基于多目标进化算法与离散事件仿真的
测试系统调度优化与敏感性分析
摘要

针对问题一：对综合测试 E 测出系统有问题的概率及其指向各子系统的比例 λ_i 进行分析。该问题的关键在于把 E 测试发现的问题区分为两类来源：单项测试 ABC 漏判留下的真实缺陷、联接系统 D 综合后产生的新增缺陷。在此基础上，考虑到 Y2、Y3、Y4 三类错误的影响，本文用条件概率把综合测试出问题并指向某子系统的事件拆开量化。本文首先计算每个子系统，在单测通过条件下，仍存在真实问题的条件概率，再将其与 D 的错误发生概率联合计算，得到 E 测试出问题的概率，最后通过归一化得到指向各源的比例系数。模型既考虑了测手的误判、漏判，也纳入了 D 的独立贡献，解析过程直接利用题中给定参数和条件概率实现。求解结果如下：综合测试测出系统有问题的概率 $P(E)=0.012183$，子系统占比 $\lambda_A=0.236471$、$\lambda_B=0.256012$、$\lambda_C=0.221502$、$\lambda_D=0.286016$。

针对问题二:对单个测试小队在给定班制（12h）下完成 100 台大型装置测试任务进行分析。该问题的关键在于，在任务的平均完成天数最小与总漏判概率最小这两项目标间，制定可执行的调度与设备更换策略。考虑到存在测试设备故障、重测与班次中断等多种随机事件，本文围绕任务完成平均天数 T、通过装置平均数 S、总漏判率 P_L、总误判率 P_W、各组有效工作时间比 YXB 等指标，将任务安排的优化问题转化为双目标 Pareto 优化问题。为能在随机环境下评估候选策略，基于离散事件仿真（DES）+蒙特卡洛抽样对测试流程与进行多次试验取样，从而估计各解的统计性能。在仿真模型设计上，将调度策略转化为给定条件下的动态规划求解，在完成时间维度采用贪婪算法。同时在设备管理上，把主动更换时长作为可调决策变量。

基于上述思想，建立了双目标 Pareto 优化模型，并使用多目标进化算法（NSGA-II 与 MOEA/D）进行求解。为保证解的稳健性与可解释性，对候选 Pareto 点应用 Bootstrap 重抽样估计置信区间与概率关联度一致性检验，剔除不稳定或统计上等价的解，得到稳健的 Pareto 前沿供决策者选择。求解结果如下：稳健的 Pareto 最优解的 $\mathrm{T}\approx 29.3-29.9$ 天，$\mathrm{S}\approx 92.5$ 台，$P_L\approx 0.0071\%$，$P_W=5.37\%$，YXB 分别为 0.725、0.585、0.718、0.787；

针对问题三:在问题二基础上增加第二测试分队并采用接续倒班，其关键在于选择合适的班次工作时长 K 以及测试设备的主动更换时点，使得在两队共享同一套测试装备的约束下，能最大化设备利用率，并在任务完成时间最短与总漏判率最低间取得平衡。本文建立了与问题二相同的性能指标，并将问题仍然表述为双目标 Pareto 优化问题，但决策变量扩展为设备主动更换时长与班次工作时长 K。优化方法上同样采用离散事件模拟（DES）、蒙特卡洛抽样和 MOEA 搜索扩展后的多目标决策空间。在问题的模型上增加了班次转换逻辑，并对模型进行扩展假设，任何任务不得在班次切换间进行。求解结果如下：最优班次工作时长 $K=12$ 小时（两班倒）；在该时长下的性能指标为 $\mathrm{T}\approx 15.27$ 天，$\mathrm{S}\approx 92.53$ 台，$P_L\approx 0.0073\%$，$P_W=5.3877\%$，YXB 分别为 0.723、0.584、0.717、0.786。

针对问题四：问题四，对各因素对测试任务平均完成时间的影响性分析，关键在于分析系统测试结果对因素的敏感度。考虑到班次工作时长 K、设备的主动更换时长、测

试大厅的测试台数以及测试设备套件数这四类变量的影响，本文建立了 Morris、SRC 和 PRCC 等指标，定量去比较各因素影响性。基于上述思想，设计 3 个影响性分析模型，包含 1 个粗粒度（4 种变量均包含）和 2 个细粒度（4 种测试设备的不同设备数量和时长）分析，并对问题 3 中的测试任务模型扩展测试台数量以及测试设备数量两个可变因素，充分考虑了多因素的影响。最后基于算法的定量结果，Morris 算法的μ_j^*和σ_j，SRC 以及 PRCC，对各因素影响性进行分析，以此来对主管部门提出对测试工作的改进建议。

关键词：离散事件仿真　遗传算法　Pareto 最优　敏感性分析

目录

1 问题综述

1.1 问题背景

大型装置通常由多个精密子系统构成，其在使用过程中对可靠性具有极高要求。为确保装置在实际应用中的稳定性和安全性，必须通过多次系统性测试来验证其性能。测试过程包括子系统级测试和系统级综合测试，涉及多个专业测试小组协作完成。测试过程中还需考虑装置运输、设备调试、异常情况处理等多重因素，整体流程复杂且充满不确定性。

任务规划问题在实际工程中常采用数学建模与优化方法进行处理。这类问题通常涉及资源分配、时序安排、风险控制等多重约束，目标往往是时间最短、成本最低或质量最优。常用方法包括排队论、随机过程建模、动态规划、仿真模拟等。尤其是在测试任务中，由于存在设备故障、测手差错、子系统问题等多类随机事件，需引入概率模型与随机优化技术，以在不确定性环境下制定鲁棒性较强的调度方案。面对测试过程中充满动态性和不确定性，需研究快速、高效的适应性任务规划技术，通过灵活配置资源支持任务方案临机调整，实现任务方案从基于预案向适应式规划的转变[1]。

1.2 问题提出

现需对某大型装置测试进行任务规划，该大型装置由 A、B、C 三个子系统组成，其测试流程满足以下 8 点约束：

- 测试大厅有两个测试台，可同时放置并测试 2 个大型装置。
- 一个测试分队包括 A、B、C 三个子系统测试小组和一个综合测试小组 E 组。
- 测试流程如图 1 所示，先对 A、B、C 子系统分别测试，全部通过后再进行综合测试。若任一测试未通过，需重测；连续两次未通过则装置退出测试。

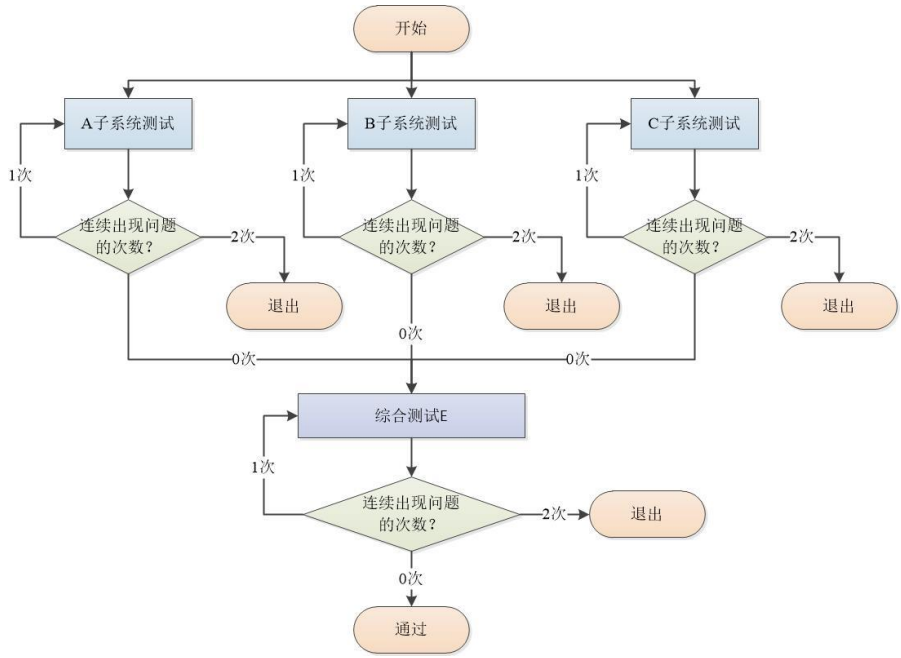

图 1 某大型装置测试流程图

- 装置进出测试大厅各需 0.5 小时，且运输过程不影响其他测试。
- 测试设备首次启用需调试，时间分别为：A 组 30 分钟、B 组 20 分钟、C 组 20 分钟、E 组 40 分钟。

- 正常测试时间分别为：A 组 2.5 小时、B 组 2 小时、C 组 2.5 小时、E 组 3 小时。
- 测试过程中可能出现异常情况：Y1 测试设备故障，故障概率随使用时间增加而上升；Y2 子系统有问题，需重测；Y3 测手操作失误，包括 Y31 误判（系统无问题但判为有问题）和 Y32 漏判（系统有问题但未测出）；Y4 综合测试测出问题，可能源于子系统漏判或整体联接问题。
- 测试任务的目标是：尽可能缩短任务完成时间，同时降低总漏判概率。

测试条件既有场地限制又有人员限制，测试流程有先后顺序要求，测试过程中还可能出现各种异常情况，基于上述约束，需解决以下 4 个问题：

(1) 问题 1 计算比例参数与概率：列出综合测试中各子系统问题比例参数 $\lambda_i(i=1,2,3,4)$ 的数学表达式和综合测试测出系统有问题的概率表达式，并代入给定数值进行计算。

(2) 问题 2 单分队测试计划：假设一个测试分队需测试 100 个装置，每班工作时间不超过 12 小时。制定测试计划并计算任务完成平均天数 T、通过测试的平均装置数 S、总漏判概率 P_L、总误判概率 P_W 指标、各测试小组的有效工作时间比 $YBX_i(i=1,2,3,4)$（即每班平均测试时间与 12 小时之比）等 8 个统计指标。

(3) 问题 3 双分队倒班测试计划：为加快进度，安排两个分队接续倒班，每班工作 $K(9\le K\le 12)$（以 0.5 小时为单位），使用同一套测试设备。确定最优 K 值，制定测试计划，并计算与问题 2 相同的统计指标。

(4) 问题 4 因素分析与改进建议：分析问题 3 中各项因素（如班次时长、设备故障、测手差错等）对平均完成时间的影响，并向主管部门提出测试工作的改进建议。

2 模型假设与符号说明

2.1 模型基本假设

(1) 假设系统有问题与测手发生差错两个事件相互独立。

(2) 假设装置没有两处及以上子系统问题（概率太小，可忽略）。

(3) 假设前一天没有完成测试的装置，保留工作进度，不将运出测试大厅。

(4) 假设 A、B、C 三个子系统的测试可同时进行。

(5)

【不要写“假设题目所给的数据没有任何错误”这些废话】

2.2 符号说明

本文定义了如下【数字】个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
$P(A)$	事件 A 发生的概率	/
$P(A\|B)$	事件 B 发生的条件下事件 A 发生的概率	/
$P(pass_i)$	通过测试 $i(i=A,B,C,E)$ 的概率	/
$P(defect_i)$	子系统 $i(i=A,B,C,D)$ 有问题的概率	/
$P(OK)$	整个系统无问题的概率	/
$P(E)$	综合测试 E 测出问题的概率	/

符号	含义	单位
$P(E_i)$	综合测试E测出问题指向子系统$i(i=A,B,C,D)$的概率	/
λ_i	综合测试测出有问题时问题指向子系统i所占比例	/
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_i	测试小组i的有效工作时间比	/
K	测试小队每班次工作时长	小时
θ_i	测试设备i主动更换时间	小时

3 问题 1 的分析与模型建立

3.1 问题 1 的分析

题目以大型装置测试任务规划为背景，介绍了测试条件、测试流程、异常情况等，问题 1 要求我们列出综合测试测出系统有问题的概率和综合测试测出有问题时各部分所占比例的表达式，并代入题目所给的数值计算。

根据测试流程，需通过 A、B、C 三个子系统测试后才能进入综合测试 E，所以要求的综合测试测出系统有问题的概率是条件概率，即 A、B、C 三个子系统测试通过条件下综合测试 E 测出系统有问题的概率。所以，需先分别计算某子系统测试通过条件下某子系统有问题的概率，再考虑综合测试 E 测出系统有问题指向某子系统的概率，每个综合测试 E 测出系统有问题指向某子系统概率之和是综合测试测出系统有问题的概率。

3.2 比例系数模型的建立

根据上述分析，首先计算条件概率，**子系统$i(i=A,B,C)$测试通过条件下子系统i有问题的概率**$P(defect_i \mid pass_i)$，根据事件独立性和贝叶斯公式，得到：

$$P(defect_i \mid pass_i)=\frac{P(pass_i \mid defect_i)\,P(defect_i)}{P(pass_i)} \tag{1}$$

根据题目所给的两个事件，Y_2 子系统有问题和 Y_3 测手操作失误，可以得到概率。子系统i有问题的条件下测试通过（漏判）的概率 $P(pass_i \mid defect_i)=P(Y_{3,i})P(Y_{32,i})$，子系统$i$有问题的概率 $P(defect_i)=P(Y_{2,i})$。而 $P(pass_i)$ 表示子系统i测试通过的概率。子系统i测试通过分为两种情况，一是子系统i有问题但测手漏判，二是子系统i无问题且测手没误判，二者相加得到子系统i测试通过的概率。

$$P(pass_i)=P(Y_{2,i})P(Y_{3,i})P(Y_{32,i})+\left[1-P(Y_{2,i})\right]\left[1-P(Y_{3,i})P(Y_{31,i})\right] \tag{2}$$

然后，考虑**综合测试E测出系统有问题指向子系统$i(i=A,B,C,D)$的概率**$P(E_i)$，需分别考虑子系统 A,B,C 和 D，因为子系统 A,B,C 需通过之前的子系统测试才能进入综合测试 E，这里子系统 A,B,C 有问题的概率是通过子系统测试条件下的条件概率。综合测试 E 测出系统有问题指向子系统 i 也分为两种情况，一是子系统 i 有问题且测手没漏判，

二是系统无问题但测手误判（假设指向每种问题的概率相等，即 0.25），二者相加得到综合测试 E 测出系统有问题指向子系统 i 的概率。

$$P(E_i)=\begin{cases}P(defect_i\,|\,pass_i)\left[1-P(Y_{3,E})P(Y_{32,E})\right]+P(OK)P(Y_{3,E})P(Y_{31,E})\times 0.25\ , & i=A,B,C\\ P(defect_D)\left[1-P(Y_{3,E})P(Y_{32,E})\right]+P(OK)P(Y_{3,E})P(Y_{31,E})\times 0.25\ , & i=D\end{cases} \tag{3}$$

其中 $P(OK)$ 表示综合测试 E 中整个系统无问题的概率。

$$P(OK)=\prod_{i=A,B,C}\left[1-P(defect_i\,|\,pass_i)\right]\times\left[1-P(defect_D)\right] \tag{4}$$

每个综合测试 E 测出系统有问题指向 $i(i=A,B,C,D)$ 子系统概率之和是**综合测试测出系统有问题的概率** $P(E)$，**综合测试测出有问题时问题指向子系统 i 所占比例** λ_i 是综合测试 E 测出系统有问题指向 i 子系统的概率与综合测试测出系统有问题的概率之比。

$$P(E)=\sum_{i=A,B,C}P(E_i) \tag{5}$$

$$\lambda_i=\frac{P(E_i)}{P(E)} \tag{6}$$

3.3 比例系数模型求解

比例系数模型较为简单，仅涉及一些概率公式的计算，算法流程如下：

Step 1 计算子系统 $i(i=A,B,C)$ 测试通过条件下子系统 i 有问题的概率 $P(defect_i\,|\,pass_i)$；

Step 2 计算综合测试 E 测出系统有问题指向子系统 $i(i=A,B,C,D)$ 的概率 $P(E_i)$；

Step 3 计算综合测试测出系统有问题的概率 $P(E)$ 和综合测试测出有问题时 i 部分所占比例 λ_i。

计算所需的参数值题目中已提供，程序求得综合测试测出系统有问题的概率 $P(E)=0.012183$，综合测试测出有问题时各部分所占比例系数结果记录在表 2 中。

表2 综合测试测出有问题时各部分所占比例系数

子系统 A 的占比 λ_A	子系统 B 的占比 λ_B	子系统 C 的占比 λ_C	子系统 D 的占比 λ_D
0.236471	0.256012	0.221502	0.286016

4 问题 2 的分析与模型建立

4.1 问题 2 的分析

问题 2 给定一个测试小队完成测试 100 个大型装置的测试任务，根据两个测试目标（任务尽快完成，总的漏判概率尽量低）制定测试工作计划并计算 8 个统计指标。

显然，这是一个**双目标优化**问题。目标为任务完成平均天数尽量少，总漏判率尽量低。约束条件为单项测试时间固定、运输和设备校对时间固定、班次工作时间固定、测试流程和重测规则固定等。

在测试任务过程中，会发生测试设备故障、子系统有问题等随机事件，需要花费时间更换设备或重测，影响任务进度。这些随机事件发生的概率在题目和问题 1 中已知，采用**蒙特卡洛方法**可将随机事件引入测试过程中。

随机事件的发生具有动态性和随机性，面对这些事件的发生，需动态规划工作计划，以实现目标优化。除了题目中固定的测试规则，可**动态决策**的内容包括**测试小队调度策略**和**测试设备更换策略**。对于测试小队的调度，采用贪婪策略，充分利用小队，减少测试时间。

测试、故障、重测等事件属于在时间和空间中发生的离散事件，而**离散事件模拟**是一种通过模拟时间和空间中发生的离散事件来研究复杂系统的仿真技术，适用于具有动态特性和随机性的系统。

所以本文基于双目标优化模型、蒙特卡洛方法和三种动态决策策略，采用离散事件模拟实现对测试流程、随机事件发生、动态规划工作计划的模拟。

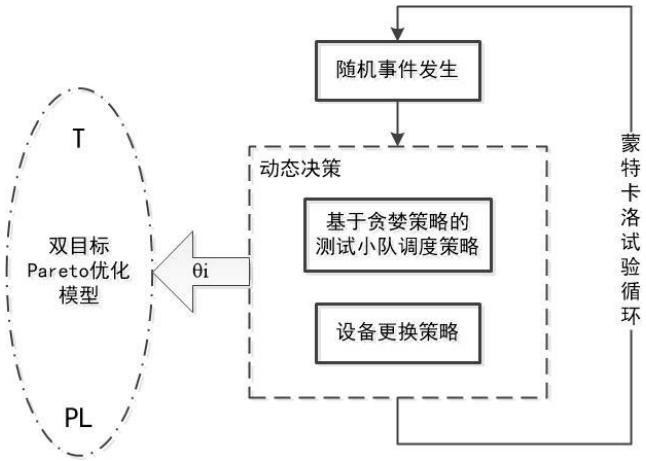

图 2 求解问题 2 的基本框架

4.2 问题 2 的指标定义

问题 2 需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 和各个专业测试组的有效工作时间比 $YXB_i(i=A,B,C,E)$ 等 8 项指标，现进行定义。

测试过程中会发生测试设备故障、子系统有问题等随机事件，采用蒙特卡洛方法进行多次试验，得到上述参数的平均值。

(1) 任务完成平均天数 T

$$T=\sum_{j}^{n_{mente}}\left(makespan_j-\max_{i\in\{A,B,C,E\}}t_{calib,i}-2\times t_{trans}\right)\Big/\left(K\times n_{mente}\right) \tag{7}$$

其中， n_{mente} 是蒙特卡洛方法的试验次数， $makespan_j$ 是在第 j 次试验中测试所需的总时间， $t_{calib,i}$ 是测试设备 i 调试校准时间， t_{trans} 是装置运入/运出测试大厅的时间， K 是班次一天工作时间。

(2) 通过测试的装置的平均数目 S

$$S=\sum_{j}^{n_{mente}}n_{pass,j}\Big/n_{mente} \tag{8}$$

其中， $n_{pass,j}$ 是在第 j 次试验中通过测试的装置数目。

(3) 总漏判概率 P_L

$$P_L = \sum_{j}^{n_{mente}} n_{neg,j} \Big/ \left(n_{device} \times n_{mente}\right) \tag{9}$$

其中，$n_{neg,j}$ 是在第 j 次试验中漏判的装置数目，n_{device} 是测试的装置综述。

(4) 总误判概率 P_W

$$P_W = \sum_{j}^{n_{mente}} n_{pos,j} \Big/ \left(n_{device} \times n_{mente}\right) \tag{10}$$

其中，$n_{pos,j}$ 是在第 j 次试验中误判的装置数目。

(5) 各个专业测试组的有效工作时间比 YXB_i

$$YXB_i = \sum_{j}^{n_{mente}} t_{test,j,i} \Big/ \left(K \times n_{mente}\right) \tag{11}$$

其中，$t_{test,j,i}$ 是在第 j 次试验中测试小组 i 测试的总时间。

4.3 问题 2：双目标 Pareto 优化模型的建立

根据问题 2 的分析，这是一个双目标优化问题，对于双目标优化的解通常是一组均衡解，兼顾两个目标函数的优化，Pareto 解集能够有效的解决该问题。

4.3.1 确定目标函数

根据题目要求，测试任务安排主要考虑的目标是：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 尽量少，总漏判概率 P_L 尽量小，确定目标函数为：

$$\min T \quad , \quad \min P_L \tag{12}$$

4.3.2 确定约束条件

根据题目描述，写出双目标优化模型的约束条件。

(1) 测试顺序约束：装置 $j(j \in [1,100])$ 必须完成 A、B、C 三个子系统测试后才能进入综合测试 E，即综合测试 E 开始时刻大于子系统测试最大结束时刻。

$$\max_{i \in \{A,B,C\}} end_{i,j} \le start_{E,j} \tag{13}$$

其中，end_i 表示测试项目 i 结束时刻，$start_E$ 表示综合测试 E 结束时刻。

(2) 测试设备约束：由于题目只提供一套测试设备，一个测试小组不能同时测试多个装置，即测试设备 $i(i \in \{A,B,C,E\})$ 必须测试完第 j 个装置才能测试下一个第 $k(k > j)$ 个装置。

$$end_{i,j} \le start_{i,k} \tag{14}$$

(3) 运输时间约束：装置 j 经过综合测试 E 后，被运出测试大厅时，运入下一个装置 $k(k \ge j+2)$，即装置 k 开始测试时刻小于等于装置 j 经过综合测试 E 时刻加上运输时间。

$$end_{E,j} + t_{trans} \le \min_{i \in \{A,B,C\}} start_{i,k} \tag{15}$$

(4) 测试时间计算：装置 j 完成测试项目 i 的时刻等于开始时刻与测试时长（包括重测）之和。

$$end_{i,j} = start_{i,j} + n_{test-i,j} \times t_{test,i} \tag{16}$$

其中，$n_{test-i,j}$ 是装置 j 在测试项目 i 中测试次数，$t_{test,i}$ 是测试项目 i 的测试时长。

(5) 其他常数和范围约束：例如，测试小队一天工作时长 $K=12$ 小时，运输时间、测试设备校准时间、项目测试时间等题目中已给出，不再一一列举。需要测试的装置总数为 100，测试装置编号 $j \in [1,100]$；装置 j 在测试项目 i 中测试次数 $n_{test-i,j} \in \{1,2\}$；测试设备至少需要工作 120 个小时才能更换，工作 240 个小时必须更换，所以测试设备 i 更换时间 $\theta_i \in [120,240]$。

结合目标函数和约束条件，最终建立的双目标优化模型如式(17)所示。

$$\begin{aligned} &\min T \quad , \quad \min P_L \\ &s.t \begin{cases} \max\limits_{i\in\{A,B,C\}} end_{i,j} \le start_{E,j} \\ end_{i,j} \le start_{i,k} \\ end_{E,j} + t_{trans} \le \min\limits_{i\in\{A,B,C\}} start_{i,k} \\ end_{i,j} = start_{i,j} + n_{test-i,j} \times t_{test,i} \\ K = 12, t_{trans} = 0.5 \\ t_{test,i} = \{2.5, 2, 2.5, 3\}, \ t_{calib,i} = \{0.5, 1/3, 1/3, 2/3\} \\ n_{test-i,j} \in \{1,2\}, \ n_{team} = 1 \\ \theta_i \in [120,240], \ j \in [1,100] \end{cases} \end{aligned} \tag{17}$$

4.3.3 构建 Pareto 解集

根据问题 2 的分析，可动态决策的内容包括测试小队调度策略和测试设备更换策略。测试小队调度采用贪婪策略，那么可优化的因素只有设备更换策略。假设 θ_i 小时后更换测试设备 i，不同的 θ 值，对应不同的解 $(\hat{T}(\pi), \hat{P}_L(\pi))$，所有解的集合称为解空间。$\hat{T}(\pi)$ 和 $\hat{P}_L(\pi)$ 分别表示 T 和 P_L 的样本均值，n_0 为样本数量。

$$\begin{cases} \hat{T}(\pi) = \dfrac{1}{n_0} \sum\limits_{i=0}^{n_0} T_i(\pi) \\ \hat{P}_L(\pi) = \dfrac{1}{n_0} \sum\limits_{i=0}^{n_0} P_{L,i}(\pi) \end{cases} \tag{18}$$

不同解之间存在优劣差别，通过以下方法选出 Pareto 最优解。

(1) 非支配排序计算 Patero 前沿

设 x, y 是双目标优化问题中的两个可行解，当且仅当

$$\forall i \in \{T, P_L\}, f_i(x) \le f(y)$$
$$\exists j \in \{T, P_L\}, f_j(x) < f(y)$$

则称 x 支配 y，记为 $x \prec y$。

删除被其它解支配的解集，剩下的解是互不支配的解，这些解的集合称为 Pareto 最优解集，表示为

$$P = \{x \in S \,|\, \forall y \in S, x \prec y\}$$

其中，S 是所有候选解的解集。

由 Pareto 最优解集 P 中所有 Pareto 最优解对应的目标函数向量组成的曲面/曲线称为 Pareto 前沿 P_f 。

(2) Bootstrap 估计置信区间

为了检验 Patero 前沿解的稳定性，对每个前沿点进行 Bootstrap 重抽样，计算 T 和 P_L 的 95%置信区间，留下稳定的 Pareto 解。

Bootstrap 的核心思想是通过有放回地重复抽样来模拟从总体中多次抽样的过程，从而估计统计量（如均值、方差、中位数等）的抽样分布。

Bootstrap 方法步骤分为以下几步：

1) 重抽样 (Resampling)：从原始的样本数据（容量为 n ）中，有放回地随机抽取 n 个观测值，生成新 Bootstrap 样本。

2) 计算统计量 (Calculate Statistic)：针对新生成的 Bootstrap 样本，计算样本均值 α 。

3) 重复 (Repeat)：将步骤 1 和步骤 2 重复执行非常多次（通常为 1000 次、10000 次甚至更多）。每次都会产生一个新的 Bootstrap 样本和一个对应的统计量值。

4) 构建抽样分布 (Form the Distribution)：将所有重复计算出的统计量值（ $\alpha_1,\alpha_2,\cdots$ ）收集起来，形成了一个经验分布，即该统计量的 Bootstrap 抽样分布。这个分布近似于该统计量在真实总体中的抽样分布。

5) 进行推断 (Make Inference)：利用 Bootstrap 分布，计算置信区间 (Confidence Interval)，使用百分位数法，取 Bootstrap 分布的 2.5%分位数和 97.5%分位数，即可构成一个 95%的置信区间。

(3) 概率关联度一致性检验

删除不稳定的 Pareto 解后，解集中还可能存在 CI 区间高度重叠的解，它们在统计上没有差异。用统计检验的方法判断两个解的差异是否显著，差异不显著的两个解称为“等价”，删除其中一个等价解，确保 Pareto 解不是“假优势”。

概率关联度一致性检验方法布置分为以下几步[2]：

1) 在相同初始条件下，分别得到参考序列 x 和比较序列 y 。

2) 对参考序列 x 和比较序列 y 进行预处理，使其满足等步长、等长度的数据序列要求。

3) 计算 k 时刻概率关联系数，根据重抽样样本，估计 y 的经验分布函数，将参考样本 x 带入经验分布函数，计算累积分布函数值，得到关联系数。

4) 检验关联系数在一定置信水平下是否服从[0,1]上的均匀分布。若通过检验，说明通过关联分析；否则，未通过概率关联分析。

(4) 形成稳健 Pareto 解集

根据上述方法，最终得到均值优良、CI 稳定、差异显著的 Pareto 解，即“最可信的 Pareto 前沿”。

4.4 离散事件模拟

根据问题 2 的分析，基于双目标优化模型、蒙特卡洛方法和三种动态决策策略，采用离散事件模拟实现对测试流程、随机事件发生、动态规划工作计划的模拟。

4.4.1 离散事件模拟和蒙特卡洛方法简介

4.4.1.1 离散事件模拟简介

离散事件模拟（Discrete Event Simulation, DES）是一种通过模拟时间和空间中发生的离散事件来研究复杂系统的仿真技术。其核心在于对系统中发生的事件进行详细的建模和分析，尤其适用于具有动态特性和随机性的系统[3]。它广泛应用于各种领域，如运营管理、制造业、交通规划、医疗系统等，以帮助预测和优化复杂系统的行为和性能。

离散事件模拟的核心思想是模拟系统在离散的时间点上发生的事件，这些事件会引起系统状态的变化。模拟通过在时间上推进，模拟系统在不同事件发生时的状态变化，从而得出系统的性能指标、行为和结果。

离散事件模拟的基本步骤是问题定义和模型建立、初始化、事件生成、事件排序、事件处理、数据收集和统计、模拟终止条件。

4.4.1.2 蒙特卡洛方法简介

蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法，以概率和统计理论方法为基础的一种计算方法。将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解。

蒙特卡罗方法可以粗略地分成两类，一类是所求解的问题本身具有内在的随机性，借助计算机的运算能力可以直接模拟这种随机的过程，如中子在反应堆中的传输过程；另一类是所求解问题可以转化为某种随机分布的特征数，比如随机事件出现的概率或者随机变量的期望值。本文使用蒙特卡洛方法模拟测试过程中测试设备故障、子系统有问题等有概率值的随机事件，属于第二类蒙特卡洛方法。

4.4.2 离散事件模拟流程

根据离散事件模拟算法的基本步骤和题目要求，结合蒙特卡洛方法模拟随机事件的发生，算法流程如下：

Step 1 每当有装置运出测试大厅时，触发“开始”（初始条件：测试大厅中已有两个装置，测试设备都完成了调试校对）；

Step 2 判断测试大厅中装置数量是否小于 2，是进入 **Step 3**，否进入 **Step 4**；

Step 3 运入 $(2-n)$ 个装置到测试大厅，其中 $n(n=0,1,2)$ 表示当前测试大厅中装置的数量，回到 **Step 1**；

Step 4 事件生成：以测试台为单位，同时发起 A、B、C 三个测试任务；

Step 5 判断测试设备 $i(i=A,B,C,E)$ 是否被占用，是进入 **Step 6**，否进入 **Step 7**；

Step 6 事件排序：测试任务 i 进入抢占池 X 等待；

Step 7 任务入栈：测试任务 i 进入任务池 Y；

Step 8 任务提取：根据测试任务 i 的完成时刻，提前提取任务状态；

Step 9 判断测试设备 i 是否发生故障，是进入 **Step 10**，否进入 **Step 11**；

Step 10 更换测试设备 i 并调试校准，重新计算任务时刻，任务重新入栈，回到 **Step 7**；

Step 11 判断任务时刻是否超过班次工作时间，是进入入 **Step 12**，否进入 **Step 13**；

Step 12 更新任务时刻到第二天工作开始时间，任务重新入栈，回到 **Step 7**；

Step 13 任务出栈：任务 i 完成后弹出任务池 Y，记录任务序号、任务完成时刻、测试结果等信息

Step 14 判断任务 i 测试出有问题的次数，0 次进入 **Step 15**，1 次回到 **Step 9** 重测，2 次“退出，运出测试大厅”；

Step 15 释放任务池任务 i 的空间，进入 **Step 16** 和 **Step 17**；

Step 16 遍历抢占池 X，判断抢占池 X 中是否有任务 i 在等待，是回到 **Step 9** 测试；

Step 17 判断当前测试是否是 E 测试，是装置通过测试“结束，运出测试大厅”，否进入 **Step 18**；

Step 18 判断装置 A、B、C 测试是否均通过，是发起测试任务 E，回到 **Step 7**。

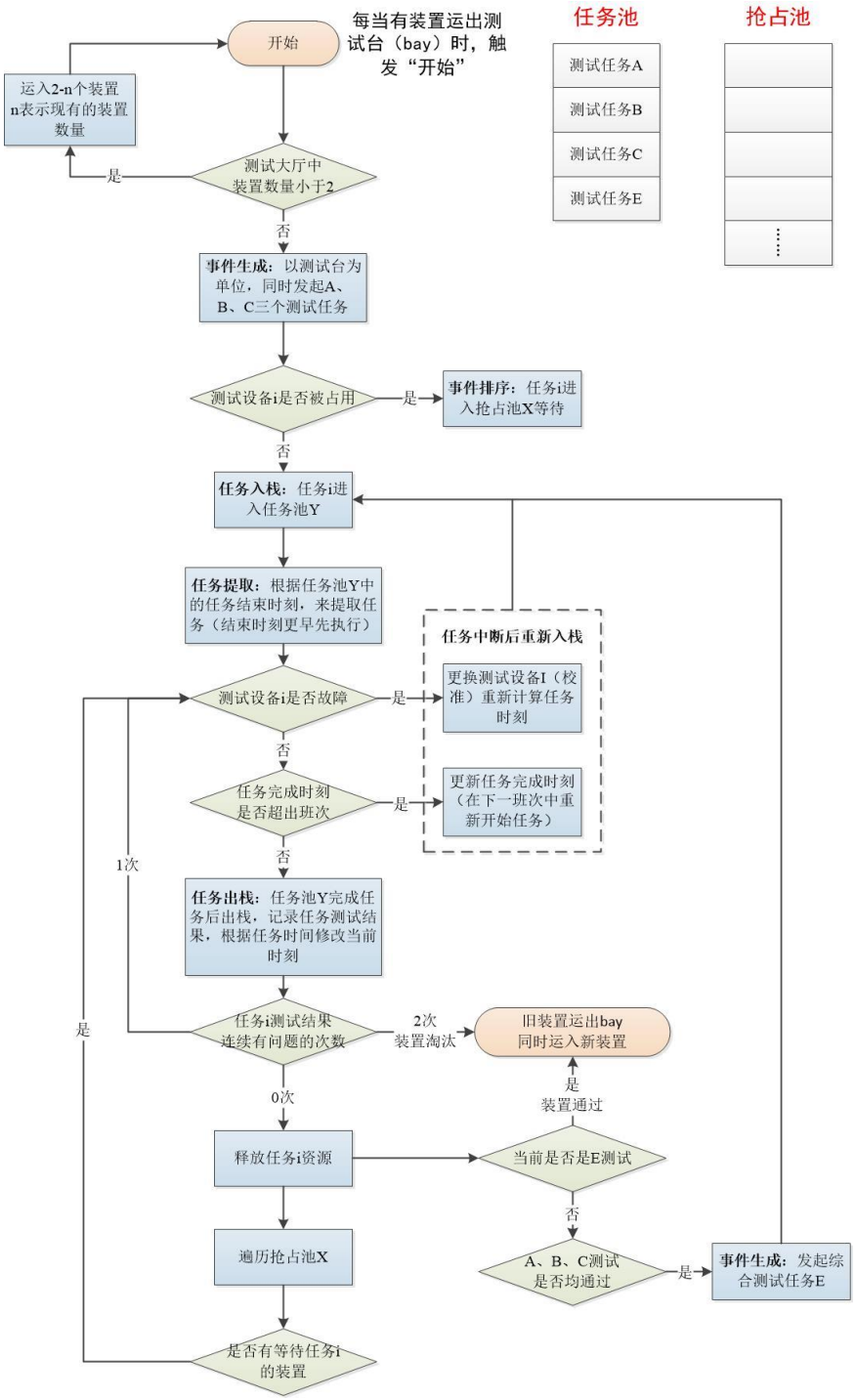

图 3 离散事件模拟流程图

4.5 问题 2：双目标 Pareto 优化模型的求解

结合蒙特卡洛方法的离散事件模拟成功模拟了测试流程和随机事件的发生、处理，但缺少优化。结合 4.3 节建立双目标 Pareto 优化模型，优化测试设备更换时间，假设测

试设备更换时间为$\theta_i(\theta_i \in [120,240])$，得到最优 Pareto 解。为了寻找最优解对应的θ_i值，本文尝试采用三种算法，一是简单的**遍历算法**，二是 **NSGA-II 算法**，三是 **MOEA/D 算法**，效率优于 NSGA-II 算法。初步采用的简单遍历算法和 NSGA-II 算法，由于求解速度太慢，被舍去。

4.5.1 MOEA/D 算法求解

MOEA/D 是一种基于分解策略的多目标优化算法，它将多个目标分解成单个目标，引入参考方向和邻居，可以高效的生成均匀的 Pareto 前沿，实验表明它的效率要好于 NSGA-II[4]。

- 参考方向：MOEA/D 算法需要提供一系列的参考方向，参考方向是一组预先定义的具有代表性的目标方向向量，一般来讲参考方向的数量等于种群的数量。
- 邻居：MOEA/D 能够比 NSGA-II 效率更高的一大优势就是引入了邻居，邻居是指与某个个体在目标空间中距离最近的一些个体。每一个目标方向向量会计算自身与其他目标方向的欧几里得距离按升序排序，取前 N 个作为自己的邻居。在 MOEA/D 算法中，每个个体只会跟自己的邻居进行信息交换和优化，而不会与整个种群进行全局交互。这种局部信息交换可以提高计算效率，同时也有助于维持种群的多样性。

MOEA/D 算法流程图如图 4 所示。

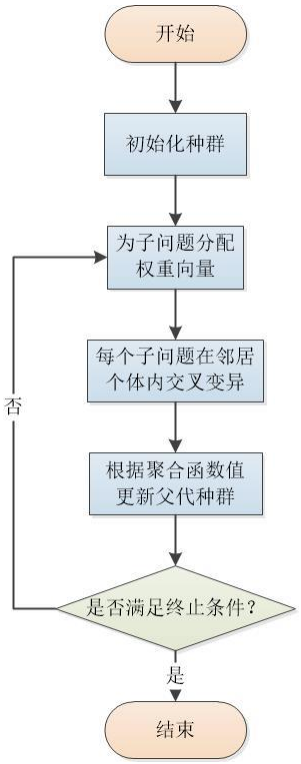

图 4 MOEA/D 算法流程图

Step 1 初始化种群：设置初始 Patero 解集$EP=\varnothing$，计算任意两个权向量之间的欧几里得距离，取前T个作为自己的邻居，计算初始化种群每个个体的目标函数FV，初始化一个参考点z^*。

Step 2 为子问题分配权重向量。

Step 3 随机选择两个邻居，产生新的个体，应用某种策略去判断或者优化新产生的个体。

Step 4 更新参考点和邻居。如果这个新产生的个体的某一个目标函数要优于参考点对应的目标函数就进行替换。

Step 5 达到最大迭代次数或者触发停止策略则输出 Pareto 解集，否则执行步骤 2。

4.5.2 双目标 Pareto 优化模型的求解结果

设定 MOEA/D 算法的迭代次数为 60，种群大小为 7（一代包含 7 个候选解），重复运行 10 次，得到 Pareto 前沿解，如图 5 所示。

图中展示的是参数空间与优化目标的响应关系。纵坐标轴表示关键决策参数，在本设计中一共设置了 4 个关键决策参数分别是 A、B、C、E 测试设备的主动更换时长。横坐标表示对应的目标函数值（任务完成时间T和漏检率P_L）。颜色深度则代表了搜索最佳前沿解所消耗的迭代次数，颜色越明亮代表迭代次数越多。

从图中可以观察到：

- 在图中曲面整体趋势中，包含有许多局部的波峰和波谷。在波峰和波谷的情况下，整体的目标函数值，即任务完成时间T和漏检率P_L是变化较少，但 Z 轴方向的测试

设备主动更换时长变化较大。说明系统对所选出的测试设备主动更换时长这一变量，具有一定的鲁棒性。

- 图中在 XY 平面的最左上角，即为目标函数的最优区域（此区域任务完成时间 T 和漏检率 P_L 最低）。在本实验中，Pareto 前沿最优解（红点区域）在 XY 平面的映射区域，也主要集中在最左上角，证明了迭代搜索最优解策略的可行性。
- 图中曲面的颜色深度反应了搜索策略对解搜索的迭代次数消耗。可以发现在波谷位置通常迭代次数较低。而波峰位置迭代次数较高。这证明了在空间搜索时使用的多目标进化算法 MOEA\D 可以有效降低陷入局部解的概率，在后续的迭代过程中跳脱局部解，从而找到在整个解空间中的 Pareto 前沿最优解。

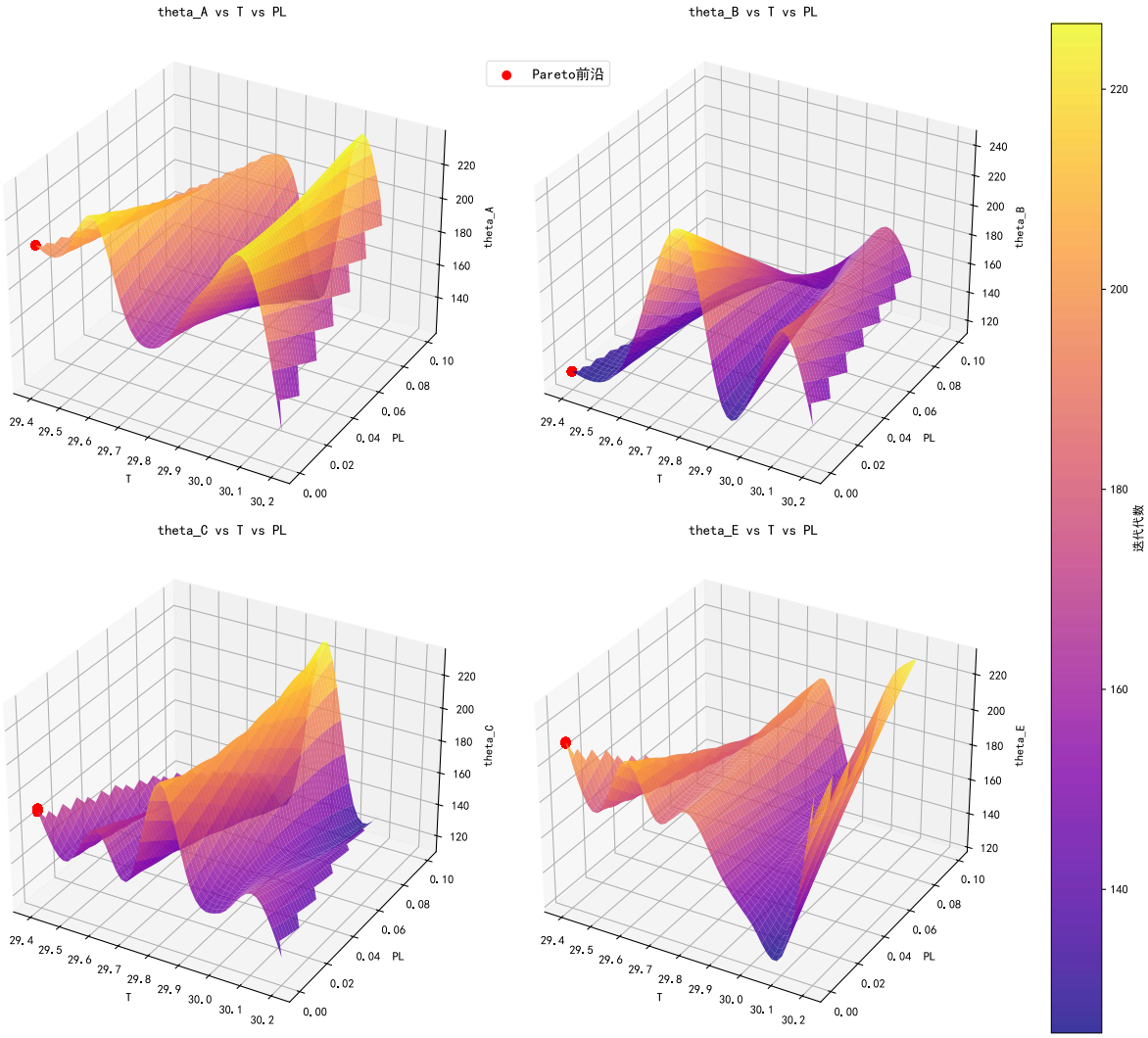

图 5 MOEA\D 算法解得的 Patero 前沿解

Pareto 前沿解再通过 Bootstrap 重抽样和概率关联度一致性检验，得到稳健的 Pareto 解集，如表 3 所示。

权衡任务完成时间和总漏判率，按照 $\min 0.5\times\frac{T}{\tilde{T}}+0.5\times\frac{P_L}{\tilde{P}_L}$ 原则，选中表 3 中蓝色行为最优解。工作计划：203.7 小时后更换测试设备 A，123.5 小时后更换测试设备 B，169.7 小时后更换测试设备 C，211.3 小时后更换测试设备 E，任务平均完成时间为 29.32 天，总漏判率为 0。

表3 MOEA/D 算法得到的稳健 Pareto 解集

测试设备 A 更换时间 θ_A	测试设备 B 更换时间 θ_B	测试设备 C 更换时间 θ_C	测试设备 E 更换时间 θ_E	任务平均完成时间 $T[95\%CI]$	总漏判率 $P_L[95\%CI]$
203.7	123.5	169.7	211.3	29.32 [28.95,29.75]	0.000150 [0.00000,0.00030]
203.7	123.7	169.8	210.7	29.45 [29.05,29.82]	0.000310 [0.00000,0.00050]
203.7	123.5	171.5	211.3	29.41 [29.00,29.80]	0.000280 [0.00000,0.00050]
203.7	123.5	170	211.3	29.43 [29.05,29.81]	0.000260 [0.00000,0.00045]
203.7	123.5	170	211.5	29.40 [29.02,29.78]	0.000240 [0.00000,0.00042]

按照标蓝的工作计划，重复多次蒙特卡洛试验，得到 8 个指标的结果如表 4 所示。

表4 MOEA/D 算法得到 8 个统计指标

T /天	S /个	P_L /%	P_W /%	YXB_1	YXB_2	YXB_3	YXB_4
29.9	92.49	0.0071	5.3732	0.725	0.585	0.718	0.787

5 问题 3 的分析与模型建立

5.1 问题 3 的分析

问题 3 安排两个分队接续倒班，每个班次工作 $K(9 \le K \le 12)$ 小时，同工序两个班次使用同一套测试装备。确定最优的 K 值（以半小时为最小单位），并制定测试工作计划并计算与问题 2 相同的 8 个指标。

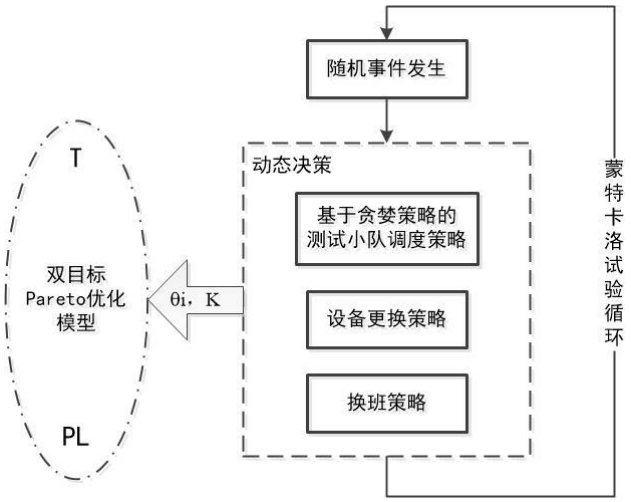

图 6 求解问题 3 的基本框架

问题 3 在问题 2 的基础上，增加另一个测试分队，并将工作时长设为 K。为了最大化利用仅有的一套测试设备，采用两班倒工作制，一天可工作 $2K$ 小时。测试目标不变，仍然建立双目标 Pareto 优化模型，改动约束条件：工作时间为 $0 \sim K, K \sim 2K$。测试流程

和随机事件不变，仍然采用基于蒙特卡洛方法的离散事件模拟。但工作时长可变后，动态优化策略可增加一个换班策略。

5.2 问题 3：双目标 Pareto 优化模型的建立

相较于问题 2，问题 3 优化模型的目标函数没变，仍然是任务完成平均天数T尽量少，总漏判概率P_L尽量小；改动一个约束条件，工作时长由 12 小时改为K小时，测试小队增加为两队。双目标 Pareto 优化模型改为：

$$\min T \quad , \quad \min P_L$$

$$s.t\begin{cases} \max\limits_{i\in\{A,B,C\}} end_{i,j} \le start_{E,j} \\ end_{i,j} \le start_{i,k} \\ end_{E,j} + t_{trans} \le \min\limits_{i\in\{A,B,C\}} start_{i,k} \\ end_{i,j} = start_{i,j} + n_{test-i,j} \times t_{test,i} \\ K \in [9,12],\ t_{trans} = 0.5 \\ t_{test,i} = \{2.5, 2, 2.5, 3\},\ t_{calib,i} = \{0.5, 1/3, 1/3, 2/3\} \\ n_{test-i,j} \in \{1, 2\},\ n_{team} = 2 \\ \theta_i \in [120, 240],\ j \in [1, 100] \end{cases} \tag{19}$$

后续构建 Patero 解集步骤与 4.3.3 节相同。

5.3 问题 3：双目标 Pareto 优化模型的求解

与问题 2 相同，基于双目标优化模型（改动）、蒙特卡洛方法和三种动态决策策略，采用离散事件模拟实现对测试流程、随机事件发生、动态规划工作计划的模拟。优化测试设备更换时间θ_i、测试小队工作时长K，得到最优 Pareto 解。为了寻找最优解对应的值，选用效果更好的 MOEA/D 算法，求解流程与 5.5 节相同。

设定 MOEA/D 算法的迭代次数为 60，种群大小为 7（一代包含 7 个候选解），重复运行 10 次，得到 Pareto 前沿解，如图 7（变量为设备主动更换时间）和图 8（变量为班次工作时长）所示。Pareto 前沿最优解（红点区域）在 XY 平面的映射区域主要集中在最左上角，即目标函数的最优区域（此区域任务完成时间T和漏检率P_L最低）。

三维散点图中 x 轴为K，y 轴为T，z 轴为P_L。在优化过程中，散点在T、P_L方向几乎不变化，呈水平平面分布。同时，每个固定的K值仅产生几个离散聚点，而不是连续分布，说明 MOEA 在局部空间内进行探索，并最终跳脱局部解找到全局 Pareto 前沿。从图中可以看到，最终 Pareto 前沿几乎都集中在$T=15, P_L=0$平面上，说明 Pareto 前沿对K的稳健性较高，算法能够稳定地找到全局最优解。

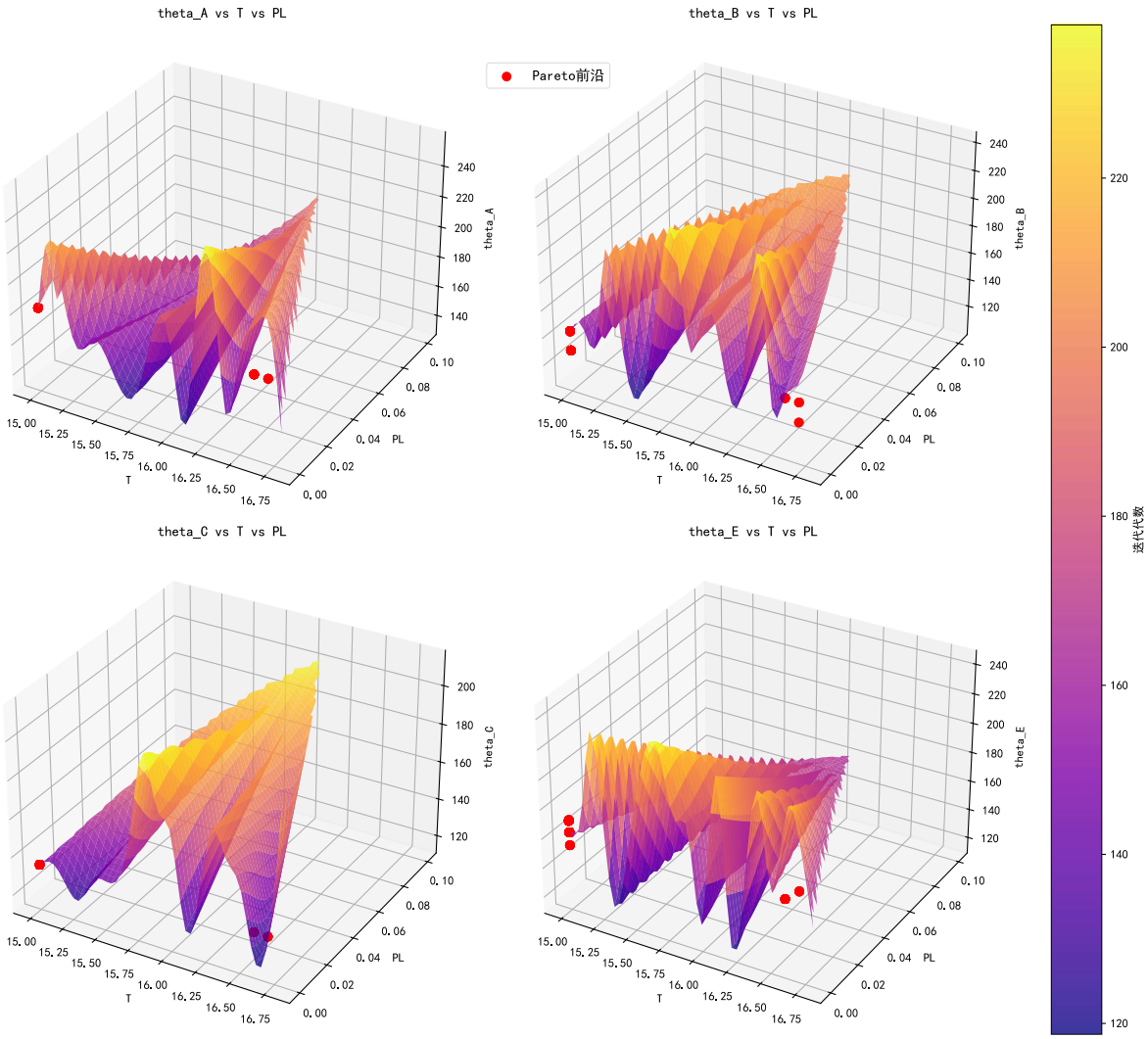

图 7 MOEA\D 算法解得的设备主动更换时间的 Patero 前沿解

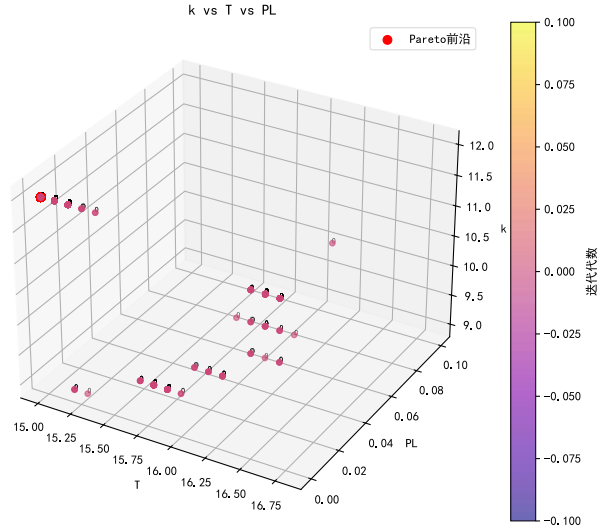

图 8 MOEA\D 算法解得的班次工作时长的 Patero 前沿解

Pareto 前沿解再通过 Bootstrap 重抽样和概率关联度一致性检验，得到稳健的 Pareto 解集，如表 5 所示。

表5 MOEA/D 算法得到的稳健 Pareto 解集

测试设备 A 更换时间 θ_A	测试设备 B 更换时间 θ_B	测试设备 C 更换时间 θ_C	测试设备 E 更换时间 θ_E	班次工作时长 K	任务平均完成时间 $T[95\%CI]$	总漏判率 $P_L[95\%CI]$
182.3	142.3	134.3	170.8	12	14.95 [14.90,15.05]	0.000120 [0.000000,0.000250]
182.3	128.5	134.3	153.5	12	15.05 [14.95,15.12]	0.000300 [0.000000,0.000450]
182.3	128.5	134.3	170.5	12	15.02 [14.93,15.11]	0.000280 [0.000000,0.000420]
182.3	142.8	134.3	170.5	12	15.06 [14.96,15.14]	0.000305 [0.000000,0.000460]
182.3	142.8	134.3	162.3	12	15.04 [14.95,15.13]	0.000267 [0.000000,0.000430]
182.3	142.8	134.3	170.3	12	15.03 [14.94,15.11]	0.000246 [0.000000,0.000400]
182.3	142.8	134.3	170.2	12	15.01 [14.92,15.09]	0.000245 [0.000000,0.000380]

权衡任务完成时间和总漏判率，按照 $\min 0.5\times\frac{T}{\tilde{T}}+0.5\times\frac{P_L}{\tilde{P}_L}$ 原则，选中表 5 中蓝色行为最优解。任务平均完成时间为 14.95 小时，总漏判率为 0.000120%。工作计划：182.3 小时后更换测试设备 A，142.3 小时后更换测试设备 B，134.3 小时后更换测试设备 C，170.8 小时后更换测试设备 E，每个班次工作时长为 12 小时。

按照此工作计划，重复多次蒙特卡洛试验，得到 8 个指标的结果如表 6 所示。

表6 MOEA/D 算法得到 8 个统计指标

T /天	S /个	P_L /%	P_W /%	YXB_1	YXB_2	YXB_3	YXB_4
15.27	92.53	0.0073	5.3877	0.723	0.584	0.717	0.786

6 问题 4：敏感性分析与建议

问题 4 要求分析讨论问 3 中各个因素对测试任务平均完成时间 T 的影响，即敏感性分析，并根据分析结果对主管提出建议。

6.1 敏感性分析

敏感性分析是指从定量分析的角度研究有关因素发生某种变化对某一个或一组关键指标影响程度的一种不确定分析技术。其实质是通过逐一改变相关变量数值的方法来解释关键指标受这些因素变动影响大小的规律。本题存在多种因素会对测试任务平均完成时间产生影响，例如测试设备数量、工作时长等增加，会给测试任务完成时间带来成倍的减少。

本文针对不同的影响因素做 3 组敏感性分析，1 组粗粒度敏感性分析和 2 组细粒度敏感性分析。细粒度敏感性分析区分了不同的测试设备。

- 粗粒度敏感度分析（4 个变量）：**测试小队班次工作时间** $K \in [9,12], \text{step } 0.5$、**测试设备使用时长** $\theta \in [120,240], \text{step } 20$（假设四个设备使用时长相同）、**测试台数量** $n_{plat} \in [2,5], \text{step } 1$、**测试设备套件数量** $n_{device} \in [1,5], \text{step } 1$（假设四个套件数量相同）。
- 细粒度敏感度分析 1（4 个变量）：**测试设备使用时长**分别设为变量 $\theta_i (i = A,B,C,E)$，其余 3 个变量固定。
- 细粒度敏感度分析 2（4 个变量）：**测试设备数量**分别设为变量 $n_{device,i} (i = A,B,C,E)$，其余 3 个变量固定。

不同因素对任务完成时间的影响效果不同，有的线性单调，有的非线性单调，有的不单调，因素之间还可能出现相互作用、相互约束的关系。本文采用以下 3 种敏感度分析方法。

(1) 标准化回归系数（Standardized Regression Coefficients，SRC）

标准化回归系数的基本思想是用线性回归近似 Y 对输入变量 X_j 的依赖关系，把回归系数标准化以比较不同变量对 Y 的相对影响大小。它给出在控制其它变量情况下，输入变量 X_j 单位标准差变化对输出 Y 的标准差变化的影响。

假设原始带截距的线性回归方程为：

$$Y = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \varepsilon$$

经过标准化 $\tilde{X}_j = \dfrac{X_j - \mu_{X_j}}{\sigma_{X_j}}, \tilde{Y} = \dfrac{Y - \mu_Y}{\sigma_Y}$（其中 μ 为均值，σ 为标准差）后回归变为：

$$\tilde{Y} = \beta_0' + \sum_{j=1}^{p} \beta_j^{std} \tilde{X}_j + \varepsilon'$$

其中 $\beta_j^{std} = \beta_j \dfrac{\sigma_{X_j}}{\sigma_Y}$ 就是标准化回归系数 SRC。

SRC 的符号表明影响方向（正表示增加该输入因数会增加测试时间，负表示减少测试时间）；绝对值表示影响的强弱（越大越重要）。但 SRC 只能度量线性关系，对强非线性不敏感。当 SRC 为零时，可能反映了输入输出关系的不单调，线性度量将正负贡献相互抵消了。

(2) 偏秩相关（Partial Rank Correlation Coefficient, PRCC）

PRCC 是在控制其它变量的影响后，衡量某输入变量（秩变换后）与输出（秩变换后）之间的线性偏相关系数。它通过秩变换（rank）使方法对单调非线性更健壮，并通过“偏相关”控制其它输入的影响，所以能衡量“在控制其它参数后，该参数与输出的单调关系”。

假设有 p 个输入 $X_1, X_2, \cdots X_p$，输出 Y。对每个变量做“秩变换”，得到：

$$R_{X_j} = \text{rank}(X_j), \quad R_Y = \text{rank}(Y)$$

对第 i 个变量，做两次线性回归。一是用线性回归把 R_{X_i} 对其它 $R_{X_{-i}}$ 回归，取残差 r_i。二是用线性回归把 R_Y 对其它 $R_{X_{-i}}$ 回归，取残差 r_Y。PRCC 为残差 r_i 与 r_Y 的 Pearson 相关系数。

PRCC 取值[−1,1]区间，绝对值大表示该参数在控制其它参数后与输出存在强单调关系（正值表示单调增加关系，负值表示单调减少关系）。因为先做秩变换，PRCC 对于单调非线性关系仍能很好的检测。但是 PRCC 假设关系是单调的，若关系是非单调（例如先下降后上升，PRCC 可能正负相消为零），PRCC 可能低估或给出误导。

(3) 主效应（level means）与差分（边际收益）

对每个因子 g 的每个离散水平 k，计算在该水平下输出 Y 的平均值（在其它因子上取平均或在实验设计中用随机/均匀采样）。差分 $\Delta(k)=\tilde{Y}(k)-\tilde{Y}(k+1)$ 则表示输入参数从 k 增到 $k+1$ 的平均边际收益。

主效应定义为：

$$\mu_g(k)\approx\frac{1}{|S_k|}\sum_{s\in S_k}Y_s$$

其中 S_k 是所有试验中 $n_g=k$ 的配置集。

边际收益定义为：

$$\Delta_g(k)=\mu_g(k)-\mu_g(k+1)$$

计算步骤是先把仿真样本按每一个因子分组计算平均值和标准误差，然后画出主效应曲线（水平 k 对应的平均测试时间）并加上标准误差，最后计算相邻水平差分。

主效应曲线直观展示水平 k 对应的平均测试时间。如果曲线呈递减并趋于平坦，说明递减边际收益——开始收益大，后续收益小。该方法对非线性、交互都能直观反映，若因素之间有显著交互，单独的主效应可能掩盖交互，需要做交互效应分析。

综上所述，上述三种方法能够分析出单个输入因素对输出单调性的影响，但输入因素之间可能存在相互作用，引入全局影响性的分析——Moriss 方法。

(4) Morris 筛选法

Morris 方法的核心思想是在输入空间上做许多单因素移动（One-At-A-Time，OAT），每次只改变一个输入变量，记录输出增量，把这些局部斜率（elementary effects）在空间上取样并统计。这样既能用较少模型运行次数得到全局敏感度排序，又能通过 EE 的离散程度判断非线性和交互性的存在。

Morris 首先的任务是样本采样。设有个 k 输入变量 X，把每个变量都线性标度到标准化区间[0,1]。在该区间上取 p 等分网格，允许取值集合为：

$$\{0,\frac{1}{p-1},\frac{2}{p-1},\ldots,1\}$$

定义 Morris 采样步长为：

$$\Delta=\frac{p}{2(p-1)}$$

保证采样的起点和终点都落在 Morris 设置的网格里。

在规定采样点的移动规则后，进行采样点的轨迹生成，一条轨迹包含 $k+1$ 个样本采样点。轨迹的生成矩阵公式如下：

$$X=x^*\mathbf{1}^\top+\frac{\Delta}{2}\Big(\big(2B-J\big)D+J\Big)P$$

轨迹的生成矩阵 X 维度大小为 $(k+1)k$； D 为 $k\times k$ 的对角矩阵，用于决定每个变量在轨迹上的步方向 $(+1,-1)$；P 为 $k\times k$ 的置换矩阵，用于随机化变量改变的顺序，即轨迹上第几步改变什么变量。

再随机采样到足够多的轨迹后，需要得到每个变量的 EE 值。轨迹上有 $k+1$ 点，记其输出为序列 $Y_0, Y_{1,}\cdots Y_k$。在轨迹上第 t 步改变的是变量 j (由 P 决定顺序)，则该步对应的 EE 为：

$$EE_j^{(r)}=\frac{Y_t-Y_{t-1}}{\Delta}$$

其中 Δ 是标准化步长，对一条轨迹会为 k 个变量各得到一个 EE。做 R 条独立轨迹，变量 j 最终得到 R 个 EE 值 $\{EE_j^{(1)},\cdots,EE_j^{(R)}\}$。

在 Morris 筛选法中，最关键的两个变量如下，首先是绝对值均值 μ_j^*，其大小衡量了相对于总体系统的重要性。当 μ_j^* 越大时，该变量对总体的影响性更大。

$$\mu_j^*=\frac{1}{R}\sum_{r=1}^{R}|EE_j^{(r)}|$$

其次是各变量的标准差，直观表示系统的离散和变动程度。当 σ_j 越大时，证明该变量在总体系统中的非线性更强，并且该变量和其他变量发生交互性影响的可能性更大。

$$\sigma_j=\sqrt{\frac{1}{R-1}\sum_{r=1}^{R}\left(EE_j^{(r)}-\mu_j\right)^2}$$

根据 Morris 筛选法，可以通过其统计量得到各变量在全局上的影响性。在本设计中，Morris 筛选法将在粗粒度敏感度分析中使用，为粗粒度条件下的 SRC 和 PRCC 分析，提供更加全局的视野。在细粒度敏感度分析中，由于设置的变量的量纲相同，仅对应于不同测试设备，故使用 SRC 和 PRCC 分析各变量的影响性，不额外设置 Morris 筛选法实验。

6.2 分析结果与建议

采用上述四种方法相结合的方式进行了 1 组粗粒度敏感性分析和 2 组细粒度敏感性分析，分析结果如下。

6.2.1 粗粒度敏感性分析结果和建议

6.2.1.1 结果分析

首先使用 Morris 算法去测试多个因素在全局中的重要性。其中设置了的 4 个全局变量，测试小队班次工作时间 $K\in[9,12],\text{step }0.5$、测试设备使用时长 $\theta\in[120,240],\text{step }20$（假设四个设备使用时长相同）、测试台数量 $n_{plat}\in[2,5],\text{step }1$、测试设备套件数量 $n_{device}\in[1,5],\text{step }1$（假设四个套件数量相同）。Morris 算法的结果有两个，μ^*（绝对效果均值）表示变量平均影响力；σ 表示效果的非线性的交互强度，结果如下表所示。

表7 Morris 方法对 4 个全局变量的分析结果

指标	μ^*	σ
班次工作时间 K	34.395	36.440
测试设备使用时长 θ	1.500	1.979
测试台数量 n_{plat}	128.731	92.858
测试设备套件数量 n_{device}	96.951	111.857

Morris 算法给出了简明情况下的影响力对比，后续使用 PRCC 和 SRC 进一步细化影响力分析。PRCC 是将测变量与输出间的秩相关（在控制其他变量影响后），带有非线性因素，值的绝对值越大说明全局上相关性越强；正号说明变量增大会使目标（任务完成时间）变大，负号说明变量增大使任务完成时间变小。SRC 是将变量标准化后做线性回归得到的系数，反映线性近似下每个变量的相对影响力。

表8 4 个全局变量的 PRCC 和 SRC 结果

指标	*PRCC*	*SRC*
班次工作时间 K	0.0658	0.0303
测试设备使用时长 θ	-0.0385	-0.0102
测试台数量 n_{plat}	-0.7732	-0.5954
测试设备套件数量 n_{device}	-0.6580	-0.5011

图 9 小队工作时长的均值和区间差值统计图

图 10 设备主动更换时长的均值和区间差值统计图

图 11 测试台数量的均值和区间差值统计图

图 12 测试设备数量的均值和区间差值统计图

（1）强影响因素：根据 Morris 结果和 PRCC 以及 SRC 的结果，测试台数量和测试设备的数量是影响任务完成时间的最重要的 2 个因子。他们的 PRCC 为强负相关，意味着增加测试台数会显著降低平均完成时间（更快完成）；并且，他们的 SRC 也很大，证明其在线性近似和单调性上也很显著。

在 Morris 分析中，μ^* 很大，证明对整个系统任务的完成有着强影响性。并且，两者的 σ 同样很大，说明 2 者具有明显非线性交互效应。由题目依据也可分析，仅增加测试设备的数量，而没有充足的测试台提供测试装置，任务完成时间是不会有很大降低。只用当测试台的数量和测试设备同步增加时，才能有效减少任务完成时间。

从边际收益来看从 2 增长到 3 个测试台，平均完成时间减少约 92.36 小时，为单步内最大收益，后续的边际收益逐渐递减。在当前系统下，把每种测试设备从 1 套增加到 2 套带来巨大改善，之后再增加套件的边际效应急剧下降，说明此处是关键拐点。

（2）弱影响因素：根据 Morris 结果和 PRCC 以及 SRC 的结果，小队工作时长和设备主动退出时长是弱影响因素。其中小队工作时长有影响，但同样保持非单调非线性。

两者的 PRCC 和 SRC 都显著接近于 0，表示了他们在系统中的线性度较小，且影响性也较小。均值曲线也很好的反映了这一点，但在范围内来看，最小和最长两个极端情况下的 K 值，对整个系统的时间缩减有所帮助。设备主动退出时长相比于小队工作时长来说更不敏感，上述 3 中敏感度分析数值都接近 0。在当前任务中，可以把它列为低优先级参数。后续可按维护成本或设备寿命/可靠性考虑设置，而不是为缩短完成时间去优化它。

6.2.1.2 给工厂的可执行决策建议

建议 1：增加测试设备的数量。由统计图可知，把每类测试的设备套件从 1 增到 2，平均任务完成时间减少约 118.2 小时，在四变量组合空间里这是最大单步收益，可以满足短期直接收益的最大化。

建议 2：增加测试台的数量。在建议 1 的基础上，要想在后续的测试中持续提高任务完成时间，需要增加测试台的数量，使其数量匹配测试设备的数量。根据结论中 2 者

的强交互性分析，建议 2 和建议 1 可以形成良好的互补关系。但增加测试台数量往往预算远大于增加测试设备的数量，所有在建议中将增加测试台的数量放在低优先级的位置。

建议 3 调整小队工作时长 K 的安排。该建议是结合实验性调整考虑，K 对总体有影响，但非单调且依赖交互。后续可在生产线上完善短期测试，在真实条件下试行候选 K 值各若干批次，记录完成时间与停滞、交接事件，观察 K 值在实际约束的稳健性。若人员排班或劳动法限制，优先选择能兼顾人员疲劳和设备校准窗口的 K 值。

建议 4：延长测试设备的主动更换时间。由上文的分析可知，测试设备的主动更换时长对整体的任务要求（减少任务完成时间）的影响性很小。故不建议为短期加速而改动该阈值，应按照设备可靠性和维护成本最优来设置，即以可靠性、成本最优为目标，而非任务完成时间。

6.2.2 细粒度敏感性分析 1 结果和建议

6.2.2.1 分析结果

细粒度敏感度分析 1 的 4 个变量分别是**测试设备使用时长** $\theta_i (i = A, B, C, E)$，所以不需要考虑全局性。

表9 4 个变量的 PRCC 和 SRC 结果

指标	*PRCC*	*SRC*
测试设备 A 使用时长 θ_A	-0.0151	-0.0178
测试设备 B 使用时长 θ_B	0.0120	-0.0291
测试设备 C 使用时长 θ_C	0.0275	-0.0366
测试设备 E 使用时长 θ_E	-0.1059	-0.1740

从 SRC 与 PRCC 看，四类设备的主动退出时长对总体完工时间的线性和秩相关性都非常小（A、B、C 的系数几乎为 0，E 虽然相对最大但也仅为 SRC≈−0.174、PRCC≈−0.106），说明在当前系统与其他参数固定的条件下，调整单台设备的更换时长对任务完成时间的直接影响可忽略不计。

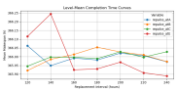

图 13 不同测试设备的主动更换时长的均值

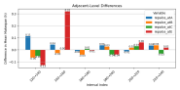

图 14 不同测试设备的主动更换时长的区间差值

主效应层面各水平的平均仅在小量级（约几十分之一到几十分之几小时）内波动，E 的全区间跨度约 0.35 小时左右，其它设备的水平间差异更小，且差分符号无稳定单调趋势，进一步支持非显著、非单调且交互性占主导的结论。

6.2.2.2 给工厂的可执行决策建议

工厂在布置任务时，可以不要为了缩短总体完工时间去优先调整 A、B、C、E 更换时长阈值；将维护阈值按设备可靠性與维护成本（寿命、维修窗口、备件成本）来确定，而不是以任务完成时间为唯一目标。对相对影响稍大 E 设备建议加强日常维护与测量质量控制，降低漏判、误判与校准波动。同时把精力和预算优先投入到仿真已显示更敏感的因素（如增加测试台数、增加设备套件或优化班次 K）以获得更明显的进度改善。

6.2.3 细粒度敏感性分析 2 结果和建议

6.2.3.1 分析结果

细粒度敏感度分析 2 的 4 个变量分别是**测试设备数量** $n_{device,i}(i=A,B,C,E)$，所以不需要考虑全局性。

表10 4 个变量的 PRCC 和 SRC 结果

指标	*PRCC*	*SRC*
测试设备 A 数量 $n_{device,A}$	-0.6547	-0.3953
测试设备 B 数量 $n_{device,B}$	-0.2305	-0.1807
测试设备 C 数量 $n_{device,C}$	-0.5072	-0.3826
测试设备 E 数量 $n_{device,E}$	-0.3678	-0.3160

从 PRCC 与 SRC 的结果看，增加各子系统的设备套件数对平均完工时间有显著负向影响，增加套件数会降低任务完成时间，其中 A 与 C 的影响最大，根据题意 A C 组测试单件耗时最长，且故障率较低，重测概率很小；因此每多一出一套设备时抢占池的瓶颈立刻释放。E 设备影响次之，其调试和测试的总时间最长，边际收益被高耗时和故障率抵消。B 设备由于单件耗时最短，且故障率最高，而对任务完成时间影响最小，即使多设 B 设备，也很快被高故障重测拖慢，性价比最低。从 1 套到 2 套件的单步收益最大，而随后的增量带来的边际收益急剧下降。因此系统存在明显的瓶颈拐点，把每类设

备数量从 1 提升到 2 是关键的高回报操作，之后呈现递减边际效应，说明增加套件数后会很快遇到其他瓶颈（如测试台、工位或班次安排）限制。

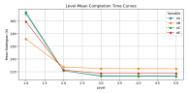

图 15 不同测试设备数量的均值

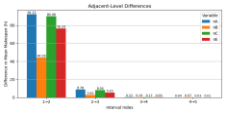

图 16 不同测试设备数量的区间差值

6.2.3.2 给工厂的可执行决策建议

优先把 A 与 C 的设备数由 1 提升到 2（若预算只够一项，先做 A 或 C，其单步节省最大），其次考虑提升 E 设备的数量。对 B 的投资回报较小可放在后续阶段或在确认其它瓶颈已移除后再增设。此外，实施时务必同步检查并扩容相关约束（测试台数量、每组工人数量、人员班次与交接、运输能力），避免出现设备闲置，但操作人数、操作装置受限的次生瓶颈。在正式大规模投入前建议做小范围试点并计算 ROI（小时节省×小时价值 vs 运营成本），并用局部响应面进一步量化协同效应以确定最优投资组合。

7 模型评价与推广

7.1 模型的优点

(1) 考虑不确定性、贴近实际：将离散事件模拟和蒙特卡洛嵌入优化环节，使得随机故障、测手差错等真实工程不确定性能被显式建模，所得目标（任务完成平均天数T尽量少，总漏判概率P_L尽量小）具有统计意义，可用于稳健决策。

(2) 多目标优化与均衡解集：采用 MOEA/D（以分解策略生成均匀 Pareto 前沿）能同时给出一组权衡解，便于在“时间一风险“之间做决策比对，而不是仅给出单一最优。
(3) 鲁棒性评估：对 Pareto 点进行 Bootstrap 重抽样并结合统计显著性检验，能识别并保留置信区间稳定、统计上有差异的稳健解，增强结论的可信度。
(4) 可解释的决策输出：最终以带置信区间的参数组作为推荐策略，便于向工程管理层交付可量化的操作建议。

7.2 模型的不足与局限

(1) 仿真噪声或参数设置可能导致指标“退化”：如果 PL 全为 0、多个解 T 值完全相同，可能是仿真中故障/漏判概率设置过小、每次评估的蒙特卡洛样本数过低或随机种子/事件模型实现导致样本变异不足，从而影响 Pareto 多样性与置信区间估计。
(2) 计算开销大：基于蒙特卡洛的 MOEA/D（多代、大种群）带来大量仿真评估，若不采用加速（代理模型、多保真等），实验耗时高，且难以在有限时间内保证空间覆盖度。
(3) 多样性维护不足：若优化操作（采样、交叉、变异、邻域策略）或初始化策略未能有效维护解的多样性，会导致 Pareto 集中、早熟收敛，从而难以发现真正的全局多样解。

7.3 模型的推广与改进建议

为增强结论可靠性、提高结果多样性与降低计算成本,建议从以下方向改进与推广：

(1) 增加或调整随机性与蒙特卡洛设置：提高 n_runs(每次评估的重复数)或对关键事件(故障、漏判)采用较大概率区间做敏感分析，可使 PL 等指标呈现更合理的变异性;同时用不同随机种子重复实验以检验稳健性。
(2) 参数/模型诊断与校准：对仿真中故障率、误判率等关键概率参数做来源检验或基于历史数据的贝叶斯校准，避免因参数过小导致指标退化。
(3) 加速与多保真策略：引入代理/元模型(GaussianProcess、树模型等)或多保真仿真(低成本、低精度版用于初筛)以减少高成本仿真次数。采用自适应采样(uncertainty sampling)在有信息处密集采样，提升效率。
(4) 改进优化多样性保持机制：调整 MOEA/D 的邻域大小、变异/交叉策略，或引入分层初始化(拉丁超立方、Sobol 序列)与群体重启策略，以避免早熟。
(5) 鲁棒与风险度量拓展：除均值与置信区间外，引入更具风险意识的目标(如百分位数、CVaR)为决策指标，提供更保守的工程建议。
(6) 开展系统级敏感性分析：使用 Morris 筛选、PRCC 等方法识别对 T、PL 影响最大的因子，再把计算资源优先分配到关键因子组合上做精细化优化。

参考文献

[1] 王雯, 赵凯南, 杨林, 等. 面向复杂场景的层次式任务规划方法[J]. 系统工程与电子技术, 2025, 47(4): 1255-1264.

[2] 宁小磊, 吴颖霞, 赵新, 等. 小样本概率关联度模型研究[J]. 西北工业大学学报, 2022, 40(5): 1164-1171.

[3] 程国勇, 陈实. 基于离散事件模拟的航站楼运行韧性分析[J]. 北京航空航天大学学报, 2024, 50(11): 3310-3318.

附　录

附录 A: 支撑材料列表

```
├─── figsQ2/                                    # Q2 问题相关图表
│   └─── theta_surface_progress.svg
├─── figsQ3/                                    # Q3 问题相关图表
│   ├─── k_scatter_progress.svg
│   ├─── k_scatter_with_gen.svg
│   ├─── k_subplots_scatter.svg
│   └─── theta_surface_progress.svg
├─── calc_static.py                             # Q2Q3 中统计计算模块
├─── data_source.py                             #  数据源处理模块
├─── fig.py                                     # Q2Q3  绘图功能模
块
├─── GA.py                                      #  遗传算法  MOEA
实现
├─── Q1.py                                      # Q1 处理
├─── Q2.py                                      # Q2 处理
├─── Q3.py                                      # Q3 处理
├─── Q4_model.py                                # Q4 系统仿真系统
├─── Q4_test_large.py                           # Q4 粗粒度测试
├─── Q4_test_replace_num.py                     # Q4ABCE 设备使用时长
├─── Q4_test_set_num.py                         # Q4ABCE 设备数量
├─── test_stack_2group.py                       # Q2 系统仿真系统
└─── test_stackV5.py                            # Q3 系统仿真系统


---------------------------------------requirements ---------------------------------------
joblib==1.5.2
matplotlib==3.8.4
numpy==2.3.2
pandas==2.3.2
pymoo==0.6.1.5
SALib==1.5.1
scikit_learn==1.7.1
scipy==1.16.1
statsmodels==0.14.5
tqdm==4.66.2
```

附录 B: 主要程序/关键代码

代码环境	操作系统: Windows 11 23H2(OS 22631.3447) 编程语言: Python 3.9.19(AnacondaNavigator 1.9.7) 编辑器: PyCharm 2020.1(Professional·Edition) 代码详见: Code

代码清单 1 问题 2 离散事件模拟系统代码

```
# test_hall_event_v4_Kshift.py
# 在 V4-fixed 基础上修改：采用 2 个小队交替工作，每个小队工作 K 小时（连续轮换），
# 若当前小队剩余时间不足以完成某次运算（运输/校准/测试），则该任务被保存至待办池，
# 并在下一个小队开始时由其接手（不会在两个小队之间分段执行同一次测试）。

import heapq
import random
import math
from collections import deque
from data_source import *    # 保留你的外部参数数据源
from Q1 import lambda_vals    # 保留 Q1 输出

# ---------- 参数区（保持不变或可调） ----------
RANDOM_SEED = 12345678
random.seed(RANDOM_SEED)

q = {'A': 0.025, 'B': 0.03, 'C': 0.02, 'D': 0.001}
e = {'A': 0.03, 'B': 0.04, 'C': 0.02, 'E': 0.02}
REPLACE_AT = {'A': 130, 'B': 140, 'C': 150, 'E': 160}
alpha = 0.5
leak = {k: e[k] * alpha for k in e}

# 问题3中调整的 K（每个小队工作时长，单位小时）
# 可设置为 9.0 <= K <= 12.0（以 0.5 为步长）
K_WORK_HOURS = 12.0


# ---------- 小队调度工具函数（基于 K 轮替） ----------
def current_team(now, K=K_WORK_HOURS):
    """返回当前时刻属于哪一个小队：0 或 1（交替）"""
    return int((now // K) % 2)

def team_time_elapsed(now, K=K_WORK_HOURS):
    """返回当前小队已工作多少小时（在 0..K)"""
    return now % K

def team_time_remaining(now, K=K_WORK_HOURS):
    """返回当前小队剩余可用时间（小时）"""
    return K - (now % K)

def next_team_start(now, K=K_WORK_HOURS):
    """返回下一个小队开始的绝对时刻（当前小队剩余时间后的时刻）"""
    rem = team_time_remaining(now, K)
    return now + rem

# ---------- 累积故障概率 -> hazard (保持你原来的线性分段实现) ----------
def make_hourly_hazard(cum_low, cum_high):
    if cum_high < cum_low:
        cum_high = cum_low
    s1 = cum_low / 120.0
    s2 = (cum_high - cum_low) / 120.0
    return s1, s2

equip_hazard = {k: make_hourly_hazard(*equip_cum[k]) for k in equip_cum}

# ---------- 设备类 ----------
```

```
class Equipment:
    def __init__(self, name, hazard_rates, min_run=120.0):
        self.name = name
        self.s1, self.s2 = hazard_rates
        self.usage = 0.0
        self.min_run = min_run
        self.replacements = 0
        self.total_calib_time = 0.0
        self.calibrated = False
        self.pre_calibrated = False

    def cumf(self, u):
        """分段线性累计故障概率函数 F(u) (u = 使用小时数)"""
        if u <= 0:
            return 0.0
        if u <= 120.0:
            return min(1.0, self.s1 * u)
        if u <= 240.0:
            return min(1.0, self.s1 * 120.0 + self.s2 * (u - 120.0))
        # 超过 240h 的情形：按题意到 240h 必须替换，理论上不会继续累计使用；
        # 若仍计算，直接返回 1.0（或返回 F(240)）。这里取 max(F(240), 1.0) 的下界保护。
        return 1.0

    def prob_fail_during(self, duration):
        """
        计算在当前 usage (= self.usage) 的基础上，接下来 duration 小时内发生故障的概率：
            P = F(usage + duration) - F(usage)
        并限定在 [0,1]。
        """
        cur = self.usage
        end = cur + duration
        # 若 end >= 240，按照题意设备在 240h 必须替换；仍然可用差分计算（或直接视为到
240 有额外风险）
        p = self.cumf(end)
        # 数值保护
        if p < 0:
            p = 0.0
        if p > 1:
            p = 1.0
        return p

    def advance(self, dur):
        self.usage += dur

    def replace(self):
        self.usage = 0.0
        self.replacements += 1
        self.calibrated = False
        self.pre_calibrated = False

# ---------------- 事件与仿真器 ----------------
class Event:
    seq = 0
    def __init__(self, time, callback, desc=""):
        Event.seq += 1
        self.time = time
        self.callback = callback
        self.desc = desc
        self.seq = Event.seq
    def __lt__(self, other):
        if self.time == other.time:
            return self.seq < other.seq
        return self.time < other.time

class Simulator:
    def __init__(self):
        self.now = 0.0
        self.pq = []
        self._debug = False
    def schedule(self, time, callback, desc=""):
        if time < self.now - 1e-9:
```

```
                raise RuntimeError("不能调度过去的时间")
            ev = Event(time, callback, desc)
            heapq.heappush(self.pq, ev)
        def run(self):
            while self.pq:
                ev = heapq.heappop(self.pq)
                self.now = ev.time
                if self. debug:
                    print(f"[RUN] t={self.now:.2f} | {ev.desc}")
                ev.callback(self.now)

# ---------- 资源 ----------
class ResourceWithQueue:
    def __init__(self, capacity):
        self.capacity = capacity
        self.available = capacity
        self.queue = deque()
    def request(self, retry_cb):
        if self.available > 0:
            self.available -= 1
            return True
        else:
            self.queue.append(retry_cb)
            return False
    def release(self, sim: Simulator, now):
        self.available += 1
        if self.queue:
            cb = self.queue.popleft()
            sim.schedule(now, cb, desc="resource retry")

# ---------- Bay ----------
class Bay:
    def __init__(self, id, slots):
        self.id = id
        self.place_taken = False
        self.current_dev = None
        self.reserved_incoming = None
        self.slots_capacity = slots
        self.slots_used = 0

# ---------- 主仿真类（K 小队轮值） ----------
class TestHallEventV4_K:
    def __init__(self, n_devices=100, replace_at=None, bay_slots=BAY_SLOTS, seed=None, verbose=True, K=K_WORK_HOURS):
        if seed is not None:
            random.seed(seed)
        self.n = n_devices
        self.replace_at = replace_at
        self.verbose = verbose
        self.K = K
        self.sim = Simulator()
        self.sim. debug = False
        self.bays = [Bay(i, bay_slots) for i in range(NUM_BAYS)]
        self.transport_capacity = NUM_BAYS
        self.transport_queue = deque()
        self.groups = {g: ResourceWithQueue(1) for g in ['A', 'B', 'C', 'E']}
        self.equip = {g: Equipment(g, equip_hazard[g]) for g in ['A', 'B', 'C', 'E']}
        self.devices = [{'id': i,
                         'qA': random.random() < q['A'],
                         'qB': random.random() < q['B'],
                         'qC': random.random() < q['C'],
                         'qD': random.random() < q['D']} for i in range(self.n)]
        self.next_idx = 0
        self.passed = 0
        self.rejected = 0
        self.completion_times = []
        self.false_neg = 0
        self.false_pos = 0
        self.tests_count = {'A': 0, 'B': 0, 'C': 0, 'E': 0}
        self.pending_attempts = deque()
```

```
    def log(self, t, msg):
        if self.verbose:
            team = current team(t, self.K)
            self name = f"[Team{team}]"
            print(f"[{t:.2f}h] {self name} {msg}")

    # ---------- 启动 ----------
    def start(self):
        # 预先 schedule bay_manager
        self.sim.schedule(0.0, self. bay manager step, desc="bay manager")
        self.sim.run()

    # ---------- Bay 预约与运入（考虑 K 小队时长） ----------
    def  bay manager step(self, t):
        while self.next idx < self.n:
            dev = self.devices[self.next idx]
            chosen = None
            for b in self.bays:
                if b.reserved incoming is None:
                    chosen = b
                    break
            if chosen is None:
                break
            chosen.reserved incoming = dev
            self.log(t, f"装置 {dev['id']} 已预约到 bay {chosen.id}")
            self.next idx += 1
            if not chosen.place taken:
                self. request transport in for bay(dev, chosen, t)

    def  request transport in for bay(self, dev, bay, t):
        # 在 K 机制下，运输只能在当前小队剩余时间足够开始并完成
        rem = team time remaining(t, self.K)
        # 若当前小队剩余时间不足以完成 transport_time，则把任务推到下一小队开始
        if transport time > rem:
            ns = next team start(t, self.K)
            self.log(t, f"transport 时间不足（剩余 {rem:.2f}h） -> 将装置 {dev['id']} 的入场推
迟到 {ns:.2f}h")
            self.sim.schedule(ns, lambda now, d=dev, b=bay: self. request transport in for bay(d,
b, now),
                              desc=f"deferred transport in dev{dev['id']} bay{bay.id}")
            return

        def start in(now, d=dev, b=bay):
            self.transport capacity -= 1
            self.log(now, f"装置 {d['id']} 开始运入 bay {b.id}（transport 占用）")
            end = now + transport time  # 在 K 模式下无需跨天处理（已检查能完成）
            def on end(et, d=d, b=b):
                self.transport capacity += 1
                b.place taken = True
                b.current dev = d
                b.reserved incoming = None
                self.log(et, f"装置 {d['id']} 已运入 bay {b.id}")
                self. start device flow(d, b, et)
            self.sim.schedule(end, on end, desc=f"transport in end dev{d['id']} bay{b.id}")

        if self.transport capacity > 0:
            self.sim.schedule(t, start in, desc=f"start transport in dev{dev['id']} bay{bay.id}")
        else:
            self.transport queue.append((dev, bay, False))
            self.log(t, f"transport 满，装置 {dev['id']} 运入排队 bay {bay.id}")

    # ---------- swap ----------
    def  start swap(self, now, incoming dev, bay):
        rem = team time remaining(now, self.K)
        if transport time > rem:
            ns = next team start(now, self.K)
            self.log(now, f"swap 时间不足 -> 推迟 swap 到 {ns:.2f}h （bay {bay.id}）")
            self.sim.schedule(ns, lambda now2, inc=incoming dev, b=bay: self. start swap(now2, i
nc, b),
                              desc=f"deferred swap bay{bay.id}")
```

```
                return
            if self.transport_capacity <= 0:
                self.transport_queue.append((incoming_dev, bay, True))
                self.log(now, f"swap 暂无 transport，入队 bay {bay.id}")
                return
            self.transport_capacity -= 1
            self.log(now, f"bay {bay.id} 开始 swap（新设备装置 {incoming_dev['id']} 输入）")
            end = now + transport_time
            def on_swap_end(et, inc=incoming_dev, b=bay):
                self.transport_capacity += 1
                old = b.current_dev
                if old is not None:
                    self.log(et, f"bay {b.id} 旧装置 {old['id']} 已运出（swap）")
                b.current_dev = inc
                b.reserved_incoming = None
                b.place_taken = True
                self.log(et, f"bay {b.id} 新设备 {inc['id']} 已就位（swap 完成）")
                self._start_device_flow(inc, b, et)
            self.sim.schedule(end, on_swap_end, desc=f"swap_end bay{bay.id} inc{incoming_dev['id']}
")

        # ---------- 请求运出（优先 swap） ----------
        def request_transport_out(self, dev, bay, t):
            # 先触发 bay_manager 以尽可能提前预约下一台装置
            self._bay_manager_step(t)
            if bay.reserved_incoming:
                incoming = bay.reserved_incoming
                # 若 swap 时间不足，则推到下个小队（避免跨小队分段运输）
                rem = team_time_remaining(t, self.K)
                if transport_time > rem:
                    ns = next_team_start(t, self.K)
                    self.log(t, f"swap 时间不足 -> 推迟 swap 到 {ns:.2f}h （bay {bay.id}）")
                    self.sim.schedule(ns, lambda now2, inc=incoming, b=bay: self._start_swap(now2, i
nc, b),
                                      desc=f"deferred_swap_at_out bay{bay.id}")
                    return
                if self.transport_capacity > 0:
                    self.sim.schedule(t, lambda now, inc=incoming, b=bay: self._start_swap(now, inc,
b),
                                      desc=f"start_swap_at_out bay{bay.id} inc{incoming['id']}")
                else:
                    self.transport_queue.append((incoming, bay, True))
                    self.log(t, f"swap 入队（等待 transport 空位） bay {bay.id} incoming {incomin
g['id']}")
                return

            # 普通 out（先检查是否能在当前小队内完成 transport）
            rem = team_time_remaining(t, self.K)
            if transport_time > rem:
                ns = next_team_start(t, self.K)
                self.log(t, f"transport_out 时间不足（剩余 {rem:.2f}h） -> 推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now, d=dev, b=bay: self._request_transport_out(d, b, no
w),
                                  desc=f"deferred_transport_out dev{dev['id']} bay{bay.id}")
                return

            def start_out(now, d=dev, b=bay):
                if self.transport_capacity > 0:
                    self.transport_capacity -= 1
                    self.log(now, f"装置 {d['id']} 开始普通运出 bay {b.id}（transport 占用）")
                    end = now + transport_time
                    def on_out_end(et, d=d, b=b):
                        self.transport_capacity += 1
                        b.place_taken = False
                        b.current_dev = None
                        self.log(et, f"装置 {d['id']} 已运出 bay {b.id}")
                        self.completion_times.append(et)
                        if b.reserved_incoming:
                            self._request_transport_in_for_bay(b.reserved_incoming, b, et)
                        self._drain_pending_and_try(et)
```

```
                        self.sim.schedule(et, self. bay manager step, desc="bay manager trigger after
 out")
                    self.sim.schedule(end, on out end, desc=f"transport out end dev{d['id']} bay{b.id}
")
                else:
                    self.transport queue.append((dev, bay, False))
                    self.log(now, f"transport 满，装置 {dev['id']} 运出排队 bay {bay.id}")

            self.sim.schedule(t, start out, desc=f"start transport out dev{dev['id']} bay{bay.id}")

        # ---------- 装置在 bay 的测试流程入口 ----------
        def  start device flow(self, dev, bay, t):
            self.log(t, f"装置 {dev['id']} 进入 bay {bay.id}，准备 A/B/C 首次尝试")
            dev state = {'dev': dev,
                         'bay': bay,
                         'attempt counts': {'A': 0, 'B': 0, 'C': 0, 'E': 0},
                         'results': {'A': None, 'B': None, 'C': None},
                         'eliminated': False}
            for g in ['A', 'B', 'C']:
                self. initiate subtest attempt(dev state, g, t)

        # ---------- 首次校准（若需要），遵循 K 小队时长约束 ----------
        def  do calibration if needed(self, g, dev state, now, on done):
            eq = self.equip[g]
            if eq.calibrated:
                on done(now)
                return
            rem = team time remaining(now, self.K)
            # 若当前小队剩余时间不足以完成校准，则把校准推到下一个小队
            if calib time[g] > rem:
                ns = next team start(now, self.K)
                self.log(now, f"{g} 首次校准推迟到 {ns:.2f}h（小队剩余 {rem:.2f}h 不足）")
                self.sim.schedule(ns, lambda now2: self. do calibration if needed(g, dev state, now2,
on done),
                                  desc=f"deferred initial calib {g} dev{dev state['dev']['id']}")
                return

            def start calib(now2, g local=g, ds=dev state):
                if self.groups[g local].available <= 0:
                    # 若组工位不可得，入队并稍后重试
                    self.pending attempts.append(lambda now3, g local=g local, ds=ds: start calib(no
w3, g local, ds))
                    return
                # 占用组工位
                self.groups[g local].request(lambda now3: self. pending attempt retry(now3))
                eq local = self.equip[g local]
                if not eq local.calibrated:
                    eq local.calibrated = True
                    self.log(now2, f"{g_local} 首次校准开始（{calib_time[g_local]:.2f}h）")
                    eq local.total calib time += calib time[g local]
                    end = now2 + calib time[g local]
                else:
                    end = now2

                def on end(ct, g local=g local):
                    self.groups[g local].release(self.sim, ct)
                    self. drain pending and try(ct)
                    self.log(ct, f"{g_local} 首次校准完成")
                    on done(ct)

                self.sim.schedule(end, on end, desc=f"initial calib end {g local} dev{dev state['dev']['i
d']}")

            self.sim.schedule(now, start calib, desc=f"try initial calib {g} dev{dev state['dev']['id']}")

        # ---------- 子测试发起（若当前小队剩余时间不足则推到下个小队） ----------
        def  initiate subtest attempt(self, dev state, group, t):
            dev = dev state['dev']
            rem = team time remaining(t, self.K)
            if test time[group] > rem:
```

```
                ns = next team start(t, self.K)
                self.log(t, f"当前小队剩余时间不足以完成 {group} 测试（剩余 {rem:.2f}h），推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now, ds=dev state, g=group: self. initiate subtest attempt(ds, g, now),
                                  desc=f"deferred subtest {group} dev{dev['id']}")
                return

            def after calib(now after):
                self.sim.schedule(now after,
                                  lambda now2, ds=dev state, g=group: self. attempt allocate and execute(now2, ds, g),
                                  desc=f"post calib try {group} dev{dev['id']}")
            self. do calibration if needed(group, dev state, t, after calib)

        # ---------- 执行一次子测试（必须能在当前小队内一次完成） ----------
        def  attempt allocate and execute(self, now, dev state, group):
            dev = dev state['dev']
            bay = dev state['bay']
            rem = team time remaining(now, self.K)
            if test time[group] > rem:
                ns = next team start(now, self.K)
                self.log(now, f"[重检查] 本小队时间不足以执行 {group}（剩余 {rem:.2f}h），推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now2, ds=dev state, g=group: self. initiate subtest attempt(ds, g, now2),
                                  desc=f"deferred check subtest {group} dev{dev['id']}")
                return
            if self.groups[group].available <= 0 or bay.slots used >= bay.slots capacity:
                self.pending attempts.append(
                    lambda now inner, ds=dev state, g=group: self. attempt allocate and execute(now inner, ds, g))
                return
            # 占用组工位与 bay slot
            self.groups[group].request(lambda now inner: self. pending attempt retry(now inner))
            bay.slots used += 1
            self.log(now, f"bay {bay.id} 占用 {group}，开始一次尝试（usage={self.equip[group].usage:.2f}h）")
            eq = self.equip[group]
            p fail = eq.prob fail during(test time[group])
            if random.random() < p fail:
                # 故障将在测试内部某点发生（只允许当整段测试可被当前小队承担）
                frac = random.random()
                fail work = test time[group] * frac
                fail time = now + fail work
                def on fail(ft, ds=dev state, g=group, fw=fail work):
                    eq.advance(fw)
                    self.log(ft, f"bay {bay.id} {g} 测试中故障 -> 替换+校准")
                    eq.replace()
                    # 替换需要完整校准时间，若当前小队剩余时间不足则把替换推到下一小队
                    rem2 = team time remaining(ft, self.K)
                    if calib time[g] > rem2:
                        ns = next team start(ft, self.K)
                        self.log(ft, f"{g} 替换校准时间不足 -> 推迟校准到 {ns:.2f}h")
                        self.sim.schedule(ns, lambda now2, ds=ds, g=g: self. on attempt aborted(now2, ds, g),
                                          desc=f"deferred calib after fail {g} dev{ds['dev']['id']}")
                        # 释放资源（按你原来的 V4 逻辑，故障后会释放组与 slot）
                        self.groups[g].release(self.sim, ft)
                        bay.slots used -= 1
                        return
                    else:
                        # 在当前小队内完成替换校准（在替换后安排重试）
                        def after calib(ct):
                            self.groups[g].release(self.sim, ct)
                            bay.slots used -= 1
                            self.sim.schedule(ct, lambda now2, ds=ds, g=g: self. on attempt aborted(now2, ds, g),
                                              desc=f"retry after aborted {g} dev{ds['dev']['id']}")
```

```
                    self.sim.schedule(ft, after calib, desc=f"calib end after fail {g} dev{ds['dev']['
id']}")
            self.sim.schedule(fail time, on fail, desc=f"fail event {group} dev{dev['id']} bay{bay.i
d}")
        else:
            end time = now + test time[group]
            def on success(et, ds=dev state, g=group):
                eq.advance(test time[g])
                self.log(et, f"bay {bay.id} 完成 {g} 测试（成功） usage={eq.usage:.2f}h")
                def release(t2):
                    self.groups[g].release(self.sim, t2)
                    bay.slots used -= 1
                    flag = self. on attempt success(t2, ds, g)
                    if not flag: # 不去遍历抢占池，把当前的遍历完（通过或淘汰）
                        pass
                    else: # 寻找抢占池
                        self. drain pending and try(t2)
                self. handle success and maybe replace(eq, g, et, release, bay, ds)
            self.sim.schedule(end time, on success, desc=f"success event {group} dev{dev['id']} b
ay{bay.id}")

    # ---------- 后置替换处理（保持原来行为） ----------
    def  handle success and maybe replace(self, eq, group, end time, release, bay, dev state):
        if eq.usage >= self.replace at[group] and eq.usage >= eq.min run:
            self.log(end time, f"bay {bay.id} {group} 测试中达到预设上限{self.replace_at[group]}
h Usage:{eq.usage}h -> 替换")
            eq.replace()
            # 替换所需校准若不能在当前小队完成则推到下个小队
            rem = team time remaining(end time, self.K)
            if calib time[group] > rem:
                eq.calibrated = True
                ns = next team start(end time, self.K) + calib time[group]
                self.log(end_time, f"{group} post-replace 校准推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now2: release(now2), desc=f"deferred post calib {gr
oup}")
            else:
                ct = end time + calib time[group]
                eq.total calib time += calib time[group]
                eq.calibrated = True
                self.log(end_time, f"{group} 替换后首次校准开始（{calib_time[group]:.2f}h）")
                self.sim.schedule(ct, lambda now2: release(now2), desc=f"post calib end {group}
")
        elif eq.usage >= 240.0:
            self.log(end_time, f"bay {bay.id} {group} 使用达到 240h -> 强制替换")
            eq.replace()
            rem = team time remaining(end time, self.K)
            if calib time[group] > rem:
                eq.calibrated = True
                ns = next team start(end time, self.K) + calib time[group]
                self.log(end_time, f"{group} 强制校准推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now2: release(now2), desc=f"deferred force calib {gr
oup}")
            else:
                ct = end time + calib time[group]
                eq.calibrated = True
                eq.total calib time += calib time[group]
                self.sim.schedule(ct, lambda now2: release(now2), desc=f"force calib end {group}
")
        else:
            release(end time)

    # ---------- pending retry ----------
    def  pending attempt retry(self, now):
        self. drain pending and try(now)

    def  drain pending and try(self, now):
        max iter = max(50, len(self.pending attempts))
        for    in range(min(max iter, len(self.pending attempts))):
            cb = self.pending attempts.popleft()
            cb(now)
```

```
    # ---------- 中断重试 ----------
    def  on attempt aborted(self, t, dev state, group):
        self.log(t, f"装置 {dev_state['dev']['id']} 的 {group} 尝试被中断或延后，安排重试")
        self.sim.schedule(t, lambda now, ds=dev state, g=group: self. initiate subtest attempt(ds, g,
now),
                          desc=f"retry after aborted {group} dev{dev state['dev']['id']}")

    # ---------- 成功判定 ----------
    def  on attempt success(self, t, dev state, group):
        dev = dev state['dev']
        bay = dev state['bay']
        g = group
        self.tests count[g] += 1
        if dev['q' + g]:
            if random.random() < leak[g]:
                dev state['results'][g] = 'pass hidden'
                self.log(t, f"装置 {dev['id']} 在 {g} 漏判（隐含通过）")
            else:
                dev state['attempt counts'][g] += 1
                self.log(t, f"装置 {dev['id']} 在 {g} 被检出 (attempts={dev_state['attempt_counts']
[g]})")
                if dev state['attempt counts'][g] >= 2:
                    self.log(t, f"装置 {dev['id']} 在 {g} 连续两次失败 -> 淘汰")
                    self. elim device(dev state, t)
                    return True # True 代表后续遍历抢占任务池
                else:
                    self.sim.schedule(t, lambda now, ds=dev state, g=g: self. initiate subtest atte
mpt(ds, g, now),
                                      desc=f"retry detected {g} dev{dev['id']}")
                    return False # False 代表后续不遍历抢占任务池
        else:
            if random.random() < e[g] * 0.5:
                self.false pos += 1
                dev state['attempt counts'][g] += 1
                self.log(t, f"装置 {dev['id']} 在 {g} 误判 (attempts={dev_state['attempt_counts']
[g]})")
                if dev state['attempt counts'][g] >= 2:
                    self.log(t, f"装置 {dev['id']} 在 {g} 误判两次 -> 淘汰")
                    self. elim device(dev state, t)
                    return True
                else:
                    self.sim.schedule(t, lambda now, ds=dev state, g=g: self. initiate subtest atte
mpt(ds, g, now),
                                      desc=f"retry mis {g} dev{dev['id']}")
                    return False
            else:

                dev state['results'][g] = 'pass'
                self.log(t, f"装置 {dev['id']} 在 {g} 判定通过")
        all done = all(dev state['results'].get(x) in ('pass', 'pass hidden') for x in ['A', 'B', 'C'])
        if all done:
            self.log(t, f"装置 {dev['id']} 三项子测试完成，准备综合测试 E")
            self. do calibration if needed('E', dev state, t,
                                           lambda now, ds=dev state: self. start comprehensi
ve(ds, now))
            return True
        else:
            return True

    # ---------- 综合测试 E ----------
    def  start comprehensive(self, dev state, t):
        dev = dev state['dev']
        bay = dev state['bay']
        rem = team time remaining(t, self.K)
        if test time['E'] > rem:
            ns = next team start(t, self.K)
            self.log(t, f"装置 {dev['id']} E 测试本小队时间不足（剩余 {rem:.2f}h），推迟到 {n
s:.2f}h")
            self.sim.schedule(ns, lambda now, ds=dev state: self. start comprehensive(ds, now),
```

```
                                        desc=f"deferred E dev{dev['id']}")
                return
            self.log(t, f"装置 {dev['id']} E 测试开始一次尝试 （usage={self.equip['E'].usage:.2f}h）" )
            def start E(now):
                if self.groups['E'].available <= 0 or bay.slots used >= bay.slots capacity:
                    # 修改
                    self.pending attempts.append(lambda now inner, ds = dev state: self. start compr
ehensive(ds, now inner))
                    return
                self.groups['E'].request(lambda now inner: self. pending attempt retry(now inner))
                bay.slots used += 1
                eq = self.equip['E']
                p fail = eq.prob fail during(test time['E'])
                if random.random() < p fail:
                    frac = random.random()
                    fail work = test time['E'] * frac
                    fail time = now + fail work
                    def on fail(ft):
                        eq.advance(fail work)
                        self.log(ft, f"E 在测试中故障 -> 替换并校准")
                        eq.replace()
                        eq.total calib time += calib time['E']
                        rem2 = team time remaining(ft, self.K)
                        if calib time['E'] > rem2:
                            ns = next team start(ft, self.K)
                            self.log(ft, f"E 故障后校准时间不足 -> 推迟到 {ns:.2f}h")
                            self.sim.schedule(ns, lambda now2, ds=dev state: self. start comprehensi
ve(ds, now2),
                                                desc=f"deferred E retry after fail dev{dev['id']}")
                            self.groups['E'].release(self.sim, ft)
                            bay.slots used -= 1
                            # self. drain pending and try(ft)
                            return
                        else:
                            calib end = ft + calib time['E']
                            def after calib(ct):
                                self.groups['E'].release(self.sim, ct)
                                bay.slots used -= 1
                                # self. drain pending and try(ct)
                                self.sim.schedule(ct, lambda now2, ds=dev state: self. start compre
hensive(ds, now2),
                                                    desc=f"retry E after fail dev{dev['id']}")
                            self.sim.schedule(calib end, after calib, desc=f"E calib end after fail de
v{dev['id']}")
                    self.sim.schedule(fail time, on fail, desc=f"E fail dev{dev['id']}")
                else:
                    end time = now + test time['E']
                    def on success(et):
                        eq.advance(test time['E'])
                        self.tests count['E'] += 1
                        def release and handle(t2):
                            self.groups['E'].release(self.sim, t2)
                            bay.slots used -= 1
                            flag = self. handle E result(t2, dev state)

                            if not flag: # 不去遍历抢占池，把当前的遍历完（通过或淘汰）
                                pass
                            else: # 寻找抢占池
                                self. drain pending and try(t2)
                        self. handle success and maybe replace(eq, 'E', et, release and handle, bay,
dev state)
                    self.sim.schedule(end time, on success, desc=f"E success dev{dev['id']}")
            self.sim.schedule(t, start E, desc=f"try start E dev{dev['id']}")

        def  handle E result(self, t, dev state):
            dev = dev state['dev']
            bay = dev state['bay']

            hiddenA = (dev state['results'].get('A') == 'pass hidden')
            hiddenB = (dev state['results'].get('B') == 'pass hidden')
            hiddenC = (dev state['results'].get('C') == 'pass hidden')
```

```
        actual_problem = hiddenA or hiddenB or hiddenC or dev['qD']

        # 情形1：真实存在问题（实际故障）
        if actual_problem:
            # 漏判（检测不到真实问题） -> 计入漏判统计并直接判为通过（即漏判视为“通过”）
            if random.random() < leak['E']:
                self.false_neg += 1
                self.log(t, f"装置 {dev['id']} E 漏判实际问题 -> 判为通过（漏判）")
                self._finalize_pass(t, dev, state)
                return True
            else:
                # 正确检出真实问题 -> 计为 E 的一次失败尝试（与 A/B/C 处理保持一致）
                dev_state['attempt_counts']['E'] += 1
                cnt = dev_state['attempt_counts']['E']
                self.log(t, f"装置 {dev['id']} E 检出问题（attempts_E={cnt}）")
                if cnt >= 2:
                    # 连续两次不合格 -> 淘汰
                    self.log(t, f"装置 {dev['id']} 在 E 连续两次不合格 -> 淘汰")
                    self._elim_device(dev, state, t)
                    return True
                else:
                    # 否则安排重测 E（不回退任何子系统）
                    self.sim.schedule(t, lambda now, ds=dev_state: self._start_comprehensive(ds, now),
                                      desc=f"retest E after detect dev{dev['id']}")
                    return False

        # 情形2：真实无问题（设备本身正常）
        else:
            # 误判（将良品判为问题） -> 计为误判统计，并当作 E 的一次失败尝试
            if random.random() < e['E'] * 0.5:
                self.false_pos += 1
                dev_state['attempt_counts']['E'] += 1
                cnt = dev_state['attempt_counts']['E']
                self.log(t, f"装置 {dev['id']} E 误判（attempts_E={cnt}）")
                if cnt >= 2:
                    # 连续两次误判 -> 淘汰
                    self.log(t, f"装置 {dev['id']} 在 E 误判连续两次 -> 淘汰")
                    self._elim_device(dev, state, t)
                    return True
                else:
                    # 否则重测 E
                    self.sim.schedule(t, lambda now, ds=dev_state: self._start_comprehensive(ds, now),
                                      desc=f"retest E after falsepos dev{dev['id']}")
                    return False
            # 正常通过
            self._finalize_pass(t, dev, state)
            return True

    def _finalize_pass(self, t, dev, state):
        dev = dev_state['dev']
        bay = dev_state['bay']
        self.passed += 1
        self.log(t, f"装置 {dev['id']} E 通过，准备运出（优先 swap）")
        if bay.reserved_incoming:
            incoming = bay.reserved_incoming
            # swap 也需满足当前小队时间
            rem = team_time_remaining(t, self.K)
            if transport_time > rem:
                ns = next_team_start(t, self.K)
                self.log(t, f"swap 时间不足 -> 推迟到 {ns:.2f}h")
                self.sim.schedule(ns, lambda now, inc=incoming, b=bay: self._start_swap(now, inc, b),
                                  desc=f"deferred_direct_swap bay{bay.id}")
                return
            if self.transport_capacity > 0:
```

```
                    self.sim.schedule(t, lambda now, inc=incoming, b=bay: self. start swap(now, inc,
b),
                                      desc=f"direct swap after pass bay{bay.id}")
               else:
                    self.transport queue.append((incoming, bay, True))
                    self.log(t, f"swap 排队（等待 transport） bay {bay.id} incoming {incoming['id']}
")
          else:
               self. request transport out(dev, bay, t)

     # 淘汰设备
     def  elim device(self, dev state, t):
          if dev state.get('eliminated', False):
               return
          dev = dev state['dev']
          bay = dev state['bay']
          dev state['eliminated'] = True
          self.rejected += 1
          self.log(t, f"装置 {dev['id']} 淘汰 -> 运出（优先 swap）")
          if bay.reserved incoming:
               incoming = bay.reserved incoming
               rem = team time remaining(t, self.K)
               if transport time > rem:
                    ns = next team start(t, self.K)
                    self.log(t, f"swap 推迟到 {ns:.2f}h（淘汰时）")
                    self.sim.schedule(ns, lambda now, inc=incoming, b=bay: self. start swap(now, inc,
 b),
                                      desc=f"deferred direct swap reject bay{bay.id}")
                    return
               if self.transport capacity > 0:
                    self.sim.schedule(t, lambda now, inc=incoming, b=bay: self. start swap(now, inc,
b),
                                      desc=f"direct swap after reject bay{bay.id}")
               else:
                    self.transport queue.append((incoming, bay, True))
                    self.log(t, f"swap 排队（淘汰时） bay {bay.id} incoming {incoming['id']}")
          else:
               self. request transport out(dev, bay, t)

     # ---------- 汇总 ----------
     def summarize(self):
          makespan = max(self.completion times)  - max(calib time.values()) - 2 * transport time if
 self.completion times else 0.0
          days = int(makespan / 24.0)+1 if makespan > 0 else 0.0
          shifts = max(1, int(makespan / self.K)+1)
          group work = {g: self.tests count[g] * test time[g] for g in self.tests count}
          yxb = {g: (group work[g] / (shifts * self.K)) for g in group work}
          return {
               'makespan hours': makespan,
               'makespan days': days,
               'passed': self.passed,
               'rejected': self.rejected,
               'false neg': self.false neg,
               'false pos': self.false pos,
               'tests count': self.tests count,
               'yxb': yxb
          }

# ---------------- 运行示例 ----------------
def demo run v4 K():
     sim = TestHallEventV4 K(n devices=N DEVICES, replace at=REPLACE AT, bay slots=BAY
SLOTS, seed=RANDOM SEED, verbose=True, K=K WORK HOURS)
     sim.start()
     summary = sim.summarize()
     print("\n=== V4-K 仿真结束，统计摘要 ===")
     for k, v in summary.items():
          print(f"{k}: {v}")
     return summary

def run multiple simulations(n simulations=100, n devices=100, replace at=None, seed base=RAND
OM SEED, verbose=False, k work hours=9.0):
```

```
    """
    多次运行仿真，计算平均统计指标（蒙特卡洛方法）

    参数:
        n_simulations: 模拟次数
        n_devices: 每次模拟的装置数量
        replace_at: 设备更换时间阈值
        seed_base: 随机种子基础值，每次运行使用不同的种子以确保结果独立性
        verbose: 是否显示每次运行的详细日志
    """
    all_results = {
        'makespan_days': [],
        'passed': [],
        'rejected': [],
        'false_neg': [],
        'false_pos': [],
        'yxb_A': [],
        'yxb_B': [],
        'yxb_C': [],
        'yxb_E': []
    }

    for i in range(n_simulations):
        # 使用不同的种子确保每次模拟的随机性独立
        current_seed = seed_base + i
        # print(f"\n--- 开始第 {i + 1}/{n_simulations} 次模拟 (Seed: {current_seed}) ---")

        # 创建并运行仿真
        sim = TestHallEventV4_K(
            n_devices=n_devices,
            replace_at=replace_at,
            bay_slots=BAY_SLOTS,
            seed=current_seed,
            verbose=verbose,  # 通常在外层循环中关闭详细日志以避免输出过多
            K=k_work_hours
        )
        sim.start()
        summary = sim.summarize()

        # 收集结果
        all_results['makespan_days'].append(summary['makespan_days'])
        all_results['passed'].append(summary['passed'])
        all_results['rejected'].append(summary['rejected'])
        all_results['false_neg'].append(summary['false_neg'])
        all_results['false_pos'].append(summary['false_pos'])
        all_results['yxb_A'].append(summary['yxb']['A'])
        all_results['yxb_B'].append(summary['yxb']['B'])
        all_results['yxb_C'].append(summary['yxb']['C'])
        all_results['yxb_E'].append(summary['yxb']['E'])

    # 计算平均值和标准差
    avg_results = {}
    std_results = {}
    for key, value_list in all_results.items():
        avg_results[key] = sum(value_list) / len(value_list)
        std_results[key] = (sum((x - avg_results[key]) ** 2 for x in value_list) / len(value_list)) *
* 0.5

    # print(f"=== 蒙特卡洛模拟完成 ({n_simulations} 次) ===")
    # print("平均统计指标:")
    # for k, v in avg_results.items():
    #     print(f"  {k}: {v:.4f}")
    # print("\n标准差:")
    # for k, v in std_results.items():
    #     print(f"  {k}: {v:.4f}")

    return avg_results, all_results


def Q3resultlog(result):
```

```
    # 打印结果表
    print("\n" + "=" * 60)
    print("问题3结果统计指标")
    print("=" * 60)
    print(f"{'指标':<10} {'值':<15} {'描述':<30}")
    print("-" * 60)
    print(f"{'T':<10} {result['T']:<15.2f} {'任务完成平均天数':<30}")
    print(f"{'S':<10} {result['S']:<15.2f} {'通过测试的装置平均数目':<30}")
    print(f"{'PL':<10} {result['PL']:<15.6f} {'总漏判概率':<30}")
    print(f"{'PW':<10} {result['PW']:<15.6f} {'总误判概率':<30}")
    print(f"{'YXB1':<10} {result['YXB1']:<15.3f} {'小组A有效工作时间比':<30}")
    print(f"{'YXB2':<10} {result['YXB2']:<15.3f} {'小组B有效工作时间比':<30}")
    print(f"{'YXB3':<10} {result['YXB3']:<15.3f} {'小组C有效工作时间比':<30}")
    print(f"{'YXB4':<10} {result['YXB4']:<15.3f} {'小组E有效工作时间比':<30}")
    print("=" * 60)

if __name__ == "__main__":
    demo_run_v4_K()
    n_runs = 100
    replace_at = {'A': 180, 'B': 140, 'C': 150, 'E': 160}
    # 运行蒙特卡洛模拟
    avg_stats, all_stats = run_multiple_simulations(n_simulations=n_runs, n_devices=N_DEVICES, r
eplace_at=replace_at, verbose=False, k_work_hours=K_WORK_HOURS)

    # 创建结果字典
    Q3_results = {
        'T': avg_stats['makespan_days'],
        'S': avg_stats['passed'],
        'PL': avg_stats['false_neg'],
        'PW': avg_stats['false_pos'],
        'YXB1': avg_stats['yxb_A'],
        'YXB2': avg_stats['yxb_B'],
        'YXB3': avg_stats['yxb_C'],
        'YXB4': avg_stats['yxb_E']
    }
    Q3resultlog(Q3_results)
```

代码清单 2 问题 2 和问题 3 的进化算法

```
    import numpy as np
    import time
    from tqdm import tqdm
    from pymoo.algorithms.moo.nsga2 import NSGA2
    from pymoo.core.problem import Problem
    from pymoo.optimize import minimize
    from pymoo.operators.crossover.sbx import SBX
    from pymoo.operators.mutation.pm import PM
    from pymoo.operators.sampling.rnd import FloatRandomSampling
    from pymoo.algorithms.moo.moead import MOEAD
    from pymoo.decomposition.tchebicheff import Tchebicheff
    from pymoo.util.reference_direction import UniformReferenceDirectionFactory



    class DiscreteSampling(FloatRandomSampling):
        def __init__(self, include_k=True, theta_step=10 / 60):
            super().__init__()
            self.include_k = include_k
            self.theta_step = theta_step

        def _do(self, problem, n_samples, **kwargs):
```

```
        X = super()._do(problem, n_samples, **kwargs)

        # 对 θ 做步长限制
        X[:, :4] = np.round(X[:, :4] / self.theta_step) * self.theta_step

        if self.include_k and X.shape[1] >= 5:
            k_values = X[:, 4]
            discrete_k = np.round(k_values * 2) / 2
            discrete_k = np.clip(discrete_k, problem.xl[4], problem.xu[4])
            X[:, 4] = discrete_k

        return X


class DiscreteCrossover(SBX):
    """离散交叉算子，处理k值的0.5步长"""

    def __init__(self, prob=0.9, eta=15, include_k=True, theta_step=10 / 60):
        super().__init__(prob=prob, eta=eta)
        self.include_k = include_k
        self.theta_step = theta_step

    def _do(self, problem, X, **kwargs):
        # 先进行连续交叉
        X_crossover = super()._do(problem, X, **kwargs)
        # 假设 crossover / mutation 类增加了 theta_step 参数
        X_crossover[:, :4] = np.round(X_crossover[:, :4] / self.theta_step) * self.theta_step

        if self.include_k and X_crossover.ndim == 2 and X_crossover.shape[1] >= 5:
            # 对k值进行离散化处理（第5列）
            k_values = X_crossover[:, 4]  # 获取所有k值
            # 四舍五入到最近的0.5
            discrete_k = np.round(k_values * 2) / 2
            # 限制在有效范围内
            discrete_k = np.clip(discrete_k, problem.xl[4], problem.xu[4])
            # 赋值回数组
            X_crossover[:, 4] = discrete_k

        return X_crossover


class DiscreteMutation(PM):
    """离散变异算子，处理k值的0.5步长"""

    def __init__(self, eta=20, include_k=True, theta_step=10 / 60):
        super().__init__(eta=eta)
        self.include_k = include_k
        self.theta_step = theta_step

    def _do(self, problem, X, **kwargs):
```

```
        # 先进行连续变异

        X_mutated = super()._do(problem, X, **kwargs)
        X_mutated[:, :4] = np.round(X_mutated[:, :4] / self.theta_step) * self.theta_step

        if self.include_k and X_mutated.ndim == 2 and X_mutated.shape[1] >= 5:
            # 对k值进行离散化处理（第5列）
            k_values = X_mutated[:, 4]  # 获取所有k值
            # 四舍五入到最近的0.5
            discrete_k = np.round(k_values * 2) / 2

            # 限制在有效范围内
            discrete_k = np.clip(discrete_k, problem.xl[4], problem.xu[4])

            # 赋值回数组
            X_mutated[:, 4] = discrete_k

        return X_mutated


class DeviceTestingProblem(Problem):
    """优化问题定义：4维或5维优化"""

    def __init__(self, n_devices, n_runs=10, include_k=True, simulation_func=None):
        self.include_k = include_k

        if include_k:
            # 5维：4个theta + 1个k
            n_var = 5
            xl = np.array([120, 120, 120, 120, 9.0])
            xu = np.array([240, 240, 240, 240, 12.0])
        else:
            # 4维：4个theta
            n_var = 4
            xl = np.array([120, 120, 120, 120])
            xu = np.array([240, 240, 240, 240])

        super().__init__(n_var=n_var, n_obj=2, n_constr=0, xl=xl, xu=xu)
        self.n_devices = n_devices
        self.n_runs = n_runs
        self.simulation_func = simulation_func
        self.evaluation_cache = {}
        self.evaluation_count = 0
        self.total_evaluations = 0
        self.start_time = time.time()

        if include_k:
            self.k_values = [9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0]
            print(f"k值的可能取值: {self.k_values}")
```

```
        self.all_stats_dict = {}  # 存储所有详细统计数据
        self.all_evaluated_offspring = []  # 保存所有被评估的个体
        self.current_generation = None  # 由外部 callback 设置

    def _validate_parameters(self, theta_A, theta_B, theta_C, theta_E, k=None):
        """验证参数是否在有效范围内，并对齐到步长"""
        step = 10 / 60  # 10分钟 = 0.1667 小时
        valid_theta_A = round(max(self.xl[0], min(self.xu[0], theta_A)) / step) * step
        valid_theta_B = round(max(self.xl[1], min(self.xu[1], theta_B)) / step) * step
        valid_theta_C = round(max(self.xl[2], min(self.xu[2], theta_C)) / step) * step
        valid_theta_E = round(max(self.xl[3], min(self.xu[3], theta_E)) / step) * step

        if self.include_k and k is not None:
            valid_k = round(k * 2) / 2
            valid_k = max(self.xl[4], min(self.xu[4], valid_k))
            return valid_theta_A, valid_theta_B, valid_theta_C, valid_theta_E, valid_k
        else:
            return valid_theta_A, valid_theta_B, valid_theta_C, valid_theta_E

    def _evaluate(self, X, out, *args, **kwargs):
        F = np.full((X.shape[0], 2), np.inf)

        with tqdm(total=X.shape[0], desc="评估种群", unit="个体", ncols=100) as pbar:
            for i in range(X.shape[0]):
                theta_A = float(X[i, 0])
                theta_B = float(X[i, 1])
                theta_C = float(X[i, 2])
                theta_E = float(X[i, 3])
                k = float(X[i, 4]) if self.include_k else None

                # 验证和调整参数
                if self.include_k:
                    theta_A, theta_B, theta_C, theta_E, k = self._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E, k
                    )
                    key = (theta_A, theta_B, theta_C, theta_E, k)
                else:
                    theta_A, theta_B, theta_C, theta_E = self._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E
                    )
                    key = (theta_A, theta_B, theta_C, theta_E)

                try:
                    if key in self.evaluation_cache:
                        # 已评估过 → 用缓存
                        F[i] = self.evaluation_cache[key]
                        stats = self.all_stats_dict.get(key, {})
                    else:
                        # 新个体 → 调用仿真函数
                        if self.simulation_func is None:
```

```
                        raise NotImplementedError("请提供simulation_func参数")

                    T, PL, stats = self.simulation_func(
                        theta_A=theta_A,
                        theta_B=theta_B,
                        theta_C=theta_C,
                        theta_E=theta_E,
                        k=k if self.include_k else None,
                        n_runs=self.n_runs
                    )

                    F[i] = [T, PL]
                    self.evaluation_cache[key] = F[i]
                    self.all_stats_dict[key] = stats

                # ===== ◈关键：无论新旧解，都保存 =====
                self.all_evaluated_offspring.append({
                    'theta_A': theta_A,
                    'theta_B': theta_B,
                    'theta_C': theta_C,
                    'theta_E': theta_E,
                    'k': k if self.include_k else None,
                    'T': float(F[i][0]),
                    'PL': float(F[i][1]),
                    'stats': stats,
                    'generation': self.current_generation
                })
                self.evaluation_count += 1

            except Exception as e:
                print(f"\n评估失败: {e}")
                import traceback;
                traceback.print_exc()
                F[i] = [np.inf, np.inf]
                self.all_stats_dict[key] = {'makespan_days': [], 'false_neg': []}

            pbar.update(1)
            pbar.set_postfix({
                'theta_A': f'{theta_A:.1f}',
                'theta_B': f'{theta_B:.1f}',
                'theta_C': f'{theta_C:.1f}',
                'theta_E': f'{theta_E:.1f}',
                'k': f'{k:.1f}' if self.include_k else 'N/A',
                'T': f'{F[i][0]:.2f}' if F[i][0] != np.inf else 'inf',
                'PL': f'{F[i][1]:.6f}' if F[i][1] != np.inf else 'inf',
                '评估数': self.evaluation_count
            })

        out["F"] = F
```

```
class OptimizationProgress:
    """自定义进度跟踪器"""

    def __init__(self, n_gen, include_k=True):

        self.n_gen = n_gen
        self.include_k = include_k
        self.start_time = time.time()
        self.best_T_history = []
        self.best_PL_history = []

    def update(self, algorithm, gen):

        valid_solutions = [ind for ind in algorithm.pop if not np.any(np.isinf(ind.F))]
        if valid_solutions:
            best_T = min(ind.F[0] for ind in valid_solutions)
            best_PL = min(ind.F[1] for ind in valid_solutions)
            self.best_T_history.append(best_T)
            self.best_PL_history.append(best_PL)
            elapsed = time.time() - self.start_time
            progress = gen / self.n_gen * 100
            print(f"\n=== 代数 {gen}/{self.n_gen} ({progress:.1f}%) ===")
            print(f"最佳 T: {best_T:.2f} 天")
            print(f"最佳 PL: {best_PL:.6f}")
            print(f"已用时间: {elapsed:.1f}s")
            print(f"总评估次数: {len(algorithm.problem.evaluation_cache)}")

            if self.include_k:
                k_values = [round(ind.X[4] * 2) / 2 for ind in valid_solutions]
                unique_k = sorted(set(k_values))
                print(f"当前k值分布: {unique_k}")

            if gen > 1:
                time_per_gen = elapsed / gen
                remaining_time = time_per_gen * (self.n_gen - gen)
                print(f"预计剩余: {remaining_time:.1f}s")

            print("=" * 40)


def run_nsga2_optimization(n_devices, n_gen=50, pop_size=40, n_runs=10,

                           include_k=True, simulation_func=None):
    """运行NSGA-II优化"""

    print("================NSGA-II优化==============")

    print(f"维度: {'5维 (4theta + k)' if include_k else '4维 (4theta)'}")
```

```
print(f"代数: {n_gen}, 种群大小: {pop_size}, 每次评估运行次数: {n_runs}")
if include_k:
    print(f"k值步长: 0.5 (9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0)")
print(f"预计总评估次数: {n_gen * pop_size}")
print("-" * 50)

# 创建优化问题
problem = DeviceTestingProblem(
    n_devices=n_devices,
    n_runs=n_runs,
    include_k=include_k,
    simulation_func=simulation_func

)

problem.total_evaluations = n_gen * pop_size
# 配置NSGA-II算法，使用离散算子
algorithm = NSGA2(
    pop_size=pop_size,
    sampling=DiscreteSampling(include_k=include_k),
    crossover=DiscreteCrossover(prob=0.9, eta=15, include_k=include_k),
    mutation=DiscreteMutation(eta=20, include_k=include_k),
    eliminate_duplicates=True
)

# 创建进度跟踪器
progress_tracker = OptimizationProgress(n_gen, include_k=include_k)
# 运行优化
print("开始NSGA-II优化...")
start_time = time.time()
# 初始化算法
algorithm.setup(problem, seed=42)
# 运行每一代
for gen in range(1, n_gen + 1):
    algorithm.next()
    progress_tracker.update(algorithm, gen)

# 获取最终结果
res = algorithm.result()
end_time = time.time()
total_time = end_time - start_time
print("\n" + "=" * 50)
print("NSGA-II优化完成!")
print(f"总耗时: {total_time:.1f} 秒")
print(f"总评估次数: {problem.evaluation_count}")
print(f"缓存中的唯一解数量: {len(problem.evaluation_cache)}")

# 获取所有有效解
valid_solutions = [ind for ind in res.pop if not np.any(np.isinf(ind.F))]
```

```
        print(f"找到的有效解数量: {len(valid_solutions)}")
        print("=" * 50)
        # 打印所有有效解的详细信息
        print("\n" + "=" * 80)
        print("所有有效解详细信息")
        print("=" * 80)
        for i, ind in enumerate(valid_solutions):
            theta_A = float(ind.X[0])
            theta_B = float(ind.X[1])
            theta_C = float(ind.X[2])
            theta_E = float(ind.X[3])

            if include_k:
                k = float(ind.X[4])
                # 验证参数
                theta_A, theta_B, theta_C, theta_E, k = problem._validate_parameters(
                    theta_A, theta_B, theta_C, theta_E, k
                )
                T = float(ind.F[0])
                PL = float(ind.F[1])
                print(f"解 {i + 1}: theta_A={theta_A:.1f}, theta_B={theta_B:.1f}, "
                      f"theta_C={theta_C:.1f}, theta_E={theta_E:.1f}, k={k:.1f}, "
                      f"T={T:.2f}天, PL={PL:.6f}")
            else:
                # 验证参数
                theta_A, theta_B, theta_C, theta_E = problem._validate_parameters(
                    theta_A, theta_B, theta_C, theta_E
                )
                T = float(ind.F[0])
                PL = float(ind.F[1])
                print(f"解 {i + 1}: theta_A={theta_A:.1f}, theta_B={theta_B:.1f}, "
                      f"theta_C={theta_C:.1f}, theta_E={theta_E:.1f}, "
                      f"T={T:.2f}天, PL={PL:.6f}")
        print("=" * 80)
        return res, problem


    def run_moead_optimization(n_devices, n_gen=50, pop_size=40, n_runs=10,
                               include_k=True, simulation_func=None):
        """运行MOEA/D优化，保存每代解"""

        print("================MOEA/D优化===============")
        print(f"维度: {'5维 (4theta + k)' if include_k else '4维 (4theta)'}")
        print(f"代数: {n_gen}, 种群大小: {pop_size}, 每次评估运行次数: {n_runs}")
        if include_k:
            print(f"k值步长: 0.5 (9.0, 9.5, 10.0, 10.5, 11.0, 11.5, 12.0)")
        print(f"预计总评估次数: {n_gen * pop_size}")
        print("-" * 50)
```

```
# 为MOEA/D生成参考方向
ref_dir_factory = UniformReferenceDirectionFactory(n_dim=2, n_points=pop_size)
ref_dirs = ref_dir_factory.do()

# 创建优化问题
problem = DeviceTestingProblem(
    n_devices=n_devices,
    n_runs=n_runs,
    include_k=include_k,
    simulation_func=simulation_func
)
problem.total_evaluations = n_gen * pop_size

# 配置MOEA/D算法
algorithm = MOEAD(
    ref_dirs=ref_dirs,
    decomposition=Tchebicheff(),
    n_neighbors=15,
    prob_neighbor_mating=0.7,
    sampling=DiscreteSampling(include_k=include_k),
    crossover=DiscreteCrossover(prob=0.9, eta=15, include_k=include_k),
    mutation=DiscreteMutation(eta=20, include_k=include_k),
    seed=42
)

# 保存每代解
all_generations = []

# 进度回调函数
def callback_func(algorithm_instance):
    current_gen = algorithm_instance.n_gen
    generation_solutions = []
    for ind in algorithm_instance.pop:
        theta_A = float(ind.X[0])
        theta_B = float(ind.X[1])
        theta_C = float(ind.X[2])
        theta_E = float(ind.X[3])
        if include_k:
            k = float(ind.X[4])
            theta_A, theta_B, theta_C, theta_E, k = problem._validate_parameters(
                theta_A, theta_B, theta_C, theta_E, k
            )
            T = float(ind.F[0])
            PL = float(ind.F[1])
            # 保存解时同时记录代数
            solution = {
                'theta_A': theta_A,
                'theta_B': theta_B,
                'theta_C': theta_C,
                'theta_E': theta_E,
```

```
                        'k': k,
                        'T': T,
                        'PL': PL,
                        'generation': current_gen  # 记录当前代数
                    }
                    generation_solutions.append(solution)
                    # generation_solutions.append((theta_A, theta_B, theta_C, theta_E, k, T, PL))
                else:
                    theta_A, theta_B, theta_C, theta_E = problem._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E
                    )
                    T = float(ind.F[0])
                    PL = float(ind.F[1])
                    # 保存解时同时记录代数
                    solution = {
                        'theta_A': theta_A,
                        'theta_B': theta_B,
                        'theta_C': theta_C,
                        'theta_E': theta_E,
                        'T': T,
                        'PL': PL,
                        'generation': current_gen  # 记录当前代数
                    }
                    generation_solutions.append(solution)
                    # generation_solutions.append((theta_A, theta_B, theta_C, theta_E, T, PL))
        all_generations.append(generation_solutions)

    # 运行优化
    print("开始MOEA/D优化...")
    start_time = time.time()
    res = minimize(
        problem,
        algorithm,
        ('n_gen', n_gen),
        verbose=False,
        seed=42,
        callback=callback_func  # 保存每代解
    )
    end_time = time.time()
    total_time = end_time - start_time

    print("\n" + "=" * 50)
    print("MOEA/D优化完成!")
    print(f"总耗时: {total_time:.1f} 秒")
    print(f"总评估次数: {problem.evaluation_count}")
    print(f"缓存中的唯一解数量: {len(problem.evaluation_cache)}")

    # 去重处理最后一代解
    unique_solutions = []
```

```
    seen = set()
    for ind in res.pop:
        theta_A = float(ind.X[0])
        theta_B = float(ind.X[1])
        theta_C = float(ind.X[2])
        theta_E = float(ind.X[3])
        if include_k:
            k = float(ind.X[4])
            theta_A, theta_B, theta_C, theta_E, k = problem._validate_parameters(
                theta_A, theta_B, theta_C, theta_E, k
            )
            T = float(ind.F[0])
            PL = float(ind.F[1])
            key = (round(theta_A, 4), round(theta_B, 4), round(theta_C, 4),
                   round(theta_E, 4), round(k, 4), round(T, 6), round(PL, 6))
            if key not in seen:
                seen.add(key)
                unique_solutions.append((theta_A, theta_B, theta_C, theta_E, k, T, PL))
        else:
            theta_A, theta_B, theta_C, theta_E = problem._validate_parameters(
                theta_A, theta_B, theta_C, theta_E
            )
            T = float(ind.F[0])
            PL = float(ind.F[1])
            key = (round(theta_A, 4), round(theta_B, 4), round(theta_C, 4),
                   round(theta_E, 4), round(T, 6), round(PL, 6))
            if key not in seen:
                seen.add(key)
                unique_solutions.append((theta_A, theta_B, theta_C, theta_E, T, PL))

    print(f"去重后的有效解数量: {len(unique_solutions)}")
    print("=" * 50)

    return res, problem, problem.all_stats_dict, all_generations


def results_to_pareto(result, problem):
    """从优化结果中提取Pareto前沿解，并去重"""
    pareto = []
    include_k = problem.include_k

    # 获取非支配解
    if hasattr(result, 'opt'):
        front = result.opt
    else:
        front = result.pop

    for ind in front:
        if not np.any(np.isinf(ind.F)):
            theta_A = float(ind.X[0])
```

```
                theta_B = float(ind.X[1])
                theta_C = float(ind.X[2])
                theta_E = float(ind.X[3])
                if include_k:
                    k = float(ind.X[4])
                    theta_A, theta_B, theta_C, theta_E, k = problem._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E, k
                    )
                    T = float(ind.F[0])
                    PL = float(ind.F[1])
                    pareto.append((theta_A, theta_B, theta_C, theta_E, k, T, PL))
                else:
                    theta_A, theta_B, theta_C, theta_E = problem._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E
                    )
                    T = float(ind.F[0])
                    PL = float(ind.F[1])
                    pareto.append((theta_A, theta_B, theta_C, theta_E, T, PL))

    # 如果没有找到Pareto解，使用所有有效解
    if not pareto:
        print("警告: 未找到Pareto前沿解，使用所有有效解")
        for ind in result.pop:
            if not np.any(np.isinf(ind.F)):
                theta_A = float(ind.X[0])
                theta_B = float(ind.X[1])
                theta_C = float(ind.X[2])
                theta_E = float(ind.X[3])
                if include_k:
                    k = float(ind.X[4])
                    theta_A, theta_B, theta_C, theta_E, k = problem._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E, k
                    )
                    T = float(ind.F[0])
                    PL = float(ind.F[1])
                    pareto.append((theta_A, theta_B, theta_C, theta_E, k, T, PL))
                else:
                    theta_A, theta_B, theta_C, theta_E = problem._validate_parameters(
                        theta_A, theta_B, theta_C, theta_E
                    )
                    T = float(ind.F[0])
                    PL = float(ind.F[1])
                    pareto.append((theta_A, theta_B, theta_C, theta_E, T, PL))

    # 去重处理
    unique_pareto = []
    seen = set()
    for sol in pareto:
        key = tuple(round(x, 6) for x in sol)  # 决策变量和目标函数都参与去重
        if key not in seen:
```

```
                seen.add(key)
                unique_pareto.append(sol)

        # 按T排序
        if include_k:
            unique_pareto.sort(key=lambda x: x[5])  # 第5个元素是T
        else:
            unique_pareto.sort(key=lambda x: x[4])  # 第4个元素是T

        return unique_pareto


    def compare_algorithms(n_devices, n_gen=30, pop_size=40, n_runs=10,

                           include_k=True, simulation_func=None):
        """比较NSGA-II和MOEA/D两种算法"""
        print("==============算法比较==============")
        # 运行NSGA-II
        print("\n1. 运行NSGA-II...")
        nsga2_result, nsga2_problem = run_nsga2_optimization(
            n_devices=n_devices,
            n_gen=n_gen,
            pop_size=pop_size,
            n_runs=n_runs,
            include_k=include_k,
            simulation_func=simulation_func
        )
        nsga2_pareto = results_to_pareto(nsga2_result, nsga2_problem)
        # 运行MOEA/D
        print("\n2. 运行MOEA/D...")
        moead_result, moead_problem = run_moead_optimization(
            n_devices=n_devices,
            n_gen=n_gen,
            pop_size=pop_size,
            n_runs=n_runs,
            include_k=include_k,
            simulation_func=simulation_func
        )
        moead_pareto = results_to_pareto(moead_result, moead_problem)
        return {
            'nsga2': {'result': nsga2_result, 'problem': nsga2_problem, 'pareto': nsga2_pareto},
            'moead': {'result': moead_result, 'problem': moead_problem, 'pareto': moead_pareto}

        }
```

代码清单 3 问题 2 和问题 3 的统计算法

```
import numpy as np
from scipy import stats
from statsmodels.stats.multitest import multipletests
```

```
def analyze_pareto_with_confidence(pareto_solutions, all_stats_dict, n_bootstrap=1000, alpha=0.05):
    """
    对Pareto解集进行Bootstrap重采样计算置信区间

    Parameters:
    -----------
    pareto_solutions : list
        Pareto解集，4维格式: (theta_A, theta_B, theta_C, theta_E, T, PL)
        或5维格式: (theta_A, theta_B, theta_C, theta_E, k, T, PL)
    all_stats_dict : dict
        所有模拟结果的详细统计信息
    n_bootstrap : int
        Bootstrap重采样次数
    alpha : float
        显著性水平

    Returns:
    --------
    list: 包含置信区间的解集
    """
    pareto_with_ci = []

    for sol in pareto_solutions:
        # 判断是4维还是5维解
        if len(sol) == 6:  # 4维: (theta_A, theta_B, theta_C, theta_E, T, PL)
            theta_A, theta_B, theta_C, theta_E, T_orig, PL_orig = sol
            k = None
            key = (theta_A, theta_B, theta_C, theta_E)
        elif len(sol) == 7:  # 5维: (theta_A, theta_B, theta_C, theta_E, k, T, PL)
            theta_A, theta_B, theta_C, theta_E, k, T_orig, PL_orig = sol
            key = (theta_A, theta_B, theta_C, theta_E, k)
        else:
            print(f"警告: 未知的解格式，长度={len(sol)}")
            continue

        # 调试：打印所有可用的键
        if not all_stats_dict:
            print("警告: all_stats_dict 为空")
            continue

        available_keys = list(all_stats_dict.keys())
        # if len(available_keys) > 0:
        #     print(f"调试: 第一个可用键 = {available_keys[0]}, 类型 = {type(available_keys
[0])}")
        #     print(f"调试: 当前查找键 = {key}, 类型 = {type(key)}")
        #
        # if key not in all_stats_dict:
        #     print(f"警告: 未找到键 {key} 的统计信息")
```

```
#         print(f"可用键的前5个: {available_keys[:5]}")
#         continue

# 获取该解的所有模拟结果
all_stats = all_stats_dict[key]

# 检查数据结构
if isinstance(all_stats, dict):
    # 2维版本的数据结构
    T_samples = all_stats.get('makespan_days', [])
    PL_samples = all_stats.get('false_neg', [])
elif isinstance(all_stats, list) and len(all_stats) > 0 and isinstance(all_stats[0], dict):
    # 4/5维版本的数据结构
    T_samples = [stats.get('makespan_days', 0) for stats in all_stats]
    PL_samples = [stats.get('false_neg', 0) for stats in all_stats]
else:
    print(f"警告: 未知的数据结构类型 {type(all_stats)}")
    continue

# 检查数据是否有效
if not T_samples or not PL_samples:
    print(f"警告: 键 {key} 的数据为空")
    continue

# Bootstrap重采样
T_bootstrap_means = []
PL_bootstrap_means = []

n_samples = len(T_samples)

for _ in range(n_bootstrap):
    # 有放回抽样
    indices = np.random.choice(n_samples, n_samples, replace=True)
    T_bootstrap = [T_samples[i] for i in indices]
    PL_bootstrap = [PL_samples[i] for i in indices]

    T_bootstrap_means.append(np.mean(T_bootstrap))
    PL_bootstrap_means.append(np.mean(PL_bootstrap))

# 计算置信区间
T_mean = np.mean(T_bootstrap_means)
T_lower = np.percentile(T_bootstrap_means, (alpha / 2) * 100)
T_upper = np.percentile(T_bootstrap_means, (1 - alpha / 2) * 100)

PL_mean = np.mean(PL_bootstrap_means)
PL_lower = np.percentile(PL_bootstrap_means, (alpha / 2) * 100)
PL_upper = np.percentile(PL_bootstrap_means, (1 - alpha / 2) * 100)

# 保存结果
```

```
            if k is None:
                # 4维结果
                result = (theta_A, theta_B, theta_C, theta_E, T_mean, T_lower, T_upper,
                          PL_mean, PL_lower, PL_upper)
            else:
                # 5维结果
                result = (theta_A, theta_B, theta_C, theta_E, k, T_mean, T_lower, T_upper,
                          PL_mean, PL_lower, PL_upper)

            pareto_with_ci.append(result)

        return pareto_with_ci


def statistical_significance_test(pareto_with_ci, all_stats_dict, alpha=0.05, factor_count=2):
    """
    统计显著性检验 - 支持2因子、4因子、5因子版本

    参数:
    pareto_with_ci: Pareto解集，包含置信区间信息
    all_stats_dict: 所有统计结果的字典
    alpha: 显著性水平，默认0.05
    factor_count: 因子数量，2/4/5
    """
    print("=" * 60)
    print(f"统计显著性检验结果（{factor_count}因子）")
    print("=" * 60)

    comparisons = []

    for i, sol1 in enumerate(pareto_with_ci):
        # 根据因子数量解析解的参数
        if factor_count == 2:
            theta1, k1, T_mean1, T_lower1, T_upper1, PL_mean1, PL_lower1, PL_upper1 = sol1
            key1 = (theta1, k1)
        elif factor_count == 4:
            theta_A1, theta_B1, theta_C1, theta_E1, T_mean1, T_lower1, T_upper1, PL_mean1, PL_lower1, PL_upper1 = sol1
            key1 = (theta_A1, theta_B1, theta_C1, theta_E1)
        elif factor_count == 5:
            theta_A1, theta_B1, theta_C1, theta_E1, k1, T_mean1, T_lower1, T_upper1, PL_mean1, PL_lower1, PL_upper1 = sol1
            key1 = (theta_A1, theta_B1, theta_C1, theta_E1, k1)
        else:
            raise ValueError("不支持的因子数量，只能是2、4或5")

        if key1 not in all_stats_dict:
            continue

        # 从 all_results 中提取数据
```

```
            results1 = all_stats_dict[key1]
            T_values1 = results1.get('makespan_days', [])
            PL_values1 = results1.get('false_neg', [])

            for j, sol2 in enumerate(pareto_with_ci):
                if i >= j:  # 避免重复比较
                    continue

                # 根据因子数量解析第二个解的参数
                if factor_count == 2:
                    theta2, k2, T mean2, T lower2, T upper2, PL mean2, PL lower2, PL upper2 =
sol2
                    key2 = (theta2, k2)
                elif factor_count == 4:
                    theta A2, theta B2, theta C2, theta E2, T mean2, T lower2, T upper2, PL mea
n2, PL_lower2, PL_upper2 = sol2
                    key2 = (theta_A2, theta_B2, theta_C2, theta_E2)
                elif factor_count == 5:
                    theta A2, theta B2, theta C2, theta E2, k2, T mean2, T lower2, T upper2, PL
mean2, PL_lower2, PL_upper2 = sol2
                    key2 = (theta_A2, theta_B2, theta_C2, theta_E2, k2)

                if key2 not in all_stats_dict:
                    continue

                # 从 all_results 中提取数据
                results2 = all_stats_dict[key2]
                T_values2 = results2.get('makespan_days', [])
                PL_values2 = results2.get('false_neg', [])

                # 使用Mann-Whitney U检验（非参数）
                try:
                    T_stat, T_p = stats.mannwhitneyu(T_values1, T_values2, alternative='two-sided')
                    PL stat, PL p = stats.mannwhitneyu(PL values1, PL values2, alternative='two-sid
ed')

                    comparisons.append((key1, key2, 'T', T_p))
                    comparisons.append((key1, key2, 'PL', PL_p))

                except Exception as e:
                    print(f"检验失败: {key1} vs {key2}, 错误: {e}")

        # 多重比较校正
        if comparisons:
            p_values = [comp[3] for comp in comparisons]
            reject, pvals_corrected, _, _ = multipletests(p_values, alpha=alpha, method='fdr_bh')

            for (key1, key2, metric, raw_p), corr_p, sig in zip(comparisons, pvals_corrected, reject):
                print(f"{key1} vs {key2}, {metric}: p={raw_p:.4f}, 校正p={corr_p:.4f} {'*' if sig els
e 'NS'}")
        else:
```

```
        print("没有可比较的解对")

def form_robust_pareto_set_nd(pareto_with_ci, all_stats_dict, alpha=0.05,include_k=False):
    """
    形成稳健Pareto解集（通用n维）
    pareto with ci: [(theta1,...,thetaN, [T mean, T lower, T upper], [PL mean, PL lower, PL upper]), ...]
    all_stats_dict: 统计信息字典
    alpha: 显著性水平
    """
    robust_pareto = []
    elimination_info = []

    for i, sol1 in enumerate(pareto_with_ci):
        # 拆分解
        *theta1, T_mean1, T_lower1, T_upper1, PL_mean1, PL_lower1, PL_upper1 = sol1
        key1 = tuple(theta1)

        if key1 not in all_stats_dict:
            continue

        is_robust = True
        dominated_by = []

        for j, sol2 in enumerate(pareto_with_ci):
            if i == j:
                continue
            *theta2, T_mean2, T_lower2, T_upper2, PL_mean2, PL_lower2, PL_upper2 = sol2
            key2 = tuple(theta2)
            if key2 not in all_stats_dict:
                continue

            # 检查是否被支配
            if is_statistically_dominated(key1, key2, all_stats_dict, alpha):
                is_robust = False
                dominated_by.append(key2)
                elimination_info.append({
                    'eliminated': key1,
                    'dominated_by': key2,
                    'reason': '统计显著性支配'
                })
                break

        if is_robust:
            robust_pareto.append(sol1)
            print(f"✓ 保留: θ={theta1}, T={T_mean1:.2f}[{T_lower1:.2f},{T_upper1:.2f}], "
                  f"PL={PL_mean1:.6f}[{PL_lower1:.6f},{PL_upper1:.6f}]")
        else:
            print(f"✗ 淘汰: θ={theta1} (被 {dominated_by} 支配)")
```

```
    return robust_pareto, elimination_info

def is_statistically_dominated(key1, key2, all_stats_dict, alpha=0.05):
    """
    判断解1是否被解2统计显著性支配 - 修正版
    """
    if key1 not in all_stats_dict or key2 not in all_stats_dict:
        return False

    # 从 all_results 中提取数据
    results1 = all_stats_dict[key1]
    results2 = all_stats_dict[key2]

    T_values1 = results1.get('makespan_days', [])
    T_values2 = results2.get('makespan_days', [])
    PL_values1 = results1.get('false_neg', [])
    PL_values2 = results2.get('false_neg', [])

    # 使用Mann-Whitney U检验
    try:
        # 检验解1的T是否显著大于解2的T
        T_stat, T_p = stats.mannwhitneyu(T_values1, T_values2, alternative='greater')
        # 检验解1的PL是否显著大于解2的PL
        PL_stat, PL_p = stats.mannwhitneyu(PL_values1, PL_values2, alternative='greater')

        # 如果解2在两个目标上都显著优于解1，则解1被支配
        if T_p < alpha and PL_p < alpha:
            return True

    except Exception as e:
        print(f"支配判断失败: {key1} vs {key2}, 错误: {e}")

    return False
```

代码清单 4 问题 4 参数计算代码

```
# ------------------ 计算 SRC（标准化回归系数） ------------------
def compute src(X, Y, var names):
    scaler = StandardScaler().fit(X)
    Xs = scaler.transform(X)
    Ys = (Y - Y.mean()) / (Y.std() if Y.std() != 0 else 1.0)
    lr = LinearRegression().fit(Xs, Ys)
    coefs = lr.coef
    return dict(zip(var names, coefs.tolist())), float(lr.intercept )

# ------------------ 计算 PRCC（偏秩相关） ------------------
def compute prcc(X, Y, var names):
    # 对每列做秩变换
    Xr = np.apply along axis(lambda col: rankdata(col), 0, X)
    Yr = rankdata(Y)
    prcc vals = []
    nvars = Xr.shape[1]
    for i in range(nvars):
        others = [j for j in range(nvars) if j != i]
        reg1 = LinearRegression().fit(Xr[:, others], Xr[:, i])
        resid x = Xr[:, i] - reg1.predict(Xr[:, others])
```

```
        reg2 = LinearRegression().fit(Xr[:, others], Yr)
        resid_y = Yr - reg2.predict(Xr[:, others])
        r = np.corrcoef(resid_x, resid_y)[0, 1]
        prcc_vals.append(float(r))
    return dict(zip(var_names, prcc_vals))

# ------------------ 主效应与边际差分（按离散化后的取值） ------------------
def compute_main_effects_and_marginals(df, var_names):
    main_effects = {}
    marginals = {}
    # df 应含离散化列后缀 "_q"（如 K_q, replace_at_q, ...）
    for v in var_names:
        col_q = v + '_q'
        if col_q not in df.columns:
            # fallback to raw
            df[col_q] = df[v]
        groups = df.groupby(col_q)['makespan_hours'].agg(['mean','count']).reset_index()
        level_means = dict(zip(groups[col_q].tolist(), groups['mean'].tolist()))
        main_effects[v] = level_means
        # 边际差分：排序后相邻差
        sorted_levels = sorted(level_means.keys())
        deltas = []
        for i in range(len(sorted_levels)-1):
            a = level_means[sorted_levels[i]]
            b = level_means[sorted_levels[i+1]]
            deltas.append(b - a)  # 增加该变量一档后的 makespan 变化（正 -> 增加 makespa
n）
        marginals[v] = {'levels': sorted_levels, 'deltas': deltas}
    return main_effects, marginals
```