrt{np(1-p)})^{2}}{n^{2}} \tag{21}$$

然而在实际操作中，抽样检测得到的次品率可能会有一定的误差，这种误差将影响后续所有决策的准确性，因此我们决定采用问题一中介绍的蒙特卡罗法来求解 p（次品率）的值。通过假定样本大小 n，我们可以利用蒙特卡罗模拟来计算 p。具体求解步骤如下：

（1）蒙特卡洛模拟

（2）计算样本统计量

（3）计算置信区间

（4）检查实际次品率是否在置信区间内

为了便于评估估计值的准确性和可靠性，观察值是否落在置信区间内是一个重要的统计检验步骤。在蒙特卡洛模拟中，置信区间为我们提供了次品率可能的波动范围，这个范围是在特定的置信水平（如 95%）下，通过多次模拟得到的次品率的统计分布所确定的。

5.4.2 结果分析

表 3 更新后的次品率

情况 次品率	零配件 1	零配件 2	成品
情况 1	0.09	0.02	0.02
情况 2	0.19	0.10	0.10
情况 3	0.10	0.02	0.02
情况 4	0.20	0.10	0.10
情况 5	0.10	0.10	0.02
情况 6	0.03	0.01	0.01

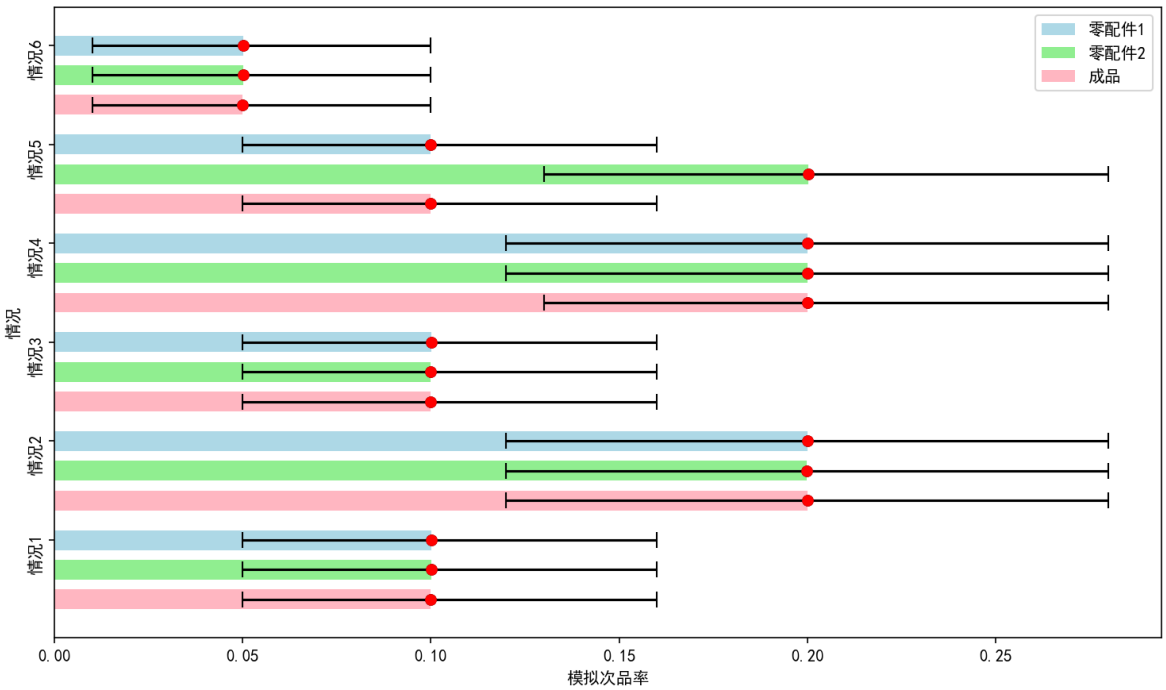


图 6　次品率求解图

对于重新完成问题二，由图可知，每个情况下各个部件的模拟次品率都落在了各自的置信区间内，这表明我们的模拟次品率估计值是可信的。将优化后的次品率代入问题二的模型可以得到如下结果图：

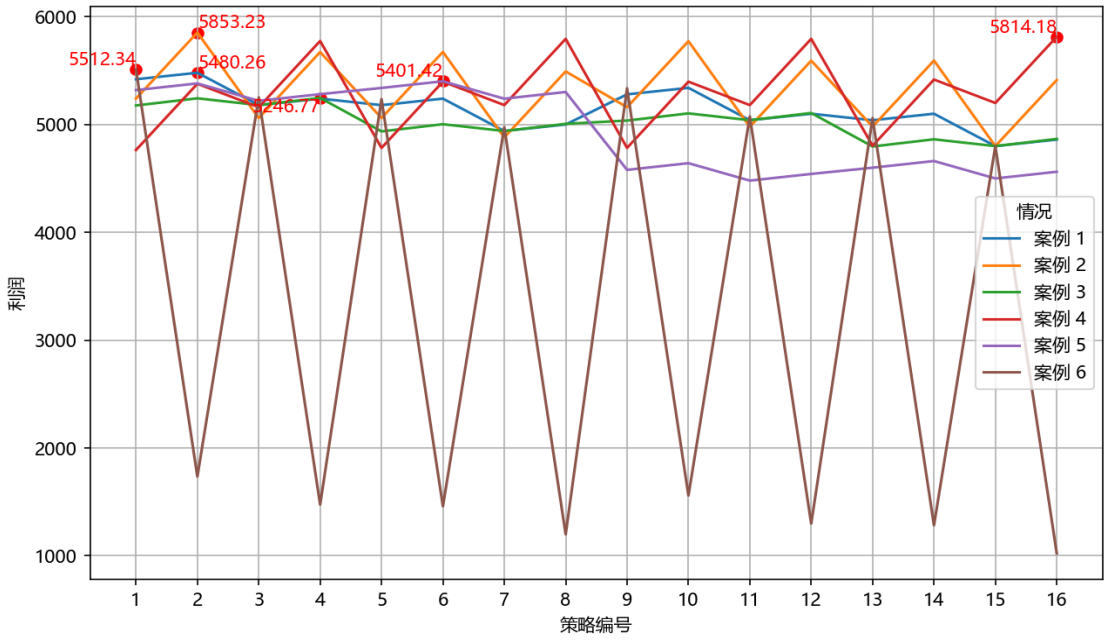


图 7　求解效益图

可以得出对第一种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5480.26 元。对于第二种情况来说最优方案为不检测零配件 1，不检测零配件 2，检测成品，拆解不合格成品，最大利润为 5853.23。对于第三种情况来说最优方案为不检测零配件 1，不检测零配

件 2，检测成品，拆解不合格成品，最大利润为 4267.77。对于第四种情况来说最优方案为检测零配件 1，检测零配件 2，检测成品，拆解不合格成品，最大利润为 5814.18。第五种情况来说最优方案为不检测零配件 1，检测零配件 2，不检测成品，拆解不合格成品，最大利润为 5401.42。第六种情况来说最优方案为不检测零配件 1，不检测零配件 2，不检测成品，不拆解不合格成品，最大利润为 5512.34。

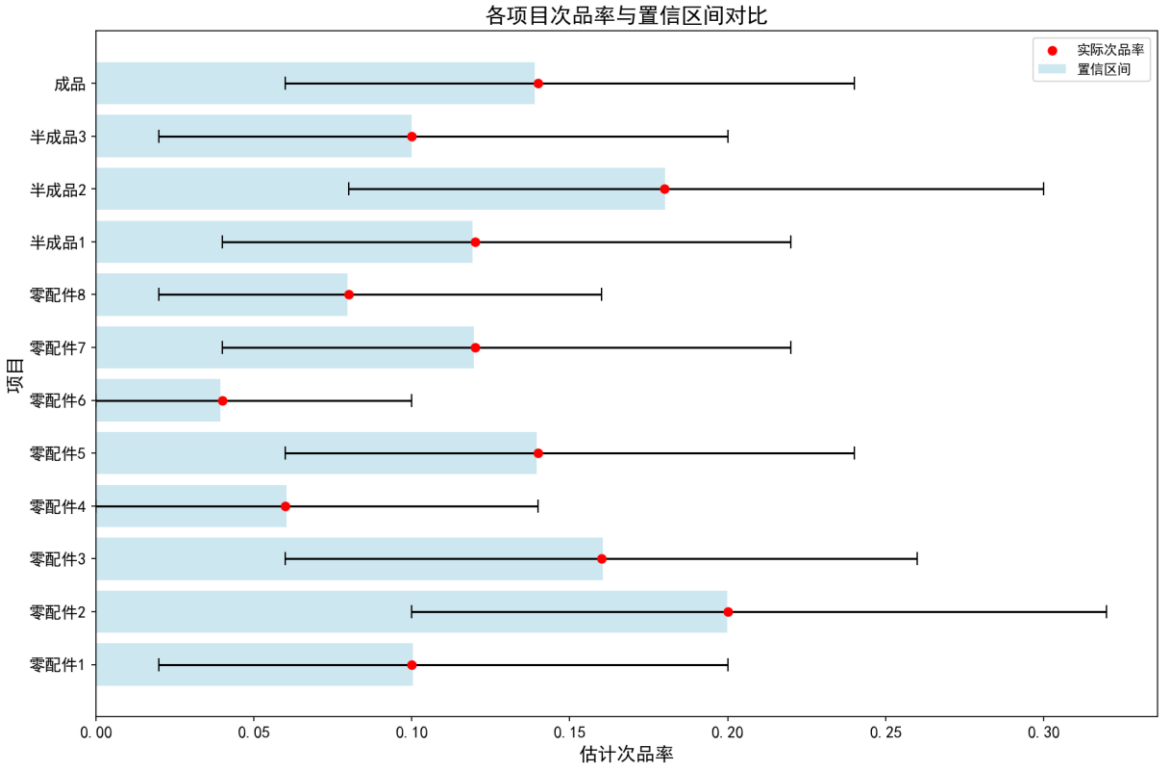


图 8　次品率求解图

对于重新完成问题三，如图所示，我们也能清晰地看出对于问题三的模拟次品率估计值是可信的。将优化后的次品率代入问题三的模型可以得到如下结果图：

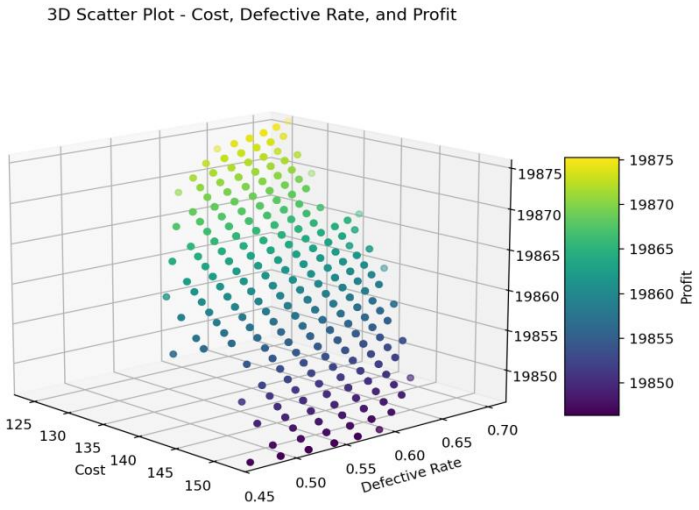


图 9　成本与成本次品率三维图

得到的结论为不检测零配件 1，检测零配件 2，不检测零配件 3，不检测零配件 4，不检测零配件 5，不检测零配件 6，不检测零配件 7，不检测零配件 8，不检测半成品 1，不检测半成品 2，不检测半成品 3，不检测成品，不拆解不合格成品。

本问是在保证其他的因素不变的情况下，单独改变了次品率，也得到了符合预期的结果，说明问题二和问题三的模型灵活性很强，能够在多变的实际情况下，快速而准确地提供有用的预测和决策支持。

六、模型的优缺点

优点：由于蚁群算法对问题空间的变化和局部最优解的影响不敏感，因此它非常适合用于第二问中最佳策略的选择。在求解第一问和第四问时，所使用的方程是以不等式形式给出的，这使得求解过程变得困难。蒙特卡洛方法则特别适合处理这类不等式问题。

缺点：蚁群算法对参数设置较为敏感，通常需要依赖经验或通过实验来确定最佳参数。

七、模型评价

（1）将遗传算法与快速收敛的局部优化方法进行混合[2]，进而得到有效的全局优化方法。

（2）遗传算法的随机搜索等特征使其具有明显的并行性[3]。可据此进行扩展与延申。

八、模型检验

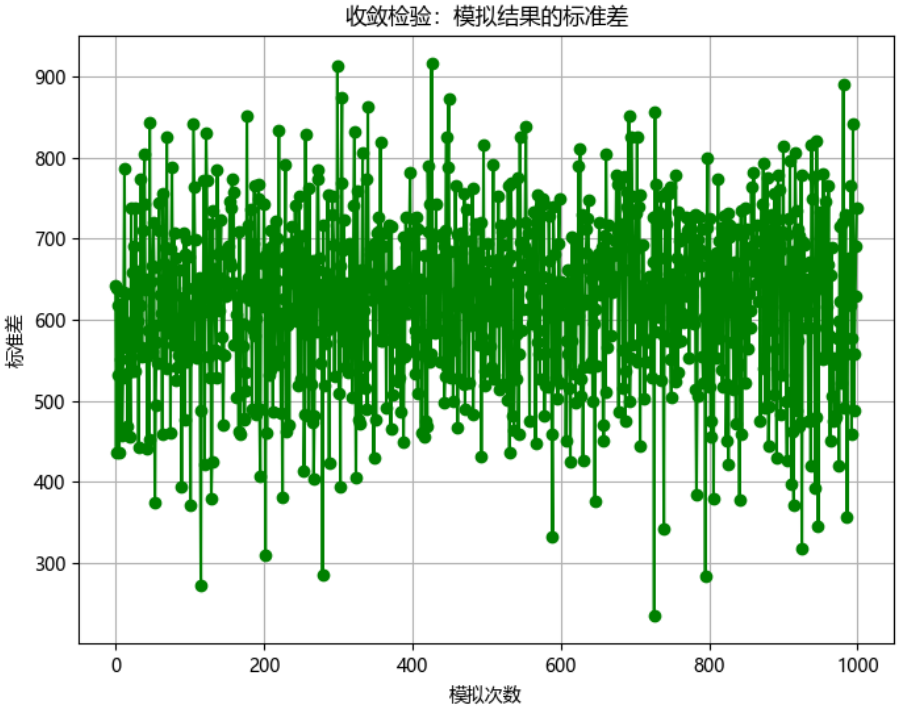


图 10 问题二模型检验

问题二中，我们使用遗传算法对利润进行了分析，关注过程中利润的变化情况。因此我们采用收敛性检验来验证利润的稳定性，并作图 9。根据图我们可以看出虽然标准差在不同的迭代次数下有波动，但整体上没有明显的上升或下降趋势。同时数据点的分布较为密集，说明在大多数迭代次数下，模型的标准差变化不大，意味着模型的性能也相对稳定。

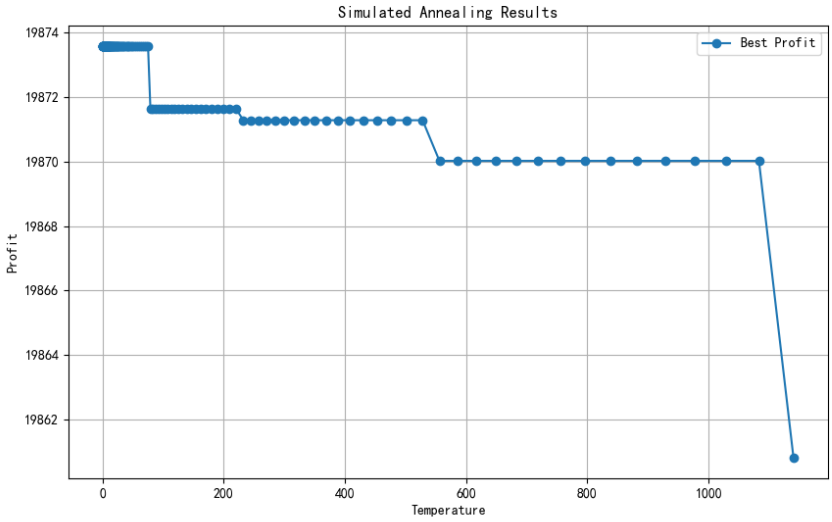


图 11 问题三模型检验

问题三中，我们还使用了模拟退火算法优化局部最优解，由图 10 我们可以知道，在温度较低的范围内（0 到 400），最佳利润保持相对稳定。 在温度达到一定值后（大约在 600 左右），最佳利润出现明显下降，而后达到一定平稳值，说明该模型趋于稳定。

九、参考文献

[1]. 朱陆陆.蒙特卡洛方法及应用[D].华中师范大学,2014.

[2]. 马永杰,云文霞.遗传算法研究进展[J].计算机应用研究,2012,29(04):1201-1206+1210.

[3]. 葛继科,邱玉辉,吴春明,等.遗传算法研究综述[J].计算机应用研究,2008,(10):2911-2916.

十、附录

附录 1

支撑材料

文件名	文件说明
Mentekaluo.py	蒙特卡洛算法代码
Genetic_heat.py	遗传算法模型代码
NSGA2.py	NSGA-2 模型代码
Exam_1.py	使用蒙特卡洛优化问题 2
Exam_2.py	使用蒙特卡洛优化问题 3
求 p.py	使用蒙特卡洛求解问题 1 的次品率
次品率和置信区间对比.py	对比图
问题 2.excel	问题 2 答案
问题三.excel	问题 3 答案
问题四.excel	问题 4 答案

附录 2

蒙特卡洛算法（问题 1）

```
def calculate_sample_size(p0, E, confidence_level):
    if confidence_level == 0.95:
        Z = 1.96
    elif confidence_level == 0.90:
        Z = 1.645
    else:
        raise ValueError("Unsupported confidence level")
    n = (Z ** 2 * p0 * (1 - p0)) / (E ** 2)
    return int(np.ceil(n))
def monte_carlo_simulation_accuracy(sample_size, true_defect_rate,
confidence_level, num_simulations=10000):
    Z_threshold = stats.norm.ppf(confidence_level)
    correct_predictions = 0
```

```
    for _ in range(num_simulations):
        sample = np.random.binomial(1, true_defect_rate, sample_size)
        defect_rate = np.mean(sample)
        Z = (defect_rate - p0) / np.sqrt(p0 * (1 - p0) / sample_size)
        predicted_reject = Z > Z_threshold
        actual_reject = true_defect_rate > p0
        if predicted_reject == actual_reject:
            correct_predictions += 1
    accuracy = correct_predictions / num_simulations
    return accuracy
p0 = 0.1
E = 0.05
confidence_levels = [0.95, 0.90]
true_defect_rate = 0.12
sample_sizes = {cl: calculate_sample_size(p0, E, cl) for cl in
confidence_levels}
print("样本大小:")
for cl, size in sample_sizes.items():
    print(f"{cl * 100}% 信度下的样本大小: {size}")
accuracies = {cl: monte_carlo_simulation_accuracy(sample_sizes[cl],
true_defect_rate, cl) for cl in confidence_levels}
print("\n模型准确度:")
for cl, acc in accuracies.items():
    print(f"{cl * 100}% 信度下的准确度: {acc:.2f}")
confidence_levels_percent = [cl * 100 for cl in confidence_levels]
accuracy_values = list(accuracies.values())
plt.figure(figsize=(8, 6))
plt.bar(confidence_levels_percent, accuracy_values, color='blue', width=5)
plt.xlabel('Confidence Level (%)')
plt.ylabel('Accuracy')
plt.title('Model Accuracy at Different Confidence Levels')
plt.ylim(0, 1)
plt.xticks(confidence_levels_percent)
plt.show()
```

附录 3

遗传算法（问题 2）

```
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
cases = [
    {'零配件 1 次品率': 0.1, '零配件 2 次品率': 0.100066, '成品次品率': 0.100060, '
零配件 1 检测成本': 2, '零配件 2 检测成本': 3,
     '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用':
5},
    {'零配件 1 次品率': 0.199834, '零配件 2 次品率': 0.200360, '成品次品率':
0.198670, '零配件 1 检测成本': 2, '零配件 2 检测成本': 3,
     '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 6, '拆解费用':
5},
    {'零配件 1 次品率': 0.100140, '零配件 2 次品率': 0.099458, '成品次品率':
0.101166, '零配件 1 检测成本': 2, '零配件 2 检测成本': 3,
     '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用':
5},
    {'零配件 1 次品率': 0.199388, '零配件 2 次品率': 0.199458, '成品次品率':
0.198960, '零配件 1 检测成本': 1, '零配件 2 检测成本': 1,
     '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 30, '拆解费用':
5},
    {'零配件 1 次品率': 0.100252, '零配件 2 次品率':  0.199966, '成品次品率':
0.100342, '零配件 1 检测成本': 8, '零配件 2 检测成本': 1,
     '成品检测成本': 2, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解费用':
5},
    {'零配件 1 次品率': 0.039728, '零配件 2 次品率': 0.040098, '成品次品率':
0.039766, '零配件 1 检测成本': 2, '零配件 2 检测成本': 3,
     '成品检测成本': 3, '装配成本': 6, '市场售价': 56, '调换损失': 10, '拆解费用':
40}
]
def calc_total_cost(case, detect_parts1=True, detect_parts2=True,
detect_final=True, dismantle=True):
    n_parts1 = 100
    n_parts2 = 100
    cost_parts1 = n_parts1 * case['零配件 1 检测成本'] if detect_parts1 else 0
    cost_parts2 = n_parts2 * case['零配件 2 检测成本'] if detect_parts2 else 0
    loss_parts1 = (n_parts1 * case['零配件 1 次品率']) * (case['装配成本']) if
not detect_parts1 else 0
    loss_parts2 = (n_parts2 * case['零配件 2 次品率']) * (case['装配成本']) if
```

```
not detect_parts2 else 0
    n_final_products = 100
    cost_final = n_final_products * case['成品检测成本'] if detect_final else
0
    loss_final = (n_final_products * case['成品次品率']) * case['调换损失'] if
not detect_final else 0
    dismantle_cost = n_final_products * case['拆解费用'] if dismantle else 0
    dismantle_revenue = (n_final_products * case['成品次品率']) * case['市场售
价'] if dismantle else 0
    total_cost = (cost_parts1 + cost_parts2 + loss_parts1 + loss_parts2 +
                cost_final + loss_final + dismantle_cost -
dismantle_revenue)
    return total_cost
strategies = list(itertools.product([False, True], repeat=4))
strategy_labels = [
    "检测零配件 1", "检测零配件 2", "检测成品", "拆解不合格成品"
]
def strategy_identifier(strategy):
    return ''.join(['1' if x else '0' for x in strategy])
all_cases_results = []
for case_index, case in enumerate(cases):
    print(f"案例 {case_index + 1} 的结果:")
    costs = [calc_total_cost(case, *strategy) for strategy in strategies]
    profits = [case['市场售价'] * 100 - cost for cost in costs]
    min_cost = min(costs)
    optimal_strategy = strategies[costs.index(min_cost)]
    all_cases_results.append({
        'case_index': case_index + 1,
        'strategies': strategies,
        'costs': costs,
        'profits': profits,
        'min_cost': min_cost,
        'optimal_strategy': optimal_strategy
    })
    for strategy_index, (strategy, cost, profit) in
enumerate(zip(strategies, costs, profits), start=1):
        strategy_desc = ", ".join([f"{strategy_labels[i]}: {'是' if
strategy[i] else '否'}" for i in range(4)])
        print(f"策略 {strategy_index}: {strategy_desc}, 总成本: {cost:.2f}, 利
润: {profit:.2f}")
    print(f"\n 最优策略: {optimal_strategy}, 最小总成本: {min_cost:.2f}, 最大利
润: {max(profits):.2f}\n")
plt.figure(figsize=(10, 6))
offsets = [-0.05, 0.05, -0.08, 0.08, -0.10, 0.10]
```

```
for idx, case_result in enumerate(all_cases_results):
    profits = case_result['profits']
    plt.plot(range(1, 17), profits, label=f"案例
{case_result['case_index']}")
    max_profit_index = np.argmax(profits) + 1
    max_profit_value = max(profits)
    plt.scatter(max_profit_index, max_profit_value, color='red')
    offset = offsets[idx % len(offsets)]
    if case_result['case_index'] == 3:
        plt.text(max_profit_index, max_profit_value + offset,
f'{max_profit_value:.2f}',
                 horizontalalignment='right', verticalalignment='top',
fontsize=10, color='red')
    elif case_result['case_index'] == 2:
        plt.text(max_profit_index, max_profit_value + offset,
f'{max_profit_value:.2f}',
                 horizontalalignment='left', verticalalignment='bottom',
fontsize=10, color='red')
    elif case_result['case_index'] == 1:
        plt.text(max_profit_index, max_profit_value + offset,
f'{max_profit_value:.2f}',
                 horizontalalignment='left', verticalalignment='bottom',
fontsize=10, color='red')
    else:
        plt.text(max_profit_index, max_profit_value + offset,
f'{max_profit_value:.2f}',
                 horizontalalignment='right', verticalalignment='bottom',
fontsize=10, color='red')

plt.xlabel('策略编号')
plt.ylabel('利润')
plt.legend(title='情况')
plt.xticks(range(1, 17))
plt.grid(True)
plt.show()
strategy_names = [strategy_identifier(strategy) for strategy in strategies]
df_all_cases = pd.DataFrame()

for case_result in all_cases_results:
    case_index = case_result['case_index']
    costs = case_result['costs']
    profits = [cases[case_index - 1]['市场售价'] * 100 - cost for cost in
costs]
    df_case = pd.DataFrame({
```

```
        '策略': strategy_names,
        '利润': profits,
        '情况': case_index
    })
    df_all_cases = pd.concat([df_all_cases, df_case], ignore_index=True)

heatmap_data = df_all_cases.pivot(index="策略", columns="情况", values="利润
")
plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data, annot=True, cmap="YlGnBu", fmt=".2f",
cbar_kws={'label': '利润'})
plt.title("不同策略的利润热力图")
plt.xlabel("情况")
plt.ylabel("策略")
plt.xticks(ticks=np.arange(len(cases)) + 0.5, labels=[f"情况 {i+1}" for i in
range(len(cases))], rotation=45)
plt.yticks(ticks=np.arange(len(strategies)) + 0.5, labels=[f"策略 {i+1}" for
i in range(len(strategies))], rotation=0)
plt.tight_layout()
plt.show()
def strategy_identifier(strategy):
    return f"({int(strategy[0])}, {int(strategy[1])}, {int(strategy[2])},
{int(strategy[3])})"
def strategy_identifier(strategy):
    return f"({int(strategy[0])}, {int(strategy[1])}, {int(strategy[2])},
{int(strategy[3])})"
def plot_profits(all_cases_results, strategies, strategy_labels):
    group1 = all_cases_results[:3]
    group2 = all_cases_results[3:]
    plot_group_profits(group1, strategies, strategy_labels, "情况 1-3 各策略总
利润")
    plot_group_profits(group2, strategies, strategy_labels, "情况 4-6 各策略总
利润")
def plot_group_profits(cases_group, strategies, strategy_labels, title):
    num_cases = len(cases_group)
    fig, axes = plt.subplots(nrows=num_cases, ncols=1, figsize=(15, 3 *
num_cases))
    if num_cases == 1:
        axes = [axes]
    for ax, case_result in zip(axes, cases_group):
        case_num = case_result['case_index']
        profits = case_result['profits']
        max_profit = max(profits)
        optimal_strategy =
```

```
case_result['strategies'][profits.index(max_profit)]
        optimal_index = case_result['strategies'].index(optimal_strategy)
        strategy_ids = [strategy_identifier(s) for s in strategies]
        sns.barplot(x=strategy_ids, y=profits, ax=ax, palette="viridis")
        ax.set_title(f"情况 {case_num} 各策略总利润")
        ax.set_xlabel("策略 (检测零配件 1, 检测零配件 2, 检测成品, 拆解不合格成品)")
        ax.set_ylabel("总利润")
        ax.bar(strategy_ids[optimal_index], profits[optimal_index],
color='red', label='最优策略 (最大利润)')
        ax.legend()
    fig.suptitle(title, fontsize=16)
    fig.tight_layout(rect=[0, 0, 1, 0.96])
    fig.subplots_adjust(hspace=0.5)
    plt.show()
plot_profits(all_cases_results, strategies, strategy_labels)
def monte_carlo_simulation_profit(case, num_simulations=1000):
    results = []
    for _ in range(num_simulations):
        case['零配件 1 次品率'] = np.random.uniform(0, 0.5)
        case['零配件 2 次品率'] = np.random.uniform(0, 0.5)
        case['成品次品率'] = np.random.uniform(0, 0.5)
        costs = [calc_total_cost(case, *strategy) for strategy in strategies]
        profits = [case['市场售价'] * 100 - cost for cost in costs]
        max_profit = max(profits)
        results.append(max_profit)
    return results
simulation_profit_results = monte_carlo_simulation_profit(cases[0])
plt.figure(figsize=(8, 6))
sns.histplot(simulation_profit_results, kde=True)
plt.title('蒙特卡罗模拟: 总利润分布')
plt.xlabel('总利润')
plt.ylabel('频率')
plt.show()
```

附录 4

NSGA-2 模型（问题 3）

```
population_size = 9000
generations = 15
mutation_rate = 0.1
crossover_rate = 0.7
initial_temperature = 1000
```

```
cooling_rate = 0.95
components = {
    '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},
}
semi_products = {
    '半成品1': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品2': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
    '半成品3': {'次品率': 0.1, '装配成本': 8, '检测成本': 4, '拆解费用': 6},
}
final_product = {
    '成品': {'次品率': 0.1, '装配成本': 8, '检测成本': 6, '拆解费用': 10, '售价':
200, '调换损失': 40}
}
def calculate_expected_cost(components, semi_products, final_product,
detect_components, detect_semi, detect_final,
                        dismantle):
    total_cost = 0
    total_defective_rate = 1
    for i, (comp_name, comp_data) in enumerate(components.items()):
        cost = comp_data['购买单价']
        defective_rate = comp_data['次品率']
        if detect_components[i]:
            defective_rate *= 0.5
            cost += comp_data['检测成本']
        total_cost += cost
        total_defective_rate *= (1 - defective_rate)
    for i, (semi_name, semi_data) in enumerate(semi_products.items()):
        cost = semi_data['装配成本']
        defective_rate = semi_data['次品率']
        if detect_semi[i]:
            defective_rate *= 0.5
            cost += semi_data['检测成本']
        total_cost += cost
        total_defective_rate *= (1 - defective_rate)
    final_defective_rate = final_product['成品']['次品率'] * (0.5 if
detect_final else 1)
    final_cost = final_product['成品']['装配成本']
```

```
    if detect_final:
        final_cost += final_product['成品']['检测成本']
    total_cost += final_cost
    total_defective_rate = 1 - (total_defective_rate * (1 -
final_defective_rate))
    exchange_loss = final_product['成品']['调换损失'] * total_defective_rate
    total_cost += exchange_loss
    if dismantle:
        total_cost += final_product['成品']['拆解费用']
    total_units = 100
    total_revenue = total_units * final_product['成品']['售价']
    profit = total_revenue - total_cost
    return total_cost, total_defective_rate, profit
def generate_all_possible_decisions():
    decisions = list(itertools.product([True, False], repeat=13))
    return decisions
all_possible_decisions = generate_all_possible_decisions()
def encode_individual(index):
    detect_strategy = all_possible_decisions[index]
    detect_components = detect_strategy[:8]
    detect_semi = detect_strategy[8:11]
    detect_final = detect_strategy[11]
    dismantle = detect_strategy[12]
    return detect_components, detect_semi, detect_final, dismantle
def initialize_population(size, all_decisions):
    population = []
    for i in range(size):
        individual = encode_individual(i % len(all_decisions))
        population.append(individual)
    return population
def fitness(individual):
    detect_components, detect_semi, detect_final, dismantle = individual
    cost, defective_rate, profit = calculate_expected_cost(components,
semi_products, final_product,
                                                    detect_components,
detect_semi, detect_final, dismantle)
    return profit
def selection(population):
    fitness_values = [1 / fitness(ind) for ind in population]
    total_fitness = sum(fitness_values)
    probabilities = [f / total_fitness for f in fitness_values]
    selected_population = []
    for _ in range(population_size):
        r = random.random()
```

```
        cumulative_prob = 0
        for i, individual in enumerate(population):
            cumulative_prob += probabilities[i]
            if r < cumulative_prob:
                selected_population.append(individual)
                break
    return selected_population
def crossover(parent1, parent2):
    if random.random() < crossover_rate:
        point = random.randint(1, len(parent1[0]) - 1)
        parent1_components = list(parent1[0])
        parent2_components = list(parent2[0])
        child1_components = parent1_components[:point] +
parent2_components[point:]
        child2_components = parent2_components[:point] +
parent1_components[point:]
        parent1_semi = list(parent1[1])
        parent2_semi = list(parent2[1])
        child1_semi = parent1_semi[:1] + parent2_semi[1:]
        child2_semi = parent2_semi[:1] + parent1_semi[1:]
        child1_final = parent1[2]
        child2_final = parent2[2]
        child1_dismantle = parent1[3]
        child2_dismantle = parent2[3]
        child1 = (tuple(child1_components), tuple(child1_semi), child1_final,
child1_dismantle)
        child2 = (tuple(child2_components), tuple(child2_semi), child2_final,
child2_dismantle)
        return child1, child2
    else:
        return parent1, parent2
def mutate(individual):
    detect_components, detect_semi, detect_final, dismantle = individual
    detect_components = list(detect_components)
    detect_semi = list(detect_semi)
    if random.random() < mutation_rate:
        component_to_mutate = random.randint(0, len(detect_components) - 1)
        detect_components[component_to_mutate] = not
detect_components[component_to_mutate]
    if random.random() < mutation_rate:
        semi_to_mutate = random.randint(0, len(detect_semi) - 1)
        detect_semi[semi_to_mutate] = not detect_semi[semi_to_mutate]
    if random.random() < mutation_rate:
        detect_final = not detect_final
```

```
    if random.random() < mutation_rate:
        dismantle = not dismantle
    return detect_components, detect_semi, detect_final, dismantle
def accept_solution(old_fitness, new_fitness, temperature):
    if new_fitness < old_fitness:
        return True
    else:
        probability = np.exp(-(new_fitness - old_fitness) / temperature)
        return random.random() < probability
def generate_strategy_description(detect_components, detect_semi,
detect_final, dismantle):
    description = ""
    for i, detect in enumerate(detect_components):
        description += f"检测零配件 {i + 1}, " if detect else f"不检测零配件 {i +
1}, "
    for i, detect in enumerate(detect_semi):
        description += f"检测半成品 {i + 1}, " if detect else f"不检测半成品 {i +
1}, "
    if detect_final:
        description += "检测成品, "
    else:
        description += "不检测成品, "
    if dismantle:
        description += "拆解不合格成品"
    else:
        description += "不拆解不合格成品"
    return description
all_possible_decisions = generate_all_possible_decisions()    results =
pd.DataFrame(all_generations_data)
    return results
results_df = genetic_simulated_annealing(all_possible_decisions)
results_df.to_excel('strategy_results.xlsx', index=False)
print("结果已保存到 'strategy_results.xlsx'")
```

附录 5

蒙特卡洛算法优求 p（问题 4）

```
from matplotlib import rcParams
def monte_carlo_simulation(k, n, num_simulations=10000):
    samples = np.random.binomial(n, k / n, size=num_simulations) / n
    return samples.mean(), samples.std(), samples
def confidence_interval(samples, confidence_level=0.95):
```

```
    lower_bound = np.percentile(samples, (1 - confidence_level) / 2 * 100)
    upper_bound = np.percentile(samples, (1 + confidence_level) / 2 * 100)
    return lower_bound, upper_bound
scenarios = {
    '情况 1': {'零配件 1': {'次品率': 0.1, '购买单价': 4, '检测成本': 2},
            '零配件 2': {'次品率': 0.1, '购买单价': 18, '检测成本': 3},
            '成品': {'次品率': 0.1, '装配成本': 6, '检测成本': 3}},
    '情况 2': {'零配件 1': {'次品率': 0.2, '购买单价': 4, '检测成本': 2},
            '零配件 2': {'次品率': 0.2, '购买单价': 18, '检测成本': 3},
            '成品': {'次品率': 0.2, '装配成本': 6, '检测成本': 3}},
    '情况 3': {'零配件 1': {'次品率': 0.1, '购买单价': 4, '检测成本': 2},
            '零配件 2': {'次品率': 0.1, '购买单价': 18, '检测成本': 3},
            '成品': {'次品率': 0.1, '装配成本': 6, '检测成本': 3}},
    '情况 4': {'零配件 1': {'次品率': 0.2, '购买单价': 4, '检测成本': 1},
            '零配件 2': {'次品率': 0.2, '购买单价': 18, '检测成本': 1},
            '成品': {'次品率': 0.2, '装配成本': 6, '检测成本': 2}},
    '情况 5': {'零配件 1': {'次品率': 0.1, '购买单价': 4, '检测成本': 8},
            '零配件 2': {'次品率': 0.2, '购买单价': 18, '检测成本': 1},
            '成品': {'次品率': 0.1, '装配成本': 6, '检测成本': 2}},
    '情况 6': {'零配件 1': {'次品率': 0.05, '购买单价': 4, '检测成本': 2},
            '零配件 2': {'次品率': 0.05, '购买单价': 18, '检测成本': 3},
            '成品': {'次品率': 0.05, '装配成本': 6, '检测成本': 3}},
}
n = 50
num_simulations = 10000
updated_results = []
def adjust_k_for_scenario(scenario_data, n):
    k1 = int(scenario_data['零配件 1']['次品率'] * n)
    k2 = int(scenario_data['零配件 2']['次品率'] * n)
    k_prod = int(scenario_data['成品']['次品率'] * n)
    return {'零配件 1': {'k': k1, 'n': n}, '零配件 2': {'k': k2, 'n': n}, '成品
': {'k': k_prod, 'n': n}}
for scenario_name, scenario_data in scenarios.items():
    sampling_data_adjusted = adjust_k_for_scenario(scenario_data, n)
    result = {'情况': scenario_name}
    k1 = sampling_data_adjusted['零配件 1']['k']
    n1 = sampling_data_adjusted['零配件 1']['n']
    mean_defect_rate1, std_defect_rate1, samples1 =
monte_carlo_simulation(k1, n1, num_simulations)
    lower_bound1, upper_bound1 = confidence_interval(samples1)
    result['零配件 1_模拟次品率'] = mean_defect_rate1
    result['零配件 1_置信区间下限'] = lower_bound1
    result['零配件 1_置信区间上限'] = upper_bound1
    k2 = sampling_data_adjusted['零配件 2']['k']
```

```
    n2 = sampling_data_adjusted['零配件 2']['n']
    mean_defect_rate2, std_defect_rate2, samples2 =
monte_carlo_simulation(k2, n2, num_simulations)
    lower_bound2, upper_bound2 = confidence_interval(samples2)
    result['零配件 2_模拟次品率'] = mean_defect_rate2
    result['零配件 2_置信区间下限'] = lower_bound2
    result['零配件 2_置信区间上限'] = upper_bound2
    k_prod = sampling_data_adjusted['成品']['k']
    n_prod = sampling_data_adjusted['成品']['n']
    mean_defect_rate_prod, std_defect_rate_prod, samples_prod =
monte_carlo_simulation(k_prod, n_prod, num_simulations)
    lower_bound_prod, upper_bound_prod = confidence_interval(samples_prod)
    result['成品_模拟次品率'] = mean_defect_rate_prod
    result['成品_置信区间下限'] = lower_bound_prod
    result['成品_置信区间上限'] = upper_bound_prod

    updated_results.append(result)
df = pd.DataFrame(updated_results)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
print(df)

fig, ax = plt.subplots(figsize=(10, 6))

colors = ['#add8e6', '#90ee90', '#ffb6c1']
components = ['零配件 1', '零配件 2', '成品']

for i, (scenario_name, scenario_data) in enumerate(df.iterrows()):
    scenario_results = scenario_data.to_dict()
    scenario_y_pos = i
    for j, component in enumerate(components):
        mean_defect_rate = scenario_results[f'{component}_模拟次品率']
        lower_bound = scenario_results[f'{component}_置信区间下限']
        upper_bound = scenario_results[f'{component}_置信区间上限']
        ax.barh(scenario_y_pos - j * 0.3, mean_defect_rate, color=colors[j],
height=0.2, label=component if i == 0 else "")
        ax.errorbar(mean_defect_rate, scenario_y_pos - j * 0.3,
                    xerr=[[mean_defect_rate - lower_bound], [upper_bound -
mean_defect_rate]],
                    fmt='o', color='black', capsize=5)
        ax.scatter(mean_defect_rate, scenario_y_pos - j * 0.3, color='red',
zorder=5)
ax.set_yticks(range(len(df)))
```

```
ax.set_yticklabels(df['情况'], rotation='vertical')
plt.xlabel('模拟次品率')
plt.ylabel('情况')
plt.legend()
plt.tight_layout()
plt.show()
```