第十届湖南省研究生数学建模竞赛承诺书

我们仔细阅读了湖南省高校研究生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们完全清楚，在竞赛中必须合法合规地使用文献资料和软件工具，不能有任何侵犯知识产权的行为。否则我们将失去评奖资格，并可能受到严肃处理。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们授权湖南省研究生数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

我们参赛选择的题号是（从组委会提供的赛题中选择一项填写）：A

我们的参赛编号（请填写完整参赛编号）：202518001013

所属学校（请填写完整的全名）：国防科技大学

参赛队员（打印后签名）：1. 佟国华

2. 徐佩茹

3. 张晓诚

指导教师或指导教师组负责人 (打印后签名)：

日期：2025 年 8 月 26 日

（请勿改动此页内容和格式。以上内容请仔细核对，如填写错误，论文可能被取消评奖资格。）

第十届湖南省研究生数学建模竞赛

题　目：面向大型装置测试的动态任务规划及敏感性分析

摘要：本文面向大型装置测试任务规划问题，以提高测试效率与降低漏判风险为目标，综合考虑子系统故障、设备故障、测试出现差错等异常风险，通过建立概率模型、离散事件仿真模型及优化算法，对多阶段测试过程进行动态任务规划及敏感性分析。

针对问题一，构建了基于贝叶斯理论概率计算模型。将大型装置子系统 A、B、C 的测试视为并行事件，并与综合测试 E 串联，利用贝叶斯概率思想，推导出综合测试发现问题概率及各责任比例参数 λ_1、λ_2、λ_3、λ_4 的数学表达式，并计算出具体比例：λ_1=0.1724，λ_2=0.2759，λ_3=0.0919，λ_4=0.4598，并进一步分析了所求各部分所占比例的合理性。

针对问题二，构建了基于离散事件模拟和蒙特卡洛的单分队任务规划模型。采用离散事件建模方法描述测试流程，综合考虑设备故障、子系统缺陷及测手差错等随机因素，并通过蒙特卡洛模拟多次仿真获得稳定统计结果，在此基础上，以设备更换时间为决策变量，以平均任务完成天数与总漏判概率为优化目标，构建其目标函数，通过循环遍历法得到更换设备时间为 230h 时的指标最优，将其作为工作计划，最终得到任务平均完成天数为 31、通过测试装置数量为 95，总漏判概率为 0.0158%，总误判概率为 1.3024% 及各小组有效工作时间比分别为 0.5514、0.4399、0.5517 和 0.6019。进一步论证分析了模型中所用蒙特卡洛算法的鲁棒性和稳定性，并将提前更换设备与不提前更换下的各类指标进行对比，验证本工作计划的优越效果。

针对问题三，构建了融合遗传算法的双分队任务规划模型。在问题二单分队模型基础上引入双分队动态协作倒班机制，以班次时长 K 为决策变量，建立双班制测试优化模型。同时，引入遗传算法对 K 值进行寻优，以获得最佳班次时长 K 为 10 小时，各项统计指标结果任务中完成平均天数为 22、通过测试的装置的平均数目为 96，总漏判概率为 0.03%、总误判概率为 1.25%，A、B、C、E 专业测试组的有效工作时间比分别为 0.7853、0.623、0.7813、0.8545。

针对问题四，基于 Sobol 算法和随机森林对问题三构建的双分队任务规划模型进行多因素敏感性分析，考虑班次时长、子系统故障率、测手差错率、设备更换时间这 4 个因素对测试任务平均完成天数的综合影响程度。分析结果表明，班次时长对测试任务平均完成天数指标具有显著影响。对此结果，进一步对问题三模型遗传算法中适应度函数的平均完成天数的加权系数占比进行单变量敏感性分析，探索最小化平均完成天数和最小化漏判概率的不同重视程度对于最终任务完成天数的影响。据此建议主管部门优先优化排班制度，并在保证测试质量的前提下，通过加强设备维护与人员培训，实现测试效率与可靠性的协同提升。

关键词：测试任务规划 条件概率 蒙特卡洛模拟 离散事件仿真 遗传算法 敏感性分析

目录

1 问题综述

1.1 问题背景

任务规划与资源调度作为运筹优化领域的核心研究方向，旨在通过算法和模型对有限资源下的任务序列进行优化安排，以实现特定目标如时间最短、成本最低、效率最高。随着计算机科学、优化算法和人工智能技术的发展，任务规划逐渐从简单的规则调度演变为融合多目标优化、机器学习、实时响应等复杂技术的综合性学科。

在工业化与信息化的浪潮中，任务规划的重要性日益凸显。生产系统、物流网络、军事部署和民用服务等领域均面临资源有限而需求复杂的挑战，如何高效分配时间、人力、设备及能源等资源，成为提升整体效能的关键。此外，随着无人机、机器人等智能设备的普及，任务规划进一步扩展到动态、不确定环境下的实时决策问题，推动了其在理论和应用层面的快速发展。任务规划与资源调度在各大领域都有重要研究。例如，对于无人机任务规划，无人机常需执行多目标、多约束的任务如区域覆盖、目标搜索、物资投递。对于资源调度，在云计算、分布式计算和通信网络中，任务规划用于优化计算资源、带宽和存储空间的分配。作业车间调度是经典的任务规划问题，这也是本文所关注的研究。车间调度问题是一个决策过程，它指的是在给定的时间段内为加工制造任务分配特定的资源，涉及多个工件在多台机器上的加工顺序安排，使得一个或多个目标达到最优。

对于作业车间调度和任务规划的研究，仍面临诸多挑战，任务规划与调度问题充斥着各种复杂的约束条件，包括工艺约束、资源约束、时间约束等。算法必须在满足所有这些“硬约束”的前提下寻找最优解。实际生产应用中通常为多目标优化问题，现实世界中的目标往往是多元且相互冲突的。常见的优化目标包括最小化最大完成时间、最小化总流程时间、最小化总设备空闲时间等。如何权衡质量与效率并找到最符合生产需求的解是一个巨大挑战。此外，现实车间充满了不确定性和动态扰动，例如，机器故障、紧急订单插入、工件交货期变更等，这比理想的静态调度要复杂得多。

为了解决这些问题，研究学者展开了广泛研究。陈勇[1]等人将模糊概念引入调度模型。尤一琛[2]等人构建了故障机器可恢复的动态柔性作业车间模型。Maroua[3]等人采用两阶段粒子群算法求解含机器故障的预测调度问题。李素粉[4]等人对于机器故障情况下的动态车间调度问题，采用平均、最大值、迟到方差和迟到作业比例来评价各种调度规则的性能，在比较研究中考虑了细分水平，车间负荷水平。采用异常事件驱动的滚动窗口机制和重新调度策略，在机器发生故障时实时更新原定进度。智能优化算法作为解决复杂调度问题的有力工具，近年来在作业车间调度领域展现出强大的应用潜力。例如，遗传算法、灰狼优化算法、粒子群等算法均被引入作业车间调度和任务规划的研究。

1.2 问题提出

本文所研究的大型装置测试任务规划问题，是一项复杂的系统工程，涉及多子系统协同检测与资源优化配置。本研究针对某大型装置（由 A、B、C 三个子系统组成）的批量测试任务，建立数学模型解决测试过程中的多目标优化问题。该问题具有以下典型特征。**（1）多阶段测试流程：**包含子系统独立测试（A、B、C）和综合测试（E）两个阶段，形成并行和串行混合的测试结构；**（2）随机性干扰因素：**系统测试时可能出现异常情形，包括设备故障（Y1）、子系统缺陷（Y2）、测试人员失误（Y3）以及综合测试异常（Y4）四类不确定性事件；**（3）资源约束条件：**测试大厅仅配备两个测试台，四个专业测试小组（A、B、C、E）共享有限工位；**（4）多目标优化需求：**需同时优化任务完成时间和漏判概率。

需要从考虑随机异常干扰、多目标规划、资源约束角度考虑，解决以下 4 个问题:

问题 1: 概率模型构建。重点在于量化综合测试对各子系统故障的检测能力。需要建立数学表达式来描述综合测试发现问题时各子系统所占的比例参数，这些参数需要满足概率分布的基本性质。通过历史数据可以获取各子系统固有故障率和测试人员差错率等先验信息，这些都将作为模型构建的输入参数。

问题 2: 单分队测试规划。针对单个测试分队承担批量测试任务的情景，要求在每日工作时间不超过十二小时的限制条件下，需要考虑测试装置的排序、测试台的分配、异常情况的处理策略等多方面因素。当中断发生时，相关测试必须重新开始的规则增加了调度复杂性。本质上是一个带随机扰动的资源约束项目调度问题。

问题 3: 双分队协同调度。问题三在测试任务中引入了资源扩展，采用两个分队轮班工作以加快测试进度。新增的决策变量是每个班次的最佳工作时长。两个分队共用同一套测试设备的设定带来了额外的协调挑战，需要设计合理的任务交接机制。这个问题将单分队调度扩展为更复杂的多模式资源调度问题。

问题 4：参数分析与改进建议。系统分析影响测试效率的关键因素，并基于分析结果提出改进建议。需要考察班次时长、设备故障率、人员操作水平等多种因素对整体测试周期的影响程度，识别系统中的瓶颈环节和潜在优化空间。

2 模型假设与符号说明

2.1 模型基本假设

对于问题一，假设如下：

(1)要计算综合测试测出系统有问题的概率表达式，则其前提是测试顺利完成，不考虑测试设备发生故障的情况，即不需要考虑 Y1 的异常情况。同时，问题一中明确指出综合测试测出各子系统存在问题的能力是相同的，并且综合测试测出的问题具有指向性。那么对于计算指向各子系统的比例 $\lambda_i, i=1,2,3,4$，我们仅需要考虑其子系统存在问题的情况。

对于问题二，假设如下：

(1)假设在装置系统测试完成后才知道通过与否，测试中无法因不通过而中断;

(2)假设完成 A、B、C 子系统测试后联接所需时间及重测间隙所用时间分别为 0.1h 和 0.5h;

(3)假设漏判概率与任务完成时长相互独立;

(4)假设工序测试因故中断不考虑不可抗因素;

(5)假设对于 1 个装置，A、B、C3 个子系统同时并行检测;

(6)假设各种异常事件的发生相互独立;

(7)假设题目中给出的所有概率均为已知且恒定不变的先验概率;

(8)假设综合测试测出有问题时只需要对综合测试进行重测，不用对子系统进行重测。

对于问题三，假设如下：

(1)假设分队 1 完成 K 小时工作后，分队 2 立即接替，忽略倒班时间;

(2)假设换班时正在进行的测试正常继续，不中断。

2.2 符号说明

本文定义了如下使用次数较多的符号，其余符号在使用时注明。

表1 符号说明

符号	含义	单位
P_A	子系统 A 本身有问题的概率	无
P_B	子系统 B 本身有问题的概率	无
P_C	子系统 C 本身有问题的概率	无
P_D	子系统 D 本身有问题的概率	无
P_L	总漏判概率	无
P_W	总误判概率	无
T	任务完成平均天数	天
S	通过测试的装置的平均数目	个
YXB	有效工作时间比	无
S_i	目标函数	无
d_i	第 i 次模拟的平均天数	天
m_i	第 i 次模拟的漏检率	无
R_i	第 i 次模拟的实验结果	无
$D_{\max}$	n 次实验 d_i 的最大值	天
$M_{\max}$	n 次实验 m_i 的最大值	无
w_d	d_i 的权重，0.7	无
w_m	m_i 的权重，03	无
S'	n 次模拟的所有目标函数值	无
h_w	每日工作时长	小时
P_{change}	更换设备的概率	无
D_e	装置列表	无
d_{e1}	装置 1	无
G	测试小组列表	无
W_i	测试台 i	无
M	测试设备列表	无
τ	事件触发时刻	小时
t	当前时间	小时
T_{in}	装置运入大厅时长	小时
$T_0(j)$	j 工序下设备重启调试时长	小时
$T_{tes}(j)$	j 工序下装置检测时长	小时
T_{out}	装置运出大厅时长	小时
ns_{ij}	i 装置 j 工序时的漏判数	台
nf_{ij}	i 装置 j 工序时的误判数	台
N_{miss}	漏判总数	台

符号	含义	单位
N_{fail}	误判总数	台
T_{all}	测试总时间	小时
r	蒙特卡洛算法产生的随机数	无
$fault_X$	发生异常情况	无
P_{Yi}	Y_i (i=1,2,3,4)异常情况发生的概率	无
T_{eqb}	当前设备的使用时长	小时
δ	循环遍历步长，10	无
T_{re}	重测次数	次
T_{all}	总的测试时长	小时
n	装置总数，100	台

3 问题一分析与模型建立

3.1 问题一分析

问题一要求我们计算综合测试测出系统有问题的概率表达式以及指向各子系统的比例 $\lambda_i, i=1,2,3,4$。如图 1 所示，综合测试“发现问题”的概率应包含两类情形：（1）系统本身有问题（可能源于已通过测试的子系统 A、B、C 的漏判，也可能是子系统 D 有问题），且综合测试 E 小组的测手正确地把问题测了出来；（2）系统本身没有任何问题（A、B、C、D 都正常），但综合测试 E 小组的测手出现误判（Y31），错误地报出“有问题”。

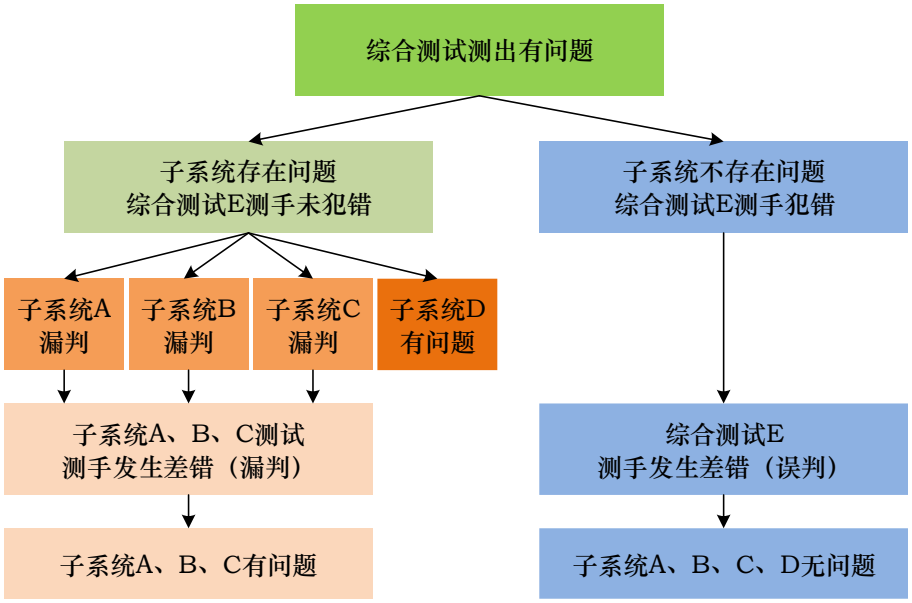


图1 问题一分析结构图

通过对问题一的分析，明确了需要计算综合测试测出问题的概率和计算出现该问题来源的比例参数（综合测试测出问题来自子系统 A、B、C、D 的概率）。因为问题一中明确指出，综合测试测出各子系统存在问题的能力是相同的，并且综合测试测出的问题具有指向性。那么对于计算指向各子系统的比例 $\lambda_i, i=1,2,3,4$，我们仅考虑子系统存在问题的情况。显然，这是一个概率问题，可以利用贝叶斯理论进行计算。因此，对于问题一，我们建立基于贝叶斯理论的概率计算模型。

3.2 基于贝叶斯理论的概率计算模型

3.2.1 贝叶斯理论简介

贝叶斯理论提供了一种更新概率的方法，通过结合新的证据或信息来修正对事件概率的信念。作为一种强大的工具，贝叶斯理论得到了广泛应用。贝叶斯理论是一种基于概率统计的推断方法，其核心思想是利用已有的先验信息和观测数据，通过贝叶斯公式更新概率分布，进而获得系统的后验概率分布，是概率论的重要内容。贝叶斯公式可以表示如下：

$$p\left(A|B\right)=\frac{p\left(B|A\right)p\left(A\right)}{p\left(B\right)} \tag{1}$$

其中，$p\left(A|B\right)$是事件 A 在给定 B 发生的情况下的后验概率，$p\left(B|A\right)$是在事件 A 发生的情况下，事件 B 发生的概率，也称为可能性。$p\left(A\right)$是事件 A 的先验概率，$p\left(B\right)$是事件 B 的先验概率。

全概率公式用于计算$p\left(B\right)$时对所有可能来源A_i求和，表示如下：

$$p\left(B\right)=\sum_i p\left(B|A_i\right)\cdot p\left(A_i\right) \tag{2}$$

3.2.2 基于贝叶斯理论的概率计算

（1）综合测试测出系统有问题的概率计算

定义综合测试测出系统有问题的概率为P_d，基于上文对问题一的分析可知：综合测试测出系统有问题由两个事件组成：A、子系统真的存在问题，B、综合测试测出有问题。易知：A 事件和 B 事件相互独立。记 A 事件发生的概率为P_{sys}，B 事件发生的概率为P_{Ed}根据贝叶斯理论得到：

$$P_d = P_{sys}\times P_{Ed} \tag{3}$$

首先分析 A 事件：子系统真的存在问题。子系统i有通过测试仍有问题的概率记为$P_{miss,i}$，测手i出现问题的概率记为$P_{i,false}$，测手i漏判的概率记为$P_{i,miss}$则有

$$\begin{aligned}
P_{miss,A} &= P_A\times P_{A,miss} = P_A\times P_{A,false}\times 50\% \\
P_{miss,B} &= P_B\times P_{B,miss} = P_B\times P_{B,false}\times 50\% \\
P_{miss,C} &= P_C\times P_{C,miss} = P_C\times P_{C,false}\times 50\%
\end{aligned} \tag{4}$$

综合检测前所有子系统不存在问题的概率记为P_{zer}，则有

$$\begin{aligned}
P_{sys} &= 1-P_{zer} \\
&= 1-\left(1-P_{miss,A}\right)\left(1-P_{miss,B}\right)\left(1-P_{miss,C}\right)\left(1-P_D\right)
\end{aligned} \tag{5}$$

略去高阶小量，得到

$$\begin{aligned}
P_{sys} &= 1-P_{zer} \\
&= 1-\left(1-P_{miss,A}\right)\left(1-P_{miss,B}\right)\left(1-P_{miss,C}\right)\left(1-P_D\right) \\
&= P_{miss,A}+P_{miss,B}+P_{miss,C}+P_D
\end{aligned} \tag{6}$$

其次分析 B 事件：综合测试测出有问题。分析可知

$$P_{Ed}=1-P_{E,miss}=1-P_{E,false}\times 50\% \tag{7}$$

（2）指向各子系统的比例计算

设

$$Q_1=P_{miss,A},Q_2=P_{miss,B},Q_3=P_{miss,C},Q_4=P_D \tag{8}$$

由于综合测试检测问题的能力相同，所以当综合测试测出问题时，指向各子系统的比例 λ_i 应该正比于该子系统实际存在问题的概率。

综合测试测出系统有问题且指向各子系统的比例 $\lambda_i,i=1,2,3,4$ 计算如下：

$$\lambda_i=\frac{Q_i}{Q_1+Q_2+Q_3+Q_4} \tag{9}$$

其中，Q_i 代表第 i 个子系统贡献的问题概率。

3.3 问题一结果与分析

将表 2 所示的问题一中的已知参量代入，可得所需结果。

表2 问题一参数取值

符号	取值（%）	符号	取值（%）
P_A	2.5	$P_{A,false}$	3
P_B	3	$P_{B,false}$	4
P_C	2	$P_{C,false}$	2
P_D	0.1	$P_{E,false}$	2

子系统 i 有通过测试仍有问题的概率为：

$$\begin{aligned}
P_{miss,A}&=P_A\times P_{A,miss}=P_A\times P_{A,false}\times 50\%=0.0375\%\\
P_{miss,B}&=P_B\times P_{B,miss}=P_B\times P_{B,false}\times 50\%=0.06\%\\
P_{miss,C}&=P_C\times P_{C,miss}=P_C\times P_{C,false}\times 50\%=0.02\%
\end{aligned} \tag{10}$$

综合测试测出有问题的概率 P_{sys} 为

$$\begin{aligned}
P_{sys}&=1-P_{zer}\\
&=1-\left(1-P_{miss,A}\right)\left(1-P_{miss,B}\right)\left(1-P_{miss,C}\right)\left(1-P_D\right)\\
&=P_{miss,A}+P_{miss,B}+P_{miss,C}+P_D\\
&=0.2175\%
\end{aligned} \tag{11}$$

综合测试测出有问题的概率 P_{false} 为

$$P_{Ed}=1-P_{E,miss}=1-P_{E,false}\times 50\%=99\% \tag{12}$$

综合测试“发现问题”的总概率 P_d 为

$$P_d=P_{sys}\times P_{Ed}=0.215325\%\approx 0.2153\% \tag{13}$$

综合测试测出系统有问题且指向各子系统的比例 $\lambda_i,i=1,2,3,4$ 结果如表 3 所示。将表 3 所示的问题一求解结果进行相加，满足(14)提到的问题一要求，验证了本文对 $\lambda_i,i=1,2,3,4$ 计算的正确性。

$$\lambda_1+\lambda_2+\lambda_3+\lambda_4=1 \tag{14}$$

表3 综合测试测出有问题时各部分所占比例

子系统 A	子系统 B	子系统 C	子系统 D
$\lambda_1=0.1724$	$\lambda_2=0.2759$	$\lambda_3=0.0919$	$\lambda_4=0.4598$

综合测试测出有问题时各部分所占比例可视化结果如图 2 所示，可以看到，子系统 D 有问题造成综合测试出现问题的概率最大。进一步分析该结果，尽管前面已通过测试的子系统 A、B、C，可能有漏判，造成综合测试测出有问题。但每个子系统在测试时，若有问题，会进行重测，且各测试小组测手出差错是小概率事件，子系统 A、B、C 重复测试之后，仍被漏判，遗留到综合测试的概率更小，如图 3 所示。因此，子系统 A、B、C 联接之后组成的子系统 D 本身有问题，对于综合测试测出有问题占比最大，符合实际。

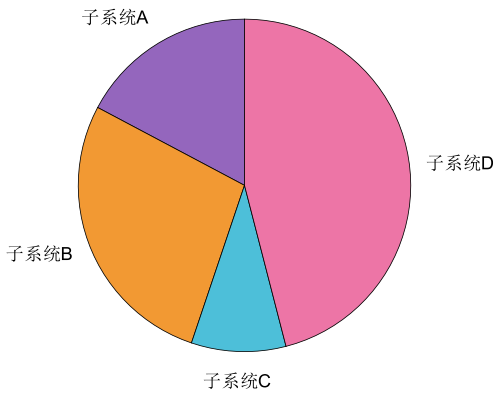


图2 综合测试测出有问题时各子系统所占比例

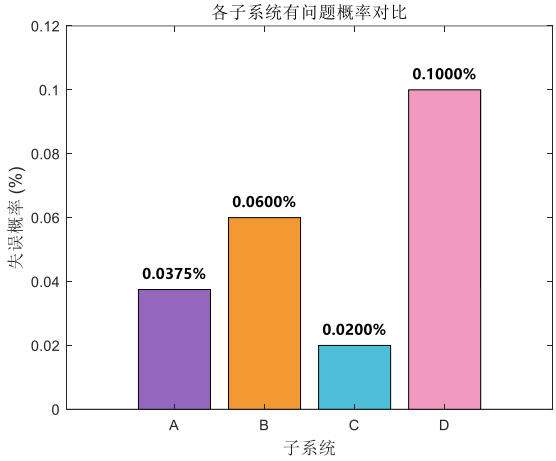
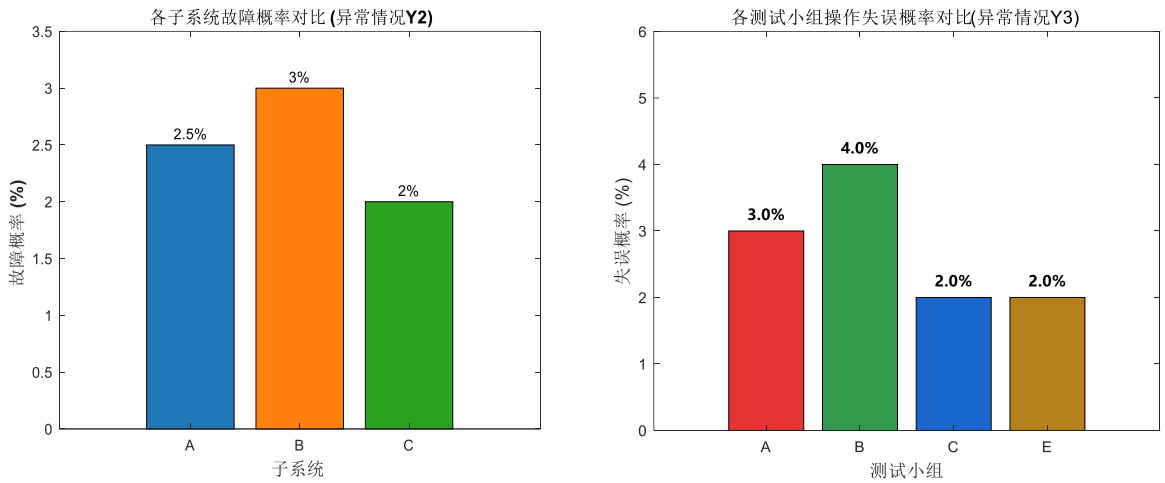


图3 各子系统的问题遗留到综合测试的概率

对于子系统 A、B、C 三者的占比，由图 4 所示的概率对比可以看出，子系统 B 本身存在问题的概率大于子系统 A、C，且 B 测试小组测手出差错概率也更大，则导致子系统 B 的问题遗留到综合测试的概率大于子系统 A、C，进一步证明了问题一所求解的综合测试测出有问题时各部分所占比例的合理性。



（a）异常情况 Y2　　（b）异常情况 Y3

图4 异常情况 Y2 和 Y3 中各组概率对比

4 问题二分析与模型建立

4.1 问题二分析

问题二要求我们在任务尽快完成，总的漏判概率尽量低，即任务完成平均天数 T 和漏判概率 P_L 尽可能小的目标下制定测试工作计划，并计算，任务完成平均天数（T）、通过测试的装置的平均数目（S）、总漏判概率（P_L）、总误判概率（P_W）、各个专业测试组的有效工作时间比（YXB）等各项统计指标。

题目未给定装置完成 A、B、C 子系统测试后联接所需时间及重测间隙所用时间，我们根据实际工程情况，假设其时间分别为 0.1h 和 0.5h，并且测试通过与否在测试结束后才知道。

主要目标任务受限于各种异常情况，其只有更换设备的时间可以人为控制，其他因素均不可控，因此：**1、本文选择将何时更换设备作为工作计划之一**，构建目标函数并分别赋予任务完成平均天数和总漏判概率权重，通过循环遍历法得到目标函数值最优时对应的设备更换时间。**2、本文计划 A、B、C、E 4 个测试小组的人数分别有 2 人，分别在测试台 1、2 进行 A、B、C、E 子系统测试**，如图 5 所示。这样可减少总测试时长。

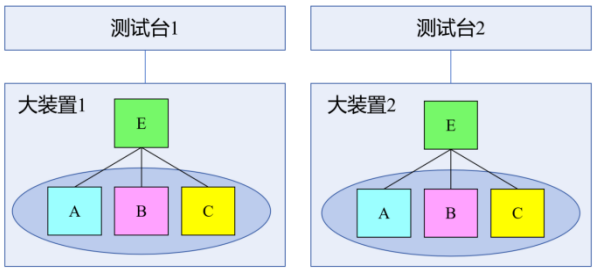


图5 测试台与测试小组示意

我们发现该大型装置检测流程符合离散事件执行规则，且无法确定和预测异常情况发生的具体时刻，因此，构建离散事件仿真模型对整个检测流程进行模拟，并通过蒙特卡洛算法模拟事件中发生的异常情况，给定工作计划，保证主要任务目标下计算题目二的各类指标。

4.2 基于离散事件模拟和蒙特卡洛模型的单分队任务规划

4.2.1 目标函数与约束条件

为了使任务尽快完成且总的漏判概率尽量低，我们构建优化指标的目标函数S_i。设有n次实验，第i次实验的结果为R_i：

$$R_i = (d_i, m_i) \tag{15}$$

式中d_i为平均天数，m_i为漏检率。

定义归一化因子：

$$\begin{cases} D_{\max} = \max\limits_{1\le i\le n} d_i \\ M_{\max} = \max\limits_{1\le i\le n} m_i \end{cases} \tag{16}$$

式中$D_{\max}$为n次实验d_i的最大值，$M_{\max}$为n次实验m_i的最大值。

对d_i和m_i进行归一化处理得：

$$\begin{cases} \tilde{d}_i = \dfrac{d_i}{D_{\max}} \\ \tilde{m}_i = \dfrac{m_i}{M_{\max}} \end{cases} \tag{17}$$

因此，目标函数S_i为：

$$S_i = w_d \tilde{d}_i + w_m \tilde{m}_i \tag{18}$$

其中w_d为d_i的权重，w_m为总漏判概率m_i的权重。由于题目给定主要目标：尽快完成任务，漏判概率尽量低，可得尽可完成任务的重要性大于漏判概率低，因此w_d=0.7，w_m=0.3：

$$\begin{cases} w_d = 0.7 \\ w_m = 0.3 \end{cases} \tag{19}$$

再进行n次实验得到S'向量：

$$S' = (S_1, S_2, \ldots, S_n) \tag{20}$$

求得S'中的最小值，此时的任务完成平均天数和总漏判率最小：

$$i^* = \arg\min_{1\le i\le n} S_i \tag{21}$$

同时存在约束条件：

$$\begin{cases} h_w \le 12 \\ P_{Yi}, i = \{1,2,3,4\} \end{cases} \tag{22}$$

即每日工作时长h_w不超过 12 小时，限制了任务完成的平均天数，而又存在 Y1、Y2、Y3、Y4 的异常情况，其中只有设备更换时间可以人为设定，其余异常情况都不可控，更换设备的概率P_{change}如下：

$$\begin{cases} P_{change}=\{0.03,0.04,0.02,0.03\}, h<120 \\ P_{change}=\{0.05,0.07,0.06,0.05\}, 120\le h<240 \\ P_{change}=1, h\ge 240 \\ P_{change}=1, Fail \end{cases} \tag{23}$$

设备使用时间越长其出故障需要更换的概率越大，仍然会限制任务完成的平均天数，人为提前更换设备可有效降低其故障概率，因此为求目标函数的最优解，需要求得何时更换设备。

4.2.2 离散事件仿真模型

（1）模型建立

为求得何时更换设备及各类指标，需要模拟整个检测流程，因此我们建立离散事件仿真模型，主要包括设置实体系统、设置离散事件、设置统计计数器。

1.设置系统实体

模型所用系统实体包括装置、测试小组、测试台、测试设备，其中 D_e 为装置列表，d_{e1} 为装置 1；G 为测试小组列表，包括 A、B、C、E 4 个测试小组；W_i 为测试台 i，表示有 2 个测试台，其状态为空闲或者占用；M 为测试设备列表，其与 4 个测试小组分别一一对应。

$$\begin{cases} D_e=\{d_{e1},d_{e2},...,d_{e100}\} \\ G=\{A,B,C,E\} \\ W_i\in\{\text{空闲, 占用}\}, i=1,2 \\ M=\{m\_A,\ m\_B,\ m\_C,\ m\_E\} \end{cases} \tag{24}$$

2.设置离散事件

在 Python 中调用 SimPy 函数库建立离散事件，离散事件列表 E 如下：

$$E=\{E_{arr},E_{setup},E_{test},E_{change},E_{retest},E_{dep}\} \tag{25}$$

分别为装置运至大厅、设备调试完成、装置测试完成、更换设备完成、装置重测完成和装置运出大厅。

时钟驱动事件公式如下：

$$\begin{cases} E_{arr}(i):\tau=t+T_{in} \\ E_{setup}(j):\tau=t+T_0(j) \\ E_{test}(i,j):\tau=t+T_{tes}(j) \\ E_{change}(j):\tau=t+T_0(j) \\ E_{retest}(i,j):\tau=t+T_{tes}(j) \\ E_{dep}(i):\tau=t+T_{out} \\ j\in\{A,B,C,E\} \end{cases} \tag{26}$$

i 为装置号，j 为当前工序，τ 为事件触发时刻，t 为当前时间，T_{in}、$T_0(j)$、$T_{tes}(j)$、T_{out} 分别表示装置运入大厅时长、j 工序下设备重启调试时长、j 工序下装置检测时长及装置运出大厅时长，各时间变量取值如表 4 ，各事件形象化表示表 4 图 6

表4 各符号取值

符号	取值（小时）
T_{in}	0.5
$T_0(j)$	$\{\frac{1}{2},\frac{1}{3},\frac{1}{3},\frac{2}{3} \mid j=A,B,C,E\}$
$T_{tes}(j)$	$\{2.5,2,2.5,3 \mid j=A,B,C,E\}$
T_{out}	0.5

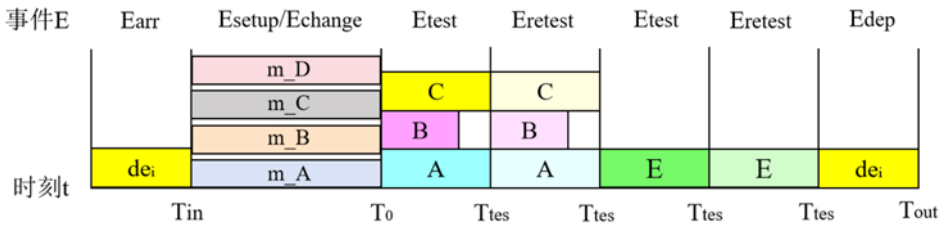


图6 离散事件示意

3.设置统计计数器

本模型需要统计的参数：漏判总数、误判总数、测试总时间和通过检测装置总数如下：

$$\begin{cases} N_{miss}=\sum_{i=1}^{100}\sum_{j=A}^{E}ns_{ij} \\ N_{fail}=\sum_{i=1}^{100}\sum_{j=A}^{E}nf_{ij} \\ T_{all}=\left\{\tau \mid E_{dep}(100)\right\} \end{cases} \tag{27}$$

其中，ns_{ij} 为 i 装置 j 工序时的漏判数，nf_{ij} 为 i 装置 j 工序时的误判数，N_{miss} 为漏判总数，N_{fail} 为误判总数，T_{all} 为测试总时间。另外，通过检测装置总数在代码中用计数变量统计得到。

（2）事件调度法

事件调度法以事件为分析系统的基本单元，通过定义事件以及每个事件发生后对系统状态的影响按时间顺序执行每个事件并策划新的事件来驱动仿真模型的运行[5]。

本文离散事件模型中系统的状态演化由仿真时钟和事件列表 E 驱动。

（3）蒙特卡洛算法

由于装置测试过程中有许多随机性异常情况，因此我们采用蒙特卡洛算法[6]模拟随机异常情况，在 1000 次循环下保证结果的可靠。

在离散事件仿真器内部，每当工序 X 开始测试时，执行单次抽样判定公式：

$$r\sim\mathcal{U}(0,1),\quad fault_X\leftarrow\mathbb{I}(r\le P_{Yi}) \tag{28}$$

式中，r 为产生的随机数，$fault_X$ 表示发生异常情况，P_{Yi} 为 Y_i (i=1,2,3,4)异常情况发生的概率，若 r 小于等于该概率则说明发生异常情况，触发 $fault_X$。

如图 8 ，当事件进行至可能发生的异常情况时，通过 Python 中 random.random()函数生成 0-1 的随机数 r，与异常事件发生概率进行对比，判断概率性异常问题是否发生，若发生，则事件调度法推动异常事件如设备故障进行，反之则推动事件正常进行如设备测试，时钟同步运行相应时长，再将该单次抽样判定循环 1000 次实现随机事件模拟。

（4）循环遍历法

前文所述通过蒙特卡洛算法模拟流程中的随机事件，通过离散事件仿真建立装置测试流程，在事件进行中，为求得更换设备时间的最优值，我们采用循环遍历法，如式(29)，在设备使用时长大于等于 120h 而小于 240h 时以 δ=10 为步长，依次循环遍历各时长，更换设备，并计算对应的目标函数值，目标函数值最小对应的更换设备时间为最优值。

$$\begin{cases} T_{eqb} \in [120, 240] \\ \delta = 10 \end{cases} \tag{29}$$

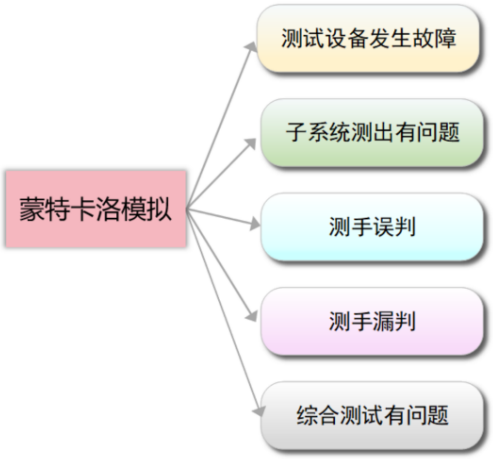


图7 蒙特卡洛模拟情况

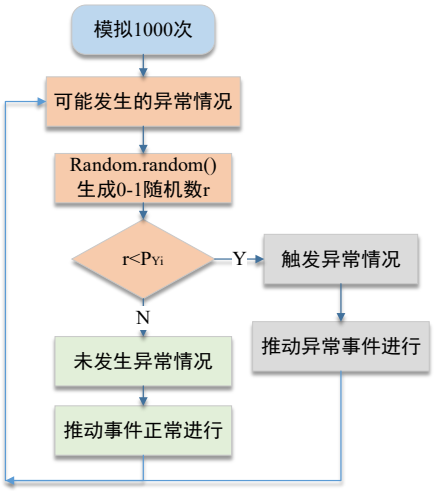


图8 蒙特卡洛模拟流程

（5）仿真流程

模型准备流程如图 9 所示，首先创建测试设备、装置和测试台，创建仿真时钟、测试队列，系统生成 100 个待检测装置并赋予 ID 号，将装置放入队列排队，再从队列中按顺序获取第 n 个装置，系统时钟运行 0.5h 模拟该装置运入大厅。在获取空闲的测试台后，装置自动分配至该测试台进行子系统的并行检测。

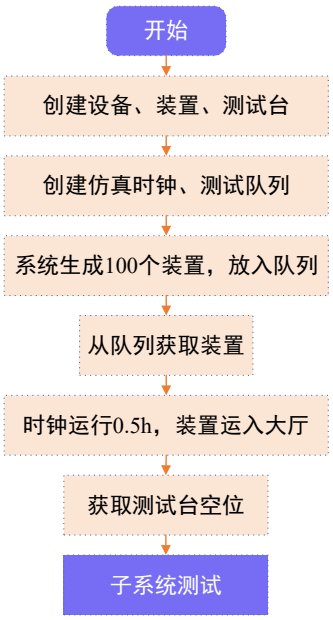


图9 离散事件仿真模型准备流程

如图 10 所示，装置 n 进入 ABC 子系统测试，且 ABC 子系统为并行测试：

Step1：编程获取当前设备的使用时长T_{eqb}，为了使目标函数最小，需考虑设备何时更换以降低出故障的概率，因此若T_{eqb}大于 120h 且小于 240h 则选择人为更换设备，此处采用循环遍历法，步长为 10h，获得不同更换设备时间下的指标计算结果，得到最优的时间。

Step2：若T_{eqb}大于 240h 需强制更换设备，此处更换设备都将增加调试时长。

Step3：进一步判断本次测试是否为首测，若为重测则判断重测次数T_{re}是否大于 1，若为真则表示重测次数已用完，检测失败，装置退出大厅，同时队列取下一个装置进入大厅进行测试，并将运输时间计入总时长。若重测次数小于等于 1 则该装置进行第二次测试。

Step4：若为首测，则记录当前起始时间T_{st}，此处引入蒙特卡洛随机模拟，判断是否漏判，若漏判则将漏判计数ns_{ij}+1，反之测试不通过，进行重测。

Step5：若子系统检测出没问题，则判断是否为误判，在一定概率下，若为误判则误判计数 +1，反之本次子系统测试通过，由于 A B C 子系统并行执行测试，因此任一系统测试失败则装置测试不通过且不进行 E 综合测试。

Step6：而若子系统均通过测试，则进行最后的综合测试，综合测试中的 D 子系统故障模拟类似前文所述，且流程与子系统检测类似，因此不再赘述。

本模型仿真中统计漏判次数、误判次数、以及测试时长计算问题二所求各项统计指标，各指标求解公式如下：

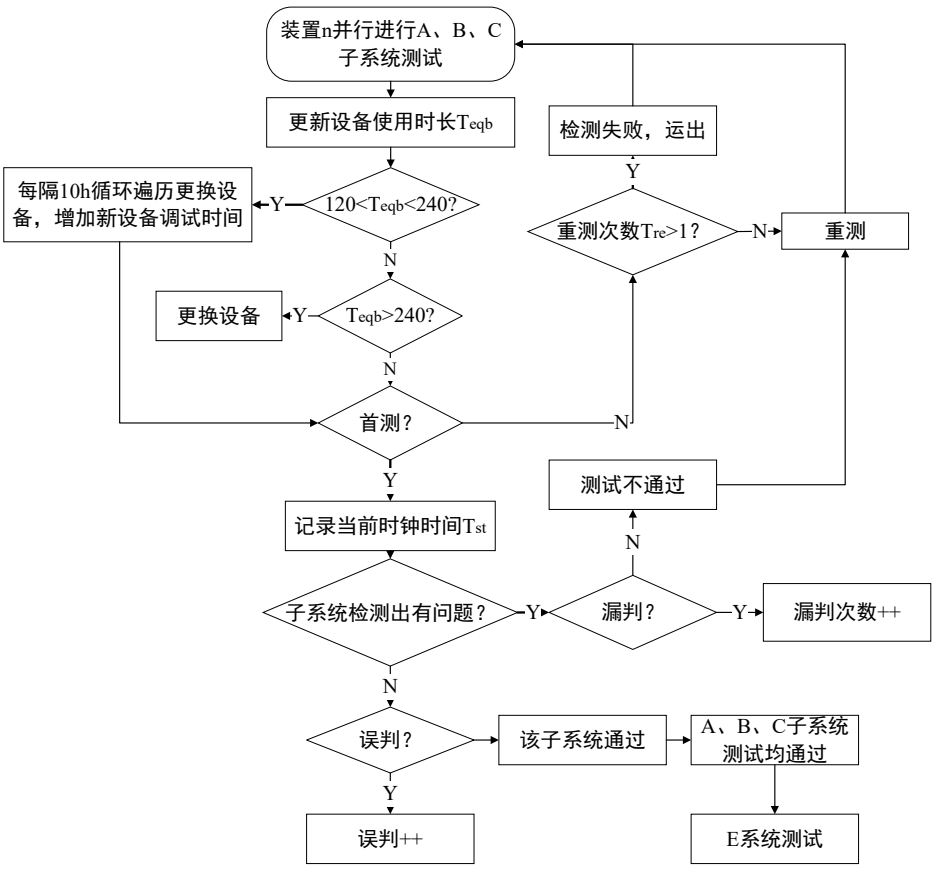


图10 仿真流程

• 任务完成平均天数T：

$$T = \frac{T_{all}}{12} \tag{30}$$

其中 T_{all} 为总的测试时长。

- 通过测试的装置的平均数目 S 为代码中通过计数变量获得。
- n 为装置数 100，总漏判概率 P_L：

$$P_L = \frac{N_{miss}}{n} \tag{31}$$

- 总误判概率 P_W：

$$P_W = \frac{N_{fail}}{n} \tag{32}$$

- 各个专业测试组的有效工作时间比 YXB：

$$\begin{cases} YXB1 = \dfrac{\overline{td_A}}{12} \\ YXB2 = \dfrac{\overline{td_B}}{12} \\ YXB3 = \dfrac{\overline{td_C}}{12} \\ YXB4 = \dfrac{\overline{td_E}}{12} \end{cases} \tag{33}$$

式中 $\overline{td_A}$、$\overline{td_B}$、$\overline{td_C}$、$\overline{td_E}$ 分别为系统 A、B、C、E 的小组每班次平均测试时间。

4.3 结果与分析

（1）设备更换时间及各类指标计算结果

循环遍历得到平均测试天数和漏判率随设备更换阈值的变化曲线分别为图 11 和图 12 ，可得更换设备时间为 190h 及 240h 时的平均测试天数最少，而更换时间为 230h 时的漏判率最低，计算目标函数值得到图 13 ，可得设备更换时间为 230h 时目标函数值最小，即此时的任务完成平均天数 T 和总漏判率 P_L 最小，满足题目对主要目标的要求，因此计划在每台设备使用时间为 230h 左右时及时更换设备。

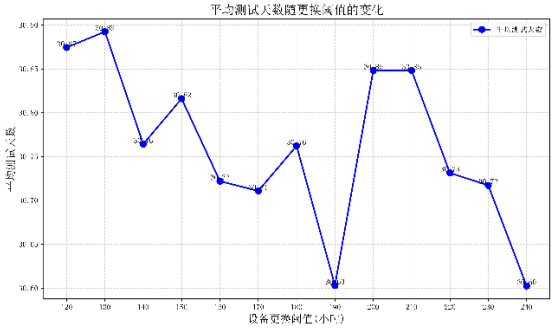


图11 平均测试天数与设备更换时间关系

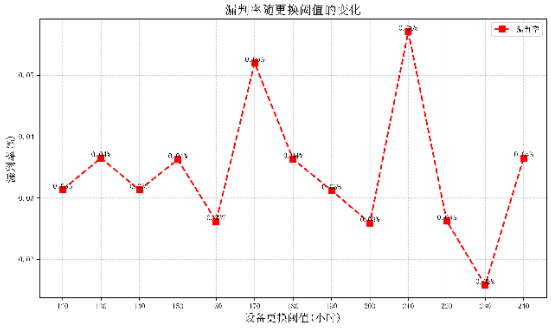


图12 漏判率与设备更换时间关系

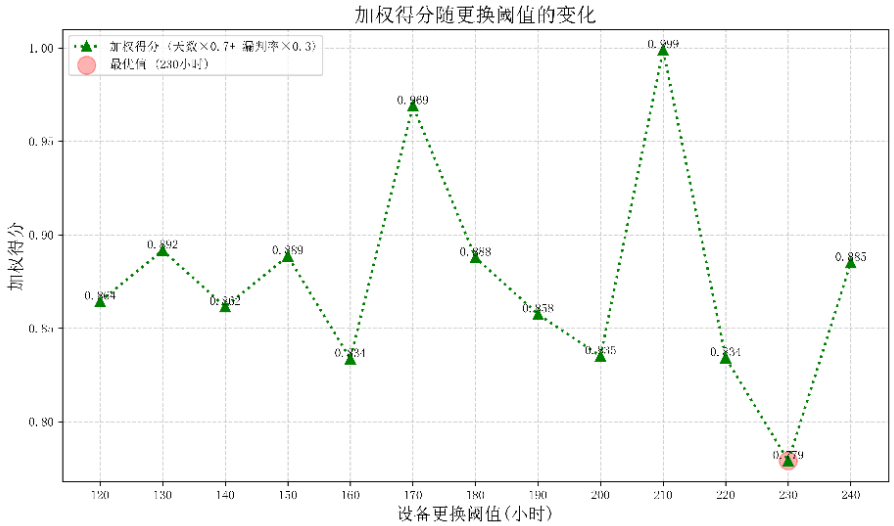


图13 目标函数值与设备更换时间关系

此时的各类指标计算结果如表 5 ，分析得到 YXB 中 YXB4 值最大，为 0.6019，YXB2 值最小，为 0.4399，而 YXB1 和 YXB3 的值很接近，这符合题目设定条件，A、B、C 子系统均通过检测才进行 E 综合测试，因此 E 发生异常情况的概率会比子系统概率更小，所对应的有效工作时间 YXB4 也就更大；B 系统检测设备出故障的概率、测手出错的概率、检测出问题的概率均大于其他系统，因此其有效工作时间 YXB2 理应最小，而 A 子系统发生异常情况概率和 C 系统近似，所以二者的有效工作时间接近。若理想化情况，不考虑所有异常问题，每天完成 4 台装置检测，完成 100 台装置检测需要 25 天，但问题求解时存在异常情况，因此所需天数 T 不可能小于 25 天，而我们计算出的结果为 31 天，从这个角度而言也是合理的。综上所述，结果合理且指标性能较好。

表5 问题 2 结果统计指标

T	S	P_L	P_W	YXB1	YXB2	YXB3	YXB4
31	95	0.0158%	1.3024%	0.5514	0.4399	0.5517	0.6019

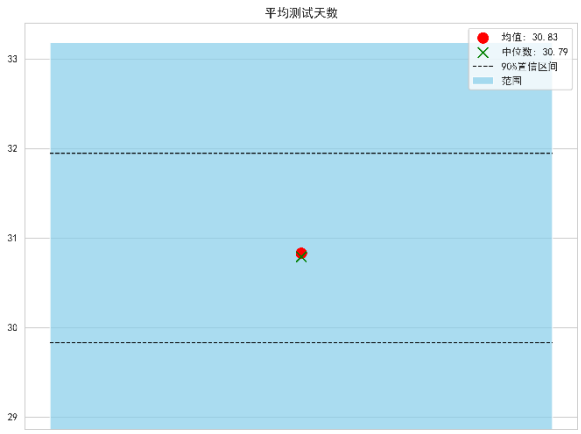
（2）工作计划

综合上述分析，为了使任务尽快完成，总的漏判概率尽量低，则需要在可控范围内提高测试小组工作效率及减小异常情况发生概率。为提高测试小组工作效率，我们计划让 2 个测试台同时进行装置检测，在此前提下，计划设备使用时长达到 230h 左右时人为更换新设备，此时装置检测总时长+总漏判概率的值最小，满足题目要求，且计算得其余各指标性能较好。

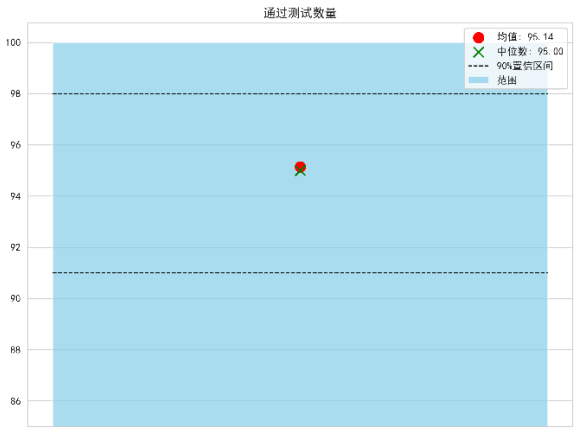
（3）蒙特卡洛算法结果可靠性验证

为验证蒙特卡洛算法随机性模拟结果的可靠性[7]，我们计算 1000 次模拟下，各指标的均值、中位数和数值范围如图 14 ，可以看出，各指标的均值和中位数非常接近，几乎完全重叠，意味着数据分布均匀，并无极端值出现。此外，由于置信区间的上限和下限与均值和中位数的差距不大，这也表明数据的可靠性较高。

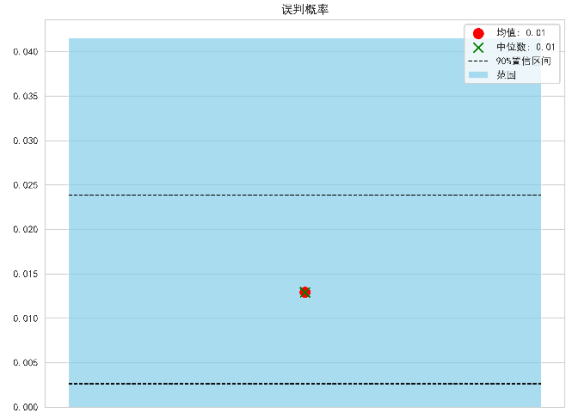
如图 15 所示为各子系统的有效工作时间比，三条雷达线均值、5th 百分位数、95th 百分位数形状相似，说明各子系统的性能分布规律一致。绿色线 95th 百分位数始终位于最外侧，橙色线 5th 百分位数在最内侧，表明系统在大多数情况下表现稳定，蓝色线均值为 0.541，接近中位水平，说明整体表现均衡。因此本模型使用蒙特卡洛算法可以生成大量随机样本，用统计结果来近似模拟概率，并且具有优异的可靠性。



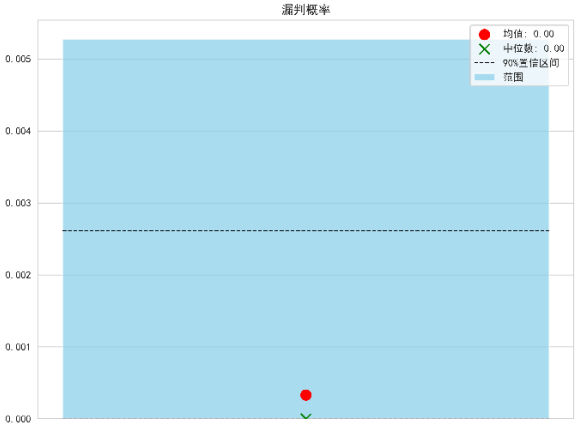
(a)平均测试天数



(b)通过测试的装置数量



(c)误判概率



(d)漏判概率

图14 蒙特卡洛算法下各指标的均值、中位数及数值范围

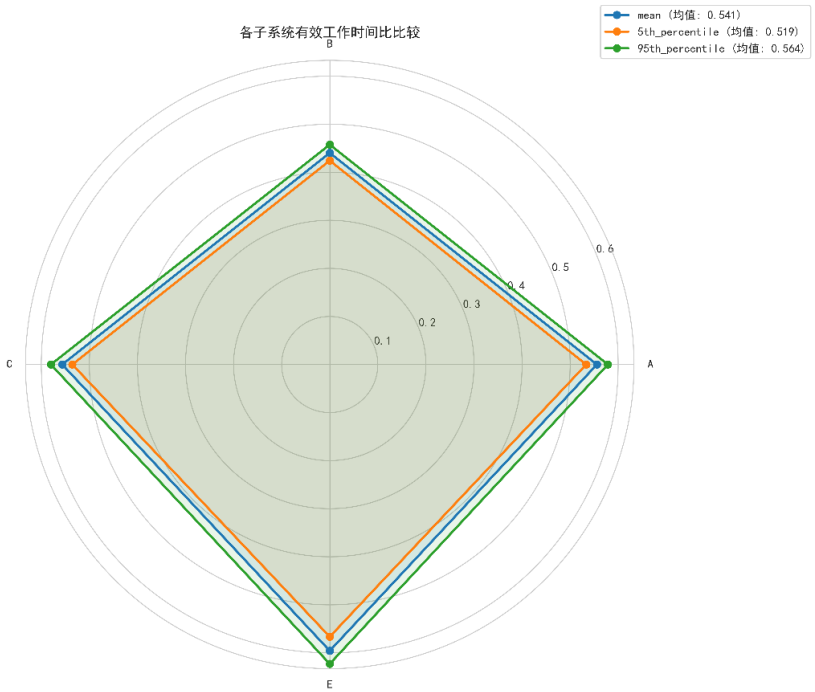


图15 各子系统有效工作时间比雷达图

（4）工作计划效果验证

我们通过对比提前人为更换设备和不提前更换设备的最终指标得到表 6 ，验证本工作计划中提前更换设备的优良效果。表 6 中序号 1 表示人为提前更换设备，序号 2 表示不提前更换，灰色部分表示二者指标值相等，黄色部分为优越方，红色字体为题目要求的主要任务目标，可明显看出提前更换设备对应的主要指标值远好于不提前更换设备，因此可以验证我们工作计划的优越效果。

表6 提前更换设备与不提前更换对应的指标对比

序号	T	S	P_L	P_W	YXB1	YXB2	YXB3	YXB4
1	31	95	0.0158%	1.3024%	0.5514	0.4399	0.5517	0.6019
2	31	95	1.34%	0.03%	0.5565	0.4409	0.5569	0.5949

5 问题三分析与模型建立

5.1 问题三分析

问题三为了加快测试进度，安排两个分队接续倒班测试 100 个装置，两个分队接续倒班，每班工作 K 小时（$K \in [9,12]$，步长 0.5）。同工序两个班次使用同一套测试设备即设备连续使用，不休息。目标为最小化任务完成平均天数，同时控制漏判概率，提高有效工作时间比。

问题三在问题二的基础上，重点需要考虑两个分队的交接班过程，优化每班工作时长 K。因此，在问题二基于离散事件模拟和蒙特卡洛模型的基础上，问题三构建了一个复杂的测试系统仿真模型，该系统采用两班倒工作制，用于对包含多个子系统的装配进行质量检测。系统核心由两个并行工作站组成，每个工作站配备四种测试设备（A、B、C 子系统测试设备和 E 综合测试设备）。模型采用离散事件仿真方法，通过 SimPy 框架实现，并引入遗传算法优化班次时长参数 K。

5.2 融合遗传算法的双分队任务规划

5.2.1 算法整体框架

问题三的融合遗传算法的双分队任务规划整体算法流程包含三大部分，如图 16 所示，参数优化循环（遗传算法）、统计仿真循环（蒙特卡洛模型）、生产测试流程（离散事件仿真）。

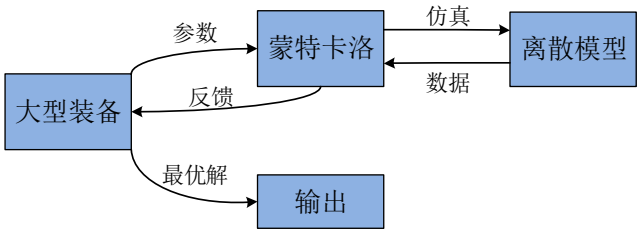


图16 融合遗传算法的双分队任务规划整体框架

具体而言，系统采用三层嵌套结构：

（1）遗传算法层：在解空间（班次时长 $K \in [9,12]$）中搜索最优解；

（2）蒙特卡洛层：对每个候选 K 值进行多次独立仿真，消除随机性误差；

（3）离散事件仿真层：模拟测试生产线动态过程。

5.2.2 离散事件仿真层

离散事件仿真层的功能是在模拟大型装置测试的流程，是一个装备完整测试流程的核心子流程。综合考虑了各种异常情况，包含设备故障、设备校准、测手失误等。这些与问题二中算法一致。特殊的是，问题三的离散事件仿真层还包括一个重要的班次转换逻辑，结构图如图 17 所示。

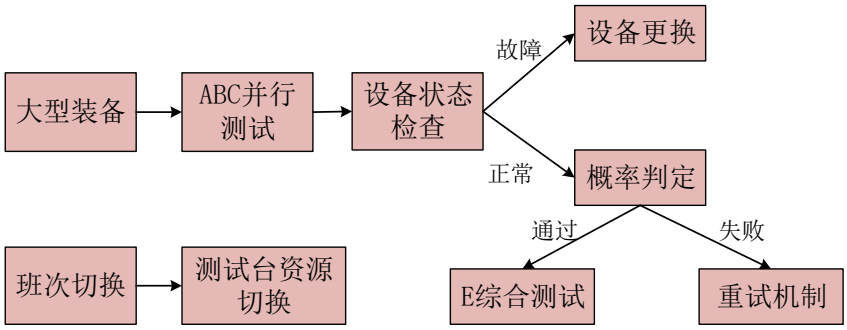


图17 离散事件仿真层结构图

倒班机制的实现是连接参数 K 与系统性能的关键桥梁，其实现细节如下：

核心逻辑：仿真环境中运行着一个独立的“班次切换控制器”进程。

周期：该进程每 K 小时即一个班次时长自动触发一次切换事件。

动作：当切换事件发生时：

（1）当前正在工作的工作站将被设置为“休眠”状态，不再接收新的装备；

（2）另一个工作站被激活，开始接收和处理新的装配体；

（3）对于“休眠”工作站中尚未完成测试的装配体，它们将继续占用设备直至当前测试完成，完成后该设备也进入休眠。

5.2.3 混合遗传算法

混合遗传算法[8]为避免过早陷入局部最优，引入精英族群，并采用自适应调整交叉和变异概率的方法来改进遗传算法，设计了综合自适应的参数调整、多种群迁移和精英族群，增强了算法的性能。

步骤 1：初始化种群。在问题的解空间内（即班次时长 K 的取值范围，为 $[9.0, 12.0]$ 小时）随机生成一组初始解。种群大小是预设的，例如包含 50 个个体。每个个体直接用一个实数表示，即一个候选的 K 值。

步骤 2：适应度评估。对当前种群中的每一个 K 值），调用蒙特卡洛层进行评估，并接收其返回的系统性能数据。遗传算法将当前需要评估的 K 值传递给蒙特卡洛模块。接收蒙特卡洛模块返回的、经过多次仿真平均后的系统性能指标，即总测试天数的平均值和漏判率的平均值。计算适应度，根据收到总测试天数的平均值和漏判率的平均值，按照定义的适应度函来计算每个个体的适应度值。对于适应度函数，问题三需要综合考虑最小化平均测试天数和最小化漏判概率。因此，定义适应度函数 F 如下：

$$F = \omega_1 T + \omega_2 P_L \tag{34}$$

其中，通过权重系数 ω 平衡测试天数和漏判概率。F 值越小，代表该对应 K 值的综合性能越好。

步骤 3：选择操作。基于“优胜劣汰”的原则，从当前种群中选择适应度高的个体作为父代，用于产生下一代。采用“锦标赛选择”策略，每次随机从种群中选取一小部分个体，在这些个体中选择适应度最好的那个作为父代之一。重复此过程，直到选够所需数量的父代个体。

步骤 4：交叉操作。将选出的父代个体两两配对，通过交叉操作生成新的子代个体，以继承父代的优良特性。采用混合交叉$(BLX-\alpha)$策略。交叉操作是创新的主要来源，它能够组合不同父代的基因信息，在解空间中进行探索，有望产生比父代更优的新解。

步骤 5：变异操作。以较小的概率对子代个体进行随机改动，以引入新的基因并维持种群多样性。本算法采用高斯变异。以一个预设的变异概率，给子代的 K 值加上一个服从均值为 0、标准差为σ的正态分布的随机数。

步骤 6：边界修复。变异后的值如果超出了 K 的合法范围$[9.0, 12.0]$，则会被拉回到最近的边界上。防止算法过早收敛于局部最优解。

步骤 7：新一代种群形成与终止检查。用经过选择、交叉、变异后产生的新子代种群替换旧的父代种群，并判断算法是否应结束。基于“精英保留”策略在替换时，将当代适应度最好的个体直接保留到下一代，确保已发现的最优解不会丢失。

步骤 8：终止条件。检查是否满足终止条件。未满足终止条件，则返回步骤二，开始新一轮的“评估-选择-交叉-变异”循环。如果满足终止条件，则算法结束。

5.3 结果与分析

为了验证所建立的融合遗传算法的双分队任务规划模型的有效性，进行仿真实验，其中，问题三的离散事件模拟和蒙特卡洛模型与问题二的基本一致，仅多一个倒班机制，体现在待求解参量 K，蒙特卡洛模拟 1000 次。其他约束条件，例如，装置运入运出时间、测试设备发生故障、测手失误概率等参量均与问题二实验保持一致，在此不再赘述。该部分仅列出所用混合遗传算法的实验参数，如表 7 所示。

表7 混合遗传算法的实验参数

参量	数值
种群大小	50
迭代代数	500
班次时长搜索空间 K	$[9.0, 12.0]$
交叉概率	0.6
变异概率	0.3
锦标赛选择大小	3
高斯变异参数方差	0.5
平均测试天数权重	0.6
漏判概率权重	0.4

混合遗传算法的双分队任务规划模型实验结果如下所示。由图 18 可以看到，遗传算法迭代 500 次后，最佳班次时长 K 值基本稳定在 9.7 小时。同时，由图 19 所示的最佳班次时长频次分布，每迭代 100 次，统计一次 K 值分布，可以发现，迭代次数在 300 次之后，以及到 500 次时，K 值主要频次分布集中在 9.5 到 10 小时之间，符合图 18 所示的迭代结果。因为问题三要求 K 值（$K \in [9,12]$，且步长 0.5），则最佳班次时长 K 值 9.7 小时，必须取为$K=10$小时。

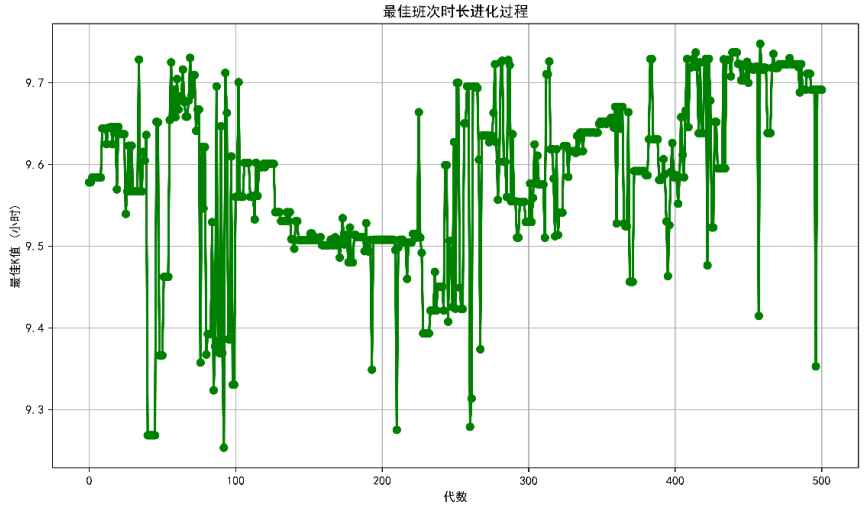


图18 最佳班次时长随迭代次数变化曲线

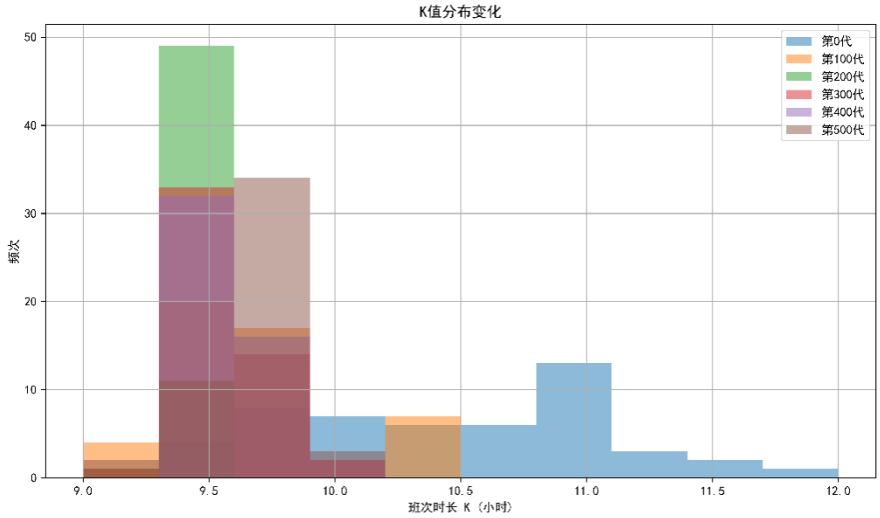


图19 最佳班次时长频次分布图

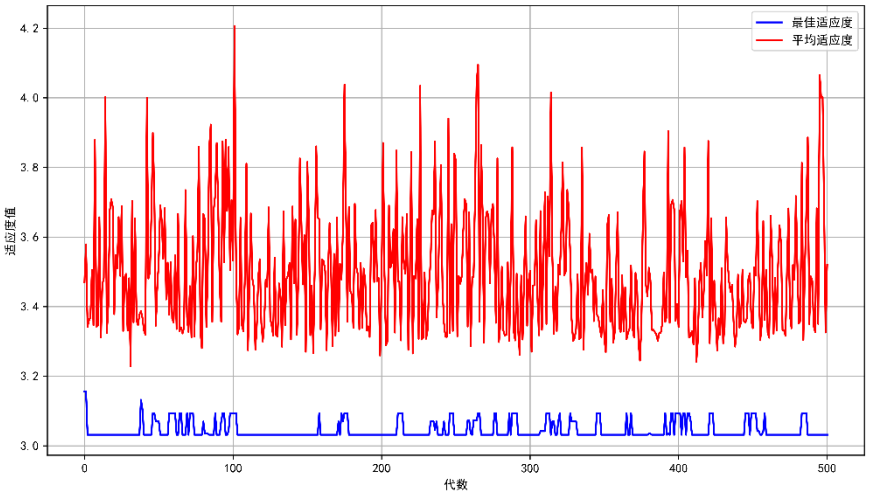


图20 遗传算法迭代过程适应度变化曲线

根据以上实验结果得到的最佳班次时长 $K=10$ 小时，可以确定问题三的工作计划为：两个分队不间断接续工作，每个班次工作时长 10 小时。每个班次单独工作时，工作安

排仍符合问题二的工作计划，测试大厅的两个测试台同时工作，每个测试台对应一个大装置的测试，每个大装置的子系统 A、B、C 并行测试，子系统 A、B、C 均通过后，进行 E 综合测试。

在此工作计划下，求得各项统计指标如表 8 所示。与问题二的单分队工作相比，问题三的双分队接续工作模式明显缩短了任务完成平均天数，且保证较低的漏判和误判概率。同时，保持较高的有效工作时间比。

表8 问题 3 结果统计指标

T	S	P_L	P_W	YXB1	YXB2	YXB3	YXB4
22	96	0.03%	1.25%	0.7853	0.623	0.7813	0.8545

6 问题四分析与模型建立

6.1 问题四分析

问题四需要基于问题三的模型和结果，分析影响测试任务平均完成时间的关键因素，并提出改进建议。问题 3 的场景是任务紧急，安排两个分队接续倒班，每班工作 K 小时（$K \in [9,12]$，步长 0.5），其影响因素主要包含班次时长即 K 值、子系统故障率、设备更换时间阈值、测试差错率等因素。这属于多变量全局敏感性分析，需要通过系统遍历多维参数空间，量化评估多参数协同作用对输出响应的非线性影响，突破线性模型的约束条件，实现复杂系统参数优化的评估。

6.2 参数敏感性分析

6.2.1 基于 Sobol 算法的多变量敏感性分析

在全局敏感性分析方法谱系中，Sobol 方法基于方差分解理论，能够有效分解输入参数的主效应指数及交互效应贡献度，特别适用于强非线性及非单调响应特征的复杂系统模型。

Sobol 法主要分为选取初始输入参数的采样点和将样本点带入模型两个阶段，其详细数学推导任启伟有研究[9]，在此不再赘述。总结而言，Sobol 法得到的总方差 D 可以分解为单个参数与多个参数相互作用的组合。

$$D = \sum_{i=1}^{n} D_i + \sum_{1<i<j\le n} D_{ij} + \ldots + D_{1,2,\ldots n} \tag{35}$$

归一化即可得到:

$$1 = \sum_{i=1}^{n} S_i + \sum_{1<i<j\le n} S_{ij} + \ldots + S_{1,2,\ldots n} \tag{36}$$

分解偏方差与总方差的比值即为灵敏度。一阶敏感性指数表示单个输入参数对输出的影响，计算公式为：

$$S_i = \frac{D_i}{D} \tag{37}$$

全局敏感性指数反映参数相互作用后对输出参数的影响程度，表示为：

$$S_{Ti} = 1 - \frac{D_{\sim i}}{D} \tag{38}$$

式中，D为输出的总方差，D_i为第i个参数的一阶方差，$D_{\sim i}$为除第i个参数外其他参数共同作用产生的方差。

6.2.2 基于随机森林的多变量敏感性分析

基于随机森林进行多变量敏感性分析[10]是另一种非常强大且实用的方法，能够处理高维、非线性且存在交互作用的数据。利用随机森林模型作为原始复杂模型的“代理模型”，然后通过分析这个代理模型的内置指标如特征重要性来评估输入变量对输出变量的影响程度和方式。训练良好的随机森林本身具有极高的预测精度，能够很好地捕捉原始模型输入与输出之间的复杂关系，同时它又提供了一套天然的工具来评估变量重要性。

基于随机森林的多变量敏感性分析分为以下步骤：

步骤 1：生成数据集。随机森林作为一种机器学习方法，首先需要为复杂模型生成一个输入—输出数据集。

输入（X）：根据每个输入变量的概率分布，使用抽样方法生成大量样本点，确保样本能很好地覆盖整个输入空间。

输出（Y）：将上述生成的每个输入样本点代入原始复杂模型中运行，得到对应的输出结果。

步骤 2：训练随机森林代理模型。使用上一步得到的数据集 [11] 来训练一个随机森林模型。随机森林通过构建大量决策树并进行集成。训练完成后，该随机森林模型就可以近似地代表原始复杂模型的输入—输出关系。

步骤 3：进行敏感性分析。随机森林提供了多种评估特征重要性的方法。基于排列的重要性是最常用、最可靠的方法，其原理直接体现了“敏感性”的概念。对于一个训练好的随机森林，在一个测试集即未参与训练的数据上评估其基准精度。然后，对某个特征的值进行随机打乱，破坏这个特征与输出之间的关系，再次用打乱后的数据评估模型精度。

6.3 结果与分析

6.3.1 基于 Sobol 算法的敏感性分析

基于问题三的模型，对班次时长、子系统故障率、测手差错率、设备更换时间 4 个因素进行基于 Sobol 算法的多变量敏感性分析。Sobol 敏感性分析的 4 个关键参数范围如表 9 所示。子系统故障率取子系统 A、B、C 故障概率构成的范围，测手差错率取 4 个测试小组 A、B、C、E 的失误率范围。采用 Saltelli 采样方法生成 1024 组参数组合，通过问题三的模型模拟测试 100 个装置来评估每个参数组合的性能（以总测试天数作为输出指标），同时计算一阶效应（单个参数的直接影响）和总效应（包含交互作用的综合影响）。

表9 Sobol 敏感性分析的 4 个关键参数

影响因素	取值范围
班次时长	[9.0, 12.0]
子系统故障率	[0.02, 0.03]
测手差错率	[0.02, 0.04]
设备更换时间	[120, 240]

基于 Sobol 算法的多变量敏感性分析结果如图 21 所示，左图为一阶效应，即单个参数对总测试时间的直接影响，右图为总效应，即包含因素间交互作用对总测试时间的综合影响。对于班次时长、子系统故障率、测手差错率、设备更换时间 4 个因素，班次时长 K 对问题三模型的总测试时间影响最大。在左图的一阶效应中，可以看到，子系统故障率、测手差错率、设备更换时间这 3 个因素的影响非常小，占比小于 0.05，说明这 3 个因素对总测试时间的直接影响很小。在右图的总效应中，班次时长 K 的影响仍然最大，同时明显发现，子系统故障率、测手差错率、设备更换时间这 3 个因素的占比有了较大提高，该对比说明，这 3 个因素更大程度上，是相互影响导致交互作用对总测试时间造成一定影响。

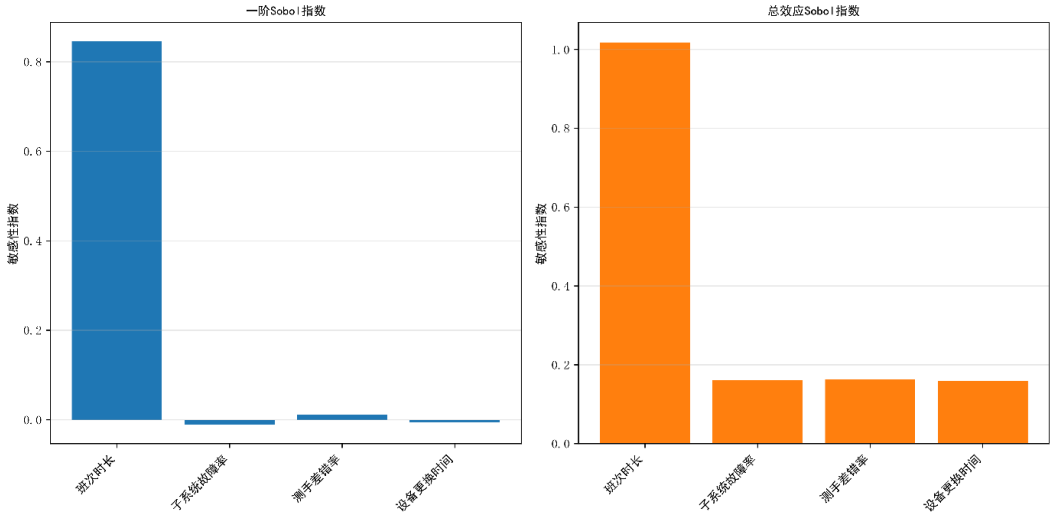


图21 Sobol 敏感性分析结果

6.3.2 基于随机森林的敏感性分析

基于问题三的模型，仍对班次时长、子系统故障率、测手差错率、设备更换时间 4 个因素进行基于随机森林的多变量敏感性分析，4 个因素的实验参数同 6.3.1 表 9 。通过仿真采样得到 1024 组参数组合，通过问题三的模型模拟测试 100 个装置，取其中 10000 组样本用于随机森林的训练和测试，具体参数设置如表 10 所示。

表10 随机森林的实验参数设置

参数	取值
总样本数	10000
训练集	8000
测试集	2000
决策树数量	200
树深度	5

基于随机森林的多变量敏感性分析结果如图 22 所示，对于班次时长、子系统故障率、测手差错率、设备更换时间 4 个因素，班次时长 K 对问题三模型的总测试时间影响最大特征重要性为 0.983，子系统故障率、测手差错率、设备更换时间这 3 个因素的特征重要性占比很小，分别为 0.004、0.008、0005。基于随机森林的多变量敏感性分析结果整体趋势与基于 Sobol 算法的多变量敏感性分析结果一致，进一步证明了本文利用两种算法对多变量敏感性分析的合理性和有效性。

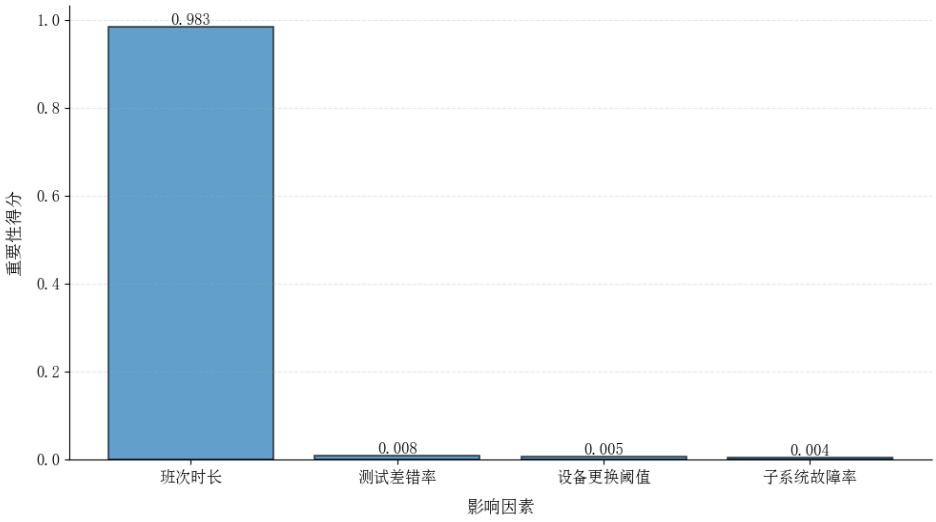


图22 基于随机森林的多变量敏感性分析结果

6.3.3 单变量敏感性分析

由基于 Sobol 算法的多变量敏感性分析和基于随机森林的多变量敏感性分析结果看到，对于问题三的模型，班次时长是对测试任务平均完成时间最重要的影响。同时，在问题三中，我们设定了式(34)的适应度函数，但对于平均完成天数和漏判概率二者的加权系数没有进一步研究。在该部分，我们借助问题三平均完成天数的加权系数占比来分析班次时长的单变量敏感性分析。

具体而言，我们将适应度函数中平均完成天数的加权系数 ω_1 作为一个单变量，则漏判概率的加权系数 $\omega_2=1-\omega_1$。取加权系数 ω_1 的范围为 0.1-0.9，步长 0.05，则得到 17 组加权系数组合 (ω_1,ω_2)，保证其他实验参数一致的条件下，对于每一组加权系数组合，均基于问题三的模型，进行 3 次实验，结果取均值。以测试任务平均完成时间为观测指标，得到如图所示的平均完成时间与平均完成天数的加权系数 ω_1 的变化趋势。

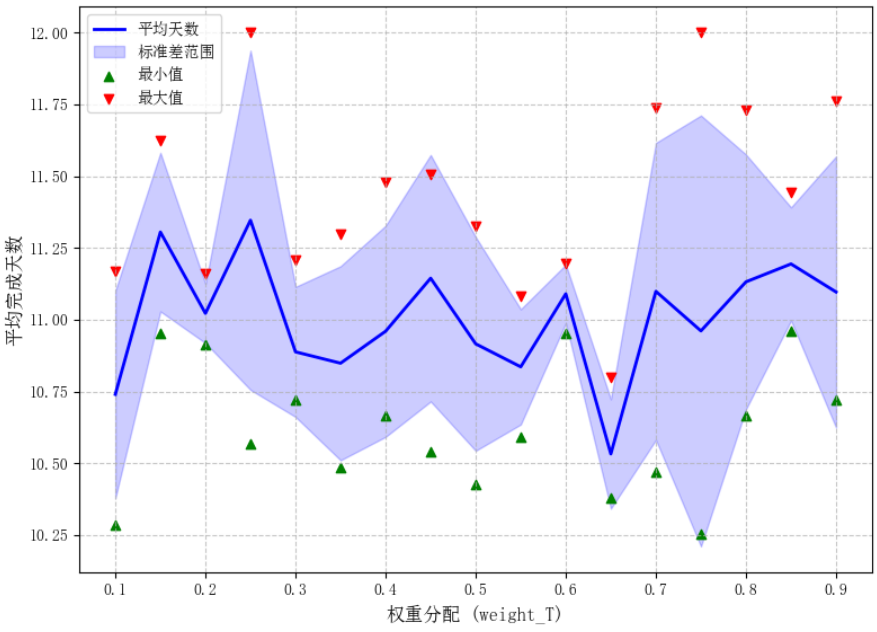


图23 平均完成天数的加权系数 ω_1 单因素敏感性分析

由图 23 所示结果，可以看到，平均完成时间与平均完成天数的加权系数ω_1变化曲线整体呈现一个“U”形。

（1）初始阶段（$\omega_1 = 0.1-0.3$）。该部分整体的平均天数较高，此时加权权重主要分配给漏判概率P_L，系统倾向于选择更保守的工作安排，测试过程更谨慎，但整体效率较低，完成天数较长。

（2）中间阶段（$\omega_1 = 0.3-0.6$）。测试完成平均天数逐渐下降并达到最优值。该段内，综合考虑了时间效率和质量控制，达到较好平衡。

（3）后期阶段（$\omega_1 = 0.6-0.9$）。平均天数再次上升，因为该段过度强调最小化漏判概率，可能导致选择过短的班次时长，重试次数增加，设备故障率可能上升，最终反而延长了总完成时间。

6.4 改进建议

由 6.3 可知，班次时长是影响测试任务平均完成天数和总漏判概率的最显著因素。其次，设备故障率、测手差错率等因素也具有一定影响。为实现任务尽快完成且漏判概率尽量低的双重目标，现提出如下多层次改进建议：

（1）优化排班制度与资源配置

实施弹性动态排班制：分析表明，固定的班次时长并非最优。建议建立基于实时任务队列的动态排班模型。在测试高峰期或任务积压时，在人员疲劳度允许的范围内，适度延长单班工时如接近 12 小时；在测试后期或任务量较少时，可恢复为标准工时如 9 或 10 小时，以降低人为差错，保障测试质量。同时，在子系统检测结束后可以立即开始新装置的测试，不必等到 E 检测结束再开启新测试，提高时间利用率。

增加人力资源投入：为解决测试效率瓶颈，建议为关键测试小组如耗时最长的综合测试组 E 配置备用团队或轮换人员。这不仅能减少因人员疲劳导致的误判和漏判，还能在需要延长班次时实现人员无缝轮换。

（2）加强测试设备维护与更新

推行预测性维护策略：鉴于设备故障率随使用时间增加而显著上升，应建立设备状态监控系统，实时记录各测试设备的工作时长和运行状态。在设备累计工时接近 120 小时的关键节点前，进行预防性维护或更换，将其故障率始终维持在较低水平，从而减少非计划性中断，提高设备综合利用率。

建设测试设备快速更换机制：优化测试工位的设计，为关键测试设备开发模块化、快速插拔的接口。缩短因设备故障或定期更换带来的停机时间，将设备更换对测试流程的干扰降至最低。

（3）提升测手专业水平与规范性

强化标准化操作与专项培训：针对误判和漏判问题，定期对测手进行双盲测试和典型案例复盘培训，尤其强化对临界状态判读的一致性训练，从源头降低主观差错率。

引入自动化判读与决策辅助系统：在条件允许的情况下，开发基于算法的自动化测试结果初步判读系统。该系统可作为测手的辅助工具，对测试数据进行初步分析，提示潜在风险点，减少人为疏忽造成的漏判，同时也能对测手的判读结果进行二次校验，降低误判风险。

7 模型评价与改进

7.1 模型的优点

（1）模型充分结合实际，简化了装置各检测流程间的过渡条件，考虑了诸多重要因素得到合理的模型，如：设备的预防性更换策略、随机性事件的概率分布。这样得到的模型贴合实际测试流程，具有较高的应用价值。

（2）模型运用了离散事件仿真和随机抽样思想，抓住影响测试效率与质量的重要因素，将复杂的带随机干扰的目标优化问题转化为可模拟计算的模型，合理设置目标函数权重、模拟次数等参数，模型的输出结果全面符合题目要求，能有效解决测试任务的实际规划问题。

（3）本文使用的蒙特卡洛算法具有高效处理随机性、可靠性强等优点，所用的混合遗传算法可有效避免过早陷入局部最优，所用的 Sobol 算法和随机森林算法适用于强非线性及非单调响应特征的复杂问题。

7.2 模型的不足

（1）实际应用中, 测试设备精度的周期性漂移可能也是重要的因素, 但本文未能考虑到这些因素的影响, 一定程度上影响了模型的准确性。

（2）本文提出的模型对于题目给定的测试分队规模和装置数量使用效果较好，但由于时间问题没有对其他情况进行检验，如测试装置数量急剧增加、测试工位数量扩展等，当前模型的调度规则和策略可能无法自动适应变化的情况。

（3）实际上，问题二的目标函数不一定是简单的加权，但本文将其作为线性加权处理，忽略了实际复杂情况的影响。

7.3 模型的改进

（1）可以采用强化学习算法来替代静态的调度规则，使模型能够在线学习并适应随机环境，动态做出更优的调度决策，从而得到更智能、更鲁棒的规划方案.

（2）模型只是对测试流程进行了仿真模拟，但无法进行预测，可以进行基于仿真的在线滚动优化，在每个决策点利用简化仿真快速预测未来一段时间的系统行为，选择能使短期目标最优的调度动作。

参考文献

[1] 陈勇, 姚玉斌, 夏翔, et al. 考虑备自投的地区电网静态安全分析设计与应用 [J]. 电力系统自动化, 2004, 28(19): 5.

[2] 尤一琛, 王艳, 纪志成. 基于随机机器故障的柔性作业车间动态调度 [J]. 江苏大学学报:自然科学版, 2021, 42(6): 8.

[3] RIAD B, EL-ARKAM M, MAROUA K, et al. Design of Type 2 Fuzzy Logic Controller for FESTO Process Workstation [J]. ICCEIS 2022, 2023.

[4] 李素粉, 朱云龙, 尹朝万. 具有随机加工时间和机器故障的流水车间调度 [J]. 计算机集成制造系统, 2005, 11(10): 1425-9.

[5] YONG L, 刘勇, 王德才, et al. 离散事件系统仿真建模与仿真策略 [J]. 西南师范大学学报：自然科学版, 2005, 30(6): 7.

[6] 徐晟. 基于改进蒙特卡洛算法的配电网可靠性评估 [D]; 长沙理工大学, 2013.

[7] 刘志祥, 王凯, 杨小聪, et al. 基于模糊故障树和蒙特卡洛方法的智能铲运系统可靠性分析 [J]. 黄金科学技术, 2023, 31(3): 477-86.

[8] 李钢, 李金勇. 混合遗传算法求解车间作业调度问题 [J]. 天津大学学报：自然科学与工程技术版, 2003, 36(2): 4.

[9] 任启伟, 陈洋波, 周浩澜, et al. 基于 Sobol 法的 TOPMODEL 模型全局敏感性分析 [J]. 人民长江, 2010, 41(19): 5.

[10] 周红, 王鸣, 柴文轩, et al. 基于随机森林的北京城区臭氧敏感性分析 [J]. 环境科学, 2024, 45(5): 2497-506.

[11] Disturbance suppression in active magnetic bearings with adaptive control and extended state observer [J]. Proceedings of the Institution of Mechanical Engineers, Part I Journal of Systems and Control Engineering, 2020, 234(2): 272-84.

附　录

附录 A: 问题二主要代码

```
import sys
import subprocess
import importlib

# 自动安装所需的库
required_libraries = ['simpy', 'seaborn', 'tabulate']


def install_packages():
    """自动安装缺失的包"""
    for lib in required_libraries:
        try:
            importlib.import_module(lib)
            print(f"✓ {lib} 已安装")
        except ImportError:
            print(f"正在安装 {lib}...")
            subprocess.check_call([sys.executable, "-m", "pip", "install", lib])
            print(f"✓ {lib} 安装完成")


# 安装必要的库
print("=" * 60)
print("正在检查并安装必要的库...")
install_packages()
print("所有必要的库已就绪！")
print("=" * 60)

# 现在导入所需的库
import simpy
import numpy as np
import pandas as pd
from enum import Enum
import random


class TestResult(Enum):
    SUCCESS = 1
    FAILURE = 2
    FALSE_ALARM = 3    # 误判
    MISSED_DETECTION = 4    # 漏判
    EQUIPMENT_FAILURE = 5    # 设备故障


class Subsystem(Enum):
    A = 0
```

```
    B = 1
    C = 2
    E = 3   # 综合测试


class Device:
    def __init__(self, subsystem_type):
        self.subsystem_type = subsystem_type
        self.usage_hours = 0.0
        self.need_replacement = False

    def update_usage(self, hours):
        self.usage_hours += hours
        if self.usage_hours >= 240:
            self.need_replacement = True

    def get_failure_probability(self):
        """根据使用时间返回不同的故障概率"""
        if self.usage_hours < 120:
            failure_probs = {
                Subsystem.A: 0.03,
                Subsystem.B: 0.04,
                Subsystem.C: 0.02,
                Subsystem.E: 0.03
            }
        else:
            failure_probs = {
                Subsystem.A: 0.05,
                Subsystem.B: 0.07,
                Subsystem.C: 0.06,
                Subsystem.E: 0.05
            }
        return failure_probs[self.subsystem_type]

    def reset(self):
        self.usage_hours = 0.0
        self.need_replacement = False


class Assembly:
    assembly_count = 0

    def __init__(self):
        Assembly.assembly_count += 1
        self.id = Assembly.assembly_count
        self.test_results = {
            Subsystem.A: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.B: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.C: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.E: {'attempts': 0, 'status': None, 'retry_count': 0}
```

```
            }
            self.status = 'WAITING'
            self.start_time = None
            self.end_time = None
            self.max_retries = 1


class Workstation:
    def __init__(self, env, station_id):
            self.env = env
            self.station_id = station_id
            self.devices = {
                Subsystem.A: Device(Subsystem.A),
                Subsystem.B: Device(Subsystem.B),
                Subsystem.C: Device(Subsystem.C),
                Subsystem.E: Device(Subsystem.E)
            }
            self.resources = {
                Subsystem.A: simpy.Resource(env, capacity=1),
                Subsystem.B: simpy.Resource(env, capacity=1),
                Subsystem.C: simpy.Resource(env, capacity=1),
                Subsystem.E: simpy.Resource(env, capacity=1)
            }
            self.calibration_times = {
                Subsystem.A: 0.5,
                Subsystem.B: 1 / 3,
                Subsystem.C: 1 / 3,
                Subsystem.E: 2 / 3
            }
            self.test_times = {
                Subsystem.A: 2.5,
                Subsystem.B: 2.0,
                Subsystem.C: 2.5,
                Subsystem.E: 3.0
            }
            self.stats = {
                'total_tests': 0,
                'successful_tests': 0,
                'equipment_failures': 0,
                'false_alarms': 0,
                'missed_detections': 0
            }

    def test_subsystem(self, assembly, subsystem, is_retry=False):
            """在指定位置测试单个子系统，支持重试"""
            start_time = self.env.now

            with self.resources[subsystem].request() as request:
                yield request
```

```
            device = self.devices[subsystem]

            if device.need_replacement:
                print(f'Time {self.env.now:.2f}: {subsystem.name}测试设备需要更换，
进行更换操作')
                replacement_time = self.calibration_times[subsystem]
                yield self.env.timeout(replacement_time)
                device.reset()
                print(f'Time {self.env.now:.2f}: {subsystem.name}测试设备更换完成')

            if not is_retry:
                assembly.test_results[subsystem]['attempts'] += 1

            needs_calibration = device.usage_hours == 0

            if needs_calibration:
                calibration_time = self.calibration_times[subsystem]
                yield self.env.timeout(calibration_time)
                print(f'Time {self.env.now:.2f}: {subsystem.name}测试设备校准完成')

            current_failure_prob = device.get_failure_probability()
            if random.random() < current_failure_prob:
                self.stats['equipment_failures'] += 1
                device.need_replacement = True
                print(f'Time {self.env.now:.2f}: {subsystem.name}测试设备发生故障')
                return TestResult.EQUIPMENT_FAILURE

            test_time = self.test_times[subsystem]
            yield self.env.timeout(test_time)

            total_usage_time = (self.calibration_times[subsystem] if needs_calibration else
0) + test_time
            device.update_usage(total_usage_time)

            if not is_retry:
                self.stats['total_tests'] += 1

            end_time = self.env.now
            actual_time = end_time - start_time

            if not hasattr(self, 'subsystem_actual_times'):
                self.subsystem_actual_times = {sub: 0 for sub in [Subsystem.A, Subsys-
tem.B, Subsystem.C, Subsystem.E]}
            self.subsystem_actual_times[subsystem] += actual_time

            if not hasattr(self, 'device_usage_stats'):
                self.device_usage_stats = {sub: [] for sub in [Subsystem.A, Subsystem.B,
Subsystem.C, Subsystem.E]}
            self.device_usage_stats[subsystem].append(device.usage_hours)
```

```
            return self.determine_test_result(subsystem, device)

    def determine_test_result(self, subsystem, device):
        """判断测试结果"""
        if subsystem == Subsystem.E:
            if random.random() < 0.076:
                return TestResult.FAILURE
            else:
                self.stats['successful_tests'] += 1
                return TestResult.SUCCESS
        else:
            subsystem_fault_prob = [0.025, 0.03, 0.02][subsystem.value]
            has_fault = random.random() < subsystem_fault_prob

            tester_error_prob = [0.03, 0.04, 0.02][subsystem.value]
            made_error = random.random() < tester_error_prob

            if has_fault:
                if made_error and random.random() < 0.5:
                    self.stats['missed_detections'] += 1
                    return TestResult.MISSED_DETECTION
                else:
                    return TestResult.FAILURE
            else:
                if made_error and random.random() < 0.5:
                    self.stats['false_alarms'] += 1
                    return TestResult.FALSE_ALARM
                else:
                    self.stats['successful_tests'] += 1
                    return TestResult.SUCCESS


class TestFacility:
    def __init__(self, env):
        self.env = env
        self.workstations = [
            Workstation(env, "Workstation_1"),
            Workstation(env, "Workstation_2")
        ]
        self.assembly_queue = simpy.Store(env)
        self.completed_assemblies = []
        self.transport_time = 0.5
        # 使用信号量控制装置处理
        self.workstation_semaphores = [
            simpy.Resource(env, capacity=1),  # 工作台 1 信号量
            simpy.Resource(env, capacity=1)  # 工作台 2 信号量
        ]

    def assembly_generator(self, num_assemblies, interval=2):
        """生成待测试的装置"""
```

```
            for i in range(num_assemblies):
                assembly = Assembly()
                assembly.start_time = self.env.now
                yield self.assembly_queue.put(assembly)
                print(f'Time {self.env.now:.2f}: Assembly {assembly.id} arrived')
                yield self.env.timeout(interval)

    def transport_in(self):
        yield self.env.timeout(self.transport_time)

    def transport_out(self):
        yield self.env.timeout(self.transport_time)

    def test_abc_concurrently(self, assembly, workstation, is_retry=False):
        """在工作台上同时测试A、B、C三个子系统，支持重试"""
        print(
            f'Time {self.env.now:.2f}: Assembly {assembly.id} 在 {workstation.station_id} 开始 ABC 测试{"（重试）" if is_retry else ""}')

        test_processes = []
        for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C]:
            process = self.env.process(workstation.test_subsystem(assembly, subsystem, is_retry))
            test_processes.append(process)

        results = yield simpy.AllOf(self.env, test_processes)

        abc_results = {}
        for i, subsystem in enumerate([Subsystem.A, Subsystem.B, Subsystem.C]):
            abc_results[subsystem] = results[test_processes[i]]
            assembly.test_results[subsystem]['status'] = results[test_processes[i]]
            if is_retry:
                assembly.test_results[subsystem]['retry_count'] += 1

        return abc_results

    def test_e_subsystem(self, assembly, workstation, is_retry=False):
        """在工作台上测试E子系统，支持重试"""
        print(
            f'Time {self.env.now:.2f}: Assembly {assembly.id} 在 {workstation.station_id} 开始 E 测试{"（重试）" if is_retry else ""}')

        result = yield self.env.process(workstation.test_subsystem(assembly, Subsystem.E, is_retry))
        assembly.test_results[Subsystem.E]['status'] = result
        if is_retry:
            assembly.test_results[Subsystem.E]['retry_count'] += 1
        return result
```

```
    def check_abc_passed(self, abc_results):
        """检查A、B、C三个子系统是否都通过"""
        for result in abc_results.values():
            if result != TestResult.SUCCESS:
                return False
        return True

    def process_assembly(self, assembly, workstation_index):
        """处理单个装置的完整测试流程"""
        print(f'Time {self.env.now:.2f}: Assembly {assembly.id} 开始处理，使用工作台{workstation_index + 1}')

        # 使用信号量控制工作台访问
        with self.workstation_semaphores[workstation_index].request() as req:
            yield req

            # 运输进入
            yield self.env.process(self.transport_in())

            workstation = self.workstations[workstation_index]

            # ABC 测试（支持重试）
            abc_passed = False
            abc_retry_count = 0

            while not abc_passed and abc_retry_count <= assembly.max_retries:
                abc_results = yield self.env.process(
                    self.test_abc_concurrently(assembly, workstation, abc_retry_count > 0))

                if self.check_abc_passed(abc_results):
                    abc_passed = True
                else:
                    abc_retry_count += 1
                    if abc_retry_count > assembly.max_retries:
                        assembly.status = 'FAILED'
                        print(f'Time {self.env.now:.2f}: Assembly {assembly.id} ABC测试重试次数超限，测试失败')
                        break
                    else:
                        print(
                            f'Time {self.env.now:.2f}: Assembly {assembly.id} ABC 测试失败，进行第{abc_retry_count}次重试')
                        yield self.env.timeout(0.5)

            if not abc_passed:
                # 运输离开
                yield self.env.process(self.transport_out())
                assembly.end_time = self.env.now
```

```
                    self.completed_assemblies.append(assembly)
                    return

                # E 测试（支持重试）
                e_passed = False
                e_retry_count = 0

                while not e_passed and e_retry_count <= assembly.max_retries:
                    e_result = yield self.env.process(self.test_e_subsystem(assembly, work-
station, e_retry_count > 0))

                    if e_result == TestResult.SUCCESS:
                        e_passed = True
                        assembly.status = 'PASSED'
                        print(f'Time {self.env.now:.2f}: Assembly {assembly.id} 测试通过')
                    else:
                        e_retry_count += 1
                        if e_retry_count > assembly.max_retries:
                            assembly.status = 'FAILED'
                            print(f'Time {self.env.now:.2f}: Assembly {assembly.id} E 测试
重试次数超限，测试失败')
                            break
                        else:
                            print(f'Time {self.env.now:.2f}: Assembly {assembly.id} E 测试
失败，进行第{e_retry_count}次重试')
                            yield self.env.timeout(0.5)

                # 记录完成时间
                assembly.end_time = self.env.now
                self.completed_assemblies.append(assembly)

                # 运输离开
                yield self.env.process(self.transport_out())

                print(f'Time {self.env.now:.2f}: Assembly {assembly.id} 已完全离开，工作台
{workstation_index + 1}空闲')

    def run_testing(self, num_devices_to_test):
        """检测指定数量的装置"""
        # 生成装置
        self.env.process(self.assembly_generator(num_devices_to_test))

        # 为每个工作台创建独立的处理进程
        for i in range(2):  # 两个工作台
            self.env.process(self.workstation_process(i, num_devices_to_test))

        # 等待所有装置完成
        while len(self.completed_assemblies) < num_devices_to_test:
```

```
                yield self.env.timeout(1)  # 每秒检查一次完成情况

    def workstation_process(self, workstation_index, num_devices_to_test):
        """工作台处理进程 - 每个工作台独立运行"""
        while len(self.completed_assemblies) < num_devices_to_test:
            # 等待装置到达
            assembly = yield self.assembly_queue.get()

            # 处理装置
            yield self.env.process(self.process_assembly(assembly, workstation_index))


def simulate(num_devices_to_test=100, shift_hours=12):
    """运行仿真 - 检测指定数量的装置"""
    env = simpy.Environment()
    facility = TestFacility(env)

    # 直接启动测试过程
    env.process(facility.run_testing(num_devices_to_test))

    env.run(until=1000)

    passed_count = sum(1 for a in facility.completed_assemblies if a.status == 'PASSED')
    completed_assemblies = [a for a in facility.completed_assemblies if a.end_time is not
None]
    total_time = max(a.end_time for a in completed_assemblies) if completed_assemblies else
0

    total_tests = 0
    false_alarms = 0
    missed_detections = 0
    equipment_failures = 0

    for workstation in facility.workstations:
        total_tests += workstation.stats['total_tests']
        false_alarms += workstation.stats['false_alarms']
        missed_detections += workstation.stats['missed_detections']
        equipment_failures += workstation.stats['equipment_failures']

    false_alarm_prob = false_alarms / total_tests if total_tests > 0 else 0
    missed_detection_prob = missed_detections / total_tests if total_tests > 0 else 0

    yxb_ratios = {}
    total_sim_time = max(
        a.end_time for a in facility.completed_assemblies if a.end_time is not None) if com-
pleted_assemblies else 0

    for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C, Subsystem.E]:
        total_pure_test_time = 0
```

```
            for workstation in facility.workstations:
                if hasattr(workstation, 'pure_test_times'):
                    total_pure_test_time += workstation.pure_test_times.get(subsystem, 0)
                else:
                    total_test_time = workstation.subsystem_actual_times.get(subsystem, 0) if
hasattr(workstation,

'subsystem_actual_times') else 0
                    total_pure_test_time += total_test_time * 0.7

            yxb_ratios[subsystem.name] = total_pure_test_time / total_sim_time if to-
tal_sim_time > 0 else 0

        total_retries = 0
        for assembly in facility.completed_assemblies:
            for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C, Subsystem.E]:
                total_retries += assembly.test_results[subsystem]['retry_count']

        avg_retries_per_assembly = total_retries / len(
            facility.completed_assemblies) if facility.completed_assemblies else 0

        device_usage_info = {}
        for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C, Subsystem.E]:
            total_usage = 0
            usage_counts = 0
            for workstation in facility.workstations:
                if hasattr(workstation, 'device_usage_stats') and subsystem in workstation.de-
vice_usage_stats:
                    total_usage += sum(workstation.device_usage_stats[subsystem])
                    usage_counts += len(workstation.device_usage_stats[subsystem])

            avg_usage = total_usage / usage_counts if usage_counts > 0 else 0
            device_usage_info[subsystem.name] = {
                'avg_usage': avg_usage,
                'total_replacements': sum(1 for ws in facility.workstations if ws.devices[subsys-
tem].need_replacement)
            }

        return {
            'total_days': total_time / shift_hours if shift_hours > 0 else 0,
            'passed_count': passed_count,
            'false_alarm_prob': false_alarm_prob,
            'missed_detection_prob': missed_detection_prob,
            'yxb_ratios': yxb_ratios,
            'total_retries': total_retries,
            'avg_retries_per_assembly': avg_retries_per_assembly,
            'device_usage_info': device_usage_info,
            'equipment_failures': equipment_failures
        }
```

```
import sys
import subprocess
import importlib
from enum import Enum
import random
import simpy
import numpy as np
import pandas as pd
from tqdm import tqdm  # 用于显示进度条


# ... [保留原有的所有类和函数定义] ...

def monte_carlo_simulation(num_simulations=100, num_devices_per_sim=100, shift_hours=12):
    """执行蒙特卡洛模拟"""
    results = {
        'total_days': [],
        'passed_count': [],
        'false_alarm_prob': [],
        'missed_detection_prob': [],
        'equipment_failures': [],
        'total_retries': [],
        'avg_retries_per_assembly': []
    }

    # 各子系统的有效工作时间比
    yxb_ratios = {
        'A': [],
        'B': [],
        'C': [],
        'E': []
    }

    # 设备使用情况
    device_usage = {
        'A': {'avg_usage': [], 'replacements': []},
        'B': {'avg_usage': [], 'replacements': []},
        'C': {'avg_usage': [], 'replacements': []},
        'E': {'avg_usage': [], 'replacements': []}
    }

    print(f"正在进行蒙特卡洛模拟 ({num_simulations} 次)...")
    for _ in tqdm(range(num_simulations)):
        sim_results = simulate(num_devices_to_test=num_devices_per_sim, shift_hours=shift_hours)

        # 收集基本结果
```

```
            for key in results:
                if key in sim_results:
                    results[key].append(sim_results[key])

            # 收集有效工作时间比
            for subsystem, ratio in sim_results['yxb_ratios'].items():
                yxb_ratios[subsystem].append(ratio)

            # 收集设备使用情况
            for subsystem, info in sim_results['device_usage_info'].items():
                device_usage[subsystem]['avg_usage'].append(info['avg_usage'])
                device_usage[subsystem]['replacements'].append(info['total_replacements'])

        # 计算统计量
        def calculate_stats(data):
            if isinstance(data, dict):
                return {k: calculate_stats(v) for k, v in data.items()}
            elif isinstance(data, list):
                return {
                    'mean': np.mean(data),
                    'std': np.std(data),
                    'min': np.min(data),
                    'max': np.max(data),
                    'median': np.median(data),
                    '5th_percentile': np.percentile(data, 5),
                    '95th_percentile': np.percentile(data, 95)
                }
            else:
                return data

        # 分析结果
        analysis = {
            'basic_metrics': calculate_stats({k: v for k, v in results.items()}),
            'yxb_ratios': calculate_stats(yxb_ratios),
            'device_usage': calculate_stats(device_usage)
        }

        return analysis


# def main():
#     """主函数"""
#     print("=" * 60)
#     print("单次模拟结果:")
#     results_q2 = simulate(num_devices_to_test=100, shift_hours=12)
#
#     print(f"平均天数: {results_q2['total_days']:.2f}")
#     print(f"通过数量: {results_q2['passed_count']}")
#     print(f"误判概率: {results_q2['false_alarm_prob']:.4f}")
```

```
#         print(f"漏判概率: {results_q2['missed_detection_prob']:.4f}")
#         print(f"设备故障次数: {results_q2['equipment_failures']}")
#         print(f"总重试次数: {results_q2['total_retries']}")
#         print(f"平均每个装置重试次数: {results_q2['avg_retries_per_assembly']:.2f}")
#
#         print("\n 设备使用情况:")
#         for subsystem, info in results_q2['device_usage_info'].items():
#             print(f"   {subsystem}: 平均使用时间={info['avg_usage']:.2f}小时, 更换次数
={info['total_replacements']}")
#
#         print("\n 有效工作时间比:")
#         for subsystem, ratio in results_q2['yxb_ratios'].items():
#             print(f"   {subsystem}: {ratio:.4f}")
#
#         print("\n" + "=" * 60)
#         print("蒙特卡洛模拟结果 (100 次重复):")
#             mc_results = monte_carlo_simulation(num_simulations=1000, num_de-
vices_per_sim=100)
#
#         def print_stat(name, stat):
#             print(f"{name}:")
#             print(f"   平均值: {stat['mean']:.4f}")
#             print(f"   标准差: {stat['std']:.4f}")
#             print(f"   范围: [{stat['min']:.4f}, {stat['max']:.4f}]")
#             print(f"   中位数: {stat['median']:.4f}")
#                 print(f"   90%置信区间: [{stat['5th_percentile']:.4f}, {stat['95th_percen-
tile']:.4f}]")
#
#         print("\n 基本指标:")
#         for metric, stat in mc_results['basic_metrics'].items():
#             print(f"\n{metric.replace('_', ' ').title()}")
#             print_stat(metric, stat)
#
#         print("\n 有效工作时间比:")
#         for subsystem, stat in mc_results['yxb_ratios'].items():
#             print(f"\n 子系统 {subsystem}")
#             print_stat("YXB 比率", stat)
#
#         print("\n 设备使用情况:")
#         for subsystem, stats in mc_results['device_usage'].items():
#             print(f"\n 子系统 {subsystem} 设备:")
#             for stat_name, stat in stats.items():
#                 print(f"\n{stat_name.replace('_', ' ').title()}")
#                 print_stat(stat_name, stat)

import matplotlib.pyplot as plt
import seaborn as sns
```

```
# def plot_monte_carlo_results(mc_results):
#     """绘制蒙特卡洛模拟结果的图表"""
#     # 设置样式 - 使用 seaborn 的默认样式
#     sns.set_style("whitegrid")
#     plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
#     plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号
#
#     # 1. 基本指标柱状图
#     basic_metrics = mc_results['basic_metrics']
#     metrics_names = {
#         'total_days': '平均测试天数',
#         'passed_count': '通过测试数量',
#         'false_alarm_prob': '误判概率',
#         'missed_detection_prob': '漏判概率',
#         'equipment_failures': '设备故障次数',
#         'total_retries': '总重试次数',
#         'avg_retries_per_assembly': '平均重试次数'
#     }
#
#     fig, axes = plt.subplots(3, 3, figsize=(18, 15))
#     axes = axes.flatten()
#
#     for i, (metric, cn_name) in enumerate(metrics_names.items()):
#         data = basic_metrics[metric]
#         ax = axes[i]
#
#         # 绘制柱状图表示范围
#         ax.bar(0, data['max'] - data['min'], bottom=data['min'],
#                width=0.6, color='skyblue', alpha=0.7, label='范围')
#
#         # 标记平均值
#         ax.scatter(0, data['mean'], color='red', s=100,
#                    label=f'均值: {data["mean"]:.2f}', zorder=5)
#
#         # 标记中位数
#         ax.scatter(0, data['median'], color='green', s=100,
#                    marker='x', label=f'中位数: {data["median"]:.2f}', zorder=5)
#
#         # 添加 90%置信区间
#         ax.plot([-0.3, 0.3], [data['5th_percentile'], data['5th_percentile']],
#                 'k--', linewidth=1, label='90%置信区间')
#         ax.plot([-0.3, 0.3], [data['95th_percentile'], data['95th_percentile']],
#                 'k--', linewidth=1)
#
#         ax.set_title(cn_name)
#         ax.set_xticks([])
#         ax.legend()
```

```
#
#       # 隐藏多余的子图
#       for j in range(i + 1, len(axes)):
#           axes[j].axis('off')
#
#       plt.tight_layout()
#       plt.savefig('basic_metrics.png', dpi=300, bbox_inches='tight')
#       plt.close()
#
#       # 2. 有效工作时间比雷达图
#       yxb_data = mc_results['yxb_ratios']
#       subsystems = list(yxb_data.keys())
#       stats = ['mean', '5th_percentile', '95th_percentile']
#
#       angles = np.linspace(0, 2 * np.pi, len(subsystems), endpoint=False)
#       angles = np.concatenate((angles, [angles[0]]))
#
#       fig = plt.figure(figsize=(10, 10))
#       ax = fig.add_subplot(111, polar=True)
#
#       for stat in stats:
#           values = [yxb_data[sub][stat] for sub in subsystems]
#           values += [values[0]]
#           ax.plot(angles, values, 'o-', linewidth=2,
#                   label=f'{stat} (均值: {np.mean(values):.3f})')
#           ax.fill(angles, values, alpha=0.1)
#
#       ax.set_thetagrids(angles[:-1] * 180 / np.pi, subsystems)
#       ax.set_title('各子系统有效工作时间比比较', pad=20)
#       ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
#       plt.tight_layout()
#       plt.savefig('yxb_ratios.png', dpi=300, bbox_inches='tight')
#       plt.close()
#
#       # 3. 设备使用情况分组柱状图
#       usage_data = mc_results['device_usage']
#       fig, axes = plt.subplots(1, 2, figsize=(15, 6))
#
#       # 平均使用时间
#       ax = axes[0]
#       x = np.arange(len(usage_data))
#       for i, (sub, data) in enumerate(usage_data.items()):
#           ax.bar(i, data['avg_usage']['mean'],
#                  yerr=data['avg_usage']['std'],
#                  capsize=5, label=sub)
#       ax.set_xticks(x)
#       ax.set_xticklabels(usage_data.keys())
#       ax.set_title('各设备平均使用时间(小时)')
#       ax.set_ylabel('小时')
```

```
#         ax.legend()
#
#         # 更换次数
#         ax = axes[1]
#         for i, (sub, data) in enumerate(usage_data.items()):
#               ax.bar(i, data['replacements']['mean'],
#                           yerr=data['replacements']['std'],
#                           capsize=5, label=sub)
#         ax.set_xticks(x)
#         ax.set_xticklabels(usage_data.keys())
#         ax.set_title('各设备平均更换次数')
#         ax.set_ylabel('次数')
#         ax.legend()
#
#         plt.tight_layout()
#         plt.savefig('device_usage.png', dpi=300, bbox_inches='tight')
#         plt.close()

def plot_monte_carlo_results(mc_results):
    """绘制蒙特卡洛模拟结果的图表"""
    # 设置样式 - 使用 seaborn 的默认样式
    sns.set_style("whitegrid")
    plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
    plt.rcParams['axes.unicode_minus'] = False  # 用来正常显示负号

    # 1. 基本指标单独柱状图
    basic_metrics = mc_results['basic_metrics']
    metrics_names = {
        'total_days': '平均测试天数',
        'passed_count': '通过测试数量',
        'false_alarm_prob': '误判概率',
        'missed_detection_prob': '漏判概率',
        'equipment_failures': '设备故障次数',
        'total_retries': '总重试次数',
        'avg_retries_per_assembly': '平均重试次数'
    }

    # 为每个指标创建单独的图表
    for metric, cn_name in metrics_names.items():
        if metric not in basic_metrics:
            continue

        data = basic_metrics[metric]

        plt.figure(figsize=(8, 6))

        # 绘制柱状图表示范围
        plt.bar(0, data['max'] - data['min'], bottom=data['min'],
```

```
                width=0.6, color='skyblue', alpha=0.7, label='范围')

        # 标记平均值
        plt.scatter(0, data['mean'], color='red', s=100,
                    label=f'均值: {data["mean"]:.2f}', zorder=5)

        # 标记中位数
        plt.scatter(0, data['median'], color='green', s=100,
                    marker='x', label=f'中位数: {data["median"]:.2f}', zorder=5)

        # 添加 90%置信区间
        plt.plot([-0.3, 0.3], [data['5th_percentile'], data['5th_percentile']],
                 'k--', linewidth=1, label='90%置信区间')
        plt.plot([-0.3, 0.3], [data['95th_percentile'], data['95th_percentile']],
                 'k--', linewidth=1)

        plt.title(cn_name)
        plt.xticks([])
        plt.legend()
        plt.tight_layout()

        # 保存为单独的文件
        plt.savefig(f'{metric}.png', dpi=600, bbox_inches='tight')
        plt.close()

# 2. 有效工作时间比雷达图
yxb_data = mc_results['yxb_ratios']
subsystems = list(yxb_data.keys())
stats = ['mean', '5th_percentile', '95th_percentile']

angles = np.linspace(0, 2 * np.pi, len(subsystems), endpoint=False)
angles = np.concatenate((angles, [angles[0]]))

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, polar=True)

for stat in stats:
        values = [yxb_data[sub][stat] for sub in subsystems]
        values += [values[0]]
        ax.plot(angles, values, 'o-', linewidth=2,
                label=f'{stat} (均值: {np.mean(values):.3f})')
        ax.fill(angles, values, alpha=0.1)

ax.set_thetagrids(angles[:-1] * 180 / np.pi, subsystems)
ax.set_title('各子系统有效工作时间比比较', pad=20)
ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
plt.tight_layout()
plt.savefig('yxb_ratios.png', dpi=600, bbox_inches='tight')
plt.close()
```

```
    # 3. 设备使用情况分组柱状图
    usage_data = mc_results['device_usage']
    fig, axes = plt.subplots(1, 2, figsize=(15, 6))

    # 平均使用时间
    ax = axes[0]
    x = np.arange(len(usage_data))
    for i, (sub, data) in enumerate(usage_data.items()):
        ax.bar(i, data['avg_usage']['mean'],
               yerr=data['avg_usage']['std'],
               capsize=5, label=sub)
    ax.set_xticks(x)
    ax.set_xticklabels(usage_data.keys())
    ax.set_title('各设备平均使用时间(小时)')
    ax.set_ylabel('小时')
    ax.legend()

    # 更换次数
    ax = axes[1]
    for i, (sub, data) in enumerate(usage_data.items()):
        ax.bar(i, data['replacements']['mean'],
               yerr=data['replacements']['std'],
               capsize=5, label=sub)
    ax.set_xticks(x)
    ax.set_xticklabels(usage_data.keys())
    ax.set_title('各设备平均更换次数')
    ax.set_ylabel('次数')
    ax.legend()

    plt.tight_layout()
    plt.savefig('device_usage.png', dpi=600, bbox_inches='tight')
    plt.close()



def main():
    """主函数"""
    # 创建并打开一个 txt 文件用于保存结果
    with open('monte_carlo_results.txt', 'w', encoding='utf-8') as f:
        print("=" * 60, file=f)
        print("单次模拟结果:", file=f)
        results_q2 = simulate(num_devices_to_test=100, shift_hours=12)

        # 使用中文指标名称
        print(f"平均测试天数: {results_q2['total_days']:.2f}", file=f)
        print(f"通过测试数量: {results_q2['passed_count']}", file=f)
```

```
        print(f"误判概率: {results_q2['false_alarm_prob']:.4f}", file=f)
        print(f"漏判概率: {results_q2['missed_detection_prob']:.4f}", file=f)
        print(f"设备故障次数: {results_q2['equipment_failures']}", file=f)
        print(f"总重试次数: {results_q2['total_retries']}", file=f)
        print(f"平均每个装置重试次数: {results_q2['avg_retries_per_assembly']:.2f}",
file=f)

        print("\n 设备使用情况:", file=f)
        for subsystem, info in results_q2['device_usage_info'].items():
            print(f"  {subsystem}: 平均使用时间={info['avg_usage']:.2f}小时, 更换次
数={info['total_replacements']}",
                  file=f)

        print("\n 有效工作时间比:", file=f)
        for subsystem, ratio in results_q2['yxb_ratios'].items():
            print(f"  {subsystem}: {ratio:.4f}", file=f)

        print("\n" + "=" * 60, file=f)
        print("蒙特卡洛模拟结果 (1000 次重复):", file=f)
        mc_results = monte_carlo_simulation(num_simulations=1000, num_de-
vices_per_sim=100)

        def print_stat(name, stat, file_obj):
            print(f"{name}:", file=file_obj)
            print(f"  平均值: {stat['mean']:.4f}", file=file_obj)
            print(f"  标准差: {stat['std']:.4f}", file=file_obj)
            print(f"  范围: [{stat['min']:.4f}, {stat['max']:.4f}]", file=file_obj)
            print(f"  中位数: {stat['median']:.4f}", file=file_obj)
            print(f"  90% 置信区间: [{stat['5th_percentile']:.4f}, {stat['95th_percen-
tile']:.4f}]", file=file_obj)

        # 使用中文指标名称
        print("\n 基本指标:", file=f)
        print("\n 平均测试天数", file=f)
        print_stat("平均测试天数", mc_results['basic_metrics']['total_days'], f)

        print("\n 通过测试数量", file=f)
        print_stat("通过测试数量", mc_results['basic_metrics']['passed_count'], f)

        print("\n 误判概率", file=f)
        print_stat("误判概率", mc_results['basic_metrics']['false_alarm_prob'], f)

        print("\n 漏判概率", file=f)
        print_stat("漏判概率", mc_results['basic_metrics']['missed_detection_prob'], f)

        print("\n 设备故障次数", file=f)
        print_stat("设备故障次数", mc_results['basic_metrics']['equipment_failures'], f)
```

```
            print("\n 总重试次数", file=f)
            print_stat("总重试次数", mc_results['basic_metrics']['total_retries'], f)

            print("\n 平均每个装置重试次数", file=f)
            print_stat(" 平 均 每 个 装 置 重 试 次 数 ", mc_results['basic_metrics']['avg_re-
tries_per_assembly'], f)

            print("\n 有效工作时间比:", file=f)
            for subsystem, stat in mc_results['yxb_ratios'].items():
                print(f"\n 子系统 {subsystem}", file=f)
                print_stat("YXB 比率", stat, f)

            print("\n 设备使用情况:", file=f)
            for subsystem, stats in mc_results['device_usage'].items():
                print(f"\n 子系统 {subsystem} 设备:", file=f)
                for stat_name, stat in stats.items():
                    # 设备使用情况的中文名称
                    cn_name = {
                        'avg_usage': '平均使用时间',
                        'total_replacements': '更换次数'
                    }.get(stat_name, stat_name)
                    print(f"\n{cn_name}", file=f)
                    print_stat(cn_name, stat, f)

        # 同时在控制台也输出结果
        print("模拟完成，结果已保存到 monte_carlo_results.txt 文件")

        # 添加绘图功能
        print("\n 正在生成可视化图表...")
        plot_monte_carlo_results(mc_results)
        print("可视化图表已保存为 PNG 文件: basic_metrics.png, yxb_ratios.png, device_us-
age.png, correlation_heatmap.png")


if __name__ == "__main__":
        main()
```

附录 B：问题三主要代码

```
import sys
import subprocess
import importlib
import random
import simpy
import numpy as np
from enum import Enum
from tqdm import tqdm
```

```
from tabulate import tabulate
from deap import base, creator, tools, algorithms
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

# 设置中文字体
rcParams['font.sans-serif'] = ['SimHei', 'DejaVu Sans']
rcParams['axes.unicode_minus'] = False

# 自动安装所需的库
required_libraries = ['simpy', 'numpy', 'tqdm', 'tabulate', 'deap', 'matplotlib', 'seaborn']


def install_packages():
    """自动安装缺失的包"""
    for lib in required_libraries:
        try:
            importlib.import_module(lib)
            print(f"✓ {lib} 已安装")
        except ImportError:
            print(f"正在安装 {lib}...")
            subprocess.check_call([sys.executable, "-m", "pip", "install", lib])
            print(f"✓ {lib} 安装完成")


# 安装必要的库
print("=" * 60)
print("正在检查并安装必要的库...")
install_packages()
print("所有必要的库已就绪！")
print("=" * 60)


class TestResult(Enum):
    SUCCESS = 1
    FAILURE = 2
    FALSE_ALARM = 3  # 误判
    MISSED_DETECTION = 4  # 漏判
    EQUIPMENT_FAILURE = 5  # 设备故障


class Subsystem(Enum):
    A = 0
    B = 1
    C = 2
    E = 3  # 综合测试
```

```
class Device:
    def __init__(self, subsystem_type):
        self.subsystem_type = subsystem_type
        self.usage_hours = 0.0
        self.need_replacement = False

    def update_usage(self, hours):
        self.usage_hours += hours
        if self.usage_hours >= 240:
            self.need_replacement = True

    def get_failure_probability(self):
        """根据使用时间返回不同的故障概率"""
        if self.usage_hours < 120:
            failure_probs = {
                Subsystem.A: 0.03,
                Subsystem.B: 0.04,
                Subsystem.C: 0.02,
                Subsystem.E: 0.03
            }
        else:
            failure_probs = {
                Subsystem.A: 0.05,
                Subsystem.B: 0.07,
                Subsystem.C: 0.06,
                Subsystem.E: 0.05
            }
        return failure_probs[self.subsystem_type]

    def reset(self):
        self.usage_hours = 0.0
        self.need_replacement = False


class Assembly:
    assembly_count = 0

    def __init__(self):
        Assembly.assembly_count += 1
        self.id = Assembly.assembly_count
        self.test_results = {
            Subsystem.A: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.B: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.C: {'attempts': 0, 'status': None, 'retry_count': 0},
            Subsystem.E: {'attempts': 0, 'status': None, 'retry_count': 0}
        }
        self.status = 'WAITING'
        self.start_time = None
        self.end_time = None
```

```
            self.max_retries = 1


class Workstation:
    def __init__(self, env, station_id):
            self.env = env
            self.station_id = station_id
            self.devices = {
                Subsystem.A: Device(Subsystem.A),
                Subsystem.B: Device(Subsystem.B),
                Subsystem.C: Device(Subsystem.C),
                Subsystem.E: Device(Subsystem.E)
            }
            self.resources = {
                Subsystem.A: simpy.Resource(env, capacity=1),
                Subsystem.B: simpy.Resource(env, capacity=1),
                Subsystem.C: simpy.Resource(env, capacity=1),
                Subsystem.E: simpy.Resource(env, capacity=1)
            }
            self.calibration_times = {
                Subsystem.A: 0.5,
                Subsystem.B: 1 / 3,
                Subsystem.C: 1 / 3,
                Subsystem.E: 2 / 3
            }
            self.test_times = {
                Subsystem.A: 2.5,
                Subsystem.B: 2.0,
                Subsystem.C: 2.5,
                Subsystem.E: 3.0
            }
            self.stats = {
                'total_tests': 0,
                'successful_tests': 0,
                'equipment_failures': 0,
                'false_alarms': 0,
                'missed_detections': 0
            }

    def test_subsystem(self, assembly, subsystem, is_retry=False):
            """在指定位置测试单个子系统，支持重试"""
            start_time = self.env.now

            with self.resources[subsystem].request() as request:
                yield request

                device = self.devices[subsystem]

                if device.need_replacement:
                    replacement_time = self.calibration_times[subsystem]
```

```
                    yield self.env.timeout(replacement_time)
                    device.reset()

                if not is_retry:
                    assembly.test_results[subsystem]['attempts'] += 1

                needs_calibration = device.usage_hours == 0

                if needs_calibration:
                    calibration_time = self.calibration_times[subsystem]
                    yield self.env.timeout(calibration_time)

                current_failure_prob = device.get_failure_probability()
                if random.random() < current_failure_prob:
                    self.stats['equipment_failures'] += 1
                    device.need_replacement = True
                    return TestResult.EQUIPMENT_FAILURE

                test_time = self.test_times[subsystem]
                yield self.env.timeout(test_time)

                total_usage_time = (self.calibration_times[subsystem] if needs_calibration else
0) + test_time
                device.update_usage(total_usage_time)

                if not is_retry:
                    self.stats['total_tests'] += 1

                end_time = self.env.now
                actual_time = end_time - start_time

                if not hasattr(self, 'subsystem_actual_times'):
                    self.subsystem_actual_times = {sub: 0 for sub in [Subsystem.A, Subsys-
tem.B, Subsystem.C, Subsystem.E]}
                self.subsystem_actual_times[subsystem] += actual_time

                return self.determine_test_result(subsystem, device)

    def determine_test_result(self, subsystem, device):
        """判断测试结果"""
        if subsystem == Subsystem.E:
            if random.random() < 0.076:
                return TestResult.FAILURE
            else:
                self.stats['successful_tests'] += 1
                return TestResult.SUCCESS
        else:
            subsystem_fault_prob = [0.025, 0.03, 0.02][subsystem.value]
            has_fault = random.random() < subsystem_fault_prob
```

```
                tester_error_prob = [0.03, 0.04, 0.02][subsystem.value]
                made_error = random.random() < tester_error_prob

                if has_fault:
                    if made_error and random.random() < 0.5:
                        self.stats['missed_detections'] += 1
                        return TestResult.MISSED_DETECTION
                    else:
                        return TestResult.FAILURE
                else:
                    if made_error and random.random() < 0.5:
                        self.stats['false_alarms'] += 1
                        return TestResult.FALSE_ALARM
                    else:
                        self.stats['successful_tests'] += 1
                        return TestResult.SUCCESS


class ShiftTestFacility:
    def __init__(self, env, shift_hours=12):
        self.env = env
        self.workstations = [
            Workstation(env, "Workstation_1"),
            Workstation(env, "Workstation_2")
        ]
        self.assembly_queue = simpy.Store(env)
        self.completed_assemblies = []
        self.transport_time = 0.5
        self.workstation_semaphores = [
            simpy.Resource(env, capacity=1),
            simpy.Resource(env, capacity=1)
        ]
        self.shift_hours = shift_hours
        self.current_shift = 0
        self.shift_change_event = env.event()

    def assembly_generator(self, num_assemblies, interval=2):
        """生成待测试的装置"""
        for i in range(num_assemblies):
            assembly = Assembly()
            assembly.start_time = self.env.now
            yield self.assembly_queue.put(assembly)
            yield self.env.timeout(interval)

    def transport_in(self):
        yield self.env.timeout(self.transport_time)

    def transport_out(self):
        yield self.env.timeout(self.transport_time)
```

```
    def test_abc_concurrently(self, assembly, workstation, is_retry=False):
        """在工作台上同时测试A、B、C三个子系统，支持重试"""
        test_processes = []
        for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C]:
            process = self.env.process(workstation.test_subsystem(assembly, subsystem,
is_retry))
            test_processes.append(process)

        results = yield simpy.AllOf(self.env, test_processes)

        abc_results = {}
        for i, subsystem in enumerate([Subsystem.A, Subsystem.B, Subsystem.C]):
            abc_results[subsystem] = results[test_processes[i]]
            assembly.test_results[subsystem]['status'] = results[test_processes[i]]
            if is_retry:
                assembly.test_results[subsystem]['retry_count'] += 1

        return abc_results

    def test_e_subsystem(self, assembly, workstation, is_retry=False):
        """在工作台上测试E子系统，支持重试"""
        result = yield self.env.process(workstation.test_subsystem(assembly, Subsystem.E,
is_retry))
        assembly.test_results[Subsystem.E]['status'] = result
        if is_retry:
            assembly.test_results[Subsystem.E]['retry_count'] += 1
        return result

    def check_abc_passed(self, abc_results):
        """检查A、B、C三个子系统是否都通过"""
        for result in abc_results.values():
            if result != TestResult.SUCCESS:
                return False
        return True

    def shift_scheduler(self):
        """倒班调度器"""
        while True:
            yield self.env.timeout(self.shift_hours)
            self.current_shift = 1 - self.current_shift
            self.shift_change_event.succeed()
            self.shift_change_event = self.env.event()
            yield self.env.timeout(self.shift_hours)
            self.current_shift = 1 - self.current_shift
            self.shift_change_event.succeed()
            self.shift_change_event = self.env.event()

    def process_assembly(self, assembly, workstation_index):
        """处理单个装置的完整测试流程"""
        with self.workstation_semaphores[workstation_index].request() as req:
```

```
                yield req
                yield self.env.process(self.transport_in())

                workstation = self.workstations[workstation_index]

                # ABC 测试
                abc_passed = False
                abc_retry_count = 0
                while not abc_passed and abc_retry_count <= assembly.max_retries:
                    abc_results = yield self.env.process(
                        self.test_abc_concurrently(assembly, workstation, abc_retry_count >
0))
                    if self.check_abc_passed(abc_results):
                        abc_passed = True
                    else:
                        abc_retry_count += 1
                        if abc_retry_count > assembly.max_retries:
                            assembly.status = 'FAILED'
                            break
                        yield self.env.timeout(0.5)

                if not abc_passed:
                    yield self.env.process(self.transport_out())
                    assembly.end_time = self.env.now
                    self.completed_assemblies.append(assembly)
                    return

                # E 测试
                e_passed = False
                e_retry_count = 0
                while not e_passed and e_retry_count <= assembly.max_retries:
                    e_result = yield self.env.process(self.test_e_subsystem(assembly, work-
station, e_retry_count > 0))
                    if e_result == TestResult.SUCCESS:
                        e_passed = True
                        assembly.status = 'PASSED'
                    else:
                        e_retry_count += 1
                        if e_retry_count > assembly.max_retries:
                            assembly.status = 'FAILED'
                            break
                        yield self.env.timeout(0.5)

                assembly.end_time = self.env.now
                self.completed_assemblies.append(assembly)
                yield self.env.process(self.transport_out())

    def run_testing(self, num_devices_to_test):
        """检测指定数量的装置"""
        self.env.process(self.assembly_generator(num_devices_to_test))
```

```
            self.env.process(self.shift_scheduler())
            for i in range(2):
                self.env.process(self.workstation_process(i, num_devices_to_test))
            while len(self.completed_assemblies) < num_devices_to_test:
                yield self.env.timeout(1)

        def workstation_process(self, workstation_index, num_devices_to_test):
            """带倒班检查的工作台处理进程"""
            while len(self.completed_assemblies) < num_devices_to_test:
                if (workstation_index % 2) != self.current_shift:
                    yield self.shift_change_event
                    continue
                assembly = yield self.assembly_queue.get()
                yield self.env.process(self.process_assembly(assembly, workstation_index))


    def simulate_with_shifts(num_devices_to_test=100, shift_hours=12):
        """带倒班的仿真运行"""
        env = simpy.Environment()
        facility = ShiftTestFacility(env, shift_hours=shift_hours)
        env.process(facility.run_testing(num_devices_to_test))
        env.run(until=1000)

        passed_count = sum(1 for a in facility.completed_assemblies if a.status == 'PASSED')
        completed_assemblies = [a for a in facility.completed_assemblies if a.end_time is not
    None]
        total_time = max(a.end_time for a in completed_assemblies) if completed_assemblies else
    0

        total_tests = 0
        false_alarms = 0
        missed_detections = 0
        equipment_failures = 0

        for workstation in facility.workstations:
            total_tests += workstation.stats['total_tests']
            false_alarms += workstation.stats['false_alarms']
            missed_detections += workstation.stats['missed_detections']
            equipment_failures += workstation.stats['equipment_failures']

        false_alarm_prob = false_alarms / total_tests if total_tests > 0 else 0
        missed_detection_prob = missed_detections / total_tests if total_tests > 0 else 0

        yxb_ratios = {}
        total_sim_time = max(a.end_time for a in completed_assemblies) if completed_assem-
    blies else 0

        for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C, Subsystem.E]:
            total_pure_test_time = 0
            for workstation in facility.workstations:
```

```
                total_test_time = workstation.subsystem_actual_times.get(subsystem, 0) if ha-
sattr(workstation,

'subsystem_actual_times') else 0
                total_pure_test_time += total_test_time * 0.7
            yxb_ratios[subsystem.name] = total_pure_test_time / (
                                total_sim_time / (24 / 12)) if total_sim_time > 0 else 0

        total_retries = 0
        for assembly in facility.completed_assemblies:
            for subsystem in [Subsystem.A, Subsystem.B, Subsystem.C, Subsystem.E]:
                total_retries += assembly.test_results[subsystem]['retry_count']

        avg_retries_per_assembly = total_retries / len(
            facility.completed_assemblies) if facility.completed_assemblies else 0

        return {
            'total_days': total_time / 24,
            'passed_count': passed_count,
            'false_alarm_prob': false_alarm_prob,
            'missed_detection_prob': missed_detection_prob,
            'yxb_ratios': yxb_ratios,
            'total_retries': total_retries,
            'avg_retries_per_assembly': avg_retries_per_assembly,
            'equipment_failures': equipment_failures
        }


# ==================== 可视化函数 ====================

class GAVisualizer:
    """遗传算法可视化类"""

    def __init__(self):
        self.fitness_history = []
        self.k_values_history = []
        self.generation_stats = []

    def record_generation(self, population, generation):
        """记录每一代的信息"""
        # 只记录已经评估过的个体
        evaluated_individuals = [ind for ind in population if hasattr(ind.fitness, 'values') and
ind.fitness.values]

        if not evaluated_individuals:
            return

        fitnesses = [ind.fitness.values[0] for ind in evaluated_individuals]
        k_values = [ind[0] for ind in evaluated_individuals]
```

```
            self.fitness_history.append(fitnesses)
            self.k_values_history.append(k_values)
            self.generation_stats.append({
                'generation': generation,
                'min_fitness': min(fitnesses) if fitnesses else float('inf'),
                'max_fitness': max(fitnesses) if fitnesses else float('-inf'),
                'avg_fitness': np.mean(fitnesses) if fitnesses else 0,
                'best_k': k_values[np.argmin(fitnesses)] if fitnesses else 0
            })

        def plot_evolution(self):
            """绘制进化过程"""
            if not self.generation_stats:
                print("没有足够的数据进行可视化")
                return

            fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))

            # 1. 适应度进化图
            generations = range(len(self.generation_stats))
            min_fitness = [stat['min_fitness'] for stat in self.generation_stats]
            avg_fitness = [stat['avg_fitness'] for stat in self.generation_stats]

            ax1.plot(generations, min_fitness, 'b-', label='最佳适应度')
            ax1.plot(generations, avg_fitness, 'r-', label='平均适应度')
            ax1.set_xlabel('代数')
            ax1.set_ylabel('适应度值')
            ax1.set_title('遗传算法进化过程')
            ax1.legend()
            ax1.grid(True)

            # 2. K 值分布图
            for gen in range(0, len(self.k_values_history), max(1, len(self.k_values_history) //
5)):
                if gen < len(self.k_values_history) and self.k_values_history[gen]:
                    ax2.hist(self.k_values_history[gen], alpha=0.5,
                             label=f'第{gen}代', bins=10, range=(9, 12))
            ax2.set_xlabel('班次时长  K (小时)')
            ax2.set_ylabel('频次')
            ax2.set_title('K 值分布变化')
            ax2.legend()
            ax2.grid(True)

            # 3. 最佳 K 值变化
            best_k_values = [stat['best_k'] for stat in self.generation_stats]
            ax3.plot(generations, best_k_values, 'g-o', linewidth=2)
            ax3.set_xlabel('代数')
            ax3.set_ylabel('最佳 K 值  (小时)')
```

```
        ax3.set_title('最佳班次时长进化过程')
        ax3.grid(True)

        # 4. 适应度箱线图
        if len(self.fitness_history) > 1:
            ax4.boxplot(self.fitness_history, positions=range(len(self.fitness_history)))
        ax4.set_xlabel('代数')
        ax4.set_ylabel('适应度值')
        ax4.set_title('适应度分布箱线图')
        ax4.grid(True)

        plt.tight_layout()
        plt.savefig('ga_evolution.png', dpi=300, bbox_inches='tight')
        plt.show()


# ==================== 遗传算法优化部分 ====================

# 首先定义目标和工具箱
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# 初始化工具箱
toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, 9.0, 12.0)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# 初始化可视化器
visualizer = GAVisualizer()


def evaluate_individual_fixed(individual):
    """修复后的评估函数"""
    K = individual[0]
    K_rounded = round(K * 2) / 2.0
    K_rounded = max(9.0, min(12.0, K_rounded))

    # 添加最小 K 值约束
    if K_rounded < 9.5:
        return (12.0,)  # 惩罚过小的 K 值

    try:
        sim_results = simulate_with_shifts(num_devices_to_test=30,
shift_hours=K_rounded)
        T = sim_results['total_days']
        PL = sim_results['missed_detection_prob']

        # 更平衡的权重分配
```

```
            weight_T = 0.5
            weight_PL = 0.5

            # 添加质量约束惩罚
            if PL > 0.08:  # 漏判概率上限
                penalty = 50 * (PL - 0.08)
            else:
                penalty = 0

            weighted_score = weight_T * T + weight_PL * PL + penalty
            return (weighted_score,)

        except Exception as e:
            print(f"评估失败: {e}")
            return (1000.0,)


def init_diverse_population(size):
    """生成更多样化的初始种群"""
    # 在合理范围内均匀分布
    base_values = [9.5, 10.0, 10.5, 11.0, 11.5, 12.0]
    population = []

    for i in range(size):
        if i < len(base_values):
            # 使用基础值
            ind = creator.Individual([base_values[i % len(base_values)]])
        else:
            # 随机生成，但偏向中间值
            center = random.choice([10.0, 10.5, 11.0])
            ind = creator.Individual([center + random.uniform(-0.5, 0.5)])

        population.append(ind)

    return population


def run_genetic_algorithm_fixed(pop_size=10, ngen=6):
    """修复后的遗传算法"""
    print("开始改进的遗传算法优化...")

    # 使用多样化的初始种群
    pop = init_diverse_population(pop_size)

    # 注册修复后的评估函数
    toolbox.register("evaluate", evaluate_individual_fixed)
    toolbox.register("mate", tools.cxBlend, alpha=0.5)
    toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.5, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
```

```
    # 评估初始种群
    print("评估初始种群...")
    fitnesses = list(toolbox.map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit

    visualizer.record_generation(pop, 0)
    hof = tools.HallOfFame(1)
    hof.update(pop)

    # 运行遗传算法
    for gen in range(1, ngen + 1):
        # 选择下一代
        offspring = toolbox.select(pop, len(pop))
        offspring = list(offspring)

        # 应用交叉和变异
        offspring = algorithms.varAnd(offspring, toolbox, cxpb=0.6, mutpb=0.3)

        # 评估新个体
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = toolbox.map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # 更新种群
        pop[:] = offspring
        hof.update(pop)
        visualizer.record_generation(pop, gen)

        print(f"第{gen}代完成，最佳 K 值: {hof[0][0]:.1f}小时，评分: {hof[0].fitness.values[0]:.4f}")

    # best_K = round(hof[0][0] * 2) / 2.0
    best_K = hof[0][0]
    best_K = max(9.5, min(12.0, best_K))  # 确保不小于 9.5

    print(f"遗传算法完成，最优 K 值: {best_K} 小时")
    return best_K


def monte_carlo_with_shifts(num_simulations=30, num_devices_per_sim=100, shift_hours=12):
    """带倒班的蒙特卡洛模拟"""
    results = {
        'total_days': [], 'passed_count': [], 'false_alarm_prob': [],
        'missed_detection_prob': [], 'equipment_failures': [],
        'total_retries': [], 'avg_retries_per_assembly': []
```

```
    }

    yxb_ratios = {'A': [], 'B': [], 'C': [], 'E': []}

    print("进行蒙特卡洛模拟...")
    for _ in tqdm(range(num_simulations)):
        sim_results = simulate_with_shifts(num_devices_per_sim, shift_hours)

        for key in results:
            if key in sim_results:
                results[key].append(sim_results[key])

        for subsystem, ratio in sim_results['yxb_ratios'].items():
            yxb_ratios[subsystem].append(ratio)

    def calculate_stats(data):
        if isinstance(data, dict):
            return {k: calculate_stats(v) for k, v in data.items()}
        elif isinstance(data, list):
            return {
                'mean': np.mean(data),
                'std': np.std(data),
                'min': np.min(data),
                'max': np.max(data),
                'median': np.median(data)
            }
        else:
            return data

    return {
        'basic_metrics': calculate_stats({k: v for k, v in results.items()}),
        'yxb_ratios': calculate_stats(yxb_ratios)
    }


def solve_problem3_with_ga():
    """使用遗传算法解决问题 3"""
    # 寻找最优 K 值
    optimal_k = run_genetic_algorithm_fixed(pop_size=50, ngen=500)

    # 可视化进化过程
    visualizer.plot_evolution()

    # 使用最优 K 值进行详细模拟
    print(f"\n 使用最优班次时长 K={optimal_k} 进行详细模拟...")
    mc_results = monte_carlo_with_shifts(num_simulations=30, num_devices_per_sim=100,
shift_hours=optimal_k)

    # 输出结果表格
```

```
    print("\n 问题 3 结果统计指标（遗传算法优化）:")
    print(f"最优班次时长 K: {optimal_k} 小时")

    table_data = [
        ["T (平均天数)", f"{mc_results['basic_metrics']['total_days']['mean']:.2f}"],
        ["S (通过数量)", f"{mc_results['basic_metrics']['passed_count']['mean']:.1f}"],
        ["PL ( 漏 判 概 率 )", f"{mc_results['basic_metrics']['missed_detection_prob']['mean']:.4f}"],
        ["Pw (误判概率)", f"{mc_results['basic_metrics']['false_alarm_prob']['mean']:.4f}"],
        ["YXB1 (A 组)", f"{mc_results['yxb_ratios']['A']['mean']:.4f}"],
        ["YXB2 (B 组)", f"{mc_results['yxb_ratios']['B']['mean']:.4f}"],
        ["YXB3 (C 组)", f"{mc_results['yxb_ratios']['C']['mean']:.4f}"],
        ["YXB4 (E 组)", f"{mc_results['yxb_ratios']['E']['mean']:.4f}"]
    ]

    print("\n" + tabulate(table_data, headers=["指标", "值"], tablefmt="grid"))

    # 保存结果
    with open('problem3_results_ga.txt', 'w', encoding='utf-8') as f:
        print("问题 3 结果统计指标（遗传算法优化）:", file=f)
        print(f"最优班次时长 K: {optimal_k} 小时", file=f)
        print("\n" + tabulate(table_data, headers=["指标", "值"], tablefmt="grid"), file=f)

    return optimal_k, mc_results


# ==================== 主程序 ====================

if __name__ == "__main__":
    # random.seed(42)
    # np.random.seed(42)

    print("=" * 60)
    print("开始解决问题 3 - 使用遗传算法优化班次时长")
    print("=" * 60)

    try:
        optimal_k, results = solve_problem3_with_ga()

        print("\n" + "=" * 60)
        print("优化完成！")
        print(f"• 最优班次时长: {optimal_k} 小时")
        print(f"• 预计完成天数: {results['basic_metrics']['total_days']['mean']:.2f} 天")
        print(f"• 平均通过数量: {results['basic_metrics']['passed_count']['mean']:.1f}")
        print("=" * 60)

    except Exception as e:
```

```
print(f'程序运行出错: {e}")
import traceback

traceback.print_exc()
```