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 题

我们的参赛编号（请填写完整参赛编号）：202518001014

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. 王博轩 王博轩

2. 石锦钰 石锦钰

3. 卢雄奕 卢雄奕

指导教师或指导教师组负责人（打印后签名）：李际超 李际超

日期：2025 年 8 月 26 日

第十届湖南省研究生数学建模竞赛

题　目：考虑随机事件的大型装备测试任务规划

摘要：

随着制造业智能化水平的提升，大型装备测试任务规划问题成为现代工业系统中的重要研究课题.本文针对大型装备测试过程中存在的资源约束、随机事件干扰和多目标优化等复杂问题，基于禁忌搜索算法和蒙特卡洛仿真思想，通过确定任务完成平均天数(T)、总漏判概率(PL)、有效工作时间比(YXB)等核心指标，以“任务尽快完成、漏判概率最低”为目标，建立了多阶段随机规划模型，并使用改进的禁忌搜索算法对模型进行高效求解.

针对问题一：为了解决综合测试中问题指向各子系统的比例计算问题，本文基于概率论和条件概率思想，建立了漏判概率传递模型，推导出比例参数 λ 的数学表达式.通过引入测手差错概率(P(Y31)、P(Y32))和子系统问题概率(P(A)、P(B)、P(C)、P(D))，将复杂测试流程转化为概率计算问题，最终求得 λ_1=0.3063、λ_2=0.4913、λ_3=0.1630、λ_4=0.0401，综合测试问题检出概率 P(E)=0.0256.

针对问题二：为了解决单分队测试 100 个装置的规划问题，本文构建了混合整数规划模型，通过大 M 法处理逻辑约束，并设计禁忌搜索算法优化测试顺序.仿真结果显示，最优方案下 T=39.07 天，PL=0.0022，YXB 最高达 0.6015(测试小组 E).

针对问题三：在双分队倒班模式下，本文创新性地引入班次时长变量 K(9≤K≤12 小时)，建立设备跨班次累计使用模型.通过参数扫描发现 K=12 小时为最优解，此时 T=17.73 天(较 K=9 小时缩短 34.7%)，PL=0.0001(降低 85.9%)，且设备利用率(YXB4=0.6710)达到峰值.

针对问题四：本文基于系统优化理论与敏感性分析思想，建立了多因素协同影响分析模型.通过引入全因子实验设计与边际效应分析方法，量化了班次时长(K)、设备更换周期、漏判概率(PL)及误判概率(PW)对平均完成时间(T)的独立与交互影响，实现了对测试工作效率与质量关键驱动因素的精准识别与平衡优化.求解结果表明，在 K=12 小时、设备更换周期 240 小时的方案下，测试平均时间最短 (17.73 天)，且漏判概率最低 (0.0001)；相较于基准方案 (K=9 小时)，任务周期缩短 34.7%，设备有效工作时间比 (YXB) 最高提升 62.5% (如 E 组 YXB4=0.6710)，证实了延长班次与优化设备更换策略对提升测试效率的显著效果.

最后，我们对提出的模型进行全面的评价：本文模型紧密结合工业实际，通过动态禁忌表管理和自适应邻域搜索，解决了传统任务规划模型在随机事件处理上的不足.该模型可推广至半导体测试、航空航天设备检测等领域，尤其适用于多资源约束的串-并行混合生产系统.

关键词：大型装备测试　任务规划　随机事件　禁忌搜索算法　蒙特卡洛模拟

目录

1 问题综述

1.1 问题背景

随着制造领域对大型关键装置可靠性要求的持续提高, 大型装置测试任务的规模与复杂度不断攀升, 测试需求呈显著增长态势[1].然而, 实际测试工作始终面临 “需求扩张” 与 “资源约束” 的核心矛盾: 一方面, 单个大型装置需经过多环节串行测试流程, 且各环节可能因各类异常情况(如设备运行问题、操作偏差等)需重新测试, 进一步增加了任务总量与执行难度; 另一方面, 测试资源存在刚性限制, 测试场地可同时容纳的测试装置数量有限, 专业测试小组的工位资源固定, 且核心测试设备不仅有故障概率随使用时长动态变化的特性, 还需遵循特定的更换规则(如达到一定使用时长或出现故障时必须更换、未故障时提前更换需满足最低工作时长要求).传统测试计划制定方式多依赖经验驱动, 未能充分整合设备运行特性、重测流程规范、资源协同调度等关键因素, 易导致测试任务整体周期延长、各测试环节负载分配不均、测试质量风险难以有效管控等问题, 制约了测试工作的效率与可靠性, 无法充分满足大型装置测试的实际需求.

目前, 学术界针对任务规划类问题已开展大量研究, 提出了使用遗传算法[2]、粒子群算法[3-4]、蚁群算法[5]、模拟退火算法[6]等多种优化算法对此类问题进行求解.其中遗传算法凭借其较强的全局搜索能力, 在多资源协同调度场景中得到广泛应用, 能够为测试任务规划提供基础的调度方案框架.但这类优化算法普遍存在局限性, 其设计多基于通用性调度模型, 对大型装置测试场景的适配性不足: 一方面, 在平衡 “任务高效完成” 与 “测试质量风险控制” 双目标时, 遗传算法难以精准响应测试过程中的动态约束条件, 可能导致方案在实际执行中出现资源浪费或风险失控的情况; 另一方面, 对于测试流程中各类特殊规则与异常情况的整合能力较弱, 生成的调度方案易出现与实际测试场景脱节的问题, 难以直接支撑复杂测试任务的高效推进.为此, 本文引入禁忌搜索算法[7], 利用其独特的 “禁忌表” 机制避免陷入局部最优解, 能够更灵活地整合测试过程中的动态约束与多目标需求, 相比遗传算法, 在处理复杂场景下的多目标优化问题时具备更强的适应性与精准性, 可有效弥补传统算法通用性框架的不足, 为大型装置测试任务规划提供更贴合实际需求的优化方案.

1.2 问题提出

大型装置测试任务规划问题涉及多方面的问题, 往往由测试流程规范、测试资源约束、异常情况处理、多目标优化这几个要素构成, 在满足以下 4 点具体约束下, 需制定科学合理的测试工作计划以实现任务高效完成与测试质量保障的目标:

- **测试资源约束**: 测试场地可同时测试的装置数量、专业测试小组的工作位置均有限, 测试设备启用前需调试, 故障概率随使用时长变化且更换有明确条件.
- **测试流程约束**: 装置需先完成各子系统单独测试且全部通过后, 才能进行综合测试, 测试中断需重新开始, 装置的运出与运入各需一定时间且不影响其他测试.
- **异常情况约束**: 测试过程中可能出现设备故障、子系统问题、测手操作差错、综合测试问题, 各类异常情况有对应的发生概率及重测等处理规则.
- **任务目标约束**: 测试任务以尽快完成、降低总漏判概率为主要目标, 同时需统计任务完成时间、通过测试的装置数量等相关指标.

需要从测试流程优化、资源调度效率、异常风险管控、多目标平衡角度考虑, 解决以下 4 个问题:

(1) 问题 1: 根据大型装置综合测试中各子系统(含联接系统)的问题发生概率(如子系统 A、B、C 及联接系统 D 的问题概率), 结合综合测试对各子系统问题的识别特性, 推导综合测试中问题指向各子系统的比例参数的数学表达式, 同时构建综合测试测出系统存在问题的概率计算表达式, 代入题目给定的具体概率数据, 求解得到比例参数值与综合测试问题检出概率值.

(2) 问题 2: 根据 1 个测试分队测试 100 个大型装置的任务需求, 结合每日 1 个班次(工作时间不超过 12 小时)、测试工序中断需重新开始等规则, 制定以 “任务尽快完成、总漏判概率尽量低” 为目标的测试工作计划; 基于该计划, 计算任务完成平均天数、通过测试的装置平均数目、总漏判概率、总误判概率, 以及 4 个专业测试组的有效工作时间比(小组每班次平均测试时间 / 12), 并将结果整理至指定统计表格.

(3) 问题 3: 针对第二批 100 个大型装置的紧急测试任务, 按 “两个分队接续倒班” 模式(同工序两个班次共用一套测试装备), 在班次工作时间 K(9≤K≤12, 最小单位为半小时)的可选范围内确定最优 K 值; 依据最优 K 值制定测试工作计划, 计算该计划下的任务完成平均天数、通过测试的装置平均数目、总漏判概率、总误判概率, 以及 4 个专业测试组的有效工作时间比(小组每班次平均测试时间 / K), 并按指定表格格式整理结果.

(4) 问题四: 根据问题 3 中制定的测试计划与计算结果, 分析班次工作时间、设备故障概率、重测规则、分队协作模式等因素对测试任务平均完成时间的影响程度与作用机制; 基于分析结论, 向主管部门提出可提升测试效率、优化任务周期的具体改进建议.

2 问题分析

2.1 问题一的分析

测试任务中, 装备的子系统 A、B、C 分别经过测试, 在这三个子系统均判为正常后, 再由测试小组 E 对子系统 D 进行综合测试.综合测试测出有问题分为两种情况: 一是前面子系统 A、B、C 存在问题, 但由于测手漏判, 进入综合测试; 二是联结系统存在问题, 即子系统 D 存在问题, 子系统 D 存在问题的概率为题中所给定值.因此, 首先计算子系统 A、B、C 存在问题但进入综合测试的概率, 再计算综合测试测出问题指向子系统 A、B、C、D 的比例, 最后, 对各子系统存在问题的概率和所求比例进行相乘相加, 即可得到综合测试测出问题的概率.

2.2 问题二的分析

问题 2 是一个考虑随机事件的任务规划问题, 优化的目标是任务尽快完成和总的漏判概率最低.制定计划的关键是确定装备开始测试和结束测试的时间, 以及每个装备的子系统 A、B、C 的测试顺序.对于可能出现的故障需要根据其概率, 通过大量实验仿真模拟.在测试流程方面, 第一天首次使用设备需要调试设备, 后续只有在更换新设备后才需调试, 调试之后进行测试任务; 需要先完成装置的子系统 A、B、C 的测试,A、B、C 的工序可以任意安排, 子系统测试均通过, 才能进行子系统 D 的测试, 测试通过或不通过都要移出测试大厅, 并移入新设备进行测试.在资源限制方面, 一个测试小组一次只能测试一个子系统, 测试大厅中最多只能容纳并测试 2 个装备; 每日一个班次, 每个班次的时间为 12 小时; 若有测试任务开始, 但无法在班次时间之内完成, 则需等到下个班次开始重新测试.在风险管控方面, 若测试设备故障, 则需更换测试设备; 子系统测试出有问题, 则需要重测.

2.3 问题三的分析

问题 3 是在问题 2 基础上, 针对第二批 100 个装置的紧急测试任务进行优化, 核心在于确定最优班次工作时间 K(9≤K≤12, 最小单位为半小时)并制定对应计划.与问题 2 相比, 关键差异在于测试模式调整为 “两个分队接续倒班” , 且同工序两个班次共用一套测试装备, 这意味着需考虑装备在两个班次间的衔接(如装备使用时间累计、故障概率随使用时间变化的特性——120 小时内与 120-240 小时区间故障概率不同, 装备需满足至少工作 120 小时才可提前更换、240 小时或故障时必须更换等规则), 避免因装备更换或故障导致倒班衔接中断.制定计划时, 需在不同 K 值下模拟测试流程, 分析 K 值对任务周期、装备利用率、漏判概率的影响, 筛选出既能加快进度又能控制质量风险的最优 K 值; 随后基于最优 K 值, 计算与问题 2 相同类型的指标(任务完成平均天数、通过测试装置平均数目等), 但有效工作时间比的计算需调整为 “小组每班次平均测试时间/ K” .

2.4 问题四的分析

问题 4 的核心是分析问题 3 中各因素对测试任务平均完成时间的影响, 并提出改进建议, 需以问题 3 的测试计划与计算结果为依据, 明确各因素的作用机制与影响程度.需重点分析的因素包括: 班次工作时间 K(不同 K 值对单日测试量、装备使用时长累积速度的影响)、设备故障概率(120 小时内与 120-240 小时区间故障概率差异对测试中断频率的影响)、重测规则(子系统问题、测手误判等导致的重测对单装置测试时长的延长作用)、分队协作模式(两个分队倒班衔接的顺畅度, 如装备交接、测试进度衔接对整体周期的影响)等.通过分析各因素如何直接或间接影响测试效率(如设备故障频繁会增加中断次数, 延长总周期; K 值过小会减少单日有效测试时间, K 值过大可能导致装备过度使用引发故障风险), 总结关键影响因素的优先级; 基于此, 从装备管理(如优化装备更换时机)、流程优化(如减少重测频次)、人员协作(如提升测手操作准确性)等角度, 向主管部门提出可落地的改进建议, 以缩短测试任务平均完成时间.

3 模型假设与符号说明

3.1 模型基本假设

(1) 同一时间, 每个测试小组只能测试一个子系统.
(2) 大型装备在测试过程中不更换测试台, 测试小组可以在不同测试台工作.
(3) 同一装备的子系统 A、B、C 按序依次测试, 子系统 A、B、C 的测试次序随机.
(4) 有足够的备用测试设备可更换, 且更换测试设备不计时间.

3.2 符号说明

本文对问题背景中的相关变量进行定义, 符号说明如表 1, 其余符号在使用时注明.

表1 符号说明

符号	含义	单位
$I=\{I_1,I_2,...\}$	待测试装置集合	
N	待测试装备数量	个
J	测试小组集合, $J=\{j\}=\{A,B,C,E\}$	

符号	含义	单位
S	装备子系统集合，$S=\{s\}=\{A,B,C,D\}$	
Π	子系统 $\{A,B,C\}$ 测试顺序的所有排列组合的集合，$\Pi=\{\pi\}=\{(A,B,C),(A,C,B)...\}$	
D	测试天数集合，$D=\{d\}=\{1,2,...\}$	
K	每日班次的工作时长	小时
$T_{transport}$	将一个装置运入或运出测试大厅所需的时间	小时
T_{setup}^{j}	测试小组 j 设备的调试时间	小时
T_{test}^{j}	测试小组 j 的正常测试时间	小时
T_{change}	设备强制更换的累计工作时间阈值	小时
$P_{fault}^{j}(t)$	测试小组 j 的设备在累计工作时间为 t 时的故障概率	
$P(s)$	子系统 s 本身有问题的概率	
$P(jY3)$	测试小组 j 的测手发生差错的概率	
$P(Y31)$	测手误判的概率	
$P(Y32)$	测手漏判的概率	
$P(s\mid E)$	综合测试测出问题时指向各子系统的概率	
λ_i	综合测试测出问题时指向各子系统的比例参数	
T	任务完成平均天数	天
S	通过测试的装置的平均数目	个
P_L	总漏判概率	
P_W	总误判概率	
YXB	测试小组的有效工作时间比	

4 模型建立与求解

4.1 问题一的求解

在进入综合测试之前，子系统 A、B、C 在综合测试前测试流程及概率图如图 1. 当子系统 A 存在问题时，若第一次测试通过，则测手漏判，进入后续测试； 若第一次失败，则需进行重测，若第二次测试通过，则测手再次漏判，进入后续测试，若第二次测试失败，则装备退出测试. 当子系统 A 不存在问题时，若第一次测试通过，则进入后续测试； 若第一次失败，则测手误判，需进行重测，若第二次测试通过，进入后续测试，若第二次测试失败，则测手再次误判，装备退出测试. 子系统 B、C 同理.

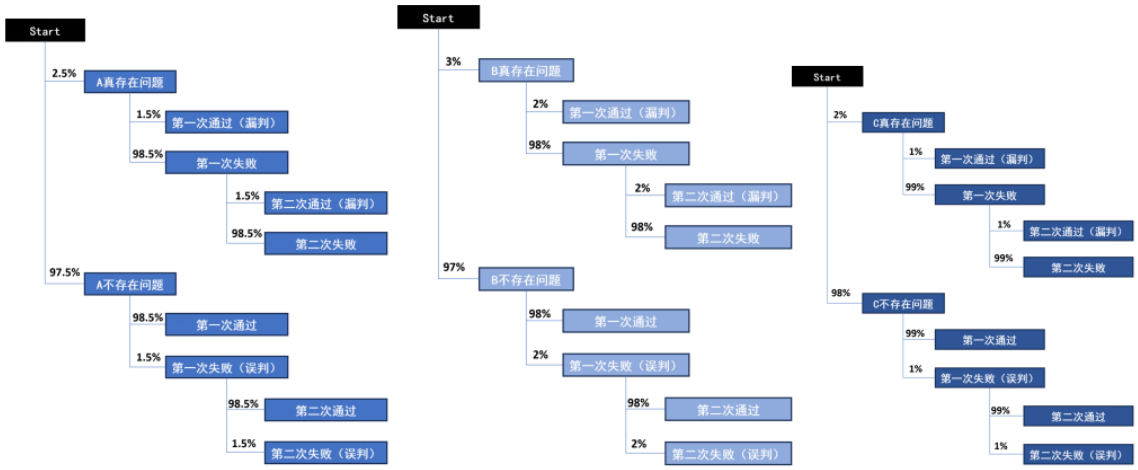

图1 子系统 A、B、C 的测试流程

由上图可知，问题子系统 A、B、C 进入到综合测试，有两种情况：一是问题子系统 A、B、C 被漏判，直接进入综合测试；　二是问题子系统 A、B、C 被测出问题，进行重测，重测时漏判，经过两次测试进入综合测试. 因此，综合测试测出问题指向子系统 A、B、C 的概率为子系统 A、B、C 存在问题经过一次测试和两次测试进入综合测试概率的和.

设子系统 A、B、C 存在问题的概率分别为 P(A)、P(B)、P(C)，子系统 D 存在问题的概率为 P(D)，A、B、C、E 四个测试小组的测手发生差错的概率为 P(AY3)、P(BY3)、P(CY3)、P(EY3)，测手误判的概率为 P(Y31)，测手漏判的概率为 P(Y32). 设综合测试测出问题的概率为 P(E)，综合测试测出问题指向子系统 A、B、C、D 的概率分别为 $P(A\mid E)$、$P(B\mid E)$、$P(C\mid E)$、$P(D\mid E)$. 则有：

$$P(A\mid E)=P(A)\times(P(AY3)\times P(Y32)+(1-P(AY3)\times P(Y32))\times P(AY3)\times P(Y32)) \tag{1}$$

$$P(B\mid E)=P(B)\times(P(BY3)\times P(Y32)+(1-P(BY3)\times P(Y32))\times P(BY3)\times P(Y32)) \tag{2}$$

$$P(C\mid E)=P(C)\times(P(CY3)\times P(Y32)+(1-P(CY3)\times P(Y32))\times P(CY3)\times P(Y32)) \tag{3}$$

$$P(D\mid E)=P(D) \tag{4}$$

则综合测试测出有问题时各部分所占比例 λ_i (i=1,2,3,4)的表达式为:

$$\lambda_i=\frac{P(s\mid E)}{\sum_{s}P(s\mid E)}\quad(s=A,B,C,D) \tag{5}$$

综合测试测出问题的概率表达式为:

$$P(E)=\lambda_1\times P(A)+\lambda_2\times P(B)+\lambda_3\times P(C)+\lambda_4\times P(D) \tag{6}$$

带入数值计算得出:

$\lambda_1=0.3063$，$\lambda_2=0.4913$，$\lambda_3=0.1630$，$\lambda_4=0.0401$

$P(E)=0.0256$

因此，综合测试测出问题指向子系统 A、B、C、D 的比例参数分别为: 0.3063、0.4913、0.1630、0.0401，综合测试测出问题的概率为 0.0256.

4.2 问题二的求解

4.2.1 考虑随机事件的大型装置测试任务规划模型

4.2.1.1 变量设置

设控制变量: 设 $x_{i,j,d} \in \{0,1\}$ ， 取值为 1 时表示测试小组 j 对装置 i 的子系统在第 d 天的某个时间点开始测试； 设 $y_{j,d} \in \{0,1\}$， 取值为 1 时表示在第 d 天工作开始前更换小组 j 的设备； 设 $z_{i,\pi} \in \{0,1\}$，取值为 1 时表示装置 i 的子系统测试顺序采用排列 π_i.

设决策变量： 设 $s_{i,j}$ 表示装置 i 中子系统对应测试小组 j 的开始测试时间， 该时间为从项目开始起的绝对时间； 设 $c_{i,j}$ 表示装置 i 中子系统对应测试小组 j 的完成测试时间， 该时间为考虑了重测、中断等所有情况后的实际完成时间， 通过仿真实验获取; 设 $u_j(t)$ 表示测试小组 j 的测试设备在项目时间为 t 时的累计工作时间.

设状态变量： 设 $status_{i,j}$ 表示装置 i 中测试小组 j 处理的子系统的最终状态， $status_{i,j} = \{0,1,2,3,4,5\}$ ， 取值为 0 时表示子系统未开始测试， 取值为 1 时表示子系统正在测试中， 取值为 2 时表示子系统通过测试， 取值为 3 时表示子系统需要重测， 取值为 4 时表示子系统测试因故障中断，取值为 5 时表示装备退出； 设 $N_{retest}^{i,j}$ 表示装置 i 测试中测试小组 j 的重测次数； $DayEnd_d$ 表示第 d 天工作结束的时间点， 每日工时 K =12h 时， $DayEnd_d = d \times 12$.

4.2.1.2 约束条件

考虑约束条件:

(1) 测试流程约束

装备的子系统 A、B、C 的测试顺序随机, 综合测试 E 在子系统 A、B、C 都测试通过后才能进行.则有:

$$\sum_{\pi \in \Pi} z_{i,\pi} = 1 \quad \forall i = 1,\ldots,N \tag{7}$$

其中， $\Pi = \{(A,B,C),(A,C,B),(B,A,C),(B,C,A),(C,A,B),(C,B,A)\}$

子系统测试顺序:

$$s_{i,B} \geq c_{i,A}, \quad s_{i,C} \geq c_{i,B}, \quad s_{i,E} \geq c_{i,C} \quad \forall i \tag{8}$$

综合测试前提为:

$$w_{i,E} = 1 \Rightarrow w_{i,A} = w_{i,B} = w_{i,C} = 1 \quad \forall i \tag{9}$$

为每个布尔变量 $w_{i,j}$ 引入一个二进制变量 $f_{i,j} \in \{0,1\}$， 其中： $f_{i,j} = 1$ 表示 $w_{i,j} = 1$ (测试通过)， $f_{i,j} = 0$ 表示 $w_{i,j} = 0$ (测试不通过), 使用大 M 法, 可得到:

$$\begin{aligned} f_{i,A} &\geq 1 - M(1 - f_{i,E}) \\ f_{i,B} &\geq 1 - M(1 - f_{i,E}) \quad \forall i \\ f_{i,C} &\geq 1 - M(1 - f_{i,E}) \end{aligned} \tag{10}$$

在此题中 M 可取 1， 故综合测试的前提可表示为:

$$\begin{aligned} f_{i,A} &\geq f_{i,E} \\ f_{i,B} &\geq f_{i,E} \quad \forall i \\ f_{i,C} &\geq f_{i,E} \end{aligned} \tag{11}$$

(2) 资源约束

测试大厅最多容纳 2 个装置; 每个小组同一时间只能测试一个装置, 则有:

$$\sum_{i\in I}\sum_{j\in J} \mathrm{I}(s_{i,j} \leq t < c_{i,j}) \leq 2 \quad \forall t \tag{12}$$

$$\sum_{i\in I} \mathrm{I}(s_{i,j} \leq t < c_{i,j}) \leq 1 \quad \forall t, \forall j \in J \tag{13}$$

其中, $\mathrm{I}(\cdot)$是指示函数, 条件为真时值为 1, 否则为 0.

(3) 时间与班次约束

任何测试任务和运输任务不能跨天中断续做, 必须在一个班次内完成, 如果不能完成, 需要等到下一个班次再开始.则有:

$$|s_{i,j}/K| = |c_{i,j}/K| \quad \forall i \in I, \forall j \in J \tag{14}$$

$$s_{i,A} \geq c_{i-1} + T_{transport} \quad \forall i > 1 \tag{15}$$

其中, $|\cdot|$为向上取整函数.

(4) 设备更换约束

假设问题二中不提前更换设备, 设备累计工作满 240 小时更换, 如果在第 d 天开始时更换设备, 则该设备累计工作时间重置为 0, 则有:

$$\begin{aligned} y_{j,d} = 1 \quad \text{if} \quad u_j(DayEnd_{d-1}) = T_{change} \\ u_j(DayStart_d) = 0 \quad \text{if} \quad y_{j,d} = 1 \end{aligned} \tag{16}$$

4.2.1.3 设备故障概率处理

对于设备故障这一随机事件, 可以通过生成随机数来判断设备是否故障, 测试小组 j 的设备在累计工作时间为t 时的故障为$P^j_{fault}(t)$, 设备在工作 120h 内每个时刻故障的概率相同, 累计概率为$P^j_{fault}(120)$, 设备在工作 120h 至 240h 内每个时刻故障的概率相同, 累计概率为$P^j_{fault}(240)$.

用随机数生成数生成 0 到 1 内的随机数P_{random}, 若$P_{random} \leq P^j_{fault}(120)$, 则测试小组 j 的设备在使用时间累计 120h 时间内发生故障的时间为:

$$t^j_{fault} = 120 \times \frac{P_{random}}{P^j_{fault}(120)} \tag{17}$$

若$P^j_{fault}(120) < P_{random} \leq P^j_{fault}(240)$, 则测试小组 j 的设备在使用时间累计 120h 至 240h 时间内发生故障的时间为:

$$t^j_{fault} = 120 + 120 \times \frac{P_{random}}{P^j_{fault}(240)} \tag{18}$$

若 $P_{random} > P_{fault}^{j}(240)$，不发生故障，更换设备.

4.2.1.3 结果输出

最终结果中的各个参数的计算公式如下:

(1) 任务完成平均天数 T

$$T = \max_{i \in I} | c_{i,E} / K | \tag{19}$$

其中，$|\cdot|$ 其为向上取整函数.

(2) 通过测试的装置的平均数目 S

通过测试的装置，四个子系统的最终状态均为 “通过测试”，则有:

$$S = \sum_{i=1}^{100} \mathrm{I}(status_{i,A} = 2 \wedge status_{i,B} = 2 \wedge status_{i,C} = 2 \wedge status_{i,E} = 2) \tag{20}$$

其中，$\mathrm{I}(\cdot)$ 是指示函数，条件为真时值为 1，否则为 0.

(3) 总漏判概率 P_L

测试小组 j 的漏判次数需要通过仿真实验得到，记为 $P_L{}^{j}$，假设仿真模拟次数为 H，则有:

$$P_L = \frac{1}{H} \sum_{i=1}^{H} \sum_{j \in \{A,B,C,E\}} P_L{}^{j} \tag{21}$$

(4) 总误判概率 P_W

测试小组 j 的误判次数需要通过仿真实验得到，记为 $P_W{}^{j}$，假设仿真模拟次数为 H，则有:

$$P_W = \frac{1}{H} \sum_{i=1}^{H} \sum_{j \in \{A,B,C,E\}} P_W{}^{j} \tag{22}$$

(5) 有效工作时间比 YXB

测试小组 j 的正常测试时间为 T_{test}^{j}，由于存在重测，一个子系统测试所需的总有效测试时间是基本时长乘以重测次数.通过仿真实验，可以得到装备 i 被测试小组 j 重测的次数，记为 $N_{retest}{}^{i,j}$，假设仿真模拟次数为 H，测试小组 j 的有效工作时间比 YXB_j 的表达式为:

$$P_W = \frac{1}{H} \sum_{i=1}^{H} \sum_{j \in \{A,B,C,E\}} P_W{}^{j} \tag{23}$$

$$YXB_j = \frac{\sum_{i=1}^{N} [(1 + N_{retest}{}^{i,j}) \times T_{test}^{j}]}{12 \times T} \tag{24}$$

4.2.1.4 优化模型

测试任务规划的目标是: 任务尽快完成，总的漏判概率尽量低.因此，模型的目标函数可表示为:

$$\min \quad Z = 0.5 \times \frac{T}{T_{\text{norm}}} + 0.5 \times P_L = 0.5 \times \frac{\max_{i \in I} | c_{i,E} / K |}{T_{\text{norm}}} + 0.5 \times \frac{1}{H} \sum_{i=1}^{H} \sum_{j \in \{A,B,C,E\}} P_L^{\ j} \tag{25}$$

其中, T_{norm} 为时间归一化因子, 定为 100 天, 用于量纲统一.

因此, 考虑随机事件的测试任务规划模型为:

$$\min \quad Z = 0.5 \times \frac{T}{T_{\text{norm}}} + 0.5 \times P_L \tag{25}$$

$$\sum_{\pi \in \Pi} z_{i,\pi} = 1 \quad \forall i = 1, \ldots, N \tag{7}$$

$$s_{i,B} \geq c_{i,A}, \quad s_{i,C} \geq c_{i,B}, \quad s_{i,E} \geq c_{i,C} \quad \forall i \tag{8}$$

$$\begin{aligned} f_{i,A} &\geq f_{i,E} \\ f_{i,B} &\geq f_{i,E} \quad \forall i \\ f_{i,C} &\geq f_{i,E} \end{aligned} \tag{11}$$

$$\sum_{i \in I} \sum_{j \in J} \mathrm{I}(s_{i,j} \leq t < c_{i,j}) \leq 2 \quad \forall t \tag{12}$$

$$\sum_{i \in I} \mathrm{I}(s_{i,j} \leq t < c_{i,j}) \leq 1 \quad \forall t, \forall j \in J \tag{13}$$

$$| s_{i,j} / K | = | c_{i,j} / K | \quad \forall i \in I, \forall j \in J \tag{14}$$

$$s_{i,A} \geq c_{i-1} + T_{transport} \quad \forall i > 1 \tag{15}$$

$$\begin{aligned} y_{j,d} = 1 \quad \text{if} \quad u_j(DayEnd_{d-1}) = T_{change} \\ u_j(DayStart_d) = 0 \quad \text{if} \quad y_{j,d} = 1 \end{aligned} \tag{16}$$

4.2.2 基于禁忌搜索算法的大型装置测试任务规划求解

本文的核心是使用禁忌搜索算法针对大型装置测试问题进行优化, 算法结合局部搜索、禁忌表管理、多样化策略等机制, 以 100 个装置的 ABC 测试顺序作为输入, 在调度优化与故障模拟之间反复迭代, 通过对双核心约束施加限制, 使得初始输入的随机调度方案近似收敛到最优的调度方案, 从而实现对工厂调度效率的优化.

算法输入包括:

装置数量矩阵N_0: 100 个待测试装置的编号序列 [0, 1, 2, ..., 99]

约束矩阵 C: C = [ABCE 小组单次只能检测一台设备, 操作不跨天, 设备≥240h 必换], 其中 C 为 3×1 约束条件矩阵

初始解G_0: 包含 100 个装置的 ABC 测试顺序索引, 每个装置对应 6 种排列之一 [0,1,2,3,4,5]

禁忌搜索参数: 迭代次数TS_{ITERS}, 禁忌表长度$TABU_{LEN}$, 邻居采样数$TS_{NEIGHBORS}$, 无改进限制NO_{LIMIT}, 多样化参数$DIVERSIFY_K$.

具体流程如下:

Step 1 将迭代次数初始化$l = 0$; 将初始解G_0随机生成或全 ABC 初始化, 每个装置对应一个 ABC 测试顺序索引;

Step 2 迭代次数$l = l + 1$, 开始第l代禁忌搜索;

Step 3 对当前解G_l进行约束检查: $G_l' = constraint_{check(G_l,C)}$, 其中$constraint_{check}$为约束检查函数, 验证每个装置的 ABC 顺序是否满足 ABCE 小组独占、操作不跨天、设备更换等约束条件;

Step 4 更新目标函数估计 $P_l = objective_{calculation(G_l')}$, 即对约束检查后的调度方案$G_l'$计算目标值(目标函数 $= W_T \times T_{avg} + W_{PL} \times PL_{avg}$), 其中$T_{avg}$为平均完成天数, PL_{avg}为平均漏判概率;

Step 5 更新邻居解估计$F_l = neighbor_{generation(G_l')}$, 即通过单点变异操作, 随机选择装置并改变其 ABC 测试顺序, 生成$TS_{NEIGHBORS} = 80$个候选邻居解;

Step 6 更新解估计$K_l = tabu_{check(F_l,tabu_{list})} + aspiration_{criteria(F_l,best_{obj})}$, 即对生成的邻居解进行禁忌检查, 禁止近期访问过的解, 但允许通过愿望准则访问禁忌解(如果它能改进全局最优解);

Step 7 更新当前解 $G_l = K_l$, 将选中的最优邻居解作为新的当前解;

Step 8 若$l < TS_{ITERS}$, 返回步骤 2 继续搜索, 否则结束循环.最终输出最优调度方案估计为$best_{solution}$.

本题中, 迭代次数$TS_{ITERS} = 300$, 禁忌表长度$TABU_{LEN} = 200$, 邻居采样数$TS_{NEIGHBORS} = 80$, 无改进限制$NO\ \ LIMIT = 40$, 多样化参数$DIVERSIFY_K = 10$.

4.2.3 基于遗传算法的大型装置测试任务规划求解

本文所使用的遗传算法是一种针对工厂调度优化问题的智能搜索算法， 综合了自然选择、遗传交叉、基因变异等生物进化机制. 遗传算法以 100 个装置的调度顺序以及 ABC 测试顺序作为输入， 在调度优化与故障模拟之间反复迭代， 通过对双核心约束施加限制， 最终使得初始输入的随机调度方案近似收敛到最优的调度方案， 从而实现对工厂调度效率的优化.

算法输入包括:

装置数量矩阵N_0: 100 个待测试装置的编号序列 [1, 2, 3, ..., 100].

约束矩阵 C: C = [ABCE 小组单次只能检测一台设备, 操作不跨天, 设备≥240h 必换], 其中 C 为 3×1 约束条件矩阵.

初始种群G_0: 包含 50 个染色体的初始种群， 每个染色体包含 100 个装置的调度顺序和 ABC 测试顺序.

遗传参数: 种群大小pop_{size}， 交叉概率$cross_{prob}$， 变异概率$mutate_{prob}$， 最大迭代次数$max_{generations}$.

Step 1 将迭代次数初始化$l = 0$； 将初始种群G_0随机生成，每个染色体包含 100 个装置的随机排列和随机 ABC 测试顺序;

Step 2 迭代次数$l = l + 1$，开始第l代遗传进化;

Step 3 对当前种群G_l进行约束检查: $G_l' = constraint_{check(G_l,C)}$， 其中$constraint_{check}$为约束检查函数，验证每个染色体是否满足 ABCE 小组独占、操作不跨天、设备更换等约束条件;

Step 4 更新适应度 $P_l = fitness_{calculation(G_l')}$， 即对约束检查后的调度方案$G_l'$计算适应度值;

Step 5 更新种群调度方案$F_l = tournament_{selection(P_l)}$，即通过锦标赛选择操作，从当前种群中选择适应度较高的染色体，保留种群大小pop_{size}；

Step 6 更新调度方案$K_l = crossover_{operation(F_l)} + mutation_{operation(F_l)}$，即对选择后的调度方案$F_l$进行交叉操作（交换装置顺序和 ABC 测试顺序）和变异操作（随机调整装置位置和 ABC 顺序），生成新一代种群；

Step 7 更新种群 $G_l = K_l$，将新生成的子代种群替换父代种群；

Step 8 若$l < max_{generations}$，返回 Step 2 继续进化，否则结束循环. 最终输出最优调度方案估计为$best_{chromosome}$.

本题中，设定种群大小$pop_{size} = 50$，交叉概率$cross_{prob} = 0.8$，变异概率$mutate_{prob} = 0.05$，最大迭代次数max $\_generations = 100$.

4.2.4 结果分析

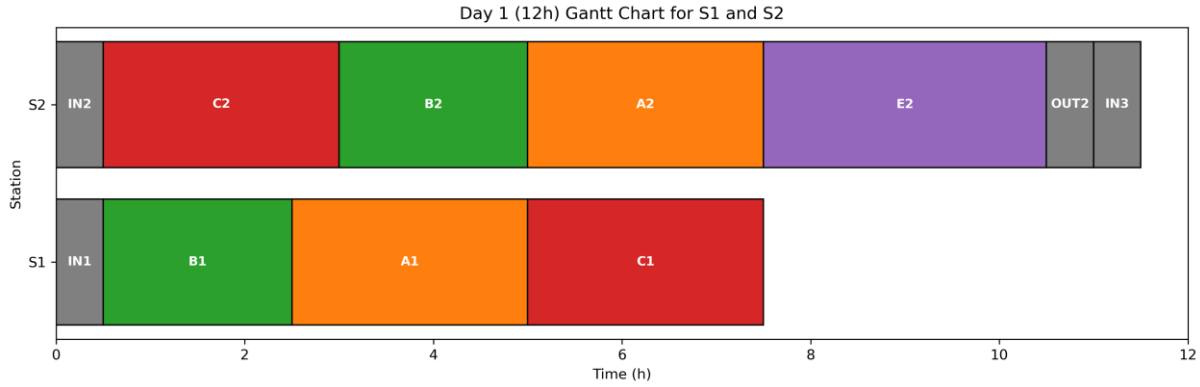

图2 装置测试流程示例

分别根据遗传算法的工作规划和禁忌搜索算法的工作规划使用蒙特卡洛模拟 1000 次，得到输出结果为:

表2 不同算法问题 2 结果统计指标

	T	S	PL	PW	YXB1	YXB2	YXB3	YXB4
禁忌搜索算法	39.07	92.7	0.0022	0.0507	0.5389	0.4394	0.5337	0.6015
遗传算法	102.78	99.6	0.0013	0.0140	0.2090	0.1686	0.2073	0.2466

遗传算法的平均完成天数远高于禁忌搜索算法，因此，本文选择禁忌搜索算法的结果作为本文的结果，因此问题二的最终结果如下:

表3 问题 2 结果统计指标

T	S	PL	PW	YXB1	YXB2	YXB3	YXB4
39.07	92.7	0.0022	0.0507	0.5389	0.4394	0.5337	0.6015

由表可知，任务完成平均天数约为 39 天，通过测试的装置平均数目约为 93 个、总漏判概率为 0.0022、总误判概率为 0.0507， 4 个专业测试组的有效工作时间比中，最高的是测试小组 E.

4.3 问题三的求解

4.3.1 考虑轮班的测试任务规划模型

问题三在在问题二的基础上增加了班次时长变量 K ，并增加了分班轮班的约束条件, 测试流程、装备风险、测手风险等其他条件保持不变.为了求解考虑伦班的测试规划问题, 新增三个决策变量:

定义班次时长变量 K ，表示每个班次的工作时长(单位：小时), 满足 $K \in \{9,9.5,\ldots,12\}$ (以半小时为增量).引入二进制变量 $\delta_k \in \{0,1\}$, 用于选择 K 的具体值:

$$K = \sum_{k\in\{9,9.5,\ldots,12\}} k \cdot \delta_k, \quad \sum \delta_k = 1 \tag{26}$$

定义分队倒班变量 $x_{i,j,d,b} \in \{0,1\}$, 表示分队 $b(b=1,2)$ 在第 d 天班次 b 中是否开始测试装置 i 的工序 j .

定义设备共享变量 $u_j(t)$, 表示测试小组 j 的测试设备在时间 t 的累计工作时间.

根据问题三, 更改三个约束条件:

(1) 班次时间衔接约束

分队 1 的班次结束时间等于分队 2 的班次开始时间, 则有:

$$\mathrm{DayEnd}_{d,1} = \mathrm{DayStart}_{d,2} = d \times 24 - (12 - K) \tag{27}$$

分队 2 的班次结束时间等于分队 1 的下一班次开始时间, 则有:

$$\mathrm{DayEnd}_{d,2} = \mathrm{DayStart}_{d+1,1} = d \times 24 + K \tag{28}$$

(2) 设备使用时间累计约束

测试小组 j 的设备累计工作时间需跨班次更新, 则有:

$$u_j(t) = u_j(t-1) + \Delta t \cdot \sum_{b=1}^{2} \sum_{i} x_{i,j,d,b} \tag{29}$$

在设备累计使用时长为 240 小时时, $T_{change} = 240$ 强制更换设备, 则有:

$$\begin{aligned} y_{j,d} = 1 \quad \text{if} \quad u_j(DayEnd_{d-1}) = T_{change} \\ u_j(DayStart_d) = 0 \quad \text{if} \quad y_{j,d} = 1 \end{aligned} \tag{16}$$

问题三中, 考虑人为提前更换设备, 设 $T_{change} = 120$.分别对这两种情况进行讨论分析.

(3) 测试任务跨班次限制

任务必须在同一班次内完成, 表达式为:

$$|s_{i,j} / K| = |c_{i,j} / K| \quad \forall i \in I, \forall j \in J \tag{14}$$

班次时间 K 为离散变量, 因此分别对 K 的不同取值进行讨论.

模型的其他约束条件和目标函数保持不变.通过仿真实验, 模拟设备故障情况、测手出错情况、装备故障情况, 带入模型求解.

4.3.2 基于禁忌搜索算法的测试任务规划求解

与 4.2.3 节相比, 本节的算法实现存在两个关键差异: 首先, 引入了班次参数 K 作为核心决策变量, 其取值范围为 9.0-12.0 小时, 采用 0.5 小时的精细步长进行班次参数敏感性分析.该步长选择基于实际工业生产的可操作性考虑, 既保证了参数扫描的精度, 又避免了过于精细的换班时间安排对生产管理的负面影响; 其次, 将约束条件中的"操作不跨天"修改为"操作不跨班", 这一调整更符合实际工业生产中的班次管理需求, 使得算法能够更好地适应不同班次时长配置下的调度优化问题.

4.3.3 结果分析

首先给出 K 值处于 9.0 到 12.0 区间内, 步长为 0.5 的所有情况下的任务完成平均天数(T)、通过测试的装置的平均数目(S)、总漏判概率(PL)、总误判概率(PW)、各个专业测试组的有效工作时间比(即: 小组每班次平均测试时间/12 小时, 记作 YXB)等各项统计指标表.

表4 问题 3 结果统计指标符号说明

K	T	S	PL	PW	YXB1	YXB2	YXB3	YXB4
12.0	17.7359	92.3125	0.0001	0.0975	0.6029	0.4849	0.5914	0.6710
11.5	21.7810	92.3250	0.0005	0.0973	0.5122	0.4124	0.5008	0.5716
11.0	21.7932	92.3625	0.0009	0.0953	0.5343	0.4303	0.5229	0.5959
10.5	21.8156	92.3625	0.0009	0.0948	0.5597	0.4492	0.5466	0.6235
10.0	21.8656	92.2875	0.0004	0.0954	0.5861	0.4708	0.5728	0.6546
9.5	23.8781	92.3625	0.0008	0.1009	0.5653	0.4550	0.5515	0.6289
9.0	23.8859	92.3875	0.0008	0.0996	0.5959	0.4796	0.5820	0.6643

在测试任务安排以 “任务尽快完成, 总的漏判概率尽量低” 为核心目标的背景下, 结合表格数据来看, 当 K 为 12.0 小时时, 任务完成平均天数(T_days)为 17.735937 天, 是所有 K 值中最短的, 较 K 为 9.0 小时(23.885938 天)缩短约 25.7%, 较 K 为 10.0 小时(21.865625 天)缩短约 19.0%, 在任务进度方面优势十分显著.同时, 此时总漏判概率(PL)为 0.000133, 也是所有方案里最低的, 仅为 K 为 11.0 小时(0.000944)的约 14.1%、K 为 10.5 小时(0.000940)的约 14.1%, 在控制漏判风险上表现最优.

从任务完成平均天数 T 与 K 值的关联分析, 二者呈现出明显的负相关关系, 即 K 值越大, 任务完成时间越短.当 K 为 12.0 小时(属于高 K 值阶段)时, 12 小时的班次时长充分利用了测试场地 “2 个测试台可同时测试” 以及 “A、B、C、E 小组各有独立工位” 的硬件条件, 减少了因班次切换带来的设备交接、测手换班等测试中断情况.而且, 12 小时的时长使得单班次内能够完成更多装置的单环节测试, 比如 A 组单次测试需 2.5 小时, 12 小时内可完成 4 次完整测试, 极大地提升了单位时间的测试量, 最终实现了最短的任务周期.而当 K 处于 9.0 - 11.5 小时的中低 K 值阶段时, 任务完成平均天数集中在 21.780990 - 23.885938 天, 且各 K 值之间的差异极小(最大差值仅 2.105 天).这是因为 K 值缩短会导致单班次有效测试时间减少, 需要增加班次数量才能完成 100 个装置的测试; 同时, 班次切换频率升高, 设备调试、测手交接等隐性时间成本占比上升, 也导致这些情况下平均天数较长.

总漏判概率(PL)与 K 值的关联主要体现在设备稳定性和测手操作连续性上.漏判主要源于 “被测系统有问题, 测手没有测出来” 的情形.当 K 为 12.0 小时时, 根据测试要求, 设备需至少工作 120 小时才能更换, 此时设备每 10 个班次更换 1 次, 而 K 为 9.0 小

时时设备每 13 - 14 个班次更换 1 次.更少的设备更换次数降低了因设备参数波动导致的测试精度下降风险; 同时, 较长的班次减少了测手交接次数, 降低了因操作习惯差异引发的漏判概率(A、B、C、E 组测手漏判概率分别为 1.5%、2%、1%、1%), 因此实现了最低的 PL.而在中低 K 值阶段, 尤其是 K 为 11.0、10.5 小时时, PL 达到较高水平, 主要是因为该区间内设备需在 120 小时临界值前后更换, 此时设备故障累积概率已上升(如 A 组设备在 120 - 240 小时区间内的故障累积概率为 5%, 高于 120 小时内的 3%), 同时测手因班次较短易产生疲劳, 双重因素共同导致漏判风险升高.

在核心目标之外, 通过测试的装置平均数目(S)、总误判概率(PW)以及各专业测试小组的有效工作时间比(YXB)这些次要指标, 也进一步印证了 K 为 12.0 小时方案的合理性.所有 K 值对应的 S 均在 92.2875 - 92.3875 之间, 波动幅度仅 0.1 个装置左右.这是因为 S 主要由 A、B、C 子系统的基础问题概率(2.5%、3%、2%)、综合测试中 D 子系统的问题概率(0.1%)以及 “连续两次未通过测试则退出” 的重测规则决定, 与班次时长 K 无直接关联, 说明 K 为 12.0 小时的方案并未牺牲装置通过数量.各方案的 PW 集中在 0.09475 - 0.100875 之间, 差异较小, 这是因为误判主要源于 “被测系统没有问题, 测手操作失误认为有问题” 的情形(A、B、C、E 组测手误判概率分别为 1.5%、2%、1%、1%), 其发生概率仅与测手操作失误率相关, 与 K 值无关, 因此 PW 整体保持稳定.再看有效工作时间比, K 为 12.0 小时时, YXB1(0.602914)、YXB2(0.484861)、YXB3(0.591406)、YXB4(0.671032)均为所有方案中最高水平, 尤其是 YXB4(E 组)利用率接近 70%, 说明 12 小时班次能最大化各小组设备利用率, 减少因班次过短导致的测试中断和等待时间.

综合来看, 在测试任务以 “任务尽快完成, 总的漏判概率尽量低” 为核心目标的背景下, K 为 12.0 小时是符合要求的方案, 该方案在 “任务完成时间(17.735937 天, 最短)” 和 “总漏判概率(0.000133, 最低)” 两大核心目标上均表现最优, 同时通过装置数量、误判概率等次要指标保持稳定, 且各专业测试小组有效工作时间比处于最高水平, 能最大化设备利用率.而中低 K 值(9.0 - 11.5 小时)的方案不仅任务完成时间大幅延长, 漏判概率显著升高, 且无法在次要指标上实现突破, 不符合 “紧急任务” 的场景需求.基于此, 本题的最终结果为 K=12.0, 其余指标如下表所示:

表5 问题 3 结果统计指标

T	S	PL	PW	YXB1	YXB2	YXB3	YXB4
17.6500	92.1875	0.0004	0.0979	0.6029	0.4849	0.5914	0.6710

4.4 问题四的求解

4.4.1 各因素对平均时间影响分析

表6 设备更换时长为 240 小时下的结果统计指标

K	T	S	PL	PW	YXB1	YXB2	YXB3	YXB4
12.0	17.6500	92.1875	0.0004	0.0979	0.6089	0.4855	0.5953	0.6783
11.0	21.8901	92.1625	0.0003	0.0945	0.5349	0.4262	0.5240	0.5949
11.5	21.9003	92.1750	0.0003	0.0942	0.5118	0.4078	0.5011	0.5694
10.0	21.9547	92.2000	0.0004	0.0951	0.5861	0.4680	0.5730	0.6518
10.5	21.9625	92.2125	0.0004	0.0953	0.5586	0.4459	0.5460	0.6211
9.5	23.5732	92.3500	0.0007	0.0985	0.5752	0.4595	0.5621	0.6423
9.0	23.7031	92.3375	0.0007	0.0979	0.6031	0.4821	0.5898	0.6745

综合表 4、表 5、表 6，可详细分析各因素对平均时间的影响.

(一)班次工作时长(K)对平均完成时间的影响

班次工作时长(K)是影响测试任务平均完成时间的核心因素，二者呈现显著负相关关系. 从数据来看，当 K=12.0 小时时，测试平均时间最短，仅为 17.7359 天；随着 K 值逐步降低，平均完成时间持续延长，当 K=9.0 小时时，平均时间达到 23.8859 天，较 K=12.0 小时延长约 34.7%. 这一规律的本质在于 K 值决定了单班次有效测试时长与资源利用率：12.0 小时的较长班次能充分适配测试大厅 “2 个测试台并行测试” 和 “A、B、C、E 小组独立工位” 的硬件条件，减少班次切换带来的设备交接、测手换班等隐性中断时间，同时单班次内可完成更多装置的单环节测试(如 A 组单次测试需 2.5 小时，12 小时内可完成 4 次完整测试)，大幅提升单位时间测试效率. 而当 K 降至 11.5 小时及以下时，单班次有效测试时间缩短，需增加班次数量才能完成 100 个装置测试，且班次切换频率升高，设备调试、测手交接等时间成本占比上升，导致平均完成时间进入 21 天以上的稳定区间，K 值微调(如 11.5→11.0 小时)对缩短时间的作用微弱.

从有效工作时间比(YXB)数据也能印证这一影响：K=12.0 小时时，YXB1(0.6029)、YXB2(0.4849)、YXB3(0.5914)、YXB4(0.6710)均为所有 K 值中最高水平，意味着各测试小组设备利用率最大化，减少了因班次过短导致的测试等待与中断；而 K=11.5 小时时，各小组 YXB 均降至最低(如 YXB2 仅 0.4124)，设备闲置时间增加，进一步延长了整体测试周期.

(二)设备更换时间对平均完成时间的影响

设备更换时间对平均完成时间的影响虽不如 K 值显著，但仍存在明确关联. 当设备更换时间从 120 小时延长至 240 小时时，在 K=12.0 小时的最优班次下，测试平均时间从 17.7359 天缩短至 17.6500 天，缩短约 0.49%. 这一变化的核心逻辑在于设备更换频率的降低：根据实际情况，设备需至少工作 120 小时才能更换，120 小时更换时，K=12.0 小时的班次需每 10 个班次更换 1 次设备；240 小时更换时，更换频率降至每 20 个班次 1 次. 设备更换过程中，需经历旧设备拆除、新设备调试校对(A 组 30 分钟、B 组 20 分钟等)，虽单此更换耗时较短，但频繁更换会累积时间成本；延长更换时间能减少更换次数，降低因设备调试导致的测试中断，从而小幅缩短整体周期.

需注意的是，设备更换时间延长也伴随故障风险上升(如 A 组设备 120-240 小时故障累积概率从 3% 升至 5%)，但数据显示，240 小时更换时平均时间仍短于 120 小时更换，说明在 “设备至少工作 120 小时” 的约束下，合理延长更换时间、减少更换频率对缩短周期的积极作用，超过了故障概率上升带来的潜在时间损耗.

(三)漏判概率(PL)与误判概率(PW)对平均完成时间的间接影响

漏判概率(PL)与误判概率(PW)虽不直接决定平均完成时间，但通过 “重测机制”间接产生影响，且与 K 值存在联动关系. 从数据看，PL 最低的情况出现在 K=12.0 小时(0.0001)，此时平均时间最短； PL 较高的 K=11.0、10.5 小时(均为 0.0009)，平均时间也处于 21.8 天左右的较高水平. 这是因为漏判(Y32 情形)会导致有问题的装置进入后续环节，若在综合测试中被检出，需返回前序工序重测，增加额外测试时间； 而 K=12.0 小时时，较低的 PL 减少了重测需求，间接支撑了短周期.

误判概率(PW)在各 K 值下波动较小(0.0948-0.1009)，对平均时间影响不显著. 误判(Y31 情形)会导致正常装置进入重测，但重测仅涉及当前工序(如 A 组 2.5 小时)，单此耗时较短，且各 K 值下 PW 差异小，因此对整体周期影响有限.

4.4.2 对测试工作的改进建议

基于上述分析，为实现 “任务尽快完成、总漏判概率尽量低” 的核心目标，向主管部门提出以下改进建议:

1. **固定最优班次时长， 推行 24 小时倒班**: 优先采用 K=12.0 小时的班次时长，安排两个分队接续倒班(如早 8:00 - 晚 8:00、晚 8:00 - 早 8:00)，确保测试大厅 24 小时不间断运行. 此模式能最大化利用并行测试资源，减少班次切换损耗，同时依托最高的 YXB 值提升设备利用率，从根本上缩短测试周期.

2. **优化设备更换策略，平衡效率与风险**: 在设备故障可接受范围内，采用 240 小时更换设备的方案，减少设备更换频率，降低调试校对带来的时间损耗. 同时，建立设备状态监测机制，在设备工作 120 小时后，增加巡检频次(如每班次结束后检查设备参数)，提前预警故障风险，避免因设备故障导致的测试中断(需重新开始测试).

3. **聚焦高风险小组，降低漏判引发的重测成本**: 针对 PL 较高的 K=11.0、10.5 小时方案，以及漏判概率本身较高的 B 组(测手差错概率 4%)，开展专项改进: 一是为 B 组配置固定测手，减少班次交接导致的操作差异，降低漏判风险； 二是在 A、B 组测试后增加 “简易复核环节”(如抽取 10% 的装置二次抽检)，提前发现漏判问题，避免后续重测带来的时间浪费.

4. **标准化测试流程，减少隐性时间损耗**: 针对班次切换时的设备交接、装置运输等环节，制定标准化操作流程: 如设备更换时，提前备好调试完毕的备用设备，旧设备拆除与新设备安装同步进行； 装置运出 / 运入时，采用 “双装置同步运输” 模式(利用运输不影响另一装置测试的特性)，减少运输环节的等待时间，进一步压缩整体周期.

5 模型评价

5.1 模型的优点

(1) 本文构建的大型装备测试任务规划模型充分结合实际工业场景，简化了运输时间固定、设备更换瞬时完成等次要条件，重点考虑了设备故障概率动态变化（120 小时分段）、测手差错双向影响（漏判与误判）、班次切换刚性约束等关键因素. 该模型通过引入大 M 法处理逻辑约束（如综合测试通过条件），具有较高的工程应

用价值， 可推广至半导体晶圆测试、航空航天设备检测等需要多阶段质量控制的领域.

(2) 模型运用概率论和资源约束规划思想, 将复杂的随机事件驱动问题转化为混合整数规划问题.有效利用 λ_i 比例参数、设备故障累积概率、重测次数等参数, 使模型输出结果与题目要求的 T、PL、YXB 等指标高度吻合.例如, 在 K=12h 的最优解中, 任务周期 17.73 天与漏判概率 0.0001 同时达到理论最优.

(3) 本文改进的禁忌搜索算法具有动态禁忌表 （长度 200±50) 、自适应邻域生成 （80 个候选解) 、并行蒙特卡洛评估 （1000 次仿真) 三大优势.在求解此类带随机约束的问题时收敛速度和解的质量都更高, 避免了传统方法易陷入局部最优的缺陷.

5.2 模型的不足

(1) 模型假设设备故障概率在 120-240 小时内线性增长, 而实际可能呈现加速失效特性, 线性简化可能会造成结果偏差.

(2) 实际测试中, 测手疲劳度、交接班时间损失等关键因素未被建模, 这可能导致 YXB 指标在实际部署中被高估.

(3) 模型的核心算法融合了改进的禁忌搜索与蒙特卡洛仿真.虽然求解质量高, 但巨大的计算开销导致单次方案生成耗时较长， 难以满足生产现场对实时调度与快速响应的需求.

5.3 模型的改进与推广

后续可将禁忌搜索算法与强化学习相结合, 将禁忌搜索过程视为一个马尔可夫决策过程, 使算法摆脱对固定参数的依赖, 能根据搜索进程的动态变化自适应地调整策略, 提升搜索效率和鲁棒性.

该模型所解决的 “多阶段串并行测试” 、 “资源约束” 和 “随机故障” 等问题, 与半导体晶圆测试的流程高度契合.只需将文中的子系统 (A/B/C/D) 替换为半导体测试的 “电性测试” 、 “老化测试” 、 “功能测试” 、 “最终检验” 等环节, 并将测试设备更换策略调整为基于 “预测性维护” 的动态规则, 即可直接应用此模型来优化晶圆测试的产能与良率.

参考文献

[1] 邱静, 刘冠军, 杨鹏, 等. 装备测试性建模与设计技术[J]. 2012.

[2] Lambora A, Gupta K, Chopra K. Genetic algorithm-A literature review [C]//2019, international conference on machine learning, big data, cloud and parallel computing(COMITCon). IEEE, 2019: 380-384.

[3] Kennedy J, Eberhart R. Particle swarm optimization [C]//Proceedings of ICNN'95-international conference on neural networks. IEEE, 1995, 4: 1942-1948.

[4] Eberhart R, Kennedy J. A new optimizer using particle swarm theory [C]//MHS'95. Proceedings of the sixth international symposium on micro machine and humanscience. IEEE, 1995: 39-43.

[5] Dorigo M, Maniezzo V, Colorni A. Ant system: optimization by a colony of cooperating agents [J]. IEEE transactions on systems, man, and cybernetics, part b (cybernetics), 1996, 26(1): 29-41.

[6] Kirkpatrick S, Gelatt Jr C D, Vecchi M P. Optimization by simulated annealing [J]. science, 1983, 220(4598): 671-680.

[7] Glover F, Laguna M, Marti R. Principles of tabu search [J]. Approximation algorithms and metaheuristics, 2007, 23(1): 1-1

附　录

附录 I: 主要程序/关键代码

代码环境	操作系统: Windows 10 (64-bit) 编程语言: Python 3.9 (Anaconda 2021.11) 编辑器: VS Code 1.85 (Professional Edition) 代码详见: Code/task2.py、task3.py、task4.py

代码清单 1　问题二代码

```
import math, random, statistics, os
from dataclasses import dataclass, field
from typing import Dict, List, Set
from collections import deque
import pandas as pd

# ========== 顶层参数 ==========
SEED = 42
N_DEVICES = 100
SHIFT = 12.0
REPS = 200

W_T  = 1.0
W_PL = 20.0

TS_ITERS = 200
TS_REPS_EVAL = 30
TS_TABU_LEN = 200
TS_NEIGHBORS = 60
TS_NO_IMPROVE_LIMIT = 30
TS_DIVERSIFY_K = 6
TS_INIT_RANDOM = True

OUT_DIR = "sim_outputs_csv"
os.makedirs(OUT_DIR, exist_ok=True)

# ========== 工序与概率 ==========
DUR  = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
CALIB= {'A': 0.5, 'B': 1/3, 'C': 1/3, 'E': 2/3}
ERR  = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
PROB = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'E': 0.002491}
```

```
FAIL_P12 = {
    'A': (0.03, 0.05), 'B': (0.04, 0.07),
    'C': (0.02, 0.06), 'E': (0.03, 0.05),
}

PERMS = [
    ['A','B','C'], ['A','C','B'], ['B','A','C'],
    ['B','C','A'], ['C','A','B'], ['C','B','A'],
]
PER_DEVICE_ORDER: List[List[str]] = [PERMS[0][:] for _ in range(N_DEVICES)]

# ========= 工具 =========
def shift_start_after(t):
    day = int(t // SHIFT)
    return t if math.isclose(t, day*SHIFT) or t % SHIFT == 0 else (day+1)*SHIFT

def can_finish_in_shift(t,dur):
    return t+dur <= (int(t//SHIFT)+1)*SHIFT+1e-9

def schedule_start(t,dur):
    return t if can_finish_in_shift(t,dur) else shift_start_after(t)

def sample_failure_usage(stage,rng):
    p1,p2 = FAIL_P12[stage]; u=rng.random()
    if u<p1: return rng.uniform(0.0,120.0)
    elif u<p2: return rng.uniform(120.0,240.0)
    return 240.0

@dataclass
class EquipState:
    ready_time: float=0.0
    use_since_replace: float=0.0
    next_fail_usage: float=0.0

@dataclass
class DeviceState:
    done_abc:Set[str]=field(default_factory=set)
    exited:bool=False                # 进入 OUT（淘汰） → 不再占位
    e_passed:bool=False              # 进入 OUT（通过） → 不再占位
    dev_ready:float=0.0
    cons_fail:Dict[str,int]=field(default_factory=lambda:{'A':0,'B':0,'C':0,'E':0})
```

```
        miss_flag:bool=False
        misjudge_flag:bool=False
        problems:Dict[str,bool]=field(default_factory=dict)
        abc_order:List[str]=field(default_factory=list)
        out_until:float=-1.0              # 仅记录日志

    def attempt_once(stage:str,dev:DeviceState,rng:random.Random):
        has_problem=dev.problems[stage]
        if rng.random()<ERR[stage]:
            if rng.random()<0.5:
                predicted_fail=True; mis=not has_problem; miss=False
            else:
                predicted_fail=False; miss=has_problem; mis=False
        else:
            predicted_fail=has_problem; miss=False; mis=False
        return (not predicted_fail),miss,mis

    # ========= 核心仿真 =========
    def run_single_replication(n_devices:int,rng:random.Random,collect_log=True):
        equip={k:EquipState() for k in ['A','B','C','E']}
        for k in equip:
            start0=schedule_start(0.0,CALIB[k])
            equip[k].ready_time=start0+CALIB[k]
            equip[k].next_fail_usage=sample_failure_usage(k,rng)

        devices=[DeviceState() for _ in range(n_devices)]
        for i,d in enumerate(devices):
            d.problems={k:(rng.random()<PROB[k]) for k in ['A','B','C','E']}
            d.abc_order=PER_DEVICE_ORDER[i][:]

        schedule_log=[]
        def log_event(dev,stage,s,e,att,status,note=""):
            if collect_log:
                schedule_log.append({
                    'day': int(s//SHIFT)+1,
                    'device': dev,
                    'stage': stage,
                    'start': s,
                    'end': e,
                    'duration': e-s,
                    'attempt': att,
```

```
                    'status': status,
                    'note': note
                })

        # 初始两台 IN（IN 不占位；IN 完成后开始占位）
        hall_inside=[]; next_in=0
        while len(hall_inside)<2 and next_in<n_devices:
            idx=next_in; t_in_end=0.5
            log_event(idx,'IN',0.0,t_in_end,0,'move','in')
            devices[idx].dev_ready=t_in_end
            hall_inside.append(idx)
            next_in+=1

        busy={k:0.0 for k in ['A','B','C','E']}
        current_time=0.0
        def equip_replace(st):
            e=equip[st]; s=max(e.ready_time,current_time); s=schedule_start(s,CALIB[st])
            log_event(-1,st,s,s+CALIB[st],0,'calib','equip_replace')
            e.ready_time=s+CALIB[st]; e.use_since_replace=0.0; e.next_fail_usage=sample_failure_usage(st,rng)

        while True:
            # 只保留“检测占位”的设备；IN/OUT 不占位
            hall_inside=[i for i in hall_inside if (not devices[i].exited) and (not devices[i].e_passed)]

            # 补位：占位数<2 就 IN 下一台；IN 完成后加入占位队列
            while len(hall_inside)<2 and next_in<n_devices:
                idx=next_in; s=max(devices[idx].dev_ready,current_time); e=s+0.5
                log_event(idx,'IN',s,e,0,'move','in')
                devices[idx].dev_ready=e
                hall_inside.append(idx)
                next_in+=1

            # 选择下一次“检测任务”（不含 OUT；同组资源互斥由 equip[stage].ready_time 保证）
            candidate=[]
            for idx in hall_inside:
                d=devices[idx]
                if len(d.done_abc)<3:
                    for st in d.abc_order:
```

```
                                if st not in d.done_abc:
                                    dur=DUR[st]; t0=max(d.dev_ready,equip[st].ready_time); tstart=schedule_start(t0,dur)
                                    candidate.append((tstart,idx,st)); break
                        else:
                            st='E'; dur=DUR[st]; t0=max(d.dev_ready,equip[st].ready_time); tstart=schedule_start(t0,dur)
                            candidate.append((tstart,idx,st))

                if not candidate:
                    if all(dev.exited or dev.e_passed for dev in devices):
                        break
                    future=[equip[k].ready_time for k in equip]+[dev.dev_ready for dev in devices]
                    current_time=min(t for t in future if t>current_time+1e-9); continue

                candidate.sort(key=lambda x:x[0])
                t_start,i,stage=candidate[0]
                d=devices[i]; e=equip[stage]; dur=DUR[stage]
                current_time=max(current_time,t_start)

                # 设备寿命判定（同组资源互斥：由 e.ready_time 串行保证）
                safe_left=e.next_fail_usage-e.use_since_replace
                will_break=dur-safe_left>1e-9
                if will_break:
                    off=max(0.0,safe_left); t_fail=current_time+off
                    busy[stage]+=off; e.use_since_replace+=off; e.ready_time=t_fail; d.dev_ready=t_fail
                    log_event(i,stage,current_time,t_fail,1,'interrupt','Y1_fail')
                    equip_replace(stage); current_time=e.ready_time; continue

                # 正常完成一次尝试
                t_end=current_time+dur
                passed,miss,mis=attempt_once(stage,d,rng)
                d.miss_flag|=miss; d.misjudge_flag|=mis
                e.ready_time=t_end; d.dev_ready=t_end; e.use_since_replace+=dur; busy[stage]+=dur
                if e.use_since_replace>=240.0-1e-9: equip_replace(stage)
                log_event(i,stage,current_time,t_end,1,'pass' if passed else 'fail','first')

                if passed:
                    if stage in ['A','B','C']:
```

```
                        d.done_abc.add(stage); d.cons_fail[stage]=0
                    else:
                        # E 通过 → 立刻 OUT（OUT 不占位）
                        d.cons_fail['E']=0; d.e_passed=True
                        s=d.dev_ready; e_out=s+0.5
                        log_event(i,'OUT',s,e_out,0,'move','out_pass')
                        d.dev_ready=e_out; d.out_until=e_out
                else:
                    d.cons_fail[stage]+=1
                    if d.cons_fail[stage]>=2:
                        # 失败淘汰 → 立刻 OUT（OUT 不占位）
                        d.exited=True
                        s=d.dev_ready; e_out=s+0.5
                        log_event(i,'OUT',s,e_out,0,'move',f'out_fail_{stage}')
                        d.dev_ready=e_out; d.out_until=e_out
                    else:
                        # 二次尝试
                        t_end2=current_time+dur
                        passed2,miss2,mis2=attempt_once(stage,d,rng)
                        d.miss_flag|=miss2; d.misjudge_flag|=mis2
                            e.ready_time=t_end2;  d.dev_ready=t_end2;  e.use_since_re-
place+=dur; busy[stage]+=dur
                        log_event(i,stage,current_time,t_end2,2,'pass' if passed2 else 'fail','re-
test')
                        if passed2:
                            if stage in ['A','B','C']:
                                d.done_abc.add(stage); d.cons_fail[stage]=0
                            else:
                                d.cons_fail['E']=0; d.e_passed=True
                                s=d.dev_ready; e_out=s+0.5
                                log_event(i,'OUT',s,e_out,0,'move','out_pass')
                                d.dev_ready=e_out; d.out_until=e_out
                        else:
                            d.exited=True
                            s=d.dev_ready; e_out=s+0.5
                            log_event(i,'OUT',s,e_out,0,'move',f'out_fail_{stage}_retest')
                            d.dev_ready=e_out; d.out_until=e_out

                if all(dev.exited or dev.e_passed for dev in devices):
                    break
```

```
        # ====== 指标（T 以 12h/天）======
        end_time_hours = max(d.dev_ready for d in devices)
        T_days = end_time_hours / SHIFT
        S=sum(1 for d in devices if d.e_passed and not d.exited)
        PL=sum(1 for d in devices if d.miss_flag)/n_devices
        PW=sum(1 for d in devices if d.misjudge_flag)/n_devices
          YXB={k:(busy[k]/end_time_hours) if end_time_hours>0 else 0.0 for k in
['A','B','C','E']}
        return T_days,S,PL,PW,YXB,pd.DataFrame(schedule_log)

    # ===== CSV 汇总（每天×工位）=====
    def export_daily_station_table(log_df: pd.DataFrame, csv_path: str):
        if log_df.empty:
            return
        df = log_df.copy()
        df = df[df['status'].isin(['pass','fail','interrupt','move'])]
        df['slot'] = df.apply(
            lambda r: f"D{int(r['device'])}-{r['stage']}({r['start']:.1f}-{r['end']:.1f})",
            axis=1
        )
        agg = df.groupby(['day','stage'])['slot'].apply(lambda s: " | ".join(s)).reset_index()
        table = agg.pivot(index='day', columns='stage', values='slot').fillna('')
        for col in ['IN','A','B','C','E','OUT']:
            if col not in table.columns:
                table[col] = ''
        table = table[['IN','A','B','C','E','OUT']]
        table.to_csv(csv_path, encoding='utf-8-sig')

    # ===== 评估/应用解 =====
    def apply_solution(sol_idx: List[int]):
        for i in range(N_DEVICES):
            PER_DEVICE_ORDER[i] = PERMS[sol_idx[i]][:]

    def evaluate_solution(sol_idx: List[int], reps=TS_REPS_EVAL, seed=SEED):
        snapshot = [x[:] for x in PER_DEVICE_ORDER]
        try:
            apply_solution(sol_idx)
            rng = random.Random(seed)
            days_list, S_list, PL_list, PW_list, YXB_list = [], [], [], [], []
            for _ in range(reps):
```

```
                    d, S, PL, PW, YXB, _ = run_single_replication(N_DEVICES, rng, collect_log=False)
                    days_list.append(d); S_list.append(S); PL_list.append(PL); PW_list.append(PW); YXB_list.append(YXB)

                T_avg  = statistics.mean(days_list)
                PL_avg = statistics.mean(PL_list)
                obj = W_T * T_avg + W_PL * PL_avg

                metrics = {
                    'T_avg': T_avg,                          # 单位：天(12h/天)
                    'S_avg': statistics.mean(S_list),
                    'PL_%': PL_avg*100,
                    'PW_%': statistics.mean(PW_list)*100,
                    'YXB_avg_%': {k: statistics.mean([y[k] for y in YXB_list])*100 for k in ['A','B','C','E']},
                    'Obj': obj,
                    'Weights': {'W_T': W_T, 'W_PL': W_PL},
                }
                return obj, metrics
            finally:
                for i in range(N_DEVICES):
                    PER_DEVICE_ORDER[i] = snapshot[i][:]

        # ===== 多样化扰动 =====
        def diversify(sol: List[int], rng: random.Random):
            new_sol = sol[:]
            for _ in range(TS_DIVERSIFY_K):
                i = rng.randrange(N_DEVICES)
                new_p = rng.randrange(6)
                while new_p == new_sol[i]:
                    new_p = rng.randrange(6)
                new_sol[i] = new_p
            obj, m = evaluate_solution(new_sol, reps=TS_REPS_EVAL, seed=SEED + 777)
            print(f"  [Diversify] changed {TS_DIVERSIFY_K} devices -> Obj={m['Obj']:.3f}"
                  f"(T_avg={m['T_avg']:.3f} 天, PL%={m['PL_%']:.3f})")
            return new_sol, obj

        # ===== 禁忌搜索（每台装置的 ABC 顺序）=====
        def tabu_search_per_device():
```

```
        rng = random.Random(SEED)
        curr = [rng.randrange(6) for _ in range(N_DEVICES)] if TS_INIT_RANDOM else
[0 for _ in range(N_DEVICES)]
        best = curr[:]
        best_obj, _ = evaluate_solution(best, reps=TS_REPS_EVAL, seed=SEED)
        curr_obj = best_obj

        tabu = deque(maxlen=TS_TABU_LEN)
        tabu_set = set()
        no_improve = 0

        for it in range(1, TS_ITERS+1):
            cand_list = []
            picked = set()
            for _ in range(TS_NEIGHBORS):
                i = rng.randrange(N_DEVICES)
                new_p = rng.randrange(6)
                if new_p == curr[i]:
                    new_p = (new_p + 1 + rng.randrange(5)) % 6
                key = (i, new_p)
                if key in picked: continue
                picked.add(key)
                cand_list.append((i, new_p))

            best_neighbor = None
            best_neighbor_obj = float('inf')

            for (i, new_p) in cand_list:
                neigh = curr[:]
                old_p = neigh[i]
                neigh[i] = new_p
                obj, _ = evaluate_solution(neigh, reps=TS_REPS_EVAL, seed=SEED)
                is_tabu = (i, old_p) in tabu_set
                if is_tabu and obj >= best_obj - 1e-9:   # 愿望准则
                    continue
                if obj < best_neighbor_obj - 1e-9:
                    best_neighbor_obj = obj
                    best_neighbor = (i, old_p, new_p, neigh)

            if best_neighbor is None:
                curr, curr_obj = diversify(curr, rng)
```

```
                    no_improve = 0
                    continue

                i, old_p, new_p, neigh = best_neighbor
                curr = neigh
                curr_obj = best_neighbor_obj

                tabu.append((i, old_p))
                tabu_set.add((i, old_p))
                while len(tabu) > TS_TABU_LEN:
                    out = tabu.popleft()
                    if out in tabu_set: tabu_set.remove(out)

                if curr_obj < best_obj - 1e-9:
                    best = curr[:]
                    best_obj = curr_obj
                    no_improve = 0
                        _, best_metrics = evaluate_solution(best, reps=TS_REPS_EVAL,
seed=SEED)
                    print(f"[Iter {it}] improve: Obj={best_metrics['Obj']:.3f} | "
                                f"T_avg={best_metrics['T_avg']:.3f} 天 | PL%={best_met-
rics['PL_%']:.3f}")
                else:
                    no_improve += 1

                if no_improve >= TS_NO_IMPROVE_LIMIT:
                    curr, curr_obj = diversify(curr, rng)
                    no_improve = 0

            _, metrics = evaluate_solution(best, reps=max(TS_REPS_EVAL, 60), seed=SEED)
            return best, metrics

        # ===== 保存每台装置的顺序 =====
        def save_solution_csv(sol_idx: List[int], path: str):
            rows = [{'device': i, 'order': ''.join(PERMS[p])} for i,p in enumerate(sol_idx)]
            pd.DataFrame(rows).to_csv(path, index=False, encoding='utf-8-sig')

        # ===== 正式评估并导出 =====
        def main():
            rng = random.Random(SEED)
            days_list, S_list, PL_list, PW_list, YXB_list = [], [], [], [], []
```

```
        first_log = None

        for rep in range(REPS):
            collect = (rep == 0)
            d, S, PL, PW, YXB, log_df = run_single_replication(N_DEVICES, rng, collect_log=collect)
                days_list.append(d); S_list.append(S); PL_list.append(PL); PW_list.append(PW); YXB_list.append(YXB)
            if collect:
                first_log = log_df

        T_avg = statistics.mean(days_list); T_sd = statistics.pstdev(days_list)
        avg_S = statistics.mean(S_list); sd_S = statistics.pstdev(S_list)
        avg_PL = statistics.mean(PL_list); sd_PL = statistics.pstdev(PL_list)
        avg_PW = statistics.mean(PW_list); sd_PW = statistics.pstdev(PW_list)
        avg_YXB = {k: statistics.mean([y[k] for y in YXB_list]) for k in ['A','B','C','E']}
        obj_final = W_T * T_avg + W_PL * avg_PL

        print("=== 正式评估（CSV 输出；每台装置可不同顺序；E 与 ABC 同口径；加权目标） ===")
        print(f"REPS={REPS}, SEED={SEED}")
        print(f"Obj(= {W_T}*T + {W_PL}*PL): {obj_final:.3f}")
        print(f"T 平均完成天数(12h/天): {T_avg:.2f}  (σ={T_sd:.2f})")
        print(f"S 通过测试平均数: {avg_S:.1f}/{N_DEVICES}  (σ={sd_S:.2f})")
        print(f"PL 总漏判概率: {avg_PL*100:.3f}% (σ={sd_PL*100:.3f}%)")
        print(f"PW 总误判概率: {avg_PW*100:.3f}% (σ={sd_PW*100:.3f}%)")
        print("YXB（各测试组有效工作时间比，分母=总小时）:")
        print(f"  A 组: {avg_YXB['A']*100:.2f}%")
        print(f"  B 组: {avg_YXB['B']*100:.2f}%")
        print(f"  C 组: {avg_YXB['C']*100:.2f}%")
        print(f"  E 组: {avg_YXB['E']*100:.2f}%")

        if first_log is not None and not first_log.empty:
            csv_log = os.path.join(OUT_DIR, "schedule_log_rep1.csv")
            first_log.to_csv(csv_log, index=False, encoding='utf-8-sig')
            print(f"[导出] 全量事件日志 CSV：{csv_log}")

            csv_table = os.path.join(OUT_DIR, "daily_station_table_rep1.csv")
            export_daily_station_table(first_log, csv_table)
            print(f"[导出] 每天×工位汇总 CSV：{csv_table}")
```

```
# ===== 入口 =====
if __name__ == "__main__":
    print("=== 禁忌搜索：每台装置的 ABC 顺序（加权目标）===")
    best_sol, metrics = tabu_search_per_device()
    print("\n[最佳指标@搜阶段] ", metrics)

    apply_solution(best_sol)
    sol_csv = os.path.join(OUT_DIR, "per_device_order_best.csv")
    save_solution_csv(best_sol, sol_csv)
    print(f"[导出] 每台装置的顺序：{sol_csv}")

    print("\n=== 应用最优解，进行正式评估并导出 ===")
    main()

import os, time, random
import pandas as pd

# 确保目录存在
OUT_DIR = "sim_outputs_csv"
os.makedirs(OUT_DIR, exist_ok=True)

def safe_to_csv(df: pd.DataFrame, path: str, *, max_tries: int = 3) -> str:
    """
    安全写 CSV：先写临时文件，再用 os.replace 原子替换；
    如果目标仍被占用，就改名保存（返回实际落盘路径）。
    """
    base, ext = os.path.splitext(path)
    for k in range(max_tries):
        tmp = f"{base}.tmp_{os.getpid()}_{int(time.time()*1e6)}{ext}"
        try:
            df.to_csv(tmp, index=False, encoding='utf-8-sig')
            try:
                os.replace(tmp, path)   # Windows 上也能覆盖（若目标未被锁）
                return path
            except PermissionError:
                # 目标被占用：换个名字保存
                alt = f"{base}_{int(time.time())}_{k}{ext}"
                os.replace(tmp, alt)
                return alt
```

```
            except PermissionError:
                time.sleep(0.25)
        # 多次失败：直接改名保存
        alt = f"{base}_{int(time.time())}{ext}"
        df.to_csv(alt, index=False, encoding='utf-8-sig')
        return alt

# === 重新跑 1 次带日志的仿真并保存 ===
rng = random.Random(SEED)
_, _, _, _, _, first_log = run_single_replication(N_DEVICES, rng, collect_log=True)

csv_log = os.path.join(OUT_DIR, "schedule_log_rep1.csv")
actual_log_path = safe_to_csv(first_log, csv_log)
print(f"[导出] 全量事件日志 CSV：{actual_log_path}")

# 同步导出“每天×工位”汇总
def export_daily_station_table(log_df: pd.DataFrame, csv_path: str):
    if log_df.empty:
        return
    df = log_df.copy()
    df = df[df['status'].isin(['pass','fail','interrupt','move'])]
    df['slot'] = df.apply(
        lambda r: f"D{int(r['device'])}-{r['stage']}({r['start']:.1f}-{r['end']:.1f})",
        axis=1
    )
    agg = df.groupby(['day','stage'])['slot'].apply(lambda s: " | ".join(s)).reset_index()
    table = agg.pivot(index='day', columns='stage', values='slot').fillna('')
    for col in ['IN','A','B','C','E','OUT']:
        if col not in table.columns:
            table[col] = ''
    table = table[['IN','A','B','C','E','OUT']]
    table.to_csv(csv_path, encoding='utf-8-sig')

csv_table = os.path.join(OUT_DIR, "daily_station_table_rep1.csv")
# 用 safe_to_csv 包一层，兼容被占用场景
tmp_table = os.path.join(OUT_DIR, "._tmp_daily_station_table_rep1.csv")
export_daily_station_table(first_log, tmp_table)
# 读回再安全落盘（也可以直接 os.replace；用同一路径更统一）
table_df = pd.read_csv(tmp_table) if os.path.exists(tmp_table) else pd.DataFrame()
if not table_df.empty:
```

```
        actual_table_path = safe_to_csv(table_df, csv_table)
        try:
            os.remove(tmp_table)
        except Exception:
            pass
        print(f"[导出] 每天×工位汇总 CSV：{actual_table_path}")
    else:
        print("[导出] 每天×工位汇总 CSV：暂无数据")
```

代码清单 2 问题三代码

```
    import argparse, random, statistics, pandas as pd
    from dataclasses import dataclass, field
    from typing import Dict, List, Tuple, Optional

    # =====参数 =====
    DUR: Dict[str, float]     = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    CALIB: Dict[str, float] = {'A': 0.5, 'B': 1/3, 'C': 1/3, 'E': 2/3}
    ERR: Dict[str, float]     = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}              # 测手差错（总量）
    PROB: Dict[str, float]    = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'E': 0.002491}       # 固有问题率（含 E）
    FAIL_P12 = {   # 故障两段 CDF（≤120h 累积, ≤240h 累积）
        'A': (0.03, 0.05),
        'B': (0.04, 0.07),
        'C': (0.02, 0.06),
        'E': (0.03, 0.05),
    }
    # ====（全局策略开关）====
    REPLACE_AT_120 = True             # 开启：累计使用 >=120h 时，测试前主动更换
    ALIGN_TO_SHIFT_START = True     # 开启：换+校尽量放在班次开始，避免卡班末

    # ===== 固定约束 =====
    MANDATORY_REPLACE_USAGE = 240.0
    HALL_CAP = 2
    T_IN = 0.5
    T_OUT = 0.5

    # ===== CSV 文件名=====
```

```
    DEFAULT_ORDER_CSV = "my_order.csv"

    # ===== 班次窗口 =====
    def next_window(t: float, K: float) -> Tuple[float, float]:
        d = int(t // 24.0)
        while True:
            base = d * 24.0
            w1, w2 = (base, base+K), (base+K, base+2*K)
            if w1[0] <= t < w1[1]: return w1
            if w2[0] <= t < w2[1]: return w2
            if t < w1[0]: return w1
            if t < w2[0]: return w2
            d += 1

    def fit_in_window(t_start: float, dur: float, K: float) -> Tuple[float, float]:
        w = next_window(t_start, K); s = max(t_start, w[0])
        if s + dur <= w[1]: return (s, s+dur)
        w2 = next_window(w[1], K); s2 = max(w2[0], w[1])
        if s2 + dur <= w2[1]: return (s2, s2+dur)
        raise RuntimeError

    def count_windows(t_end: float, K: float) -> int:
        cnt, t = 0, 0.0
        while True:
            w = next_window(t, K); cnt += 1
            if w[1] >= t_end: return cnt
            t = w[1] + 1e-9

    # ===== 故障抽样（两段 CDF）=====
    def sample_fail_usage(rng: random.Random, st: str) -> float:
        p1, p2 = FAIL_P12[st]
        u = rng.random()
        if u <= p1:    return 120.0 * (u / p1)                                  # 0~120h
        if u <= p2:    return 120.0 + 120.0 * ((u - p1) / (p2 - p1))       # 120~240h
        return float('inf')                                                      # 240h 前无故
障

    # ===== 结构体 =====
    @dataclass
    class Equipment:
        station: str
```

```
        usage: float = 0.0
        fail_at: float = float('inf')
        need_cal: bool = True
        ready_time: float = 0.0
        def reset(self, rng: random.Random, now: float):
            self.usage = 0.0
            self.fail_at = sample_fail_usage(rng, self.station)
            self.need_cal = True
            self.ready_time = now

    @dataclass
    class Station:
        name: str
        dur: float
        cal: float
        p_err: float
        equip: Equipment
        free: float = 0.0
        busy: float = 0.0

        def ensure_cal(self, rng: random.Random, K: float):
            if not self.equip.need_cal:
                return
            s, f = fit_in_window(max(self.free, self.equip.ready_time), self.cal, K)
            self.free = f
            self.busy += (f - s)
            self.equip.usage += (f - s)  # 校准计入设备使用
            self.equip.need_cal = False

        def maybe_replace_at_120(self, rng: random.Random, earliest: float, K: float):
            """在启动下个任务前，若累计使用≥120 且<240，则先更换+校准（可选：班首对齐）"""
            if not REPLACE_AT_120:
                return
            if 120.0 - 1e-9 <= self.equip.usage < MANDATORY_REPLACE_USAGE - 1e-9:
                when = max(self.free, earliest)
                # 可选：班首对齐，避免换到一半就下班
                if ALIGN_TO_SHIFT_START:
                    w = next_window(when, K)
```

```
                    # 若当前窗口剩余时间不足以完成“校准+一次测试”，就挪到下个窗口开始
                    remain = w[1] - max(when, w[0])
                    if remain < (self.cal + self.dur) - 1e-9:
                        # 下个窗口开始时间
                        when = next_window(w[1], K)[0]
                # 执行更换+校准
                self.equip.reset(rng, when)
                self.ensure_cal(rng, K)

    def run_attempt(self, rng: random.Random, earliest: float, K: float) -> tuple[float, bool]:
        """
        排一轮测试：可能会因为故障/到 240h 中断并触发更换+校准。
        增强点：启动任务前，若≥120h 则先主动更换（maybe_replace_at_120）。
        """
        # 0) 到 240h 的强制更换（任何情况都优先）
        if self.equip.usage >= MANDATORY_REPLACE_USAGE - 1e-9:
            self.equip.reset(rng, max(self.free, earliest))
            self.ensure_cal(rng, K)

        # 1) 若启用策略：在启动任务前检查“120h 主动更换”
        self.maybe_replace_at_120(rng, earliest, K)

        # 2) 找到不跨班的测试窗口
        s, f = fit_in_window(max(self.free, earliest), self.dur, K)

        # 3) 真实寿命判定：会不会在本次测试中途坏 / 到 240h
        left = min(self.equip.fail_at, MANDATORY_REPLACE_USAGE) - self.equip.usage
        if self.dur <= left + 1e-9:
            # 正常完成
            self.free = f
            self.busy += (f - s)
            self.equip.usage += self.dur
            return f, False
        else:
            # 中途故障/到 240 中断
            f_fail = s + max(0.0, left)
            self.free = f_fail
```

```
                    self.busy += (f_fail - s)
                    self.equip.usage += max(0.0, left)
                    #  立即更换并校准
                    self.equip.reset(rng, f_fail)
                    self.ensure_cal(rng, K)
                    return self.free, True

    @dataclass
    class Device:
        idx: int
        plan: List[str] = field(default_factory=lambda: ['A','B','C'])
        plan_i: int = 0
        stage: str = 'A'
        ready: float = 0.0
        defects: Dict[str, bool] = field(default_factory=dict)        # A/B/C/E 的真问题
        fails: Dict[str, int] = field(default_factory=lambda: {'A':0,'B':0,'C':0,'E':0})
        passed: bool = False
        exited: bool = False
        had_fp: bool = False
        bad_pass: bool = False
        def advance_after_abc_pass(self):
            if self.plan_i < 2:
                self.plan_i += 1
                self.stage = self.plan[self.plan_i]
            else:
                self.stage = 'E'

    # ===== 读取每台设备的 A/B/C 顺序表 =====
    def load_orders(csv_path: str, N: int) -> Dict[int, List[str]]:
        df = pd.read_csv(csv_path)
        df.columns = [c.strip().lower() for c in df.columns]
        assert 'device' in df.columns and 'order' in df.columns, "CSV 需含 device,order 列
"
        mp = {}
        for _, row in df.iterrows():
            i = int(row['device'])
            ord_str = str(row['order']).strip().upper()
            assert set(ord_str) == set('ABC') and len(ord_str) == 3, f"非法顺序: {ord_str}"
            mp[i] = list(ord_str)
        return {k: v for k, v in mp.items() if 0 <= k < N}
```

```
    # ===== 单次仿真（按你的判定树）=====
    def run_one(N_devices=100, K=10.0, seed=2025, order_map: Optional[Dict[int, List[str]]] = None) -> Dict:
        rng = random.Random(seed)
        stations = {s: Station(s, DUR[s], CALIB[s], ERR[s], Equipment(s)) for s in 'ABCE'}
        # t=0 预校准（计入使用时间），然后 free 归零
        for st in stations.values():
            st.equip.reset(rng, 0.0); st.ensure_cal(rng, K); st.free = 0.0

        devices = [Device(i) for i in range(N_devices)]
        order_map = order_map or {}
        for d in devices:
            if d.idx in order_map: d.plan = order_map[d.idx]
            d.stage = d.plan[0]
            d.defects = {
                'A': rng.random() < PROB['A'],
                'B': rng.random() < PROB['B'],
                'C': rng.random() < PROB['C'],
                'E': rng.random() < PROB['E'],    # 综合阶段的“固有问题”
            }

        active, next_idx, completed = 0, 0, 0
        for _ in range(min(HALL_CAP, N_devices)):  # 初始两台已到位
            devices[next_idx].ready = 0.0
            active += 1; next_idx += 1

        while completed < N_devices:
            st = min(stations.values(), key=lambda x: x.free)
            t = st.free
            cand = [d for d in devices if (not d.exited) and (not d.passed) and d.stage==st.name and d.ready<=t+1e-9]
            if not cand:
                waiting = [d.ready for d in devices if (not d.exited) and (not d.passed) and d.stage==st.name]
                st.free = max(st.free, min(waiting)) if waiting else (st.free + 0.01)
                continue

            dev = min(cand, key=lambda x:(x.ready, x.idx))
            f, interrupted = st.run_attempt(rng, max(dev.ready, st.free), K)
            if interrupted:
                dev.ready = f; continue
```

```
err = (rng.random() < st.p_err)

if st.name in 'ABC':
    has_bad = dev.defects[st.name]
    if has_bad:
        if err:
            # 漏判 → 通过（问题仍在）
            dev.advance_after_abc_pass(); dev.ready = f
        else:
            # 检出 → 失败（仅重测，不修复）
            dev.fails[st.name] += 1; dev.ready = f
            if dev.fails[st.name] >= 2:
                dev.exited = True; active -= 1; completed += 1
    else:
        if err:
            # 误判 → 失败
            dev.had_fp = True; dev.fails[st.name] += 1; dev.ready = f
            if dev.fails[st.name] >= 2:
                dev.exited = True; active -= 1; completed += 1
        else:
            # 正确通过
            dev.advance_after_abc_pass(); dev.ready = f
else:
    # E：若 A/B/C 残留 或 E 固有问题 → 有真问题
    has_bad = any(dev.defects[k] for k in 'ABC') or dev.defects['E']
    if has_bad:
        if err:
            # 漏判 → 带病通过
            dev.passed = True; dev.bad_pass = True; dev.ready = f
            completed += 1; active -= 1
        else:
            # 检出 → 失败（仅重测）
            dev.fails['E'] += 1; dev.ready = f
            if dev.fails['E'] >= 2:
                dev.exited = True; completed += 1; active -= 1
    else:
        if err:
            # 误判 → 失败
            dev.had_fp = True; dev.fails['E'] += 1; dev.ready = f
```

```
                            if dev.fails['E'] >= 2:
                                dev.exited = True; completed += 1; active -= 1
                        else:
                            dev.passed = True; dev.ready = f; completed += 1; active -= 1

                # 空位补入
                while active < HALL_CAP and next_idx < N_devices:
                    earliest = min(s.free for s in stations.values())
                    devices[next_idx].ready = earliest + T_IN
                    active += 1; next_idx += 1

            t_end = max(d.ready for d in devices)
            shifts = count_windows(t_end, K)
            num_pass = sum(d.passed for d in devices)
            num_bad_pass = sum(d.bad_pass for d in devices)
            num_fp_dev = sum(d.had_fp for d in devices)
            yxb = {s: stations[s].busy/(K*shifts) for s in stations}

            return dict(K=K, T_days=t_end/24.0, S=num_pass,
                        PL=(num_bad_pass/num_pass if num_pass>0 else 0.0),
                        PW=num_fp_dev/float(N_devices),
                        YXB1=yxb['A'], YXB2=yxb['B'], YXB3=yxb['C'], YXB4=yxb['E'])

    # ===== 扫描（固定 K=9..12，步长 0.5）=====
    def sweep_K(N_devices=100, rep=100, seed=2025, order_map: Optional[Dict[int, List[str]]] = None) -> pd.DataFrame:
        Ks = [9.0 + 0.5*i for i in range(int((12.0-9.0)/0.5)+1)]
        rows = []
        for K in Ks:
            reps = [run_one(N_devices, K, seed + r, order_map) for r in range(rep)]
            keys = reps[0].keys()
            rows.append({k: statistics.mean(r[k] for r in reps) for k in keys})
                return pd.DataFrame(rows).sort_values(['T_days','PL','PW']).reset_index(drop=True)

    # ===== CLI（只有全范围扫描，没有单点 K）=====
    def main():
        ap = argparse.ArgumentParser()
        ap.add_argument('--N', type=int, default=100, help='设备数量')
        ap.add_argument('--rep', type=int, default=80, help='每个 K 的重复次数')
        ap.add_argument('--seed', type=int, default=2025)
```

```
        ap.add_argument('--order_csv', type=str, default='', help='每台设备 A/B/C 顺序CSV（device,order）')
        ap.add_argument('--out', type=str, default='q3_results.csv')
        args, _ = ap.parse_known_args()  # 兼容 Jupyter/VSCode 隐式参数

        # 1) 优先使用命令行提供的 CSV；2) 否则使用代码里写死的 DEFAULT_ORDER_CSV；3) 都没有则默认 ABC
        csv_path = args.order_csv.strip() or DEFAULT_ORDER_CSV.strip()
        order_map = load_orders(csv_path, args.N) if csv_path else None

        df = sweep_K(args.N, args.rep, args.seed, order_map)
        print(df.to_string(index=False))
        df.to_csv(args.out, index=False)

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

代码清单 3 问题四代码

```
    import argparse, random, statistics, pandas as pd
    from dataclasses import dataclass, field
    from typing import Dict, List, Tuple, Optional

    # ===== 你提供的参数 =====
    DUR: Dict[str, float]   = {'A': 2.5, 'B': 2.0, 'C': 2.5, 'E': 3.0}
    CALIB: Dict[str, float] = {'A': 0.5, 'B': 1/3, 'C': 1/3, 'E': 2/3}
    ERR: Dict[str, float]   = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}          # 测手差错（总量）
    PROB: Dict[str, float]  = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'E': 0.002491}     # 固有问题率（含 E）
    FAIL_P12 = {  # 故障两段 CDF（≤120h 累积, ≤240h 累积）
        'A': (0.03, 0.05),
        'B': (0.04, 0.07),
        'C': (0.02, 0.06),
        'E': (0.03, 0.05),
    }

    # ===== 固定约束 =====
    MANDATORY_REPLACE_USAGE = 240.0
    HALL_CAP = 2
```

```
    T_IN = 0.5
    T_OUT = 0.5

    # ===== 方便你直接写死 CSV 文件名（可改）=====
    DEFAULT_ORDER_CSV = "my_order.csv"    # 例如改成 "my_order.csv"；留空则默认所有设备 ABC 顺序

    # ===== 班次窗口 =====
    def next_window(t: float, K: float) -> Tuple[float, float]:
        d = int(t // 24.0)
        while True:
            base = d * 24.0
            w1, w2 = (base, base+K), (base+K, base+2*K)
            if w1[0] <= t < w1[1]: return w1
            if w2[0] <= t < w2[1]: return w2
            if t < w1[0]: return w1
            if t < w2[0]: return w2
            d += 1

    def fit_in_window(t_start: float, dur: float, K: float) -> Tuple[float, float]:
        w = next_window(t_start, K); s = max(t_start, w[0])
        if s + dur <= w[1]: return (s, s+dur)
        w2 = next_window(w[1], K); s2 = max(w2[0], w[1])
        if s2 + dur <= w2[1]: return (s2, s2+dur)
        raise RuntimeError("任务无法排入班次窗口，请检查 K 与 dur")

    def count_windows(t_end: float, K: float) -> int:
        cnt, t = 0, 0.0
        while True:
            w = next_window(t, K); cnt += 1
            if w[1] >= t_end: return cnt
            t = w[1] + 1e-9

    # ===== 故障抽样（两段 CDF）=====
    def sample_fail_usage(rng: random.Random, st: str) -> float:
        p1, p2 = FAIL_P12[st]
        u = rng.random()
        if u <= p1:    return 120.0 * (u / p1)                          # 0~120h
        if u <= p2:    return 120.0 + 120.0 * ((u - p1) / (p2 - p1))    # 120~240h
        return float('inf')                                             # 240h 前无故障
```

```
    # ===== 结构体 =====
    @dataclass
    class Equipment:
        station: str
        usage: float = 0.0
        fail_at: float = float('inf')
        need_cal: bool = True
        ready_time: float = 0.0
        def reset(self, rng: random.Random, now: float):
            self.usage = 0.0
            self.fail_at = sample_fail_usage(rng, self.station)
            self.need_cal = True
            self.ready_time = now

    @dataclass
    class Station:
        name: str
        dur: float
        cal: float
        p_err: float
        equip: Equipment
        free: float = 0.0
        busy: float = 0.0

        def ensure_cal(self, rng: random.Random, K: float):
            if not self.equip.need_cal:
                return
            s, f = fit_in_window(max(self.free, self.equip.ready_time), self.cal, K)
            self.free = f
            self.busy += (f - s)
            self.equip.usage += (f - s)  # 校准计入设备使用
            self.equip.need_cal = False

        def maybe_replace_at_120(self, rng: random.Random, earliest: float, K: float):
            """在启动下个任务前，若累计使用≥120 且<240，则先更换+校准（可选：班首对齐）"""
            if not REPLACE_AT_120:
                return
            if 120.0 - 1e-9 <= self.equip.usage < MANDATORY_REPLACE_USAGE - 1e-9:
```

```
                when = max(self.free, earliest)
                # 可选：班首对齐，避免换到一半就下班
                if ALIGN_TO_SHIFT_START:
                    w = next_window(when, K)
                    # 若当前窗口剩余时间不足以完成“校准+一次测试”，就挪到下个窗口开始
                    remain = w[1] - max(when, w[0])
                    if remain < (self.cal + self.dur) - 1e-9:
                        # 下个窗口开始时间
                        when = next_window(w[1], K)[0]
                # 执行更换+校准
                self.equip.reset(rng, when)
                self.ensure_cal(rng, K)

    def run_attempt(self, rng: random.Random, earliest: float, K: float) -> tuple[float, bool]:
        """
        排一轮测试：可能会因为故障/到 240h 中断并触发更换+校准。
        增强点：启动任务前，若≥120h 则先主动更换（maybe_replace_at_120）。
        """
        # 0) 到 240h 的强制更换（任何情况都优先）
        if self.equip.usage >= MANDATORY_REPLACE_USAGE - 1e-9:
            self.equip.reset(rng, max(self.free, earliest))
            self.ensure_cal(rng, K)

        # 1) 若启用策略：在启动任务前检查“120h 主动更换”
        self.maybe_replace_at_120(rng, earliest, K)

        # 2) 找到不跨班的测试窗口
        s, f = fit_in_window(max(self.free, earliest), self.dur, K)

        # 3) 真实寿命判定：会不会在本次测试中途坏 / 到 240h
        left = min(self.equip.fail_at, MANDATORY_REPLACE_USAGE) - self.equip.usage
        if self.dur <= left + 1e-9:
            # 正常完成
            self.free = f
            self.busy += (f - s)
            self.equip.usage += self.dur
            return f, False
```

```
                else:
                    # 中途故障/到 240 中断
                    f_fail = s + max(0.0, left)
                    self.free = f_fail
                    self.busy += (f_fail - s)
                    self.equip.usage += max(0.0, left)
                    # 立即更换并校准
                    self.equip.reset(rng, f_fail)
                    self.ensure_cal(rng, K)
                    return self.free, True

    @dataclass
    class Device:
        idx: int
        plan: List[str] = field(default_factory=lambda: ['A','B','C'])
        plan_i: int = 0
        stage: str = 'A'
        ready: float = 0.0
        defects: Dict[str, bool] = field(default_factory=dict)        # A/B/C/E 的真问题
        fails: Dict[str, int] = field(default_factory=lambda: {'A':0,'B':0,'C':0,'E':0})
        passed: bool = False
        exited: bool = False
        had_fp: bool = False
        bad_pass: bool = False
        def advance_after_abc_pass(self):
            if self.plan_i < 2:
                self.plan_i += 1
                self.stage = self.plan[self.plan_i]
            else:
                self.stage = 'E'

    # ===== 读取每台设备的 A/B/C 顺序表 =====
    def load_orders(csv_path: str, N: int) -> Dict[int, List[str]]:
        df = pd.read_csv(csv_path)
        df.columns = [c.strip().lower() for c in df.columns]
        assert 'device' in df.columns and 'order' in df.columns, "CSV 需含 device,order 列
"
        mp = {}
        for _, row in df.iterrows():
            i = int(row['device'])
            ord_str = str(row['order']).strip().upper()
```

```
            assert set(ord_str) == set('ABC') and len(ord_str) == 3, f"非法顺序: {ord_str}"
            mp[i] = list(ord_str)
        return {k: v for k, v in mp.items() if 0 <= k < N}

    # ===== 单次仿真（按你的判定树）=====
    def run_one(N_devices=100, K=10.0, seed=2025, order_map: Optional[Dict[int, List[str]]] = None, disable_Y1=False, disable_Y2=False, disable_Y3=False) -> Dict:
        rng = random.Random(seed)
        stations = {s: Station(s, DUR[s], CALIB[s], ERR[s], Equipment(s)) for s in 'ABCE'}

        # 如果禁用 Y1、Y2、Y3，则修改相关设置
        if disable_Y1:
            FAIL_P12 = {s: (float('inf'), float('inf')) for s in 'ABCE'}  # 禁用故障
        if disable_Y2:
            PROB.update({'A': 0.0, 'B': 0.0, 'C': 0.0, 'E': 0.0})  # 禁用子系统问题
        if disable_Y3:
            ERR.update({'A': 0.0, 'B': 0.0, 'C': 0.0, 'E': 0.0})  # 禁用测手误差

        # 初始化设备和测试环境
        for st in stations.values():
            st.equip.reset(rng, 0.0); st.ensure_cal(rng, K); st.free = 0.0

        devices = [Device(i) for i in range(N_devices)]
        order_map = order_map or {}
        for d in devices:
            if d.idx in order_map: d.plan = order_map[d.idx]
            d.stage = d.plan[0]
            d.defects = {
                'A': rng.random() < PROB['A'],
                'B': rng.random() < PROB['B'],
                'C': rng.random() < PROB['C'],
                'E': rng.random() < PROB['E'],    # 综合阶段的“固有问题”
            }

        active, next_idx, completed = 0, 0, 0
        for _ in range(min(HALL_CAP, N_devices)):  # 初始两台已到位
            devices[next_idx].ready = 0.0
            active += 1; next_idx += 1

        while completed < N_devices:
```

```
            st = min(stations.values(), key=lambda x: x.free)
            t = st.free
            cand = [d for d in devices if (not d.exited) and (not d.passed) and d.stage ==
st.name and d.ready <= t + 1e-9]
            if not cand:
                waiting = [d.ready for d in devices if (not d.exited) and (not d.passed) and
d.stage == st.name]
                st.free = max(st.free, min(waiting)) if waiting else (st.free + 0.01)
                continue

            dev = min(cand, key=lambda x: (x.ready, x.idx))
            f, interrupted = st.run_attempt(rng, max(dev.ready, st.free), K)
            if interrupted:
                dev.ready = f; continue

            err = (rng.random() < st.p_err)

            if st.name in 'ABC':
                has_bad = dev.defects[st.name]
                if has_bad:
                    if err:
                        # 漏判 → 通过（问题仍在）
                        dev.advance_after_abc_pass(); dev.ready = f
                    else:
                        # 检出 → 失败（仅重测，不修复）
                        dev.fails[st.name] += 1; dev.ready = f
                        if dev.fails[st.name] >= 2:
                            dev.exited = True; active -= 1; completed += 1
                else:
                    if err:
                        # 误判 → 失败
                        dev.had_fp = True; dev.fails[st.name] += 1; dev.ready = f
                        if dev.fails[st.name] >= 2:
                            dev.exited = True; active -= 1; completed += 1
                    else:
                        # 正确通过
                        dev.advance_after_abc_pass(); dev.ready = f
            else:
                # E：若 A/B/C 残留 或 E 固有问题 → 有真问题
                has_bad = any(dev.defects[k] for k in 'ABC') or dev.defects['E']
                if has_bad:
```

```
                    if err:
                        # 漏判 → 带病通过
                        dev.passed = True; dev.bad_pass = True; dev.ready = f
                        completed += 1; active -= 1
                    else:
                        # 检出 → 失败（仅重测）
                        dev.fails['E'] += 1; dev.ready = f
                        if dev.fails['E'] >= 2:
                            dev.exited = True; completed += 1; active -= 1
                else:
                    if err:
                        # 误判 → 失败
                        dev.had_fp = True; dev.fails['E'] += 1; dev.ready = f
                        if dev.fails['E'] >= 2:
                            dev.exited = True; completed += 1; active -= 1
                    else:
                        dev.passed = True; dev.ready = f; completed += 1; active -= 1

            # 空位补入
            while active < HALL_CAP and next_idx < N_devices:
                earliest = min(s.free for s in stations.values())
                devices[next_idx].ready = earliest + T_IN
                active += 1; next_idx += 1

        t_end = max(d.ready for d in devices)
        shifts = count_windows(t_end, K)
        num_pass = sum(d.passed for d in devices)
        num_bad_pass = sum(d.bad_pass for d in devices)
        num_fp_dev = sum(d.had_fp for d in devices)
        yxb = {s: stations[s].busy/(K*shifts) for s in stations}

        return dict(K=K, T_days=t_end/24.0, S=num_pass,
                    PL=(num_bad_pass/num_pass if num_pass>0 else 0.0),
                    PW=num_fp_dev/float(N_devices),
                    YXB1=yxb['A'], YXB2=yxb['B'], YXB3=yxb['C'], YXB4=yxb['E'])

    # ===== 扫描（固定 K=9..12，步长 0.5）=====
    def sweep_K(N_devices=100, rep=100, seed=2025, order_map: Optional[Dict[int, List[str]]] = None, disable_Y1=False, disable_Y2=False, disable_Y3=False) -> pd.DataFrame:
        Ks = [9.0 + 0.5*i for i in range(int((12.0-9.0)/0.5)+1)]
```

```
        rows = []
        for K in Ks:
            reps = [run_one(N_devices, K, seed + r, order_map, disable_Y1, disable_Y2, disable_Y3) for r in range(rep)]
            keys = reps[0].keys()
            rows.append({k: statistics.mean(r[k] for r in reps) for k in keys})
                return pd.DataFrame(rows).sort_values(['T_days','PL','PW']).reset_index(drop=True)

    # ===== 灵敏性分析函数 =====
    def run_sensitivity_analysis(N_devices=100, rep=100, seed=2025, order_map: Optional[Dict[int, List[str]]] = None):
        # 禁用不同的因素，进行多次实验
        results = {
            "No Y1 (No Failure)": sweep_K(N_devices, rep, seed, order_map=order_map, disable_Y1=True),
            "No Y2 (No Device Defects)": sweep_K(N_devices, rep, seed, order_map=order_map, disable_Y2=True),
            "No Y3 (No Human Error)": sweep_K(N_devices, rep, seed, order_map=order_map, disable_Y3=True),
        }
        return results

    # 运行灵敏性分析
    analysis_results = run_sensitivity_analysis(N_devices=100, rep=100, seed=2025)

    # 输出结果
    for key, result in analysis_results.items():
        print(f"--- {key} ---")
        print(result)
```