基于抽样方法选择与模拟退火迭代算法的生产决策模型研究

摘 要

在实际生产过程中，从零配件的选购到多阶段生产决策的每一步，均直接影响企业的利润收益。本文基于全流程视角，针对零配件选购和多阶段生产过程中的决策问题，通过建立适当的数学模型优化决策流程，以最大化企业利润。

**针对问题一**，设计了一种抽样检测方案，在最小检测次数下，以较高概率作出接受或拒绝决策，并计算两种情况下的最小检测次数值。该问题被建模为统计学中的二项分布概率问题，采用简单随机抽样生成样本数据，结合序贯概率比检验（SPRT）模型，逐步计算样本的似然比值，与预设阈值比较后决定下一步操作，最终验证了最小检测次数的合理性，并通过OC曲线和ASN曲线辅助判断模型效果。

**针对问题二**，研究了一个包含多决策点的多阶段生产决策问题。本文通过构建0/1规划的成本效益模型，将决策变量分为是否拆解零配件、是否检测成品，以及是否对不合格成品进行拆解处理。利用动态规划方法进行求解，最终得到每种情况的最佳决策及其对应的最大利润。

**针对问题三**，复杂化多阶段决策模型，增加了更多零配件与工序，使生产流程呈现网络化结构。为解决成本效益模型无法直接适用的问题，采用模拟退火迭代算法优化决策效率与正确性。通过实例验证，模拟退火迭代算法在大规模生产决策中的优越性得以体现。

**针对问题四**，在问题二和问题三的基础上，引入抽样检测确定次品率的方法，以更贴近实际生产环境。本文优化了模型，使其能够动态处理次品率波动情况下的决策选择，并通过实例验证优化模型在复杂实际场景中的应用效果。

关键词：抽样检测方案、序贯概率比检验、多阶段决策、成本效益分析、模拟退火迭代算法

1. 问题重述

1.1问题的背景

最近，一款电子产品因市场需求旺盛而畅销热卖，某企业因此计划生产该产品以实现盈利。据悉，生产此类电子产品至少需要两种零配件（零配件1和零配件2）。生产流程包括企业自行采购这两种零配件，并组装成品进行销售。

在具体的生产过程中，如果两种零配件中有一种不合格，则组装出的成品必定不合格。但即便两种零配件均合格，成品也不一定能够通过质量检验。对于不合格的成品，企业可以选择直接报废处理，丢弃产品，或者花费一定费用进行拆解。拆解过程中，合格的零配件不会受损，因此可以重新投入下一轮生产，从而节省采购新零配件的成本。

1.2需要解决的问题

问题一：抽样检测方案的设计：由于供应商的零件有一定的次品率，虽然供应商提供次品率标称值为，但该数值的真实性无法保障。因此企业想要对其产品进行抽样检测，需要我们提供一种抽样检测的方案，具体要求为在尽可能少的检测次数下（节省检测成本)，能够根据跟定的不同信度水平，帮助企业做出是否接受供应商提供的某一批零件的决定。问题设计统计假设理论、抽样方法等知识，需要我们根据计算结果提供一种科学的抽样检测方案，并检验正确性。

问题二：现在进入正式的生产流程决策：两种零配件和成品次品率都已给出，我们需要根据对零配件1和2是否进行检测、对成品是否进行检测、对检测到的不合格成品和用户退回的不合格产品是否进行拆解再使用这三个过程进行四次决策，依旧给定的不同次品率的数据组，对不同种情况分别进行计算，得出每组最大收益，进行给出每种情况的具体决策选择。问题设计多阶段决策过程，需要我们建立一个全面、准确的决策模型，以尽量得到最大化的利润。

问题三：进一步增加多阶段决策的复杂性：在真实的生产过程中，该电子产品的生产不仅仅只需要两个零配件、一次组装，而是多道工序、多次配件组装。现在题目中已经给出在2道工序、8个零配件的生产流程示意图，以及生产过程所需要的零配件和成品次品率等参考数据。该题的要求是在问题二的基础上，增加考虑项即多个半成品各自是否检测、是否拆解的决策选择，在问题二的成本效益模型上结合优化算法，在复杂的生产网络中找到全局最优解，做出决策。并以此推导估计出道工序、个零配件的生产流程所应做出决策的简易思路。

问题四：问题中他提出上文中零配件、半成品、成品的次品率均为抽样检测得到的，希望我们据此重新计算问题二、三，即该项参考值并不准确。这就要求我们要将次品率看作一个范围波动的值，在这种不确定的波动范围内，去作出较为正确的决策。

二、问题分析

这道题目是一道关于电子产品生产的企业决策问题，涉及到了统计学、概率学、多过程决策等多方面的知识，涉及到的主要问题包括零部件选购过程中的抽样检测方法的选择，生产过程中的工序决策以及在次品率范围波动情况下的工序决策问题。

**针对问题一**：题目中要求设计一个抽样检测方案,在此方案下能够利用最少的检测次数确保检测结果能够较为可靠的反映真实的次品率,以便决定是否接受供应商提供的零配件批次。这个问题主要涉及到了统计学和概率论中的假设检验理论。在此问题中，我们可以使用二项分布及正态分布近似的模型，通过设定显著性水平和检验功效，计算得出最优的样本量。因为检测费要由企业自己承担，所以要尽可能保证较少的检测次数，但是在此问题中，除了较少的检测次数，我们还需要考虑做出错误决策的风险概率。因此，我们可以再使用序贯概率比检验算法来进行验证。总的来说，可以先结合两种情形，利用所建模型求出样本容量，再对结果进行正确性检验。

**针对问题二：**这是一个典型的多阶段决策问题，共有零配件检测、成品装配、成品检测、用户不合格品调换和不合格品拆解处理等多个环节，其中在零配件1和2各自是否检测、成品是否检测、不合格品是否拆解等环节进行决策。这里解题的思路模型就是一个全面、多决策变量的01规划模型，将各个环节的决策都考虑在内，将利润最大化作为目标函数，各阶段决策的01状态都作为决策变量，约束条件较多，比如每一个决策变量都为01状态以及不同情况下最终得到的产品中合格率的约束和选择不同策略时对应的调换成本、拆解成本等的约束。总体来说就是根据各阶段决策变量状态的不同，分别计算出某种情况下各种决策所能得到的目标收益，最后横向比较得到最终决策，进而计算出每一种情况的最优策略。

**针对问题三：**问题三是在问题二的基础上进一步加大数据量，涉及到更多决策点，使得实际问题模型更加复杂、多样化，形成了网络状的生产流程。具体分析问题：由于决策变量的增加，问题二中建立的成本效益模型就将不再适用，因此，需要对原模型进行抽象化分析，以便其能够对问题三中给定情形进行计算。但由于最终问题还要上升到m道工序、n件零配件的巨大数据量，因此还需要再对抽象后的成本收益模型进行优化，比如使用模拟退火迭代算法，该算法可以通过随机地动态搜索和概率接受机制来不断逼近全局最优解，进而得出最终决策。最终还需要将模拟退火算法求得的最终解与新成本效益模型下的最终解进行比较，以此佐证该算法的能够在较短时间内得到接近最优解的决策方案，即该算法高效率的优越性。

**针对问题四**：问题四其实就是在问题二、三的基础上进行拓展，利用问题一的背景进行问题二、三情况的求解。主要涉及如何在存在不确定性的情况下对零配件、半成品和成品的次品率进行估算，进而做出决策。问题中提到其次品率的标称值并不准确，而是在一定区间内波动，并且各个零配件、半成品、成品等的次品率也在范围性波动。因此，我们可以选择使用抽样检测方法，来动态得到每组样品的次品率，利用正态分布的概率模型分布和蒙特卡洛方法进行方差估计，得到合理的方差值来表示次品率的波动。同时,利用抽样检测得到的次品率这一信息，辅助的帮助优化问题二，三的求解，当抽样检测得到的次品率不能够被接受时，模型应该能够反映这一情况并动态的做出相应的决策。

三、模型假设

1.检验准确性假设：在模型中，我们假设随机抽样后的检测过程中，对不合格产品能够100%准确识别。

2.独立性假设：我们假设在生产过程中的生产工序间是相互独立的，不会发生相互影响。

3.稳定性假设：生产阶段中的产品的各种生产概率值和各个数值均为恒定的，不受外界影响（如零配件、半成品、成品的组装产生次品率等值为固定值）

四、符号说明

|  |  |  |  |
| --- | --- | --- | --- |
| 符号 | 含义 |  | 购买零件的成本 |
| E | 允许的误差范围 |  | 检测零件和成品的成本 |
| n | 样本量 |  | 组装成品的成本 |
| α | 第一类犯错概率 |  | 拆解成品所造成的成本 |
| β | 第二类犯错概率 |  | 因次品到用户手中进行调换造成的损失 |
| (i=1,2…) | 零配件的购买成本 | （i） | 第i个零件的的购买成本 |
| (i=1,2…) | 零配件的检测成本 | （i） | 第i个零件或半成品工序的检测成本 |
|  | 成品的检测成本 | （i） | 第i个成品或半成品工序的装配成本 |
|  | 成品售价 | （i） | 对得到的不合格品进行拆解的成本 |
|  | 成品次品率 |  | 对产品的次品率进行修正至多零件检测之后的值 |

1. 模型建立与求解

5.1问题一模型的建立与求解

5.1.1假设检验：二项分布模型

现在企业需要用一项抽样检测方案检验测定一批零配件次品率是否超过标称值。在这里，我们通过假设检验的方法来实现这一目标。假设检验的构建如下：

情况一：

零假设：零配件次品率超过标称值，则拒收这批零配件；

备择假设：零配件次品率不超过标称值，则接受这批零配件；

具体意义：在这种情况下，如果次品率超过标称值，则拒收这批零配件；否则，接受这批零配件。

情况二：

零假设：零配件次品率不超过标称值，则接收这批零配件；

备择假设：零配件次品率超过标称值，则接受这批零配件；

具体意义：在这种情况下，如果次品率不超过标称值，则接受这批零配件；否则，拒绝这批零配件。

5.1.2计算样本量

为了在尽量减少检测次数以减少检测成本和确保检测结果的正确性、可靠性的双重目标下找到最优的平衡点，我们需要对所需抽样的最小样本量进行计算。而该计算主要基于设定的显著性水平a和给定的次品率标称值，以及允许的误差范围E。因此，使用如下公式计算样本量n：

在其中：

：置信水平下对应的值；

：样品次品率的标称值；

：允许的误差范围,这里取第一类错误概率

现假设供应商提供的零配件总数为,在按上述公式计算完后，再次利用下述公式进行优化：

其中：为按第一个公式计算出的值。

5.1.3确定抽样检测方案

在得到了所需抽样的最少样本量后，我们选择简单随机抽样的方法，即每次通过蒙特卡洛模拟抽取5个样本，即统计每次抽取抽样完成后的总次品数，再根据次品率进行下一步的计算。此时样本次品率的估计结果值为：

在其中：

：当前一次抽样中的样本量

：在当前抽的n个样本中的次品个数

5.1.4序贯概率比检验模型

基于上述思路，我们可以建立一个序贯概率比检验模型来解决这个问题。在这里，SPRT模型的核心思想是在每次抽样检测后，计算出对应的似然比，并将其与预先设定计算好的上下限进行比较，以此来决定是否还要继续抽样和是否要做出接受/拒绝的决定。

在此模型中 似然比函数为：

但在实际计算中,为了方便计算,可以对似然比函数进行取对数处理

而在此公式中，我们用到的参数：

：可接受质量水平，在这里为供应商声称的标称值；

拒绝质量水平,在这里假设商家可以接受的值为0.15；

为抽样决策中当前已经抽取的样本个数；

当前抽取的样本个数中的次品个数，即值

上下限阈值为：

上限

下限

同样的,对上下限取对数处理

上限

下限

在此公式中，我们用到的参数：

为第一类犯错概率

为第二类犯错概率

在题目中由信度确定，在这里假设企业可接受的为0.10

此检验模型的检验规则为：

1. 若,则停止继续抽样，且拒绝这批零配件；
2. 若,则停止继续抽样，且接受这批零配件；
3. 若,则继续下一组抽样；

5.1.5动态抽样算法的具体步骤

基于上文建立的SPRT模型与抽样检测方法，我们需要设计一个具体的算法来实现此动态抽样决策。该算法的具体步骤为：

1. 初始化定值参数：，计算出和；
2. 利用已知参数计算出所需的最小样本量；
3. 进行动态抽样并更新次品量：进行一次随机抽样，并更新单次抽样不合格品数量和单次抽样次品率值；
4. 利用更新的次品量计算似然比；
5. 根据似然比的数值进行行为判断决策：
6. 在未达到终止条件前，不停重复3-5步骤，直至得到最终结果

5.1.7问题一求解结果的分析验证

在该类涉及统计意义中设计方案决策的问题中，完成模型建立和算法设计后，我们还需要对求解结果进行分析验证。主要包括检查是否符合预期的检验性能和效果，验证过程可能包括对实际样本数据的模拟测试，分析假设检验的实际效果，以及调整模型参数以优化检测方案。

上述动态抽样算法的优势在于并不一定要预先确定样本量，而是可以根据一次次简单随机抽样得到的结果动态调整决定是否还要继续抽取，该方法可以在一定程度上帮助得到在统计模型意义下所需的最小的样本量。

在此模型中算得两种情况得最小检测次数分别为为97和23。

此外，我们还进行了SPRT模型延伸理论中OC曲线和ASN曲线的模拟求解。因为OC函数可以描述在不同实际次品率下，接受批次的概率，其能够很好的帮助我们检验模拟过程中SPRT模型所得结果的可靠性。而ASN函数则可以得到在不同实际次品率下平均需要的样本量，其也是帮助我们检验所得结果可靠性的一大参考值。

具体算法检验实现结果如图1所示（0C函数图像）：

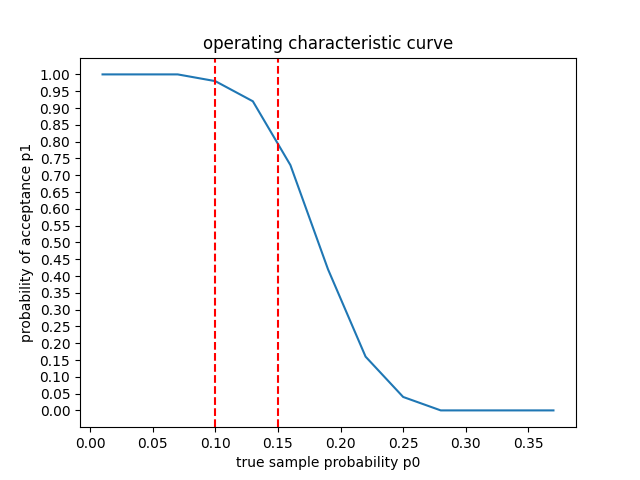


图1 OC曲线值

由图1可得：当真实次品率接近或低于0.10的情况下，接受概率高达以上，而在真是次品率大于0.15的情况下，接受概率急剧下降，甚至趋近于0，表示不接受。通过该OC曲线图可以得出：我们的SPRT模型在实际的抽样模拟过程中具有相当高的能力能够有效辨别配件中的真实次品率，从而做出正确决策。

ASN函数的图像如下图2所示;

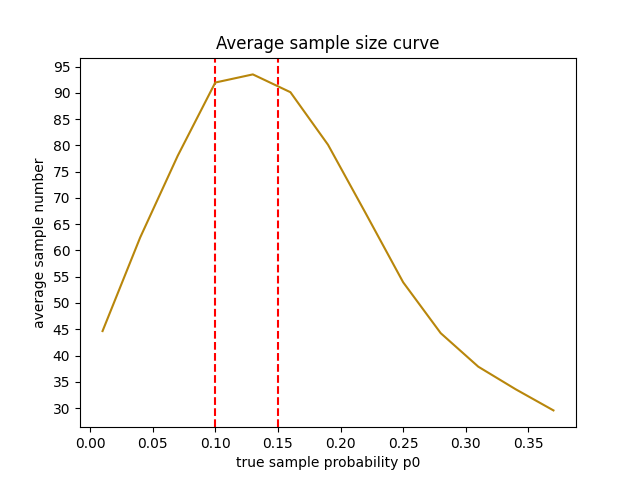


图2 ACN曲线值

由图2可得：当零配件中的真实次品率低于0.10或者高于0.15时所需的平均样本量较小，因为若真实次品率与标称值相差过大，则会在一定程度上减少检测次数，只需较少的检测次数，即可得到正确率足够高的决策。而当零配件中的真实次品率在0.10和0，15之间时，所需的平均样本量较大，这也对应着真实情况中，当真实次品率与标称值过于接近时，需要更多的检测样本，才能保障做出的决策的正确性。

问题二模型建立与求解

5.2.1决策组合情况

表1 不同决策组合表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 零件1是否检测 | 零件2是否检测 | 成品是否检测 | 次品是否拆解 |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 不检测  不检测  不检测  不检测  不检测  不检测  不检测  不检测  检测  检测  检测  检测  检测  检测  检测  检测 | 不检测  不检测  不检测  不检测  检测  检测  检测  检测  不检测  不检测  不检测  不检测  检测  检测  检测  检测 | 不检测  不检测  检测  检测  不检测  不检测  检测  检测  不检测  不检测  检测  检测  不检测  不检测  检测  检测 | 不拆解  拆解  不拆解  拆解  不拆解  拆解  不拆解  拆解  不拆解  拆解  不拆解  拆解  不拆解  拆解  不拆解  拆解 |

5.2.2 0/1规划下成本效益模型的建立

**1.总花费成本**

在其中：

：购买零配件1、2所花费的总成本

：包括对零配件1、2检测的总成本和对成品检测的成本

：对销售出去到达用户手中需要进行调换所产生的损失成本

：对检测和调换得到的不合格成品的拆解成本

**2.总购买成本**

购买成本即为购买零部件1和2原材料所要花费的成本，根据以下计算公式：

其中，n为假定的购买量，C1、C2分别为零配件1和2的的购买单价。在此问中，应保证第一次购入的的零配件1和零配件2的数量相同。

**3.零配件检测的决策**

现在企业将要决定是否对零配件1和2进行检测，若进行检测，则会产生检测费用，但得到的成品却不会因被检测的该零配件而发生装配失败，从而避免装配成本的损失和之后的拆解决策。若不进行检测，则会引发后续装配成本的损失和拆解费用的决策。检测成本的计算公式为：

其中，，用0、1的变量取值表示零配件1和2是否检测的状态决策，为上文中提及的假定购买量。

**4.装配成本的计算**

装配成品量就为在对问题一、二做完是否检测的决策后，能够装配出的成品的最大值。具体的计算思路为在做完零配件的检测决策后，无论得到的零配件是否掺杂次品，只需要对获得的两种零配件数量取最小值即可得到能够组装出的成品数量。因此装配成本即为所需装配的成品量与单件装配成本的乘积值。具体计算公式如下：

当然，在这里是假设在一种零配件检验、另一种零配件不检验的情况下，得到零配件的数量较大即未检测的一方，其与另一零配件组装时，其提供的配件数量中的次品率依然为原值。

**5.成品检测的决策**

由于在将得到的零配件进行组装后，除两种零配件均检测，排除零配件本身不合格造成成品组装失败的情况，在两种零配件均为合格的情况下进行组装依然会有一定的概率组装除出不合格的次品,即题中所给的次品率值的意义。因此，无论之前是否对零配件1和2进行检测，都要面对成品是否检测的决策问题，且其与零配件的检测决策问题完全独立。若对成品进行检测，则可以避免对用户发出不合格产品，从而避免调换损失和之后的拆解决策。若不进行检测，则要承担相应的费用成本。具体的计算公式如下：

=

其中T，用0、1的变量取值表示成品是否检测的状态决策

**6.不合格成本拆解的决策**

对于检测出的或者生产出未检测销售后，由于产品质量不合格而需要调换的不合格产品，企业可以选择拆解或者直接报废处理。若发生用户调换不合格产品，不仅需要承担调换损失，还需要归还该件产品的销售收益。若直接报废处理，将不会得到任何收益，但好处是可以省下这一批不合格产品的拆解费用。相应地，若选择拆解，需要承担拆解费用，但好处是可以得到一批新的零配件1和2，这些新配件可以当作节约购买成本，投入下一轮的生产中，以此获取利益。

但在此情况中，由于无论前两次检测决策是否决策都会产生次品，因为即使两种零配件都完好，依然有一定概率产生次品，且即使进行成品检测，也会得到不合格产品，只是并不会产生调换损失而已，因此永远会产生不合格产品。因此，该情况会不断循环下去，直到产生数量无限趋近于0或者拆解费用已经大于能获得的节约购买成本的期望。

这里考虑到零件1、2的购买成本与拆解费用的关系，以及简化模型与计算的需要，因此模型中只进行一次循环求解计算。

因此，调换成本的计算公式为：

其中是调换损失

对于拆解后的零件，只考虑进行一次循环，在循环时，由于拆解的零件是来自于不合格品，此时在再一次进行生产成品时，应对零件的次品率进行修正

得到拆解的零件数=

带入上面式子得到,，同时修正拆解后组装的成品次品率

得到总的拆解损失为

=++

其中为拆解损失。

7..**总销售收入**

根据真实的生产规律以及上述分析，容易得出总销售收入的值就为最后真实的成品销售量与单件成品的市场售价的乘积值，用公式表示即为：

**7.目标函数的建立**

在总的多阶段决策过程中，目标函数始终为利润最大化。

5.2.3问题二求解结果及分析

根据以上模型思路及代码实现，可以求得六种情况下每种决策的最终受益值，具体结果如图3：

图3 情况1各策略利润

由图3分析可得，在情况一中方案3编号的所获利益最大，即不检测零配件1和2，但对成品进行检测，同时对不合格成品进行拆解。

同理，可以由该方法计算出六种情况下各自决策下获得的最大利润，如下图4所示：

图4 不同情况的利润最值

由图4及分析可得：

1. 情况1、3的最佳决策为不检测零配件1和2，但对成品进行检测，同时对不合格成品进行拆解；
2. 情况2、4的最佳决策为检测零配件1和2，并且对成品进行检测以及对不合格成品进行拆解；
3. 情况5的最佳决策为不检测零配件1，但检测零配件2，以及对成品进行检测和对不合格成品进行拆解。
4. 情况6的最佳决策为不检测零配件1和2，对成品进行检测，以及不对不合格成品进行拆解；

问题三模型建立与求解

5.3.1策略分析

问题三本质是在问题二的基础上增加了工序与零配件的数量，从而变为更复杂的网络状的生产流程的决策问题。对于这种多配件、多工序的多阶段决策问题，问题二中的0/1规划算法求解答案的思路已经不再适用，因为单单题中给出的2道工序、8件零配件的问题，若要一一表示状态，已经高达8192种不同情形，虽然该情况下依旧可以进行可以利用问题二中的模型进行求解。但对于m道工序、n件零配件的情况来说，0/1规划算法的计算量更为浩大，因此要想求出有效解，则需要在更新成本效益模型的基础上，寻找新的最优算法来进行求解。

5.3.2 成本效益模型的更新

基本模型与问题二中所列数学表达式基本相同，只需将部分表达式中有关零件和工艺的具体变量，再次抽象得到通式即可。

**1.总成本的更新**

总成本的来源基本未发生变化，但每一部分都需进行抽象化，以此转换为多零配件、多工序的通用模型，具体表达式如下：

在其中：

：第i个零配件的购买成本；

：第i个零配件或半成品工序的检测成本；

：第i个零配件或半成品工序的装配成本；

：：对得到的所有不合格成品进行拆解的成本

**2.成品次品率的更新**

问题二中的的更新也为来源未发生变化，但需抽象化为多零配件检验后的结果，具体表达式如下：

在其中：表示第i个零配件的次品率；

**3.检测成本的更新**

在其中：

：第i个零配件的数量

：第i个零配件的单价检测成本

：第i个零配件产生的不合格成品所要调换需花费的费用

**4.装配成本的更新**

在其中：

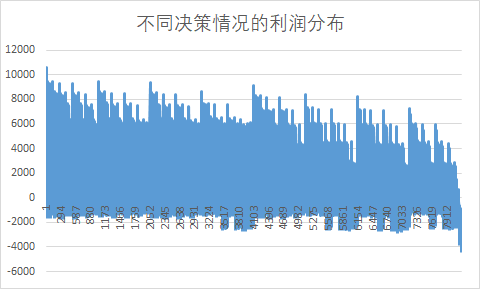
：最终得到的装配出的成品的数量；

：第i个零配件装配成半成品或成品花费的成本

：装配成半成品或成品的工序所产生的次品率

5.3.3成本效益模型的求解结果

因为上文中已经分析出在问题二的成本效益模型的基础下进行更新得到的新的成本效益模型，因此直接将题目中所给8个零配件、2道工序的生产情况用该新模型进行求解，得出答案即可。具体数值如图5所示：



**图5 新成本效益模型求解利润最值**

5.3.4模拟退火迭代算法的优化策略

**1.算法模型建立**

（1）初始化：随机生成一个二进制向量，表示所有决策变量的状态（检测/不检测为某一位上的0/1取值），并以此作为当前的最优解d（0），并且设置一个较高的初始温度T（0），以便后续模拟退火过程。

（2）进行迭代过程，对于每次迭代k，执行以下步骤：

1.生成领域解：随机选择当前解d上的一个位置i，将该位置上的决策值反转，即0/1转换，其他位置上则保持不变；

2.计算新解的目标函数值：利用更新的成本效益模型计算新解的利润；

3.判断是否接受新解即计算接受概率：若新解的目标函数值大于当前解的目标函数值，则接受新解；否则，将以一定的概率接受新解。

接受新解的概率值公式为：

4.更新温度：温度更新公式为：

在其中：

为冷却率,在这里取

为第k次迭代时的温度

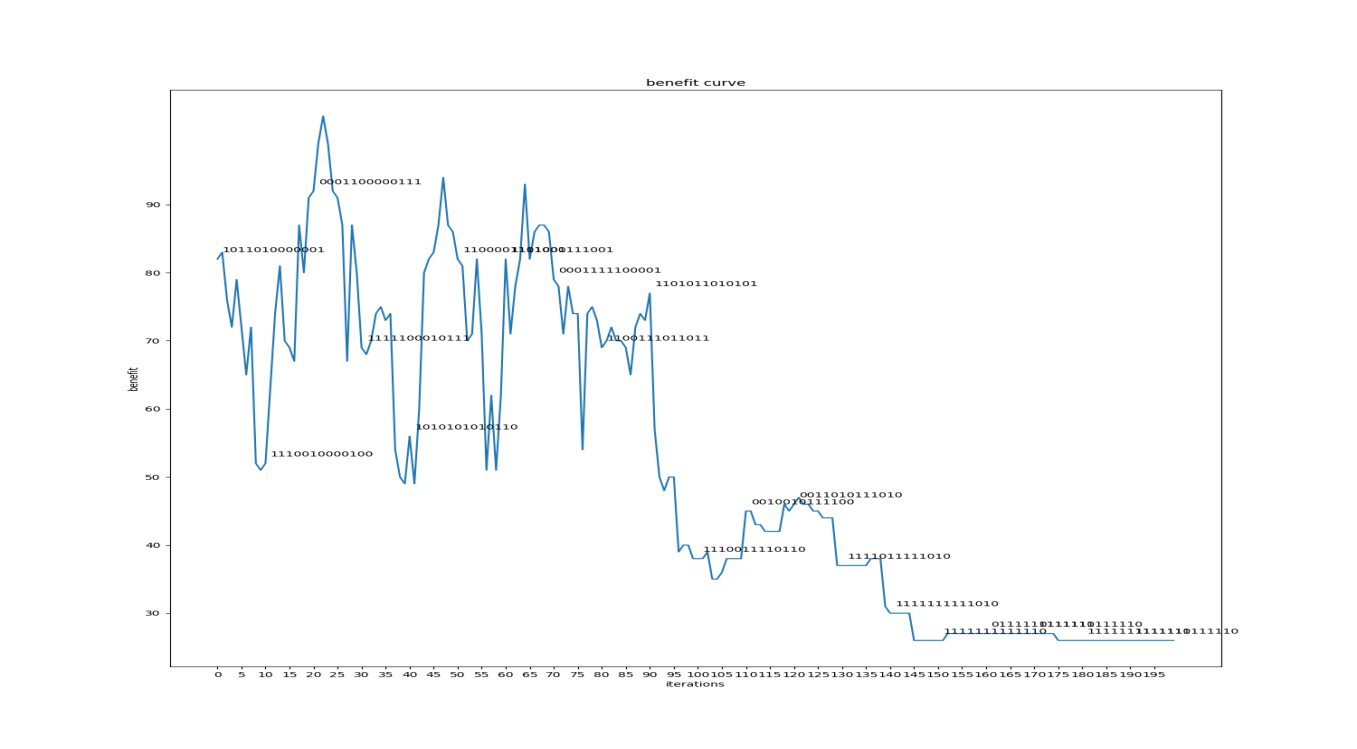
（4）迭代终止：一直迭代到温度下降到一定阈值或者达到预设的迭代次数，即停止迭代；

（5）得到最终的决策方案和对应的总利润值；

**2.算法实现过程**

首先，我们将每个决策变量——即零件、半成品和成品的检测与拆卸决策都视为独立的向量。而目标函数的目的是计算每个可能解的利润，并追求利润最大化。在算法预启动时，我们先随机确定一个初始解，并设定一个较高的起始温度。随后，我们进行循环的迭代过程：通过随机扰动当前的解以产生新的解。根据新解的目标函数值和当前的温度水平，结合新解的接受概率公式，决定每一步是否采纳该新解。随着迭代次数的增加，我们逐渐降低系统温度，促使算法向最优解逐步逼近。

**5.3.5模拟退火算法下的结果及分析**



**图6 迭代下模拟退火状态的改变**

由上图6分析可得：该算法具体的实现过程为：在模拟退火的过程中，随着不断迭代，产生新解，不断判断是否接受新解。并且在上图6中可以明显看到当前解的状态的波动变化，并由该算法运行得到最终的决策为

**零件1-8全都进行检测，检测半成品，不检测成品但拆解成品**

而在新成本效益模型中，该方案的目标函数值排第二，且与目标函数值最高的情况一（即对零配件、半成品、成品全检测，并且拆解成品）的目标函数值差值较小，因此，最终的决策即为该决策。

在零配件的数据量和工序量不断增大的趋势下，利用模拟退火优化算法能够有效优化做出决策的时间成本等，且正确率较高。

问题四问题建立与求解

5.4.1策略分析

问题4是在问题2和问题3的基础上进行的一个扩展。对于零配件、半成品和成品的次品率，我们没有一个固定的标称值，而是需要通过抽样检测的方法来估算次品率。这意味着这些零配件、半成品和成品的次品率会处于一个相对于问题2和问题3中标称值的范围内，而且我们抽取的次品率应符合相对标称值的正态分布。尽管模型的建立和求解与问题2和问题3基本相同，但我们需要在其基础上模拟抽样检测方法，以获得零配件、半成品和成品的次品率。这一变化引入了更接近实际生产场景的复杂性：在实际生产中，企业只能依赖有限的抽样检测数据。由于抽样检测结果具有波动性和不确定性，企业在制定决策时必须考虑次品率波动对生产成本和质量控制的潜在影响。

5.4.2 抽样检测得到次品率

**1.抽样检测的假设与方法**

从经验上来讲,抽样检测得到的零配件，半成品和成品的次品率应该是服从正态分布的，因此这样的假设是合理的,设抽取的样本量为,次品数为,则次品率估计为

且应该有

在其中：

为标称次品率

为方差

**2.使用蒙特卡罗方法进行方差估计**

方差的值是不确定的，因此为了尽可能准确地模拟出抽样得到的次品率结果，我们需要进行方差的估计。在这里，我们可以采用蒙特卡罗模拟方法来估算方差。蒙特卡罗模拟通过对数据进行大量的随机抽取，生成范围性波动的次品率数据。利用这些数据，我们能够计算出方差的估计值，从而确定一个合理的方差值。

5.4.3 问题2，3模型的优化与改进

**1.确定次品率**

在这里，我们可以采取问题1中提供的抽样检测方案去确定解决问题2中的零配件和成品的次品率。

**2.零配件的采购与筛选**

由于采用抽样检测来确定零配件的次品率会引入更多的不确定性。在这种情况下，原本基于给定标称值次品率进行最优决策的方法就可能不再适用。如果在进行抽样检测方案的检验后，发现次品率较高，我们就可以灵活地选择拒收这批零配件，以确保在进行最优决策时，次品率能够保持在一个可接受的范围内。

**3.装配环节的质量把控**

在先前问题2，3的求解中，我们基于一个样品率的值去求解最优的决策策略,但这样得到的决策策略可能并不是最优的,在现在可以通过抽样检测确定当前这批部件的次品率后,比如我们在中间求解的过程中,根据抽样检测确定的次品率来做出一些动态的调整策略,如果在一次抽样检测中,我们得到了一个较高的且不能被接受的次品率的数值,我们可以放弃使用这批零配件或者中间件,这样的灵活处理能够帮助企业得到更优的效益，在此基础上，最终最优的决策策略也有可能会发生变化。

**4.成品的检测与市场投放**

在问题2和问题3的求解中，我们基于一个固定的次品率值来制定最优决策策略。然而，这样得到的策略可能并不最优。现在，通过抽样检测确定当前批次部件的次品率后，我们可以在求解过程中根据检测结果进行动态调整。如果一次抽样检测结果显示次品率较高且不可接受，我们可以选择放弃使用这批零配件或中间件。这种灵活的处理方式能够帮助企业获得更优的效益，并可能导致最终最优决策策略的调整。

六、模型优缺点分析

6.1问题一模型优缺点分析

在问题一中，我们建立了一个假设检验和序贯概率比检验的模型，以此模型进行计算来决定是否接受供应商提供的某一批零配件。

该模型中的显著优点有：

（1）该思路简单清晰，运行效率较高：利用统计学中的二项分布模型结合公式进行计算，再进行简单的模拟随机抽样，利用序贯概率比检验，以此动态做出决策。思路简单清晰，且在现有数据量下运行效率极高。

（2）灵活性和准确度较高：模型中采用动态随机抽样的方法，不断进行序贯概率检验以辅助下一步的行动决策，能够有效保障模型中动态决策的调整和具体结果的正确性。

（3）适用性较高：因为其中利用到了统计学中的二项分布的模型并且由固定公式计算得到n，因此其适用度较高，可以适应各种不同规模的数据量。

该模型中的部分缺点有：

1. SPRT模型本身的局限性：在某些极端情况下，计算出的样本量过高，导致计算过程较长，计算过于复杂。
2. 抽样过程过于理想化：在模型中，我们假设了每次抽样都相互独立，即每次抽取都与之前的抽取无关，在这里，未考虑真实抽样过程中两次抽样存在一定关联规则的情况，导致最终结果出现错误。
3. 答案存在误差：公式中的参数值为近似值，所以求得的n也为近似值，可能会因此产生误差；

6.2问题二模型优缺点分析

在问题二中，我们建立了一个多阶段决策问题的解题策略，具体模型为在0/1规划下的成本效益模型。

该模型中的显著优点有：

1. 全面性：在该决策问题的分析中，依据零配件是否检测、成品是否检测、不合格成品是否拆解分为不同种情况，进行了较为全面的决策分析。
2. 适用性：在该模型中，我们运用了0/1规划的思想，通过不断的分析，将不同决策点的状态变量进行0/1规划，放入最终的数学表达式，且表达式中其他的参数也均为变量表示，可以根据实际情况调整参数，普遍适用性较强。

该模型中的部分缺点有：

只能计算数据规模较小的情况：当出现多个决策变量和大量的可能决策组合的模型时，计算过程可能会变成极其复杂和耗时，甚至出现错误。

6.3问题三模型优缺点分析

在问题三中，我们将问题二得到的成本效益模型进行了抽象化，得到适用性更强、更通用的新成本效益模型，之后又建立了以新成本效益模型为接受条件的参考值，建立了模拟退火迭代模型。

该模型中的显著优点有：

1. 通用性：该问题的模型求解思路中，先更新成本效益模型，对题目中所给的情况即数据量还可计算的情况进行求解，随后，考虑到模型的通用性，又在原模型的基础上，引入模拟退火迭代算法进行优化，使得该模型真正能够计算不同的数据情况，通用性较好；
2. 模型实用，计算效率高：由于该模型可以应对不同的数据情况，通用性较好，因此该模型极为实用，并且在模拟退火算法引入后，也使得该模型能够在可接受的时间内处理较大规模的问题，计算效率较高；
3. 计算结果较为准确：该模拟退火算法在生产过程中的迭代选择中动态地做出决策，并且采用概率接受原则，此模型允许在局部最优解附近进行搜索，也大大增加了搜索到全局最优解的概率；

该模型中的部分缺点有：

极端情况下，可能搜索到错误的全局最优解：即虽然该模拟退火算法的采用动态决策过程以及概率接受原则，大大提高了准确性，但还存在可能受到初始温度、冷却率等参数的影响，导致在极端情况下取到错误的决策结果；

6.4问题四模型优缺点分析

问题四通过题意得出该问题要求在次品率波动的情况下，做出较为稳健的决策。

该模型思路中的显著优点有：

1. 真实反映生产流程：引入抽样检测的不确定性和数值的的区间波动性，更加贴近真实生活，具有实用性；
2. 灵活性：在蒙特卡洛模拟随机抽样的过程中，能够根据每一次的实际检测结果动态调整决策，大大提高了决策的灵活性；
3. 精度较高：利用蒙特卡罗方法帮助准确估计方差，从而得到次品率大致的波动区间，极大地改进了次品率估算的精度；

该模型思路中的部分缺点：

虽然引入动态调整和蒙特卡洛模拟方法能够增加计算的精确度，但也增加了模型的计算复杂度和实现难度。

参考文献

[1] 黄文健,黄瑾珉,曹承昊.序贯概率比检验在故障诊断中的应用[J].化学工程与装备,2018,(01):19-22.DOI:10.19566/j.cnki.cn35-1285/tq.2018.01.005.

[2] 李俞利,胡宏昌.序贯L\_q似然比型检验及其应用[J].湖北师范大学学报(自然科学版),2022,42(04):1-5+27.

[3] 凌乔楠.模型选择后多个预测值同时置信区间的有效推断[D].华东师范大学,2020.DOI:10.27149/d.cnki.ghdsu.2020.000689.

[4] 元一平.基于改进负荷削减与序贯链条抽样的电网可靠性评估研究[D].重庆大学,2017.

[5] 杨延璞,雷紫荆,兰晨昕,等.融合贝叶斯网络与前景理论的产品工业设计多阶段决策方法[J].图学学报,2022,43(03):537-547.

[6] 薄胜,李媛,刘海伦.基于马尔可夫决策的钢铁产成品订单分配模型研究[J].河北省科学院学报,2024,41(02):29-36.DOI:10.16191/j.cnki.hbkx.2024.02.005.

[7] 雷霆,朱承,张维明.基于马尔科夫决策的目标选择策略[J].国防科技大学学报,2014,36(02):161-167.

[8] 罗艺,江凌云.基于马尔科夫决策过程的服务迁移策略[J].计算机工程与设计,2022,43(11):3015-3021.DOI:10.16208/j.issn1000-7024.2022.11.003.

[9] 温建博.基于多目标优化的UAV辅助无线可充电传感网络充电路径规划研究[D].吉林大学,2024.DOI:10.27162/d.cnki.gjlin.2024.001789.

[10] 胡鸿鹏,蒋水华,陈东,等.基于改进贝叶斯更新方法的边坡参数概率反分析及可靠度评估[J].岩土力学,2024,45(03):835-845.DOI:10.16285/j.rsm.2023.0485.

附录

|  |  |  |
| --- | --- | --- |
| 文件名 |  | 类型 |
| 问题1.py |  | python代码文件 |
| 问题1（2）.py |  | python代码文件 |
| 问题2代码.py |  | python代码文件 |
| 问题3.1模拟退火.py |  | python代码文件 |
| 问题3.2代码.py |  | python代码文件 |
| 问题4代码.py |  | python代码文件 |
| 问题二策略结果.xlsx |  | 数据分析结果文件 |
| 问题三结果.xlsx |  | 数据分析结果文件 |
| 问题4.1结果.xlsx |  | 数据分析结果文件 |
| 问题4(2)策略结果.xlsx |  | 数据分析结果文件 |

问题1代码：

|  |
| --- |
| *import* numpy *as* np  *from* scipy.stats *import* binomtest  *import* matplotlib.pyplot *as* plt  p0 = 0.1  p1 = 0.15  a = 0.05  *# 不妨认为第二类错误概率β与α相同*  b = 0.10  *# 似然函数的边界*  ceil, floor = np.log((1 - b) / a), np.log(b / (1 - a))  *# 定义似然函数(对数化)*  def L(*x*, *n*):  *return* *x* \* np.log(p1 / p0) + (*n* - *x*) \* np.log((1 - p1) / (1 - p0))  total = 10000  Z\_95 = 1.65  *# 使用无限公式计算的*  n1 = int(((Z\_95 \*\* 2) \* p0 \* (1 - p0)) / (a \*\* 2))  *# 有限调整*  n2 = int(n1 / (1 + (n1 - 1) / total))  print("无限公式n =", n1, "有限公式n =", n2)  def extract(*cnt*: int) -> int:      c0 = 0  *for* i *in* range(*cnt*):          sec = np.random.randint(0, data.size)  *if* data[sec] == 0:              c0 += 1  *return* c0  def main() -> (bool,int):  *# 定义单次抽取的步长*      dc = 5      c0, cnt = 0, 0  *if* io:          print(f"似然边界为{floor} ~ {ceil}")  *while* cnt + dc <= 9999999999:          c0 += extract(dc)          cnt += dc          lv = L(c0, cnt)  *if* io:              print(f"{cnt}里有次品{c0},样本次品率{c0 / cnt},似然值为{lv}")  *if* lv <= floor:  *if* io:                  print(f"抽{cnt},次品{c0},接受这批零件")  *return* True,cnt  *elif* lv >= ceil:  *if* io:                  print(f"抽{cnt},次品{c0},拒绝这批零件")  *return* False,cnt  *if* cnt < n2:          c0 += extract(n2 - cnt)          cnt = n2      p = c0 / cnt      res = binomtest(c0, cnt, p0, *alternative*="less")      ret = False  *if* res.pvalue > a:  *if* io:              print("抽完，但是接受这批零件")          ret = True,cnt  *else*:  *if* io:              print("抽完，但是拒绝这批零件")  *if* io:          print(f"{cnt}里有次品{c0},样本次品率{p},可能性概率为{res.pvalue}")  *return* ret,cnt  *if* \_\_name\_\_ == "\_\_main\_\_":      pz = 0.10      pz\_start = 0.01      pz\_end = 0.40      OCx = []      OCy = []      ASNx = []      ASNy = []      io = False      pi = pz\_start;  *while* pi < pz\_end:          data = np.array([0] \* int(pi \* total) + [1] \* int((1 - pi) \* total))          np.random.shuffle(data)          OCx.append(pi)          ASNx.append(pi)          ok = 0          s = 0  *for* i *in* range(100):              flag,cnt = main()  *if* flag:                  ok += 1              s += cnt          OCy.append(ok / 100)          ASNy.append(s / 100)          pi += 0.03      plt.title("operating characteristic curve")      plt.xlabel("true sample probability p0")      plt.ylabel("probability of acceptance p1")      plt.yticks(*ticks* = np.arange(0, 1 + 0.05, 0.05))      plt.plot(OCx, OCy)      plt.axvline(p0, *color* = 'r', *linestyle* = "--")      plt.axvline(p1, *color* = 'r', *linestyle* = "--")      plt.figure()      plt.title("Average sample size curve")      plt.xlabel("true sample probability p0")      plt.ylabel("average sample number")  *#plt.yticks([v for v in range(0,500,10)])*      plt.axvline(p0, *color* = 'r', *linestyle* = "--")      plt.axvline(p1, *color* = 'r', *linestyle* = "--")      plt.plot(ASNx, ASNy,*color* = "#B8860B")      plt.show() |

问题1（2）代码：

|  |
| --- |
| *import* numpy *as* np  *from* scipy.stats *import* binomtest  *import* matplotlib.pyplot *as* plt  p0 = 0.1  p1 = 0.15  a = 0.10  *# 不妨认为第二类错误概率β与α相同*  b = 0.10  *# 似然函数的边界*  ceil, floor = np.log((1 - b) / a), np.log(b / (1 - a))  *# 定义似然函数(对数化)*  def L(*x*, *n*):  *return* *x* \* np.log(p1 / p0) + (*n* - *x*) \* np.log((1 - p1) / (1 - p0))  total = 10000  Z\_95 = 1.65  *# 使用无限公式计算的*  n1 = int(((Z\_95 \*\* 2) \* p0 \* (1 - p0)) / (a \*\* 2))  *# 有限调整*  n2 = int(n1 / (1 + (n1 - 1) / total))  print("无限公式n =", n1, "有限个体优化计算后n =", n2)  def extract(*cnt*: int) -> int:      c0 = 0  *for* i *in* range(*cnt*):          sec = np.random.randint(0, data.size)  *if* data[sec] == 0:              c0 += 1  *return* c0  def main() -> (bool,int):  *# 定义单次抽取的步长*      dc = 5      c0, cnt = 0, 0  *if* io:          print(f"似然边界为{floor} ~ {ceil}")  *while* cnt + dc <= 9999999999:          c0 += extract(dc)          cnt += dc          lv = L(c0, cnt)  *if* io:              print(f"{cnt}里有次品{c0},样本次品率{c0 / cnt},似然值为{lv}")  *if* lv <= floor:  *if* io:                  print(f"抽{cnt},次品{c0},接受这批零件")  *return* True,cnt  *elif* lv >= ceil:  *if* io:                  print(f"抽{cnt},次品{c0},拒绝这批零件")  *return* False,cnt  *if* cnt < n2:          c0 += extract(n2 - cnt)          cnt = n2      p = c0 / cnt      res = binomtest(c0, cnt, p0, *alternative*="less")      ret = False  *if* res.pvalue > a:  *if* io:              print("抽完，但是接受这批零件")          ret = True,cnt  *else*:  *if* io:              print("抽完，但是拒绝这批零件")  *if* io:          print(f"{cnt}里有次品{c0},样本次品率{p},可能性概率为{res.pvalue}")  *return* ret,cnt  *if* \_\_name\_\_ == "\_\_main\_\_":      pz = 0.10      pz\_start = 0.01      pz\_end = 0.40      OCx = []      OCy = []      ASNx = []      ASNy = []      io = False      pi = pz\_start;  *while* pi < pz\_end:          data = np.array([0] \* int(pi \* total) + [1] \* int((1 - pi) \* total))          np.random.shuffle(data)          OCx.append(pi)          ASNx.append(pi)          ok = 0          s = 0  *for* i *in* range(100):              flag,cnt = main()  *if* flag:                  ok += 1              s += cnt          OCy.append(ok / 100)          ASNy.append(s / 100)          pi += 0.03      plt.title("operating characteristic curve")      plt.xlabel("true sample probability p0")      plt.ylabel("probability of acceptance p1")      plt.yticks(*ticks* = np.arange(0, 1 + 0.05, 0.05))      plt.plot(OCx, OCy)      plt.axvline(p0, *color* = 'r', *linestyle* = "--")      plt.axvline(p1, *color* = 'r', *linestyle* = "--")      plt.figure()      plt.title("Average sample size curve")      plt.xlabel("true sample probability p0")      plt.ylabel("average sample number")  *#plt.yticks([v for v in range(0,500,10)])*      plt.axvline(p0, *color* = 'r', *linestyle* = "--")      plt.axvline(p1, *color* = 'r', *linestyle* = "--")      plt.plot(ASNx, ASNy,*color* = "#B8860B")      plt.show() |

问题2代码：

|  |
| --- |
| *import* numpy *as* np  class Package:      def \_\_init\_\_(*self*, *bad*, *cost*, *dect*):  *self*.bad = *bad*  *self*.cost = *cost*  *self*.dect = *dect*  class Result:      def \_\_init\_\_(*self*, *bad*, *cost*, *dect*, *sell*):  *self*.bad = *bad*  *self*.cost = *cost*  *self*.dect = *dect*  *self*.sell = *sell*  def lowestcost(*cases*, *z1*, *z2*, *z3*, *r*, *s*):      costs = float('inf')      choose = -1  *for* i *in* range(16):          x1 = Package(*z1*.bad, *z1*.cost, *z1*.dect)          x2 = Package(*z2*.bad, *z2*.cost, *z2*.dect)          x3 = Result(*z3*.bad, *z3*.cost, *z3*.dect, *z3*.sell)          c1, c2, c3, c4 = *cases*[i]          n = 100  *#零件的成本*          costpackage = (x1.cost + x2.cost) \* n          nr = min(n \* (1 - (1 - c1) \* x1.bad), n \* (1 - (1 - c2) \* x2.bad))          nr0 = nr  *#检测费用与组装费用*          costdect = (x1.dect \* c1 + x2.dect \* c2) \* n + nr \* x3.dect          costcombin = nr \* x3.cost          x3.bad = 1 - (1 - x3.bad) \* (1 - (1 - c1) \* x1.bad) \* (1 - (1 - c2) \* x2.bad)  *#调换损失*          costchange = *r* \* nr \* x3.bad \* (1 - c3)          nr0 -= nr \* x3.bad \* (1 - c3)  *#拆解费用*          costapart = c4 \* nr \* x3.bad \* *s*          x1.bad = x1.bad / x3.bad          x2.bad = x2.bad / x3.bad          n = c4 \* nr \* x3.bad          nr = min(n \* (1 - (1 - c1) \* x1.bad), n \* (1 - (1 - c2) \* x2.bad))          x3.bad = 1 - (1 - x3.bad) \* (1 - (1 - c1) \* x1.bad) \* (1 - (1 - c2) \* x2.bad)  *#拆解后再次进行加工引起的成本*          costapart += (x1.dect \* c1 + x2.dect \* c2) \* n + nr \* x3.dect + nr \* x3.cost + *r* \* nr \* x3.bad \* (1 - c3)          nr0 -= nr \* x3.bad \* (1 - c3)          nr0 += nr  *#总成本减去总营销额*          costall = costpackage + costdect + costcombin + costchange + costapart - nr0 \* x3.sell          print(costall, *end*=' ')  *if* costall < costs:              costs = costall              choose = i      print()  *return* choose  cases = [[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 0, 1], [0, 1, 1, 0], [0, 1, 1, 1], [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0], [1, 0, 1, 1], [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 0], [1, 1, 1, 1]]  ans = []  y1 = Package(0.1, 4, 2)  y2 = Package(0.1, 18, 3)  y3 = Result(0.1, 6, 3, 56)  r = 6  s = 5  ans.append(lowestcost(cases, y1, y2, y3, r, s))  y1 = Package(0.2, 4, 2)  y2 = Package(0.2, 18, 3)  y3 = Result(0.2, 6, 3, 56)  ans.append(lowestcost(cases, y1, y2, y3, r, s))  y1 = Package(0.1, 4, 2)  y2 = Package(0.1, 18, 3)  y3 = Result(0.1, 6, 3, 56)  r = 30  ans.append(lowestcost(cases, y1, y2, y3, r, s))  y1 = Package(0.2, 4, 1)  y2 = Package(0.2, 18, 1)  y3 = Result(0.2, 6, 2, 56)  ans.append(lowestcost(cases, y1, y2, y3, r, s))  y1 = Package(0.1, 4, 8)  y2 = Package(0.2, 18, 1)  y3 = Result(0.1, 6, 2, 56)  r = 10  ans.append(lowestcost(cases, y1, y2, y3, r, s))  y1 = Package(0.05, 4, 2)  y2 = Package(0.05, 18, 3)  y3 = Result(0.05, 6, 3, 56)  r = 10  s = 40  ans.append(lowestcost(cases, y1, y2, y3, r, s))  *#对每种情况得出的方案编号*  *for* i *in* range(len(ans)):      print(ans[i], *end*=' ') |

问题三代码：

|  |
| --- |
| *import* random  *import* math  *import* matplotlib.pyplot *as* plt  *# 定义组件的成本和缺陷率*  comps = [      {"defect\_rate": 0.10, "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 8, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 12, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 8, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 12, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": 0.10, "purchase\_cost": 8, "inspection\_cost": 1}  ]  *# 定义半成品的成本和缺陷率*  semip = [      {"defect\_rate": 0.10, "assembly\_cost": 8, "inspection\_cost": 4},      {"defect\_rate": 0.10, "assembly\_cost": 8, "inspection\_cost": 4},      {"defect\_rate": 0.10, "assembly\_cost": 8, "inspection\_cost": 4}  ]  *# 定义最终产品的成本和市场信息*  final = {      "defect\_rate": 0.10, "assembly\_cost": 8, "inspection\_cost": 6,      "market\_price": 200, "replacement\_cost": 40, "disassembly\_cost": 20  }  *# 计算总成本的函数*  def cacc(*comps*, *semip*, *final*, *dec*):      comp\_d = *dec*[:8]      semip\_d = *dec*[8:11]      final\_d = *dec*[11]      dis\_d = *dec*[12]      component\_c = sum(d \* comp["inspection\_cost"] + (1 - d) \* comp["purchase\_cost"] *for* d, comp *in* zip(comp\_d, *comps*))      semip\_c = sum(d \* semi["inspection\_cost"] + (1 - d] \* semi["assembly\_cost"] *for* d, semi *in* zip(semip\_d, semip))  *final\_c* = final\_d \* *final*["inspection\_cost"] + (1 - final\_d) \* *final*["assembly\_cost"]  *dis\_c* = dis\_d \* *final*["disassembly\_cost"]  *total\_cost* = component\_c + semip\_c + final\_c + dis\_c  *return* total\_cost  *# 初始化解的函数*  *def* init():  *return* [random.choice([0, 1]) *for* \_ *in* range(13)]  *# 生成邻域解的函数*  *def* shllf(*solution*):  *new* = *solution*[:]  *index* = random.randint(0, len(*solution*) - 1)      new[index] = 1 - new[index]  *return* new  *# 将解转换为字符串的函数*  *def* to\_str(*a*) -> str:      s: *str* = ""  *for* v *in* *a*:          s += str(v)  *return* s  *# 模拟退火算法的主函数*  *def* SA(*comps*, *semip*, *final*, *init\_tmp*, *cooling\_rate*, *mit*,*cx*,*cy*,*ex*,*ey*):  *solution\_c* = init()  *cost\_c* = cacc(*comps*, *semip*, *final*, solution\_c)  *solution\_b* = solution\_c  *cost\_b* = cost\_c  *temp* = *init\_tmp*  *for* i *in* range(*mit*):  *new\_solution* = shllf(solution\_c)  *new\_cost* = cacc(*comps*, *semip*, *final*, new\_solution)  *if* new\_cost < cost\_c *or* random.random() < math.exp((cost\_c - new\_cost) / temp):  *solution\_c* = new\_solution  *cost\_c* = new\_cost  *if* new\_cost < cost\_b:  *solution\_b* = new\_solution  *cost\_b* = new\_cost          temp \*= *cooling\_rate*  *cx*.append(i)  *cy*.append(cost\_c)  *ex*.append(i)  *ey*.append(to\_str(solution\_c))  *if* i % 30 == 0:              plt.annotate(to\_str(solution\_c), *xy*=(i, cost\_c), *xytext*=(i + 1, cost\_c + 1))              print(f"Current Solution: {solution\_c}")              print(f"Iteration {i}: Current Cost = {cost\_c}")  *return* solution\_b, cost\_b  *# 主函数*  *def* main():      init\_tmp: *float* = 1000.      cooling\_rate: *float* = 0.95      mit: *int* = 200 + 1  *cx* = []  *cy* = []  *ex* = []  *ey* = []      plt.figure(*figsize*=(15, 15))      plt.title("benefit curve")      plt.xlabel("iterations")      plt.ylabel("benefit")      solution\_b, *cost\_b* = SA(comps, semip, final, init\_tmp, cooling\_rate, mit, cx, cy, ex, ey)      print(f"Best Solution: {solution\_b}")      print(f"Minimum Cost: {cost\_b}")      plt.xticks([v *for* v *in* range(0, mit + 5, 5)])      plt.yticks([v *for* v *in* range(0, 100 + 10, 10)])      plt.plot(cx, cy)      plt.figure(*figsize*=(15, 15))      plt.title("state iteration curve")      plt.xlabel("iterations")      plt.ylabel("state")      plt.xticks([v *for* v *in* range(0, mit + 5, 5)])      plt.plot(ex, ey, *color*="#B8860B")      plt.show()  *if* \_\_name\_\_ == "\_\_main\_\_":      main() |

问题四代码：

|  |
| --- |
| *import* random  *import* math  *import* matplotlib.pyplot *as* plt  *import* numpy *as* np  *import* pandas *as* pd  p = 0.1  z = 1.65  n = 97  CI\_u = p + z \* np.sqrt(p \* (1 - p) / n)  rd = np.random.normal  sigma = 0.015  rdv = lambda : rd(p, sigma)  comps = [      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 8, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 12, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 8, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 12, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 2, "inspection\_cost": 1},      {"defect\_rate": rd(p ,sigma), "purchase\_cost": 8, "inspection\_cost": 1}  ]  semip = [      {"defect\_rate": rd(p ,sigma), "assembly\_cost": 8, "inspection\_cost": 4},      {"defect\_rate": rd(p ,sigma), "assembly\_cost": 8, "inspection\_cost": 4},      {"defect\_rate": rd(p ,sigma), "assembly\_cost": 8, "inspection\_cost": 4}  ]  final = {      "defect\_rate": rd(p ,sigma), "assembly\_cost": 8, "inspection\_cost": 6,      "market\_price": 200, "replacement\_cost": 40, "disassembly\_cost": 20  }  def cacc(*comps*, *semip*, *final*, *dec*):      comp\_d = *dec*[:8]      semip\_d = *dec*[8:11]      final\_d = *dec*[11]      dis\_d = *dec*[12]      component\_c = sum(d \* comp["inspection\_cost"] + (1 - d) \* comp["purchase\_cost"] *for* d, comp *in* zip(comp\_d, *comps*))      semip\_c = sum(d \* semi["inspection\_cost"] + (1 - d) \* semi["assembly\_cost"] *for* d, semi *in* zip(semip\_d, *semip*))      final\_c = final\_d \* *final*["inspection\_cost"] + (1 - final\_d) \* *final*["assembly\_cost"]      dis\_c = dis\_d \* *final*["disassembly\_cost"]      total\_cost = component\_c + semip\_c + final\_c + dis\_c  *return* total\_cost  def init():  *return* [random.choice([0, 1]) *for* \_ *in* range(13)]  *# 生成邻域解*  def shllf(*solution*):      new = *solution*[:]      index = random.randint(0, len(*solution*) - 1)      new[index] = 1 - new[index]  *return* new  def to\_str(*a*) -> str:      s: str = ""  *for* v *in* *a*:          s += str(v)  *return* s  dic = {  }  x = []  y = []  def SA(*comps*, *semip*, *final*, *init\_tmp*, *cooling\_rate*, *mit* ,*cx* ,*cy* ,*ex* ,*ey*):      solution\_c = init()      cost\_c = cacc(*comps*, *semip*, *final*, solution\_c)      solution\_b = solution\_c      cost\_b = cost\_c      temp = *init\_tmp*  *for* i *in* range(*mit*):          new\_solution = shllf(solution\_c)          new\_cost = cacc(*comps*, *semip*, *final*, new\_solution)  *if* new\_cost < cost\_c or random.random() < math.exp((cost\_c - new\_cost) / temp):              solution\_c = new\_solution              cost\_c = new\_cost  *if* new\_cost < cost\_b:              solution\_b = new\_solution              cost\_b = new\_cost          temp \*= *cooling\_rate*  *cx*.append(i)  *cy*.append(cost\_c)  *ex*.append(i)  *ey*.append(to\_str(solution\_c))          x.append(int(to\_str(solution\_c),2))          y.append(cost\_c)  *if* i % 30 == 0:              plt.annotate(to\_str(solution\_c) ,*xy* = (i ,cost\_c) ,*xytext* = (i + 1 ,cost\_c + 1),                           )              print(f"Current Solution: {solution\_c}")              print(f"Iteration {i}: Current Cost = {cost\_c}")  *return* solution\_b, cost\_b  def main():      init\_tmp: float = 1000.      cooling\_rate: float = 0.95      mit: int = 200 + 1      cx = []      cy = []      ex = []      ey = []      plt.figure(*figsize* = (15 ,15))      plt.title("benefit curve")      plt.xlabel("iterations")      plt.ylabel("benefit")      solution\_b, cost\_b = SA(comps, semip, final, init\_tmp, cooling\_rate, mit ,cx ,cy ,ex ,ey)      print(f"Best Solution: {solution\_b}")      print(f"Minimum Cost: {cost\_b}")      plt.xticks([v *for* v *in* range(0 ,mit + 5 ,5)])      plt.yticks([v *for* v *in* range(0 ,100 + 10 ,10)])      plt.plot(cx ,cy)      plt.show()      dic["决策策略"] = x      dic["经济效益"] = y      df = pd.DataFrame.from\_dict(dic)      df.to\_excel("问题4(2)策略结果.xlsx",*index* = False)  *if* \_\_name\_\_ == "\_\_main\_\_":      main() |