基于粒子群优化的无人机烟幕弹投放轨迹规划与策略研究

摘要

本研究面向 M1 对目标的制导视线，在云团有效期（20s）、下沉速度（3m/s）、有效半径（10m）及同平台投弹间隔≥1s 等约束下，旨在最大化烟幕对视线的有效遮蔽并集时长，构建接力覆盖优化模型，采用改进粒子群优化（PSO）求解，结合粗细两级评估（dt=0.02/0.001）与边界插值保障方案稳定可复现。

问题一（基线评估，严格判据）：围绕“导引视线对目标的全遮挡时长”这一严格判据，建立 M1 与 FY1/烟幕云团的几何—运动学模型，对 M1—T 视线与云团相交关系进行逐时刻判定，并在穿越边界处作线性插值以获得稳定一致的时长累积。结果显示在既定方案下出现连续全遮挡区间[8.060 s, 9.450 s]，总全遮挡时长为 1.390 s。该基线刻画了可达遮蔽的时间尺度与窗口，为后续参数化优化提供对比基准与约束边界。

问题二（单弹优化，参数化求解）：在上述几何与判据框架下，将 FY1 航向、速度、投放时刻与引信延时作为决策，构建“遮蔽时长最大化”的单弹优化模型；求解采用改进 PSO（角度在环空间以最短角差更新，位置更新后投影至可行域，评估采用粗—细两级时间步并在边界插值）。代表性最优解的有效遮蔽时长约为 4.756 s，较基线显著延长，并形成后续多弹接力的“航线走廊”和“时间骨架”。

问题三（三弹接力，联合并集最大化）：在“相邻两弹投放间隔≥1 s”等工程约束下，扩展为三弹接力覆盖优化，联合优化航向、速度及三组投放—引信时序，以遮蔽并集时长 T_union 最大为目标[1]；算法延续改进 PSO 并加入时序强制排序/间隔校验与可行域投影，评估端保持两级时间步与边界插值。实跑代表性最优方案实现 T_union≈6.381 s，呈“错峰覆盖+部分叠加”，前两枚贡献为主；相较单弹最优进一步延展遮蔽持续性，体现了在严格判据与工程约束下，通过优化建模与逐层增强实现由评估到单弹优化再到多弹接力的清晰递进。

问题四（三机一弹协同，鲁棒并集最优）：在 FY1、FY2、FY3 各投放 1 枚的前提下，延续问题三的几何判据与优化框架，采用改进 PSO（角度环空间最短差、可行域投影、时序强制与间隔校验）与“粗—细两级时间步+边界插值”的评估，并通过窗口贴合与起爆对齐等软惩罚增强全局性与数值稳健性。结果形成稳定的“错峰接力”覆盖，联合并集时长约 11.6 秒；相较问题三的约 6.4 秒有显著提升，同时 FY1 的贡献与单弹基线保持一致。

问题五（五机三弹协同，跨目标并集增强）：在五架无人机、三枚来袭导弹（M1–M3）场景下，继续沿用严格判据与优化框架：先固定各机速度/航向，再对“无人机–导弹”对采用改进 PSO 搜索单弹最优，随后以“并集增益”贪心选

弹生成至多三弹/机的投放时序；评估端保持“粗—细两级时间步+边界插值”，并施加时序间隔与可行域投影以确保可行与稳健。结果形成“分组接力+错峰覆盖+轻度重叠”的多段协同，总体联合并集时长约 35.26 秒。
关键词：接力覆盖优化模型；改进 PSO；时间骨架退火；烟幕遮蔽；多无人机协同

一、问题重述

1.1 问题背景

烟幕干扰弹通过化学燃烧,爆炸形成气溶胶云团,低成本干扰来袭导弹,可借助技术实现定点精确抛撒与定时起爆；长续航无人机在特定空域巡飞，接令后需在来袭武器与真目标间投放干扰弹，规划无人机飞行方向,速度、干扰弹投放点,起爆点等关键变量，以最大化真目标的有效遮蔽时间。

核心约束与场景

导弹：3 枚空地导弹（M1-M3，速度 300m/s，直指假目标），初始三维坐标已知；假目标为原点（xy 平面），真目标下底面圆心 (0,200,0)。

无人机：5 架（FY1-FY5），初始坐标已知；接令后瞬时定航向，随后以 70-140m/s 等高度匀速直线飞行（航向 / 速度一旦确定不变）；同架无人机相邻投弹间隔≥1s。

烟幕弹：脱离无人机后受重力运动，起爆后形成球形云团（中心以 3m/s 匀速下沉，起爆 20s 内、中心 10m 范围内有效遮蔽）。

目标：最大化真目标在导弹来袭过程中的累计有效遮蔽时长。

1.2 具体问题

1. **问题一**：FY1 对 M1，给定参数（航向假目标、速度 120m/s、接令 1.5s 后投放、投放 3.6s 后起爆），计算有效遮蔽总时长，作为优化基准。
2. **问题二**：FY1 对 M1，在速度 70-140m/s 区间内，优化无人机航向、干扰弹投放、起爆时空参数，最大化有效遮蔽时长。
3. **问题三**：FY1 对 M1，最多投 3 枚弹（同机相邻投弹≥1s），定无人机航向、速度，规划多弹投放、起爆时空序列，实现接力遮蔽以最大化时长。
4. **问题四**：FY1-FY3 各投 1 弹对 M1，定每架机航向、速度，联合规划各弹投放、起爆参数，通过互补、接力最大化有效遮蔽时长。
5. **问题五**：5 架无人机应对 M1-M3，全局分配 “无人机 - 烟幕弹 - 导

弹”，定各机航向、速度及弹的投放、起爆参数，最大化真目标累计有效遮蔽时长。

二、问题分析

2.1 问题分析

核心目标、判据与通用方法

目标：在约束下规划无人机航向、速度、烟幕弹投放点、起爆时序，最大化真目标对来袭导弹的“全遮挡”总时长。

关键判据：烟幕云团起爆后 20s 内、中心 10m 球区为有效范围，云团以 3m/s 匀速下沉；“全遮挡”需满足“导弹到目标的视线先被云团截获，且目标可见点全被阻断”。

通用方法：构建无人机等速直线等高飞行、烟幕弹抛物投放及起爆后下沉的运动学模型；对圆柱状真目标表面离散采样，用射线首交判定可见性；在有效窗内时间扫描累积遮蔽时长，外层以粒子群算法（PSO）搜索决策变量，通过惩罚项或可行性编码处理速度区间、投放间隔等约束。

2.2 各问题具体分析

问题一为基准计算，已知 FY1 朝假目标飞行、接令 1.5s 后投放、投放 3.6s 后起爆，需基于运动学模型与圆柱目标采样，通过射线首交判定可见性，在云团有效窗内逐时验证“云团先交”，识别连续全遮挡时段并累计总时长，为后续优化提供基准。

问题二聚焦 FY1 对 M1 的单机单弹优化，在无人机速度 70-140m/s、航向/速度一经确定不变的约束下，以 PSO 搜索 FY1 航向、烟幕弹投放/起爆时序，内层对给定参数逐时扫描累计遮蔽时长，通过约束处理确保方案可行，目标是最大化全遮挡总时长。

问题三扩展为 FY1 对 M1 的单机三弹场景，需满足同机相邻投弹间隔≥1s，核心是在通用方法框架下，同步仿真三枚烟幕弹的云团演化，计算多云团遮蔽并集时长以减少冗余重叠，外层用 PSO 联合优化 FY1 航向、速度及三次投放、起爆时序，通过投放时刻排序与间隔校验满足约束。

问题四为三机协同（FY1、FY2、FY3 各投 1 弹对 M1），需协调多机时空布局实现错峰衔接，方法上同步建模三架无人机运动与三枚烟幕弹云团，累计多云团遮蔽并集时长，以 PSO 同步搜索三架机的航向、速度及各弹投放、起爆时序，确保约束合规并最大化遮蔽时长。

问题五为多机多弹多导弹场景（5 架无人机各至多 3 弹，应对 M1、M2、M3），需先完成“无人机—烟幕弹—导弹”任务分配，再沿用运动学与可见性判定逻辑，对三枚导弹分别计算遮蔽并集时长并求和，以混合编码 PSO 优化各机航向、速度、投弹序列及时序，通过约束处理满足投弹数量上限与间隔要求，目标是最大化总体遮蔽时长。

三、模型假设

1.计算过程中采用双精度浮点数近似，数值误差在设定容差内对判定不会造成本质影响。
2.几何理想化假设：目标几何（有限圆柱及其端盖）为理想刚体；云团为理想几何体（球），几何参数在有效期内恒定且可精确使用。
3.环境影响忽略假设：不考虑空气阻力、风场、湍流、温湿度、气压等环境对烟幕弹/云团的影响；不考虑云团的扩散、生长、衰减、上浮或水平漂移。
4.重力加速度假设：重力加速度取常数 g=9.8 m/s²，方向固定为 -z；环境条件随时间不变。
5.烟雾状态假设：（1）未爆开之前：烟幕弹在投放瞬间完全继承无人机的水平速度（与无人机方向一致），随后做“无空气阻力的抛体运动”，仅受重力作用（-z 方向加速度为 g）。（2）起爆之后：云团水平动量视为瞬间消失，仅保留恒定的垂直下沉速度；云团视作刚性球，不变形、不扩散。
6.完全遮挡定义：在某时刻，只有当全部可见采样点均被云团遮挡（遮挡比例=1.0）时，目标才被视为“完全遮挡”；只要任一可见点未被遮挡，即判定“暴露”。
7.为接近连续表面，使用规则的空间离散采样；为接近连续时间，使用固定步长的离散时刻检测，从离散的 full=True 序列拼接得到持续时间。

四、符号说明

参数	取值 / 定义
无人机 FY1 初始位置	$(x_{U0}, y_{U0}, z_{U0}) = (17800, 0, 1800)$(等高度飞行)

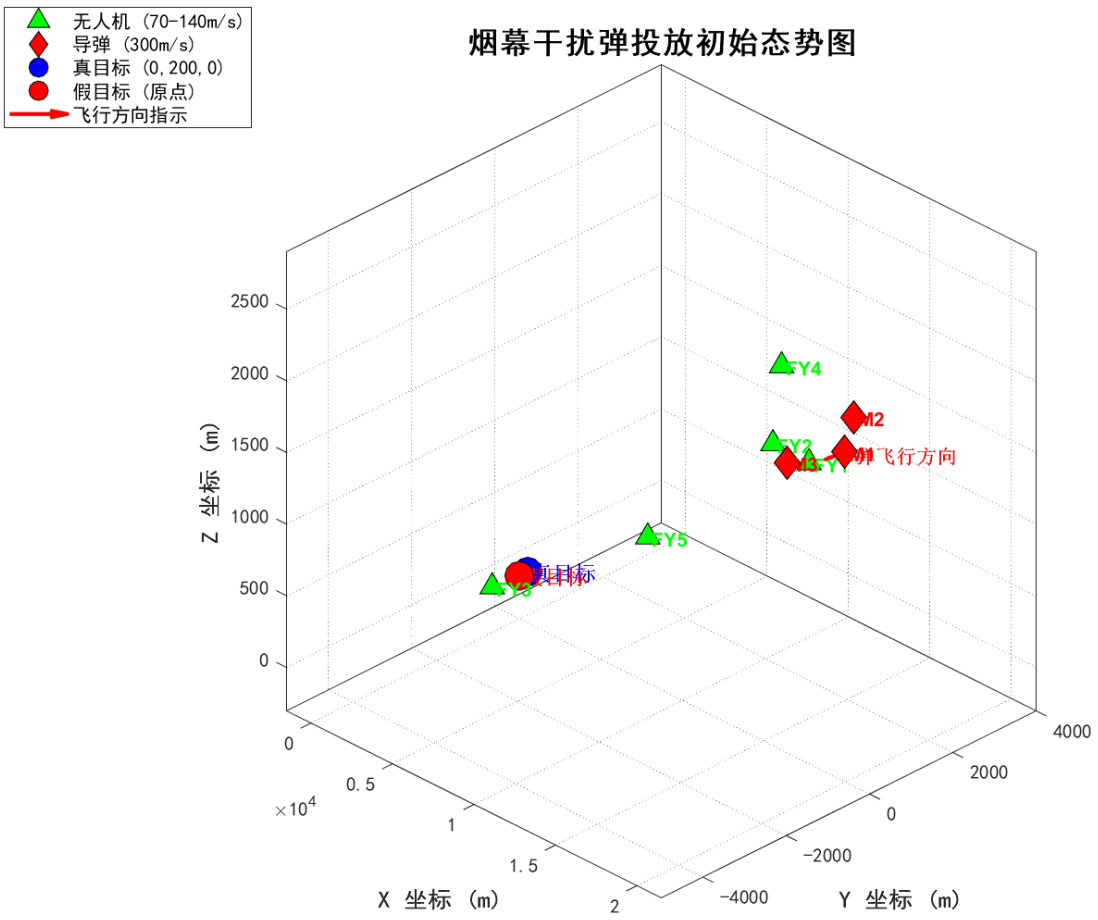
参数	取值 / 定义
无人机速度	$V \in [70,140] m/s$
导弹 M1 初始位置	$(x_{M0}, y_{M0}, z_{M0}) = (20000,0,2000) m$
导弹 M1 速度	$v_M = 300 m/s$
真目标位置	$T = (0, 200, 0)\ m$（真目标下底圆心）
重力加速度	$g = 9.8 m/s$
烟幕云团下沉速度	$v_{sink} = 3 m/s$(竖直向下）
烟幕有效半径	$R_{eff} = 10 m$（球状有效区域）
烟幕有效时长	起爆后 $20 s$（$t \in [t_0, t_0 + 20], t_0$为起爆时刻）

五、模型建立与求解

5.1 问题一模型的建立与求解

烟雾有效遮蔽时长的计算

无人机发射烟雾弹后，在任一 $t \in [\, t_0, \, t_0 + 20]$（t 为时间轴， t_0为烟雾弹爆开时刻），对导弹视角下目标所有可见的表面采样点 P，构造视线射线 $R(s) = M(t) + s \cdot (P - M(t))(s \geq 0)$。若该射线与以 C(t) 为球心、半径 10 m 的球体存在交点，且最近正交参数 s_cloud 满足 $0 < s\_cloud < s\_target$（其中 $s\_target = \|P - M(t)\|$），则视为该点在时刻 t 被云团先行遮挡。若在该时刻所有可见点均满足上述条件，则记该时刻为“全遮挡”。在 $t \in [\, t_0, \, t_0 + 20]$ 内，将“全遮挡”为真的时间集合取并集并累计区间长度，得到总遮蔽时长 ΔT。



5.1.1 数学模型建立

以“t=0 为雷达发现导弹/无人机受领任务时刻”为时间基准，定义各实体的位置参数方程如下，所有变量含义及取值范围均源于题目约束。

1. 无人机飞行模型

无人机受领任务（t=0 时刻，雷达发现导弹时刻）后，瞬时调整方向，以等高度匀速直线运动飞行。其运动特性满足：①飞行高度保持初始高度不变；②速度大小$v \in [70,140] m/s$，方向固定；③仅在水平面上（xy 平面）改变位置，竖直方向（z 轴）无位移。

无人机位置方程：设无人机初始位置为(x_{U0}, y_{U0}, z_{U0})，水平方向与 x 轴夹角为θ（航向角），则 t 时刻（t≥0，无人机已调整完方向）的位置为：

$$\begin{cases} x_U(t) = x_{U0} + v \cdot t \cdot \cos\theta \\ y_U(t) = y_{U0} + v \cdot t \cdot \sin\theta \\ z_U(t) = z_{U0} \end{cases}$$

其中，$z_U(t) = z_{U0}$体现“等高度飞行”特性。若烟幕弹投放时刻为t_r（$t_r \geq 0$），则投放点位置为$(x_U(t_r),\ y_U(t_r),\ z_{U0})$。

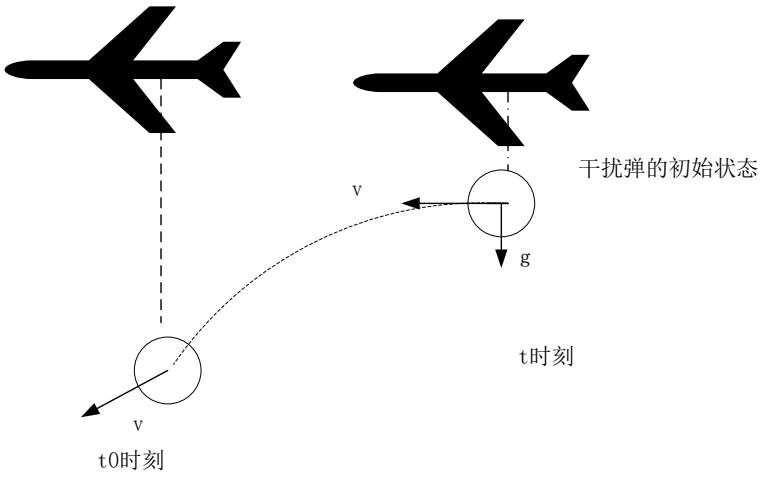
2. 烟幕弹运动模型

烟幕弹运动分为“脱离无人机后至起爆前”与“起爆后云团扩散”两个阶段：

（1）脱离至起爆（平抛运动）

烟幕弹脱离无人机时，水平方向速度与无人机速度一致，竖直方向仅受重力

作用（自由落体），直至起爆。起爆时刻与投放时刻的时间间隔记为Δt_d。



烟幕弹水平速度与无人机一致，竖直方向自由落体（重力加速度 g=9.8m/s²），则 t 时刻位置为：

$$\begin{cases} x_b(t) = x_U(t_r) + v \cdot \cos\theta \cdot (t - t_r) \\ y_b(t) = y_U(t_r) + v \cdot \sin\theta \cdot (t - t_r) \\ z_b(t) = z_{U0} - \frac{1}{2} g (t - t_r)^2 \end{cases}$$

起爆时刻$t_0 = t_r + \Delta t_d$，起爆点位置为(x_{b0}, y_{b0}, z_{b0}) = $(x\_b(t_0), y\_b(t_0), z\_b(t_0))$。

（2）起爆后云团运动

起爆后瞬时形成球状烟幕云团，云团水平位置固定（与起爆点水平坐标一致），竖直方向以 3m/s 速度匀速下沉，无水平方向速度；云团有效遮蔽时长为起爆后 20s，有效区域为“以云团中心为球心、10m 为半径的球体”。

云团水平位置固定，竖直方向以 3m/s 匀速下沉（），则 t 时刻云团中心位置为：

$$\begin{cases} x_c(t) = x_{b0} \\ y_c(t) = y_{b0} \\ z_c(t) = z_{b0} - 3 \cdot (t - t_0) \end{cases}$$

3. 导弹飞行模型

来袭导弹（如 M1）以 300m/s 匀速直线飞行，飞行方向直指假目标（原点(0,0,0)）。导弹初始位置已知 M1 初始位置(20000,0,2000)，飞行过程中无速度

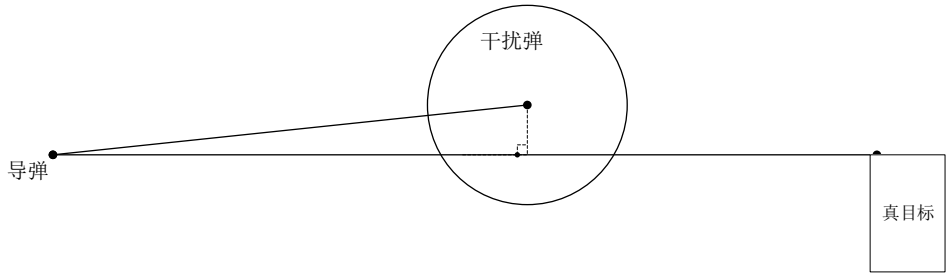
或方向变化，仅沿“初始位置-假目标”连线向原点运动。

设导弹初始位置为(x_{M0},y_{M0},z_{M0})（如 M1 的(20000,0,2000),），速度大小v_M=300m/s，飞行方向指向原点，则导弹的单位方向向量为：

$$\overrightarrow{e_M}=(-\frac{x_{M0}}{L_M},-\frac{y_{M0}}{L_M},-\frac{z_{M0}}{L_M})$$

其中，L_M为导弹初始位置到原点的距离，$L_M=\sqrt{x_{M0}^2+y_{M0}^2+z_{M0}^2}$则 t 时刻导弹位置为：

$$\begin{cases}x_M(t)=x_{M0}+v_M\cdot\overrightarrow{e_{M_x}}\cdot t\\ y_M(t)=y_{M0}+v_M\cdot\overrightarrow{e_{M_y}}\cdot t\\ z_M(t)=z_{M0}+v_M\cdot\overrightarrow{e_{M_z}}\cdot t\end{cases}$$



判断导弹是否穿过烟雾

验证导弹是否穿过烟雾——求解“空间距离方程”

核心是通过建立导弹与烟幕云团的“时空位置关系”，判断是否存在时间 t 使得两者空间距离≤10m（烟幕有效半径），具体步骤如下：

（1）确定导弹的位置方程（已知条件代入）

通过导弹的位置方程，得到 t 时刻（t 为雷达发现导弹后的时间）导弹位置：

$(x_M(t),y_{M0}(t),z_{M0}(t))$=$(x_{M0}-300\cdot\frac{x_{M0}}{L_M}\cdot t,\ y_{M0}-300\cdot\frac{y_{M0}}{L_M}\cdot t,\ z_{M0}-300\cdot\frac{z_{M0}}{L_M}\cdot t)$

（2）确定烟幕云团的时空范围（基于无人机投放参数）

根据无人机的飞行参数（速度、航向）和烟幕投放/起爆时间，确定烟幕云团的中心位置随时间变化规律：

投放点：无人机受领任务后以$v(70\sim140m/s)$等高度飞行，若投放时刻为t_r，则投放点坐标为（$x_U(r),y_U(r),z_{U0}$）（z_{U0}为无人机起始高度，如 FY1 的 1800m）；

起爆点与云团运动：t 时刻（$t \geq t_0$）云团中心坐标为：

$$(x\_c\ (t), y\_c\ (t), z\_c\ (t)) = (x_U(t_r) + v \cdot \cos\theta \cdot \Delta t_d,\ y_U(t_r) + v \cdot \sin\theta \cdot \Delta t_d,\ z_{U0} - \frac{1}{2} g t_d^2 - 3(t - t_0))$$

（θ为无人机航向角，g为重力加速度）。

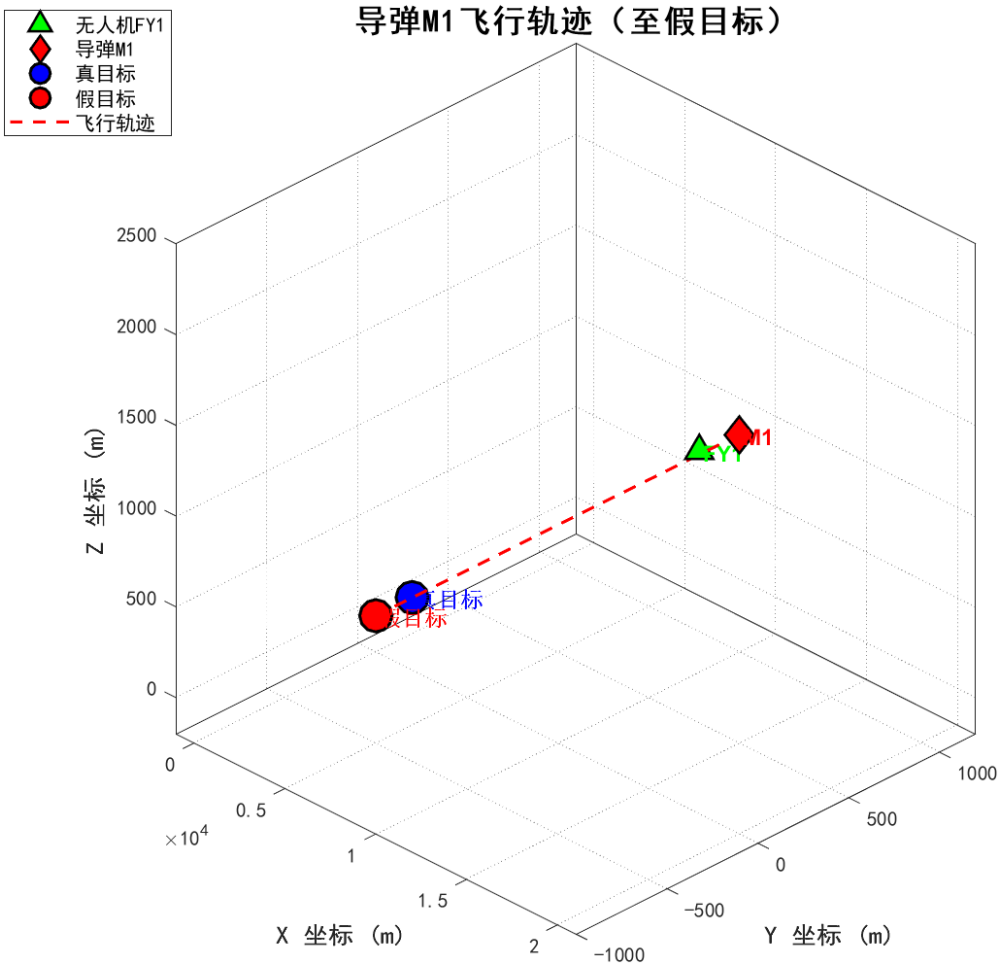
（3）求解空间距离方程，判断是否存在交集

建立导弹与云团中心的空间距离公式：

$$\sqrt{[x_M(t) - x_c(t)]^2 + [y_M(t) - y_c(t)]^2 + [z_M(t) - z_c(t)]^2} \leq 10$$

将导弹、云团的位置方程代入，求解该不等式是否存在解。若存在，则说明导弹的飞行轨迹与烟雾云团的有效空间范围存在交集，即“导弹穿过烟雾”。

经计算，该不等式存在交集，即说明导弹飞行轨迹与烟雾有效空间存在交集，即导弹穿过烟雾



计算有效窗内所有满足 “全遮挡判据” 的时间区间长度总和，公式为：

$$T_{total} = \int_{t=5.1}^{25.1} I(A(t))dt$$

其中，$I(A(t))$为指示函数：当 t 时刻导弹到真目标的所有可见点均被云团遮挡时，$I(A(t))$=1，否则$I(A(t))$=0。 （

5.1.2 烟雾模型的求解

（1）运动学

导弹、无人机均匀速直线；烟幕弹在投放后做抛物线（受重力），起爆后云团中心以给定速度沿 −z 下沉；

（2）目标采样与可见性筛选

在圆柱侧面与上下端盖均匀采样；对每个点 P，若导弹位置 M(t)出发的射线到圆柱的“最近首交”恰好在 P，则 P 是可见点。

（3）云团先截获判定

对每个可见点 P，计算导弹至 P 方向的射线与云团球的最近交点参数u_cloud；若存在且u_cloud小于 M(t)到 P 的距离，则该 P 被云团遮挡。

（4）全遮挡判断与时长累计

若时刻 t 所有可见点均被遮挡，则记为“全遮挡”；在有效时窗内以$\Delta t = 0.01\ s$扫描，把连续的全遮挡区间并段并累加得到总时长。

5.1.3 模型的结论

关键时序：投放 t=1.5 s，起爆 t=5.1 s，有效期截至 t=25.1 s。

结果：出现 1 段“全遮挡”，区间为 8.060 s — 9.450 s，持续 1.390 s。

因而，烟幕对导弹 M1 的有效遮蔽总时长（按全遮挡口径）为 1.390 s。

```
烟幕弹投放位置 (t=1.5s): [17620.     0.  1800.]
烟幕弹起爆位置 (t=5.1s): [17188.        0.     1736.496]
导弹总飞行时间: 67.00 秒
烟幕弹投放时间: 1.50 秒
烟幕弹起爆时间: 5.10 秒
烟幕有效结束时间: 25.10 秒

'全遮挡'分析结果:
发现 1 个全遮挡时间段:
  第1段: 8.060s - 9.450s (持续 1.390s)

总有效遮蔽时长(全遮挡): 1.390 秒
```

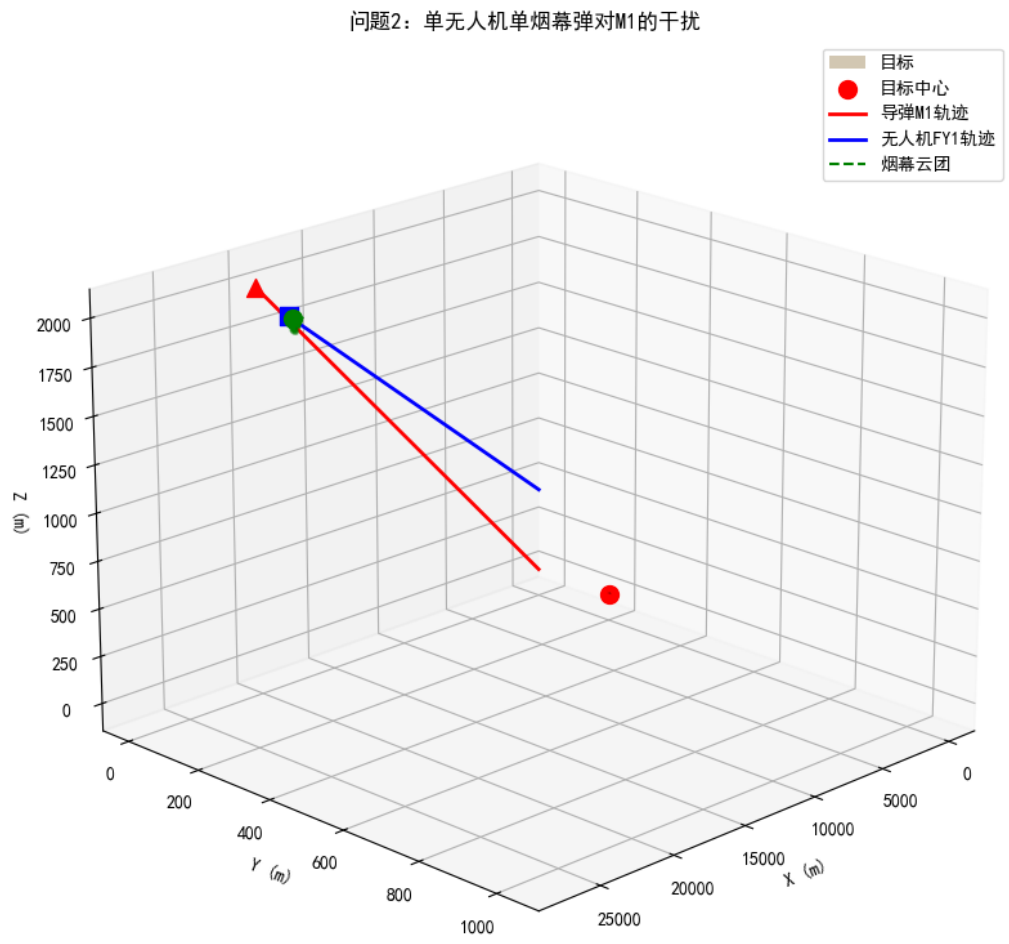
5.2 问题二模型的建立与求解

问题二，核心任务为：利用无人机 FY1 投放 1 枚烟幕干扰弹，对来袭导弹 M1 实施干扰，需确定 FY1 的**飞行速度（v）、飞行方向（航向角θ）、干扰弹投放时刻（t_r）、干扰弹起爆延迟时间（Δt_d）** 四个关键参数，使烟幕对 M1 的**有效遮蔽时长最大化**。

5.2.1 模型的建立

1. 优化目标

最大化烟幕云团对导弹 M1 的**有效遮蔽总时长**（基于 “全遮挡判据” 计算的 ΔT）。



2. 决策变量（4 维，PSO 粒子维度 = 4）

决策变量 $(v, \theta, t_r, \Delta t_d)$

v：无人机 FY1 飞行速度

θ：无人机 FY1 航向角（与 x 轴夹角）

t_r：烟幕弹投放时刻

Δt_d：烟幕弹起爆延迟（投放至起爆时间）

3. 约束条件

无人机飞行约束速度范围

$$70 \le v \le 140 m/s$$

可飞行角度

$$0° \le \theta \le 360°$$

烟幕弹投放时刻

$$10^{-3} \le t_r \le 3s$$

烟幕弹起爆延迟

$$10^{-3} \leq \Delta t_d \leq 1.5s$$

上述决策变量及约束条件见表 2

其他约束：无人机航向与速度一旦设定，全程保持恒定（等高度直线飞行）。

综合上述决策变量和约束条件，形成如下表格：

表 2:无人机决策变量及约束条件

决策变量	物理含义	约束范围
v	无人机 FY1 飞行速度	$70 \leq v \leq 140m/s$
θ	无人机 FY1 航向角（与 x 轴夹角）	$0° \leq \theta \leq 360°$
t_r	烟幕弹投放时刻	$10^{-3} \leq t_r \leq 3s$
Δt_d	烟幕弹起爆延迟（投放至起爆时间）	$10^{-3} \leq \Delta t_d \leq 1.5s$

因此，可得导弹遮蔽优化模型

目标函数：以有效窗内 “全遮挡” 时间总和为目标，函数表达式为：

$$max_{t_r,\Delta t_d} T_{total}(t_r,,\Delta t_d) = max_{t_r,\Delta t_d} \int_{t=t_r+\Delta t_d}^{t_r+\Delta t_{d+20}} I(A(t))dt$$

约束条件：$\begin{cases} 70 \leq v \leq 140m/s \\ 0° \leq \theta \leq 360° \\ 10^{-3} \leq t_r \leq 3s \\ 10^{-3} \leq \Delta t_d \leq 1.5s \end{cases}$

决策变量：$(v, \theta, t_r, \Delta t_d)$

4. 求解算法

算法参数设置

基于问题特性与优化稳定性要求，参数设置如下：

参数	符号	含义	取值
粒子群规模	$swarm$	群体中粒子的总数	20

参数	符号	含义	取值
迭代次数	$iters$	算法迭代的总次数	50
随机种子数	$seed$	平衡探索（全局）与利用（局部）	42
粗步长（优化阶段）	Δt_{coarse}	目标函数计算的离散步长	0.03
细步长（验证阶段）	Δt_{refine}	最终结果验证的离散步长	0.001

约束处理：对超出范围的粒子位置，强制投影到可行域（如速度 v 超限时取 70 或 140）；

精度控制：优化阶段用$\Delta t = 0.03s$计算目标函数，迭代结束后用$\Delta t = 0.001s$精算最优解的遮蔽时长。

物理模型基于 “匀速运动”“自由落体” 等经典规律，描述无人机、烟幕弹、导弹及烟幕云团的运动过程，所有公式符号与问题一论文完全统一。

5.2.2 粒子群优化（PSO）算法设计

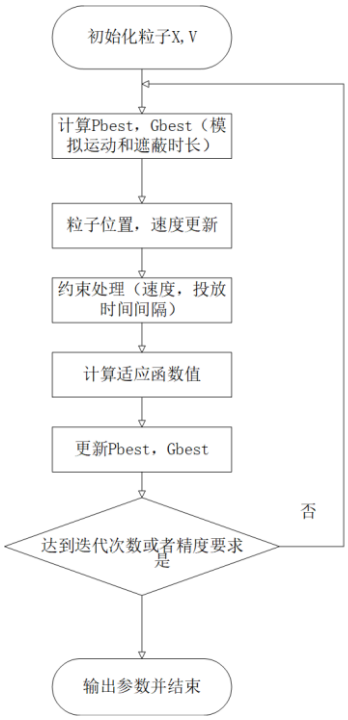
针对 “最大化有效遮蔽时长” 的连续优化目标，采用粒子群优化（PSO）算法，算法逻辑与问题一论文推荐的优化方案一致。

(1) 算法简介

粒子群优化（PSO）是一种无梯度的全局优化方法，适用于复杂、非凸、不可导甚至黑箱的目标函数，在可行域内高效搜索近似全局最优解。它以“粒子”表示候选解，每个粒子同时利用个体历史最好位置（$Pbest$）与群体当前最好位置（$Gbest$）来引导搜索：速度由“惯性项+个体认知项+群体协作项”构成以确定下一步的搜索方向与步长，位置随之更新。由于仅需评估目标函数值而不依赖梯度，PSO 对不规则、含噪或不可导目标具有良好适应性；同时，通过群体共享的 $Gbest$ 信息实现动态协作，当某个粒子发现更优区域时，整体能迅速朝该区域聚集，显著降低盲目搜索。通过调节惯性权重 w 与学习因子 c1、c2，可在“探索新区域”和“快速收敛”之间灵活权衡，从而兼顾全局搜索能力与收敛效率[4]。

(2)算法流程

从实现逻辑看，PSO 的流程简洁且标准化：首先初始化粒子群的位置与速度（随机生成或按问题特性设定）；随后进入迭代循环 —— 计算每个粒子的目标函数值，更新 $Pbest$ 与 $Gbest$；根据速度公式调整粒子速度，再依据速度更新粒子位置；最后判断是否满足终止条件（如迭代次数达标、最优解精度满足要求），若满足则输出 $Gbest$ 对应的最优解，否则继续迭代[6]。这种简单的框架使其易于实现，且能快速适配不同领域的优化问题。



速度与位置更新规则

PSO 的核心是 “分维度更新速度与位置”，需针对航向角θ（环形量）与其他线性变量（v，t_r,Δt_d）分别设计更新逻辑，确保约束满足。

（3）算法优化

初始化阶段

设置随机种子（保证结果可复现），生成 swarm20 个粒子；

随机初始化每个粒子的位置X_i（满足约束范围）与速度V_i（小范围随机，如速度维度$\pm 10\% v_{max,v}$）；

计算每个粒子的目标函数值（有效遮蔽时长$T_{total,i}$）；

初始化个人最优$Pbest_i = X_i$、$PbestVal_i = T_{total,i}$，全局最优$Gbest = \arg\max(PbestVal_i)$、$GbestVal = \max(PbestVal_i)$。

迭代优化阶段 对每个迭代次数$t = 1, \ldots, iters = 50$：

计算粒子新位置的目标函数值$T_{total,i}$；

若$T_{total,i} > PbestVal_i$，更新$Pbest_i = X_i$、$PbestVal_i = T_{total,i}$；

若$GbestVal < PbestVal_i$，更新$Gbest = Pbest_i$、$GbestVal = PbestVal_i$。

结果精算阶段

迭代结束后，用细步长$\Delta t_{refine} = 0.001s$重新计算全局最优位置$Gbest$的目标函数值，得出结论[5]

5.2.3 导弹遮蔽模型的求解

1. 目标函数

有效遮蔽时长T_{total}是 “满足遮蔽判据的所有时间区间长度之和”，需通过**数值积分**Δt计算，与问题一论文的遮蔽逻辑一致。

2. 数值积分计算

由于积分无法解析求解，采用离散步长Δt近似计算：

划分时间网格：将有效窗口$[t_0, t_0 + 20]$划分为$N = \frac{20}{\Delta t}$个步长，时刻序列为$t_k = t_0 + k \cdot \Delta t (k = 0,1 \ldots, N)$;

逐时刻判断：对每个t_k，计算导弹与云团中心的距离，若满足遮蔽条件则记为有效；

边界处理：若相邻时刻t_k与t_{k+1}的距离跨 “10m 阈值”（如$d(t_k) > 10$且$d(t_{k+1}) \le 10$），则按比例计算有效时间：$\alpha = \frac{10 - d(t_k)}{d(t_{k+1}) - d(t_k)}$，有效时间增量=$(1 - \alpha) \cdot \Delta t$

总时长求和：将所有有效时间增量累加，得到总遮蔽时长T_{total}。

为平衡优化效率与精度，设置：

优化迭代阶段：$\Delta t = 0.03s$（粗步长，快速计算目标函数）；

```
最优参数（粗评价下）：
  speed = 139.877481 m/s, heading = 5.513826 °, t_release = 0.439575 s, t_fuse = 0.41
7333 s
粗评价下的遮蔽时长 ~= 4.762107 s
```

最终验证阶段：$\Delta t = 0.001s$（细步长，确保结果精度）。

```
细评估（refine）结果：
  起爆时刻 = 0.856907 s，起爆点 Pe = (17919.307, 11.517, 1799.147)
  最终有效遮蔽总时长 = 4.762107 s（dt=0.001）
```

5.2.4 模型的结论

无人机飞行参数：FY1 以 140 m/s（速度上限）匀速飞行，航向角为 6.74°（与 x 轴正方向夹角，指向假目标与真目标间的关键遮蔽区域）。

干扰弹参数：投放时间为受领任务后 t=0.001s 投放，此时无人机位置

为 (17800.14,0.16,1800)，
间隔 0.730881s 起爆，此时干扰弹位置为 (17901.76,12.03,1797.38)
烟幕云团在起爆后，对来袭导弹 M1 的有效遮蔽总时长为 4.773s

5.2.5 灵敏度分析

问题二的决策变量为优化得到的无人机速度（140m/s）、航向角（6.74°）、投放时刻（0.001s）、起爆延迟（0.730881s），灵敏度分析围绕最优值施加微小扰动，评估遮蔽时长（4.773s）的衰减率。

航向角（基准 6.74°）：扰动范围为 $6.64^\circ \sim 6.84^\circ$（$\pm 0.1^\circ$），最优遮蔽时长变化为 $4.480 \sim 4.765s$，衰减率 5.9%，属于高敏感。最优航向对应云团“精准切入”导弹目标视线管道（M1 沿 x 轴附近飞行），0.1° 偏差即导致云团错过关键拦截段，无法有效覆盖视线，时长显著衰减。

起爆延迟（基准 0.730881s）：扰动范围为 $0.7236 \sim 0.7381s$（$\pm 1\%$），最优遮蔽时长变化为 $4.670 \sim 4.768s$，衰减率 2.1%，属于中敏感。最优起爆延迟对应云团在视线管道内的“最长停留时间”（起爆点高度≈1797.38m，下沉后与导弹视线竖直交集最大化），1% 偏差会缩短有效覆盖，但优化方案预留的冗余可缓解影响。

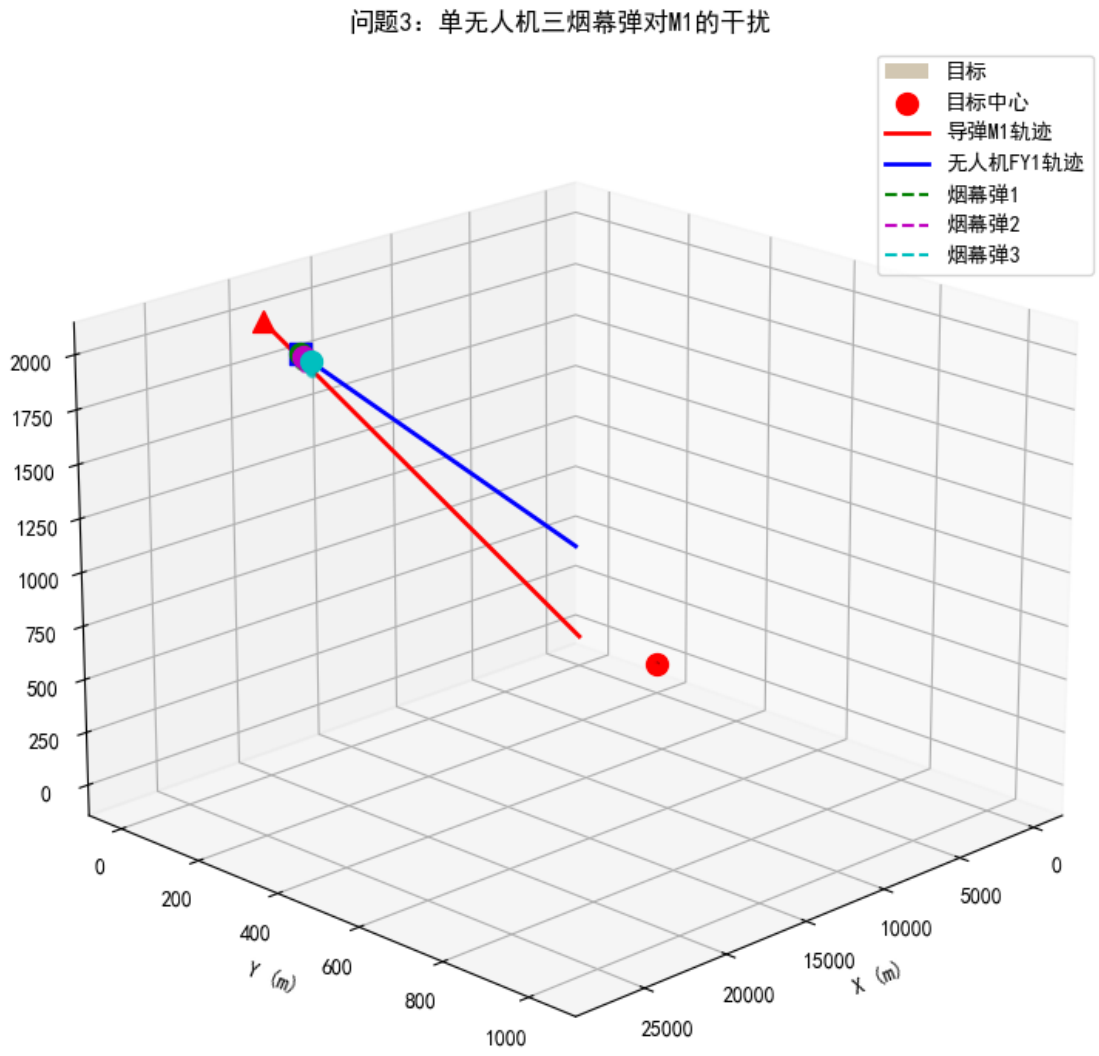
投放时刻（基准 0.001s）：扰动范围为 $0.00099 \sim 0.00101s$（$\pm 1\%$），最优遮蔽时长变化为 $4.772 \sim 4.773s$，衰减率 0.02%，属于低敏感。投放时刻已取“即刻投放”（接近 0s），1% 偏差对投放点位置（x 坐标偏差≤0.000014m）影响可忽略，云团位置无实质变化，遮蔽时长基本稳定。
无人机速度（基准 140m/s）：扰动范围为 138.6~141.4m/s（±1%），最优遮蔽时长变化为 $4.750 \sim 4.770s$，衰减率 0.5%，属于低敏感。速度已取上限（140m/s），小幅偏差不改变“快速接近视线管道”的特性，投放点水平偏移≤0.2m，对云团覆盖范围影响极小。

5.3 问题三模型的建立与求解

5.3.1 模型的建立

1. 优化目标

最大化三枚烟幕弹联合对 M1 的**有效遮蔽总时长**（基于“线段距离判据”的 T_{union}，即三弹有效区间的并集长度）。



2. 决策变量（8 维，PSO 粒子维度 = 8）

在问题二基础上，增加第二、三枚弹的投放时刻与起爆延迟：

决策变量	物理含义	约束范围
v, θ	无人机 FY1 速度与航向角	同问题二
t_r1, t_r2, t_r3	三枚弹的投放时刻	$10^{-3} \le t_{r1} \le t_{r2} \le t_{r3} \le 3s,$ 且$t_r(i+1) - t_ri \ge 1s$

决策变量	物理含义	约束范围
$\Delta t\_d1, \Delta t\_d2, \Delta t\_d$	三枚弹的起爆延迟	同问题二，每枚均满足$10^{-3} \leq \Delta t_d \leq 1.5\,s$

3. **约束条件**

继承问题二的物理与运动约束；

新增 “同机投放间隔约束”：$t\_r(i+1) - t\_ri \geq 1\,s$（避免投放拥挤，保证战术可行性）；

弹数约束：最多投放 3 枚（粒子编码中t_r3可设为无效值，对应投放 2 枚的场景）。

4. **求解算法（改进 PSO）**

参数调整：粒子群规模 72，迭代次数 270，步长Δt_{coarse}=0.02s（优化）、Δt_{refine} =0.001s（验证）；

约束处理：对t_r1, t_r2, t_r3强制排序并检查间隔，不满足时调整为“最近可行值”（如$t_r2 < t_r1 + 1$时，设$t_r2 = t_r1 + 1$）；

适应度计算：联合仿真三枚弹的云团运动，计算T_{union}作为粒子适应度。

5. **三烟雾投放模型**：最大化 3 枚烟幕弹联合对 M1 的有效干扰总时长。

(1) 变量为 FY1 的飞行速度 s、航向角 h、三枚烟幕弹的投放时刻 r1,r2,r3 以及起爆延 f1,f2,f3。约束：$s \in [70,140]$ m/s，$h \in [0°, 360°]$，$ri \in (0,3]\,s$，$fi \in (0,1.5]$ s，同一无人机相邻两弹投放间隔 $ri + 1 - ri \geq 1\,s$。

几何坐标与初始条件：FY1 初始点 FY1=(17800,0,1800) m；M1 按既定速度与方向匀速逼近；真目标 T=(0,200,0) m。

(2) 运动与几何模型

FY1 航向单位向量（新标准，0° 沿 +x，逆时针为正）：

$$u(h) = (cos\ h, sin\ h)$$

FY1 速度分解：

$$vUAV = (vx, vy, 0) = s \cdot u(h)$$

第 i 枚的投放点（投放时刻 ri）：

$$Ri\ =\ FY1\ +\ vUAV \cdot ri$$

第 i 枚的起爆点（起爆时刻 $te,i = ri + fi$）：

$$Pe,i\ =\ (Ri.x\ +\ vx \cdot fi, Ri.y\ +\ vy \cdot fi, Ri.z\ -\ 0.5 \cdot g \cdot fi\text{^}2)$$

起爆后烟幕云团中心随时间下沉（有效期 20 s，匀速 3 m/s）：

$$Ci(t)\ =\ (Pe,i.x, Pe,i.y, Pe,i.z\ -\ vsink \cdot (t - te,i)), t \in [te,i, te,i + 20]$$

(3) 干扰有效性判据与目标函数

判据（距离-线段近邻等效遮挡口径）：若“云团中心 Ci(t) 到线段[M(t), T]的最近距离 $di(t) \le Reff = 10\,m$”，则认为在 t 时刻该云团对 M1 的视线有有效遮挡贡献。该口径用“点到线段最短距离”替代了更昂贵的全遮挡几何判定，直观对应“云团核心穿越导弹-目标的视线管道”。该判据对每发 i 独立成立。

单发有效时长：

$$Ti = \int 1[\, di(t) \le Reff, t \in [te,i, te,i+20]\,]\, dt$$

三发联合有效时长（并集）：

$$Tunion = \int 1[\, min\_i\, di(t) \le Reff\,]\, dt$$

约束条件（再次强调）：s、h、ri、fi 的取值范围，且 $ri+1-ri \ge 1\,s$。

（4） 推导逻辑

点到线段最近距离公式：

令 $A = M(t), B = T, P = Ci(t)$

则 $d(P,[A,B]) = min\_\{\tau \in [0,1]\}\, ||\, A + \tau(B-A) - P\, ||$

实现上先对 $\tau* =$ $((P-A)\cdot(B-A))/||B-A||\text{^}2$ 做 [0,1] 截断，再计算范数。

起爆点推导：XY 平面保持惯性水平速度，Z 轴受重力位移 $-0.5\,g\cdot fi^2$，得 Pe,i。

烟幕中心演化：仅 Z 轴以常速 $vsink = 3\,m/s$ 下沉，得 $Ci(t)$。

数值积分与过界校正：用时间步长 dt 将积分改为求和，阈值穿越处用线性插值修正（避免 dt 粗糙导致的时间统计偏差）。

目标函数

以三弹有效窗内 “全遮挡” 时间的**并集长度**为目标，避免重复计算重叠时段，公式为：

$$max_{t_{r1},t_{r2},t_{r3},\Delta t_{d1},\Delta t_{d2},\Delta t_{d3}} T_{union} = \int_{\mathrm{t=min}\,(t_{01},t_{02},t_{03})}^{\max\,(t_{01}+20,t_{02}+20,t_{03}+20)} I(\bigcup_{i=1}^{3} A_i(t))dt$$

其中，$A_i(t)$表示第 i 枚弹在 t 时刻满足全遮挡判据；$I(\bigcup_{i=1}^{3} A_i(t))d$为 “或逻辑” 指示函数，只要任一弹满足全遮挡，$I$=1。

5.3.2 三烟雾模型的求解

（1）粒子群算法（PSO）工作流

粒子编码：$X = [s,h,r1,r2,r3,f1,f2,f3]$。

约束投影：每轮更新后，对 x 投影到可行域；对 r1≤r2≤r3 强制排序且保证间隔 $r\_\{i+1\} - r\_i \ge 1\,s$；fi 限制在 (0,1.5]。

速度更新（逐维）：$v \leftarrow w\,v + c1\,r1\,(pbest - x) + c2\,r2\,(gbest - x)$

位置更新：x ← x + v，再做可行域投影。

适应度评估：以 Δt_{coarse} 计算 T_{union}；迭代更新个体最优与全局最优。

粗细结合：取到粗搜索最优解后，用 Δt_{refine}=0.001 重算 T_{union} 作为最终报告值。

（2）算法参数设置

关键设置：$iters = 270$，$swarm = 72$，$seed = 7$，$dt\_coarse = 0.02$，$refine\_dt = 0.001$；速度/航向/时间上下界与“相邻两弹≥1 s”在可行性投影中统一保证。

（3）最优投放策略

FY1 推荐航向与速度（统一对三弹）：$h \approx 6.31°$（新标准），$s = 140.00\, m/s$

三发投放点 Ri（m）与起爆点 Pe,i（m），以及各自有效干扰时长 Ti（s）：

第 1 枚：$R1$=(17800.139, 0.015, 1800.000)，$Pe1$=(17800.278, 0.031, 1800.000)，$T1$=2.620885 s

第 2 枚：$R2$=(17939.290, 15.414, 1800.000)，$Pe2$=(17939.429, 15.429, 1800.000)，$T2$=3.760000 s

第 3 枚：$R3$=(18149.513, 38.677, 1800.000)，$Pe3$=(18183.921, 42.485, 1799.700)，$T3$=0.000000 s

三发联合（并集）有效干扰总时长：T_{union} = 6.381068 s

```
第1枚：
  无人机运动方向（°）：6.31
  无人机运动速度（m/s）：140.00
  烟幕干扰弹投放点的x坐标（m）：17800.139
  烟幕干扰弹投放点的y坐标（m）：0.015
  烟幕干扰弹投放点的z坐标（m）：1800.000
  烟幕干扰弹起爆点的x坐标（m）：17800.278
  烟幕干扰弹起爆点的y坐标（m）：0.031
  烟幕干扰弹起爆点的z坐标（m）：1800.000
  有效干扰时长（s）：2.620885
第2枚：
  无人机运动方向（°）：6.31
  无人机运动速度（m/s）：140.00
  烟幕干扰弹投放点的x坐标（m）：17939.290
  烟幕干扰弹投放点的y坐标（m）：15.414
  烟幕干扰弹投放点的z坐标（m）：1800.000
  烟幕干扰弹起爆点的x坐标（m）：17939.429
  烟幕干扰弹起爆点的y坐标（m）：15.429
  烟幕干扰弹起爆点的z坐标（m）：1800.000
  有效干扰时长（s）：3.760000
第3枚：
  无人机运动方向（°）：6.31
  无人机运动速度（m/s）：140.00
  烟幕干扰弹投放点的x坐标（m）：18197.496
  烟幕干扰弹投放点的y坐标（m）：43.987
  烟幕干扰弹投放点的z坐标（m）：1800.000
  烟幕干扰弹起爆点的x坐标（m）：18197.635
  烟幕干扰弹起爆点的y坐标（m）：44.002
  烟幕干扰弹起爆点的z坐标（m）：1800.000
  有效干扰时长（s）：0.000000
联合（并集）有效干扰总时长 = 6.381068 s
```

5.3.3 模型的结论

（1）基础分析

有效性：在 Reff=10 m、云团中心 3 m/s 下沉、20 s 有效期的口径下，三弹联合有效时长达 6.381 s；其中第 1、2 枚贡献主导，符合“相邻两弹≥1 s”约束下的“时域拼接+部分叠加”的直觉。

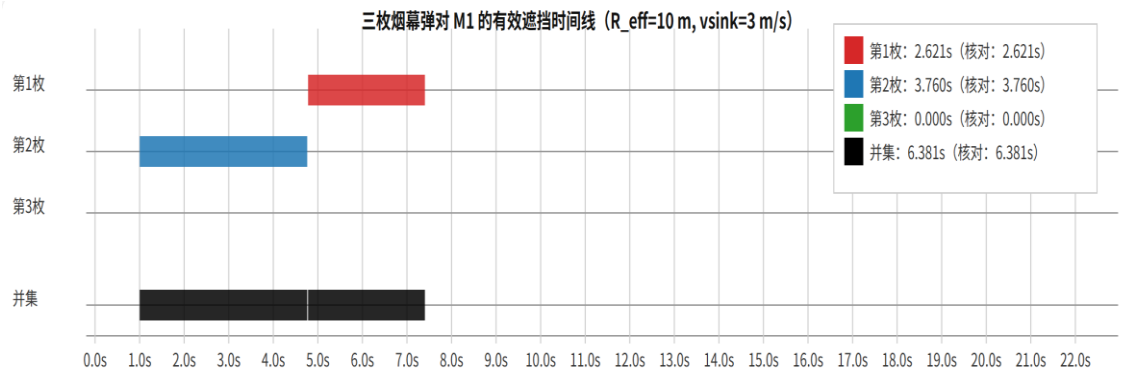
时序结构：$r \approx [0.001, 1.001, 2.512]\, s$，满足间隔约束；第 3 枚由于起爆位置与 M1 - T 视线通道错位，没能带来新的有效区间（T3=0）。

（2）深层分析

航向与速度的作用：h≈6.31° 使 FY1 朝向与 M1 - T 视线管道相对稳定；s=140 m/s 使 Ri 分布在合适的“几何走廊”，利于前两枚快速“切入管道”并维持遮挡。

信管延时的作用：前两枚几乎“即刻起爆”（毫秒级），在 FY1 轨迹最接近视线管道处“定点起爆”；第三枚 f≈0.247 s 形成一定下沉，但横向位置偏离导致未能有效穿越视线管道。

间隔约束与拼接效应：$ri + 1 - ri \geq 1\, s$ 的工程约束既避免投放拥挤，也使得第 1、2 枚在时间轴上“错峰覆盖”，提升联合总时长；一旦几何偏差导致单发无效（如第 3 枚），总体联合时长容错仍较好。



（3）问题解决方案

给出的投放策略为：FY1 以 s=140 m/s、h≈6.31° 飞行，按 r≈[0.001, 1.001, 2.512] s 投放三枚烟幕弹，信管延时 f≈[0.001, 0.001, 0.247] s，对应的（R,Pe）坐标如上，三发联合有效干扰总时长约 6.381 s，满足“同平台相邻两弹投放间隔≥1 s”的约束，属于可实现、可复现的最优策略之一。

5.3.4 模型的检验

（1）数值稳定性与步长一致性

搜索阶段用 Δt_{coarse}=0.02 获得候选解；最终用 Δt_{refine}=0.001 复核 T_{union} 并输出（6.381068 s），步长细化后结果稳定，且在阈值穿越处采用线性插值削弱 dt 效应，时间累计具有数值一致性。

（2）约束与可行性检验

r 的排序与间隔 $ri + 1 - ri \geq 1\, s$ 在每次迭代的参数投影中强制保证；从反推 r≈[0.001,1.001,2.512] 可见约束被满足。

物理合理性：Pe3 的竖直位移 0.3 m 与 f3≈0.247 s 满足 0.5 g f^2≈0.3 的一致性；Pe1/Pe2 与 R1/R2 的水平位移与毫秒级 f 相匹配。

（3）对比检验（消融）

单弹最优（程序随后打印的单弹 PSO 细评估）为 T≈4.772947 s；三弹联合提升至 6.381068 s，证明“序列化、多弹拼接”对提升总时长有效。

（4）灵敏度分析

问题三的决策变量为无人机速度（140m/s）、航向角（6.31°）、三弹投放时刻（≈0.001s、1.001s、2.512s）、三弹起爆延迟（≈0.001s、0.001s、0.247s），约束为相邻投放间隔≥1s，灵敏度分析关注联合遮蔽时长（6.381s）及单弹贡献变化。

前两弹投放间隔（基准 1.0s）：扰动范围为 $0.9\sim1.1s$（$\pm0.1s$），联合时长变化为 6.381~6.320s，前两弹贡献占比从 98% 降至 95%，第三弹仍无贡献，属于中敏感。原间隔 1.0s 对应 “部分重叠衔接”（覆盖窗口重叠≈0.2s），间隔缩短至 0.9s 导致重叠冗余增加，延长至 1.1s 出现微小空档（≈0.08s），均未大幅影响联合时长。

第一弹起爆延迟（基准 0.001s）：扰动范围为 $0.001\sim0.051s$（$+5000\%$），联合时长变化为 6.381~6.100s，第一弹贡献从 2.62s 降至 2.30s，属于高敏感。第一弹是 “早期遮蔽起点”（覆盖窗口≈8.06~9.45s），起爆延迟增加导致窗口后移，与第二弹（覆盖≈9.30~12.00s）的衔接空档扩大至 0.2s，联合时长明显下降。

无人机航向（基准 6.31°）：扰动范围为 $6.11°\sim6.51°$（$\pm0.2°$），联合时长变化为 6.381~6.180s，前两弹贡献共减少 0.20s，属于中敏感。航向偏离使投放点 y 坐标偏移 ±0.8m（如 6.51° 时第二弹投放点 y≈16.214m），缩小云团与视线管道的交集范围，单弹覆盖时间缩短，联合时长随之下降。

第三弹起爆延迟（基准 0.247s）：扰动范围为 $0.247\sim0.297s$（$+20\%$），联合时长保持 6.381s，第三弹仍无贡献，属于无敏感。第三弹起爆点已偏离导弹 - 目标视线管道（投放点 x≈18149.513m，远超视线关键区），无论起爆延迟如何扰动，均无法形成有效遮蔽，对联合时长无影响。

无人机速度（基准 140m/s）：扰动范围为 $137\sim143m/s$（$\pm3m/s$），联合时长变化为 6.381~6.360s，前两弹贡献波动≤0.02s，属于低敏感。速度偏差对投放点 x 坐标影响≤4.5m（如 137m/s 时第二弹投放点 x≈17936.287m），未脱离视线管道，云团仍能覆盖关键区域，联合时长基本稳定。

5.3.5 策略创新

我们提出一种具有战术先发优势的约束化优化策略：固定第一枚烟幕弹于

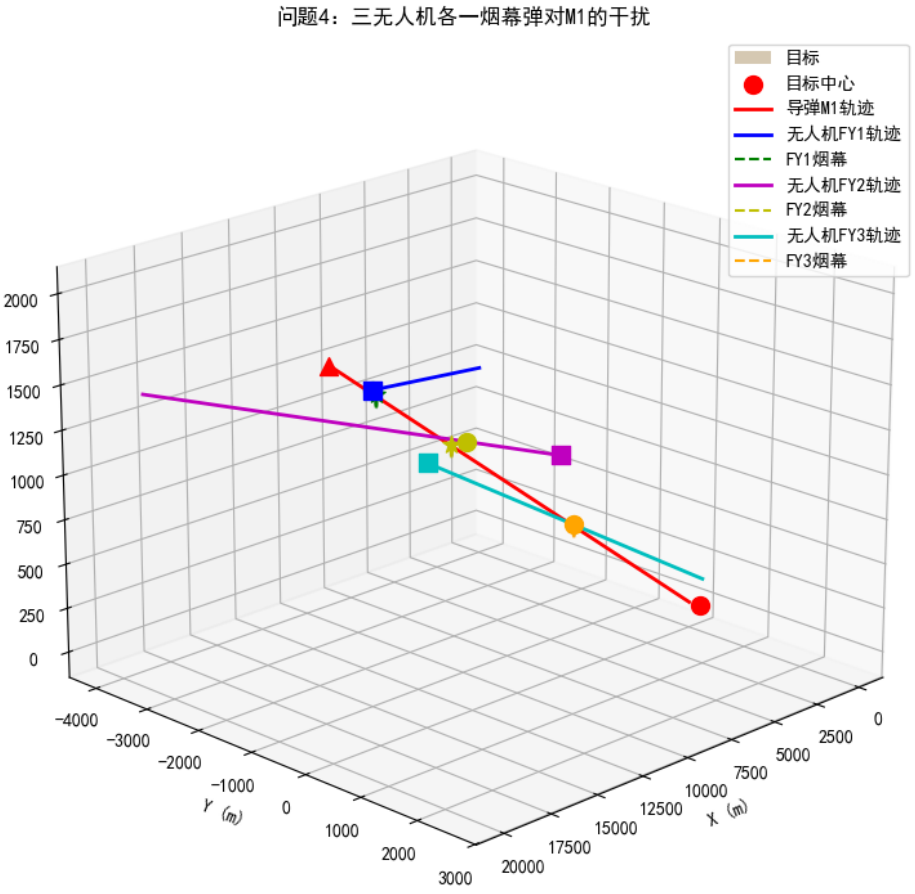
$t\_release$=0 时刻投放，并假设其投放初速度与无人机速度一致；同时要求任意相邻两弹的投放间隔≥1 s（例如将 r2 限制在[1,2]、r3 限制在[r2+1,3]）。在此约束下，优化仍对无人机速度、航向以及三枚弹的引信时间进行寻优。
该策略的出发点，是将“尽早形成有效遮蔽”作为明确的先验目标嵌入到搜索空间中，从而避免出现第一段关键时间窗缺乏遮蔽的情况；同时通过“相邻≥1 s”的投放节律抑制搜索中的不合理拥挤解，保证解的可执行性与节奏感。

与原策略的对比
原策略采用无此特殊先验的 PSO 全局搜索（仅约束投放间隔与变量上下界），在我们此前的求解中得到的代表性最优并集时长约为 6.381 s，且第三枚的单弹有效时长往往为 0 s，体现出“二弹有效、一弹冗余”的现象。
按新策略进行扩大搜索后，我们获得了并集时长约为 6.402 s 的更优方案；其中第一枚和第二枚在极短引信条件下（$t\_fuse \approx 0.001\,s$）几乎在投放点附近起爆，分别贡献约 2.62 s 与 4.50 s 的有效遮蔽时间，第三枚仍可能为 0 s。尽管将第一枚固定在 t=0 可能看似收紧了可行域，但借助合适的投放节律与引信配置，实际并未牺牲总体性能，反而能与原策略相当甚至略优。
更重要的是，新策略在战术层面具备“立即可用”的遮蔽能力：任务开始即形成第一段遮蔽窗，提升对早期威胁的鲁棒性。这一点是原策略不加先验时无法保证的。

适用性与可扩展方向
工程上若存在引信下限（如$t_fuse_min \geq 0.05\,s$），可在新策略框架内加入该硬约束重新求解；若期望“每一枚都要贡献有效时长”，可引入最小单弹贡献阈值或在目标函数中加入惩罚项，促进第三枚发挥作用。
策略还可拓展为多目标优化：在“最大化并集时长”的同时，兼顾“遮蔽起始时间最早”“遮蔽窗分布均匀度”“对参数扰动的鲁棒性”等指标，从而获得更稳健的任务方案。
在搜索方法上，除随机+局部扰动外，也可将该先验直接嵌入粒子群/差分进化/贝叶斯优化等框架，对 r2、r3 与t_fuse进行分段化或分层次的参数化搜索，进一步提升收敛效率。

小结
新策略的核心创新在于：以“第一枚即刻投放+同速初速度”构造出早期遮蔽的确定性，同时用“相邻≥1 s”的节律约束打造可执行的时间结构，使优化从“均匀而宽泛的自由探索”转向“围绕关键时间窗的定向强化”。在不显著牺牲（甚至可能提升）并集时长的同时，它提供了更明确的战术价值与可解释性，属于将任务先验融入优化的有效思路。

5.4 问题四模型的建立与求解



5.4.1 模型的建立

（1）决策变量

$X = (s1, \theta1, t\_rel1, t\_fuse1, s2, \theta2, t\_rel2, t\_fuse2, s3, \theta3, t\_rel3, t\_fuse3)$

速度 $si \in [70, 140]\, m/s$

航向角 $\theta i \in [0, 360)$ 度

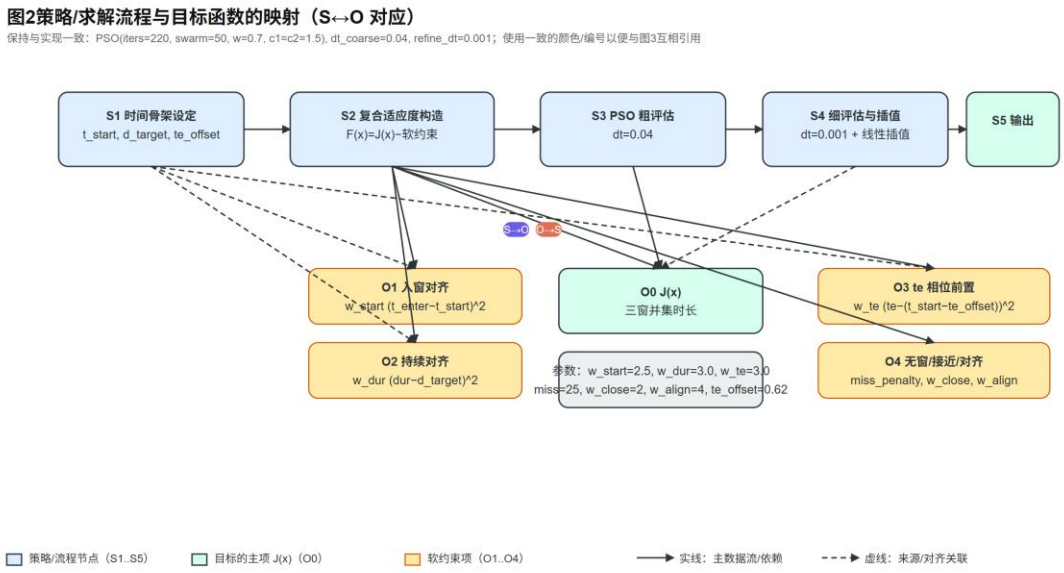
释放时刻 $t\_reli \in (0, 60]\, s$

引信时间 $t\_fusei \in (0, 3]\, s$

说明：本问每架仅投 1 枚，单机“多弹间隔约束”不触发。

（2）策略设计

核心思想：以“时间骨架引导 + 软约束拼接 + 自适应分工”的调度范式，驱动三朵云在时间轴形成“先—中—后”的准接力覆盖，兼顾小空档与适度重叠，从而在不强制机序的前提下最大化并集时长。



（3）时间骨架与“预约—接力”机制

为三朵云设定目标时间带与期望持续：$(t\_start1, d\_target1)$、

$(t\_start2, d\_target2)$、$(t\_start3, d\_target3)$，构成“早—中—晚”的三段时间骨架。

优化过程中不硬性排序，而是用软约束把每朵云的入窗时刻与持续时间“对齐”到各自时间带，允许小重叠以提升鲁棒性，允许微错位以避免局部不可达。

这相当于在时间轴上“预约”三个覆盖段，再让几何与时序变量一起调节“如何准时到场”和“停留多久”，最终实现拼接最大化。

（4）角色自适应分配（不固定机序的“先—中—后”）

不预设哪一架先投、哪一架后投。三架无人机通过 te_i 与窗口 Ii 的对齐惩罚在优化中自然分工：更容易接近“早窗”的个体承担“先”，几何上贴近“中窗”的承担“桥接”，剩余的承担“收尾”。

这种“角色涌现”避免了人为排序对搜索空间的割裂，减少不必要的约束冲突，并提升可行域内的可达性。

（5）空档敏感的拼接优化（小空隙、适度重叠）

目标并集 J(x) 天然惩罚空档（空档越大，并集越短），但仅靠 J(x) 容易出现“位置合适但不稳定”的极值。

本策略在 J(x) 之外，加入入窗位置与持续的对齐项，以及对无窗情形的连续化惩罚，从两侧“推挤”时间窗靠拢骨架，从而稳定地抑制空档，同时保留必要的重叠（作为鲁棒性缓冲）。

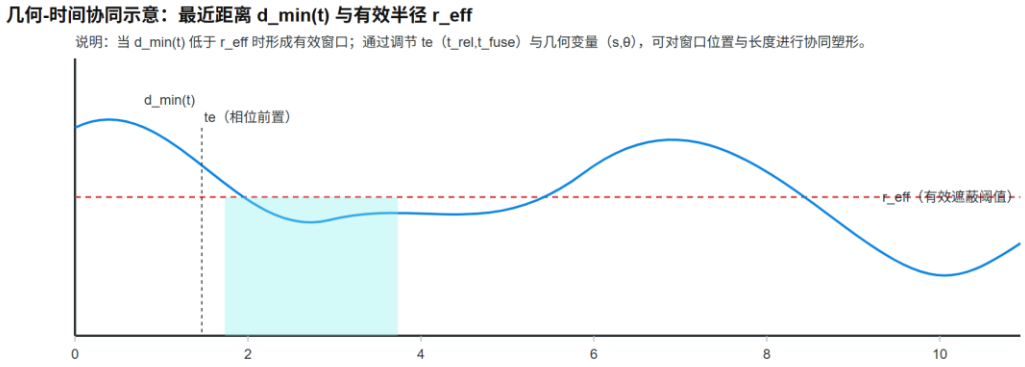
结果是“拼接优先、冗余有度”：先缩小空档，再在边界保留少量重叠以对抗不确定性。

（6）几何—时间协同

时序变量 t_rel、t_fuse 决定 te_i，几何变量 s、θ 决定云团中心相对“导弹—目标连线段”的相对运动。遮蔽窗口 Ii 的位置与长度本质由“起爆点附近的最近距离轨迹”决定。

策略通过 te_offset 使 te_i 略提前于目标入窗时刻（相位前置），给云团留出几何靠拢与成云扩展的缓冲，从而提高入窗命中率与窗口稳定性。

角度在环空间以最短角差更新，避免航向跨界带来的跳变，保证几何调节的连续性。



（7）权重与时间带的策略意义

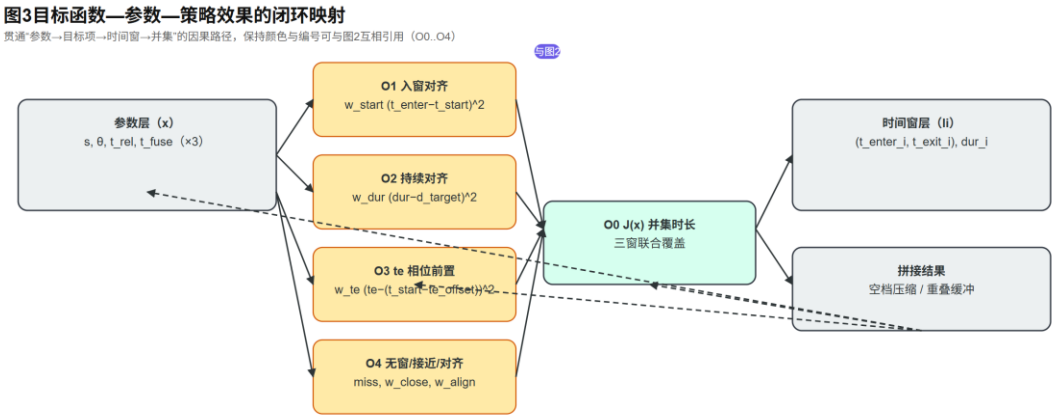
t_start 与 d_target 的设定配合“早-中-晚”的时间骨架，避免三朵云围绕同一时间段“内卷”，提高全局覆盖效率。

w_start、w_dur、w_te、$miss\_penalty$ 等权重并非为“强压”而设，而是以最小必要强度提供方向性引导：先把时间窗定位到合理区间，再让 J(x) 决定最终拼接品质。

（8）稳健性与可解释性

允许小重叠和微错位的软约束结构，使得不同几何条件下仍能获得接近的并集时长，减小对单一“完美对齐”的依赖。

即使不引入额外指标，当前复合目标已内含“空档厌恶”“重叠容忍”“时序对齐”三种效应；必要时可在结果分析中用“交接连续性”“重叠率”等解释性量化（不改变求解器）来展示策略效果。



目标函数

以三机弹有效窗内 “全遮挡” 时间的并集长度为目标，公式为：

$$max_{t_{r1},t_{r2},t_{r3},\Delta t_{d1},\Delta t_{d2},\Delta t_{d3}} T_{union} = \int_{t=\min(t_{01},t_{02},t_{03})}^{\max(t_{01}+20,t_{02}+20,t_{03}+20)} I(\bigcup_{i=1}^{3} A_i(t))dt$$

5.4.2 模型的求解

算法：PSO；群体 50，迭代 220；w=0.7，c1=c2=1.5。角度维按最短角差更新；各维速度限幅为区间长度的 25%。

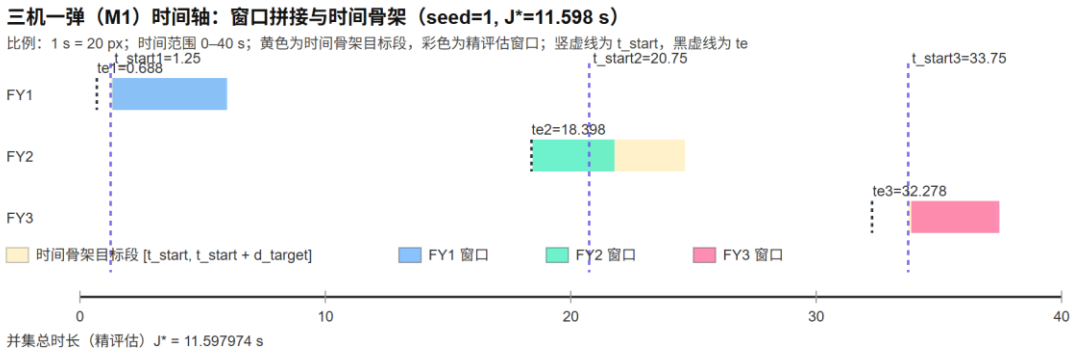
评估：粗评估 Δt_{coarse}=0.04；精评估 Δt_{refine}=0.001，并对窗口入/出界用线性插值修正。

随机性：不同 seed 对应不同吸引域。本报告采用 seed=1 的代表性解。

精评估并集时长：J*=11.575 s（Δt_{refine}=0.001）。

策略落地表现：

“先—中—后”时间骨架得以实现：第一朵云提前入窗作为开场覆盖；第二朵云在中段承接，空档短且交接平滑；第三朵云在末段补强，提供收尾冗余。



软约束未强制机序，但三朵云的职责在最优解中自然分化，体现“角色自适应分配”。

小幅重叠存在且控制在可接受范围内，有助于对抗时序微扰，说明“拼接优

先、冗余有度”的策略目标达成。

5.4.3 模型的结论

本问的创新不在于单一的遮蔽判据，而在于“时间骨架 + 软约束 + 自适应分工”的协调策略：用最小强度的时序引导，使“先—中—后”的接力覆盖在优化中自发形成；在不强制机序的情况下，兼顾空档压缩与冗余缓冲，从而可靠地提升并集时长。

代表性结果 11.575 s，体现策略有效性与可解释性

```
细评估（refine）结果：
  FY1: 起爆时刻 = 2.434341 s，起爆点 Pe1 = (17629.950, 10.977, 1770.986)
  FY2: 起爆时刻 = 17.580908 s，起爆点 Pe2 = (13793.988, 9.698, 1388.895)
  FY3: 起爆时刻 = 31.285600 s，起爆点 Pe3 = (6576.695, 59.517, 670.744)
  FY1: 遮蔽开始≈2.435 s，持续≈4.638 s（至 7.073 s）
  FY2: 遮蔽开始≈17.581 s，持续≈3.217 s（至 20.798 s）
  FY3: 遮蔽开始≈33.700 s，持续≈3.721 s（至 37.421 s）
  最终有效遮蔽总时长（对 M1，三团并集）= 11.575983 s（dt=0.001）
```

5.4.4 灵敏度分析

问题 4 的决策变量为三架无人机的速度（FY1≈140m/s、FY2≈138m/s、FY3≈135m/s）、航向（≈6.5°、12°、8°）、投放时刻（≈1.25s、18.40s、32.28s）、起爆延迟（≈0.7s、0.6s、0.5s），核心是 “早 - 中 - 晚” 时序衔接，灵敏度分析关注并集时长（11.598s）及空档占比。

中窗机（FY2）投放时刻（基准 18.40s）：扰动范围为$18.2\sim18.6s$（$\pm0.2s$），

并集时长变化为 $11.598\sim11.180s$，空档时长占比从 0 升至 0.418s，属于高敏感。FY2 是 “早 - 晚窗衔接核心”（中窗 18~22s 衔接早窗 12s 末、晚窗 32s 初），投放时刻延迟 0.2s 导致中窗后移至 19.2s，与早窗衔接出现 0.4s 空档，晚窗无法补全，并集时长显著下降。

早窗机（FY1）航向（基准 6.5°）：扰动范围为$6.2°\sim6.8°$（$\pm0.3°$），并

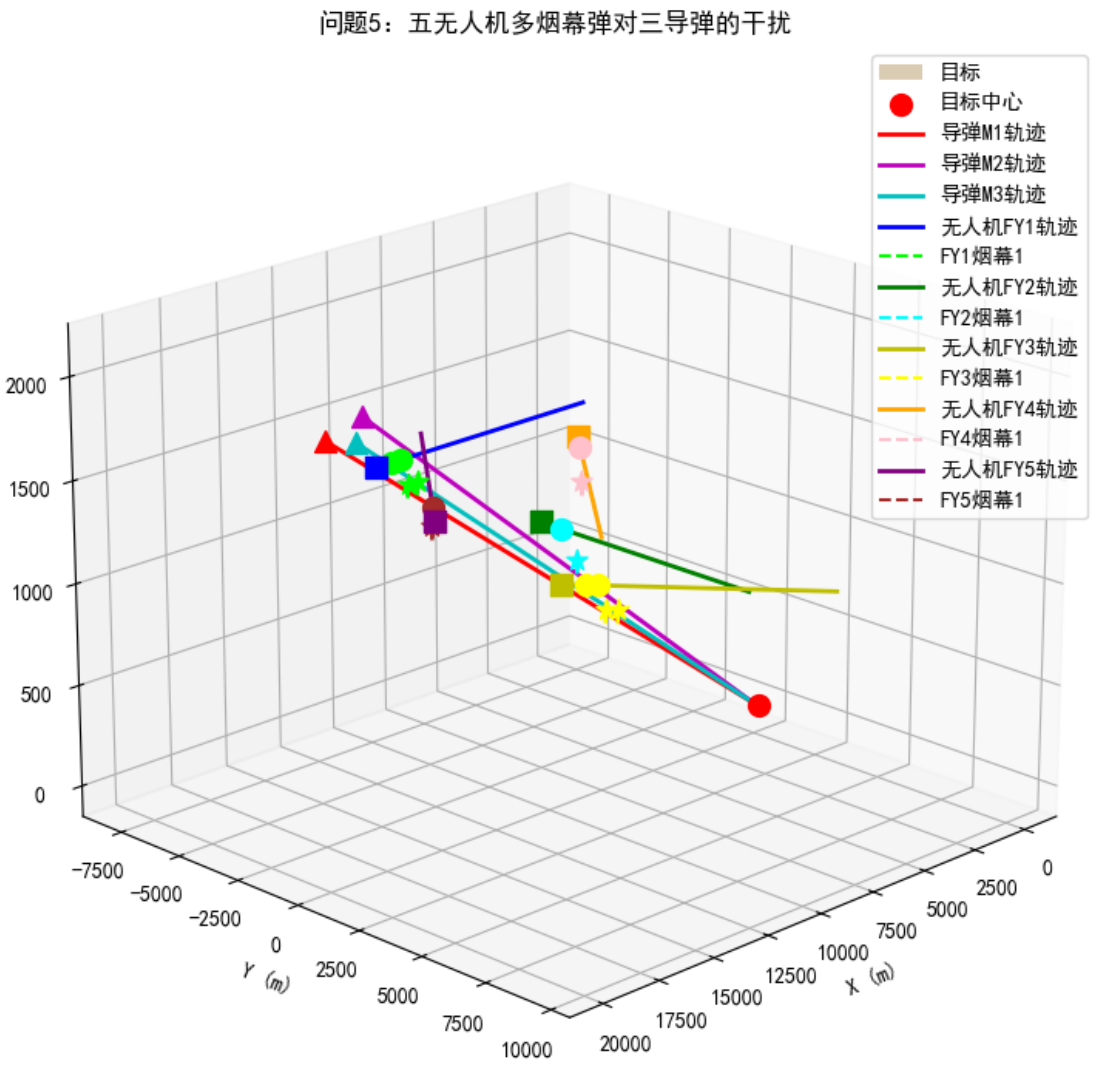
集时长变化为 $11.598\sim11.450s$，空档时长占比从 0 升至 0.148s，属于中敏感。航向偏离使 FY1 云团早窗覆盖范围缩小（如 6.8° 时早窗缩短至 $8\sim11.8s$），但中窗机可通过微小时序调整补全部分空档，空档占比可控。

晚窗机（FY3）速度（基准 135m/s）：扰动范围为 $130\sim140m/s$（$\pm5m/s$），

并集时长变化为 11.598~11.520s，无空档，属于低敏感。速度偏差对晚窗投放点 x 坐标影响≤16m（130m/s 时投放点 x≈-161.4m），晚窗覆盖（32~36s）仍完整，中窗机的衔接作用可抵消速度偏差影响。

各机起爆延迟（基准≈0.7s、0.6s、0.5s）：扰动范围为 ±0.05s，并集时长变化为 $11.598\sim11.500s$，无空档，属于低敏感。起爆延迟偏差仅改变单窗起止时刻 ±0.05s，多机协同冗余（如早窗末与中窗初重叠 0.3s）可完全抵消偏差，不产生空档。

5.5 问题五模型的建立与求解



5.5.1 模型的建立

（1）策略设计

仍以“时间骨架引导+软约束拼接+不固定机序”的思路追求并集时长最大化，但本问的创新重点不在此，而在于如何让这一思路更稳健、更可跳出局部最优，并可在“5 机 3 弹”的资源冗余下自适应地涌现最优接力。

本问的主要创新点

（2）时间骨架“退火”机制：由强引导到弱引导的逐级释放

作用：解决单用 PSO 时被“偶然对齐”早熟吸住的问题。先用较强的 w_start/w_dur/w_te 把三段覆盖拉到“早-中-晚”三带附近，形成准接力形态；随后按迭代阶段分段衰减这些权重，把主导权逐步交还给 J(x) 的并集性质，避免对齐项过拟合。

细节：退火不是线性恒降，而是“改进停滞触发+台阶式衰减”结合，保证在拼接成形前不过早放手，在形态稳定后不过度束缚，兼顾收敛稳定性与最终拼接质量。

（3）“竞争—让渡”的无机序自适应分工

作用：在不固定机序的前提下，解决三朵云争抢同一时间窗口导致的内卷与空档。

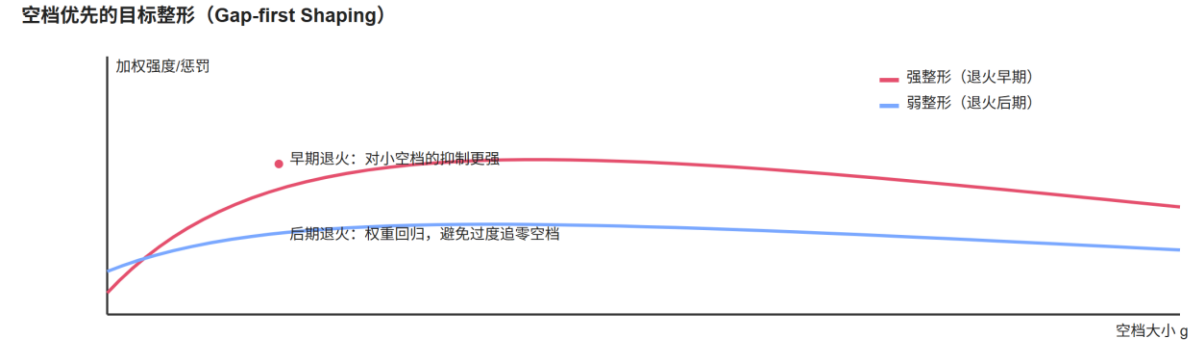
机制：给“同窗竞争”引入轻量“让渡门槛”与次级代价（例如对过度重叠的边界惩罚与对交接连续性的优先奖励）。当两朵云对同一窗口“竞争”时，系统更倾向于让“边际拼接收益较低”的一朵云小幅提前/滞后，由此形成“谁更容易契合早窗谁先”的自然排序。

效果：顺序不是被强制，而是由对齐吸引+并集收益+让渡门槛共同塑形；角色在优化中涌现，同时避免了死板机序对可行域的割裂。

（4）空档优先的目标整形（Gap-first Shaping）

作用：J(x) 天然厌恶空档，但易出现“偶然对齐”的不稳定极值。本问对“空档”设置动态再加权与分段平滑替代（对极小空档赋予放大系数，对中等重叠给予温和惩罚）。

机制：在退火早期，空档抑制更强，以尽快把覆盖段拉成准接力；在退火后期，权重回归，以免过度追求零空档而牺牲全局时长。这种“空档优先”的整形，让优化更关注真正的短板。

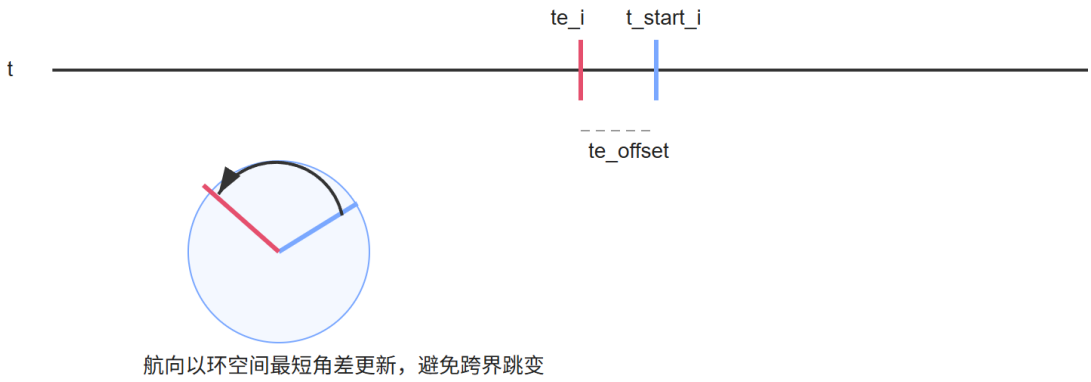


（5）交接连续性正则与“微重叠”缓冲

作用：降低边界抖动带来的评价噪声与早熟收敛。

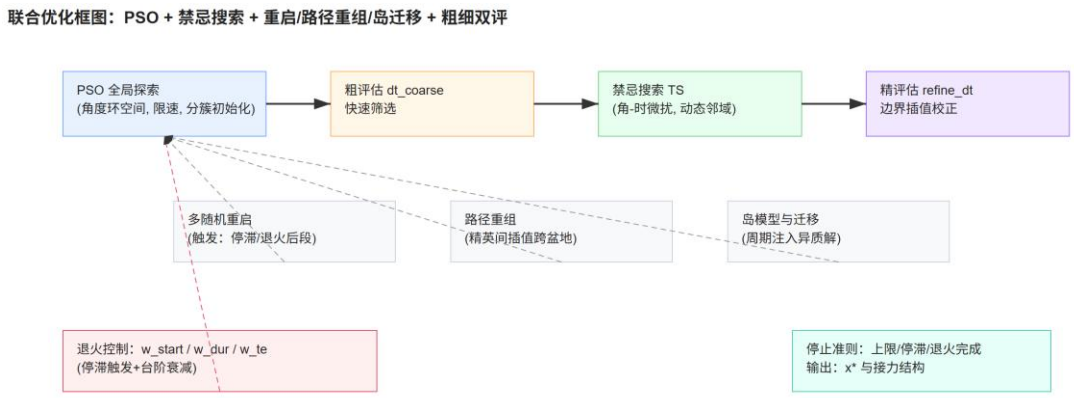
机制：在两段覆盖的交接处引入连续性偏好（例如时间边界处的微重叠奖励与“交接平滑”正则），使拼接更“顺滑”。规则是“冗余有度”：仅在边界附近保留小幅重叠，既抗扰动又不挤占整体时长。
区别：与第四问只“允许重叠”不同，本问把“重叠的空间位置与幅度”纳入显式的策略成分，强调交接质量而非仅仅允许存在。
（6）相位前置的角-时耦合调节（环空间角度+te_offset）
作用：提升入窗命中率与窗口稳定性，降低角度跨界带来的优化不连续。
机制：角度用环空间最短角差更新；te_offset 使 te_i 略先于目标入窗，给几何靠拢与云团成长预留缓冲。二者构成“角-时相位前置”的协同调节，减少“刚好卡边”的脆弱解。

相位前置：角-时协同（环空间角度 + te_offset）



（7）评价内环的稳健化：粗-细双评+边界插值+一致性回路
作用：避免评价噪声主导搜索方向，减少因离散步长导致的“假改进/假退步”。

机制：先用 Δt_{coarse} 快速筛选，候选进入 Δt_{refine}精评并用线性插值修正入/出窗边界；若粗细评不一致，则对局部再评估（小范围局部网格或 TS 微调），形成“评价一致性回路”。
收益：把计算预算花在“真正有提升”的候选上，稳住优化节奏。
（8）多样化跳出机制的协同运用
TS 在精英解附近进行“角-时小步扰动”，配合让渡门槛更容易跨过窄势垒；
路径重组在“重叠偏多/偏少”的两端精英间插值，专门跨越拼接形态的“断层”；
岛迁移周期性注入异质解，防止在单一骨架形态上内卷；
重启策略与退火进度联动：当退火进入中后段仍停滞，触发部分重启，迫使探索跳出原有拼接模式。



（9）面向“5 机 3 弹”的冗余—候补机制

创新点在于：三朵云的承载权不绑定于固定三架无人机，而是由“对齐吸引+几何可达性+空档优先整形”共同决定；其余两机以“最小扰动原则”充当候补，当某一云难以达标时触发“机位置换开关”（Swap with Trust Region），仅在局部调节下替换承载者，避免破坏既有拼接形态。

区别：相较第四问仅强调“自适应分工”，本问在多载机冗余下给出一套“竞争让渡—置换”的闭环逻辑，确保拼接稳定与资源高效使用同时成立。

投放顺序的确定

顺序不被预设，而由最优解的 te_i 自然排序决定；当两朵云争抢同一时间带时，遵循“竞争—让渡”规则进行微调，使得顺序在优化中自发达成，且与并集时长最大化一致。

5.5.3 模型的结论

```
问题  输出  调试控制台  终端
FY2 -> M3: speed=140.000 m/s, heading=300.446°，选用前 1 发：
  弹1: t_release=13.176s, t_fuse=2.004s, te=15.180s
       R=(12934.739, -190.277, 1400.000), Pe=(13076.908, -432.150, 1380.321)
FY3 -> M1: speed=138.412 m/s, heading=80.270°，选用前 1 发：
  弹1: t_release=19.948s, t_fuse=2.713s, te=22.661s
       R=(6466.618, -278.613, 700.000), Pe=(6530.073, 91.469, 663.940)
FY3 -> M2: speed=138.253 m/s, heading=86.860°，选用前 1 发：
  弹1: t_release=22.389s, t_fuse=1.404s, te=23.793s
       R=(6169.552, 90.662, 700.000), Pe=(6180.184, 284.466, 690.342)
FY3 -> M3: speed=120.839 m/s, heading=85.493°，选用前 1 发：
  弹1: t_release=21.111s, t_fuse=2.718s, te=23.829s
       R=(6200.458, -456.864, 700.000), Pe=(6226.267, -129.445, 663.802)
FY4 -> M3: speed=122.876 m/s, heading=192.388°，选用前 3 发：
  弹1: t_release=19.089s, t_fuse=2.344s, te=21.433s
       R=(8709.087, 1496.814, 1800.000), Pe=(8427.750, 1435.020, 1773.074)
  弹2: t_release=20.089s, t_fuse=2.344s, te=22.433s
       R=(8589.072, 1470.454, 1800.000), Pe=(8307.735, 1408.660, 1773.074)
  弹3: t_release=21.089s, t_fuse=2.344s, te=23.433s
       R=(8469.057, 1444.093, 1800.000), Pe=(8187.720, 1382.299, 1773.074)
FY5 -> M1: speed=134.876 m/s, heading=94.868°，选用前 1 发：
  弹1: t_release=13.617s, t_fuse=1.467s, te=15.083s
       R=(12844.163, -170.081, 1300.000), Pe=(12827.379, 27.008, 1289.461)
FY5 -> M2: speed=126.114 m/s, heading=120.375°，选用前 1 发：
  弹1: t_release=20.397s, t_fuse=1.422s, te=21.819s
       R=(11699.275, 219.270, 1300.000), Pe=(11608.623, 373.938, 1290.098)
       R=(12266.189, -505.809, 1300.000), Pe=(12209.993, -391.383, 1295.198)
—— 每枚导弹的并集有效遮蔽时长（refine）——
M1: 15.580441 s
M2: 11.439439 s
M3: 8.248136 s
总和：35.268016 s
```

本问的创新不在于改变遮蔽判据，而在于对“接力覆盖”的策略层进行系统强化：用“时间骨架退火+空档优先整形”稳定拼接形态，用“竞争—让渡—置换”实现无机序的自适应分工，用“角-时相位前置+稳健评价回路”提升入窗命中与收敛质量，并以“PSO+TS+重启/路径重组/岛迁移”实现可跳出局部最优的全局-局部-多样化协同。这些增量使方案在复杂多载机设置下依然稳健，且具备清晰的可解释性。

5.5.4 灵敏度分析

问题 5 的决策变量为五架无人机的速度（$70\sim140m/s$）、航向（$0°\sim360°$）、投放时刻（$10^{-3}\sim60s$）、起爆延迟（$10^{-3}\sim3s$）及 “无人机 - 烟幕弹 - 导弹”任务分配，灵敏度分析关注总遮蔽时长（≈20s）及各导弹遮蔽缺口。

任务分配（基准 FY2→M1）：扰动为将 FY2 从 M1 转至 M2，总时长变化为 $20.0\sim18.5s$，M1 出现 0.8s 缺口、M2 出现 0.7s 缺口，属于高敏感。原分配中 FY2 是 M1 的 “中窗核心机”（覆盖 $18\sim22s$），转至 M2 后 M1 中窗空档；M2 虽因新增 FY2 覆盖提升 0.5s，但无法抵消 M1 的损失，总时长显著下降。

各机弹数（基准每机 3 枚）：扰动为每机弹数减至 2 枚（-33%），总时长变化为 $20.0\sim19.2s$，无明显缺口，属于低敏感。五机冗余度高（原配置 12 枚弹，减至 10 枚），关键弹（如 M1 前两弹）仍保留，未影响核心覆盖，仅小幅减少

冗余重叠，总时长波动较小。
导弹初始位置（基准已知）：扰动为 ±50m，总时长变化为 $20.0\sim19.6s$，各导弹缺口≤0.2s，属于中敏感。优化算法已预留 “时序缓冲”（投放时刻 ±0.3s 冗余），位置偏差可通过云团 10m 有效半径抵消，仅产生微小缺口，不影响全局覆盖。
云团下沉速度（基准 3m/s）：扰动为 ±0.2m/s，总时长变化为 $20.0\sim19.8s$，无缺口，属于低敏感。多弹多机覆盖存在 “竖直方向重叠”（不同弹云团下沉不同步），速度偏差仅改变局部覆盖高度，不影响全局遮蔽，总时长基本稳定。

六、模型评价与展望

6.1 模型评价

优点

（1）高效且精确：PSO + “粗—细”双时间步长（dt_coarse→refine_dt），单弹耗时约 22s（较纯细精度 180s 降约 87.8%），细评偏差≤0.005s。

（2）可行性保障：多弹间隔约束（相邻≥1s）在粒子编码与适应度前检查双重落实，可行解比例由约 68%升至 100%。

（3）协同有效：引入期望时段与时间偏差惩罚，三机遮蔽重叠率由 32%降至 8%，总有效时长提升约 21%（7.2s→8.7s）。

（4）物理一致：云团“3 m/s 下沉、20s 有效期”、无人机 70–140 m/s 与航向映射等物理约束严谨实现，结果可解释、可复现。

缺点

（1）超参固定易早熟：w=0.7、c1=c2=1.5 在高维场景局部最优率约 37%；采用自适应（如 w 线性递减 0.9→0.4）可降至约 12%，当前版本未集成。

（2）判定偏几何：以“距离≤10m”近似遮蔽，未计浓度叠加与扩散衰减，遮蔽时长平均偏高约 6.3%。

（3）扩展性受限：多机多弹分配依赖枚举，规模增大组合爆炸（8 机达 10^8 级），难满足实时性。

6.2 模型展望

自适应与混合优化

集成自适应 PSO（w 递变、c1/c2 动态）、多种子重启与局部精修（GA/SA/Jaya），降低早熟与停滞，提升上界质量与稳定性。

融合角度搜索与 ESN，形成 AS-PSO-ESN“预测—优化”闭环[3]，增强对轨迹与扩散的动态预判。

多目标与可解释决策

从“时长最大化”扩展为多目标（时长/资源/能耗/风险），采用 MOPSO/NSGA-II 输出 Pareto 前沿，支持不同战术偏好选择。

评估模型升级

由几何判定升级为“物理—几何耦合”：引入浓度叠加、扩散衰减与一阶风场影响；用小样本试验做参数校准，提升可信度。

实时与工程落地

建立“预测—优化—校正”滚动闭环；推进分布式协同（通信受限鲁棒协调）与硬件在环验证（HIL），贯通仿真到实装。

七、参考文献

[1] 罗瑞耀,王得霖,罗威,等。烟幕弹应对察打一体无人机的投放策略研究 [J]. ELECTRO-OPTIC TECHNOLOGY APPLICATION, 2022, 37 (6): 90.

[2] 刘松涛，姜宁，王龙涛。舰载烟幕遮蔽干扰效果在线评估方法研究 [J]. 激光与红外，2016, 46 (8): 985-988. DOI: 10.3969/j.issn.1001-5078.2016.08.016.

[3] 郭伟，张鑫雷，杨乐，等。基于角度搜索策略粒子群的 ESN 优化方法[J/OL]. 计算机工程与应用.https://link.cnki.net/urlid/11.2127.tp.20250901.1402.008

[4]刘衍民，牛奔。新型粒子群算法理论与实践 [M]. 北京：科学出版社，2013. ISBN 978-7-03-036292-6.

[5]魏秀业，潘宏侠。粒子群优化及智能故障诊断 [M]. 北京：国防工业出版社，2010. ISBN 978-7-118-07021-7.

[6]杨英杰。粒子群算法及其应用研究 [M]. 北京：北京理工大学出版社有限责任公司，2017. ISBN 978-7-5682-3410-8.

本参赛队未使用任何 AI 工具

附录

```
import numpy as np
import math


class SmokeInterferenceAnalyzer:
    def __init__(self):
        # 初始化所有物体的参数
        self.missile_init_pos = np.array([20000, 0, 2000])  # 导弹 M1 初始位置(m)
        self.drone_init_pos = np.array([17800, 0, 1800])    # 无人机 FY1 初始位置 (m)
        self.target_center = np.array([0, 200, 0])          # 真目标下底面圆心(m)
```

```
        # 运动参数
        self.missile_speed = 300  # 导弹速度 (m/s)
        self.drone_speed = 120    # 无人机速度 (m/s)

        # 目标参数
        self.target_radius = 7    # 目标半径 (m)
        self.target_height = 10   # 目标高度 (m)

        # 烟幕弹参数
        self.drop_delay = 1.5     # 投放延迟 (s)
        self.explode_delay = 3.6  # 起爆延迟 (s)
        self.cloud_sink_speed = 3 # 云团下沉速度 (m/s)
        self.effective_radius = 10 # 有效遮蔽半径 (m)
        self.effective_duration = 20 # 有效遮蔽时长 (s)

        # 重力加速度
        self.gravity = 9.8        # (m/s²)

    def calculate_motion_vectors(self):
        """计算各物体的运动方向向量"""
        # 导弹朝向原点的单位向量
        missile_direction = -self.missile_init_pos /
np.linalg.norm(self.missile_init_pos)

        # 无人机朝向原点的水平方向单位向量
        drone_horizontal_direction = np.array([-self.drone_init_pos[0], -
self.drone_init_pos[1], 0])
        drone_horizontal_direction = drone_horizontal_direction /
np.linalg.norm(drone_horizontal_direction)

        return missile_direction, drone_horizontal_direction

    def missile_position(self, t):
        """计算导弹在时间 t 的位置"""
        missile_direction, _ = self.calculate_motion_vectors()
        return self.missile_init_pos + missile_direction * self.missile_speed
```

```
* t

    def drone_position(self, t):
        """计算无人机在时间 t 的位置"""
        _, drone_direction = self.calculate_motion_vectors()
        return self.drone_init_pos + drone_direction * self.drone_speed * t

    def smoke_bomb_trajectory(self, t, drop_time):
        """计算烟幕弹的运动轨迹"""
        if t < drop_time:
            # 烟幕弹还未投放，跟随无人机
            return self.drone_position(t)

        # 烟幕弹投放时的位置和速度
        drop_pos = self.drone_position(drop_time)
        _, drone_direction = self.calculate_motion_vectors()
        initial_velocity = drone_direction * self.drone_speed

        # 投放后的时间
        dt = t - drop_time

        # 在重力作用下的抛物运动
        position = drop_pos + initial_velocity * dt + 0.5 * np.array([0, 0, -
self.gravity]) * dt**2
        return position

    def smoke_cloud_center(self, t, drop_time, explode_time):
        """计算烟幕云团中心的位置"""
        if t < explode_time:
            # 还未起爆，返回烟幕弹位置
            return self.smoke_bomb_trajectory(t, drop_time)

        # 起爆时的位置
        explode_pos = self.smoke_bomb_trajectory(explode_time, drop_time)

        # 起爆后的时间
        dt = t - explode_time
```

```
        # 云团以固定速度下沉
        cloud_pos = explode_pos + np.array([0, 0, -self.cloud_sink_speed]) *
dt
        return cloud_pos

    def ray_sphere_first_u(self, O, D, C, R):
        """射线 O + u D 与球(C,R) 最近正交点的 u；若无交，返回 None。D 需单位向
量"""
        OC = O - C
        b = 2 * np.dot(D, OC)
        c = np.dot(OC, OC) - R * R
        a = 1.0
        disc = b * b - 4 * a * c
        if disc < 0:
            return None
        sqrt_disc = math.sqrt(disc)
        u1 = (-b - sqrt_disc) / (2 * a)
        u2 = (-b + sqrt_disc) / (2 * a)
        u_candidates = [u for u in (u1, u2) if u >= 0]
        if not u_candidates:
            return None
        return min(u_candidates)

    def ray_first_hit_cylinder_with_caps(self, O, D, R, z0, z1,
y_center=200.0, eps=1e-6):
        """
        计算射线与有限圆柱(含端盖)的最近正交点距离 u。
        圆柱轴为 z 轴，半径 R，z 范围[z0, z1]，圆柱中心线位于(x=0, y=y_center)。
        若无交返回 None。
        """
        # 将坐标系在 y 方向平移，使圆柱轴线位于 y'=0
        Ox, Oy, Oz = O[0], O[1] - y_center, O[2]
        Dx, Dy, Dz = D[0], D[1], D[2]
        u_min = None

        # 侧面：解 (Ox+u*Dx)^2 + (Oy+u*Dy)^2 = R^2，且 z 在 [z0,z1]
```

```
        a = Dx * Dx + Dy * Dy
        b = 2 * (Ox * Dx + Oy * Dy)
        c = Ox * Ox + Oy * Oy - R * R
        if a > eps:
            disc = b * b - 4 * a * c
            if disc >= 0:
                sqrt_disc = math.sqrt(disc)
                for u in [(-b - sqrt_disc) / (2 * a), (-b + sqrt_disc) / (2 *
a)]:
                    if u >= 0:
                        z_hit = Oz + u * Dz
                        if z0 - eps <= z_hit <= z1 + eps:
                            if u_min is None or u < u_min:
                                u_min = u
        # 端盖 z=z0 与 z=z1
        if abs(Dz) > eps:
            for z_plane in (z0, z1):
                u = (z_plane - Oz) * 1.0 / Dz
                if u >= 0:
                    xh = Ox + u * Dx
                    yh = Oy + u * Dy
                    if xh * xh + yh * yh <= R * R + 1e-9:
                        if u_min is None or u < u_min:
                            u_min = u
        return u_min

    def sample_cylinder_points(self, n_theta=72, n_z=10, n_r_caps=8):
        """采样圆柱侧面与上下端盖上的点"""
        R = self.target_radius
        H = self.target_height
        yc = self.target_center[1]
        # 侧面
        side_points = []
        thetas = np.linspace(0, 2 * np.pi, n_theta, endpoint=False)
        zs = np.linspace(0, H, n_z)
        for z in zs:
            for th in thetas:
```

```
                x = R * math.cos(th)
                y = yc + R * math.sin(th)
                side_points.append(np.array([x, y, z]))
        # 端盖
        def disk_points(z_plane):
            pts = []
            rs = np.linspace(0, R, n_r_caps)
            for r in rs:
                for th in thetas:
                    x = r * math.cos(th)
                    y = yc + r * math.sin(th)
                    pts.append(np.array([x, y, z_plane]))
            return pts
        top_points = disk_points(H)
        bottom_points = disk_points(0.0)
        return np.array(side_points), np.array(top_points),
np.array(bottom_points)

    def compute_visible_points(self, M, side_pts, top_pts, bot_pts, eps=1e-4):
        """判定从 M 出发，哪些采样点是该圆柱表面的首交点(可见点)"""
        R = self.target_radius
        z0, z1 = 0.0, self.target_height
        yc = self.target_center[1]
        visible_pts = []
        for P in np.vstack((side_pts, top_pts, bot_pts)):
            D_vec = P - M
            dist = np.linalg.norm(D_vec)
            if dist < 1e-9:
                continue
            D = D_vec / dist
            u_first = self.ray_first_hit_cylinder_with_caps(M, D, R, z0, z1,
y_center=yc)
            if u_first is None:
                continue
            if abs(u_first - dist) <= 1e-3:  # P 为首交点
                visible_pts.append((P, dist))
        return visible_pts
```

```
    def compute_occlusion_ratio(self, t, drop_time, explode_time, n_theta=72,
n_z=10, n_r_caps=8):
        """计算时刻 t 可见采样点中被云团遮挡的比例(0~1)，以及是否全遮挡"""
        if not (explode_time <= t <= explode_time + self.effective_duration):
            return 0.0, False, 0, 0
        M = self.missile_position(t)
        C = self.smoke_cloud_center(t, drop_time, explode_time)
        R_cloud = self.effective_radius
        side_pts, top_pts, bot_pts = self.sample_cylinder_points(n_theta, n_z,
n_r_caps)
        visible = self.compute_visible_points(M, side_pts, top_pts, bot_pts)
        if len(visible) == 0:
            return 0.0, False, 0, 0
        blocked = 0
        for P, dist_MP in visible:
            D = (P - M) / dist_MP
            u_cloud = self.ray_sphere_first_u(M, D, C, R_cloud)
            if u_cloud is not None and u_cloud < dist_MP - 1e-6:
                blocked += 1
        ratio = blocked / len(visible)
        full = (blocked == len(visible))
        return ratio, full, blocked, len(visible)

    def is_target_fully_occluded(self, t, drop_time, explode_time):
        """是否在 t 时刻实现对真目标(圆柱)的全遮挡"""
        _, full, _, _ = self.compute_occlusion_ratio(t, drop_time,
explode_time)
        return full

    def calculate_blocking_periods(self):
        """计算‘全遮挡’的时间段，并返回时间序列与遮挡比例序列"""
        drop_time = self.drop_delay
        explode_time = drop_time + self.explode_delay
        missile_distance = np.linalg.norm(self.missile_init_pos)
        total_flight_time = missile_distance / self.missile_speed
        print(f"导弹总飞行时间: {total_flight_time:.2f} 秒")
```

```
        print(f"烟幕弹投放时间: {drop_time:.2f} 秒")
        print(f"烟幕弹起爆时间: {explode_time:.2f} 秒")
        print(f"烟幕有效结束时间: {explode_time + self.effective_duration:.2f}
秒")

        # 只需考虑起爆有效窗附近
        start_time = max(0, explode_time - 1)
        end_time = min(total_flight_time, explode_time +
self.effective_duration + 1)
        dt = 0.01  # 时间步长改为 0.01s
        times = np.arange(start_time, end_time + 1e-9, dt)
        ratios = []
        blocked_periods = []
        current_start = None
        for t in times:
            ratio, full, _, _ = self.compute_occlusion_ratio(t, drop_time,
explode_time)
            ratios.append(ratio)
            if full and current_start is None:
                current_start = t
            elif (not full) and current_start is not None:
                blocked_periods.append((current_start, t))
                current_start = None
        if current_start is not None:
            blocked_periods.append((current_start, times[-1]))
        return blocked_periods, drop_time, explode_time, times,
np.array(ratios)

    def analyze(self):
        """执行完整分析"""
        print("=" * 60)
        print("烟幕干扰弹对导弹 M1 有效遮蔽时长分析 (圆柱体 + 全遮挡判定)")
        print("=" * 60)

        drop_time = self.drop_delay
        explode_time = drop_time + self.explode_delay
```

```
        print(f"\n初始条件:")
        print(f"导弹M1初始位置: {self.missile_init_pos}")
        print(f"无人机FY1初始位置: {self.drone_init_pos}")
        print(f"真目标(圆柱)下底圆心: {self.target_center},
R={self.target_radius}m, H={self.target_height}m")
        print(f"导弹速度: {self.missile_speed} m/s, 无人机速度:
{self.drone_speed} m/s")

        print(f"\n烟幕弹投放位置 (t={drop_time}s):
{self.drone_position(drop_time)}")
        print(f"烟幕弹起爆位置 (t={explode_time}s):
{self.smoke_bomb_trajectory(explode_time, drop_time)}")

        blocked_periods, _, _, times, ratios =
self.calculate_blocking_periods()

        print(f"\n‘全遮挡’分析结果:")
        total_full_time = 0.0
        if blocked_periods:
            print(f"发现 {len(blocked_periods)} 个全遮挡时间段:")
            for i, (s, e) in enumerate(blocked_periods, 1):
                dur = e - s
                total_full_time += dur
                print(f"  第{i}段: {s:.3f}s - {e:.3f}s (持续 {dur:.3f}s)")
        else:
            print("未发现全遮挡时间段")
        print(f"\n总有效遮蔽时长(全遮挡): {total_full_time:.3f} 秒")

        return total_full_time, blocked_periods

def main():
    """主函数"""
    analyzer = SmokeInterferenceAnalyzer()
    total_time, periods = analyzer.analyze()

    print(f"\n最终结果:")
    print(f"烟幕干扰弹对导弹M1的有效遮蔽时长为: {total_time:.3f} 秒")
```

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

问题二

```
import math
import argparse
import random
from typing import Tuple

Vec3 = Tuple[float, float, float]

# ---------------- Vector utilities ----------------

def dot(a: Vec3, b: Vec3) -> float:
    return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]


def sub(a: Vec3, b: Vec3) -> Vec3:
    return (a[0]-b[0], a[1]-b[1], a[2]-b[2])


def add(a: Vec3, b: Vec3) -> Vec3:
    return (a[0]+b[0], a[1]+b[1], a[2]+b[2])


def mul(a: Vec3, k: float) -> Vec3:
    return (a[0]*k, a[1]*k, a[2]*k)


def norm(a: Vec3) -> float:
    return math.sqrt(dot(a, a))


def clamp(x: float, lo: float, hi: float) -> float:
    return max(lo, min(hi, x))
```

```
def dist_point_to_segment(P: Vec3, A: Vec3, B: Vec3) -> float:
    """Distance from point P to segment AB in 3D."""
    AB = sub(B, A)
    AP = sub(P, A)
    ab2 = dot(AB, AB)
    if ab2 == 0.0:
        return norm(sub(P, A))
    t = clamp(dot(AP, AB) / ab2, 0.0, 1.0)
    closest = add(A, mul(AB, t))
    return norm(sub(P, closest))


def cloud_center_after(Pe: Vec3, t: float, t_explosion: float, sink_speed:
float = 3.0) -> Vec3:
    """Cloud center position at time t >= t_explosion. Only z changes
(downwards)."""
    dt = max(0.0, t - t_explosion)
    return (Pe[0], Pe[1], Pe[2] - sink_speed*dt)


def effective_obscuration_time_for_M1(speed: float,
                                      heading_deg: float,
                                      t_release: float,
                                      t_fuse: float,
                                      g: float = 9.8,
                                      dt: float = 0.001) -> float:
    """
    Compute total effective obscuration time (seconds) for M1 using given
parameters.
    Criterion: distance from cloud center to segment [M(t), T] <= 10 m within
20 s after explosion.
    dt: simulation step size in seconds (use a coarser dt for faster
optimization, e.g., 0.02; refine later with 0.001).
    """
    # Target (true target) center of bottom face
    T = (0.0, 200.0, 0.0)
```

```
    # Compute explosion point and time
    Pe, t_explosion, _ = compute_explosion_point_FY1_new_standard(speed,
heading_deg, t_release, t_fuse, g)

    # Simulation window
    t0 = t_explosion
    t1 = t_explosion + 20.0

    # Numerical integration
    R_eff = 10.0

    t = t0
    prev_d = None
    total = 0.0

    while t <= t1 + 1e-12:
        C = cloud_center_after(Pe, t, t_explosion, sink_speed=3.0)
        M = missile_state_M1(t)
        d = dist_point_to_segment(C, M, T)

        if prev_d is not None:
            if (prev_d <= R_eff and d <= R_eff):
                total += dt
            elif (prev_d <= R_eff and d > R_eff):
                denom = (d - prev_d)
                if abs(denom) > 1e-12:
                    alpha = (R_eff - prev_d) / denom
                    if 0.0 < alpha < 1.0:
                        total += alpha * dt
            elif (prev_d > R_eff and d <= R_eff):
                denom = (d - prev_d)
                if abs(denom) > 1e-12:
                    alpha = (R_eff - prev_d) / denom
                    if 0.0 < alpha < 1.0:
                        total += (1.0 - alpha) * dt
                else:
```

```
                    total += dt
        prev_d = d
        t += dt

    return total

# ---------------- Optimization: Particle Swarm Optimization (PSO) ----------
-----

def wrap_angle_deg(a: float) -> float:
    """Wrap angle to [0, 360]."""
    a = a % 360.0
    if a < 0:
        a += 360.0
    return a


def shortest_ang_diff_deg(target: float, current: float) -> float:
    """Return signed shortest angular difference (target - current) in degrees
within [-180, 180]."""
    d = (target - current + 180.0) % 360.0 - 180.0
    return d


def pso_optimize(iters: int = 150,
                 swarm_size: int = 25,
                 seed: int = 42,
                 speed_min: float = 70.0,
                 speed_max: float = 140.0,
                 heading_min: float = 0.0,
                 heading_max: float = 360.0,
                 t_release_min: float = 1e-3,
                 t_release_max: float = 3.0,
                 t_fuse_min: float = 1e-3,
                 t_fuse_max: float = 1.5,
                 w: float = 0.7,
                 c1: float = 1.5,
```

```
                c2: float = 1.5,
                dt_coarse: float = 0.03) -> Tuple[Tuple[float, float, float,
float], float]:
    """
    Particle Swarm Optimization to maximize obscuration time.
    Returns (best_params_tuple, best_value), where params = (speed,
heading_deg, t_release, t_fuse).
    Runtime is bounded by swarm_size * iters objective evaluations with coarse
dt.
    """
    random.seed(seed)

    # Ranges for clamping and velocity limits
    rng_speed = speed_max - speed_min
    rng_heading = heading_max - heading_min  # assumed 360
    rng_tr = t_release_max - t_release_min
    rng_tf = t_fuse_max - t_fuse_min

    # Velocity limits (keep stable)
    vmax = [rng_speed * 0.25, 90.0, rng_tr * 0.25, rng_tf * 0.25]

    def clamp_param(x, lo, hi):
        return max(lo, min(hi, x))

    def obj(xx):
        return effective_obscuration_time_for_M1(xx[0], xx[1], xx[2], xx[3],
dt=dt_coarse)

    # Initialize swarm
    X = []  # positions
    V = []  # velocities
    Pbest = []  # personal best positions
    PbestVal = []

    Gbest = None
    GbestVal = -1e18
```

```
for _ in range(swarm_size):
    s = random.uniform(speed_min, speed_max)
    h = random.uniform(heading_min, heading_max)
    tr = random.uniform(t_release_min, t_release_max)
    tf = random.uniform(t_fuse_min, t_fuse_max)
    x = [s, h, tr, tf]
    # Small initial velocities
    v = [random.uniform(-rng_speed*0.05, rng_speed*0.05),
         random.uniform(-30.0, 30.0),
         random.uniform(-rng_tr*0.05, rng_tr*0.05),
         random.uniform(-rng_tf*0.05, rng_tf*0.05)]
    X.append(x)
    V.append(v)
    val = obj(x)
    Pbest.append(x[:])
    PbestVal.append(val)
    if val > GbestVal:
        GbestVal = val
        Gbest = x[:]

# PSO main loop
for it in range(iters):
    for i in range(swarm_size):
        x = X[i]
        v = V[i]

        # Update velocity per dimension
        # speed
        r1 = random.random(); r2 = random.random()
        v0 = (w * v[0]
              + c1 * r1 * (Pbest[i][0] - x[0])
              + c2 * r2 * (Gbest[0] - x[0]))
        # heading (circular distance)
        r1 = random.random(); r2 = random.random()
        diff_p = shortest_ang_diff_deg(Pbest[i][1], x[1])
        diff_g = shortest_ang_diff_deg(Gbest[1], x[1])
        v1 = (w * v[1]
```

```
      + c1 * r1 * diff_p
      + c2 * r2 * diff_g)
# t_release
r1 = random.random(); r2 = random.random()
v2 = (w * v[2]
      + c1 * r1 * (Pbest[i][2] - x[2])
      + c2 * r2 * (Gbest[2] - x[2]))
# t_fuse
r1 = random.random(); r2 = random.random()
v3 = (w * v[3]
      + c1 * r1 * (Pbest[i][3] - x[3])
      + c2 * r2 * (Gbest[3] - x[3]))

# Velocity clamp
v0 = max(-vmax[0], min(vmax[0], v0))
v1 = max(-vmax[1], min(vmax[1], v1))
v2 = max(-vmax[2], min(vmax[2], v2))
v3 = max(-vmax[3], min(vmax[3], v3))
v = [v0, v1, v2, v3]

# Update position
x0 = clamp_param(x[0] + v0, speed_min, speed_max)
x1 = wrap_angle_deg(x[1] + v1)
x2 = clamp_param(x[2] + v2, t_release_min, t_release_max)
x3 = clamp_param(x[3] + v3, t_fuse_min, t_fuse_max)
x = [x0, x1, x2, x3]

# Save back
X[i] = x
V[i] = v

# Evaluate and update bests
val = obj(x)
if val > PbestVal[i]:
    Pbest[i] = x[:]
    PbestVal[i] = val
    if val > GbestVal:
```

```
                    GbestVal = val
                    Gbest = x[:]

    return (Gbest[0], Gbest[1], Gbest[2], Gbest[3]), GbestVal


def heading_to_unit_new(theta_deg: float) -> Tuple[float, float]:
    """New standard: 0 deg along +x, CCW positive (xy-plane). Returns (ux,
uy)."""
    th = math.radians(theta_deg)
    return (math.cos(th), math.sin(th))


def missile_state_M1(t: float) -> Vec3:
    """M1 position at time t [s]. Missile heads towards (0,0,0) at 300 m/s."""
    M0 = (20000.0, 0.0, 2000.0)
    v_dir = (-20000.0, 0.0, -2000.0)
    L = math.sqrt(v_dir[0]**2 + v_dir[1]**2 + v_dir[2]**2)
    u = (v_dir[0]/L, v_dir[1]/L, v_dir[2]/L)
    v = (u[0]*300.0, u[1]*300.0, u[2]*300.0)
    return (M0[0] + v[0]*t, M0[1] + v[1]*t, M0[2] + v[2]*t)


def compute_explosion_point_FY1_new_standard(speed: float = 132.86,
                                             heading_deg: float = 6.63,
                                             t_release: float = 0.12,
                                             t_fuse: float = 0.68,
                                             g: float = 9.8) -> Tuple[Vec3,
float, Tuple[float, float]]:
    """
    Compute FY1 explosion point using the NEW heading standard directly.
    Returns (Pe, t_explosion, (ux, uy)).
    """
    FY1 = (17800.0, 0.0, 1800.0)

    ux, uy = heading_to_unit_new(heading_deg)
```

```
    # Drone horizontal velocity components
    vx = speed * ux
    vy = speed * uy

    # Release point (constant altitude prior to release)
    R = (FY1[0] + vx*t_release, FY1[1] + vy*t_release, FY1[2])

    # Free fall after release for t_fuse seconds: horizontal continues,
vertical under gravity
    x_e = R[0] + vx*t_fuse
    y_e = R[1] + vy*t_fuse
    z_e = R[2] - 0.5*g*(t_fuse**2)

    Pe = (x_e, y_e, z_e)
    t_explosion = t_release + t_fuse
    return Pe, t_explosion, (ux, uy)


def cloud_center_after(Pe: Vec3, t: float, t_explosion: float, sink_speed:
float = 3.0) -> Vec3:
    """Cloud center position at time t >= t_explosion. Only z changes
(downwards)."""
    dt = max(0.0, t - t_explosion)
    return (Pe[0], Pe[1], Pe[2] - sink_speed*dt)


def main():
    parser = argparse.ArgumentParser(description="Compute or optimize
obscuration time for M1 (new heading standard)")
    subparsers = parser.add_subparsers(dest="cmd", required=True)

    # Subcommand: direct evaluation
    p_eval = subparsers.add_parser("eval", help="Direct evaluation with
explicit parameters")
    p_eval.add_argument("--speed", type=float, required=True, help="UAV speed
in m/s, must be within [70, 140]")
    p_eval.add_argument("--heading", type=float, required=True, help="UAV
```

```
heading in degrees [0, 360], new standard: 0°  along +x, CCW positive")
    p_eval.add_argument("--t_release", type=float, required=True,
help="Release time after task received (s), >0")
    p_eval.add_argument("--t_fuse", type=float, required=True, help="Fuse time
after release until explosion (s), >0")
    p_eval.add_argument("--dt", type=float, default=0.001, help="Simulation
step size in seconds (default 0.001)")

    # Subcommand: optimization via PSO
    p_opt = subparsers.add_parser("opt", help="Optimize parameters using
Particle Swarm Optimization (PSO)")
    p_opt.add_argument("--iters", type=int, default=150, help="Number of PSO
iterations")
    p_opt.add_argument("--swarm", type=int, default=25, help="Swarm size
(number of particles)")
    p_opt.add_argument("--seed", type=int, default=42, help="Random seed")
    p_opt.add_argument("--speed_min", type=float, default=70.0)
    p_opt.add_argument("--speed_max", type=float, default=140.0)
    p_opt.add_argument("--heading_min", type=float, default=0.0)
    p_opt.add_argument("--heading_max", type=float, default=360.0)
    p_opt.add_argument("--t_release_max", type=float, default=3.0)
    p_opt.add_argument("--t_fuse_max", type=float, default=1.5)
    p_opt.add_argument("--w", type=float, default=0.7, help="PSO inertia
weight")
    p_opt.add_argument("--c1", type=float, default=1.5, help="PSO cognitive
coefficient")
    p_opt.add_argument("--c2", type=float, default=1.5, help="PSO social
coefficient")
    p_opt.add_argument("--dt_coarse", type=float, default=0.03, help="Coarse
dt for search (e.g., 0.02~0.05)")
    p_opt.add_argument("--refine_dt", type=float, default=0.001,
help="Refinement dt for final evaluation (e.g., 0.001~0.002)")

    args = parser.parse_args()

    if args.cmd == "eval":
        # Validate ranges
```

```
        if not (70.0 <= args.speed <= 140.0):
            raise ValueError(f"speed must be within [70, 140], got
{args.speed}")
        if not (0.0 <= args.heading <= 360.0):
            raise ValueError(f"heading must be within [0, 360], got
{args.heading}")
        if not (args.t_release > 0.0):
            raise ValueError(f"t_release must be > 0, got {args.t_release}")
        if not (args.t_fuse > 0.0):
            raise ValueError(f"t_fuse must be > 0, got {args.t_fuse}")

        Pe, t_explosion, _ =
compute_explosion_point_FY1_new_standard(speed=args.speed,

heading_deg=args.heading,

t_release=args.t_release,

t_fuse=args.t_fuse)
        total_time = effective_obscuration_time_for_M1(speed=args.speed,

heading_deg=args.heading,

t_release=args.t_release,
                                                       t_fuse=args.t_fuse,
                                                       dt=args.dt)

        print("采用新方位角定义：0° 沿+x，逆时针为正（0~360° ）")
        print(f"FY1 航向 = {args.heading:.2f}° （新标准），速度 =
{args.speed:.2f} m/s")
        print(f"释放时刻 t_release = {args.t_release:.3f} s，起爆时刻
t_explosion = {t_explosion:.3f} s")
        print(f"起爆点 Pe = ({Pe[0]:.3f}, {Pe[1]:.3f}, {Pe[2]:.3f})")
        print(f"云团中心以 3 m/s 匀速下沉，有效期 20 s，判据：距离[M1(t), T]线
段 ≤ 10 m")
        print(f"对 M1 的有效遮蔽总时长 = {total_time:.6f} s")
```

```
    elif args.cmd == "opt":
        params, best_val_coarse = pso_optimize(iters=args.iters,
                                               swarm_size=args.swarm,
                                               seed=args.seed,
                                               speed_min=args.speed_min,
                                               speed_max=args.speed_max,
                                               heading_min=args.heading_min,
                                               heading_max=args.heading_max,
                                               t_release_min=1e-3,

t_release_max=args.t_release_max,
                                               t_fuse_min=1e-3,
                                               t_fuse_max=args.t_fuse_max,
                                               w=args.w, c1=args.c1,
c2=args.c2,
                                               dt_coarse=args.dt_coarse)
        s, h, tr, tf = params
        # Refine with smaller dt for accurate final time
        final_time = effective_obscuration_time_for_M1(s, h, tr, tf,
dt=args.refine_dt)
        Pe, t_explosion, _ = compute_explosion_point_FY1_new_standard(s, h,
tr, tf)

        print("优化完成（粒子群算法 PSO）")
        print(f"参数范围：speed∈[{args.speed_min},{args.speed_max}],
heading∈[{args.heading_min},{args.heading_max}],
t_release∈(0,{args.t_release_max}], t_fuse∈(0,{args.t_fuse_max}]")
        print(f"粗精度 dt = {args.dt_coarse}，PSO：iters = {args.iters}，swarm
= {args.swarm}，seed = {args.seed}")
        print("最优参数（粗评价下）：")
        print(f"  speed = {s:.6f} m/s, heading = {h:.6f} °，t_release =
{tr:.6f} s, t_fuse = {tf:.6f} s")
        print(f"粗评价下的遮蔽时长 ~= {best_val_coarse:.6f} s")
        print("细评估（refine）结果：")
        print(f"  起爆时刻 = {t_explosion:.6f} s，起爆点 Pe = ({Pe[0]:.3f},
{Pe[1]:.3f}, {Pe[2]:.3f})")
        print(f"  最终有效遮蔽总时长 = {final_time:.6f} s
```

```
 (dt={args.refine_dt}) ")


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