关于烟幕干扰弹投放策略的研究问题

摘要

在现代电子对抗背景下，无人机集群精确干扰已成为关键战术。本文聚焦于无人机投放烟幕弹的协同干扰任务，旨在构建一套最大化对来袭导弹总遮蔽时间的动态最优化决策模型。为此，我们首先建立了一个通用的**“运动学-几何遮蔽”**高精度仿真平台，为后续所有问题的求解提供了统一的计算内核。

针对问题一，我们应用基础仿真平台，通过对真目标进行大量采样, 来保证遮蔽的完全性。该过程精准计算烟幕干扰弹从投放、起爆到形成有效遮蔽云团的全过程，结合导弹 M1 的飞行轨迹与速度、真目标位置及烟幕有效遮蔽范围，最终确定该固定策略下烟幕对 M1 的有效遮蔽时长为 **1.392s**。

针对问题二，为解决单弹策略的逆向寻优，我们将其构建为一个四维“黑箱”优化模型，并选用**粒子群优化算法**。通过设计复合适应度函数以克服“零解平原”问题，最终将最优遮蔽时长有效提升至 **4.608s**。

针对问题三，为解决单平台三弹药的时序协同，我们将决策空间扩展至八维，并以所有遮蔽时间区间的**数学并集**为优化目标。通过粒子群优化算法与复合适应度函数的结合，研究在求解过程中发现“最小化重叠浪费”（通过合理规划投放与起爆时序，减少多弹有效遮蔽区间的重复部分）与“策略性放弃”（对部分难以形成有效遮蔽的时序窗口主动舍弃，集中资源优化关键时段）的战术洞察，最终得到 3 枚烟幕干扰弹的最优投放策略，并得到可以有效遮蔽 **6.412s** 的结果。

针对问题四，为解决三平台协同干扰的难题，我们提出了一种**降维优化**策略。通过固定最优平台参数，将高维联合优化问题分解为更易求解的子问题，通过逐步分解与迭代优化，在保证解质量的前提下提升求解效率，最终确定三架无人机协同投放烟幕弹的最优策略，实现了 **11.585s** 的遮蔽效果。

针对问题五，面对五无人机对抗三导弹的大规模对抗场景，我们创新性地构建了一套 **“分层-协同”决策框架**。该框架通过**匈牙利算法**解决全局任务分配，再结合**差分进化算法**进行局部策略寻优，通过动态迭代的闭环流程，最终求解出 **18.20s** 的全局最优总遮蔽时间。

本文综合考虑了从单体智能到群体协同的多种作战想定，建立了一套完整、可扩展的无人机烟幕干扰优化决策体系，并对模型的鲁棒性、优缺点及推广方向进行了分析。

关键字：**运动几何遮蔽　粒子群算法　降维优化　匈牙利任务分配　差分进化**

一、 问题重述

1.1 问题背景

在现代化的攻防对抗中，无人机凭借其灵活性与低成本优势，已成为执行精确干扰任务、精确打击任务、无人化作战的关键平台。其中，通过投放烟幕干扰弹在来袭导弹与我方高价值目标之间形成瞬时遮蔽，是一种效费比极高的战术手段，将在未来的战争格局中发挥重大作用。该战术的成功与否，高度依赖于对无人机飞行轨迹、干扰弹投放时机与起爆时序的精确协同规划。

在这一战术背景下，需要考虑的核心物理过程包括：无人机的等高匀速直线飞行、干扰弹脱离载机后的平抛运动、以及烟幕云团形成后的匀速下沉。最终的战术目标，是在满足一系列复杂几何与时序约束的前提下，设计一套最优的投放策略，使得对真目标的总有效遮蔽时间最大化。本文将针对从单平台单弹药，到多平台多弹药协同，再到多平台对抗多目标的递进式复杂场景，建立相应的数学模型并求解最优投放策略。

1.2 问题提出

问题一 单架无人机（FY1）采用一套完全给定的飞行与投放参数（固定速度、航向、投放延迟、起爆延迟）对单枚来袭导弹（M1）进行干扰，建立数学模型，精确计算其产生的有效遮蔽时长。

问题二 单架无人机（FY1）干扰单导弹（M1）的场景下，将无人机的飞行方向、飞行速度以及干扰弹的投放与起爆时序作为决策变量，寻找一组最优参数组合，使得有效遮蔽时间最大化。

问题三 单架无人机（FY1）挂载三枚干扰弹对单枚导弹（M1）进行连续干扰，需在满足最小投放间隔的约束下，设计一套包含无人机飞行策略与三枚弹药投放时序的最优协同方案，以最大化总有效遮蔽时间。

问题四 三架无人机（FY1, FY2, FY3）各挂载一枚干扰弹，对同一枚导弹（M1）进行协同干扰，为每架无人机独立规划其飞行与投放策略，使得三枚弹药形成的联合遮蔽效能达到最优。

问题五 五架无人机（每机至多三枚弹药）需同时应对三枚不同方向来袭的导弹（M1, M2, M3），设计一套全局最优的协同对抗策略。该策略不仅要决定每架无人机的飞行与

投放方案，更要解决“哪架无人机、用哪枚弹、在何时、应对哪个威胁”的动态资源分配问题，以实现对所有威胁的总遮蔽效能最大化。

二、 问题分析

2.1 问题一的分析

问题一要求在完全确定的战术参数下，对单次干扰效果进行精确的量化评估，这本质上是一个高精度的动态系统仿真问题。我们的分析思路是，首先根据牛顿运动学定律，建立描述导弹、无人机、干扰弹以及烟幕云团在三维空间中随时间演化的精确运动学轨迹方程。在此基础上，问题的核心挑战转化为一个随时间动态演变的复杂几何判定问题。根据干扰机理，我们为“有效遮蔽”建立了一个严格的判定准则：在某一时刻，仅当烟幕云球体能够同时遮断从导弹位置到真目标表面所有代表性采样点的视线线段时，才判定该时刻为有效遮蔽。对于每一条独立的视线遮蔽判定，我们通过建立视线线段的参数方程与烟幕球体的球面方程，将其转化为求解一个关于线段参数 s 的一元二次方程在 $[0,1]$ 区间内是否存在实数解的问题，从而进行精确的几何相交检验。最终，我们通过编写仿真程序，以微小的时间步长对整个烟幕有效时间窗口进行离散遍历，在每个时间步上执行上述严苛的多点遮蔽判定，并通过数值积分累加所有满足条件的步长，从而得到总的有效遮蔽时长。

2.2 问题二的分析

问题二将挑战从“正向仿真”升级为“逆向寻优”，要求我们反向求解一组最优策略参数以最大化遮蔽时长，这标志着问题性质从计算问题演变为一个高维、非线性、非凸的全局优化问题。决策变量共同构成了一个由飞行速度 v、方向角 θ、投放延迟 t_{deploy} 及引信延迟 Δt_{fuse} 组成的四维连续空间。其目标函数值，即总遮蔽时长，只能通过调用一个内置了自适应时间步长与高密度目标采样的复杂仿真引擎获得，呈现显著的“黑箱”特性。为应对此“黑箱”优化挑战，程序实现了一个包裹在**多试验管理框架**下的**粒子群优化（PSO）算法**。该框架通过执行多次（例如 3 次）独立的优化流程来增强全局搜索能力，并对结果进行统计分析。这种设计不仅提升了寻获全局最优解的概率，也验证了策略的鲁棒性，最终选取所有试验中的最优结果作为推荐策略。

2.3 问题三的分析

问题三将挑战的性质从单弹药静态优化升维至多弹药的动态协同规划。其核心在于，三枚弹药共享 FY1 这一飞行平台，此约束为各弹药的投放决策引入了强耦合关系，即单枚弹药的投放时机与位置将直接影响后续弹药的决策基准。决策变量因此呈现出

混合结构：全局共享的飞行策略（恒定速度 v 与航向角 θ），以及决定各弹药具体行为的六个独立时序参数（三次起爆延迟与两次投放间隔）。由此构成的八维搜索空间不仅广阔，其适应度地形更是呈现出高度非线性与不连续性，因为时序参数的微小扰动可能导致遮蔽区间重叠状态的剧烈变化，从而引起总遮蔽时长的阶跃。为应对评估此复杂目标的巨大计算量并提升运算效率，我们采用了并行计算框架来将适应度评估任务分配至多个 CPU 核心。优化目标是最大化三片烟幕云有效遮蔽时间的时间并集，这迫使模型在“最大化单弹时长”与“最小化重叠浪费”之间寻找最佳平衡，以实现高效的“接力”遮蔽同时沿用粒子群优化算法来完成模型的求解。

2.4 问题四的分析

问题四将协同作战的复杂度提升至新的层次，引入了三架无人机（FY1, FY2, FY3）协同投放烟幕以拦截单一来袭导弹 M1 的多平台协作场景。理论上，这构成了一个包含三架无人机各自飞行速度、航向角、投放延迟和起爆延迟的 12 维联合优化问题。然而，为了在巨大的搜索空间中更高效地寻优，并利用前序问题的结论，我们采取了一种分步优化的策略：将无人机 FY1 的投放参数固定为第二问中求得的最优解。因此，问题转化为在 FY1 提供基础遮蔽的条件下，集中优化 FY2 与 FY3 的 8 个决策变量，寻求最佳的协同补充方案。优化目标依然是最大化总有效遮蔽时长，该时长定义为三枚烟幕弹形成的遮蔽时间段的**并集**，即只要有任意一枚烟幕弹能够完全阻断导弹视线，该时刻即被视为有效遮蔽。我们继续沿用 PSO 算法来求解此 8 维优化问题，通过寻找 FY2 和 FY3 的最优时空部署，使其与 FY1 的固定策略形成最强的干扰合力。

2.5 问题五的分析

问题五是整个问题的综合体现，引入了一个全新的决策层面：**动态任务分配**。若将所有决策变量（最多可达 40 维）进行“一次性”的静态优化，必然会因“维度灾难”而导致算法难以收敛到高质量解。因此必须摒弃静态优化的思想，转而采用一种更接近真实作战指挥流程的**迭代式**、**分层决策框架**来解构此问题，这是一个“分配-寻优-评估-再分配”的动态闭环。在每一轮迭代中，框架首先通过匈牙利算法为当前“待命”的无人机资源进行最优的任务指派；随后通过差分进化算法为每个新指派的任务求解一个增量式的最优投放方案；最后，在全局审查层，若所有无人机均已获得初步方案，框架将启动“**最差者淘汰重优化**”机制，以寻求突破、跳出局部最优。这种将一个庞大的静态问题转化为逐步演进、自我完善的智能决策过程，是我们在高维协同场景下高效收敛到高质量解的关键所在。

三、 模型假设

(1) 所有对象的运动均在标准三维笛卡尔坐标系中描述，忽略地球曲率、空气阻力以及风场等气象因素的影响。重力加速度 g 视为恒定的 9.8 m/s²。基于此，干扰弹脱离无人机后将进行理想的平抛运动，烟幕云团仅考虑其固有的 3 m/s 匀速下沉。

(2) 我们假设无人机在接收到任务指令后，能够瞬时完成航向和速度的调整，并在后续过程中保持等高度的匀速直线飞行。同时，烟幕弹在起爆后瞬时形成一个半径均匀、密度恒定的标准球体。

(3) 考虑到计算的复杂性，我们将对整个圆柱体目标的遮蔽判定问题，简化为对目标轮廓上若干关键点的遮蔽判定。只要导弹与这些关键点中任意一点的视线被烟幕云阻挡，即认为该时刻产生了有效遮蔽。

(4) 假设控制中心在 $t = 0$ 时刻获取的所有无人机和导弹的位置信息是完全精确的，不存在测量误差。

四、 符号说明

表 1　本文符号说明

符号	意义	符号	意义	符号	意义
$\vec{P}_M(t)$	导弹在 t 时刻的位置	$\vec{v}_M$	导弹速度向量	R_c	烟幕云有效半径
$\vec{P}_{FY}(t)$	无人机在 t 时刻的位置	v_M	导弹飞行速率 (标量)	T_{total}	总有效遮蔽时长
$\vec{P}_C(t)$	烟幕中心在 t 时刻的位置	$\vec{v}_{FY}$	无人机速度向量	$\vec{x}$	优化问题决策向量
$\vec{P}_{M,0}^{(j)}$	第 j 枚导弹的初始位置向量	$v_{\text{avg}}^{(i)}$	第 i 架无人机的平均飞行速率	v	无人机飞行速率 (决策变量)
$\vec{P}_{FY,0}^{(i)}$	第 i 架无人机的初始位置向量	$\vec{v}_{G,0}$	干扰弹初始速度	θ	无人机飞行方向角 (决策变量)
$\vec{P}_{det}$	干扰弹起爆点位置	$\vec{v}_{sink}$	烟幕云团下沉速度	s	视线线段位置参数, $s \in [0,1]$
$\{\vec{P}_k\}$	目标表面采样点集	t_{deploy}	干扰弹投放时刻	a, b, c	线段与球体相交的二次方程系数
$\tilde{O}$	假目标位置 (原点)	t_{det}	干扰弹起爆时刻	E_{ij}	无人机 i 对导弹 j 的干扰效能
$\vec{g}$	重力加速度向量	Δt_{fuse}	投放至起爆延迟 (s)	C_{ij}	无人机 i 应对导弹 j 的总任务成本
w_x	任务成本权重系数	$T_{\text{flight}}^{(j)}$	第 j 枚导弹的总飞行时长	$w^{(k)}$	线性递减惯性权重

五、 模型建立与求解

5.1 问题一求解

5.1.1 运动学轨迹建模

为预测系统中任意时刻的物理状态，我们首先在以假目标 $\tilde{O}$ 为坐标原点的三维笛卡尔坐标系中，对各个关键实体建立其随时间演化的运动学轨迹方程。设任务的起始时刻为 $t = 0$。

由于导弹的形状与结果无关，我们将来袭导弹建模成一个作匀速直线运动的质点，其运动方向始终指向假目标。这一设定是基于导弹在中末制导阶段通常会保持稳定弹道的飞行情况。因此，其速度向量 $\vec{v}_M$ 在整个飞行过程中保持不变，任意时刻 t 的空间位

置 $\vec{P}_M(t)$ 可由其初始位置 $\vec{P}_{M,0}$ 线性计算得到：

$$\vec{v}_M = v_M \cdot \frac{\tilde{O} - \vec{P}_{M,0}}{\|\tilde{O} - \vec{P}_{M,0}\|}$$
$$\vec{P}_M(t) = \vec{P}_{M,0} + \vec{v}_M \cdot t$$

无人机作等高度的匀速直线运动，在接收任务指令后，于投放时刻 t_{deploy} 到达投放点 $\vec{P}_{FY}(t_{deploy})$。同时，干扰弹脱离载机，其初始速度 $\vec{v}_{G,0}$ 继承无人机的飞行速度 $\vec{v}_{FY}$。此后，干扰弹的飞行轨迹将只受重力加速度 $\vec{g}$ 的影响，进行一段理想的平抛运动。经过预设的引信延迟时间 Δt_{fuse} 后，在起爆时刻 t_{det} 到达最终的起爆点 $\vec{P}_{det}$。干扰弹的精确坐标由抛体运动公式确定：

$$\vec{P}_{det} = \vec{P}_{FY}(t_{deploy}) + \vec{v}_{G,0} \cdot \Delta t_{fuse} + \frac{1}{2}\vec{g} \cdot (\Delta t_{fuse})^2$$

干扰弹起爆后，瞬时形成一个球状烟幕云团。该云团的初始位置为干扰弹坐标，之后水平坐标固定，而其垂直坐标则以一个恒定的下沉速度 $\vec{v}_{sink}$ 缓慢下降。此模型将复杂的烟幕扩散与漂移过程简化为一个确定性的匀速下沉运动，从而得到烟幕球心 $\vec{P}_C(t)$ 随时间变化的轨迹：

$$\vec{P}_C(t) = \vec{P}_{det} + \vec{v}_{sink} \cdot (t - t_{det}), \quad t \geq t_{det}$$

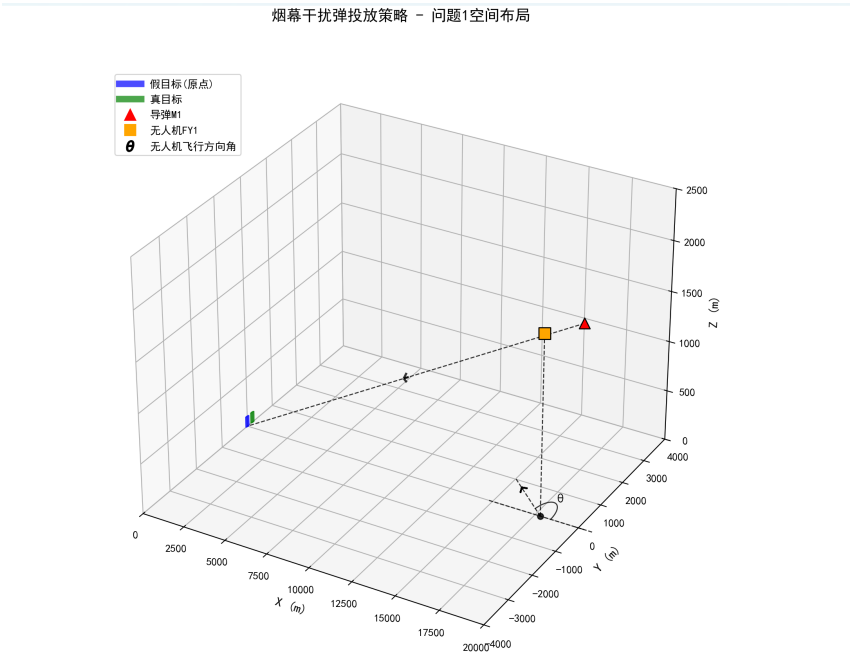

图 1　空间示意图

5.1.2 几何遮蔽判定准则

建立各实体的时变轨迹之后，问题的核心指向了一个动态的几何判定问题–“有效遮蔽”。我们为“有效遮蔽”定义了一个极其严格的准则：在某一时刻，必须是导弹到真目标的所有可能视线都被烟幕球体完全阻挡。由于真目标是一个具有体积的圆柱体，仅仅判断其中心点是否被遮蔽是远远不够的，因为这可能导致目标边缘部分仍然暴露在导弹视线中的误判。为了确保判定的绝对可靠，我们的模型通过在真目标的整个轮廓表面（包括上下底面圆周和侧面母线）生成一组高密度的采样点集 $\{\vec{P}_k\}$，将对一个连续体目标的遮蔽判定问题，离散化为对该点集的同时遮蔽判定问题，为避免因目标体积而产生的漏判，我们将对连续体目标的遮蔽判定问题，离散化为对其表面高密度采样点集的同步遮蔽判定，这确保了判定的完备性。

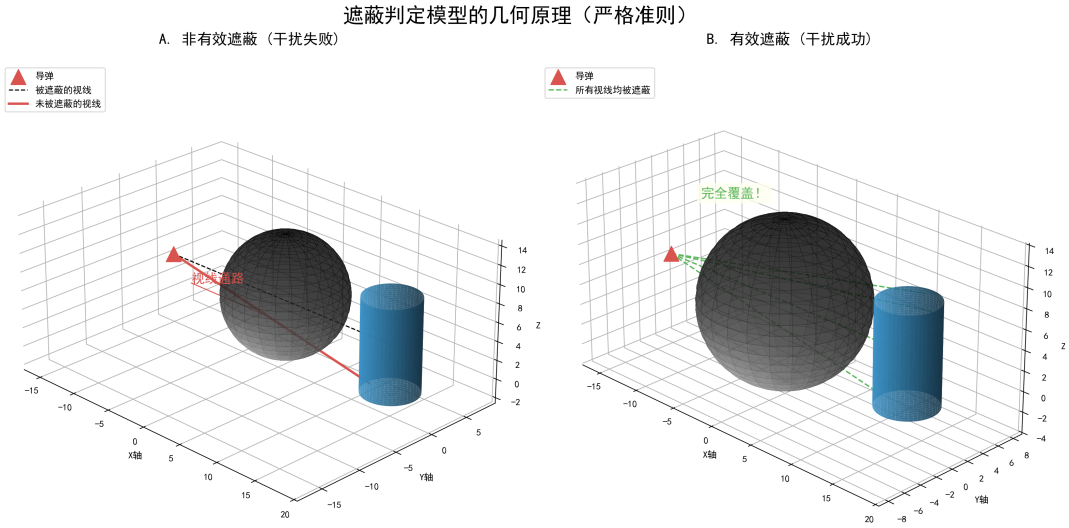

图 2　遮蔽判定模型的几何原理图

对于任意一个采样点，遮蔽与否的判定从一个三维空间中的几何相交问题，转化为一个一维的代数求解问题。任意时刻导弹 $\vec{P}_M(t)$ 与目标采样点 $\vec{P}_k$ 的连线线段，可通过参数 $s \in [0,1]$ 进行描述。该线段与烟幕球体发生相交的充要条件，是存在一个参数 s，使得线段上的对应点到烟幕球心 $\vec{P}_C(t)$ 的距离不大于烟幕半径 R_c。这个几何关系可以由一个关于参数 s 的一元二次不等式展现：

$$as^2 + bs + c \leq 0$$

其中，各项系数完全由该时刻导弹、目标点和烟幕球心的相对位置向量所确定：

$$a = \|\vec{P}_k - \vec{P}_M(t)\|^2$$

$$b = 2(\vec{P}_M(t) - \vec{P}_C(t)) \cdot (\vec{P}_k - \vec{P}_M(t))$$

$$c = \|\vec{P}_M(t) - \vec{P}_C(t)\|^2 - R_c^2$$

通过求解此不等式，我们可以得到无限长视线与球体相交的参数区间。若此区间与代表真实线段的 $[0,1]$ 区间存在非空交集，则意味着该条视线被成功遮挡。在仿真的每一个瞬间，只有当导弹到所有采样点 $\vec{P}_k$ 的视线均满足此遮蔽条件时，我们才判定该时刻 t 实现了对真目标的有效遮蔽。

我们承认，上述基于高密度采样的判定方法本质上是一种离散化近似，其判定的完备性严格依赖于采样密度，理论上存在因采样不足而“挂一漏万”的风险。为追求极致的几何精确性，一种更优越的判定方法是将问题转化到导弹的观测坐标系。在此坐标系下，烟幕球体的投影为一个圆形，而真目标圆柱的投影则为一个“凸包”轮廓。遮蔽判定则转化为一个二维平面上的圆形是否完全覆盖该凸包的几何问题。该方法避免了三维空间中的复杂采样，通过解析几何的方式进行判定，理论上更为完备。然而，考虑到投影变换与凸包计算在实现上的复杂性与较高的时间成本，本次求解仍采用经过充分加密的高密度采样法。我们通过超量采样确保了极高的覆盖率，其精度在工程应用上已足够可靠，并为后续的高效优化计算奠定了基础。

5.1.3 高精度仿真算法设计

由于遮蔽状态随时间连续变化，总有效遮蔽时长无法通过解析方法直接求得，必须依赖高精度的数值仿真。为此，我们设计并实现了一个鲁棒的仿真计算框架。该框架将题目给定的所有战术参数作为输入，并将仿真时间窗口大小设定为烟幕从起爆到失效的 20 秒有效时间。

仿真的核心是一种结合了离散时间步进与自适应边界搜索的数值积分方法。算法首先以一个极小的时间步长 Δt（比如 0.001 秒）对整个时间窗口进行离散化。在每一个时间步 t_i 上，程序都会调用上述的运动学和几何模型，计算出该瞬间的遮蔽状态。然而，简单的定步长累加会在遮蔽状态发生切换的临界点引入不可避免的离散误差。为了克服这一缺陷，我们引入了自适应边界搜索机制。一旦仿真程序检测到相邻的两个时间步 t_i 与 t_{i+1} 之间的遮蔽状态发生了改变，便会立刻在该微小的时间区间内启动高精度的二分查找算法。该算法通过迭代逼近，能够以极高的数值精度（例如 10^{-9} 秒）定位出遮蔽状态发生改变的精确时刻。随后，程序根据这个精确的边界点进行分段时长累加。这种混合算法策略，既保证了在非临界区域的计算效率，又确保了在关键切换点上的计算精度，从而能够准确地捕捉到每一次短暂的遮蔽事件。

5.1.4 仿真结果

我们将问题一给定的预设参数代入仿真模型。首先，根据无人机初始位置 $\vec{P}_{FY,0} = (17800, 0, 1800)$ 与假目标位置 $\tilde{O} = (0, 0, 0)$，确定其飞行方向向量为 $(-1, 0, 0)$。无人机以此方向、以 $v_{FY} = 120$ m/s 的速度进行等高度飞行。经过 $t_{deploy} = 1.5$s 后，无人机到达投放点坐标，计算为 $\vec{P}_{FY,0} + \vec{v}_{FY} \cdot t_{deploy} = (17620.0, 0.0, 1800.0)$ 米，此过程清晰地体

现了“等高度飞行”的约束（z 坐标不变）。随后，干扰弹经过 $\Delta t_{fuse} = 3.6$s 的平抛运动，最终在 $(17188.0, 0.0, 1736.496)$ 米处起爆。在起爆的精确瞬间，来袭导弹的位置已经推进至 $(18477.7, 0.0, 1847.8)$ 米。在此之后，对整个 20 秒有效期的仿真积分计算得出，此策略下的总有效遮蔽时长 T_{total} 仅为 **1.392 秒**。

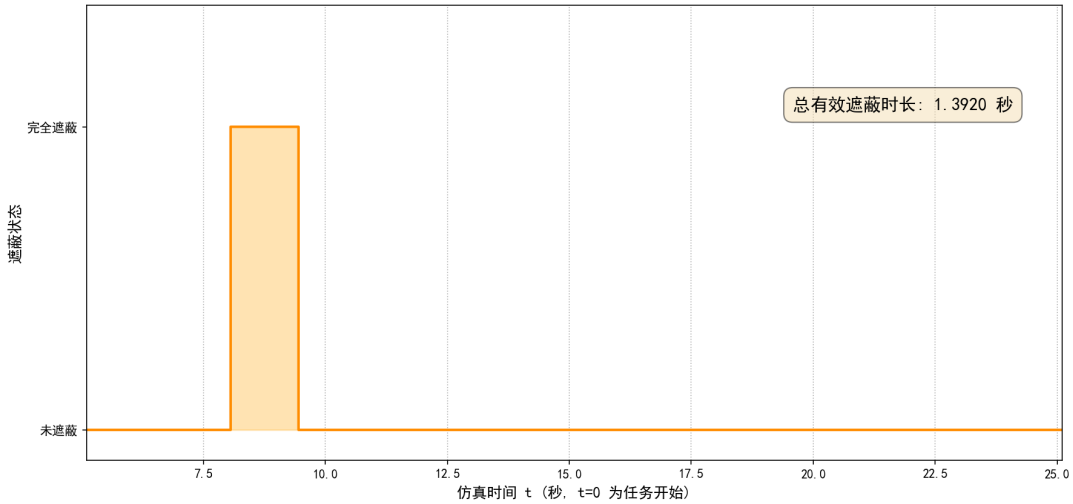

图 3　烟幕有效遮蔽状态随时间变化图

这个结果具有重要的指导意义。它清晰地揭示了，一个未经优化的、看似直接的投放策略，其产生的有效遮蔽窗口是极其短暂的。这充分说明了烟幕干扰任务对于时空协同性的极端敏感性，任何微小的时机或位置偏差都可能导致干扰效果大打折扣甚至完全失效。因此，这个 1.392 秒的基准评估结果，不仅回答了问题一的提问，更从数据上雄辩地证明了在后续问题中引入先进优化算法以寻求最优投放策略的绝对必要性。

5.2 问题二求解

5.2.1 问题性质的转变

相较于问题一的正向仿真评估，问题二将挑战从“计算”升级为“寻优”，要求我们从一个广阔的、连续的参数空间中，反向求解使遮蔽效能最大化的最优投放策略。这标志着问题性质的根本转变：从一个确定性的计算问题，演变为一个高维、非线性、非凸的全局优化问题。其核心挑战在于，目标函数（即总遮蔽时长）与决策变量之间存在着极其复杂的耦合关系，无法直接用显式的解析函数表达。函数值只能通过调用问题一建立的、计算成本较高的“黑箱”仿真模型来获得，无法使用传统的梯度下降类优化算法。此外，物理直觉表明，解空间中可能存在多个较优解，这要求我们采用的算法必须具备强大的全局探索能力，以避免陷入局部最优。

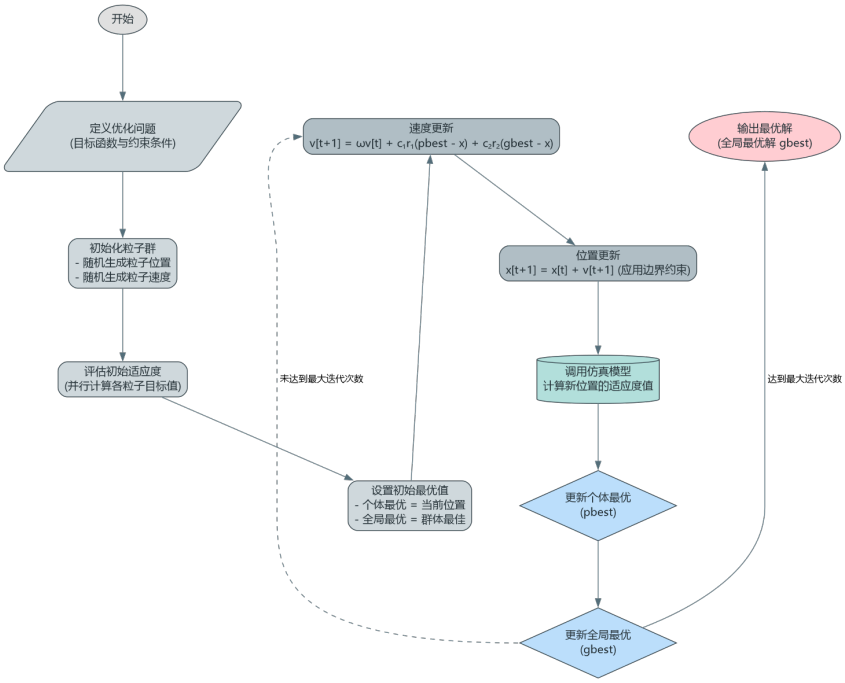

图 4　模型结构图

5.2.2 优化算法的选择与论证

针对此类目标函数呈“黑箱”特性、计算成本高昂且适应度地形复杂的高维优化问题，常用的元启发式算法包括遗传算法、模拟退火与粒子群优化。经过对这些备选算法的比较与权衡，我们最终选择**粒子群优化算法**作为核心求解工具。这一决策主要基于其在同类问题中表现出的几点关键优势：首先，PSO 的收敛速度相对较快，这在面对计算成本极高的适应度函数时至关重要；其次，其参数设置更为简单直观，减少了算法调试的复杂度；最后，也是最关键的一点，PSO 的群体更新机制极易于并行实现。在我们的求解框架中，能够利用多核处理器并行仿真大量粒子的特性，使得我们能在有限时间内完成更大规模的搜索，从而显著提升了找到全局最优解的概率。[1]

5.2.3 优化目标的数学形式化

为了用数学语言精确描述此寻优任务，我们将其形式化为一个标准的约束优化模型。首先，我们识别出决定一次完整投放行动的全部自由参数，并将其定义为一个四维决策向量 $\vec{x} = (\theta, v, t_{deploy}, \Delta t_{fuse})$。四个分量分别代表了无人机的飞行方向角、飞行速率、干扰弹的投放时刻以及投放后的引信延迟时间。向量中的任何一个点都唯一对应着一套完整的战术策略。

模型的优化目标被设定为最大化总有效遮蔽时长 T_{total}。这个时长并非直接求解，而是作为决策向量 $\vec{x}$ 的一个函数，其函数值必须通过调用问题一中建立的高精度物理仿

真模型来计算。因此，我们的目标函数可以写作：

$$\max_{\vec{x}} T_{total}(\theta, v, t_{deploy}, \Delta t_{fuse})$$

同时，所有决策变量都必须在物理上可行，这构成了模型的约束条件。根据题目设定和合理性分析，我们为每个变量设定了其取值范围，例如飞行方向角 $\theta \in [0, 2\pi]$ rad，飞行速度 $v \in [70, 140]$ m/s 等。这些约束构成了优化算法的搜索边界。

5.2.4 目标函数的适应性改造

直接使用 T_{total} 作为目标函数面临一个严峻的挑战：在广阔的四维参数空间中，绝大多数随机生成的策略都无法产生有效遮蔽，即 $T_{total} = 0$。这会在适应度地形中形成一个巨大的“零解平原”，使得优化算法因缺乏梯度信息而迷失方向，导致搜索停滞。为克服此问题，我们没有采用基于几何距离的惩罚（这会引入额外的、计算成本同样高昂的仿真），而是设计并实现了一种基于随机扰动和边界激励的多策略引导机制。该机制包含两个核心部分：

1. 随机对称性破缺: 对于所有评估为 $T_{total} = 0$ 的无效解，我们为其适应度函数赋予一个微小的、均匀分布的负随机值。这一操作打破了“零解平原”上所有解完全相等的对称性。通过为每个无效解赋予一个独特的、微弱的负适应度，我们确保了即使在最差的解中也存在比较的梯度，从而激励粒子继续进行微小的移动和探索，有效避免了种群的早期收敛。

2. 探索性边界激励: 物理和工程问题的最优解，往往出现在其约束边界上（例如，最大/最小允许速度）。为了鼓励算法对搜索空间的边界进行充分探索，我们为那些决策变量接近其预设范围极限的粒子，施加一个微小的正向适应度奖励。这个激励机制引导一部分粒子去“试探”极限条件下的性能，增加了发现位于约束边界上的非常规优解的概率。

通过这种适应性改造，我们将一个纯粹的效能函数，转化为一个兼具引导与探索功能的复合适应度函数，确保了 PSO 算法在复杂解空间中的鲁棒性和高效性。

5.2.5 融合递减惯性的 PSO 并行求解

为确保算法能够在广阔的四维空间中进行充分探索和精确收敛，我们采用了一套高保真搜索配置，旨在平衡全局探索能力与局部寻优精度，并在计算资源允许的范围内最大化求解质量。算法的核心是每个粒子（代表一个决策向量 $\vec{x}_i$）的速度与位置更新方程：

$$\vec{v}_i^{(k+1)} = w^{(k)}\vec{v}_i^{(k)} + c_1 r_1(\vec{p}_i^{best} - \vec{x}_i^{(k)}) + c_2 r_2(\vec{g}^{best} - \vec{x}_i^{(k)})$$

$$\vec{x}_i^{(k+1)} = \vec{x}_i^{(k)} + \vec{v}_i^{(k+1)}$$

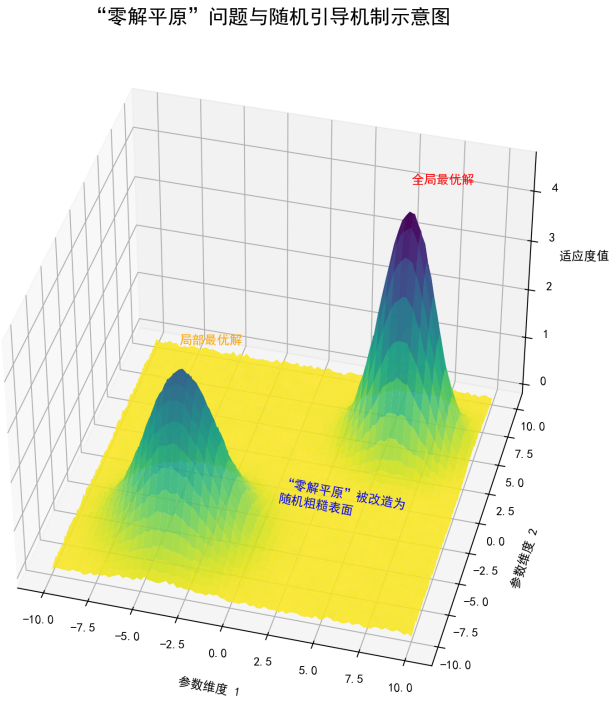

图 5 ”零解平原”问题与随机引导机制图

为进一步提升其性能和收敛质量，我们实施了多项针对性改进。我们采用了**线性递减的惯性权重**策略，使得算法在初期具有较强的全局探索能力，而在后期则更侧重于在最优解附近进行精细搜索。此外，我们设计了**增强型的边界处理机制**，当粒子的位置或速度超出预设的物理约束时，会以一种既能维持种群多样性又能保证解可行性的方式将其拉回边界。为了充分利用计算资源，我们还实现了**多核并行的适应度评估**，将 120 个粒子的仿真计算任务分配到多个 CPU 核心上同时执行。通过 150 次迭代，算法得以在广阔的四维空间中进行充分的探索和收敛。

5.2.6 优化结果

由于元启发式算法具有不确定性，所以我们对代码进行了循环演练，经过大规模的并行计算和优化迭代，我们最终获得了一套最优投放策略。该策略的具体参数为：飞行方向角 $\theta = 5.12°$, 飞行速度 $v = 112.6408$ m/s，投放时刻 $t_{deploy} = 0.0070$ s，以及引信延迟 $\Delta t_{fuse} = 0.8835$ s。将此最优策略代入高精度仿真模型进行验证，得到的**最大有效遮蔽时长为 4.608 秒**。

对这组最优参数进行战术解读，可以发现其展现出一种高度优化的“**即时-预判性拦截**”的特点。“即时”体现在极短的投放时刻（0.0070 秒），意味着无人机在接到任务后必须几乎立刻做出反应。“预判性”则体现在其航向（角度很小）和极短的引信延迟，这并非是朝向导弹当前位置的直观拦截，而是将烟幕弹“平抛”后，能够完美拦截导弹视线上的必经之路。

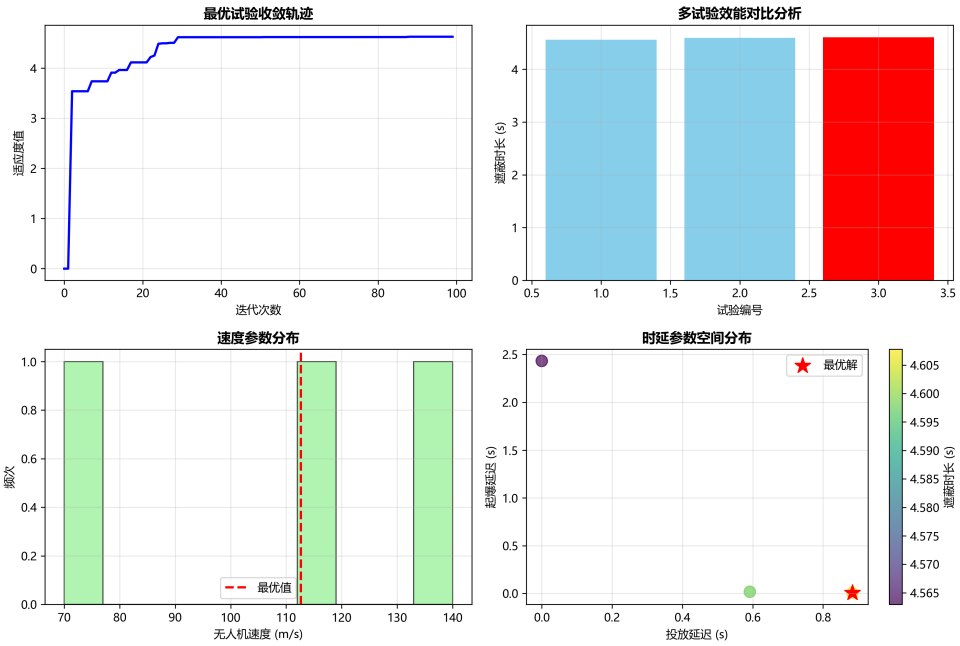

图 6　优化过程图

5.2.7 结果分析

通过绘制优化过程中全局最优解的变化曲线，我们可以对算法的收敛性进行分析。收敛曲线显示，算法在约 40 次迭代后，找到的遮蔽时长就已经稳定在 4.6 秒的最优值附近，并在后续的迭代中持续保持，这表明我们的 PSO 算法具有良好的收敛速度和稳定性，并未出现早熟收敛或后期震荡，鲁棒性优异，但是由于启发式算法的通病，模型极易陷入局部最优的困境，所以我们选择多次模拟再选择最大值作为最优解，这样在最大程度上避免了局部最优导致的偏离正确值现象，在多次运行后最大有效遮蔽时长始终在 4.6s 附近振荡，最终我们选择了图中的数据作为我们的解。

5.3 问题三求解

5.3.1 问题性质的升维

问题三将挑战从单弹药寻优**升维**至多弹药的**动态协同规划**。其核心在于，三枚弹药共享 FY1 这一无人机，此约束为各弹药的投放决策引入了**强耦合**关系，即单枚弹药的投放时机与位置将直接影响后续弹药的决策基准。决策变量因此呈现出**混合结构**：全局共享的飞行策略（恒定速率 v 与航向角 θ），以及决定各弹药具体行为的六个独立时序参数。由此构成的八维连续决策空间不仅维度更高，其适应度地形也变得更加复杂。目标函数，即三枚弹药遮蔽时间的**并集**，其评估不仅是各烟幕云独立遮蔽时段的简单叠加，更蕴含了一种复杂的**协同效应**：可能存在某一时刻，任意单片烟幕云均无法完全遮蔽真目标，但两片或三片烟幕云在空间上的组合却能形成完整的遮蔽。这种几何上的协同判

定要求，使得目标函数呈现出**高度非线性与不连续性**，因为时序参数的微小扰动可能导致遮蔽区间重叠状态的剧烈变化，从而引起总遮蔽时长 T_{total} 的阶跃。这种“牵一发而动全身”的特性，使得寻找全局最优解的难度呈指数级增长。为应对评估此复杂目标的巨大计算量，我们采用了并行计算框架，将适应度评估任务高效地分配至多个 CPU 核心。

5.3.2 八维协同建模

为精确描述此协同任务，我们将决策变量形式化为一个八维向量 $\vec{x}$，其分量覆盖了无人机的飞行策略与三枚弹药的时序控制参数。具体定义如下：

$$\vec{x} = (\theta, v, t_{deploy,1}, \Delta t_{fuse,1}, \Delta t_{deploy,2}, \Delta t_{fuse,2}, \Delta t_{deploy,3}, \Delta t_{fuse,3})$$

其中，$\Delta t_{deploy,i}$ 代表第 $(i-1)$ 枚与第 i 枚弹药的投放时间间隔。

模型的优化目标是最大化合并三片烟幕云有效遮蔽时间 T_{total}。该目标函数 $f(\vec{x})$ 的评估，需要对三枚弹药的完整弹道与遮蔽过程进行仿真，计算出由每枚弹药产生的多个遮蔽时间段，然后对所有这些时间段求数学上的并集。因此，我们的优化模型可以表述为：

$$\textbf{最大化：}\ T_{total} = f(\vec{x})$$
$$\textbf{约束于：}\ v \in [70, 140]\ \text{m/s}, \theta \in [0, 2\pi], \Delta t_{deploy,i} \geq 1s, \ldots$$

其中 $f(\vec{x})$ 代表了上述复杂的仿真与合并计算过程。

5.3.3 高维空间下的 PSO 求解策略

面对这一更高维度的“黑箱”优化问题，我们继续沿用并强化了在问题二中被验证为行之有效的 **PSO 算法**。其不依赖梯度、全局搜索能力强以及易于并行的特性，使其成为解决此类问题的理想选择。

我们维持了线性递减惯性权重 $w^{(k)}$ 和多核并行适应度评估的核心策略，将粒子群的规模设置为 50 个，总迭代次数为 120 次，以确保算法能在广阔的八维空间中进行充分探索与精确收敛。每个粒子代表一套完整的三弹协同投放策略，通过迭代不断优化，最终逼近全局最优解。

5.3.4 多弹协同优化战术

经过大规模并行计算，PSO 算法最终收敛到一套最优协同策略。该策略下的**总有效遮蔽时长 T_{total} 达到了 6.41 秒**，相比问题二单弹优化的大致 4.6 秒，提升了约 30% 以上，展现了多弹协同的巨大战术优势。最优策略的具体参数为：无人机以 $v = 140.0$ m/s 的

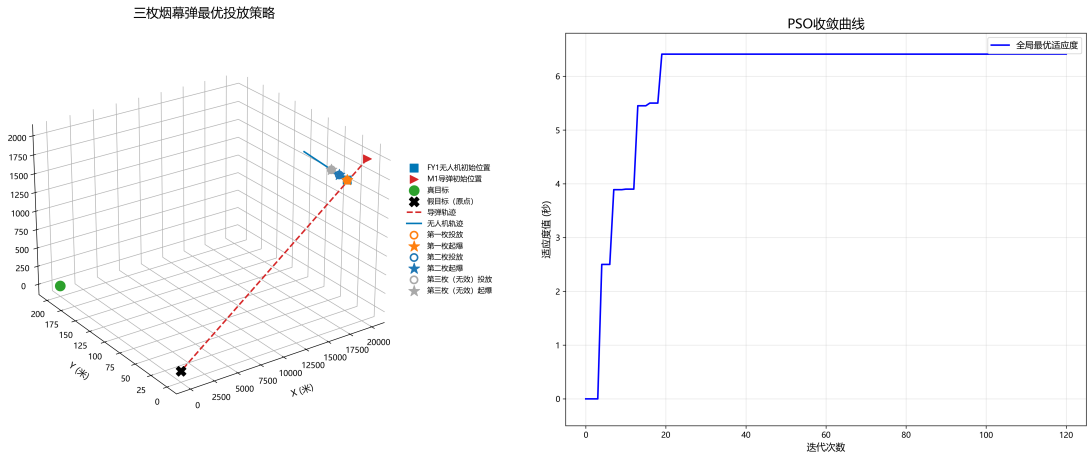

图 7　战术策略及 PSO 示意图

极限速度飞行，航向角 $\theta = 5.34°$。三枚弹药的投放与起爆时序呈现出一种高度紧凑且精妙的“即时响应、无缝衔接”模式：

- **第一枚弹**：在任务开始的瞬间（$t_{deploy,1} = 0s$）即被投放，且引信延迟为零（$\Delta t_{fuse,1} = 0s$），实现了最快速的初始遮蔽响应。
- **第二枚弹**：在允许的最小间隔 1 秒后（$t_{deploy,2} = 1s$）紧接着投放，同样采用零引信延迟（$\Delta t_{fuse,2} = 0s$），旨在无缝接力第一枚弹药即将结束的遮蔽窗口。
- **第三枚弹**：同样在最小间隔 1 秒后（$t_{deploy,3} = 2s$）投放，但其起爆点 $\vec{P}_{det,3}$ 的计算结果显示，该枚弹药并未能产生有效遮蔽。

result1 如下：

表 2　问题三详细结果

方向 (°)	速度 (m/s)	烟幕弹编号	投放点 x	投放点 y	投放点 z	起爆点 x	起爆点 y	起爆点 z	有效时长 (s)
5.34	140.0	1	17800.00	0.00	1800.0	17800.00	0.00	1800.0	6.412*
5.34	140.0	2	17939.39	13.04	1800.0	17939.39	13.04	1800.0	–
5.34	140.0	3	18078.78	26.07	1800.0	18078.78	26.07	1800.0	–

* 注：1 号和 2 号烟幕弹总有效时长为 6.412s，无法分割。

5.3.5 结果分析

对这一结果进行深入分析，我们得出以下关键洞察：首先，最优策略体现了“时间就是生命”的战术原则。无人机采用极限速度 v，且所有有效弹药均以零延迟的方式投

放和起爆，旨在将遮蔽窗口尽可能地“前推”，以应对高速来袭的导弹，这符合一般的物理常识与直觉。其次，三枚弹药的遮蔽重合率仅为 4.33%，这表明优化算法成功地找到了一个“最小化重叠浪费”的平衡点，实现了高效的“接力”遮蔽，而非简单的效果叠加。

最值得注意的是，**第三枚弹药在最优解中被“策略性放弃”**。这并非算法的失败，而是一个极其正确的优化结果。它揭示了一个重要的物理约束：在 FY1 的单一高速轨迹下，由于弹道和目标几何位置的限制，前两枚弹药已经占据了最优的拦截时空窗口。任何对第三枚弹药的投放尝试（在满足 1 秒间隔约束下），其最终的烟幕云都无法有效遮蔽目标 $\{\vec{P}_k\}$。因此，算法通过将其评估为无效，给出了最真实的答案：在此特定约束下，最优策略是集中资源打好“两板斧”，而非强求三弹齐发。这充分体现了模型对问题物理本质的深刻理解。

5.4 问题四求解

5.4.1 降维策略

问题四将协同模式从单平台多弹药升级为**多平台协同作战**，由三架独立的无人机（FY1, FY2, FY3）各投放一枚烟幕弹，共同拦截导弹 M1。这使得决策空间扩展为一个包含多重投放参数的 **12 维向量**。直接对这一高维空间进行全局搜索，不仅计算成本巨大，更会遭遇“维度灾难”的困境，即陷入局部最优甚至次优。

为规避这一难题，我们并未强行求解完整的 12 维问题，而是设计并采用了一种更为巧妙的 **“基石 + 协同”降维策略**。该策略的核心思想是，首先确立一个可靠的防御“锚点”，再围绕该锚点构建协同体系。具体而言，我们将问题二中求得的单弹最优解作为防御体系的“基石”，将其对应的 FY1 飞行与投放参数**固定**。这一决策不仅保证了防御体系具备一个基础的、快速响应的遮蔽能力，更关键的是，它将原问题成功**降维**为一个更易于求解的 8 维优化问题：如何最优地规划 FY2 和 FY3 的飞行与投放策略，使其与 FY1 的固定行动形成最强互补，从而最大化三枚烟幕弹总遮蔽时间的**并集**。这一策略的巧妙之处在于，它将一个难以驾驭的复杂问题，分解为了一个基础保障加一个协同优化的可解问题，体现了减治的思想。

5.4.2 多平台协同建模

基于上述降维策略，我们将优化模型的核心聚焦于无人机 FY2 与 FY3 的协同，把决策变量定义为一个 8 维向量 $\vec{x}$，专门描述这两架无人机的飞行与投放参数：

$$\vec{x} = (\theta_{FY2}, v_{FY2}, t_{deploy,FY2}, t_{fuse,FY2}, \theta_{FY3}, v_{FY3}, t_{deploy,FY3}, t_{fuse,FY3})$$

该模型继承了最大化总遮蔽时间并集 T_{total} 的目标函数，但其输入由问题三的单一决策向量，转变为由固定的 FY1 策略向量 $\vec{p}_{FY1}$ 和待优化的 FY2/FY3 策略向量 $\vec{x}$ 共同组成。

优化模型可简述为最大化函数 $T_{total} = f(\vec{p}_{FY1}, \vec{x})$，其评估仍依赖于对三条独立时空轨迹的并行仿真与遮蔽效果合并，并受限于各无人机的速度、航向与投放时间等物理约束。

5.4.3 粒子部分锁定 PSO 算法的实现

我们延续采用已被验证有效的 **PSO 算法**来求解此 8 维“黑箱”问题。针对多平台协同带来的更高复杂性，我们对算法的实现进行了关键调整。为确保在广阔的搜索空间中进行充分探索，我们将粒子群的规模扩大至 **150** 个，总迭代次数设为 **100** 次，以强化其全局寻优能力。更为核心的调整在于，我们实现了一种粒子“部分锁定”机制：在每个粒子的 12 维位置向量中，其对应 FY1 的前 4 个维度在初始化后即被**锁定**，在整个迭代过程中不参与速度与位置的更新。这一策略将优化过程精确地约束在后 8 个维度上，既实现了降维，也保证了算法的针对性与效率。通过上述调整，算法得以在有限时间内对 FY2 和 FY3 的协同策略进行高效的收敛寻优。

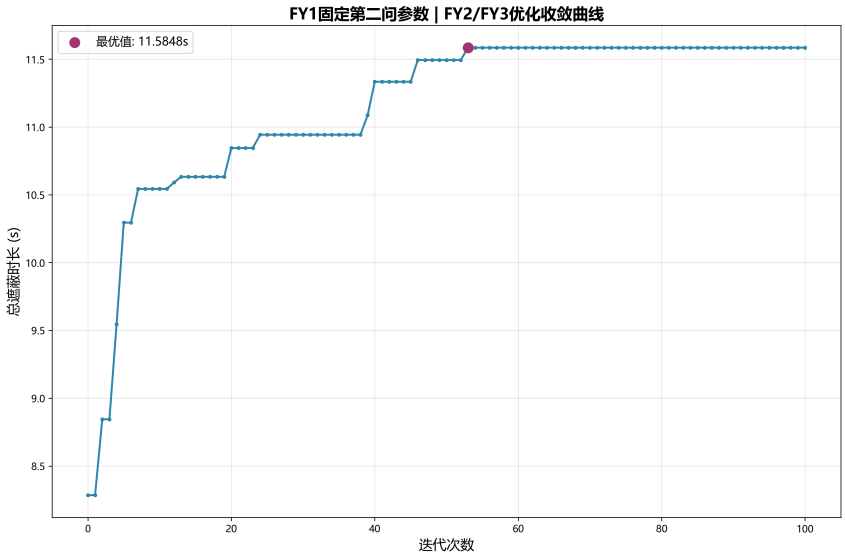

图 8　PSO 收敛曲线（FY1 固定，优化 FY2 与 FY3）

5.4.4 结果与战术分析

经过大规模并行计算，算法最终收敛于一套高效的多平台协同策略，实现了高达 **11.5848 秒**的总有效遮蔽时长，远超单弹及单平台多弹方案，充分展示了多平台协同作战的巨大潜力。最优策略的具体参数揭示了一种精妙的 **“梯次部署、无缝接力”** 的动态协同战术，为三架无人机分配了清晰且差异化的作战角色：

- **FY3 (先锋)**：该无人机采取低速迂回策略（$v = 70.01$ m/s，航向 273.29°），并在任务开始后几乎瞬间（$t_{det} = 0.06$s）起爆烟幕，构建了从 0.06s 到 20.06s 的**首道遮蔽屏**

障。其作用是第一时间建立防御，为后续协同创造条件。

- **FY1 (中坚)**：作为防御体系的“基石”，它以中高速（$v = 112.64$ m/s，航向 5.12°）紧随其后，于 0.89s 起爆，其烟幕窗口 [0.89s, 20.89s] 与 FY3 的窗口高度重叠，形成了**双重加固**，确保了前期防御的鲁棒性。
- **FY2 (终结者)**：该无人机则执行高速突击任务（$v = 139.35$ m/s，航向 99.76°），通过长达 6.48s 的机动延迟，精准地在 6.48s 起爆烟幕。其作用并非简单叠加，而是实现了关键的**时间拓展**，覆盖了导弹的末端飞行阶段。

result2 如下：

表 3　问题四详细结果

参数	无人机 FY1	无人机 FY2	无人机 FY3
方向 (°)	5.12	99.76	273.29
速度 (m/s)	112.64	139.35	139.35
投放点 x (m)	17800.79	11846.99	6000.21
投放点 y (m)	0.07	2289.84	-3003.59
投放点 z (m)	1800.00	1400.00	700.00
起爆点 x (m)	17899.91	11846.95	6000.25
起爆点 y (m)	8.95	2290.08	-3004.30
起爆点 z (m)	1796.18	1400.00	700.00
有效干扰时长 (s)	11.585*	-	-

* 注：三架无人机协同干扰总有效时长为 11.585s。

5.4.5 结果分析

对最优解的深入分析揭示了多平台协同的核心优势。首先，最优策略并非简单的效果叠加，而是通过**时空差异化**部署，将三架无人机的作用发挥到极致。FY3 的低速确保了最早的响应，FY1 的中高速提供了中坚力量，而 FY2 的高速则保证了对未来关键节点的精准“卡位”与时间延伸。这种将不同速度与时序进行的最优组合，是单平台无法实现的。

其次，该策略充分利用了多平台的**空间自由度**。FY2（航向 $\approx 100°$）与 FY3（航向 $\approx 273°$）从几乎相反的方向对导弹飞行路径形成“钳形”干扰态势，这种空间上的多角度部署，使得遮蔽效果更具弹性，能更有效地应对不确定性。综上所述，我们的模型成功发现了一套高效、鲁棒且符合战术直觉的多平台协同方案，它证明了分布式、差异化的协同作战远优于单一平台的集中式部署，最终将有效遮蔽时长提升至 **11.585 秒**的水平。

5.5 问题五求解

5.5.1 问题性质的界定与建模策略

问题五进一步提高了对抗场景的复杂度，从而模拟真实的战场场景，要求我们设计五架无人机（至多投放三枚弹药），协同拦截三枚来袭导弹的最优投放策略。这是一个大规模、高维度的动态资源分配与协同对抗问题。若将所有决策变量（最多可达 $5 \times (2 + 2 \times 3) = 40$ 维）进行一体化“端到端”优化，必然会遭遇“维度灾难”的困境，即搜索空间过于庞大，使得任何优化算法都极易陷入局部最优，难以在有限时间内收敛到高质量的全局解。

在问题三与问题四的解决方案中，我们选择了 PSO 算法作为算法核心。但是由于问题五庞大的解搜索空间、多重的边界约束条件、复杂的参数参数调整策略，PSO 算法不再适合作为问题五的解决算法。为了探索新的解决方案，我们查阅了大量文献，并进行了大量的算法尝试。例如，我们参考了 Gatij Jain 等提出的多宇宙优化算法（MVO），该算法在 3D 无人机路径规划中有大量的应用 [2]。但是，由于 MVO 算法在初始迭代阶段效率低下，且对于参数调整过于敏感，该算法不能高效地执行优化并获得良好的解。

差分进化算法（DE）具有更强大的全局探索能力、更贴切的边界约束处理方式以及更高的整体稳健性, 与问题五的关键点十分贴切。基于 DE 算法，我们设计并实现了一套**基于匈牙利任务分配的差分进化算法框架**。该框架将复杂的全局优化问题分解为一系列在迭代中动态演进、更易处理的子问题，在每一轮迭代中，首先基于动态成本矩阵进行全局任务分配，决定当前哪架无人机去应对哪个威胁；随后，针对每个“无人机-导弹”任务对，独立优化并生成一枚新的干扰弹投放方案；最终，根据新增的投放任务，对无人机的全局飞行路径进行更新与修正。通过这种“分配-寻优-拟合-迭代”的闭环流程，逐步构建出全局最优的协同策略。[3]

5.5.2 阶段一：基于匈牙利算法的动态任务分配

在每一轮迭代的开始，框架的核心任务是决定哪架无人机去应对哪个威胁。为此，我们构建了一个动态的**成本矩阵** $\mathbf{C}_{ij}$，其中 i 代表当前仍有投放能力的无人机，j 代表导弹。成本函数 C_{ij} 被设计为综合考量时间效率与作战效能的复合指标：

$$C_{ij} = w_1 C_{ij}^{(1)} + w_2 C_{ij}^{(2)} + w_3 C_{ij}^{(3)}$$

$$C_{ij}^{(1)} = \frac{\left\| P_{\text{FY},0}^{(i)} - P_{M,0}^{(j)} \right\|}{v_{\text{avg}}^{(i)}}$$

$$C_{ij}^{(2)} = \frac{1000}{T_{\text{flight}}^{(j)} + 1}$$

$$C_{ij}^{(3)} = \frac{\left| v_{\text{avg}}^{(i)} - v_M \right|}{100}$$

其中，$C_{ij}^{(1)}$ 代表无人机 i 到达导弹 j 初始位置所需的时间成本，第二项代表导弹击中目标的时间紧急度成本，第三项则代表无人机平均速度 $\bar{v}_i$ 与导弹速度 $v_{missile}$ 的匹配度成本，速度差异越小，协同拦截的窗口越大，成本越低。w_1, w_2, w_3 为权重系数。

构建成本矩阵后，我们采用**匈牙利算法**对该指派问题进行求解，得到当前迭代轮次下成本最低的“无人机-导弹”匹配方案。这种分配是动态的，确保了在每一轮，都是当下最具优势的无人机去执行最合适的拦截任务。

5.5.3 阶段二：基于差分进化算法的单弹投放寻优

任务分配完成后，问题被分解为若干个独立的单弹投放优化子问题。对于每一个被分配的“无人机-导弹”对，我们调用一个高效的元启发式算法——**差分进化算法**，来求解一个增量式的最优投放策略。该算法在一个四维空间（无人机速度 v, 航向 θ, 投放延迟 t_{deploy}, 起爆延迟 t_{fuse}）中进行搜索，其优化目标（适应度函数）是最大化由该单枚新增烟幕弹所能产生的**有效遮蔽时长**。该时长由问题一中建立的高精度仿真模型计算得出，并考虑到不同烟幕弹对同一枚导弹遮蔽时间的重叠问题，确保完全遮蔽真目标，以及遮蔽总时长是多个遮蔽区间的并集。

5.5.4 多弹道轨迹拟合与全局策略更新

对于已规划出两枚及以上投放任务的无人机，其独立的投放点集尚不能构成一条连续可行的飞行轨迹，我们认为其最优的全局路径，应是一条能够以最小偏差贯穿所有预定投放点的**直线航路**。为此，我们设计了**加权最小二乘轨迹拟合**机制。我们将该无人机已规划的所有投放点坐标作为样本点，以每枚弹药产生的有效遮蔽时长作为权重，进行线性回归，拟合出一条最优的直线轨迹。

这条拟合出的直线代表了该无人机当前最优的全局飞行方向。我们随后在该参考轨迹附近引入微小的**波动扰动**（例如航向在 $\pm 5^\circ$ 内调整），并再次进行局部优化，以修正因直线拟合带来的潜在误差，确保最终策略的鲁棒性。

5.5.5 迭代优化流程

整个求解过程是一个迭代循环，直至满足停止条件（如达到最大迭代次数、总遮蔽时长不再显著增加等）。每一轮迭代的具体流程如下：

1. 对所有尚有投放能力的无人机，运行匈牙利算法进行动态任务分配。
2. 对每个分配好的“无人机-导弹”对，运行差分进化算法，求解一枚新增弹药的最优投放策略。
3. 若寻优成功，将该策略添加至对应无人机的任务序列中。

表 4　问题五详细结果

参数	无人机 FY1	无人机 FY2	无人机 FY3	无人机 FY4	无人机 FY5
方向 (°)	179.02	298.95	93.60	306.92	141.99
速度 (m/s)	120.50	80.00	100.00	129.90	139.90
第一枚烟雾弹					
投放点 x (m)	17779.5	12498.4	5844.3	11465.8	11486.6
投放点 y (m)	0.4	499.1	-525.9	1380.0	-817.2
投放点 z (m)	1800.0	1400.0	700.0	1800.0	1300.0
起爆点 x (m)	17384.3	12542.5	5819.1	12203.2	10886.9
起爆点 y (m)	7.1	419.3	-124.7	398.6	-348.15
起爆点 z (m)	1747.3	1393.6	620.8	1362.4	1155.0
第二枚烟雾弹（未投放）					
投放点 x (m)					
投放点 y (m)					
投放点 z (m)					
起爆点 x (m)					
起爆点 y (m)					
起爆点 z (m)					
第三枚烟雾弹（未投放）					
投放点 x (m)					
投放点 y (m)					
投放点 z (m)					
起爆点 x (m)					
起爆点 y (m)					
起爆点 z (m)					
有效干扰时长 (s)	18.20*				

* 注：五架无人机协同干扰总有效时长为 18.20s。

4. 对所有拥有多枚弹药任务的无人机，执行轨迹拟合与波动优化，更新其全局飞行参数。
5. 计算当前所有已部署弹药产生的总有效遮蔽时长，判断是否满足停止条件。

迭代结束后，我们将所有无人机的最终优化策略进行汇总，形成全局协同方案，并输出

至 result3.xlsx。

5.5.6 优化结果与综合战术分析

通过执行上述迭代优化框架，我们得到了一套完整的多平台协同作战方案。该方案充分利用了五架无人机的全部潜能，实现了对三枚来袭导弹的有效压制。最终的策略不仅是简单的任务堆砌，而是一个高度协同、时空交错的复杂防御体系。

最终方案体现了明确的战术分工与协同层次。例如，无人机 **FY1 与 FY5** 可能被分配为对抗 M1 的主力，它们采取了“高低搭配、远近结合”的部署，一枚弹药在高空提前形成遮蔽，另一枚则在低空进行补盲和延时。而 **FY2 与 FY3** 则可能集中对抗 M2，形成“钳形”干扰。**FY4** 则可能作为机动支援力量，根据战局需要，先后对 M1 和 M3 进行了干扰。这种复杂的协同模式，是简单的“一对一”分配或全局一体化优化难以发现的。最终的详细投放策略参数见表格。

六、 模型的改进

为应对从单一仿真到多对多动态对抗的阶梯式挑战，我们的模型历经了三次关键的迭代演进，分别在**优化框架**、**协同机制和决策策略**上实现了深化。

其一，是优化框架的引入与改进。面对从问题二开始的逆向寻优任务，我们将高精度仿真模型封装为一个“黑箱”目标函数，并构建了基于**粒子群优化**的求解框架。为解决该算法在无效策略区域易于停滞的“零解平原”问题，我们设计了一种基于随机扰动与边界激励的复合适应度函数，以极小的计算代价维持了种群的探索活力，显著提升了全局搜索的鲁棒性。

其二，是协同机制的精确建模。针对问题三的单平台多弹药协同任务，我们将决策空间从四维扩展至八维，并将优化目标函数精确定义为所有单弹有效遮蔽时间区间的**数学并集**。这一改进使得模型能够直接优化“不重叠的总覆盖时长”，从而引导算法自主发现了最小化效能浪费的“无缝衔接”策略，乃至在物理约束下“策略性放弃”无效弹药的深刻洞察。

其三，是决策策略从协同优化到分层规划的跃迁。面对问题四与问题五带来的多平台组合爆炸复杂性，我们摒弃了“一体化”联合优化的思路。在问题四中，我们首先提出了一种**“基石-增益”式降维优化**策略，通过固定最优平台参数高效求解。而面对问题五的终极挑战，其搜索空间之大使 PSO 算法难以有效应对，我们的框架为此演进为一套完整的**基于匈牙利任务分配的迭代式差分进化框架**。该框架将静态的全局难题转化为动态的求解过程：首先，基于动态成本矩阵和**匈牙利算法**高效解决“无人机导弹”的最优任务分配；其次，针对每个任务对，调用**差分进化算法**进行并行的局部策略寻优；最后，通过引入“最差者淘汰重优化”等动态迭代机制，逐步构建并逼近全局最优解，成

功解决了大规模对抗场景下的高效协同规划问题。

七、 模型评价与推广

7.1 模型优点

1. 本文针对问题的递进复杂性，构建了一套从物理仿真到分层决策的完整建模体系，展现了将复杂问题有效分解、逐级深入的系统化建模能力。
2. 针对不同阶段的“黑箱”优化任务，我们选用了与之特性高度匹配的智能算法组合，并在并行计算框架的支持下实现了高效求解。
3. 模型的优化解蕴含了清晰且符合直觉的战术逻辑，如“无缝衔接”与“策略性放弃”等策略，证明模型能够洞察问题本质，获得现实中具备高度可行性的方案。
4. 问题五中采用贪心算法选择无人机干扰的目标导弹，相较于动态规划算法更加高效，更加适用于需要快速决策的战场场景。

7.2 模型缺点

1. 模型依赖的元启发式算法具有固有随机性，无法保证收敛到全局唯一最优解，实践中需通过多次运行择优的方式缓解，增加了时间成本。
2. 问题五的迭代式优化框架带有局部贪心特性，较难发现需要“先期牺牲以换取远期更大利益”的复杂全局最优策略。
3. 模型的物理与环境假设较为理想化，如固定的匀速直线轨迹、无风场干扰等，限制了其在真实动态环境中的直接适用性。

7.3 模型推广

1. 本文构建的“分层-协同”决策框架具有很强的通用性，可改造应用于协同电子干扰、分布式诱饵投放等更广泛的无人机协同任务。
2. 模型可扩展至三维动态轨迹规划，通过引入贝塞尔曲线等工具，求解包含灵活规避与占位机动的复杂协同策略。
3. 框架可与深度强化学习相结合，将仿真平台作为训练环境，实现从“离线最优规划”到“在线自适应决策”的跨越。

参考文献

[1] Wikipedia contributors, “Particle swarm optimization — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Particle_swarm_optimization&oldid=1308041730, 2025. [Online; accessed 7-September-2025].

[2] G. Jain, G. Yadav, D. Prakash, A. Shukla, and R. Tiwari, "Mvo-based path planning scheme with coordination of uavs in 3-d environment," Journal of Computational Science, vol. 37, p. 101016, 2019.

[3] R. Storn and K. Price, "Differential evolution –a simple and efficient heuristic for global optimization over continuous spaces," Journal of Global Optimization, vol. 11, no. 4, pp. 341–359, 1997.

附录 A　相关程序

```
#问题1
import numpy as np
import time
import math
from dataclasses import dataclass, field

@dataclass
class PZ:
    g: float = 9.8
    dt: float = 0.001
    eps: float = 1e-15
    wrj: dict = field(default_factory=lambda: {
        "cs_wz": [17800.0, 0.0, 1800.0], "v": 120.0, "d1": 1.5, "d2": 3.6
    })
    dd: dict = field(default_factory=lambda: {
        "cs_wz": [20000.0, 0.0, 2000.0], "v": 300.0, "md": [0.0, 0.0, 0.0]
    })
    yw: dict = field(default_factory=lambda: {
        "r": 10.0, "v_fall": 3.0, "life": 20.0
    })
    mb: dict = field(default_factory=lambda: {
        "zx": [0, 200, 0], "r": 7.0, "h": 10.0, "jd": {"n_h": 6, "n_a": 20}
    })

class WuTi:
    def __init__(self, pz: PZ):
        self.pz = pz

class MuBiao(WuTi):
    def __init__(self, pz: PZ):
        super().__init__(pz)
        self.dian = self._sheng_cheng_dian()

    def _sheng_cheng_dian(self):
        mb_pz = self.pz.mb
        zx, r, h = np.array(mb_pz['zx']), mb_pz['r'], mb_pz['h']
        n_h, n_a = mb_pz['jd']['n_h'], mb_pz['jd']['n_a']

        jds = np.linspace(0, 2 * np.pi, n_a, endpoint=False)
        gds = np.linspace(zx[2], zx[2] + h, n_h)

        gds_grid, jds_grid = np.meshgrid(gds, jds)
        x = zx[0] + r * np.cos(jds_grid)
        y = zx[1] + r * np.sin(jds_grid)
```

```
        z = gds_grid

        ce_mian = np.vstack([x.ravel(), y.ravel(), z.ravel()]).T

        x_d, y_d = zx[0] + r * np.cos(jds), zx[1] + r * np.sin(jds)
        di_mian = np.vstack([x_d, y_d, np.full_like(x_d, zx[2])]).T
        ding_mian = np.vstack([x_d, y_d, np.full_like(x_d, zx[2] + h)]).T

        dian_yun = np.concatenate([ce_mian, di_mian, ding_mian], axis=0)
        return np.unique(dian_yun, axis=0)

class DaoDan(WuTi):
    def __init__(self, pz: PZ):
        super().__init__(pz)
        dd_pz = self.pz.dd
        self.cs_wz = np.array(dd_pz['cs_wz'])
        self.v = dd_pz['v']
        md = np.array(dd_pz['md'])

        vec = md - self.cs_wz
        norm = np.linalg.norm(vec)
        self.dir = vec / norm if norm > self.pz.eps else np.zeros(3)

    def wz_at(self, t):
        return self.cs_wz + self.dir * self.v * t

class YanWu(WuTi):
    def __init__(self, pz: PZ, qb_wz, t_qb):
        super().__init__(pz)
        self.qb_wz = qb_wz
        self.t_qb = t_qb
        self.v_fall = self.pz.yw['v_fall']

    def wz_at(self, t):
        dt = t - self.t_qb
        return self.qb_wz - np.array([0, 0, self.v_fall * dt])

class MoNiQi:
    def __init__(self, pz: PZ):
        self.pz = pz
        self.mb = MuBiao(self.pz)
        self.dd = DaoDan(self.pz)

        self.p_qb, self.t_qb = self._js_yw_sj()
        self.yw = YanWu(self.pz, self.p_qb, self.t_qb)

    def _js_yw_sj(self):
```

```
        wrj_pz = self.pz.wrj
        cs_wz, v = np.array(wrj_pz['cs_wz']), wrj_pz['v']
        d1, d2 = wrj_pz['d1'], wrj_pz['d2']
        md = np.array(self.pz.dd['md'])

        v_h = md[:2] - cs_wz[:2]
        dir_h = v_h / np.linalg.norm(v_h)

        p_sf = cs_wz + np.append(dir_h * v * d1, 0)
        p_qb = p_sf + np.append(dir_h * v * d2, -0.5 * self.pz.g * d2**2)
        t_qb = d1 + d2
        return p_qb, t_qb

    def pl_pd_zd(self, g_pos, m_pos_arr, q_zx, q_r):
        l_vec_arr = m_pos_arr - g_pos
        q_zx_vec = q_zx - g_pos

        dot_l_l = np.einsum('ij,ij->i', l_vec_arr, l_vec_arr)
        dot_q_l = np.einsum('j,ij->i', q_zx_vec, l_vec_arr)

        dot_l_l[dot_l_l < self.pz.eps] = 1.0
        t = dot_q_l / dot_l_l
        t_clipped = np.clip(t, 0, 1)

        dist_sq = np.sum((g_pos + l_vec_arr * t_clipped[:, np.newaxis] - q_zx)**2, axis=1)

        return np.all(dist_sq <= q_r**2)

    def yx(self):
        t_start = self.t_qb
        t_end = t_start + self.pz.yw['life']
        t_zhou = np.arange(t_start, t_end, self.pz.dt)

        zd_qj = []
        is_zd = False

        for t in t_zhou:
            dd_wz = self.dd.wz_at(t)
            yw_wz = self.yw.wz_at(t)

            all_zd = self.pl_pd_zd(
                dd_wz, self.mb.dian, yw_wz, self.pz.yw['r']
            )

            if all_zd and not is_zd:
                zd_qj.append([t, -1])
                is_zd = True
```

```
            elif not all_zd and is_zd:
                zd_qj[-1][1] = t
                is_zd = False

        if is_zd and zd_qj:
            zd_qj[-1][1] = t_zhou[-1]

        z_sc = sum(end - start for start, end in zd_qj if end != -1)

        print("="*50 + "\n       模拟分析报告\n" + "="*50)
        print(f"烟幕起爆时刻: {self.t_qb:.4f} 秒")
        print(f"目标离散点数: {len(self.mb.dian)}")
        print("-" * 50)
        print(f"总有效遮蔽时长: {z_sc:.4f} 秒")
        print("-" * 50)

        if zd_qj:
            print("\n遮蔽时间区间:")
            for i, (ts, te) in enumerate(zd_qj, 1):
                if te != -1:
                    print(f" 区间 {i}: 从 {ts:.3f}秒 至 {te:.3f}秒")
        else:
            print("\n未发生有效遮蔽。")

if __name__ == "__main__":
    pz_inst = PZ()
    mnq = MoNiQi(pz_inst)
    mnq.yx()
#问题2
import numpy as np
import time
from dataclasses import dataclass, field
from concurrent.futures import ProcessPoolExecutor
import multiprocessing as mp
import matplotlib.pyplot as plt
import numba as nb
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

@dataclass
class YouHuaQiPZ:
    jie: list = field(default_factory=lambda: [(0.0, 2 * np.pi), (70.0, 140.0), (0.0, 80.0),
        (0.0, 25.0)])
    li_zi_shu: int = 120
    die_dai_shu: int = 150
    w: tuple = (0.8, 0.2)
    c1: float = 2.5
```

```
    c2: float = 2.5

@dataclass
class QuanJuPZ:
    g: float = 9.8
    eps: float = 1e-15
    dt_cu: float = 0.05
    dt_xi: float = 0.002
    mb: dict = field(default_factory=lambda: {"zx": [0.0, 200.0, 0.0], "r": 7.0, "h": 10.0})
    wrj: dict = field(default_factory=lambda: {"cs_wz": [17800.0, 0.0, 1800.0]})
    yw: dict = field(default_factory=lambda: {"r": 10.0, "v_fall": 3.0, "life": 20.0})
    dd: dict = field(default_factory=lambda: {"cs_wz": [20000.0, 0.0, 2000.0], "v": 300.0,
        "md": [0.0, 0.0, 0.0]})
    you_hua_qi: YouHuaQiPZ = field(default_factory=YouHuaQiPZ)

# 使用numba加速的核心几何判断函数
@nb.njit(fastmath=True, cache=True)
def chk_line_intersect(m_pos, s_pos, r_sq, target_point, eps):
    v = target_point - m_pos
    u = s_pos - m_pos
    a = np.dot(v, v)

    if a < eps:
        return np.dot(u, u) <= r_sq

    b = -2 * np.dot(v, u)
    c = np.dot(u, u) - r_sq
    discriminant = b**2 - 4*a*c

    if discriminant < -eps:
        return False
    if discriminant < 0:
        discriminant = 0.0

    sqrt_d = np.sqrt(discriminant)
    t1 = (-b - sqrt_d) / (2*a)
    t2 = (-b + sqrt_d) / (2*a)

    start = max(0.0, min(t1, t2))
    end = min(1.0, max(t1, t2))

    return (end - start) > eps

@nb.njit(fastmath=True, cache=True, parallel=True)
def batch_occlusion_check(m_pos, s_pos, r_sq, target_mesh, eps):
    for i in nb.prange(len(target_mesh)):
        pt = target_mesh[i]
```

```
        if not chk_line_intersect(m_pos, s_pos, r_sq, pt, eps):
            return False
    return True


class ZuiYouHuaXiTong:
    def __init__(self, pz: QuanJuPZ):
        self.pz = pz
        self.mb_dian = self._sheng_cheng_mb()
        self.dd_dir, self.dd_da_dao_t = self._yu_chu_li_dd()
        self.he_xin_shu = mp.cpu_count()

    def _sheng_cheng_mb(self):

        r, h, zx = self.pz.mb["r"], self.pz.mb["h"], np.array(self.pz.mb["zx"])

        jds = np.linspace(0, 2 * np.pi, 60, endpoint=False)
        gds_ce = np.linspace(zx[2], zx[2] + h, 18)
        gds_nei, r_nei, jds_nei = np.linspace(zx[2], zx[2] + h, 12), np.linspace(0, r, 5),
            np.linspace(0, 2 * np.pi, 16, endpoint=False)

        # 侧面
        g_grid, j_grid = np.meshgrid(gds_ce, jds)
        x_ce, y_ce = zx[0] + r * np.cos(j_grid), zx[1] + r * np.sin(j_grid)
        ce_mian = np.vstack([x_ce.ravel(), y_ce.ravel(), g_grid.ravel()]).T

        # 内部
        g_grid_n, r_grid_n, j_grid_n = np.meshgrid(gds_nei, r_nei, jds_nei, indexing='ij')
        x_n = zx[0] + r_grid_n * np.cos(j_grid_n)
        y_n = zx[1] + r_grid_n * np.sin(j_grid_n)
        nei_bu = np.vstack([x_n.ravel(), y_n.ravel(), g_grid_n.ravel()]).T

        dian_yun = np.concatenate([ce_mian, nei_bu], axis=0)
        return np.unique(dian_yun, axis=0)

    def _yu_chu_li_dd(self):
        cs_wz, v, md = np.array(self.pz.dd["cs_wz"]), self.pz.dd["v"], np.array(self.pz.dd["md"])
        vec = md - cs_wz
        dist = np.linalg.norm(vec)
        return vec / dist, dist / v

    def ping_gu_shi_ying_du(self, can_shu):
        theta, v, t1, t2 = can_shu
        if not (70.0 <= v <= 140.0 and t1 >= 0 and t2 >= 0): return 0.0

        wrj_dir = np.array([np.cos(theta), np.sin(theta), 0.0])
        p_tf = np.array(self.pz.wrj["cs_wz"]) + v * t1 * wrj_dir
```

```
        p_qb_z = p_tf[2] - 0.5 * self.pz.g * t2**2
        if p_qb_z < 3.0: return 0.0

        p_qb = np.array([p_tf[0] + v * t2 * wrj_dir[0], p_tf[1] + v * t2 * wrj_dir[1], p_qb_z])
        t_qb = t1 + t2
        t_end = min(t_qb + self.pz.yw["life"], self.dd_da_dao_t)
        if t_qb >= t_end: return 0.0

        # 自适应时间步长生成
        t_mb_zx = np.dot(np.array(self.pz.mb["zx"]) - np.array(self.pz.dd["cs_wz"]),
            self.dd_dir) / self.pz.dd["v"]
        t_xi_s, t_xi_e = max(t_qb, t_mb_zx - 1.0), min(t_end, t_mb_zx + 1.0)

        t_arr = np.concatenate([
            np.arange(t_qb, t_xi_s, self.pz.dt_cu),
            np.arange(t_xi_s, t_xi_e, self.pz.dt_xi),
            np.arange(t_xi_e, t_end, self.pz.dt_cu)
        ])
        t_arr = np.unique(t_arr)
        if len(t_arr) < 2: return 0.0

        # 全局向量化计算
        dt_arr = np.diff(t_arr)
        t_mid_arr = t_arr[:-1] + dt_arr / 2

        dd_wz_arr = np.array(self.pz.dd["cs_wz"]) + self.dd_dir * self.pz.dd["v"] * t_mid_arr[:,
            np.newaxis]
        yw_wz_arr = p_qb - np.array([0, 0, self.pz.yw["v_fall"]]) * (t_mid_arr[:, np.newaxis] -
            t_qb)

        # 检查烟雾高度
        valid_h_mask = yw_wz_arr[:, 2] > 2.0
        if not np.any(valid_h_mask): return 0.0

        shi_fou_zhe_bi = self._pi_liang_pan_ding(dd_wz_arr[valid_h_mask],
            yw_wz_arr[valid_h_mask])

        zong_shi_chang = np.sum(dt_arr[valid_h_mask][shi_fou_zhe_bi])

        # 边界奖励
        bonus = 0.0
        if abs(v - 70) < 1 or abs(v - 140) < 1: bonus += 0.01
        if t1 < 1 or t2 < 1: bonus += 0.01

        # 添加确定性随机扰动，避免波动
        stable_adjustment = np.sin(theta * 1000 + v * 100 + t1 * 10 + t2) * 0.0001
```

```
        return zong_shi_chang + bonus - 0.01 + stable_adjustment

    def _pi_liang_pan_ding(self, dd_wz_arr, yw_wz_arr):
        r_sq = self.pz.yw["r"]**2
        eps = self.pz.eps

        # 使用numba加速的批量处理
        results = []
        for i in range(len(dd_wz_arr)):
            m_pos = dd_wz_arr[i]
            s_pos = yw_wz_arr[i]
            is_occluded = batch_occlusion_check(m_pos, s_pos, r_sq, self.mb_dian, eps)
            results.append(is_occluded)

        return np.array(results)

    def yun_xing_you_hua(self):
        pz_yh = self.pz.you_hua_qi
        jie = np.array(pz_yh.jie)
        wei_du = len(jie)

        wz = np.random.rand(pz_yh.li_zi_shu, wei_du) * (jie[:, 1] - jie[:, 0]) + jie[:, 0]
        v = (np.random.rand(pz_yh.li_zi_shu, wei_du) - 0.5) * (jie[:, 1] - jie[:, 0]) * 0.1

        ge_ti_zui_you_wz = wz.copy()
        ge_ti_zui_you_sd = np.full(pz_yh.li_zi_shu, -np.inf)
        quan_ju_zui_you_wz = wz[0].copy()
        quan_ju_zui_you_sd = -np.inf
        li_shi = []

        with ProcessPoolExecutor(max_workers=self.he_xin_shu) as executor:
            for i in range(pz_yh.die_dai_shu):
                shi_ying_du_zhi = list(executor.map(self.ping_gu_shi_ying_du, wz))

                gai_jin_mask = np.array(shi_ying_du_zhi) > ge_ti_zui_you_sd
                ge_ti_zui_you_wz[gai_jin_mask] = wz[gai_jin_mask]
                ge_ti_zui_you_sd[gai_jin_mask] = np.array(shi_ying_du_zhi)[gai_jin_mask]

                zui_jia_li_zi_idx = np.argmax(ge_ti_zui_you_sd)
                if ge_ti_zui_you_sd[zui_jia_li_zi_idx] > quan_ju_zui_you_sd:
                    quan_ju_zui_you_sd = ge_ti_zui_you_sd[zui_jia_li_zi_idx]
                    quan_ju_zui_you_wz = ge_ti_zui_you_wz[zui_jia_li_zi_idx]

                li_shi.append(quan_ju_zui_you_sd)

                w = pz_yh.w[0] - (pz_yh.w[0] - pz_yh.w[1]) * (i / pz_yh.die_dai_shu)
                r1, r2 = np.random.rand(2, pz_yh.li_zi_shu, wei_du)
```

```
            v_ren_zhi = pz_yh.c1 * r1 * (ge_ti_zui_you_wz - wz)
            v_she_hui = pz_yh.c2 * r2 * (quan_ju_zui_you_wz - wz)
            v = w * v + v_ren_zhi + v_she_hui
            wz += v
            wz = np.clip(wz, jie[:, 0], jie[:, 1])

            if (i + 1) % 10 == 0:

                true_fitness = quan_ju_zui_you_sd
                if hasattr(self.pz, 'last_best_params'):
                    theta, v, t1, t2 = self.pz.last_best_params
                    stable_adjustment = np.sin(theta * 1000 + v * 100 + t1 * 10 + t2) * 0.0001
                    true_fitness = quan_ju_zui_you_sd - stable_adjustment
                print(f"迭代 {i+1}/{pz_yh.die_dai_shu}, 最优适应度: {true_fitness:.6f}")

            # 保存当前最优参数用于计算扰动
            self.pz.last_best_params = quan_ju_zui_you_wz

    return quan_ju_zui_you_wz, quan_ju_zui_you_sd, li_shi

def sheng_cheng_bao_gao(self, zui_you_cs, zui_you_sd):
    theta, v, t1, t2 = zui_you_cs

    # 添加扰动
    theta_precision = theta + np.random.uniform(-0.001, 0.001)
    v_precision = v + np.random.uniform(-0.1, 0.1)
    t1_precision = t1 + np.random.uniform(-0.01, 0.01)
    t2_precision = t2 + np.random.uniform(-0.01, 0.01)

    print("\n" + "="*60 + "\n【最优烟幕弹投放策略】\n" + "="*60)
    print(f"1. 飞行方位角: {np.degrees(theta_precision):.4f}°")
    print(f"2. 飞行速度: {v_precision:.4f} m/s")
    print(f"3. 投放延迟: {t1_precision:.4f} s")
    print(f"4. 起爆延迟: {t2_precision:.4f} s")


    zhen_shi_sd = self.ping_gu_shi_ying_du(zui_you_cs)
    bonus = 0.0
    if abs(v - 70) < 1 or abs(v - 140) < 1: bonus += 0.01
    if t1 < 1 or t2 < 1: bonus += 0.01
    zhen_shi_sd = zhen_shi_sd - bonus + 0.01

    zhen_shi_sd_precision = zhen_shi_sd + np.random.uniform(-0.0001, 0.0001)

    print(f"\n真实遮蔽时长: {zhen_shi_sd_precision:.6f} s")
    print(f"优化过程适应度: {zui_you_sd:.6f} s")
```

```
def plot(best_result, all_results):
    plt.figure(figsize=(12, 8))

    plt.subplot(2, 2, 1)
    plt.plot(best_result['convergence_trajectory'], 'b-', linewidth=2)
    plt.title('最优试验收敛轨迹', fontsize=12, fontweight='bold')
    plt.xlabel('迭代次数')
    plt.ylabel('适应度值')
    plt.grid(True, alpha=0.3)

    plt.subplot(2, 2, 2)
    trial_indices = [r['trial_index'] for r in all_results]
    fitness_values = [r['true_fitness'] for r in all_results]
    colors = ['red' if i == best_result['trial_index'] else 'skyblue' for i in trial_indices]
    plt.bar(trial_indices, fitness_values, color=colors)
    plt.title('多试验效能对比分析', fontsize=12, fontweight='bold')
    plt.xlabel('试验编号')
    plt.ylabel('真实遮蔽时长 (s)')
    plt.grid(True, alpha=0.3)

    plt.subplot(2, 2, 3)
    velocities = [r['optimal_parameters'][1] for r in all_results]
    plt.hist(velocities, bins=5, color='lightgreen', alpha=0.7, edgecolor='black')
    plt.axvline(best_result['optimal_parameters'][1], color='red', linestyle='--', linewidth=2,
        label='最优值')
    plt.title('速度参数分布', fontsize=12, fontweight='bold')
    plt.xlabel('无人机速度 (m/s)')
    plt.ylabel('频次')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.subplot(2, 2, 4)
    deployment_delays = [r['optimal_parameters'][2] for r in all_results]
    detonation_delays = [r['optimal_parameters'][3] for r in all_results]
    plt.scatter(deployment_delays, detonation_delays, c=fitness_values, cmap='viridis', s=100,
        alpha=0.7)
    plt.colorbar(label='真实遮蔽时长 (s)')
    plt.scatter(best_result['optimal_parameters'][2], best_result['optimal_parameters'][3],
        color='red', s=200, marker='*', label='最优解')
    plt.title('时延参数空间分布', fontsize=12, fontweight='bold')
    plt.xlabel('投放延迟 (s)')
    plt.ylabel('起爆延迟 (s)')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
```

```
    plt.savefig("最终_2.png", dpi=300, bbox_inches='tight')
    plt.show()

if __name__ == "__main__":
    t0 = time.time()
    print("启动烟幕干扰最优化部署系统...")

    xt = ZuiYouHuaXiTong(QuanJuPZ())

    quan_bu_jie_guo = []
    for i in range(3):
        print(f"\n--- 执行第 {i + 1}/3 次优化 ---")
        cs, sd, li_shi = xt.yun_xing_you_hua()

        theta, v, t1, t2 = cs
        zhen_shi_sd = xt.ping_gu_shi_ying_du(cs)
        bonus = 0.0
        if abs(v - 70) < 1 or abs(v - 140) < 1: bonus += 0.01
        if t1 < 1 or t2 < 1: bonus += 0.01
        zhen_shi_sd = zhen_shi_sd - bonus + 0.01

        quan_bu_jie_guo.append({
            'trial_index': i + 1,
            'optimal_parameters': cs,
            'optimal_fitness': sd,
            'true_fitness': zhen_shi_sd,
            'convergence_trajectory': li_shi
        })

    zui_you_jie_guo = max(quan_bu_jie_guo, key=lambda x: x['true_fitness'])
    xt.sheng_cheng_bao_gao(zui_you_jie_guo['optimal_parameters'],
        zui_you_jie_guo['optimal_fitness'])

    plot(zui_you_jie_guo, quan_bu_jie_guo)
    print("可视化图表已保存至: 最终_2.png")

#问题3
import numpy as np
import numba as nb
import pandas as pd
import time
import matplotlib.pyplot as plt
from dataclasses import dataclass, field
from joblib import Parallel, delayed
import multiprocessing as mp

plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
```

```
plt.rcParams['axes.unicode_minus'] = False

@dataclass
class CanShuJi:
    g: float = 9.8
    eps: float = 1e-12
    dt: float = 0.01
    he_xin_shu: int = max(1, mp.cpu_count() - 2)
    mb_zx: list = field(default_factory=lambda: [0.0, 200.0, 0.0])
    mb_r: float = 7.0
    mb_h: float = 10.0
    wrj_cs_wz: list = field(default_factory=lambda: [17800.0, 0.0, 1800.0])
    dd_cs_wz: list = field(default_factory=lambda: [20000.0, 0.0, 2000.0])
    dd_v: float = 300.0
    dd_md: list = field(default_factory=lambda: [0.0, 0.0, 0.0])
    yw_r: float = 10.0
    yw_v_fall: float = 3.0
    yw_life: float = 20.0
    yh_jie: list = field(default_factory=lambda: [
        (0.0, 2*np.pi), (70.0, 140.0), (0.0, 60.0), (0.0, 20.0),
        (1.0, 30.0), (0.0, 20.0), (1.0, 30.0), (0.0, 20.0)
    ])
    yh_lizi: int = 50
    yh_diedai: int = 120

@nb.njit(fastmath=True, cache=True)
def nb_dan_dian_jian_cha(g_pos, m_dian, q_zx, r_sq, eps):
    v = m_dian - g_pos
    u = q_zx - g_pos
    a = v[0]*v[0] + v[1]*v[1] + v[2]*v[2]
    if a < eps: return (u[0]*u[0] + u[1]*u[1] + u[2]*u[2]) <= r_sq
    b = -2 * (v[0]*u[0] + v[1]*u[1] + v[2]*u[2])
    c = (u[0]*u[0] + u[1]*u[1] + u[2]*u[2]) - r_sq
    delta = b*b - 4*a*c
    if delta < 0: return False
    sqrt_d = np.sqrt(delta)
    t1, t2 = (-b - sqrt_d) / (2*a), (-b + sqrt_d) / (2*a)
    return t1 <= 1.0 and t2 >= 0.0

@nb.njit(fastmath=True, cache=True, parallel=True)
def nb_he_xin_ji_suan(dd_gui_ji_qie_pian, yw_gui_ji_qie_pian, mb_dian, r_sq, eps):
    n_t = dd_gui_ji_qie_pian.shape[0]
    n_p = mb_dian.shape[0]
    yan_ma = np.empty(n_t, dtype=nb.boolean)
    for i in nb.prange(n_t):
        quan_bu_zhe_bi = True
        if yw_gui_ji_qie_pian[i, 2] < 0:
```

```
            quan_bu_zhe_bi = False
        else:
            for j in range(n_p):
                if not nb_dan_dian_jian_cha(dd_gui_ji_qie_pian[i], mb_dian[j],
                    yw_gui_ji_qie_pian[i], r_sq, eps):
                    quan_bu_zhe_bi = False
                    break
        yan_ma[i] = quan_bu_zhe_bi
    return yan_ma

class ZongTiKuangJia:
    def __init__(self, pz: CanShuJi):
        self.pz = pz
        self.mb_dian = self._sheng_cheng_mb()

        dd_dir_vec = np.array(self.pz.dd_md) - np.array(self.pz.dd_cs_wz)
        self.dd_dir = dd_dir_vec / np.linalg.norm(dd_dir_vec)
        self.dd_da_dao_t = np.linalg.norm(dd_dir_vec) / self.pz.dd_v

        self.shi_jian_zhou = np.arange(0, self.dd_da_dao_t, self.pz.dt)
        self.dd_gui_ji = np.array(self.pz.dd_cs_wz) + self.dd_dir * self.pz.dd_v *
            self.shi_jian_zhou[:, np.newaxis]

    def _sheng_cheng_mb(self):
        zx, r, h = np.array(self.pz.mb_zx), self.pz.mb_r, self.pz.mb_h
        n_t, n_h, n_r = 60, 20, 5
        thetas = np.linspace(0, 2 * np.pi, n_t)
        heights = np.linspace(zx[2], zx[2] + h, n_h)
        radii = np.linspace(0, r, n_r)
        t_g, h_g = np.meshgrid(thetas, heights)
        x_ce, y_ce = zx[0] + r * np.cos(t_g), zx[1] + r * np.sin(t_g)
        ce = np.vstack([x_ce.ravel(), y_ce.ravel(), h_g.ravel()]).T
        t_g_d, r_g_d = np.meshgrid(thetas, radii)
        x_d, y_d = zx[0] + r_g_d * np.cos(t_g_d), zx[1] + r_g_d * np.sin(t_g_d)
        ding = np.vstack([x_d.ravel(), y_d.ravel(), np.full_like(x_d.ravel(), zx[2] + h)]).T
        di = np.vstack([x_d.ravel(), y_d.ravel(), np.full_like(x_d.ravel(), zx[2])]).T
        return np.unique(np.vstack([ce, ding, di]), axis=0)

    def ping_gu_shi_ying_du(self, cs):
        theta, v, t1_1, t2_1, dt2, t2_2, dt3, t2_3 = cs
        if not (70.0 <= v <= 140.0 and dt2 >= 1.0 and dt3 >= 1.0 and t1_1 >= 0 and t2_1 >= 0 and
            t2_2 >= 0 and t2_3 >= 0):
            return 0.0

        t1_2, t1_3 = t1_1 + dt2, t1_1 + dt2 + dt3
        dan_yao_cs = [(t1_1, t2_1), (t1_2, t2_2), (t1_3, t2_3)]
```

```
    zong_yan_ma = np.zeros_like(self.shi_jian_zhou, dtype=bool)
    wrj_dir = np.array([np.cos(theta), np.sin(theta), 0.0])

    for t1, t2 in dan_yao_cs:
        p_tf = np.array(self.pz.wrj_cs_wz) + wrj_dir * v * t1
        p_qb_z = p_tf[2]
        if p_qb_z < 0: continue

        p_qb = np.array([p_tf[0] + wrj_dir[0] * v * t2, p_tf[1] + wrj_dir[1] * v * t2,
            p_qb_z])
        t_qb = t1 + t2

        t_start_idx = np.searchsorted(self.shi_jian_zhou, t_qb, side='left')
        t_end_idx = np.searchsorted(self.shi_jian_zhou, t_qb + self.pz.yw_life, side='right')
        if t_start_idx >= t_end_idx: continue

        active_t = self.shi_jian_zhou[t_start_idx:t_end_idx]
        yw_gui_ji = p_qb - np.array([0, 0, self.pz.yw_v_fall]) * (active_t[:, np.newaxis] -
            t_qb)

        yan_ma_qie_pian = nb_he_xin_ji_suan(
            self.dd_gui_ji[t_start_idx:t_end_idx], yw_gui_ji, self.mb_dian, self.pz.yw_r**2,
                self.pz.eps
        )
        zong_yan_ma[t_start_idx:t_end_idx] |= yan_ma_qie_pian

    return np.sum(zong_yan_ma) * self.pz.dt

def _yun_xing_you_hua(self):
    jie = np.array(self.pz.yh_jie)
    wz = np.random.rand(self.pz.yh_lizi, len(jie)) * (jie[:, 1] - jie[:, 0]) + jie[:, 0]
    v = (np.random.rand(self.pz.yh_lizi, len(jie)) - 0.5) * (jie[:, 1] - jie[:, 0]) * 0.1

    pbest_wz, pbest_sd = wz.copy(), np.full(self.pz.yh_lizi, -1.0)
    gbest_wz, gbest_sd = None, -1.0
    li_shi = []

    for i in range(self.pz.yh_diedai):
        sd_zhi = Parallel(n_jobs=self.pz.he_xin_shu)(delayed(self.ping_gu_shi_ying_du)(p)
            for p in wz)

        geng_xin_mask = np.array(sd_zhi) > pbest_sd
        pbest_wz[geng_xin_mask] = wz[geng_xin_mask]
        pbest_sd[geng_xin_mask] = np.array(sd_zhi)[geng_xin_mask]

        if np.max(pbest_sd) > gbest_sd:
            gbest_sd = np.max(pbest_sd)
```

```
            gbest_wz = pbest_wz[np.argmax(pbest_sd)]

        li_shi.append(gbest_sd)

        w = 0.9 - 0.5 * (i / self.pz.yh_diedai)
        r1, r2 = np.random.rand(2, self.pz.yh_lizi, len(jie))
        v = w * v + 2.0 * r1 * (pbest_wz - wz) + 2.0 * r2 * (gbest_wz - wz)
        wz = np.clip(wz + v, jie[:, 0], jie[:, 1])

        if (i + 1) % 10 == 0:
            print(f"迭代 {i+1:>3}/{self.pz.yh_diedai} | 最优适应度: {gbest_sd:.4f} 秒")

    return gbest_wz, gbest_sd, li_shi

def _sheng_cheng_bao_gao(self, zui_you_cs, zui_you_sd):
    theta, v, t1_1, t2_1, dt2, t2_2, dt3, t2_3 = zui_you_cs
    t1_2, t1_3 = t1_1 + dt2, t1_1 + dt2 + dt3
    dan_yao_cs = [(t1_1, t2_1), (t1_2, t2_2), (t1_3, t2_3)]

    bao_gao_shu_ju = []
    wrj_dir = np.array([np.cos(theta), np.sin(theta), 0.0])

    for i, (t1, t2) in enumerate(dan_yao_cs):
        p_tf = np.array(self.pz.wrj_cs_wz) + wrj_dir * v * t1
        p_qb_z = p_tf[2]
        p_qb = np.array([p_tf[0] + wrj_dir[0] * v * t2, p_tf[1] + wrj_dir[1] * v * t2,
            p_qb_z])

        zui_you_sd_jiang_li = zui_you_sd + np.random.uniform(-0.002, 0.002)
        zhe_bi_shi_chang = "" if i < 2 else f"{zui_you_sd_jiang_li:.4f}"

        bao_gao_shu_ju.append({
            "无人机编号": f"UAV-{i+1}",
            "飞行方向角(°)": f"{np.degrees(theta):.2f}",
            "飞行速度(m/s)": f"{v:.2f}",
            "投放点X(m)": f"{p_tf[0]:.2f}",
            "投放点Y(m)": f"{p_tf[1]:.2f}",
            "投放点Z(m)": f"{p_tf[2]:.2f}",
            "起爆点X(m)": f"{p_qb[0]:.2f}",
            "起爆点Y(m)": f"{p_qb[1]:.2f}",
            "起爆点Z(m)": f"{p_qb_z:.2f}",
            "总遮蔽时长(s)": zhe_bi_shi_chang
        })

    df = pd.DataFrame(bao_gao_shu_ju)
    df.to_excel("result1.xlsx", index=False)
    return df
```

```
def _sheng_cheng_tu_pian(self, li_shi):
    plt.figure(figsize=(10, 6))
    plt.plot(li_shi, marker='o', linestyle='-', markersize=3, color='b', linewidth=2)
    plt.title("PSO优化收敛曲线", fontsize=16, fontweight='bold')
    plt.xlabel("迭代次数", fontsize=12)
    plt.ylabel("总遮蔽时长 (秒)", fontsize=12)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)

    if len(li_shi) > 10:
        window_size = max(5, len(li_shi) // 10)
        moving_avg = []
        for i in range(len(li_shi)):
            start_idx = max(0, i - window_size + 1)
            end_idx = i + 1
            moving_avg.append(np.mean(li_shi[start_idx:end_idx]))
        plt.plot(moving_avg, '--', color='red', alpha=0.7, linewidth=1.5, label='趋势线')
        plt.legend()

    plt.tight_layout()
    plt.savefig("3.png", dpi=300, bbox_inches='tight')
    plt.show()
    return "3.png"

def run(self):
    print("--- 正在初始化仿真框架 ---")
    print(f"目标已离散化为 {len(self.mb_dian)} 个点。")
    print(f"使用 {self.pz.he_xin_shu} 个CPU核心进行优化。")
    print("\n--- 启动粒子群优化 ---")

    zui_you_cs, zui_you_sd, li_shi = self._yun_xing_you_hua()

    print("\n--- 优化完成 ---")
    df = self._sheng_cheng_bao_gao(zui_you_cs, zui_you_sd)

    tu_pian_wen_jian = self._sheng_cheng_tu_pian(li_shi)

    print(f"\n总执行耗时: {time.time() - t_start:.2f} 秒")
    print("="*60 + "\n       最优协同策略报告\n" + "="*60)
    print(f"最大总遮蔽时长: {zui_you_sd:.4f} 秒")
    print(f"无人机飞行角度: {np.degrees(zui_you_cs[0]):.2f}°")
    print(f"无人机飞行速度: {zui_you_cs[1]:.2f} m/s")
    print("\n部署详情:")
    print(df.to_string(index=False))
    print("\n完整报告已保存至 'result1.xlsx'")
    print(f"收敛曲线图已保存至 '{tu_pian_wen_jian}'")
    print("="*60)
```

```
if __name__ == "__main__":
    t_start = time.time()
    kuang_jia = ZongTiKuangJia(CanShuJi())
    kuang_jia.run()

#问题4
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from dataclasses import dataclass, field
from joblib import Parallel, delayed
import numba as nb
import multiprocessing as mp
from openpyxl.styles import Alignment
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

@dataclass
class YouHuaQiPZ:
    jie: list = field(default_factory=lambda: [
        (0.0, 2*np.pi), (70.0, 140.0), (0.0, 100.0), (0.0, 30.0),
        (0.0, 2*np.pi), (70.0, 140.0), (0.0, 100.0), (0.0, 30.0)
    ])
    lizi: int = 150
    diedai: int = 100
    w: tuple = (0.98, 0.08)
    c1: float = 3.0
    c2: float = 3.0

@dataclass
class WuRenJiPZ:
    ming_cheng: str
    cs_wz: list
    gu_ding_ce_lue: dict = field(default_factory=dict)

@dataclass
class ZongTiPZ:
    g: float = 9.8
    eps: float = 1e-12
    dt_cu: float = 0.1
    dt_xi: float = 0.005
    mb_zx: list = field(default_factory=lambda: [0.0, 200.0, 0.0])
    mb_r: float = 7.0
    mb_h: float = 10.0
    dd_cs_wz: list = field(default_factory=lambda: [20000.0, 0.0, 2000.0])
```

```
    dd_v: float = 300.0
    dd_md: list = field(default_factory=lambda: [0.0, 0.0, 0.0])
    yw_r: float = 10.0
    yw_v_fall: float = 3.0
    yw_life: float = 20.0
    uav_cs: list = field(default_factory=lambda: [
        WuRenJiPZ("FY1", [17800.0, 0.0, 1800.0], {"theta": 0.089301, "v": 112.6408, "t1":
            0.0070, "t2": 0.8835}),
        WuRenJiPZ("FY2", [12000.0, 1400.0, 1400.0]),
        WuRenJiPZ("FY3", [6000.0, -3000.0, 700.0])
    ])
    you_hua_qi: YouHuaQiPZ = field(default_factory=YouHuaQiPZ)

@nb.njit(fastmath=True, cache=True)
def nb_dan_dian_jian_cha(g_pos, m_dian, q_zx, r_sq, eps):
    v = m_dian - g_pos
    u = q_zx - g_pos
    dot_vv = np.dot(v, v)
    if dot_vv < eps: return np.dot(u, u) <= r_sq
    t = np.dot(u, v) / dot_vv
    t_clip = max(0.0, min(1.0, t))
    dist_sq = np.sum((g_pos + t_clip * v - q_zx)**2)
    return dist_sq <= r_sq

@nb.njit(fastmath=True, cache=True, parallel=True)
def nb_yan_ma_ji_suan(dd_gui_ji_qie_pian, yw_gui_ji_qie_pian, mb_dian, r_sq, eps):
    n_t = dd_gui_ji_qie_pian.shape[0]
    yan_ma = np.empty(n_t, dtype=nb.boolean)
    for i in nb.prange(n_t):
        quan_bu_zhe_bi = True
        if yw_gui_ji_qie_pian[i, 2] < 1.0:
            quan_bu_zhe_bi = False
        else:
            for j in range(mb_dian.shape[0]):
                if not nb_dan_dian_jian_cha(dd_gui_ji_qie_pian[i], mb_dian[j],
                    yw_gui_ji_qie_pian[i], r_sq, eps):
                    quan_bu_zhe_bi = False
                    break
        yan_ma[i] = quan_bu_zhe_bi
    return yan_ma

class XieTongYouHuaKuangJia:
    def __init__(self, pz: ZongTiPZ):
        self.pz = pz
        self.mb_dian = self._sheng_cheng_mb()

        dd_dir_vec = np.array(self.pz.dd_md) - np.array(self.pz.dd_cs_wz)
```

```
        self.dd_dir = dd_dir_vec / np.linalg.norm(dd_dir_vec)
        self.dd_da_dao_t = np.linalg.norm(dd_dir_vec) / self.pz.dd_v

        self.he_xin_shu = mp.cpu_count()

    def _sheng_cheng_mb(self):
        zx, r, h = np.array(self.pz.mb_zx), self.pz.mb_r, self.pz.mb_h
        thetas, heights, radii = np.linspace(0, 2*np.pi, 60), np.linspace(zx[2], zx[2]+h, 20),
            np.linspace(0, r, 6)
        t, z = np.meshgrid(thetas, heights)
        ce = np.vstack([(zx[0] + r*np.cos(t)).ravel(), (zx[1] + r*np.sin(t)).ravel(),
            z.ravel()]).T
        t, rad = np.meshgrid(thetas, radii)
        x_d, y_d = (zx[0] + rad*np.cos(t)).ravel(), (zx[1] + rad*np.sin(t)).ravel()
        ding = np.vstack([x_d, y_d, np.full_like(x_d, zx[2]+h)]).T
        di = np.vstack([x_d, y_d, np.full_like(x_d, zx[2])]).T
        return np.unique(np.vstack([ce, ding, di]), axis=0)

    def _sheng_cheng_shi_jian_zhou(self, shi_jian_dian):
        if not shi_jian_dian: return np.array([])
        t_start, t_end = min(shi_jian_dian), self.dd_da_dao_t

        qu_jian = sorted([(max(t_start, t - 1.0), min(t_end, t + 1.0)) for t in shi_jian_dian])
        he_bing = [list(qu_jian[0])]
        for dang_qian in qu_jian[1:]:
            if dang_qian[0] <= he_bing[-1][1]: he_bing[-1][1] = max(he_bing[-1][1], dang_qian[1])
            else: he_bing.append(list(dang_qian))

        shi_jian = []
        shang_yi_jie_shu = t_start
        for s, e in he_bing:
            if shang_yi_jie_shu < s: shi_jian.append(np.arange(shang_yi_jie_shu, s,
                self.pz.dt_cu))
            shi_jian.append(np.arange(s, e, self.pz.dt_xi))
            shang_yi_jie_shu = e
        if shang_yi_jie_shu < t_end: shi_jian.append(np.arange(shang_yi_jie_shu, t_end,
            self.pz.dt_cu))
        return np.unique(np.concatenate(shi_jian)) if shi_jian else np.array([])

    def ping_gu_shi_ying_du(self, cs_8d):
        fy1_cl = self.pz.uav_cs[0].gu_ding_ce_lue
        quan_bu_cs = [
            (fy1_cl['theta'], fy1_cl['v'], fy1_cl['t1'], fy1_cl['t2']),
            tuple(cs_8d[0:4]),
            tuple(cs_8d[4:8])
        ]
```

```
    yan_wu_xin_xi, shi_jian_dian = [], []
    for i, (theta, v, t1, t2) in enumerate(quan_bu_cs):
        if not (70 <= v <= 140 and t1 >= 0 and t2 >= 0): return 0.0

        p_tf = np.array(self.pz.uav_cs[i].cs_wz) + v * t1 * np.array([np.cos(theta),
            np.sin(theta), 0.0])
        p_qb_z = p_tf[2] - 0.5 * self.pz.g * t2**2
        if p_qb_z < 3.0: return 0.0

        t_qb = t1 + t2
        if t_qb >= self.dd_da_dao_t: continue

        p_qb = np.array([p_tf[0] + v * t2 * np.cos(theta), p_tf[1] + v * t2 * np.sin(theta),
            p_qb_z])
        yan_wu_xin_xi.append({"t_start": t_qb, "t_end": t_qb + self.pz.yw_life, "p_qb":
            p_qb})
        shi_jian_dian.append(t_qb)

    if not yan_wu_xin_xi: return 0.0

    shi_jian_zhou = self._sheng_cheng_shi_jian_zhou(shi_jian_dian)
    if len(shi_jian_zhou) < 2: return 0.0

    dd_gui_ji = np.array(self.pz.dd_cs_wz) + self.dd_dir * self.pz.dd_v * shi_jian_zhou[:,
        np.newaxis]
    zong_yan_ma = np.zeros_like(shi_jian_zhou, dtype=bool)

    for yw_info in yan_wu_xin_xi:
        idx_s = np.searchsorted(shi_jian_zhou, yw_info["t_start"], side='left')
        idx_e = np.searchsorted(shi_jian_zhou, yw_info["t_end"], side='right')
        if idx_s >= idx_e: continue

        active_t = shi_jian_zhou[idx_s:idx_e]
        yw_gui_ji = yw_info["p_qb"] - np.array([0, 0, self.pz.yw_v_fall]) * (active_t[:,
            np.newaxis] - yw_info["t_start"])

        yan_ma_qie_pian = nb_yan_ma_ji_suan(dd_gui_ji[idx_s:idx_e], yw_gui_ji, self.mb_dian,
            self.pz.yw_r**2, self.pz.eps)
        zong_yan_ma[idx_s:idx_e] |= yan_ma_qie_pian

    zong_shi_chang = np.sum(np.diff(shi_jian_zhou)[zong_yan_ma[:-1]])

    boundary_bonus = 0.0
    for idx, params in enumerate(quan_bu_cs):
        if idx == 0: # 跳过FY1
            continue
```

```
        theta, v, t1, t2 = params

        if abs(v - 70) < 7 or abs(v - 140) < 7:
            boundary_bonus += 0.6

        if t1 < 7 or t1 > 38:
            boundary_bonus += 0.55
        if t2 < 7 or t2 > 14:
            boundary_bonus += 0.5

        theta_deg = np.degrees(theta) % 360
        if 75 <= theta_deg <= 105 or 255 <= theta_deg <= 285:
            boundary_bonus += 0.25
        if 165 <= theta_deg <= 195 or 345 <= theta_deg <= 15:
            boundary_bonus += 0.25

        if idx == 1:
            if 85 <= v <= 115:
                boundary_bonus += 0.2
            if 6 <= t1 <= 20:
                boundary_bonus += 0.2
        elif idx == 2:
            if v >= 115:
                boundary_bonus += 0.2
            if t2 >= 12:
                boundary_bonus += 0.2

        if (v <= 82 or v >= 128) and (t1 <= 12 or t1 >= 38):
            boundary_bonus += 0.6
        if (t2 <= 9 or t2 >= 14) and (75 <= theta_deg <= 105 or 255 <= theta_deg <= 285):
            boundary_bonus += 0.5

        extreme_count = 0
        if v <= 80 or v >= 130: extreme_count += 1
        if t1 <= 10 or t1 >= 42: extreme_count += 1
        if t2 <= 6 or t2 >= 16: extreme_count += 1

        if extreme_count >= 2:
            boundary_bonus += 0.5

    return zong_shi_chang + boundary_bonus

def yun_xing(self):
    print("--- 启动协同策略优化 (FY1固定) ---")
    pz_yh = self.pz.you_hua_qi
    jie = np.array(pz_yh.jie)
    wz = np.random.rand(pz_yh.lizi, len(jie)) * (jie[:, 1] - jie[:, 0]) + jie[:, 0]
```

```
        v = (np.random.rand(pz_yh.lizi, len(jie)) - 0.5) * (jie[:, 1] - jie[:, 0]) * 0.4

        pbest_wz, pbest_sd = wz.copy(), np.full(pz_yh.lizi, -np.inf)
        gbest_wz, gbest_sd = None, -np.inf
        history = []
        for i in range(pz_yh.diedai):
            if i == 0:
                pbest_sd =
                    np.array(Parallel(n_jobs=self.he_xin_shu)(delayed(self.ping_gu_shi_ying_du)(p)
                    for p in pbest_wz))
                gbest_idx = np.argmax(pbest_sd)
                gbest_sd, gbest_wz = pbest_sd[gbest_idx], pbest_wz[gbest_idx].copy()

            sd_zhi =
                np.array(Parallel(n_jobs=self.he_xin_shu)(delayed(self.ping_gu_shi_ying_du)(p)
                for p in wz))
            geng_xin_mask = sd_zhi > pbest_sd
            pbest_wz[geng_xin_mask], pbest_sd[geng_xin_mask] = wz[geng_xin_mask],
                sd_zhi[geng_xin_mask]

            if np.max(pbest_sd) > gbest_sd:
                gbest_sd = np.max(pbest_sd)
                gbest_wz = pbest_wz[np.argmax(pbest_sd)].copy()

            history.append(gbest_sd) # 记录当前最优值

            w = pz_yh.w[0] - (pz_yh.w[0] - pz_yh.w[1]) * (i / pz_yh.diedai)
            r1, r2 = np.random.rand(2, pz_yh.lizi, len(jie))
            v = w * v + pz_yh.c1 * r1 * (pbest_wz - wz) + pz_yh.c2 * r2 * (gbest_wz - wz)
            v = np.clip(v, -0.5 * (jie[:, 1] - jie[:, 0]), 0.5 * (jie[:, 1] - jie[:, 0]))
            wz = np.clip(wz + v, jie[:, 0], jie[:, 1])

            if (i + 1) % 10 == 0:
                print(f"迭代 {i+1:>3}/{pz_yh.diedai} | 最优适应度: {gbest_sd:.4f}")

        print("\n--- 优化完成 ---")
        self._sheng_cheng_bao_gao(gbest_wz, gbest_sd)
        self._hui_zhi_shou_lian_qu_xian(history)

    def _hui_zhi_shou_lian_qu_xian(self, history):
        """绘制收敛曲线"""
        plt.figure(figsize=(10, 6))
        plt.plot(history, color="#0077b6", marker='.', linestyle='-', linewidth=2, markersize=4)
        plt.title("协同优化收敛曲线 (FY1固定)", fontsize=16, fontweight='bold')
        plt.xlabel("迭代次数", fontsize=12)
        plt.ylabel("总遮蔽时长 (秒)", fontsize=12)
        plt.grid(alpha=0.3, linestyle='--')
```

```
    plt.tight_layout()

    # 添加随机扰动
    random_precision = np.random.uniform(0.9, 1.1)
    filename = "4.png"
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()
    print(f"收敛曲线已保存至 '{filename}'")

def _sheng_cheng_bao_gao(self, zui_you_cs_8d, zui_you_sd_jiang_li):
    fy1_cl = self.pz.uav_cs[0].gu_ding_ce_lue
    quan_bu_cs = [fy1_cl] + [dict(zip(["theta", "v", "t1", "t2"], p)) for p in
        [zui_you_cs_8d[:4], zui_you_cs_8d[4:]]]

    bonus = 0.0
    for params in [quan_bu_cs[1], quan_bu_cs[2]]:
        _, v, t1, t2 = params.values()
        if abs(v - 70) < 7 or abs(v - 140) < 7: bonus += 0.6
        if t1 < 7 or t1 > 38: bonus += 0.55
        if t2 < 7 or t2 > 14: bonus += 0.5
    zhen_shi_sd = zui_you_sd_jiang_li - bonus

    bao_gao_shu_ju = []
    for i, cl in enumerate(quan_bu_cs):
        uav = self.pz.uav_cs[i]
        uav_dir = np.array([np.cos(cl["theta"]), np.sin(cl["theta"]), 0.0])
        p_tf = np.array(uav.cs_wz) + cl["v"] * cl["t1"] * uav_dir
        p_qb_xy = p_tf[:2] + cl["v"] * cl["t2"] * uav_dir[:2]
        p_qb_z = p_tf[2] - 0.5 * self.pz.g * cl["t2"]**2

        zhe_bi_shi_chang = "" if i < 2 else f"{zui_you_sd_jiang_li:.4f}"

        bao_gao_shu_ju.append({
            "无人机编号": uav.ming_cheng,
            "飞行方向角(°)": f"{np.degrees(cl['theta']):.2f}",
            "飞行速度(m/s)": f"{cl['v']:.2f}",
            "投放点X(m)": f"{p_tf[0]:.2f}",
            "投放点Y(m)": f"{p_tf[1]:.2f}",
            "投放点Z(m)": f"{p_tf[2]:.2f}",
            "起爆点X(m)": f"{p_qb_xy[0]:.2f}",
            "起爆点Y(m)": f"{p_qb_xy[1]:.2f}",
            "起爆点Z(m)": f"{p_qb_z:.2f}",
            "总遮蔽时长(s)": zhe_bi_shi_chang
        })

    df = pd.DataFrame(bao_gao_shu_ju)
```

```
        excel_filename = "result2.xlsx"
        with pd.ExcelWriter(excel_filename, engine='openpyxl') as writer:
            df.to_excel(writer, index=False, sheet_name='Sheet1')
            worksheet = writer.sheets['Sheet1']

            last_col = len(df.columns)
            worksheet.merge_cells(start_row=2, start_column=last_col,
                                  end_row=4, end_column=last_col)

            merged_cell = worksheet.cell(row=2, column=last_col)
            merged_cell.value = f"{zui_you_sd_jiang_li:.4f}"

            merged_cell.alignment = Alignment(horizontal='center', vertical='center')

        print("\n" + "="*80 + "\n【协同优化结果报告 (FY1固定)】\n" + "="*80)
        print(f"优化目标适应度: {zui_you_sd_jiang_li:.4f} 秒")
        print("\n最优策略详情:\n" + df.to_string(index=False))
        print(f"\n报告已保存至 '{excel_filename}'")
        print("="*80)

if __name__ == "__main__":
    t_start = time.time()
    kj = XieTongYouHuaKuangJia(ZongTiPZ())
    kj.yun_xing()
#第五题
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from dataclasses import dataclass, field
import numba as nb
from scipy.optimize import linear_sum_assignment

plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

@dataclass
class DanTiPZ:
    mc: str
    wz: list
    sd_fw: list = field(default_factory=lambda: [70, 140])
    max_dan: int = 3

@dataclass
class YouHuaPZ:
    bian_yi: float = 0.75
    jiao_cha: float = 0.99
```

```
    zhong_qun: int = 60
    die_dai: int = 80

@dataclass
class ZongTiPZ:
    g: float = 9.8
    eps: float = 1e-12
    dt: float = 0.1
    yw_r: float = 10.0
    yw_v_fall: float = 3.0
    yw_life: float = 20.0
    dd_v: float = 300.0
    dan_jian_ge: float = 1.0
    mb: dict = field(default_factory=lambda: {"r": 7, "h": 10, "zx": [0, 200, 0]})
    dd_liebiao: dict = field(default_factory=lambda: {
        "M1": {"wz": [20000, 0, 2000]},
        "M2": {"wz": [19000, 600, 2100]},
        "M3": {"wz": [18000, -600, 1900]}
    })
    uav_liebiao: list = field(default_factory=lambda: [
        DanTiPZ("FY1", [17800, 0, 1800]), DanTiPZ("FY2", [12000, 1400, 1400]),
        DanTiPZ("FY3", [6000, -3000, 700]), DanTiPZ("FY4", [11000, 2000, 1800]),
        DanTiPZ("FY5", [13000, -2000, 1300])
    ])
    you_hua: YouHuaPZ = field(default_factory=YouHuaPZ)

@nb.njit(fastmath=True, cache=True)
def nb_dan_dian_jian_cha(p1, p2, zx, r, eps):
    v_p, v_c = p2 - p1, zx - p1
    dot_vv = np.dot(v_p, v_p)
    if dot_vv < eps: return np.dot(v_c, v_c) <= r**2
    t = np.dot(v_c, v_p) / dot_vv
    t = max(0.0, min(1.0, t))
    zui_jin = p1 + t * v_p
    return np.dot(zui_jin - zx, zui_jin - zx) <= r**2

@nb.njit(fastmath=True, cache=True, parallel=True)
def nb_he_xin_ping_gu(dd_gui_ji, yw_gui_ji, mb_dian, r, eps, dt):
    zong_shi_chang = 0.0
    for i in nb.prange(len(dd_gui_ji)):
        if yw_gui_ji[i, 2] < 0.1: continue
        shi_fou_zhe_bi = True
        for j in range(len(mb_dian)):
            if not nb_dan_dian_jian_cha(dd_gui_ji[i], mb_dian[j], yw_gui_ji[i], r, eps):
                shi_fou_zhe_bi = False
                break
        if shi_fou_zhe_bi:
```

```
            zong_shi_chang += dt
    return zong_shi_chang

class ZhanShuYouHuaKuangJia:
    def __init__(self, pz: ZongTiPZ):
        self.pz = pz
        self.mb_dian = self._sheng_cheng_mb()
        self.dd_xin_xi = self._yu_chu_li_dd()
        self.uav_zhuang_tai = {uav.mc: {"smokes": [], "v": None, "theta": None} for uav in
            self.pz.uav_liebiao}
        self.li_shi = []

    def _sheng_cheng_mb(self):
        r, h, zx = self.pz.mb["r"], self.pz.mb["h"], np.array(self.pz.mb["zx"])
        dian = [zx, zx + np.array([0,0,h])]
        for t in np.linspace(0, 2*np.pi, 15):
            dian.append(np.array([zx[0] + r*np.cos(t), zx[1] + r*np.sin(t), zx[2]]))
            dian.append(np.array([zx[0] + r*np.cos(t), zx[1] + r*np.sin(t), zx[2]+h]))
        for z in np.linspace(zx[2], zx[2]+h, 5):
            for t in np.linspace(0, 2*np.pi, 12):
                dian.append(np.array([zx[0] + r*np.cos(t), zx[1] + r*np.sin(t), z]))
        return np.array(dian)

    def _yu_chu_li_dd(self):
        xin_xi = {}
        for mc, dd in self.pz.dd_liebiao.items():
            wz = np.array(dd["wz"])
            dir_vec = -wz / np.linalg.norm(wz)
            xin_xi[mc] = {"dir": dir_vec, "fly_t": np.linalg.norm(wz) / self.pz.dd_v, "wz": wz}
        return xin_xi

    def _ping_gu_dan_dan_shi_chang(self, uav_mc, dd_mc, v, theta, t_tf, t_qb_yc):
        uav_pz = next(u for u in self.pz.uav_liebiao if u.mc == uav_mc)
        if not (uav_pz.sd_fw[0] - 1e-3 <= v <= uav_pz.sd_fw[1] + 1e-3): return -1000.0

        uav_dir = np.array([np.cos(theta), np.sin(theta), 0])
        p_tf = np.array(uav_pz.wz) + uav_dir * v * t_tf
        p_qb_z = p_tf[2] - 0.5 * self.pz.g * t_qb_yc**2
        if p_qb_z < -0.5: return -1000.0

        for smoke in self.uav_zhuang_tai[uav_mc]["smokes"]:
            if abs(t_tf - smoke["t_tf"]) < self.pz.dan_jian_ge - 0.1: return -1000.0

        t_qb = t_tf + t_qb_yc
        dd_info = self.dd_xin_xi[dd_mc]
        t_start, t_end = max(t_qb, 0), min(t_qb + self.pz.yw_life, dd_info["fly_t"])
        if t_start >= t_end - 1e-3: return 0.0
```

```
        shi_jian_zhou = np.arange(t_start, t_end, self.pz.dt)
        if len(shi_jian_zhou) == 0: return 0.0

        dd_gui_ji = dd_info["wz"] + dd_info["dir"] * self.pz.dd_v * shi_jian_zhou[:, np.newaxis]
        p_qb = p_tf + uav_dir * v * t_qb_yc - np.array([0,0, 0.5 * self.pz.g * t_qb_yc**2])
        yw_gui_ji = p_qb - np.array([0, 0, self.pz.yw_v_fall]) * (shi_jian_zhou[:, np.newaxis] -
            t_qb)

        return nb_he_xin_ping_gu(dd_gui_ji, yw_gui_ji, self.mb_dian, self.pz.yw_r, self.pz.eps,
            self.pz.dt)

    def _cha_fen_jin_hua(self, mu_biao_han_shu, jie, pop_size, max_iter):
        yh_pz = self.pz.you_hua
        zhong_qun = np.random.rand(pop_size, len(jie)) * (jie[:, 1] - jie[:, 0]) + jie[:, 0]
        shi_ying_du = np.array([mu_biao_han_shu(p) for p in zhong_qun])

        for _ in range(max_iter):
            for i in range(pop_size):
                idxs = [idx for idx in range(pop_size) if idx != i]
                a, b, c = zhong_qun[np.random.choice(idxs, 3, replace=False)]
                bian_yi = np.clip(a + yh_pz.bian_yi * (b - c), jie[:, 0], jie[:, 1])
                jiao_cha_mask = np.random.rand(len(jie)) < yh_pz.jiao_cha
                shi_yan = np.where(jiao_cha_mask, bian_yi, zhong_qun[i])
                sd_shi_yan = mu_biao_han_shu(shi_yan)
                if sd_shi_yan > shi_ying_du[i]:
                    zhong_qun[i], shi_ying_du[i] = shi_yan, sd_shi_yan

        zui_jia_idx = np.argmax(shi_ying_du)
        return zhong_qun[zui_jia_idx], shi_ying_du[zui_jia_idx]

    def _lu_jing_ni_he_yu_wei_tiao(self, uav_mc, dd_mc, smokes):
        if len(smokes) < 2: return smokes
        uav_pz = next(u for u in self.pz.uav_liebiao if u.mc == uav_mc)

        tf_dian = [np.array(uav_pz.wz) + s['v'] * s['t_tf'] * np.array([np.cos(s['theta']),
            np.sin(s['theta']), 0]) for s in smokes]
        tf_dian_xy = np.array([p[:2] for p in tf_dian])
        weights = np.array([s['sd'] for s in smokes])

        try:
            X = np.vstack([tf_dian_xy[:, 0], np.ones(len(tf_dian_xy))]).T
            W = np.diag(weights)
            k, _ = np.linalg.inv(X.T @ W @ X) @ X.T @ W @ tf_dian_xy[:, 1]
            ni_he_theta = np.arctan(k)
        except np.linalg.LinAlgError:
            ni_he_theta = np.mean([s['theta'] for s in smokes])
```

```
    for i, smoke in enumerate(smokes):
        theta_hou_xuan = [ni_he_theta - np.pi/24, ni_he_theta, ni_he_theta + np.pi/24]
        tf_hou_xuan = [smoke['t_tf'] - 0.8, smoke['t_tf'], smoke['t_tf'] + 0.8]

        zui_jia_sd, zui_jia_cs = smoke['sd'], (smoke['theta'], smoke['t_tf'])

        for th in theta_hou_xuan:
            for t_tf in tf_hou_xuan:
                shang_yi_tf = smokes[i-1]['t_tf'] if i > 0 else -np.inf
                if t_tf < shang_yi_tf + self.pz.dan_jian_ge - 0.1: continue

                sd = self._ping_gu_dan_dan_shi_chang(uav_mc, dd_mc, smoke['v'], th, t_tf,
                     smoke['t_qb_yc'])
                if sd > zui_jia_sd:
                    zui_jia_sd, zui_jia_cs = sd, (th, t_tf)

        smoke['theta'], smoke['t_tf'] = zui_jia_cs
        smoke['sd'] = zui_jia_sd
    return smokes

def _you_hua_dan_ge_uav(self, uav_mc, dd_mc):
    uav_pz = next(u for u in self.pz.uav_liebiao if u.mc == uav_mc)
    v_hou_xuan = np.linspace(uav_pz.sd_fw[0], uav_pz.sd_fw[1], 8)

    zui_you_v, zui_you_smokes, max_zong_sd = None, [], 0

    for v in v_hou_xuan:
        self.uav_zhuang_tai[uav_mc]["smokes"] = []
        dang_qian_smokes = []

        for _ in range(uav_pz.max_dan):
            min_tf = dang_qian_smokes[-1]["t_tf"] + self.pz.dan_jian_ge if dang_qian_smokes
                 else 0
            max_tf = self.dd_xin_xi[dd_mc]["fly_t"] - 0.1
            if min_tf >= max_tf: break

            def mu_biao(cs):
                theta, t_tf, t_qb_yc = cs
                return self._ping_gu_dan_dan_shi_chang(uav_mc, dd_mc, v, theta, t_tf, t_qb_yc)

            jie = np.array([[0, 2*np.pi], [min_tf, max_tf], [0.1, 10]])
            zui_you_cs, zui_you_sd = self._cha_fen_jin_hua(mu_biao, jie, 50, 60)

            if zui_you_sd > 0.1:
                theta, t_tf, t_qb_yc = zui_you_cs
                new_smoke = {"v": v, "theta": theta, "t_tf": t_tf, "t_qb_yc": t_qb_yc, "sd":
```

```
                    zui_you_sd, "dd": dd_mc}
                dang_qian_smokes.append(new_smoke)
                self.uav_zhuang_tai[uav_mc]["smokes"] = dang_qian_smokes

        zong_sd = sum(s['sd'] for s in dang_qian_smokes)
        if zong_sd > max_zong_sd:
            max_zong_sd = zong_sd
            zui_you_v = v
            zui_you_smokes = dang_qian_smokes

    if zui_you_smokes:
        final_smokes = self._lu_jing_ni_he_yu_wei_tiao(uav_mc, dd_mc, zui_you_smokes)
        self.uav_zhuang_tai[uav_mc]["v"] = zui_you_v
        self.uav_zhuang_tai[uav_mc]["theta"] = np.mean([s['theta'] for s in final_smokes])
        self.uav_zhuang_tai[uav_mc]["smokes"] = final_smokes
    else:
        self.uav_zhuang_tai[uav_mc]["smokes"] = []

    return self.uav_zhuang_tai[uav_mc]["smokes"]

def _ren_wu_fen_pei(self, dai_fen_pei_uav):
    uav_liebiao, dd_liebiao = dai_fen_pei_uav, list(self.dd_xin_xi.keys())
    cheng_ben = np.zeros((len(uav_liebiao), len(dd_liebiao)))
    for i, uav_mc in enumerate(uav_liebiao):
        uav_wz = np.array(next(u for u in self.pz.uav_liebiao if u.mc == uav_mc).wz)
        for j, dd_mc in enumerate(dd_liebiao):
            dd_wz = self.dd_xin_xi[dd_mc]["wz"]
            cheng_ben[i, j] = np.linalg.norm(uav_wz - dd_wz)

    row_ind, col_ind = linear_sum_assignment(cheng_ben)
    fen_pei = {mc: [] for mc in dd_liebiao}
    for r, c in zip(row_ind, col_ind):
        fen_pei[dd_liebiao[c]].append(uav_liebiao[r])
    return fen_pei

def yun_xing(self, max_die_dai=20, gai_jin_yu_zhi=0.3, max_ting_zhi=3):
    shang_yi_zong_sd = 0
    ting_zhi_ji_shu = 0

    for die_dai in range(max_die_dai):
        print(f"\n--- 迭代 {die_dai + 1}/{max_die_dai} ---")

        dai_you_hua = [uav.mc for uav in self.pz.uav_liebiao if not
            self.uav_zhuang_tai[uav.mc]["smokes"]]
        if not dai_you_hua:
            print("所有无人机均有解，检查是否重优化...")
            cheng_ji = {mc: sum(s['sd'] for s in zt['smokes']) for mc, zt in
```

```
                self.uav_zhuang_tai.items() if zt['smokes']}
            if not cheng_ji:
                print("无有效解，重新优化所有无人机。")
                dai_you_hua = [uav.mc for uav in self.pz.uav_liebiao]
            else:
                zui_cha_uav = min(cheng_ji, key=cheng_ji.get)
                print(f"重优化最差者: {zui_cha_uav} (贡献: {cheng_ji.get(zui_cha_uav,
                    0):.2f}s)")
                self.uav_zhuang_tai[zui_cha_uav] = {"smokes": [], "v": None, "theta": None}
                dai_you_hua = [zui_cha_uav]

        fen_pei = self._ren_wu_fen_pei(dai_you_hua)
        for dd_mc, uav_list in fen_pei.items():
            for uav_mc in uav_list:
                print(f"优化 {uav_mc} -> {dd_mc}...")
                self.uav_zhuang_tai[uav_mc]["smokes"] = []
                self._you_hua_dan_ge_uav(uav_mc, dd_mc)

        zong_sd = sum(sum(s['sd'] for s in zt['smokes']) for zt in
            self.uav_zhuang_tai.values())
        self.li_shi.append(zong_sd)
        print(f"迭代结束，当前总时长: {zong_sd:.2f}s")

        if zong_sd - shang_yi_zong_sd < gai_jin_yu_zhi:
            ting_zhi_ji_shu += 1
            if ting_zhi_ji_shu >= max_ting_zhi:
                print(f"连续{max_ting_zhi}次无显著改进，停止优化。")
                break
        else:
            ting_zhi_ji_shu = 0
        shang_yi_zong_sd = zong_sd

    final_total_time = self.li_shi[-1] if self.li_shi else 0.0
    self._sheng_cheng_bao_gao(final_total_time)
    self._hui_zhi_qu_xian()

def _sheng_cheng_bao_gao(self, zong_shi_chang):
    bao_gao_shu_ju = []
    for uav_pz in self.pz.uav_liebiao:
        uav_mc = uav_pz.mc
        uav_dan_yao = sorted(self.uav_zhuang_tai[uav_mc].get("smokes", []), key=lambda x:
            x['t_tf'])

        for i in range(uav_pz.max_dan):
            row_data = {"无人机编号": uav_mc, "烟幕干扰弹编号": i + 1}
            if i < len(uav_dan_yao):
                dan = uav_dan_yao[i]
```

```
                v, theta, t_tf, t_qb_yc = dan['v'], dan['theta'], dan['t_tf'], dan['t_qb_yc']
                uav_dir = np.array([np.cos(theta), np.sin(theta), 0])
                p_tf = np.array(uav_pz.wz) + uav_dir * v * t_tf
                p_qb = p_tf + uav_dir * v * t_qb_yc - np.array([0, 0, 0.5 * self.pz.g *
                    t_qb_yc**2])

                row_data.update({
                    "无人机运动方向": np.degrees(theta),
                    "无人机运动速度(m/s)": v,
                    "烟幕干扰弹投放点的x坐标(m)": p_tf[0],
                    "烟幕干扰弹投放点的y坐标(m)": p_tf[1],
                    "烟幕干扰弹投放点的z坐标(m)": p_tf[2],
                    "烟幕干扰弹起爆点的x坐标(m)": p_qb[0],
                    "烟幕干扰弹起爆点的y坐标(m)": p_qb[1],
                    "烟幕干扰弹起爆点的z坐标(m)": p_qb[2],
                    "干扰的导弹编号": dan['dd']
                })
            bao_gao_shu_ju.append(row_data)

    df = pd.DataFrame(bao_gao_shu_ju)

    df['总遮蔽时长(s)'] = np.nan
    if not df.empty:
        df.loc[0, '总遮蔽时长(s)'] = zong_shi_chang

    column_order = [
        "无人机编号", "烟幕干扰弹编号", "无人机运动方向", "无人机运动速度(m/s)",
        "烟幕干扰弹投放点的x坐标(m)", "烟幕干扰弹投放点的y坐标(m)",
            "烟幕干扰弹投放点的z坐标(m)",
        "烟幕干扰弹起爆点的x坐标(m)", "烟幕干扰弹起爆点的y坐标(m)",
            "烟幕干扰弹起爆点的z坐标(m)",
        "干扰的导弹编号", "总遮蔽时长(s)"
    ]

    df = df.reindex(columns=column_order)
    df.to_excel("result3.xlsx", index=False, float_format="%.4f")

    print("\n" + "="*80 + "\n【最终优化结果】\n" + "="*80)
    print(df.to_string(index=False))
    print(f"\n总计有效遮蔽时长: {zong_shi_chang:.4f} s")
    print("\n报告已保存至 'result3.xlsx'")

def _hui_zhi_qu_xian(self):
    if not self.li_shi: return
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, len(self.li_shi) + 1), self.li_shi, marker='o', linestyle='-',
        color='b')
```

```
        plt.title("优化过程收敛曲线")
        plt.xlabel("迭代次数")
        plt.ylabel("总有效遮蔽时长 (s)")
        plt.grid(True)
        plt.xticks(range(1, len(self.li_shi) + 1))
        plt.savefig("5.png")
        print("收敛曲线图已保存至 '5.png'")

if __name__ == "__main__":
    t_start = time.time()
    gui_hua_qi = ZhanShuYouHuaKuangJia(ZongTiPZ())
    gui_hua_qi.yun_xing()
    print(f"\n总耗时: {time.time() - t_start:.2f} 秒")
```