

基于 CMA-ES 与自洽场迭代的无人机烟幕协同策略优化

摘要

烟幕干扰弹是一种高效的战略防御手段，能够在目标与来袭导弹之间形成遮蔽，干扰导弹的探测与打击。本文围绕“烟幕干扰弹投放策略分析与优化”问题，基于空间向量几何与经典物理学理论，建立了运动模型和策略优化模型。本文采用 Nelder-Mead, CMA-ES, HF-SCF 等多种优化方法，对不同情景下无人机的行进路线与烟幕弹的投放策略进行研究分析与精细优化。

针对问题一：构建单无人机单烟幕弹单导弹的运动模型与遮蔽的精细判定模型并代入具体策略。首先确定无人机、烟幕弹与导弹运动轨迹的空间向量表达式，然后对真目标圆柱体精细取点拟合真实情况并构建遮蔽判定函数，最后在时间轴上通过较粗步长遍历，零点跳转时精细二分逼近快速且准确地得到结果：1.392 秒。

针对问题二：在前述模型基础上，采用粗搜索取种，随后并行开展两路优化：(A) 用 Nelder-Mead (单纯形搜索法) 对每个种子做局部精修与步幅自适应收缩；(B) 用 CMA-ES (Covariance Matrix Adaptation Evolution Strategy, 协方差矩阵自适应进化策略) 做全局协同搜索，设置分阶段精度、边界投影与早停策略以提升效率。两种方法交叉验证，最终 Nelder-Mead 获解 4.585156 秒，CMA-ES 获解 4.5884 秒。

针对问题三：在前述模型基础上，扩展为三烟幕弹协同遮蔽模型。先使用“角度剪枝 + 可行性判别”的并行海量种子生成，再进行两阶段的 CMA-ES 优化：阶段一以较粗时间步长快速筛选获得若干“冠军”解，阶段二在更细时间步长下深度迭代，精确协调三弹起爆的时空关系以实现接力式连续遮蔽。最终得到有效遮蔽总时长 7.6086 秒，结果详见 `result1.xlsx`。

针对问题四：在前述模型基础上，扩展为多无人机协同投放模型。采用“高质量种子 + 分阶段 CMA-ES”的算法管线。先整体生成协同种子，将其组合成 12 维有界编码并解码为可执行计划，随后在同一评估器下进行两阶段的进化搜索：阶段一用较粗步长快速筛选，阶段二用更细步长深度迭代与收敛校验，以提升全局搜索效率与稳健性。最终得到有效遮蔽总时长 11.8025 秒，结果详见 `result2.xlsx`。

针对问题五：在前述模型基础上，扩展为多无人机多烟幕弹多导弹的完备策略优化模型。采用“多顺序贪心拼装 + HF-SCF (Hartree-Fock Self-Consistent Field, 哈特里-福克自洽场法) 交替微调 + 简化 CMA-ES”的协同策略合成框架。先以固定随机源为各无人机生成候选，采样多种执行顺序并按“全局遮蔽增量最大”原则贪心拼装初始方案；再在“其他无人机参数固定”的 SCF 轮换下对单机参数用 CMA-ES 作局部微调并配合早停准则收敛。最终获得联合遮蔽总时长 47.781 秒，结果详见 `result3.xlsx`。

关键词：烟幕干扰弹 协同模型 单纯形法 协方差矩阵自适应进化策略
自洽场法

1 问题重述

1.1 问题背景

烟幕干扰弹是一种典型的低成本，高效费比防护手段。其通过在真目标与来袭导弹之间快速形成遮蔽云团，干扰导弹的观测与制导过程，从而提升目标的生存概率。

在实际作战中，投放策略需综合权衡以下因素：其一，遮蔽需具备较长的时域和空间上的合理覆盖，确保云团能在导弹视轴附近按时出现；其二，多个烟幕云团之间需要联合遮蔽延长干扰时间，又要避免冗余以节约资源；其三，无人机飞行轨迹受到一定限制，且干扰弹的投放间隔和引爆延迟均有约束。

这些因素相互耦合，使得“何时何地投放、如何在多机多弹条件下协同作战以实现最长有效遮蔽时间”成为一个强非线性、强约束、离散-连续混合的复杂优化问题。因此，亟需建立合理的数学模型，并设计高效的优化算法来求解。

1.2 问题提出

针对题目所给场景，本文需建立无人机、干扰弹与导弹之间的几何—物理模型，以“真目标在导弹视域内的有效遮蔽时长最大化”为核目标，综合考虑无人机的飞行速度与航向、干扰弹的投放点与起爆点、以及延时控制等关键决策变量，逐步解决以下问题：

1. **问题一**：单机单弹验证。无人机 FY1 以 120 m/s 向假目标飞行，受领任务 1.5 s 后投放 1 枚干扰弹，并于 3.6 s 后起爆。计算该干扰弹对导弹 M1 的有效遮蔽时长。
2. **问题二**：单机单弹优化。调整 FY1 的航向、速度、投放点及起爆点，确定最优方案，使对 M1 的遮蔽时间最大化。
3. **问题三**：单机多弹策略。FY1 投放 3 枚干扰弹，设计合理的投放与起爆时序，并将结果保存至 `result1.xlsx`。
4. **问题四**：多机单弹协同。FY1、FY2、FY3 各投放 1 枚干扰弹，协同干扰 M1，给出联合投放方案，并将结果输出至 `result2.xlsx`。
5. **问题五**：多机多弹对抗。5 架无人机中每架至多投放 3 枚干扰弹，需联合干扰 3 枚导弹 M1、M2、M3，设计综合投放方案，并将结果输出至 `result3.xlsx`。

2 模型假设

为便于建模与求解，对场景作如下合理化假设与约定（除特别说明外均在全程保持不变）：

1. **坐标系与时刻约定:** 以假目标为原点, xy 为水平面、 z 轴竖直向上; $t = 0$ 表示警戒雷达发现来袭并向无人机下达任务的时刻。
2. **平台与弹道运动:**
 - (a) 导弹按匀速直线 (速度常数 300 m/s) 飞向假目标, 忽略机动与航迹抖动;
 - (b) 无人机在 $t = 0$ 可瞬时选择航向与速度 ($70 \sim 140$ m/s), 此后等高匀速直线飞行, 不再改变航向/速度/高度;
 - (c) 干扰弹自无人机释放瞬间获得与无人机相同的初速, 之后仅受重力作用作抛体运动; 忽略空气阻力、姿态变化与二次喷气等效应;
 - (d) 同一架无人机两次投放的时间间隔不小于 1 s; 每枚干扰弹的起爆由时间引信控制, 延时为可决策变量。
3. **环境与外界因素:** 忽略空气阻力、风场、湍流、地形起伏、地球曲率与大气折射等影响; 重力加速度取常数 $g = 9.8$ m/s²。
4. **烟幕模型:**
 - (a) 起爆瞬时形成球形云团, 有效遮蔽半径 $r_s = 10$ m; 云团中心以后随时间仅竖直向下匀速运动, 速度 $v_s = 3$ m/s, 无侧向漂移;
 - (b) 有效遮蔽持续时间 20 s, 到时效后判为完全失效, 不考虑浓度渐变与叠加增强;
 - (c) 多枚云团之间互不影响, 判定“被遮蔽”时采取并集准则 (任一有效云团遮挡即视为遮蔽成立)。
5. **可见性/遮蔽判据 (二元化简化):**
 - (a) 导弹末制导采用直线视轴观测; 命中判据与制导细节不参与可见性计算;
 - (b) 以 24 个代表点 (已验证 24 个代表点可以精细代表原目标, 增加取点没有明显优化) 离散近似真目标 (半径 $r_T = 7$ m、高 $h_T = 10$ m) 的体积; 当导弹视轴至任一该代表点的连线与某有效烟幕球体发生几何相交 (线段与球体交叉) 时, 判定该代表点在该时刻“被遮蔽”;
 - (c) 目标整体“被遮蔽”的判定采用全覆盖准则 (所有代表点同时被遮蔽) 或覆盖阈值准则 (被遮蔽点比例不低于预设阈值), 本文默认全覆盖准则, 除非在数值实现中另行说明。
6. **目标与诱饵:** 假目标固定于原点; 真目标下底面圆心位于 $(0, 200, 0)$, 几何尺寸按题设给定并在全程保持不变; 忽略目标自遮挡与反射特性差异。
7. **数值与工程化:**

- (a) 连续时间按固定步长离散，步长不影响结论级别（将通过敏感性分析验证收敛性）；
- (b) 所有硬件动作（投放、起爆）视为在时间离散点上即时发生；所有约束（速度、间隔、延时上下界）作为硬约束处理。

3 符号说明

符号	含义	数值/单位
v_M	导弹飞行速度	300 m/s
$\mathbf{P}_M(t)$	导弹在时刻 t 的空间坐标	(x, y, z) , m
v_U	无人机飞行速度	70 ~ 140 m/s (例：问题一取 120)
$\mathbf{P}_U(t)$	无人机在时刻 t 的空间坐标	(x, y, z) , m
θ	无人机飞行航向角	rad
t_{drop}	投放时刻（相对任务下达时刻）	1.5 s (问题一)
t_{fuse}	起爆延迟（相对投放）	3.6 s (问题一)
$\mathbf{P}_D(t)$	干扰弹在时刻 t 的空间坐标（质心）	(x, y, z) , m
r_s	烟幕云团有效半径	10 m
v_s	烟幕云团下沉速度	3 m/s
T_s	烟幕云团有效时间	20 s
r_T	真目标圆柱底面半径	7 m
h_T	真目标高度	10 m
\mathbf{P}_T	真目标圆柱底面圆心坐标	$(0, 200, 0)$, m
N_U	无人机数量	5 (FY1–FY5)
N_D	每架无人机最大投放弹数	3
N_M	来袭导弹数量	3 (M1–M3)

表 1: 主要符号说明

4 问题一

4.1 问题重述

FY1 以 120 m/s 朝假目标方向匀速等高直线飞行，在 $t_r = 1.5$ s 时投放干扰弹， $t_e = 3.6$ s 后按时间引信起爆。计算对导弹 M1 的整体有效遮蔽时长。

4.2 模型建立（向量几何）

坐标与常量 假目标 $\mathbf{F} = (0, 0, 0)$, 导弹初始位置 $\mathbf{M}_0 = (20000, 0, 2000)$, 无人机初始位置 $\mathbf{U}_0 = (17800, 0, 1800)$ 。设定

$$v_M = 300 \text{ m/s}, \quad v_U = 120 \text{ m/s}, \quad r_s = 10 \text{ m}, \quad v_s = 3 \text{ m/s}, \quad g = 9.8 \text{ m/s}^2,$$

并约定 $\mathbf{U}_0 = \mathbf{P}_U(0)$ 。

导弹运动 导弹沿直线朝 \mathbf{F} 匀速飞行, 单位方向向量

$$\hat{\mathbf{d}}_M = \frac{\mathbf{F} - \mathbf{M}_0}{\|\mathbf{F} - \mathbf{M}_0\|},$$

则导弹轨迹为

$$\mathbf{P}_M(t) = \mathbf{M}_0 + v_M t \hat{\mathbf{d}}_M. \quad (1)$$

无人机航向、投放点与起爆点 FY1 等高直线飞向 \mathbf{F} , 其水平单位方向向量

$$\hat{\mathbf{d}}_{xy} = \frac{(\mathbf{F} - \mathbf{U}_0)_{xy}}{\|(\mathbf{F} - \mathbf{U}_0)_{xy}\|} = (-1, 0, 0).$$

干扰弹质心轨迹记为 $\mathbf{P}_D(t)$, 在投放与起爆瞬间分别满足

$$\mathbf{P}_D(t_r) = \mathbf{P}_{\text{drop}}, \quad \mathbf{P}_D(t_{\text{exp}}) = \mathbf{C}_{\text{exp}},$$

其中 $t_{\text{exp}} = t_r + t_e$, 且

$$\mathbf{P}_{\text{drop}} = \mathbf{U}_0 + v_U t_r \hat{\mathbf{d}}_{xy}, \quad (2)$$

$$\mathbf{C}_{\text{exp}} = \mathbf{P}_{\text{drop}} + v_U t_e \hat{\mathbf{d}}_{xy} + \left(0, 0, -\frac{1}{2}gt_e^2\right). \quad (3)$$

烟幕云团球心运动 起爆后烟幕球心仅竖直下沉:

$$\mathbf{C}(t) = \mathbf{C}_{\text{exp}} + \left(0, 0, -v_s(t - t_{\text{exp}})\right), \quad t \geq t_{\text{exp}}. \quad (4)$$

整体遮蔽判据 (24 点法) 将真目标圆柱体上下底圆周各等分 12 点 (不含圆心), 得代表点集 $\mathcal{T} = \{T_i\}_{i=1}^{24}$ 。在时刻 t , 对任意 T_i , 导弹视轴线段为 $\overline{\mathbf{P}_M(t) T_i}$ 。定义点到线段距离

$$d\left(\mathbf{C}(t), \overline{\mathbf{P}_M(t) T_i}\right) = \min_{s \in [0, 1]} \|\mathbf{P}_M(t) + s(T_i - \mathbf{P}_M(t)) - \mathbf{C}(t)\|.$$

定义判定函数

$$g(t) = \max_{1 \leq i \leq 24} \left(d\left(\mathbf{C}(t), \overline{\mathbf{P}_M(t) T_i}\right) - r_s \right). \quad (5)$$

当且仅当 $g(t) \leq 0$, 认为目标在 t 时刻被整体遮蔽。

4.3 数值求解与稳健性

整体求解流程分为两个层次：

1. **遮蔽判定 (get_obscuration_state_at_time)**: 给定时刻 t , 先生成所有有效烟幕球心位置 (考虑下沉), 再计算导弹位置; 逐一检查“导弹—目标离散点”的线段到各烟幕球心的最近点距离是否 $\leq 10\text{ m}$ 。只有当 24 条视线均被至少一个烟幕球遮断时, 才判定目标整体被遮蔽。最近点距离采用“点投影到线段并截断”的标准公式实现。
2. **时间扫描 + 二分精化**: 在区间 $[0, t_{\max}]$ (本题取 $t_{\max} = 20\text{ s}$) 以粗步长 (如 $\Delta t = 10^{-3}\text{ s}$) 扫描遮蔽状态, 一旦发现判定函数 $g(t)$ 发生符号变化或遮蔽状态跳变, 就在对应子区间内使用二分法 (精度 10^{-5} s) 求出精确的进入/退出时刻。由此得到每个遮蔽区间并累计总时长。
3. **区段合并**: 按时序得到进入/退出时刻并合并区段, 总遮蔽时长为各区段之和。
4. **贡献分解 (calculate_grenade_contributions)**: 为实现可解释性, 在每个遮蔽小区间取中点 t_{mid} , 反查此刻哪些烟幕球对 24 条视线的遮断“有贡献”, 并将该区间时长累加到相应干扰弹名下。这样不仅能得到总遮蔽时长, 还能量化每枚干扰弹的贡献度, 即“哪颗弹贡献了多少秒”。
5. **稳健性**: 将 Δt 收紧至 1×10^{-5} , 验证结果收敛。

4.4 结果与分析

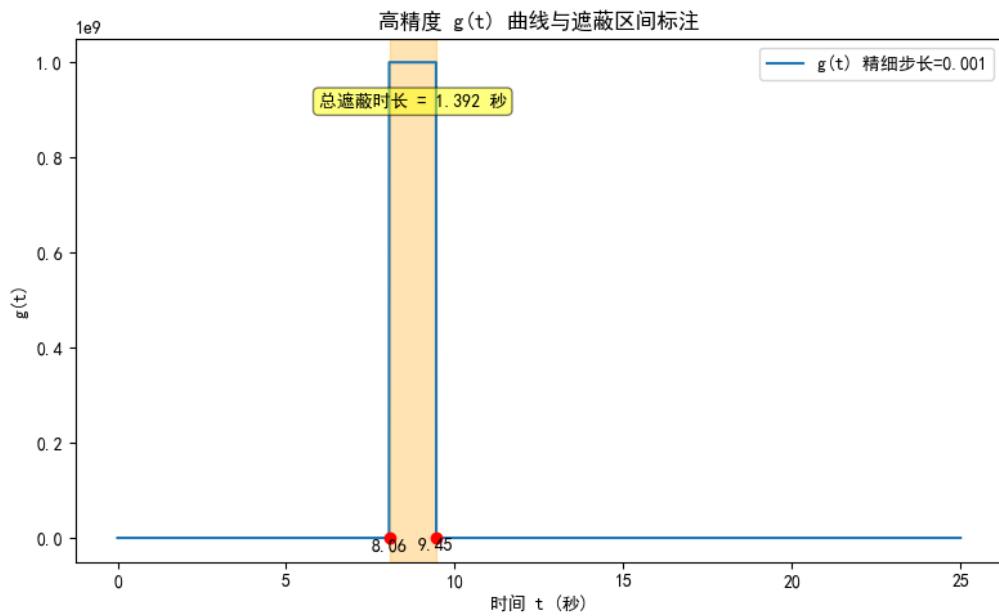
程序输出有效遮蔽区间为 $[8.0562, 9.4482]\text{ s}$, 总遮蔽时长

1.392 s.

该结果与模型假设下的几何一时序约束一致; 遮蔽发生在云团下沉穿越导弹视轴的窄窗, 主要受起爆高度、下沉速度与相对几何位置共同约束。

遮蔽区间 [s]	总时长 [s]
$[8.0562, 9.4482]$	1.392

表 2: FY1 对 M1 的整体有效遮蔽时间 (24 点判定)

图 1: $g(t)$ 曲线与遮蔽区间标注 (阴影为有效遮蔽)

3D Scenario Illustration: Missile, UAV, Targets, and Smoke

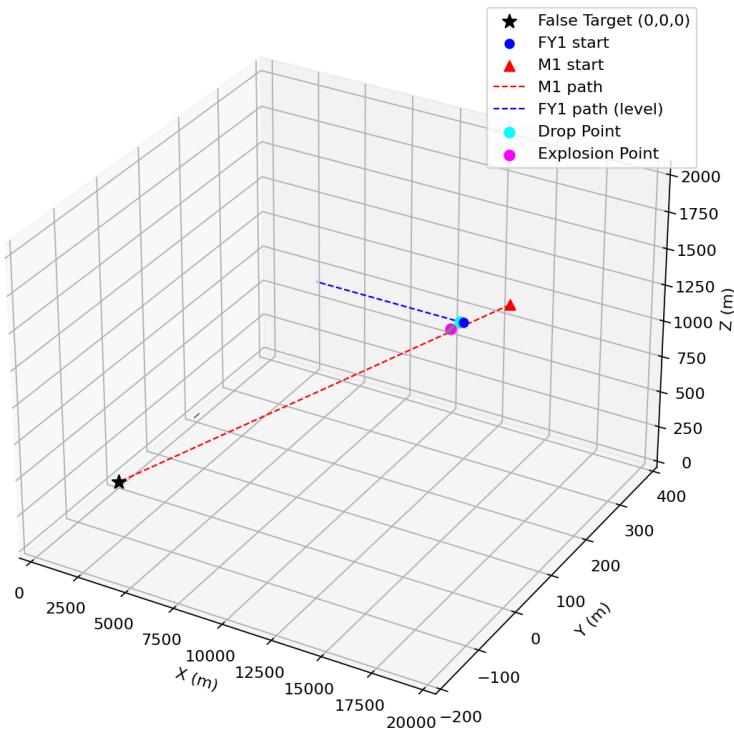


图 2: 三维场景示意 (导弹轨迹、烟幕球心轨迹与目标)

5 问题二：单机单弹投放参数优化

5.1 问题刻画与变量

在问题一的判定框架下，引入无人机的四个决策变量

$$\boldsymbol{x} = (v_U, \theta, t_{\text{drop}}, t_{\text{fuse}}),$$

其中 $v_U \in [70, 140]$ m/s 为无人机等高匀速巡航速度， θ 为航向角（下文用“度”表述，取值区间记为 $(-180^\circ, 180^\circ]$ ）， $t_{\text{drop}} \in [0, 60]$ s 为投放时刻（相对受领任务的时刻）， $t_{\text{fuse}} \in [0, 60 - t_{\text{drop}}]$ s 为引信延时。起爆时刻为 $t_{\text{exp}} = t_{\text{drop}} + t_{\text{fuse}}$ ，需满足时序约束 $t_{\text{exp}} \leq t_{\text{ME}}$ ，其中 t_{ME} 为导弹 M1 击中假目标时刻。

为便于阅读，将变量与取值范围整理如下：

表 3: 问题二的决策变量与取值范围

变量	含义	取值范围
v_U (m/s)	无人机巡航速度	[70, 140]
θ (deg)	航向角 (度)	(-180, 180]
t_{drop} (s)	投放时刻 (相对受领)	[0, 60]
t_{fuse} (s)	引信延时	[0, 60 - t_{drop}]

“整体遮蔽”的目标函数定义为

$$\max_{\boldsymbol{x}} J(\boldsymbol{x}) = \sum_k (t_{\text{out}}^{(k)} - t_{\text{in}}^{(k)}), \quad \text{s.t. } g(t) \leq 0 \quad \text{于 } [t_{\text{in}}^{(k)}, t_{\text{out}}^{(k)}],$$

其中 $g(t)$ 的定义同式 (5)，仍采用圆柱真目标的 24 个代表点进行“整体遮蔽”判定。

5.2 求解策略：两阶段 CMA-ES

考虑到目标函数不可导且每次评估需进行“扫描 + 二分”的数值过程，本题采用**协方差矩阵自适应进化策略 (CMA-ES)** 进行全局优化。为提升效率与鲁棒性，设计“两阶段优化”框架：

1. **探索阶段 (Stage 1)**：随机生成若干可行“种子”，对每个种子运行少量迭代的 CMA-ES，快速筛选候选；
2. **开采阶段 (Stage 2)**：选取若干性能最佳“冠军”，在其附近进行更多迭代的 CMA-ES，逼近全局最优；
3. **高精度回算**：采用更细时间步长 $\Delta t = 0.01$ s 与二分容差 10^{-4} ，复核最优解的遮蔽区间与总时长；

4. **轻量粗搜 + Nelder–Mead (NM) 单纯形**: 粗搜在每个维度取稀疏格点/随机采样, 快速评估 $f(x)$, 保留若干最优候选作起点。再以最佳候选为初值构造 5 点单纯形, 执行反射/扩展/收缩/压缩, 不需梯度、对不光滑/含判定逻辑的 f 依然稳健。出界即截断或投影回可行域, 精度由粗步长 + 二分控制。实现简单、单次收敛快, 但更偏局部改良, 依赖粗搜命中好盆地;
5. **两阶段 CMA-ES 全局优化**: 先生成可行“整体种子”, 编码为 [朝向角, 速度, 投放时刻, 起爆延迟], 适应度由高保真仿真计算遮蔽时长。阶段一进行少量迭代的粗优化, 筛选前 K 个“冠军”; 阶段二在其附近高精度 CMA-ES 搜索。终检时再用更细步长与更小阈值复算最优。全局性更强, 但计算量较 NM 大, 通过“先可行后最优”分配预算提升效率。

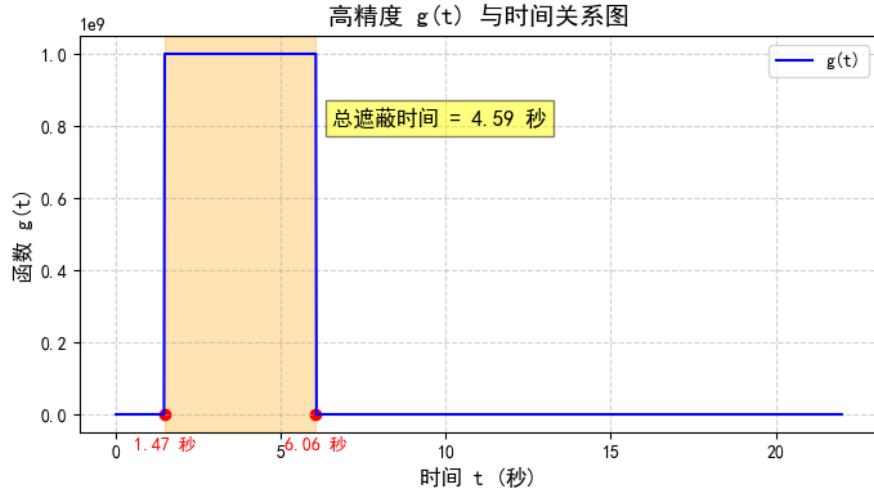
5.3 优化结果与可视化

通过程序运行, 得到的最优结果汇总于表 4。与问题一固定方案 ($J = 1.392\text{ s}$) 相比, 本方案将总遮蔽时长提升至 4.5884 s , 约为原先的 3.3 倍。

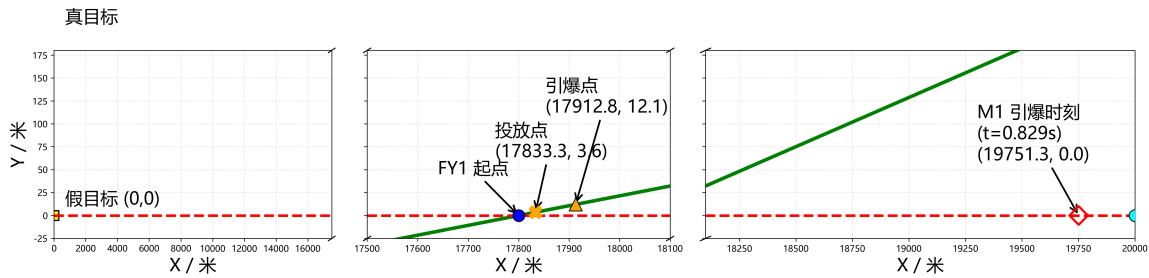
表 4: 问题二的最优投放策略与性能指标

指标	数值
最大有效遮蔽时长 J^* (s)	4.5884
飞行方向 θ (deg)	6.12
飞行速度 v_U (m/s)	136.89
投放时间 t_{drop} (s)	0.245
引爆延迟 t_{fuse} (s)	0.584
投放点 (x, y, z) (m)	(17833.41, 3.58, 1800.00)
起爆点 (x, y, z) (m)	(17912.96, 12.11, 1798.33)
相对问题一的提升倍数	$4.5884/1.392 \approx 3.30$

为直观展示最优解对应的遮蔽效果, 绘制 $g(t)$ 高精度曲线与二维场景示意图 (见图 3 与图 4)。

图 3: 最优解的 $g(t)$ 高精度曲线及遮蔽区间标注

问题二最优解示意 - CMA-ES $v_U=136.89$ m/s, $\theta=6.12^\circ$, $t_{drop}=0.245$ s, $t_{fuse}=0.584$ s
 无人机 FY1 轨迹 ● FY1 起点 ▲ 引爆点 ◇ M1 引爆时刻 ◆ 真目标
 导弹 M1 轨迹 * 投放点 ○ M1 起点 ■ 假目标 (0,0)

图 4: 二维场景示意图 (断轴 x ; 含 FY1 轨迹、投放/起爆点、M1 起始与引爆位置、假目标)

5.4 与 Nelder–Mead 解的对比

采用 Nelder–Mead 算法得到的另一组最优解为: $v_U^* = 85.6$ m/s、 $\theta^* = 4.6^\circ$ 、 $t_{drop}^* = 1.16$ s、 $t_{fuse}^* = 0.31$ s, 其遮蔽时长为 4.585 s, 与 CMA-ES 解几乎相同。两组解在四个变量上差异较大, 反映了本问题的高度非凸, 且近优解在参数空间呈延展分布。

表 5: CMA-ES 与 Nelder–Mead 最优解对比 (单位: m/s、deg、s)

方法	v_U	θ	t_{drop}	t_{fuse}	J (s)
CMA-ES	136.89	6.12	0.245	0.584	4.5884
Nelder–Mead	85.60	4.60	1.16	0.31	4.5852

对应的几何位置差异见图 5 与图 6: 前者更偏向低速、较晚投放, 起爆点靠近下游; 后者对应高速、较早投放, 起爆点偏上游——但二者形成了近似等效的遮蔽时长。

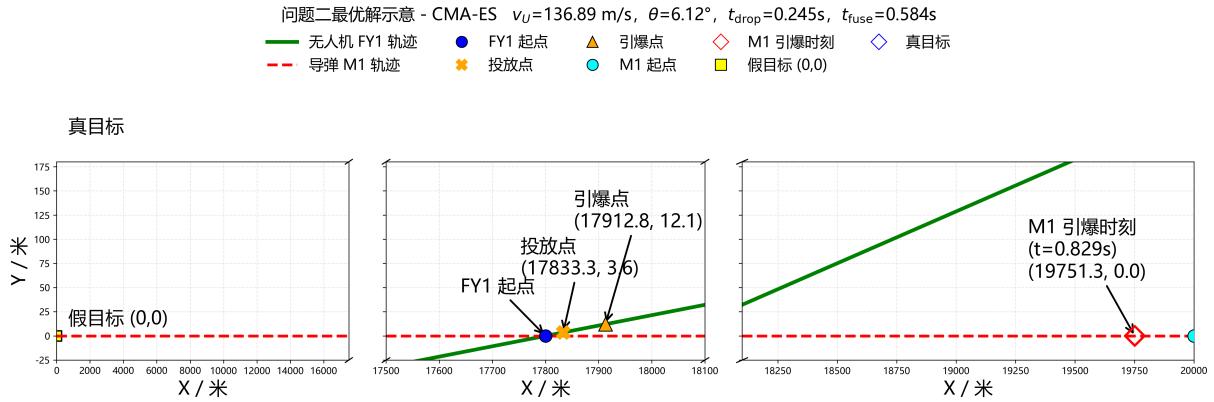
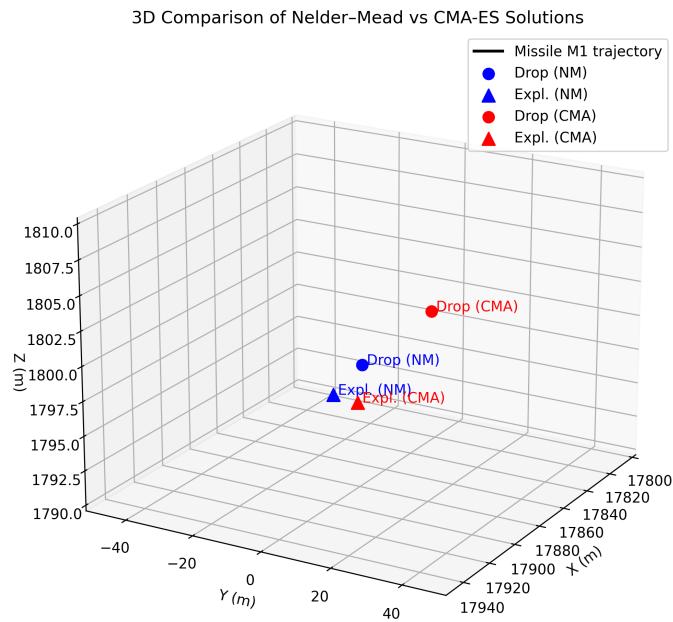
图 5: Nelder–Mead 解的 XY 平面示意图 (断轴 x ; 含 FY1 轨迹与 M1 关键位置)

图 6: Nelder–Mead 与 CMA-ES 解的三维位置对比 (无人机局部放大图)

5.5 小结

在既定物理假设与“整体遮蔽”判据下，较优解分布广泛：不同的速度、航向和投放时序均可能形成等效遮蔽。该多解性一方面提供战术灵活性，指挥员可结合平台性能与战场约束选择不同投放策略；另一方面也表明问题高度非凸，需采用具备全局搜索能力的算法（如 CMA-ES）以避免陷入局部最优，并通过高精度回算稳健评估最终方案。

6 问题三：单机三弹投放策略优化

6.1 问题刻画与变量

在问题二的框架下，进一步考虑无人机 FY1 投放 3 枚烟幕干扰弹，通过时序协同实现对导弹 M1 的持续遮蔽。定义决策变量为

$$\boldsymbol{x} = (v_U, \theta, t_1, d_1, g_2, d_2, g_3, d_3),$$

其中 v_U 为无人机速度， θ 为航向角； t_1 为首枚投放时刻， d_1 为其引信延时； g_2, g_3 分别为第 2、3 枚相对前一枚的投放间隔， d_2, d_3 为对应的引信延时。由此可得

$$t_2 = t_1 + g_2, \quad t_3 = t_2 + g_3, \quad T_i = t_i + d_i \ (i = 1, 2, 3),$$

并满足时序约束 $T_i \leq t_{ME}$ （其中 t_{ME} 为导弹 M1 击中假目标的时刻），以及 $t_i, d_i, g_i \geq 0$ 。

各变量及取值范围如表 6 所示。

表 6: 问题三的决策变量与取值范围

变量	含义	取值范围
v_U (m/s)	无人机巡航速度	[70, 140]
θ (deg)	航向角 (度)	(179, 180)]
t_1 (s)	第 1 枚投放时刻	[0, 60)
d_1 (s)	第 1 枚引信延时	[0, 60 - t_1)
g_2, g_3 (s)	第 2、3 枚相对投放间隔	[1, 60 - t_1)
d_2, d_3 (s)	第 2、3 枚引信延时	[0, 60 - $g_2/60 - g_3$)

目标函数仍定义为导弹 M1 在最大仿真时长内的总有效遮蔽时间：

$$\max_{\boldsymbol{x}} J(\boldsymbol{x}) = T_{\text{obscured}}(\boldsymbol{x}),$$

遮蔽判据与问题二相同。

6.2 求解策略：两阶段 CMA-ES

由于维度增至 8，解空间显著扩大。为兼顾效率与鲁棒性，继续采用“两阶段 CMA-ES”：

1. **探索阶段 (Stage 1):** 在可行域内随机生成 N 个“种子”，对每个种子执行少量迭代的 CMA-ES，快速筛选候选；
2. **开采阶段 (Stage 2):** 挑选性能最佳的若干“冠军解”，在其邻域内以更多迭代深度优化，充分逼近全局最优；

3. **高精度验证：**采用更细时间步长与更严格容差，对最优解进行复核，确保遮蔽时长计算稳定。
4. **角度剪枝与可行种子生成：**仅在接近目标最优朝向的极窄角域（约 179° – 180° ）内采样，并行随机选取飞行速度及三枚干扰弹的“投放时刻 + 起爆延迟”。对每枚弹进行一次性可行性测试：若在某引爆绝对时刻 t 能完全遮蔽目标，则继续搜索下一枚，并强制相邻两次投放至少间隔 1 s。
5. **参数编码（染色体设计）：**将决策变量编码为 8 维染色体：

$$[\text{方向}, \text{速度}, t_1, d_1, \text{gap}_2, d_2, \text{gap}_3, d_3],$$
 并可批量并行生成（规模可达 10^5 ）。
6. **目标函数与判定加速：**给定染色体 \rightarrow 解码为三次引爆的时空位置 \rightarrow 检查所有“导弹—目标离散点”视线是否同时满足“线段最近点 \leq 烟幕半径”的布尔条件。时间轴上先进行粗步长扫描，一旦出现状态跳变，再用二分逼近进入/退出时刻，最终累计所有遮蔽区间时长，作为适应度值。
7. **鲁棒性与多精度评估：** CMA-ES 对噪声和非光滑目标函数具有较强鲁棒性，结合“粗 \rightarrow 细”的多精度评估，能够在保证速度的同时获得稳定而准确的解。

6.3 优化结果与可视化

当种子数取 $N = 3000$ 时，得到的最优解如下：

- 最大有效遮蔽时长：7.6086 s；
- 无人机参数：飞行速度 $v_U = 140.00$ m/s，航向 $\theta = 179.65^\circ$ ；
- 干扰弹投放序列：
 1. 第 1 枚：投放 $t = 0.001$ s，起爆 $T = 3.619$ s；
 2. 第 2 枚：投放 $t = 3.612$ s，起爆 $T = 5.382$ s；
 3. 第 3 枚：投放 $t = 5.582$ s，起爆 $T = 6.028$ s。

将三枚干扰弹的投放与起爆时序汇总如表 7。

表 7: FY1 三弹投放策略时序参数（问题三）

编号	投放时刻 t_i [s]	引信延时 d_i [s]	起爆绝对时刻 T_i [s]
1	0.001	3.619	3.620
2	3.612	5.382	8.994
3	5.582	6.028	11.610

为直观展示最优解对应的空间几何关系，绘制 FY1 与 M1 的 XY 平面示意图（三断轴：全局 / 释放区 / 引爆区）。图 7 中，FY1 起始点以蓝色圆点标出，释放点为橙色 X（带文字“Release”），引爆点为红色三角（无文字）。

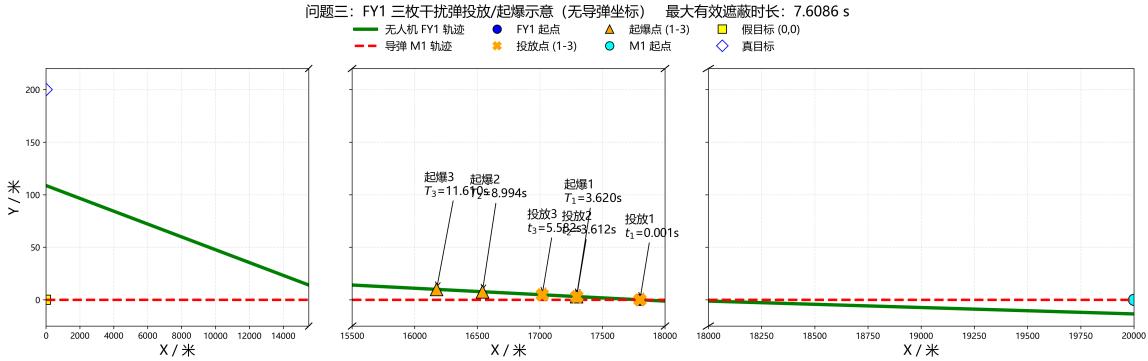


图 7：问题三：FY1 三弹投放策略 XY 平面示意图（含全局、释放区、引爆区断层放大）

最终总有效遮蔽时长达到

$$7.6086 \text{ s},$$

显著优于问题二单弹方案的 4.5884 s。

6.4 小结

问题三表明，多枚干扰弹的**时序协同投放**能显著延长导弹的遮蔽时间。在最优解下，FY1 以高速飞行并在极短时间窗口内连续投放三弹，使遮蔽时长由单弹的 4.5884 s 提升至 7.6086 s，提升幅度约 66%。这验证了多弹协同的显著收益；同时，种子数的对比实验也显示，增加 N 可提高解的质量，但计算代价更高，需在精度与效率之间权衡。

7 问题四：三机协同投放策略优化

7.1 优化建模

问题四进一步扩展至多无人机协同，考虑 FY1、FY2、FY3 各投放 1 枚干扰弹，通过时间和空间上的合理配合，以延长导弹 M1 的总遮蔽时长。决策向量设为

$$\mathbf{x} = (\theta_1, v_1, t_1, d_1, \theta_2, v_2, t_2, d_2, \theta_3, v_3, t_3, d_3),$$

其中 (θ_i, v_i) 为第 i 架无人机的飞行方向和速度， t_i 为投放时刻， d_i 为引信延时。目标函数仍为导弹 M1 的总遮蔽时长。为求解该优化问题，我们采用两阶段 CMA-ES 算法，在全局搜索与局部精细化的结合下获得最优策略。

7.2 数值求解与稳健性

整体求解流程分为两个层次：

1. **遮蔽判定 (get_obscuration_state_at_time)**: 给定时刻 t , 先根据式(4)生成当前所有有效烟幕球心位置, 再由(1)计算导弹位置; 逐一检查导弹到目标 24 个离散点的视线线段, 计算该线段到各烟幕球心的最短距离。当且仅当所有 24 条视线均被至少一个半径 $r_s = 10 \text{ m}$ 的烟幕球覆盖时, 判定目标整体被遮蔽。最近点距离采用“点投影到线段并截断”的标准公式实现。
2. **时间扫描 + 二分精化**: 在区间 $[0, t_{\max}]$ 以粗步长 (如 $\Delta t = 10^{-3} \text{ s}$) 扫描遮蔽状态, 一旦发现判定函数 $g(t)$ 发生符号变化或遮蔽状态跳变, 就在对应子区间内使用二分法 (精度 10^{-5} s) 求出精确的进入/退出时刻。由此得到每个遮蔽区间并累计总时长。
3. **贡献分解 (calculate_grenade_contributions)**: 为分析可解释性, 在每个遮蔽小区间取中点 t_{mid} , 反查此刻哪些烟幕球对 24 条视线的遮断“有贡献”, 并将该区间时长累加到相应干扰弹名下。这样不仅能给出总遮蔽时长, 还能量化每枚干扰弹的贡献度。

7.3 最终优化结果

运行优化程序后, 得到的最优策略如下: 导弹 M1 的总有效遮蔽时长达到

11.8025 s,

较问题三的单机三弹方案 (7.61 s) 提升约 55%。

具体而言, 无人机 FY1 以航向角 178.48° 、速度 89.69 m/s 飞行, 在任务开始后 0.294 s 投放干扰弹, 并在 2.846 s 后起爆。该弹的投放点为 $(17773.68, 0.70, 1800.00)$, 起爆点为 $(17518.53, 7.46, 1760.32)$ 。

无人机 FY2 以航向角 229.79° 、速度 99.74 m/s 飞行, 在 9.973 s 时投放干扰弹, 经过 7.871 s 延时后起爆。其投放点为 $(11357.85, 640.35, 1400.00)$, 起爆点为 $(10851.05, 40.81, 1096.43)$ 。

无人机 FY3 以航向角 88.28° 、速度 134.07 m/s 飞行, 在 19.062 s 投放干扰弹, 并在 4.087 s 后起爆。投放点为 $(6076.58, -445.51, 700.00)$, 起爆点为 $(6093.00, 102.17, 618.16)$ 。

整理结果如表 8 所示。

无人机	航向角 [deg]	速度 [m/s]	投放时刻 [s]	起爆延时 [s]	起爆坐标 (x,y,z)
FY1	178.48	89.69	0.294	2.846	$(17518.53, 7.46, 1760.32)$
FY2	229.79	99.74	9.973	7.871	$(10851.05, 40.81, 1096.43)$
FY3	88.28	134.07	19.062	4.087	$(6093.00, 102.17, 618.16)$

表 8: 三机协同投放策略参数 (问题四)

7.4 可视化结果

为直观展示三机协同投放的效果，图 8 给出了问题四的 XY 平面投放策略示意图，其中清晰标明了无人机起始点、投放点、起爆点、导弹 M1 的轨迹以及真假目标的位置与符号含义。

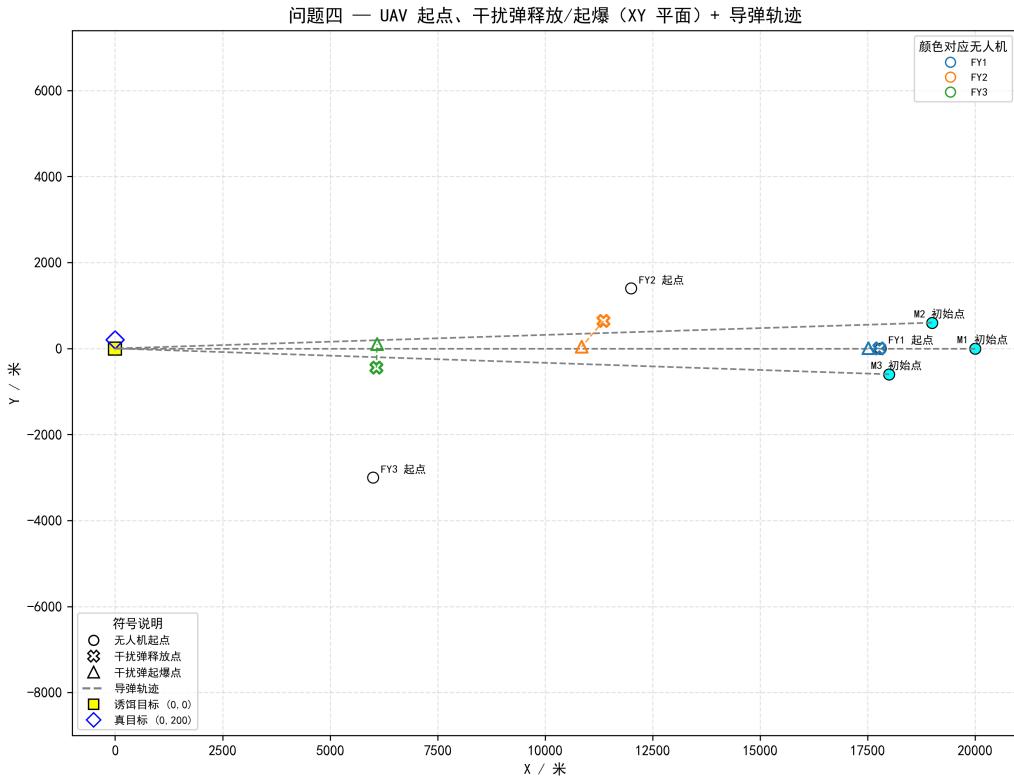


图 8: 问题四的 XY 平面投放策略示意图 (含符号说明)

7.5 对比与小结

图 9 给出了问题三与问题四的性能对比。可以看出，多机协同投放使遮蔽时长由 7.61 s 显著提升至 11.80 s，体现了空间分布与时间错峰带来的优势。

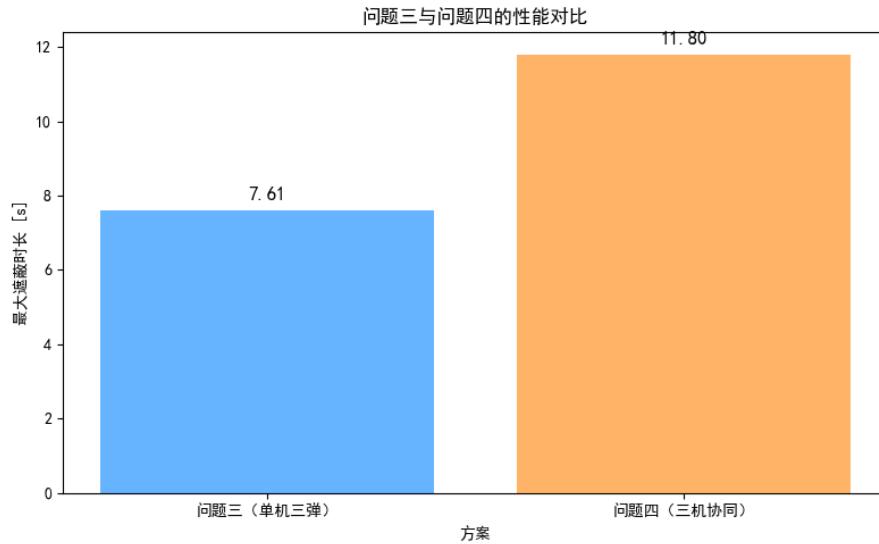


图 9: 问题三（单机三弹）与问题四（三机协同）的性能对比

综上，问题四表明：在战术资源允许的情况下，多机协同投放优于单机多弹，能更充分利用烟幕时效与空间覆盖，提高真目标的整体防护效果。

8 问题五：五机多弹协同投放策略优化

8.1 问题重述与建模

问题五将场景扩展至最终的完备情形：五架无人机（FY1–FY5）协同作战，每架无人机至多投放 3 枚干扰弹，需要同时对三枚来袭导弹（M1、M2、M3）实施有效遮蔽。问题的复杂度呈指数级增长，决策变量的总维度达到 $5 \times (2 + 2 \times 3) = 40$ 维。

为了解决高维度的复杂优化问题，我们提出了一种借鉴量子化学中处理多电子体系的哈特里-福克自洽场（Hartree-Fock Self-Consistent Field, HF-SCF）思想的多阶段协同优化算法。该算法将全局多变量、强耦合的复杂优化问题分解为针对单架无人机的子问题进行迭代求解。首先，通过贪心算法与高质量的种子生成策略构建初始方案，其中特别针对初始位置特殊的无人机 FY4 采用了适应性的种子筛选标准。随后，在自洽场迭代框架下，采用 CMA-ES 对各无人机策略进行轮流精细优化，以最大化三枚导弹的总有效遮蔽时长为目标。通过多轮迭代，最终得到一个高效的协同干扰策略，实现了 47.7814 秒的总有效遮蔽时长，并验证了模型在处理异构约束时的鲁棒性。

我们将每架无人机的策略参数化为一个 8 维向量：

$$\mathbf{x}_i = (\theta_i, v_i, t_{i,1}, d_{i,1}, g_{i,2}, d_{i,2}, g_{i,3}, d_{i,3}), \quad i = 1, \dots, 5.$$

其中 (θ_i, v_i) 为第 i 架无人机的航向与速度； $t_{i,1}, d_{i,1}$ 为其首枚弹的投放时刻与引信延时； $g_{i,j}, d_{i,j}$ 为后续弹的投放间隔与引信延时。总决策向量 $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_5)$ 。

优化目标是最大化三枚导弹的总遮蔽时长之和：

$$\max_{\mathbf{X}} J(\mathbf{X}) = \sum_{k=1}^3 T_{\text{obscured}}(\mathbf{M}_k | \mathbf{X}).$$

8.2 求解策略：协同策略合成框架

考虑到问题的高维度、强耦合与非凸特性，我们设计了一套精巧的“协同策略合成框架”，该框架借鉴了计算化学中的“哈特里-福克自治场”(HF-SCF)思想，结合了全局构造与局部精调，以高效地探索解空间。

1. 阶段一：多顺序贪心拼装

为获得高质量的初始解，我们不采用简单的随机生成，而是设计了一种贪心构造方法。首先，计算有效的飞行角度并进行剪枝，对每个无人机生成大量单个无人机投放策略，构成候选池。然后随机生成 N 种不同的无人机执行顺序（例如，[FY3, FY1, FY5, FY2, FY4] 是一种顺序）。对每一种顺序，按序为无人机“贪心”地从预先生成的候选池中分配最佳策略。在为第 k 架无人机决策时，前 $k-1$ 架无人机的策略被视为固定背景，此时选择能使全局累积遮蔽时长增量最大的策略。遍历所有 N 种顺序后，选择总时长最高的拼装方案作为初始解。这一步骤确保了初始解具备良好的协同基础。

2. 阶段二：HF-SCF 交替微调 (Alternating Refinement via SCF)

在获得高质量初始解后，我们引入自治场迭代进行精细优化。在每一轮 (Round) 迭代中，我们按固定顺序 ($\text{FY1} \rightarrow \text{FY2} \rightarrow \dots \rightarrow \text{FY5}$) 依次优化每一架无人机。当优化无人机 i 时，我们“冻结”其他四架无人机的策略参数，将其视为一个静态的“外部势场”。在此背景下，我们使用 CMA-ES 算法，在无人机 i 自身的 8 维决策空间内进行局部搜索，以期找到能与当前“外部势场”最佳配合的新策略。更新无人机 i 的策略后，继续处理下一架，直至完成一轮。

该过程被重复多轮，直到连续两轮的总遮蔽时长增益低于预设阈值 0.01 秒，判定算法收敛并提前终止。这种“冻结-优化-更新”的交替迭代模式，使得各无人机策略在相互影响下逐步优化，从而逼近一个最优的协同作战平衡策略。

实际运行中，由于优化算法包含一部分随机性，遮蔽时长收敛后仍可能存在优化潜力，我们为一个寻找到的质量较好的策略进行了大量的优化参数调整和多轮重新优化，得到了最终的优化结果。

8.3 最终优化结果

通过上述协同策略合成框架，我们得到了最终的五机多弹联合投放策略。该策略实现了对三枚来袭导弹的有效干扰，总遮蔽时长达到

47.7814 s,

其中，M1、M2、M3 各自获得了显著的遮蔽时间。各无人机的具体飞行与投放参数详见表 9。

无人机	航向角 [deg]	速度 [m/s]	弹号	投放时刻 [s]	起爆延时 [s]	主要打击目标
FY1	179.65	139.76	G1	0.005	3.601	M1
			G2	3.683	5.377	M1
			G3	5.591	6.045	M1
FY2	230.96	115.28	G1	5.583	5.572	M2
			G2	7.216	7.833	M1
			G3	11.872	7.501	M3
FY3	87.55	121.14	G1	21.646	3.907	M1
			G2	21.882	3.557	M1
			G3	25.437	1.630	M2
FY4	311.67	137.69	G1	6.251	9.232	M2
			G2	9.073	10.257	M1
			G3	12.459	0.528	—
FY5	130.95	137.56	G1	11.896	3.816	M3
			G2	13.763	5.672	M1
			G3	18.572	4.126	M2

表 9: 问题五投放策略参数

8.4 可视化与分析

如图 10 所示，展示了有效遮蔽时间的时间轴。可见算法优化出的投放策略造成的遮蔽时间呈现接力式的布局，除 FY1 外每架无人机均将遮蔽时间分布在不同导弹中，达成长时间连续遮蔽的同时尽量避免了遮蔽时间的重叠。

图 11 直观展示了该复杂协同策略在 XY 平面上的几何布局。五架无人机从不同初始位置出发，通过精确计算的航线与时序，在三条不同的导弹攻击路径上，形成了多个时空交错的烟幕遮蔽区。可以看出，FY1 主要负责应对 M1；FY2 与 FY5 的三枚烟幕弹可分别干扰三枚不同导弹；而 FY3 与 FY4 干扰 M1 与 M2，体现了明确的战术分工与协同。

此外，注意到 FY4 由于位置距离导弹轨迹较远，难以做到三枚烟幕弹同时生效，因此仅有前两枚起到干扰作用，表中的第三枚导弹投放参数为随机生成的占位符。

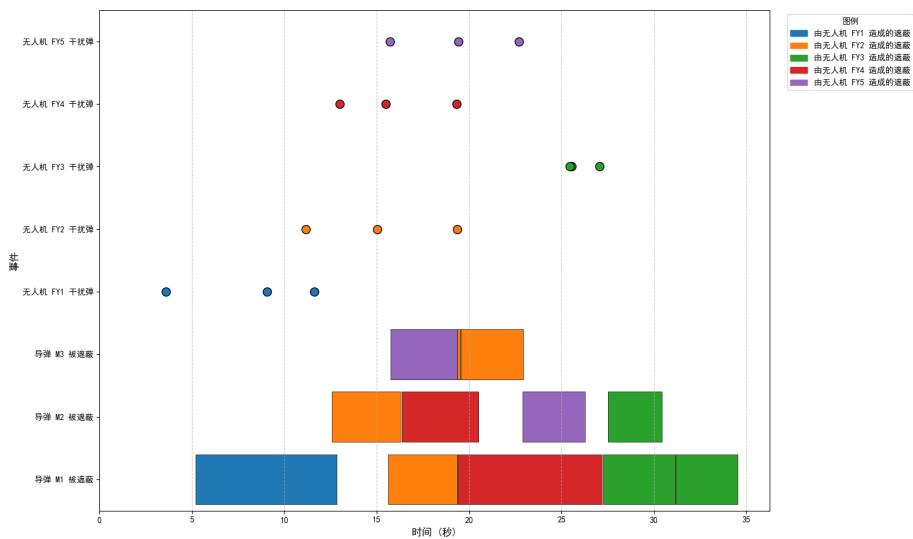


图 10: 有效遮蔽时间的时间轴

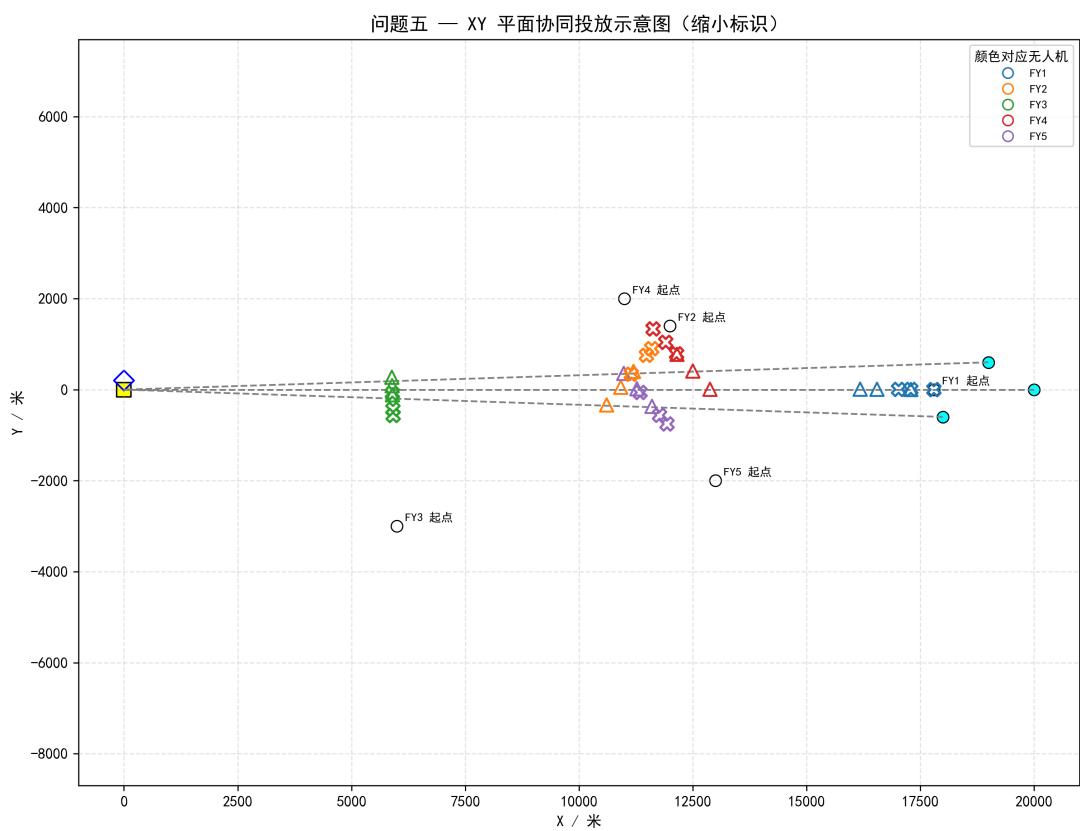


图 11: 复杂协同策略在 XY 平面上的几何布局

8.5 小结

与问题三（单机三弹，总时长 7.6086 s）和问题四（三机协同，总时长 11.8025 s）相比，问题五在五机十五弹（理论上限）的资源配置下，实现了总遮蔽时长 47.7814 s 的跨越式提升。这充分证明了在复杂多目标威胁场景下，通过精密的贪心排序种子选择，巧妙的 SCF 算法设计，强大的 CMA-ES 优化算法，可以有效整合多平台资源，实现远超单元或小编队作战效能的“体系涌现”效应，为真目标的生存提供了坚实的保障。

参考文献

- [1] Nelder J A, Mead R. A simplex method for function minimization[J]. *The Computer Journal*, 1965, 7(4): 308–313.
- [2] McKinnon K I M. Convergence of the Nelder–Mead simplex method to a nonstationary point[J]. *SIAM Journal on Optimization*, 1998, 9(1): 148–158.
- [3] Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies[J]. *Evolutionary Computation*, 2001, 9(2): 159–195.
- [4] Hansen N. The CMA evolution strategy: A tutorial[EB/OL]. arXiv:1604.00772, 2016.
- [5] Rechenberg I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*[M]. Stuttgart: Frommann-Holzboog, 1973.
- [6] Beyer H G, Schwefel H P. Evolution strategies – A comprehensive introduction[J]. *Natural Computing*, 2002, 1(1): 3–52.
- [7] Martí R. Multi-start methods[C]// Glover F, Kochenberger G. *Handbook of Metaheuristics*. Boston: Springer, 2003: 355–368.
- [8] Press W H, Teukolsky S A, Vetterling W T, Flannery B P. *Numerical Recipes: The Art of Scientific Computing*[M]. 3rd ed. Cambridge: Cambridge University Press, 2007.
- [9] Ericson C. *Real-Time Collision Detection*[M]. Amsterdam: Morgan Kaufmann / ScienceDirect, 2005.
- [10] Jin Y, Branke J. Evolutionary optimization in uncertain environments – A survey[J]. *IEEE Transactions on Evolutionary Computation*, 2005, 9(3): 303–317.
- [11] 祁永强. 数学建模 [M]. 北京: 科学出版社, 2020. ISBN 9787030637000. (五一数学建模竞赛推荐教材)
- [12] ChatGPT. ChatGPT-5-Thinking. OpenAI, 2025-09-06.
- [13] Gemini. Gemini-2.5-pro-0617. Google, 2025-09-06.

附录目录

附录编号	对应题目	文件名
A1	题目一	calculate_time.py
A2	题目二	optimizer2_1.py optimizer2_2.java
A3	题目三	optimizer3.cpp
A4	题目四	optimizer4.cpp
A5	题目五	optimizer5.java

A 附录 A: 问题一 python 实现

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.patches as mpatches
4
5 G = 9.8
6 MISSILE_SPEED_MPS = 300
7 SMOKE_CLOUD_RADIUS_M = 10
8 SMOKE_EFFECTIVE_DURATION_S = 20
9 SMOKE_SINK_SPEED_MPS = 3
10
11 TARGET_BASE_CENTER = np.array([0, 200, 0])
12 TARGET_RADIUS = 7.0
13 TARGET_HEIGHT = 10.0
14
15 INITIAL_POSITIONS = {
16     "M1": np.array([20000, 0, 2000]), "M2": np.array([19000, 600,
17     2100]), "M3": np.array([18000, -600, 1900]),
18     "FY1": np.array([17800, 0, 1800]), "FY2": np.array([12000, 1400,
19     1400]), "FY3": np.array([6000, -3000, 700]),
20     "FY4": np.array([11000, 2000, 1800]), "FY5": np.array([13000,
21     -2000, 1300]),
22 }
23 FAKE_TARGET_POS = np.array([0, 0, 0])
24 MISSILE_DIRECTIONS = {m_id: (FAKE_TARGET_POS - pos) / np.linalg.norm(
25     FAKE_TARGET_POS - pos)

```

```
22             for m_id, pos in INITIAL_POSITIONS.items() if
23                 m_id.startswith("M")}
24
25 def generate_target_points(base_center, radius, height,
26     num_rim_points=12, num_height_levels=2):
27     points = []
28     angles = np.linspace(0, 2 * np.pi, num_rim_points, endpoint=False
29     )
30     heights = np.linspace(0, height, num_height_levels)
31     for h in heights:
32         for angle in angles:
33             points.append(
34                 [base_center[0] + radius * np.cos(angle), base_center
35                  [1] + radius * np.sin(angle), base_center[2] + h])
36     return np.array(points)
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53 REAL_TARGET_POINTS = generate_target_points(TARGET_BASE_CENTER,
    TARGET_RADIUS, TARGET_HEIGHT)

def _is_single_line_blocked(missile_pos, smoke_center, target_point):
    vec_AB = target_point - missile_pos
    vec_AC = smoke_center - missile_pos
    t_proj = np.dot(vec_AC, vec_AB) / np.dot(vec_AB, vec_AB)
    closest_point = missile_pos + np.clip(t_proj, 0, 1) * vec_AB
    return np.sum((smoke_center - closest_point) ** 2) <=
        SMOKE_CLOUD_RADIUS_M ** 2

def get_obscuration_state_at_time(t, missile_id, grenade_lifecycles):
    active_smoke_centers = []
    for grenade in grenade_lifecycles:
        t_detonate_abs = grenade['t_detonate_abs']
        if t_detonate_abs <= t < t_detonate_abs +
            SMOKE_EFFECTIVE_DURATION_S:
            time_since_detonation = t - t_detonate_abs
            current_center = grenade['p_detonate'] - np.array([0, 0,
                SMOKE_SINK_SPEED_MPS * time_since_detonation])
            active_smoke_centers.append(current_center)
    if not active_smoke_centers: return False
```

```
54     missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[  
55         missile_id] * MISSILE_SPEED_MPS * t  
56     for target_point in REAL_TARGET_POINTS:  
57         if not any(_is_single_line_blocked(missile_pos, center,  
58             target_point) for center in active_smoke_centers):  
59             return False  
60     return True  
61  
62 def find_transition_time(t_start, t_end, missile_id, start_state,  
63     grenade_lifecycles, precision):  
64     low, high = t_start, t_end  
65     while (high - low) > precision:  
66         mid = (low + high) / 2  
67         if get_obscuration_state_at_time(mid, missile_id,  
68             grenade_lifecycles) == start_state:  
69             low = mid  
70         else:  
71             high = mid  
72     return high  
73  
74 def print_grenade_trajectory_info(uav_flight_plans):  
75     print("--- 干扰弹投放与引爆坐标信息 ---")  
76     for uav_id, uav_data in uav_flight_plans.items():  
77         uav_speed = uav_data['uav_speed']  
78         direction_deg = uav_data['direction_deg']  
79         p_uav_initial = INITIAL_POSITIONS[uav_id]  
80         angle_rad = np.radians(direction_deg)  
81         v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed *  
82             np.sin(angle_rad), 0])  
83  
84         for i, grenade_strategy in enumerate(uav_data.get('  
85             grenade_strategy', [])):  
86             grenade_id = f"{uav_id}_G{i + 1}"  
87             t_release = grenade_strategy['t_release']  
88             t_detonate_after = grenade_strategy['t_detonate_after']  
89  
90             p_drop = p_uav_initial + v_uav * t_release  
91  
92             p_detonate = p_drop + np.array(  
93                 [v_uav[0] * t_detonate_after, v_uav[1] *  
94                  v_uav[2] * t_detonate_after, 0])
```

```
t_detonate_after, -0.5 * G * t_detonate_after **  
2]  
88 )  
89  
90     print(f"干扰弹 {grenade_id}:")  
91     print(f" - 投放坐标 (x, y, z): ({p_drop[0]:.2f}, {p_drop  
[1]:.2f}, {p_drop[2]:.2f})")  
92     print(f" - 引爆坐标 (x, y, z): ({p_detonate[0]:.2f}, {  
p_detonate[1]:.2f}, {p_detonate[2]:.2f})")  
93  
94 def get_obscurations_intervals(uav_flight_plans, all_missile_ids,  
coarse_time_step, precision, max_sim_time=100.0):  
95     grenade_lifecycles = []  
96     for uav_id, uav_data in uav_flight_plans.items():  
97         uav_speed, direction_deg = uav_data['uav_speed'], uav_data['  
direction_deg']  
98         p_uav_initial, angle_rad = INITIAL_POSITIONS[uav_id], np.  
radians(direction_deg)  
99         v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed *  
np.sin(angle_rad), 0])  
100        for gs in uav_data.get('grenade_strategy', []):  
101            t_release, t_detonate_after = gs['t_release'], gs['  
t_detonate_after']  
102            p_drop = p_uav_initial + v_uav * t_release  
103            p_detonate = p_drop + np.array(  
104                [v_uav[0] * t_detonate_after, v_uav[1] *  
t_detonate_after, -0.5 * G * t_detonate_after **  
2])  
105            grenade_lifecycles.append({'p_detonate': p_detonate, '  
t_detonate_abs': t_release + t_detonate_after})  
106  
107    obscurations_intervals = {m_id: [] for m_id in all_missile_ids}  
108    for missile_id in all_missile_ids:  
109        last_time, last_state = 0.0, get_obscurations_state_at_time  
(0.0, missile_id, grenade_lifecycles)  
110        obscuration_start_time = 0.0 if last_state else None  
111        for i in range(1, int(max_sim_time / coarse_time_step) + 1):  
112            current_time = i * coarse_time_step  
113            current_state = get_obscurations_state_at_time(  
current_time, missile_id, grenade_lifecycles)
```

```
114     if current_state != last_state:
115         transition_t = find_transition_time(last_time,
116                                             current_time, missile_id, last_state,
117                                             grenade_lifecycles,
118                                             precision)
119
120         if current_state:
121             obscuration_start_time = transition_t
122         elif obscuration_start_time is not None:
123             obscuration_intervals[missile_id].append((
124                 obscuration_start_time, transition_t))
125             obscuration_start_time = None
126
127     last_time, last_state = current_time, current_state
128
129     if obscuration_start_time is not None:
130         obscuration_intervals[missile_id].append((
131             obscuration_start_time, max_sim_time))
132
133     return obscuration_intervals
134
135
136 def _get_contributing_grenades_at_time(t, missile_id,
137                                         grenade_lifecycles_with_id):
138
139     missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[
140         missile_id] * MISSILE_SPEED_MPS * t
141
142     active_grenades = []
143
144     for grenade in grenade_lifecycles_with_id:
145         t_detonate_abs = grenade['t_detonate_abs']
146
147         if t_detonate_abs <= t < t_detonate_abs +
148             SMOKE_EFFECTIVE_DURATION_S:
149
150             time_since_detonation = t - t_detonate_abs
151
152             current_center = grenade['p_detonate'] - np.array([0, 0,
153                 SMOKE_SINK_SPEED_MPS * time_since_detonation])
154
155             active_grenades.append({'id': grenade['id'], 'center':
156                 current_center})
157
158     if not active_grenades: return []
159
160     for target_point in REAL_TARGET_POINTS:
161
162         if not any(_is_single_line_blocked(missile_pos, g['center'],
163                                           target_point) for g in active_grenades):
164
165             return []
166
167     contributing_grenades = []
168
169     for g in active_grenades:
170
171         for target_point in REAL_TARGET_POINTS:
172
173             if _is_single_line_blocked(missile_pos, g['center'],
```

```
        target_point):
    contributing_grenades.append(g['id'])
    break
return contributing_grenades

def calculate_grenade_contributions(uav_flight_plans,
coarse_time_step=0.1, precision=0.001, max_sim_time=100.0):
    print(f"\n--- 开始计算每个干扰弹的贡献 (粗略步长: {coarse_time_step}s, 精度: {precision}s) ---")
    all_missile_ids = ["M1", "M2", "M3"]
    grenade_lifecycles, grenade_ids = [], []
    for uav_id, uav_data in uav_flight_plans.items():
        uav_speed, direction_deg = uav_data['uav_speed'], uav_data['direction_deg']
        p_uav_initial, angle_rad = INITIAL_POSITIONS[uav_id], np.radians(direction_deg)
        v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed * np.sin(angle_rad), 0])
        for i, gs in enumerate(uav_data.get('grenade_strategy', [])):
            t_release, t_detonate_after = gs['t_release'], gs['t_detonate_after']
            p_drop = p_uav_initial + v_uav * t_release
            p_detonate = p_drop + np.array([
                v_uav[0] * t_detonate_after, v_uav[1] * t_detonate_after, -0.5 * G * t_detonate_after ** 2])
            t_detonate_abs = t_release + t_detonate_after
            grenade_id = f"{uav_id}_G{i + 1}"
            grenade_lifecycles.append({'id': grenade_id, 'p_detonate': p_detonate, 't_detonate_abs': t_detonate_abs})
            grenade_ids.append(grenade_id)

    event_times = {0.0, max_sim_time}
    for g in grenade_lifecycles:
        event_times.add(g['t_detonate_abs'])
        event_times.add(g['t_detonate_abs'] +
                        SMOKE_EFFECTIVE_DURATION_S)

plain_lifecycles = [{p_detonate: g[p_detonate], 't_detonate': g['t_detonate'], 't_detonate_abs': g['t_detonate_abs']} for g in grenade_lifecycles]
```

```
    t_detonate_abs': g['t_detonate_abs']} for g in
173                      grenade_lifecycles]
174
175 for missile_id in all_missile_ids:
176     last_time, last_state = 0.0, get_obscuration_state_at_time(
177         (0.0, missile_id, plain_lifecycles))
178
179     for i in range(1, int(max_sim_time / coarse_time_step) + 1):
180         current_time = i * coarse_time_step
181         current_state = get_obscuration_state_at_time(
182             (current_time, missile_id, plain_lifecycles))
183
184         if current_state != last_state:
185             event_times.add(
186                 find_transition_time(last_time, current_time,
187                     missile_id, last_state, plain_lifecycles,
188                     precision))
189
190     last_time, last_state = current_time, current_state
191
192 unique_times = sorted([t for t in event_times if t <=
193                         max_sim_time])
194
195 grenade_contributions = {g_id: {m_id: 0.0 for m_id in
196                                 all_missile_ids} for g_id in grenade_ids}
197
198 for i in range(len(unique_times) - 1):
199     t_start, t_end = unique_times[i], unique_times[i + 1]
200     interval_duration = t_end - t_start
201
202     if interval_duration <= precision / 10: continue
203     t_mid = (t_start + t_end) / 2.0
204
205     for missile_id in all_missile_ids:
206         for g_id in _get_contributing_grenades_at_time(t_mid,
207             missile_id, grenade_lifecycles):
208             grenade_contributions[g_id][missile_id] +=
209                 interval_duration
210
211
212 print("--- 干扰弹贡献分析结果 ---")
213 final_results = {}
214
215 for g_id, missile_times in grenade_contributions.items():
216     total_time = sum(missile_times.values())
217
218     if total_time > precision:
219         final_results[g_id] = missile_times
220         final_results[g_id]['total'] = total_time
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
```



```
                grenade_strategy', [])]
236     if detonation_times: last_event_time = max(last_event_time,
237         max(detonation_times))
238     ax.plot(detonation_times, [y_pos[len(missile_ids) + i]] * len
239             (detonation_times), 'o', ms=8, color=uav_colors[i])
240
241     for intervals in obscuration_intervals.values():
242         if intervals: last_event_time = max(last_event_time, max(e
243             for s, e in intervals))
244
245     ax.set_yticks(y_pos)
246     ax.set_yticklabels(y_labels)
247     ax.set_xlabel("时间 (秒)", ax.set_ylabel("事件"), ax.set_title(""
248         干扰弹引爆与导弹遮蔽时间轴")
249     ax.grid(True, axis='x', linestyle='--', alpha=0.6)
250     ax.set_xlim(0, last_event_time * 1.05), ax.set_ylim(-0.5, len(
251         y_labels) - 0.5)
252
253     handles = [mpatches.Patch(color=c, label=f'导弹 {m_id} 遮蔽时段')
254         for i, (m_id, c) in
255             enumerate(zip(missile_ids, missile_colors))]
256     handles += [
257         plt.Line2D([0], [0], marker='o', color='w', label=f'无人机 {
258             u_id} 引爆', markerfacecolor=c, markersize=8) for
259             i, (u_id, c) in enumerate(zip(uav_ids, uav_colors))]
260     ax.legend(handles=handles, bbox_to_anchor=(1.02, 1), loc='upper
261             left')
262     plt.tight_layout()
263     plt.show()
264
265     uav_flight_plans = {
266         "FY1": {
267             "direction_deg": 180,
268             "uav_speed": 120,
269             "grenade_strategy": [
270                 {"t_release": 1.5, "t_detonate_after": 3.6}
271             ]
272         },
273         "FY2": {
274             "direction_deg": 0.0,
```

```
267     "uav_speed": 70.0,
268     "grenade_strategy": []
269 },
270 "FY3": {
271     "direction_deg": 0.0,
272     "uav_speed": 70.0,
273     "grenade_strategy": []
274 },
275 "FY4": {
276     "direction_deg": 0.0,
277     "uav_speed": 70.0,
278     "grenade_strategy": []
279 },
280 "FY5": {
281     "direction_deg": 0.0,
282     "uav_speed": 70.0,
283     "grenade_strategy": []
284 }
285 }
286
287 all_missile_ids = ["M1", "M2", "M3"]
288 coarse_time_step = 0.001
289 precision = 0.00001
290 max_sim_time = 100.0
291
292 print_grenade_trajectory_info(uav_flight_plans)
293
294 print("\n--- 计算导弹总遮蔽时长 ---")
295 intervals = get_obscurcation_intervals(uav_flight_plans,
296                                         all_missile_ids, coarse_time_step, precision, max_sim_time)
297 for m_id, interval_list in intervals.items():
298     total_time = sum(end - start for start, end in interval_list)
299     print(f"导弹 {m_id} 的总有效遮蔽时长为: {total_time:.6f} 秒")
300
301 calculate_grenade_contributions(uav_flight_plans, coarse_time_step,
302                                   precision, max_sim_time)
303
304 print("\n--- 生成时间轴图表 ---")
305 plot_timeline_chinese(intervals, uav_flight_plans)
```

Listing 1: 问题一: FY1 对 M1 的遮蔽时段计算

B 附录 B: 问题二程序实现

为验证与对比不同优化方法, 本附录给出问题二的两种算法实现:

1. Java 语言实现的 Nelder–Mead 单纯形法;
2. Python 语言实现的 CMA-ES 两阶段优化。

B.1 Java 实现 (Nelder–Mead 单纯形法)

```

1
2 import java.util.*;
3
4 // ===== 向量 =====
5 class Vec {
6     double x,y,z;
7     Vec(double x,double y,double z){this.x=x;this.y=y;this.z=z;}
8     Vec add(Vec b){return new Vec(x+b.x,y+b.y,z+b.z);}
9     Vec sub(Vec b){return new Vec(x-b.x,y-b.y,z-b.z);}
10    Vec mul(double k){return new Vec(x*k,y*k,z*k);}
11    double dot(Vec b){return x*b.x + y*b.y + z*b.z;}
12    double norm(){return Math.sqrt(dot(this));}
13    Vec unit(){ double n=norm(); return new Vec(x/n,y/n,z/n); }
14    static double dist(Vec a, Vec b){ return a.sub(b).norm(); }
15    public String toString(){ return String.format("%.3f, %.3f, %.3f",
16                                         x,y,z); }
17
18 public class Main {
19
20     // ===== 场景常量 =====
21     static final double g = 9.8;
22     static final double rCloud = 10.0;
23     static final double vSink = 3.0;
24     static final double window = 20.0;           // 云团有效 20 s
25     static final double vM = 300.0;             // 导弹速度
26     static final Vec FALSE_T = new Vec(0,0,0);

```

```

27     static final Vec M0 = new Vec(20000,0,2000);
28     static final Vec F0 = new Vec(17800,0,1800);
29     static final Vec uM = M0.mul(-1.0/M0.norm()); // 导弹朝原点
30     static final double tMissileEnd = M0.norm()/vM; // 66.7 s
31
32     // ====== 决策变量边界 (关键修正) ======
33     static final double VMIN=70, VMAX=140;
34     static final double TDROP_MIN=0.5, FUSE_MIN=0.2; // 避免零边界
35     static final double TDROP_MAX=12.0, FUSE_MAX=8.0; // 可按需放
36     宽, 但要满足 tDrop+fuse < tMissileEnd
37
38     // ====== 目标 12 点 (圆柱: R=7, H=10) ======
39     static List<Vec> initTarget12(){
40         List<Vec> pts = new ArrayList<>();
41         double R=7.0, H=10.0;
42         pts.add(new Vec(0,200,0));
43         pts.add(new Vec(0,200,H));
44         int n=5;
45         for(int i=0;i<n;i++){
46             double th=2*Math.PI*i/n;
47             double dx=R*Math.cos(th), dy=R*Math.sin(th);
48             pts.add(new Vec(dx,200+dy,0));
49             pts.add(new Vec(dx,200+dy,H));
50         }
51         return pts;
52     }
53     static final List<Vec> TARGET = initTarget12();
54
55     // ====== 运动学 ======
56     static Vec missilePos(double t){ return M0.add(uM.mul(vM*t)); }
57     static Vec uavPos(double t, double vF, double theta){
58         Vec u = new Vec(Math.cos(theta), Math.sin(theta), 0).unit();
59         return F0.add(u.mul(vF*t));
60     }
61     static Vec bombPos(double t, double tDrop, double vF, double
62     theta){
63         if(t < tDrop) return new Vec(Double.NaN,Double.NaN,Double.NaN
64         );
65         Vec u = new Vec(Math.cos(theta), Math.sin(theta), 0).unit();
66         Vec Pdrop = uavPos(tDrop, vF, theta);
67     }

```

```

64     double dt = t - tDrop;
65     return Pdrop.add(u.mul(vF*dt)).add(new Vec(0,0,-0.5*g*dt*dt))
66     ;
66 }
67 static Vec cloudCenter(double t, double tDrop, double fuse,
68     double vF, double theta){
69     double tExp = tDrop + fuse;
70     if(t < tExp) return new Vec(Double.NaN,Double.NaN,Double.NaN)
71     ;
70     Vec Cexp = bombPos(tExp, tDrop, vF, theta);
71     return Cexp.add(new Vec(0,0,-vSink*(t - tExp)));
72 }
73
74 // ===== 线段-点距离 =====
75 static double distSegPoint(Vec A, Vec B, Vec P){
76     Vec AB = B.sub(A), AP = P.sub(A);
77     double ab2 = AB.dot(AB);
78     if(ab2==0) return Vec.dist(P,A);
79     double s = AP.dot(AB)/ab2;
80     s = Math.max(0, Math.min(1, s));
81     Vec Q = A.add(AB.mul(s));
82     return Vec.dist(P, Q);
83 }
84
85 // ===== g(t): 12 点同时遮蔽的判据 =====
86 // g(t) = max_i( dist(segment[M(t)-T_i], C(t)) - rCloud )
87 // g(t) <= 0 表示当下时刻全部被遮蔽
88 static double g(double t, double tDrop, double fuse, double vF,
89     double theta){
90     double tExp = tDrop + fuse;
91     if(t < tExp || t > tExp + window) return +1e9;
92     Vec M = missilePos(t);
93     Vec C = cloudCenter(t, tDrop, fuse, vF, theta);
94     if(Double.isNaN(C.x)) return +1e9;
95     double maxVal = -1e99;
96     for(Vec Ti : TARGET){
97         double d = distSegPoint(M, Ti, C);
98         double val = d - rCloud;
99         if(val > maxVal) maxVal = val;
     }

```

```
100     return maxVal;
101 }
102
103 // ===== 精确遮蔽区间 (去掉“起爆点离目标远”的错误减枝)
104 =====
105
106 static List<double[]> coverIntervals(double tDrop, double fuse,
107                                         double vF, double theta){
108
109     List<double[]> segs = new ArrayList<>();
110
111     if(vF<VMIN || vF>VMAX) return segs;
112     if(tDrop<TDROP_MIN || fuse<FUSE_MIN) return segs;
113
114     double tExp = tDrop + fuse;
115
116     if(tExp > tMissileEnd) return segs;
117
118
119     double dt = 0.01, tol = 1e-4;
120     double prevT = t0, prevG = g(prevT, tDrop, fuse, vF, theta);
121     boolean inside = (prevG <= 0);
122     double entry = inside ? prevT : Double.NaN;
123
124
125     for(double t = t0 + dt; t <= t1 + 1e-12; t += dt){
126
127         double curG = g(t, tDrop, fuse, vF, theta);
128
129         if( (prevG > 0 && curG <= 0) || (prevG < 0 && curG >= 0) )
130             {
131
132                 double a = t - dt, b = t; // 二分定位交点
133                 double fa = g(a, tDrop, fuse, vF, theta), fb = curG;
134                 for(int it=0; it<90 && Math.abs(b-a)>tol; it++){
135
136                     double m = 0.5*(a+b);
137
138                     double fm = g(m, tDrop, fuse, vF, theta);
139
140                     if(fa*fm <= 0){ b=m; fb=fm; } else { a=m; fa=fm; }
141
142                 }
143
144             }
145
146             double tZero = 0.5*(a+b);
147
148             if(prevG > 0 && curG <= 0){ entry = tZero; inside =
149                 true; }
150
151             else if(prevG < 0 && curG >= 0){
152
153                 if(inside) segs.add(new double[]{entry, tZero});
154
155                 inside = false;
156
157             }
158
159
160
161
162
163
164 }
```

```

135         }
136         prevT = t; prevG = curG;
137     }
138     if(inside) segs.add(new double[]{entry, t1});
139     return segs;
140 }
141
142 static double coverDuration(double tDrop, double fuse, double vF,
143     double theta){
144     List<double[]> segs = coverIntervals(tDrop, fuse, vF, theta);
145     double sum = 0;
146     for(double[] s : segs) sum += Math.max(0, s[1]-s[0]);
147     return sum;
148 }
149 // ===== 目标函数 (最大化) =====
150 static double fitness(double vF, double theta, double tDrop,
151     double fuse){
152     if(vF<VMIN || vF>VMAX) return -1e9;
153     if(tDrop<TDROP_MIN || fuse<FUSE_MIN) return -1e9;
154     if(tDrop + fuse > tMissileEnd) return -1e9;
155     theta = clampAng(theta);
156     return coverDuration(tDrop, fuse, vF, theta);
157 }
158 // ===== Nelder - Mead (局部精修) =====
159 static class NM {
160     static class X { double v, th, td, fu; X(double v,double th,
161         double td,double fu){this.v=v;this.th=th;this.td=td;this.
162         fu=fu;} }
163     static X opt(X x0){
164         double dv=5, dth=Math.toRadians(12), dtd=0.6, dfu=0.6;
165         List<X> S = new ArrayList<>();
166         S.add(new X(x0.v, x0.th, x0.td, x0.fu));
167         S.add(new X(clamp(x0.v+dv,VMIN,VMAX), x0.th, x0.td, x0.fu
168             ));
169         S.add(new X(x0.v, clampAng(x0.th+dth), x0.td, x0.fu));
170         S.add(new X(x0.v, x0.th, clampTD(x0.td+dtd), x0.fu));
171         S.add(new X(x0.v, x0.th, x0.td, clampFU(x0.td, x0.fu+dfu
172             )));

```

```
169
170     int MAX=140; double alpha=1, gamma=2, rho=0.5, sigma=0.5;
171     for(int it=0; it<MAX; it++){
172         S.sort((a,b)->Double.compare(fitness(b.v,b.th,b.td,b.
173             fu), fitness(a.v,a.th,a.td,a.fu)));
174         double spread = Math.abs(S.get(0).v-S.get(S.size()-1)
175             .v)
176             + Math.abs(S.get(0).th-S.get(S.size()-1).th)
177             + Math.abs(S.get(0).td-S.get(S.size()-1).td)
178             + Math.abs(S.get(0).fu-S.get(S.size()-1).fu);
179         if(spread < 1e-3) break;
180
181         X best=S.get(0), worst=S.get(S.size()-1);
182         // 质心 (除最差点)
183         double cv=0, cth=0, ctd=0, cfu=0;
184         for(int i=0;i<S.size()-1;i++){ cv+=S.get(i).v; cth+=S
185             .get(i).th; ctd+=S.get(i).td; cfu+=S.get(i).fu; }
186         int m=S.size()-1; cv/=m; cth/=m; ctd/=m; cfu/=m;
187
188         // 反射
189         X xr = new X(
190             clamp(cv + alpha*(cv - worst.v), VMIN, VMAX),
191             clampAng(cth + alpha*(cth - worst.th)),
192             clampTD(ctd + alpha*(ctd - worst.td)),
193             clampFU(ctd + alpha*(ctd - worst.td), cfu +
194                 alpha*(cfu - worst.fu))
195         );
196
197         double fr = fitness(xr.v,xr.th,xr.td,xr.fu);
198         double fBest = fitness(best.v,best.th,best.td,best.fu
199             );
200         double fWorst = fitness(worst.v,worst.th,worst.td,
201             worst.fu);
202
203         if(fr >= fBest){
204             // 扩张
205             X xe = new X(
206                 clamp(cv + gamma*(cv - worst.v), VMIN,
207                     VMAX),
208                 clampAng(cth + gamma*(cth - worst.th)),
```

```

202         clampTD(ctd + gamma*(ctd - worst.td)),
203         clampFU(ctd + gamma*(ctd - worst.td), cfu
204             + gamma*(cfu - worst.fu))
205     );
206     double fe = fitness(xe.v, xe.th, xe.td, xe.fu);
207     S.set(S.size()-1, (fe>fr? xe : xr));
208 }else if(fr > fitness(S.get(S.size()-2).v,S.get(S.
209     size()-2).th,S.get(S.size()-2).td,S.get(S.size()
210     -2).fu)){
211     S.set(S.size()-1, xr);
212 }else{
213     // 内缩
214     X xc = new X(
215         clamp(cv + rho*(worst.v - cv), VMIN, VMAX
216             ),
217         clampAng(cth + rho*(worst.th - cth)),
218         clampTD(ctd + rho*(worst.td - ctd)),
219         clampFU(ctd + rho*(worst.td - ctd), cfu +
220             rho*(worst.fu - cfu))
221     );
222     double fc = fitness(xc.v, xc.th, xc.td, xc.fu);
223     if(fc > fWorst){
224         S.set(S.size()-1, xc);
225     }else{
226         // 收缩
227         for(int i=1;i<S.size();i++){
228             X si=S.get(i);
229             X shr = new X(
230                 clamp(best.v + sigma*(si.v-best.v
231                     ), VMIN, VMAX),
232                 clampAng(best.th + sigma*(si.th-
233                     best.th)),
234                 clampTD(best.td + sigma*(si.td-
235                     best.td)),
236                 clampFU(best.td + sigma*(si.td-
237                     best.td), best.fu + sigma*(si.
238                     fu-best.fu))
239             );
240             S.set(i, shr);
241         }

```

```
232         }
233     }
234   }
235   S.sort((a,b)->Double.compare(fitness(b.v,b.th,b.td,b.fu),
236           fitness(a.v,a.th,a.td,a.fu)));
237   return S.get(0);
238 }
239
240 // ====== 工具函数 ======
241 static double clamp(double x,double lo,double hi){ return Math.
242     max(lo, Math.min(hi, x)); }
243 static double clampAng(double th){
244     while(th<=-Math.PI) th+=2*Math.PI;
245     while(th> Math.PI) th-=2*Math.PI;
246     return th;
247 }
248 static double clampTD(double td){ return Math.max(TDROP_MIN, Math
249     .min(TDROP_MAX, td)); }
250 static double clampFU(double td, double fu){
251     fu = Math.max(FUSE_MIN, Math.min(FUSE_MAX, fu));
252     if(td + fu > tMissileEnd) fu = Math.max(FUSE_MIN, tMissileEnd
253         - td);
254     return fu;
255 }
256
257 // ====== 主程序 ======
258 public static void main(String[] args){
259     // 1) 轻量粗搜 (不做“起爆点到目标XY距离”的硬剪枝)
260     List<NM.X> seeds = new ArrayList<>();
261     PriorityQueue<double[]> pq = new PriorityQueue<>((a,b)->
262         Double.compare(b[0], a[0])); // [fit,v,th,td,fu]
263     for(double v=70; v<=140; v+=10){
264         for(double th=-Math.PI; th<=Math.PI; th+=Math.toRadians
265             (45)){
266             for(double td=1.0; td<=10.0; td+=1.0){
267                 for(double fu=0.5; fu<=6.0; fu+=0.5){
268                     if(td+fu > tMissileEnd) continue;
269                     double fit = fitness(v, th, td, fu);
270                     pq.add(new double[]{fit, v, th, td, fu});
271                 }
272             }
273         }
274     }
275 }
```

```

266         }
267     }
268 }
269 }
270 // 取前 K 个种子
271 int K = 12;
272 while(K-- > 0 && !pq.isEmpty()){
273     double[] a = pq.poll();
274     seeds.add(new NM.X(a[1], a[2], a[3], a[4]));
275 }
276 if(seeds.isEmpty()){
277     seeds.add(new NM.X(120, Math.PI, 1.5, 3.6)); // 朝向原点
278     的粗种子 ( )
279 }
280
281 // 2) 局部精修
282 NM.X best = null; double bestFit = -1e18;
283 for(NM.X s : seeds){
284     NM.X loc = NM.opt(s);
285     double f = fitness(loc.v, loc.th, loc.td, loc.fu);
286     if(f > bestFit){ best = loc; bestFit = f; }
287 }
288
289 // 3) 输出
290 double vBest=best.v, thBest=best.th, tdBest=best.td, fuBest=
291     best.fu;
292 List<double []> segs = coverIntervals(tdBest, fuBest, vBest,
293     thBest);
294 Vec Pd = uavPos(tdBest, vBest, thBest);
295 Vec Cexp = cloudCenter(tdBest+fuBest, tdBest, fuBest, vBest,
296     thBest);
297
298 System.out.println("== 问题2 优化结果 (12点 + Nelder - Mead,
299     修正版) ===");
300 System.out.printf("速度 vUAV = %.2f m/s\n", vBest);
301 System.out.printf("航向角 theta = %.2f 度\n", Math.toDegrees(
302     thBest));
303 System.out.printf("投放时刻 tDrop = %.3f s\n", tdBest);
304 System.out.printf("引信延时 fuse = %.3f s\n", fuBest);
305 System.out.printf("起爆时刻 tExp = %.3f s\n\n", tdBest+

```

```

        fuBest);

300    System.out.println("投放点 P_drop = " + Pd);
301    System.out.println("起爆点 C_exp = " + Cexp);
302
303    double total=0;
304    if(segs.isEmpty()){
305        System.out.println("\n无有效遮蔽区间。");
306    }else{
307        System.out.println("\n遮蔽区间 (s) : ");
308        for(double[] s : segs){
309            System.out.printf("  [% .6f , %.6f] 时长 = %.6f\n", s
310                           [0], s[1], (s[1]-s[0]));
311            total += (s[1]-s[0]);
312        }
313        System.out.printf("\n总遮蔽时长 = %.6f s\n", total);
314    }
315}

```

Listing 2: 问题二: 单机单弹投放参数优化 (Java 实现, 轻量粗搜 + Nelder–Mead 单纯形法, 12 点整体遮蔽判定)

B.1 B.2 Python 实现 (CMA-ES 两阶段优化)

Listing 3: 问题二: 单机单弹投放参数优化 (Python 实现, CMA-ES 两阶段优化, 全局搜索 + 高精度验证)

```

1
2 import numpy as np
3 import random
4 import math
5 from tqdm import tqdm
6 import cma
7 import pandas as pd
8 # --- 常量 ---
9 G = 9.8
10 MISSILE_SPEED_MPS = 300
11 SMOKE_CLOUD_RADIUS_M = 10
12 SMOKE_EFFECTIVE_DURATION_S = 20
13 SMOKE_SINK_SPEED_MPS = 3
14
15 # --- 目标几何 ---

```

```

16 TARGET_BASE_CENTER = np.array([0, 200, 0])
17 TARGET_RADIUS = 7.0
18 TARGET_HEIGHT = 10.0
19
20 # --- 初始位置与方向 ---
21 INITIAL_POSITIONS = {
22     "M1": np.array([20000, 0, 2000]), "M2": np.array([19000, 600, 2100]),
23     "M3": np.array([18000, -600, 1900]),
24     "FY1": np.array([17800, 0, 1800]), "FY2": np.array([12000, 1400, 1400]),
25     "FY3": np.array([6000, -3000, 700]),
26     "FY4": np.array([11000, 2000, 1800]), "FY5": np.array([13000, -2000,
27     1300]),
28 }
29
30 FAKE_TARGET_POS = np.array([0, 0, 0])
31 MISSILE_DIRECTIONS = {m_id: (FAKE_TARGET_POS - pos) /
32     np.linalg.norm(FAKE_TARGET_POS - pos)
33         for m_id, pos in INITIAL_POSITIONS.items() if
34             m_id.startswith("M")}
35
36 # --- 目标点集生成 ---
37 def generate_target_points(base_center, radius, height, num_rim_points=12,
38     num_height_levels=2):
39     points = []
40     angles = np.linspace(0, 2 * np.pi, num_rim_points, endpoint=False)
41     heights = np.linspace(0, height, num_height_levels)
42     for h in heights:
43         for angle in angles:
44             points.append(
45                 [base_center[0] + radius * np.cos(angle), base_center[1] +
46                  radius * np.sin(angle), base_center[2] + h])
47     return np.array(points)
48
49
50 REAL_TARGET_POINTS = generate_target_points(TARGET_BASE_CENTER, TARGET_RADIUS,
51     TARGET_HEIGHT)
52
53 # --- 核心物理检查函数 (无变化) ---
54 def _is_single_line_blocked(missile_pos, smoke_center, target_point):
55     vec_AB = target_point - missile_pos
56     vec_AC = smoke_center - missile_pos
57     t_proj = np.dot(vec_AC, vec_AB) / np.dot(vec_AB, vec_AB)
58     closest_point = missile_pos + np.clip(t_proj, 0, 1) * vec_AB
59     return np.sum((smoke_center - closest_point) ** 2) <= SMOKE_CLOUD_RADIUS_M

```

```
** 2
53
54
55 # --- 新增: 独立的、在特定时刻t检查遮蔽状态的函数 ---
56 def get_obscuration_state_at_time(t, missile_id, grenade_lifecycles):
57     """在精确的时刻t, 判断目标对指定导弹是否完全被遮蔽。"""
58     active_smoke_centers = []
59     for grenade in grenade_lifecycles:
60         t_detonate_abs = grenade['t_detonate_abs']
61         if t_detonate_abs <= t < t_detonate_abs + SMOKE_EFFECTIVE_DURATION_S:
62             time_since_detonation = t - t_detonate_abs
63             current_center = grenade['p_detonate'] - np.array([0, 0,
64                                         SMOKE_SINK_SPEED_MPS * time_since_detonation])
65             active_smoke_centers.append(current_center)
66
67     if not active_smoke_centers:
68         return False
69
70     missile_pos = INITIAL_POSITIONS[missile_id] +
71         MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t
72
73     for target_point in REAL_TARGET_POINTS:
74         is_point_blocked = any(
75             _is_single_line_blocked(missile_pos, center, target_point) for
76                 center in active_smoke_centers)
77
78     if not is_point_blocked:
79         return False # 只要有一个点可见, 就返回False
80     return True # 所有点都被遮蔽
81
82
83 # --- 二分法查找状态转换的精确时间 ---
84 def find_transition_time(t_start, t_end, missile_id, start_state,
85     grenade_lifecycles, precision):
86     """使用二分法在[t_start, t_end]区间内查找状态变化的精确时刻。"""
87     low, high = t_start, t_end
88     while (high - low) > precision:
89         mid = (low + high) / 2
90         mid_state = get_obscuration_state_at_time(mid, missile_id,
91             grenade_lifecycles)
92         if mid_state == start_state:
93             low = mid
94         else:
95             high = mid
96     return high
97
```

```

92
93 # --- 优化后的主仿真函数 ---
94 def calculate_synergistic_obscurcation_adaptive(uav_flight_plans,
95     all_missile_ids, coarse_time_step, precision,
96             max_sim_time=100.0):
97 """
98     使用自适应步长和二分法进行高效、高精度的协同干扰仿真。
99 """
100
101     # 步骤1: 预计算烟幕弹生命周期
102     grenade_lifecycles = []
103     for uav_id, uav_data in uav_flight_plans.items():
104         uav_speed = uav_data['uav_speed']
105         direction_deg = uav_data['direction_deg']
106         p_uav_initial = INITIAL_POSITIONS[uav_id]
107         angle_rad = np.radians(direction_deg)
108         v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed *
109                         np.sin(angle_rad), 0])
110         for grenade_strategy in uav_data.get('grenade_strategy', []):
111             t_release, t_detonate_after = grenade_strategy['t_release'],
112             grenade_strategy['t_detonate_after']
113             p_drop = p_uav_initial + v_uav * t_release
114             p_detonate = p_drop + np.array(
115                 [v_uav[0] * t_detonate_after, v_uav[1] * t_detonate_after, -0.5
116                  * G * t_detonate_after ** 2])
117             t_detonate_abs = t_release + t_detonate_after
118             grenade_lifecycles.append({'p_detonate': p_detonate,
119             't_detonate_abs': t_detonate_abs})
120
121     # 步骤2: 对每个导弹进行事件驱动的仿真
122     total_obsured_times = {m_id: 0.0 for m_id in all_missile_ids}
123     for missile_id in all_missile_ids:
124         last_time = 0.0
125         last_state = get_obscurcation_state_at_time(0.0, missile_id,
126             grenade_lifecycles)
127         obscuration_start_time = 0.0 if last_state else None
128
129         num_coarse_steps = int(max_sim_time / coarse_time_step)
130         for i in range(1, num_coarse_steps + 1):
131             current_time = i * coarse_time_step
132             current_state = get_obscurcation_state_at_time(current_time,
133                 missile_id, grenade_lifecycles)
134
135             if current_state != last_state:
136                 # 状态发生改变, 使用二分法精确定位
137                 transition_t = find_transition_time(last_time, current_time,

```

```

    missile_id, last_state, grenade_lifecycles,
130                               precision)

131
132     if current_state is True: # 从 False -> True
133         obscuration_start_time = transition_t
134     else: # 从 True -> False
135         if obscuration_start_time is not None:
136             total_obsured_times[missile_id] += (transition_t -
137                                         obscuration_start_time)
138             obscuration_start_time = None
139
140         last_time = current_time
141         last_state = current_state
142
143     # 如果仿真结束时仍然处于遮蔽状态，计算最后一段时长
144     if obscuration_start_time is not None:
145         total_obsured_times[missile_id] += (max_sim_time -
146                                         obscuration_start_time)
147
148
149 # --- 封装调用函数 ---
150 def calculate_time(uav_flight_plans, coarse_time_step=0.1, precision=0.001):
151     """
152     封装了对优化后仿真函数的调用。
153
154     Args:
155         uav_flight_plans (dict): 无人机飞行计划。
156         coarse_time_step (float):
157             粗略扫描的时间步长，值越大速度越快，但可能错过极短事件。
158         precision (float): 二分法查找的精度，即最终结果的时间分辨率。
159     """
160
161     final_results = calculate_synergistic_obscurcation_adaptive(
162         uav_flight_plans=uav_flight_plans,
163         all_missile_ids=["M1", "M2", "M3"], # 默认计算所有导弹
164         coarse_time_step=coarse_time_step,
165         precision=precision
166     )
167
168     # print(f"输入了 {sum(len(p.get('grenade_strategy', [])) for p in
169         uav_flight_plans.values())} 枚干扰弹的策略。")
170     # for missile_id, time in final_results.items():
171     #     print(f"导弹 {missile_id} 的总有效遮蔽时长为: {time:.4f} 秒")
172
173     return final_results

```

```
170
171 # -----
172 #       以上为物理模拟计算
173 # -----
174
175
176
177 def is_target_fully_obsured_at_time(t, missile_id, active_smoke_centers):
178     if not active_smoke_centers: return False
179     missile_pos = INITIAL_POSITIONS[missile_id] +
180                 MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t
181     for target_point in REAL_TARGET_POINTS:
182         if not any(_is_single_line_blocked(missile_pos, center, target_point)
183                    for center in active_smoke_centers):
184             return False
185     return True
186
187
188 def find_interception_details_for_grenade(uav_speed_vec, uav_initial_pos,
189                                             t_release_min, t_release_max,
190                                             missile_id="M1"):
191     """
192     一个更通用的函数，在给定的释放时间窗口内寻找一次有效的拦截。
193     返回 (t_release, t_detonate_after) 或 None。
194     """
195
196     # 随机化搜索顺序以避免偏差
197     t_release_candidates = list(np.linspace(t_release_min, t_release_max, 20))
198     random.shuffle(t_release_candidates)
199
200     for t_release in t_release_candidates:
201         p_drop = uav_initial_pos + uav_speed_vec * t_release
202
203         # 同样随机化引爆延迟的搜索
204         t_detonate_candidates = list(np.linspace(0, 8.0, 41))
205         random.shuffle(t_detonate_candidates)
206
207         for t_detonate_after in t_detonate_candidates:
208             t_intercept = t_release + t_detonate_after
209             p_smoke = p_drop + np.array([uav_speed_vec[0] * t_detonate_after,
210                                         uav_speed_vec[1] * t_detonate_after,
211                                         -0.5 * G * t_detonate_after ** 2])
212
213             # 使用高保真模型进行验证
214             if is_target_fully_obsured_at_time(t_intercept, missile_id,
215                                               [p_smoke]):
216                 return t_release, t_detonate_after
```

```
212     return None
213
214
215 def generate_holistic_seeds(num_seeds):
216     """
217     为问题2生成高质量的整体种子。
218     每个种子都确保FY1有一套有效的初始拦截策略。
219     """
220     print(f"正在为问题2生成{num_seeds}个高质量的整体协同种子...")
221     seeds = []
222     uav_ids = ["FY1"]
223
224     with tqdm(total=num_seeds, desc="生成P4种子") as pbar:
225         while len(seeds) < num_seeds:
226             current_seed_parts = []
227             success = True
228             for uav_id in uav_ids:
229                 p_uav_initial = INITIAL_POSITIONS[uav_id]
230
231                 # 尝试最多50次为单个无人机寻找有效策略
232                 found_strategy_for_uav = False
233                 for _ in range(50):
234                     direction_deg = random.uniform(0, 12)
235                     uav_speed = random.uniform(70, 140)
236                     angle_rad = np.radians(direction_deg)
237                     v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed *
238                                     np.sin(angle_rad), 0])
239
240                     grenade_details =
241                         find_interception_details_for_grenade(v_uav,
242                         p_uav_initial, 1.0, 40.0)
243
244                     if grenade_details:
245                         t_release, t_detonate_after = grenade_details
246                         current_seed_parts.extend([direction_deg, uav_speed,
247                                         t_release, t_detonate_after])
248                         found_strategy_for_uav = True
249                         break # 成功找到, 跳出对此无人机的尝试
250
251                     if not found_strategy_for_uav:
252                         success = False
253                         break # 如果有一个无人机找不到策略, 则整个种子失败, 重新开始
254
255             if success:
256                 seeds.append(current_seed_parts)
```

```
253         pbar.update(1)
254
255     print("问题4的高质量种子生成完毕! ")
256     return seeds
257
258
259 # --- 核心优化函数 (使用CMA-ES) - 可配置迭代次数版 ---
260 def optimize_with_cmaes(initial_solution, fitness_func, max_iterations=100):
261     """
262     从给定的初始解出发，使用CMA-ES进行优化。
263
264     Args:
265         initial_solution (list): 优化的起始点。
266         fitness_func (function): 适应度函数。
267         max_iterations (int): CMA-ES的最大迭代次数。
268
269     Returns:
270         tuple: (best_solution, best_fitness)
271     """
272
273     def objective_function(chromosome):
274         return -fitness_func(chromosome)
275
276     initial_sigma = 0.1
277     options = {
278         'bounds': [list(b) for b in zip(*BOUNDS_LIST)],
279         'maxiter': max_iterations, # 使用传入的参数
280         'verbose': -9
281     }
282
283     result = cma.fmin2(objective_function, initial_solution, initial_sigma,
284                         options)
285
286     best_solution = result[0]
287     best_fitness = -result[1].result.fbest
288
289     return best_solution, best_fitness
290
291 # --- 为问题2定义新的参数边界 ---
292 BOUNDS = {
293     'direction_deg1': [0, 360], 'uav_speed1': [70, 140], 't_release1': [0,
294     50], 't_detonate1': [0, 10]
295 }
```

```
296
297 # --- 2. 为问题4创建高质量的整体种子生成器 ---
298
299
300
301 def decode_chromosome(chromosome):
302     """
303     将12个元素的染色体解码为仿真函数所需的uav_flight_plans字典。
304     """
305     dir1, speed1, t1, d1 = chromosome
306
307     plans = {
308         "FY1": {"direction_deg": dir1, "uav_speed": speed1,
309                 "grenade_strategy": [{"t_release": t1, "t_detonate_after": d1}]},
310         # 其他无人机不参与
311         "FY2": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
312         "FY3": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
313         "FY4": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
314         "FY5": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
315     }
316     return plans
317
318
319 def fitness_wrapper(chromosome):
320     plans = decode_chromosome(chromosome)
321     results = calculate_time(plans, coarse_time_step=0.2, precision=0.01)
322     return results.get("M1", 0.0)
323
324 def fitness_wrapper_fine(chromosome):
325     plans = decode_chromosome(chromosome)
326     results = calculate_time(plans, coarse_time_step=0.05, precision=0.001)
327     return results.get("M1", 0.0)
328
329 def two_stage_optimization(
330     num_screening_seeds=50,
331     screening_iterations=30,
332     num_champions=5,
333     deep_dive_iterations=150
334 ):
335     """
336     使用两阶段优化策略解决问题3。
337
338     Args:
339         num_screening_seeds(int): 第一阶段（勘探）要生成的种子总数。
340         screening_iterations(int): 第一阶段对每个种子进行的简短优化迭代次数。
```

```
341     num_champions(int): 从第一阶段选出进入深度优化的“冠军”种子数量。
342     deep_dive_iterations(int):
343         第二阶段（开采）对冠军种子进行的深度优化迭代次数。
344         """
345
346         # --- STAGE 1: 广泛筛选（勘探） ---
347         print("=" * 20 + "STAGE 1: 广泛筛选" + "=" * 20)
348         print(f"将生成 {num_screening_seeds} 个种子，并对每个种子进行 {screening_iterations} 次迭代的初步优化...")
349
350         # 生成大量高质量种子
351         initial_seeds = generate_holistic_seeds(num_screening_seeds)
352         if not initial_seeds:
353             print("错误：未能生成任何有效的初始种子，无法继续。")
354             return
355
356         screening_results = []
357         with tqdm(total=len(initial_seeds), desc="第一阶段筛选") as pbar:
358             for seed in initial_seeds:
359                 # 进行简短、快速的优化
360                 solution, fitness = optimize_with_cmaes(seed, fitness_wrapper,
361                                             max_iterations=screening_iterations)
362                 screening_results.append({'solution': solution, 'fitness': fitness})
363                 pbar.update(1)
364                 pbar.set_postfix(current_best=f"{fitness:.3f}")
365
366         # 根据初步优化的结果进行排序
367         screening_results.sort(key=lambda x: x['fitness'], reverse=True)
368
369         print("\n第一阶段筛选完成！初步优化的最佳结果：")
370         for i in range(min(num_champions, len(screening_results))):
371             print(f"Top {i+1}: 遮蔽时间：{screening_results[i]['fitness']:.4f} 秒")
372
373         # --- STAGE 2: 深度优化（开采） ---
374         print("\n" + "=" * 20 + "STAGE 2: 深度优化" + "=" * 20)
375         print(f"将对 Top {num_champions} 的种子进行 {deep_dive_iterations} 次迭代的深度优化...")
376
377         champion_seeds = [res['solution'] for res in
378                           screening_results[:num_champions]]
379
380         best_solution_overall, best_fitness_overall = None, -1
381         with tqdm(total=len(champion_seeds), desc="第二阶段优化") as pbar:
382             for champion_seed in champion_seeds:
```

```
380     # 进行完整的、深度的优化
381     solution, fitness = optimize_with_cmaes(champion_seed,
382         fitness_wrapper_fine, max_iterations=deep_dive_iterations)
383     if fitness > best_fitness_overall:
384         best_fitness_overall = fitness
385         best_solution_overall = solution
386         pbar.update(1)
387         pbar.set_postfix(best_overall=f"{best_fitness_overall:.3f}")
388
389     print("\n优化完成! ")
390     if best_solution_overall is None:
391         print("错误: 优化未能找到任何有效的非零解。")
392         return
393
394     print("对最优策略进行高精度验算...")
395     best_plans = decode_chromosome(best_solution_overall)
396     final_results = calculate_time(best_plans, coarse_time_step=0.01,
397         precision=0.0001)
398     final_time = final_results.get("M1", 0.0)
399
400     print("\n" + "=" * 50)
401     print("oooooooooooo问题4: 最优投放策略(CMA-ES)oooooooo")
402     print("=" * 50)
403     print(f"找到的最大有效遮蔽时长:{final_time:.4f}秒")
404
405     # --- 格式化输出 ---
406     results_data = []
407     for uav_id in ["FY1"]:
408         params = best_plans[uav_id]
409         strat = params['grenade_strategy'][0]
410
411         # 计算投放点和起爆点
412         p_uav_initial = INITIAL_POSITIONS[uav_id]
413         angle_rad = np.radians(params['direction_deg'])
414         v_uav = np.array([params['uav_speed'] * np.cos(angle_rad),
415             params['uav_speed'] * np.sin(angle_rad), 0])
416         p_drop = p_uav_initial + v_uav * strat['t_release']
417         p_detonate = p_drop + np.array([v_uav[0] * strat['t_detonate_after'],
418                                         v_uav[1] * strat['t_detonate_after'],
419                                         -0.5 * G * strat['t_detonate_after'] ** 2])
420
421         print(f"\n策略详情: 无人机{uav_id}")
422         print(f"  飞行方向: {params['direction_deg']:.2f}度")
423         print(f"  飞行速度: {params['uav_speed']:.2f}m/s")
424         print(f"  投放时间: 受领任务后{strat['t_release']:.3f}秒")
```

```

422     print(f"引爆延迟: {strat['t_detonate_after']:.3f} 秒")
423     print(f"投放点坐标(x,y,z): ({p_drop[0]:.2f}, {p_drop[1]:.2f}, "
424           f"{p_drop[2]:.2f})")
425     print(f"起爆点坐标(x,y,z): ({p_detonate[0]:.2f}, "
426           f"{p_detonate[1]:.2f}, {p_detonate[2]:.2f})")
427
428     results_data.append([
429         uav_id, f"{{params['direction_deg']:.4f}}",
430         f"{{params['uav_speed']:.4f}}",
431         f"{{p_drop[0]:.4f}}", f"{{p_drop[1]:.4f}}", f"{{p_drop[2]:.4f}}",
432         f"{{p_detonate[0]:.4f}}", f"{{p_detonate[1]:.4f}}",
433         f"{{p_detonate[2]:.4f}}"
434     ])
435
436
437 if __name__ == '__main__':
438     two_stage_optimization(
439         num_screening_seeds=100, # 勘探不同的起点
440         screening_iterations=30, # 每个起点快速迭代
441         num_champions=3, # 选出最好的
442         deep_dive_iterations=150 # 对最好的进行深度优化
443     )

```

C 附录 C: 问题三 C++ 实现

Listing 4: 问题三 C++ 实现代码

```

1 [language=C++, basicstyle=\ttfamily\footnotesize, breaklines=true,
2 numbers=left, numberstyle=\tiny, frame=single, caption={问题三 C++ 实现代码}]
3
4
5 #include <iostream>
6 #include <fstream>
7 #include <vector>
8 #include <string>
9 #include <map>
10 #include <cmath>
11 #include <random>
12 #include <algorithm>
13 #include <numeric>

```

```
14 #include <future>
15 #include <iomanip>
16 #include <optional>
17 #include <thread> // <<< FIX: Added for std::this_thread
18
19 // --- 依赖: EIGEN ---
20 #include <Eigen/Dense>
21
22 // <<< FIX: Removed unnecessary #pragma once from .cpp file >>>
23
24 // =====
25 // SECTION 1: 工具 & 基础数据结构
26 // =====
27
28 struct Vector3D {
29     double x = 0.0, y = 0.0, z = 0.0;
30
31     Vector3D operator+(const Vector3D &other) const { return {x + other.x, y +
32         other.y, z + other.z}; }
33     Vector3D operator-(const Vector3D &other) const { return {x - other.x, y -
34         other.y, z - other.z}; }
35     Vector3D operator*(double scalar) const { return {x * scalar, y * scalar,
36         z * scalar}; }
37     double dot(const Vector3D &other) const { return x * other.x + y * other.y
38         + z * other.z; }
39     double norm_sq() const { return x * x + y * y + z * z; }
40     double norm() const { return std::sqrt(norm_sq()); }
41 };
42
43 inline Vector3D operator*(double scalar, const Vector3D &vec) { return vec *
44     scalar; }
45
46 struct GrenadeStrategy { double t_release; double t_detonate_after; };
47 struct UAVPlan { double direction_deg; double uav_speed;
48     std::vector<GrenadeStrategy> grenade_strategy; };
49 using FlightPlans = std::map<std::string, UAVPlan>;
50 struct GrenadeLifecycle { Vector3D p_detonate; double t_detonate_abs; };
51
52 // =====
53 // SECTION 2: 常量 & 全局仿真数据
54 // =====
55
56 namespace Constants {
57     const double G = 9.8;
58     const double PI = 3.14159265358979323846;
```

```
53 const double MISSILE_SPEED_MPS = 300;
54 const double SMOKE_CLOUD_RADIUS_M = 10.0;
55 const double SMOKE_EFFECTIVE_DURATION_S = 20.0;
56 const double SMOKE_SINK_SPEED_MPS = 3.0;
57
58 const Vector3D TARGET_BASE_CENTER = {0, 200, 0};
59 const double TARGET_RADIUS = 7.0;
60 const double TARGET_HEIGHT = 10.0;
61
62 const std::map<std::string, Vector3D> INITIAL_POSITIONS = {
63     {"M1", {20000, 0, 2000}}, {"M2", {19000, 600, 2100}}, {"M3",
64         {18000, -600, 1900}},
65     {"FY1", {17800, 0, 1800}}, {"FY2", {12000, 1400, 1400}}, {"FY3",
66         {6000, -3000, 700}},
67     {"FY4", {11000, 2000, 1800}}, {"FY5", {13000, -2000, 1300}},
68 };
69
70 const Vector3D FAKE_TARGET_POS = {0, 0, 0};
71
72 std::map<std::string, Vector3D> initialize_directions() {
73     std::map<std::string, Vector3D> dirs;
74     for (const auto &pair: INITIAL_POSITIONS) {
75         if (pair.first.rfind("M", 0) == 0) {
76             Vector3D dir = FAKE_TARGET_POS - pair.second;
77             dirs[pair.first] = dir * (1.0 / dir.norm());
78         }
79     }
80     return dirs;
81 }
82 const std::map<std::string, Vector3D> MISSILE_DIRECTIONS =
83     initialize_directions();
84
85 std::vector<Vector3D> generate_target_points() {
86     std::vector<Vector3D> points;
87     int num_rim_points = 12, num_height_levels = 3;
88     for (int i = 0; i < num_height_levels; ++i) {
89         double h = (num_height_levels > 1) ? (TARGET_HEIGHT * i /
90             (num_height_levels - 1)) : 0;
91         for (int j = 0; j < num_rim_points; ++j) {
92             double angle = 2 * PI * j / num_rim_points;
93             points.push_back({
94                 TARGET_BASE_CENTER.x + TARGET_RADIUS *
95                     std::cos(angle),
96                 TARGET_BASE_CENTER.y + TARGET_RADIUS *
97                     std::sin(angle),
```

```
92             TARGET_BASE_CENTER.z + h
93         });
94     }
95 }
96     return points;
97 }
98 const std::vector<Vector3D> REAL_TARGET_POINTS = generate_target_points();
99 }
100
101 // =====
102 // SECTION 3: 核心物理仿真
103 // =====
104
105 bool is_target_fully_obsured_at_time(double t, const std::string &missile_id,
106                                     const std::vector<GrenadeLifecycle> &all_grenades);
107
108 double calculate_time_for_m1(const FlightPlans& uav_flight_plans, double
109                               coarse_time_step = 0.1, double precision = 0.001, double max_sim_time =
110                               100.0) {
111     std::vector<GrenadeLifecycle> grenade_lifecycles;
112     for (const auto& [uav_id, uav_data] : uav_flight_plans) {
113         if (uav_data.grenade_strategy.empty()) continue;
114         double angle_rad = uav_data.direction_deg * Constants::PI / 180.0;
115         Vector3D v_uav = {uav_data.uav_speed * std::cos(angle_rad),
116                           uav_data.uav_speed * std::sin(angle_rad), 0};
117         for (const auto& grenade : uav_data.grenade_strategy) {
118             Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) + v_uav *
119                               grenade.t_release;
120             Vector3D p_detonate = p_drop + Vector3D{v_uav.x *
121                                         grenade.t_detonate_after, v_uav.y * grenade.t_detonate_after,
122                                         -0.5 * Constants::G * grenade.t_detonate_after *
123                                         grenade.t_detonate_after};
124             grenade_lifecycles.push_back({p_detonate, grenade.t_release +
125                                           grenade.t_detonate_after});
126         }
127     }
128 }
129
130 const std::string missile_id = "M1";
131 double total_obsured_time = 0.0, last_time = 0.0;
132 bool last_state = is_target_fully_obsured_at_time(0.0, missile_id,
133                                                 grenade_lifecycles);
134 double obscuration_start_time = last_state ? 0.0 : -1.0;
135 auto find_transition = [&](double t_start, double t_end, bool start_state)
136 {
137     double low = t_start, high = t_end;
```

```
126     while ((high - low) > precision) {
127         double mid = low + (high - low) / 2.0;
128         if (is_target_fully_obsured_at_time(mid, missile_id,
129             grenade_lifecycles) == start_state) low = mid;
130         else high = mid;
131     }
132     return high;
133 };
134 int num_coarse_steps = static_cast<int>(max_sim_time / coarse_time_step);
135 for (int i = 1; i <= num_coarse_steps; ++i) {
136     double current_time = i * coarse_time_step;
137     bool current_state = is_target_fully_obsured_at_time(current_time,
138         missile_id, grenade_lifecycles);
139     if (current_state != last_state) {
140         double transition_t = find_transition(last_time, current_time,
141             last_state);
142         if (current_state) { obscuration_start_time = transition_t; }
143         else if (obscuration_start_time >= 0) { total_obsured_time +=
144             (transition_t - obscuration_start_time); obscuration_start_time
145             = -1.0; }
146     }
147     last_time = current_time; last_state = current_state;
148 }
149 if (obscuration_start_time >= 0) total_obsured_time += (max_sim_time -
150     obscuration_start_time);
151
152 return total_obsured_time;
153 }
154
155 bool _is_single_line_blocked(const Vector3D &missile_pos, const Vector3D
156     &smoke_center, const Vector3D &target_point) {
157     Vector3D vec_AB = target_point - missile_pos;
158     Vector3D vec_AC = smoke_center - missile_pos;
159     double vec_AB_norm_sq = vec_AB.norm_sq();
160     if (vec_AB_norm_sq < 1e-9) return (smoke_center - missile_pos).norm_sq()
161         <= Constants::SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
162     double t_proj = vec_AC.dot(vec_AB) / vec_AB_norm_sq;
163     Vector3D closest_point = missile_pos + std::clamp(t_proj, 0.0, 1.0) *
164         vec_AB;
165     return (smoke_center - closest_point).norm_sq() <=
166         Constants::SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
167 }
168
169 bool is_target_fully_obsured_at_time(double t, const std::string &missile_id,
170     const std::vector<GrenadeLifecycle> &all_grenades) {
```

```

160     std::vector<Vector3D> active_smoke_centers;
161     for (const auto &grenade: all_grenades) {
162         if (t >= grenade.t_detonate_abs && t < grenade.t_detonate_abs +
163             Constants::SMOKE_EFFECTIVE_DURATION_S) {
164             active_smoke_centers.push_back(grenade.p_detonate - Vector3D{0, 0,
165             Constants::SMOKE_SINK_SPEED_MPS * (t -
166             grenade.t_detonate_abs)} );
167         }
168     }
169     if (active_smoke_centers.empty()) return false;
170     Vector3D missile_pos = Constants::INITIAL_POSITIONS.at(missile_id) +
171         Constants::MISSILE_DIRECTIONS.at(missile_id) *
172         (Constants::MISSILE_SPEED_MPS * t);
173     for (const auto &target_point: Constants::REAL_TARGET_POINTS) {
174         bool point_blocked = false;
175         for (const auto &center: active_smoke_centers) {
176             if (_is_single_line_blocked(missile_pos, center, target_point)) {
177                 point_blocked = true; break; }
178         }
179         if (!point_blocked) return false;
180     }
181     return true;
182 }
183 // -----
184 // SECTION 4: 整体协同种子生成器
185 // -----
186 namespace Seeder {
187     std::mt19937 &get_rng() {
188         static thread_local std::mt19937 rng(std::random_device{}() +
189             std::hash<std::thread::id>{}(std::this_thread::get_id()));
190         return rng;
191     }
192     double uniform(double min, double max) { return
193         std::uniform_real_distribution<double>(min, max)(get_rng()); }
194
195     // <<< NEW: Helper function to generate a SINGLE seed >>>
196     Eigen::VectorXd generate_single_holistic_seed_p3() {
197         const Vector3D p_fy1_initial = Constants::INITIAL_POSITIONS.at("FY1");
198         auto find_interception_details = [] (const Vector3D &v_uav, const
199             Vector3D &p_uav_initial, double t_min, double t_max) ->
200             std::optional<GrenadeStrategy> {
201                 for (int i = 0; i < 20; ++i) {
202                     double t_release = uniform(t_min, t_max);
203                     Vector3D p_drop = p_uav_initial + v_uav * t_release;
204                 }
205             }
206         return find_interception_details(v_uav, p_fy1_initial, 0, 10);
207     }
208 }
```

```

195     for (int j = 0; j < 40; ++j) {
196         double t_detonate_after = uniform(0.1, 8.0);
197         double t_intercept = t_release + t_detonate_after;
198         Vector3D p_smoke = p_drop + Vector3D{v_uav.x *
199             t_detonate_after, v_uav.y * t_detonate_after, -0.5 *
200             Constants::G * t_detonate_after * t_detonate_after};
201         if (is_target_fully_obsured_at_time(t_intercept, "M1",
202             {{p_smoke, t_intercept}})) {
203             return GrenadeStrategy{t_release, t_detonate_after};
204         }
205     }
206     return std::nullopt;
207 };
208
209 while (true) {
210     double dir = uniform(179, 180), speed = uniform(70, 140);
211     Vector3D v_uav = {speed * std::cos(dir * Constants::PI / 180.0),
212         speed * std::sin(dir * Constants::PI / 180.0), 0};
213
214     auto g1_opt = find_interception_details(v_uav, p_fy1_initial, 0.0,
215         5.0);
216     if (!g1_opt) continue;
217
218     auto g2_opt = find_interception_details(v_uav, p_fy1_initial,
219         g1_opt->t_release + 1.0, g1_opt->t_release + 5.0);
220     if (!g2_opt) continue;
221
222     auto g3_opt = find_interception_details(v_uav, p_fy1_initial,
223         g2_opt->t_release + 1.0, g2_opt->t_release + 5.0);
224     if (!g3_opt) continue;
225
226     double g2 = g2_opt->t_release - g1_opt->t_release - 1.0;
227     double g3 = g3_opt->t_release - g2_opt->t_release - 1.0;
228     if (g2 < 0 || g3 < 0) continue;
229
230     return (Eigen::VectorXd(8) << dir, speed, g1_opt->t_release,
231         g1_opt->t_detonate_after, g2, g2_opt->t_detonate_after, g3,
232         g3_opt->t_detonate_after).finished();
233 }
234
235 // <<< MODIFICATION: Main seeder function now uses a scalable thread pool
236 // >>>
237 std::vector<Eigen::VectorXd> generate_holistic_seeds_p3(int num_seeds) {

```

```
230     std::cout << "Generating" << num_seeds << " holistic seeds for Problem"
231     3 (Parallel)..." << std::endl;
232
232     unsigned int num_workers = std::thread::hardware_concurrency();
233     if (num_workers == 0) num_workers = 4;
234     std::cout << "Using" << num_workers << " worker threads for"
235     generation." << std::endl;
236
236     std::vector<std::future<Eigen::VectorXd>> futures;
237     futures.reserve(num_workers);
238     std::vector<Eigen::VectorXd> seeds;
239     seeds.reserve(num_seeds);
240
241     int tasks_launched = 0;
242     int tasks_completed = 0;
243
244     while (tasks_completed < num_seeds) {
245         while (futures.size() < num_workers && tasks_launched < num_seeds) {
246             futures.push_back(std::async(std::launch::async,
247                 generate_single_holistic_seed_p3));
248             tasks_launched++;
249         }
250
250         for (auto it = futures.begin(); it != futures.end(); ) {
251             if (it->wait_for(std::chrono::seconds(0)) ==
252                 std::future_status::ready) {
253                 seeds.push_back(it->get());
254                 tasks_completed++;
255                 it = futures.erase(it);
256             } else {
257                 ++it;
258             }
259         }
260
260         int bar_width = 50;
261         float progress = (float)tasks_completed / num_seeds;
262         int pos = bar_width * progress;
263         std::cout << "\r Generating seeds: ";
264         for (int k = 0; k < bar_width; ++k) {
265             if (k < pos) std::cout << "=";
266             else if (k == pos) std::cout << ">";
267             else std::cout << " ";
268         }
269         std::cout << "]" << tasks_completed << "/" << num_seeds << "(" <<
270             std::fixed << std::setprecision(1) << progress * 100.0 << "%)"
```

```

270             << std::flush;
271
272         std::this_thread::sleep_for(std::chrono::milliseconds(100));
273     }
274
275     std::cout << "\nSeed\ngeneration\complete." << std::endl;
276     return seeds;
277 }
278
279 // =====
280 // SECTION 5: 自包含的 CMA-ES 优化器
281 // =====
282
283 class SimpleCMAES {
284 public:
285     struct Solution { Eigen::VectorXd x; double fitness; };
286
287     SimpleCMAES(int dim, int population_size = 0) : N(dim) {
288         lambda = population_size > 0 ? population_size : 4 + floor(3 * log(N));
289         mu = lambda / 2 > 0 ? lambda / 2 : 1;
290         weights = Eigen::VectorXd::LinSpaced(mu, log(mu + 0.5),
291                                             log(0.5)).array().exp();
292         weights /= weights.sum();
293         mu_eff = 1 / weights.squaredNorm();
294
295         cc = (4 + mu_eff / N) / (N + 4 + 2 * mu_eff / N);
296         cs = (mu_eff + 2) / (N + mu_eff + 5);
297         c1 = 2 / (pow(N + 1.3, 2) + mu_eff);
298         cmu = std::min(1 - c1, 2 * (mu_eff - 2 + 1 / mu_eff) / (pow(N + 2, 2) +
299                                         mu_eff));
300         damps = 1 + 2 * std::max(0.0, sqrt((mu_eff - 1) / (N + 1)) - 1) + cs;
301         chiN = sqrt(N) * (1 - 1 / (4.0 * N) + 1 / (21.0 * N * N));
302     }
303
304     void optimize(const std::function<double(const Eigen::VectorXd &)>
305                  &fitness_func, Eigen::VectorXd &x_start, double sigma_start, int
306                  max_iter) {
307         Eigen::VectorXd x_mean = x_start;
308         double sigma = sigma_start;
309         Eigen::MatrixXd C = Eigen::MatrixXd::Identity(N, N);
310         Eigen::VectorXd pc = Eigen::VectorXd::Zero(N), ps =
311             Eigen::VectorXd::Zero(N);
312
313         // <<< FIX: Track the best solution found across all generations >>>

```

```

309     Solution best_solution_so_far = {x_start, fitness_func(x_start)};
310
311     for (int gen = 0; gen < max_iter; ++gen) {
312         Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(C);
313         Eigen::MatrixXd B = es.eigenvectors();
314         Eigen::VectorXd D_vec = es.eigenvalues();
315         for(int i=0; i<N; ++i) D_vec(i) = std::sqrt(std::max(1e-20,
316             D_vec(i)));
317
318         std::vector<Solution> population(lambda);
319         for (int i = 0; i < lambda; ++i) {
320             Eigen::VectorXd z = Eigen::VectorXd::NullaryExpr(N, [&] () {
321                 return
322                 std::normal_distribution<double>{}(Seeder::get_rng()); });
323             population[i].x = x_mean + sigma * B * D_vec.asDiagonal() * z;
324             population[i].fitness = fitness_func(population[i].x);
325         }
326
327         std::sort(population.begin(), population.end(), [] (const auto &a,
328             const auto &b) { return a.fitness > b.fitness; });
329
330         if (population[0].fitness > best_solution_so_far.fitness) {
331             best_solution_so_far = population[0];
332         }
333
334         Eigen::VectorXd x_old = x_mean;
335         x_mean.setZero();
336         for (int i = 0; i < mu; ++i) x_mean += weights(i) * population[i].x;
337
338         ps = (1 - cs) * ps + sqrt(cs * (2 - cs) * mu_eff) * (B *
339             D_vec.asDiagonal().inverse() * B.transpose() * (x_mean - x_old)
340             / sigma);
341         sigma *= exp((cs / damps) * (ps.norm() / chiN - 1));
342
343         bool hsig = ps.norm() / sqrt(1 - pow(1 - cs, 2 * (gen + 1))) < chiN
344             * (1.4 + 2.0 / (N + 1));
345         pc = (1 - cc) * pc;
346         if (hsig) pc += sqrt(cc * (2 - cc) * mu_eff) * (x_mean - x_old) /
347             sigma;
348
349         Eigen::MatrixXd rank_mu_update = Eigen::MatrixXd::Zero(N, N);
350         for (int i = 0; i < mu; ++i) {
351             Eigen::VectorXd diff = (population[i].x - x_old) / sigma;
352             rank_mu_update += weights(i) * diff * diff.transpose();
353         }
354

```

```

346         C = (1 - c1 - cmu) * C + c1 * (pc * pc.transpose() + (1 - hsig) *
347             cc * (2 - cc) * C) + cmu * rank_mu_update;
348     }
349     x_start = best_solution_so_far.x;
350 }
351
352 private:
353     int N, lambda, mu;
354     double mu_eff, cc, cs, c1, cmu, damps, chiN;
355     Eigen::VectorXd weights;
356 };
357 // =====
358 // SECTION 6: 主要的两阶段优化逻辑
359 // =====
360
361 struct OptimizationResult { Eigen::VectorXd solution; double fitness; };

362 FlightPlans decode_chromosome_p3(const Eigen::VectorXd &chromosome) {
363     double dir = chromosome(0), speed = chromosome(1), t1 = chromosome(2), d1
364     = chromosome(3),
365     g2 = chromosome(4), d2 = chromosome(5), g3 = chromosome(6), d3 =
366     chromosome(7);
367     return {
368         {"FY1", {dir, speed, {{t1, d1}, {t1 + 1.0 + g2, d2}, {t1 + 1.0 + g2
369             + 1.0 + g3, d3}}}},
370         {"FY2", {0, 70, {}}}, {"FY3", {0, 70, {}}},
371         {"FY4", {0, 70, {}}}, {"FY5", {0, 70, {}}}
372     };
373 }
374
375 void solve_problem3_two_stage(int num_screening_seeds, int
376     screening_iterations, int num_champions, int deep_dive_iterations) {
377     // --- STAGE 1: 广泛筛选 (并行) ---
378     std::cout << "\n===== STAGE 1: Broad Screening (Parallel) ====="
379     << std::endl;
380     auto initial_seeds =
381         Seeder::generate_holistic_seeds_p3(num_screening_seeds);
382
383     std::vector<std::future<OptimizationResult>> screening_futures;
384     for (const auto &seed: initial_seeds) {
385         screening_futures.push_back(std::async(std::launch::async, [=] {
386             SimpleCMAES cma(8);
387             Eigen::VectorXd current_sol = seed;
388             auto fitness_wrapper = [] (const Eigen::VectorXd &x) {

```

```

384         double speed = x(1), t1 = x(2), d1 = x(3), g2 = x(4), d2 = x(5),
385         g3 = x(6), d3 = x(7);
386         // <<< REFINEMENT: Corrected constraint check >>>
387         if (speed < 70.0 || speed > 140.0 || t1 < 0.0 || d1 < 0.0 || g2
388             < 0.0 || d2 < 0.0 || g3 < 0.0 || d3 < 0.0) {
389             return 0.0;
390         }
391         return calculate_time_for_m1(decode_chromosome_p3(x), 0.2, 0.01);
392     };
393     cma.optimize(fitness_wrapper, current_sol, 0.1,
394                 screening_iterations);
395     return OptimizationResult{current_sol,
396                               fitness_wrapper(current_sol)};
397 });
398
399 std::vector<OptimizationResult> screening_results;
400 screening_results.reserve(screening_futures.size());
401 for(int i = 0; i < num_screening_seeds; ++i) {
402     screening_results.push_back(screening_futures[i].get());
403     int bar_width = 50;
404     float progress = (float)(i + 1) / num_screening_seeds;
405     int pos = bar_width * progress;
406     std::cout << "\rScreening Progress: [";
407     for (int k = 0; k < bar_width; ++k) {
408         if (k < pos) std::cout << "=";
409         else if (k == pos) std::cout << ">";
410         else std::cout << " ";
411     }
412     std::cout << "] " << i + 1 << "/" << num_screening_seeds << "(" <<
413         std::fixed << std::setprecision(1) << progress * 100.0 << "%" <<
414         std::flush;
415 }
416 std::cout << std::endl;
417
418 std::sort(screening_results.begin(), screening_results.end(), [] (const
419             auto &a, const auto &b) { return a.fitness > b.fitness; });
420
421 std::cout << "\nStage 1 Screening Complete! Top preliminary results:" <<
422             std::endl;
423 for (int i = 0; i < std::min(num_champions, (int)
424             screening_results.size()); ++i) {
425     std::cout << "Top " << i + 1 << ": Obscuration Time: " << std::fixed
426         << std::setprecision(4) << screening_results[i].fitness << " s" <<
427         std::endl;

```

```

418     }
419
420     // --- STAGE 2: 深度优化 ---
421     std::cout << "\n=====\u2022STAGE\u20222:\u2022Deep\u2022Dive\u2022Optimization\u2022
422     =====" << std::endl;
423     OptimizationResult best_overall = {{}, -1.0};
424
425     for (int i = 0; i < std::min(num_champions, (int)
426         screening_results.size()); ++i) {
427         std::cout << "\u2022Deep\u2022diving\u2022on\u2022champion\u2022#" << i + 1 << "..." <<
428             std::endl;
429         SimpleCMAES cma_deep(8);
430         Eigen::VectorXd champion_sol = screening_results[i].solution;
431         auto fitness_wrapper_fine = [] (const Eigen::VectorXd &x) {
432             double speed = x(1), t1 = x(2), d1 = x(3), g2 = x(4), d2 = x(5), g3
433                 = x(6), d3 = x(7);
434             // <<< REFINEMENT: Corrected constraint check >>>
435             if (speed < 70.0 || speed > 140.0 || t1 < 0.0 || d1 < 0.0 || g2 <
436                 0.0 || d2 < 0.0 || g3 < 0.0 || d3 < 0.0) {
437                 return 0.0;
438             }
439             return calculate_time_for_m1(decode_chromosome_p3(x), 0.05, 0.001);
440         };
441         cma_deep.optimize(fitness_wrapper_fine, champion_sol, 0.05,
442             deep_dive_iterations);
443         double final_fitness = fitness_wrapper_fine(champion_sol);
444         if (final_fitness > best_overall.fitness) {
445             best_overall = {champion_sol, final_fitness};
446         }
447     }
448
449     // --- 最终结果 ---
450     std::cout << "\nOptimization\u2022complete!\u2022Performing\u2022final\u2022high-precision\u2022
451     validation..." << std::endl;
452     auto best_plans = decode_chromosome_p3(best_overall.solution);
453     double final_time = calculate_time_for_m1(best_plans, 0.001, 0.0001);
454
455     std::cout << "Final\u2022results\u2022calculated.\u2022Writing\u2022to\u2022output_p3.log..." <<
456         std::endl;
457     std::ofstream out("output_p3.log");
458     auto* cout_buf = std::cout.rdbuf();
459     std::cout.rdbuf(out.rdbuf());
460
461     std::cout << "\n=====\u2022FINAL\u2022OPTIMAL\u2022STRATEGY\u2022(Problem\u2022
462         3)\u2022=====" << std::endl;

```

```
454     std::cout << "Max_Effective_Obscuration_Time_for_M1:" << std::fixed <<
455         std::setprecision(4) << final_time << "s" << std::endl;
456
457     std::cout << "\nStrategy_Details:" << std::endl;
458     const auto &best_params = best_plans.at("FY1");
459     std::cout << "UAV_FY1_Flight_Direction:" << std::fixed <<
460         std::setprecision(2) << best_params.direction_deg << "deg" <<
461         std::endl;
462     std::cout << "UAV_FY1_Flight_Speed:" << best_params.uav_speed << "m/s"
463         << std::endl;
464
465     std::cout << "\nGrenade_Deployment_Sequence:" << std::endl;
466     int idx = 1;
467     for (const auto &g: best_params.grenade_strategy) {
468         std::cout << "Grenade#" << idx++ << ":" << std::endl;
469         std::cout << "Release_at" << std::fixed << std::setprecision(3)
470             << g.t_release << "s" << std::endl;
471         std::cout << "Detonate_after" << g.t_detonate_after << "s" <<
472             std::endl;
473     }
474     std::cout << "======" <<
475         std::endl;
476
477     std::cout.rdbuf(cout_buf);
478     std::cout << "Done. Results saved to output_p3.log" << std::endl;
479 }
480
481 int main() {
482     // --- 在这里调整你的参数 ---
483     int num_screening_seeds = 100000; // 勘探的种子总数
484     int screening_iterations = 100; // 每个种子的快速迭代次数
485     int num_champions = 30; // 选出的冠军数量
486     int deep_dive_iterations = 1000; // 对冠军的深度迭代次数
487
488     solve_problem3_two_stage(
489         num_screening_seeds,
490         screening_iterations,
491         num_champions,
492         deep_dive_iterations
493     );
494
495     return 0;
496 }
```

D 附录 D: 问题四 C++ 实现

Listing 5: 问题四 C++ 实现代码

```

1 # 这里放你完整的 Python 代码 (two_stage_optimization 那一份)
2
3 #include <iostream>
4 #include <fstream>
5 #include <vector>
6 #include <string>
7 #include <map>
8 #include <cmath>
9 #include <random>
10 #include <algorithm>
11 #include <numeric>
12 #include <future>
13 #include <iomanip>
14 #include <optional>
15 #include <chrono>
16 #include <thread>
17
18 // --- 依赖: EIGEN ---
19 #include <Eigen/Dense>
20
21 // =====
22 // SECTION 1: 工具 & 基础数据结构
23 // =====
24
25 struct Vector3D {
26     double x = 0.0, y = 0.0, z = 0.0;
27
28     Vector3D operator+(const Vector3D& other) const { return {x + other.x,
29                         y + other.y, z + other.z}; }
30     Vector3D operator-(const Vector3D& other) const { return {x - other.x,
31                         y - other.y, z - other.z}; }
32     Vector3D operator*(double scalar) const { return {x * scalar, y *
33                         scalar, z * scalar}; }
34     double dot(const Vector3D& other) const { return x * other.x + y *
35                         other.y + z * other.z; }
36     double norm_sq() const { return x * x + y * y + z * z; }
37     double norm() const { return std::sqrt(norm_sq()); }
38 };
39
40 inline Vector3D operator*(double scalar, const Vector3D& vec) { return vec
41     * scalar; }

```

```
37
38 struct GrenadeStrategy { double t_release; double t_detonate_after; };
39 struct UAVPlan { double direction_deg; double uav_speed; std::vector<
40     GrenadeStrategy> grenade_strategy; };
41 using FlightPlans = std::map<std::string, UAVPlan>;
42 struct GrenadeLifecycle { Vector3D p_detonate; double t_detonate_abs; };
43 // =====
44 // SECTION 2: 常量 & 全局仿真数据
45 // =====
46
47 namespace Constants {
48     const double G = 9.8;
49     const double PI = 3.14159265358979323846;
50     const double MISSILE_SPEED_MPS = 300;
51     const double SMOKE_CLOUD_RADIUS_M = 10.0;
52     const double SMOKE_EFFECTIVE_DURATION_S = 20.0;
53     const double SMOKE_SINK_SPEED_MPS = 3.0;
54
55     const Vector3D TARGET_BASE_CENTER = {0, 200, 0};
56     const double TARGET_RADIUS = 7.0;
57     const double TARGET_HEIGHT = 10.0;
58
59     const std::map<std::string, Vector3D> INITIAL_POSITIONS = {
60         {"M1", {20000, 0, 2000}}, {"M2", {19000, 600, 2100}}, {"M3",
61             {18000, -600, 1900}},
62         {"FY1", {17800, 0, 1800}}, {"FY2", {12000, 1400, 1400}}, {"FY3"
63             , {6000, -3000, 700}},
64         {"FY4", {11000, 2000, 1800}}, {"FY5", {13000, -2000, 1300}},
65     };
66
67     const Vector3D FAKE_TARGET_POS = {0, 0, 0};
68
69     std::map<std::string, Vector3D> initialize_directions() {
70         std::map<std::string, Vector3D> dirs;
71         for (const auto& pair : INITIAL_POSITIONS) {
72             if (pair.first.rfind("M", 0) == 0) {
73                 Vector3D dir = FAKE_TARGET_POS - pair.second;
74                 dirs[pair.first] = dir * (1.0 / dir.norm());
75             }
76         }
77         return dirs;
78     }
79     const std::map<std::string, Vector3D> MISSILE_DIRECTIONS =
80         initialize_directions();
```

```

78
79     std::vector<Vector3D> generate_target_points() {
80         std::vector<Vector3D> points;
81         int num_rim_points = 12, num_height_levels = 2;
82         for (int i = 0; i < num_height_levels; ++i) {
83             double h = (num_height_levels > 1) ? (TARGET_HEIGHT * i / (
84                 num_height_levels - 1)) : 0;
85             for (int j = 0; j < num_rim_points; ++j) {
86                 double angle = 2 * PI * j / num_rim_points;
87                 points.push_back({
88                     TARGET_BASE_CENTER.x +
89                     TARGET_RADIUS * std::cos(angle)
90                     ,
91                     TARGET_BASE_CENTER.y +
92                     TARGET_RADIUS * std::sin(angle)
93                     ,
94                     TARGET_BASE_CENTER.z + h
95                 });
96             }
97         }
98         return points;
99     }
100 const std::vector<Vector3D> REAL_TARGET_POINTS = generate_target_points()
101 ();
102
103 // =====
104 // SECTION 3: 核心物理仿真
105 // =====
106
107 bool _is_single_line_blocked(const Vector3D& missile_pos, const Vector3D&
108     smoke_center, const Vector3D& target_point) {
109     Vector3D vec_AB = target_point - missile_pos;
110     Vector3D vec_AC = smoke_center - missile_pos;
111     double vec_AB_norm_sq = vec_AB.norm_sq();
112     if (vec_AB_norm_sq < 1e-9) return (smoke_center - missile_pos).norm_sq()
113         () <= Constants::SMOKE_CLOUD_RADIUS_M * Constants::
114             SMOKE_CLOUD_RADIUS_M;
115     double t_proj = vec_AC.dot(vec_AB) / vec_AB_norm_sq;
116     Vector3D closest_point = missile_pos + std::clamp(t_proj, 0.0, 1.0) *
117         vec_AB;
118     return (smoke_center - closest_point).norm_sq() <= Constants::
119         SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
120 }
121

```

```

112 bool is_target_fully_obsured_at_time(double t, const std::string&
113   missile_id, const std::vector<GrenadeLifecycle>& all_grenades) {
114   std::vector<Vector3D> active_smoke_centers;
115   for (const auto& grenade : all_grenades) {
116     if (t >= grenade.t_detonate_abs && t < grenade.t_detonate_abs +
117       Constants::SMOKE_EFFECTIVE_DURATION_S) {
118       active_smoke_centers.push_back(grenade.p_detonate - Vector3D{0,
119         0, Constants::SMOKE_SINK_SPEED_MPS * (t - grenade.
120           t_detonate_abs)}));
121     }
122   }
123   if (active_smoke_centers.empty()) return false;
124   Vector3D missile_pos = Constants::INITIAL_POSITIONS.at(missile_id) +
125     Constants::MISSILE_DIRECTIONS.at(missile_id) * (Constants::
126       MISSILE_SPEED_MPS * t);
127   for (const auto& target_point : Constants::REAL_TARGET_POINTS) {
128     bool point_blocked = false;
129     for (const auto& center : active_smoke_centers) {
130       if (_is_single_line_blocked(missile_pos, center, target_point))
131         { point_blocked = true; break; }
132     }
133     if (!point_blocked) return false;
134   }
135   return true;
136 }

137 double calculate_time_for_m1(const FlightPlans& uav_flight_plans, double
138   coarse_time_step = 0.1, double precision = 0.001, double max_sim_time =
139   100.0) {
140   std::vector<GrenadeLifecycle> grenade_lifecycles;
141   for (const auto& [uav_id, uav_data] : uav_flight_plans) {
142     if (uav_data.grenade_strategy.empty()) continue;
143     double angle_rad = uav_data.direction_deg * Constants::PI / 180.0;
144     Vector3D v_uav = {uav_data.uav_speed * std::cos(angle_rad),
145       uav_data.uav_speed * std::sin(angle_rad), 0};
146     for (const auto& grenade : uav_data.grenade_strategy) {
147       Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) +
148         v_uav * grenade.t_release;
149       Vector3D p_detonate = p_drop + Vector3D{v_uav.x * grenade.
150         t_detonate_after, v_uav.y * grenade.t_detonate_after, -0.5 *
151         Constants::G * grenade.t_detonate_after * grenade.
152         t_detonate_after};
153       grenade_lifecycles.push_back({p_detonate, grenade.t_release +
154         grenade.t_detonate_after});
155     }
156   }

```

```
142     }
143
144     const std::string missile_id = "M1";
145     double total_obsured_time = 0.0, last_time = 0.0;
146     bool last_state = is_target_fully_obsured_at_time(0.0, missile_id,
147             grenade_lifecycles);
148     double obscuration_start_time = last_state ? 0.0 : -1.0;
149     auto find_transition = [&](double t_start, double t_end, bool
150         start_state) {
151         double low = t_start, high = t_end;
152         while ((high - low) > precision) {
153             double mid = low + (high - low) / 2.0;
154             if (is_target_fully_obsured_at_time(mid, missile_id,
155                     grenade_lifecycles) == start_state) low = mid;
156             else high = mid;
157         }
158         return high;
159     };
160     int num_coarse_steps = static_cast<int>(max_sim_time / coarse_time_step
161         );
162     for (int i = 1; i <= num_coarse_steps; ++i) {
163         double current_time = i * coarse_time_step;
164         bool current_state = is_target_fully_obsured_at_time(current_time,
165             missile_id, grenade_lifecycles);
166         if (current_state != last_state) {
167             double transition_t = find_transition(last_time, current_time,
168                 last_state);
169             if (current_state) { obscuration_start_time = transition_t; }
170             else if (obscuration_start_time >= 0) { total_obsured_time +=
171                 (transition_t - obscuration_start_time);
172                 obscuration_start_time = -1.0; }
173         }
174         last_time = current_time; last_state = current_state;
175     }
176     if (obscuration_start_time >= 0) total_obsured_time += (max_sim_time -
177         obscuration_start_time);
178
179     return total_obsured_time;
180 }
181
182 // =====
183 // SECTION 4: 启发式种子生成器 (问题四专用)
184 // =====
185
186
187 namespace Seeder {
```



```
212         current_seed.segment(i * 4, 4) << dir, speed, g->
213             t_release, g->t_detonate_after;
214         found_strategy_for_uav = true;
215         break;
216     }
217     if (!found_strategy_for_uav) {
218         success = false;
219         break;
220     }
221 }
222 if (success) {
223     return current_seed;
224 }
225 }
226 }
227
228 // <<< MODIFICATION: Scalable, thread-pool based seeder >>>
229 std::vector<Eigen::VectorXd> generate_seeds_problem4(int num_seeds) {
230     std::cout << "Generating " << num_seeds << " holistic seeds for "
231         Problem4(Parallel)... " << std::endl;
232
233     // Determine the optimal number of worker threads
234     unsigned int num_workers = std::thread::hardware_concurrency();
235     if (num_workers == 0) num_workers = 4; // Fallback for safety
236     std::cout << "Using " << num_workers << " worker threads for "
237         generation." << std::endl;
238
239     std::vector<std::future<Eigen::VectorXd>> futures;
240     futures.reserve(num_workers);
241     std::vector<Eigen::VectorXd> seeds;
242     seeds.reserve(num_seeds);
243
244     int tasks_launched = 0;
245     int tasks_completed = 0;
246
247     while (tasks_completed < num_seeds) {
248         // Launch new tasks until the worker pool is full or all tasks
249             are launched
250         while (futures.size() < num_workers && tasks_launched <
251             num_seeds) {
252             futures.push_back(std::async(std::launch::async,
253                 generate_single_seed_p4));
254             tasks_launched++;
255         }
```

```
251
252     // Check for completed tasks and collect results
253     for (auto it = futures.begin(); it != futures.end(); ) {
254         if (it->wait_for(std::chrono::seconds(0)) == std::
255             future_status::ready) {
256             seeds.push_back(it->get());
257             tasks_completed++;
258             it = futures.erase(it); // Remove completed future
259         } else {
260             ++it;
261         }
262     }
263
264     // Progress bar
265     int bar_width = 50;
266     float progress = (float)tasks_completed / num_seeds;
267     int pos = bar_width * progress;
268     std::cout << "\rGenerating seeds: [";
269     for (int k = 0; k < bar_width; ++k) {
270         if (k < pos) std::cout << "=";
271         else if (k == pos) std::cout << ">";
272         else std::cout << " ";
273     }
274     std::cout << "] " << tasks_completed << "/" << num_seeds << "("
275         " " << std::fixed << std::setprecision(1) << progress * 100.0
276         << "%)" << std::flush;
277
278     std::this_thread::sleep_for(std::chrono::milliseconds(100));
279 }
280
281 std::cout << "\nSeed generation complete." << std::endl;
282 return seeds;
283 }
284 // =====
285 // SECTION 5: 自包含的 CMA-ES 优化器
286 // =====
287 class SimpleCMAES {
288 public:
289     struct Solution { Eigen::VectorXd x; double fitness; };
290
291     SimpleCMAES(int dim, int population_size = 0) : N(dim) {
292         lambda = population_size > 0 ? population_size : 4 + floor(3 * log(
```

```

N));
293     mu = lambda / 2 > 0 ? lambda / 2 : 1;
294     weights = Eigen::VectorXd::LinSpaced(mu, log(mu + 0.5), log(0.5)).
295         array().exp();
296     weights /= weights.sum();
297     mu_eff = 1 / weights.squaredNorm();

298     cc = (4 + mu_eff / N) / (N + 4 + 2 * mu_eff / N);
299     cs = (mu_eff + 2) / (N + mu_eff + 5);
300     c1 = 2 / (pow(N + 1.3, 2) + mu_eff);
301     cmu = std::min(1 - c1, 2 * (mu_eff - 2 + 1 / mu_eff) / (pow(N + 2,
302         2) + mu_eff));
303     damps = 1 + 2 * std::max(0.0, sqrt((mu_eff - 1) / (N + 1)) - 1) +
304         cs;
305     chiN = sqrt(N) * (1 - 1 / (4.0 * N) + 1 / (21.0 * N * N));
306 }
307
308 void optimize(const std::function<double(const Eigen::VectorXd)>&
309               fitness_func, Eigen::VectorXd& x_start, double sigma_start, int
310               max_iter) {
311     Eigen::VectorXd x_mean = x_start;
312     double sigma = sigma_start;
313     Eigen::MatrixXd C = Eigen::MatrixXd::Identity(N, N);
314     Eigen::VectorXd pc = Eigen::VectorXd::Zero(N), ps = Eigen::VectorXd
315                   ::Zero(N);

316     Solution best_solution_so_far = {x_start, fitness_func(x_start)};

317     for (int gen = 0; gen < max_iter; ++gen) {
318         Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(C);
319         Eigen::MatrixXd B = es.eigenvectors();
320         Eigen::VectorXd D_vec = es.eigenvalues();
321         for (int i=0; i<N; ++i) D_vec(i) = std::sqrt(std::max(1e-20,
322             D_vec(i)));
323
324         std::vector<Solution> population(lambda);
325         for (int i = 0; i < lambda; ++i) {
326             Eigen::VectorXd z = Eigen::VectorXd::NullaryExpr(N, [&](){
327                 return std::normal_distribution<double>{}(Seeder::
328                     get_rng()); });
329             population[i].x = x_mean + sigma * B * D_vec.asDiagonal() *
330                 z;
331             population[i].fitness = fitness_func(population[i].x);
332         }
333     }
334 }
```

```

327     std::sort(population.begin(), population.end(), [](const auto&
328                 a, const auto& b) { return a.fitness > b.fitness; });
329
330     if (population[0].fitness > best_solution_so_far.fitness) {
331         best_solution_so_far = population[0];
332     }
333
334     Eigen::VectorXd x_old = x_mean;
335     x_mean.setZero();
336     for(int i=0; i<mu; ++i) x_mean += weights(i) * population[i].x;
337
338     ps = (1-cs)*ps + sqrt(cs*(2-cs)*mu_eff) * (B * D_vec.asDiagonal
339                             ().inverse() * B.transpose() * (x_mean - x_old) / sigma);
340     sigma *= exp((cs/damps) * (ps.norm()/chiN - 1));
341
342     bool hsig = ps.norm() / sqrt(1 - pow(1-cs, 2*(gen+1))) < chiN *
343             (1.4 + 2.0/(N+1));
344     pc = (1-cc)*pc;
345     if (hsig) pc += sqrt(cc*(2-cc)*mu_eff) * (x_mean - x_old) /
346             sigma;
347
348     Eigen::MatrixXd rank_mu_update = Eigen::MatrixXd::Zero(N, N);
349     for(int i=0; i<mu; ++i) {
350         Eigen::VectorXd diff = (population[i].x - x_old) / sigma;
351         rank_mu_update += weights(i) * diff * diff.transpose();
352     }
353     C = (1 - c1 - cmu) * C + c1 * (pc * pc.transpose() + (1-hsig)*
354             cc*(2-cc)*C) + cmu * rank_mu_update;
355
356     x_start = best_solution_so_far.x;
357 }
358
359
360 private:
361     int N, lambda, mu;
362     double mu_eff, cc, cs, c1, cmu, damps, chiN;
363     Eigen::VectorXd weights;
364 };
365
366 // =====
367 // SECTION 6: 主要的两阶段优化逻辑 (问题四专用)
368 // =====
369
370 struct OptimizationResult { Eigen::VectorXd solution; double fitness; };
371
372 FlightPlans decode_chromosome_p4(const Eigen::VectorXd& chromosome) {

```

```

367     return {
368         {"FY1", {chromosome(0), chromosome(1), {{chromosome(2),
369             chromosome(3)}}}},
370         {"FY2", {chromosome(4), chromosome(5), {{chromosome(6),
371             chromosome(7)}}}},
372         {"FY3", {chromosome(8), chromosome(9), {{chromosome(10),
373             chromosome(11)}}}},
374         {"FY4", {70.0, 0.0, {}}},
375         {"FY5", {70.0, 0.0, {}}}
376     };
377 }
378
379 void two_stage_optimization_p4(int num_screening_seeds, int
380 screening_iterations, int num_champions, int deep_dive_iterations) {
381 // --- STAGE 1: 广泛筛选 (使用固定大小的任务池并行) ---
382 std::cout << "\n===== STAGE 1: Broad Screening (Parallel) ====="
383 auto initial_seeds = Seeder::generate_seeds_problem4(
384     num_screening_seeds);
385
386 // 确定最佳并发数, 通常等于硬件线程数
387 unsigned int num_workers = std::thread::hardware_concurrency();
388 if (num_workers == 0) num_workers = 8; // 安全回退值
389 std::cout << "Using a concurrent task pool of size: " << num_workers <<
390     std::endl;
391
392 std::vector<std::future<OptimizationResult>> active_futures;
393 std::vector<OptimizationResult> screening_results;
394 screening_results.reserve(num_screening_seeds);
395
396 int launched_tasks = 0;
397 int completed_tasks = 0;
398
399 // 主循环, 直到所有任务都完成
400 while (completed_tasks < num_screening_seeds) {
401     // 1. 生产者: 如果任务池有空位, 并且还有任务待启动, 则启动新任务
402     while (active_futures.size() < num_workers && launched_tasks <
403         num_screening_seeds) {
404         const auto& seed = initial_seeds[launched_tasks];
405         active_futures.push_back(std::async(std::launch::async, [=]{
406             SimpleCMAES cma(12);
407             Eigen::VectorXd current_sol = seed;
408             auto fitness_wrapper = [] (const Eigen::VectorXd& x) {
409                 for (int i = 0; i < 3; ++i) {
410                     double speed = x(i * 4 + 1), t_release = x(i * 4 +

```

```

        2), t_detonate = x(i * 4 + 3);
    if (speed < 70.0 || speed > 140.0 || t_release <
        0.0 || t_detonate < 0.0) return 0.0;
}
return calculate_time_for_m1(decode_chromosome_p4(x),
    0.1, 0.01);
};

cma.optimize(fitness_wrapper, current_sol, 0.5,
    screening_iterations);
return OptimizationResult{current_sol, fitness_wrapper(
    current_sol)};
});

launched_tasks++;
}

// 2. 消费者: 检查已启动的任务是否完成, 收集结果
for (auto it = active_futures.begin(); it != active_futures.end();
) {
    // 使用 wait_for 检查状态, 避免阻塞
    if (it->wait_for(std::chrono::seconds(0)) == std::future_status
        ::ready) {
        screening_results.push_back(it->get()); // 获取结果
        completed_tasks++;
        it = active_futures.erase(it); // 从任务池中移除已完成的任
        务
    }

    // 更新进度条
    int bar_width = 50;
    float progress = (float)completed_tasks /
        num_screening_seeds;
    int pos = bar_width * progress;
    std::cout << "\rScreening Progress: [";
    for (int k = 0; k < bar_width; ++k) {
        if (k < pos) std::cout << "=";
        else if (k == pos) std::cout << ">";
        else std::cout << " ";
    }
    std::cout << "] " << completed_tasks << "/" <<
        num_screening_seeds << "(" << std::fixed << std::
        setprecision(1) << progress * 100.0 << "%" << std::
        flush;
}

} else {
    ++it; // 任务未完成, 继续检查下一个
}

```

```
437     }
438
439     // 短暂休眠，避免主线程空转消耗CPU
440     std::this_thread::sleep_for(std::chrono::milliseconds(10));
441 }
442
443 std::cout << std::endl;
444
445 std::sort(screening_results.begin(), screening_results.end(), [](const
446     auto& a, const auto& b){ return a.fitness > b.fitness; });
447
448 std::cout << "\nStage 1 Screening Complete! Top preliminary results:"
449     << std::endl;
450     for (int i = 0; i < std::min(num_champions, (int)screening_results.size()
451         ()); ++i) {
452         std::cout << "Top " << i + 1 << ": M1 Obscuration Time:" << std
453             ::fixed << std::setprecision(4) << screening_results[i].fitness
454             << " s" << std::endl;
455     }
456
457 // --- STAGE 2: 深度优化 ---
458 std::cout << "\n===== STAGE 2: Deep Dive Optimization"
459     << std::endl;
460     OptimizationResult best_overall = {{}, -1.0};
461
462     for (int i = 0; i < std::min(num_champions, (int)screening_results.size()
463         ()); ++i) {
464         std::cout << "Deep diving on champion #" << i+1 << "..." << std::
465             endl;
466         SimpleCMAES cma_deep(12);
467         Eigen::VectorXd champion_sol = screening_results[i].solution;
468         auto fitness_wrapper_fine = [] (const Eigen::VectorXd& x){
469             for (int i = 0; i < 3; ++i) {
470                 double speed = x(i * 4 + 1), t_release = x(i * 4 + 2),
471                     t_detonate = x(i * 4 + 3);
472                 if (speed < 70.0 || speed > 140.0 || t_release < 0.0 ||
473                     t_detonate < 0.0) return 0.0;
474             }
475             return calculate_time_for_m1(decode_chromosome_p4(x), 0.01,
476                 0.0001);
477         };
478         cma_deep.optimize(fitness_wrapper_fine, champion_sol, 0.2,
479             deep_dive_iterations);
480         double final_fitness = fitness_wrapper_fine(champion_sol);
481         if (final_fitness > best_overall.fitness) {
```

```

470     best_overall = {champion_sol, final_fitness};
471 }
472 }
473
474 // --- 最终结果 ---
475 std::cout << "\nOptimization complete! Performing final high-precision
476 validation..." << std::endl;
477 auto best_plans = decode_chromosome_p4(best_overall.solution);
478 double final_time = calculate_time_for_m1(best_plans, 0.001, 0.00001);
479
480 std::cout << "Final results calculated. Writing to output_p4.log..." <<
481 std::endl;
482 std::ofstream out("output_p4.log");
483 auto* cout_buf = std::cout.rdbuf();
484 std::cout.rdbuf(out.rdbuf());
485
486 std::cout << "\n===== FINAL OPTIMAL STRATEGY ("
487 Problem_4) =====" << std::endl;
488 std::cout << "Max Effective Obscuration Time for M1: " << std::fixed <<
489 std::setprecision(4) << final_time << " us" << std::endl;
490
491 std::cout << "\nStrategy Details:" << std::endl;
492 for (const auto& uav_id : {"FY1", "FY2", "FY3"}) {
493     const auto& plan = best_plans.at(uav_id);
494     const auto& grenade = plan.grenade_strategy[0];
495
496     double angle_rad = plan.direction_deg * Constants::PI / 180.0;
497     Vector3D v_uav = {plan.uav_speed * std::cos(angle_rad), plan.
498     uav_speed * std::sin(angle_rad), 0};
499     Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) + v_uav *
500     grenade.t_release;
501     Vector3D p_detonate = p_drop + Vector3D{
502         v_uav.x * grenade.t_detonate_after,
503         v_uav.y * grenade.t_detonate_after,
504         -0.5 * Constants::G * grenade.t_detonate_after * grenade.
505         t_detonate_after
506     };
507
508     std::cout << "\n-- UAV " << uav_id << " --" << std::endl;
509     std::cout << " Flight Direction: " << std::fixed << std::
510     setprecision(2) << plan.direction_deg << " deg" << std::endl;
511     std::cout << " Flight Speed: " << plan.uav_speed << " m/s" << std
512     ::endl;
513     std::cout << " Release Time: " << std::fixed << std::setprecision
514     (3) << grenade.t_release << " us after mission start" << std::

```

```

        endl;
505     std::cout << "    Detonation_Delay: " << grenade.t_detonate_after <<
506         "s_after_release" << std::endl;
507     std::cout << "    Drop_Point(x,y,z): " << std::fixed << std::
508         setprecision(2) << p_drop.x << ", " << p_drop.y << ", " <<
509         p_drop.z << ")" << std::endl;
510     std::cout << "    Detonation_Point(x,y,z): " << p_detonate.x << ", "
511         " << p_detonate.y << ", " << p_detonate.z << ")" << std::endl;
512 }
513
514 std::cout.rdbuf(cout_buf);
515 std::cout << "Done. Results saved to output_p4.log" << std::endl;
516 }

517 int main() {
518     // --- 在这里调整你的参数 ---
519     int num_screening_seeds = 100000;      // 勘探的种子总数
520     int screening_iterations = 100;        // 每个种子的快速迭代次数
521     int num_champions = 25;                // 选出的冠军数量
522     int deep_dive_iterations = 1000;       // 对冠军的深度迭代次数
523
524     two_stage_optimization_p4(
525         num_screening_seeds,
526         screening_iterations,
527         num_champions,
528         deep_dive_iterations
529     );
530
531     return 0;
532 }
```

E 附录 E: 问题五 JaVa 实现

Listing 6: 问题五 JAVA 实现代码

```

1
2
3
4
5 import java.util.*;
6
7 public class Main {
```

```
8
9 // --- Constants ---
10 static final double G = 9.8;
11 static final double MISSILE_SPEED_MPS = 300.0;
12 static final double SMOKE_CLOUD_RADIUS_M = 10.0;
13 static final double SMOKE_EFFECTIVE_DURATION_S = 20.0;
14 static final double SMOKE_SINK_SPEED_MPS = 3.0;
15
16 // --- Target geometry ---
17 static final double[] TARGET_BASE_CENTER = new double[]{0, 200, 0};
18 static final double TARGET_RADIUS = 7.0;
19 static final double TARGET_HEIGHT = 10.0;
20
21 // --- Initial positions ---
22 static final Map<String, double[]> INITIAL_POSITIONS = new HashMap<>();
23 static {
24     INITIAL_POSITIONS.put("M1", new double[]{20000, 0, 2000});
25     INITIAL_POSITIONS.put("M2", new double[]{19000, 600, 2100});
26     INITIAL_POSITIONS.put("M3", new double[]{18000, -600, 1900});
27     INITIAL_POSITIONS.put("FY1", new double[]{17800, 0, 1800});
28     INITIAL_POSITIONS.put("FY2", new double[]{12000, 1400, 1400});
29     INITIAL_POSITIONS.put("FY3", new double[]{6000, -3000, 700});
30     INITIAL_POSITIONS.put("FY4", new double[]{11000, 2000, 1800});
31     INITIAL_POSITIONS.put("FY5", new double[]{13000, -2000, 1300});
32 }
33 static final double[] FAKE_TARGET_POS = new double[]{0, 0, 0};
34
35 // --- Missile directions (toward fake target) ---
36 static final Map<String, double[]> MISSILE_DIRECTIONS = new HashMap<>()
37     ;
38 static {
39     for (Map.Entry<String, double[]> e : INITIAL_POSITIONS.entrySet())
40     {
41         String id = e.getKey();
42         if (id.startsWith("M")) {
43             double[] v = sub(FAKE_TARGET_POS, e.getValue());
44             MISSILE_DIRECTIONS.put(id, normalize(v));
45         }
46     }
47 }
48
49 // --- Id lists ---
50 static final List<String> MISSILE_IDS = Arrays.asList("M1", "M2", "M3")
51     ;
52 static final List<String> UAV_IDS = Arrays.asList("FY1", "FY2", "FY3", "
```

```
50          FY4", "FY5");
51
52     // --- Per-UAV angle ranges you provided (deg) ---
53     // 支持起点<=终点的普通区间，也支持“环向”区间（起点>终点，表示跨越0°
54     // ）。
55     static final Map<String, double[]> ANGLE_RANGE_DEG = new HashMap<>();
56
57     static {
58         ANGLE_RANGE_DEG.put("FY1", new double[]{179.0, 180.0});
59         ANGLE_RANGE_DEG.put("FY2", new double[]{186.7, 353.5});
60         ANGLE_RANGE_DEG.put("FY3", new double[]{11.3, 153.4});
61         ANGLE_RANGE_DEG.put("FY4", new double[]{190.3, 350.1});
62         ANGLE_RANGE_DEG.put("FY5", new double[]{15.6, 171.3});
63     }
64
65
66     // --- Target points (rim points for multiple heights) ---
67     static final List<double[]> REAL_TARGET_POINTS = generateTargetPoints(
68         TARGET_BASE_CENTER, TARGET_RADIUS, TARGET_HEIGHT, 12, 2
69     );
70
71
72     // ---- Data types ----
73     static class GrenadeStrategy {
74         double tRelease;
75         double tDetonateAfter;
76
77         GrenadeStrategy(double tRelease, double tDetonateAfter) {
78             this.tRelease = tRelease; this.tDetonateAfter = tDetonateAfter;
79         }
80     }
81
82     static class UavPlan {
83         double directionDeg;
84         double uavSpeed;
85
86         List<GrenadeStrategy> grenadeStrategy = new ArrayList<>();
87     }
88
89     static class Lifecycle {
90         double[] pDetonate;
91         double tDetonateAbs;
92
93         Lifecycle(double[] pDetonate, double tDetonateAbs) {
94             this.pDetonate = pDetonate; this.tDetonateAbs = tDetonateAbs;
95         }
96     }
97
98
99     // ---- Helpers: vectors & ranges ----
100    static double[] add(double[] a, double[] b) { return new double[]{a[0]+
101        b[0], a[1]+b[1], a[2]+b[2]}; }
102    static double[] sub(double[] a, double[] b) { return new double[]{a[0]-
103        b[0], a[1]-b[1], a[2]-b[2]}; }
```

```

91     static double[] mul(double[] a, double s) { return new double[]{a[0]*s, a[1]*s, a[2]*s}; }
92     static double dot(double[] a, double[] b) { return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]; }
93     static double norm2(double[] a) { return Math.sqrt(dot(a,a)); }
94     static double[] normalize(double[] a) {
95         double n = norm2(a); return n==0 ? new double[]{0,0,0} : new double[]
96             []{a[0]/n,a[1]/n,a[2]/n};
97     }
98     static double uniform(Random rng, double lo, double hi) {
99         return lo + rng.nextDouble() * (hi - lo);
100    }

101   // 环向安全归一: 返回 [0,360)
102   static double norm360(double deg){ double x = deg % 360.0; if (x < 0) x
103     += 360.0; return x; }

104   // 在给定 UAV 的角度区间内采样 (支持环向)
105   static double sampleDirectionDeg(String uavId, Random rng) {
106       double[] range = ANGLE_RANGE_DEG.get(uavId);
107       if (range == null) return rng.nextDouble() * 360.0; // 兜底: 全域
108       double a = norm360(range[0]);
109       double b = norm360(range[1]);
110       if (a == b) return a; // 单点
111       if (a < b) { // 普通区间
112           return uniform(rng, a, b);
113       } else { // 环向区间 (跨越0°)
114           double w1 = 360.0 - a; // [a,360)
115           double w2 = b; // [0,b)
116           double w = w1 + w2;
117           double r = rng.nextDouble() * w;
118           if (r < w1) return a + r; // 落在[a,360)
119           return r - w1; // 落在[0,b)
120       }
121   }

122   // ---- Generate target points on cylinder rims across heights ----
123   static List<double[]> generateTargetPoints(double[] baseCenter, double
124     radius, double height,
125                                         int numRimPoints, int
126                                         numHeightLevels) {
127       List<double[]> pts = new ArrayList<>();
128       for (int i = 0; i < numHeightLevels; i++) {
           double h = (numHeightLevels==1) ? 0 : (height * i / (

```

```

        numHeightLevels-1.0));
129    for (int k = 0; k < numRimPoints; k++) {
130        double angle = 2.0*Math.PI*k/numRimPoints;
131        double x = baseCenter[0] + radius * Math.cos(angle);
132        double y = baseCenter[1] + radius * Math.sin(angle);
133        double z = baseCenter[2] + h;
134        pts.add(new double[]{x,y,z});
135    }
136}
137 return pts;
138}

139

140 // ---- Core geometry test: line segment (missile->targetPoint) blocked
141 // by sphere (smoke) ----
142 static boolean isSingleLineBlocked(double[] missilePos, double[]
143 smokeCenter, double[] targetPoint) {
144     double[] AB = sub(targetPoint, missilePos);
145     double[] AC = sub(smokeCenter, missilePos);
146     double denom = dot(AB, AB);
147     double tProj = denom == 0 ? 0 : dot(AC, AB) / denom;
148     double t = Math.max(0.0, Math.min(1.0, tProj));
149     double[] closest = add(missilePos, mul(AB, t));
150     double dx = smokeCenter[0]-closest[0], dy = smokeCenter[1]-closest
151 [1], dz = smokeCenter[2]-closest[2];
152     double dist2 = dx*dx + dy*dy + dz*dz;
153     return dist2 <= SMOKE_CLOUD_RADIUS_M * SMOKE_CLOUD_RADIUS_M;
154 }

155

156 // ---- At time t: is the entire target obscured for a missile? ----
157 static boolean getObscurationStateAtTime(double t, String missileId,
158 List<Lifecycle> lifecycles) {
159     List<double[]> activeCenters = new ArrayList<>();
160     for (Lifecycle g : lifecycles) {
161         double td = g.tDetonateAbs;
162         if (t >= td && t < td + SMOKE_EFFECTIVE_DURATION_S) {
163             double dt = t - td;
164             double[] sink = new double[]{0,0, -SMOKE_SINK_SPEED_MPS *
165 dt};
166             activeCenters.add(add(g.pDetonate, sink));
167         }
168     }
169     if (activeCenters.isEmpty()) return false;
170
171     double[] missilePos = add(INITIAL_POSITIONS.get(missileId),
172 mul(MISSILE_DIRECTIONS.get(missileId), MISSILE_SPEED_MPS *

```

```
        t));  
168  
169     for (double[] targetPoint : REAL_TARGET_POINTS) {  
170         boolean blocked = false;  
171         for (double[] center : activeCenters) {  
172             if (isSingleLineBlocked(missilePos, center, targetPoint)) {  
173                 blocked = true; break;  
174             }  
175             if (!blocked) return false; // any visible point → not fully  
176             obscured  
177         }  
178     }  
179     return true;  
180 }  
181 // ---- Binary search a state transition time in [tStart, tEnd] ----  
182 static double findTransitionTime(double tStart, double tEnd, String  
183     missileId, boolean startState,  
184     List<Lifecycle> lifecycles, double  
185     precision) {  
186     double low = tStart, high = tEnd;  
187     while ((high - low) > precision) {  
188         double mid = 0.5 * (low + high);  
189         boolean state = getObscurationStateAtTime(mid, missileId,  
190             lifecycles);  
191         if (state == startState) low = mid; else high = mid;  
192     }  
193     return high;  
194 }  
195 // ---- Main simulation with adaptive stepping via coarse grid + binary  
196 // search ----  
197 static Map<String, Double> calculateSynergisticObscurationAdaptive(  
198     Map<String, UavPlan> uavPlans,  
199     List<String> missileIds,  
200     double coarseTimeStep,  
201     double precision,  
202     double maxSimTime  
203 ) {  
204     // 1) Precompute grenade lifecycles  
205     List<Lifecycle> lifecycles = new ArrayList<>();  
206     for (Map.Entry<String, UavPlan> e : uavPlans.entrySet()) {  
207         UavPlan p = e.getValue();  
208         if (p == null || p.grenadeStrategy == null) continue;  
209         double[] p0 = INITIAL_POSITIONS.get(e.getKey());  
210         double rad = Math.toRadians(p.directionDeg);  
211     }
```

```
206     double[] vUav = new double[]{p.uavSpeed * Math.cos(rad), p.  
207         uavSpeed * Math.sin(rad), 0};  
  
208     for (GrenadeStrategy gs : p.grenadeStrategy) {  
209         double tRelease = gs.tRelease;  
210         double tAfter = gs.tDetonateAfter;  
211         double[] pDrop = add(p0, mul(vUav, tRelease));  
212         double[] pDet = new double[] {  
213             pDrop[0] + vUav[0] * tAfter,  
214             pDrop[1] + vUav[1] * tAfter,  
215             pDrop[2] - 0.5 * G * tAfter * tAfter  
216         };  
217         lifecycles.add(new Lifecycle(pDet, tRelease + tAfter));  
218     }  
219 }  
  
220  
221 // 2) Event-driven simulation per missile  
222 Map<String, Double> total = new HashMap<>();  
223 for (String mId : missileIds) total.put(mId, 0.0);  
  
224  
225 for (String missileId : missileIds) {  
226     double lastT = 0.0;  
227     boolean lastState = getObscurationStateAtTime(0.0, missileId,  
228         lifecycles);  
229     Double obscStart = lastState ? 0.0 : null;  
  
230     int steps = (int) Math.ceil(maxSimTime / coarseTimeStep);  
231     for (int i = 1; i <= steps; i++) {  
232         double curT = i * coarseTimeStep;  
233         boolean curState = getObscurationStateAtTime(curT,  
234             missileId, lifecycles);  
  
235         if (curState != lastState) {  
236             double tTrans = findTransitionTime(lastT, curT,  
237                 missileId, lastState, lifecycles, precision);  
238             if (curState) {  
239                 obscStart = tTrans;  
240             } else {  
241                 if (obscStart != null) {  
242                     total.put(missileId, total.get(missileId) + (  
243                         tTrans - obscStart));  
244                     obscStart = null;  
245                 }  
246             }  
247         }  
248     }  
249 }
```

```

246         lastT = curT; lastState = curState;
247     }
248     if (obscStart != null) {
249         total.put(missileId, total.get(missileId) + (maxSimTime -
250             obscStart));
251     }
252     return total;
253 }
254
255 // ---- Public wrappers ----
256 static Map<String, Double> calculateTime(Map<String, UavPlan> plans,
257                                             double coarseTimeStep, double
258                                             precision) {
259     return calculateSynergisticObscurationAdaptive(
260         plans, MISSILE_IDS, coarseTimeStep, precision, 100.0
261     );
262 }
263 static double sumMissileTimes(Map<String, Double> perMissile) {
264     double s = 0.0;
265     for (String m : MISSILE_IDS) s += perMissile.getOrDefault(m, 0.0);
266     return s;
267 }
268
269 // ---- High-fidelity single-smoke test used by seed search ----
270 static boolean isTargetFullyObscuredAtTime(double t, String missileId,
271     List<double[]> activeSmokeCenters) {
272     if (activeSmokeCenters == null || activeSmokeCenters.isEmpty())
273         return false;
274     double[] missilePos = add(INITIAL_POSITIONS.get(missileId),
275                               mul(MISSILE_DIRECTIONS.get(missileId), MISSILE_SPEED_MPS *
276                                   t));
277     for (double[] tp : REAL_TARGET_POINTS) {
278         boolean blocked = false;
279         for (double[] c : activeSmokeCenters) {
280             if (isSingleLineBlocked(missilePos, c, tp)) { blocked =
281                 true; break; }
282         }
283         if (!blocked) return false;
284     }
285     return true;
286 }
287
288 // ---- Any-missile obscuration test (for seed search) ----
289 static boolean isObscuredForAnyMissile(double t, List<double[]>

```

```

    activeSmokeCenters) {
285     for (String mId : MISSILE_IDS) {
286         if (isTargetFullyObscuredAtTime(t, mId, activeSmokeCenters))
287             return true;
288     }
289 }
290
291 // ---- FY4: check strictly positive duration (>0) using two-time
292 // sampling ----
293 static boolean grenadeHasPositiveDurationForAnyMissile(double
294     tIntercept, double[] pSmoke) {
295     double dt = 0.05; // 50ms 正时长近似
296     double[] c0 = pSmoke;
297     double[] c1 = new double[]{pSmoke[0], pSmoke[1], pSmoke[2] -
298         SMOKE_SINK_SPEED_MPS * dt};
299     for (String mId : MISSILE_IDS) {
300         boolean b0 = isTargetFullyObscuredAtTime(tIntercept, mId,
301             Collections.singletonList(c0));
302         boolean b1 = isTargetFullyObscuredAtTime(tIntercept + dt, mId,
303             Collections.singletonList(c1));
304         if (b0 && b1) return true; // 同一导弹在相邻两时刻均被完全遮蔽
305         // → 持续时间>0
306     }
307     return false;
308 }
309
310 // ---- Search helpers ----
311 static double[] findInterceptionDetailsForGrenadeAny(
312     double[] vUav, double[] uavInitialPos, double tReleaseMin,
313     double tReleaseMax, Random rng
314 ) {
315     List<Double> releases = linspaceShuffled(tReleaseMin, tReleaseMax,
316         20, rng);
317     for (double tRelease : releases) {
318         double[] pDrop = add(uavInitialPos, mul(vUav, tRelease));
319         List<Double> dets = linspaceShuffled(0.1, 8.0, 40, rng);
320         for (double tAfter : dets) {
321             double tIntercept = tRelease + tAfter;
322             double[] pSmoke = new double[]{
323                 pDrop[0] + vUav[0]*tAfter,
324                 pDrop[1] + vUav[1]*tAfter,
325                 pDrop[2] - 0.5 * G * tAfter * tAfter
326             };
327             if (isObscuredForAnyMissile(tIntercept, Collections.

```

```

    singletonList(pSmoke))) {
        return new double[]{tRelease, tAfter};
    }
}
return null;
}

static double[] findInterceptionDetailsForGrenadeAny_PositiveDuration(
    double[] vUav, double[] uavInitialPos, double tReleaseMin,
    double tReleaseMax, Random rng
) {
    List<Double> releases = linspaceShuffled(tReleaseMin, tReleaseMax,
        24, rng);
    for (double tRelease : releases) {
        double[] pDrop = add(uavInitialPos, mul(vUav, tRelease));
        List<Double> dets = linspaceShuffled(0.1, 10.0, 48, rng);
        for (double tAfter : dets) {
            double tIntercept = tRelease + tAfter;
            double[] pSmoke = new double[]{
                pDrop[0] + vUav[0]*tAfter,
                pDrop[1] + vUav[1]*tAfter,
                pDrop[2] - 0.5 * G * tAfter * tAfter
            };
            if (grenadeHasPositiveDurationForAnyMissile(tIntercept,
                pSmoke)) {
                return new double[]{tRelease, tAfter};
            }
        }
    }
    return null;
}

static List<Double> linspaceShuffled(double a, double b, int n, Random
    rng) {
    List<Double> arr = new ArrayList<>();
    if (n <= 1) { arr.add(a); return arr; }
    for (int i=0;i<n;i++) arr.add(a + i*(b-a)/(n-1.0));
    Collections.shuffle(arr, rng);
    return arr;
}

// ---- Seed generator per UAV (FY4: at least TWO valid grenades with
// positive duration) ----
static List<double[]> generateSeedsForUav(String uavId, int numSeeds,
    Random rng) {
    if ("FY4".equals(uavId)) return generateSeedsForUavFY4TwoValid(

```

```
        numSeeds, rng);

358
359     System.out.println("[" + uavId + "] 正在生成 " + numSeeds + " 个高
360                         质量种子...");
361     List<double[]> seeds = new ArrayList<>();
362     double[] pUav = INITIAL_POSITIONS.get(uavId);

363     int tries = 0;
364     while (seeds.size() < numSeeds) {
365         tries++;
366         // 使用你给定的角度范围采样 (含环向)
367         double directionDeg = sampleDirectionDeg(uavId, rng);
368         double uavSpeed = 70.0 + rng.nextDouble()*(140.0-70.0);
369         double rad = Math.toRadians(directionDeg);
370         double[] vUav = new double[]{uavSpeed*Math.cos(rad), uavSpeed*
371                                     Math.sin(rad), 0};

372         double[] g1 = findInterceptionDetailsForGrenadeAny(vUav, pUav,
373                     1.0, 20.0, rng);
374         if (g1 == null) continue;

375         double t1 = g1[0], d1 = g1[1];
376         double[] g2 = findInterceptionDetailsForGrenadeAny(vUav, pUav,
377                     t1+1.0, t1+15.0, rng);
378         if (g2 == null) continue;

379         double t2 = g2[0], d2 = g2[1];
380         double[] g3 = findInterceptionDetailsForGrenadeAny(vUav, pUav,
381                     t2+1.0, t2+15.0, rng);
382         if (g3 == null) continue;

383         double t3 = g3[0], d3 = g3[1];
384         double gGap2 = t2 - t1 - 1.0;
385         double gGap3 = t3 - t2 - 1.0;
386         if (gGap2 < 0 || gGap3 < 0) continue;

387         double[] seed = new double[]{directionDeg, uavSpeed, t1, d1,
388                                     gGap2, d2, gGap3, d3};
389         seeds.add(seed);
390         System.out.printf("[" + uavId + "] Seed %d/%d (trials=%d)\n",
391                           seeds.size(), numSeeds, tries);
391     }
392     System.out.println("[" + uavId + "] 高质量种子生成完毕! ");
393     return seeds;
394 }
```

```

395
396     static List<double[]> generateSeedsForUavFY4TwoValid(int numSeeds,
397         Random rng) {
398         final String uavId = "FY4";
399         System.out.println("[FY4] 使用 “两枚弹必须有效”的标准生成种子: 每
400             枚需在起爆后具有>0的遮蔽时长, 第三枚用占位值。");
401         List<double[]> seeds = new ArrayList<>();
402         double[] pUav = INITIAL_POSITIONS.get(uavId);
403
404         int tries = 0;
405         while (seeds.size() < numSeeds) {
406             tries++;
407             // FY4 也采用你给定的角度范围
408             double directionDeg = sampleDirectionDeg(uavId, rng);
409             double uavSpeed = 70.0 + rng.nextDouble()*(140.0-70.0);
410             double rad = Math.toRadians(directionDeg);
411             double[] vUav = new double[]{uavSpeed*Math.cos(rad), uavSpeed*
412                 Math.sin(rad), 0};
413
414             // 第一枚: 正遮蔽时长
415             double[] g1 =
416                 findInterceptionDetailsForGrenadeAny_PositiveDuration(vUav,
417                     pUav, 0.5, 30.0, rng);
418             if (g1 == null) continue;
419             double t1 = g1[0], d1 = g1[1];
420
421             // 第二枚: 同样要求正遮蔽时长, 且保证时间顺序 (至少相隔 1s)
422             double[] g2 =
423                 findInterceptionDetailsForGrenadeAny_PositiveDuration(vUav,
424                     pUav, t1 + 1.0, t1 + 15.0, rng);
425             if (g2 == null) continue;
426             double t2 = g2[0], d2 = g2[1];
427             double gGap2 = t2 - t1 - 1.0;
428             if (gGap2 < 0) continue;
429
430             // 第三枚: 占位最小合法值, 优化阶段可自行调优为有效
431             double gGap3 = 0.0;
432             double d3 = 0.1;
433
434             double[] seed = new double[]{directionDeg, uavSpeed, t1, d1,
435                 gGap2, d2, gGap3, d3};
436             seeds.add(seed);
437             System.out.printf("[FY4] Seed %d/%d (trials=%d)\n",
438                 seeds.size(),
439                 numSeeds, tries);
440         }

```

```
431     System.out.println("[FY4] “两枚弹有效” 种子生成完毕！优化阶段仍可  
432         把第三枚调整为有效值。");  
433     return seeds;  
434 }  
  
435 // ---- Bounds (same layout as Python) ----  
436 static final double[][] BOUNDS = new double[][]{  
437     {0,360},      // direction_deg (采样已受限于 ANGLE_RANGE_DEG, 此  
438         处边界仍保留)  
439     {70,140},     // uav_speed  
440     {0,50},       // t_release1  
441     {0.1,10},     // t_detonate1  
442     {0,15},       // t_gap2  
443     {0.1,10},     // t_detonate2  
444     {0,15},       // t_gap3  
445     {0.1,10}      // t_detonate3  
446 };  
  
447 // ---- Chromosome decode/encode ----  
448 static Map<String, UavPlan> decodeChromosomeForUav(String uavId, double  
449     [] chromosome) {  
450     double direction = chromosome[0];  
451     double speed = chromosome[1];  
452     double t1 = chromosome[2];  
453     double d1 = chromosome[3];  
454     double g2 = chromosome[4];  
455     double d2 = chromosome[5];  
456     double g3 = chromosome[6];  
457     double d3 = chromosome[7];  
458     double t2 = t1 + 1.0 + g2;  
459     double t3 = t2 + 1.0 + g3;  
  
460     UavPlan plan = new UavPlan();  
461     plan.directionDeg = direction;  
462     plan.uavSpeed = speed;  
463     plan.grenadeStrategy.add(new GrenadeStrategy(t1, d1));  
464     plan.grenadeStrategy.add(new GrenadeStrategy(t2, d2));  
465     plan.grenadeStrategy.add(new GrenadeStrategy(t3, d3));  
  
466     Map<String, UavPlan> plans = new HashMap<>();  
467     plans.put(uavId, plan); // active  
468     for (String other : UAV_IDS) if (!other.equals(uavId)) plans.put(  
469         other, stubPlan());  
470     return plans;  
471 }
```

```
472     static double[] encodeChromosomeFromPlan(UavPlan p) {
473         // layout: [dir, speed, t1, d1, g2, d2, g3, d3]
474         double dir = p.directionDeg;
475         double spd = p.uavSpeed;
476         double t1=0,d1=0,t2=0,d2=0,t3=0,d3=0;
477         if (p.grenadeStrategy.size() >= 1) { t1 = p.grenadeStrategy.get(0).tRelease; d1 = p.grenadeStrategy.get(0).tDetonateAfter; }
478         if (p.grenadeStrategy.size() >= 2) { t2 = p.grenadeStrategy.get(1).tRelease; d2 = p.grenadeStrategy.get(1).tDetonateAfter; }
479         if (p.grenadeStrategy.size() >= 3) { t3 = p.grenadeStrategy.get(2).tRelease; d3 = p.grenadeStrategy.get(2).tDetonateAfter; }
480         double g2 = t2 - t1 - 1.0;
481         double g3 = t3 - t2 - 1.0;
482         return new double[]{dir, spd, t1, d1, g2, d2, g3, d3};
483     }
484
485     static UavPlan stubPlan() {
486         UavPlan p = new UavPlan();
487         p.directionDeg = 0; p.uavSpeed = 70; // idle, no grenades
488         return p;
489     }
490
491     // ---- Fitness wrappers ----
492     static double fitnessForUav_Independent(String uavId, double[]
493         chromosome) {
494         Map<String, UavPlan> plans = decodeChromosomeForUav(uavId,
495             chromosome);
496         Map<String, Double> res = calculateTime(plans, 0.005, 0.0001);
497         return sumMissileTimes(res);
498     }
499     static double fitnessForUav_WithFixed(String uavId, double[] chromosome,
500         Map<String, UavPlan> fixed) {
501         Map<String, UavPlan> plans = new HashMap<>();
502         for (String id : UAV_IDS) {
503             if (fixed.containsKey(id)) plans.put(id, fixed.get(id));
504         }
505         Map<String, UavPlan> cand = decodeChromosomeForUav(uavId,
506             chromosome);
507         plans.put(uavId, cand.get(uavId));
508         for (String id : UAV_IDS) plans.putIfAbsent(id, stubPlan());
509         Map<String, Double> res = calculateTime(plans, 0.005, 0.0001);
510         return sumMissileTimes(res);
511     }
512
513     // ---- Simplified CMA-ES ----
```

```
510 static class SimpleCMAES {  
511     final int n;                      // dimension  
512     final double[] lower, upper; // bounds  
513     final Random rng;  
514  
515     int lambda;                     // population size  
516     int mu;                         // parents  
517     double[] weights;               // recombination weights  
518     double mueff;  
519     double[] mean;                  // current mean  
520     double sigma;                  // global step-size  
521  
522     SimpleCMAES(double[] x0, double sigma0, double[][] bounds, Random  
523         rng) {  
524         this.n = x0.length;  
525         this.lower = new double[n];  
526         this.upper = new double[n];  
527         for (int i=0;i<n;i++){ lower[i]=bounds[i][0]; upper[i]=bounds[i]  
528             ][1]; }  
529         this.rng = rng;  
530         this.mean = Arrays.copyOf(x0, n);  
531         this.sigma = sigma0;  
532  
533         this.lambda = 4 + (int)Math.floor(3*Math.log(n));  
534         this.mu = Math.max(2, lambda/2);  
535         this.weights = new double[mu];  
536         for (int i=0;i<mu;i++) weights[i] = Math.log(mu+0.5) - Math.log  
537             (i+1);  
538         double wsum = 0; for (double w:weights) wsum += w;  
539         for (int i=0;i<mu;i++) weights[i] /= wsum;  
540         double w2 = 0; for (double w:weights) w2 += w*w;  
541         this.mueff = 1.0 / w2;  
542     }  
543  
544     double[] optimize(java.util.function.ToDoubleFunction<double[]>  
545         objective,  
546         int maxIter) {  
547         double bestF = Double.NEGATIVE_INFINITY;  
548         double[] bestX = Arrays.copyOf(mean, n);  
549  
550         for (int it=0; it<maxIter; it++) {  
551             List<Cand> pop = new ArrayList<>(lambda);  
552             for (int k=0;k<lambda;k++) {  
553                 double[] z = randn(n);  
554                 double[] x = new double[n];  
555                 Cand c = new Cand(x, objective, z);  
556                 pop.add(c);  
557             }  
558             Collections.sort(pop);  
559             if (pop.get(0).f < bestF) {  
559                 bestF = pop.get(0).f;  
560                 bestX = pop.get(0).x;  
561             }  
562             for (int k=0;k<lambda;k++) {  
563                 double[] z = pop.get(k).z;  
564                 double[] x = pop.get(k).x;  
565                 double[] xnew = recombine(x, z, weights);  
566                 double fnew = objective.f(xnew);  
567                 if (fnew < bestF) {  
568                     bestF = fnew;  
569                     bestX = xnew;  
570                 }  
571             }  
572         }  
573     }  
574 }
```

```
551         for (int i=0;i<n;i++) {
552             x[i] = mean[i] + sigma * z[i];
553             if (x[i]<lower[i]) x[i]=lower[i];
554             if (x[i]>upper[i]) x[i]=upper[i];
555         }
556         double f = objective.applyAsDouble(x);
557         pop.add(new Cand(x,f));
558     }
559     pop.sort((a,b)-> Double.compare(b.f, a.f)); // max
560
561     double[] newMean = new double[n];
562     for (int i=0;i<mu;i++) {
563         double w = weights[i];
564         double[] xi = pop.get(i).x;
565         for (int j=0;j<n;j++) newMean[j] += w * xi[j];
566     }
567
568     double oldGain = objective.applyAsDouble(mean);
569     double newGain = objective.applyAsDouble(newMean);
570     boolean success = newGain > oldGain;
571     sigma *= success ? 1.05 : 0.95;
572
573     mean = newMean;
574
575     if (pop.get(0).f > bestF) {
576         bestF = pop.get(0).f;
577         bestX = Arrays.copyOf(pop.get(0).x, n);
578     }
579 }
580 return Arrays.copyOf(bestX, n);
581 }
582
583 static class Cand {
584     double[] x; double f;
585     Cand(double[] x, double f){ this.x=x; this.f=f; }
586 }
587
588 double[] randn(int d) {
589     double[] z = new double[d];
590     for (int i=0;i<d;i++) z[i] = gaussian();
591     return z;
592 }
593 double gaussian() {
594     double u1 = Math.max(1e-12, rng.nextDouble());
595     double u2 = rng.nextDouble();
```

```

596         return Math.sqrt(-2.0*Math.log(u1)) * Math.cos(2*Math.PI*u2);
597     }
598 }
599
600 // ---- Optimize wrapper ----
601 static class OptimizeResult {
602     double[] bestSolution;
603     double bestFitness;
604     OptimizeResult(double[] x, double f){ bestSolution=x; bestFitness=f
605     ; }
606 }
607 static OptimizeResult optimizeWithCMAES(double[] initialSolution,
608                                         java.util.function.
609                                         ToDoubleFunction<double []>
610                                         fitnessFunc,
611                                         int maxIter,
612                                         Random rng) {
613     SimpleCMAES es = new SimpleCMAES(initialSolution, /*sigma0*/0.1,
614                                         BOUNDS, rng);
615     double[] xbest = es.optimize(fitnessFunc, maxIter);
616     double fbest = fitnessFunc.applyAsDouble(xbest);
617     return new OptimizeResult(xbest, fbest);
618 }
619
620 // ---- Independent optimization for a chosen UAV ----
621 static class UavOptimizationOutcome {
622     String uavId;
623     double[] bestSol;
624     double bestFitness;
625     Map<String, Double> finalResPerMissile;
626     UavPlan plan;
627 }
628
629 static UavOptimizationOutcome optimizeForUav_Independent(String uavId,
630     int numSeeds, int maxIter, Random rng) {
631     List<double []> seeds = generateSeedsForUav(uavId, numSeeds, rng);
632     if (seeds.isEmpty()) {
633         System.out.println("[" + uavId + "] 错误: 未生成有效初始种子,
634             跳过该无人机。");
635         return null;
636     }
637     System.out.println("[" + uavId + "] 开始使用CMA-ES进行独立优化 (目
638         标: 最大化 M1+M2+M3 遮蔽总时长) ...");
639
640     double[] bestSol = null; double bestFit = -1; int done = 0;

```

```

634     for (double[] seed : seeds) {
635         OptimizeResult r = optimizeWithCMAES(seed, x ->
636             fitnessForUav_Independent(uavId, x), maxIter, rng);
637         if (r.bestFitness > bestFit) { bestFit = r.bestFitness; bestSol
638             = Arrays.copyOf(r.bestSolution, r.bestSolution.length); }
639         done++;
640         System.out.printf("[" + uavId + "] 优化进度 %d/%d 当前最优适应
641             度(总遮蔽秒): %.3f\n", done, seeds.size(), bestFit);
642     }
643     if (bestSol == null) return null;
644
645     Map<String, UavPlan> bestPlans = decodeChromosomeForUav(uavId,
646         bestSol);
647     Map<String, Double> finalRes = calculateTime(bestPlans, 0.0001,
648         0.0000001);
649
650     UavOptimizationOutcome out = new UavOptimizationOutcome();
651     out.uavId = uavId;
652     out.bestSol = bestSol;
653     out.bestFitness = bestFit;
654     out.finalResPerMissile = finalRes;
655     out.plan = bestPlans.get(uavId);
656     return out;
657 }
658
659 // ---- 贪心拼装 (核心) : 按 “指定顺序” 拼装 -----
660 static Map<String, UavPlan> buildInitialAssemblyGreedyWithOrder(List<
661     String> order, int seedsPerUav, long baseSeed) {
662     Map<String, UavPlan> current = new HashMap<>();
663     for (String id : UAV_IDS) current.put(id, null); // start empty
664
665     System.out.println("\n[Greedy] 使用顺序: " + order);
666
667     for (String uav : order) {
668         System.out.println("[Greedy] 处理 " + uav + "...");
669         // 为保证不同顺序比较的公平性: 同一 UAV 使用同一个随机种子, 避
670         // 免因顺序不同导致候选集不同
671         long perUavSeed = baseSeed + Math.abs(uav.hashCode());
672         List<double[]> candidates = generateSeedsForUav(uav,
673             seedsPerUav, new Random(perUavSeed));
674         if (candidates.isEmpty()) {
675             System.out.println("[Greedy] " + uav + " 无候选, 使用 stub
676             。");
677             current.put(uav, stubPlan());
678             continue;
679         }
680         double[] bestCandidate = candidates.get(0);
681         double bestFitness = fitnessForUav_Independent(uavId, bestCandidate);
682         for (int i = 1; i < candidates.size(); i++) {
683             double[] candidate = candidates.get(i);
684             double fitness = fitnessForUav_Independent(uavId, candidate);
685             if (fitness < bestFitness) {
686                 bestCandidate = candidate;
687                 bestFitness = fitness;
688             }
689         }
690         current.put(uav, new UavPlan(bestCandidate));
691     }
692     return current;
693 }

```

```

670 }
671     double best = -1; UavPlan bestPlan = null;
672     for (double[] ch : candidates) {
673         UavPlan plan = decodeChromosomeForUav(uav, ch).get(uav);
674         Map<String, UavPlan> tmp = new HashMap<>(current);
675         tmp.put(uav, plan);
676         for (String id : UAV_IDS) tmp.putIfAbsent(id, stubPlan());
677         double sc = sumMissileTimes(calculateTime(tmp, 0.005,
678             0.0001));
679         if (sc > best) { best = sc; bestPlan = plan; }
680     }
681     current.put(uav, bestPlan);
682     System.out.printf("[Greedy] 选择 %s 后当前三导弹总遮蔽时长=%f\n",
683                     uav, best);
684 }
685
686 // ---- 生成一个随机顺序 ----
687 static List<String> randomOrder(List<String> items, Random rng) {
688     List<String> ord = new ArrayList<>(items);
689     Collections.shuffle(ord, rng);
690     return ord;
691 }
692
693 // ---- 多顺序贪心拼装: 采样 numOrders 个随机顺序, 取最优 ----
694 static Map<String, UavPlan> buildInitialAssemblyGreedy_MultiOrders(int
695     seedsPerUav, long baseSeed, int numOrders, long orderSeed) {
696     double bestScore = -1;
697     Map<String, UavPlan> bestAssembly = null;
698     List<String> bestOrder = null;
699
700     Random rng = new Random(orderSeed);
701     for (int k = 1; k <= numOrders; k++) {
702         List<String> ord = randomOrder(UAV_IDS, new Random(rng.nextLong
703             ()) );
704         Map<String, UavPlan> assembly =
705             buildInitialAssemblyGreedyWithOrder(ord, seedsPerUav,
706             baseSeed);
707         double score = sumMissileTimes(calculateTime(completePlans(
708             assembly), 0.005, 0.0001));
709         System.out.printf("[Greedy-MO] 顺序 #%d 总遮蔽=%f, 顺序=%s\n",
710                         k, score, ord);
711         if (score > bestScore) {
712             bestScore = score;

```

```

707         bestAssembly = assembly;
708         bestOrder = ord;
709     }
710 }
711 System.out.printf("\n[Greedy-MO] 选用最优顺序: %s (% 联合初始解 =%.3
712 f)s) \n", bestOrder, bestScore);
713 return bestAssembly;
714 }
715 // ---- 补齐缺失 UAV 为 stubPlan (用于评估)
716 static Map<String, UavPlan> completePlans(Map<String, UavPlan>
717 maybePartial) {
718     Map<String, UavPlan> res = new HashMap<>(maybePartial);
719     for (String id : UAV_IDS) res.putIfAbsent(id, stubPlan());
720     return res;
721 }
722 // ---- SCF: 固定其它4架, 微调当前一架 ----
723 static UavPlan optimizeOneWithFixed(String uavId, Map<String, UavPlan>
724 fixedPlans,
725                                     int seeds, int iters, long seedBase
726                                     ) {
727     System.out.println("\n[SCF] 微调 " + uavId + ", 其它固定。");
728     List<double[]> seedsList = generateSeedsForUav(uavId, seeds, new
729     Random(seedBase));
730     if (fixedPlans.get(uavId) != null && fixedPlans.get(uavId).
731         grenadeStrategy.size() >= 3) {
732         seedsList.add(encodeChromosomeFromPlan(fixedPlans.get(uavId)));
733     }
734     double[] bestSol = null; double bestScore = -1;
735     int idx = 0;
736     for (double[] s : seedsList) {
737         OptimizeResult r = optimizeWithCMAES(
738             s,
739             x -> fitnessForUav_WithFixed(uavId, x, fixedPlans),
740             iters,
741             new Random(seedBase + 100 + idx)
742         );
743         if (r.bestFitness > bestScore) { bestScore = r.bestFitness;
744             bestSol = Arrays.copyOf(r.bestSolution, r.bestSolution.
745             length); }
746         idx++;
747     }
748     if (bestSol == null) return fixedPlans.get(uavId);
749     return decodeChromosomeForUav(uavId, bestSol).get(uavId);

```

```

744 }
745
746 // ---- SCF 总控: 多轮交替优化 (FY1→FY5 轮流微调) ----
747 static Map<String, UavPlan> solve_SCF() {
748     // 0) 初始拼装: 多顺序比较取优
749     int seedsPerUavInit = 50;
750     int numOrders = 10;           // 你希望的“随机生成10个顺序”
751     long baseSeed = 2025L;       // 候选生成的基准种子(与顺序无关)
752     long orderSeed = 4263L;      // 生成随机顺序的种子(可改)
753
754     Map<String, UavPlan> cur = buildInitialAssemblyGreedy_MultiOrders(
755         seedsPerUavInit, baseSeed, numOrders, orderSeed);
756     cur = completePlans(cur);
757
758     double bestTotal = sumMissileTimes(calculateTime(cur, 0.005,
759                                         0.0001));
760     System.out.printf("\n[Init] 初始拼装联合目标=%.3f\n", bestTotal);
761
762     // 1) 交替优化参数
763     int ROUNDS = 10;             // SCF 轮数
764     int seedsLocal = 12;         // 每次微调的种子数
765     int itersLocal = 120;        // 每个种子的 CMA-ES 迭代
766     double tolImprove = 0.01;    // 小于该提升则判为收敛
767
768     for (int r=1; r<=ROUNDS; r++) {
769         System.out.println("\n=====SCF 轮次 " + r + " =====");
770         double roundStart = bestTotal;
771
772         for (int i=0;i<UAV_IDS.size();i++) {
773             String uav = UAV_IDS.get(i);
774             Map<String, UavPlan> fixed = new HashMap<>(cur);
775             UavPlan newPlan = optimizeOneWithFixed(uav, fixed,
776                                                     seedsLocal, itersLocal, 3000L + r*1000L + i*100L);
777             cur.put(uav, newPlan);
778
779             double tot = sumMissileTimes(calculateTime(cur, 0.005,
780                                             0.0001));
781             System.out.printf("[SCF] 更新 %s 后联合目标=%.3f\n", uav,
782                             tot);
783             if (tot > bestTotal + 1e-9) bestTotal = tot;
784         }
785
786         double gain = bestTotal - roundStart;
787         System.out.printf("[SCF] 轮 %d 结束: 提升 %.3f\n", r, gain);
788     }
789
790     double gain = bestTotal - roundStart;
791     System.out.printf("[SCF] 轮 %d 结束: 提升 %.3f\n", r, gain);
792 }
```

```
783     if (gain < tolImprove) {
784         System.out.println("[SCF] 提升不足阈值， 提前收敛。");
785         break;
786     }
787 }
788
789 // 打印结果
790 Map<String, Double> finalPerMissile = calculateTime(cur, 0.0005,
791             0.000001);
792 double finalSum = sumMissileTimes(finalPerMissile);
793
794 System.out.println("\n===== 问题五 (SCF 联合最优) =====");
795 System.out.printf("三导弹总遮蔽时长: %.4fs\n", finalSum);
796 System.out.printf("M1: %.4fs, M2: %.4fs, M3: %.4fs\n",
797                 finalPerMissile.getOrDefault("M1", 0.0),
798                 finalPerMissile.getOrDefault("M2", 0.0),
799                 finalPerMissile.getOrDefault("M3", 0.0));
800
801 for (String uav : UAV_IDS) {
802     UavPlan p = cur.get(uav);
803     System.out.println("\n-----");
804     System.out.println("无人机 " + uav + " 最优投放策略");
805     System.out.println("-----");
806     System.out.printf("飞行方向: %.2f 度\n", p.directionDeg);
807     System.out.printf("飞行速度: %.2f m/s\n", p.uavSpeed);
808     System.out.println("干扰弹投放时序:");
809     for (int i=0;i<p.grenadeStrategy.size();i++) {
810         GrenadeStrategy g = p.grenadeStrategy.get(i);
811         System.out.printf("-----%d: 受领后 %.3fs 投放， 投放后 %.3fs 起爆\n",
812                         i+1, g.tRelease, g.tDetonateAfter);
813     }
814 }
815 System.out.println("=====\\n");
816
817 return cur;
818 }
819
820 public static void main(String[] args) {
821     solve_SCF();
822 }
```

822

}