

基于集合估计与覆盖优化的 无线电干扰源自动定位清除

摘 要

针对源数量、位置和有效接收半径未知条件下的无线电干扰源自动排查问题,本文建立由几何定位、主动测量、覆盖搜索与动作调度组成的闭环模型。模型以有界测向误差维护真实位置的候选集合,以清除半径约束控制操作时机,并将移动、检测、换频及清除统一换算为虚拟耗时。

针对问题一,将每次示向度转化为前向角域的半平面约束,通过边界交点枚举、可行性筛选和凸包构造得到定位区域,区分空集、无界及退化情形。证明区域直径在顶点对中取得,给出直径圆覆盖的充要判据;等边三角形反例说明直径不超过 40 m 仍不足以保证一次清除,建立最小包围圆半径不超过 20 m 的清除充分条件。

针对问题二,由首次测向与接收半径上界构造候选集,定义保证接收区域为所有候选位置的 1000 m 圆盘之交。证明截断角域的最小包围半径为 $1500/(2 \cos \delta)$,当有效误差半角为 1.005° 时,得到半径约 249.885 m 的充分安全候选盘。在线采用更紧的凸外包圆,在盘内比较 37 个候选点的采样最坏后验半径与移动时间。

针对问题三,构造原点与六个环站组成的七点覆盖,在环半径 1159.7 m 时证明最坏覆盖距离不超过 984.539 m。结合保守网格的正负观测更新、信息增益选点与开放路径滚动调度,按各频道的候选集与清除结果判定完成状态。三次正式测试分别清除 13、15、10 个源,平均定位清除时间为 238.075、232.889、290.686 s/个。

针对问题四,以位置与朝向的联合约束处理定向遮挡,采用 31 点离散集合覆盖、可见性筛选与网格遍历清除,并利用凸组合性质证明 49 点方格方案的连续覆盖性。三次正式测试分别清除 13、10、14 个源,平均定位清除时间为 720.171、927.130、614.884 s/个。

六次正式测试均主动退出且清除失败计数为零。正式成绩仅取自官方模拟器 test 目录下的六份正式导出包及其匹配统计;正式数据未公开源总数,故本文只用清除数量、平均定位清除时间与程序运行时间评价,不据此宣称全清率。

对比实验以仓库内的配对基准集复核上述方法:问题一朴素基线在随机算例上直径平均绝对误差 4.583 m、覆盖判定准确率 52.5%,最大误差 14.599 m;问题二忽略接收保证的基线保证接收率为 0.0%,只保证接收而缺少横向基线的基线最坏后验半径为 340.747 m;问题三默认策略相对 49 点网格的平均虚拟时间下降 78.4%;问题四直接搬用问题三策略时出现 24 次缺席误判。该组数据来自自建冻结基准集与演练局留档,只用于机制说明和相对行为比较,与正式成绩分节陈述,不合并统计。

四问的求解核心已整理为约 5000 行的单文件程序，并按四问主线在附录中列出核心函数与一句话说明；评测机、命令行外壳、采集与绘图脚本、测试以及未被采用的方案不计入正文。

关键词：有界误差；交会定位；最小包围圆；主动测量；定向覆盖

1 问题重述

1.1 问题背景

无线电干扰源的精准定位是频谱管理的末端任务。在初步监测和区域排查已确定目标分布范围后，仍需携带测向设备逐点检测，以缩小位置不确定性并完成清除。本题希望由一只搭载测向机和光学定位装置的机器狗自动完成这一过程，减少人工进入危险环境的需要，并缩短全部任务的完成时间 [1]。

目标区域为 $\Omega = \overline{B}(0, 1800)$ ，单位为米，坐标轴分别指向东、北。每个源占用 1 至 20 中的一个独立频道，源总数为 10 至 16。有效接收半径未知，介于 1000 与 1500 m 之间。检测返回无信号、近距离或示向度；示向度误差不超过 $\pm 1^\circ$ ，同一地点的重复检测不会改变误差。全向源在整个圆周发射；定向源仅在其未知朝向两侧各 90° 的闭半平面内发射。

机器狗从原点、频道 1 出发，以 5 m/s 沿直线移动，停机后才能检测。一次检测耗时 5 s，切换检测频道耗时 1 s。距源不超过 20 m 时，光学定位与清除共耗时 5 s；清除失败仅支付 3 s 光学检查时间。清除是否成功与源朝向无关。距源不超过 5 m 且能够接收信号时，返回近距离信息，可直接执行清除。圆域约束源的位置，机器狗允许到圆域外检测 [2]。

1.2 研究任务

- (1) 已知多个测点及同一源的示向度，计算交会定位区域的直径，判断以该直径为直径的圆能否覆盖定位区域。
- (2) 已知一个全向源的首次示向度，确定第二检测点的候选区域与选点策略，在保证接收的条件下获得较好的定位效果。
- (3) 对数量未知的全向源设计自动发现、定位和清除算法，在完成任务的前提下尽量减少总耗时，并报告官方测试结果。
- (4) 将策略推广到全向与定向源混合的情形，处理无信号的歧义，给出相应的搜索、定位、清除算法及正式测试结果。

问题三、四各完成三次正式测试。本文依据六份正式日志及其对应统计，报告各次清除数量、平均定位清除时间和程序运行时间。

2 问题分析

四问共用一条从题面抽象到在线执行的链路，如图 1。最上层是题面给定的区域、运动与动作耗时、源与频道范围；中间是在线系统的证据层、决策层与执行层，分别维护候选集与幸存集、生成选点与任务表、串行执行动作；其下是四问各自的主线，最下

层是发现、定位与清除三个闭环。后文的模型与算法都落在这条链路的某一层上：问题一、二给出证据层与决策层的几何判据，问题三、四给出三个闭环在时间约束下的组织方式。

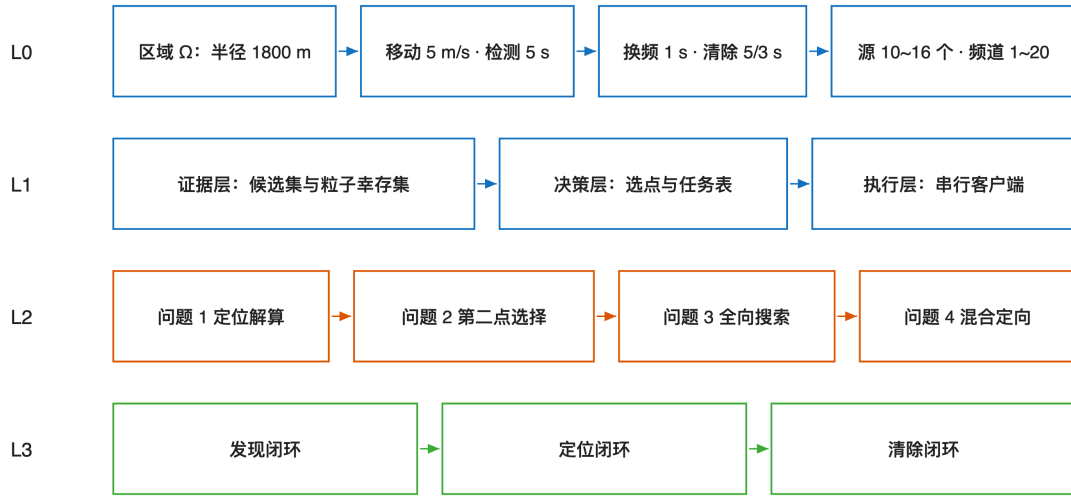


图 1 四问共用的模型链路：题面抽象（L0）、在线三层（L1）、四问主线（L2）与发现、定位、清除三个闭环（L3）。

四问构成由单源几何分析到多源在线决策的递进关系。问题一的核心是将带误差的射线交会改写为凸集合计算。单次示向度只限制方向，不能给出唯一位置；即使定位区域的直径较小，其最远顶点对中点也未必能覆盖全部候选位置，因此应分别讨论直径与包围半径。

问题二同时受几何收缩与接收范围制约。只追求较大的交会角可能使第二点远离源，导致没有信号；直接前往候选集中心则可能缺少横向基线。因此先构造对未知真源均安全的测量区域，再在其中优化后验不确定性。“保证能够接收”具有确定性证明，“有限候选中收缩较好”属于数值决策。

问题三的搜索过程需要同时维护各频道的存在性与定位状态。本文以覆盖点集和逐频道检测记录确定搜索范围，以负观测排除不可能位置，以最小包围圆触发清除，并通过滚动排序减少发现任务与定位任务之间的往返。

问题四中，无信号既可能来自距离，也可能来自朝向。其发现要求从“圆盘覆盖位置”提升为“前向半圆盘覆盖位置与朝向”。定位时应保留可能遮挡的解释；若无线电观测停滞，则利用清除不受朝向限制的性质，覆盖仍可能含源的小区域。整个闭环的关系如图 2。

3 模型假设

假设 1：源的位置、频道、发射朝向和有效接收半径在一次测试中保持不变，不同频道互不干扰。这与静态测试案例及题设的频道独立性一致。

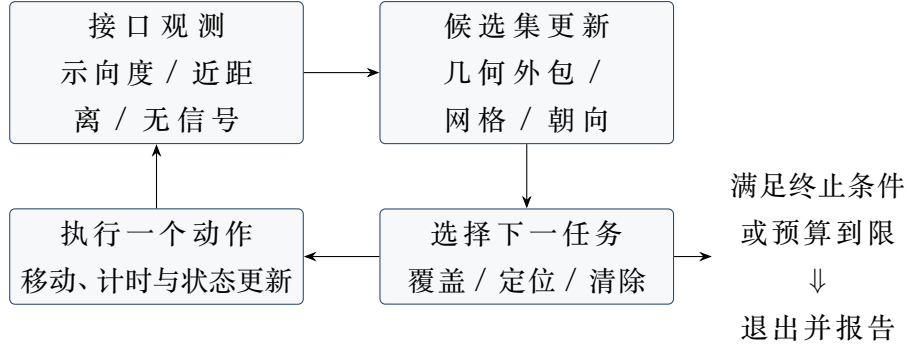


图2 基于观测更新、集合估计与任务排序的闭环决策过程。

假设2：采用二维欧氏距离，忽略高度差；机器人匀速直线移动，检测、换频、光学定位与清除按给定时间串行执行。

假设3：几何保证只使用有界误差和接收半径上下界。为覆盖接口两位小数的表示误差，在线角域取 $\delta_{\text{eff}} = 1^\circ + 0.005^\circ$ 。该附加量用于处理示向度的数值舍入误差。

假设4：定位过程依据检测点坐标、频道及观测结果更新候选集合。

假设5：信息增益与可见性评分采用均匀候选权重及均匀接收半径作为决策先验；几何可行性由测向误差和接收距离约束确定。

前三项中的物理约束取自题目和附件，数值余量与评分先验属于实现选择。

4 符号说明

数学推导中三角函数的自变量采用弧度；带 $^\circ$ 的量先换算为弧度。记 $[a, b] = a_x b_y - a_y b_x$ 为二维叉积， $\overline{B}(c, r)$ 为闭圆盘。

5 计时模型与会话状态机

5.1 会话状态与合法动作

题目接口只有 `/enter`、`/measure`、`/clear` 与 `/exit` 四条指令。会话状态依次为未进入、进入倒计时、接口开放与断连，进入倒计时为5s，如图3。只有 `accepted=true` 的响应推进虚拟时钟；被拒绝的请求返回的虚拟时刻为零，不能当作当前时刻使用，也不改变位置与测向机频道。一次合法 `/measure` 结束后，测向机频道更新为本次检测频道；`/clear` 给出的是目标频道，既不改变测向机频道，也不产生换频耗时。因此清除之后原地再测的耗时取决于上一次 `/measure` 的频道，而不是清除的频道。

5.2 动作耗时与累计虚拟时间

题面将全部动作折算为虚拟耗时，见表2。检测无论返回无信号、近距离还是示向度，均计5s；清除成功计5s，其中光学定位3s、激光清除2s，未命中目标时只支付

表 1 主要符号及单位

符号	含义	单位
Ω, R	源位置圆域及其半径, $R = 1800$	m
g, s_i, q	真源位置、已有检测点、待选检测点	m
θ_i, δ	示向度及误差半角	度或弧度
$u(\alpha)$	单位方向向量 $(\cos \alpha, \sin \alpha)$	—
W_i, P	第 i 个前向角域及纯角域交集	—
K_c, P_c^{out}	频道 c 的可行位置集及保守凸外包	—
$D(P), m, \rho$	区域直径、最小包围圆圆心及半径	m
R_-, R_+	接收半径下、上界, 分别为 1000、1500	m
$\mathcal{Q}(K)$	对集合 K 保证接收的检测点区域	—
ℓ, ϵ_g	网格边长及格心位置裕量 $\ell/\sqrt{2}$	m
ψ, H	源朝向及朝向离散档数	度, 个
L, T	移动总距离及总虚拟耗时	m, s
N_m, N_s, N_c^+, N_c^-	检测、换频、成功清除、失败清除次数	次
$\bar{T}, T_{\text{real}}$	平均定位清除时间及程序运行时间	s/个, s

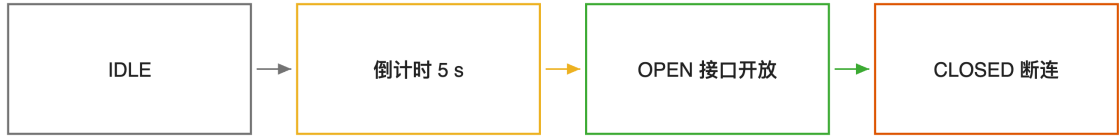


图 3 会话状态机。进入后经 5 s 倒计时接口开放, 主动退出或断连后进入关闭状态。

光学检查的 3 s。移动与换频没有独立指令, 由相邻动作的位置与频道变化推断。

表 2 动作耗时规则。移动耗时等于直线距离除以 5 m/s。

动作	指令	虚拟耗时/s	说明
进入	/enter	0	初始位置 $(0, 0)$ 、初始频道 1
检测	/measure	5+ 移动 + 换频	三种检测结果耗时相同
清除	/clear	5+ 移动 / 3+ 移动	成功 / 未命中目标, 均不切换测向频道
退出	/exit	0	路径终点自由, 无需返回原点
换频	由检测推断	1	相邻 /measure 的频道变化
移动	由动作推断	距离 / 5	相邻动作的位置变化

设整个会话的移动距离为 L , 检测、换频、成功清除与失败清除的次数分别为 N_m, N_s, N_c^+, N_c^- 。忽略每段移动时间的微秒量化时, 总虚拟耗时为

$$T = \frac{L}{5} + 5N_m + N_s + 5N_c^+ + 3N_c^-. \quad (1)$$

式 (1) 是后文任务排序的服务时间来源, 也是由官方累计量反推移动时间 (式 (40)) 的

依据。图 4 用附件的五步示例说明记账方式：首次检测含移动 100 s 与检测 5 s，单步 105 s；原地换到频道 2 检测单步 6 s，累计 111 s；清除目标频道 3 未命中，移动 80 s 加光学检查 3 s，单步 83 s，累计 194 s；清除不改变测向机频道，故随后原地以频道 2 复测只需 5 s，累计 199 s，与附件一致。

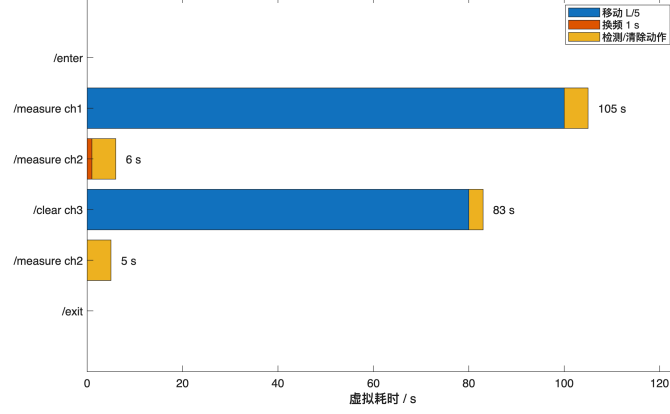


图 4 附件计时示例的逐动作构成。移动项为距离除以 5 m/s，标题数字为该步合计。

5.3 虚拟时间、现实预算与终止

虚拟时间度量任务完成速度，现实预算约束程序能否继续发出请求，两者不能混用。程序以 /enter 返回的 `remaining_real_duration_s` 为准，在每个动作前预留退出时间；虚拟时间以整数微秒累计，在线预测只用于规划与一致性核对。因此后文的“耗时”一律指式 (1) 的虚拟时间，程序运行时间在结果表中单列。策略在预算不足或协议异常时主动退出并报告未解决频道，不以程序正常结束代替任务完成。

6 问题一：有界误差交会定位与覆盖圆

6.1 前向角域的半平面模型

给定检测点 $s_i = (s_{ix}, s_{iy})$ 与读数 θ_i ，误差约束意味着真实方位位于 $[\theta_i - \delta, \theta_i + \delta]$ 内。因此可行位置属于前向角域

$$W_i = \left\{ g : \begin{cases} [u(\theta_i - \delta), g - s_i] \geq 0, \\ u(\theta_i + \delta), g - s_i \leq 0, \\ u(\theta_i) \cdot (g - s_i) \geq 0 \end{cases} \right\}. \quad (2)$$

三条不等式均可写为 $ax + by \leq c$ ，其法向依次为

$$\begin{aligned} &(\sin(\theta_i - \delta), -\cos(\theta_i - \delta)), \\ &(-\sin(\theta_i + \delta), \cos(\theta_i + \delta)), \\ &(-\cos \theta_i, -\sin \theta_i), \end{aligned} \quad (3)$$

右端为相应法向与 s_i 的内积。在 $0 < \delta < 90^\circ$ 时，前两条已限定前向窄角域；第三条显式保留方向性，并使零误差退化情形对应射线而非整条直线。采用正弦、余弦表示可自动处理 $0^\circ/360^\circ$ 跨界。

问题一的多边形定位区域定义为

$$P = \bigcap_{i=1}^n W_i = \{g : Ag \leq b\}. \quad (4)$$

式 (4) 根据各次测向角域确定问题一的定位区域。后续问题将在此基础上加入 Ω 与接收距离约束。半平面交是闭凸集，可能为空、单点、线段、有界多边形或无界集。

6.2 区域构造、空集及无界情形

令约束数 $k = 3n$ 。对任意两条不平行边界 $ax + by = c$ 、 $dx + ey = f$ ，候选交点为

$$x = \frac{ce - bf}{ae - bd}, \quad y = \frac{af - cd}{ae - bd}. \quad (5)$$

仅保留满足全部不等式的交点，再经去重和单调链凸包构造得到顶点集合。实现先平移与缩放坐标，以降低千米尺度坐标造成的数值误差；近平行边界仍参与求交，因为其远端交点可能决定细长区域的直径。

对非空集合定义衰退锥

$$\mathcal{R} = \{d : Ad \leq 0\}. \quad (6)$$

若 $\mathcal{R}$ 含非零向量，则 $g + td \in P$ 对所有 $t \geq 0$ 成立，区域无界，直径为 $+\infty$ 。二维多面锥只需检查约束边界的两个切向 $(b, -a)$ 与 $(-b, a)$ 是否满足全部齐次不等式。本题至少一个前向窄角域会排除直线性，故非空交集存在极点；在此特定输入族中，没有可行边界交点可以判为空集。无观测输入单独返回整个平面，不按空集处理。

有界非空区域中，一个顶点对应单点，两个顶点对应线段，其余为凸多边形。空集没有本文定义的有效直径与覆盖圆，程序以显式状态报告观测不相容，便于检查频道、误差界或输入数据。

6.3 直径与直径圆的判定

命题 1 (顶点直径). 设 $P = \text{conv}\{v_1, \dots, v_m\}$ 非空有界，则

$$D(P) = \max_{i,j} \|v_i - v_j\|. \quad (7)$$

证明. 任意 $x, y \in P$ 可表示为凸组合 $x = \sum_i \lambda_i v_i$ 、 $y = \sum_j \mu_j v_j$ 。由于系数非负且各自和为 1，

$$\|x - y\| = \left\| \sum_{i,j} \lambda_i \mu_j (v_i - v_j) \right\| \leq \sum_{i,j} \lambda_i \mu_j \|v_i - v_j\| \leq \max_{i,j} \|v_i - v_j\|.$$

最远顶点对本身属于 P ，故上界可达。 $\square$

因此无需在整个区域内部搜索，只需枚举顶点对，计算量为 $O(m^2)$ 。

命题 2 (直径圆覆盖的充要条件). 取最远顶点对 a, b ，令 $D = \|a - b\|$ 、 $o = (a + b)/2$ 。存在直径等于 D 的圆盘覆盖 P ，当且仅当

$$\max_{v \in V(P)} \|v - o\| \leq D/2. \quad (8)$$

证明. 若半径 $D/2$ 的圆盘中心为 z 且包含 a, b ，则 $D \leq \|a - z\| + \|z - b\| \leq D$ 。等号迫使 $z = o$ 。反之，圆盘为凸集，包含全部顶点即包含它们的凸包。□

若条件不成立，输出圆外顶点及超出量 $e = \max_v \|v - o\| - D/2$ ，即得到可独立检查的否定证据。不能仅以直径端点在圆周上判断整个区域被覆盖。图 5(a) 给出一组三点交会算例：左图为三条前向角域及其交集，右图为定位区域的放大，虚线为直径圆、实线为最小包围圆。该算例的定位区域由四个顶点确定，直径为 23.997 m，以该直径为直径的圆半径为 11.998 m，而最远顶点到圆心的距离为 12.602 m，故直径圆并不覆盖定位区域；其最小包围圆半径为 12.069 m，满足 20 m 的清除判据。这说明直径不超过 40 m 与能否清除是两个不同的判据，覆盖判定必须逐顶点检查。

6.4 反例与可清除半径

构造等边三角形

$$A = (0, 0), \quad B = (40, 0), \quad C = (20, 20\sqrt{3}). \quad (9)$$

其直径为 40 m。以 AB 为直径的圆心为 $(20, 0)$ ，半径为 20 m，但 C 到圆心的距离为 $20\sqrt{3}$ m，位于圆外。其最小包围圆圆心为 $(20, 20/\sqrt{3})$ ，半径 $40/\sqrt{3} \approx 23.094$ m，同样大于清除半径。图 5(b) 展示两类圆的差别。

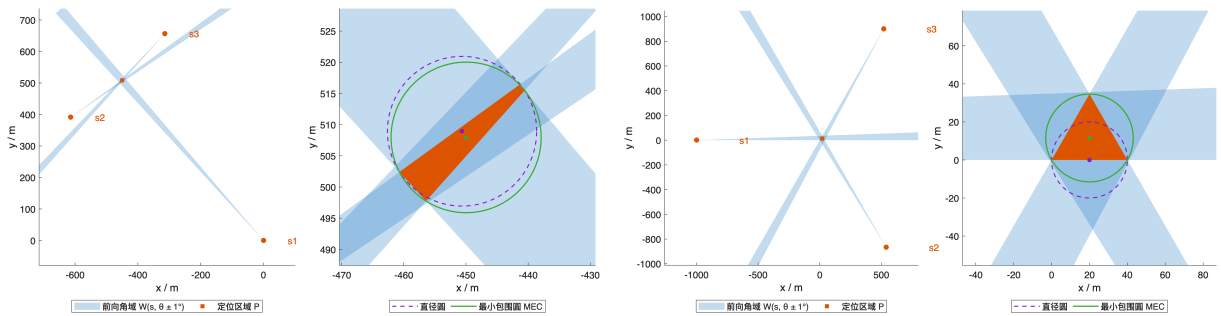


图 5 问题一的两组几何。(a) 三点交会算例：定位区域及其直径圆与最小包围圆，观测取自官方演练案例 U5ZW-P9KN-TCAG-R7ST 的频道 9，几何量由生产求解器 `mcm26b.geometry.solve` 现算；(b) 等边三角形解析反例，虚线直径圆 $r = 20$ m 不覆盖顶点 C ，实线最小包围圆 $\rho = 40/\sqrt{3} \approx 23.094$ m 覆盖全部顶点，不是模拟器观测。

该形状可由题设的 $\pm 1^\circ$ 角域构造出来。例如检测点 $(-1000, 0)$ 、 $(540, -500\sqrt{3})$ 、 $(520, 520\sqrt{3})$ ，对应示向度为 1° 、 121° 、 241° 。各角域的一侧边界给出三角形的三条支撑线，另一侧边界包含整个三角形，故交集恰为上述三角形。

对一般候选区域，定义最小包围圆

$$\rho(P) = \min_{z \in \mathbb{R}^2} \max_{g \in P} \|g - z\|, \quad m \in \arg \min_z \max_{g \in P} \|g - z\|. \quad (10)$$

其边界由至多三个支撑点确定，可用 Welzl 随机增量算法计算 [4]。实现采用固定随机种子及显式栈，并在输出前检查全部顶点的包含关系。在精确模型中，若 $\rho(P) \leq 20$ ，于 m 清除即可覆盖所有可能真源；考虑浮点计算后采用 $\rho(P) + \eta_{\text{num}} \leq 20$ 的充分判据。 $D(P) \leq 40$ 是必要条件，但并非充分条件。

6.5 算法一及复杂度

算法 1: 有界测向角域交会与覆盖圆判定 (对应 algo-1)

输入：同一频道的检测点、示向度和误差半角。

输出：区域状态、顶点、直径、直径圆判定与最小包围圆。

1. 若观测为空，返回无界；否则归一化坐标，生成 $3n$ 条半平面。
 2. 枚举非平行边界交点，仅保留满足全部约束的点。
 3. 无可行点时返回空集；存在非零衰退切向时返回无界及无穷直径。
 4. 对可行交点去重、求凸包，并区分单点、线段和二维多边形。
 5. 枚举最远顶点对，计算 D 、中点 o ，检查全部顶点距 o 是否不超过 $D/2$ 。
 6. 求最小包围圆，复核包含关系与数值余量，判断是否满足清除条件。
-

边界交点枚举需 $O(k^2)$ 次，每点校验 k 条约束，主计算量为 $O(k^3)$ ；凸包最坏需 $O(k^2 \log k)$ ，直径枚举需 $O(m^2)$ 。Welzl 算法在随机排列下具有期望线性时间。该求解流程适合少量观测，当前入口限制最多 100 个检测点；更大规模可替换半平面交算法。

7 问题二：保证接收的第二检测点选择

7.1 候选位置集合与安全区域

设首次有效示向度为 (s, θ) 。有效读数除限定角域外，还说明源距检测点不超过接收半径上界，故

$$K_1 = \Omega \cap W(s, \theta, \delta_{\text{eff}}) \cap \overline{B}(s, R_+). \quad (11)$$

其中 $R_+ = 1500$ 为有效接收半径上界。direction 对应距离大于 5 m；式 (11) 采用其凸松弛，使真实可行位置始终包含在所维护的候选集合中。

对非空紧集 K 定义

$$F_K(q) = \max_{g \in K} \|q - g\|, \quad \mathcal{Q}(K) = \{q : F_K(q) \leq R_-\} = \bigcap_{g \in K} \overline{B}(g, R_-). \quad (12)$$

该区域内的任何点均在所有可能真源的最小接收范围内。 F_K 是凸函数且为 1-Lipschitz, 因而 $\mathcal{Q}(K)$ 是闭凸集。由式 (10) 可知

$$\mathcal{Q}(K) \neq \emptyset \iff \rho(K) \leq R_- . \quad (13)$$

这一表述将第二点的鲁棒可行性转化为最小包围圆判据 [5]。

在线用外切正多边形表示圆盘, 再逐半平面裁剪。若边数为 M 、圆半径为 r , 外切多边形顶点的半径为 $r \sec(\pi/M)$, 其最大径向超出量准确写为

$$e_M(r) = r(\sec(\pi/M) - 1). \quad (14)$$

默认 $M = 192$ 。由于整个圆盘已包含在多边形中, 裁剪后得到 $K_1 \subseteq P^{\text{out}}$ 。算法求外包多边形 P^{out} 的最小包围圆 $\bar{B}(m, \rho_{\text{out}})$ 。

由三角不等式, 充分安全盘为

$$\mathcal{Q}_{\text{in}} = \bar{B}(m, R_- - \rho_{\text{out}} - \eta), \quad \eta = 10^{-3} \text{ m}. \quad (15)$$

任意 $q \in \mathcal{Q}_{\text{in}}$ 与真源 g 满足 $\|q - g\| \leq \|q - m\| + \|m - g\| \leq R_- - \eta$ 。因此第二次检测一定为 `direction` 或 `near`。该构造适用于 $\rho_{\text{out}} + \eta \leq R_-$; 当候选区域较大时, 程序先在其圆心追加观测, 再更新可行区域。

7.2 首次截断角域的解析候选盘

为给出不依赖离散多边形的显式答案, 暂去掉 Ω , 考虑 $S = W(s, \theta, \delta) \cap \bar{B}(s, R_+)$ 。当 $0 \leq \delta \leq 45^\circ$, 令

$$a = \frac{R_+}{2 \cos \delta}, \quad c = s + a u(\theta). \quad (16)$$

命题 3 (截断角域最小包围圆). 上述 S 的最小包围圆为 $\bar{B}(c, a)$ 。

证明. 任取 $g = s + tu(\theta + \varphi)$, 其中 $0 \leq t \leq R_+$ 、 $|\varphi| \leq \delta$ 。有 $\|g - c\|^2 = t^2 + a^2 - 2at \cos \varphi$ 。该式关于 t 为凸二次式, 最大值在区间端点取得。 $t = 0$ 时为 a^2 ; $t = R_+$ 时不超过 $R_+^2 + a^2 - 2aR_+ \cos \delta = a^2$, 故全域被包含。

再取边界三点 $s, p_\pm = s + R_+ u(\theta \pm \delta)$ 。权重 $\lambda_\pm = 1/(4 \cos^2 \delta)$ 、 $\lambda_0 = 1 - 1/(2 \cos^2 \delta)$ 非负且和为 1, 其加权中心为 c 。三点均距 c 为 a , 对任意圆心 z , 加权平方距离等于 $a^2 + \|z - c\|^2$, 故至少一点距 z 不小于 a 。任何包含三点的圆半径均不小于 a 。□

由于 $K_1 \subseteq S$, 于是

$$q \in \bar{B}\left(s + \frac{1500}{2 \cos \delta} u(\theta), 1000 - \frac{1500}{2 \cos \delta}\right) \quad (17)$$

表 3 解析安全盘参数，由式 (17) 直接计算

误差半角	中心沿首次方向偏移 a/m	候选盘半径/ m
1°	750.114246	249.885754
1.005°	750.115391	249.884609

即为题目所求的一种明确候选区域，实际使用时再内缩 1 mm。该圆盘构成 $\mathcal{Q}(K_1)$ 的一个显式凸子集。

在线算法继续使用 Ω 裁剪后的完整候选形状，通常能够获得更紧的源包围圆与更大的充分安全盘。解析公式负责解释保证为何存在，多边形流程负责利用已有的全部几何约束。

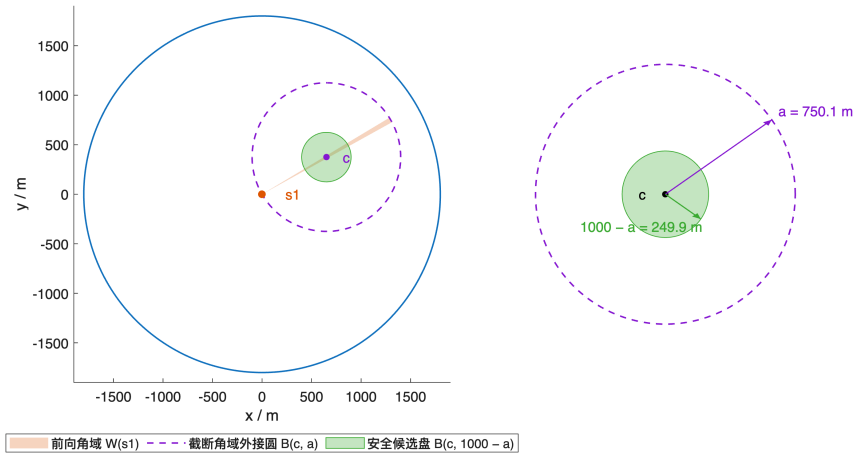


图 6 第二点解析候选区域。左图为首次截断角域 S 、其最小包围圆 $\bar{B}(c, a)$ 与保证接收的候选盘 $\bar{B}(c, 1000 - a)$ ；右图为半径分解， $a = 1500/(2 \cos \delta)$ 。取 $\delta = 1^\circ$ 时 $a \approx 750.114 \text{ m}$ 、 $1000 - a \approx 249.886 \text{ m}$ ，与表 3 一致。

7.3 定位效果指标与有限候选求解

理想的鲁棒选点模型为

$$\min_{q \in \mathcal{Q}_{\text{in}}} \left(\sup_{y \in \mathcal{Y}(q, K_1)} \rho(K_1 \cap O(q, y)), \frac{\|q - p\|}{5} \right)_{\text{lex}}, \quad (18)$$

其中 p 是当前机器人位置， y 表示可能观测， $O(q, y)$ 为相应观测可行集，下标表示按字典序比较：先最小化最坏后验半径，再减少移动耗时。方向分支取相应角域，近距离分支取 $\bar{B}(q, 5)$ 。这比仅优化交会角更直接地对应最终清除需求。

连续极小极大问题求解成本较高，当前算法采用有限候选近似。在式 (15) 的盘心及半径比例 0.4, 0.75, 1.0 的三个环上各取 12 个方向，角度为 $(k + \frac{1}{2})2\pi/12$ ，共 37 点。对每个候选 q ，在 P^{out} 边界以不大于 60 m 的间距采样，包含全部顶点。对边界采样点 g 枚举读数 $\text{bearing}(q, g) \pm \delta_{\text{eff}}$ ，计算新角域交后的最小包围半径，并另外计算 near 分支；取所得最大值作为 $\hat{J}(q)$ 。

选取使 $(\hat{J}(q), \|q - p\|/5)$ 最小的点，然后将边界采样间距加密至 15 m 复核其评分。 $\hat{J}$ 是最坏后验半径的离散近似，用于比较有限候选点的定位效果；接收条件由式 (15) 确定。

算法 2：保证接收的主动选点（对应 algo-2）

输入：首次测点与示向度，或任意已更新的候选凸外包；当前位置。

输出：下一检测点、采样后验评分及接收保证状态。

1. 初次调用时构造式 (11) 的外包多边形；后续调用使用已有外包。
 2. 计算最小包围圆和安全盘半径 $b = 1000 - \rho_{\text{out}} - 10^{-3}$ 。
 3. 若 $b < 0$ ，在候选区域圆心追加观测；否则生成 37 个安全候选点。
 4. 对每枚枚举边界样本与方向端点读数，补入近距离分支，得到 $\hat{J}$ 。
 5. 按后验评分优先、移动时间次优选点，并以 15 m 采样间距复核。
 6. 获得新观测后更新候选集；near 可就地清除，direction 则根据最小包围半径决定清除或追加测量。
-

若候选数为 N_q 、边界样本数为 N_g 、多边形顶点数为 m ，一次裁剪为 $O(m)$ ，总计算量主要为 $O(N_q N_g (m + C_{\text{MEC}}(m)))$ 。该算法采用问题一的角域模型与最小包围圆方法，通过多边形裁剪逐次更新候选集。

8 问题三：全向多源的覆盖搜索与滚动调度

8.1 统一时间目标与任务状态

按式 (1) 的计时记账，优化目标是在全部源清除的约束下最小化虚拟耗时 T 。题目统计的平均定位清除时间是 $\bar{T} = T/N_c^+$ ；不能用每次清除动作的固定 5 s 代替。清除请求中的频道是目标频道，不改变测向机频道；进入与退出均不增加虚拟时间，因此路径终点自由，无需返回原点。

各频道的状态按“未知、已发现、可清除、已清除、已证无源”管理。首次获得 direction 或 near 即确立该频道存在源。最小包围圆判据决定清除时机，成功响应将状态更新为已清除。频道的候选集为空，或其所有覆盖站检测结果均为无信号时，可判定该频道无源。

8.2 七点覆盖的解析证明

取站点集合

$$S(a) = \{(0, 0)\} \cup \left\{ a \left(\cos \frac{k\pi}{3}, \sin \frac{k\pi}{3} \right) : k = 0, \dots, 5 \right\}. \quad (19)$$

命题 4 (七点全向覆盖). 当 $0 < a/\sqrt{3} \leq R$ 时，任意 $g \in \Omega$ 到最近站点的距离不超过

$$D_{\text{cov}}(a) = \max \left\{ \frac{a}{\sqrt{3}}, \sqrt{R^2 + a^2 - \sqrt{3}Ra} \right\}. \quad (20)$$

证明. 设 g 距原点为 r , 其方位与最近环站的夹角不超过 $\pi/6$ 。若 $r \leq a/\sqrt{3}$, 选取原点即可。若 $r \geq a/\sqrt{3}$, 由余弦定理, 距最近环站的平方不超过 $r^2 + a^2 - \sqrt{3}ar$ 。该凸二次式在 $[a/\sqrt{3}, R]$ 上的最大值于端点取得, 分别为 $a^2/3$ 与 $R^2 + a^2 - \sqrt{3}aR$ 。□

当前 bayes 策略取 $a = 1159.7$ m, 由式 (20) 得 $D_{\text{cov}} = 984.539136$ m, 比最小接收半径小约 15.461 m。因此每个合法全向源至少在一站可被接收。某频道在七站全部为无信号, 就构成该频道不存在源的严格证据。以“站点、频道”为索引建立检测状态矩阵, 记录各站对各频道的检测结果。

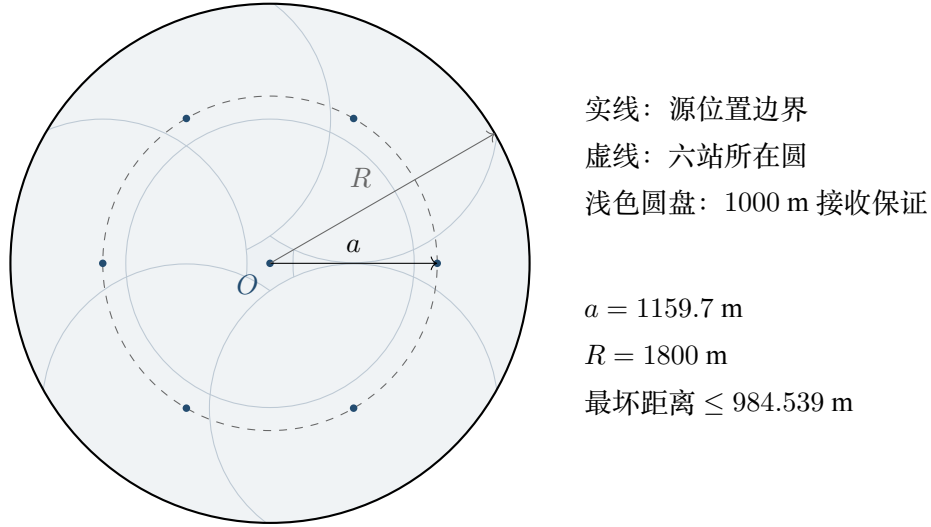


图 7 七点全向覆盖的站点构造与接收圆盘。

图 7 给出站点构造与最小接收圆盘的关系。顺次从原点访问六个环站的开放路径长为 $6a$ 。任务总路径由覆盖站间移动以及定位、清除所需的局部移动共同构成。

8.3 正负观测的双重候选表示

每个已发现频道维护凸外包 P_c^{out} 。direction 将候选集与 $\overline{B}(s, 1500)$ 外切多边形及测向角域相交; near 与 $\overline{B}(s, 5)$ 外切多边形相交。全向源的 no_signal 意味着

$$\|g - s\| > r_{\text{eff}} \geq 1000, \quad (21)$$

排除圆盘可使可行区域呈非凸形状。为此, 采用网格候选集 $\mathcal{G}_c$ 表示负观测带来的空间约束, 同时保留便于几何运算的凸外包。

网格边长为 $\ell = 14.65$ m, 每格中心与格内点的最大距离为 $\epsilon_g = \ell/\sqrt{2}$ 。初始保留所有可能与 Ω 相交的格, 按以下充分排除条件更新格心 z :

$$\begin{aligned} \text{no_signal: } & \|z - s\| \leq 1000 - \epsilon_g \Rightarrow \text{删除;} \\ \text{near: } & \|z - s\| > 5 + \epsilon_g \Rightarrow \text{删除;} \\ \text{direction: } & \|z - s\| > 1500 + \epsilon_g \Rightarrow \text{删除.} \end{aligned} \quad (22)$$

方向观测另外检查角差。若 $d = \|z - s\| > \epsilon_g$, 源所在格引起的方位偏差可由 $\arcsin(\epsilon_g/d)$ 上界; 实现采用距离内缩后的角误差界, 并在测点附近保留所有可能方位。只有角差超过 δ_{eff} 加格内角裕量时才删除。

上述排除条件保持真源所在网络的可行性。因此网格候选集为空时, 该频道无源。若全部候选格心均满足 $\|z - s\| > 1500 + \epsilon_g$, 则在 s 测量必无信号, 且所有候选网格已在最小接收圆盘外, 其观测的信息增益为零, 可在检测状态矩阵中标记该推断结果, 从而节省相应检测与换频时间。

8.4 信息增益选点与清除判据

问题二的算法 2 提供满足接收约束的检测点; 问题三的 BayesLoop 进一步采用网格候选集上的信息增益。存在保证接收圆盘时, 在其内部生成环形候选并补充当前接近方向上的测点; 其余情况下, 在网格最小包围圆附近生成环形采样点。候选集合同时包含当前位置与探索点, 分别根据预期信息增益和移动代价评分。对候选 q 计算

$$J_{\text{IG}}(q) = \hat{\mathbb{E}} \left[\log \frac{|\mathcal{G}_c^{\text{post}}(q, Y)|}{|\mathcal{G}_c|} \right] + \lambda \left(\frac{\|p - q\|}{5} + \frac{\|q - m_c\|}{5} \right), \quad (23)$$

其中 $\lambda = 0.05$, 空分支用正的小值截断。较小的候选集基数比对应较大信息增益, 移动项限制为测量而产生的绕行。评分最多取 800 个子样本及 7 个代表假设, 并以均匀 $r_{\text{eff}} \in [1000, 1500]$ 估计接收概率。该决策采用离散假设上的加权期望, 接收半径的均匀分布作为选点先验。

清除判断使用完整候选集合。凸候选集的清除判据为

$$\rho_c + b_c + 0.5 \leq 20, \quad (24)$$

其中 $b_c \geq 0$ 为代码的累计附加余量。凸多边形包含真源, ρ_c 是其最小包围半径; b_c 与 0.5 m 共同提供附加数值裕量。若负观测使网格更紧, 也可使用

$$\rho_{\text{grid}} + \epsilon_g + 0.5 \leq 20 \quad (25)$$

的网格清除判据, 其中 ρ_{grid} 为候选格心的最小包围半径。满足任一充分条件即可选择清除点。利用剩余距离裕量, 可将圆心向当前位置平移 Δ ; 只要平移后的距离上界仍满足对应不等式, 三角不等式便保证清除条件成立。

8.5 开放路径上的滚动任务排序

统一任务表包含三类任务: 覆盖站的批量扫频、已发现源的定位测量、满足半径条件的清除。站内检测优先使用测向机当前频道, 减少换频次数。对任务序列 π , 排序代价为

$$C(\pi) = \frac{\|p - q_{\pi_1}\|}{5} + \sum_{j=1}^{k-1} \frac{\|q_{\pi_j} - q_{\pi_{j+1}}\|}{5} + \sum_{j=1}^k t_{\pi_j}(c_{j-1}), \quad (26)$$

其中服务时间 t_{π_j} 依赖进入任务时的频道，清除不改变该频道。从最近邻初序开始，默认最多进行 4 轮开放路径 2-opt 改进；每次只执行当前序列的首任务，得到新观测后重新生成和排序。这样新发现的源、收缩后的清除点和已证无源的频道都能及时改变路线。

已停靠位置可以顺带补测：要求此前未在该位置测过、距离保证成立、尚处于定位阶段，且预测半径有足够收缩。追加测量的费用为检测与换频时间，其收益来自节省额外移动。题设固定同地误差，因此优先选择新的检测位置。

8.6 算法三与完成条件

算法 3：七点覆盖、网格更新与滚动调度（对应 algo-3）

输入：题设参数、七站计划、合法观测接口。 输出：逐频道处理状态与会话统计。

1. 检验七点覆盖条件，初始化频道状态、检测状态矩阵和网格候选集。
 2. 进入后同步记录位置、频道及剩余现实预算，必要时执行有信息的机会测量。
 3. 构建覆盖、定位和清除任务；跳过能够证明无信息的检测。
 4. 按最近邻与开放路径 2-opt 排序，执行首任务，更新候选集和频道状态。
 5. 若满足清除判据，执行清除；成功则更新状态，失败则追加观测。
 6. 若已清除 16 个源，或每个频道均已清除或有严格无源证据，则退出。否则返回步骤 2；预算不足或异常时退出并报告未解决频道。
-

“清除数达到 16”利用题设数量上界。对任意已存在的 16 个不同频道，也可提前排除其余未知频道，但已发现而未清除的源仍必须继续处理。

覆盖条件用于判定搜索范围，清除充分条件用于确定操作位置。当前每轮追加有效定位测量上限为 8；停滞、协议异常或预算耗尽时，记录各频道状态并终止。程序以接口返回的现实预算为准，并预留退出时间，实际计时在整数微秒状态上累计，避免用实机运行秒数代替虚拟动作耗时。

8.7 运行过程示例

图 8 回放一局全向演练（案例 BBM9-HZG2-77N5-NKKC，10 个源）：左图为移动轨迹、检测示向度与已清除频道，右上方为事件流水，下方为虚拟时间轴上的检测与清除时刻。该局说明覆盖扫描、定位测量与清除三类任务的交替关系，以及为接收向外发射的源而短时越出 Ω 边界的路径形态；它属于演练留档，不计入正式成绩。

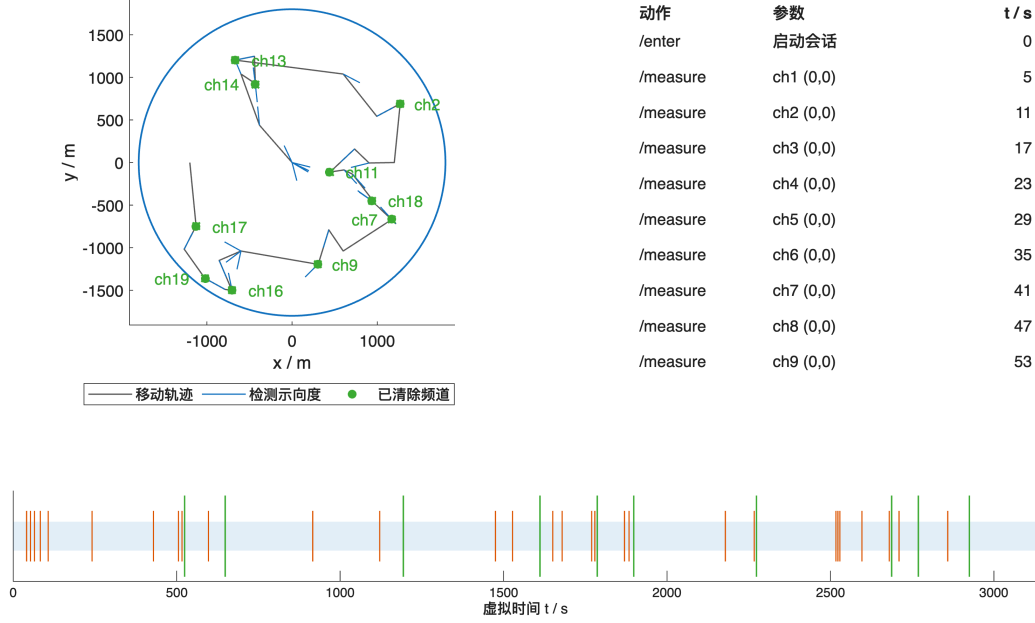


图 8 全向源演练局 (BBM9-HZG2-77N5-NKKC) 的运行回放。该局为演练案例，非正式测试记录。

9 问题四：混合源的定向覆盖与网格遍历清除

9.1 定向观测模型与无信号歧义

设定向源位于 g ，朝向为 ψ 。在检测点 q 可见的充要条件为

$$\|q - g\| \leq r_{\text{eff}}, \quad (q - g) \cdot u(\psi) \geq 0. \quad (27)$$

边界属于可见半平面。可见且距离不超过 5 m 时返回 near，否则返回 direction；任何一个条件失败均为 no_signal。问题二的可行区域满足距离约束，定向接收还需满足前向半平面约束。对于已发现频道，位于该区域内的负观测反映朝向遮挡；区域外的负观测则同时包含距离与朝向两种解释。

向外发射的边界源需要在圆域外布置接收站点。适用于混合源的站点集合 S 应满足

$$\forall g \in \Omega, \forall \psi \in [0, 2\pi), \exists s \in S: \quad \|s - g\| \leq 1000, \quad (s - g) \cdot u(\psi) \geq 0. \quad (28)$$

这是一项针对“位置、朝向”二元状态的全覆盖要求。

9.2 方格点集连续覆盖性

取 $S_{49} = \{(700i, 700j) : i, j = -3, \dots, 3\}$ 。其网格方域包含 Ω ，每个源落在某个边长为 700 m 的闭方格内。

命题 5 (方格四角的定向覆盖). S_{49} 满足式 (28)，对全向与任意朝向的定向源均保证发现。

证明. 设包含 g 的方格四角为 $v_1, \dots, v_4$ 。四角距 g 均不超过对角线长 $700\sqrt{2} < 1000$ 。由于 g 是四角的凸组合，存在 $\lambda_i \geq 0$ 、 $\sum_i \lambda_i = 1$ 使 $\sum_i \lambda_i (v_i - g) = 0$ 。对任意朝向 u 取内

积，得 $\sum_i \lambda_i (v_i - g) \cdot u = 0$ ，故不可能四个角均满足 $(v_i - g) \cdot u < 0$ ；至少一角处于前向闭半平面，且同时满足距离条件。□

四角的距离上界与凸组合性质共同保证连续覆盖。该方案具有显式站点坐标和覆盖距离上界。

9.3 基于集合覆盖的 31 点方案

当前 `directional` 实现采用 `q4_mincover_v2` 计划。在候选站点 s_j 和离散目标状态 (g_i, ψ_i) 上定义关联矩阵

$$A_{ij} = \mathbf{1}\{\|s_j - g_i\| \leq r_{\text{design}}, (s_j - g_i) \cdot u(\psi_i) \geq \|s_j - g_i\| \sin \gamma_{\text{design}}\}. \quad (29)$$

站点筛选可写为集合覆盖模型

$$\min \sum_j z_j, \quad \sum_j A_{ij} z_j \geq 1 \quad (\forall i), \quad z_j \in \{0, 1\}. \quad (30)$$

采用集合覆盖的贪心算法与冗余站点删除求近似解 [6]，得到 31 个站点。设计阶段使用 965 m 距离与 5° 前向角余量，最终按 15 m 位置间距、72 个朝向采样复核。验证结果覆盖全部 3 256 200 个离散状态，最坏覆盖距离约为 990.051 m。

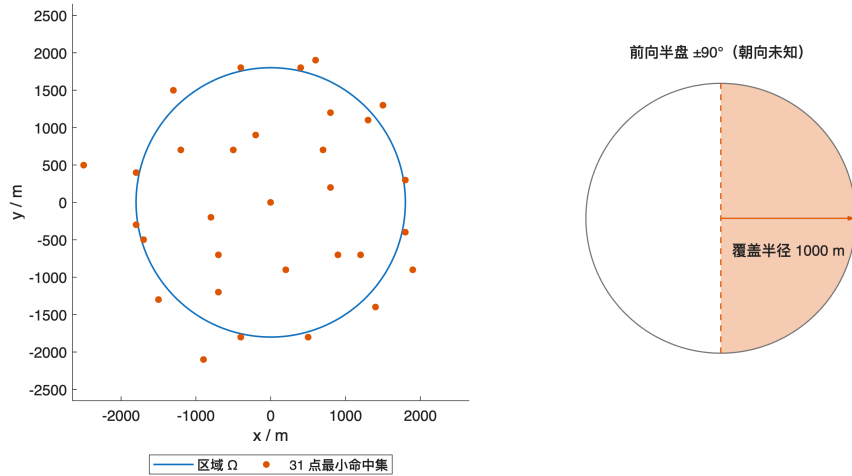


图 9 31 点最小命中集的几何分布（左）与定向源的前向半盘语义（右）。站点取自计划文件，部分站点位于 Ω 之外，用于接收向外发射的边界源；离散校验不等同于连续覆盖证明。

从离散覆盖推广到连续覆盖时，需同时计入位置与朝向的离散化误差。设任意实际位置距某采样位置不超过 ε ，实际朝向距其采样朝向不超过 α ，覆盖站点距采样位置为 d 。由三角不等式及单位向量差的上界，一组充分条件为

$$d + \varepsilon \leq 1000, \quad (s - g_i) \cdot u(\psi_i) \geq \varepsilon + 2d \sin(\alpha/2). \quad (31)$$

采样网格应覆盖 Ω 及其边界。本文对 31 点方案采用离散集合覆盖模型，对 49 点方案采用上述连续覆盖定理；连续近似误差的控制将在未来工作中讨论。

9.4 位置与朝向联合候选的保守更新

对定向假设，在每个候选位置网格的中心 z 附加 $H = 24$ 个朝向档，档宽为 15° 。令 M_z 为仍可能的朝向掩码。方向与近距离观测先更新位置约束，再保留可能覆盖检测点的朝向；无信号时只有在整格处于 1000 m 保证接收盘内，才可删除必然可见的朝向。

记 $\beta = \text{bearing}(z \rightarrow s)$ 、 $\Delta_\psi = 7.5^\circ$ ，格内方位余量为 α_z ，查表舍入余量为 $\eta_\psi = 1.5^\circ$ 。以 d_o 表示圆周最小角差，保守保留条件为

$$\begin{aligned} \text{正观测: } d_o(\psi_k, \beta) &\leq 90^\circ + \Delta_\psi + \eta_\psi + \alpha_z, \\ \text{盘内负观测: } d_o(\psi_k, \beta) &> 90^\circ - \Delta_\psi - \eta_\psi - \alpha_z. \end{aligned} \quad (32)$$

第二行保留含有不可见方向的朝向区间，排除整个区间均可见的候选。若格心距测点过近，格内方位可覆盖整个圆周，此时保留全部朝向。在定向假设下，上述更新保持真源所在位置与朝向网格的可行性。混合类型的集合估计扩展见第 12.2 节。

9.5 可见性筛选与探点选择

联合候选的用途首先是排序探点。对候选 q ，以均匀接收半径假设给出距离权重

$$w_r(d) = \begin{cases} 1, & d \leq 1000, \\ (1500 - d)/500, & 1000 < d < 1500, \\ 0, & d \geq 1500. \end{cases} \quad (33)$$

记 $\tilde{\mathcal{G}}_c$ 为位置样本集， $b_z(q)$ 为与检测点 q 相容的候选朝向档数，采用如下可见性权重：

$$V(q) = \frac{1}{|\tilde{\mathcal{G}}_c|} \sum_{z \in \tilde{\mathcal{G}}_c} w_r(\|z - q\|) \frac{b_z(q)}{24}. \quad (34)$$

分母 24 为初始朝向档数， $b_z(q)$ 由保留朝向集合与预计算可见角区间的交集确定。该权重用于衡量各候选点的相对检测价值。

候选族包括算法 2 给出的基础点、关于候选圆心的对跖点，以及 $0.5\rho, \rho, 1.5\rho$ 半径上的八方向环点。排除已被遮挡的重复点后，保留评分达到当前最大值 70% 的点，并从中选择

$$\frac{\|p - q\|}{5} + \frac{\|q - m\|}{5} + 6 \quad (35)$$

最小者。常数 6 是检测加换频的选点评分费用，真实结算仍按当时频道计算。该规则优先避免完全遮挡的点，再减少移动及后续接近距离。联合集不可用时，返回算法 2、对跖探点和环探的确定性顺序。持续遮挡时转入基于距离约束的网格遍历清除。

9.6 网格遍历清除与计算预算

当一轮有效定位测量达到 8 次，或连续遮挡探点达到 4 次且候选区域仍较大，对保守凸候选多边形建立边长 20 m 的方格覆盖。保留所有与多边形相交的格子，并以格心

作为清除点。格内任一点到格心距离满足

$$\|g - z\| \leq 10\sqrt{2} < 20. \quad (36)$$

因清除与源朝向无关，只要真源保留在多边形中，逐格清除必有一次到达其 20 m 内。按最近邻顺序访问，首次成功即停止该频道的网格遍历。

若外包框宽、高为 w, h ，保守格数上界为

$$B_{\text{cell}} = \left(\left\lceil \frac{w}{20} \right\rceil + 2 \right) \left(\left\lceil \frac{h}{20} \right\rceil + 2 \right). \quad (37)$$

当前格数硬上限为 400，现实预算需覆盖 $N_{\text{cell}} t_{\text{RTT},95} + t_{\text{reserve}}$ 。超过格数或预算上限即跳过并报告未解决；全部清除点失败则保留候选集并记录异常。在候选区域包含真源、格数满足上限且预算足够的条件下，逐格访问可保证完成该源的清除。

9.7 算法四与终止条件

算法 4：定向集合覆盖、选点与网格遍历清除（对应 **algo-4**）

输入：混合源接口，定向覆盖计划与预算。 输出：已清除频道、未解决频道、停止依据。

1. 加载覆盖计划；默认 31 点按声明分辨率验证，49 点备选有连续证明。
 2. 选择最近的待测站，当前频道优先，批量检测仍未知的频道。
 3. 更新凸位置候选；首次正观测后建立朝向候选，用于探点可见性筛选。
 4. 按移动代价处理已发现频道；满足半径判据则清除，否则继续探点。
 5. 遮挡或定位停滞时检查格数与预算，允许则执行 20 m 方格遍历清除。
 6. 覆盖任务结束后处理未解决频道；主动退出时报告清除数及停止依据。
-

问题四使用“本站批量扫频、最近候选圆心服务”的调度骨架，并未直接复用问题三的统一信息增益任务表。若要求严格的混合源全清证据，除采用连续覆盖计划外，还须补全类型联合表示或限制联合集排空的适用范围，并确保全部定位清除任务完成。本文区分当前实现、可证明条件与尚需完善之处，不把后者追溯为正式测试时已执行的功能。

9.8 运行过程示例

图 10 回放一局含定向源的演练（案例 GHSN-M38E-74GD-YLUD）：左图在轨迹、检测示向度与已清除频道之外，画出每个定向源的前向半盘（ $\pm 90^\circ$ ，半径取该源的 r_{eff} ），箭头方向即朝向。图中存在落在接收半径之内、却因位于背向半平面而无信号的检测点，对应式 (27) 的可见性条件，保留这一解释正是问题四与问题三在候选更新上的区别；该局同样不计入正式成绩。

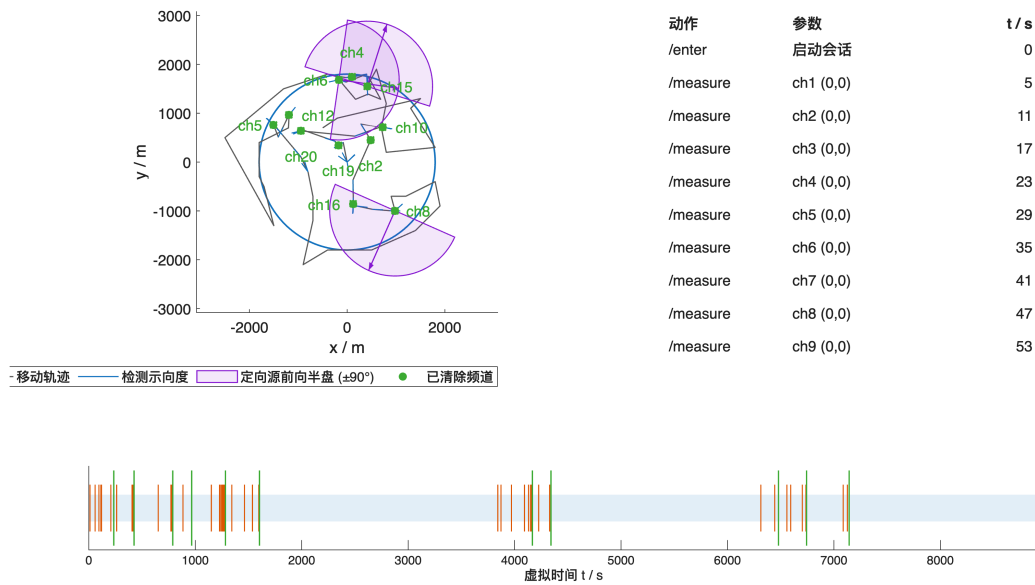


图 10 含定向源演练局 (GHSN-M38E-74GD-YLUD) 的运行回放，紫色半盘为定向源的前向可见区域。该局为演练案例，非正式测试记录。

10 官方正式测试结果与时间分解

10.1 数据选择及对应关系核对

本节实验样本严格限定为官方模拟器 Jammers-simulator-full/test 下的六份正式导出包，问题三、四各有三次。文件名、案例编码与完整包校验值见结果表与附录 A。本节表格不采用 project/results 中的自建评测数据，也不将历史演练案例当作正式测试补入：自建配对对比见第 11 节，演练局示例见问题三、四的运行过程示例，三类数据分开陈述，不合并统计。

评价指标包括清除数量 N_c^+ 、平均定位清除时间 $\bar{T}$ 、程序运行时间 T_{real} ，以及检测、换频和清除的动作次数。其中 $\bar{T} = T/N_c^+$ ，虚拟时间 T 按整数微秒累计， T_{real} 采用官方记录的毫秒值。六次测试均以主动退出结束，统计提交状态为已确认。

10.2 问题三正式结果

表 4 问题三正式测试结果（时间单位：s，平均时间单位：s/个）

测试	测试案例编码	清除数	平均定位清除时间	程序运行时间
测试 1	9HU7-UWJR-BPTR-VYAT	13	238.075	5.596
测试 2	7TCG-C9EC-53VF-HZ5H	15	232.889	3.024
测试 3	UVUZ-B4MQ-XSUV-HQFP	10	290.686	2.680

三次正式测试分别清除 13、15、10 个源，累计 38 个，平均定位清除时间分别为 238.075、232.889、290.686 s/个。三次总虚拟时间之和为 9495.176084 s，按清除数量加

权的平均时间为

$$\bar{T}_{3,w} = \frac{\sum_{i=1}^3 T_{3,i}}{\sum_{i=1}^3 N_{3,i}^+} = \frac{9495.176084}{38} = 249.873 \text{ s/个}. \quad (38)$$

将三个案例等权平均，得到 253.884 s/个。加权平均衡量全部已清除源的平均时间成本，等权平均衡量三个案例的平均表现。

第三次的总虚拟时间为 2906.863 s，在本组三次测试中最短；其清除数为 10，因而每源平均时间为 290.686 s/个。清除数量与总时间共同决定每源时间，表 4 同时列出这些指标以便比较。程序运行时间为 2.680 至 5.596 s，反映了本组案例中的实际计算与通信开销。

10.3 问题四正式结果

表 5 问题四正式测试结果（时间单位：s，平均时间单位：s/个）

测试	测试案例编码	清除数	平均定位清除时间	程序运行时间
测试 1	VQUM-ETQ5-74PP-ZQ3J	13	720.171	7.668
测试 2	5K5T-TRWZ-BPR2-ECAV	10	927.130	5.724
测试 3	K58B-V5VC-E4EF-CRTW	14	614.884	6.051

表 5 给出问题四的三次正式结果。三次测试分别清除 13、10、14 个源，累计 37 个，平均定位清除时间分别为 720.171、927.130、614.884 s/个。总虚拟时间之和为 27241.909918 s，相应的加权平均时间为

$$\bar{T}_{4,w} = \frac{\sum_{i=1}^3 T_{4,i}}{\sum_{i=1}^3 N_{4,i}^+} = \frac{27241.909918}{37} = 736.268 \text{ s/个}. \quad (39)$$

三个案例的等权平均时间为 754.062 s/个。

第三次清除数为 14，总虚拟时间为 8608.382 s，每源平均时间为 614.884 s/个；在本组三个案例中，其清除数量最多，平均定位清除时间最短。程序运行时间为 5.724 至 7.668 s。各案例的结果变化反映了源分布、有效接收范围和朝向约束对搜索过程的共同影响。

10.4 动作耗时分解

将式 (1) 重排，得到累计移动时间

$$T_{\text{move}} = T - 5N_m - N_s - 5N_c^+ - 3N_c^-. \quad (40)$$

表 6 中 Q3-1 表示问题三第一次正式测试，其余编号依此类推。六次测试的清除失败计数均为零。各动作次数与式 (40) 共同确定图 11 的耗时分解。

表 6 六次正式测试的动作计数与移动耗时

案例	检测数	换频数	失败清除	总虚拟时间/s	移动时间/s	移动占比
Q3-1	105	96	0	3094.978	2408.978	77.84%
Q3-2	117	108	0	3493.335	2725.335	78.02%
Q3-3	117	107	0	2906.863	2164.863	74.47%
Q4-1	349	275	0	9362.226	7277.226	77.73%
Q4-2	361	315	0	9271.302	7101.302	76.59%
Q4-3	264	213	0	8608.382	7005.382	81.38%

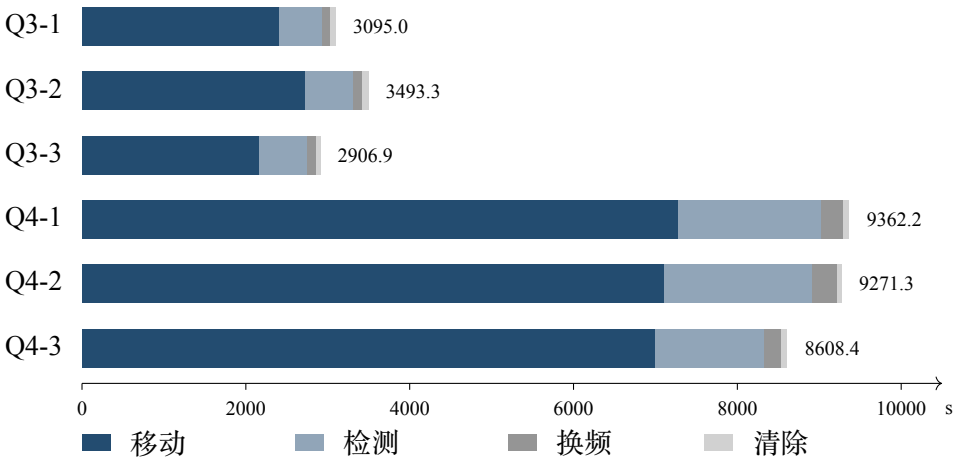


图 11 正式测试的累计虚拟耗时分解。移动时间由总耗时与各动作计数计算。

图 12 按同一记账口径给出一局全向演练的累计时间构成：横轴为动作序号，堆叠高度为累计虚拟时间，移动项在整局中占主要部分，清除阶段表现为阶梯状跃升。该图用于说明式 (1) 的分项在同一局内如何随时间累积，其数据来自演练留档，不进入正式统计。

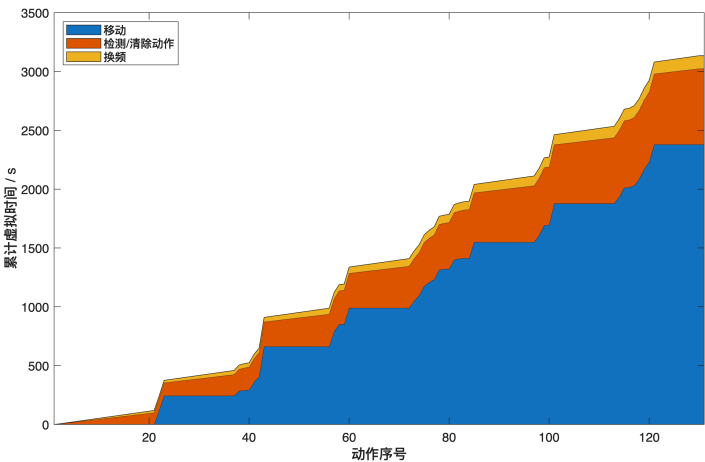


图 12 一局全向演练（BBM9-HZG2-77N5-NKKC，10 个源）的累计虚拟时间构成。该局为演练案例，不计入正式测试结果。

六次测试的移动时间占总虚拟耗时的 74.47% 至 81.38%，是主要的时间成本，减少移动因而也是缩短总耗时的主要方向，但检测与换频仍需一并计入路径选择。由 $L_{eq} = 5T_{move}$ 反推等效移动距离：问题三依次为 12.045、13.627、10.824 km，问题四依次为 36.386、35.507、35.027 km。该等效距离由官方动作计数与移动速度换算得到，不是独立轨迹测得的长度。所有平均值先以官方整数单位求和、相除，最后保留三位小数。

10.5 实验结果分析

六次正式测试累计清除 75 个源，程序运行时间介于 2.680 与 7.668 s。在所观测的三组全向案例中，七点覆盖与信息增益选点相结合，得到 232.889 至 290.686 s/个的平均定位清除时间；三组混合源案例采用定向集合覆盖与网格遍历清除，得到 614.884 至 927.130 s/个的相应结果。

两类问题的共同特征是移动耗时占比较高。路径排序应统筹覆盖站、定位测点和清除点的位置，同时计入频道切换与检测费用。平均定位清除时间、累计清除数和程序运行时间分别反映时间成本、任务处理规模与计算通信开销，三者共同构成本文的实验评价。

11 算法对比实验：最优实现与基线

11.1 对比口径与有效性判定

本节数据来自仓库内的配对对比框架 `project/`，用于回答上述几何判据与闭环策略相对朴素做法改进在哪里；它与第 10 节的六份官方正式导出完全分开，两类数字不合并、不相加。框架构建三种角色：最优实现直接复用 `src/mcm26b` 中已由测试固化的生产代码，只增加输入输出适配；基线实现是问题下更朴素但能独立跑通的对照，不共享最优实现的关键步骤；独立参考与分析算法正交，用于判定谁对谁错。问题三、四在内存回环评测机上运行，与官方接口使用同一请求与响应边界：源真值只在评测机一侧，策略端只读响应，因此全清率与缺席误判可以核查。一局有效必须同时满足全清、无缺席误判、正常 `/exit`、零异常与逐微秒计时复算一致，否则记为无效，失败样本不静默丢弃；所有对比按场景配对，并给出均值、中位与最大值。源位置均匀、误差场逐点固定等分布属于自建测试假设，题面并未规定，本节结论只描述各方法的相对行为。

表 7 中问题二、四的基线分别取“候选圆心”与“问题三策略直接搬用”，它们是各自问中最需要区分的对照，其余各表的配对数字同样如此选择。本节全部数字由 `scripts/build_paper_support.py` 从 `project/results/*.json` 生成，正文与表格均不手抄。

11.2 问题一：直径与覆盖判定

表 8 显示基线误差的来源是代理方式而非数值精度：它以轴对齐包围盒的对角线代替区域直径，把边长 40 m 的等边三角形判成 50 m，把零误差退化线段判成 0 m；覆盖

表 7 四问最优实现与基线的核心指标总览（自建基准集）

问	核心指标	最优实现	基线实现
问题	核心指标	最优实现	基线实现
1	直径平均绝对误差 / 覆盖判定准确率	0.000 m / 100.0%	4.583 m / 52.5%
2	保证接收率 / 最坏后验半径均值	100.0% / 73.890 m	100.0% / 340.747 m
3	全清率 / 平均虚拟耗时	100.0% / 3188.734 s	100.0% / 14935.659 s
4	全清率 / 缺席误判总数	100.0% / 0	77.0% / 24

表 8 问题一固定示例的直径与覆盖判定（单位：m， ΔD 为相对参考解的误差）

算例	D 参考	D 最优	D 基线	$ \Delta D $ 基线	覆盖参考	覆盖最优	覆盖基线
三点交会	26.986	26.986	21.213	5.773	否	否	是
直径圆反例	40.000	40.000	50.000	10.000	否	否	是
近共线检测点	—	—	—	—	—	—	—
零误差退化线段	100.000	100.000	0.000	100.000	是	是	是
不相容观测	—	—	—	—	—	—	—

表 9 问题一的汇总指标（直径误差单位：m，耗时单位：ms）

数据子集 / 方法	状态判定	直径平均误差	直径最大误差	覆盖判定	平均耗时
最优（固定 5 例）	100.0%	0.000	0.000	100.0%	0.064
最优（随机 40 例）	100.0%	0.000	0.000	100.0%	0.086
基线（固定 5 例）	80.0%	7.886	10.000	33.3%	14.202
基线（随机 40 例）	100.0%	4.583	14.599	52.5%	15.933

判定以“包围盒中心加代理直径圆”代替逐顶点检查，于是对任意包围盒几乎恒判覆盖。在 5 个固定示例上，基线的状态判定准确率为 80.0%，平均直径误差 7.886 m，最大误差 10.000 m，覆盖判定准确率仅 33.3%；在 40 个随机交会算例上状态判定回到 100.0%，但平均直径误差仍为 4.583 m，最大误差 14.599 m，覆盖判定准确率 52.5%。最优实现与独立参考解在全部算例上的状态、直径与覆盖判定一致；平均耗时为 0.086 与 15.933 ms（随机算例），说明精确几何并未以计算量为代价。

11.3 问题二：第二检测点选择

表 10 的算例由随机首次示向度生成，首次候选集的平均包围半径为 681.467 m。垂直 1000 m 的基线交会角最大，却忽略接收保证，保证接收率为 0.0%：真源位于角域远端时第二点必然无信号，其最坏后验均值 80.111 m 是在丢失样本上算出的，不能与其它方法直接比较。候选圆心基线保证接收（100.0%），但不增加横向基线，最坏后验半径均值仍为 340.747 m，终值比 0.500，只收缩一半。默认选点器在保证接收下取得最坏

表 10 问题二各方法在 30 个随机首次示向度算例上的汇总（半径与收缩单位：m，移动时间单位：s）

方法	保证接收率	最坏后验均值	中位	平均收缩	终值比	移动均值
算法 2（默认选点）	100.0%	73.890	90.899	607.577	0.104	170.44
参考：盘内网格枚举	100.0%	75.627	91.349	605.841	0.106	170.28
基线：垂直 1000 m	0.0%	80.111	90.775	601.356	0.116	200.00
基线：候选圆心	100.0%	340.747	375.088	340.721	0.500	136.29

后验均值 73.890 m、终值比 0.104，与保证接收盘内 9×9 网格枚举的参考解（75.627 m）相差约 2.3%，说明 37 点环形候选已接近该目标函数下的网格最优。这里的最坏后验是对候选外包多边形的采样评估，不是连续极小极大的全局上界；“保证接收”一项由安全盘独立成立。

11.4 问题三：全向闭环

表 11 问题三三种方法在 18 个冻结场景上的汇总（时间单位：s，位移单位：km）

方法	有效局	全清率	缺席误判	T 均值/s	T 中位/s	T 最大/s	检测	位移/km	后备清除
bayes（默认策略）	18/18	100.0%	0	3188.734	3111.256	3663.707	111.1	12.338	0.0
seven（七点批量）	18/18	100.0%	0	5361.707	5367.455	6462.351	100.8	23.508	0.0
grid（49 点网格）	18/18	100.0%	0	14935.659	15275.057	16691.406	430.6	61.550	0.0

表 11 汇总三种方法。它们在 18 个冻结场景（源数 10、13、16，种子与源数版本化）上全部完成全清，缺席误判均为零，差别只在时间：默认 bayes 平均 3188.734 s，七点批量 seven 5361.707 s，49 点网格 grid 14935.659 s。按场景配对，bayes 相对 grid 的平均虚拟时间下降 78.4%、中位下降 80.0%，在 18/18 场景上更快，seven 相对 grid 下降 63.7%。分项上，默认策略检测 111.111 次、位移 12.338 km，网格基线需要 430.556 次检测、61.550 km 位移；单局在线计算均在 1.6 s 以内（默认策略 0.767 s），不构成瓶颈。七点方案用解析覆盖保证发现，站点最少，但按站点批量扫频需要绕行，位移约为默认策略的两倍。可见三者的差别不是“能否完成”，而是覆盖证明与信息增益选点各自省下了多少移动与检测。

11.5 问题四：混合源闭环

表 12 汇总三种方法。11 个场景取自 core 与 boundary 两组，boundary 全部是背离圆心发射的边界定向源，属最不利工况。directional 在 11/11 局上全部有效，全清率 100.0%，平均后备清除 24.818 次，对应遮挡导致的定位停滞。把问题三的 bayes 直接搬用时全清率降到 77.0%，并出现 24 次缺席误判：七站全向缺席证明把朝外发射的定向源判为不存在，因此它用时最短（3890.801 s），但结论是错的。grid 的 49 点覆盖对定向源

表 12 问题四三种方法在 11 个场景上的汇总（时间单位：s，位移单位：km）

方法	有效局	全清率	缺席误判	T 均值/s	T 中位/s	T 最大/s	检测	位移/km	后备清除
directional（默认策略）	11/11	100.0%	0	9069.871	8803.261	12081.148	355.6	34.268	24.8
grid（问题三网格）	0/11	66.6%	0	18179.173	18345.164	20457.858	573.7	73.876	0.0
bayes（问题三策略）	0/11	77.0%	24	3890.801	3646.178	5054.382	132.0	15.319	0.0

发现完备，全清率 66.6%、平均虚拟时间 18179.173 s，却没有朝向推理与后备清除，定位停滞时只能停在“已发现”状态，0/11 局通过有效性判定。directional 平均 9069.871 s，比 bayes 高 142.9%，位移 34.268 km 对 15.319 km；由于基线 11 局全部无效，这一比值不构成“默认策略更慢”的证据，在定向源上应先保证判对、再比较快慢。

11.6 对比实验的结论与边界

四组配对比较支持的结论是：问题一的误差来自代理近似，改用精确顶点枚举与最小包围圆后消失；问题二要求同时满足接收保证与横向基线，只取其一都会被表 10 的某一系列暴露；问题三的三种方法都能全清，默认策略的收益来自信息增益选点与滚动排序而非覆盖证明本身；问题四的差别在正确性，搬用全向缺席证明会“快而错”，只扩大覆盖而不补朝向推理与后备清除则“稳而不全”。

边界有四条。随机分布是自建测试假设，场景数、源数与种子见 project/results/*.json，不能据此推断官方测试的分布。基线不是稻草人：候选圆心、七点批量与网格覆盖都能独立跑通并在部分场景有效，它们暴露的是各自缺少的那一步。问题二的最坏后验与问题四的可见性评分都是启发式评估量，不是全局最优值或校准概率。本节指标与第 10 节的官方统计口径不同，两组数字不可互相换算。

12 模型评价与推广

12.1 模型的有效性

本文采用集合估计处理有界测向误差，将每次观测转化为真实位置的可行约束。问题一通过凸包顶点计算直径，以最小包围圆半径给出清除充分条件；问题二将接收距离要求表示为闭圆盘的交集，并由三角不等式构造显式可行子集。几何定位、接收约束与清除距离由统一的集合运算联系起来。

多源搜索采用覆盖点集与检测状态矩阵维护各频道的搜索进度。网格表示可处理负观测产生的非凸可行区域，信息增益用于选择有利于收缩候选集的测点。开放路径排序将移动、检测、换频和清除纳入统一目标，并在新观测到来后重新求解任务次序。

对于定向源，位置与朝向的联合约束提供可见性信息，凸组合性质给出方格点集的连续覆盖证明，网格遍历清除则利用清除半径与方向无关的物理条件。六次正式测试提供了算法在全向及混合源案例中的清除数量和时间成本。

12.2 未来工作

提高离散选点的近似精度。算法 2 以边界采样和读数端点枚举近似最坏后验半径，信息增益评分采用离散先验，任务排序采用最近邻初始化与 2-opt 局部搜索。这些方法尚未给出整个动态任务的全局最优性保证。后续可用区间分支定界控制后验半径的近似误差，结合动态规划或滚动时域优化，研究测点选择与路径规划的联合求解。

建立 31 点方案的连续覆盖误差界。现有方案在 15 m 网格和 72 个朝向上完成离散验证。若采用位置误差界 $15/\sqrt{2}$ ，则 $990.051 + 15/\sqrt{2} \approx 1000.658$ m，超过最小接收半径；最终验证的前向角余量为 0° ，也不足以控制全部朝向扰动。因此当前离散结果尚不能推出连续覆盖。后续可按式 (31) 同时约束位置与朝向误差，或采用已证明的 49 点方案，在相同场景中比较站点数量与耗时。

扩展混合类型的集合估计。当前实现主要维护定向假设，并以 `joint_empty` 处理部分候选集为空的情形，尚未单独维护全向分支。全向源可从多个方向被接收，定向候选集为空不足以判定该频道无源。后续应将类型加入状态，维护

$$\mathcal{H}_c = \mathcal{H}_c^{\text{omni}} \cup \mathcal{H}_c^{\text{dir}}, \quad (41)$$

并分别更新两类可行集合。例如源周围三个相隔 120° 的测点可以同时接收全向信号，却不能同时属于一个固定前向半平面。类型并集模型可保留这一解释，使源存在性判断与观测条件一致。

研究资源约束下的终止性。连续覆盖与单次清除充分条件尚不足以保证所有案例均在有限预算内完成。持续遮挡、几何退化及较大的候选区域会增加定位步数，当前 400 格上限也会限制遍历清除。后续可设计自适应网格细化与计算预算分配，并研究最坏情况下的动作次数上界。

扩展统计评价。本文的六次正式测试完成了两问各三次案例的结果汇总，但样本量仍不足以代表所有位置与朝向分布。官方统计中的源总数及类型组成尚未公开，因此全清率与分类型指标有待补充。未来可在可重复场景中开展配对试验，同时保存策略版本、源数量和轨迹等信息，以估计置信区间并比较各模块的贡献。

第六，配对对比依赖自建基准集。第 11 节的最优与基线对比运行在内存回环评测机上，源位置均匀、误差场逐点固定等分布是测试假设，题面并未规定；场景数有限，基线在部分场景同样有效。该节结论说明的是各方法缺少哪一步，而不是普适的性能排序，也不能与官方正式统计互相换算。

12.3 参数影响的解析讨论

主要参数的影响可由模型公式分析。由 $a(\delta) = 1500/(2 \cos \delta)$ 得 $a'(\delta) = 750 \sec \delta \tan \delta$ ，其中 δ 以弧度计。误差半角增大会扩大源位置的包围圆，并缩小保证接收圆盘。表 3 给出了两位小数舍入余量对首次可行区域的影响。

由式 (20), 环半径影响覆盖路径长度和最大覆盖距离。默认半径保留约 15.461 m 的距离裕量。减小网格边长可降低 ϵ_g 并收紧位置误差上界, 而二维网格数量大致与 ℓ^{-2} 成正比, 因此应在精度、内存和计算时间间取得平衡。

12.4 模型推广

该模型适用于有界方向误差下的静态信号源排查、室内设备巡检和移动传感器主动定位。推广时, 根据新的设备参数校准接收半径、测向误差和动作时间; 对于移动目标或额外遮挡, 可扩展状态转移与观测约束, 继续采用集合估计、覆盖优化和路径排序进行联合决策。

参考文献

- [1] 全国大学生数学建模竞赛组委会. 2026 年高教社杯全国大学生数学建模竞赛 B 题: 无线电干扰源的快速自动定位与清除 [Z]. 2026.
- [2] 赛题配套材料. 模拟器使用说明; 模拟器通信接口说明及编程指南 [Z]. 2026.
- [3] 全国大学生数学建模竞赛组委会. 全国大学生数学建模竞赛论文格式规范 (2026 年修订稿) [Z]. 2026.
- [4] WELZL E. Smallest enclosing disks (balls and ellipsoids)[C]// New Results and New Trends in Computer Science. LNCS 555. Berlin: Springer, 1991: 359–370. <https://doi.org/10.1007/BFb0038202>.
- [5] BOYD S, VANDENBERGHE L. Convex Optimization[M]. Cambridge: Cambridge University Press, 2004. <https://web.stanford.edu/~boyd/cvxbook/>.
- [6] CHVÁTAL V. A greedy heuristic for the set-covering problem[J]. Mathematics of Operations Research, 1979, 4(3): 233–235. <https://doi.org/10.1287/moor.4.3.233>.
- [7] 官方无线电干扰源环境模拟器. 问题三、四正式行为日志与配套统计 (客户端 1.1.0, 六个案例) [DS]. 2026-09-12–2026-09-13. 案例编码及文件校验值见附录 A。

A 正式数据清单与复算说明

六份正式导出文件及其 SHA-256 校验值列于表 13。论文采用原项目模板，按 2026 年格式规范设置 A4 纸张、四边距 2.5 cm，电子版从摘要页开始连续编号 [3]。所给往届论文用于章节组织与版式参考。

表 13 所选正式导出文件与完整包哈希

案例	原始文件名及 SHA-256
Q3-1	formal-p3-1-9HU7-UWJR-BPTR-VYAT.jlog fbd7ab92f78a019887f603172ab6a39e274adb1f917fc2681f1e8073ff47d911
Q3-2	formal-p3-2-7TCG-C9EC-53VF-HZ5H.jlog 5d83ce87455ce22fadbcdbdb60814d9b1f0be54de7bc9d0463777ba47b3922ef
Q3-3	formal-p3-3-UVUZ-B4MQ-XSUV-HQFP.jlog eeffe419c3df005d0ea78c7ec2c10afed29c4bec782327538537dd7ba7914424
Q4-1	formal-p4-1-VQUM-ETQ5-74PP-ZQ3J.jlog 9a102295d324b5fb5aeed1c6036d13ae7e08a7d7fddb60ceb8dead24e62b15d3
Q4-2	formal-p4-2-5K5T-TRWZ-BPR2-ECAV.jlog 370aac9c2c5d324dd382e57e541ef0da7c5dd4759660d9ea0cba8373ce0ea9a1
Q4-3	formal-p4-3-K58B-V5VC-E4EF-CRTW.jlog 1c7f1d111f8fd1a1d0f608d9847c0946b1b0a45bdb7b3f487e0c6d4f4586885f

复算脚本 `scripts/audit_paper_formal.py` 将正式导出、结果侧文件与官方统计记录匹配，生成结构化数据、CSV 和 LaTeX 表格。在项目根目录执行：

```
python scripts/audit_paper_formal.py
python scripts/audit_paper_formal.py --snapshot docs/paper/data/formal-results.json
```

第一条命令完成来源匹配与统计计算；第二条根据已有快照重新计算指标，并核对现有正式包的校验值。耗时图与汇总量由同一统计快照生成。

第 11 节的配对对比表由 `scripts/build_paper_support.py` 从 `project/results/*.json` 生成，复跑对比框架本身执行 `uv run python -m project.run`。正文插图由 `scripts/build_figure_data.py` 准备数据、`scripts/matlab/run_all_figures.m` 出图，逐图来源与生成命令见 `docs/figures/manifest.json`。其中回放画布与实测计时两张插图取自评测机演练留档，只用于说明机制与时间构成，不参与正式统计。

B 支撑材料与运行方法

表 14 支撑材料目录及用途

文件或目录	用途
src/mcm26b/	四问几何、信念、覆盖、调度与串行通信实现
src/mcm26b/guarantee/plans/	第四问已实现的覆盖计划及验证参数
project/	最优与基线配对对比框架及其冻结基准集结果
scripts/audit_paper_formal.py	正式文件对应核对、匿名快照及表格复算
scripts/build_paper_support.py	对比表、矢量图、源码附录与支撑包生成
scripts/build_figure_data.py、	插图数据准备与 MATLAB 出图脚本
scripts/matlab/	
docs/paper/data/	六个正式案例的可读统计、表格及校验值
docs/paper/sections/	分章论文源文件
docs/paper/figures/	正式累计耗时图的矢量源文件
docs/figures/	正文插图（PNG）及其数据清单与生成命令
Jammers-simulator-full/test/	六份原名官方正式日志
support-manifest.json	支撑包文件清单及逐文件 SHA-256

Python 运行时需要 requests, numpy 提供可选加速。将 src 加入 Python 模块路径后, 可调用 `python -m mcm26b.geometry` 进行问题一几何分析, 或通过 `mcm26b.cli` 运行在线策略。在线测试由模拟器界面启动, 接口就绪后执行策略入口。具体运行指令和编译步骤见支撑包的 README.md。

C 核心程序代码

四问的求解核心集中在约 5000 行的单文件程序中, 按公共层、几何与集合估计层、覆盖保证层、策略层四层组织; 评测机、命令行外壳、数据采集与绘图脚本、单元测试以及未被采用的方案不属于求解核心, 不在此列出。完整可执行程序为支撑材料中的 `mcm26b-.ipynb`, `src/mcm26b/` 保留与之逐字对应的模块化版本, 两者函数名一一对应。下列代码为无改动的摘录: 函数体与注释与程序一致, 只按核心程度取舍。未收录的部分包括独立评测机 (按附件协议复现模拟器, 与求解代码零耦合)、命令行入口、数据采集与论文绘图脚本、单元测试, 以及经实测未采用的候选方案。

动作计时与开放路径排序

三段耗时按整数微秒累计; 后备清除的走格顺序用最近邻打底并做 2-opt 改进。
CostBreakdown: 一次合法动作的三段耗时: 移动 + 换频道 (仅检测) + 动作本身。

```
1 class CostBreakdown:
2     """一次合法动作的三段耗时 [题面]: 移动 + 换频道 (仅检测) + 动作本身。"""
3
4     move_s: float
5     switch_s: float
6     action_s: float
7     @property
8     def total_s(self) -> float:
```

```

9         return self.total_us / 1_000_000
10
11     @property
12     def total_us(self) -> int:
13         """逐段量化后按整数微秒累计，与官方虚拟时钟一致。"""
14         return sum(round(value * 1_000_000)
15                     for value in (self.move_s, self.switch_s, self.action_s))

```

measure_cost: 合法 /measure 的耗时；移动按直线距离，换频道只看前后频道是否不同。

```

1 def measure_cost(prev_pos: Vec2, pos: Vec2, prev_channel: int, channel: int,
2                 cost: CostConfig = CostConfig()) -> CostBreakdown:
3     """合法 /measure 的耗时。移动按直线距离；换频道只看前后频道是否不同。"""
4     return CostBreakdown(
5         move_s=dist(prev_pos, pos) / cost.speed_mps,
6         switch_s=0.0 if channel == prev_channel else cost.switch_s,
7         action_s=cost.measure_s,
8     )

```

clear_cost: 合法 /clear 的耗时；清除不切换测向机频道，故不含换频项。

```

1 def clear_cost(prev_pos: Vec2, pos: Vec2, success: bool,
2               cost: CostConfig = CostConfig()) -> CostBreakdown:
3     """合法 /clear 的耗时。"""
4     return CostBreakdown(
5         move_s=dist(prev_pos, pos) / cost.speed_mps,
6         switch_s=0.0,
7         action_s=cost.clear_success_s if success else cost.clear_fail_s,
8     )

```

move_time: 任意两点间的移动耗时，供规划使用。

```

1 def move_time(a: Vec2, b: Vec2, cost: CostConfig = CostConfig()) -> float:
2     """任意两点间的移动耗时（规划用）。"""
3     return dist(a, b) / cost.speed_mps

```

nearest_neighbor_order: 最近邻开放路径，作为路径排序的启发式上界。

```

1 def nearest_neighbor_order(points: list[Vec2],
2                             start: Vec2 = (0.0, 0.0)) -> list[Vec2]:
3     """最近邻开放路径 ( $O(n^2)$ ，启发式上界)。"""
4     remaining = list(points)
5     order: list[Vec2] = []
6     position = start
7     while remaining:
8         nxt = min(remaining, key=lambda q: math.dist(position, q))
9         remaining.remove(nxt)
10        order.append(nxt)
11        position = nxt
12    return order

```

two_opt_open: 开放路径 2-opt 改进；终点自由，只反转内部片段。

```

1 def two_opt_open(order: list[Vec2], start: Vec2 = (0.0, 0.0),
2                 max_passes: int = 8) -> list[Vec2]:
3     """开放路径 2-opt (终点自由：只反转内部片段，不做首尾相接)。"""
4     best = list(order)
5     n = len(best)
6     if n < 3:
7         return best
8     for _ in range(max_passes):
9         improved = False
10        for i in range(n - 1):
11            prev = best[i - 1] if i > 0 else start
12            pi = best[i]
13            d_prev_i = math.dist(prev, pi)
14            for j in range(i + 1, n):
15                pj = best[j]
16                nxt = best[j + 1] if j + 1 < n else None
17                d_prev_j = math.dist(prev, pj)
18                removed = d_prev_i
19                added = d_prev_j
20                if nxt is not None:
21                    removed += math.dist(pj, nxt)
22                    added += math.dist(pi, nxt)
23                if added < removed - 1e-9:

```

```

24         best[i:j + 1] = best[i:j + 1][::-1]
25         improved = True
26         pi = best[i]
27         d_prev_i = math.dist(prev, pi)
28     if not improved:
29         break
30     return best

```

问题一：角域交会、直径与最小包围圆

半平面交求定位区域，顶点对枚举直径，Welzl 算法求最小包围圆，并输出直径圆覆盖判定与清除半径判据。

Measurement：一次示向度观测：测点坐标、示向度与误差半角（题设为 1° ）。

```

1 class Measurement:
2
3     x: float
4     y: float
5     bearing_deg: float
6     half_angle_deg: float = 1.0

```

halfplanes：一次观测的三个半平面约束：角域两侧边界与前向约束，法向量已单位化。

```

1 def halfplanes(m: Measurement) -> list[tuple[float, float, float]]:
2     """Return a*x+b*y<=c with unit normals, including the forward..."""
3     theta = m.bearing_deg % 360
4     lower, upper = radians(theta - m.half_angle_deg), radians(theta + m.half_angle_deg)
5     middle = radians(theta)
6     normals = [(sin(lower), -cos(lower)), (-sin(upper), cos(upper)),
7               (-cos(middle), -sin(middle))]
8     return [(a, b, a * m.x + b * m.y) for a, b in normals]

```

convex_hull：单调链凸包（逆时针、无重复末点），输入坐标先平移缩放到同一量级。

```

1 def convex_hull(points: list[Point], eps: float = EPS) -> list[Point]:
2     """Monotone chain, CCW, no repeated final vertex; retain segm..."""
3     ordered = sorted(points)
4     unique = []
5     for p in ordered:
6         if not unique or distance(p, unique[-1]) > eps:
7             unique.append(p)
8     if len(unique) <= 2:
9         return unique
10
11     def chain(seq):
12         out = []
13         for p in seq:
14             while len(out) >= 2:
15                 a = (out[-1][0] - out[-2][0], out[-1][1] - out[-2][1])
16                 b = (p[0] - out[-1][0], p[1] - out[-1][1])
17                 # Distance-to-line tolerance, not an unscaled area tolerance.
18                 if cross(a, b) > eps * max(hypot(*a), hypot(*b), eps):
19                     break
20                 out.pop()
21             out.append(p)
22         return out
23
24     return chain(unique)[-1] + chain(reversed(unique))[:-1]

```

diameter：顶点对枚举区域直径；有界凸集的最远点必为顶点。

```

1 def diameter(points: list[Point]):
2     if not points:
3         raise ValueError("空点集没有本程序定义下的直径")
4     a = b = points[0]
5     value = 0.0
6     for p, q in combinations(points, 2):
7         d = distance(p, q)
8         if d > value:
9             value, a, b = d, p, q
10    return value, (a, b)

```

smallest_enclosing_circle：Welzl 随机增量最小包围圆；固定随机种子保证结果可复现。

```

1 def smallest_enclosing_circle(points: list[Point], eps: float = EPS):

```

```

2     """Welzl's randomized incremental algorithm, using an explici..."""
3     if not points:
4         raise ValueError("空集不计算最小包围圆")
5     if not isfinite(eps) or eps <= 0:
6         raise ValueError("eps必须为有限正数")
7     if any(len(p) != 2 or any(isinstance(x, bool) or not isinstance(x, (int, float))
8         or not isfinite(x) for x in p) for p in points):
9         raise ValueError("点集必须由有限二维坐标组成")
10    origin = points[0]
11    scale = max(distance(origin, p) for p in points)
12    if scale == 0:
13        return tuple(origin), 0.0
14    local = [(p[0]-origin[0])/scale, (p[1]-origin[1])/scale) for p in points]
15    tolerance = min(eps/scale, EPS)
16    Random(20260910).shuffle(local)
17    # Frame = (remaining prefix length, forced boundary, returning from child).
18    stack = [(len(local), (), False)]
19    circle = None
20    while stack:
21        n, boundary, returning = stack.pop()
22        if n == 0 or len(boundary) == 3:
23            # Forced support points must lie ON the boundary. For three points
24            # use their circumcircle, even when their unconstrained MEC is smaller.
25            circle = _circumcircle(*boundary) if len(boundary) == 3 else None
26            if circle is None and boundary:
27                circle = _enclosing_circle_reference(list(boundary), tolerance)
28            elif not returning:
29                stack.append((n, boundary, True))
30                stack.append((n-1, boundary, False))
31            elif not _circle_covers(circle, local[n-1], tolerance):
32                stack.append((n-1, boundary + (local[n-1],), False))
33    assert circle is not None
34    center = (origin[0]+circle[0][0]*scale, origin[1]+circle[0][1]*scale)
35    # Recompute after transforming back: returned circle contains all input points
36    # to distance() precision, including any roundoff from the large translation.
37    return center, max(distance(center, p) for p in points)

```

solve: 问题一主入口: 半平面交求定位区域, 区分空集、无界与退化情形, 再给出直径、直径圆覆盖判定与最小包围圆。

```

1 def solve(measurements: list[Measurement], eps: float = EPS) -> dict:
2     """Solve Q1's pure angular intersection (no disk/range prior...)"""
3     if not isfinite(eps) or eps <= 0:
4         raise ValueError("eps必须为有限正数")
5     if not measurements:
6         return {"status": "unbounded", "vertices": [], "diameter_m": None,
7             "diameter_infinite": True, "recession_direction": [1.0, 0.0],
8             "message": "没有观测, 整个平面均可行"}
9     if len(measurements) > 100:
10        raise ValueError("参考实现最多接收100个检测点; 更大规模应使用半平面交算法")
11    ox = sum(m.x for m in measurements) / len(measurements)
12    oy = sum(m.y for m in measurements) / len(measurements)
13    scale = max(1.0, max(hypot(m.x - ox, m.y - oy) for m in measurements))
14    local = [Measurement((m.x - ox) / scale, (m.y - oy) / scale,
15        m.bearing_deg, m.half_angle_deg) for m in measurements]
16    planes = [h for m in local for h in halfplanes(m)]
17
18    def inside(p):
19        tol = eps * max(1.0, abs(p[0]), abs(p[1]))
20        return all(a * p[0] + b * p[1] <= c + tol for a, b, c in planes)
21
22    feasible_vertices = []
23    for (a, b, c), (d, e, f) in combinations(planes, 2):
24        det = a * e - b * d
25        if det == 0.0:
26            continue # 精确平行: 无交点 (近平行对的交点是合法远端顶点, 保留)
27        p = ((c * e - b * f) / det, (a * f - c * d) / det)
28        if isfinite(p[0]) and isfinite(p[1]) and inside(p):
29            feasible_vertices.append(p)
30
31    if not feasible_vertices:
32        return {"status": "empty", "vertices": [], "diameter_m": None,
33            "diameter_infinite": False, "tolerance_m": eps * scale,
34            "message": "测向角域没有公共点; 检查频道、数据及误差界"}
35
36    # A nontrivial 2D recession cone has a ray tangent to an input halfplane.
37    # Use a much tighter angular threshold than the positional feasibility epsilon.

```

```

38 ray = None
39 for a, b, _ in planes:
40     for direction in ((b, -a), (-b, a)):
41         if all(d * direction[0] + e * direction[1] <= 2e-14
42             for d, e, _ in planes):
43             ray = direction
44             break
45     if ray is not None:
46         break
47
48 hull = convex_hull(feasible_vertices, eps)
49
50 def world(p):
51     return [ox + p[0] * scale, oy + p[1] * scale]
52
53 result = {"status": "unbounded" if ray else
54     ("point" if len(hull) == 1 else "segment" if len(hull) == 2 else "polygon"),
55     "vertices": [world(p) for p in hull], "tolerance_m": eps * scale,
56     "diameter_infinite": ray is not None}
57 if ray is not None:
58     result.update(diameter_m=None, recession_direction=list(ray),
59         message="存在非零衰退方向，区域无界，直径为无穷大")
60     return result
61
62 d, pair = diameter(hull)
63 midpoint = ((pair[0][0] + pair[1][0]) / 2, (pair[0][1] + pair[1][1]) / 2)
64 distances = [distance(midpoint, p) for p in hull]
65 witness_index = max(range(len(hull)), key=distances.__getitem__)
66 max_distance = distances[witness_index]
67 center, radius = smallest_enclosing_circle(hull, eps)
68 area = abs(sum(cross(hull[i], hull[(i + 1) % len(hull)]))
69     for i in range(len(hull)))) / 2
70 result.update(diameter_m=d * scale, diameter_pair=[world(p) for p in pair],
71     area_m2=area * scale ** 2,
72     diameter_circle={"center": world(midpoint), "radius_m": d * scale / 2,
73         "max_vertex_distance_m": max_distance * scale,
74         "covers": max_distance <= d / 2 + eps * 4,
75         "coverage_tolerance_m": 4 * eps * scale,
76         "max_excess_m": (max_distance-d/2) * scale,
77         "outside_vertex_indices": [i for i, value in enumerate(distances)
78             if value > d/2 + 4*eps],
79         "witness_vertex": world(hull[witness_index])
80             if max_distance > d/2 + 4*eps else None},
81     minimum_enclosing_circle={"center": world(center), "radius_m": radius * scale},
82     guaranteed_clearable_at_mec_center=radius * scale <= 20 - eps * scale * 4,
83     clearance_certificate={"model": "angular_intersection_only",
84         "clear_radius_m": 20.0,
85         "numerical_margin_m": 4*eps*scale,
86         "radius_with_margin_m": (radius+4*eps)*scale,
87         "remaining_margin_m": 20-(radius+4*eps)*scale,
88         "center": world(center)})
89     return result

```

问题二：候选集与保证接收的第二检测点

候选集用圆盘外切多边形与角域半平面裁剪表示；安全盘半径取 $1000 - \rho$ ，盘内候选按最坏后验包围半径、同分按移动代价取优。

CandidateRegion：某频道候选集合的保守凸外包多边形（顶点按 CCW；允许退化为 1-2 点）。

```

1 class CandidateRegion:
2     """某频道候选集合的保守凸外包多边形（顶点按 CCW；允许退化为 1-2 点）。"""
3
4     @classmethod
5     def disk(cls, center: Vec2, radius_m: float, sides: int,
6         slack_m: float | None = None) -> "CandidateRegion":
7         if slack_m is None:
8             slack_m = radius_m * (1 - math.cos(math.pi / sides))
9         return cls(circumscribed_polygon(center, radius_m, sides), slack_m)
10
11     @classmethod
12     def initial(cls, station: Vec2, config: ProblemConfig) -> "CandidateRegion":
13         """首个 direction 后的基础集:  $\Omega$  B(station, r_max) 的保守外包。"""
14         field_cfg, sides = config.field, config.search.polygon_sides
15         slack = field_cfg.omega_radius_m * (1 - math.cos(math.pi / sides))
16         region = cls.disk((0.0, 0.0), field_cfg.omega_radius_m, sides, slack)

```

```

17     region.clip_disk(station, field_cfg.r_eff_max_m, sides)
18     return region
19
20 def clip_disk(self, center: Vec2, radius_m: float, sides: int) -> None:
21     """与圆盘 B(center, radius) 的外切多边形求交（近似误差入账）。"""
22     for a, b, c in polygon_halfplanes(circumscribed_polygon(center, radius_m, sides)):
23         self._clip(a, b, c)
24     self.slack_m += radius_m * (1 - math.cos(math.pi / sides))
25
26 def clip_wedge(self, station: Vec2, bearing_deg: float,
27               half_angle_deg: float) -> None:
28     """与测向角域 W(station, bearing±half_angle) 求交（半角须 <90°）。
29
30     两个边界半平面与 localization.halfplanes 同口径，0°/360° 跨界由
31     sin/cos 直接处理，无需分支。
32     """
33     if not 0 < half_angle_deg < 90:
34         raise ValueError("角域半角须在 (0, 90) 度")
35     lower = math.radians(bearing_deg - half_angle_deg)
36     upper = math.radians(bearing_deg + half_angle_deg)
37     sx, sy = float(station[0]), float(station[1])
38     self._clip(math.sin(lower), -math.cos(lower),
39               math.sin(lower) * sx - math.cos(lower) * sy)
40     self._clip(-math.sin(upper), math.cos(upper),
41               -math.sin(upper) * sx + math.cos(upper) * sy)
42
43 def mec(self) -> tuple[Vec2, float]:
44     """最小包围圆（圆心，半径）。半径是真实候选集 的上界。"""
45     if self._mec is None:
46         if not self.polygon:
47             raise InconsistentBelief("候选区域为空")
48         self._mec = smallest_enclosing_circle(self.polygon)
49     return self._mec
50
51 def witnesses(self, spacing_m: float) -> list[Vec2]:
52     """顶点 + 边界按间距细分的采样点（最坏后验评估用）。"""
53     if len(self.polygon) < 3:
54         return list(self.polygon)
55     points = list(self.polygon)
56     count = len(self.polygon)
57     for index in range(count):
58         p, q = self.polygon[index], self.polygon[(index + 1) % count]
59         length = math.dist(p, q)
60         steps = int(length // spacing_m)
61         for k in range(1, steps + 1):
62             ratio = k / (steps + 1)
63             points.append((p[0] + ratio * (q[0] - p[0]),
64                           p[1] + ratio * (q[1] - p[1])))
65     return _dedupe(points, tol=1e-9)
66
67 def contains(self, point: Vec2, tol_m: float = 0.05) -> bool:
68     """点是否在候选集内（带容差；用于真源保留性检验，不做删点依据）。"""
69     if not self.polygon:
70         return False
71     if len(self.polygon) == 1:
72         return math.dist(point, self.polygon[0]) <= tol_m + self.slack_m
73     if len(self.polygon) == 2:
74         p, q = self.polygon
75         return _point_segment_distance(point, p, q) <= tol_m + self.slack_m
76     for index, p in enumerate(self.polygon):
77         q = self.polygon[(index + 1) % len(self.polygon)]
78         if (q[0] - p[0]) * (point[1] - p[1]) - (q[1] - p[1]) * (point[0] - p[0])
79             < -tol_m - self.slack_m:
80             return False
81     return True

```

ClearCertificate: 清除充分条件：候选集外包半径与近似误差之和不超过清除半径。

```

1 class ClearCertificate:
2     """清除证书：候选集外包半径（含近似误差）不超过 20m 阈值。"""
3
4     channel: int
5     center: Vec2
6     radius_m: float
7     limit_m: float
8     certified: bool

```

clear_certificate: 由候选集最小包围圆判定是否可以就地清除。

```
1 def clear_certificate(region: CandidateRegion, channel: int,
2                       config: ProblemConfig = ProblemConfig()) -> ClearCertificate:
3     """_poly <= 20 - 外包近似误差 - 工程余量 才允许发起清除。"""
4     center, radius = region.mec()
5     limit = (config.field.clear_radius_m - region.slack_m
6             - config.search.clear_margin_m)
7     return ClearCertificate(channel, center, radius, limit, radius <= limit)
```

candidate_points: 保证接收盘内的候选点: 盘心加各比例半径上的均匀方向点。

```
1 def candidate_points(center: Vec2, safe_radius_m: float, rings, angles: int) -> list[Vec2]:
2     """保证接收盘内的候选点: 盘心 + 各比例半径 × 均匀方向。"""
3     points = [tuple(center)]
4     for fraction in rings:
5         radius = fraction * safe_radius_m
6         if radius <= 1e-9:
7             continue
8         step = 2 * math.pi / angles
9         for k in range(angles):
10            angle = step * k + step / 2
11            points.append((center[0] + radius * math.cos(angle),
12                          center[1] + radius * math.sin(angle)))
13     return points
```

safe_disk_radius: 保证接收盘半径 $1000 - \rho$; 小于零表示没有保证接收的区域。

```
1 def safe_disk_radius(region: CandidateRegion,
2                      config: ProblemConfig = ProblemConfig()) -> float:
3     """保证接收盘 B(mec, safe) 的半径; <0 表示无保证 ( 超过最小接收半径)。"""
4     return config.field.r_eff_min_m - region.mec()[1] - 1e-3
```

select_measurement_point: 在保证接收盘内选下一检测点: 主指标为最坏后验包围半径, 同分取移动代价较小者。

```
1 def select_measurement_point(region: CandidateRegion, from_position: Vec2,
2                             config: ProblemConfig = ProblemConfig()) -> SecondPointPlan:
3     """在保证接收盘内选择下一检测点 (P3 核心; 对任意候选集通用)。"""
4     field, search = config.field, config.search
5     center, radius = region.mec()
6     half_angle = field.bearing_error_half_deg + search.bearing_rounding_margin_deg
7
8     def move_cost(point: Vec2) -> float:
9         return move_time(from_position, point, config.cost)
10
11     # 1mm 数值余量在 safe_disk_radius 内记账。
12     safe_radius = safe_disk_radius(region, config)
13     if safe_radius < 0:
14         return SecondPointPlan(tuple(center), radius, move_cost(center),
15                                "无保证接收盘 (>1000): 探索 MEC 圆心, 可能 no_signal")
16     coarse = region.witnesses(search.witness_spacing_m)
17     samples = candidate_points(center, safe_radius,
18                               search.candidate_rings, search.candidate_angles)
19     near_radius, sides = field.near_radius_m, search.polygon_sides
20     best_key, best_point, best_worst = None, None, None
21     for point in samples:
22         worst = _worst_posterior_radius(region, point, coarse, half_angle,
23                                         near_radius, sides)
24         key = (round(worst, 6), round(move_cost(point), 6))
25         if best_key is None or key < best_key:
26             best_key, best_point, best_worst = key, point, worst
27     dense = _worst_posterior_radius(region, best_point,
28                                     region.witnesses(search.witness_spacing_m / 4),
29                                     half_angle, near_radius, sides)
30     reason = (f"安全盘 B(centre,{safe_radius:.0f}m) 内 {len(samples)} 候选; "
31              f"粗采样最坏后验 {best_worst:.1f}m, 1/4 间距加密复核 {dense:.1f}m")
32     return SecondPointPlan(tuple(best_point), dense, move_cost(best_point), reason)
```

select_second_point: 问题二入口: 由首次示向度构造候选集并给出第二检测点。

```
1 def select_second_point(client: SimulatorClient, first_position: Vec2,
2                         first_bearing_deg: float,
3                         config: ProblemConfig = ProblemConfig()) -> SecondPointPlan:
4     """P3 入口: 由首次 direction 构造 K1, 返回保证接收的第二检测点方案。"""
5     region = CandidateRegion.initial(first_position, config)
6     half_angle = config.field.bearing_error_half_deg
```

```

7         + config.search.bearing_rounding_margin_deg
8     region.clip_wedge(first_position, first_bearing_deg, half_angle)
9     return select_measurement_point(region, client.state.position, config)

```

问题三：覆盖计划、检测记录与清除证书

覆盖计划与运行期抽样校验、逐频道检测记录，以及七点解析覆盖与全向 / 定向两种覆盖语义的证书判据。

CoverageCertificate: 覆盖证书：采样分辨率、位置与朝向目标数、缺口数与最坏命中距离。

```

1 class CoverageCertificate:
2     """覆盖证书：采样语义 + 缺口 + 最坏命中距离。"""
3
4     kind: str                # "full_disk" | "forward_half_disk"
5     target_step_m: float
6     headings: int
7     r_eff_min_m: float
8     targets: int
9     uncovered: int
10    worst_hit_distance_m: float
11    nn_tour_m: float
12    generator_sha256: str = ""
13    verify_heading_margin_deg: float = 0.0
14    @property
15    def valid(self) -> bool:
16        return self.uncovered == 0 and self.targets > 0
17
18    def as_dict(self) -> dict:
19        return {"kind": self.kind, "target_step_m": self.target_step_m,
20                "headings": self.headings, "r_eff_min_m": self.r_eff_min_m,
21                "targets": self.targets, "uncovered": self.uncovered,
22                "worst_hit_distance_m": round(self.worst_hit_distance_m, 3),
23                "nn_tour_m": round(self.nn_tour_m, 1),
24                "nn_tour_virtual_s": round(self.nn_tour_m / SPEED_MPS, 1),
25                "generator_sha256": self.generator_sha256,
26                "verify_heading_margin_deg": self.verify_heading_margin_deg}

```

CoveragePlan: 覆盖计划：点集、覆盖语义与证书。

```

1 class CoveragePlan:
2     """一份覆盖计划：点集 + 语义 + 证书（可为解析证明）。"""
3
4     name: str
5     problem: int
6     points: tuple[Vec2, ...]
7     kind: str                # "analytic" | "certified"
8     coverage: str            # "full_disk" | "forward_half_disk"
9     note: str = ""
10    certificate: CoverageCertificate | None = None
11    generator: dict = field(default_factory=dict)
12    def as_dict(self) -> dict:
13        return {"name": self.name, "problem": self.problem, "kind": self.kind,
14                "coverage": self.coverage, "note": self.note,
15                "generator": self.generator,
16                "certificate": (self.certificate.as_dict()
17                                if self.certificate is not None else None),
18                "points": [[p[0], p[1]] for p in self.points]}

```

coverage_points: 49 点覆盖网格，单元对角 $700\sqrt{2} \approx 989.95 \leq 1000$ m。

```

1 def coverage_points(config: ProblemConfig = ProblemConfig()) -> list[Vec2]:
2     """49 点覆盖网格生成（纯几何，可独立测试）。"""
3     step = config.search.grid_step_m
4     half = config.search.grid_half_cells
5     return [(step * i, step * j)
6             for j in range(half, -half - 1, -1)
7             for i in range(-half, half + 1)]

```

CoverageLedger: 逐频道检测记录，按（频道, 网格点）二维记账。

```

1 class CoverageLedger:
2     """逐频道覆盖账本：（频道，网格点）-> 是否已测。"""
3
4     config: ProblemConfig = field(default_factory=ProblemConfig)
5     points: list[Vec2] = field(default_factory=coverage_points)

```

```

6     measured: set = field(default_factory=set)          # {(channel, point_index)}
7     skipped: dict = field(default_factory=dict)        # channel -> 原因
8     def mark_measured(self, channel: int, point_index: int) -> None:
9         self.measured.add((channel, point_index))
10
11     def measured_at(self, channel: int, point_index: int) -> bool:
12         return (channel, point_index) in self.measured
13
14     def skip_channel(self, channel: int, reason: str) -> None:
15         """记录整频道跳过原因 (如 already_cleared / located_pending_clear) 。"""
16         self.skipped[channel] = reason
17
18     def pending_points(self, channel: int) -> list[int]:
19         """该频道尚未测量的网格点下标。扫描调度策略未实现 (P4) 。"""
20         return [index for index in range(len(self.points))
21                 if not self.measured_at(channel, index)]
22
23     def coverage_complete(self, channel: int) -> bool:
24         """该频道覆盖是否完成 (含显式跳过) 。"""
25         return channel in self.skipped or not self.pending_points(channel)

```

seven_points: 原点与环上六个等分点的七站覆盖集。

```

1 def seven_points(ring_radius_m: float = RING_RADIUS_M) -> list[tuple[float, float]]:
2     """原点 + 环上 6 等分点的七站覆盖集。"""
3     return [(0.0, 0.0)] + [(ring_radius_m * math.cos(k * math.pi / 3),
4                             ring_radius_m * math.sin(k * math.pi / 3)) for k in range(6)]

```

seven_plan: 第三问解析覆盖计划; 环半径 1159.7 m 时最坏覆盖距离不超过 984.539 m。

```

1 def seven_plan(ring_radius_m: float) -> CoveragePlan:
2     """第三问解析计划 (覆盖证明由 validate_seven_coverage 给出) 。"""
3     return CoveragePlan(
4         name=f"seven-r{ring_radius_m:.1f}", problem=3, kind="analytic",
5         coverage="full_disk",
6         note=("解析证明  $\max(r/\sqrt{3}, \sqrt{R^2+r^2-\sqrt{3}Rr}) < 1000$ ; "
7              "r [1122.9, 1732] 时成立、巡检半径越小巡回越短"),
8         points=tuple(seven_points(ring_radius_m)))

```

verify_full_disk: 第三问覆盖语义: 圆域内每点都落在某个被访点的最小接收圆盘内。

```

1 def verify_full_disk(points: list[Vec2], step_m: float = VERIFY_STEP_M,
2                       r_eff: float = R_EFF_MIN_M) -> CoverageCertificate:
3     """第三问语义:  $\Omega$  内每点都在某个被访点的  $r_{\text{eff}}$  圆盘内。"""
4     points = list(points)
5     positions = omega_positions(step_m)
6     table = _bucket(points, r_eff)
7     missing, worst = 0, 0.0
8     for (px, py) in positions:
9         ci, cj = int(px // r_eff), int(py // r_eff)
10        best = None
11        for di in (-1, 0, 1):
12            for dj in (-1, 0, 1):
13                for c_index in table.get((ci + di, cj + dj), ()):
14                    cx, cy = points[c_index]
15                    distance = math.hypot(cx - px, cy - py)
16                    if distance <= r_eff and (best is None or distance < best):
17                        best = distance
18        if best is None:
19            missing += 1
20        else:
21            worst = max(worst, best)
22    return CoverageCertificate("full_disk", step_m, 1, r_eff, len(positions),
23                              missing, worst, nn_tour_length(points))

```

verify_forward_half_disk: 第四问覆盖语义: 任意位置与朝向都存在前向半盘内的被访点。

```

1 def verify_forward_half_disk(points: list[Vec2], step_m: float = VERIFY_STEP_M,
2                               headings: int = VERIFY_HEADINGS,
3                               r_eff: float = R_EFF_MIN_M,
4                               heading_margin_deg: float = 0.0) -> CoverageCertificate:
5     """第四问语义: (p,h) 存在前向半盘内的被访点。"""
6     points = list(points)
7     targets = forward_targets(step_m, headings)
8     table = _bucket(points, r_eff)
9     missing, worst = 0, 0.0
10    sin_margin = math.sin(math.radians(max(heading_margin_deg, 0.0)))

```

```

11     for (px, py, heading) in targets:
12         ux, uy = math.cos(heading), math.sin(heading)
13         ci, cj = int(px // r_eff), int(py // r_eff)
14         best = None
15         for di in (-1, 0, 1):
16             for dj in (-1, 0, 1):
17                 for c_index in table.get((ci + di, cj + dj), ()):
18                     cx, cy = points[c_index]
19                     dx, dy = cx - px, cy - py
20                     distance_sq = dx * dx + dy * dy
21                     if distance_sq > r_eff * r_eff:
22                         continue
23                     if dx * ux + dy * uy < sin_margin * math.sqrt(distance_sq) - 1e-9:
24                         continue
25                     distance = math.sqrt(distance_sq)
26                     if best is None or distance < best:
27                         best = distance
28         if best is None:
29             missing += 1
30         else:
31             worst = max(worst, best)
32     return CoverageCertificate("forward_half_disk", step_m, headings, r_eff,
33                               len(targets), missing, worst,
34                               nn_tour_length(points))

```

plan_instance: 按名称取覆盖计划；证书计划在运行期按同一分辨率复核。

```

1 def plan_instance(name: str, verify: bool = True) -> CoveragePlan:
2     """统一入口：按名称取覆盖计划（内置解析计划 / 内嵌证书计划）。"""
3     if name == "square49":
4         return square_49_plan()
5     if name == "seven":
6         return seven_plan(1200.0)
7     if name not in EMBEDDED_PLANS:
8         raise FileNotFoundError(f"未找到覆盖计划 {name!r}；现有：{plan_names()}")
9     plan = embedded_plan(name)
10    if verify and plan.kind == "certified":
11        certificate = verify_plan(plan)
12        if not certificate.valid:
13            raise ValueError(
14                f"覆盖计划 {name!r} 运行期校验失败： "
15                f"{certificate.uncovered}/{certificate.targets} 个 (p,h) 未被覆盖")
16    plan = CoveragePlan(
17        name=plan.name, problem=plan.problem, kind=plan.kind,
18        coverage=plan.coverage, note=plan.note, generator=plan.generator,
19        certificate=CoverageCertificate(
20            kind=certificate.kind, target_step_m=certificate.target_step_m,
21            headings=certificate.headings, r_eff_min=certificate.r_eff_min_m,
22            targets=certificate.targets, uncovered=certificate.uncovered,
23            worst_hit_distance_m=certificate.worst_hit_distance_m,
24            nn_tour_m=certificate.nn_tour_m,
25            generator_sha256=(plan.certificate.generator_sha256
26                             if plan.certificate else ""),
27            points=plan.points)
28    return plan

```

问题三：网格候选集

逐频道的网格幸存集，按格心裕量 $\ell/\sqrt{2}$ 内缩后再删格，据此支持负观测更新、跳测规则与缺席证明。

ChannelParticles: 单个频道的网格候选集：格心裕量 $\ell/\sqrt{2}$ ，所有删除判定内缩该裕量，故真源所在格永不被删。

```

1 class ChannelParticles:
2     """单个频道的网格幸存集。"""
3
4     def cell_center(self, index: int) -> Vec2:
5         i, j = divmod(index, self.width)
6         return (self.x0 + (i + 0.5) * self.spacing_m,
7                 self.x0 + (j + 0.5) * self.spacing_m)
8
9     def discard(self, indices: list[int]) -> None:
10        """公开删除入口：联合信念把"朝向不可能"的格一并删除时使用。"""
11        self._discard(indices)
12

```

```

13 def _discard(self, indices: list[int]) -> None:
14     if not indices:
15         return
16     coarse_w = self._coarse_width()
17     for index in indices:
18         if index not in self.alive:
19             continue
20         self.alive.discard(index)
21         key = (index // self.width // _COARSE_FACTOR) * coarse_w
22             + (index % self.width) // _COARSE_FACTOR
23         count = self._coarse.get(key, 0) - 1
24         if count <= 0:
25             self._coarse.pop(key, None)
26         else:
27             self._coarse[key] = count
28     self.version += 1
29     self._mec = None
30
31 def _kill_disk(self, center: Vec2, radius_m: float) -> None:
32     """删除格心距 center <= radius_m 的全部幸存格（整格判据见模块注释）。"""
33     if not self.alive:
34         return
35     reach = radius_m + _DIST_EPS_M
36     ilo = max(0, int(math.floor((center[0] - reach - self.x0) / self.spacing_m)))
37     ihi = min(self.width - 1,
38              int(math.ceil((center[0] + reach - self.x0) / self.spacing_m)))
39     jlo = max(0, int(math.floor((center[1] - reach - self.x0) / self.spacing_m)))
40     jhi = min(self.width - 1,
41              int(math.ceil((center[1] + reach - self.x0) / self.spacing_m)))
42     limit_sq = reach * reach
43     doomed = []
44     for i in range(ilo, ihi + 1):
45         dx = self.x0 + (i + 0.5) * self.spacing_m - center[0]
46         base = i * self.width
47         for j in range(jlo, jhi + 1):
48             index = base + j
49             if index not in self.alive:
50                 continue
51             dy = self.x0 + (j + 0.5) * self.spacing_m - center[1]
52             if dx * dx + dy * dy <= limit_sq:
53                 doomed.append(index)
54     self._discard(doomed)
55
56 def _kill_outside_disk(self, center: Vec2, radius_m: float) -> None:
57     """删除格心距 center > radius_m 的全部幸存格（保留盘内）。"""
58     if not self.alive:
59         return
60     reach = radius_m + _DIST_EPS_M
61     doomed = [index for index in self.alive
62               if math.dist(self.cell_center(index), center) > reach]
63     self._discard(doomed)
64
65 def _kill_outside_wedge(self, station: Vec2, reading_deg: float) -> None:
66     """删除角向偏离读数超出 _eff + 角向格裕量的幸存格。
67
68     角向格裕量 = asin(m / max(dist-m, 1)): 格内任一点相对格心的方位
69     偏差不超过它（点距测点还可再近 m, 故分母再内缩 m）。
70     """
71     if not self.alive:
72         return
73     sx, sy = float(station[0]), float(station[1])
74     doomed = []
75     for index in self.alive:
76         gx, gy = self.cell_center(index)
77         dist = math.hypot(gx - sx, gy - sy)
78         if dist <= self.margin_m + 1.0:
79             continue # 格贴测点: 格内方位可指向任意方向, 不可删
80         angular_margin = math.degrees(
81             math.asin(min(0.9999, self.margin_m / (dist - self.margin_m))))
82         gap = angular_gap(math.degrees(math.atan2(gy - sy, gx - sx)), reading_deg)
83         if gap > self.half_angle_deg + angular_margin + _ANG_EPS_DEG:
84             doomed.append(index)
85     self._discard(doomed)
86
87 def apply_measure(self, station: Vec2, result: str,
88                  svd_deg: float | None = None, *,
89                  directional: bool = False) -> None:

```

```

90     """按读数更新幸存集。真源保留性由 Voronoi 裕量保证（模块注释）。
91
92     ``directional=True``（第四问）：no_signal **不删空间候选**——定向源
93     的静默还可能来自朝向遮挡，单独删空间候选会误删真源：朝向维度由
94     :class:`~mcm26b.belief.joint.JointBelief` 维护。
95     """
96     if result == NO_SIGNAL:
97         if directional:
98             return
99         # no_signal dist > r_eff >= 1000: 整格落入 B(s, 1000-m) 的删。
100         self._kill_disk(station, self.r_eff_min_m - self.margin_m)
101     elif result == NEAR:
102         self._kill_outside_disk(station, self.near_radius_m + self.margin_m)
103     elif result == DIRECTION:
104         if svd_deg is None:
105             raise ValueError("direction 读数必须携带 svd_deg")
106         # direction 隐含 dist <= r_max [题面]
107         self._kill_outside_disk(station, self.r_eff_max_m + self.margin_m)
108         self._kill_outside_wedge(station, svd_deg)
109     else:
110         raise ValueError(f"未知测向结果: {result!r}")
111
112     def alive_points(self, limit: int | None = None) -> list[Vec2]:
113         sorted_alive = sorted(self.alive)
114         if limit is not None and len(sorted_alive) > limit:
115             step = len(sorted_alive) / limit
116             indices = {sorted_alive[int(k * step) % len(sorted_alive)]
117                        for k in range(limit)}
118             indices = sorted(indices)
119         else:
120             indices = sorted_alive
121         return [self.cell_center(index) for index in indices]
122
123     def nearest_alive_distance(self, point: Vec2, limit_m: float) -> float | None:
124         """最近幸存格心距离；> limit_m 时返回 None（粗格环扫 + 下界剪枝）。
125
126         返回值语义：恰为最近幸存格心距离（若 <= limit_m）；None 表示所有
127         幸存格心都超出 limit_m。粗格聚合半径已计入扫描范围，不漏判。
128         """
129         if not self.alive:
130             return None
131         coarse_cell_m = self.spacing_m * _COARSE_FACTOR
132         coarse_reach = coarse_cell_m * math.sqrt(2) / 2
133         cw = self._coarse_width()
134         ci_base = int(math.floor((point[0] - self.x0) / coarse_cell_m))
135         cj_base = int(math.floor((point[1] - self.x0) / coarse_cell_m))
136         ring_max = int(math.ceil((limit_m + coarse_reach) / coarse_cell_m)) + 1
137         best: float | None = None
138         for ring in range(ring_max + 1):
139             lower_bound = max(0.0, ring - 1) * coarse_cell_m - coarse_reach
140             if best is not None and lower_bound >= best:
141                 break # 更外环的所有格心都 >= 下界 >= 当前最优
142             for ci in range(max(0, ci_base - ring), min(cw - 1, ci_base + ring) + 1):
143                 for cj in range(max(0, cj_base - ring), min(cw - 1, cj_base + ring) + 1):
144                     if max(abs(ci - ci_base), abs(cj - cj_base)) != ring:
145                         continue # 只扫当前环（Chebyshev 环）
146                     if (ci * cw + cj) not in self._coarse:
147                         continue
148                     ccx = self.x0 + (ci + 0.5) * coarse_cell_m
149                     ccy = self.x0 + (cj + 0.5) * coarse_cell_m
150                     if math.hypot(ccx - point[0], ccy - point[1])
151                         > limit_m + coarse_reach:
152                         continue
153                     i0, j0 = ci * _COARSE_FACTOR, cj * _COARSE_FACTOR
154                     for i in range(i0, min(i0 + _COARSE_FACTOR, self.width)):
155                         base = i * self.width
156                         for j in range(j0, min(j0 + _COARSE_FACTOR, self.width)):
157                             index = base + j
158                             if index not in self.alive:
159                                 continue
160                             dist = math.dist(self.cell_center(index), point)
161                             if dist <= limit_m and (best is None or dist < best):
162                                 best = dist
163         return best
164
165     def guaranteed_no_signal(self, station: Vec2) -> bool:
166         """真源与该测点距离必 > r_max (>= r_eff) 必 no_signal, 测了也无信息。"""

```

```

167         return self.nearest_alive_distance(
168             station, self.r_eff_max_m + self.margin_m) is None
169
170     def mec(self) -> tuple[Vec2, float]:
171         """幸存集最小包围圆（圆心，半径）；按 version 缓存。"""
172         if self._mec is None:
173             if not self.alive:
174                 raise ValueError("幸存集为空（该频道已可证明缺席）")
175             points = [self.cell_center(index) for index in self.alive]
176             self._mec = smallest_enclosing_circle(points)
177         return self._mec
178
179     def subsample(self, size: int) -> list[Vec2]:
180         """按索引序均匀抽取的子采样（索引序 空间序，代表点空间铺开）。"""
181         return self.alive_points(limit=size)
182
183     def expected_survival(self, point: Vec2, subsample,
184                          representatives) -> float:
185         """E[log 幸存占比]：越小该测点的期望信息增益越大。
186
187         r_eff 不确定性解析积分：dist (1000,1500) 时按 r_eff~U(1000,1500)
188         取 p_ns=(dist-1000)/500 为 no_signal 分支概率。
189         - no_signal 分支：删 B(point, 1000-m)，幸存 = 子采样中盘外比例；
190         - direction 分支：以代表点方位为期望读数，角域 B(point,1500+m) 内存存；
191         - near 分支 (dist <= 5+m)：可行集收缩到 5m 盘，幸存占比趋近 0。
192         """
193         if not self.alive or len(subsample) == 0 or len(representatives) == 0:
194             return 0.0
195         if _np is not None and isinstance(subsample, _np.ndarray)
196             and isinstance(representatives, _np.ndarray):
197             return self._expected_survival_vector(point, subsample, representatives)
198         eps = 1.0 / len(subsample)
199         keep_ns = max(1.0 - self.fraction_within(subsample, point,
200                                                    self.r_eff_min_m - self.margin_m),
201                      eps)
202         log_keep_ns = math.log(keep_ns)
203         total = 0.0
204         for rep in representatives:
205             dist = math.hypot(rep[0] - point[0], rep[1] - point[1])
206             if dist <= self.near_radius_m + self.margin_m:
207                 total += math.log(eps)
208                 continue
209             p_ns = min(1.0, max(0.0, (dist - self.r_eff_min_m)
210                               / (self.r_eff_max_m - self.r_eff_min_m)))
211             reading = math.atan2(rep[1] - point[1], rep[0] - point[0])
212             survivors = 0
213             for p in subsample:
214                 if math.dist(p, point) > self.r_eff_max_m + self.margin_m:
215                     continue
216                 gap = angular_gap(math.degrees(
217                     math.atan2(p[1] - point[1], p[0] - point[0])),
218                     math.degrees(reading))
219                 if gap <= self.half_angle_deg + _ANG_EPS_DEG:
220                     survivors += 1
221             keep_dir = max(survivors / len(subsample), eps)
222             total += p_ns * log_keep_ns + (1.0 - p_ns) * math.log(keep_dir)
223         return total / max(len(representatives), 1)
224
225     def fraction_within(self, samples, point: Vec2, radius_m: float) -> float:
226         if len(samples) == 0:
227             return 0.0
228         reach = radius_m + _DIST_EPS_M
229         if _np is not None and isinstance(samples, _np.ndarray):
230             distance = _np.hypot(samples[:, 0] - point[0], samples[:, 1] - point[1])
231             return float(_np.count_nonzero(distance <= reach)) / len(samples)
232         count = sum(1 for p in samples if math.dist(p, point) <= reach)
233         return count / len(samples)
234
235     def expected_posterior_radius(self, point: Vec2, subsample,
236                                  representatives) -> float:
237         """一次测量后期望后验半径（米）：选点主指标，直指清除证书阈值。
238
239         对每个代表假设（真值候选）按 r_eff~U(1000,1500) 解析积分
240         no_signal/direction 分支，near 分支 (dist <= 5+m) 后验为 5m 盘；
241         各分支半径用幸存子采样的质心最大距离近似（偏差 << 格距，排序用）。
242         与几何版 worst-posterior 同族，但作用于含负观测的非凸幸存集，
243         且以期望代替最坏（代表点覆盖幸存集主轴）。

```

```

244     """
245     if not self.alive or len(subsample) == 0 or len(representatives) == 0:
246         return 0.0
247     if _np is not None and isinstance(subsample, _np.ndarray)
248         and isinstance(representatives, _np.ndarray):
249         return self._expected_posterior_radius_vector(
250             point, subsample, representatives)
251     eps_radius = self.spacing_m / 2.0
252     total = 0.0
253     for rep in representatives:
254         dist = math.hypot(rep[0] - point[0], rep[1] - point[1])
255         if dist <= self.near_radius_m + self.margin_m:
256             total += self.near_radius_m + self.margin_m
257             continue
258         p_ns = min(1.0, max(0.0, (dist - self.r_eff_min_m)
259                             / (self.r_eff_max_m - self.r_eff_min_m)))
260     members_ns = [p for p in subsample
261                   if math.dist(p, point) > self.r_eff_min_m - self.margin_m]
262     reading = math.atan2(rep[1] - point[1], rep[0] - point[0])
263     members_dir = []
264     for p in subsample:
265         if math.dist(p, point) > self.r_eff_max_m + self.margin_m:
266             continue
267         gap = angular_gap(math.degrees(
268             math.atan2(p[1] - point[1], p[0] - point[0])),
269             math.degrees(reading))
270         if gap <= self.half_angle_deg + _ANG_EPS_DEG:
271             members_dir.append(p)
272     r_ns = self._branch_radius(members_ns) if members_ns else eps_radius
273     r_dir = self._branch_radius(members_dir) if members_dir else eps_radius
274     total += p_ns * r_ns + (1.0 - p_ns) * r_dir
275     return total / max(len(representatives), 1)

```

问题三：信息增益选点与滚动任务排序

统一任务表由站点扫频、定位测量与清除三类任务组成，每执行一项即重排；定位点按期望信息增益加移动代价评分。

BayesLoop：问题三闭环：网格候选集的正负观测更新、信息增益选点与滚动任务排序。

```

1 class BayesLoop(OmnidirectionalLoop):
2     """问题3贝叶斯闭环：负观测感知 + 信息增益选点 + 滚动任务排序。"""
3
4     def _build_tasks(self) -> list[Task]:
5         tasks: list[Task] = []
6         for index, point in enumerate(self.ledger.points):
7             pending = self._fused_channels_at(index)
8             if not pending:
9                 continue
10            switches = len(pending) - (1 if self.client.state.channel in pending else 0)
11            tasks.append(Task("station", tuple(point),
12                             self.config.cost.measure_s * len(pending)
13                             + self.config.cost.switch_s * max(switches, 0),
14                             station_index=index, station_channels=tuple(pending)))
15        for channel in self.belief.channels_by_status(ChannelStatus.DETECTED):
16            cert = self._best_certificate(channel)
17            if cert is not None and cert.certified
18                and channel not in self._clear_failed:
19                tasks.append(Task("clear", cert.center,
20                                 self.config.cost.clear_success_s,
21                                 channel=channel))
22            continue
23        if self._defer_to_station(channel):
24            continue # 下一条方位线可由"本就要去的站点"顺路提供
25        point = self._select_point(channel)
26        if point is None:
27            continue
28        tasks.append(Task("locate", point,
29                          self.config.cost.measure_s + self.config.cost.switch_s,
30                          channel=channel))
31    return tasks
32
33    def _execute_station(self, index: int) -> None:
34        """单站批量扫频：只测待测频道（跳测规则已过滤），扫完即返回重排。
35
36        P0-1：本站同时测量"已发现未定证"频道，使第二/第三条方位线在巡回

```

```

37 途中顺路取得；证书一旦成立即就地清除，不额外绕行。
38 """
39 point = self.ledger.points[index]
40 pending = self._fused_channels_at(index)
41 unknown = [c for c, b in self.belief.channels.items()
42             if b.status is ChannelStatus.UNKNOWN]
43 self.skipped_measures += len(unknown) - len(
44     [c for c in pending if self.belief.channels[c].status
45      is ChannelStatus.UNKNOWN])
46 # 当前频道优先，减少换频；同站 near 直接清除不扫描频
47 pending.sort(key=lambda c: (c != self.client.state.channel, c))
48 for channel in pending:
49     if self.stop_reason is not None or self._count_limit_reached()
50        or self._virtual_cap_reached():
51         return
52     detected_before = (self.belief.channels[channel].status
53                        is ChannelStatus.DETECTED)
54     outcome = self.client.measure(point, channel)
55     self.ledger.mark_measured(channel, index)
56     self._record_measure(channel, point, outcome)
57     if outcome.result in (DIRECTION, NEAR):
58         self._may_clear.add(channel)
59     if outcome.result == NEAR:
60         region = self.belief.candidate_set(channel)
61         cert = clear_certificate(region, channel, self.config)
62         if cert.certified:
63             if self._attempt_clear(channel, point, cert):
64                 self._clear_failed.discard(channel)
65             else:
66                 self._clear_failed.add(channel)
67                 self._may_clear.discard(channel)
68         else:
69             self._anomaly("near_without_certificate", channel, position=point)
70         continue
71     if not detected_before:
72         continue
73     if outcome.result == NO_SIGNAL:
74         # 该频道已被发现：保证接收盘内不可能 no_signal，属守恒性异常
75         region = self.belief.candidate_set(channel)
76         safe = (safe_disk_radius(region, self.config)
77                if region is not None else -1.0)
78         if safe >= 0 and math.dist(point, region.mec()[0]) <= safe:
79             self._anomaly("no_signal_in_safe_disk", channel,
80                           position=point)
81         continue
82     cert = self._best_certificate(channel)
83     if cert is not None and cert.certified:
84         if self._attempt_clear(channel, cert.center, cert):
85             self._clear_failed.discard(channel)
86         else:
87             self._clear_failed.add(channel)
88             self._may_clear.discard(channel)
89
90 def _record_measure(self, channel: int, position: Vec2, outcome) -> None:
91     """belief (凸证书) 与 PF (幸存集) 同步更新；PF 空即证缺席。"""
92     self.belief.update_measure(channel, position, outcome.result,
93                               outcome.svd_deg)
94     # 同一 0.5m 测点 × 同一频道的重复检测：历史上一次官方演练有 98.84%
95     # 的虚拟时间耗在空复扫上，本计数是该问题的回归指标。
96     key = _point_key(position)
97     seen = self._measured_points.setdefault(channel, set())
98     if key in seen:
99         self.redundant_rescan_measures += 1
100     seen.add(key)
101     self._clear_failed.discard(channel) # 新证据落地：允许再次尝试清除
102     belief = self.belief.channels[channel]
103     if belief.status in (ChannelStatus.CLEARED, ChannelStatus.ABSENT):
104         return # 已证毕：清除/缺席后的 no_signal 与距离无关，不进 PF
105     pf = self.particles[channel]
106     pf.apply_measure(position, outcome.result, outcome.svd_deg)
107     if not pf:
108         if belief.status is ChannelStatus.UNKNOWN:
109             self.belief.mark_absent(channel)
110             self.absence_evidence[channel] = "pf_empty"
111             if channel not in self.pf_absent_channels:
112                 self.pf_absent_channels.append(channel)
113     else:

```

```

114         self._anomaly("pf_empty_while_detected", channel, position=position)
115     self._apply_count_bound()
116
117     def _best_certificate(self, channel: int) -> ClearCertificate | None:
118         """凸证书优先：负观测挖洞使 PF 幸存集更紧时用网格证书补位。"""
119         region = self.belief.candidate_set(channel)
120         cert = clear_certificate(region, channel, self.config)
121         if region is not None else None
122         if cert is not None and cert.certified:
123             return self._near_clear_target(cert)
124         if self.config.bayes.grid_cert_enabled:
125             pf = self.particles[channel]
126             if len(pf):
127                 center, radius = pf.mec()
128                 limit = (self.config.field.clear_radius_m - pf.margin_m
129                         - self.config.search.clear_margin_m)
130                 if radius <= limit:
131                     return self._near_clear_target(
132                         ClearCertificate(channel, center, radius, limit, True))
133         return cert
134
135     def _select_point(self, channel: int) -> Vec2 | None:
136         """定位测量点：期望 log 幸存占比 + 移动时间 最小的候选。"""
137         context = self._probe_context(channel)
138         if context is None:
139             return None
140         candidates, info_score, center, position = context
141         bayes = self.config.bayes
142
143         def score(q: Vec2) -> float:
144             # 一次定位后仍需接近源才能清除；仅计到检测点的移动会偏好绕行。
145             move = bayes.lambda_move_s * (move_time(position, q, self.config.cost)
146                                           + move_time(q, center, self.config.cost))
147             return info_score(q) + move
148
149         return min(candidates, key=lambda q: (score(q), _point_key(q)))
150
151     def _localize_and_clear(self, channel: int) -> bool:
152         """阻塞式定位子循环（resolve 阶段安全网）：PF 选点，失败退凸选点。"""
153         search = self.config.search
154         self._may_clear.add(channel)
155         for _ in range(search.max_localize_steps):
156             if self._virtual_cap_reached():
157                 return False
158             cert = self._best_certificate(channel)
159             if cert is not None and cert.certified and channel in self._may_clear:
160                 if self._attempt_clear(channel, cert.center, cert):
161                     self._clear_failed.discard(channel)
162                     return True
163                 self._clear_failed.add(channel)
164                 self._may_clear.discard(channel)
165                 continue
166             point = self._select_point(channel)
167             if point is None:
168                 region = self.belief.candidate_set(channel)
169                 if region is None:
170                     return False
171                 point = select_measurement_point(
172                     region, self.client.state.position, self.config).position
173             outcome = self.client.measure(point, channel)
174             self._record_measure(channel, point, outcome)
175             self._may_clear.add(channel)
176             if outcome.result == NO_SIGNAL:
177                 region = self.belief.candidate_set(channel)
178                 if region is not None and safe_disk_radius(region, self.config) >= 0
179                     and math.dist(point, region.mec()[0])
180                     <= safe_disk_radius(region, self.config):
181                     self._anomaly("no_signal_in_safe_disk", channel, position=point)
182             cert = self._best_certificate(channel)
183             if cert is not None and cert.certified:
184                 if self._attempt_clear(channel, cert.center, cert):
185                     self._clear_failed.discard(channel)
186                     return True
187                 self._clear_failed.add(channel)
188                 self._may_clear.discard(channel)
189         return self.belief.channels[channel].status is ChannelStatus.CLEARED

```

问题四：朝向候选、可见性选点与网格遍历清除

每个幸存格附朝向档掩码构成联合候选集，可见性权重用于探点排序；定位停滞时以 20 m 方格遍历清除兜底。

JointBelief：单频道的位置与朝向联合候选集；朝向分 24 档，负观测只在整格必落在最小接收半径内时才约束朝向。

```
1 class JointBelief:
2     """单频道（位置 × 朝向）联合可行集。"""
3
4     def is_absent(self) -> bool:
5         """联合集空 该频道不存在源（真源保留定理的二维推广）。"""
6         return not self.position.alive
7
8     def sample(self, size: int = 256) -> list[int]:
9         """均匀抽样幸存格（评分用；证书始终用完整集合）。"""
10        cached = self._sample_cache
11        if cached is not None and cached[0] == self.position.version:
12            return cached[1]
13        alive = sorted(self.position.alive)
14        if len(alive) > size:
15            step = len(alive) / size
16            alive = [alive[int(k * step) % len(alive)] for k in range(size)]
17        self._sample_cache = (self.position.version, alive)
18        return alive
19
20    def expected_posterior_radius(self, reader: Vec2, size: int | None = None) -> float:
21        """一次测量后的期望后验 MEC 半径（米）；D1 值函数。
22
23        与 :meth:`ChannelParticles.expected_posterior_radius` 同族，差别只在
24        no_signal 分支概率：定向源不仅会因距离 > r_eff 静默，还会因朝向遮挡
25        静默，因此
26
27        
$$p_{\text{vis}}(p) = P(r_{\text{eff}} \mid p-q) \times \text{cover\_ratio}(p, q)$$

28
29        其中 ``cover_ratio`` 是该格朝向掩码中"可能覆盖 q 方位"的档位占比。
30        分支半径语义：
31
32        - direction（可见，非 near）：读数为 q 处的方位 ±，后验收缩到该方位
33          楔形与当前可行集之交，半径取幸存子采样的质心最大距离；
34        - no_signal：定向源的静默**不删空间候选**，位置后验半径保持当前值；
35        - near：后验为 5m 盘。
36
37        返回值只用于**选点排序**，不参与任何安全判定；证书仍由凸/网格口径给出。
38        """
39        pf = self.position
40        if not pf.alive:
41            return 0.0
42        if size is None:
43            size = self.config.bayes.score_subsample
44        indices = self.sample(max(2, size))
45        if not indices:
46            return 0.0
47        stride = max(len(indices) // max(self.config.bayes.mc_representatives, 1), 1)
48        representatives = indices[::stride]
49        if _np is not None:
50            return self._epr_vector(reader, indices, representatives)
51        return self._epr_python(reader, indices, representatives) # pragma: no cover
52
53    def prob_detect(self, reader: Vec2, size: int = 256) -> float:
54        """在 ``reader`` 处能看见任一可行（位置，朝向）的概率质量（启发式）。
55
56        r_eff ~ U[1000, 1500] 解析积分；朝向按可行档位占比计权。
57        只用于选点排序，不参与任何安全判定。
58        """
59        cells = self.sample(size)
60        if not cells:
61            return 0.0
62        total = 0.0
63        span = self.field.r_eff_max_m - self.field.r_eff_min_m
64        for index in cells:
65            gx, gy = self.position.cell_center(index)
66            distance = math.hypot(reader[0] - gx, reader[1] - gy)
67            if distance > self.field.r_eff_max_m:
68                continue
69            if distance <= self.field.r_eff_min_m:
```

```

70         p_range = 1.0
71     else:
72         p_range = (self.field.r_eff_max_m - distance) / span
73     if p_range <= 0.0:
74         continue
75     bearing = math.degrees(math.atan2(reader[1] - gy, reader[0] - gx))
76     cover = _COVER_TABLE[_table_index(bearing)]
77     bits = bin(self._mask(index) & cover).count("1")
78     if bits:
79         total += p_range * bits / HEADING_BINS
80     return total / len(cells)
81
82 def apply(self, station: Vec2, result: str,
83          svd_deg: float | None = None) -> None:
84     """按读数更新联合可行集（保守，真源（位置，朝向）永不被删）。"""
85     if result == DIRECTION:
86         self.position.apply_measure(station, DIRECTION, svd_deg,
87                                   directional=True)
88         self._apply_heading(station, keep=True)
89     elif result == NEAR:
90         # near 同样要求可见（评测机先判朝向再判距离）
91         self.position.apply_measure(station, NEAR, directional=True)
92         self._apply_heading(station, keep=True)
93     elif result == NO_SIGNAL:
94         # 距离阈值必须取**保守侧**：只有"整格都必然落在 r_eff_min 圆盘内"
95         # 时，no_signal 才一定由朝向遮挡解释。格心到格内任一点 margin，
96         # 故阈值取 r_eff_min - margin；取 + margin 会把 1000-1010m 的模糊带
97         # 也当成"必须不覆盖"，从而可能删掉真源的朝向档（破坏真源保留性）。
98         self._apply_heading(station, keep=False,
99                             within_m=max(
100                                 0.0, self.field.r_eff_min_m
101                                 - self.position.margin_m))
102     else:
103         raise ValueError(f"未知测向结果: {result!r}")
104
105 def summary(self) -> dict:
106     """联合集规模快照（写入策略 summary，供审计与论文复算）。"""
107     cells = self.sample(256)
108     bins = [bin(self._mask(index)).count("1") for index in cells]
109     return {"cells": len(self.position),
110            "sampled_mean_heading_bins": (round(sum(bins) / len(bins), 2)
111                                           if bins else 0.0),
112            "heading_bins_total": HEADING_BINS}

```

DirectionalExtension：问题四闭环：定向覆盖计划、遮挡解释与网格遍历清除兜底。

```

1 class DirectionalExtension(OmnidirectionalLoop):
2     """问题4混合源闭环（P5）：全向基线 + 定向观测处理 + 后备清除。"""
3
4     def _localize_and_clear(self, channel: int) -> bool:
5         """定位子循环：选点测量收敛候选集，拿到清除证书后清除。
6
7         与 P4 的差异：保证接收盘内的 no_signal 说明探点被朝向遮挡
8         [观测推理]，走"对路复核 -> 环探"升级梯子（不计入定位步数，
9         由 max_blocked_probes 限流），之后仍未收敛再转后备清除。
10        """
11         search = self.config.search
12         self._may_clear.add(channel)
13         steps = 0
14         while steps < search.max_localize_steps:
15             if self._virtual_cap_reached():
16                 return False
17             region = self.belief.candidate_set(channel)
18             if region is None:
19                 return False
20             cert = clear_certificate(region, channel, self.config)
21             if cert.certified and channel in self._may_clear:
22                 if self._attempt_clear(channel, cert.center, cert):
23                     return True
24                 self._may_clear.discard(channel) # 证书失败：补测后才可重试
25                 continue
26             if len(self.blocked_probes.get(channel, [])) >= search.max_blocked_probes:
27                 break # 梯子用尽仍被遮挡：定位停滞，转后备清除
28             point = self._next_probe(region, channel)
29             within_range = (math.dist(point, region.mec()[0])
30                             <= safe_disk_radius(region, self.config))
31             outcome = self.client.measure(point, channel)

```

```

32     self._observe(channel, point, outcome.result, outcome.svd_deg)
33     self._may_clear.add(channel)
34     if outcome.result == NO_SIGNAL:
35         if within_range and channel not in self.directional_observed:
36             self.directional_observed.append(channel)
37             self.blocked_probes.setdefault(channel, []).append(point)
38             continue # 盘外无信号也可能是距离不足，不据此断言定向
39             self.blocked_probes.pop(channel, None)
40             steps += 1
41     if self.belief.channels[channel].status is ChannelStatus.CLEARED:
42         return True
43     region = self.belief.candidate_set(channel)
44     if region is None:
45         return False
46     cert = clear_certificate(region, channel, self.config)
47     if cert.certified and channel in self._may_clear:
48         if self._attempt_clear(channel, cert.center, cert):
49             return True
50         self._may_clear.discard(channel)
51     return self._fallback_clear(channel, region)
52
53 def _observe(self, channel: int, point: Vec2, result: str,
54             svd_deg: float | None) -> None:
55     """同步更新凸候选集与（位置 × 朝向）联合可行集。
56
57     联合集只在频道首次被看见（direction/near）时建立：从未被看见的
58     频道由覆盖证明负责缺席，不需要朝向维度。联合集一旦为空，直接给出
59     该频道的缺席证明（严格，与覆盖证明同一条件），可提前停止对其检测。
60     """
61     self.belief.update_measure(channel, point, result, svd_deg)
62     if not self.config.search.directional_joint_belief:
63         return
64     joint = self.joint.get(channel)
65     if joint is None:
66         if result not in (DIRECTION, NEAR):
67             return
68         joint = JointBelief(self.config)
69         self.joint[channel] = joint
70     joint.apply(point, result, svd_deg)
71     if joint.is_absent() and self.belief.channels[channel].status
72         is not ChannelStatus.CLEARED:
73         self.belief.mark_absent(channel)
74         self.absence_evidence[channel] = "joint_empty"
75         if channel not in self.joint_absent_channels:
76             self.joint_absent_channels.append(channel)
77
78 def _next_probe(self, region: CandidateRegion, channel: int) -> Vec2:
79     """定向遮挡下的探点升级梯子 [工程]。
80
81     - 未被遮挡过: worst-posterior 选点器（保证接收盘内）；
82     - 第 1 次被遮挡: 对 MEC 圆心取对跖方向，偏移裁进保证接收盘；
83     - 之后: 绕 MEC 圆心、半径 = 的环探，方位自首个被遮挡点起
84       每次 +45°。几何边界必须如实陈述：环上可见弧没有下界——
85       源可位于环上（lg-cl = ）且朝向沿外切线，可见集退化为单个
86       切点，固定步进环探可以全部漏过。梯子只提高提前命中概率，
87       不构成接收或命中保证：完备性完全由后备清除兜底。
88     """
89     visibility = self._visibility_probe(region, channel)
90     if visibility is not None:
91         return visibility
92     blocked = self.blocked_probes.get(channel, [])
93     center, radius = region.mec()
94     if not blocked:
95         return select_measurement_point(region, self.client.state.position,
96                                       self.config).position
97     safe = safe_disk_radius(region, self.config)
98     first = blocked[0]
99     if len(blocked) == 1 and safe >= 0:
100         dx, dy = center[0] - first[0], center[1] - first[1]
101         norm = math.hypot(dx, dy)
102         if norm <= 1e-9:
103             # 首个被遮挡探点恰在圆心：方向信息为零，沿 +x 走满安全盘。
104             return (center[0] + safe, center[1])
105         scale = min(1.0, safe / norm)
106         # 对跖方向，偏移模长 = min(norm, safe): 圆心对称点裁进保证接收盘。
107         return (center[0] + dx * scale, center[1] + dy * scale)
108     base = math.atan2(first[1] - center[1], first[0] - center[0])

```

```

109     azimuth = base + math.radians(45.0 * (len(blocked) - 1))
110     return (center[0] + radius * math.cos(azimuth),
111           center[1] + radius * math.sin(azimuth))
112
113 def _fallback_clear(self, channel: int, region: CandidateRegion) -> bool:
114     """定位停滞的后备：候选集有限圆盘覆盖逐格清除（格数即请求上界）。"""
115     search, session = self.config.search, self.config.session
116     width = max(p[0] for p in region.polygon) - min(p[0] for p in region.polygon)
117     height = max(p[1] for p in region.polygon) - min(p[1] for p in region.polygon)
118     cells = fallback_clear_cells(region.polygon, self.config)
119     entry = {"channel": channel, "cells": len(cells),
120            "bound": fallback_cell_bound(width, height, self.config),
121            "posterior_radius_m": round(region.mec()[1], 3)}
122     self.fallback_log.append(entry)
123     if len(cells) > search.fallback_max_cells:
124         # 宽候选集盲扫违背预算纪律：留待定位收敛，不得硬扫 [计划$4]。
125         entry["outcome"] = "skipped_cell_cap"
126         self._anomaly("fallback_cell_cap_exceeded", channel,
127                      cells=len(cells), cap=search.fallback_max_cells)
128         return False
129     # 现实预算核对：N 次请求 × 单请求高分位估计 + 退出余量，不足不启动。
130     remaining = self.client.real_remaining_s
131     if remaining is None or len(cells) * session.real_rtt_p95_s
132       + session.reserve_exit_s > remaining:
133         entry["outcome"] = "skipped_budget"
134         return False
135     attempts = 0
136     order = self._fallback_order(cells)
137     for cell in order:
138         if self._virtual_cap_reached():
139             entry["outcome"] = "skipped_virtual_budget"
140             entry["attempts"] = attempts
141             return False
142         attempts += 1
143         outcome = self.client.clear(cell, channel)
144         if outcome.result == CLEAR_SUCCESS:
145             self.belief.mark_cleared(channel)
146             self.cleared.append(channel)
147             entry["outcome"] = "cleared"
148             entry["attempts"] = attempts
149             return True
150     # 候选集保守包含真源，格心距格内任一点  $\leq 10\sqrt{2} < 20\text{m}$ ：全部
151     # 失败说明守恒性被打破，按异常留痕，不得据此删除候选。
152     entry["outcome"] = "exhausted"
153     entry["attempts"] = attempts
154     self._anomaly("fallback_exhausted_without_success", channel,
155                  cells=len(cells))
156     return False

```