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”为核心目标，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 $120\,\mathrm{m/s}$ 向假目标飞行，受领任务 $1.5\,\mathrm{s}$ 后投放 1 枚干扰弹，并于 $3.6\,\mathrm{s}$ 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 $300\,\mathrm{m/s}$）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（$70 \sim 140\,\mathrm{m/s}$），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 $1\,\mathrm{s}$；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 $g = 9.8\,\mathrm{m/s^2}$。
4. **烟幕模型**：
   (a) 起爆瞬时形成球形云团，*有效遮蔽半径* $r_s = 10\,\mathrm{m}$；云团中心以后随时间仅竖直向下匀速运动，速度 $v_s = 3\,\mathrm{m/s}$，无侧向漂移；
   (b) 有效遮蔽持续时间 $20\,\mathrm{s}$，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 $r_T = 7\,\mathrm{m}$、高 $h_T = 10\,\mathrm{m}$）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	$300\,\mathrm{m/s}$
$\mathbf{P}_M(t)$	导弹在时刻 t 的空间坐标	(x, y, z)，m
v_U	无人机飞行速度	$70 \sim 140\,\mathrm{m/s}$（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 $t_r = 1.5\,\mathrm{s}$ 时投放干扰弹，$t_e = 3.6\,\mathrm{s}$ 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

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

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

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

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

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

其中 $t_{\exp}=t_r+t_e$，且

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

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

烟幕云团球心运动 起爆后烟幕球心仅竖直下沉：

$$\mathbf{C}(t)=\mathbf{C}_{\exp}+\Big(0,0,-v_s\,(t-t_{\exp})\Big),\qquad t\ge t_{\exp}. \tag{4}$$

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

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

定义判定函数

$$g(t)=\max_{1\le i\le 24}\bigg(d\Big(\mathbf{C}(t),\,\overline{\mathbf{P}_M(t)\,T_i}\Big)-r_s\bigg). \tag{5}$$

当且仅当 $g(t)\le 0$，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 $\leq 10\,\mathrm{m}$。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 $t_{\max} = 20\,\mathrm{s}$）以粗步长（如 $\Delta t = 10^{-3}\,\mathrm{s}$）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 $10^{-5}\,\mathrm{s}$）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 $[8.0562, 9.4482]\,\mathrm{s}$，总遮蔽时长

$$\boxed{1.392\,\mathrm{s}}.$$

该结果与模型假设下的几何—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）

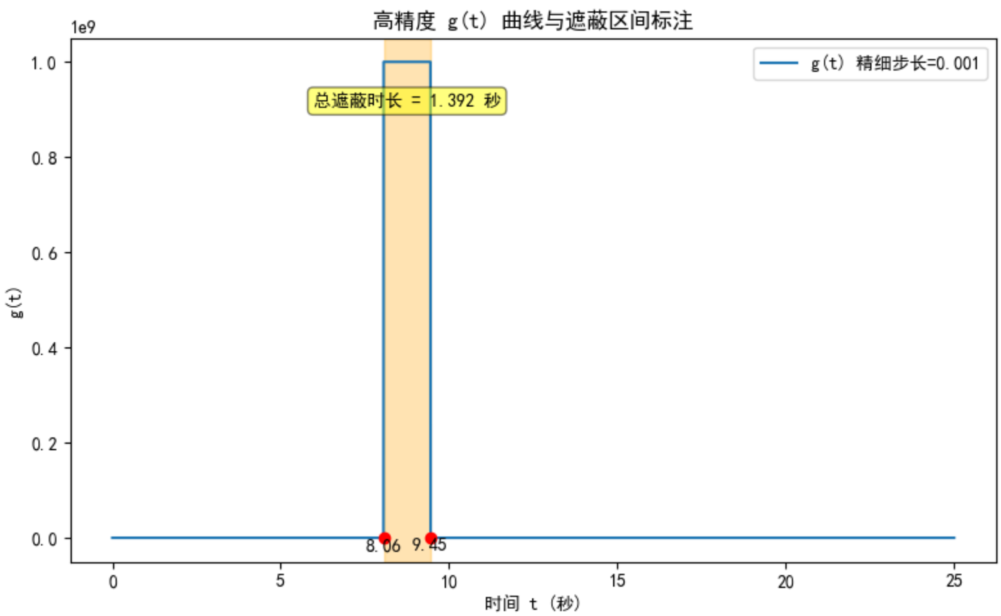

图 1: $g(t)$ 曲线与遮蔽区间标注（阴影为有效遮蔽）

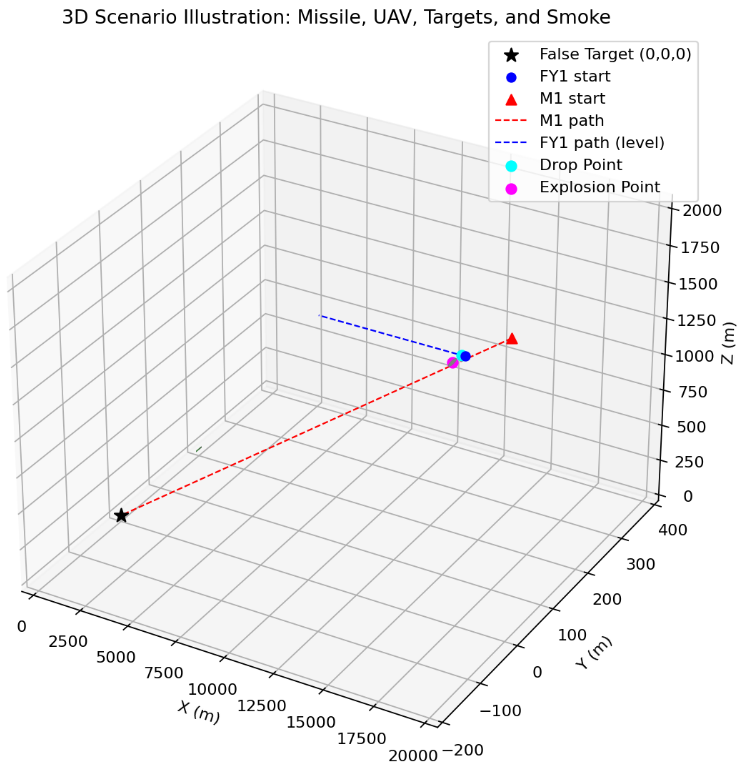

图 2: 三维场景示意（导弹轨迹、烟幕球心轨迹与目标）

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

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

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 $(-180°, 180°]$），$t_{\text{drop}} \in [0, 60]$ 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 $t_{\text{exp}} \le t_{\text{ME}}$，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	$[0, 60 - t_{\text{drop}}]$

“整体遮蔽”的目标函数定义为

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

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 $\Delta t = 0.01\,\text{s}$ 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（$J = 1.392\,\mathrm{s}$）相比，本方案将总遮蔽时长提升至 $4.5884\,\mathrm{s}$，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 $J^\star$ (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）。

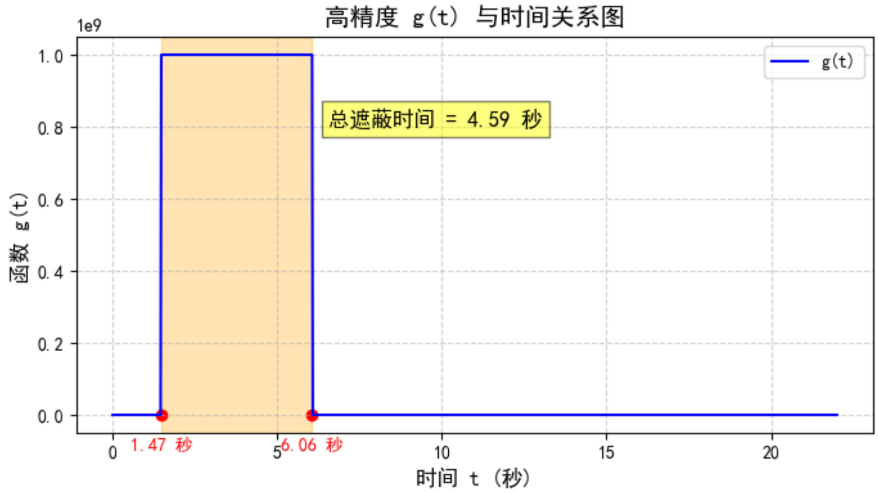

图 3: 最优解的 $g(t)$ 高精度曲线及遮蔽区间标注

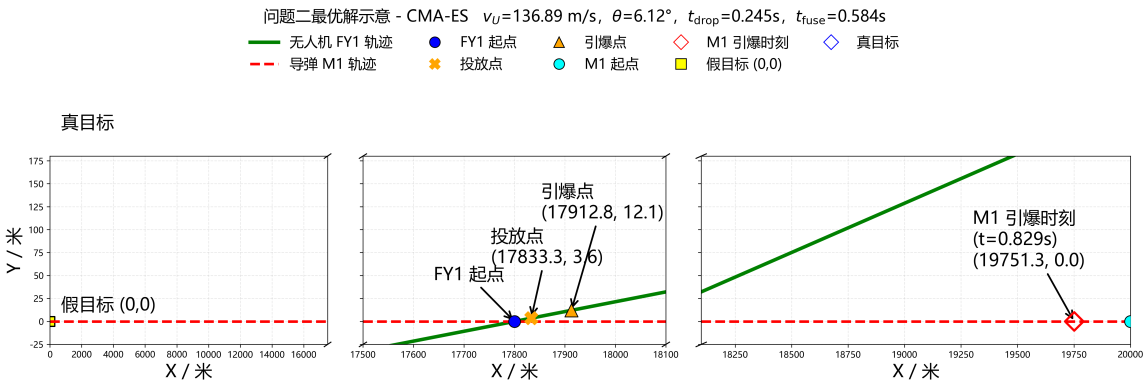

图 4: 二维场景示意图（断轴 x；含 FY1 轨迹、投放/起爆点、M1 起始与引爆位置、假目标）

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：$v_U^\star = 85.6\,\mathrm{m/s}$、$\theta^\star = 4.6°$、$t_{\mathrm{drop}}^\star = 1.16\,\mathrm{s}$、$t_{\mathrm{fuse}}^\star = 0.31\,\mathrm{s}$，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。

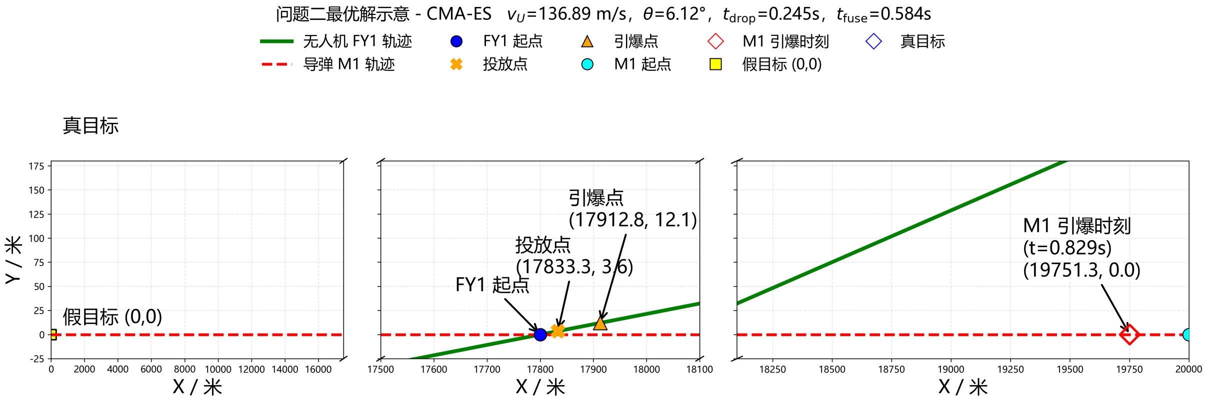

图 5: Nelder–Mead 解的 XY 平面示意图（断轴 x；含 FY1 轨迹与 M1 关键位置）

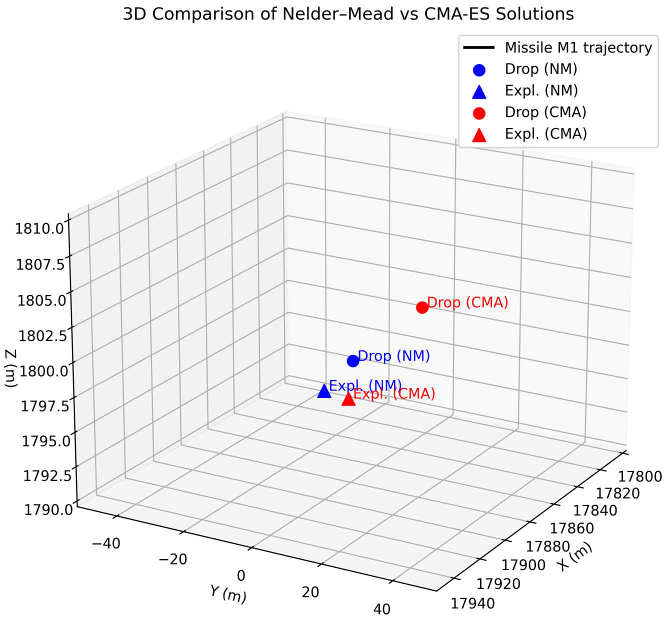

图 6: Nelder–Mead 与 CMA-ES 解的三维位置对比（无人机局部放大图）

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

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

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 $T_i \le t_{\mathrm{ME}}$（其中 t_{ME} 为导弹 M1 击中假目标的时刻），以及 $t_i, d_i, g_i \ge 0$。

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 → 解码为三次引爆的时空位置 → 检查所有"导弹—目标离散点"视线是否同时满足"线段最近点 ≤ 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 → 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 $\theta = 179.65°$；
- 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）。

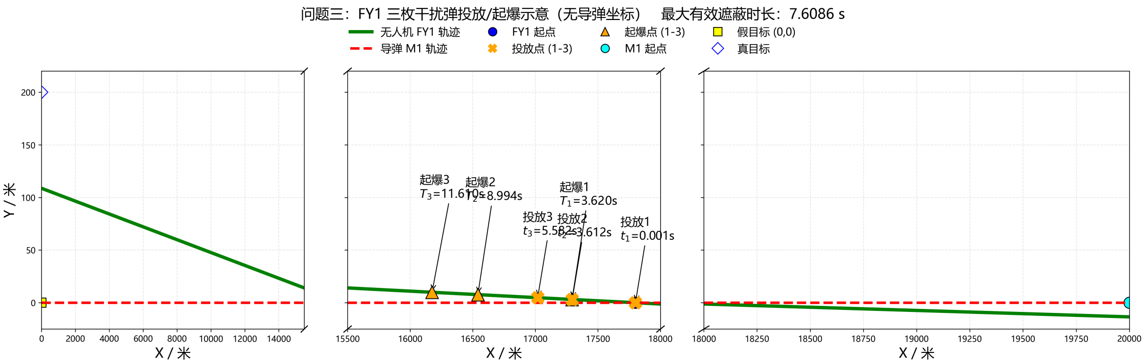

图 7: 问题三：FY1 三弹投放策略 XY 平面示意图（含全局、释放区、引爆区断层放大）

最终总有效遮蔽时长达到

$$\boxed{7.6086\ \mathrm{s}},$$

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

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

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 $r_s = 10\,\mathrm{m}$ 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 $\Delta t = 10^{-3}\,\mathrm{s}$）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 $10^{-5}\,\mathrm{s}$）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

$$\boxed{11.8025\ \mathrm{s}},$$

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。

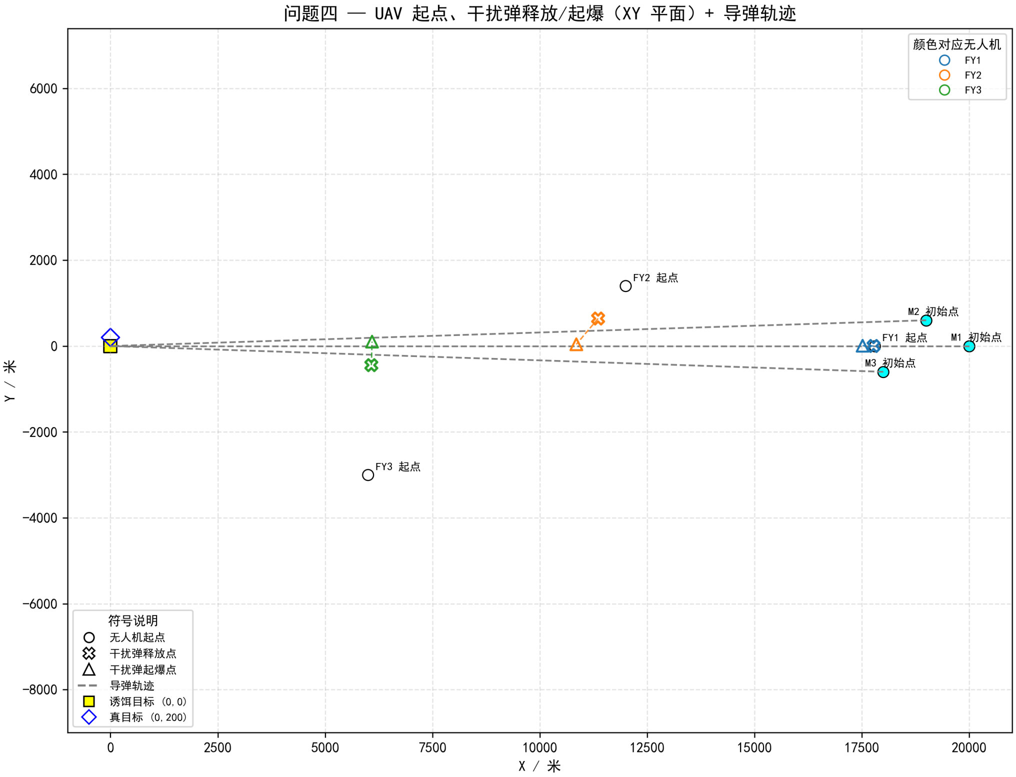

图 8: 问题四的 XY 平面投放策略示意图（含符号说明）

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。

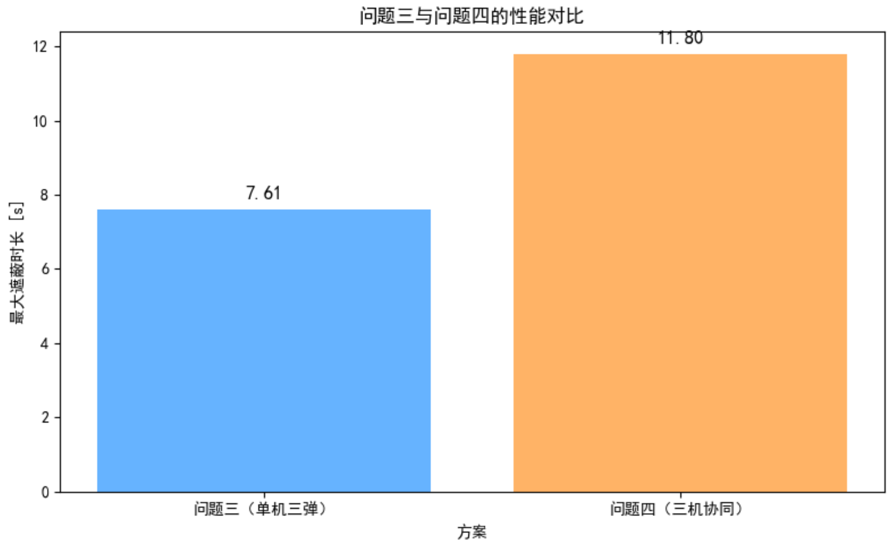

图 9: 问题三（单机三弹）与问题四（三机协同）的性能对比

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：

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

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 $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_5)$。

优化目标是最大化三枚导弹的总遮蔽时长之和：

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

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–福克自洽场”（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，我们引入自洽场迭代进行精细优化。在每一轮（Round）迭代中，我们按固定顺序（FY1 $\rightarrow$ FY2 $\rightarrow \cdots \rightarrow$ 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

$$\boxed{47.7814\ \mathrm{s}},$$

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。

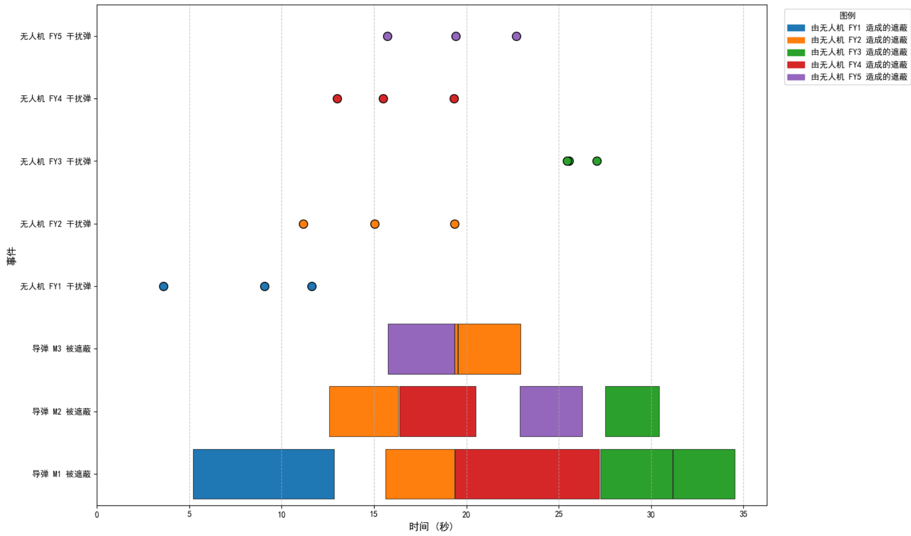

图 10: 有效遮蔽时间的时间轴

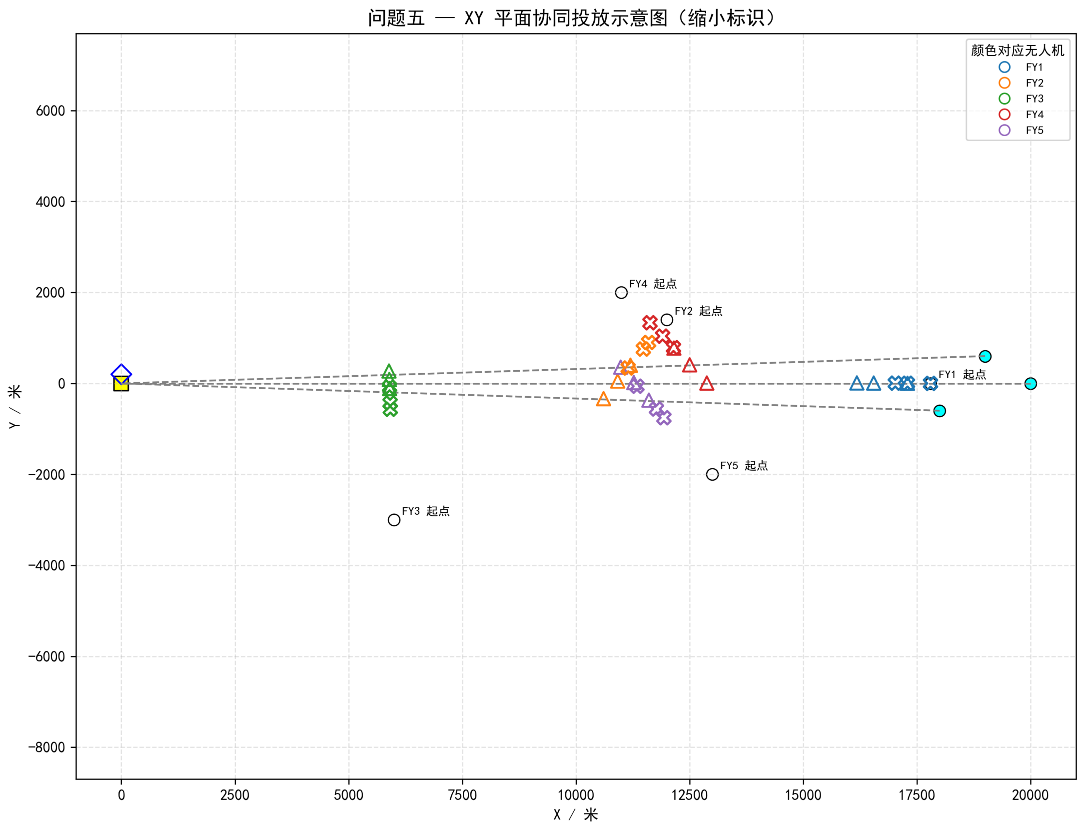

图 11: 复杂协同策略在 XY 平面上的几何布局

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

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

G = 9.8
MISSILE_SPEED_MPS = 300
SMOKE_CLOUD_RADIUS_M = 10
SMOKE_EFFECTIVE_DURATION_S = 20
SMOKE_SINK_SPEED_MPS = 3

TARGET_BASE_CENTER = np.array([0, 200, 0])
TARGET_RADIUS = 7.0
TARGET_HEIGHT = 10.0

INITIAL_POSITIONS = {
    "M1": np.array([20000, 0, 2000]), "M2": np.array([19000, 600,
        2100]), "M3": np.array([18000, -600, 1900]),
    "FY1": np.array([17800, 0, 1800]), "FY2": np.array([12000, 1400,
        1400]), "FY3": np.array([6000, -3000, 700]),
    "FY4": np.array([11000, 2000, 1800]), "FY5": np.array([13000,
        -2000, 1300]),
}
FAKE_TARGET_POS = np.array([0, 0, 0])
MISSILE_DIRECTIONS = {m_id: (FAKE_TARGET_POS - pos) / np.linalg.norm(
    FAKE_TARGET_POS - pos)
```

```
                      for m_id, pos in INITIAL_POSITIONS.items() if m_id.startswith("M")}


def generate_target_points(base_center, radius, height, num_rim_points=12, num_height_levels=2):
    points = []
    angles = np.linspace(0, 2 * np.pi, num_rim_points, endpoint=False)
    heights = np.linspace(0, height, num_height_levels)
    for h in heights:
        for angle in angles:
            points.append(
                [base_center[0] + radius * np.cos(angle), base_center[1] + radius * np.sin(angle), base_center[2] + h])
    return np.array(points)


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
```

```python
    missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t
    for target_point in REAL_TARGET_POINTS:
        if not any(_is_single_line_blocked(missile_pos, center, target_point) for center in active_smoke_centers):
            return False
    return True

def find_transition_time(t_start, t_end, missile_id, start_state, grenade_lifecycles, precision):
    low, high = t_start, t_end
    while (high - low) > precision:
        mid = (low + high) / 2
        if get_obscuration_state_at_time(mid, missile_id, grenade_lifecycles) == start_state:
            low = mid
        else:
            high = mid
    return high

def print_grenade_trajectory_info(uav_flight_plans):
    print("--- 干扰弹投放与引爆坐标信息 ---")
    for uav_id, uav_data in uav_flight_plans.items():
        uav_speed = uav_data['uav_speed']
        direction_deg = uav_data['direction_deg']
        p_uav_initial = INITIAL_POSITIONS[uav_id]
        angle_rad = np.radians(direction_deg)
        v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed * np.sin(angle_rad), 0])

        for i, grenade_strategy in enumerate(uav_data.get('grenade_strategy', [])):
            grenade_id = f"{uav_id}_G{i + 1}"
            t_release = grenade_strategy['t_release']
            t_detonate_after = grenade_strategy['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]
            )

            print(f"干扰弹 {grenade_id}:")
            print(f"  - 投放坐标 (x, y, z): ({p_drop[0]:.2f}, {p_drop
                [1]:.2f}, {p_drop[2]:.2f})")
            print(f"  - 引爆坐标 (x, y, z): ({p_detonate[0]:.2f}, {
                p_detonate[1]:.2f}, {p_detonate[2]:.2f})")

def get_obscuration_intervals(uav_flight_plans, all_missile_ids,
    coarse_time_step, precision, max_sim_time=100.0):
    grenade_lifecycles = []
    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 gs in 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])
            grenade_lifecycles.append({'p_detonate': p_detonate, '
                t_detonate_abs': t_release + t_detonate_after})

    obscuration_intervals = {m_id: [] for m_id in all_missile_ids}
    for missile_id in all_missile_ids:
        last_time, last_state = 0.0, get_obscuration_state_at_time
            (0.0, missile_id, grenade_lifecycles)
        obscuration_start_time = 0.0 if last_state else None
        for i in range(1, int(max_sim_time / coarse_time_step) + 1):
            current_time = i * coarse_time_step
            current_state = get_obscuration_state_at_time(
                current_time, missile_id, grenade_lifecycles)
```

```python
            if current_state != last_state:
                transition_t = find_transition_time(last_time, current_time, missile_id, last_state, grenade_lifecycles,
                                                    precision)
                if current_state:
                    obscuration_start_time = transition_t
                elif obscuration_start_time is not None:
                    obscuration_intervals[missile_id].append((obscuration_start_time, transition_t))
                    obscuration_start_time = None
            last_time, last_state = current_time, current_state
        if obscuration_start_time is not None:
            obscuration_intervals[missile_id].append((obscuration_start_time, max_sim_time))
    return obscuration_intervals

def _get_contributing_grenades_at_time(t, missile_id, grenade_lifecycles_with_id):
    missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t
    active_grenades = []
    for grenade in grenade_lifecycles_with_id:
        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_grenades.append({'id': grenade['id'], 'center': current_center})
    if not active_grenades: return []
    for target_point in REAL_TARGET_POINTS:
        if not any(_is_single_line_blocked(missile_pos, g['center'], target_point) for g in active_grenades):
            return []
    contributing_grenades = []
    for g in active_grenades:
        for target_point in REAL_TARGET_POINTS:
            if _is_single_line_blocked(missile_pos, g['center'],
```

```python
               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_abs': g['t_detonate_abs']} for g in grenade_lifecycles]
    for missile_id in all_missile_ids:
        last_time, last_state = 0.0, get_obscuration_state_at_time(0.0, missile_id, plain_lifecycles)
        for i in range(1, int(max_sim_time / coarse_time_step) + 1):
            current_time = i * coarse_time_step
            current_state = get_obscuration_state_at_time(current_time, missile_id, plain_lifecycles)
            if current_state != last_state:
                event_times.add(
                    find_transition_time(last_time, current_time, missile_id, last_state, plain_lifecycles, precision))
            last_time, last_state = current_time, current_state

    unique_times = sorted([t for t in event_times if t <= max_sim_time])
    grenade_contributions = {g_id: {m_id: 0.0 for m_id in all_missile_ids} for g_id in grenade_ids}

    for i in range(len(unique_times) - 1):
        t_start, t_end = unique_times[i], unique_times[i + 1]
        interval_duration = t_end - t_start
        if interval_duration <= precision / 10: continue
        t_mid = (t_start + t_end) / 2.0
        for missile_id in all_missile_ids:
            for g_id in _get_contributing_grenades_at_time(t_mid, missile_id, grenade_lifecycles):
                grenade_contributions[g_id][missile_id] += interval_duration

    print("--- 干扰弹贡献分析结果 ---")
    final_results = {}
    for g_id, missile_times in grenade_contributions.items():
        total_time = sum(missile_times.values())
        if total_time > precision:
            final_results[g_id] = missile_times
            final_results[g_id]['total'] = total_time

```

```
    if not final_results:
        print("没有干扰弹产生有效遮蔽。")
    else:
        sorted_results = sorted(final_results.items(), key=lambda item: item[1]['total'], reverse=True)
        for g_id, data in sorted_results:
            total = data['total']
            details = ", ".join([f"{m_id}: {t:.2f}s" for m_id, t in data.items() if m_id != 'total' and t > 0])
            print(f"干扰弹 {g_id}: 总计干扰 {total:.4f}s ({details})")
    return final_results

def plot_timeline_chinese(obscuration_intervals, uav_flight_plans):
    try:
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False
    except Exception:
        print("设置中文字体失败，将使用默认字体。")

    missile_ids, uav_ids = sorted(obscuration_intervals.keys()), sorted(uav_flight_plans.keys())
    y_labels = [f"导弹 {m_id} 被遮蔽" for m_id in missile_ids] + [f"无人机 {u_id} 干扰弹" for u_id in uav_ids]
    y_pos = np.arange(len(y_labels))
    fig, ax = plt.subplots(figsize=(16, 8))

    missile_colors = plt.cm.viridis(np.linspace(0, 0.5, len(missile_ids)))
    for i, m_id in enumerate(missile_ids):
        ax.broken_barh([(s, e - s) for s, e in obscuration_intervals[m_id]], (y_pos[i] - 0.4, 0.8),
                       facecolors=missile_colors[i])

    uav_colors = plt.cm.plasma(np.linspace(0.5, 1, len(uav_ids)))
    last_event_time = 0.0
    for i, uav_id in enumerate(uav_ids):
        detonation_times = [gs['t_release'] + gs['t_detonate_after'] for gs in
                                uav_flight_plans[uav_id].get('
```

```
                                    grenade_strategy', [])]
        if detonation_times: last_event_time = max(last_event_time,
            max(detonation_times))
        ax.plot(detonation_times, [y_pos[len(missile_ids) + i]] * len
            (detonation_times), 'o', ms=8, color=uav_colors[i])

    for intervals in obscuration_intervals.values():
        if intervals: last_event_time = max(last_event_time, max(e
            for s, e in intervals))

    ax.set_yticks(y_pos)
    ax.set_yticklabels(y_labels)
    ax.set_xlabel("时间 (秒)"), ax.set_ylabel("事件"), ax.set_title("
        干扰弹引爆与导弹遮蔽时间轴")
    ax.grid(True, axis='x', linestyle='--', alpha=0.6)
    ax.set_xlim(0, last_event_time * 1.05), ax.set_ylim(-0.5, len(
        y_labels) - 0.5)

    handles = [mpatches.Patch(color=c, label=f'导弹 {m_id} 遮蔽时段')
        for i, (m_id, c) in
               enumerate(zip(missile_ids, missile_colors))]
    handles += [
        plt.Line2D([0], [0], marker='o', color='w', label=f'无人机 {
            u_id} 引爆', markerfacecolor=c, markersize=8) for
        i, (u_id, c) in enumerate(zip(uav_ids, uav_colors))]
    ax.legend(handles=handles, bbox_to_anchor=(1.02, 1), loc='upper
        left')
    plt.tight_layout()
    plt.show()

uav_flight_plans = {
    "FY1": {
        "direction_deg": 180,
        "uav_speed": 120,
        "grenade_strategy": [
            {"t_release": 1.5, "t_detonate_after": 3.6}
        ]
    },
    "FY2": {
        "direction_deg": 0.0,
```

```
            "uav_speed": 70.0,
            "grenade_strategy": []
        },
        "FY3": {
            "direction_deg": 0.0,
            "uav_speed": 70.0,
            "grenade_strategy": []
        },
        "FY4": {
            "direction_deg": 0.0,
            "uav_speed": 70.0,
            "grenade_strategy": []
        },
        "FY5": {
            "direction_deg": 0.0,
            "uav_speed": 70.0,
            "grenade_strategy": []
        }
}

all_missile_ids = ["M1", "M2", "M3"]
coarse_time_step = 0.001
precision = 0.00001
max_sim_time = 100.0

print_grenade_trajectory_info(uav_flight_plans)

print("\n--- 计算导弹总遮蔽时长 ---")
intervals = get_obscuration_intervals(uav_flight_plans,
    all_missile_ids, coarse_time_step, precision, max_sim_time)
for m_id, interval_list in intervals.items():
    total_time = sum(end - start for start, end in interval_list)
    print(f"导弹 {m_id} 的总有效遮蔽时长为: {total_time:.6f} 秒")

calculate_grenade_contributions(uav_flight_plans, coarse_time_step,
    precision, max_sim_time)

print("\n--- 生成时间轴图表 ---")
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）

```java

import java.util.*;

// ========= 向量 =========
class Vec {
    double x,y,z;
    Vec(double x,double y,double z){this.x=x;this.y=y;this.z=z;}
    Vec add(Vec b){return new Vec(x+b.x,y+b.y,z+b.z);}
    Vec sub(Vec b){return new Vec(x-b.x,y-b.y,z-b.z);}
    Vec mul(double k){return new Vec(x*k,y*k,z*k);}
    double dot(Vec b){return x*b.x + y*b.y + z*b.z;}
    double norm(){return Math.sqrt(dot(this));}
    Vec unit(){ double n=norm(); return new Vec(x/n,y/n,z/n); }
    static double dist(Vec a, Vec b){ return a.sub(b).norm(); }
    public String toString(){ return String.format("(%.3f, %.3f, %.3f
        )", x,y,z); }
}

public class Main {

    // ========= 场景常量 =========
    static final double g = 9.8;
    static final double rCloud = 10.0;
    static final double vSink = 3.0;
    static final double window = 20.0;      // 云团有效 20 s
    static final double vM = 300.0;         // 导弹速度
    static final Vec FALSE_T = new Vec(0,0,0);
```

```
    static final Vec M0 = new Vec(20000,0,2000);
    static final Vec F0 = new Vec(17800,0,1800);
    static final Vec uM = M0.mul(-1.0/M0.norm()); // 导弹朝原点
    static final double tMissileEnd = M0.norm()/vM; //   66.7 s

    // ========= 决策变量边界（关键修正）=========
    static final double VMIN=70, VMAX=140;
    static final double TDROP_MIN=0.5, FUSE_MIN=0.2; // 避免零边界
    static final double TDROP_MAX=12.0, FUSE_MAX=8.0; // 可按需放宽，但要满足 tDrop+fuse   tMissileEnd

    // ========= 目标 12 点（圆柱：R=7, H=10）=========
    static List<Vec> initTarget12(){
        List<Vec> pts = new ArrayList<>();
        double R=7.0, H=10.0;
        pts.add(new Vec(0,200,0));
        pts.add(new Vec(0,200,H));
        int n=5;
        for(int i=0;i<n;i++){
            double th=2*Math.PI*i/n;
            double dx=R*Math.cos(th), dy=R*Math.sin(th);
            pts.add(new Vec(dx,200+dy,0));
            pts.add(new Vec(dx,200+dy,H));
        }
        return pts;
    }
    static final List<Vec> TARGET = initTarget12();

    // ========= 运动学 =========
    static Vec missilePos(double t){ return M0.add(uM.mul(vM*t)); }
    static Vec uavPos(double t, double vF, double theta){
        Vec u = new Vec(Math.cos(theta), Math.sin(theta), 0).unit();
        return F0.add(u.mul(vF*t));
    }
    static Vec bombPos(double t, double tDrop, double vF, double theta){
        if(t < tDrop) return new Vec(Double.NaN,Double.NaN,Double.NaN);
        Vec u = new Vec(Math.cos(theta), Math.sin(theta), 0).unit();
        Vec Pdrop = uavPos(tDrop, vF, theta);
```

```java
        double dt = t - tDrop;
        return Pdrop.add(u.mul(vF*dt)).add(new Vec(0,0,-0.5*g*dt*dt))
            ;
    }
    static Vec cloudCenter(double t, double tDrop, double fuse,
        double vF, double theta){
        double tExp = tDrop + fuse;
        if(t < tExp) return new Vec(Double.NaN,Double.NaN,Double.NaN)
            ;
        Vec Cexp = bombPos(tExp, tDrop, vF, theta);
        return Cexp.add(new Vec(0,0,-vSink*(t - tExp)));
    }

    // ========= 线段-点距离 =========
    static double distSegPoint(Vec A, Vec B, Vec P){
        Vec AB = B.sub(A), AP = P.sub(A);
        double ab2 = AB.dot(AB);
        if(ab2==0) return Vec.dist(P,A);
        double s = AP.dot(AB)/ab2;
        s = Math.max(0, Math.min(1, s));
        Vec Q = A.add(AB.mul(s));
        return Vec.dist(P, Q);
    }

    // ========= g(t): 12 点同时遮蔽的判据 =========
    // g(t) = max_i( dist(segment[M(t)-T_i], C(t)) - rCloud )
    // g(t) <= 0 表示当下时刻全部被遮蔽
    static double g(double t, double tDrop, double fuse, double vF,
        double theta){
        double tExp = tDrop + fuse;
        if(t < tExp || t > tExp + window) return +1e9;
        Vec M = missilePos(t);
        Vec C = cloudCenter(t, tDrop, fuse, vF, theta);
        if(Double.isNaN(C.x)) return +1e9;
        double maxVal = -1e99;
        for(Vec Ti : TARGET){
            double d = distSegPoint(M, Ti, C);
            double val = d - rCloud;
            if(val > maxVal) maxVal = val;
        }
```

```java
        return maxVal;
    }

    // ========= 精确遮蔽区间（去掉“起爆点离目标远”的错误减枝） =========
    static List<double[]> coverIntervals(double tDrop, double fuse, double vF, double theta){
        List<double[]> segs = new ArrayList<>();
        if(vF<VMIN || vF>VMAX) return segs;
        if(tDrop<TDROP_MIN || fuse<FUSE_MIN) return segs;
        double tExp = tDrop + fuse;
        if(tExp > tMissileEnd) return segs;

        double t0 = tExp, t1 = Math.min(tMissileEnd, tExp + window);
        if(t1 <= t0) return segs;

        double dt = 0.01, tol = 1e-4;
        double prevT = t0, prevG = g(prevT, tDrop, fuse, vF, theta);
        boolean inside = (prevG <= 0);
        double entry = inside ? prevT : Double.NaN;

        for(double t = t0 + dt; t <= t1 + 1e-12; t += dt){
            double curG = g(t, tDrop, fuse, vF, theta);
            if( (prevG > 0 && curG <= 0) || (prevG < 0 && curG >= 0) ){
                double a = t - dt, b = t; // 二分定位交点
                double fa = g(a, tDrop, fuse, vF, theta), fb = curG;
                for(int it=0; it<90 && Math.abs(b-a)>tol; it++){
                    double m = 0.5*(a+b);
                    double fm = g(m, tDrop, fuse, vF, theta);
                    if(fa*fm <= 0){ b=m; fb=fm; } else { a=m; fa=fm; }
                }
                double tZero = 0.5*(a+b);
                if(prevG > 0 && curG <= 0){ entry = tZero; inside = true; }
                else if(prevG < 0 && curG >= 0){
                    if(inside) segs.add(new double[]{entry, tZero});
                    inside = false;
                }
```

```java
            }
            prevT = t; prevG = curG;
        }
        if(inside) segs.add(new double[]{entry, t1});
        return segs;
    }

    static double coverDuration(double tDrop, double fuse, double vF, double theta){
        List<double[]> segs = coverIntervals(tDrop, fuse, vF, theta);
        double sum = 0;
        for(double[] s : segs) sum += Math.max(0, s[1]-s[0]);
        return sum;
    }

    // ========= 目标函数（最大化） =========
    static double fitness(double vF, double theta, double tDrop, double fuse){
        if(vF<VMIN || vF>VMAX) return -1e9;
        if(tDrop<TDROP_MIN || fuse<FUSE_MIN) return -1e9;
        if(tDrop + fuse > tMissileEnd) return -1e9;
        theta = clampAng(theta);
        return coverDuration(tDrop, fuse, vF, theta);
    }

    // ========= Nelder-Mead（局部精修） =========
    static class NM {
        static class X { double v, th, td, fu; X(double v,double th,double td,double fu){this.v=v;this.th=th;this.td=td;this.fu=fu;} }
        static X opt(X x0){
            double dv=5, dth=Math.toRadians(12), dtd=0.6, dfu=0.6;
            List<X> S = new ArrayList<>();
            S.add(new X(x0.v, x0.th, x0.td, x0.fu));
            S.add(new X(clamp(x0.v+dv,VMIN,VMAX), x0.th, x0.td, x0.fu));
            S.add(new X(x0.v, clampAng(x0.th+dth), x0.td, x0.fu));
            S.add(new X(x0.v, x0.th, clampTD(x0.td+dtd), x0.fu));
            S.add(new X(x0.v, x0.th, x0.td, clampFU(x0.td, x0.fu+dfu)));
```

```java

            int MAX=140; double alpha=1, gamma=2, rho=0.5, sigma=0.5;
            for(int it=0; it<MAX; it++){
                S.sort((a,b)->Double.compare(fitness(b.v,b.th,b.td,b.fu), fitness(a.v,a.th,a.td,a.fu)));
                double spread = Math.abs(S.get(0).v-S.get(S.size()-1).v)
                        + Math.abs(S.get(0).th-S.get(S.size()-1).th)
                        + Math.abs(S.get(0).td-S.get(S.size()-1).td)
                        + Math.abs(S.get(0).fu-S.get(S.size()-1).fu);
                if(spread < 1e-3) break;

                X best=S.get(0), worst=S.get(S.size()-1);
                // 质心 (除最差点)
                double cv=0, cth=0, ctd=0, cfu=0;
                for(int i=0;i<S.size()-1;i++){ cv+=S.get(i).v; cth+=S.get(i).th; ctd+=S.get(i).td; cfu+=S.get(i).fu; }
                int m=S.size()-1; cv/=m; cth/=m; ctd/=m; cfu/=m;

                // 反射
                X xr = new X(
                        clamp(cv + alpha*(cv - worst.v), VMIN, VMAX),
                        clampAng(cth + alpha*(cth - worst.th)),
                        clampTD(ctd + alpha*(ctd - worst.td)),
                        clampFU(ctd + alpha*(ctd - worst.td), cfu + alpha*(cfu - worst.fu))
                );

                double fr = fitness(xr.v,xr.th,xr.td,xr.fu);
                double fBest = fitness(best.v,best.th,best.td,best.fu);
                double fWorst = fitness(worst.v,worst.th,worst.td,worst.fu);

                if(fr >= fBest){
                    // 扩张
                    X xe = new X(
                            clamp(cv + gamma*(cv - worst.v), VMIN, VMAX),
                            clampAng(cth + gamma*(cth - worst.th)),
```

```
                        clampTD(ctd + gamma*(ctd - worst.td)),
                        clampFU(ctd + gamma*(ctd - worst.td), cfu + gamma*(cfu - worst.fu))
                );
                double fe = fitness(xe.v,xe.th,xe.td,xe.fu);
                S.set(S.size()-1, (fe>fr? xe : xr));
            }else if(fr > fitness(S.get(S.size()-2).v,S.get(S.size()-2).th,S.get(S.size()-2).td,S.get(S.size()-2).fu)){
                S.set(S.size()-1, xr);
            }else{
                // 内缩
                X xc = new X(
                        clamp(cv + rho*(worst.v - cv), VMIN, VMAX),
                        clampAng(cth + rho*(worst.th - cth)),
                        clampTD(ctd + rho*(worst.td - ctd)),
                        clampFU(ctd + rho*(worst.td - ctd), cfu + rho*(worst.fu - cfu))
                );
                double fc = fitness(xc.v,xc.th,xc.td,xc.fu);
                if(fc > fWorst){
                    S.set(S.size()-1, xc);
                }else{
                    // 收缩
                    for(int i=1;i<S.size();i++){
                        X si=S.get(i);
                        X shr = new X(
                                clamp(best.v + sigma*(si.v-best.v), VMIN, VMAX),
                                clampAng(best.th + sigma*(si.th-best.th)),
                                clampTD(best.td + sigma*(si.td-best.td)),
                                clampFU(best.td + sigma*(si.td-best.td), best.fu + sigma*(si.fu-best.fu))
                        );
                        S.set(i, shr);
                    }
```

```
                    }
                }
            }
            S.sort((a,b)->Double.compare(fitness(b.v,b.th,b.td,b.fu), fitness(a.v,a.th,a.td,a.fu)));
            return S.get(0);
        }
    }

    // ========= 工具函数 =========
    static double clamp(double x,double lo,double hi){ return Math.max(lo, Math.min(hi, x)); }
    static double clampAng(double th){
        while(th<=-Math.PI) th+=2*Math.PI;
        while(th> Math.PI) th-=2*Math.PI;
        return th;
    }
    static double clampTD(double td){ return Math.max(TDROP_MIN, Math.min(TDROP_MAX, td)); }
    static double clampFU(double td, double fu){
        fu = Math.max(FUSE_MIN, Math.min(FUSE_MAX, fu));
        if(td + fu > tMissileEnd) fu = Math.max(FUSE_MIN, tMissileEnd - td);
        return fu;
    }

    // ========= 主程序 =========
    public static void main(String[] args){
        // 1) 轻量粗搜 (不做“起爆点到目标XY距离”的硬剪枝)
        List<NM.X> seeds = new ArrayList<>();
        PriorityQueue<double[]> pq = new PriorityQueue<>((a,b)->Double.compare(b[0], a[0])); // [fit,v,th,td,fu]
        for(double v=70; v<=140; v+=10){
            for(double th=-Math.PI; th<=Math.PI; th+=Math.toRadians(45)){
                for(double td=1.0; td<=10.0; td+=1.0){
                    for(double fu=0.5; fu<=6.0; fu+=0.5){
                        if(td+fu > tMissileEnd) continue;
                        double fit = fitness(v, th, td, fu);
                        pq.add(new double[]{fit, v, th, td, fu});
```

```
                    }
                }
            }
        }
        // 取前 K 个种子
        int K = 12;
        while(K-- > 0 && !pq.isEmpty()){
            double[] a = pq.poll();
            seeds.add(new NM.X(a[1], a[2], a[3], a[4]));
        }
        if(seeds.isEmpty()){
            seeds.add(new NM.X(120, Math.PI, 1.5, 3.6)); // 朝向原点的粗种子 (    )
        }

        // 2) 局部精修
        NM.X best = null; double bestFit = -1e18;
        for(NM.X s : seeds){
            NM.X loc = NM.opt(s);
            double f  = fitness(loc.v, loc.th, loc.td, loc.fu);
            if(f > bestFit){ best = loc; bestFit = f; }
        }

        // 3) 输出
        double vBest=best.v, thBest=best.th, tdBest=best.td, fuBest=best.fu;
        List<double[]> segs = coverIntervals(tdBest, fuBest, vBest, thBest);
        Vec Pd = uavPos(tdBest, vBest, thBest);
        Vec Cexp = cloudCenter(tdBest+fuBest, tdBest, fuBest, vBest, thBest);

        System.out.println("=== 问题2 优化结果 (12点 + Nelder-Mead, 修正版) ===");
        System.out.printf("速度 vUAV = %.2f m/s\n", vBest);
        System.out.printf("航向角 theta = %.2f 度\n", Math.toDegrees(thBest));
        System.out.printf("投放时刻 tDrop = %.3f s\n", tdBest);
        System.out.printf("引信延时 fuse  = %.3f s\n", fuBest);
        System.out.printf("起爆时刻 tExp  = %.3f s\n\n", tdBest+
```

```java
                fuBest);
        System.out.println("投放点 P_drop = " + Pd);
        System.out.println("起爆点 C_exp  = " + Cexp);

        double total=0;
        if(segs.isEmpty()){
            System.out.println("\n无有效遮蔽区间。");
        }else{
            System.out.println("\n遮蔽区间(s):");
            for(double[] s : segs){
                System.out.printf("  [%.6f , %.6f]  时长 = %.6f\n", s
                    [0], s[1], (s[1]-s[0]));
                total += (s[1]-s[0]);
            }
            System.out.printf("\n总遮蔽时长 = %.6f s\n", total);
        }
    }
}
```

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）

```python

import numpy as np
import random
import math
from tqdm import tqdm
import cma
import pandas as pd
# --- 常量 ---
G = 9.8
MISSILE_SPEED_MPS = 300
SMOKE_CLOUD_RADIUS_M = 10
SMOKE_EFFECTIVE_DURATION_S = 20
SMOKE_SINK_SPEED_MPS = 3

# --- 目标几何 ---
```

```
TARGET_BASE_CENTER = np.array([0, 200, 0])
TARGET_RADIUS = 7.0
TARGET_HEIGHT = 10.0

# --- 初始位置与方向 ---
INITIAL_POSITIONS = {
    "M1": np.array([20000, 0, 2000]), "M2": np.array([19000, 600, 2100]), "M3": np.array([18000, -600, 1900]),
    "FY1": np.array([17800, 0, 1800]), "FY2": np.array([12000, 1400, 1400]), "FY3": np.array([6000, -3000, 700]),
    "FY4": np.array([11000, 2000, 1800]), "FY5": np.array([13000, -2000, 1300]),
}
FAKE_TARGET_POS = np.array([0, 0, 0])
MISSILE_DIRECTIONS = {m_id: (FAKE_TARGET_POS - pos) / np.linalg.norm(FAKE_TARGET_POS - pos)
                      for m_id, pos in INITIAL_POSITIONS.items() if m_id.startswith("M")}


# --- 目标点集生成 ---
def generate_target_points(base_center, radius, height, num_rim_points=12, num_height_levels=2):
    points = []
    angles = np.linspace(0, 2 * np.pi, num_rim_points, endpoint=False)
    heights = np.linspace(0, height, num_height_levels)
    for h in heights:
        for angle in angles:
            points.append(
                [base_center[0] + radius * np.cos(angle), base_center[1] + radius * np.sin(angle), base_center[2] + h])
    return np.array(points)


REAL_TARGET_POINTS = generate_target_points(TARGET_BASE_CENTER, TARGET_RADIUS, TARGET_HEIGHT)


# --- 核心物理检查函数 (无变化) ---
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


# --- 新增: 独立的、在特定时刻t检查遮蔽状态的函数 ---
def get_obscuration_state_at_time(t, missile_id, grenade_lifecycles):
    """在精确的时刻t，判断目标对指定导弹是否完全被遮蔽。"""
    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

    missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t

    for target_point in REAL_TARGET_POINTS:
        is_point_blocked = any(
            _is_single_line_blocked(missile_pos, center, target_point) for center in active_smoke_centers)
        if not is_point_blocked:
            return False # 只要有一个点可见，就返回False
    return True # 所有点都被遮蔽


# --- 二分法查找状态转换的精确时间 ---
def find_transition_time(t_start, t_end, missile_id, start_state, grenade_lifecycles, precision):
    """使用二分法在[t_start,␣t_end]区间内查找状态变化的精确时刻。"""
    low, high = t_start, t_end
    while (high - low) > precision:
        mid = (low + high) / 2
        mid_state = get_obscuration_state_at_time(mid, missile_id, grenade_lifecycles)
        if mid_state == start_state:
            low = mid
        else:
            high = mid
    return high

```

```python

# --- 优化后的主仿真函数 ---
def calculate_synergistic_obscuration_adaptive(uav_flight_plans, all_missile_ids, coarse_time_step, precision,
                                               max_sim_time=100.0):
    """
    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。
    """
    # 步骤1: 预计算烟幕弹生命周期
    grenade_lifecycles = []
    for uav_id, uav_data in uav_flight_plans.items():
        uav_speed = uav_data['uav_speed']
        direction_deg = uav_data['direction_deg']
        p_uav_initial = INITIAL_POSITIONS[uav_id]
        angle_rad = np.radians(direction_deg)
        v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed * np.sin(angle_rad), 0])
        for grenade_strategy in uav_data.get('grenade_strategy', []):
            t_release, t_detonate_after = grenade_strategy['t_release'], grenade_strategy['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_lifecycles.append({'p_detonate': p_detonate, 't_detonate_abs': t_detonate_abs})

    # 步骤2: 对每个导弹进行事件驱动的仿真
    total_obscured_times = {m_id: 0.0 for m_id in all_missile_ids}
    for missile_id in all_missile_ids:
        last_time = 0.0
        last_state = get_obscuration_state_at_time(0.0, missile_id, grenade_lifecycles)
        obscuration_start_time = 0.0 if last_state else None

        num_coarse_steps = int(max_sim_time / coarse_time_step)
        for i in range(1, num_coarse_steps + 1):
            current_time = i * coarse_time_step
            current_state = get_obscuration_state_at_time(current_time, missile_id, grenade_lifecycles)

            if current_state != last_state:
                # 状态发生改变，使用二分法精确定位
                transition_t = find_transition_time(last_time, current_time,
```

```
                    missile_id, last_state, grenade_lifecycles,
                                                    precision)

                if current_state is True: # 从 False -> True
                    obscuration_start_time = transition_t
                else: # 从 True -> False
                    if obscuration_start_time is not None:
                        total_obscured_times[missile_id] += (transition_t - obscuration_start_time)
                        obscuration_start_time = None

            last_time = current_time
            last_state = current_state

        # 如果仿真结束时仍然处于遮蔽状态，计算最后一段时长
        if obscuration_start_time is not None:
            total_obscured_times[missile_id] += (max_sim_time - obscuration_start_time)

    return total_obscured_times


# --- 封装调用函数 ---
def calculate_time(uav_flight_plans, coarse_time_step=0.1, precision=0.001):
    """
    封装了对优化后仿真函数的调用。

    Args:
        uav_flight_plans (dict): 无人机飞行计划。
        coarse_time_step (float): 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。
        precision (float): 二分法查找的精度，即最终结果的时间分辨率。
    """
    final_results = calculate_synergistic_obscuration_adaptive(
        uav_flight_plans=uav_flight_plans,
        all_missile_ids=["M1", "M2", "M3"], # 默认计算所有导弹
        coarse_time_step=coarse_time_step,
        precision=precision
    )

    # print(f"输入了 {sum(len(p.get('grenade_strategy', [])) for p in uav_flight_plans.values())} 枚干扰弹的策略。")
    # for missile_id, time in final_results.items():
    #     print(f"导弹 {missile_id} 的总有效遮蔽时长为：{time:.4f} 秒")
    return final_results
```

```python

# ----------------------------------------
#           以上为物理模拟计算
# ----------------------------------------



def is_target_fully_obscured_at_time(t, missile_id, active_smoke_centers):
    if not active_smoke_centers: return False
    missile_pos = INITIAL_POSITIONS[missile_id] + MISSILE_DIRECTIONS[missile_id] * MISSILE_SPEED_MPS * t
    for target_point in REAL_TARGET_POINTS:
        if not any(_is_single_line_blocked(missile_pos, center, target_point) for center in active_smoke_centers):
            return False
    return True

def find_interception_details_for_grenade(uav_speed_vec, uav_initial_pos,
                                          t_release_min, t_release_max,
                                          missile_id="M1"):
    """
␣␣␣␣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。
␣␣␣␣返回␣(t_release,␣t_detonate_after)␣或␣None。
␣␣␣␣"""
    # 随机化搜索顺序以避免偏差
    t_release_candidates = list(np.linspace(t_release_min, t_release_max, 20))
    random.shuffle(t_release_candidates)

    for t_release in t_release_candidates:
        p_drop = uav_initial_pos + uav_speed_vec * t_release

        # 同样随机化引爆延迟的搜索
        t_detonate_candidates = list(np.linspace(0, 8.0, 41))
        random.shuffle(t_detonate_candidates)

        for t_detonate_after in t_detonate_candidates:
            t_intercept = t_release + t_detonate_after
            p_smoke = p_drop + np.array([uav_speed_vec[0] * t_detonate_after,
                                         uav_speed_vec[1] * t_detonate_after,
                                         -0.5 * G * t_detonate_after ** 2])

            # 使用高保真模型进行验证
            if is_target_fully_obscured_at_time(t_intercept, missile_id, [p_smoke]):
                return t_release, t_detonate_after
```

```
    return None


def generate_holistic_seeds(num_seeds):
    """
␣␣␣␣为问题2生成高质量的整体种子。
␣␣␣␣每个种子都确保FY1有一套有效的初始拦截策略。
␣␣␣␣"""
    print(f"正在为问题2生成␣{num_seeds}␣个高质量的整体协同种子...")
    seeds = []
    uav_ids = ["FY1"]

    with tqdm(total=num_seeds, desc="生成P4种子") as pbar:
        while len(seeds) < num_seeds:
            current_seed_parts = []
            success = True
            for uav_id in uav_ids:
                p_uav_initial = INITIAL_POSITIONS[uav_id]

                # 尝试最多50次为单个无人机寻找有效策略
                found_strategy_for_uav = False
                for _ in range(50):
                    direction_deg = random.uniform(0, 12)
                    uav_speed = random.uniform(70, 140)
                    angle_rad = np.radians(direction_deg)
                    v_uav = np.array([uav_speed * np.cos(angle_rad), uav_speed * np.sin(angle_rad), 0])

                    grenade_details = find_interception_details_for_grenade(v_uav, p_uav_initial, 1.0, 40.0)

                    if grenade_details:
                        t_release, t_detonate_after = grenade_details
                        current_seed_parts.extend([direction_deg, uav_speed, t_release, t_detonate_after])
                        found_strategy_for_uav = True
                        break # 成功找到，跳出对此无人机的尝试

                if not found_strategy_for_uav:
                    success = False
                    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

            if success:
                seeds.append(current_seed_parts)
```

```
            pbar.update(1)

    print("问题4的高质量种子生成完毕! ")
    return seeds


# --- 核心优化函数 (使用CMA-ES) - 可配置迭代次数版 ---
def optimize_with_cmaes(initial_solution, fitness_func, max_iterations=100):
    """
␣␣␣␣从给定的初始解出发，使用CMA-ES进行优化。

␣␣␣␣Args:
␣␣␣␣␣␣␣␣initial_solution␣(list):␣优化的起始点。
␣␣␣␣␣␣␣␣fitness_func␣(function):␣适应度函数。
␣␣␣␣␣␣␣␣max_iterations␣(int):␣CMA-ES的最大迭代次数。

␣␣␣␣Returns:
␣␣␣␣␣␣␣␣tuple:␣(best_solution,␣best_fitness)
␣␣␣␣"""

    def objective_function(chromosome):
        return -fitness_func(chromosome)

    initial_sigma = 0.1
    options = {
        'bounds': [list(b) for b in zip(*BOUNDS_LIST)],
        'maxiter': max_iterations, # 使用传入的参数
        'verbose': -9
    }

    result = cma.fmin2(objective_function, initial_solution, initial_sigma, options)

    best_solution = result[0]
    best_fitness = -result[1].result.fbest

    return best_solution, best_fitness

# --- 为问题2定义新的参数边界 ---
BOUNDS = {
    'direction_deg1': [0, 360], 'uav_speed1': [70, 140], 't_release1': [0, 50], 't_detonate1': [0, 10]
}
BOUNDS_LIST = list(BOUNDS.values())

```

```python

# --- 2. 为问题4创建高质量的整体种子生成器 ---



def decode_chromosome(chromosome):
    """
␣␣␣␣将12个元素的染色体解码为仿真函数所需的␣uav_flight_plans␣字典。
␣␣␣␣"""
    dir1, speed1, t1, d1 = chromosome

    plans = {
        "FY1": {"direction_deg": dir1, "uav_speed": speed1,
                "grenade_strategy": [{"t_release": t1, "t_detonate_after": d1}]},
        # 其他无人机不参与
        "FY2": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
        "FY3": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
        "FY4": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
        "FY5": {"direction_deg": 0, "uav_speed": 70, "grenade_strategy": []},
    }
    return plans


def fitness_wrapper(chromosome):
    plans = decode_chromosome(chromosome)
    results = calculate_time(plans, coarse_time_step=0.2, precision=0.01)
    return results.get("M1", 0.0)

def fitness_wrapper_fine(chromosome):
    plans = decode_chromosome(chromosome)
    results = calculate_time(plans, coarse_time_step=0.05, precision=0.001)
    return results.get("M1", 0.0)

def two_stage_optimization(
        num_screening_seeds=50,
        screening_iterations=30,
        num_champions=5,
        deep_dive_iterations=150
):
    """
␣␣␣␣使用两阶段优化策略解决问题3。

␣␣␣␣Args:
␣␣␣␣␣␣␣␣num_screening_seeds␣(int):␣第一阶段（勘探）要生成的种子总数。
␣␣␣␣␣␣␣␣screening_iterations␣(int):␣第一阶段对每个种子进行的简短优化迭代次数。
```

```
␣␣␣␣␣␣␣␣num_champions␣(int):␣从第一阶段选出进入深度优化的“冠军”种子数量。
␣␣␣␣␣␣␣␣deep_dive_iterations␣(int):␣第二阶段（开采）对冠军种子进行的深度优化迭代次数。
␣␣␣␣"""

    # --- STAGE 1: 广泛筛选（勘探）---
    print("=" * 20 + "␣STAGE␣1:␣广泛筛选␣" + "=" * 20)
    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...")

    # 生成大量高质量种子
    initial_seeds = generate_holistic_seeds(num_screening_seeds)
    if not initial_seeds:
        print("错误：未能生成任何有效的初始种子，无法继续。")
        return

    screening_results = []
    with tqdm(total=len(initial_seeds), desc="第一阶段筛选") as pbar:
        for seed in initial_seeds:
            # 进行简短、快速的优化
            solution, fitness = optimize_with_cmaes(seed, fitness_wrapper, max_iterations=screening_iterations)
            screening_results.append({'solution': solution, 'fitness': fitness})
            pbar.update(1)
            pbar.set_postfix(current_best=f"{fitness:.3f}")

    # 根据初步优化的结果进行排序
    screening_results.sort(key=lambda x: x['fitness'], reverse=True)

    print("\n第一阶段筛选完成！初步优化的最佳结果：")
    for i in range(min(num_champions, len(screening_results))):
        print(f"␣␣Top␣{i␣+␣1}:␣遮蔽时间:␣{screening_results[i]['fitness']:.4f}␣秒")

    # --- STAGE 2: 深度优化（开采）---
    print("\n" + "=" * 20 + "␣STAGE␣2:␣深度优化␣" + "=" * 20)
    print(f"将对Top␣{num_champions}␣的种子进行␣{deep_dive_iterations}␣次迭代的深度优化...")

    champion_seeds = [res['solution'] for res in screening_results[:num_champions]]

    best_solution_overall, best_fitness_overall = None, -1
    with tqdm(total=len(champion_seeds), desc="第二阶段优化") as pbar:
        for champion_seed in champion_seeds:
```

```
            # 进行完整的、深度的优化
            solution, fitness = optimize_with_cmaes(champion_seed, fitness_wrapper_fine, max_iterations=deep_dive_iterations)
            if fitness > best_fitness_overall:
                best_fitness_overall = fitness
                best_solution_overall = solution
            pbar.update(1)
            pbar.set_postfix(best_overall=f"{best_fitness_overall:.3f}")

    print("\n优化完成！")
    if best_solution_overall is None:
        print("错误：优化未能找到任何有效的非零解。")
        return

    print("对最优策略进行高精度验算...")
    best_plans = decode_chromosome(best_solution_overall)
    final_results = calculate_time(best_plans, coarse_time_step=0.01, precision=0.0001)
    final_time = final_results.get("M1", 0.0)

    print("\n" + "=" * 50)
    print("          问题4：最优投放策略 (CMA-ES)          ")
    print("=" * 50)
    print(f"找到的最大有效遮蔽时长: {final_time:.4f} 秒")

    # --- 格式化输出 ---
    results_data = []
    for uav_id in ["FY1"]:
        params = best_plans[uav_id]
        strat = params['grenade_strategy'][0]

        # 计算投放点和起爆点
        p_uav_initial = INITIAL_POSITIONS[uav_id]
        angle_rad = np.radians(params['direction_deg'])
        v_uav = np.array([params['uav_speed'] * np.cos(angle_rad), params['uav_speed'] * np.sin(angle_rad), 0])
        p_drop = p_uav_initial + v_uav * strat['t_release']
        p_detonate = p_drop + np.array([v_uav[0] * strat['t_detonate_after'],
                                        v_uav[1] * strat['t_detonate_after'],
                                        -0.5 * G * strat['t_detonate_after'] ** 2])

        print(f"\n策略详情: 无人机 {uav_id}")
        print(f"  飞行方向: {params['direction_deg']:.2f} 度")
        print(f"  飞行速度: {params['uav_speed']:.2f} m/s")
        print(f"  投放时间: 受领任务后 {strat['t_release']:.3f} 秒")
```

```
        print(f"␣␣引爆延迟:␣投放后␣{strat['t_detonate_after']:.3f}␣秒")
        print(f"␣␣投放点坐标␣(x,y,z):␣({p_drop[0]:.2f},␣{p_drop[1]:.2f},␣
            {p_drop[2]:.2f})")
        print(f"␣␣起爆点坐标␣(x,y,z):␣({p_detonate[0]:.2f},␣
            {p_detonate[1]:.2f},␣{p_detonate[2]:.2f})")

        results_data.append([
            uav_id, f"{params['direction_deg']:.4f}",
                f"{params['uav_speed']:.4f}",
            f"{p_drop[0]:.4f}", f"{p_drop[1]:.4f}", f"{p_drop[2]:.4f}",
            f"{p_detonate[0]:.4f}", f"{p_detonate[1]:.4f}",
                f"{p_detonate[2]:.4f}"
        ])

    print("=" * 50)

    return best_plans, final_time


if __name__ == '__main__':
    two_stage_optimization(
        num_screening_seeds=100, # 勘探不同的起点
        screening_iterations=30, # 每个起点快速迭代
        num_champions=3, # 选出最好的
        deep_dive_iterations=150 # 对最好的进行深度优化
    )
```

C 附录 C：问题三 C++ 实现

Listing 4: 问题三 C++ 实现代码

```
[language=C++,basicstyle=\ttfamily\footnotesize,breaklines=true,
numbers=left,numberstyle=\tiny,frame=single,caption={问题三 C++ 实现代码}]


#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include <cmath>
#include <random>
#include <algorithm>
#include <numeric>
```

```
#include <future>
#include <iomanip>
#include <optional>
#include <thread> // <<< FIX: Added for std::this_thread

// --- 依赖: EIGEN ---
#include <Eigen/Dense>

// <<< FIX: Removed unnecessary #pragma once from .cpp file >>>

// ============================================================
// SECTION 1: 工具 & 基础数据结构
// ============================================================

struct Vector3D {
    double x = 0.0, y = 0.0, z = 0.0;

    Vector3D operator+(const Vector3D &other) const { return {x + other.x, y +
        other.y, z + other.z}; }
    Vector3D operator-(const Vector3D &other) const { return {x - other.x, y -
        other.y, z - other.z}; }
    Vector3D operator*(double scalar) const { return {x * scalar, y * scalar,
        z * scalar}; }
    double dot(const Vector3D &other) const { return x * other.x + y * other.y
        + z * other.z; }
    double norm_sq() const { return x * x + y * y + z * z; }
    double norm() const { return std::sqrt(norm_sq()); }
};

inline Vector3D operator*(double scalar, const Vector3D &vec) { return vec *
    scalar; }

struct GrenadeStrategy { double t_release; double t_detonate_after; };
struct UAVPlan { double direction_deg; double uav_speed;
    std::vector<GrenadeStrategy> grenade_strategy; };
using FlightPlans = std::map<std::string, UAVPlan>;
struct GrenadeLifecycle { Vector3D p_detonate; double t_detonate_abs; };

// ============================================================
// SECTION 2: 常量 & 全局仿真数据
// ============================================================

namespace Constants {
    const double G = 9.8;
    const double PI = 3.14159265358979323846;
```

```
const double MISSILE_SPEED_MPS = 300;
const double SMOKE_CLOUD_RADIUS_M = 10.0;
const double SMOKE_EFFECTIVE_DURATION_S = 20.0;
const double SMOKE_SINK_SPEED_MPS = 3.0;

const Vector3D TARGET_BASE_CENTER = {0, 200, 0};
const double TARGET_RADIUS = 7.0;
const double TARGET_HEIGHT = 10.0;

const std::map<std::string, Vector3D> INITIAL_POSITIONS = {
      {"M1", {20000, 0,   2000}}, {"M2", {19000, 600, 2100}}, {"M3",
          {18000, -600, 1900}},
      {"FY1", {17800, 0,  1800}}, {"FY2", {12000, 1400, 1400}}, {"FY3",
          {6000, -3000, 700}},
      {"FY4", {11000, 2000, 1800}}, {"FY5", {13000, -2000, 1300}},
};

const Vector3D FAKE_TARGET_POS = {0, 0, 0};

std::map<std::string, Vector3D> initialize_directions() {
   std::map<std::string, Vector3D> dirs;
   for (const auto &pair: INITIAL_POSITIONS) {
      if (pair.first.rfind("M", 0) == 0) {
         Vector3D dir = FAKE_TARGET_POS - pair.second;
         dirs[pair.first] = dir * (1.0 / dir.norm());
      }
   }
   return dirs;
}
const std::map<std::string, Vector3D> MISSILE_DIRECTIONS =
    initialize_directions();

std::vector<Vector3D> generate_target_points() {
   std::vector<Vector3D> points;
   int num_rim_points = 12, num_height_levels = 3;
   for (int i = 0; i < num_height_levels; ++i) {
      double h = (num_height_levels > 1) ? (TARGET_HEIGHT * i /
          (num_height_levels - 1)) : 0;
      for (int j = 0; j < num_rim_points; ++j) {
         double angle = 2 * PI * j / num_rim_points;
         points.push_back({
                              TARGET_BASE_CENTER.x + TARGET_RADIUS *
                                  std::cos(angle),
                              TARGET_BASE_CENTER.y + TARGET_RADIUS *
                                  std::sin(angle),
```

```cpp
                                    TARGET_BASE_CENTER.z + h
                            });
            }
        }
        return points;
    }
    const std::vector<Vector3D> REAL_TARGET_POINTS = generate_target_points();
}

// ============================================================
// SECTION 3: 核心物理仿真
// ============================================================

bool is_target_fully_obscured_at_time(double t, const std::string &missile_id,
    const std::vector<GrenadeLifecycle> &all_grenades);

double calculate_time_for_m1(const FlightPlans& uav_flight_plans, double
    coarse_time_step = 0.1, double precision = 0.001, double max_sim_time =
    100.0) {
    std::vector<GrenadeLifecycle> grenade_lifecycles;
    for (const auto& [uav_id, uav_data] : uav_flight_plans) {
        if (uav_data.grenade_strategy.empty()) continue;
        double angle_rad = uav_data.direction_deg * Constants::PI / 180.0;
        Vector3D v_uav = {uav_data.uav_speed * std::cos(angle_rad),
            uav_data.uav_speed * std::sin(angle_rad), 0};
        for (const auto& grenade : uav_data.grenade_strategy) {
            Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) + v_uav *
                grenade.t_release;
            Vector3D p_detonate = p_drop + Vector3D{v_uav.x *
                grenade.t_detonate_after, v_uav.y * grenade.t_detonate_after,
                -0.5 * Constants::G * grenade.t_detonate_after *
                grenade.t_detonate_after};
            grenade_lifecycles.push_back({p_detonate, grenade.t_release +
                grenade.t_detonate_after});
        }
    }

    const std::string missile_id = "M1";
    double total_obscured_time = 0.0, last_time = 0.0;
    bool last_state = is_target_fully_obscured_at_time(0.0, missile_id,
        grenade_lifecycles);
    double obscuration_start_time = last_state ? 0.0 : -1.0;
    auto find_transition = [&](double t_start, double t_end, bool start_state)
        {
        double low = t_start, high = t_end;
```

```
        while ((high - low) > precision) {
            double mid = low + (high - low) / 2.0;
            if (is_target_fully_obscured_at_time(mid, missile_id, grenade_lifecycles) == start_state) low = mid;
            else high = mid;
        }
        return high;
    };
    int num_coarse_steps = static_cast<int>(max_sim_time / coarse_time_step);
    for (int i = 1; i <= num_coarse_steps; ++i) {
        double current_time = i * coarse_time_step;
        bool current_state = is_target_fully_obscured_at_time(current_time, missile_id, grenade_lifecycles);
        if (current_state != last_state) {
            double transition_t = find_transition(last_time, current_time, last_state);
            if (current_state) { obscuration_start_time = transition_t; }
            else if (obscuration_start_time >= 0) { total_obscured_time += (transition_t - obscuration_start_time); obscuration_start_time = -1.0; }
        }
        last_time = current_time; last_state = current_state;
    }
    if (obscuration_start_time >= 0) total_obscured_time += (max_sim_time - obscuration_start_time);

    return total_obscured_time;
}

bool _is_single_line_blocked(const Vector3D &missile_pos, const Vector3D &smoke_center, const Vector3D &target_point) {
    Vector3D vec_AB = target_point - missile_pos;
    Vector3D vec_AC = smoke_center - missile_pos;
    double vec_AB_norm_sq = vec_AB.norm_sq();
    if (vec_AB_norm_sq < 1e-9) return (smoke_center - missile_pos).norm_sq() <= Constants::SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
    double t_proj = vec_AC.dot(vec_AB) / vec_AB_norm_sq;
    Vector3D closest_point = missile_pos + std::clamp(t_proj, 0.0, 1.0) * vec_AB;
    return (smoke_center - closest_point).norm_sq() <= Constants::SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
}

bool is_target_fully_obscured_at_time(double t, const std::string &missile_id, const std::vector<GrenadeLifecycle> &all_grenades) {
```

```cpp
    std::vector<Vector3D> active_smoke_centers;
    for (const auto &grenade: all_grenades) {
        if (t >= grenade.t_detonate_abs && t < grenade.t_detonate_abs +
            Constants::SMOKE_EFFECTIVE_DURATION_S) {
            active_smoke_centers.push_back(grenade.p_detonate - Vector3D{0, 0,
                Constants::SMOKE_SINK_SPEED_MPS * (t -
                grenade.t_detonate_abs)});
        }
    }
    if (active_smoke_centers.empty()) return false;
    Vector3D missile_pos = Constants::INITIAL_POSITIONS.at(missile_id) +
        Constants::MISSILE_DIRECTIONS.at(missile_id) *
        (Constants::MISSILE_SPEED_MPS * t);
    for (const auto &target_point: Constants::REAL_TARGET_POINTS) {
        bool point_blocked = false;
        for (const auto &center: active_smoke_centers) {
            if (_is_single_line_blocked(missile_pos, center, target_point)) {
                point_blocked = true; break; }
        }
        if (!point_blocked) return false;
    }
    return true;
}

// =============================================================
// SECTION 4: 整体协同种子生成器
// =============================================================
namespace Seeder {
    std::mt19937 &get_rng() {
        static thread_local std::mt19937 rng(std::random_device{}() +
            std::hash<std::thread::id>{}(std::this_thread::get_id()));
        return rng;
    }
    double uniform(double min, double max) { return
        std::uniform_real_distribution<double>(min, max)(get_rng()); }

    // <<< NEW: Helper function to generate a SINGLE seed >>>
    Eigen::VectorXd generate_single_holistic_seed_p3() {
        const Vector3D p_fy1_initial = Constants::INITIAL_POSITIONS.at("FY1");
        auto find_interception_details = [](const Vector3D &v_uav, const
            Vector3D &p_uav_initial, double t_min, double t_max) ->
            std::optional<GrenadeStrategy> {
            for (int i = 0; i < 20; ++i) {
                double t_release = uniform(t_min, t_max);
                Vector3D p_drop = p_uav_initial + v_uav * t_release;
```

```cpp
            for (int j = 0; j < 40; ++j) {
                double t_detonate_after = uniform(0.1, 8.0);
                double t_intercept = t_release + t_detonate_after;
                Vector3D p_smoke = p_drop + Vector3D{v_uav.x * t_detonate_after, v_uav.y * t_detonate_after, -0.5 * Constants::G * t_detonate_after * t_detonate_after};
                if (is_target_fully_obscured_at_time(t_intercept, "M1", {{p_smoke, t_intercept}})) {
                    return GrenadeStrategy{t_release, t_detonate_after};
                }
            }
        }
        return std::nullopt;
    };

    while (true) {
        double dir = uniform(179, 180), speed = uniform(70, 140);
        Vector3D v_uav = {speed * std::cos(dir * Constants::PI / 180.0), speed * std::sin(dir * Constants::PI / 180.0), 0};

        auto g1_opt = find_interception_details(v_uav, p_fy1_initial, 0.0, 5.0);
        if (!g1_opt) continue;

        auto g2_opt = find_interception_details(v_uav, p_fy1_initial, g1_opt->t_release + 1.0, g1_opt->t_release + 5.0);
        if (!g2_opt) continue;

        auto g3_opt = find_interception_details(v_uav, p_fy1_initial, g2_opt->t_release + 1.0, g2_opt->t_release + 5.0);
        if (!g3_opt) continue;

        double g2 = g2_opt->t_release - g1_opt->t_release - 1.0;
        double g3 = g3_opt->t_release - g2_opt->t_release - 1.0;
        if (g2 < 0 || g3 < 0) continue;

        return (Eigen::VectorXd(8) << dir, speed, g1_opt->t_release, g1_opt->t_detonate_after, g2, g2_opt->t_detonate_after, g3, g3_opt->t_detonate_after).finished();
    }
}

// <<< MODIFICATION: Main seeder function now uses a scalable thread pool >>>
std::vector<Eigen::VectorXd> generate_holistic_seeds_p3(int num_seeds) {
```

```
    std::cout << "Generating␣" << num_seeds << "␣holistic␣seeds␣for␣Problem␣
        3␣(Parallel)..." << std::endl;

    unsigned int num_workers = std::thread::hardware_concurrency();
    if (num_workers == 0) num_workers = 4;
    std::cout << "Using␣" << num_workers << "␣worker␣threads␣for␣
        generation." << std::endl;

    std::vector<std::future<Eigen::VectorXd>> futures;
    futures.reserve(num_workers);
    std::vector<Eigen::VectorXd> seeds;
    seeds.reserve(num_seeds);

    int tasks_launched = 0;
    int tasks_completed = 0;

    while (tasks_completed < num_seeds) {
        while (futures.size() < num_workers && tasks_launched < num_seeds) {
            futures.push_back(std::async(std::launch::async,
                generate_single_holistic_seed_p3));
            tasks_launched++;
        }

        for (auto it = futures.begin(); it != futures.end(); ) {
            if (it->wait_for(std::chrono::seconds(0)) ==
                std::future_status::ready) {
                seeds.push_back(it->get());
                tasks_completed++;
                it = futures.erase(it);
            } else {
                ++it;
            }
        }

        int bar_width = 50;
        float progress = (float)tasks_completed / num_seeds;
        int pos = bar_width * progress;
        std::cout << "\r␣␣Generating␣seeds:␣[";
        for (int k = 0; k < bar_width; ++k) {
            if (k < pos) std::cout << "=";
            else if (k == pos) std::cout << ">";
            else std::cout << "␣";
        }
        std::cout << "]␣" << tasks_completed << "/" << num_seeds << "␣(" <<
            std::fixed << std::setprecision(1) << progress * 100.0 << "%)"
```

```
                << std::flush;

            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }

        std::cout << "\nSeed␣generation␣complete." << std::endl;
        return seeds;
    }
}

// ============================================================
// SECTION 5: 自包含的 CMA-ES 优化器
// ============================================================

class SimpleCMAES {
public:
    struct Solution { Eigen::VectorXd x; double fitness; };

    SimpleCMAES(int dim, int population_size = 0) : N(dim) {
        lambda = population_size > 0 ? population_size : 4 + floor(3 * log(N));
        mu = lambda / 2 > 0 ? lambda / 2 : 1;
        weights = Eigen::VectorXd::LinSpaced(mu, log(mu + 0.5), log(0.5)).array().exp();
        weights /= weights.sum();
        mu_eff = 1 / weights.squaredNorm();

        cc = (4 + mu_eff / N) / (N + 4 + 2 * mu_eff / N);
        cs = (mu_eff + 2) / (N + mu_eff + 5);
        c1 = 2 / (pow(N + 1.3, 2) + mu_eff);
        cmu = std::min(1 - c1, 2 * (mu_eff - 2 + 1 / mu_eff) / (pow(N + 2, 2) + mu_eff));
        damps = 1 + 2 * std::max(0.0, sqrt((mu_eff - 1) / (N + 1)) - 1) + cs;
        chiN = sqrt(N) * (1 - 1 / (4.0 * N) + 1 / (21.0 * N * N));
    }

    void optimize(const std::function<double(const Eigen::VectorXd &)> &fitness_func, Eigen::VectorXd &x_start, double sigma_start, int max_iter) {
        Eigen::VectorXd x_mean = x_start;
        double sigma = sigma_start;
        Eigen::MatrixXd C = Eigen::MatrixXd::Identity(N, N);
        Eigen::VectorXd pc = Eigen::VectorXd::Zero(N), ps = Eigen::VectorXd::Zero(N);

        // <<< FIX: Track the best solution found across all generations >>>
```

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

    for (int gen = 0; gen < max_iter; ++gen) {
        Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(C);
        Eigen::MatrixXd B = es.eigenvectors();
        Eigen::VectorXd D_vec = es.eigenvalues();
        for(int i=0; i<N; ++i) D_vec(i) = std::sqrt(std::max(1e-20,
            D_vec(i)));

        std::vector<Solution> population(lambda);
        for (int i = 0; i < lambda; ++i) {
            Eigen::VectorXd z = Eigen::VectorXd::NullaryExpr(N, [&]() {
                return
                std::normal_distribution<double>{}(Seeder::get_rng()); });
            population[i].x = x_mean + sigma * B * D_vec.asDiagonal() * z;
            population[i].fitness = fitness_func(population[i].x);
        }

        std::sort(population.begin(), population.end(), [](const auto &a,
            const auto &b) { return a.fitness > b.fitness; });

        if (population[0].fitness > best_solution_so_far.fitness) {
            best_solution_so_far = population[0];
        }

        Eigen::VectorXd x_old = x_mean;
        x_mean.setZero();
        for (int i = 0; i < mu; ++i) x_mean += weights(i) * population[i].x;

        ps = (1 - cs) * ps + sqrt(cs * (2 - cs) * mu_eff) * (B *
            D_vec.asDiagonal().inverse() * B.transpose() * (x_mean - x_old)
            / sigma);
        sigma *= exp((cs / damps) * (ps.norm() / chiN - 1));

        bool hsig = ps.norm() / sqrt(1 - pow(1 - cs, 2 * (gen + 1))) < chiN
            * (1.4 + 2.0 / (N + 1));
        pc = (1 - cc) * pc;
        if (hsig) pc += sqrt(cc * (2 - cc) * mu_eff) * (x_mean - x_old) /
            sigma;

        Eigen::MatrixXd rank_mu_update = Eigen::MatrixXd::Zero(N, N);
        for (int i = 0; i < mu; ++i) {
            Eigen::VectorXd diff = (population[i].x - x_old) / sigma;
            rank_mu_update += weights(i) * diff * diff.transpose();
        }
```

```
            C = (1 - c1 - cmu) * C + c1 * (pc * pc.transpose() + (1 - hsig) * cc * (2 - cc) * C) + cmu * rank_mu_update;
        }
        x_start = best_solution_so_far.x;
    }

private:
    int N, lambda, mu;
    double mu_eff, cc, cs, c1, cmu, damps, chiN;
    Eigen::VectorXd weights;
};

// ============================================================
// SECTION 6: 主要的两阶段优化逻辑
// ============================================================

struct OptimizationResult { Eigen::VectorXd solution; double fitness; };

FlightPlans decode_chromosome_p3(const Eigen::VectorXd &chromosome) {
    double dir = chromosome(0), speed = chromosome(1), t1 = chromosome(2), d1 = chromosome(3),
            g2 = chromosome(4), d2 = chromosome(5), g3 = chromosome(6), d3 = chromosome(7);
    return {
            {"FY1", {dir, speed, {{t1, d1}, {t1 + 1.0 + g2, d2}, {t1 + 1.0 + g2 + 1.0 + g3, d3}}}},
            {"FY2", {0, 70,   {}}}, {"FY3", {0, 70,  {}}},
            {"FY4", {0, 70,   {}}}, {"FY5", {0, 70,  {}}}
    };
}

void solve_problem3_two_stage(int num_screening_seeds, int screening_iterations, int num_champions, int deep_dive_iterations) {
    // --- STAGE 1: 广泛筛选 (并行) ---
    std::cout << "\n================␣STAGE␣1:␣Broad␣Screening␣(Parallel)␣================" << std::endl;
    auto initial_seeds = Seeder::generate_holistic_seeds_p3(num_screening_seeds);

    std::vector<std::future<OptimizationResult>> screening_futures;
    for (const auto &seed: initial_seeds) {
        screening_futures.push_back(std::async(std::launch::async, [=] {
            SimpleCMAES cma(8);
            Eigen::VectorXd current_sol = seed;
            auto fitness_wrapper = [](const Eigen::VectorXd &x) {
```

```
            double speed = x(1), t1 = x(2), d1 = x(3), g2 = x(4), d2 = x(5),
                g3 = x(6), d3 = x(7);
            // <<< REFINEMENT: Corrected constraint check >>>
            if (speed < 70.0 || speed > 140.0 || t1 < 0.0 || d1 < 0.0 || g2
                < 0.0 || d2 < 0.0 || g3 < 0.0 || d3 < 0.0) {
                return 0.0;
            }
            return calculate_time_for_m1(decode_chromosome_p3(x), 0.2, 0.01);
        };
        cma.optimize(fitness_wrapper, current_sol, 0.1,
            screening_iterations);
        return OptimizationResult{current_sol,
            fitness_wrapper(current_sol)};
    }));
}

std::vector<OptimizationResult> screening_results;
screening_results.reserve(screening_futures.size());
for(int i = 0; i < num_screening_seeds; ++i) {
    screening_results.push_back(screening_futures[i].get());
    int bar_width = 50;
    float progress = (float)(i + 1) / 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 << "] " << i + 1 << "/" << num_screening_seeds << " (" <<
        std::fixed << std::setprecision(1) << progress * 100.0 << "%)" <<
        std::flush;
}
std::cout << std::endl;

std::sort(screening_results.begin(), screening_results.end(), [](const
    auto &a, const auto &b) { return a.fitness > b.fitness; });

std::cout << "\nStage 1 Screening Complete! Top preliminary results:" <<
    std::endl;
for (int i = 0; i < std::min(num_champions, (int)
    screening_results.size()); ++i) {
    std::cout << "  Top " << i + 1 << ": Obscuration Time: " << std::fixed
        << std::setprecision(4) << screening_results[i].fitness << " s" <<
        std::endl;
```

```cpp
    }

    // --- STAGE 2: 深度优化 ---
    std::cout << "\n================ STAGE 2: Deep Dive Optimization ================" << std::endl;
    OptimizationResult best_overall = {{}, -1.0};

    for (int i = 0; i < std::min(num_champions, (int) screening_results.size()); ++i) {
        std::cout << "  Deep diving on champion #" << i + 1 << "..." << std::endl;
        SimpleCMAES cma_deep(8);
        Eigen::VectorXd champion_sol = screening_results[i].solution;
        auto fitness_wrapper_fine = [](const Eigen::VectorXd &x) {
            double speed = x(1), t1 = x(2), d1 = x(3), g2 = x(4), d2 = x(5), g3 = x(6), d3 = x(7);
            // <<< REFINEMENT: Corrected constraint check >>>
            if (speed < 70.0 || speed > 140.0 || t1 < 0.0 || d1 < 0.0 || g2 < 0.0 || d2 < 0.0 || g3 < 0.0 || d3 < 0.0) {
                return 0.0;
            }
            return calculate_time_for_m1(decode_chromosome_p3(x), 0.05, 0.001);
        };
        cma_deep.optimize(fitness_wrapper_fine, champion_sol, 0.05, deep_dive_iterations);
        double final_fitness = fitness_wrapper_fine(champion_sol);
        if (final_fitness > best_overall.fitness) {
            best_overall = {champion_sol, final_fitness};
        }
    }

    // --- 最终结果 ---
    std::cout << "\nOptimization complete! Performing final high-precision validation..." << std::endl;
    auto best_plans = decode_chromosome_p3(best_overall.solution);
    double final_time = calculate_time_for_m1(best_plans, 0.001, 0.0001);

    std::cout << "Final results calculated. Writing to output_p3.log..." << std::endl;
    std::ofstream out("output_p3.log");
    auto* cout_buf = std::cout.rdbuf();
    std::cout.rdbuf(out.rdbuf());

    std::cout << "\n====================== FINAL OPTIMAL STRATEGY (Problem 3) ======================" << std::endl;
```

```cpp
    std::cout << "Max Effective Obscuration Time for M1: " << std::fixed << std::setprecision(4) << final_time << " s" << std::endl;

    std::cout << "\nStrategy Details:" << std::endl;
    const auto &best_params = best_plans.at("FY1");
    std::cout << "  UAV FY1 Flight Direction: " << std::fixed << std::setprecision(2) << best_params.direction_deg << " deg" << std::endl;
    std::cout << "  UAV FY1 Flight Speed: " << best_params.uav_speed << " m/s" << std::endl;

    std::cout << "\n  Grenade Deployment Sequence:" << std::endl;
    int idx = 1;
    for (const auto &g: best_params.grenade_strategy) {
        std::cout << "    - Grenade #" << idx++ << ":" << std::endl;
        std::cout << "      Release at " << std::fixed << std::setprecision(3) << g.t_release << "s" << std::endl;
        std::cout << "      Detonate after " << g.t_detonate_after << "s" << std::endl;
    }
    std::cout << "========================================================" << std::endl;

    std::cout.rdbuf(cout_buf);
    std::cout << "Done. Results saved to output_p3.log" << std::endl;
}

int main() {
    // --- 在这里调整你的参数 ---
    int num_screening_seeds = 100000; // 勘探的种子总数
    int screening_iterations = 100; // 每个种子的快速迭代次数
    int num_champions = 30;      // 选出的冠军数量
    int deep_dive_iterations = 1000; // 对冠军的深度迭代次数

    solve_problem3_two_stage(
            num_screening_seeds,
            screening_iterations,
            num_champions,
            deep_dive_iterations
    );

    return 0;
}
```

D 附录 D：问题四 C++ 实现

Listing 5: 问题四 C++ 实现代码

```
# 这里放你完整的 Python 代码 (two_stage_optimization 那一份)

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <map>
#include <cmath>
#include <random>
#include <algorithm>
#include <numeric>
#include <future>
#include <iomanip>
#include <optional>
#include <chrono>
#include <thread>

// --- 依赖: EIGEN ---
#include <Eigen/Dense>

// ============================================================
// SECTION 1: 工具 & 基础数据结构
// ============================================================

struct Vector3D {
    double x = 0.0, y = 0.0, z = 0.0;

    Vector3D operator+(const Vector3D& other) const { return {x + other.x,
        y + other.y, z + other.z}; }
    Vector3D operator-(const Vector3D& other) const { return {x - other.x,
        y - other.y, z - other.z}; }
    Vector3D operator*(double scalar) const { return {x * scalar, y *
        scalar, z * scalar}; }
    double dot(const Vector3D& other) const { return x * other.x + y *
        other.y + z * other.z; }
    double norm_sq() const { return x * x + y * y + z * z; }
    double norm() const { return std::sqrt(norm_sq()); }
};

inline Vector3D operator*(double scalar, const Vector3D& vec) { return vec
    * scalar; }
```

```cpp

struct GrenadeStrategy { double t_release; double t_detonate_after; };
struct UAVPlan { double direction_deg; double uav_speed; std::vector<
    GrenadeStrategy> grenade_strategy; };
using FlightPlans = std::map<std::string, UAVPlan>;
struct GrenadeLifecycle { Vector3D p_detonate; double t_detonate_abs; };

// ============================================================
// SECTION 2: 常量 & 全局仿真数据
// ============================================================

namespace Constants {
    const double G = 9.8;
    const double PI = 3.14159265358979323846;
    const double MISSILE_SPEED_MPS = 300;
    const double SMOKE_CLOUD_RADIUS_M = 10.0;
    const double SMOKE_EFFECTIVE_DURATION_S = 20.0;
    const double SMOKE_SINK_SPEED_MPS = 3.0;

    const Vector3D TARGET_BASE_CENTER = {0, 200, 0};
    const double TARGET_RADIUS = 7.0;
    const double TARGET_HEIGHT = 10.0;

    const std::map<std::string, Vector3D> INITIAL_POSITIONS = {
            {"M1", {20000, 0, 2000}}, {"M2", {19000, 600, 2100}}, {"M3",
                {18000, -600, 1900}},
            {"FY1", {17800, 0, 1800}}, {"FY2", {12000, 1400, 1400}}, {"FY3"
                , {6000, -3000, 700}},
            {"FY4", {11000, 2000, 1800}}, {"FY5", {13000, -2000, 1300}},
    };

    const Vector3D FAKE_TARGET_POS = {0, 0, 0};

    std::map<std::string, Vector3D> initialize_directions() {
        std::map<std::string, Vector3D> dirs;
        for (const auto& pair : INITIAL_POSITIONS) {
            if (pair.first.rfind("M", 0) == 0) {
                Vector3D dir = FAKE_TARGET_POS - pair.second;
                dirs[pair.first] = dir * (1.0 / dir.norm());
            }
        }
        return dirs;
    }
    const std::map<std::string, Vector3D> MISSILE_DIRECTIONS =
        initialize_directions();
```

```

    std::vector<Vector3D> generate_target_points() {
        std::vector<Vector3D> points;
        int num_rim_points = 12, num_height_levels = 2;
        for (int i = 0; i < num_height_levels; ++i) {
            double h = (num_height_levels > 1) ? (TARGET_HEIGHT * i / (
               num_height_levels - 1)) : 0;
            for (int j = 0; j < num_rim_points; ++j) {
                double angle = 2 * PI * j / num_rim_points;
                points.push_back({
                                          TARGET_BASE_CENTER.x +
                                             TARGET_RADIUS * std::cos(angle)
                                             ,
                                          TARGET_BASE_CENTER.y +
                                             TARGET_RADIUS * std::sin(angle)
                                             ,
                                          TARGET_BASE_CENTER.z + h
                                  });
            }
        }
        return points;
    }
    const std::vector<Vector3D> REAL_TARGET_POINTS = generate_target_points
       ();
}

// ============================================================
// SECTION 3: 核心物理仿真
// ============================================================

bool _is_single_line_blocked(const Vector3D& missile_pos, const Vector3D&
   smoke_center, const Vector3D& target_point) {
    Vector3D vec_AB = target_point - missile_pos;
    Vector3D vec_AC = smoke_center - missile_pos;
    double vec_AB_norm_sq = vec_AB.norm_sq();
    if (vec_AB_norm_sq < 1e-9) return (smoke_center - missile_pos).norm_sq
       () <= Constants::SMOKE_CLOUD_RADIUS_M * Constants::
       SMOKE_CLOUD_RADIUS_M;
    double t_proj = vec_AC.dot(vec_AB) / vec_AB_norm_sq;
    Vector3D closest_point = missile_pos + std::clamp(t_proj, 0.0, 1.0) *
       vec_AB;
    return (smoke_center - closest_point).norm_sq() <= Constants::
       SMOKE_CLOUD_RADIUS_M * Constants::SMOKE_CLOUD_RADIUS_M;
}

```

```cpp
bool is_target_fully_obscured_at_time(double t, const std::string& missile_id, const std::vector<GrenadeLifecycle>& all_grenades) {
    std::vector<Vector3D> active_smoke_centers;
    for (const auto& grenade : all_grenades) {
        if (t >= grenade.t_detonate_abs && t < grenade.t_detonate_abs + Constants::SMOKE_EFFECTIVE_DURATION_S) {
            active_smoke_centers.push_back(grenade.p_detonate - Vector3D{0, 0, Constants::SMOKE_SINK_SPEED_MPS * (t - grenade.t_detonate_abs)});
        }
    }
    if (active_smoke_centers.empty()) return false;
    Vector3D missile_pos = Constants::INITIAL_POSITIONS.at(missile_id) + Constants::MISSILE_DIRECTIONS.at(missile_id) * (Constants::MISSILE_SPEED_MPS * t);
    for (const auto& target_point : Constants::REAL_TARGET_POINTS) {
        bool point_blocked = false;
        for (const auto& center : active_smoke_centers) {
            if (_is_single_line_blocked(missile_pos, center, target_point)) { point_blocked = true; break; }
        }
        if (!point_blocked) return false;
    }
    return true;
}

double calculate_time_for_m1(const FlightPlans& uav_flight_plans, double coarse_time_step = 0.1, double precision = 0.001, double max_sim_time = 100.0) {
    std::vector<GrenadeLifecycle> grenade_lifecycles;
    for (const auto& [uav_id, uav_data] : uav_flight_plans) {
        if (uav_data.grenade_strategy.empty()) continue;
        double angle_rad = uav_data.direction_deg * Constants::PI / 180.0;
        Vector3D v_uav = {uav_data.uav_speed * std::cos(angle_rad), uav_data.uav_speed * std::sin(angle_rad), 0};
        for (const auto& grenade : uav_data.grenade_strategy) {
            Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) + v_uav * grenade.t_release;
            Vector3D p_detonate = p_drop + Vector3D{v_uav.x * grenade.t_detonate_after, v_uav.y * grenade.t_detonate_after, -0.5 * Constants::G * grenade.t_detonate_after * grenade.t_detonate_after};
            grenade_lifecycles.push_back({p_detonate, grenade.t_release + grenade.t_detonate_after});
        }
```

```
    }

    const std::string missile_id = "M1";
    double total_obscured_time = 0.0, last_time = 0.0;
    bool last_state = is_target_fully_obscured_at_time(0.0, missile_id,
        grenade_lifecycles);
    double obscuration_start_time = last_state ? 0.0 : -1.0;
    auto find_transition = [&](double t_start, double t_end, bool
        start_state) {
        double low = t_start, high = t_end;
        while ((high - low) > precision) {
            double mid = low + (high - low) / 2.0;
            if (is_target_fully_obscured_at_time(mid, missile_id,
                grenade_lifecycles) == start_state) low = mid;
            else high = mid;
        }
        return high;
    };
    int num_coarse_steps = static_cast<int>(max_sim_time / coarse_time_step
        );
    for (int i = 1; i <= num_coarse_steps; ++i) {
        double current_time = i * coarse_time_step;
        bool current_state = is_target_fully_obscured_at_time(current_time,
             missile_id, grenade_lifecycles);
        if (current_state != last_state) {
            double transition_t = find_transition(last_time, current_time,
                last_state);
            if (current_state) { obscuration_start_time = transition_t; }
            else if (obscuration_start_time >= 0) { total_obscured_time +=
                (transition_t - obscuration_start_time);
                obscuration_start_time = -1.0; }
        }
        last_time = current_time; last_state = current_state;
    }
    if (obscuration_start_time >= 0) total_obscured_time += (max_sim_time -
         obscuration_start_time);

    return total_obscured_time;
}

// ============================================================
// SECTION 4: 启发式种子生成器 (问题四专用)
// ============================================================

namespace Seeder {
```

```
    std::mt19937& get_rng() {
        static thread_local std::mt19937 rng(std::random_device{}() + std::
            hash<std::thread::id>{}(std::this_thread::get_id()));
        return rng;
    }
    double uniform(double min, double max) { return std::
        uniform_real_distribution<double>(min, max)(get_rng()); }

    Eigen::VectorXd generate_single_seed_p4() {
        const std::vector<std::string> uav_ids = {"FY1", "FY2", "FY3"};
        auto find_interception_details = [](const Vector3D& v_uav, const
            Vector3D& p_uav_initial) -> std::optional<GrenadeStrategy> {
            for (int i = 0; i < 20; ++i) {
                double t_release = uniform(1.0, 40.0);
                Vector3D p_drop = p_uav_initial + v_uav * t_release;
                for (int j = 0; j < 40; ++j) {
                    double t_detonate_after = uniform(0.1, 8.0);
                    double t_intercept = t_release + t_detonate_after;
                    Vector3D p_smoke = p_drop + Vector3D{v_uav.x *
                        t_detonate_after, v_uav.y * t_detonate_after, -0.5 *
                         Constants::G * t_detonate_after * t_detonate_after
                        };
                    if (is_target_fully_obscured_at_time(t_intercept, "M1",
                         {{p_smoke, t_intercept}})) {
                        return GrenadeStrategy{t_release, t_detonate_after
                            };
                    }
                }
            }
            return std::nullopt;
        };

        while(true) {
            Eigen::VectorXd current_seed(12);
            bool success = true;
            for(size_t i = 0; i < uav_ids.size(); ++i) {
                const auto& uav_id = uav_ids[i];
                bool found_strategy_for_uav = false;
                for (int attempt = 0; attempt < 50; ++attempt) {
                    double dir = uniform(0, 360), speed = uniform(70, 140);
                    Vector3D v_uav = {speed * std::cos(dir * Constants::PI
                        / 180.0), speed * std::sin(dir * Constants::PI /
                        180.0), 0};
                    if (auto g = find_interception_details(v_uav, Constants
                        ::INITIAL_POSITIONS.at(uav_id))) {
```

```
                        current_seed.segment(i * 4, 4) << dir, speed, g->t_release, g->t_detonate_after;
                        found_strategy_for_uav = true;
                        break;
                    }
                }
                if (!found_strategy_for_uav) {
                    success = false;
                    break;
                }
            }
            if (success) {
                return current_seed;
            }
        }
    }

    // <<< MODIFICATION: Scalable, thread-pool based seeder >>>
    std::vector<Eigen::VectorXd> generate_seeds_problem4(int num_seeds) {
        std::cout << "Generating " << num_seeds << " holistic seeds for Problem 4 (Parallel)..." << std::endl;

        // Determine the optimal number of worker threads
        unsigned int num_workers = std::thread::hardware_concurrency();
        if (num_workers == 0) num_workers = 4; // Fallback for safety
        std::cout << "Using " << num_workers << " worker threads for generation." << std::endl;

        std::vector<std::future<Eigen::VectorXd>> futures;
        futures.reserve(num_workers);
        std::vector<Eigen::VectorXd> seeds;
        seeds.reserve(num_seeds);

        int tasks_launched = 0;
        int tasks_completed = 0;

        while (tasks_completed < num_seeds) {
            // Launch new tasks until the worker pool is full or all tasks are launched
            while (futures.size() < num_workers && tasks_launched < num_seeds) {
                futures.push_back(std::async(std::launch::async, generate_single_seed_p4));
                tasks_launched++;
            }
```

```cpp

            // Check for completed tasks and collect results
            for (auto it = futures.begin(); it != futures.end(); ) {
                if (it->wait_for(std::chrono::seconds(0)) == std::future_status::ready) {
                    seeds.push_back(it->get());
                    tasks_completed++;
                    it = futures.erase(it); // Remove completed future
                } else {
                    ++it;
                }
            }

            // Progress bar
            int bar_width = 50;
            float progress = (float)tasks_completed / num_seeds;
            int pos = bar_width * progress;
            std::cout << "\r  Generating seeds: [";
            for (int k = 0; k < bar_width; ++k) {
                if (k < pos) std::cout << "=";
                else if (k == pos) std::cout << ">";
                else std::cout << " ";
            }
            std::cout << "] " << tasks_completed << "/" << num_seeds << " (" << std::fixed << std::setprecision(1) << progress * 100.0 << "%)" << std::flush;

            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }

        std::cout << "\nSeed generation complete." << std::endl;
        return seeds;
    }
}

// ============================================================
// SECTION 5: 自包含的 CMA-ES 优化器
// ============================================================

class SimpleCMAES {
public:
    struct Solution { Eigen::VectorXd x; double fitness; };

    SimpleCMAES(int dim, int population_size = 0) : N(dim) {
        lambda = population_size > 0 ? population_size : 4 + floor(3 * log(
```

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

        cc = (4 + mu_eff / N) / (N + 4 + 2 * mu_eff / N);
        cs = (mu_eff + 2) / (N + mu_eff + 5);
        c1 = 2 / (pow(N + 1.3, 2) + mu_eff);
        cmu = std::min(1 - c1, 2 * (mu_eff - 2 + 1 / mu_eff) / (pow(N + 2, 2) + mu_eff));
        damps = 1 + 2 * std::max(0.0, sqrt((mu_eff - 1) / (N + 1)) - 1) + cs;
        chiN = sqrt(N) * (1 - 1 / (4.0 * N) + 1 / (21.0 * N * N));
    }

    void optimize(const std::function<double(const Eigen::VectorXd&)>& fitness_func, Eigen::VectorXd& x_start, double sigma_start, int max_iter) {
        Eigen::VectorXd x_mean = x_start;
        double sigma = sigma_start;
        Eigen::MatrixXd C = Eigen::MatrixXd::Identity(N, N);
        Eigen::VectorXd pc = Eigen::VectorXd::Zero(N), ps = Eigen::VectorXd::Zero(N);

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

        for (int gen = 0; gen < max_iter; ++gen) {
            Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> es(C);
            Eigen::MatrixXd B = es.eigenvectors();
            Eigen::VectorXd D_vec = es.eigenvalues();
            for(int i=0; i<N; ++i) D_vec(i) = std::sqrt(std::max(1e-20, D_vec(i)));

            std::vector<Solution> population(lambda);
            for (int i = 0; i < lambda; ++i) {
                Eigen::VectorXd z = Eigen::VectorXd::NullaryExpr(N, [&](){ return std::normal_distribution<double>{}(Seeder::get_rng()); });
                population[i].x = x_mean + sigma * B * D_vec.asDiagonal() * z;
                population[i].fitness = fitness_func(population[i].x);
            }

```

```
            std::sort(population.begin(), population.end(), [](const auto&
                a, const auto& b) { return a.fitness > b.fitness; });

            if (population[0].fitness > best_solution_so_far.fitness) {
                best_solution_so_far = population[0];
            }

            Eigen::VectorXd x_old = x_mean;
            x_mean.setZero();
            for(int i=0; i<mu; ++i) x_mean += weights(i) * population[i].x;

            ps = (1-cs)*ps + sqrt(cs*(2-cs)*mu_eff) * (B * D_vec.asDiagonal
                ().inverse() * B.transpose() * (x_mean - x_old) / sigma);
            sigma *= exp((cs/damps) * (ps.norm()/chiN - 1));

            bool hsig = ps.norm() / sqrt(1 - pow(1-cs, 2*(gen+1))) < chiN *
                 (1.4 + 2.0/(N+1));
            pc = (1-cc)*pc;
            if (hsig) pc += sqrt(cc*(2-cc)*mu_eff) * (x_mean - x_old) /
                sigma;

            Eigen::MatrixXd rank_mu_update = Eigen::MatrixXd::Zero(N, N);
            for(int i=0; i<mu; ++i) {
                Eigen::VectorXd diff = (population[i].x - x_old) / sigma;
                rank_mu_update += weights(i) * diff * diff.transpose();
            }
            C = (1 - c1 - cmu) * C + c1 * (pc * pc.transpose() + (1-hsig)*
                cc*(2-cc)*C) + cmu * rank_mu_update;
        }
        x_start = best_solution_so_far.x;
    }

private:
    int N, lambda, mu;
    double mu_eff, cc, cs, c1, cmu, damps, chiN;
    Eigen::VectorXd weights;
};

// ============================================================
// SECTION 6: 主要的两阶段优化逻辑 (问题四专用)
// ============================================================

struct OptimizationResult { Eigen::VectorXd solution; double fitness; };

FlightPlans decode_chromosome_p4(const Eigen::VectorXd& chromosome) {
```

```
    return {
            {"FY1", {chromosome(0), chromosome(1), {{chromosome(2), chromosome(3)}}}},
            {"FY2", {chromosome(4), chromosome(5), {{chromosome(6), chromosome(7)}}}},
            {"FY3", {chromosome(8), chromosome(9), {{chromosome(10), chromosome(11)}}}},
            {"FY4", {70.0, 0.0, {}}},
            {"FY5", {70.0, 0.0, {}}}
    };
}

void two_stage_optimization_p4(int num_screening_seeds, int screening_iterations, int num_champions, int deep_dive_iterations) {
    // --- STAGE 1: 广泛筛选 (使用固定大小的任务池并行) ---
    std::cout << "\n================␣STAGE␣1:␣Broad␣Screening␣(Parallel)␣================" << std::endl;
    auto initial_seeds = Seeder::generate_seeds_problem4(num_screening_seeds);

    // 确定最佳并发数, 通常等于硬件线程数
    unsigned int num_workers = std::thread::hardware_concurrency();
    if (num_workers == 0) num_workers = 8; // 安全回退值
    std::cout << "Using␣a␣concurrent␣task␣pool␣of␣size:␣" << num_workers << std::endl;

    std::vector<std::future<OptimizationResult>> active_futures;
    std::vector<OptimizationResult> screening_results;
    screening_results.reserve(num_screening_seeds);

    int launched_tasks = 0;
    int completed_tasks = 0;

    // 主循环, 直到所有任务都完成
    while (completed_tasks < num_screening_seeds) {
        // 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
        while (active_futures.size() < num_workers && launched_tasks < num_screening_seeds) {
            const auto& seed = initial_seeds[launched_tasks];
            active_futures.push_back(std::async(std::launch::async, [=]{
                SimpleCMAES cma(12);
                Eigen::VectorXd current_sol = seed;
                auto fitness_wrapper = [](const Eigen::VectorXd& x){
                    for (int i = 0; i < 3; ++i) {
                        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
            }
```

```
        }

        // 短暂休眠，避免主线程空转消耗CPU
        std::this_thread::sleep_for(std::chrono::milliseconds(10));
    }

    std::cout << std::endl;

    std::sort(screening_results.begin(), screening_results.end(), [](const auto& a, const auto& b){ return a.fitness > b.fitness; });

    std::cout << "\nStage␣1␣Screening␣Complete!␣Top␣preliminary␣results:" << std::endl;
    for (int i = 0; i < std::min(num_champions, (int)screening_results.size()); ++i) {
        std::cout << "␣␣Top␣" << i + 1 << ":␣M1␣Obscuration␣Time:␣" << std::fixed << std::setprecision(4) << screening_results[i].fitness << "␣s" << std::endl;
    }

    // --- STAGE 2: 深度优化 ---
    std::cout << "\n================␣STAGE␣2:␣Deep␣Dive␣Optimization␣================" << std::endl;
    OptimizationResult best_overall = {{}, -1.0};

    for (int i = 0; i < std::min(num_champions, (int)screening_results.size()); ++i) {
        std::cout << "␣␣Deep␣diving␣on␣champion␣#" << i+1 << "..." << std::endl;
        SimpleCMAES cma_deep(12);
        Eigen::VectorXd champion_sol = screening_results[i].solution;
        auto fitness_wrapper_fine = [](const Eigen::VectorXd& x){
            for (int i = 0; i < 3; ++i) {
                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), 0.01, 0.0001);
        };
        cma_deep.optimize(fitness_wrapper_fine, champion_sol, 0.2, deep_dive_iterations);
        double final_fitness = fitness_wrapper_fine(champion_sol);
        if (final_fitness > best_overall.fitness) {
```

```
            best_overall = {champion_sol, final_fitness};
        }
    }

    // --- 最终结果 ---
    std::cout << "\nOptimization␣complete!␣Performing␣final␣high-precision␣validation..." << std::endl;
    auto best_plans = decode_chromosome_p4(best_overall.solution);
    double final_time = calculate_time_for_m1(best_plans, 0.001, 0.00001);

    std::cout << "Final␣results␣calculated.␣Writing␣to␣output_p4.log..." << std::endl;
    std::ofstream out("output_p4.log");
    auto* cout_buf = std::cout.rdbuf();
    std::cout.rdbuf(out.rdbuf());

    std::cout << "\n=======================␣FINAL␣OPTIMAL␣STRATEGY␣(Problem␣4)␣=======================" << std::endl;
    std::cout << "Max␣Effective␣Obscuration␣Time␣for␣M1:␣" << std::fixed << std::setprecision(4) << final_time << "␣s" << std::endl;

    std::cout << "\nStrategy␣Details:" << std::endl;
    for (const auto& uav_id : {"FY1", "FY2", "FY3"}) {
        const auto& plan = best_plans.at(uav_id);
        const auto& grenade = plan.grenade_strategy[0];

        double angle_rad = plan.direction_deg * Constants::PI / 180.0;
        Vector3D v_uav = {plan.uav_speed * std::cos(angle_rad), plan.uav_speed * std::sin(angle_rad), 0};
        Vector3D p_drop = Constants::INITIAL_POSITIONS.at(uav_id) + v_uav * grenade.t_release;
        Vector3D p_detonate = p_drop + Vector3D{
                v_uav.x * grenade.t_detonate_after,
                v_uav.y * grenade.t_detonate_after,
                -0.5 * Constants::G * grenade.t_detonate_after * grenade.t_detonate_after
        };

        std::cout << "\n---␣UAV␣" << uav_id << "␣---" << std::endl;
        std::cout << "␣␣Flight␣Direction:␣" << std::fixed << std::setprecision(2) << plan.direction_deg << "␣deg" << std::endl;
        std::cout << "␣␣Flight␣Speed:␣" << plan.uav_speed << "␣m/s" << std::endl;
        std::cout << "␣␣Release␣Time:␣" << std::fixed << std::setprecision(3) << grenade.t_release << "␣s␣after␣mission␣start" << std::::
```

```
            endl;
        std::cout << "  Detonation Delay: " << grenade.t_detonate_after << " s after release" << std::endl;
        std::cout << "  Drop Point (x,y,z): (" << std::fixed << std::setprecision(2) << p_drop.x << ", " << p_drop.y << ", " << p_drop.z << ")" << std::endl;
        std::cout << "  Detonation Point (x,y,z): (" << p_detonate.x << ", " << p_detonate.y << ", " << p_detonate.z << ")" << std::endl;
    }
    std::cout << "======================================================" << std::endl;

    std::cout.rdbuf(cout_buf);
    std::cout << "Done. Results saved to output_p4.log" << std::endl;
}

int main() {
    // --- 在这里调整你的参数 ---
    int num_screening_seeds = 100000;    // 勘探的种子总数
    int screening_iterations = 100;   // 每个种子的快速迭代次数
    int num_champions = 25;           // 选出的冠军数量
    int deep_dive_iterations = 1000;  // 对冠军的深度迭代次数

    two_stage_optimization_p4(
            num_screening_seeds,
            screening_iterations,
            num_champions,
            deep_dive_iterations
    );

    return 0;
}
```

E 附录 E：问题五 JaVa 实现

Listing 6: 问题五 JAVA 实现代码

```




import java.util.*;

public class Main {
```

```java

    // --- Constants ---
    static final double G = 9.8;
    static final double MISSILE_SPEED_MPS = 300.0;
    static final double SMOKE_CLOUD_RADIUS_M = 10.0;
    static final double SMOKE_EFFECTIVE_DURATION_S = 20.0;
    static final double SMOKE_SINK_SPEED_MPS = 3.0;

    // --- Target geometry ---
    static final double[] TARGET_BASE_CENTER = new double[]{0, 200, 0};
    static final double TARGET_RADIUS = 7.0;
    static final double TARGET_HEIGHT = 10.0;

    // --- Initial positions ---
    static final Map<String, double[]> INITIAL_POSITIONS = new HashMap<>();
    static {
        INITIAL_POSITIONS.put("M1", new double[]{20000, 0, 2000});
        INITIAL_POSITIONS.put("M2", new double[]{19000, 600, 2100});
        INITIAL_POSITIONS.put("M3", new double[]{18000, -600, 1900});
        INITIAL_POSITIONS.put("FY1", new double[]{17800, 0, 1800});
        INITIAL_POSITIONS.put("FY2", new double[]{12000, 1400, 1400});
        INITIAL_POSITIONS.put("FY3", new double[]{6000, -3000, 700});
        INITIAL_POSITIONS.put("FY4", new double[]{11000, 2000, 1800});
        INITIAL_POSITIONS.put("FY5", new double[]{13000, -2000, 1300});
    }
    static final double[] FAKE_TARGET_POS = new double[]{0, 0, 0};

    // --- Missile directions (toward fake target) ---
    static final Map<String, double[]> MISSILE_DIRECTIONS = new HashMap<>()
        ;
    static {
        for (Map.Entry<String, double[]> e : INITIAL_POSITIONS.entrySet())
            {
            String id = e.getKey();
            if (id.startsWith("M")) {
                double[] v = sub(FAKE_TARGET_POS, e.getValue());
                MISSILE_DIRECTIONS.put(id, normalize(v));
            }
        }
    }

    // --- Id lists ---
    static final List<String> MISSILE_IDS = Arrays.asList("M1", "M2", "M3")
        ;
    static final List<String> UAV_IDS = Arrays.asList("FY1","FY2","FY3","
```

```java
        FY4","FY5");

    // --- Per-UAV angle ranges you provided (deg) ---
    // 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°
        ）。
    static final Map<String, double[]> ANGLE_RANGE_DEG = new HashMap<>();
    static {
        ANGLE_RANGE_DEG.put("FY1", new double[]{179.0, 180.0});
        ANGLE_RANGE_DEG.put("FY2", new double[]{186.7, 353.5});
        ANGLE_RANGE_DEG.put("FY3", new double[]{11.3, 153.4});
        ANGLE_RANGE_DEG.put("FY4", new double[]{190.3, 350.1});
        ANGLE_RANGE_DEG.put("FY5", new double[]{15.6, 171.3});
    }

    // --- Target points (rim points for multiple heights) ---
    static final List<double[]> REAL_TARGET_POINTS = generateTargetPoints(
            TARGET_BASE_CENTER, TARGET_RADIUS, TARGET_HEIGHT, 12, 2
    );

    // ---- Data types ----
    static class GrenadeStrategy {
        double tRelease;
        double tDetonateAfter;
        GrenadeStrategy(double tRelease, double tDetonateAfter) {
            this.tRelease = tRelease; this.tDetonateAfter = tDetonateAfter;
        }
    }
    static class UavPlan {
        double directionDeg;
        double uavSpeed;
        List<GrenadeStrategy> grenadeStrategy = new ArrayList<>();
    }
    static class Lifecycle {
        double[] pDetonate;
        double tDetonateAbs;
        Lifecycle(double[] pDetonate, double tDetonateAbs) {
            this.pDetonate = pDetonate; this.tDetonateAbs = tDetonateAbs;
        }
    }

    // ---- Helpers: vectors & ranges ----
    static double[] add(double[] a, double[] b) { return new double[]{a[0]+
        b[0], a[1]+b[1], a[2]+b[2]}; }
    static double[] sub(double[] a, double[] b) { return new double[]{a[0]-
        b[0], a[1]-b[1], a[2]-b[2]}; }
```

```
    static double[] mul(double[] a, double s)   { return new double[]{a[0]*s, a[1]*s, a[2]*s}; }
    static double dot(double[] a, double[] b)   { return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]; }
    static double norm2(double[] a)             { return Math.sqrt(dot(a,a)); }
    static double[] normalize(double[] a) {
        double n = norm2(a); return n==0 ? new double[]{0,0,0} : new double[]{a[0]/n,a[1]/n,a[2]/n};
    }
    static double uniform(Random rng, double lo, double hi) {
        return lo + rng.nextDouble() * (hi - lo);
    }

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

    // 在给定 UAV 的角度区间内采样 (支持环向)
    static double sampleDirectionDeg(String uavId, Random rng) {
        double[] range = ANGLE_RANGE_DEG.get(uavId);
        if (range == null) return rng.nextDouble() * 360.0; // 兜底: 全域
        double a = norm360(range[0]);
        double b = norm360(range[1]);
        if (a == b) return a;           // 单点
        if (a < b) {                     // 普通区间
            return uniform(rng, a, b);
        } else {                         // 环向区间 (跨越0°)
            double w1 = 360.0 - a;       // [a,360)
            double w2 = b;               // [0,b)
            double w = w1 + w2;
            double r = rng.nextDouble() * w;
            if (r < w1) return a + r;    // 落在[a,360)
            return r - w1;               // 落在[0,b)
        }
    }

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

```
                numHeightLevels-1.0));
            for (int k = 0; k < numRimPoints; k++) {
                double angle = 2.0*Math.PI*k/numRimPoints;
                double x = baseCenter[0] + radius * Math.cos(angle);
                double y = baseCenter[1] + radius * Math.sin(angle);
                double z = baseCenter[2] + h;
                pts.add(new double[]{x,y,z});
            }
        }
        return pts;
    }

    // ---- Core geometry test: line segment (missile->targetPoint) blocked
         by sphere (smoke) ----
    static boolean isSingleLineBlocked(double[] missilePos, double[]
        smokeCenter, double[] targetPoint) {
        double[] AB = sub(targetPoint, missilePos);
        double[] AC = sub(smokeCenter, missilePos);
        double denom = dot(AB, AB);
        double tProj = denom == 0 ? 0 : dot(AC, AB) / denom;
        double t = Math.max(0.0, Math.min(1.0, tProj));
        double[] closest = add(missilePos, mul(AB, t));
        double dx = smokeCenter[0]-closest[0], dy = smokeCenter[1]-closest
            [1], dz = smokeCenter[2]-closest[2];
        double dist2 = dx*dx + dy*dy + dz*dz;
        return dist2 <= SMOKE_CLOUD_RADIUS_M * SMOKE_CLOUD_RADIUS_M;
    }

    // ---- At time t: is the entire target obscured for a missile? ----
    static boolean getObscurationStateAtTime(double t, String missileId,
        List<Lifecycle> lifecycles) {
        List<double[]> activeCenters = new ArrayList<>();
        for (Lifecycle g : lifecycles) {
            double td = g.tDetonateAbs;
            if (t >= td && t < td + SMOKE_EFFECTIVE_DURATION_S) {
                double dt = t - td;
                double[] sink = new double[]{0,0, -SMOKE_SINK_SPEED_MPS *
                    dt};
                activeCenters.add(add(g.pDetonate, sink));
            }
        }
        if (activeCenters.isEmpty()) return false;

        double[] missilePos = add(INITIAL_POSITIONS.get(missileId),
                mul(MISSILE_DIRECTIONS.get(missileId), MISSILE_SPEED_MPS *
```

```
t));

        for (double[] targetPoint : REAL_TARGET_POINTS) {
            boolean blocked = false;
            for (double[] center : activeCenters) {
                if (isSingleLineBlocked(missilePos, center, targetPoint)) { blocked = true; break; }
            }
            if (!blocked) return false; // any visible point → not fully obscured
        }
        return true;
    }

    // ---- Binary search a state transition time in [tStart, tEnd] ----
    static double findTransitionTime(double tStart, double tEnd, String missileId, boolean startState,
                                     List<Lifecycle> lifecycles, double precision) {
        double low = tStart, high = tEnd;
        while ((high - low) > precision) {
            double mid = 0.5 * (low + high);
            boolean state = getObscurationStateAtTime(mid, missileId, lifecycles);
            if (state == startState) low = mid; else high = mid;
        }
        return high;
    }

    // ---- Main simulation with adaptive stepping via coarse grid + binary search ----
    static Map<String, Double> calculateSynergisticObscurationAdaptive(
            Map<String, UavPlan> uavPlans,
            List<String> missileIds,
            double coarseTimeStep,
            double precision,
            double maxSimTime
    ) {
        // 1) Precompute grenade lifecycles
        List<Lifecycle> lifecycles = new ArrayList<>();
        for (Map.Entry<String, UavPlan> e : uavPlans.entrySet()) {
            UavPlan p = e.getValue();
            if (p == null || p.grenadeStrategy == null) continue;
            double[] p0 = INITIAL_POSITIONS.get(e.getKey());
            double rad = Math.toRadians(p.directionDeg);
```

```
            double[] vUav = new double[]{p.uavSpeed * Math.cos(rad), p.uavSpeed * Math.sin(rad), 0};

            for (GrenadeStrategy gs : p.grenadeStrategy) {
                double tRelease = gs.tRelease;
                double tAfter   = gs.tDetonateAfter;
                double[] pDrop = add(p0, mul(vUav, tRelease));
                double[] pDet = new double[]{
                        pDrop[0] + vUav[0] * tAfter,
                        pDrop[1] + vUav[1] * tAfter,
                        pDrop[2] - 0.5 * G * tAfter * tAfter
                };
                lifecycles.add(new Lifecycle(pDet, tRelease + tAfter));
            }
        }

        // 2) Event-driven simulation per missile
        Map<String, Double> total = new HashMap<>();
        for (String mId : missileIds) total.put(mId, 0.0);

        for (String missileId : missileIds) {
            double lastT = 0.0;
            boolean lastState = getObscurationStateAtTime(0.0, missileId, lifecycles);
            Double obscStart = lastState ? 0.0 : null;

            int steps = (int)Math.ceil(maxSimTime / coarseTimeStep);
            for (int i = 1; i <= steps; i++) {
                double curT = i * coarseTimeStep;
                boolean curState = getObscurationStateAtTime(curT, missileId, lifecycles);

                if (curState != lastState) {
                    double tTrans = findTransitionTime(lastT, curT, missileId, lastState, lifecycles, precision);
                    if (curState) {
                        obscStart = tTrans;
                    } else {
                        if (obscStart != null) {
                            total.put(missileId, total.get(missileId) + (tTrans - obscStart));
                            obscStart = null;
                        }
                    }
                }
```

```java
                lastT = curT; lastState = curState;
            }
            if (obscStart != null) {
                total.put(missileId, total.get(missileId) + (maxSimTime - obscStart));
            }
        }
        return total;
    }

    // ---- Public wrappers ----
    static Map<String, Double> calculateTime(Map<String, UavPlan> plans,
                                              double coarseTimeStep, double precision) {
        return calculateSynergisticObscurationAdaptive(
                plans, MISSILE_IDS, coarseTimeStep, precision, 100.0
        );
    }
    static double sumMissileTimes(Map<String, Double> perMissile) {
        double s = 0.0;
        for (String m : MISSILE_IDS) s += perMissile.getOrDefault(m, 0.0);
        return s;
    }

    // ---- High-fidelity single-smoke test used by seed search ----
    static boolean isTargetFullyObscuredAtTime(double t, String missileId, List<double[]> activeSmokeCenters) {
        if (activeSmokeCenters == null || activeSmokeCenters.isEmpty()) return false;
        double[] missilePos = add(INITIAL_POSITIONS.get(missileId),
                mul(MISSILE_DIRECTIONS.get(missileId), MISSILE_SPEED_MPS * t));
        for (double[] tp : REAL_TARGET_POINTS) {
            boolean blocked = false;
            for (double[] c : activeSmokeCenters) {
                if (isSingleLineBlocked(missilePos, c, tp)) { blocked = true; break; }
            }
            if (!blocked) return false;
        }
        return true;
    }

    // ---- Any-missile obscuration test (for seed search) ----
    static boolean isObscuredForAnyMissile(double t, List<double[]>
```

```java
            activeSmokeCenters) {
        for (String mId : MISSILE_IDS) {
            if (isTargetFullyObscuredAtTime(t, mId, activeSmokeCenters))
                return true;
        }
        return false;
    }

    // ---- FY4: check strictly positive duration (>0) using two-time
        sampling ----
    static boolean grenadeHasPositiveDurationForAnyMissile(double
        tIntercept, double[] pSmoke) {
        double dt = 0.05; // 50ms 正时长近似
        double[] c0 = pSmoke;
        double[] c1 = new double[]{pSmoke[0], pSmoke[1], pSmoke[2] -
            SMOKE_SINK_SPEED_MPS * dt};
        for (String mId : MISSILE_IDS) {
            boolean b0 = isTargetFullyObscuredAtTime(tIntercept, mId,
                Collections.singletonList(c0));
            boolean b1 = isTargetFullyObscuredAtTime(tIntercept + dt, mId,
                Collections.singletonList(c1));
            if (b0 && b1) return true; // 同一导弹在相邻两时刻均被完全遮蔽
                → 持续时间>0
        }
        return false;
    }

    // ---- Search helpers ----
    static double[] findInterceptionDetailsForGrenadeAny(
            double[] vUav, double[] uavInitialPos, double tReleaseMin,
                double tReleaseMax, Random rng
    ) {
        List<Double> releases = linspaceShuffled(tReleaseMin, tReleaseMax,
            20, rng);
        for (double tRelease : releases) {
            double[] pDrop = add(uavInitialPos, mul(vUav, tRelease));
            List<Double> dets = linspaceShuffled(0.1, 8.0, 40, 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 (isObscuredForAnyMissile(tIntercept, Collections.
```

```java
                    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);

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

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

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

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

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

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

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

```
    static List<double[]> generateSeedsForUavFY4TwoValid(int numSeeds,
        Random rng) {
        final String uavId = "FY4";
        System.out.println("[FY4]␣使用“两枚弹必须有效”的标准生成种子：每
            枚需在起爆后具有␣>0␣的遮蔽时长，第三枚用占位值。");
        List<double[]> seeds = new ArrayList<>();
        double[] pUav = INITIAL_POSITIONS.get(uavId);

        int tries = 0;
        while (seeds.size() < numSeeds) {
            tries++;
            // FY4 也采用你给定的角度范围
            double directionDeg = sampleDirectionDeg(uavId, rng);
            double uavSpeed = 70.0 + rng.nextDouble()*(140.0-70.0);
            double rad = Math.toRadians(directionDeg);
            double[] vUav = new double[]{uavSpeed*Math.cos(rad), uavSpeed*
                Math.sin(rad), 0};

            // 第一枚：正遮蔽时长
            double[] g1 =
                findInterceptionDetailsForGrenadeAny_PositiveDuration(vUav,
                pUav, 0.5, 30.0, rng);
            if (g1 == null) continue;
            double t1 = g1[0], d1 = g1[1];

            // 第二枚：同样要求正遮蔽时长，且保证时间顺序（至少相隔 1s）
            double[] g2 =
                findInterceptionDetailsForGrenadeAny_PositiveDuration(vUav,
                pUav, t1 + 1.0, t1 + 15.0, rng);
            if (g2 == null) continue;
            double t2 = g2[0], d2 = g2[1];
            double gGap2 = t2 - t1 - 1.0;
            if (gGap2 < 0) continue;

            // 第三枚：占位最小合法值，优化阶段可自行调优为有效
            double gGap3 = 0.0;
            double d3 = 0.1;

            double[] seed = new double[]{directionDeg, uavSpeed, t1, d1,
                gGap2, d2, gGap3, d3};
            seeds.add(seed);
            System.out.printf("[FY4]␣Seed␣%d/%d␣␣(trials=%d)\n", seeds.size
                (), numSeeds, tries);
        }
```

```java
        System.out.println("[FY4] “两枚弹有效” 种子生成完毕! 优化阶段仍可把第三枚调整为有效值。");
        return seeds;
    }

    // ---- Bounds (same layout as Python) ----
    static final double[][] BOUNDS = new double[][]{
            {0,360},     // direction_deg (采样已受限于 ANGLE_RANGE_DEG, 此处边界仍保留)
            {70,140},    // uav_speed
            {0,50},      // t_release1
            {0.1,10},    // t_detonate1
            {0,15},      // t_gap2
            {0.1,10},    // t_detonate2
            {0,15},      // t_gap3
            {0.1,10}     // t_detonate3
    };

    // ---- Chromosome decode/encode ----
    static Map<String, UavPlan> decodeChromosomeForUav(String uavId, double[] chromosome) {
        double direction = chromosome[0];
        double speed     = chromosome[1];
        double t1        = chromosome[2];
        double d1        = chromosome[3];
        double g2        = chromosome[4];
        double d2        = chromosome[5];
        double g3        = chromosome[6];
        double d3        = chromosome[7];
        double t2 = t1 + 1.0 + g2;
        double t3 = t2 + 1.0 + g3;

        UavPlan plan = new UavPlan();
        plan.directionDeg = direction;
        plan.uavSpeed = speed;
        plan.grenadeStrategy.add(new GrenadeStrategy(t1, d1));
        plan.grenadeStrategy.add(new GrenadeStrategy(t2, d2));
        plan.grenadeStrategy.add(new GrenadeStrategy(t3, d3));

        Map<String, UavPlan> plans = new HashMap<>();
        plans.put(uavId, plan); // active
        for (String other : UAV_IDS) if (!other.equals(uavId)) plans.put(other, stubPlan());
        return plans;
    }
```

```java
    static double[] encodeChromosomeFromPlan(UavPlan p) {
        // layout: [dir, speed, t1, d1, g2, d2, g3, d3]
        double dir = p.directionDeg;
        double spd = p.uavSpeed;
        double t1=0,d1=0,t2=0,d2=0,t3=0,d3=0;
        if (p.grenadeStrategy.size() >= 1) { t1 = p.grenadeStrategy.get(0).tRelease; d1 = p.grenadeStrategy.get(0).tDetonateAfter; }
        if (p.grenadeStrategy.size() >= 2) { t2 = p.grenadeStrategy.get(1).tRelease; d2 = p.grenadeStrategy.get(1).tDetonateAfter; }
        if (p.grenadeStrategy.size() >= 3) { t3 = p.grenadeStrategy.get(2).tRelease; d3 = p.grenadeStrategy.get(2).tDetonateAfter; }
        double g2 = t2 - t1 - 1.0;
        double g3 = t3 - t2 - 1.0;
        return new double[]{dir, spd, t1, d1, g2, d2, g3, d3};
    }

    static UavPlan stubPlan() {
        UavPlan p = new UavPlan();
        p.directionDeg = 0; p.uavSpeed = 70; // idle, no grenades
        return p;
    }

    // ---- Fitness wrappers ----
    static double fitnessForUav_Independent(String uavId, double[] chromosome) {
        Map<String, UavPlan> plans = decodeChromosomeForUav(uavId, chromosome);
        Map<String, Double> res = calculateTime(plans, 0.005, 0.0001);
        return sumMissileTimes(res);
    }
    static double fitnessForUav_WithFixed(String uavId, double[] chromosome, Map<String, UavPlan> fixed) {
        Map<String, UavPlan> plans = new HashMap<>();
        for (String id : UAV_IDS) {
            if (fixed.containsKey(id)) plans.put(id, fixed.get(id));
        }
        Map<String, UavPlan> cand = decodeChromosomeForUav(uavId, chromosome);
        plans.put(uavId, cand.get(uavId));
        for (String id : UAV_IDS) plans.putIfAbsent(id, stubPlan());
        Map<String, Double> res = calculateTime(plans, 0.005, 0.0001);
        return sumMissileTimes(res);
    }

    // ---- Simplified CMA-ES ----
```

```
    static class SimpleCMAES {
        final int n;                 // dimension
        final double[] lower, upper; // bounds
        final Random rng;

        int lambda;                  // population size
        int mu;                      // parents
        double[] weights;            // recombination weights
        double mueff;
        double[] mean;               // current mean
        double sigma;                // global step-size

        SimpleCMAES(double[] x0, double sigma0, double[][] bounds, Random
            rng) {
            this.n = x0.length;
            this.lower = new double[n];
            this.upper = new double[n];
            for (int i=0;i<n;i++){ lower[i]=bounds[i][0]; upper[i]=bounds[i
                ][1]; }
            this.rng = rng;
            this.mean = Arrays.copyOf(x0, n);
            this.sigma = sigma0;

            this.lambda = 4 + (int)Math.floor(3*Math.log(n));
            this.mu = Math.max(2, lambda/2);
            this.weights = new double[mu];
            for (int i=0;i<mu;i++) weights[i] = Math.log(mu+0.5) - Math.log
                (i+1);
            double wsum = 0; for (double w:weights) wsum += w;
            for (int i=0;i<mu;i++) weights[i] /= wsum;
            double w2 = 0; for (double w:weights) w2 += w*w;
            this.mueff = 1.0 / w2;
        }

        double[] optimize(java.util.function.ToDoubleFunction<double[]>
            objective,
                          int maxIter) {
            double bestF = Double.NEGATIVE_INFINITY;
            double[] bestX = Arrays.copyOf(mean, n);

            for (int it=0; it<maxIter; it++) {
                List<Cand> pop = new ArrayList<>(lambda);
                for (int k=0;k<lambda;k++) {
                    double[] z = randn(n);
                    double[] x = new double[n];
```

```
                for (int i=0;i<n;i++) {
                    x[i] = mean[i] + sigma * z[i];
                    if (x[i]<lower[i]) x[i]=lower[i];
                    if (x[i]>upper[i]) x[i]=upper[i];
                }
                double f = objective.applyAsDouble(x);
                pop.add(new Cand(x,f));
            }
            pop.sort((a,b)-> Double.compare(b.f, a.f)); // max

            double[] newMean = new double[n];
            for (int i=0;i<mu;i++) {
                double w = weights[i];
                double[] xi = pop.get(i).x;
                for (int j=0;j<n;j++) newMean[j] += w * xi[j];
            }

            double oldGain = objective.applyAsDouble(mean);
            double newGain = objective.applyAsDouble(newMean);
            boolean success = newGain > oldGain;
            sigma *= success ? 1.05 : 0.95;

            mean = newMean;

            if (pop.get(0).f > bestF) {
                bestF = pop.get(0).f;
                bestX = Arrays.copyOf(pop.get(0).x, n);
            }
        }
        return Arrays.copyOf(bestX, n);
    }

    static class Cand {
        double[] x; double f;
        Cand(double[] x, double f){ this.x=x; this.f=f; }
    }

    double[] randn(int d) {
        double[] z = new double[d];
        for (int i=0;i<d;i++) z[i] = gaussian();
        return z;
    }
    double gaussian() {
        double u1 = Math.max(1e-12, rng.nextDouble());
        double u2 = rng.nextDouble();
```

```
            return Math.sqrt(-2.0*Math.log(u1)) * Math.cos(2*Math.PI*u2);
        }
    }

    // ---- Optimize wrapper ----
    static class OptimizeResult {
        double[] bestSolution;
        double bestFitness;
        OptimizeResult(double[] x, double f){ bestSolution=x; bestFitness=f
            ; }
    }
    static OptimizeResult optimizeWithCMAES(double[] initialSolution,
                                            java.util.function.
                                                ToDoubleFunction<double[]>
                                                fitnessFunc,
                                            int maxIter,
                                            Random rng) {
        SimpleCMAES es = new SimpleCMAES(initialSolution, /*sigma0*/0.1,
            BOUNDS, rng);
        double[] xbest = es.optimize(fitnessFunc, maxIter);
        double fbest = fitnessFunc.applyAsDouble(xbest);
        return new OptimizeResult(xbest, fbest);
    }

    // ---- Independent optimization for a chosen UAV ----
    static class UavOptimizationOutcome {
        String uavId;
        double[] bestSol;
        double bestFitness;
        Map<String, Double> finalResPerMissile;
        UavPlan plan;
    }

    static UavOptimizationOutcome optimizeForUav_Independent(String uavId,
        int numSeeds, int maxIter, Random rng) {
        List<double[]> seeds = generateSeedsForUav(uavId, numSeeds, rng);
        if (seeds.isEmpty()) {
            System.out.println("[" + uavId + "]␣错误：未生成有效初始种子，
                跳过该无人机。");
            return null;
        }
        System.out.println("[" + uavId + "]␣开始使用CMA-ES进行独立优化（目
            标：最大化␣M1+M2+M3␣遮蔽总时长）...");

        double[] bestSol = null; double bestFit = -1; int done = 0;
```

```
        for (double[] seed : seeds) {
            OptimizeResult r = optimizeWithCMAES(seed, x -> fitnessForUav_Independent(uavId, x), maxIter, rng);
            if (r.bestFitness > bestFit) { bestFit = r.bestFitness; bestSol = Arrays.copyOf(r.bestSolution, r.bestSolution.length); }
            done++;
            System.out.printf("[" + uavId + "]␣优化进度␣%d/%d␣␣当前最优适应度(总遮蔽秒):␣%.3f\n", done, seeds.size(), bestFit);
        }
        if (bestSol == null) return null;

        Map<String, UavPlan> bestPlans = decodeChromosomeForUav(uavId, bestSol);
        Map<String, Double> finalRes = calculateTime(bestPlans, 0.0001, 0.0000001);

        UavOptimizationOutcome out = new UavOptimizationOutcome();
        out.uavId = uavId;
        out.bestSol = bestSol;
        out.bestFitness = bestFit;
        out.finalResPerMissile = finalRes;
        out.plan = bestPlans.get(uavId);
        return out;
    }

    // ---- 贪心拼装（核心）：按“指定顺序”拼装 ----
    static Map<String, UavPlan> buildInitialAssemblyGreedyWithOrder(List<String> order, int seedsPerUav, long baseSeed) {
        Map<String, UavPlan> current = new HashMap<>();
        for (String id : UAV_IDS) current.put(id, null); // start empty

        System.out.println("\n[Greedy]␣使用顺序: " + order);

        for (String uav : order) {
            System.out.println("[Greedy]␣处理␣" + uav + "␣...");
            // 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, 避免因顺序不同导致候选集不同
            long perUavSeed = baseSeed + Math.abs(uav.hashCode());
            List<double[]> candidates = generateSeedsForUav(uav, seedsPerUav, new Random(perUavSeed));
            if (candidates.isEmpty()) {
                System.out.println("[Greedy]␣" + uav + "␣无候选, 使用␣stub。");
                current.put(uav, stubPlan());
                continue;
```

```java
            }
            double best = -1; UavPlan bestPlan = null;
            for (double[] ch : candidates) {
                UavPlan plan = decodeChromosomeForUav(uav, ch).get(uav);
                Map<String, UavPlan> tmp = new HashMap<>(current);
                tmp.put(uav, plan);
                for (String id : UAV_IDS) tmp.putIfAbsent(id, stubPlan());
                double sc = sumMissileTimes(calculateTime(tmp, 0.005, 0.0001));
                if (sc > best) { best = sc; bestPlan = plan; }
            }
            current.put(uav, bestPlan);
            System.out.printf("[Greedy]␣选择␣%s␣后当前三导弹总遮蔽时长=%.3f␣s\n", uav, best);
        }
        return current;
    }

    // ---- 生成一个随机顺序 ----
    static List<String> randomOrder(List<String> items, Random rng) {
        List<String> ord = new ArrayList<>(items);
        Collections.shuffle(ord, rng);
        return ord;
    }

    // ---- 多顺序贪心拼装：采样 numOrders 个随机顺序，取最优 ----
    static Map<String, UavPlan> buildInitialAssemblyGreedy_MultiOrders(int seedsPerUav, long baseSeed, int numOrders, long orderSeed) {
        double bestScore = -1;
        Map<String, UavPlan> bestAssembly = null;
        List<String> bestOrder = null;

        Random rng = new Random(orderSeed);
        for (int k = 1; k <= numOrders; k++) {
            List<String> ord = randomOrder(UAV_IDS, new Random(rng.nextLong()));
            Map<String, UavPlan> assembly = buildInitialAssemblyGreedyWithOrder(ord, seedsPerUav, baseSeed);
            double score = sumMissileTimes(calculateTime(completePlans(assembly), 0.005, 0.0001));
            System.out.printf("[Greedy-MO]␣顺序␣#%d␣总遮蔽=%.3f␣s,␣顺序=%s\n", k, score, ord);
            if (score > bestScore) {
                bestScore = score;
```

```java
                bestAssembly = assembly;
                bestOrder = ord;
            }
        }
        System.out.printf("\n[Greedy-MO] 选用最优顺序: %s  (联合初始解=%.3f s)\n", bestOrder, bestScore);
        return bestAssembly;
    }

    // ---- 补齐缺失 UAV 为 stubPlan (用于评估)
    static Map<String, UavPlan> completePlans(Map<String, UavPlan> maybePartial) {
        Map<String, UavPlan> res = new HashMap<>(maybePartial);
        for (String id : UAV_IDS) res.putIfAbsent(id, stubPlan());
        return res;
    }

    // ---- SCF: 固定其它4架, 微调当前一架 ----
    static UavPlan optimizeOneWithFixed(String uavId, Map<String, UavPlan> fixedPlans,
                                        int seeds, int iters, long seedBase) {
        System.out.println("\n[SCF] 微调 " + uavId + ", 其它固定。");
        List<double[]> seedsList = generateSeedsForUav(uavId, seeds, new Random(seedBase));
        if (fixedPlans.get(uavId) != null && fixedPlans.get(uavId).grenadeStrategy.size() >= 3) {
            seedsList.add(encodeChromosomeFromPlan(fixedPlans.get(uavId)));
        }
        double[] bestSol = null; double bestScore = -1;
        int idx = 0;
        for (double[] s : seedsList) {
            OptimizeResult r = optimizeWithCMAES(
                    s,
                    x -> fitnessForUav_WithFixed(uavId, x, fixedPlans),
                    iters,
                    new Random(seedBase + 100 + idx)
            );
            if (r.bestFitness > bestScore) { bestScore = r.bestFitness; bestSol = Arrays.copyOf(r.bestSolution, r.bestSolution.length); }
            idx++;
        }
        if (bestSol == null) return fixedPlans.get(uavId);
        return decodeChromosomeForUav(uavId, bestSol).get(uavId);
```

```java
    }

    // ---- SCF 总控: 多轮交替优化 (FY1→FY5 轮流微调) ----
    static Map<String, UavPlan> solve_SCF() {
        // 0) 初始拼装: 多顺序比较取优
        int seedsPerUavInit = 50;
        int numOrders = 10;        // 你希望的“随机生成10个顺序”
        long baseSeed = 2025L;     // 候选生成的基准种子 (与顺序无关)
        long orderSeed = 4263L;    // 生成随机顺序的种子 (可改)

        Map<String, UavPlan> cur = buildInitialAssemblyGreedy_MultiOrders(
            seedsPerUavInit, baseSeed, numOrders, orderSeed);
        cur = completePlans(cur);

        double bestTotal = sumMissileTimes(calculateTime(cur, 0.005,
            0.0001));
        System.out.printf("\n[Init] 初始拼装联合目标=%.3f s\n", bestTotal);

        // 1) 交替优化参数
        int ROUNDS = 10;             // SCF 轮数
        int seedsLocal = 12;        // 每次微调的种子数
        int itersLocal = 120;       // 每个种子的 CMA-ES 迭代
        double tolImprove = 0.01;   // 小于该提升则判为收敛

        for (int r=1; r<=ROUNDS; r++) {
            System.out.println("\n===================== SCF 轮次 " + r + "
                =====================");
            double roundStart = bestTotal;

            for (int i=0;i<UAV_IDS.size();i++) {
                String uav = UAV_IDS.get(i);
                Map<String, UavPlan> fixed = new HashMap<>(cur);
                UavPlan newPlan = optimizeOneWithFixed(uav, fixed,
                    seedsLocal, itersLocal, 3000L + r*1000L + i*100L);
                cur.put(uav, newPlan);

                double tot = sumMissileTimes(calculateTime(cur, 0.005,
                    0.0001));
                System.out.printf("[SCF] 更新 %s 后联合目标=%.3f s\n", uav,
                     tot);
                if (tot > bestTotal + 1e-9) bestTotal = tot;
            }

            double gain = bestTotal - roundStart;
            System.out.printf("[SCF] 轮 %d 结束: 提升 %.3f s\n", r, gain);
```

```java
            if (gain < tolImprove) {
                System.out.println("[SCF]␣提升不足阈值，提前收敛。");
                break;
            }
        }

        // 打印结果
        Map<String, Double> finalPerMissile = calculateTime(cur, 0.0005, 0.000001);
        double finalSum = sumMissileTimes(finalPerMissile);

        System.out.println("\n====================␣␣问题五（SCF联合最优）␣␣====================");
        System.out.printf("三导弹总遮蔽时长:␣%.4f␣s\n", finalSum);
        System.out.printf("M1:␣%.4f␣s,␣␣M2:␣%.4f␣s,␣␣M3:␣%.4f␣s\n",
                finalPerMissile.getOrDefault("M1",0.0),
                finalPerMissile.getOrDefault("M2",0.0),
                finalPerMissile.getOrDefault("M3",0.0));

        for (String uav : UAV_IDS) {
            UavPlan p = cur.get(uav);
            System.out.println("\n--------------------------------------------------");
            System.out.println("␣␣无人机␣" + uav + "␣最优投放策略");
            System.out.println("--------------------------------------------------");
            System.out.printf("␣␣飞行方向:␣%.2f␣度\n", p.directionDeg);
            System.out.printf("␣␣飞行速度:␣%.2f␣m/s\n", p.uavSpeed);
            System.out.println("␣␣干扰弹投放时序:");
            for (int i=0;i<p.grenadeStrategy.size();i++) {
                GrenadeStrategy g = p.grenadeStrategy.get(i);
                System.out.printf("␣␣␣␣-␣#%d:␣受领后␣%.3f␣s␣投放，投放后␣%.3f␣s␣起爆\n",
                        i+1, g.tRelease, g.tDetonateAfter);
            }
        }
        System.out.println("==================================================\n");

        return cur;
    }

    public static void main(String[] args) {
        solve_SCF();
    }
```

```
}
```