

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 and 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中对取得, 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。

实验数据取自官方模拟器 test 目录的六份正式导出及对应统计, 累计清除 75 个源, 清除失败计数为零。两问按清除数量加权的平均耗时分别为 249.873 与 736.268 s/个, 移动时间占各次总耗时的 74.47% 至 81.38%。

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 1。

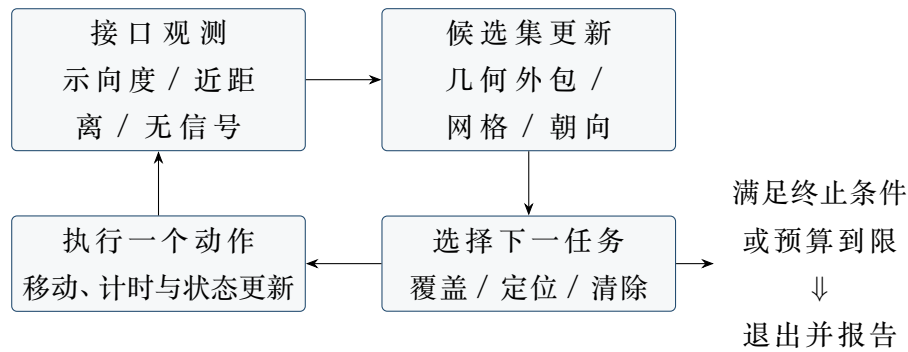


图 1 基于观测更新、集合估计与任务排序的闭环决策过程。

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。
- 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

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

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, $\bar{B}(c, r)$ 为闭圆盘。

5 问题一：有界误差交会定位与覆盖圆

5.1 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

$$W_i = \left\{ g : \begin{cases} [u(\theta_i - \delta), g - s_i] \geq 0, \\ [u(\theta_i + \delta), g - s_i] \leq 0, \\ u(\theta_i) \cdot (g - s_i) \geq 0 \end{cases} \right\}. \quad (1)$$

三条不等式均可写为 $ax + by \leq c$, 其法向依次为

$$\begin{aligned} &(\sin(\theta_i - \delta), -\cos(\theta_i - \delta)), \\ &(-\sin(\theta_i + \delta), \cos(\theta_i + \delta)), \\ &(-\cos \theta_i, -\sin \theta_i), \end{aligned} \quad (2)$$

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

$$P = \bigcap_{i=1}^n W_i = \{g : Ag \leq b\}. \quad (3)$$

式 (3) 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。

5.2 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

$$x = \frac{ce - bf}{ae - bd}, \quad y = \frac{af - cd}{ae - bd}. \quad (4)$$

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

$$\mathcal{R} = \{d : Ad \leq 0\}. \quad (5)$$

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。

5.3 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

$$D(P) = \max_{i,j} \|v_i - v_j\|. \quad (6)$$

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

$$\max_{v \in V(P)} \|v - o\| \leq D/2. \quad (7)$$

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。 $\square$

若条件不成立，输出圆外顶点及超出量 $e = \max_v \|v - o\| - D/2$ ，用于量化该圆盘的覆盖误差。

5.4 反例与可清除半径

构造等边三角形

$$A = (0, 0), \quad B = (40, 0), \quad C = (20, 20\sqrt{3}). \quad (8)$$

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。其最小包围圆心为 $(20, 20/\sqrt{3})$ ，半径 $40/\sqrt{3} \approx 23.094$ m，同样大于清除半径。图 2 展示两类圆的差别。

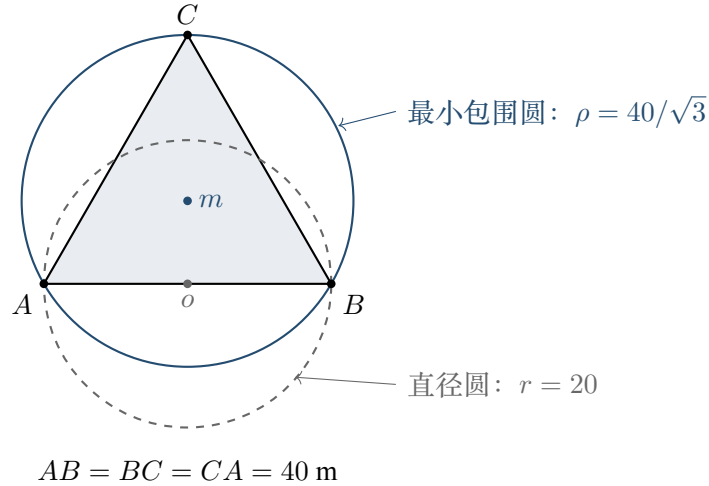


图 2 等边三角形的直径圆与最小包围圆。

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

$$\rho(P) = \min_{z \in \mathbb{R}^2} \max_{g \in P} \|g - z\|, \quad m \in \arg \min_z \max_{g \in P} \|g - z\|. \quad (9)$$

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。

5.5 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。

6 问题二：保证接收的第二检测点选择

6.1 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

$$K_1 = \Omega \cap W(s, \theta, \delta_{\text{eff}}) \cap \overline{B}(s, R_+). \quad (10)$$

其中 $R_+ = 1500$ 为有效接收半径上界。`direction` 对应距离大于 5 m；式 (10) 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

$$F_K(q) = \max_{g \in K} \|q - g\|, \quad \mathcal{Q}(K) = \{q : F_K(q) \leq R_-\} = \bigcap_{g \in K} \overline{B}(g, R_-). \quad (11)$$

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 (9) 可知

$$\mathcal{Q}(K) \neq \emptyset \iff \rho(K) \leq R_-. \quad (12)$$

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

$$e_M(r) = r(\sec(\pi/M) - 1). \quad (13)$$

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 $\overline{B}(m, \rho_{\text{out}})$ 。

由三角不等式，充分安全盘为

$$\mathcal{Q}_{\text{in}} = \overline{B}(m, R_- - \rho_{\text{out}} - \eta), \quad \eta = 10^{-3} \text{ m}. \quad (14)$$

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。

6.2 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 $S = W(s, \theta, \delta) \cap \overline{B}(s, R_+)$ 。当 $0 \leq \delta \leq 45^\circ$ ，令

$$a = \frac{R_+}{2 \cos \delta}, \quad c = s + a u(\theta). \quad (15)$$

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

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

$$q \in \overline{B}\left(s + \frac{1500}{2 \cos \delta} u(\theta), 1000 - \frac{1500}{2 \cos \delta}\right) \quad (16)$$

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。

表 2 解析安全盘参数，由式 (16) 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

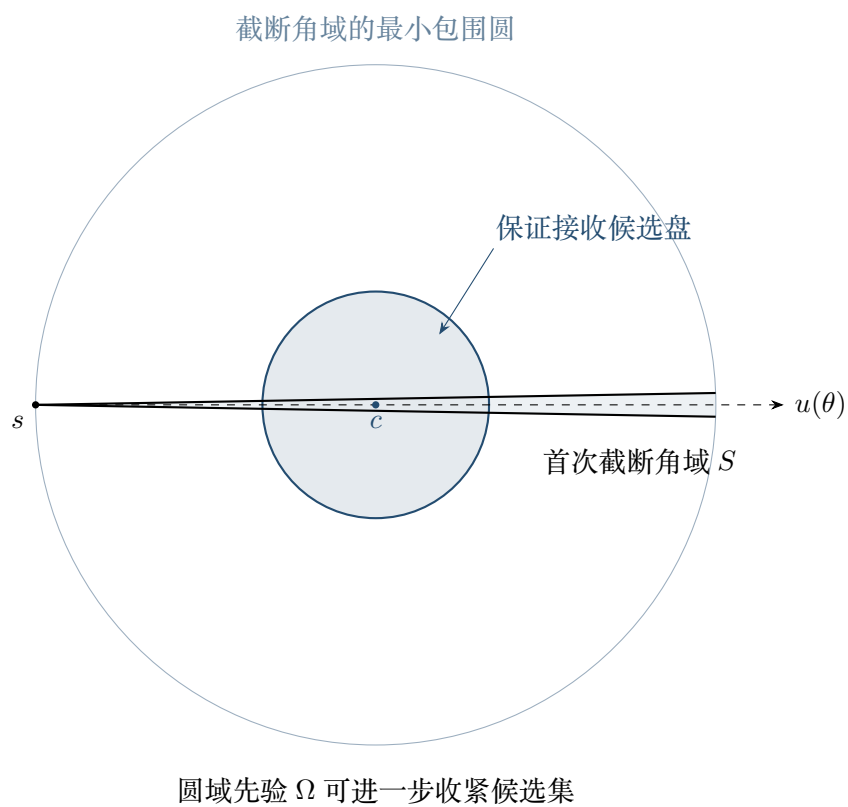


图 3 第二点解析候选区域的几何关系，采用 1° 半角、同一坐标比例绘制。

6.3 定位效果指标与有限候选求解

理想的鲁棒选点模型为

$$\min_{q \in \mathcal{Q}_{in}} \left(\sup_{y \in \mathcal{Y}(q, K_1)} \rho(K_1 \cap O(q, y)), \frac{\|q - p\|}{5} \right)_{\text{lex}}, \quad (17)$$

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 (14) 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 (14) 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 (10) 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. 对每点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。

7 问题三：全向多源的覆盖搜索与滚动调度

7.1 统一时间目标与任务状态

设整个会话移动距离为 L , 检测、换频、成功和失败清除次数分别为 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

$$T = \frac{L}{5} + 5N_m + N_s + 5N_c^+ + 3N_c^-. \quad (18)$$

优化目标是在全部源清除的约束下最小化 T 。题目统计的平均定位清除时间为 $\bar{T} = T/N_c^+$, 包含搜索、定位与清除的全部耗时。清除请求中的频道指定操作对象, 测向机保持原频道; 进入与退出的虚拟耗时为零, 因此采用终点自由的开放路径模型。

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。

7.2 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 $a = 1159.7 \text{ m}$, 由式 (20) 得 $D_{\text{cov}} = 984.539136 \text{ m}$, 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。

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。

7.3 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 $\bar{B}(s, 1500)$ 外切多边形及测向角域相交; near 与 $\bar{B}(s, 5)$ 外切多边形相交。全向源的 `no_signal` 意味着

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

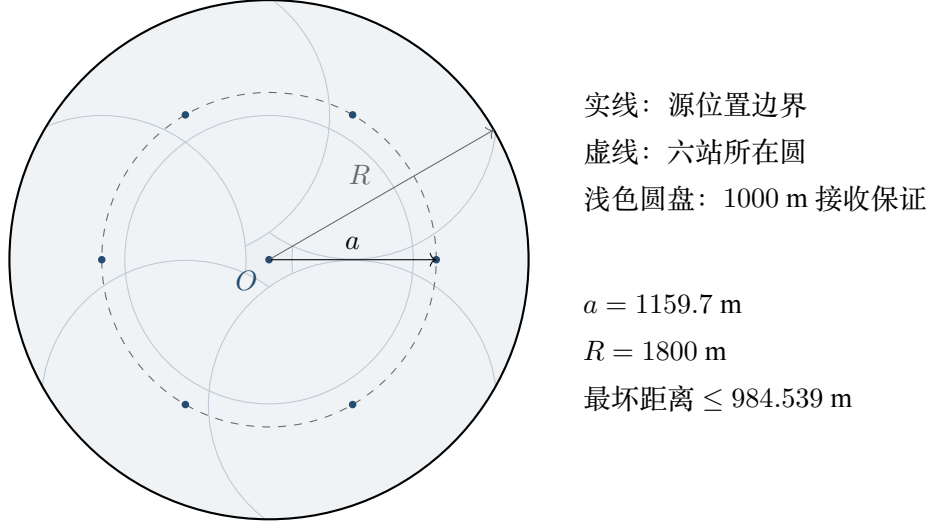


图4 七点全向覆盖的站点构造与接收圆盘。

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 ：

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

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格的可行性。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。

7.4 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

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

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。

7.5 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。

7.6 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，虚拟动作耗时以整数微秒累计，程序运行时间由官方统计单独记录。

8 问题四：混合源的定向覆盖与网格遍历清除

8.1 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。

8.2 方格点集的连续覆盖性

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。 $\square$

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。

8.3 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。

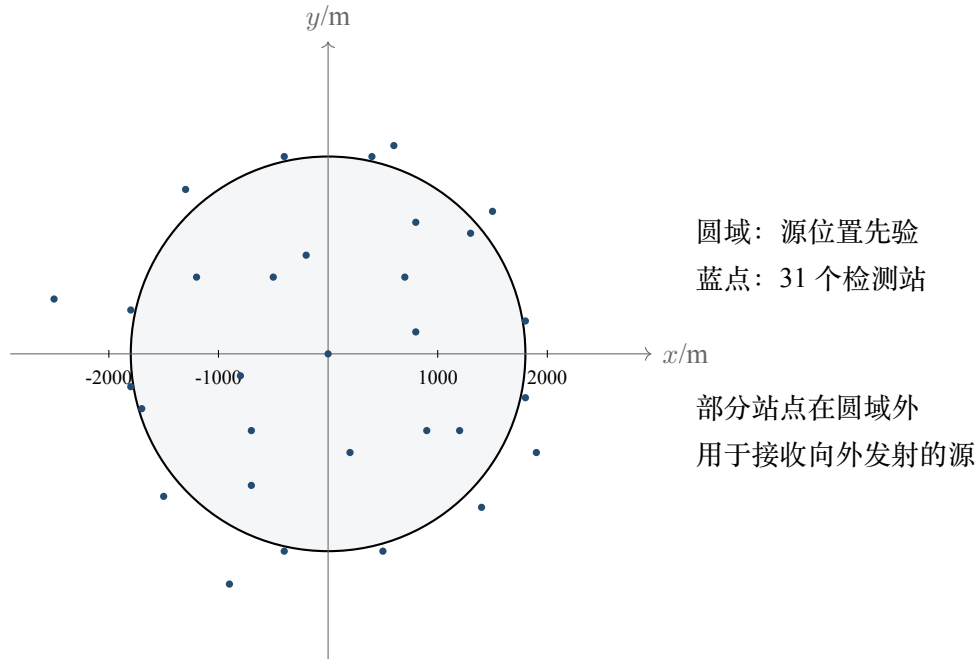


图 5 第四问集合覆盖模型得到的 31 个检测站及源位置区域。

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。

8.4 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

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

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 10.2 节。

8.5 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。

8.6 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。

8.7 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。
-

问题四采用站内批量检测与最近邻任务排序。算法结合位置集合、朝向约束和剩余预算确定测点及清除顺序，并以各频道的处理状态作为终止时的输出。

9 官方正式测试结果与时间分解

9.1 实验数据与评价指标

实验采用官方模拟器 Jammers-simulator-full/test 目录中的六份正式导出，包括问题三、四各三次测试。各文件经 SHA-256 校验后，与官方原始导出、结果侧文件及正式

统计记录建立对应关系 [7]。案例编码与文件校验值列于附录 A。

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。

9.2 问题三正式结果

表 3 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 3 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。

9.3 问题四正式结果

表 4 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

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。

9.4 动作耗时分解

将式 (18) 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 5 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 6 的耗时分解。

表 5 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%

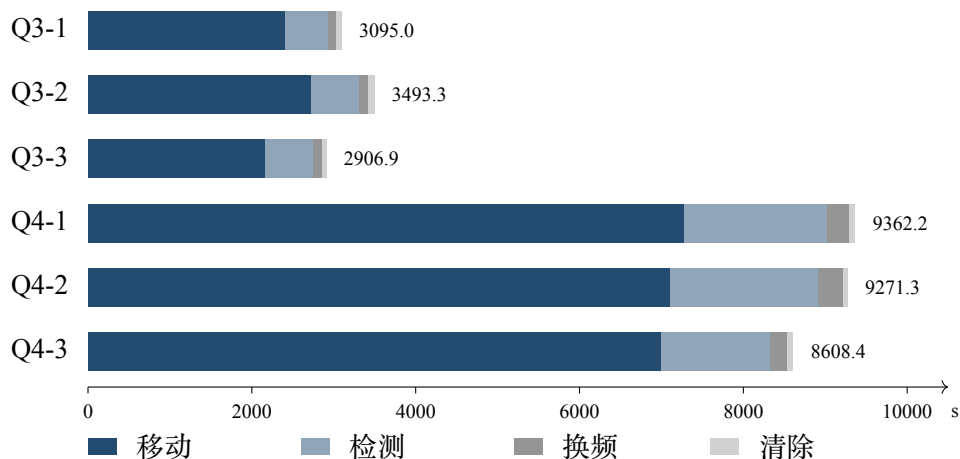


图 6 正式测试的累计虚拟耗时分解。移动时间由总耗时与各动作计数计算。

六次测试中，移动时间占总虚拟耗时的 74.47% 至 81.38%，是主要时间成本。问题三的检测次数为 105、117、117，问题四为 349、361、264；混合源案例中，朝向约束增

加了观测与位置选择的复杂度。因此，后续优化可同时考虑减少移动距离与提高单次观测的信息增益。

由 $L_{eq} = 5T_{move}$ 得问题三的等效移动距离依次为 12.045、13.627、10.824 km，问题四依次为 36.386、35.507、35.027 km。该距离由官方时间计数及移动速度计算。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。

9.5 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。

10 模型评价与推广

10.1 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。

10.2 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。

10.3 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 2 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、内存和计算时间取得平衡。

10.4 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 6。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 6 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。

B 支撑材料与运行方法

表 7 支撑材料目录及用途

文件或目录	用途
src/mcm26b/	几何计算、集合估计、覆盖优化、任务排序与通信实现
src/mcm26b/guarantee/plans/	第四问站点方案与离散验证参数
scripts/audit_paper_formal.py	正式文件核对、统计计算与表格生成
scripts/build_paper_support.py	矢量图、源码附录与支撑包生成
docs/paper/data/	六个正式案例的统计快照、表格及校验值
docs/paper/sections/	分章论文源文件
docs/paper/figures/	几何示意图与正式耗时图的矢量源文件
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 完整程序源码

以下列出当前策略、配套模块及论文数据处理程序的完整源码。附录通过生成的显示副本将数学符号转换为 LaTeX 命令, 支撑包保留对应的可执行 Python 文件。实验结果见正文正式测试表及附录 A 的数据清单。

src/mcm26b/__init__.py

```
1  """mcm26b: CUMCM 2026 B题"无线电干扰源的快速自动定位与清除"求解包。
2
3  模块分层 (docs/B题建模分析与实施计划.md §7 的七模块划分):
4  - config      全项目参数中心 (题面常量与工程参数, 唯一字面量来源)
5  - protocol    P0/P1 字段表、计时模型、状态机与串行幂等客户端
6  - geometry    P2 角域交会定位、直径、最小包围圆与 P2 证书 (已完工)
7  - sim         自建模拟器、故障注入与演练日志回放 (测试专用)
8  - belief      P3 逐频道候选集合 (骨架)
9  - coverage    P4 49 点覆盖记录 (骨架+网格生成)
10 - planner     P3 选点 / P4 全向闭环 / P5 定向扩展 (骨架)
11 - evaluation  日志复算与统计
12 """
13
14 __version__ = "0.2.0"
15
16 from .config import DEFAULT_CONFIG, ProblemConfig
17 from .geometry import Measurement, first_bearing_candidate_disk, solve, \
18     smallest_enclosing_circle
19
20 __all__ = ["DEFAULT_CONFIG", "Measurement", "ProblemConfig", "solve",
21            "first_bearing_candidate_disk", "smallest_enclosing_circle"]
```

src/mcm26b/belief/__init__.py

```
1  """P3 逐频道信念状态：候选集合 K_c 维护与频道处理状态。
2
3  核心不变式（验收标准）：
4  - 保守性：真源永远属于维护中的候选集合。候选集用"圆盘外切多边形  $\cap$ 
5    测向角域半平面"的凸多边形表示——外切多边形严格包含圆盘，半平面裁剪
6    带正容差，因此每步更新只会放大而不会缩小真实候选集；
7  - direction 读数： $K_c \subseteq \Omega \cap W(s, \theta \pm \delta_{\text{eff}}) \cap B(s, 1500)$ ，后续读数再交
8    同样的约束； $\delta_{\text{eff}}$  = 误差半角 + 示向度舍入裕量（服务器保留两位小数，
9    舍入本身引入  $\leq 0.005^\circ$  偏差）；
10 - no_signal 读数（P4，全向源）：基线不据此删空间候选——对定向源它是
11   "位置 $\times$ 朝向"联合约束，单独维护空间候选会误删真源（P5 才启用）；
12   "该频道不存在源"的证明由覆盖账本（coverage.CoverageLedger）给出；
13 - near 读数：源在 5m 内，候选集收缩为  $B(s, 5)$ ，可直接产生清除证书；
14 - /clear 成功：状态转 CLEARED；失败只说明"该频道 20m 内无可清除源"，
15   不得直接删除超出 20m 的候选，必须先补测再重试。
16 """
17
18 from __future__ import annotations
19
20 import math
21 from dataclasses import dataclass, field
22 from enum import Enum
23
24 from ..config import ProblemConfig, Vec2
25 from ..geometry.localization import smallest_enclosing_circle
26 from ..protocol.rules import DIRECTION, NEAR, NO_SIGNAL
27
28 # 半平面裁剪的正容差（米）。坐标量级  $\sim 2100$ ，浮点误差远小于它；
29 # 容差只向"保留更多候选"的方向倾斜，不破坏保守性。
30 _CLIP_TOL_M = 1e-6
31 _DEDUPE_TOL_M = 1e-7
32
33
34 class InconsistentBelief(RuntimeError):
35     """读数把候选集剪空等违背保守性的情形：须停机核查而非继续跑。"""
36
37
38 class ChannelStatus(Enum):
39     UNKNOWN = "unknown"          # 尚无有效证据
40     DETECTED = "detected"         # 收到过 direction/near，待定位
41     LOCATED = "located"           # 候选集包围圆半径  $\leq 20$ ，可清除
42     CLEARED = "cleared"           # 清除成功
43     ABSENT = "absent"             # 覆盖证明不存在（P4）
44
45
46 def circumscribed_polygon(center: Vec2, radius_m: float, sides: int) -> list[Vec2]:
47     """圆的外切正多边形顶点（外接半径  $r/\cos(\pi/n)$ ），严格包含圆盘。
48
49     近似误差（外接圆超出圆盘的部分）为  $r(1/\cos(\pi/n) - 1) \leq r(1 - \cos(\pi/n)) \cdot \sec^2$ ，
50     调用方用  $(1 - \cos(\pi/n))$  口径记账即可（略保守偏大）。
51     """
52     if sides < 8:
53         raise ValueError("外切多边形边数过少，近似误差不可控")
54     circumradius = radius_m / math.cos(math.pi / sides)
55     step = 2 * math.pi / sides
56     return [(center[0] + circumradius * math.cos(step * i + step / 2),
57             center[1] + circumradius * math.sin(step * i + step / 2))
58             for i in range(sides)]
59
60
61 def polygon_halfplanes(polygon: list[Vec2]) -> list[tuple[float, float, float]]:
62     """CCW 多边形 -> 外法线半平面列表 (a, b, c)，含义  $a \cdot x + b \cdot y \leq c$ 。"""
63     planes = []
64     count = len(polygon)
65     for index in range(count):
66         p, q = polygon[index], polygon[(index + 1) % count]
67         dx, dy = q[0] - p[0], q[1] - p[1]
68         norm = math.hypot(dx, dy)
```

```

69         if norm <= 1e-12:
70             continue
71         a, b = dy / norm, -dx / norm      # CCW 的右法线朝外
72         planes.append((a, b, a * p[0] + b * p[1]))
73     return planes
74
75
76 def clip_halfplane(polygon: list[Vec2], a: float, b: float, c: float,
77                   tol: float = _CLIP_TOL_M) -> list[Vec2]:
78     """Sutherland-Hodgman 凸多边形半平面裁剪，保留  $a \cdot x + b \cdot y \leq c + \text{tol}$ 。
79
80     边界交点取在无容差直线上；进入判定带正容差，保证"贴边"的真点不丢。
81     """
82     if not polygon:
83         return []
84     output: list[Vec2] = []
85     count = len(polygon)
86     for index in range(count):
87         p, q = polygon[index], polygon[(index + 1) % count]
88         fp = a * p[0] + b * p[1] - c
89         fq = a * q[0] + b * q[1] - c
90         p_in, q_in = fp <= tol, fq <= tol
91         if p_in:
92             output.append(p)
93         if p_in != q_in:
94             ratio = fp / (fp - fq)
95             output.append((p[0] + ratio * (q[0] - p[0]),
96                           p[1] + ratio * (q[1] - p[1])))
97     return _dedupe(output)
98
99
100 def _dedupe(points: list[Vec2], tol: float = _DEDUPE_TOL_M) -> list[Vec2]:
101     unique: list[Vec2] = []
102     for point in points:
103         if not unique or math.dist(point, unique[-1]) > tol:
104             unique.append(point)
105     if len(unique) > 1 and math.dist(unique[0], unique[-1]) <= tol:
106         unique.pop()
107     return unique
108
109
110 class CandidateRegion:
111     """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）。"""
112
113     def __init__(self, polygon: list[Vec2], slack_m: float = 0.0):
114         self.polygon = list(polygon)
115         self.slack_m = float(slack_m)      # 外包近似引入的半径向误差记账
116         self._mec: tuple[Vec2, float] | None = None
117
118     # ----- 构造
119     @classmethod
120     def disk(cls, center: Vec2, radius_m: float, sides: int,
121              slack_m: float | None = None) -> "CandidateRegion":
122         if slack_m is None:
123             slack_m = radius_m * (1 - math.cos(math.pi / sides))
124         return cls(circumscribed_polygon(center, radius_m, sides), slack_m)
125
126     @classmethod
127     def initial(cls, station: Vec2, config: ProblemConfig) -> "CandidateRegion":
128         """首个 direction 后的基础集： $\Omega \cap B(\text{station}, r_{\text{max}})$  的保守外包。"""
129         field_cfg, sides = config.field, config.search.polygon_sides
130         slack = field_cfg.omega_radius_m * (1 - math.cos(math.pi / sides))
131         region = cls.disk((0.0, 0.0), field_cfg.omega_radius_m, sides, slack)
132         region.clip_disk(station, field_cfg.r_eff_max_m, sides)
133         return region
134
135     # ----- 收缩
136     def _clip(self, a: float, b: float, c: float) -> None:
137         polygon = clip_halfplane(self.polygon, a, b, c)
138         if not polygon:

```

```

139         raise InconsistentBelief(
140             f"候选区域被半平面 ({a:.6f},{b:.6f},{c:.6f}) 剪空: 读数与保守集矛盾")
141     self.polygon = polygon
142     self._mec = None
143
144     def clip_disk(self, center: Vec2, radius_m: float, sides: int) -> None:
145         """与圆盘 B(center, radius) 的外切多边形求交 (近似误差入账)。"""
146         for a, b, c in polygon_halfplanes(circumscribed_polygon(center, radius_m, sides)):
147             self._clip(a, b, c)
148         self.slack_m += radius_m * (1 - math.cos(math.pi / sides))
149
150     def clip_wedge(self, station: Vec2, bearing_deg: float,
151                   half_angle_deg: float) -> None:
152         """与测向角域 W(station, bearing+half_angle) 求交 (半角须 <90°)。
153
154         两个边界半平面与 localization.halfplanes 同口径, 0°/360° 跨界由
155         sin/cos 直接处理, 无需分支。
156         """
157         if not 0 < half_angle_deg < 90:
158             raise ValueError("角域半角须在 (0, 90) 度")
159         lower = math.radians(bearing_deg - half_angle_deg)
160         upper = math.radians(bearing_deg + half_angle_deg)
161         sx, sy = float(station[0]), float(station[1])
162         self._clip(math.sin(lower), -math.cos(lower),
163                   math.sin(lower) * sx - math.cos(lower) * sy)
164         self._clip(-math.sin(upper), math.cos(upper),
165                   -math.sin(upper) * sx + math.cos(upper) * sy)
166
167     # ----- 查询
168     def mec(self) -> tuple[Vec2, float]:
169         """最小包围圆 (圆心, 半径)。半径是真实候选集  $\rho$  的上界。"""
170         if self._mec is None:
171             if not self.polygon:
172                 raise InconsistentBelief("候选区域为空")
173             self._mec = smallest_enclosing_circle(self.polygon)
174         return self._mec
175
176     def witnesses(self, spacing_m: float) -> list[Vec2]:
177         """顶点 + 边界按间距细分的采样点 (最坏后验评估用)。"""
178         if len(self.polygon) < 3:
179             return list(self.polygon)
180         points = list(self.polygon)
181         count = len(self.polygon)
182         for index in range(count):
183             p, q = self.polygon[index], self.polygon[(index + 1) % count]
184             length = math.dist(p, q)
185             steps = int(length // spacing_m)
186             for k in range(1, steps + 1):
187                 ratio = k / (steps + 1)
188                 points.append((p[0] + ratio * (q[0] - p[0]),
189                               p[1] + ratio * (q[1] - p[1])))
190         return _dedupe(points, tol=1e-9)
191
192     def contains(self, point: Vec2, tol_m: float = 0.05) -> bool:
193         """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)。"""
194         if not self.polygon:
195             return False
196         if len(self.polygon) == 1:
197             return math.dist(point, self.polygon[0]) <= tol_m + self.slack_m
198         if len(self.polygon) == 2:
199             p, q = self.polygon
200             return _point_segment_distance(point, p, q) <= tol_m + self.slack_m
201         for index, p in enumerate(self.polygon):
202             q = self.polygon[(index + 1) % len(self.polygon)]
203             if (q[0] - p[0]) * (point[1] - p[1]) - (q[1] - p[1]) * (point[0] - p[0]) \
204                 < -tol_m - self.slack_m:
205                 return False
206         return True
207
208

```

```

209 def _point_segment_distance(point: Vec2, p: Vec2, q: Vec2) -> float:
210     vx, vy = q[0] - p[0], q[1] - p[1]
211     length_sq = vx * vx + vy * vy
212     if length_sq <= 1e-18:
213         return math.dist(point, p)
214     t = max(0.0, min(1.0, ((point[0] - p[0]) * vx + (point[1] - p[1]) * vy) / length_sq))
215     return math.dist(point, (p[0] + t * vx, p[1] + t * vy))
216
217
218 @dataclass
219 class ChannelBelief:
220     channel: int
221     status: ChannelStatus = ChannelStatus.UNKNOWN
222     region: CandidateRegion | None = None
223     measurements: list = field(default_factory=list) # 逐步观测记录 (审计用)
224
225
226 class BeliefState:
227     """全部 20 个频道的信念集合。策略端唯一的事实来源。"""
228
229     def __init__(self, config: ProblemConfig = ProblemConfig()):
230         self.config = config
231         self.channels = {channel: ChannelBelief(channel)
232                         for channel in range(config.field.channel_min,
233                                             config.field.channel_max + 1)}
234
235     @property
236     def effective_half_angle_deg(self) -> float:
237         """示向度误差半角 + 舍入裕量: 服务器保留两位小数引入  $\leq 0.005^\circ$  偏差。"""
238         return (self.config.field.bearing_error_half_deg
239                 + self.config.search.bearing_rounding_margin_deg)
240
241     def update_measure(self, channel: int, position: Vec2, result: str,
242                       svd_deg: float | None) -> None:
243         """按读数收敛候选集合; 真源保留性由保守表示保证。"""
244         belief = self.channels[channel]
245         belief.measurements.append({"position": tuple(position), "result": result,
246                                   "svd_deg": svd_deg})
247
248         if result == DIRECTION:
249             if svd_deg is None:
250                 raise ValueError("direction 读数必须携带 svd_deg")
251             sides = self.config.search.polygon_sides
252             if belief.region is None:
253                 belief.region = CandidateRegion.initial(position, self.config)
254             else:
255                 # 后续 direction 同样隐含距离  $\leq r_{\max}$  [题面]
256                 belief.region.clip_disk(position, self.config.field.r_eff_max_m, sides)
257                 belief.region.clip_wedge(position, svd_deg, self.effective_half_angle_deg)
258                 belief.status = ChannelStatus.DETECTED
259             elif result == NEAR:
260                 sides = self.config.search.polygon_sides
261                 if belief.region is None:
262                     belief.region = CandidateRegion.disk(
263                         tuple(position), self.config.field.near_radius_m, sides)
264                 else:
265                     belief.region.clip_disk(position, self.config.field.near_radius_m, sides)
266                     belief.status = ChannelStatus.DETECTED
267             elif result == NO_SIGNAL:
268                 # P3/P4 基线: 不据此删空间候选 (缺席证明由覆盖账本给出)。
269                 pass
270             else:
271                 raise ValueError(f"未知测向结果: {result!r}")
272
273     def candidate_set(self, channel: int) -> CandidateRegion | None:
274         """该频道当前候选集合 (保守外包形式); 尚无 direction/near 时为 None。"""
275         return self.channels[channel].region
276
277     def mark_cleared(self, channel: int) -> None:
278         self.channels[channel].status = ChannelStatus.CLEARED

```

```

279     def mark_absent(self, channel: int) -> None:
280         self.channels[channel].status = ChannelStatus.ABSENT
281
282     def channels_by_status(self, *statuses: ChannelStatus) -> list[int]:
283         return sorted(channel for channel, belief in self.channels.items()
284                        if belief.status in statuses)

```

src/mcm26b/belief/joint.py

```

1  """位置×朝向联合可行集（第四问定向源）。
2
3  现状问题（docs/第三四问架构优化与运行提速方案.md §1.3 F3）：第四问的失败**全部**
4  来自定向源（评测机两局：全向 17/17 清除，定向仅 2/6）。原因不是覆盖不足，而是
5  策略只知道“源可能在某个空间区域”，不知道“它朝着哪边”——于是选点只能靠固定
6  45° 阶梯试探，一旦试探失败就退化到 ≤400 格 20 m 盲清。
7
8  本模块把“看不看得见”变成可推理的量：为已在  $\Omega$  上维护的网格幸存集
9  （:class:`~mcm26b.particles.ChannelParticles`）附加每格 24 档朝向可行性掩码，
10 得到（位置 × 朝向）联合可行集。观测语义严格对齐评测机
11 ``referee.simulator._observe``：
12
13 - 距离 >  $r_{\text{eff}}$  → no_signal；定向且源→读点方向与朝向夹角 >  $90^\circ$  → no_signal；
14 - 距离 ≤ 5 m → near（**仍需可见**：3 m 处背向的定向源同样返回 no_signal）；
15 - 其余 → direction（读数 = 源→读点方位 + 有界误差）。
16
17 更新规则（全部保守，只删“必然不可能”的格/档）：
18
19 =====
20 观测           位置           朝向
21 =====
22 ``direction``@s 距离 ≤ 1500、角向  $|gap| \leq \delta_{\text{eff}} + \text{格裕量}$  必须**可能**覆盖 s
23 ``near``@s      距离 ≤ 5 必须**可能**覆盖 s
24 ``no_signal``@s **不删**（可能是朝向遮挡或距离不足） 必须**确定不**覆盖 s
25                （仅当整格必在  $r_{\text{eff\_min}}$  内才约束朝向）
26 =====
27
28 阈值全部取保守侧，并计入**格内角裕量**
29 ``asin(margin / max(dist - margin, 1))``（格内任一点相对格心方位偏差上界）：
30
31 - “必须可能覆盖”：``gap ≤ 90° + 半档 + 查表舍入 + 格内角裕量``（宁可保留）；
32 - “必须确定不覆盖”：先删去 ``gap ≤ 90° - 半档 - 查表舍入 - 格内角裕量`` 的档，
33   只保留确定看不见 s 的档（同样宁可保留）。
34
35 因此真源（位置，朝向）永不被删——这是网格幸存集真源保留定理的二维推广。
36 （2026-09-12 修正：no_signal 分支原先误用“确定覆盖”掩码做交集，等价于把
37 规则反向；keep=True 分支也漏计了格内角裕量。两处都会删掉真源朝向档，
38 现由 ``tests/test_joint_belief.py`` 的随机观测序列保留性测试固化。）
39
40 用途：
41 1. ``prob_detect``：该点能看见该频道任一可行（位置，朝向）的概率质量，
42   用于**可见性选点**（取代固定阶梯）——只影响选点顺序，不承担安全责任；
43 2. ``expected_posterior_radius``：一次测量后的期望后验 MEC 半径（D1 值函数）；
44 3. ``is_absent``：联合集空  $\implies$  该频道不存在源（严格，与覆盖证明同一条件）；
45 4. 位置边际仍由凸证书负责 20 m 清除判定，本模块不改变任何证书口径。
46 """
47
48 from __future__ import annotations
49
50 import math
51
52 from ..config import ProblemConfig, Vec2
53 from ..particles import ChannelParticles, _branch_radius_array
54 from ..protocol.rules import DIRECTION, NEAR, NO_SIGNAL
55
56 HEADING_BINS = 24 # 15° 一档
57 _BIN_HALF_DEG = 180.0 / HEADING_BINS # 7.5°
58 _FULL_MASK = (1 << HEADING_BINS) - 1
59 _GRID_DEG = 3.0 # 角度查表分辨率
60 _GRID_N = int(round(360.0 / _GRID_DEG))

```

```

61
62 try:                                     # numpy 仅作可选加速后端 (与 particles 一致)
63     import numpy as _np
64     from ..particles import _np as _particles_np
65     if _particles_np is None:             # MCM26B_NO_NUMPY 或缺失: 强制纯 Python
66         _np = None
67 except Exception:                         # pragma: no cover - 环境无 numpy
68     _np = None
69
70
71 def heading_gap(a_deg: float, b_deg: float) -> float:
72     """两个朝向角的最小夹角 (度)。"""
73     return abs((a_deg - b_deg + 180.0) % 360.0 - 180.0)
74
75
76 def _cover_mask(bearing_deg: float, half_deg: float) -> int:
77     """与该方位夹角不超过 ``half_deg`` 的朝向档掩码。"""
78     mask = 0
79     for k in range(HEADING_BINS):
80         if heading_gap(360.0 * k / HEADING_BINS, bearing_deg) <= half_deg:
81             mask |= 1 << k
82     return mask
83
84
85 # 查表: 以 bearing 为轴、半角 90° 的可见半平面。
86 # - COVER: **可能**覆盖该方位 (含半档与查表舍入的保守侧);
87 # - SILENT: **确定**覆盖该方位 (内缩半档与舍入) ——离散对照表, 供审计与
88 #   文档比对; 实际裁剪走 _cover_mask / _far_mask, 因为还必须逐格计入
89 #   "格内角裕量" (格心到格内任一点的方位偏差上界), 无法预先制表。
90 _COVER_TABLE = tuple(
91     _cover_mask(_GRID_DEG * i, 90.0 + _BIN_HALF_DEG + _GRID_DEG / 2)
92     for i in range(_GRID_N))
93 _SILENT_TABLE = tuple(
94     _cover_mask(_GRID_DEG * i, 90.0 - _BIN_HALF_DEG - _GRID_DEG / 2)
95     for i in range(_GRID_N))
96
97
98 def _far_mask(bearing_deg: float, exclude_half_deg: float) -> int:
99     """与方位夹角**大于** ``exclude_half_deg`` 的朝向档掩码 (no_signal 用)。
100
101     与 :func:`_cover_mask` 互补: no_signal 只允许保留"确定看不见"的朝向档。
102     """
103     mask = 0
104     for k in range(HEADING_BINS):
105         if heading_gap(360.0 * k / HEADING_BINS, bearing_deg) > exclude_half_deg:
106             mask |= 1 << k
107     return mask
108
109 if _np is not None:                       # 12 位半字 popcount 查表 (朝向掩码 24 位) 与半平面查表数组
110     _POP16 = _np.array([bin(value).count("1") for value in range(1 << 12)],
111                        dtype=_np.int64)
112     _COVER_ARR = _np.array(_COVER_TABLE, dtype=_np.int64)
113 else:                                     # pragma: no cover - 无 numpy 分支
114     _POP16 = _COVER_ARR = None
115
116
117 def _popcount(mask: int) -> int:
118     return bin(mask).count("1")
119
120
121 def _table_index(bearing_deg: float) -> int:
122     return int(round((bearing_deg % 360.0) / _GRID_DEG)) % _GRID_N
123
124
125 class JointBelief:
126     """单频道 (位置 × 朝向) 联合可行集。"""
127
128     def __init__(self, config: ProblemConfig = ProblemConfig()):
129         self.config = config
130         self.field = config.field

```

```

131     self.position = ChannelParticles(config)
132     self.masks: dict[int, int] = {}          # 格索引 -> 朝向掩码 (缺省为全档)
133     self._sample_cache: tuple[int, list[int]] | None = None
134
135     # ----- 查询
136     def __len__(self) -> int:
137         return len(self.position)
138
139     def is_absent(self) -> bool:
140         """联合集空  $\implies$  该频道不存在源 (真源保留定理的二维推广) 。”"""
141         return not self.position.alive
142
143     def _mask(self, index: int) -> int:
144         return self.masks.get(index, _FULL_MASK)
145
146     def sample(self, size: int = 256) -> list[int]:
147         """均匀抽样幸存格 (评分用; 证书始终用完整集合) 。”"""
148         cached = self._sample_cache
149         if cached is not None and cached[0] == self.position.version:
150             return cached[1]
151         alive = sorted(self.position.alive)
152         if len(alive) > size:
153             step = len(alive) / size
154             alive = [alive[int(k * step) % len(alive)] for k in range(size)]
155         self._sample_cache = (self.position.version, alive)
156         return alive
157
158     def _mask_popcount(self, index: int) -> int:
159         return _popcount(self._mask(index))
160
161     def expected_posterior_radius(self, reader: Vec2, size: int | None = None) -> float:
162         """一次测量后的期望后验 MEC 半径 (米); D1 值函数。
163
164         与 :meth:`ChannelParticles.expected_posterior_radius` 同族, 差别只在
165         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
166         静默, 因此
167
168             p_vis(p) = P(r_eff ≥ |p-q|) × cover_ratio(p, q)
169
170         其中 ``cover_ratio`` 是该格朝向掩码中"可能覆盖 q 方位"的档位占比。
171         分支半径语义:
172
173         - direction (可见, 非 near): 读数为 q 处的方位 ±δ, 后验收缩到该方位
174           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;
175         - 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;
176         - near: 后验为 5m 盘。
177
178         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。
179         """
180         pf = self.position
181         if not pf.alive:
182             return 0.0
183         if size is None:
184             size = self.config.bayes.score_subsample
185         indices = self.sample(max(2, size))
186         if not indices:
187             return 0.0
188         stride = max(len(indices) // max(self.config.bayes.mc_representatives, 1), 1)
189         representatives = indices[::stride]
190         if _np is not None:
191             return self._epr_vector(reader, indices, representatives)
192         return self._epr_python(reader, indices, representatives) # pragma: no cover
193
194     def _epr_vector(self, reader: Vec2, indices: list[int],
195                    representatives: list[int]) -> float:
196         pf = self.position
197         points = _np.array([pf.cell_center(index) for index in indices], dtype=float)
198         masks = _np.array([self._mask(index) for index in indices], dtype=_np.int64)
199         reps = _np.array([pf.cell_center(index) for index in representatives],
200                          dtype=float)

```

```

201     rep_masks = _np.array([self._mask(index) for index in representatives],
202                           dtype=_np.int64)
203     dx, dy = points[:, 0] - reader[0], points[:, 1] - reader[1]
204     dist = _np.hypot(dx, dy)
205     bearing = _np.degrees(_np.arctan2(dy, dx))
206     rdx, rdy = reps[:, 0] - reader[0], reps[:, 1] - reader[1]
207     rdist = _np.hypot(rdx, rdy)
208     rbearing = _np.degrees(_np.arctan2(rdy, rdx))
209     cover = _COVER_ARR[_np.round(_np.mod(rbearing, 360.0) / _GRID_DEG)
210                       .astype(_np.int64) % _GRID_N]
211     visible_mask = rep_masks & cover
212     bits = (_POP16[visible_mask & 0xFFFF]
213            + _POP16[(visible_mask >> 12) & 0xFFFF])
214     cover_ratio = bits / float(HEADING_BINS)
215     p_silent_range = _np.clip((rdist - self.field.r_eff_min_m)
216                               / (self.field.r_eff_max_m
217                                  - self.field.r_eff_min_m), 0.0, 1.0)
218     p_vis = (1.0 - p_silent_range) * cover_ratio
219     p_ns = 1.0 - p_vis
220     gaps = _np.abs((bearing[None, :] - rbearing[:, None] + 180.0) % 360.0 - 180.0)
221     in_wedge = ((gaps <= self.config.field.bearing_error_half_deg
222                  + self.config.search.bearing_rounding_margin_deg + 1e-9)
223                 & (dist[None, :] <= self.field.r_eff_max_m + pf.margin_m))
224     eps_radius = pf.spacing_m / 2.0
225     radii = []
226     for row in in_wedge:
227         member = points[row]
228         radii.append(_branch_radius_array(member)
229                     if len(member) else eps_radius)
230     radii = _np.array(radii, dtype=float)
231     current = pf.mec()[1]
232     near = rdist <= self.field.near_radius_m + pf.margin_m
233     values = _np.where(near, self.field.near_radius_m + pf.margin_m,
234                        p_ns * current + p_vis * radii)
235     return float(values.mean())
236
237 def _epr_python(self, reader: Vec2, indices: list[int],
238                representatives: list[int]) -> float: # pragma: no cover
239     """纯 Python 回退（与向量版逐分支同语义，供无 numpy 环境与对拍）。"""
240     pf = self.position
241     points = [pf.cell_center(index) for index in indices]
242     masks = [self._mask(index) for index in indices]
243     eps_radius = pf.spacing_m / 2.0
244     current = pf.mec()[1]
245     half_angle = (self.config.field.bearing_error_half_deg
246                  + self.config.search.bearing_rounding_margin_deg)
247     total = 0.0
248     for rep_index in representatives:
249         rx, ry = pf.cell_center(rep_index)
250         distance = math.hypot(rx - reader[0], ry - reader[1])
251         if distance <= self.field.near_radius_m + pf.margin_m:
252             total += self.field.near_radius_m + pf.margin_m
253             continue
254         reading = math.degrees(math.atan2(ry - reader[1], rx - reader[0]))
255         cover = _COVER_TABLE[_table_index(reading)]
256         bits = _popcount(self._mask(rep_index) & cover)
257         ratio = bits / float(HEADING_BINS)
258         if distance <= self.field.r_eff_min_m:
259             p_range = 1.0
260         elif distance >= self.field.r_eff_max_m:
261             p_range = 0.0
262         else:
263             p_range = ((self.field.r_eff_max_m - distance)
264                        / (self.field.r_eff_max_m - self.field.r_eff_min_m))
265         p_vis = p_range * ratio
266         members = []
267         for point, mask in zip(points, masks):
268             if math.dist(point, reader) > self.field.r_eff_max_m + pf.margin_m:
269                 continue
270             bearing = math.degrees(math.atan2(point[1] - reader[1],

```

```

271         point[0] - reader[0]))
272     if heading_gap(bearing, reading) <= half_angle + 1e-9:
273         members.append(point)
274     if members:
275         mx = sum(p[0] for p in members) / len(members)
276         my = sum(p[1] for p in members) / len(members)
277         radius = max(math.hypot(p[0] - mx, p[1] - my) for p in members)
278     else:
279         radius = eps_radius
280     total += (1.0 - p_vis) * current + p_vis * radius
281     return total / max(len(representatives), 1)
282
283
284 def prob_detect(self, reader: Vec2, size: int = 256) -> float:
285     """在 ``reader`` 处能看见任一可行（位置，朝向）的概率质量（启发式）。
286
287      $r_{\text{eff}} \sim U[1000, 1500]$  解析积分；朝向按可行档位占比计权。
288     只用于选点排序，不参与任何安全判定。
289     """
290     cells = self.sample(size)
291     if not cells:
292         return 0.0
293     total = 0.0
294     span = self.field.r_eff_max_m - self.field.r_eff_min_m
295     for index in cells:
296         gx, gy = self.position.cell_center(index)
297         distance = math.hypot(reader[0] - gx, reader[1] - gy)
298         if distance > self.field.r_eff_max_m:
299             continue
300         if distance <= self.field.r_eff_min_m:
301             p_range = 1.0
302         else:
303             p_range = (self.field.r_eff_max_m - distance) / span
304         if p_range <= 0.0:
305             continue
306         bearing = math.degrees(math.atan2(reader[1] - gy, reader[0] - gx))
307         cover = _COVER_TABLE[_table_index(bearing)]
308         bits = bin(self._mask(index) & cover).count("1")
309         if bits:
310             total += p_range * bits / HEADING_BINS
311     return total / len(cells)
312
313 # ----- 观测更新
314 def _apply_heading(self, station: Vec2, keep: bool,
315                    within_m: float | None = None) -> None:
316     """按可见性约束裁剪朝向档（保守：只删"必然不可能"的档）。
317
318     ``keep=True`` (direction / near)  $\implies$  朝向必须**可能**覆盖 station:
319     保留与方位夹角  $\leq 90^\circ + \text{半档} + \text{查表舍入} + \text{格内角裕量}$  的档。
320
321     ``keep=False`` (no_signal 且整格必在  $r_{\text{eff\_min}}$  内)  $\implies$  朝向必须
322     **确定不覆盖** station: 只保留与方位夹角  $> 90^\circ - \text{半档} - \text{查表舍入}$ 
323     - 格内角裕量的档。
324
325     ``格内角裕量 = asin(margin / max(dist - margin, 1))``: 格内任一点
326     相对格心的方位偏差上界。两个方向都必须计入它，否则会误删真源的
327     朝向档（keep=True 漏计）或保留必然可见的档（keep=False 漏计）。
328     """
329     margin_m = self.position.margin_m
330     doomed: list[int] = []
331     for index in self.position.alive:
332         gx, gy = self.position.cell_center(index)
333         dx, dy = station[0] - gx, station[1] - gy
334         distance = math.hypot(dx, dy)
335         if within_m is not None and distance > within_m:
336             continue
337         bearing = math.degrees(math.atan2(dy, dx))
338         if distance <= margin_m + 1.0:
339             # 格贴测点：格内方位可指向任意方向，任何一侧都不可据此裁剪
340             continue

```

```

341         angular_margin = math.degrees(
342             math.asin(min(0.9999, margin_m / (distance - margin_m))))
343     if keep:
344         updated = self._mask(index) & _cover_mask(
345             bearing, 90.0 + _BIN_HALF_DEG + _GRID_DEG / 2 + angular_margin)
346     else:
347         updated = self._mask(index) & _far_mask(
348             bearing, 90.0 - _BIN_HALF_DEG - _GRID_DEG / 2 - angular_margin)
349     if updated == 0:
350         doomed.append(index)
351         self.masks.pop(index, None)
352     elif updated != _FULL_MASK:
353         self.masks[index] = updated
354     if doomed:
355         self.position.discard(doomed)
356
357     def apply(self, station: Vec2, result: str,
358              svd_deg: float | None = None) -> None:
359         """按读数更新联合可行集（保守，真源（位置，朝向）永不被删）。"""
360         if result == DIRECTION:
361             self.position.apply_measure(station, DIRECTION, svd_deg,
362                                         directional=True)
363             self._apply_heading(station, keep=True)
364         elif result == NEAR:
365             # near 同样要求可见（评测机先判朝向再判距离）
366             self.position.apply_measure(station, NEAR, directional=True)
367             self._apply_heading(station, keep=True)
368         elif result == NO_SIGNAL:
369             # 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"
370             # 时，no_signal 才一定由朝向遮挡解释。格心到格内任一点 ≤ margin，
371             # 故阈值取 r_eff_min - margin；取 + margin 会把 1000-1010m 的模糊带
372             # 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）。
373             self._apply_heading(station, keep=False,
374                                 within_m=max(
375                                     0.0, self.field.r_eff_min_m
376                                     - self.position.margin_m))
377         else:
378             raise ValueError(f"未知测向结果：{result!r}")
379
380         # ----- 诊断
381         def summary(self) -> dict:
382             """联合集规模快照（写入策略 summary，供审计与论文复算）。"""
383             cells = self.sample(256)
384             bins = [bin(self._mask(index)).count("1") for index in cells]
385             return {"cells": len(self.position),
386                     "sampled_mean_heading_bins": (round(sum(bins) / len(bins), 2)
387                                                    if bins else 0.0),
388                     "heading_bins_total": HEADING_BINS}

```

src/mcm26b/cli.py

```

1  """统一命令行入口：mcm26b <子命令>。
2
3  - q1          问题一角域交会求解（P2 已完工）
4  - sim-check   用评测机复现附件计时示例（P0 自检）
5  - run         闭环：发现一定位—清除—退出（--problem 3 全向 / 4 混合源；
6               --self-test 走独立评测机 referee）
7
8  解耦约定：运行时代码零导入评测机；referee/harness 仅在 --self-test 与
9  sim-check 分支内懒加载（测试侧 dev 依赖）。
10 ""
11
12 from __future__ import annotations
13
14 import argparse
15 import json
16 import socket
17 import sys
18 import threading

```

```

19 import time
20 from dataclasses import replace
21 from pathlib import Path
22
23 from .config import ProblemConfig
24 from .geometry.__main__ import main as q1_main
25 from .planner import DirectionalExtension, OmnidirectionalLoop
26 from .planner.bayes_loop import BayesLoop
27 from .planner.seven import SevenBatch
28 from .protocol.client import SimulatorClient
29 from .protocol.log import SessionLog
30 from .protocol.rules import EXIT_USER
31
32 Q3_STRATEGIES = {"grid": OmnidirectionalLoop, "seven": SevenBatch, "bayes": BayesLoop}
33
34
35 def _load_referee():
36     """懒加载独立评测机与测试侧适配层（运行时/正式对接不触发）。"""
37     from .harness import (LocalSimTransport, mixed_scene, random_omni_scene,
38                           referee_spec)
39     from referee import LocalSimulator
40     return (LocalSimulator, LocalSimTransport, mixed_scene, random_omni_scene,
41           referee_spec)
42
43
44 def _referee_missing():
45     print("未安装独立评测机包 referee (mcm26b-referee)。请先 `uv sync` 安装"
46           " dev 依赖, 或 pip install mcm26b-referee; 正式对接官方模拟器无需本包。",
47           file=sys.stderr)
48     return 2
49
50
51 # ----- 一键本地评测
52 def _port_free(port: int) -> bool:
53     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as probe:
54         probe.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
55         try:
56             probe.bind(("127.0.0.1", port))
57             return True
58         except OSError:
59             return False
60
61
62 def _any_free_port() -> int:
63     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as probe:
64         probe.bind(("127.0.0.1", 0))
65         return probe.getsockname()[1]
66
67
68 def _acquire_port(name: str, requested: int) -> int:
69     """端口被占时积极询问：交互终端逐次询问（回车=自动改空闲端口），
70     非交互环境（脚本/CI）直接改用空闲端口并告知，绝不静默失败。"""
71     if _port_free(requested):
72         return requested
73     if sys.stdin.isatty():
74         while True:
75             auto = _any_free_port()
76             try:
77                 answer = input(f"[一键评测] {name}端口 {requested} 已被占用: "
78                               f"回车改用空闲端口 {auto}, 或输入其他端口: ").strip()
79             except (EOFError, KeyboardInterrupt):
80                 return auto
81             if not answer:
82                 return auto
83             if answer.isdigit() and 1024 <= int(answer) <= 65535:
84                 if _port_free(int(answer)):
85                     return int(answer)
86                 print(f"端口 {answer} 同样被占用, 请再试。")
87                 continue
88     print("请输入 1024..65535 的端口号, 或直接回车让评测机自选。")

```

```

89     auto = _any_free_port()
90     print(f"[一键评测] {name}端口 {requested} 已被占用（非交互环境），"
91           f"自动改用空闲端口 {auto}。", file=sys.stderr)
92     return auto
93
94
95 def _start_local_judge(args):
96     """启动独立评测机（机器狗接口 + WebUI 回放）并确认开始一局演练。
97
98     返回 (judge, session, action_port, ui_port)；失败返回 None（已打印原因）。
99     """
100     try:
101         from referee import Referee
102         from referee.server import ActionInterfaceServer
103         from referee.webui import WebUIServer
104     except ImportError:
105         _referee_missing()
106         return None
107     action_port = _acquire_port("机器狗接口", args.judge_port)
108     ui_port = None
109     judge = Referee(action_port=action_port, data_dir=Path(args.judge_data_dir))
110     ActionInterfaceServer(judge)
111     if not args.judge_no_ui:
112         ui_port = _acquire_port("WebUI", args.judge_ui_port)
113         ui = WebUIServer(judge, ui_port)
114         threading.Thread(target=ui.serve_forever, daemon=True,
115                           name="mcm26b-local-judge-ui").start()
116     module = f"q{args.problem}_practice"
117     session = judge.start(module, args.robot_id, seed=args.seed,
118                           countdown_s=args.judge_countdown_s)
119     deadline = time.monotonic() + session.countdown_s + 10
120     while session.phase == "countdown" and time.monotonic() < deadline:
121         time.sleep(0.05)
122     if session.phase != "open":
123         print("[一键评测] 评测机未能开放接口，已中止。", file=sys.stderr)
124         judge.abort()
125         return None
126     print(f"[一键评测] 案例编码 {session.case_code} · {session.module.title}"
127           f" · 接口 http://127.0.0.1:{action_port}")
128     if ui_port is not None:
129         print(f"[一键评测] WebUI http://127.0.0.1:{ui_port}"
130               f"（实时界面与\"回放\"Tab 逐步回看，数据目录 {args.judge_data_dir}）")
131     return judge, session, action_port, ui_port
132
133
134 def _settle_local_judge(judge, timeout_s: float = 3.0) -> None:
135     """等看门狗确认本局结束（/exit 后毫秒级）；仅超时才兜底手工中止。"""
136     deadline = time.monotonic() + timeout_s
137     while judge.session.phase != "finished" and time.monotonic() < deadline:
138         time.sleep(0.05)
139     if judge.session.phase != "finished":
140         judge.abort()
141
142
143 def _finish_local_judge(judge, session, summary: dict) -> None:
144     """等待评测机确认结束，把结束原因、真值统计与回放入口并入 summary。"""
145     _settle_local_judge(judge, timeout_s=10.0)
146     state = judge.snapshot()
147     truth_channels = sorted(s.channel for s in session.sources)
148     cleared = sorted(summary.get("cleared_channels", []))
149     summary["local_judge"] = {
150         "case_code": session.case_code,
151         "module": session.module.title,
152         "end_reason": state.get("end_reason"),
153         "end_reason_text": state.get("end_reason_text"),
154         "virtual_time_s": state.get("virtual_time_s"),
155         "request_count": state.get("request_count"),
156         "truth": {"source_channels": truth_channels,
157                  "source_count": len(truth_channels),
158                  "omni_count": sum(1 for s in session.sources if not s.directional),

```

```

159         "directional_count": sum(1 for s in session.sources if s.directional)),
160         "all_cleared": cleared == truth_channels,
161         "log": state.get("log"),
162     }
163
164
165 def _sim_check(args) -> int:
166     """附件1表2/附件2第10节计时示例: 期望 105/111/194/199 秒。"""
167     try:
168         local_simulator, local_sim_transport, *_ = _load_referee()
169     except ImportError:
170         return _referee_missing()
171     simulator = local_simulator([], robot_id=args.robot_id)
172     client = SimulatorClient(args.robot_id, transport=local_sim_transport(simulator))
173     client.enter()
174     steps = []
175     steps.append(client.measure((300, 400), 1).virtual_time_s) # 移动500m+检测5s
176     steps.append(client.measure((300, 400), 2).virtual_time_s) # 换频道1s+检测5s
177     steps.append(client.clear((300, 0), 3).virtual_time_s) # 移动400m+未发现3s
178     steps.append(client.measure((300, 0), 2).virtual_time_s) # 原地检测频道2
179     info = client.exit()
180     expected = [105, 111, 194, 199]
181     ok = [round(value, 6) for value in steps] == expected \
182         and round(info.virtual_time_s, 6) == 199 and info.reason == EXIT_USER
183     print(json.dumps({"expected": expected, "got": steps,
184                     "final_virtual_time_s": info.virtual_time_s,
185                     "exit_reason": info.reason, "pass": ok},
186                   ensure_ascii=False))
187     return 0 if ok else 1
188
189
190 def _run(args) -> int:
191     """问题3/4闭环。
192
193     三种对接方式: --local-judge 一键本地评测 (自动起 referee 并对接);
194     --self-test 内存回环自检 (不占端口); 默认对接 --base-url 的模拟器
195     (官方或本地 referee)。真值只存在于测试端入口, 策略端仅见响应。
196
197     实机预算: --self-test 时评测机按 --real-budget-s (默认 1800s) 返回
198     remaining_real_duration_s; 其余方式以服务器 /enter 返回为准 (题面
199     程序运行上限 20 分钟, 不可延长)。
200     """
201     config = ProblemConfig()
202     session = replace(config.session,
203                      base_url=args.base_url or config.session.base_url)
204     config = replace(config,
205                    session=session,
206                    search=replace(config.search,
207                                   bearing_rounding_margin_deg=args.bearing_margin_deg,
208                                   persistent_search=args.persistent))
209     if args.persistent:
210         # 历史教训: 一次官方演练在 persistent 复扫下把 98.84% 的虚拟时间
211         # (346170 s) 花在空复扫上。这里给出单轮上界, 明确这是"扫到限时"模式。
212         stations = 49 if args.problem == 4 else 7
213         per_round = config.field.channel_max * stations * (
214             config.cost.measure_s + config.cost.switch_s)
215         print(f"[警告] --persistent: 找全后仍复扫未清除频道直到限时。"
216             f"单轮上界约 {per_round:.0f}s 虚拟时间"
217             f"({config.field.channel_max} 频道 × {stations} 站点 × "
218             f"{config.cost.measure_s + config.cost.switch_s:.0f}s), "
219             f"且会持续多轮; 成绩口径请保持默认 (关闭)。", file=sys.stderr)
220     truth = None
221     transport = None
222     real_budget_s = None
223     judge = judge_session = None
224     if args.local_judge:
225         if args.self_test:
226             print("--local-judge 与 --self-test 不能同时使用。", file=sys.stderr)
227             return 2
228         if args.base_url:

```

```

229         print("--local-judge 忽略 --base-url (接口地址由评测机自动分配)。",
230               file=sys.stderr)
231     started = _start_local_judge(args)
232     if started is None:
233         return 2
234     judge, judge_session, action_port, _ui_port = started
235     config = replace(config, session=replace(
236         config.session, base_url=f"http://127.0.0.1:{action_port}"))
237 elif args.self_test:
238     try:
239         (local_simulator, local_sim_transport,
240          mixed_scene, random_omni_scene, referee_spec) = _load_referee()
241     except ImportError:
242         return _referee_missing()
243     real_budget_s = (args.real_budget_s if args.real_budget_s is not None
244                     else config.session.self_test_real_budget_s)
245     config = replace(config,
246                     field=replace(config.field,
247                                   max_real_duration_s=real_budget_s))
248     if args.problem == 4:
249         sources = mixed_scene(seed=args.seed, count=args.sources,
250                               directional=args.directional, config=config)
251     else:
252         sources = random_omni_scene(seed=args.seed, count=args.sources,
253                                     config=config)
254     truth = sorted(source.channel for source in sources)
255     simulator = local_simulator(sources, robot_id=args.robot_id,
256                                spec=referee_spec(config, max_real_duration_s=real_budget_s,
257                                                    stress_mode=True))
258     transport = local_sim_transport(simulator)
259 elif args.real_budget_s is not None:
260     print("--real-budget-s 仅在 --self-test 下生效; 正式测试以服务器 "
261           "/enter 返回的 remaining_real_duration_s 为准。", file=sys.stderr)
262     log = SessionLog(path=Path(args.log)) if args.log else SessionLog()
263     client = SimulatorClient(args.robot_id, config=config, transport=transport, log=log)
264     if args.problem == 4:
265         strategy = "directional"
266         loop_cls = DirectionalExtension
267     else:
268         strategy = args.strategy
269         loop_cls = Q3_STRATEGIES[strategy]
270     try:
271         summary = loop_cls(client, config).run()
272     finally:
273         if judge is not None:
274             _settle_local_judge(judge)
275     summary["robot_id"] = args.robot_id
276     return _finalize_run(args, summary, strategy, truth=truth, judge=judge,
277                         judge_session=judge_session, real_budget_s=real_budget_s)
278
279
280 def _finalize_run(args, summary, strategy, *, truth, judge, judge_session,
281                  real_budget_s) -> int:
282     """补齐 summary 元信息、核对真值并输出; 返回进程退出码。"""
283     summary["strategy"] = strategy
284     if real_budget_s is not None:
285         summary["self_test_real_budget_s"] = real_budget_s
286     if args.summary_steps is not None and len(summary.get("steps", [])) > args.summary_steps:
287         summary["steps"] = summary["steps"][:args.summary_steps]
288         summary["steps_truncated"] = True
289     all_cleared = None
290     if truth is not None:
291         # 仅测试端核对: 清除集合是否与真值一致。
292         all_cleared = sorted(summary["cleared_channels"]) == truth
293         summary["truth"] = {"source_channels": truth,
294                             "source_count": len(truth),
295                             "all_cleared": all_cleared}
296     if judge is not None:
297         _finish_local_judge(judge, judge_session, summary)
298     all_cleared = summary["local_judge"]["all_cleared"]

```

```

299     print(json.dumps(summary, ensure_ascii=False, indent=2))
300     return 0 if summary["completed"] and (all_cleared is not False) else 1
301
302
303 def main(argv=None) -> int:
304     parser = argparse.ArgumentParser(prog="mcm26b", description=__doc__)
305     parser.add_argument("--config", default="",
306                         help="预留：覆盖 ProblemConfig 的 JSON 文件（参数化入口）")
307     subparsers = parser.add_subparsers(dest="command", required=True)
308
309     q1 = subparsers.add_parser("q1", help="问题一角域交会求解")
310     q1.add_argument("input", type=str)
311     q1.add_argument("-o", "--output", type=str, default=None)
312     q1.add_argument("--rounding-margin-deg", type=float, default=0)
313     q1.set_defaults(func=lambda a: q1_main([a.input] + (["-o", a.output] if a.output else []))
314                    + (["--rounding-margin-deg", str(a.rounding_margin_deg)]
315                      if a.rounding_margin_deg else [])) or 0)
316
317     check = subparsers.add_parser("sim-check", help="自建模拟器复现附件计时示例")
318     check.add_argument("--robot-id", default="2026001")
319     check.set_defaults(func=_sim_check)
320
321     run = subparsers.add_parser("run", help="闭环：发现一定位一清除一退出（问题3/4）")
322     run.add_argument("--robot-id", required=True, help="登录参赛队号，逐字节一致")
323     run.add_argument("--base-url", default=None,
324                     help="模拟器接口地址（默认取配置 http://127.0.0.1:2026）")
325     run.add_argument("--log", default=None, help="会话日志 JSONL 输出路径")
326     run.add_argument("--problem", type=int, choices=(3, 4), default=3,
327                     help="问题3=全向源，问题4=混合定向源（默认 3）")
328     run.add_argument("--strategy", choices=tuple(Q3_STRATEGIES), default="bayes",
329                     help="问题3策略：bayes=粒子滤波+信息增益+滚动调度（默认，"
330                          "自建消融相对 seven 再降 19.5%），seven=七点批量，"
331                          "grid=49点网格基线；仅 --problem 3 生效")
332     run.add_argument("--bearing-margin-deg", type=float, default=0.005,
333                     help="示向度舍入裕量（服务器保留两位小数；默认 0.005°）")
334     run.add_argument("--real-budget-s", type=float, default=None,
335                     help="自检实机预算秒数（默认 1800=30 分钟；仅 --self-test 生效，"
336                          "正式测试以服务器 /enter 返回为准）")
337     run.add_argument("--summary-steps", type=int, default=None,
338                     help="summary 中保留的逐步计时条数上限（默认全部；完整明细在日志）")
339     run.add_argument("--persistent", action="store_true",
340                     help="复扫保险模式：找全证明后继续复扫未清除频道直到限时"
341                          "（默认关闭：虚拟时间即成绩口径，找全即 /exit）")
342     run.add_argument("--local-judge", action="store_true",
343                     help="一键本地评测：自动启动独立评测机 referee（机器狗接口+"
344                          "WebUI 实时/回放），确认开始一局演练并对接跑完整闭环；"
345                          "端口被占时主动询问换端口（非交互环境自动改用空闲端口）")
346     run.add_argument("--judge-port", type=int, default=2026,
347                     help="一键评测的机器狗接口端口（默认 2026；被占时询问/自选）")
348     run.add_argument("--judge-ui-port", type=int, default=8026,
349                     help="一键评测的 WebUI 端口（默认 8026；被占时询问/自选）")
350     run.add_argument("--judge-no-ui", action="store_true",
351                     help="一键评测不启动 WebUI（只保留机器狗接口）")
352     run.add_argument("--judge-countdown-s", type=float, default=5.0,
353                     help="一键评测的确认开始倒计时秒数（题面默认 5 秒，可缩短加速）")
354     run.add_argument("--judge-data-dir", default="output/referee",
355                     help="一键评测的会话日志目录（默认 output/referee，供 WebUI 回放）")
356     run.add_argument("--self-test", action="store_true",
357                     help="走独立评测机 referee 与随机场景（测试端自检，不连官方）")
358     run.add_argument("--seed", type=int, default=20260910, help="自检场景种子")
359     run.add_argument("--sources", type=int, default=None,
360                     help="自检源数量（默认按题面在 10~16 中随机）")
361     run.add_argument("--directional", type=int, default=3,
362                     help="问题4自检场景中定向源数量（--problem 4 生效）")
363     run.set_defaults(func=_run)
364
365     args = parser.parse_args(argv)
366     try:
367         return args.func(args)
368     except NotImplementedError as exc:

```

```

369         print(f"未实现: {exc}", file=sys.stderr)
370         return 2
371
372
373 if __name__ == "__main__":
374     sys.exit(main())

```

src/mcm26b/config.py

```

1  """全项目参数中心：题面常量、协议限制、会话策略与搜索规划参数。
2
3  任何魔法数字只允许在这里出现一次。各模块通过 :class:`ProblemConfig` 读取；
4  命令行需要覆盖时用 ``dataclasses.replace`` 派生新配置，不散落字面量。
5
6  来源标注约定：
7  - ``[题面]`` 出自 issue/B题.pdf、附件1《模拟器使用说明》、附件2《通信接口说明》原文；
8  - ``[工程]`` 本队工程参数，可依演练实测调整，不代表官方规定。
9  """
10
11 from __future__ import annotations
12
13 from dataclasses import dataclass
14
15 Vec2 = tuple[float, float]
16
17
18 @dataclass(frozen=True)
19 class FieldConfig:
20     """题目物理世界与协议硬限制。[题面]"""
21
22     omega_radius_m: float = 1800.0 # 目标区域半径，圆心为原点
23     channel_min: int = 1 # 有效频道范围
24     channel_max: int = 20
25     source_count_min: int = 10 # 一局源总数范围（接口不直接告知）
26     source_count_max: int = 16
27     r_eff_min_m: float = 1000.0 # 有效接收半径下界：no_signal 只能排除该圆盘
28     r_eff_max_m: float = 1500.0 # 有效接收半径上界：direction 隐含距离上界
29     near_radius_m: float = 5.0 # "距离过近"阈值，不返回示向度
30     clear_radius_m: float = 20.0 # 清除半径；与定向朝向无关
31     bearing_error_half_deg: float = 1.0 # 示向度误差幅度（题设模型）
32     bearing_decimals: int = 2 # 返回示向度小数位
33     coord_abs_limit_m: float = 2_000_000.0 # 提交坐标的绝对值上限（超出返回 HTTP 400）
34     directional_half_angle_deg: float = 90.0 # 定向源朝向两侧各90度覆盖（含边界）
35     max_virtual_duration_s: float = 360_000.0 # 虚拟世界限时（100小时）
36     max_real_duration_s: float = 1200.0 # /enter 成功后程序运行上限（20分钟）
37     test_window_s: float = 1500.0 # 确认开始后的测试窗口（25分钟）
38
39
40 @dataclass(frozen=True)
41 class CostConfig:
42     """虚拟世界动作耗时模型。[题面] 附件1表1、附件2第4节。"""
43
44     speed_mps: float = 5.0 # 移动速度；耗时=直线距离/speed
45     measure_s: float = 5.0 # 每次检测动作耗时（无论结果）
46     switch_s: float = 1.0 # 任意两频道间切换耗时，仅由合法 /measure 触发
47     clear_success_s: float = 5.0 # 清除成功耗时 = 光学定位3s + 激光清除2s
48     clear_fail_s: float = 3.0 # 未发现可清除目标 = 仅光学定位3s
49
50
51 @dataclass(frozen=True)
52 class SessionConfig:
53     """会话与网络策略。arena/robot/request 规则为 [题面]，其余为 [工程]。"""
54
55     arena_id: str = "default" # [题面] 固定值
56     base_url: str = "http://127.0.0.1:2026" # [题面] 模拟器默认接口地址
57     robot_id_max_bytes: int = 64 # [题面] UTF-8 长度 1..64
58     request_id_max_bytes: int = 128 # [题面] UTF-8 长度 1..128
59     request_body_max_bytes: int = 65_536 # [题面] 超出返回 HTTP 413
60     connect_timeout_s: float = 5.0 # [工程] 连接超时

```

```

61     read_timeout_s: float = 15.0          # [工程] 读超时; 覆盖"服务器已处理但响应丢失"
62     max_attempts: int = 3                 # [工程] 单动作最多尝试次数 (含首次)
63     retry_backoff_s: float = 0.25         # [工程] 重试退避基数, 按  $2^n$  放大
64     reserve_exit_s: float = 60.0          # [工程] 预留退出与故障恢复的现实秒数
65     real_rtt_p95_s: float = 1.0           # [工程] 现实单请求高分位估计, 预算核算用
66     # [工程] 自检/演练实机预算: 题面正式测试程序运行上限 1200s 不可延长;
67     # 自检经 --real-budget-s 覆盖 LocalSimulator 返回的 remaining_real_duration_s,
68     # 默认 1800s (30 分钟实机), 用于验证"限时内持续运行直到找全"的行为。
69     self_test_real_budget_s: float = 1800.0
70
71
72 @dataclass(frozen=True)
73 class SearchConfig:
74     """搜索与覆盖规划参数。[工程] 推导依据见 docs/B题建模分析与实施计划.md。"""
75
76     grid_step_m: float = 700.0            # P4 覆盖网格步长; 单元对角  $700 \times \sqrt{2} \approx 989.95 \leq 1000$ 
77     grid_half_cells: int = 3              #  $i, j \in [-3, 3]$ , 49 点覆盖  $[-2100, 2100]^2 \supseteq \Omega$ 
78     fallback_clear_grid_m: float = 20.0   # P5 后备清除网格;  $10 \times \sqrt{2} \leq 20$ 
79     # 舍入裕量: 服务器对示向度保留两位小数, 引入  $\leq 0.005^\circ$  偏差 [题面];
80     # 不加裕量时合法真源可能被角域裁掉, 故库默认与 CLI run 默认一致取 0.005 [工程]。
81     bearing_rounding_margin_deg: float = 0.005
82     # ---- P3/P4 在线定位与闭环参数 [工程] ----
83     polygon_sides: int = 192              # 圆盘外切多边形边数;  $\Omega$  近似误差  $1800(1 - \cos(\pi/192)) \approx 0.24\text{m}$ 
84     witness_spacing_m: float = 60.0       # 最坏后验评估的边界采样间距; 选定后再按 1/4 加密复核
85     candidate_rings: tuple = (0.0, 0.4, 0.75, 1.0) # 安全接收盘内候选点的半径比例
86     candidate_angles: int = 12            # 每环候选方向数
87     clear_margin_m: float = 0.5           # 清除证书的额外工程余量 (多边形近似误差之外)
88     max_localize_steps: int = 8           # 单频道定位阶段允许的追加测量上限
89     max_anomalies_per_channel: int = 3    # 单频道异常记录上限, 防止异常刷屏
90     # ---- P5 定向扩展参数 [工程] ----
91     max_blocked_probes: int = 4           # 定位单轮内被朝向遮挡的探点预算
92     # (首次选点+对跖复核+2 次环探; 达限转后备清除)
93     fallback_max_cells: int = 400        # 后备清除格数硬顶; 超过则不盲扫 (宽候选集继续定位)
94     # ---- 覆盖计划 (保证层) [工程, 2026-09-12 架构优化 P0-2 / S2 硬化] ----
95     # 第四问发现点集:
96     #   square49      旧的充分条件表 (700m 方格, 巡回 36.4km);
97     #   q4_mincover_v1 75m/24 档贪心最小集 (31 点, 巡回 22.9km) ——细网格复检
98     #   在 25m/48 档有 125 个 (p,h) 缺口, 是采样过拟合解;
99     #   q4_mincover_v2 **设计裕量**最小集 (31 点,  $r_{\text{design}}=965\text{m}$ 、前向角裕量  $5^\circ$ ,
100    #   搜索分辨率 50m/48 档 + 证书分辨率精细剪枝),
101    #   15m/72 档 (325.6 万 (p,h) 缺口 0, 最坏命中 990m, 巡回 23.9km。
102    # 运行期按同一采样语义复核, 未通过即拒绝启动。
103    q4_coverage_plan: str = "q4_mincover_v2"
104    verify_coverage_plan: bool = True      # 运行期抽样校验 (进程内缓存一次)
105    # ---- 第四问定向源联合信念 [工程, 2026-09-12 架构优化 P0-3] ----
106    # 为已发现的定向频道维护 (位置  $\times$  朝向) 联合可行集: 用可见性概率选点,
107    # 取代固定  $45^\circ$  阶梯探点; 联合集空即给出缺席证明。安全性仍由凸/网格
108    # 清除证书与覆盖证明负责, 本开关只影响选点顺序。
109    directional_joint_belief: bool = True
110    # ---- 第四问定向源探点值函数 [工程, 2026-09-12 图论与 DP 方案 S3/D1] ----
111    # 旧评分  $\text{prob\_detect} / (\text{移动} + 6)$  只问"这点看不看得见源", 不问"测完剩多少
112    # 不确定度", 会反复选中同一个近处被遮挡点。D1 口径: 先按可见概率过滤
113    # (只保留  $\text{prob\_detect} \geq \text{ratio} \times \text{最大值的候选}$ ), 再最小化
114    #  $E[\text{后验 MEC 半径} | q] + \lambda \cdot (\text{当前位置} \rightarrow q \rightarrow \text{MEC 圆心 的移动秒数} + \text{检测})$ 
115    # 9 场景 core 配对 (修正联合信念朝向规则之后):
116    #  $\lambda=0.05$ : 后备 6.8 次 / T 9740 s (值函数主导, 为信息多绕路)
117    #  $\lambda=5$ : 后备 4.3 次 / T 8604 s (移动项主导, 可见性过滤 + 就近抵近)
118    # 仅移动项 (nearest_visible): 后备 4.3 次 / T 8602 s, 墙钟最低
119    # 相对旧 visibility 口径 (后备 125.6 次、T 9225 s) 三者都更好, 且  $\lambda$  越大
120    # 越优说明本题"看得见 + 少走路"比"信息量最大"更重要, 故默认取极限口径。
121    probe_score_mode: str = "nearest_visible" # visibility | expected_radius | nearest_visible
122    probe_lambda_move_s: float = 5.0         # expected_radius 口径下的移动折算
123    # 可见概率过滤: 只保留  $\text{prob\_detect} \geq \text{ratio} \times \text{最大值的候选}$ 再比期望半径,
124    # 避免"期望上有点信息、实际必被朝向遮挡"的探点。
125    probe_visibility_ratio: float = 0.7
126    probe_exclude_blocked: bool = True      # 已测过且被遮挡的探点不再重复选取
127    approach_fractions: tuple = ()         # D1 抵近点插值比例 (0=圆心); 默认关闭
128    # ---- 后备清除路径优化 [工程, 2026-09-12 S4/D3] ----
129    # 实测反例: 后备清除在命中真源时立即结束, "整条走路路径最短"并不是正确
130    # 目标——Or-opt 会把真源推到序列更后面。9 场景配对: 最近邻 125.7 次失败

```

```

131 # vs Or-opt 161.4 次 (T 8676 → 8940 s)。默认关闭, 保留代码与开关。
132 fallback_optimize_tour: bool = False
133 # ---- 持续搜索 [工程, 2026-09-11 用户决策修订] ----
134 # 虚拟时间即评测成绩口径 (表1: 平均定位清除时间): 找全证明达成后
135 # 立即 /exit, 不复扫。复扫保险模式 (防缺席证明假设偏差) 保留, 经
136 # `mcm26b run --persistent` 显式开启: 只复扫未清除频道, 直到清除数
137 # 达 16、实机预算耗尽或虚拟安全余量不足。
138 persistent_search: bool = False
139 virtual_reserve_s: float = 10_000.0 # 虚拟时长安全余量: 剩余不足时停止发起动作,
140 # 避免触及 100 小时上限被模拟器强制结束
141
142
143 @dataclass(frozen=True)
144 class BayesConfig:
145     """P4b 贝叶斯网格滤波策略参数 (问题3新策略 bayes)。[工程]
146
147     网格粒子滤波的严格性依赖 Voronoi 裕量: 真实源所在格心与源的距离
148      $\leq \sqrt{2}/2 * \text{particle\_spacing\_m}$ , 所有 kill 判定都内缩该裕量,
149     因此幸存集永远包含真源 (定理见 docs/algo-3-bayes.md §2)。
150     """
151
152     particle_spacing_m: float = 14.65 # 网格间距; Voronoi 裕量  $\approx 10.35\text{m}$ ;
153     # 全域约 4.7 万格/频道, 查询走粗格索引
154     # ---- 信息增益评分 [工程, Optuna TPE 调参回填 2026-09-11] ----
155     # 原调参记录见 output/q3/bayes-tuning/; 当前代码的配对验收记录见
156     # output/q3/optimization-20260911/, 随机分布与信息评分仅为工程假设。
157     score_mode: str = "entropy" # entropy=期望 log 幸存占比 (信息增益) |
158     # radius=期望后验半径 (直指证书)
159     score_subsample: int = 800 # 期望幸存熵估计的粒子子采样规模
160     mc_representatives: int = 7 # 假设真值代表点数 (均匀抽取幸存格)
161     lambda_move_s: float = 0.05 # 选点评分中移动时间的折算系数 (分值/秒);
162     # 2026-09-12 配对复跑: 0.05 优于 0.0105 (20 场均值 -2.8%),
163     # entropy 口径下信息项为  $O(1)$  对数值,  $\lambda$  须小型但不可过小
164     # ---- 路线感知探点 [工程, 2026-09-12 P2] ----
165     # 旧评分只看 current→q + q→MEC, 容易选到“信息增益略高但要多绕几百米”
166     # 的探点。开启后, 在候选评分中加入 q 替换当前位置后对当前任务路线代价
167     # 的增量; A/B 以冻结基准集配对验收。
168     # 2026-09-12 冻结基准集复跑 (18 场景): 关闭 3193.1 s vs 开启 3210.3 s,
169     # 移动 12339.8 m vs 12446.8 m —— 权重  $1e-3$  的路线项未带来净收益, 默认关闭。
170     route_aware_probe: bool = False
171     route_aware_insert_weight: float = 0.001 # 路线代价增量的折算系数 (小权重, 仅微调)
172     route_aware_clear_weight: float = 0.0 # 额外 q→MEC 清除接近项 (基础移动项已含)
173     wide_rings: tuple = (1.0, 1.5, 2.0) # 无保证接收盘时的探索环 ( $\times$  PF MEC 半径)
174     # ---- 站点与覆盖 [工程] ----
175     station_ring_radius_m: float = 1159.7 # 七点环半径; 构造时检验全向覆盖解析上界
176     # ---- 发现-定位融合巡回 [工程, 2026-09-12 P0-1, 实测无效, 默认关闭] ----
177     # 设想: 站点扫描频顺带取第二/第三条方位线, 取代专用探点。
178     # 实测 (output/suite/q3-fuse-ab*, 20 场景配对, 2026-09-12):
179     # 仅追加站点方位线: 3454.6s vs 3426.9s (+0.8%, 检测 +10.8 次, 位移仅 -188m);
180     # 追加 + 抑制专用探点: 4989.0s vs 3426.9s (+45.6%, 20/20 场景更慢)。
181     # 原因: 七站环上只有 2-3 个站能对同一源取得有效方位线, 且站间交会角
182     # 不足以保证 20m 证书, 专用信息增益探点仍然必要; 抑制探点只会白等巡回。
183     # 代码保留以便复现与后续研究, 成绩口径必须保持 False。
184     fuse_localize_into_tour: bool = False
185     # ---- 任务排序 [工程] ----
186     two_opt_enabled: bool = True # 开放路径 2-opt 改进 (最近邻打底)
187     two_opt_passes: int = 4 # 2-opt 最大改进轮数
188     # ---- 精确任务序 (Held-Karp, 含换频状态) [工程, 2026-09-12 S5/T1] ----
189     # 任务表典型 6-12 项:  $|T| \leq \text{exact\_order\_limit}$  时用精确 DP (状态=已访问集合
190     #  $\times$  结束时测向频道), 换频代价精确计入; 否则回退  $NN + 2\text{-opt}$ 。0 表示禁用。
191     # 实测 (18 场景): 精确 DP 3203.2 s vs 2-opt 3193.1 s —— 逐位最优的"当前
192     # 任务表"在滚动时域下反而略差 (贪心排序的次优性起到探索作用), 默认 0;
193     # 代码与对拍测试保留, 可随时开启复核。
194     exact_order_limit: int = 0
195     # ---- PF 清除证书 [工程] ----
196     grid_cert_enabled: bool = True # 允许用幸存集 MEC 颁发清除证书 (凸证书之外)
197     opportunistic_measure: bool = True # 已停靠位置顺路交会; 仍逐频道计费
198     # 预测后验半径至少减少 (1-ratio) 才补测: 0.5 优于 0.65 (18 场景 3205.4 vs 3210.3 s)
199     opportunistic_radius_ratio: float = 0.5
200

```

```

201
202 @dataclass(frozen=True)
203 class ProblemConfig:
204     """聚合配置。各模块函数的第一个默认参数，便于单测替换。"""
205
206     field: FieldConfig = FieldConfig()
207     cost: CostConfig = CostConfig()
208     session: SessionConfig = SessionConfig()
209     search: SearchConfig = SearchConfig()
210     bayes: BayesConfig = BayesConfig()
211
212
213 DEFAULT_CONFIG = ProblemConfig()

```

src/mcm26b/coverage/__init__.py

```

1  """P4 全域覆盖：49 点网格与逐频道覆盖记录。
2
3  覆盖基线（推导见 docs/B题建模分析与实施计划.md §4）：
4   $S = \{(700i, 700j) : i, j \in [-3, 3]\}$ ,  $|S| = 49$ , 覆盖  $[-2100, 2100]^2 \supseteq \Omega$ ,
5  且允许访问目标圆域外（边界朝外发射的定向源不漏）。
6  单元内任意点到 4 顶点距离  $\leq 700 \cdot \sqrt{2} \approx 989.95 \leq 1000 \leq$  最小接收半径。
7
8  验收标准：
9  - "路过某点"不算"测过某频道"：必须按（频道，点）二维记录；
10 - 已清除/已定位频道可跳过剩余点，但跳过原因必须留痕；
11 - 自适应跳点（P6）必须保留逐频道覆盖证据，否则不得启用。
12 """
13
14 from __future__ import annotations
15
16 from dataclasses import dataclass, field
17
18 from ..config import ProblemConfig, Vec2
19
20
21 def coverage_points(config: ProblemConfig = ProblemConfig()) -> list[Vec2]:
22     """49 点覆盖网格生成（纯几何，可独立测试）。"""
23     step = config.search.grid_step_m
24     half = config.search.grid_half_cells
25     return [(step * i, step * j)
26             for j in range(half, -half - 1, -1)
27             for i in range(-half, half + 1)]
28
29
30 @dataclass
31 class CoverageLedger:
32     """逐频道覆盖账本：（频道，网格点）-> 是否已测。"""
33
34     config: ProblemConfig = field(default_factory=ProblemConfig)
35     points: list[Vec2] = field(default_factory=coverage_points)
36     measured: set = field(default_factory=set) # {(channel, point_index)}
37     skipped: dict = field(default_factory=dict) # channel -> 原因
38
39     def mark_measured(self, channel: int, point_index: int) -> None:
40         self.measured.add((channel, point_index))
41
42     def measured_at(self, channel: int, point_index: int) -> bool:
43         return (channel, point_index) in self.measured
44
45     def skip_channel(self, channel: int, reason: str) -> None:
46         """记录整频道跳过原因（如 already_cleared / located_pending_clear）。"""
47         self.skipped[channel] = reason
48
49     def pending_points(self, channel: int) -> list[int]:
50         """该频道尚未测量的网格点下标。扫描调度策略未实现（P4）。"""
51         return [index for index in range(len(self.points))
52                 if not self.measured_at(channel, index)]
53

```

```

54     def coverage_complete(self, channel: int) -> bool:
55         """该频道覆盖是否完成（含显式跳过）。"""
56         return channel in self.skipped or not self.pending_points(channel)

```

src/mcm26b/evaluation/__init__.py

```

1  """评估与统计：从自身动作日志离线复算虚拟时刻与统计量。
2
3  "演练统计"验收要求：总时间与分项耗时可独立复算 [题面/计划 §6]。
4  recompute_virtual_time 是回放测试与本模块共用的唯一复算实现，
5  客户端日志格式为 SessionLog 中 accepted 事件的 {path, request, response}。
6  """
7
8  from __future__ import annotations
9
10 from ..config import CostConfig
11 from ..protocol.rules import CLEAR_SUCCESS
12 from ..protocol.timing import clear_cost, measure_cost
13
14
15 def recompute_virtual_time(events: list[dict],
16                             cost: CostConfig = CostConfig()) -> dict:
17     """从动作日志复算逐步虚拟时刻。
18
19     events 每项形如 {path, request, response} (accepted=true)。
20     返回逐步表与最终时刻；任何一步与服务器响应偏差超过容差记入 mismatches。
21     """
22     position = (0.0, 0.0)
23     channel = 1
24     virtual = 0.0
25     virtual_us = 0
26     steps, mismatches = [], []
27     for index, event in enumerate(events):
28         path, request = event["path"], event["request"]
29         response = event["response"]
30         if response.get("accepted") is not True:
31             mismatches.append({"index": index, "issue": "响应未被接受"})
32             continue
33         if path == "/enter":
34             position, channel, virtual = (0.0, 0.0), 1, response["virtual_time_s"]
35             virtual_us = round(virtual * 1_000_000)
36         elif path == "/measure":
37             pos = (request["position"]["x"], request["position"]["y"])
38             predicted = measure_cost(position, pos, channel, request["channel"], cost)
39             position, channel = pos, request["channel"]
40             virtual_us += predicted.total_us
41             virtual = virtual_us / 1_000_000
42         elif path == "/clear":
43             pos = (request["position"]["x"], request["position"]["y"])
44             success = response.get("clear_result") == CLEAR_SUCCESS
45             predicted = clear_cost(position, pos, success, cost)
46             position = pos
47             virtual_us += predicted.total_us
48             virtual = virtual_us / 1_000_000
49         elif path == "/exit":
50             pass # /exit 不推进虚拟时钟 [题面]
51         else:
52             mismatches.append({"index": index, "issue": f"未知路径 {path}"})
53             continue
54         delta = response["virtual_time_s"] - virtual
55         if abs(delta) > 1e-6:
56             mismatches.append({"index": index, "path": path,
57                                 "recomputed_s": virtual,
58                                 "recorded_s": response["virtual_time_s"],
59                                 "gap_s": delta})
60         steps.append({"index": index, "path": path,
61                       "predicted_s": round(virtual, 9),
62                       "recorded_s": response["virtual_time_s"]})
63     return {"steps": steps, "final_virtual_time_s": virtual,

```

```

64         "mismatches": mismatches}
65
66
67 def summarize(events: list[dict]) -> dict:
68     """动作计数摘要：检测/清除/移动距离等基础统计。"""
69     position = (0.0, 0.0)
70     measures, clears, cleared, distance_m = 0, 0, 0, 0.0
71     for event in events:
72         request = event["request"]
73         if event["path"] in ("/measure", "/clear"):
74             pos = (request["position"]["x"], request["position"]["y"])
75             distance_m += ((pos[0] - position[0]) ** 2
76                           + (pos[1] - position[1]) ** 2) ** 0.5
77             position = pos
78         if event["path"] == "/measure":
79             measures += 1
80         elif event["path"] == "/clear":
81             clears += 1
82             if event["response"].get("clear_result") == CLEAR_SUCCESS:
83                 cleared += 1
84     return {"measures": measures, "clears": clears, "cleared": cleared,
85           "distance_m": distance_m}

```

src/mcm26b/geometry/__init__.py

```

1  """P2 几何内核：B题问题一有界误差方位交会定位。仅用标准库，已完工。
2
3  问题二首次测向安全候选盘证书一并放在本包（planning 模块）。
4  """
5
6  from .localization import Measurement, solve, smallest_enclosing_circle
7  from .planning import first_bearing_candidate_disk
8
9  __all__ = ["Measurement", "solve", "smallest_enclosing_circle", "first_bearing_candidate_disk"]

```

src/mcm26b/geometry/__main__.py

```

1  """python -m mcm26b.geometry observations.json [-o result.json]"""
2
3  import argparse
4  import json
5  from dataclasses import replace
6  from math import isfinite
7  from pathlib import Path
8  from .localization import Measurement, solve
9
10
11 def main(argv=None):
12     parser = argparse.ArgumentParser(description="B题问题一：角域交、直径与包围圆")
13     parser.add_argument("input", type=Path, help="包含measurements数组的JSON文件")
14     parser.add_argument("-o", "--output", type=Path)
15     parser.add_argument("--rounding-margin-deg", type=float, default=0,
16                         help="额外误差半角，默认0；可显式设0.005进行舍入敏感性分析")
17     args = parser.parse_args(argv)
18     try:
19         source = json.loads(args.input.read_text(encoding="utf-8-sig"))
20         measurements = [Measurement(**row) for row in source["measurements"]]
21         if not isfinite(args.rounding_margin_deg) or args.rounding_margin_deg < 0:
22             raise ValueError("额外误差半角必须为有限非负数")
23         measurements = [replace(m, half_angle_deg=m.half_angle_deg+args.rounding_margin_deg
24                                for m in measurements)]
25         result = solve(measurements)
26         result["error_policy"] = {
27             "additional_half_angle_deg": args.rounding_margin_deg,
28             "effective_half_angles_deg": [m.half_angle_deg for m in measurements],
29         }
30     except (ValueError, TypeError, KeyError, OSError) as exc:
31         parser.error(str(exc))

```

```

32     text = json.dumps(result, ensure_ascii=False, indent=2, allow_nan=False)
33     if args.output:
34         args.output.parent.mkdir(parents=True, exist_ok=True)
35         args.output.write_text(text + "\n", encoding="utf-8")
36     print(text)
37
38
39 if __name__ == "__main__":
40     main()

```

src/mcm26b/geometry/localization.py

```

1  """Intersect forward bearing wedges without an artificial bounding box.
2
3  All calculations are floating point with normalized station coordinates.
4  The reference algorithm deliberately favors auditability over asymptotic speed.
5  Near-parallel plane pairs are intersected instead of skipped (only exactly
6  parallel pairs are discarded), and the feasibility tolerance scales with the
7  candidate vertex magnitude, so legitimate far-away vertices of long thin
8  regions are never silently dropped.
9  """
10
11 from dataclasses import dataclass
12 from itertools import combinations
13 from math import cos, sin, radians, hypot, isfinite
14 from random import Random
15
16 Point = tuple[float, float]
17 EPS = 1e-9
18 PARALLEL_EPS = 1e-12
19
20
21 @dataclass(frozen=True)
22 class Measurement:
23     x: float
24     y: float
25     bearing_deg: float
26     half_angle_deg: float = 1.0
27
28     def __post_init__(self):
29         values = (self.x, self.y, self.bearing_deg, self.half_angle_deg)
30         if any(isinstance(v, bool) or not isinstance(v, (int, float))
31             or not isfinite(v) for v in values):
32             raise ValueError("坐标、示向度和误差半角必须是有限数值")
33         if not 0 <= self.half_angle_deg < 90:
34             raise ValueError("误差半角必须在 [0, 90) 度；题设为1度")
35
36
37 def cross(a: Point, b: Point) -> float:
38     return a[0] * b[1] - a[1] * b[0]
39
40
41 def distance(a: Point, b: Point) -> float:
42     return hypot(a[0] - b[0], a[1] - b[1])
43
44
45 def halfplanes(m: Measurement) -> list[tuple[float, float, float]]:
46     """Return a*x+b*y<=c with unit normals, including the forward constraint."""
47     theta = m.bearing_deg % 360
48     lower, upper = radians(theta - m.half_angle_deg), radians(theta + m.half_angle_deg)
49     middle = radians(theta)
50     normals = [(sin(lower), -cos(lower)), (-sin(upper), cos(upper)),
51               (-cos(middle), -sin(middle))]
52     return [(a, b, a * m.x + b * m.y) for a, b in normals]
53
54
55 def convex_hull(points: list[Point], eps: float = EPS) -> list[Point]:
56     """Monotone chain, CCW, no repeated final vertex; retain segment endpoints."""
57     ordered = sorted(points)

```

```

58     unique = []
59     for p in ordered:
60         if not unique or distance(p, unique[-1]) > eps:
61             unique.append(p)
62     if len(unique) <= 2:
63         return unique
64
65     def chain(seq):
66         out = []
67         for p in seq:
68             while len(out) >= 2:
69                 a = (out[-1][0] - out[-2][0], out[-1][1] - out[-2][1])
70                 b = (p[0] - out[-1][0], p[1] - out[-1][1])
71                 # Distance-to-line tolerance, not an unscaled area tolerance.
72                 if cross(a, b) > eps * max(hypot(*a), hypot(*b), eps):
73                     break
74                 out.pop()
75             out.append(p)
76         return out
77
78     return chain(unique)[-1] + chain(reversed(unique))[-1]
79
80
81 def diameter(points: list[Point]):
82     if not points:
83         raise ValueError("空点集没有本程序定义下的直径")
84     a = b = points[0]
85     value = 0.0
86     for p, q in combinations(points, 2):
87         d = distance(p, q)
88         if d > value:
89             value, a, b = d, p, q
90     return value, (a, b)
91
92
93 def _circumcircle(a: Point, b: Point, c: Point):
94     u = (b[0] - a[0], b[1] - a[1])
95     v = (c[0] - a[0], c[1] - a[1])
96     det = 2 * cross(u, v)
97     if abs(det) <= PARALLEL_EPS * max(hypot(*u) * hypot(*v), 1e-300):
98         return None
99     uu, vv = u[0] ** 2 + u[1] ** 2, v[0] ** 2 + v[1] ** 2
100     center = (a[0] + (uu * v[1] - vv * u[1]) / det,
101              a[1] + (u[0] * vv - v[0] * uu) / det)
102     return center, distance(center, a)
103
104
105 def _enclosing_circle_reference(points: list[Point], eps: float = EPS):
106     """Enumerate all 1/2/3-point supports; O(m^4), appropriate for small m.
107
108     Radius is recomputed against every vertex to preserve numerical containment.
109     Coordinates should be normalized when there are very large offsets.
110     """
111     if not points:
112         raise ValueError("空集不计算最小包围圆")
113     xs, ys = [p[0] for p in points], [p[1] for p in points]
114     best_c = ((min(xs) + max(xs)) / 2, (min(ys) + max(ys)) / 2)
115     best_r = max(distance(best_c, p) for p in points)
116
117     def consider(center, radius):
118         nonlocal best_c, best_r
119         if radius > best_r + eps:
120             return
121         actual = max(distance(center, p) for p in points)
122         if actual <= radius + eps and actual < best_r:
123             best_c, best_r = center, actual
124
125     for p in points:
126         consider(p, 0.0)
127     for a, b in combinations(points, 2):

```

```

128         consider(((a[0] + b[0]) / 2, (a[1] + b[1]) / 2), distance(a, b) / 2)
129     for a, b, c in combinations(points, 3):
130         circle = _circumcircle(a, b, c)
131         if circle is not None:
132             consider(*circle)
133     return best_c, best_r
134
135
136 def _circle_covers(circle, point: Point, tolerance: float) -> bool:
137     """Whether the circle already covers point; None means no circle yet."""
138     if circle is None:
139         return False
140     return distance(circle[0], point) <= circle[1] + tolerance
141
142
143 def smallest_enclosing_circle(points: list[Point], eps: float = EPS):
144     """Welzl's randomized incremental algorithm, using an explicit call stack.
145
146     The fixed shuffle makes results reproducible. Expected linear work refers
147     to randomized input order, not a worst-case guarantee for this fixed seed.
148     Normalize internally so this public helper also handles translated inputs.
149     The exhaustive reference is retained for independent regression comparisons.
150     """
151     if not points:
152         raise ValueError("空集不计算最小包围圆")
153     if not isfinite(eps) or eps <= 0:
154         raise ValueError("eps必须为有限正数")
155     if any(len(p) != 2 or any(isinstance(x, bool) or not isinstance(x, (int, float))
156                               or not isfinite(x) for x in p) for p in points):
157         raise ValueError("点集必须由有限二维坐标组成")
158     origin = points[0]
159     scale = max(distance(origin, p) for p in points)
160     if scale == 0:
161         return tuple(origin), 0.0
162     local = [(p[0]-origin[0])/scale, (p[1]-origin[1])/scale) for p in points]
163     tolerance = min(eps/scale, EPS)
164     Random(20260910).shuffle(local)
165     # Frame = (remaining prefix length, forced boundary, returning from child).
166     stack = [(len(local), (), False)]
167     circle = None
168     while stack:
169         n, boundary, returning = stack.pop()
170         if n == 0 or len(boundary) == 3:
171             # Forced support points must lie ON the boundary. For three points
172             # use their circumcircle, even when their unconstrained MEC is smaller.
173             circle = _circumcircle(*boundary) if len(boundary) == 3 else None
174             if circle is None and boundary:
175                 circle = _enclosing_circle_reference(list(boundary), tolerance)
176         elif not returning:
177             stack.append((n, boundary, True))
178             stack.append((n-1, boundary, False))
179         elif not _circle_covers(circle, local[n-1], tolerance):
180             stack.append((n-1, boundary + (local[n-1],), False))
181     assert circle is not None
182     center = (origin[0]+circle[0][0]*scale, origin[1]+circle[0][1]*scale)
183     # Recompute after transforming back: returned circle contains all input points
184     # to distance() precision, including any roundoff from the large translation.
185     return center, max(distance(center, p) for p in points)
186
187
188 def solve(measurements: list[Measurement], eps: float = EPS) -> dict:
189     """Solve Q1's pure angular intersection (no disk/range prior added).
190
191     Nonempty intersections of <180-degree forward wedges are pointed polyhedra,
192     so they possess a vertex, even if unbounded. Feasibility follows from vertex
193     enumeration; unboundedness follows from a nonzero recession direction.
194     Empty diameter is null, unbounded diameter is null with diameter_infinite=true.
195
196     Near-parallel plane pairs are intersected rather than skipped: their
197     intersection can be a legitimate vertex of a very long thin region

```

```

198 (e.g. stations 1cm apart with bearings 1 and 358.9999999999 deg give a
199 vertex ~5.7e10 m away). Only exactly-parallel pairs (det == 0) have no
200 intersection. The feasibility tolerance scales with the vertex magnitude,
201 since absolute residuals grow with coordinates.
202 """
203 if not isfinite(eps) or eps <= 0:
204     raise ValueError("eps必须为有限正数")
205 if not measurements:
206     return {"status": "unbounded", "vertices": [], "diameter_m": None,
207            "diameter_infinite": True, "recession_direction": [1.0, 0.0],
208            "message": "没有观测, 整个平面均可行"}
209 if len(measurements) > 100:
210     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")
211 ox = sum(m.x for m in measurements) / len(measurements)
212 oy = sum(m.y for m in measurements) / len(measurements)
213 scale = max(1.0, max(hypot(m.x - ox, m.y - oy) for m in measurements))
214 local = [Measurement((m.x - ox) / scale, (m.y - oy) / scale,
215                    m.bearing_deg, m.half_angle_deg) for m in measurements]
216 planes = [h for m in local for h in halfplanes(m)]
217
218 def inside(p):
219     tol = eps * max(1.0, abs(p[0]), abs(p[1]))
220     return all(a * p[0] + b * p[1] <= c + tol for a, b, c in planes)
221
222 feasible_vertices = []
223 for (a, b, c), (d, e, f) in combinations(planes, 2):
224     det = a * e - b * d
225     if det == 0.0:
226         continue # 精确平行: 无交点 (近平行对的交点是合法远端顶点, 保留)
227     p = ((c * e - b * f) / det, (a * f - c * d) / det)
228     if isfinite(p[0]) and isfinite(p[1]) and inside(p):
229         feasible_vertices.append(p)
230
231 if not feasible_vertices:
232     return {"status": "empty", "vertices": [], "diameter_m": None,
233            "diameter_infinite": False, "tolerance_m": eps * scale,
234            "message": "测向角域没有公共点; 检查频道、数据及误差界"}
235
236 # A nontrivial 2D recession cone has a ray tangent to an input halfplane.
237 # Use a much tighter angular threshold than the positional feasibility epsilon.
238 ray = None
239 for a, b, _ in planes:
240     for direction in ((b, -a), (-b, a)):
241         if all(d * direction[0] + e * direction[1] <= 2e-14
242              for d, e, _ in planes):
243             ray = direction
244             break
245 if ray is not None:
246     break
247
248 hull = convex_hull(feasible_vertices, eps)
249
250 def world(p):
251     return [ox + p[0] * scale, oy + p[1] * scale]
252
253 result = {"status": "unbounded" if ray else
254          ("point" if len(hull) == 1 else "segment" if len(hull) == 2 else "polygon"),
255          "vertices": [world(p) for p in hull], "tolerance_m": eps * scale,
256          "diameter_infinite": ray is not None}
257 if ray is not None:
258     result.update(diameter_m=None, recession_direction=list(ray),
259                  message="存在非零衰退方向, 区域无界, 直径为无穷大")
260 return result
261
262 d, pair = diameter(hull)
263 midpoint = ((pair[0][0] + pair[1][0]) / 2, (pair[0][1] + pair[1][1]) / 2)
264 distances = [distance(midpoint, p) for p in hull]
265 witness_index = max(range(len(hull)), key=distances.__getitem__)
266 max_distance = distances[witness_index]
267 center, radius = smallest_enclosing_circle(hull, eps)

```

```

268     area = abs(sum(cross(hull[i], hull[(i + 1) % len(hull)]))
269                     for i in range(len(hull)))) / 2
270     result.update(diameter_m=d * scale, diameter_pair=[world(p) for p in pair],
271                  area_m2=area * scale ** 2,
272                  diameter_circle={"center": world(midpoint), "radius_m": d * scale / 2,
273                                   "max_vertex_distance_m": max_distance * scale,
274                                   "covers": max_distance <= d / 2 + eps * 4,
275                                   "coverage_tolerance_m": 4 * eps * scale,
276                                   "max_excess_m": (max_distance-d/2) * scale,
277                                   "outside_vertex_indices": [i for i, value in enumerate(distances)
278                                                             if value > d/2 + 4*eps],
279                                   "witness_vertex": world(hull[witness_index])
280                                                         if max_distance > d/2 + 4*eps else None},
281                  minimum_enclosing_circle={"center": world(center), "radius_m": radius * scale},
282                  guaranteed_clearable_at_mec_center=radius * scale <= 20 - eps * scale * 4,
283                  clearance_certificate={"model": "angular_intersection_only",
284                                          "clear_radius_m": 20.0,
285                                          "numerical_margin_m": 4*eps*scale,
286                                          "radius_with_margin_m": (radius+4*eps)*scale,
287                                          "remaining_margin_m": 20-(radius+4*eps)*scale,
288                                          "center": world(center)})
289     return result

```

src/mcm26b/geometry/planning.py

```

1  """Distance certificates for Q2; does not change Q1's angular polygon model."""
2
3  from math import cos, sin, radians, isfinite
4  from .localization import Measurement
5
6
7  def first_bearing_candidate_disk(measurement: Measurement,
8                                  max_range_m: float = 1500,
9                                  guaranteed_range_m: float = 1000) -> dict:
10     """Enclose the entire truncated first-bearing sector, then erode its disk.
11
12     For half-angle <=45 degrees the sector MEC has center distance and radius
13     a=R/(2*cos(delta)). Intersecting with Omega only shrinks the source set.
14     A missing sufficient disk is NOT proof that Q is empty. The certificate
15     guarantees distance only: directional transmitters may still be invisible.
16     """
17     if any(isinstance(v, bool) or not isinstance(v, (int, float)) or not isfinite(v)
18           or v <= 0 for v in (max_range_m, guaranteed_range_m)):
19         raise ValueError("接收半径必须为有限正数")
20     if measurement.half_angle_deg > 45:
21         raise ValueError("此窄角域证书仅支持半角不超过45度; 题设为1度")
22     delta, theta = radians(measurement.half_angle_deg), radians(measurement.bearing_deg % 360)
23     a = max_range_m/(2*cos(delta))
24     center = [measurement.x+a*cos(theta), measurement.y+a*sin(theta)]
25     available = guaranteed_range_m-a
26     return {"source_model": "first_bearing_sector_with_range_upper_bound",
27            "sector_enclosing_circle": {"center": center, "radius_m": a},
28            "sufficient_candidate_disk": {"center": center, "radius_m": available}
29            if available >= 0 else None,
30            "candidate_nonempty_certified": available >= 0,
31            "distance_only": True,
32            "max_range_m": max_range_m, "guaranteed_range_m": guaranteed_range_m}

```

src/mcm26b/guarantee/__init__.py

```

1  """保证层: 覆盖计划 (CoveragePlan)、证书与运行期校验。
2
3  对外只暴露覆盖计划与校验入口; `coverage` 包仍负责逐频道覆盖账本。
4  """
5
6  from .cover import (CERTIFY_HEADINGS, CERTIFY_STEP_M, OMEGA_RADIUS_M,
7                     R_EFF_MIN_M, CoverageCertificate,
8                     CoveragePlan, PLAN_DIR, VERIFY_HEADINGS, VERIFY_STEP_M,

```

```

9         candidate_lattice, coverage_masks, forward_targets,
10        greedy_cover, load_plan, nn_tour_length, omega_positions,
11        plan_instance, plan_names, seven_plan, seven_points,
12        square_49_plan, verify_full_disk, verify_forward_half_disk,
13        verify_plan, write_plan)
14
15    __all__ = [
16        "CERTIFY_HEADINGS", "CERTIFY_STEP_M", "OMEGA_RADIUS_M", "R_EFF_MIN_M",
17        "CoverageCertificate", "CoveragePlan",
18        "PLAN_DIR", "VERIFY_HEADINGS", "VERIFY_STEP_M", "candidate_lattice",
19        "coverage_masks", "forward_targets", "greedy_cover", "load_plan",
20        "nn_tour_length", "omega_positions", "plan_instance", "plan_names",
21        "seven_plan", "seven_points", "square_49_plan", "verify_full_disk",
22        "verify_forward_half_disk", "verify_plan", "write_plan",
23    ]

```

src/mcm26b/guarantee/cover.py

```

1  """保证层：可离线搜索、可在运行期校验的覆盖计划（CoveragePlan）。
2
3  两类覆盖语义（题面约束不同，绝不可混用）：
4
5  - ``full_disk``（第三问，全向源）：在  $s$  处 no_signal 排除  $B(s, r_{\text{eff\_min}})$ ，
6    因此"全点皆 no_signal  $\implies$  不存在源"要求
7     $\bigcup B(c, 1000) \supseteq \Omega$ 。解析最优解是"原点 + 半径  $r$  的 6 等分环"，
8     $r$  的可行区间由 ``validate_seven_coverage`` 给出（本模块提供数值复核）。
9  - ``forward_half_disk``（第四问，定向源）：源可能背向读点，no_signal 不再
10    排除整个圆盘，缺席证明要求
11     $\forall (p, h) \in \Omega \times S^1, \exists c \in S: |c-p| \leq 1000 \wedge (c-p) \cdot u(h) \geq 0$ 。
12    现役 700 m 方格 49 点是充分条件（单元对角  $\leq 1000$ ），但远非必要；
13    本模块用贪心击中集把点数压到 31，并在同一采样空间上校核。
14
15  设计原则（docs/第三问架构优化与运行提速方案.md §2.3）：
16  **保证 = 离线搜索 + 运行期校验**。点表以 JSON 落盘并携带证书（采样分辨率、
17  缺口数、最坏命中距离、生成参数哈希），运行期用同一套采样语义重新校核，
18  未通过即拒绝启动。证书只能用于选点优化，不改变任何安全性证明。
19  """
20
21  from __future__ import annotations
22
23  import hashlib
24  import json
25  import math
26  from dataclasses import dataclass, field, fields
27  from functools import lru_cache
28  from pathlib import Path
29
30  from ..config import Vec2
31
32  PLAN_DIR = Path(__file__).resolve().parent / "plans"
33  R_EFF_MIN_M = 1000.0 # [题面] 最小有效接收半径
34  OMEGA_RADIUS_M = 1800.0 # [题面] 目标圆域半径
35  SPEED_MPS = 5.0 # [题面] 用于把巡回长度折算成虚拟移动秒数
36
37  # 采样分辨率。**校核必须与证书同分辨率**：更粗的采样会放过"删掉一个关键点
38  # 仍然看似覆盖"的点集，使校核退化为装饰（实测 100 m/12 档即漏判）。
39  # 75 m / 24 档下一次全域校核约 1 s，且按计划名缓存，每进程只付一次。
40  CERTIFY_STEP_M = 75.0
41  CERTIFY_HEADINGS = 24
42  VERIFY_STEP_M = CERTIFY_STEP_M
43  VERIFY_HEADINGS = CERTIFY_HEADINGS
44
45  @dataclass(frozen=True)
46  class CoverageCertificate:
47      """覆盖证书：采样语义 + 缺口 + 最坏命中距离。"""
48
49      kind: str # "full_disk" | "forward_half_disk"
50      target_step_m: float

```

```

52     headings: int
53     r_eff_min_m: float
54     targets: int
55     uncovered: int
56     worst_hit_distance_m: float
57     nn_tour_m: float
58     generator_sha256: str = ""
59     # 证书硬化: 校核时使用的前向角裕量 (度)。与 r_eff_min_m 的设计裕量
60     # 一起把"采样格覆盖"提升为"连续空间覆盖", 见 verify_forward_half_disk。
61     verify_heading_margin_deg: float = 0.0
62
63     @property
64     def valid(self) -> bool:
65         return self.uncovered == 0 and self.targets > 0
66
67     def as_dict(self) -> dict:
68         return {"kind": self.kind, "target_step_m": self.target_step_m,
69                 "headings": self.headings, "r_eff_min_m": self.r_eff_min_m,
70                 "targets": self.targets, "uncovered": self.uncovered,
71                 "worst_hit_distance_m": round(self.worst_hit_distance_m, 3),
72                 "nn_tour_m": round(self.nn_tour_m, 1),
73                 "nn_tour_virtual_s": round(self.nn_tour_m / SPEED_MPS, 1),
74                 "generator_sha256": self.generator_sha256,
75                 "verify_heading_margin_deg": self.verify_heading_margin_deg}
76
77
78 @dataclass(frozen=True)
79 class CoveragePlan:
80     """一份覆盖计划: 点集 + 语义 + 证书 (可为解析证明)。"""
81
82     name: str
83     problem: int
84     points: tuple[Vec2, ...]
85     kind: str                # "analytic" | "certified"
86     coverage: str            # "full_disk" | "forward_half_disk"
87     note: str = ""
88     certificate: CoverageCertificate | None = None
89     generator: dict = field(default_factory=dict)
90
91     def __len__(self) -> int:
92         return len(self.points)
93
94     def as_dict(self) -> dict:
95         return {"name": self.name, "problem": self.problem, "kind": self.kind,
96                 "coverage": self.coverage, "note": self.note,
97                 "generator": self.generator,
98                 "certificate": (self.certificate.as_dict()
99                                if self.certificate is not None else None),
100                 "points": [[p[0], p[1]] for p in self.points]}
101
102
103 # ----- 采样
104 def omega_positions(step_m: float) -> list[Vec2]:
105     """ $\Omega$  内均匀方格位置采样。"""
106     span = int(OMEGA_RADIUS_M / step_m) + 1
107     return [(step_m * i, step_m * j)
108             for i in range(-span, span + 1) for j in range(-span, span + 1)
109             if (step_m * i) ** 2 + (step_m * j) ** 2 <= OMEGA_RADIUS_M ** 2]
110
111
112 def forward_targets(step_m: float, headings: int) -> list[tuple[float, float, float]]:
113     """(p, h) 目标采样: 位置在  $\Omega$  内, 朝向均分整圆。"""
114     return [(x, y, 2.0 * math.pi * k / headings)
115             for (x, y) in omega_positions(step_m) for k in range(headings)]
116
117
118 def candidate_lattice(spacing_m: float, radius_m: float) -> list[Vec2]:
119     """半径 radius_m 圆盘内的方格候选点 (离线搜索用)。"""
120     span = int(radius_m / spacing_m) + 1
121     return [(spacing_m * i, spacing_m * j)

```

```

122         for i in range(-span, span + 1) for j in range(-span, span + 1)
123             if (spacing_m * i) ** 2 + (spacing_m * j) ** 2 <= radius_m ** 2]
124
125
126 def _bucket(points: list[Vec2], cell: float) -> dict[tuple[int, int], list[int]]:
127     table: dict[tuple[int, int], list[int]] = {}
128     for index, (x, y) in enumerate(points):
129         table.setdefault((int(x // cell), int(y // cell)), []).append(index)
130     return table
131
132
133 def coverage_masks(targets, candidates: list[Vec2],
134                   r_eff: float = R_EFF_MIN_M,
135                   heading_margin_deg: float = 0.0) -> list[int]:
136     """每个候选点覆盖的目标位掩码 (Python 大整数, 按位)。
137
138     只在与候选点空间桶相邻的格内做距离/前向判定, 避免  $|T| \cdot |C|$  笛卡尔积。
139
140     ``r_eff`` / ``heading_margin_deg`` 是**设计裕量**: 离线搜索时把"命中"
141     定义得比物理要求更严 (距离  $\leq r\_eff$ 、前向夹角  $\leq 90^\circ - \text{margin}$ ), 使采样格
142     上的覆盖自动蕴含相邻未采样 (p,h) 也被覆盖 (S2 证书硬化)。
143
144     """
145     table = _bucket(candidates, r_eff)
146     masks = [0] * len(candidates)
147     sin_margin = math.sin(math.radians(max(heading_margin_deg, 0.0)))
148     for t_index, (px, py, heading) in enumerate(targets):
149         ux, uy = math.cos(heading), math.sin(heading)
150         ci, cj = int(px // r_eff), int(py // r_eff)
151         bit = 1 << t_index
152         for di in (-1, 0, 1):
153             for dj in (-1, 0, 1):
154                 for c_index in table.get((ci + di, cj + dj), ()):
155                     cx, cy = candidates[c_index]
156                     dx, dy = cx - px, cy - py
157                     distance_sq = dx * dx + dy * dy
158                     if distance_sq > r_eff * r_eff:
159                         continue
160                     if dx * ux + dy * uy < sin_margin * math.sqrt(distance_sq) - 1e-9:
161                         continue
162                     masks[c_index] |= bit
163
164     return masks
165
166 def greedy_cover(masks: list[int], n_targets: int) -> list[int]:
167     """贪心集合覆盖: 每次取"新增覆盖目标数"最大的候选, 返回索引表。"""
168     full = (1 << n_targets) - 1
169     covered = 0
170     chosen: list[int] = []
171     remaining = list(range(len(masks)))
172     while covered != full and remaining:
173         best_index, best_gain = -1, 0
174         for c_index in remaining:
175             gain = (masks[c_index] & ~covered).bit_count()
176             if gain > best_gain:
177                 best_index, best_gain = c_index, gain
178         if best_index < 0:
179             break
180         chosen.append(best_index)
181         covered |= masks[best_index]
182         remaining.remove(best_index)
183     return chosen
184
185 def nn_tour_length(points: list[Vec2], start: Vec2 = (0.0, 0.0)) -> float:
186     """最近邻开放路径长度 (巡回长度上界)。"""
187     remaining = list(points)
188     position = start
189     total = 0.0
190     while remaining:
191         nxt = min(remaining, key=lambda q: (q[0] - position[0]) ** 2

```

```

192         + (q[1] - position[1]) ** 2)
193     remaining.remove(nxt)
194     total += math.dist(position, nxt)
195     position = nxt
196     return total
197
198
199 # ----- 校验
200 def _hit_distance(points: list[Vec2], position: Vec2, heading: float,
201                 r_eff: float) -> float | None:
202     """最近的前向命中距离；没有任何前向命中点则返回 None。"""
203     ux, uy = math.cos(heading), math.sin(heading)
204     best = None
205     for (cx, cy) in points:
206         dx, dy = cx - position[0], cy - position[1]
207         if dx * dx + dy * dy > r_eff * r_eff:
208             continue
209         if dx * ux + dy * uy < -1e-9:
210             continue
211         distance = math.hypot(dx, dy)
212         if best is None or distance < best:
213             best = distance
214     return best
215
216
217 def _nearest_distance(points: list[Vec2], position: Vec2,
218                     r_eff: float) -> float | None:
219     """最近距离（不要求前向）；超过 r_eff 视作未命中。"""
220     best = None
221     for (cx, cy) in points:
222         distance = math.hypot(cx - position[0], cy - position[1])
223         if distance <= r_eff and (best is None or distance < best):
224             best = distance
225     return best
226
227
228 def verify_forward_half_disk(points: list[Vec2], step_m: float = VERIFY_STEP_M,
229                             headings: int = VERIFY_HEADINGS,
230                             r_eff: float = R_EFF_MIN_M,
231                             heading_margin_deg: float = 0.0) -> CoverageCertificate:
232     """第四问语义： $\forall (p, h)$  存在前向半盘内的被访点。
233
234     ``heading_margin_deg`` 是**证书硬化**用的前向角裕量：把"命中"收紧为
235     "前向夹角  $\leq 90^\circ - \text{margin}$ "。与 ``r_eff`` 的设计裕量合用时，采样格上的
236     覆盖即可推出**连续空间**覆盖：真实  $(p', h')$  距采样格  $\leq \text{step} \cdot \sqrt{2}/2$ 、
237     方位差  $\leq 180/\text{headings}$  度，只要
238
239     
$$r_{\text{eff}} + \text{step} \cdot \sqrt{2}/2 \leq 1000 \quad \text{且} \quad \text{margin} \geq 180/\text{headings}/2$$

240
241     就能保证  $|c-p'| \leq 1000$  且  $(c-p') \cdot u(h') \geq 0$ 。这正是 S2 取代"粗采样
242     假通过"的方式。默认 0 表示按物理条件校核。
243     """
244     points = list(points)
245     targets = forward_targets(step_m, headings)
246     table = _bucket(points, r_eff)
247     missing, worst = 0, 0.0
248     sin_margin = math.sin(math.radians(max(heading_margin_deg, 0.0)))
249     for (px, py, heading) in targets:
250         ux, uy = math.cos(heading), math.sin(heading)
251         ci, cj = int(px // r_eff), int(py // r_eff)
252         best = None
253         for di in (-1, 0, 1):
254             for dj in (-1, 0, 1):
255                 for c_index in table.get((ci + di, cj + dj), ()):
256                     cx, cy = points[c_index]
257                     dx, dy = cx - px, cy - py
258                     distance_sq = dx * dx + dy * dy
259                     if distance_sq > r_eff * r_eff:
260                         continue
261                     if dx * ux + dy * uy < sin_margin * math.sqrt(distance_sq) - 1e-9:

```

```

262         continue
263         distance = math.sqrt(distance_sq)
264         if best is None or distance < best:
265             best = distance
266     if best is None:
267         missing += 1
268     else:
269         worst = max(worst, best)
270     return CoverageCertificate("forward_half_disk", step_m, headings, r_eff,
271                               len(targets), missing, worst,
272                               nn_tour_length(points))
273
274
275 def verify_full_disk(points: list[Vec2], step_m: float = VERIFY_STEP_M,
276                     r_eff: float = R_EFF_MIN_M) -> CoverageCertificate:
277     """第三问语义:  $\Omega$  内每点都在某个被访点的  $r_{\text{eff}}$  圆盘内。"""
278     points = list(points)
279     positions = omega_positions(step_m)
280     table = _bucket(points, r_eff)
281     missing, worst = 0, 0.0
282     for (px, py) in positions:
283         ci, cj = int(px // r_eff), int(py // r_eff)
284         best = None
285         for di in (-1, 0, 1):
286             for dj in (-1, 0, 1):
287                 for c_index in table.get((ci + di, cj + dj), ()):
288                     cx, cy = points[c_index]
289                     distance = math.hypot(cx - px, cy - py)
290                     if distance <= r_eff and (best is None or distance < best):
291                         best = distance
292     if best is None:
293         missing += 1
294     else:
295         worst = max(worst, best)
296     return CoverageCertificate("full_disk", step_m, 1, r_eff, len(positions),
297                               missing, worst, nn_tour_length(points))
298
299
300 def verify_plan(plan: CoveragePlan, step_m: float | None = None,
301                headings: int | None = None) -> CoverageCertificate:
302     """按计划声明的覆盖语义校核。
303
304     默认使用**证书自带的采样分辨率** (离线 75 m / 24 档): 更粗的运行期采样
305     会放过"删掉一个关键点仍看似覆盖"的点集, 使校核失去意义。校核按计划名
306     缓存(`load_plan`的 lru_cache), 每进程只付一次成本。
307     """
308     certificate = plan.certificate
309     if step_m is None:
310         step_m = certificate.target_step_m if certificate else VERIFY_STEP_M
311     if headings is None:
312         headings = (certificate.headings if certificate
313                    else VERIFY_HEADINGS)
314     if plan.coverage == "forward_half_disk":
315         # 硬化证书: 按证书自带的距离/角度设计裕量重新校核 (旧证书默认
316         # r_eff=1000、margin=0, 与历史口径逐位一致)。
317         return verify_forward_half_disk(
318             plan.points, step_m, headings,
319             r_eff=(certificate.r_eff_min_m if certificate else R_EFF_MIN_M),
320             heading_margin_deg=(certificate.verify_heading_margin_deg
321                                if certificate else 0.0))
322     return verify_full_disk(plan.points, step_m, certificate.r_eff_min_m
323                             if certificate else R_EFF_MIN_M)
324
325
326 # ----- 计划
327 def seven_points(ring_radius_m: float) -> list[Vec2]:
328     """解析七站: 原点 + 环上 6 等分点。"""
329     return [(0.0, 0.0)] + [(ring_radius_m * math.cos(k * math.pi / 3),
330                             ring_radius_m * math.sin(k * math.pi / 3))
331               for k in range(6)]

```

```

332
333
334 def seven_plan(ring_radius_m: float) -> CoveragePlan:
335     """第三问解析计划（覆盖证明由 validate_seven_coverage 给出）。"""
336     return CoveragePlan(
337         name=f"seven-r{ring_radius_m:.1f}", problem=3, kind="analytic",
338         coverage="full_disk",
339         note=(f"解析证明  $\max(r/\sqrt{3}, \sqrt{R^2+r^2-\sqrt{3}Rr}) < 1000$ ; "
340             f" $r \in [1122.9, 1732]$  时成立，巡检半径越小巡回越短"),
341         points=tuple(seven_points(ring_radius_m)))
342
343
344 def square_49_plan() -> CoveragePlan:
345     """第四问旧计划：700 m 方格 49 点（充分条件，保留作回退与对照）。"""
346     return CoveragePlan(
347         name="square49", problem=4, kind="certified",
348         coverage="forward_half_disk",
349         note=(f"单元对角  $700\sqrt{2} \approx 989.95 \leq 1000$  的充分条件；点数为最小命中集的 1.6 倍",
350             f"# 顺序与 coverage.coverage_points() 完全一致（y 由大到小），保持历史",
351             f"# 基线 grid49_immediate 的逐位可复现。"),
352         points=tuple((700.0 * i, 700.0 * j)
353             for j in range(3, -4, -1) for i in range(-3, 4)))
354
355
356 def _load_plan_file(path: Path) -> CoveragePlan:
357     payload = json.loads(path.read_text(encoding="utf-8"))
358     raw_cert = payload.get("certificate") or {}
359     if raw_cert:
360         # as_dict() 含 nn_tour_virtual_s 等派生列：只取数据类实际字段。
361         known = {field.name for field in fields(CoverageCertificate)}
362         certificate = CoverageCertificate(
363             **{key: value for key, value in raw_cert.items() if key in known})
364     else:
365         certificate = None
366     return CoveragePlan(
367         name=payload["name"], problem=int(payload["problem"]),
368         kind=payload.get("kind", "certified"),
369         coverage=payload.get("coverage", "forward_half_disk"),
370         note=payload.get("note", ""),
371         certificate=certificate,
372         generator=payload.get("generator", {}),
373         points=tuple((float(x), float(y)) for x, y in payload["points"]))
374
375
376 @lru_cache(maxsize=None)
377 def load_plan(name: str, verify: bool = True) -> CoveragePlan:
378     """按名称加载覆盖计划；``verify`` 为真时做运行期抽样校验（进程内缓存）。"""
379     path = PLAN_DIR / f"{name}.json"
380     if not path.exists():
381         raise FileNotFoundError(f"未找到覆盖计划 {name!r} ({path}) ")
382     plan = _load_plan_file(path)
383     if verify and plan.kind == "certified":
384         certificate = verify_plan(plan)
385         if not certificate.valid:
386             raise ValueError(
387                 f"覆盖计划 {name!r} 运行期校验失败: "
388                 f"{certificate.uncovered}/{certificate.targets} 个 (p,h) 未被覆盖")
389     plan = CoveragePlan(
390         name=plan.name, problem=plan.problem, kind=plan.kind,
391         coverage=plan.coverage, note=plan.note,
392         generator=plan.generator,
393         certificate=CoverageCertificate(
394             kind=certificate.kind, target_step_m=certificate.target_step_m,
395             headings=certificate.headings, r_eff_min_m=certificate.r_eff_min_m,
396             targets=certificate.targets, uncovered=certificate.uncovered,
397             worst_hit_distance_m=certificate.worst_hit_distance_m,
398             nn_tour_m=certificate.nn_tour_m,
399             generator_sha256=(plan.certificate.generator_sha256
400                             if plan.certificate else ""),
401         points=plan.points)

```

```

402     return plan
403
404
405 def plan_instance(name: str, verify: bool = True) -> CoveragePlan:
406     """统一入口：支持内置计划名与磁盘上的证书计划。"""
407     if name == "square49":
408         return square_49_plan()
409     if name == "seven":
410         return seven_plan(1200.0)
411     return load_plan(name, verify)
412
413
414 def plan_names() -> list[str]:
415     """全部可加载的计划名（含内置）。"""
416     builtin = ["square49", "seven"]
417     return builtin + sorted(p.stem for p in PLAN_DIR.glob("*.json"))
418
419
420 def write_plan(plan: CoveragePlan, path: Path) -> None:
421     """落盘计划 + 证书（生成器用）。"""
422     path.parent.mkdir(parents=True, exist_ok=True)
423     payload = plan.as_dict()
424     payload["generator"] = dict(payload.get("generator") or {})
425     payload["sha256"] = hashlib.sha256(
426         json.dumps(payload["points"]).encode("utf-8")).hexdigest()
427     path.write_text(json.dumps(payload, ensure_ascii=False, indent=2) + "\n",
428                     encoding="utf-8")

```

src/mcm26b/harness/__init__.py

```

1  """测试侧脚手架：把独立评测机（referee 包）接入本队传输与测试。
2
3  分层边界（与官方模拟器同构）：
4  - 评测机本体在独立包 ``referee``（零 mcm26b 依赖），运行时代码绝不导入它；
5  - 本包只在测试/自检入口把 referee 的接口边界适配成 :class:`Transport`，
6    并提供故障注入与官方演练日志回放。
7
8  策略端只见 HTTP 响应；真值 Source 只存在于 referee 与测试端。
9  """
10
11 from .playback import PlaybackTransport
12 from .scenes import mixed_scene, random_omni_scene, referee_spec
13 from .simulator import LocalSimulator
14 from .suites import SceneSpec, suite, suite_manifest
15 from .transport import LocalSimTransport, ScriptedFaultTransport
16
17 __all__ = ["LocalSimTransport", "LocalSimulator", "PlaybackTransport",
18           "ScriptedFaultTransport", "SceneSpec", "mixed_scene",
19           "random_omni_scene", "referee_spec", "suite", "suite_manifest"]

```

src/mcm26b/harness/playback.py

```

1  """回放传输：把既有演练日志作为服务器响应原样回放给客户端。
2
3  用于 P1 验收"本地状态机与官方响应一致"：官方演练记录
4  （output/q1/practice/*_actions.json）逐条喂给客户端，客户端重建的
5  状态必须与当时服务器返回的虚拟时刻逐步吻合。
6
7  按录制顺序匹配（客户端的 request_id 生成规则与采集器不同，
8  故不参与匹配；故障重试场景请用 ScriptedFaultTransport）。
9  """
10
11 from __future__ import annotations
12
13 import json
14 from urllib.parse import urlsplit
15
16 from ..protocol.transport import Transport, TransportResponse

```

```

17
18
19 class PlaybackTransport(Transport):
20     """events: 演练日志 [{path, request, response, ...}, ...]。"""
21
22     def __init__(self, events: list[dict]):
23         self.events = list(events)
24         self.index = 0
25         self.served: list[dict] = []
26
27     def post(self, url, body, headers, timeout):
28         if self.index >= len(self.events):
29             raise AssertionError("回放日志已耗尽仍有请求")
30         event = self.events[self.index]
31         path = urlsplit(url).path
32         if path != event["path"]:
33             raise AssertionError(
34                 f"回放顺序不一致: 客户端请求 {path}, 录制预期 {event['path']}")
35         self.index += 1
36         self.served.append(event)
37         return TransportResponse(
38             200, json.dumps(event["response"], ensure_ascii=False).encode("utf-8"))

```

src/mcm26b/harness/scenes.py

```

1  """自检场景生成（测试端专用）：委托 referee.scenes，映射本队配置。
2
3  真值只进入测试与 CLI ``--self-test`` 入口，策略端仅见模拟器响应；
4  默认采用满足题面范围的自建随机场景：源总数 10~16 随机、位置在目标圆域内
5  均匀分布、频道互异（1..20 抽样）、有效接收半径 [1000,1500] 均匀。
6  均匀分布是测试假设，题面并未规定官方场景的概率分布。
7  """
8
9  from __future__ import annotations
10
11  from referee import scenes as _referee_scenes
12
13  from ..config import ProblemConfig
14
15
16  def referee_spec(config: ProblemConfig = ProblemConfig(), *,
17                  max_real_duration_s: float | None = None):
18     """把本队 ProblemConfig 的题面常量映射为 referee 的 Spec。
19
20     max_real_duration_s 仅用于 --self-test 的实机预算覆盖（默认 1800s）；
21     正式对接以服务器 /enter 返回为准，评测机默认保持题面 1200s。
22     """
23     from referee import CostSpec, FieldSpec, Spec
24     field, cost = config.field, config.cost
25     return Spec(
26         field=FieldSpec(
27             omega_radius_m=field.omega_radius_m,
28             channel_min=field.channel_min, channel_max=field.channel_max,
29             source_count_min=field.source_count_min,
30             source_count_max=field.source_count_max,
31             r_eff_min_m=field.r_eff_min_m, r_eff_max_m=field.r_eff_max_m,
32             near_radius_m=field.near_radius_m, clear_radius_m=field.clear_radius_m,
33             bearing_error_half_deg=field.bearing_error_half_deg,
34             bearing_decimals=field.bearing_decimals,
35             coord_abs_limit_m=field.coord_abs_limit_m,
36             directional_half_angle_deg=field.directional_half_angle_deg,
37             max_virtual_duration_s=field.max_virtual_duration_s,
38             max_real_duration_s=(field.max_real_duration_s if max_real_duration_s is None
39                                 else max_real_duration_s),
40             test_window_s=field.test_window_s),
41         cost=CostSpec(speed_mps=cost.speed_mps, measure_s=cost.measure_s,
42                      switch_s=cost.switch_s, clear_success_s=cost.clear_success_s,
43                      clear_fail_s=cost.clear_fail_s))
44

```

```

45
46 def random_omni_scene(seed: int = 20260910, count: int | None = None,
47                       config: ProblemConfig = ProblemConfig()):
48     """全向源场景 [题面 问题3]; 签名沿用重构前 mcm26b.sim.scenes。"""
49     return _referee_scenes.random_omni_scene(seed, count, referee_spec(config))
50
51
52 def mixed_scene(seed: int = 20260910, count: int | None = None,
53                directional: int = 3, config: ProblemConfig = ProblemConfig()):
54     """问题4混合场景: count 个源中 directional 个为定向源。"""
55     return _referee_scenes.mixed_scene(seed, count, directional, referee_spec(config))

```

src/mcm26b/harness/simulator.py

```

1  """兼容构造入口: 本队 ProblemConfig -> referee.LocalSimulator。
2
3  评测机实现在独立包 referee (config= 换成了 spec=); 本子类只做配置映射,
4  让既有测试与脚本继续以 ``config=ProblemConfig`` 构造, 映射规则见
5  :func:`harness.scenes.referee_spec`。真值类型 Source 亦来自 referee。
6  """
7
8  from __future__ import annotations
9
10 from referee import LocalSimulator as _RefereeLocalSimulator
11
12 from ..config import ProblemConfig
13 from .scenes import referee_spec
14
15
16 class LocalSimulator(_RefereeLocalSimulator):
17
18     def __init__(self, sources, *, robot_id: str = "2026001",
19                  config: ProblemConfig = ProblemConfig(), **kwargs):
20         super().__init__(sources, robot_id=robot_id,
21                          spec=referee_spec(config), **kwargs)

```

src/mcm26b/harness/suites.py

```

1  """冻结基准集: 第三/四问的固定场景清单与真值构造 (测试端专用)。
2
3  存在意义 (docs/第三四问架构优化与运行提速方案.md §1.4 B6 / §P1-3) :
4  历史评估集是 20 个**单源全向**场景, 使 `avg_locate_clear_time_s` 恒等于
5  `virtual_time_s`, 也无法支撑任何第四问结论。基准集必须覆盖题面声明的
6  源数区间与定向源存在性, 并且**版本化、可复现、不可随意增删**——
7  任何"更快"的声明都要在同一份清单上做配对比较。
8
9  真值只在本模块与评测机之间流动; 策略端只通过 Transport 看到响应。
10 """
11
12 from __future__ import annotations
13
14 import math
15 from dataclasses import dataclass
16
17 from referee import Source
18
19 from ..config import ProblemConfig
20 from .scenes import mixed_scene, random_omni_scene
21
22 SUITE_VERSION = 1
23
24 # 第三问: 沿用历史验证集的种子与源数 (seeds 100 - 105 × 10/13/16)
25 Q3_SEEDS = (100, 101, 102, 103, 104, 105)
26 Q3_COUNTS = (10, 13, 16)
27 # 第四问: 三种"定向占比"组成 (少量 / 约半数 / 多数)
28 Q4_SEEDS = (200, 201, 202)
29 Q4_CONFIGS = ((10, 3), (13, 6), (16, 8))
30 # 误差场种子偏移: 与 benchmark_q3_bayes.py 同口径, 保证逐点固定误差可复现

```

```

31 ERROR_SEED_BASE = 20260910
32
33
34 @dataclass(frozen=True)
35 class SceneSpec:
36     """一个冻结场景：标识、题号、源数、定向数与种子。"""
37
38     scene_id: str
39     problem: int
40     kind: str          # "random" | "boundary"
41     seed: int
42     error_seed: int
43     count: int
44     directional: int = 0
45
46     def as_dict(self) -> dict:
47         return {"scene_id": self.scene_id, "problem": self.problem,
48                 "kind": self.kind, "seed": self.seed,
49                 "error_seed": self.error_seed, "count": self.count,
50                 "directional": self.directional}
51
52     def sources(self, config: ProblemConfig | None = None) -> list[Source]:
53         """构造真值（仅测试端调用）。"""
54         config = config or ProblemConfig()
55         if self.kind == "random" and self.problem == 3:
56             return random_omni_scene(seed=self.seed, count=self.count,
57                                     config=config)
58
59         if self.kind == "random":
60             return mixed_scene(seed=self.seed, count=self.count,
61                               directional=self.directional, config=config)
62
63         return _boundary_sources(self, config)
64
65     def _boundary_sources(spec: SceneSpec, config: ProblemConfig) -> list[Source]:
66         """圆周边界压力场景：源全部位于  $\Omega$  边界、 $r_{\text{eff}}$  取下界。
67
68         第三问：全向，扇区接缝角度，检验七站覆盖的边界情形；
69         第四问：全部定向且朝向**背离圆心**（最不利于发现），检验
70         "朝外发射的边界源不漏"。
71         """
72         field = config.field
73         radius = field.omega_radius_m
74         sources: list[Source] = []
75         for index in range(spec.count):
76             angle = math.pi / 6 + 2 * math.pi * index / spec.count
77             x, y = radius * math.cos(angle), radius * math.sin(angle)
78             directional = index < spec.directional
79             sources.append(Source(
80                 channel=index + 1, x=x, y=y, r_eff_m=field.r_eff_min_m,
81                 directional=directional,
82                 heading_deg=(math.degrees(angle) % 360.0) if directional else 0.0))
83         return sources
84
85     def core_scenes(problem: int) -> list[SceneSpec]:
86         """主体场景：覆盖题面源数区间与定向占比组成。"""
87         scenes: list[SceneSpec] = []
88         if problem == 3:
89             for seed in Q3_SEEDS:
90                 for count in Q3_COUNTS:
91                     scenes.append(SceneSpec(
92                         scene_id=f"q3-core-s{seed}-n{count}", problem=3,
93                         kind="random", seed=seed, error_seed=ERROR_SEED_BASE + seed,
94                         count=count))
95         else:
96             for seed in Q4_SEEDS:
97                 for count, directional in Q4_CONFIGS:
98                     scenes.append(SceneSpec(
99                         scene_id=f"q4-core-s{seed}-n{count}-d{directional}",
100                         problem=4, kind="random", seed=seed,

```

```

101         error_seed=ERROR_SEED_BASE + seed, count=count,
102         directional=directional))
103     return scenes
104
105
106 def boundary_scenes(problem: int) -> list[SceneSpec]:
107     """圆周边界压力场景（每间 2 场：10 源与 16 源）。"""
108     if problem == 3:
109         return [SceneSpec(f"q3-boundary-n{count}", 3, "boundary", 0,
110                          ERROR_SEED_BASE, count)
111                 for count in (10, 16)]
112     return [SceneSpec(f"q4-boundary-n{count}-d{directional}", 4, "boundary", 0,
113                      ERROR_SEED_BASE, count, directional)
114            for count, directional in ((10, 5), (16, 8))]
115
116
117 def suite(name: str, problem: int) -> list[SceneSpec]:
118     """按名称取场景清单：core / boundary / all。"""
119     if name == "core":
120         return core_scenes(problem)
121     if name == "boundary":
122         return boundary_scenes(problem)
123     if name == "all":
124         return core_scenes(problem) + boundary_scenes(problem)
125     raise ValueError(f"未知基准集 {name!r}（可选 core/boundary/all）")
126
127
128 def suite_manifest(name: str, problem: int) -> dict:
129     """清单头：版本、问题、场景表（写入 manifest.json）。"""
130     scenes = suite(name, problem)
131     return {"suite_version": SUITE_VERSION, "suite": name, "problem": problem,
132            "scenes": [scene.as_dict() for scene in scenes]}

```

src/mcm26b/harness/transport.py

```

1  """传输适配：评测机接口边界 <-> 本队 Transport 接口 + 故障注入脚本。
2
3  LocalSimTransport 把 referee 包的 handle_raw（与 HTTP 服务器共用的同一
4  边界实现）包成内存回环传输，不经过网络栈；故障注入语义与既有测试一致：
5  - ``"drop_before"`` 连接失败，动作未送达服务器（可安全重试）；
6  - ``"timeout_after"`` 服务器已受理并生效，但响应丢失（必须靠幂等回放）；
7  - ``("status", n)`` 服务器返回指定 HTTP 状态码（如 500/429）；
8  - ``None`` 正常放行。
9  脚本耗尽后一律放行。
10 """
11
12 from __future__ import annotations
13
14 from urllib.parse import urlsplit
15
16 from referee.interface import handle_raw
17
18 from ..protocol.transport import Transport, TransportError, TransportResponse
19
20
21 class LocalSimTransport(Transport):
22     """内存回环：把独立评测机接到 Transport 接口（与真实 HTTP 同一边界）。"""
23
24     def __init__(self, simulator):
25         # simulator: referee.LocalSimulator（类型注解走字符串，避免导入开销）
26         self.simulator = simulator
27         self.received: list[tuple[str, dict]] = []
28
29     def post(self, url, body, headers, timeout):
30         return self.request("POST", url, body, headers, timeout)
31
32     def request(self, method, url, body, headers, _timeout=None):
33         """验证原始请求边界：内存回环不模拟 TCP、UI 或连接限流。"""
34         target = urlsplit(url)

```

```

35     raw_target = target.path + (f"?{target.query}" if target.query else "")
36     result = handle_raw(self.simulator, method, raw_target,
37                          list(headers.items()), bytes(body),
38                          on_request=lambda path, payload, _res:
39                          self.received.append((path, payload)))
40     return TransportResponse(result.status, result.body)
41
42
43 class ScriptedFaultTransport(Transport):
44     """按脚本注入故障后转发给内层传输。"""
45
46     def __init__(self, inner: Transport, script: list):
47         self.inner = inner
48         self.script = list(script)
49         self.sent_requests: list[bytes] = []
50
51     def post(self, url, body, headers, timeout):
52         self.sent_requests.append(body)
53         fault = self.script.pop(0) if self.script else None
54         if fault == "drop_before":
55             raise TransportError("injected: connection refused before send")
56         if fault == "timeout_after":
57             self.inner.post(url, body, headers, timeout) # 服务器已受理
58             raise TransportError("injected: read timeout after server applied")
59         if isinstance(fault, tuple) and fault and fault[0] == "status":
60             body = b'{"accepted": false, "real_timestamp_ms": 0, "virtual_time_s": 0}'
61             return TransportResponse(int(fault[1]), body)
62         return self.inner.post(url, body, headers, timeout)

```

src/mcm26b/particles.py

```

1  """P4b 贝叶斯网格粒子滤波：逐频道幸存集维护与负观测更新。
2
3  与 belief.CandidateRegion（保守凸外包，无法表达非凸“挖洞”）互补：
4  网格幸存集精确表达“角域  $\cap$  距离盘  $\cap$  全部 no_signal 排除盘”的非凸可行集，
5  从而支持三件凸表示做不到的事——
6
7  1. 缺席证明：幸存集为空  $\implies$  可行集为空  $\implies$  该频道无源（严格，无需覆盖账本）；
8  2. 跳测规则：幸存格到测点的最小距离  $> r_{\max} + \text{裕量} \implies$  该次测量必 no_signal，
9     且不产生任何新排除，可直接跳过（省 5s 检测 + 1s 换频）；
10 3. 负观测更新：no_signal 排除  $B(s, 1000)$ ，凸表示不删点，网格直接删格。
11
12 严格性（真源保留定理，docs/algo-3-bayes.md §2）：格距  $h$  的 Voronoi 裕量
13  $m = \sqrt{2}/2 \cdot h$  是格心到格内任一点的最大距离。所有 kill 判定均内缩  $m$ ，
14 即“仅当整格与可行集不相交时才删格”。真源所在格与可行集之交含真源本身，
15 故该格永不被删——任意观测序列后幸存集恒包含真源，且：
16 - 清除证书：MEC(幸存). $R + m \leq 20 - \text{工程余量} \implies$  真源距清除点  $\leq 20$ ；
17 - 缺席证明：幸存集为空  $\implies$  可行集为空。
18
19 观测语义与独立评测机 referee.LocalSimulator 一致 [题面]：
20 - no_signal: 距离  $> r_{\text{eff}} \geq 1000 \implies$  可行集  $\cap B(s, 1000) = \emptyset$ ；
21 - direction: 距离  $\leq 1500$ ，示向度误差  $|\text{读数} - \text{真值}| \leq \delta_{\text{eff}}$ （含舍入裕量）；
22 - near: 距离  $\leq 5$ 。
23
24 性能设计：kill 与查询全部走  $O(1)$  集合运算；最近距离查询用粗粒度占用
25 索引（每格聚合  $CF^2$  个细格）螺旋扫描 + 早退，服务跳测规则的高频调用。
26 """
27
28 from __future__ import annotations
29
30 import math
31 import os
32
33 from .config import ProblemConfig, Vec2
34 from .geometry.localization import smallest_enclosing_circle
35 from .protocol.rules import DIRECTION, NEAR, NO_SIGNAL
36
37 _ANG_EPS_DEG = 1e-9 # 角向 kill 比较的浮点余量
38 _DIST_EPS_M = 1e-9

```

```

39 _COARSE_FACTOR = 4      # 粗格每边聚合的细格数
40
41 try:                      # numpy 仅作**可选**加速后端；缺失时全部回退纯 Python
42     import numpy as _np
43 except Exception:         # pragma: no cover - 环境无 numpy 时走纯 Python 分支
44     _np = None
45 if os.environ.get("MCM26B_NO_NUMPY"): # 强制纯 Python：对拍与回退验证用
46     _np = None
47
48
49 def angular_gap(a_deg: float, b_deg: float) -> float:
50     """两个方位角（度）的最小夹角。"""
51     return abs((a_deg - b_deg + 180.0) % 360.0 - 180.0)
52
53
54 def as_sample_array(samples):
55     """把 [(x,y), ...] 采样转成 (n,2) float64 数组；numpy 缺失时原样返回。
56
57     评分为 0(代表点 × 子采样) 的向量运算，是单局 CPU 的主要开销
58     (实测占 72%)。调用方在每次选点前转换一次，评分函数按类型分派，
59     两条实现 (numpy / 纯 Python) 语义逐位对齐，见 tests/test_particles.py。
60     """
61     if _np is None or isinstance(samples, _np.ndarray) or len(samples) == 0:
62         return samples
63     return _np.asarray(samples, dtype=float)
64
65
66 def _branch_radius_array(members) -> float:
67     """分支后验半径的数组版：到质心的最大距离。"""
68     centroid = members.mean(axis=0)
69     return float(_np.max(_np.hypot(members[:, 0] - centroid[0],
70                                   members[:, 1] - centroid[1])))
71
72
73 class ChannelParticles:
74     """单个频道的网格幸存集。
75
76     网格为覆盖  $\Omega$  外扩一格的轴对齐方格（边长  $h$ ），索引  $i*W + j$ ；
77     alive 是幸存格索引集合。初始集 = 格心距原点  $\leq \Omega$  半径 + 裕量的
78     全部格子，因此真源 ( $|p| \leq 1800$  [题面]) 所在格必在初始集内。
79     version 随每次删除自增，调用方可据此缓存派生量 (MEC、跳测判定)。
80     """
81
82     def __init__(self, config: ProblemConfig = ProblemConfig(),
83                  spacing_m: float | None = None):
84         field = config.field
85         self.config = config
86         self.spacing_m = float(spacing_m if spacing_m is not None
87                                else config.bayes.particle_spacing_m)
88         if not 1.0 <= self.spacing_m <= 200.0:
89             raise ValueError("网格间距须在 [1, 200] 米")
90         self.margin_m = self.spacing_m * math.sqrt(2) / 2 # Voronoi 裕量
91         self.r_eff_min_m = field.r_eff_min_m
92         self.r_eff_max_m = field.r_eff_max_m
93         self.near_radius_m = field.near_radius_m
94         self.half_angle_deg = (field.bearing_error_half_deg
95                                + config.search.bearing_rounding_margin_deg)
96         # 覆盖  $[-R-m, R+m]^2$  的格点阵列；格  $(i,j)$  的中心在
97         #  $(x_0 + (i+0.5)h, x_0 + (j+0.5)h)$ ，中心到格内任一点  $\leq h \cdot \sqrt{2}/2 =$  裕量
98         span = field.omega_radius_m + self.margin_m
99         self.x0 = -span
100         self.width = int(math.ceil(2 * span / self.spacing_m)) + 1
101         limit_sq = (span + 1e-9) ** 2
102         self.alive: set[int] = set()
103         self._coarse: dict[int, int] = {} # 粗格索引 -> 幸存细格数
104         coarse_w = self._coarse_width()
105         for i in range(self.width):
106             x = self.x0 + (i + 0.5) * self.spacing_m
107             for j in range(self.width):
108                 if x * x + (self.x0 + (j + 0.5) * self.spacing_m) ** 2 <= limit_sq:

```

```

109         index = i * self.width + j
110         self.alive.add(index)
111         key = (i // _COARSE_FACTOR) * coarse_w + j // _COARSE_FACTOR
112         self._coarse[key] = self._coarse.get(key, 0) + 1
113     self.version = 0
114     self._mec: tuple[Vec2, float] | None = None
115
116     # ----- 索引与坐标
117     def _coarse_width(self) -> int:
118         return (self.width + _COARSE_FACTOR - 1) // _COARSE_FACTOR
119
120     def cell_center(self, index: int) -> Vec2:
121         i, j = divmod(index, self.width)
122         return (self.x0 + (i + 0.5) * self.spacing_m,
123                 self.x0 + (j + 0.5) * self.spacing_m)
124
125     def discard(self, indices: list[int]) -> None:
126         """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。"""
127         self._discard(indices)
128
129     def _discard(self, indices: list[int]) -> None:
130         if not indices:
131             return
132         coarse_w = self._coarse_width()
133         for index in indices:
134             if index not in self.alive:
135                 continue
136             self.alive.discard(index)
137             key = (index // self.width // _COARSE_FACTOR) * coarse_w \
138                 + (index % self.width) // _COARSE_FACTOR
139             count = self._coarse.get(key, 0) - 1
140             if count <= 0:
141                 self._coarse.pop(key, None)
142             else:
143                 self._coarse[key] = count
144         self.version += 1
145         self._mec = None
146
147     # ----- 观测更新
148     def _kill_disk(self, center: Vec2, radius_m: float) -> None:
149         """删除格心距 center <= radius_m 的全部幸存格（整格判据见模块注释）。"""
150         if not self.alive:
151             return
152         reach = radius_m + _DIST_EPS_M
153         ilo = max(0, int(math.floor((center[0] - reach - self.x0) / self.spacing_m)))
154         ihi = min(self.width - 1,
155                   int(math.ceil((center[0] + reach - self.x0) / self.spacing_m)))
156         jlo = max(0, int(math.floor((center[1] - reach - self.x0) / self.spacing_m)))
157         jhi = min(self.width - 1,
158                   int(math.ceil((center[1] + reach - self.x0) / self.spacing_m)))
159         limit_sq = reach * reach
160         doomed = []
161         for i in range(ilo, ihi + 1):
162             dx = self.x0 + (i + 0.5) * self.spacing_m - center[0]
163             base = i * self.width
164             for j in range(jlo, jhi + 1):
165                 index = base + j
166                 if index not in self.alive:
167                     continue
168                 dy = self.x0 + (j + 0.5) * self.spacing_m - center[1]
169                 if dx * dx + dy * dy <= limit_sq:
170                     doomed.append(index)
171         self._discard(doomed)
172
173     def _kill_outside_disk(self, center: Vec2, radius_m: float) -> None:
174         """删除格心距 center > radius_m 的全部幸存格（保留盘内）。"""
175         if not self.alive:
176             return
177         reach = radius_m + _DIST_EPS_M
178         doomed = [index for index in self.alive

```

```

179         if math.dist(self.cell_center(index), center) > reach]
180     self._discard(doomed)
181
182 def _kill_outside_wedge(self, station: Vec2, reading_deg: float) -> None:
183     """删除角向偏离读数超出  $\delta_{\text{eff}}$  + 角向格裕量的幸存格。
184
185     角向格裕量 =  $\text{asin}(m / \max(\text{dist}-m, 1))$ : 格内任一点相对格心的方位
186     偏差不超过它 (点距测点还可再近  $m$ , 故分母再内缩  $m$ )。
187     """
188     if not self.alive:
189         return
190     sx, sy = float(station[0]), float(station[1])
191     doomed = []
192     for index in self.alive:
193         gx, gy = self.cell_center(index)
194         dist = math.hypot(gx - sx, gy - sy)
195         if dist <= self.margin_m + 1.0:
196             continue # 格贴测点: 格内方位可指向任意方向, 不可删
197         angular_margin = math.degrees(
198             math.asin(min(0.9999, self.margin_m / (dist - self.margin_m))))
199         gap = angular_gap(math.degrees(math.atan2(gy - sy, gx - sx)), reading_deg)
200         if gap > self.half_angle_deg + angular_margin + _ANG_EPS_DEG:
201             doomed.append(index)
202     self._discard(doomed)
203
204 def apply_measure(self, station: Vec2, result: str,
205                  svd_deg: float | None = None, *,
206                  directional: bool = False) -> None:
207     """按读数更新幸存集。真源保留性由 Voronoi 裕量保证 (模块注释)。
208
209     ``directional=True`` (第四问): no_signal **不删空间候选**——定向源
210     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
211     :class:`~mcm26b.belief.joint.JointBelief` 维护。
212     """
213     if result == NO_SIGNAL:
214         if directional:
215             return
216         # no_signal  $\implies$  dist >  $r_{\text{eff}}$  >= 1000: 整格落入  $B(s, 1000-m)$  的删。
217         self._kill_disk(station, self.r_eff_min_m - self.margin_m)
218     elif result == NEAR:
219         self._kill_outside_disk(station, self.near_radius_m + self.margin_m)
220     elif result == DIRECTION:
221         if svd_deg is None:
222             raise ValueError("direction 读数必须携带 svd_deg")
223         # direction 隐含 dist <=  $r_{\text{max}}$  [题面]
224         self._kill_outside_disk(station, self.r_eff_max_m + self.margin_m)
225         self._kill_outside_wedge(station, svd_deg)
226     else:
227         raise ValueError(f"未知测向结果: {result!r}")
228
229 # ----- 查询
230 def __len__(self) -> int:
231     return len(self.alive)
232
233 def alive_points(self, limit: int | None = None) -> list[Vec2]:
234     sorted_alive = sorted(self.alive)
235     if limit is not None and len(sorted_alive) > limit:
236         step = len(sorted_alive) / limit
237         indices = {sorted_alive[int(k * step) % len(sorted_alive)]
238                   for k in range(limit)}
239         indices = sorted(indices)
240     else:
241         indices = sorted_alive
242     return [self.cell_center(index) for index in indices]
243
244 def nearest_alive_distance(self, point: Vec2, limit_m: float) -> float | None:
245     """最近幸存格心距离; > limit_m 时返回 None (粗格环扫 + 下界剪枝)。
246
247     返回值语义: 恰为最近幸存格心距离 (若 <= limit_m); None 表示所有
248     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。

```

```

249     """
250     if not self.alive:
251         return None
252     coarse_cell_m = self.spacing_m * _COARSE_FACTOR
253     coarse_reach = coarse_cell_m * math.sqrt(2) / 2
254     cw = self._coarse_width()
255     ci_base = int(math.floor((point[0] - self.x0) / coarse_cell_m))
256     cj_base = int(math.floor((point[1] - self.x0) / coarse_cell_m))
257     ring_max = int(math.ceil((limit_m + coarse_reach) / coarse_cell_m)) + 1
258     best: float | None = None
259     for ring in range(ring_max + 1):
260         lower_bound = max(0.0, ring - 1) * coarse_cell_m - coarse_reach
261         if best is not None and lower_bound >= best:
262             break # 更外环的所有格心都 >= 下界 >= 当前最优
263         for ci in range(max(0, ci_base - ring), min(cw - 1, ci_base + ring) + 1):
264             for cj in range(max(0, cj_base - ring), min(cw - 1, cj_base + ring) + 1):
265                 if max(abs(ci - ci_base), abs(cj - cj_base)) != ring:
266                     continue # 只扫当前环 (Chebyshev 环)
267                 if (ci * cw + cj) not in self._coarse:
268                     continue
269                 ccx = self.x0 + (ci + 0.5) * coarse_cell_m
270                 ccy = self.x0 + (cj + 0.5) * coarse_cell_m
271                 if math.hypot(ccx - point[0], ccy - point[1]) \
272                     > limit_m + coarse_reach:
273                     continue
274                 i0, j0 = ci * _COARSE_FACTOR, cj * _COARSE_FACTOR
275                 for i in range(i0, min(i0 + _COARSE_FACTOR, self.width)):
276                     base = i * self.width
277                     for j in range(j0, min(j0 + _COARSE_FACTOR, self.width)):
278                         index = base + j
279                         if index not in self.alive:
280                             continue
281                         dist = math.dist(self.cell_center(index), point)
282                         if dist <= limit_m and (best is None or dist < best):
283                             best = dist
284     return best
285
286 def guaranteed_no_signal(self, station: Vec2) -> bool:
287     """真源与该测点距离必 > r_max (>= r_eff) ==> 必 no_signal, 测了也无信息。"""
288     return self.nearest_alive_distance(
289         station, self.r_eff_max_m + self.margin_m) is None
290
291 def mec(self) -> tuple[Vec2, float]:
292     """幸存集最小包围圆 (圆心, 半径); 按 version 缓存。"""
293     if self._mec is None:
294         if not self.alive:
295             raise ValueError("幸存集为空 (该频道已可证明缺席)")
296         points = [self.cell_center(index) for index in self.alive]
297         self._mec = smallest_enclosing_circle(points)
298     return self._mec
299
300 # ----- 信息增益
301 def subsample(self, size: int) -> list[Vec2]:
302     """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)。"""
303     return self.alive_points(limit=size)
304
305 # ----- 向量化评分 (可选)
306 def _expected_survival_vector(self, point: Vec2, sub, representatives) -> float:
307     """`expected_survival` 的向量化实现 (与纯 Python 版逐分支同语义)。"""
308     count = len(sub)
309     eps = 1.0 / count
310     dx, dy = sub[:, 0] - point[0], sub[:, 1] - point[1]
311     dist = _np.hypot(dx, dy)
312     within = int(_np.count_nonzero(
313         dist <= self.r_eff_min_m - self.margin_m + _DIST_EPS_M))
314     log_keep_ns = math.log(max(1.0 - within / count, eps))
315     bearing = _np.degrees(_np.arctan2(dy, dx))
316     in_range = dist <= self.r_eff_max_m + self.margin_m
317     rdx, rdy = representatives[:, 0] - point[0], representatives[:, 1] - point[1]
318     rdist = _np.hypot(rdx, rdy)

```

```

319     rbearing = _np.degrees(_np.arctan2(rdy, rdx))
320     gaps = _np.abs((bearing[None, :] - rbearing[:, None] + 180.0) % 360.0 - 180.0)
321     survivors = _np.count_nonzero(
322         (gaps <= self.half_angle_deg + _ANG_EPS_DEG) & in_range[None, :], axis=1)
323     p_ns = _np.clip((rdist - self.r_eff_min_m)
324                     / (self.r_eff_max_m - self.r_eff_min_m), 0.0, 1.0)
325     keep_dir = _np.maximum(survivors / count, eps)
326     values = p_ns * log_keep_ns + (1.0 - p_ns) * _np.log(keep_dir)
327     near = rdist <= self.near_radius_m + self.margin_m
328     values = _np.where(near, math.log(eps), values)
329     return float(values.mean())
330
331 def _expected_posterior_radius_vector(self, point: Vec2, sub,
332                                     representatives) -> float:
333     """`expected_posterior_radius` 的向量化实现。"""
334     eps_radius = self.spacing_m / 2.0
335     dx, dy = sub[:, 0] - point[0], sub[:, 1] - point[1]
336     dist = _np.hypot(dx, dy)
337     bearing = _np.degrees(_np.arctan2(dy, dx))
338     keep_ns = dist > self.r_eff_min_m - self.margin_m
339     r_ns = (_branch_radius_array(sub[keep_ns]) if keep_ns.any()
340            else eps_radius)
341     rdx, rdy = representatives[:, 0] - point[0], representatives[:, 1] - point[1]
342     rdist = _np.hypot(rdx, rdy)
343     rbearing = _np.degrees(_np.arctan2(rdy, rdx))
344     gaps = _np.abs((bearing[None, :] - rbearing[:, None] + 180.0) % 360.0 - 180.0)
345     dir_mask = ((gaps <= self.half_angle_deg + _ANG_EPS_DEG)
346                & (dist[None, :] <= self.r_eff_max_m + self.margin_m))
347     r_dir = _np.array(_branch_radius_array(sub[row]) if row.any()
348                      else eps_radius for row in dir_mask)
349     p_ns = _np.clip((rdist - self.r_eff_min_m)
350                     / (self.r_eff_max_m - self.r_eff_min_m), 0.0, 1.0)
351     near = rdist <= self.near_radius_m + self.margin_m
352     values = _np.where(near, self.near_radius_m + self.margin_m,
353                        p_ns * r_ns + (1.0 - p_ns) * r_dir)
354     return float(values.mean())
355
356 def expected_survival(self, point: Vec2, subsample,
357                      representatives) -> float:
358     """E[log 幸存占比]: 越小该测点的期望信息增益越大。
359
360     r_eff 不确定性解析积分: dist∈(1000,1500] 时按 r_eff~U(1000,1500)
361     取 p_ns=(dist-1000)/500 为 no_signal 分支概率。
362     - no_signal 分支: 删 B(point, 1000-m), 幸存 = 子采样中盘外比例;
363     - direction 分支: 以代表点方位为期望读数, 角域 ∩ B(point,1500+m) 内幸存;
364     - near 分支 (dist <= 5+m): 可行集收缩到 5m 盘, 幸存占比趋近 0。
365     """
366     if not self.alive or len(subsample) == 0 or len(representatives) == 0:
367         return 0.0
368     if _np is not None and isinstance(subsample, _np.ndarray) \
369        and isinstance(representatives, _np.ndarray):
370         return self._expected_survival_vector(point, subsample, representatives)
371     eps = 1.0 / len(subsample)
372     keep_ns = max(1.0 - self.fraction_within(subsample, point,
373                                              self.r_eff_min_m - self.margin_m),
374                  eps)
375     log_keep_ns = math.log(keep_ns)
376     total = 0.0
377     for rep in representatives:
378         dist = math.hypot(rep[0] - point[0], rep[1] - point[1])
379         if dist <= self.near_radius_m + self.margin_m:
380             total += math.log(eps)
381         continue
382     p_ns = min(1.0, max(0.0, (dist - self.r_eff_min_m)
383                       / (self.r_eff_max_m - self.r_eff_min_m)))
384     reading = math.atan2(rep[1] - point[1], rep[0] - point[0])
385     survivors = 0
386     for p in subsample:
387         if math.dist(p, point) > self.r_eff_max_m + self.margin_m:
388             continue

```

```

389         gap = angular_gap(math.degrees(
390             math.atan2(p[1] - point[1], p[0] - point[0])),
391             math.degrees(reading))
392         if gap <= self.half_angle_deg + _ANG_EPS_DEG:
393             survivors += 1
394         keep_dir = max(survivors / len(subsample), eps)
395         total += p_ns * log_keep_ns + (1.0 - p_ns) * math.log(keep_dir)
396     return total / max(len(representatives), 1)
397
398 def fraction_within(self, samples, point: Vec2, radius_m: float) -> float:
399     if len(samples) == 0:
400         return 0.0
401     reach = radius_m + _DIST_EPS_M
402     if _np is not None and isinstance(samples, _np.ndarray):
403         distance = _np.hypot(samples[:, 0] - point[0], samples[:, 1] - point[1])
404         return float(_np.count_nonzero(distance <= reach)) / len(samples)
405     count = sum(1 for p in samples if math.dist(p, point) <= reach)
406     return count / len(samples)
407
408 def _branch_radius(self, members) -> float:
409     """分支后验的半径代理：幸存者采样到其质心的最大距离（O(n)，免 Welzl）。"""
410     if len(members) == 0:
411         return 0.0
412     if _np is not None and isinstance(members, _np.ndarray):
413         return _branch_radius_array(members)
414     mx = sum(p[0] for p in members) / len(members)
415     my = sum(p[1] for p in members) / len(members)
416     return max(math.hypot(p[0] - mx, p[1] - my) for p in members)
417
418 def expected_posterior_radius(self, point: Vec2, subsample,
419     representatives) -> float:
420     """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。
421
422     对每个代表假设（真值候选）按  $r_{\text{eff}} \sim U(1000, 1500)$  解析积分
423     no_signal/direction 分支，near 分支（dist <= 5+m）后验为 5m 盘；
424     各分支半径用幸存者采样的质心最大距离近似（偏差 << 格距，排序用）。
425     与几何版 worst-posterior 同族，但作用于含负观测的非凸幸存集，
426     且以期望代替最坏（代表点覆盖幸存集主轴）。
427     """
428     if not self.alive or len(subsample) == 0 or len(representatives) == 0:
429         return 0.0
430     if _np is not None and isinstance(subsample, _np.ndarray) \
431         and isinstance(representatives, _np.ndarray):
432         return self._expected_posterior_radius_vector(
433             point, subsample, representatives)
434     eps_radius = self.spacing_m / 2.0
435     total = 0.0
436     for rep in representatives:
437         dist = math.hypot(rep[0] - point[0], rep[1] - point[1])
438         if dist <= self.near_radius_m + self.margin_m:
439             total += self.near_radius_m + self.margin_m
440             continue
441         p_ns = min(1.0, max(0.0, (dist - self.r_eff_min_m)
442             / (self.r_eff_max_m - self.r_eff_min_m)))
443         members_ns = [p for p in subsample
444             if math.dist(p, point) > self.r_eff_min_m - self.margin_m]
445         reading = math.atan2(rep[1] - point[1], rep[0] - point[0])
446         members_dir = []
447         for p in subsample:
448             if math.dist(p, point) > self.r_eff_max_m + self.margin_m:
449                 continue
450             gap = angular_gap(math.degrees(
451                 math.atan2(p[1] - point[1], p[0] - point[0])),
452                 math.degrees(reading))
453             if gap <= self.half_angle_deg + _ANG_EPS_DEG:
454                 members_dir.append(p)
455         r_ns = self._branch_radius(members_ns) if members_ns else eps_radius
456         r_dir = self._branch_radius(members_dir) if members_dir else eps_radius
457         total += p_ns * r_ns + (1.0 - p_ns) * r_dir
458     return total / max(len(representatives), 1)

```

src/mcm26b/planner/__init__.py

```
1  """P3/P4/P5 策略层：主动选点、全向闭环与定向扩展。
2
3  P3 第二检测点（验收：保证收到信号；Q 空时退化；near 分支）：
4  -  $K1 = \Omega \cap W(s1, \theta1 \pm \delta) \cap B(s1, 1500)$ ；安全候选盘证书见
5    geometry.planning.first_bearing_candidate_disk ( $\delta \leq 1.005^\circ$  时非空)；
6  - 在保证接收盘  $B(\text{mec\_center}, 1000 - \rho)$  内环形采样候选点，以
7    "最坏后验包围圆半径"为主指标、移动代价为次指标；
8  - 采样是启发式（目标边界点  $\times$  读数区间端点），不是严格最坏情形
9    最优解；选定后按 1/4 间距加密复核并如实上报数值；
10 - 返回 near 时：候选源在 5m 内，直接对当前频道尝试 /clear。
11
12 P4 全向闭环（验收：10/16/0 源场景、边界源、预算耗尽与故障注入）：
13 发现 (coverage.CoverageLedger)  $\rightarrow$  定位 (belief 收敛)  $\rightarrow$  清除  $\rightarrow$  持续复扫
14  $\rightarrow$  /exit。全部频道"已清除或证明不存在"后默认立即退出；显式开启
15 persistent_search 时只复扫未清除频道，扫到限时为止（清除数达 16 /
16 实机预算耗尽 / 虚拟安全余量不足）。completed 表示任务证明完成（结束
17 时每频道均 CLEARED/ABSENT 且无中断）；accounted_virtual_time_s 记录
18 首次证明找全的虚拟时刻，与"复扫到限时才停"相互独立。
19 断连/协议错误/守恒性破坏中断时同样尽力 /exit 并产出未完成 summary
20 （stop_reason 记中断类别，附逐频道状态快照），不静默丢失会话。
21
22 P5 定向扩展（问题4混合源闭环）：
23 - 49 点覆盖的发现保证对定向源同样成立（源在单元凸包内，任意发射半平面
24   含至少一个顶点），覆盖阶段不区分源类型；
25 - 保证接收盘内的 no_signal 只能由朝向遮挡解释 [观测推理]：频道记为
26   "观测到定向"，对跟踪点复核一次；联合约束基线不删空间候选（真源不误删）；
27 - 定位停滞时对有界候选集构造  $\leq 20\text{m}$  有限圆盘覆盖 (fallback_clear_cells)，
28   预算核对后逐格 /clear（清除与朝向无关）；格数超硬顶或预算不足即跳过，
29   绝不盲扫。
30 ""
31
32 from __future__ import annotations
33
34 import math
35 from dataclasses import dataclass
36
37 from ..belief import (BeliefState, CandidateRegion, ChannelStatus,
38                      InconsistentBelief, clip_halfplane)
39 from ..belief.joint import JointBelief
40 from ..config import ProblemConfig, Vec2
41 from ..coverage import CoverageLedger
42 from ..evaluation import recompute_virtual_time, summarize
43 from ..guarantee import plan_instance, seven_plan, square_49_plan
44 from ..protocol.client import (DeadlineExceeded, ProtocolError,
45                               SimulatorClient)
46 from ..protocol.rules import CLEAR_SUCCESS, DIRECTION, NEAR, NO_SIGNAL
47 from ..protocol.timing import move_time
48 from ..protocol.transport import TransportError
49 from ..routing import improve_open_path
50
51
52 def bearing_deg(a: Vec2, b: Vec2) -> float:
53     """从 a 指向 b 的方位角 (0-360, 0 为 x 正方向, 逆时针)。"""
54     return math.degrees(math.atan2(b[1] - a[1], b[0] - a[0])) % 360.0
55
56
57 def _error_brief(exc: BaseException) -> dict:
58     """异常摘要 (summary 留档用)：类型名 + 截断消息。"""
59     return {"type": type(exc).__name__, "message": str(exc)[:500]}
60
61
62 @dataclass(frozen=True)
63 class SecondPointPlan:
64     """下一检测点的候选与依据 (P3 输出)。"""
65
66     position: Vec2
67     worst_posterior_radius_m: float
68     move_cost_s: float
```

```

69     reason: str
70
71
72 @dataclass(frozen=True)
73 class ClearCertificate:
74     """清除证书：候选集外包半径（含近似误差）不超过 20m 阈值。"""
75
76     channel: int
77     center: Vec2
78     radius_m: float
79     limit_m: float
80     certified: bool
81
82
83 def clear_certificate(region: CandidateRegion, channel: int,
84                      config: ProblemConfig = ProblemConfig()) -> ClearCertificate:
85     """ $\rho_{poly} \leq 20$  - 外包近似误差 - 工程余量 才允许发起清除。"""
86     center, radius = region.mec()
87     limit = (config.field.clear_radius_m - region.slack_m
88             - config.search.clear_margin_m)
89     return ClearCertificate(channel, center, radius, limit, radius <= limit)
90
91
92 def candidate_points(center: Vec2, safe_radius_m: float, rings, angles: int) -> list[Vec2]:
93     """保证接收盘内的候选点：盘心 + 各比例半径 * 均匀方向。"""
94     points = [tuple(center)]
95     for fraction in rings:
96         radius = fraction * safe_radius_m
97         if radius <= 1e-9:
98             continue
99         step = 2 * math.pi / angles
100         for k in range(angles):
101             angle = step * k + step / 2
102             points.append((center[0] + radius * math.cos(angle),
103                           center[1] + radius * math.sin(angle)))
104     return points
105
106
107 def _worst_posterior_radius(region: CandidateRegion, q: Vec2,
108                            witnesses: list[Vec2], half_angle_deg: float,
109                            near_radius_m: float, disk_sides: int) -> float:
110     """对候选点 q，遍历"目标边界点 * 读数分支"的最坏后验包围半径。
111
112     读数分支：direction 的区间端点  $\varphi(g) \pm \delta$ （端点是每个 g 的最不利读数），
113     以及 near——源落在 q 的 near_radius_m 内时后验为  $B(q, near) \cap$  候选集。
114     保证接收盘内 no_signal 不可能发生，故不枚举该分支。
115     这是启发式近似，不构成最优性证明 [工程]。
116     """
117     posterior_radii = []
118     for g in witnesses:
119         phi = bearing_deg(q, g)
120         for reading in (phi - half_angle_deg, phi + half_angle_deg):
121             probe = CandidateRegion(list(region.polygon), region.slack_m)
122             probe.clip_wedge(q, reading, half_angle_deg)
123             if probe.polygon:
124                 posterior_radii.append(probe.mec()[1])
125     near_probe = CandidateRegion(list(region.polygon), region.slack_m)
126     try:
127         near_probe.clip_disk(q, near_radius_m, disk_sides)
128     except InconsistentBelief:
129         pass # B(q, near) 与候选集不相交：near 分支不可能发生，跳过
130     else:
131         if near_probe.polygon:
132             posterior_radii.append(near_probe.mec()[1])
133     if not posterior_radii:
134         return region.mec()[1] # 无有效采样点时退回当前半径
135     return max(posterior_radii)
136
137
138 def safe_disk_radius(region: CandidateRegion,

```

```

139         config: ProblemConfig = ProblemConfig()) -> float:
140     """保证接收盘 B(mec, safe) 的半径; <0 表示无保证 ( $\rho$  超过最小接收半径)。
141
142     盘内任意点到候选集每点距离  $\leq \text{safe} + \rho + 1e-3 < 1000 \leq$  最小接收半径,
143     故必然收到 direction/near. 1mm 数值余量避免浮点贴边被  $r_{\text{eff}}=1000$  的源
144     判为 no_signal。
145     """
146     return config.field.r_eff_min_m - region.mec()[1] - 1e-3
147
148
149 def select_measurement_point(region: CandidateRegion, from_position: Vec2,
150                             config: ProblemConfig = ProblemConfig()) -> SecondPointPlan:
151     """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)。
152
153     保证接收盘 B(mec_center, 1000- $\rho$ ): 盘内任意点到候选集每点距离
154      $\leq (1000-\rho) + \rho = 1000 \leq$  最小接收半径, 故必然收到 direction/near,
155     不会浪费一次 no_signal。  $\rho>1000$  时退化为探索 MEC 圆心 (可能 no_signal)。
156     """
157     field, search = config.field, config.search
158     center, radius = region.mec()
159     half_angle = field.bearing_error_half_deg + search.bearing_rounding_margin_deg
160
161     def move_cost(point: Vec2) -> float:
162         return move_time(from_position, point, config.cost)
163
164     # 1mm 数值余量在 safe_disk_radius 内记账。
165     safe_radius = safe_disk_radius(region, config)
166     if safe_radius < 0:
167         return SecondPointPlan(tuple(center), radius, move_cost(center),
168                                 "无保证接收盘 ( $\rho>1000$ ): 探索 MEC 圆心, 可能 no_signal")
169     coarse = region.witnesses(search.witness_spacing_m)
170     samples = candidate_points(center, safe_radius,
171                               search.candidate_rings, search.candidate_angles)
172     near_radius, sides = field.near_radius_m, search.polygon_sides
173     best_key, best_point, best_worst = None, None, None
174     for point in samples:
175         worst = _worst_posterior_radius(region, point, coarse, half_angle,
176                                         near_radius, sides)
177         key = (round(worst, 6), round(move_cost(point), 6))
178         if best_key is None or key < best_key:
179             best_key, best_point, best_worst = key, point, worst
180     dense = _worst_posterior_radius(region, best_point,
181                                     region.witnesses(search.witness_spacing_m / 4),
182                                     half_angle, near_radius, sides)
183     reason = (f"安全盘 B(centre,{safe_radius:.0f}m) 内 {len(samples)} 候选; "
184              f"粗采样最坏后验 {best_worst:.1f}m, 1/4 间距加密复核 {dense:.1f}m")
185     return SecondPointPlan(tuple(best_point), dense, move_cost(best_point), reason)
186
187
188 def select_second_point(client: SimulatorClient, first_position: Vec2,
189                        first_bearing_deg: float,
190                        config: ProblemConfig = ProblemConfig()) -> SecondPointPlan:
191     """P3 入口: 由首次 direction 构造 K1, 返回保证接收的第二检测点方案。"""
192     region = CandidateRegion.initial(first_position, config)
193     half_angle = config.field.bearing_error_half_deg \
194         + config.search.bearing_rounding_margin_deg
195     region.clip_wedge(first_position, first_bearing_deg, half_angle)
196     return select_measurement_point(region, client.state.position, config)
197
198
199 class OmnidirectionalLoop:
200     """问题3全向"发现一定位—清除—持续复扫"闭环 (P4 基线 + 持续搜索)。
201
202     串行驱动 SimulatorClient: 覆盖阶段在 49 点网格上对每个 UNKNOWN 频道
203     检测: 发现 direction 即进入该频道的定位子循环 (选点→测量→收敛→
204     证书清除), near 直接清除。全部频道"已清除或证明不存在"后按
205     ``search.persistent_search`` (默认关闭) 可选择只复扫
206     未清除频道, 直到清除数达 16、实机预算耗尽或虚拟安全余量不足。
207     stop_reason 记录循环结束原因; ``completed`` 表示任务证明完成
208     (结束时每频道均 CLEARED/ABSENT 且无中断), 与"复扫到限时才停"

```

```

209 相互独立: accounted_virtual_time_s 记录首次证明找全的虚拟时刻。
210 """
211
212 def __init__(self, client: SimulatorClient,
213              config: ProblemConfig = ProblemConfig()):
214     self.client = client
215     self.config = config
216     self.belief = BeliefState(config)
217     # 覆盖计划是"可离线搜索、可在运行期校验"的一等对象; 基类默认保持
218     # 历史 49 点表 (grid49_immediate 基线口径不变), 各问题策略自行覆盖。
219     self.coverage_plan = square_49_plan()
220     self.ledger = CoverageLedger(config, points=list(self.coverage_plan.points))
221     self.cleared: list[int] = []
222     self.anomalies: list[dict] = []
223     self.stop_reason: str | None = None
224     # 逐频道缺席证据来源 (pf_empty / coverage), 供审计与论文逐条追溯。
225     self.absence_evidence: dict[int, str] = {}
226     # "同一 0.5m 测点 × 同一频道"的重复检测计数: 历史上一次官方演练
227     # 有 98.84% 的虚拟时间耗在空复扫上, 本计数是该问题的回归指标。
228     self.redundant_rescan_measures = 0
229     # 首次达成"每频道均已清除或证明不存在"的虚拟时刻 (找全证明)。
230     self.accounted_virtual_time_s: float | None = None
231     # 证书有效但 /clear 失败后必须先补测再重试, 避免在同一证书上
232     # 反复消耗 3 秒/次的失败清除。
233     self._may_clear: set[int] = set()
234     # 断连/协议错误/守恒性破坏等中断: 留档异常, 收尾时写入 summary。
235     self._abort_detail: BaseException | None = None
236
237 # ----- 主流程
238 def run(self) -> dict:
239     """主闭环。任何异常都必须尽力 /exit 并产出 summary, 不得静默丢会话。"""
240     try:
241         self.client.enter()
242         self._coverage_phase()
243         if self.stop_reason is None:
244             self._mark_absent_after_coverage()
245             self._resolve_detected()
246             self._record_accounted()
247             if self.config.search.persistent_search:
248                 self._persistent_phase()
249             if self.stop_reason is None:
250                 if len(self.cleared) >= self.config.field.source_count_max:
251                     self.stop_reason = "sixteen_cleared"
252                 elif self._all_accounted():
253                     self.stop_reason = "all_resolved"
254             else:
255                 self.stop_reason = "unresolved_channels"
256     except DeadlineExceeded:
257         self.stop_reason = "real_budget_exhausted"
258     except (TransportError, ProtocolError, InconsistentBelief) as exc:
259         self._abort_detail = exc
260         if isinstance(exc, TransportError):
261             self.stop_reason = "transport_failure"
262         elif isinstance(exc, InconsistentBelief):
263             self.stop_reason = "inconsistent_belief"
264         else:
265             self.stop_reason = "protocol_error"
266     except Exception as exc: # 意外异常同样收尾, 不阻断 /exit 与结果汇总
267         self._abort_detail = exc
268         self.stop_reason = "unexpected_error"
269     info = None
270     exit_error = None
271     try:
272         info = self.client.exit()
273     except Exception as exc: # 接口不可用: 记录后仍产出未完成状态
274         exit_error = exc
275     summary = self._summary(info)
276     if exit_error is not None:
277         summary["exit_error"] = _error_brief(exit_error)
278     return summary

```

```

279
280 # ----- 预算与证明
281 def _min_virtual_reserve(self) -> float:
282     """单动作耗时上界的保守估计，用于虚拟安全余量。
283
284     上界 = 2 × 最远站点距离 / 速度 + 检测 + 换频（机器人不会跑到覆盖
285     计划包围区之外执行任务）。配置值通常更宽松；本函数保证即使配置被
286     调小，余量也足以完成最后一个动作而不触及 100 小时上限。
287     """
288     reach = max((math.dist((0.0, 0.0), point) for point in self.ledger.points),
289                 default=0.0)
290     action_bound = (2.0 * reach / self.config.cost.speed_mps
291                     + self.config.cost.measure_s + self.config.cost.switch_s)
292     return max(self.config.search.virtual_reserve_s, action_bound)
293
294 def _virtual_cap_reached(self) -> bool:
295     """虚拟安全余量不足：置 stop_reason 并返回 True，调用方立即收尾。
296
297     余量须大于单次动作耗时上界（见 :meth:`_min_virtual_reserve`），
298     `search.virtual_reserve_s=10000s` [工程] 提供充分裕度，避免触及
299     附件2 §4.5 的 100 小时上限被模拟器强制结束。
300     """
301     if (self.client.state.virtual_time_s + self._min_virtual_reserve()
302         < self.config.field.max_virtual_duration_s):
303         return False
304     self.stop_reason = "virtual_time_cap"
305     return True
306
307 def _all_accounted(self) -> bool:
308     return all(belief.status in (ChannelStatus.CLEARED, ChannelStatus.ABSENT)
309               for belief in self.belief.channels.values())
310
311 def _record_accounted(self) -> None:
312     """首次"每频道均已清除或证明不存在"的时刻留档（找全证明）。"""
313     if self.accounted_virtual_time_s is None and self._all_accounted():
314         self.accounted_virtual_time_s = self.client.state.virtual_time_s
315
316 # ----- 覆盖阶段
317 def _pending_channels_at(self, index: int) -> list[int]:
318     return [channel for channel, belief in self.belief.channels.items()
319            if belief.status is ChannelStatus.UNKNOWN
320            and not self.ledger.measured_at(channel, index)]
321
322 def _next_point_index(self) -> int | None:
323     """尚有 UNKNOWN 频道待测的最近网格点（贪心最近邻 [工程]）。"""
324     position = self.client.state.position
325     best_index, best_distance = None, 0.0
326     for index, point in enumerate(self.ledger.points):
327         if not self._pending_channels_at(index):
328             continue
329         distance = math.dist(position, point)
330         if best_index is None or distance < best_distance:
331             best_index, best_distance = index, distance
332     return best_index
333
334 def _coverage_phase(self) -> None:
335     while True:
336         if self._count_limit_reached():
337             return
338         index = self._next_point_index()
339         if index is None:
340             return
341         point = self.ledger.points[index]
342         for channel in self._pending_channels_at(index):
343             if self._virtual_cap_reached():
344                 return
345             outcome = self.client.measure(point, channel)
346             self.ledger.mark_measured(channel, index)
347             self.belief.update_measure(channel, point, outcome.result,
348                                     outcome.svd_deg)

```

```

349         if outcome.result == DIRECTION:
350             self._localize_and_clear(channel)
351         elif outcome.result == NEAR:
352             region = self.belief.candidate_set(channel)
353             cert = clear_certificate(region, channel, self.config)
354             if cert.certified:
355                 self._attempt_clear(channel, cert.center, cert)
356             else: # 理论上不发生 (near 候选半径 5m) ; 保守记录
357                 self._anomaly("near_without_certificate", channel,
358                             position=point)
359         if self.stop_reason is not None or self._count_limit_reached():
360             return
361
362 # ----- 持续复扫
363 def _persistent_phase(self) -> None:
364     """全部频道已清除/证明不存在后的持续复扫 [用户决策 2026-09-11]。
365
366     只扫未清除频道 (DETECTED 继续定位, ABSENT 全网格复测, 防模拟器
367     或覆盖证明假设偏差), 直到清除数达 16 或虚拟安全余量不足;
368     实机预算由客户端 DeadlineExceeded 兜底 (扫到限时为止)。
369     """
370     while self.stop_reason is None:
371         if self._count_limit_reached():
372             return
373         if self._virtual_cap_reached():
374             return
375         self._persistent_round()
376
377 def _persistent_round(self) -> None:
378     """一轮全网格复扫: 网格点最近邻序, 逐点测量全部未清除频道。"""
379     measured: set[tuple[int, int]] = set()
380     while self.stop_reason is None:
381         index = self._next_resweep_point(measured)
382         if index is None:
383             return
384         point = self.ledger.points[index]
385         for channel in self._resweep_channels_at(index, measured):
386             if self._virtual_cap_reached():
387                 return
388             measured.add((channel, index))
389             was_absent = (self.belief.channels[channel].status
390                          is ChannelStatus.ABSENT)
391             outcome = self.client.measure(point, channel)
392             self.belief.update_measure(channel, point, outcome.result,
393                                       outcome.svd_deg)
394             if was_absent and outcome.result in (DIRECTION, NEAR):
395                 # 复测推翻了覆盖缺席证明: 守恒性留痕, 按新证据定位清除。
396                 self._anomaly("absent_detected_on_resweep", channel,
397                             position=point, result=outcome.result)
398             if outcome.result == DIRECTION:
399                 self._localize_and_clear(channel)
400             elif outcome.result == NEAR:
401                 region = self.belief.candidate_set(channel)
402                 cert = clear_certificate(region, channel, self.config)
403                 if cert.certified:
404                     self._attempt_clear(channel, cert.center, cert)
405                 else:
406                     self._anomaly("near_without_certificate", channel,
407                                 position=point)
408             if self.stop_reason is not None:
409                 return
410
411 def _next_resweep_point(self, measured: set[tuple[int, int]]) -> int | None:
412     """尚有未清除频道待复测的最近网格点 (贪心最近邻 [工程])。"""
413     position = self.client.state.position
414     best_index, best_distance = None, 0.0
415     for index, point in enumerate(self.ledger.points):
416         if not self._resweep_channels_at(index, measured):
417             continue
418         distance = math.dist(position, point)

```

```

419         if best_index is None or distance < best_distance:
420             best_index, best_distance = index, distance
421     return best_index
422
423     def _resweep_channels_at(self, index: int,
424                             measured: set[tuple[int, int]]) -> list[int]:
425         return [channel for channel, belief in self.belief.channels.items()
426                 if belief.status is not ChannelStatus.CLEARED
427                    and (channel, index) not in measured]
428
429     def _count_limit_reached(self) -> bool:
430         if len(self.cleared) >= self.config.field.source_count_max:
431             self.stop_reason = "sixteen_cleared"
432             return True
433         return False
434
435     # ----- 定位与清除
436     def _localize_and_clear(self, channel: int) -> bool:
437         """定位子循环：选点测量收敛候选集，拿到清除证书后清除。
438
439         返回是否已清除；步数耗尽仍未收敛则保持 DETECTED 留待二轮。
440         进入本函数前该频道刚有一次 direction 读数，允许立即清除。
441         """
442         search = self.config.search
443         self._may_clear.add(channel)
444         for _ in range(search.max_localize_steps):
445             if self._virtual_cap_reached():
446                 return False
447             region = self.belief.candidate_set(channel)
448             if region is None:
449                 return False
450             cert = clear_certificate(region, channel, self.config)
451             if cert.certified and channel in self._may_clear:
452                 if self._attempt_clear(channel, cert.center, cert):
453                     return True
454                 self._may_clear.discard(channel)
455                 continue
456             plan = select_measurement_point(region, self.client.state.position,
457                                             self.config)
458             outcome = self.client.measure(plan.position, channel)
459             self.belief.update_measure(channel, plan.position, outcome.result,
460                                       outcome.svd_deg)
461             self._may_clear.add(channel)
462             if outcome.result == NO_SIGNAL:
463                 self._anomaly("no_signal_in_safe_disk", channel,
464                               position=plan.position, detail=plan.reason)
465             return self.belief.channels[channel].status is ChannelStatus.CLEARED
466
467     def _attempt_clear(self, channel: int, target: Vec2,
468                       cert: ClearCertificate) -> bool:
469         outcome = self.client.clear(target, channel)
470         if outcome.result == CLEAR_SUCCESS:
471             self.belief.mark_cleared(channel)
472             self.cleared.append(channel)
473             return True
474         if cert.certified:
475             self._anomaly("clear_failed_with_certificate", channel, position=target,
476                           radius_m=round(cert.radius_m, 3), limit_m=cert.limit_m)
477         return False
478
479     def _anomaly(self, kind: str, channel: int, **detail) -> None:
480         """异常只在守恒性被打破或证书失效时出现；每频道限量记录防刷屏。"""
481         recorded = sum(1 for item in self.anomalies if item["channel"] == channel)
482         if recorded >= self.config.search.max_anomalies_per_channel:
483             return
484         self.anomalies.append({"kind": kind, "channel": channel, **detail})
485
486     # ----- 收尾
487     def _mark_absent_after_coverage(self) -> None:
488         """全向源缺席证明：全网格无非 no_signal 读数的频道不存在源 [题面]。"""

```

```

489         for channel, belief in self.belief.channels.items():
490             if belief.status is ChannelStatus.UNKNOWN \
491                 and self.ledger.coverage_complete(channel):
492                 self.belief.mark_absent(channel)
493                 self.absence_evidence[channel] = "coverage"
494
495     def _resolve_detected(self) -> None:
496         """覆盖完成后，对仍未清除的 DETECTED 频道做第二轮定位。
497
498         不再在此决定停止：持续搜索模式下由 _persistent_phase 接力，
499         非持续模式由 run() 依据 _all_accounted 判定 all_resolved。
500         """
501         for channel in self.belief.channels_by_status(ChannelStatus.DETECTED):
502             if self._count_limit_reached() or self._virtual_cap_reached():
503                 return
504             self._localize_and_clear(channel)
505             if self.stop_reason is not None:
506                 return
507
508     def _summary(self, info) -> dict:
509         """结果汇总。info 为 None 表示 /exit 未能完成（断连等），按未完成上报。
510
511         completed = 无中断且结束时每频道均 CLEARED/ABSENT（任务证明完成）；
512         stop_reason 另行记录循环结束原因（持续复扫模式下通常是
513         virtual_time_cap / real_budget_exhausted / sixteen_cleared）。
514         """
515         events = self.client.log.action_events()
516         stats = summarize(events)
517         report = recompute_virtual_time(events)
518         steps = self._step_report()
519         cleared = self.belief.channels_by_status(ChannelStatus.CLEARED)
520         absent = self.belief.channels_by_status(ChannelStatus.ABSENT)
521         resolved = set(cleared) | set(absent)
522         unresolved = [channel for channel in sorted(self.belief.channels)
523                       if channel not in resolved]
524         virtual_time_s = round((info.virtual_time_s if info is not None
525                                else self.client.state.virtual_time_s), 3)
526         # 清除数达 16 即数量上界证明（其余频道无论状态如何都不可能源）；
527         # 否则要求每频道均 CLEARED/ABSENT。
528         completed = (self._abort_detail is None
529                     and (len(self.cleared) >= self.config.field.source_count_max
530                          or not unresolved))
531         # 完成依据必须显式区分"数量上界"与"逐频道缺席/清除证明"，
532         # 否则论文与审计会把 16 源截断误读为逐频道证明。
533         if self._abort_detail is not None:
534             completion_basis = "incomplete"
535         elif len(self.cleared) >= self.config.field.source_count_max:
536             completion_basis = "count_cap"
537         elif not unresolved:
538             completion_basis = "per_channel_proof"
539         else:
540             completion_basis = "incomplete"
541         summary = {
542             "completed": completed,
543             "completion_basis": completion_basis,
544             "stop_reason": self.stop_reason,
545             "exit_reason": "exit_unreachable" if info is None else info.reason,
546             "virtual_time_s": virtual_time_s,
547             "accounted_virtual_time_s": (round(self.accounted_virtual_time_s, 3)
548                                         if self.accounted_virtual_time_s is not None
549                                         else None),
550             "persistent_search": self.config.search.persistent_search,
551             "virtual_reserve_s": round(self._min_virtual_reserve(), 3),
552             "coverage_plan": self.coverage_plan.name,
553             "coverage_points": len(self.coverage_plan.points),
554             "coverage_plan_points": [[p[0], p[1]] for p in self.coverage_plan.points],
555             "coverage_plan_certificate": (
556                 self.coverage_plan.certificate.as_dict()
557                 if self.coverage_plan.certificate is not None else None),
558             "cleared_channels": cleared,

```

```

559         "absent_channels": absent,
560         "unresolved_channels": unresolved,
561         "absence_evidence": {str(channel): self.absence_evidence[channel]
562                             for channel in sorted(absent)},
563         "redundant_rescan_measures": self.redundant_rescan_measures,
564         "measures": stats["measures"],
565         "clears": stats["clears"],
566         "clear_success": stats["cleared"],
567         "total_requests": stats["measures"] + stats["clears"],
568         "move_distance_m": round(stats["distance_m"], 1),
569         # 题面口径: 定位清除总时间 (全程虚拟时长) / 被清除干扰源个数。
570         "avg_locate_clear_time_s": (round(virtual_time_s / len(cleared), 3)
571                                   if cleared else None),
572         "step_totals": steps["totals"],
573         "steps": steps["rows"],
574         "anomalies": self.anomalies,
575         "virtual_recompute_mismatches": len(report["mismatches"]),
576         "channel_status": self._channel_status_snapshot(),
577     }
578     if self._abort_detail is not None:
579         summary["abort_detail"] = _error_brief(self._abort_detail)
580     return summary
581
582 def _step_report(self) -> dict:
583     """每步计时汇总 [用户决策 2026-09-11]: 实机墙钟 + 比赛虚拟耗时分项。"""
584     totals = {"count": 0, "real_elapsed_total_s": 0.0,
585              "real_since_enter_total_s": 0.0,
586              "virtual_move_s": 0.0, "virtual_switch_s": 0.0,
587              "virtual_measure_action_s": 0.0, "virtual_clear_action_s": 0.0,
588              "virtual_total_s": 0.0}
589     rows = []
590     for index, step in enumerate(self.client.log.step_events()):
591         cost = step["virtual_cost"]
592         totals["count"] += 1
593         totals["real_elapsed_total_s"] += step["real_elapsed_s"]
594         totals["real_since_enter_total_s"] = step.get(
595             "real_since_enter_s", totals["real_since_enter_total_s"])
596         totals["virtual_move_s"] += cost["move_s"]
597         totals["virtual_switch_s"] += cost["switch_s"]
598         if step["path"] == "/measure":
599             totals["virtual_measure_action_s"] += cost["action_s"]
600         elif step["path"] == "/clear":
601             totals["virtual_clear_action_s"] += cost["action_s"]
602         totals["virtual_total_s"] += cost["total_s"]
603         rows.append({"index": index, "action": step["path"].rstrip("/"),
604                    "result": step.get("result"),
605                    "channel": step.get("channel"),
606                    "position": step.get("position"),
607                    "real_elapsed_ms": round(step["real_elapsed_s"] * 1000.0, 3),
608                    "real_since_enter_s": step.get("real_since_enter_s"),
609                    "move_s": round(cost["move_s"], 3),
610                    "switch_s": round(cost["switch_s"], 3),
611                    "action_s": round(cost["action_s"], 3),
612                    "step_virtual_s": round(cost["total_s"], 3),
613                    "virtual_time_s": step["virtual_time_s"]})
614     for key, value in totals.items():
615         if isinstance(value, float):
616             totals[key] = round(value, 3)
617     return {"totals": totals, "rows": rows}
618
619 def _channel_status_snapshot(self) -> dict:
620     """逐频道状态快照: 中断收尾时的未完成状态留档 (每频道一条)。"""
621     snapshot = {}
622     for channel, belief in sorted(self.belief.channels.items()):
623         entry = {"status": belief.status.value}
624         if belief.region is not None:
625             entry["posterior_radius_m"] = round(belief.region.mec()[1], 3)
626         snapshot[str(channel)] = entry
627     return snapshot
628

```

```

629
630 class DirectionalExtension(OmnidirectionalLoop):
631     """问题4混合源闭环（P5）：全向基线 + 定向观测处理 + 后备清除。
632
633     与 P4 的三点差异（协议衔接见 docs/协议规则表与状态机.md §7）：
634     - 49 点覆盖的发现保证对定向源同样成立，覆盖阶段不区分源类型；
635     - 保证接收盘内的 no_signal 只能由朝向遮挡解释 [观测推理]：该频道
636       记为"观测到定向"，随后对 MEC 圆心取对跖点复核；联合约束基线
637       不据此删除空间候选（真源不误删）；
638     - 定位停滞（候选集有界但未达清除证书）时构造  $\leq 20\text{m}$  有限圆盘覆盖，
639       预算核对了逐格 /clear：清除与朝向无关 [题面]，是定向源不依赖
640       可见性的完备后备。格数超硬顶或现实预算不足则跳过并留痕，绝不盲扫。
641     """
642
643     def __init__(self, client: SimulatorClient,
644                 config: ProblemConfig = ProblemConfig()):
645         super().__init__(client, config)
646         # P0-2：第四问切换到同一保证下的最小命中集（默认 q4_mincover_v1）。
647         self.coverage_plan = plan_instance(config.search.q4_coverage_plan,
648                                           verify=config.search.verify_coverage_plan)
649         self.ledger = CoverageLedger(config, points=list(self.coverage_plan.points))
650         self.directional_observed: list[int] = [] # 观测到定向性的频道
651         self.blocked_probes: dict[int, list[Vec2]] = {}
652         self.fallback_log: list[dict] = [] # 后备清除逐次留痕
653         # P0-3：（位置 × 朝向）联合可行集，仅在频道首次被看见时惰性建立，
654         # 使其初始化与更新成本只落在"确实存在定向源"的频道上。
655         self.joint: dict[int, JointBelief] = {}
656         self.joint_absent_channels: list[int] = []
657         self.visibility_probes = 0 # 由联合信念选出的探点数
658
659     def _coverage_phase(self) -> None:
660         """49站的定向覆盖证明不变：每站先串行扫频，再就近服务已发现源。
661
662         避免检测一个源后远行清除，再返回同站检测下一个频道。
663         no_signal不用于删空间候选，也不使用第三问的七站缺席证明。
664         """
665         while not self._count_limit_reached() and self.stop_reason is None:
666             index = self._next_point_index()
667             if index is None:
668                 return
669             point = self.ledger.points[index]
670             pending = self._pending_channels_at(index)
671             pending.sort(key=lambda c: (c != self.client.state.channel, c))
672             for channel in pending:
673                 if self._virtual_cap_reached():
674                     return
675                 outcome = self.client.measure(point, channel)
676                 self.ledger.mark_measured(channel, index)
677                 self._observe(channel, point, outcome.result, outcome.svd_deg)
678                 if outcome.result == NEAR:
679                     cert = clear_certificate(
680                         self.belief.candidate_set(channel), channel, self.config)
681                     if cert.certified:
682                         self._attempt_clear(channel, point, cert)
683                 if self._count_limit_reached():
684                     return
685             queue = self.belief.channels_by_status(ChannelStatus.DETECTED)
686             while queue and self.stop_reason is None:
687                 channel = min(queue, key=lambda c: (
688                     math.dist(self.client.state.position,
689                               self.belief.candidate_set(c).mec()[0]), c))
690                 queue.remove(channel)
691                 self._localize_and_clear(channel)
692                 if self._count_limit_reached():
693                     return
694
695     # ----- 定位与清除
696     def _localize_and_clear(self, channel: int) -> bool:
697         """定位子循环：选点测量收敛候选集，拿到清除证书后清除。
698

```

```

699 与 P4 的差异: 保证接收盘内的 no_signal 说明探点被朝向遮挡
700 [观测推理], 走"对跖复核 -> 环探"升级梯子 (不计入定位步数,
701 由 max_blocked_probes 限流), 之后仍未收敛再转后备清除。
702 """
703 search = self.config.search
704 self._may_clear.add(channel)
705 steps = 0
706 while steps < search.max_localize_steps:
707     if self._virtual_cap_reached():
708         return False
709     region = self.belief.candidate_set(channel)
710     if region is None:
711         return False
712     cert = clear_certificate(region, channel, self.config)
713     if cert.certified and channel in self._may_clear:
714         if self._attempt_clear(channel, cert.center, cert):
715             return True
716         self._may_clear.discard(channel) # 证书失败: 补测后才可重试
717         continue
718     if len(self.blocked_probes.get(channel, [])) >= search.max_blocked_probes:
719         break # 梯子用尽仍被遮挡: 定位停滞, 转后备清除
720     point = self._next_probe(region, channel)
721     within_range = (math.dist(point, region.mec()[0])
722                     <= safe_disk_radius(region, self.config))
723     outcome = self.client.measure(point, channel)
724     self._observe(channel, point, outcome.result, outcome.svd_deg)
725     self._may_clear.add(channel)
726     if outcome.result == NO_SIGNAL:
727         if within_range and channel not in self.directional_observed:
728             self.directional_observed.append(channel)
729             self.blocked_probes.setdefault(channel, []).append(point)
730             continue # 盘外无信号也可能是距离不足, 不据此断言定向
731             self.blocked_probes.pop(channel, None)
732             steps += 1
733     if self.belief.channels[channel].status is ChannelStatus.CLEARED:
734         return True
735     region = self.belief.candidate_set(channel)
736     if region is None:
737         return False
738     cert = clear_certificate(region, channel, self.config)
739     if cert.certified and channel in self._may_clear:
740         if self._attempt_clear(channel, cert.center, cert):
741             return True
742         self._may_clear.discard(channel)
743     return self._fallback_clear(channel, region)
744
745 def _observe(self, channel: int, point: Vec2, result: str,
746             svd_deg: float | None) -> None:
747     """同步更新凸候选集与 (位置 × 朝向) 联合可行集。
748
749     联合集只在频道首次被看见 (direction/near) 时建立: 从未被看见的
750     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
751     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。
752     """
753     self.belief.update_measure(channel, point, result, svd_deg)
754     if not self.config.search.directional_joint_belief:
755         return
756     joint = self.joint.get(channel)
757     if joint is None:
758         if result not in (DIRECTION, NEAR):
759             return
760         joint = JointBelief(self.config)
761         self.joint[channel] = joint
762     joint.apply(point, result, svd_deg)
763     if joint.is_absent() and self.belief.channels[channel].status \
764         is not ChannelStatus.CLEARED:
765         self.belief.mark_absent(channel)
766         self.absence_evidence[channel] = "joint_empty"
767         if channel not in self.joint_absent_channels:
768             self.joint_absent_channels.append(channel)

```

```

769
770 def _probe_candidates(self, region: CandidateRegion, channel: int) -> list[Vec2]:
771     """可见性选点的候选点族：保证接收点 + 对跖点 + 环探 + **抵近点**。
```

772

773 D1 关键结构性改动：旧点族全部围绕 MEC 圆心固定环采样，远距离不收敛时

774 没有"主动靠近"这一动作，只能退化到 20m 网格盲清。现在补上

775 ``center +  $\alpha \cdot (\text{当前位置} - \text{center})$ `` ($\alpha \in \text{approach_fractions}$, $\alpha=0$ 即圆心)

776 的抵近序列，使策略可以在保持可见的前提下逐次逼近真源，最终以正常

777 角域证书或 near 收尾。

778 """

779 center, radius = region.mec()

780 position = self.client.state.position

781 candidates = [select_measurement_point(

782 region, position, self.config).position]

783 safe = safe_disk_radius(region, self.config)

784 blocked = self.blocked_probes.get(channel, [])

785 if blocked and safe >= 0:

786 dx, dy = center[0] - blocked[0][0], center[1] - blocked[0][1]

787 norm = math.hypot(dx, dy)

788 if norm > 1e-9:

789 scale = min(1.0, safe / norm)

790 candidates.append((center[0] + dx * scale, center[1] + dy * scale))

791 # 抵近点：从 MEC 圆心到当前点的插值序列（含圆心本身）。

792 for fraction in self.config.search.approach_fractions:

793 candidates.append((center[0] + (position[0] - center[0]) * fraction,

794 center[1] + (position[1] - center[1]) * fraction))

795 for factor in (0.5, 1.0, 1.5):

796 ring = max(radius * factor, 2.0 * self.config.field.near_radius_m)

797 for step in range(8):

798 angle = math.pi * step / 4.0

799 candidates.append((center[0] + ring * math.cos(angle),

800 center[1] + ring * math.sin(angle)))

801 unique: list[Vec2] = []

802 seen: set[tuple[int, int]] = set()

803 blocked_keys = {(int(round(point[0] * 2)), int(round(point[1] * 2)))

804 for point in blocked} \

805 if self.config.search.probe_exclude_blocked else set()

806 for point in candidates:

807 key = (int(round(point[0] * 2)), int(round(point[1] * 2)))

808 if key in seen or key in blocked_keys:

809 continue # 已测过且被遮挡的点：再测不会有任何新信息

810 seen.add(key)

811 unique.append(point)

812 return unique

813

814 def _visibility_probe(self, region: CandidateRegion, channel: int) -> Vec2 | None:

815 """按联合信念选探点；联合信念不可用时返回 None（退回梯子）。

816

817 三种口径（只影响选点顺序，不承担任何安全责任）：

818

819 - ``nearest\_visible``（默认）：先按可见概率过滤（只保留

820 ``prob\_detect  $\geq$  ratio \* max`` 的候选，避免选中"实际必然被朝向遮挡"的

821 点），再取 ``当前位置+q+MEC 圆心`` 移动最省者。9 场景 core 实测：

822 后备盲清 125.7 $\rightarrow$ 4.3 次/局、T 9225 $\rightarrow$ 8602 s、墙钟最低。

823 - ``expected\_radius``：在同样过滤后最小化

824 ``E[后验 MEC 半径](q) +  $\lambda \cdot \text{移动}$ ``。 $\lambda=5$ 时与上面几乎同分（8604 s），

825 λ 很小时（0.05）值函数主导，为信息多绕路（9740 s）——本题

826 "看得见且少走路"比"单步信息量最大"更重要。

827 - ``visibility``（旧口径）：最大化 ``prob\_detect/(移动+6)``，会反复选中

828 同一个近处被遮挡点（实测 9225 s、后备 125.6 次）。

829 """

830 joint = self.joint.get(channel)

831 if joint is None or joint.is_absent():

832 return None

833 position = self.client.state.position

834 service_s = self.config.cost.measure_s + self.config.cost.switch_s

835 candidates = self._probe_candidates(region, channel)

836 if not candidates:

837 return None

838 mode = self.config.search.probe_score_mode

```

839     if mode == "visibility":
840         best_point, best_score = None, None
841         for point in candidates:
842             probability = joint.prob_detect(point)
843             if probability <= 0.0:
844                 continue
845             score = -(probability / (move_time(position, point, self.config.cost)
846                                     + service_s))
847             if best_score is None or score < best_score:
848                 best_point, best_score = point, score
849         if best_point is not None:
850             self.visibility_probes += 1
851         return best_point
852     scored = [(joint.prob_detect(point), point) for point in candidates]
853     peak = max(probability for probability, _ in scored)
854     ratio = self.config.search.probe_visibility_ratio
855     keep = [point for probability, point in scored
856             if probability >= ratio * peak] or [point for _, point in scored]
857     center = region.mec()[0]
858     weight = self.config.search.probe_lambda_move_s
859
860     def approach_cost(point: Vec2) -> float:
861         return (move_time(position, point, self.config.cost)
862                + move_time(point, center, self.config.cost) + service_s)
863
864     if mode == "nearest_visible":
865         #  $\lambda$  很大时值函数项对 argmin 已无影响: 直接取"可见候选中移动最省",
866         # 省掉每个候选一次期望后验半径计算 (单局墙钟 -3 s)。
867         best = min(keep, key=lambda point: (approach_cost(point), point))
868         self.visibility_probes += 1
869         return best
870     best = min(keep, key=lambda point: (
871         joint.expected_posterior_radius(point) + weight * approach_cost(point),
872         point))
873     self.visibility_probes += 1
874     return best
875
876
877
878 def _next_probe(self, region: CandidateRegion, channel: int) -> Vec2:
879     """定向遮挡下的探点升级梯子 [工程]。
880
881     - 未被遮挡过: worst-posterior 选点器 (保证接收盘内);
882     - 第 1 次被遮挡: 对 MEC 圆心取对跖方向, 偏移裁进保证接收盘;
883     - 之后: 绕 MEC 圆心、半径 =  $\rho$  的环探, 方位自首个被遮挡点起
884       每次 +45°。几何边界必须如实陈述: 环上可见弧没有下界——
885       源可位于环上 ( $|g-c| = \rho$ ) 且朝向沿外切线, 可见集退化为单个
886       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,
887       不构成接收或命中保证; 完备性完全由后备清除兜底。
888     """
889     visibility = self._visibility_probe(region, channel)
890     if visibility is not None:
891         return visibility
892     blocked = self.blocked_probes.get(channel, [])
893     center, radius = region.mec()
894     if not blocked:
895         return select_measurement_point(region, self.client.state.position,
896                                       self.config).position
897     safe = safe_disk_radius(region, self.config)
898     first = blocked[0]
899     if len(blocked) == 1 and safe >= 0:
900         dx, dy = center[0] - first[0], center[1] - first[1]
901         norm = math.hypot(dx, dy)
902         if norm <= 1e-9:
903             # 首个被遮挡探点恰在圆心: 方向信息为零, 沿 +x 走满安全盘。
904             return (center[0] + safe, center[1])
905         scale = min(1.0, safe / norm)
906         # 对跖方向, 偏移模长 = min(norm, safe): 圆心对称点裁进保证接收盘。
907         return (center[0] + dx * scale, center[1] + dy * scale)
908     base = math.atan2(first[1] - center[1], first[0] - center[0])

```

```

909         azimuth = base + math.radians(45.0 * (len(blocked) - 1))
910         return (center[0] + radius * math.cos(azimuth),
911                 center[1] + radius * math.sin(azimuth))
912
913     def _fallback_clear(self, channel: int, region: CandidateRegion) -> bool:
914         """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)。"""
915         search, session = self.config.search, self.config.session
916         width = max(p[0] for p in region.polygon) - min(p[0] for p in region.polygon)
917         height = max(p[1] for p in region.polygon) - min(p[1] for p in region.polygon)
918         cells = fallback_clear_cells(region.polygon, self.config)
919         entry = {"channel": channel, "cells": len(cells),
920                 "bound": fallback_cell_bound(width, height, self.config),
921                 "posterior_radius_m": round(region.mec()[1], 3)}
922         self.fallback_log.append(entry)
923         if len(cells) > search.fallback_max_cells:
924             # 宽候选集扫描违背预算纪律: 留待定位收敛, 不得硬扫 [计划$4]。
925             entry["outcome"] = "skipped_cell_cap"
926             self._anomaly("fallback_cell_cap_exceeded", channel,
927                           cells=len(cells), cap=search.fallback_max_cells)
928             return False
929         # 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。
930         remaining = self.client.real_remaining_s
931         if remaining is None or len(cells) * session.real_rtt_p95_s \
932             + session.reserve_exit_s > remaining:
933             entry["outcome"] = "skipped_budget"
934             return False
935         attempts = 0
936         order = self._fallback_order(cells)
937         for cell in order:
938             if self._virtual_cap_reached():
939                 entry["outcome"] = "skipped_virtual_budget"
940                 entry["attempts"] = attempts
941                 return False
942             attempts += 1
943             outcome = self.client.clear(cell, channel)
944             if outcome.result == CLEAR_SUCCESS:
945                 self.belief.mark_cleared(channel)
946                 self.cleared.append(channel)
947                 entry["outcome"] = "cleared"
948                 entry["attempts"] = attempts
949                 return True
950             # 候选集保守包含真源, 格心距格内任一点  $\leq 10\sqrt{2} < 20m$ : 全部
951             # 失败说明守恒性被打破, 按异常留痕, 不得据此删除候选。
952             entry["outcome"] = "exhausted"
953             entry["attempts"] = attempts
954             self._anomaly("fallback_exhausted_without_success", channel,
955                           cells=len(cells))
956             return False
957
958     def _fallback_order(self, cells: list[Vec2]) -> list[Vec2]:
959         """后备清除的走格顺序: 最近邻打底 + 2-opt (可选 0r-opt) 改进。
960
961         旧实现每步重新 ``min(remaining, key=dist)`` , 等价于最近邻开放路径
962         (典型为最优的 1.2-1.3 倍) 且每次重扫  $O(k^2)$ 。这里一次性求解改进序
963         (D3), 把格间移动压到接近最优; 格数不变, 故完备性不变。
964         """
965         if not cells:
966             return []
967         if not self.config.search.fallback_optimize_tour or len(cells) < 4:
968             position = self.client.state.position
969             remaining = list(cells)
970             order: list[Vec2] = []
971             while remaining:
972                 nxt = min(remaining, key=lambda c: (math.dist(position, c), c))
973                 remaining.remove(nxt)
974                 order.append(nxt)
975                 position = nxt
976             return order
977         return improve_open_path([tuple(c) for c in cells],
978                                   self.client.state.position)

```

```

979
980 # ----- 收尾
981 def _summary(self, info) -> dict:
982     report = super()._summary(info)
983     report["directional_observed_channels"] = list(self.directional_observed)
984     report["fallback_clears"] = self.fallback_log
985     # 后备 /clear 请求计入总请求口径, 仅统计检测次数会低估现实耗时。
986     report["fallback_clear_attempts"] = sum(
987         entry.get("attempts", 0) for entry in self.fallback_log)
988     report["visibility_probes"] = self.visibility_probes
989     report["joint_absent_channels"] = list(self.joint_absent_channels)
990     report["joint_final"] = {str(channel): belief.summary()
991                             for channel, belief in sorted(self.joint.items())}
992     return report
993
994
995 def fallback_cell_bound(width_m: float, height_m: float,
996                        config: ProblemConfig = ProblemConfig()) -> int:
997     """外包框枚举的格数上界 ( $\lceil w/cell \rceil + 2$ ) ( $\lceil h/cell \rceil + 2$ ) (计划S4 规模检查)。"""
998     cell = config.search.fallback_clear_grid_m
999     return (math.ceil(width_m / cell) + 2) * (math.ceil(height_m / cell) + 2)
1000
1001
1002 def fallback_clear_cells(polygon: list[Vec2],
1003                         config: ProblemConfig = ProblemConfig()) -> list[Vec2]:
1004     """候选外包多边形的  $\leq 20m$  有限圆盘覆盖格心列表 (P5 后备清除)。
1005
1006     全局半开  $20m$  网格索引; 多边形与格子的相交判定带正容差 (宁可多测),
1007     多边形边界恰在格线上时两侧格子都保留, 因此格内任意点距某格心
1008      $\leq 10 \cdot \sqrt{2} < 20m$ , 逐格 /clear 至少一次命中。列表长度即 /clear
1009     请求预算上界; 先核对 fallback_cell_bound, 超预算不得逐格盲扫。
1010     """
1011     if not polygon:
1012         return []
1013     cell = config.search.fallback_clear_grid_m
1014     xs = [p[0] for p in polygon]
1015     ys = [p[1] for p in polygon]
1016     i0, i1 = math.floor(min(xs) / cell), math.floor(max(xs) / cell)
1017     j0, j1 = math.floor(min(ys) / cell), math.floor(max(ys) / cell)
1018     eps = 1e-9
1019     centers: list[Vec2] = []
1020     for i in range(i0, i1 + 1):
1021         for j in range(j0, j1 + 1):
1022             clipped = list(polygon)
1023             for a, b, c in ((1.0, 0.0, (i + 1) * cell + eps),
1024                             (-1.0, 0.0, -(i * cell - eps)),
1025                             (0.0, 1.0, (j + 1) * cell + eps),
1026                             (0.0, -1.0, -(j * cell - eps))):
1027                 clipped = clip_halfplane(clipped, a, b, c, tol=eps)
1028             if not clipped:
1029                 break
1030             if clipped:
1031                 centers.append(((i + 0.5) * cell, (j + 0.5) * cell))
1032     return centers

```

src/mcm26b/planner/bayes_loop.py

```

1  """P4b 贝叶斯网格滤波闭环 (问题3新策略 ``bayes``)。
2
3  在 :class:`OmnidirectionalLoop` 骨架 (run/异常收尾/summary/预算/异常留痕)
4  上的三层重构, 对应 docs/algo-3-bayes.md 的架构:
5
6  1. 感知层: 每频道一个网格幸存集 (particles.ChannelParticles), 全部读数
7     (含 no_signal 负观测) 即时更新; 幸存集为空即缺席证明, 无需等覆盖账本;
8  2. 决策层: 统一任务循环——站点扫频、定位测量、清除三类任务合成任务表,
9     开放路径最近邻 + 2-opt 排序, 每执行一个任务滚动重排 (动态 TSP);
10 3. 选点层: 定位测量点在保证接收盘 (凸证书) 或 PF MEC 探索环 (宽候选集)
11    内按"期望 log 幸存占比 +  $\lambda \cdot$  移动时间"评分 (信息增益贪心)。
12

```

```

13 与基线 SevenBatch 的差异：负观测跳站/跳测（必 no_signal 的（站,频道）对
14 不消耗检测与换频）、PF 缺席证明提前证毕、全局任务排序替代"站内最近邻"。
15 清除证书仍以凸证书为主（PF 网格证书仅在负观测挖洞使幸存集更紧时提前
16 触发），两者均满足 Voronoi/保守裕量，不牺牲全清率。
17 """
18
19 from __future__ import annotations
20
21 import math
22 from dataclasses import dataclass
23
24 from ..belief import CandidateRegion, ChannelStatus
25 from ..config import ProblemConfig, Vec2
26 from ..coverage import CoverageLedger
27 from ..particles import ChannelParticles, as_sample_array
28 from ..protocol.client import SimulatorClient
29 from ..protocol.rules import DIRECTION, NEAR, NO_SIGNAL
30 from ..protocol.timing import move_time
31 from . import (ClearCertificate, OmnidirectionalLoop, clear_certificate,
32               candidate_points, safe_disk_radius, select_measurement_point)
33 from .seven import validate_seven_coverage
34 from ..guarantee import seven_plan
35
36
37 @dataclass(frozen=True)
38 class Task:
39     """统一任务表的一项。service_s 为不含移动的固有耗时（检测+换频/清除）。"""
40
41     kind: str # "station" | "locate" | "clear"
42     position: Vec2
43     service_s: float
44     channel: int | None = None
45     station_index: int | None = None
46     station_channels: tuple[int, ...] = ()
47
48
49 def _point_key(point: Vec2) -> tuple[int, int]:
50     """候选点去重键：0.5m 网格取整（同一物理点不重复测量）。"""
51     return (int(round(point[0] * 2)), int(round(point[1] * 2)))
52
53
54 class BayesLoop(OmnidirectionalLoop):
55     """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。"""
56
57     def __init__(self, client: SimulatorClient,
58                 config: ProblemConfig = ProblemConfig()):
59         super().__init__(client, config)
60         validate_seven_coverage(config.bayes.station_ring_radius_m, config)
61         self.coverage_plan = seven_plan(config.bayes.station_ring_radius_m)
62         self.ledger = CoverageLedger(
63             config, points=list(self.coverage_plan.points))
64         self.particles = {channel: ChannelParticles(config)
65                          for channel in self.belief.channels}
66         self._measured_points: dict[int, set[tuple[int, int]]] = {}
67         self._clear_failed: set[int] = set() # 证书清除失败后须先补测再重试
68         self._stall: dict[int, int] = {} # 连续无收缩计数（异常留痕用）
69         self._skip_cache: dict[tuple[int, int], tuple[int, bool]] = {}
70         self.skipped_measures = 0 # 跳测规则省下的检测次数
71         self.no_shrink_measures = 0 # 未收缩幸存集的定位检测（诊断用）
72         self.pf_absent_channels: list[int] = [] # 由幸存集为空证明缺席的频道
73         self.opportunistic_measures = 0
74
75     # ----- 主循环
76     def _coverage_phase(self) -> None:
77         """统一任务循环：发现 + 定位 + 清除滚动调度，直到任务表为空。"""
78         while self.stop_reason is None:
79             if self._count_limit_reached() or self._virtual_cap_reached():
80                 return
81             if not self._step_once():
82                 return

```

```

83
84 def _step_once(self) -> bool:
85     """构建任务表、排序并执行首任务；无可执行任务返回 False。"""
86     if self.config.bayes.opportunistic_measure:
87         self._measure_at_stop()
88         tasks = self._build_tasks()
89         if not tasks:
90             return False
91         tasks = self._refine_locate_tasks(tasks)
92         order = self._order_tasks(tasks)
93         self._execute_task(order[0])
94         return True
95
96 def _measure_at_stop(self) -> None:
97     """利用已停靠位置补交会方位；评分只作选测，证书仍用完整角域。
98
99     只测尚未在此处测过、有望显著收缩且保证距离可接收的已发现频道。
100     逐频道调用measure，换频与5秒检测照常计费；不在移动中检测。
101     此距离保证仅适用于第三问，不能用于未知定向源。
102     """
103     point = self.client.state.position
104     channels = self.belief.channels_by_status(ChannelStatus.DETECTED)
105     channels.sort(key=lambda c: (c != self.client.state.channel, c))
106     for channel in channels:
107         if self._virtual_cap_reached():
108             return
109         if _point_key(point) in self._measured_points.get(channel, set()):
110             continue
111         cert = self._best_certificate(channel)
112         if cert is not None and cert.certified:
113             continue
114         region = self.belief.candidate_set(channel)
115         farthest = max(math.dist(point, v) for v in region.polygon)
116         if farthest > self.config.field.r_eff_min_m - 1e-3:
117             continue
118         radius = region.mec()[1]
119         vertices = region.polygon
120         stride = max(1, len(vertices) // self.config.bayes.mc_representatives)
121         radii = []
122         delta = (self.config.field.bearing_error_half_deg
123                 + self.config.search.bearing_rounding_margin_deg)
124         for target in vertices[::stride]:
125             theta = math.degrees(math.atan2(target[1]-point[1], target[0]-point[0]))
126             probe = CandidateRegion(list(vertices), region.slack_m)
127             probe.clip_wedge(point, theta, delta)
128             radii.append(probe.mec()[1] if probe.polygon else radius)
129         if sum(radii) / len(radii) > radius * self.config.bayes.opportunistic_radius_ratio:
130             continue
131         outcome = self.client.measure(point, channel)
132         self._record_measure(channel, point, outcome)
133         self._may_clear.add(channel)
134         self.opportunistic_measures += 1
135
136 # ----- 任务构建
137 def _station_pending(self, index: int) -> list[int]:
138     """站点待测频道：UNKNOWN、未测过、非"必 no_signal"且未证缺席。"""
139     point = self.ledger.points[index]
140     pending = []
141     for channel, belief in self.belief.channels.items():
142         if belief.status is not ChannelStatus.UNKNOWN:
143             continue
144         if self.ledger.measured_at(channel, index):
145             continue
146         if self._skip_station_channel(channel, index, point):
147             continue
148         pending.append(channel)
149     return pending
150
151 def _fused_channels_at(self, index: int) -> list[int]:
152     """本站要测的频道 = 未发现(UNKNOWN) + 已发现但尚未定证(DETECTED)。

```

```

153
154 P0-1: 把定位并入覆盖巡回。站点三角化可行性 (方案 §1.3 F11) :
155 99.28% 的源位置有  $\geq 2$  个站在  $r_{\text{eff\_max}}=1500\text{m}$  内, 最佳交会角中位
156  $122.9^\circ$ 、5 分位  $58.5^\circ$ 、因此"顺路取第二条方位线"对绝大多数源成立。
157 已有证书、已在同一  $0.5\text{m}$  测点测过、或该点必 no_signal 的频道跳过。
158
159 实测结论 (2026-09-12, `output/suite/q3-fuse-ab*`) : 本机制**不划算**。
160 七站环上通常只有 2-3 个站能对该源取得有效方位线, 交会角也不足以
161 直接换到 20m 证书, 专用信息增益探点仍然必要: 仅在站点顺带测量等于
162 平白多花约 11 次检测 ( $+0.8\%$ )、若同时抑制专用探点则  $+45.6\%$  (20/20
163 场景更慢)。默认关闭, 保留代码以便复现与后续研究。
164 """
165 point = self.ledger.points[index]
166 channels = self._station_pending(index)
167 selected = set(channels)
168 if not self.config.bayes.fuse_localize_into_tour:
169     return channels
170 for channel in self.belief.channels_by_status(ChannelStatus.DETECTED):
171     if channel in selected or self.ledger.measured_at(channel, index):
172         continue
173     if _point_key(point) in self._measured_points.get(channel, set()):
174         continue
175     cert = self._best_certificate(channel)
176     if cert is not None and cert.certified:
177         continue # 已有证书: 交给清除任务, 不再多测
178     if self._skip_station_channel(channel, index, point):
179         continue # 幸存集都在 1500m 外: 该点必 no_signal
180     channels.append(channel)
181     selected.add(channel)
182 return channels
183
184 def _skip_station_channel(self, channel: int, index: int, point: Vec2) -> bool:
185     """跳测规则 (按 pf.version 缓存; 幸存集单调收缩, 判定单调)。"""
186     pf = self.particles[channel]
187     if not pf:
188         return True
189     cached = self._skip_cache.get((channel, index))
190     if cached is not None and cached[0] == pf.version:
191         return cached[1]
192     value = pf.guaranteed_no_signal(point)
193     self._skip_cache[(channel, index)] = (pf.version, value)
194     return value
195
196 def _build_tasks(self) -> list[Task]:
197     tasks: list[Task] = []
198     for index, point in enumerate(self.ledger.points):
199         pending = self._fused_channels_at(index)
200         if not pending:
201             continue
202         switches = len(pending) - (1 if self.client.state.channel in pending else 0)
203         tasks.append(Task("station", tuple(point),
204             self.config.cost.measure_s * len(pending)
205             + self.config.cost.switch_s * max(switches, 0),
206             station_index=index, station_channels=tuple(pending)))
207     for channel in self.belief.channels_by_status(ChannelStatus.DETECTED):
208         cert = self._best_certificate(channel)
209         if cert is not None and cert.certified \
210             and channel not in self._clear_failed:
211             tasks.append(Task("clear", cert.center,
212                 self.config.cost.clear_success_s,
213                 channel=channel))
214         continue
215     if self._defer_to_station(channel):
216         continue # 下一条方位线可由"本就要去的站点"顺路提供
217     point = self._select_point(channel)
218     if point is None:
219         continue
220     tasks.append(Task("locate", point,
221         self.config.cost.measure_s + self.config.cost.switch_s,
222         channel=channel))

```

```

223         return tasks
224
225     def _defer_to_station(self, channel: int) -> bool:
226         """该频道能否由"本就要去的站点"提供下一条方位线。
227
228         P0-1 的关键: 仅仅在站点顺带测量是不够的 (那只是多做一次检测),
229         必须同时**抑制专用探点任务**, 否则滚动排序仍会优先飞向更近的探点。
230         判据要求该站本来就有 UNKNOWN 频道待测 (覆盖巡回必须去), 因此不会
231         为了等一条方位线而制造新的绕行, 也不会出现"无任务可做"的死锁:
232         站点扫完后 ``_station_pending`` 为空, 探点任务自然恢复。
233         """
234         if not self.config.bayes.fuse_localize_into_tour:
235             return False
236         for index, point in enumerate(self.ledger.points):
237             if not self._station_pending(index):
238                 continue
239             if self.ledger.measured_at(channel, index):
240                 continue
241             if _point_key(point) in self._measured_points.get(channel, set()):
242                 continue
243             if self._skip_station_channel(channel, index, point):
244                 continue
245             return True
246         return False
247
248     # ----- 任务排序
249     def _order_tasks(self, tasks: list[Task]) -> list[Task]:
250         """开放路径排序: 小任务表用精确 DP, 否则最近邻 + 2-opt。
251
252         Held-Karp (S5/T1) 把"已访问集合 × 结束时测向频道"并入状态, 因此换频
253         代价是精确的, 而不是 :meth:`_path_cost` 里"按频道排序"的贪心近似。
254         """
255         if len(tasks) <= 2:
256             ordered = sorted(tasks, key=lambda t: (
257                 move_time(self.client.state.position, t.position, self.config.cost),
258                 t.kind, t.channel or 0))
259             return min((ordered, ordered[::-1]), key=self._path_cost)
260         limit = self.config.bayes.exact_order_limit
261         if 0 < limit and len(tasks) <= limit:
262             exact = self._exact_order(tasks)
263             if exact is not None:
264                 return exact
265         order = self._nearest_order(tasks)
266         if not self.config.bayes.two_opt_enabled:
267             return order
268         return self._two_opt(order)
269
270     def _service_and_exit(self, task: Task,
271                          channel: int) -> tuple[float, int]:
272         """任务在进入频道 ``channel`` 时的固有耗时与离开后的测向频道。"""
273         cost = self.config.cost
274         if task.kind == "locate":
275             return (cost.measure_s + (cost.switch_s if channel != task.channel else 0.0),
276                     task.channel)
277         if task.kind == "station" and task.station_channels:
278             pending = sorted(task.station_channels,
279                             key=lambda c: (c != channel, c))
280             switches = len(pending) - (1 if channel in pending else 0)
281             return (cost.measure_s * len(pending) + cost.switch_s * switches,
282                     pending[-1])
283         return task.service_s, channel
284
285     def _exact_order(self, tasks: list[Task]) -> list[Task] | None:
286         """Held-Karp 精确开放路径 (起点固定、终点自由, 含换频状态)。
287
288         状态 ``(mask, c)`` = 已访问任务集 mask、离开最后一个任务后测向频道为 c;
289         频道取值域只需 {当前频道} ∪ {locate 频道} ∪ {station 离开频道}。
290         规模 ``2^n · n · |C|``; n 超过 :attr:`BayesConfig.exact_order_limit` 时由
291         调用方回退启发式。返回 ``None`` 表示状态过大 (防御性)。
292         """

```

```

293     n = len(tasks)
294     if n <= 1:
295         return list(tasks)
296     start = self.client.state.position
297     init = self.client.state.channel
298     channels = {init}
299     for task in tasks:
300         if task.kind == "locate":
301             channels.add(task.channel)
302         elif task.kind == "station" and task.station_channels:
303             # 离开频道 = 按"当前频道优先"排序后的最后一个：通常是最大值；
304             # 进入频道恰为最大值时退化为次大值，故两者都要进状态空间。
305             ordered = sorted(task.station_channels)
306             channels.add(ordered[-1])
307             if len(ordered) > 1:
308                 channels.add(ordered[-2])
309     channels = sorted(channels)
310     width = len(channels)
311     if width * (1 << n) * n > 4_000_000:
312         return None
313     index_of = {channel: i for i, channel in enumerate(channels)}
314     c_init = index_of[init]
315     service: list[list[float]] = []
316     exits: list[list[int]] = []
317     for task in tasks:
318         row_service, row_exit = [], []
319         for channel in channels:
320             value, exit_channel = self._service_and_exit(task, channel)
321             row_service.append(value)
322             row_exit.append(index_of[exit_channel])
323         service.append(row_service)
324         exits.append(row_exit)
325     move = [[move_time(tasks[i].position, tasks[j].position, self.config.cost)
326              for j in range(n)] for i in range(n)]
327     from_start = [move_time(start, task.position, self.config.cost)
328                   for task in tasks]
329     inf = float("inf")
330     # dp[(mask, ci)] = 最小代价; parent[(mask, ci)] = (上一 mask, 上一 ci)
331     dp: dict[tuple[int, int], float] = {}
332     last_task: dict[tuple[int, int], int] = {}
333     parent: dict[tuple[int, int], tuple[int, int] | None] = {}
334     for k in range(n):
335         key = (1 << k, exits[k][c_init])
336         value = from_start[k] + service[k][c_init]
337         if value < dp.get(key, inf):
338             dp[key] = value
339             last_task[key] = k
340             parent[key] = None
341     for mask in range(1, 1 << n):
342         for ci in range(width):
343             state = (mask, ci)
344             cost = dp.get(state)
345             if cost is None:
346                 continue
347             for k in range(n):
348                 if (mask >> k) & 1:
349                     continue
350                 nxt = (mask | (1 << k), exits[k][ci])
351                 value = cost + move[last_task[state]][k] + service[k][ci]
352                 if value < dp.get(nxt, inf) - 1e-12:
353                     dp[nxt] = value
354                     last_task[nxt] = k
355                     parent[nxt] = state
356     full = (1 << n) - 1
357     best_key, best_cost = None, inf
358     for ci in range(width):
359         state = (full, ci)
360         if state in dp and dp[state] < best_cost:
361             best_key, best_cost = state, dp[state]
362     if best_key is None:

```

```

363         return None
364     order: list[Task] = []
365     key: tuple[int, int] | None = best_key
366     while key is not None:
367         order.append(tasks[last_task[key]])
368         key = parent[key]
369     order.reverse()
370     return order
371
372
373
374 def _nearest_order(self, tasks: list[Task]) -> list[Task]:
375     remaining = list(tasks)
376     order = []
377     position = self.client.state.position
378     while remaining:
379         best = min(remaining, key=lambda t: (
380             move_time(position, t.position, self.config.cost),
381             t.kind, t.channel or 0))
382         remaining.remove(best)
383         order.append(best)
384         position = best.position
385     return order
386
387 def _path_cost(self, order: list[Task]) -> float:
388     total = 0.0
389     position = self.client.state.position
390     channel = self.client.state.channel
391     for task in order:
392         total += move_time(position, task.position, self.config.cost)
393         if task.kind == "locate":
394             total += self.config.cost.measure_s
395             total += self.config.cost.switch_s * (channel != task.channel)
396             channel = task.channel
397         elif task.kind == "station" and task.station_channels:
398             pending = sorted(
399                 task.station_channels,
400                 key=lambda c, channel=channel: (c != channel, c))
401             total += self.config.cost.measure_s * len(pending)
402             total += self.config.cost.switch_s * (len(pending) - (channel in pending))
403             channel = pending[-1]
404         else:
405             total += task.service_s # 清除不改变测向频道
406             position = task.position
407     return total
408
409 def _two_opt(self, order: list[Task]) -> list[Task]:
410     best = list(order)
411     best_cost = self._path_cost(best)
412     for _ in range(self.config.bayes.two_opt_passes):
413         improved = False
414         for i in range(len(best) - 1):
415             for j in range(i + 1, len(best)):
416                 candidate = best[:i] + best[i:j + 1][::-1] + best[j + 1:]
417                 cost = self._path_cost(candidate)
418                 if cost < best_cost - 1e-6:
419                     best, best_cost, improved = candidate, cost, True
420             if not improved:
421                 break
422     return best
423
424 # ----- 任务执行
425 def _execute_task(self, task: Task) -> None:
426     if task.kind == "station":
427         self._execute_station(task.station_index)
428     elif task.kind == "clear":
429         cert = self._best_certificate(task.channel)
430         if cert is not None and cert.certified:
431             if not self._attempt_clear(task.channel, cert.center, cert):
432                 self._clear_failed.add(task.channel)

```

```

433         self._may_clear.discard(task.channel)
434     else:
435         self._clear_failed.discard(task.channel)
436 elif task.kind == "locate":
437     self._execute_locate(task.channel, task.position)
438
439 def _execute_station(self, index: int) -> None:
440     """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。
441
442     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
443     途中顺路取得；证书一旦成立即就地清除，不额外绕行。
444     """
445     point = self.ledger.points[index]
446     pending = self._fused_channels_at(index)
447     unknown = [c for c, b in self.belief.channels.items()
448                if b.status is ChannelStatus.UNKNOWN]
449     self.skipped_measures += len(unknown) - len(
450         [c for c in pending if self.belief.channels[c].status
451          is ChannelStatus.UNKNOWN])
452     # 当前频道优先，减少换频；同站 near 直接清除不打破扫频
453     pending.sort(key=lambda c: (c != self.client.state.channel, c))
454     for channel in pending:
455         if self.stop_reason is not None or self._count_limit_reached() \
456             or self._virtual_cap_reached():
457             return
458         detected_before = (self.belief.channels[channel].status
459                             is ChannelStatus.DETECTED)
460         outcome = self.client.measure(point, channel)
461         self.ledger.mark_measured(channel, index)
462         self._record_measure(channel, point, outcome)
463         if outcome.result in (DIRECTION, NEAR):
464             self._may_clear.add(channel)
465         if outcome.result == NEAR:
466             region = self.belief.candidate_set(channel)
467             cert = clear_certificate(region, channel, self.config)
468             if cert.certified:
469                 if self._attempt_clear(channel, point, cert):
470                     self._clear_failed.discard(channel)
471             else:
472                 self._clear_failed.add(channel)
473                 self._may_clear.discard(channel)
474         else:
475             self._anomaly("near_without_certificate", channel, position=point)
476         continue
477     if not detected_before:
478         continue
479     if outcome.result == NO_SIGNAL:
480         # 该频道已被发现：保证接收盘内不可能 no_signal，属守恒性异常
481         region = self.belief.candidate_set(channel)
482         safe = (safe_disk_radius(region, self.config)
483                 if region is not None else -1.0)
484         if safe >= 0 and math.dist(point, region.mec()[0]) <= safe:
485             self._anomaly("no_signal_in_safe_disk", channel,
486                           position=point)
487         continue
488     cert = self._best_certificate(channel)
489     if cert is not None and cert.certified:
490         if self._attempt_clear(channel, cert.center, cert):
491             self._clear_failed.discard(channel)
492         else:
493             self._clear_failed.add(channel)
494             self._may_clear.discard(channel)
495
496 def _execute_locate(self, channel: int, point: Vec2) -> None:
497     """单步定位测量：执行后立即重排（滚动时域），收敛则顺路清除。"""
498     pf = self.particles[channel]
499     before = len(pf)
500     outcome = self.client.measure(point, channel)
501     self._record_measure(channel, point, outcome)
502     self._may_clear.add(channel)

```

```

503         if outcome.result == NO_SIGNAL:
504             region = self.belief.candidate_set(channel)
505             if region is not None and safe_disk_radius(region, self.config) >= 0 \
506                 and math.dist(point, region.mec()[0]) \
507                 <= safe_disk_radius(region, self.config):
508                 self._anomaly("no_signal_in_safe_disk", channel, position=point)
509         after = len(pf)
510         cert = self._best_certificate(channel)
511         if cert is not None and cert.certified:
512             if self._attempt_clear(channel, cert.center, cert):
513                 self._clear_failed.discard(channel)
514                 self._stall.pop(channel, None)
515                 return
516             self._clear_failed.add(channel)
517             self._may_clear.discard(channel)
518         if after == before:
519             self._stall[channel] = self._stall.get(channel, 0) + 1
520             self.no_shrink_measures += 1 # 只是浪费不破坏守恒性, 不进异常表
521         else:
522             self._stall[channel] = 0
523
524     # ----- 感知与证书
525     def _record_measure(self, channel: int, position: Vec2, outcome) -> None:
526         """belief (凸证书) 与 PF (幸存集) 同步更新; PF 空即证缺席。"""
527         self.belief.update_measure(channel, position, outcome.result,
528                                   outcome.svd_deg)
529         # 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%
530         # 的虚拟时间耗在空复扫上, 本计数是该问题的回归指标。
531         key = _point_key(position)
532         seen = self._measured_points.setdefault(channel, set())
533         if key in seen:
534             self.redundant_rescan_measures += 1
535         seen.add(key)
536         self._clear_failed.discard(channel) # 新证据落地: 允许再次尝试清除
537         belief = self.belief.channels[channel]
538         if belief.status in (ChannelStatus.CLEARED, ChannelStatus.ABSENT):
539             return # 已证毕: 清除/缺席后的 no_signal 与距离无关, 不进 PF
540         pf = self.particles[channel]
541         pf.apply_measure(position, outcome.result, outcome.svd_deg)
542         if not pf:
543             if belief.status is ChannelStatus.UNKNOWN:
544                 self.belief.mark_absent(channel)
545                 self.absence_evidence[channel] = "pf_empty"
546                 if channel not in self.pf_absent_channels:
547                     self.pf_absent_channels.append(channel)
548             else:
549                 self._anomaly("pf_empty_while_detected", channel, position=position)
550         self._apply_count_bound()
551
552     def _apply_count_bound(self) -> None:
553         """存在证据达到题面源数上界时, 其余 UNKNOWN 可由计数直接判缺席。
554
555         依据 [题面]: 每频道至多一个源、全局限定 10..16 个源。一旦
556         DETECTED/CLEARED 的不同频道数达到 16、任何尚未出现有效信号的
557         频道都不可能是源, 无须等 7 站覆盖账本跑完。该判定与位置无关,
558         是严格的计数证明; evidence 与 coverage/pf_empty 分开留痕。
559         """
560         present = [channel for channel, belief in self.belief.channels.items()
561                    if belief.status in (ChannelStatus.DETECTED,
562                                         ChannelStatus.CLEARED)]
563         if len(present) < self.config.field.source_count_max:
564             return
565         for channel, belief in self.belief.channels.items():
566             if belief.status is ChannelStatus.UNKNOWN:
567                 belief.status = ChannelStatus.ABSENT
568                 self.absence_evidence[channel] = "count_bound"
569
570     def _best_certificate(self, channel: int) -> ClearCertificate | None:
571         """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。"""
572         region = self.belief.candidate_set(channel)

```

```

573     cert = clear_certificate(region, channel, self.config) \
574         if region is not None else None
575     if cert is not None and cert.certified:
576         return self._near_clear_target(cert)
577     if self.config.bayes.grid_cert_enabled:
578         pf = self.particles[channel]
579         if len(pf):
580             center, radius = pf.mec()
581             limit = (self.config.field.clear_radius_m - pf.margin_m
582                     - self.config.search.clear_margin_m)
583             if radius <= limit:
584                 return self._near_clear_target(
585                     ClearCertificate(channel, center, radius, limit, True))
586     return cert
587
588 def _near_clear_target(self, cert: ClearCertificate) -> ClearCertificate:
589     """沿当前来向利用20m清除容差：三角不等式保留整集覆盖证明。"""
590     position = self.client.state.position
591     distance = math.dist(position, cert.center)
592     shift = min(distance, max(0.0, cert.limit_m - cert.radius_m - 1e-6))
593     if distance <= 1e-9 or shift <= 0:
594         return cert
595     target = tuple(c + (p-c)*shift/distance for p, c in zip(position, cert.center))
596     return ClearCertificate(cert.channel, target, cert.radius_m+shift, cert.limit_m, True)
597
598 # ----- 信息增益选点
599 def _probe_context(self, channel: int):
600     """构造某频道下一次定位测量的候选集与信息增益评分函数。
601
602     返回 ``(candidates, info_score, mec_center, current_position)``;
603     候选为空返回 ``None``。``_select_point`` 与路线感知精修共用此逻辑，
604     保证两者只差“移动代价如何折算”。
605     """
606     bayes, search = self.config.bayes, self.config.search
607     pf = self.particles[channel]
608     if not pf:
609         return None
610     region = self.belief.candidate_set(channel)
611     center, radius = pf.mec()
612     measured = self._measured_points.get(channel, set())
613     position = self.client.state.position
614     candidates: list[Vec2] = []
615     safe = safe_disk_radius(region, self.config) if region is not None else -1.0
616     if region is not None and safe >= 0:
617         # 保证接收盘（凸证书口径）：任意候选必产生 direction/near
618         candidates = candidate_points(tuple(region.mec()[0]), safe,
619                                       search.candidate_rings,
620                                       search.candidate_angles)
621         # 以实际来向生成对称交会候选，避免固定全局角度网格的旋转偏差。
622         geometric_center = region.mec()[0]
623         base_angle = math.atan2(position[1] - geometric_center[1],
624                                position[0] - geometric_center[0])
625         for fraction in search.candidate_rings:
626             if fraction <= 0:
627                 continue
628             for offset in (-math.pi/3, -math.pi/6, 0, math.pi/6, math.pi/3):
629                 angle = base_angle + offset
630                 candidates.append((geometric_center[0] + safe * fraction * math.cos(angle),
631                                   geometric_center[1] + safe * fraction * math.sin(angle)))
632         # 已停靠位置若本身在安全接收盘内，是零移动的合法测点；显式加入
633         # 可避免为了固定角度环而白白横移。
634         if math.dist(position, geometric_center) <= safe + 1e-9:
635             candidates.append(tuple(position))
636     else:
637         # 宽候选集：PF MEC 探索环（负观测挖洞后幸存集的包围圆）
638         for fraction in bayes.wide_rings:
639             ring = max(radius * fraction, 2.0 * pf.margin_m)
640             candidates.extend(candidate_points(center, ring, (1.0,),
641                                               search.candidate_angles))
642     candidates.append(tuple(center))

```

```

643         candidates.append(tuple(position))
644     candidates = [tuple(c) for c in candidates
645                    if _point_key(c) not in measured]
646     if not candidates:
647         return None
648     # 采样一次转成数组 (numpy 可用时), 使候选评分走向量路径;
649     # numpy 缺失时 as_sample_array 原样返回列表, 语义不变。
650     subsample = as_sample_array(pf.subsample(bayes.score_subsample))
651     stride = max(len(subsample) // bayes.mc_representatives, 1)
652     representatives = subsample[:, :stride]
653
654     def info_score(q: Vec2) -> float:
655         if bayes.score_mode == "entropy":
656             return pf.expected_survival(q, subsample, representatives)
657         return pf.expected_posterior_radius(q, subsample, representatives)
658
659     return candidates, info_score, center, position
660
661 def _select_point(self, channel: int) -> Vec2 | None:
662     """定位测量点: 期望 log 幸存占比 +  $\lambda$  · 移动时间 最小的候选。"""
663     context = self._probe_context(channel)
664     if context is None:
665         return None
666     candidates, info_score, center, position = context
667     bayes = self.config.bayes
668
669     def score(q: Vec2) -> float:
670         # 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。
671         move = bayes.lambda_move_s * (move_time(position, q, self.config.cost)
672                                       + move_time(q, center, self.config.cost))
673         return info_score(q) + move
674
675     return min(candidates, key=lambda q: (score(q), _point_key(q)))
676
677 def _refine_locate_tasks(self, tasks: list[Task]) -> list[Task]:
678     """路线感知精修定位任务点 (P2, 2026-09-12) 。
679
680     旧评分只比较 ``current->q + q->MEC`` , 是局部代理; 滚动 2-opt 的真实
681     代价还包含 ``q->next_task`` 。这里对每个 locate 任务在候选点上计算
682     “替换该任务后整条当前路线代价的增量”, 再与信息增益、MEC 接近项合并
683     选点。只改变选点启发式, 不改变覆盖/清除证明。
684     """
685     bayes = self.config.bayes
686     if not bayes.route_aware_probe:
687         return tasks
688     locate_indices = [index for index, task in enumerate(tasks)
689                      if task.kind == "locate"]
690     if not locate_indices:
691         return tasks
692     base_order = self._order_tasks(tasks)
693     base_cost = self._path_cost(base_order)
694     refined = list(tasks)
695     for task_index in locate_indices:
696         task = tasks[task_index]
697         context = self._probe_context(task.channel)
698         if context is None:
699             continue
700         candidates, info_score, center, position = context
701         try:
702             order_index = next(index for index, item in enumerate(base_order)
703                               if item is task)
704         except StopIteration: # pragma: no cover - 同对象应始终存在
705             order_index = next(index for index, item in enumerate(base_order)
706                               if item.kind == "locate"
707                               and item.channel == task.channel)
708         best_score: float | None = None
709         best_point: Vec2 | None = None
710         for point in candidates:
711             trial_order = list(base_order)
712             trial_order[order_index] = Task(

```

```

713         "locate", tuple(point), task.service_s, channel=task.channel)
714     route_delta = self._path_cost(trial_order) - base_cost
715     move_proxy = bayes.lambda_move_s * (
716         move_time(position, point, self.config.cost)
717         + move_time(point, center, self.config.cost))
718     clear_near = move_time(point, center, self.config.cost)
719     value = (info_score(point)
720             + move_proxy
721             + bayes.route_aware_insert_weight * route_delta
722             + bayes.route_aware_clear_weight * clear_near)
723     if best_score is None or value < best_score:
724         best_score, best_point = value, tuple(point)
725     if best_point is not None:
726         refined[task_index] = Task(
727             "locate", best_point, task.service_s, channel=task.channel)
728     return refined
729
730 # ----- 安全网与复扫
731 def _localize_and_clear(self, channel: int) -> bool:
732     """阻塞式定位子循环 (resolve 阶段安全网): PF 选点, 失败退凸选点。"""
733     search = self.config.search
734     self._may_clear.add(channel)
735     for _ in range(search.max_localize_steps):
736         if self._virtual_cap_reached():
737             return False
738         cert = self._best_certificate(channel)
739         if cert is not None and cert.certified and channel in self._may_clear:
740             if self._attempt_clear(channel, cert.center, cert):
741                 self._clear_failed.discard(channel)
742                 return True
743             self._clear_failed.add(channel)
744             self._may_clear.discard(channel)
745             continue
746         point = self._select_point(channel)
747         if point is None:
748             region = self.belief.candidate_set(channel)
749             if region is None:
750                 return False
751             point = select_measurement_point(
752                 region, self.client.state.position, self.config).position
753         outcome = self.client.measure(point, channel)
754         self._record_measure(channel, point, outcome)
755         self._may_clear.add(channel)
756         if outcome.result == NO_SIGNAL:
757             region = self.belief.candidate_set(channel)
758             if region is not None and safe_disk_radius(region, self.config) >= 0 \
759                 and math.dist(point, region.mec()[0]) \
760                 <= safe_disk_radius(region, self.config):
761                 self._anomaly("no_signal_in_safe_disk", channel, position=point)
762         cert = self._best_certificate(channel)
763         if cert is not None and cert.certified:
764             if self._attempt_clear(channel, cert.center, cert):
765                 self._clear_failed.discard(channel)
766                 return True
767             self._clear_failed.add(channel)
768             self._may_clear.discard(channel)
769     return self.belief.channels[channel].status is ChannelStatus.CLEARED
770
771 def _persistent_round(self) -> None:
772     """一轮复扫: 与基线一致的全站复测, 但读数同步进 PF (防假设偏差)。"""
773     measured: set[tuple[int, int]] = set()
774     while self.stop_reason is None:
775         index = self._next_resweep_point(measured)
776         if index is None:
777             return
778         point = self.ledger.points[index]
779         for channel in self._resweep_channels_at(index, measured):
780             if self._virtual_cap_reached():
781                 return
782             measured.add((channel, index))

```

```

783         was_absent = (self.belief.channels[channel].status
784                        is ChannelStatus.ABSENT)
785         outcome = self.client.measure(point, channel)
786         self._record_measure(channel, point, outcome)
787         if was_absent and outcome.result in (DIRECTION, NEAR):
788             # 复测推翻了缺席证明：守恒性留痕，按新证据定位清除。
789             self._anomaly("absent_detected_on_resweep", channel,
790                           position=point, result=outcome.result)
791         if outcome.result == DIRECTION:
792             self._localize_and_clear(channel)
793         elif outcome.result == NEAR:
794             cert = self._best_certificate(channel)
795             if cert is not None and cert.certified:
796                 if not self._attempt_clear(channel, cert.center, cert):
797                     self._clear_failed.add(channel)
798             else:
799                 self._anomaly("near_without_certificate", channel,
800                               position=point)
801         if self.stop_reason is not None:
802             return
803
804         # ----- 收尾
805         def _summary(self, info) -> dict:
806             report = super()._summary(info)
807             report["strategy"] = "bayes"
808             report["opportunistic_measures"] = self.opportunistic_measures
809             report["skipped_measures"] = self.skipped_measures
810             report["no_shrink_measures"] = self.no_shrink_measures
811             report["pf_absent_channels"] = sorted(self.pf_absent_channels)
812             report["pf_alive_final"] = {
813                 str(channel): len(self.particles[channel])
814                 for channel in sorted(self.particles)
815                 if self.belief.channels[channel].status is ChannelStatus.DETECTED}
816             return report

```

src/mcm26b/planner/seven.py

```

1  """P4 七点覆盖策略族：解析证明的六边形发现站 + 同站批量扫频。
2
3  从 scripts/benchmark_q3_search.py 提升为库代码（A2 基线，官方演练
4  12/12 全清、虚拟 5232s）。覆盖几何：原点 + 半径 RING_RADIUS_M 环上
5  6 等分点；域半径R、环半径r时，最近站距离的解析
6  上界  $\max(r/\sqrt{3}, \sqrt{R^2+r^2-\sqrt{3}Rr}) < 1000\text{m}$  [题面
7   $r_{\text{eff}}$  下界]，故任意合法源必被至少一站检出——缺席证明只需该频道
8  7 站全 no_signal (CoverageLedger.coverage_complete)。
9
10 调度变体：
11 - SevenImmediate: 站内即时进入定位子循环（A1）；
12 - SevenBatch: 站内只扫频，扫完按"距候选集 MEC 圆心最近"服务队列（A2）；
13 - SevenSweep: 全域扫完再统一服务（A3）。
14 """
15
16 from __future__ import annotations
17
18 import math
19
20 from ..belief import ChannelStatus
21 from ..config import ProblemConfig
22 from ..coverage import CoverageLedger
23 from ..guarantee import seven_plan
24 from ..protocol.rules import NEAR
25 from . import OmnidirectionalLoop, clear_certificate
26
27 RING_RADIUS_M = 1200.0 # [工程] 解析证明见 docs/algo-3.md §5 与 geometry_check
28
29
30 def seven_points(ring_radius_m: float = RING_RADIUS_M) -> list[tuple[float, float]]:
31     """原点 + 环上 6 等分点的七站覆盖集。"""
32     return [(0.0, 0.0)] + [(ring_radius_m * math.cos(k * math.pi / 3),

```

```

33         ring_radius_m * math.sin(k * math.pi / 3)) for k in range(6)]
34
35
36 def validate_seven_coverage(ring_radius_m: float, config: ProblemConfig) -> float:
37     """六个60°扇区内的最近站最远距离上界；仅证明全向源覆盖。
38
39     原点/环站Voronoi交点给出 $r/\sqrt{3}$ ，目标圆边界的扇区接缝给出另一项。
40     环半径满足 $0 < r \leq \sqrt{3}R$ 时这两项覆盖全部极值。
41     """
42     radius = config.field.omega_radius_m
43     if not 0 < ring_radius_m <= math.sqrt(3)*radius:
44         raise ValueError("七站环半径不满足全域覆盖证明的前提")
45     bound = max(ring_radius_m / math.sqrt(3), math.sqrt(
46         radius**2 + ring_radius_m**2 - math.sqrt(3)*radius*ring_radius_m))
47     if bound > config.field.r_eff_min_m - 1e-6:
48         raise ValueError("七站未覆盖整个目标域，不能据此证明频道缺席")
49     return bound
50
51
52 class SevenImmediate(OmnidirectionalLoop):
53     """七点覆盖 + 站内即时定位（基类覆盖阶段逻辑不变，仅换站集）。"""
54
55     def __init__(self, client, config: ProblemConfig = ProblemConfig(),
56                 ring_radius_m: float = RING_RADIUS_M):
57         super().__init__(client, config)
58         validate_seven_coverage(ring_radius_m, config)
59         self.coverage_plan = seven_plan(ring_radius_m)
60         self.ledger = CoverageLedger(config=config,
61                                     points=list(self.coverage_plan.points))
62
63
64 class SevenBatch(SevenImmediate):
65     """站内只扫频，扫完按最近候选集圆心顺序服务检测队列（A2 主策略）。"""
66
67     defer_all = False
68
69     def _serve_queue(self):
70         # Each detected channel gets at most one localization call in this pass.
71         queue = self.belief.channels_by_status(ChannelStatus.DETECTED)
72         while queue:
73             c = min(queue, key=lambda c: (math.dist(self.client.state.position,
74                                                     self.belief.candidate_set(c).mec()[0]), c))
75             queue.remove(c)
76             self._localize_and_clear(c)
77             if self._count_limit_reached():
78                 break
79
80     def _coverage_phase(self):
81         while not self._count_limit_reached():
82             index = self._next_point_index()
83             if index is None:
84                 return
85             point = self.ledger.points[index]
86             pending = self._pending_channels_at(index)
87             # Keeping the current measuring channel first saves a switch when possible.
88             pending.sort(key=lambda c: (c != self.client.state.channel, c))
89             for c in pending:
90                 outcome = self.client.measure(point, c)
91                 self.ledger.mark_measured(c, index)
92                 self.belief.update_measure(c, point, outcome.result, outcome.svd_deg)
93                 if outcome.result == NEAR:
94                     cert = clear_certificate(self.belief.candidate_set(c), c, self.config)
95                     # Same-station near clear; do not displace the station scan.
96                     if cert.certified:
97                         self._attempt_clear(c, point, cert)
98                 if self._count_limit_reached():
99                     return
100         if not self.defer_all:
101             self._serve_queue()
102

```

```

103     def _resolve_detected(self):
104         self._serve_queue()
105         unresolved = [c for c, b in self.belief.channels.items()
106                       if b.status not in (ChannelStatus.CLEARED, ChannelStatus.ABSENT)]
107         return "all_resolved" if not unresolved else "unresolved_channels"
108
109
110 class SevenSweep(SevenBatch):
111     """Finish discovery at all necessary stations before servicing the queue."""
112
113     defer_all = True

```

src/mcm26b/protocol/__init__.py

```

1  """PO/P1 协议层：字段表、规则表、计时模型、状态机与客户端。"""
2
3  from .client import (ActionRejected, ClearOutcome, DeadlineExceeded,
4                      ExitInfo, MeasureOutcome, ProtocolError,
5                      ProtocolViolation, SimulatorClient)
6  from .log import SessionLog
7  from .state import RobotState, StateError
8  from .timing import CostBreakdown, clear_cost, measure_cost, move_time
9  from .transport import RequestsTransport, Transport, TransportError, TransportResponse
10
11 __all__ = ["ActionRejected", "ClearOutcome", "CostBreakdown", "DeadlineExceeded",
12           "ExitInfo", "MeasureOutcome", "ProtocolError", "ProtocolViolation",
13           "RequestsTransport", "RobotState", "SessionLog", "SimulatorClient",
14           "StateError", "Transport", "TransportError", "TransportResponse",
15           "clear_cost", "measure_cost", "move_time"]

```

src/mcm26b/protocol/client.py

```

1  """P1 协议基线：串行、幂等、带现实预算的模拟器客户端。
2
3  协议硬规则 [题面] 与实现对应：
4  - 每个新动作使用新的 request_id；仅在网络失败后重试"完全相同的动作"时，
5    复用原 request_id 与原请求体 —— 本类通过一次性序列化出 bytes 并原样重发保证；
6  - 必须同时检查 HTTP 状态与 accepted；accepted=false 不更新状态，
7    其 virtual_time_s=0 不当作当前虚拟时刻；
8  - 逐次等待响应、不得并发；本类完全同步，一个动作未决不发起下一个；
9  - 现实截止用 /enter 响应的 remaining_real_duration_s 建立本地单调时钟，
10   预留退出余量；/exit 不做预算拦截。
11  """
12
13  from __future__ import annotations
14
15  import json
16  import math
17  import uuid
18  from dataclasses import dataclass
19  from time import monotonic, sleep
20
21  from ..config import ProblemConfig, Vec2
22  from .log import SessionLog
23  from .rules import (CLEAR_RESULTS, DIRECTION, EXIT_USER, MEASURE_RESULTS,
24                     POSITION_FIELDS, STATUS_OK, STATUS_RETRYABLE)
25  from .state import RobotState
26  from .timing import CostBreakdown
27  from .transport import RequestsTransport, Transport, TransportError
28
29
30  class ProtocolError(RuntimeError):
31      """协议层失败，含 HTTP 状态与响应体；默认不重试（确定性错误）。"""
32
33      def __init__(self, message: str, status: int | None = None, body=None):
34          super().__init__(message)
35          self.status = status
36          self.body = body

```

```

37
38
39 class ActionRejected(ProtocolError):
40     """HTTP 200 但 accepted=false: 动作未生效, 状态不得更新。"""
41
42     def __init__(self, message: str, body=None):
43         super().__init__(message, status=STATUS_OK, body=body)
44
45
46 class ProtocolViolation(ProtocolError):
47     """响应与协议约定不符 (字段缺失、值非法、虚拟时刻回退等)。"""
48
49
50 class DeadlineExceeded(RuntimeError):
51     """现实预算不足: 预留退出余量后剩余时间不够执行新动作。"""
52
53
54 @dataclass(frozen=True)
55 class MeasureOutcome:
56     result: str
57     svd_deg: float | None
58     virtual_time_s: float
59     predicted: CostBreakdown
60     raw: dict
61
62
63 @dataclass(frozen=True)
64 class ClearOutcome:
65     result: str
66     virtual_time_s: float
67     predicted: CostBreakdown
68     raw: dict
69
70
71 @dataclass(frozen=True)
72 class ExitInfo:
73     reason: str
74     virtual_time_s: float
75     raw: dict
76
77
78 def _validate_identifier(name: str, value: str, max_bytes: int) -> None:
79     if not isinstance(value, str) or not value:
80         raise ValueError(f"{name} 必须是非空字符串")
81     encoded = value.encode("utf-8")
82     if not 1 <= len(encoded) <= max_bytes:
83         raise ValueError(f"{name} 的 UTF-8 长度须在 1..{max_bytes} 字节")
84     if any(code < 0x20 or code == 0x7F for code in encoded):
85         raise ValueError(f"{name} 不能包含控制字符")
86
87
88 def _validate_position(position: dict, limit: float) -> None:
89     if set(position) != set(POSITION_FIELDS):
90         raise ValueError("position 字段必须是且只能是 x、y")
91     for key in ("x", "y"):
92         value = position[key]
93         if isinstance(value, bool) or not isinstance(value, (int, float)):
94             raise ValueError(f"position.{key} 必须是数值")
95         if not math.isfinite(value):
96             raise ValueError(f"position.{key} 必须是有限数值")
97         if abs(value) > limit:
98             raise ValueError(f"position.{key} 绝对值不得超过 {limit} 米")
99
100
101 def _validate_channel(channel) -> int:
102     # 写成 JSON number 1.0 且恰为整数也接受 [题面]; 1.5 拒绝。
103     if isinstance(channel, bool) or not isinstance(channel, (int, float)):
104         raise ValueError("channel 必须是数值")
105     if isinstance(channel, float) and not channel.is_integer():
106         raise ValueError("channel 必须是整数")

```

```

107     channel = int(channel)
108     if not 1 <= channel <= 20:
109         raise ValueError("channel 必须在 1..20 范围内")
110     return channel
111
112
113 class SimulatorClient:
114     """四条指令的同步客户端。状态只在 accepted=true 后推进。"""
115
116     def __init__(self, robot_id: str, *,
117                  config: ProblemConfig = ProblemConfig(),
118                  transport: Transport | None = None,
119                  log: SessionLog | None = None,
120                  clock=monotonic):
121         _validate_identifier("robot_id", robot_id, config.session.robot_id_max_bytes)
122         self.robot_id = robot_id
123         self.config = config
124         self.session = config.session
125         self.transport = transport if transport is not None else RequestsTransport()
126         self.log = log if log is not None else SessionLog()
127         self.clock = clock
128         self.state = RobotState()
129         self._prefix = uuid.uuid4().hex[:12]
130         self._seq = 0
131         self._deadline: float | None = None          # 本地单调时钟截止
132         self._server_deadline_ms: float | None = None # 服务器时间戳交叉核对
133         self._enter_monotonic: float | None = None   # 实机计时起点 (/enter 成功)
134
135     # ----- 公共动作 API
136     def enter(self) -> dict:
137         started = self.clock()
138         payload = self._payload("/enter")
139         response = self._call("/enter", payload)
140         self.state.apply_enter(response)
141         remaining = response["remaining_real_duration_s"]
142         if not isinstance(remaining, (int, float)) or remaining < 0:
143             raise ProtocolViolation("remaining_real_duration_s 缺失或非法", body=response)
144         self._deadline = self.clock() + remaining
145         self._server_deadline_ms = (response["real_timestamp_ms"]
146                                     + remaining * 1000.0)
147         self._enter_monotonic = self.clock()
148         self._log_step("/enter", payload, response, started,
149                       predicted=CostBreakdown(0.0, 0.0, 0.0))
150         return response
151
152     def measure(self, position: Vec2, channel: int) -> MeasureOutcome:
153         started = self.clock()
154         payload = self._payload("/measure", position=position, channel=channel)
155         response = self._call("/measure", payload)
156         predicted = self.state.apply_measure(payload, response,
157                                              cost=self.config.cost)
158         result = response.get("measure_result")
159         if result not in MEASURE_RESULTS:
160             raise ProtocolViolation(f"measure_result 非法: {result!r}", body=response)
161         svd = response.get("svd_deg")
162         if result == DIRECTION:
163             if not isinstance(svd, (int, float)) or isinstance(svd, bool):
164                 raise ProtocolViolation("direction 响应缺少数值 svd_deg", body=response)
165         self._log_step("/measure", payload, response, started, result=result,
166                       predicted=predicted, channel=payload["channel"])
167         return MeasureOutcome(result, float(svd) if svd is not None else None,
168                               response["virtual_time_s"], predicted, response)
169
170     def clear(self, position: Vec2, channel: int) -> ClearOutcome:
171         started = self.clock()
172         payload = self._payload("/clear", position=position, channel=channel)
173         response = self._call("/clear", payload)
174         predicted = self.state.apply_clear(payload, response,
175                                              cost=self.config.cost)
176         result = response.get("clear_result")

```

```

177         if result not in CLEAR_RESULTS:
178             raise ProtocolViolation(f"clear_result 非法: {result!r}", body=response)
179         self._log_step("/clear", payload, response, started, result=result,
180                       predicted=predicted, channel=payload["channel"])
181         return ClearOutcome(result, response["virtual_time_s"], predicted, response)
182
183     def exit(self) -> ExitInfo:
184         started = self.clock()
185         payload = self._payload("/exit")
186         response = self._call("/exit", payload)
187         self.state.apply_exit(response)
188         reason = response.get("exit_reason")
189         if reason != EXIT_USER:
190             raise ProtocolViolation(f"exit_reason 非法: {reason!r}", body=response)
191         self._log_step("/exit", payload, response, started,
192                       predicted=CostBreakdown(0.0, 0.0, 0.0))
193         return ExitInfo(reason, response["virtual_time_s"], response)
194
195     # ----- 预算控制
196     @property
197     def real_remaining_s(self) -> float | None:
198         """按本地单调时钟估计的剩余现实秒数（未 enter 时为 None）。"""
199         return None if self._deadline is None else self._deadline - self.clock()
200
201     def _log_step(self, path: str, payload: dict, response: dict,
202                  started: float, *, result: str | None = None,
203                  predicted: CostBreakdown, channel: int | None = None) -> None:
204         """每步计时留档 [用户决策 2026-09-11]: 实机墙钟耗时 + 比赛虚拟耗时分项
205         （移动=直线距离/5、切频道 1s、检测 5s、清除成功 5s/未发现 3s）。
206
207         step 事件不含 response 字段，不会混入 action_events / 虚拟时刻复算。
208         """
209         now = self.clock()
210         entry: dict = {"event": "step", "path": path,
211                       "request_id": payload["request_id"],
212                       "real_elapsed_s": round(now - started, 6),
213                       "virtual_time_s": response["virtual_time_s"],
214                       "virtual_cost": {"move_s": round(predicted.move_s, 6),
215                                       "switch_s": round(predicted.switch_s, 6),
216                                       "action_s": round(predicted.action_s, 6),
217                                       "total_s": round(predicted.total_s, 6)}}
218         if self._enter_monotonic is not None:
219             entry["real_since_enter_s"] = round(now - self._enter_monotonic, 6)
220         if "position" in payload:
221             entry["position"] = {"x": payload["position"]["x"],
222                                 "y": payload["position"]["y"]}
223         if channel is not None:
224             entry["channel"] = channel
225         if result is not None:
226             entry["result"] = result
227         self.log.append(entry)
228
229     def _check_budget(self, path: str) -> None:
230         if path == "/exit" or self._deadline is None:
231             return
232         margin = self.session.reserve_exit_s
233         if self.real_remaining_s <= margin:
234             raise DeadlineExceeded(
235                 f"剩余现实时间 {self.real_remaining_s:.1f}s 不足以预留 {margin:.1f}s 退出余量")
236
237     # ----- 内部机制
238     def _payload(self, _path: str, *, position: Vec2 | None = None,
239                 channel: int | None = None) -> dict:
240         payload = {"arena_id": self.session.arena_id,
241                   "robot_id": self.robot_id,
242                   "request_id": self._next_request_id()}
243         if position is not None:
244             x, y = float(position[0]), float(position[1])
245             _validate_position({"x": x, "y": y}, self.config.field.coord_abs_limit_m)
246             payload["position"] = {"x": x, "y": y}

```

```

247         if channel is not None:
248             payload["channel"] = _validate_channel(channel)
249         return payload
250
251     def _next_request_id(self) -> str:
252         self._seq += 1
253         request_id = f"{self._prefix}-{self._seq:04d}"
254         _validate_identifier("request_id", request_id,
255                             self.session.request_id_max_bytes)
256         return request_id
257
258     def _call(self, path: str, payload: dict) -> dict:
259         """串行执行一个动作；网络失败原样重发，HTTP 确定性错误立即抛出。"""
260         self._check_budget(path)
261         body = json.dumps(payload, ensure_ascii=False, allow_nan=False).encode("utf-8")
262         if len(body) > self.session.request_body_max_bytes:
263             raise ValueError("请求体超过 65536 字节")
264         headers = {"Content-Type": "application/json"}
265         url = self.session.base_url + path
266         attempts, last_error = 0, None
267         while attempts < self.session.max_attempts:
268             attempts += 1
269             started = self.clock()
270             self.log.append({"event": "attempt", "path": path,
271                             "request_id": payload["request_id"],
272                             "attempt": attempts, "request": payload})
273             try:
274                 result = self.transport.post(
275                     url, body, headers,
276                     timeout=(self.session.connect_timeout_s,
277                             self.session.read_timeout_s))
278             except TransportError as exc:
279                 last_error = exc
280                 self.log.append({"event": "transport_error", "path": path,
281                                 "request_id": payload["request_id"],
282                                 "attempt": attempts, "error": str(exc),
283                                 "elapsed_s": self.clock() - started})
284                 if attempts < self.session.max_attempts:
285                     sleep(self.session.retry_backoff_s * (2 ** (attempts - 1)))
286                 continue
287             if result.status in STATUS_RETRYABLE:
288                 self.log.append({"event": "retryable_status", "path": path,
289                                 "request_id": payload["request_id"],
290                                 "attempt": attempts, "status": result.status})
291                 if attempts < self.session.max_attempts:
292                     sleep(self.session.retry_backoff_s * (2 ** (attempts - 1)))
293                 continue
294             if result.status != STATUS_OK:
295                 raise ProtocolError(
296                     f"{path} 返回 HTTP {result.status}", status=result.status,
297                     body=self._safe_json(result.body))
298             response = self._safe_json(result.body)
299             if response is None or response.get("accepted") is not True:
300                 raise ActionRejected(
301                     f"{path} 未被接受: {response}", body=response)
302             self._verify_clock(response)
303             self.log.append({"event": "accepted", "path": path,
304                             "request_id": payload["request_id"],
305                             "request": payload, "response": response,
306                             "attempts": attempts,
307                             "elapsed_s": self.clock() - started})
308             return response
309             raise TransportError(
310                 f"{path} 重试 {attempts} 次仍失败: {last_error}") from last_error
311
312     def _verify_clock(self, response: dict) -> None:
313         """accepted=true 的虚拟时刻必须单调不减；现实时间戳用于漂移核对。"""
314         virtual = response.get("virtual_time_s")
315         if not isinstance(virtual, (int, float)) or isinstance(virtual, bool):
316             raise ProtocolViolation("accepted 响应缺少 virtual_time_s", body=response)

```

```

317         if virtual < self.state.virtual_time_s - 1e-9:
318             raise ProtocolViolation(
319                 f"虚拟时刻回退: {self.state.virtual_time_s} -> {virtual}",
320                 body=response)
321         if self._server_deadline_ms is not None:
322             drift = (self._server_deadline_ms - response["real_timestamp_ms"]) / 1000.0
323             if drift < self.session.reserve_exit_s / 2:
324                 # 服务器视角余量告急 (本地时钟漂移或请求排队); 仅记录, 交由上层决策。
325                 self.log.append({"event": "server_time_warning",
326                                 "server_remaining_s": drift})
327
328     @staticmethod
329     def _safe_json(body: bytes):
330         try:
331             return json.loads(body.decode("utf-8"))
332         except (UnicodeDecodeError, json.JSONDecodeError):
333             return None

```

src/mcm26b/protocol/log.py

```

1  """会话日志: 机器狗程序必须自行记录指令序列与响应 [题面] 附件2 12节。
2
3  事件逐条追加写入 JSONL, 进程崩溃时已完成的动作不丢;
4  内存中的 events 同时供回放测试与 evaluation 复算使用。
5  """
6
7  from __future__ import annotations
8
9  import json
10 from dataclasses import dataclass, field
11 from pathlib import Path
12 from time import time
13
14
15 @dataclass
16 class SessionLog:
17     path: Path | None = None
18     events: list = field(default_factory=list)
19
20     def __post_init__(self):
21         if self.path is not None:
22             self.path.parent.mkdir(parents=True, exist_ok=True)
23
24     def append(self, event: dict) -> None:
25         event = {**event, "logged_at_unix_s": time()}
26         self.events.append(event)
27         if self.path is not None:
28             with self.path.open("a", encoding="utf-8") as handle:
29                 handle.write(json.dumps(event, ensure_ascii=False, allow_nan=False) + "\n")
30
31     def action_events(self) -> list[dict]:
32         """仅最终被接受的动作事件 (每个 request_id 一条, 含重试次数)。"""
33         return [event for event in self.events if event.get("response") is not None
34                 and event["response"].get("accepted") is True]
35
36     def step_events(self) -> list[dict]:
37         """每步计时事件 (客户端 _log_step 写入, 按发生顺序)。"""
38         return [event for event in self.events if event.get("event") == "step"]

```

src/mcm26b/protocol/rules.py

```

1  """PO 规则固定: 四条指令的字段表、结果码与错误语义, 全部落成可执行数据。
2
3  依据 issue/附件/附件2.docx 《模拟器通信接口说明及编程指南》第 5-9 节。
4  客户端用它构造与预校验请求, 自建模拟器用它校验并拒绝未知字段;
5  表格本身即交付物"字段表/规则表", 人类可读版本见 docs/协议规则与状态机.md。
6  """
7

```

```

8  from __future__ import annotations
9
10 from dataclasses import dataclass
11
12 # ---- 结果码 [题面] -----
13
14 NO_SIGNAL = "no_signal"      # 收不到信号: 无未清除源 / 超接收半径 / 定向覆盖角外
15 NEAR = "near"                # 距离 <=5m 且在覆盖角内; 不返回 svd_deg
16 DIRECTION = "direction"      # 有信号; 附 svd_deg (含 ±1° 误差, 保留两位小数)
17 MEASURE_RESULTS = (NO_SIGNAL, NEAR, DIRECTION)
18
19 CLEAR_SUCCESS = "success"     # 20m 内发现并清除; 耗时 5s
20 CLEAR_NO_TARGET = "no_target_in_range" # 未发现 (无源/已清除/超20m); 耗时 3s
21 CLEAR_RESULTS = (CLEAR_SUCCESS, CLEAR_NO_TARGET)
22
23 EXIT_USER = "user_exit"      # /exit 成功响应中的固定退出原因
24
25 # accepted=false 的响应只含这 3 个字段; virtual_time_s 恒为 0, 不是当前时刻 [题面]
26 BASE_RESPONSE_FIELDS = ("accepted", "real_timestamp_ms", "virtual_time_s")
27
28
29 # ---- 请求字段表 -----
30
31
32 @dataclass(frozen=True)
33 class FieldSpec:
34     """单个请求字段的约束。[题面] 附件2 5.1 节与各指令小节。"""
35
36     kind: type                # str / dict; 数值字段由 sim 按整型/浮点细则单独校验
37     required: bool = True
38     max_bytes: int | None = None # 字符串字段的 UTF-8 长度上限
39
40
41 COMMON_FIELDS: dict[str, FieldSpec] = {
42     "arena_id": FieldSpec(str), # 必须为 "default"
43     "robot_id": FieldSpec(str, max_bytes=64), # 逐字节等于登录参赛队号
44     "request_id": FieldSpec(str, max_bytes=128), # 会话内幕等键, 新动作新 ID
45 }
46
47 POSITION_FIELDS: dict[str, FieldSpec] = {
48     "x": FieldSpec(float), # 有限 JSON number, |x| <= 2_000_000
49     "y": FieldSpec(float),
50 }
51
52 # 路径 -> 除公共字段外的动作字段: /enter 与 /exit 无附加字段
53 ACTION_FIELDS: dict[str, dict[str, FieldSpec]] = {
54     "/enter": {},
55     "/measure": {"position": FieldSpec(dict), "channel": FieldSpec(float)},
56     "/clear": {"position": FieldSpec(dict), "channel": FieldSpec(float)},
57     "/exit": {},
58 }
59 ACTION_PATHS = tuple(ACTION_FIELDS)
60
61 # channel 写成 JSON number 1.0 且恰为整数也接受; 1.5 返回 HTTP 400 [题面]
62
63
64 # ---- 响应字段表 (accepted=true 时的附加字段) -----
65
66 RESPONSE_FIELDS: dict[str, tuple[str, ...]] = {
67     "/enter": ("max_virtual_duration_s", "max_real_duration_s", "remaining_real_duration_s"),
68     "/measure": ("measure_result",), # direction 时另有 svd_deg
69     "/clear": ("clear_result",),
70     "/exit": ("exit_reason",),
71 }
72
73
74 # ---- 错误语义 [题面] 附件2 5.3 节 -----
75
76 STATUS_OK = 200
77 STATUS_BAD_REQUEST = 400 # JSON 语法/重复键/缺字段/类型/频道或坐标范围

```

```

78 STATUS_CONFLICT = 409                # 同 request_id 对应不同动作，或并发不同动作
79 STATUS_RETRYABLE = (429, 500)        # 流量保护/幂等记录上限、服务器内部错误：可原样重试
80
81
82 def http_status_meaning(status: int) -> str:
83     table = {
84         200: "动作已执行，或 JSON 合法但被业务拒绝（看 accepted）",
85         400: "请求结构错误；不占用 request_id，修正后可复用该 ID",
86         404: "路径未知或不精确（不接受尾随斜线与查询参数）",
87         405: "已知路径使用了非 POST 方法",
88         409: "request_id 幂等冲突或并发送不同动作",
89         413: "请求体超过 65536 字节",
90         415: "Content-Type 或 Content-Encoding 不受支持",
91         429: "连接/流量保护触发，或本局幂等记录达到上限",
92         500: "模拟器内部错误",
93     }
94     return table.get(status, "未知状态码")

```

src/mcm26b/protocol/state.py

```

1  """PO/P1 机器狗本地状态机：位置、测向频道、虚拟时刻与进出状态。
2
3  更新规则 [题面]:
4  - 只有 accepted=true 的响应才允许调用 apply_*; accepted=false 一律不更新，
5    其 virtual_time_s=0 不是当前虚拟时刻；
6  - 虚拟时刻以最近一次 accepted=true 的响应为准，本地预测值仅作一致性核对；
7  - /clear 不改变测向机当前频道，也不产生切换耗时；
8  - 只有合法 /measure 会把测向机频道更新为本次检测频道。
9  """
10
11 from __future__ import annotations
12
13 from dataclasses import dataclass, field
14
15 from ..config import CostConfig, Vec2
16 from .rules import CLEAR_SUCCESS
17 from .timing import CostBreakdown, clear_cost, measure_cost
18
19
20 class StateError(RuntimeError):
21     """状态机被以非法顺序驱动（例如未 enter 就 apply_measure）。"""
22
23
24 @dataclass
25 class RobotState:
26     position: Vec2 = (0.0, 0.0)    # /enter 成功后的初始位置 [题面]
27     channel: int = 1               # /enter 成功后的初始频道 [题面]
28     virtual_time_s: float = 0.0
29     entered: bool = False
30     exited: bool = False
31     action_count: int = 0
32     # 预测与实测的偏差记录: strict 模式下出现偏差会直接抛错。
33     mismatches: list = field(default_factory=list)
34
35     # ----- enter/exit
36     def apply_enter(self, response: dict, *, strict: bool = False) -> None:
37         self._require_accepted(response)
38         if strict and (self.entered or self.exited):
39             raise StateError("重复 /enter 或已退出后再 enter")
40         self.entered = True
41         self.virtual_time_s = response["virtual_time_s"]
42
43     def apply_exit(self, response: dict) -> None:
44         self._require_accepted(response)
45         self.exited = True
46         self.virtual_time_s = response["virtual_time_s"]
47
48     # ----- measure
49     def apply_measure(self, request: dict, response: dict, *,

```

```

50         cost: CostConfig = CostConfig(),
51         strict: bool = False) -> CostBreakdown:
52     self._require_accepted(response)
53     if strict and (not self.entered or self.exited):
54         raise StateError("/measure 要求已 enter 且未退出")
55     pos = (request["position"]["x"], request["position"]["y"])
56     channel = request["channel"]
57     predicted = measure_cost(self.position, pos, self.channel, channel, cost)
58     predicted = self._advance(pos, channel, response, predicted, strict)
59     return predicted
60
61 # ----- clear
62 def apply_clear(self, request: dict, response: dict, *,
63                 cost: CostConfig = CostConfig(),
64                 strict: bool = False) -> CostBreakdown:
65     self._require_accepted(response)
66     if strict and (not self.entered or self.exited):
67         raise StateError("/clear 要求已 enter 且未退出")
68     pos = (request["position"]["x"], request["position"]["y"])
69     success = response["clear_result"] == CLEAR_SUCCESS
70     predicted = clear_cost(self.position, pos, success, cost)
71     # 频道保持不变: /clear 不切换测向机频道 [题面]
72     return self._advance(pos, self.channel, response, predicted, strict)
73
74 # ----- 共用
75 def _advance(self, pos: Vec2, channel: int, response: dict,
76              predicted: CostBreakdown, strict: bool) -> CostBreakdown:
77     # 先完成全部校验再落子: strict 模式下失败时不留半更新状态。
78     actual_delta = response["virtual_time_s"] - self.virtual_time_s
79     if abs(actual_delta - predicted.total_s) > 1e-6:
80         message = (f"耗时预测不一致: 预测 {predicted.total_s:.6f}s, "
81                   f"实测 {actual_delta:.6f}s ({predicted})")
82         if strict:
83             raise StateError(message)
84         # 非严格模式: 状态仍以响应为准, 偏差记入日志供复盘。
85         self.mismatches.append(
86             {"action": self.action_count + 1, "predicted_s": predicted.total_s,
87              "actual_s": actual_delta, "message": message})
88     self.position = pos
89     self.channel = channel
90     self.virtual_time_s = response["virtual_time_s"]
91     self.action_count += 1
92     return predicted
93
94 @staticmethod
95 def _require_accepted(response: dict) -> None:
96     if response.get("accepted") is not True:
97         raise StateError("accepted=false 的响应不允许更新状态")

```

src/mcm26b/protocol/timing.py

```

1  """PO 纯计时模型: /measure 与 /clear 的耗时分解, 不涉及网络与状态。
2
3  附件1表2 与附件2第10节的计时示例是唯一权威锚点:
4  逐步得到 105、111、194、199 秒 (tests/test_protocol_timing.py 固化)。
5  客户端用本模块预测动作代价, 模拟器用同一函数推进虚拟时钟,
6  保证"预测-执行-复算"三方一致。
7  """
8
9  from __future__ import annotations
10
11  from dataclasses import dataclass
12  from math import dist
13
14  from ..config import CostConfig, Vec2
15
16
17  @dataclass(frozen=True)
18  class CostBreakdown:

```

```

19     """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。"""
20
21     move_s: float
22     switch_s: float
23     action_s: float
24
25     @property
26     def total_s(self) -> float:
27         return self.total_us / 1_000_000
28
29     @property
30     def total_us(self) -> int:
31         """逐段量化后按整数微秒累计, 与官方虚拟时钟一致。"""
32         return sum(round(value * 1_000_000)
33                     for value in (self.move_s, self.switch_s, self.action_s))
34
35
36 def measure_cost(prev_pos: Vec2, pos: Vec2, prev_channel: int, channel: int,
37                 cost: CostConfig = CostConfig()) -> CostBreakdown:
38     """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。"""
39     return CostBreakdown(
40         move_s=dist(prev_pos, pos) / cost.speed_mps,
41         switch_s=0.0 if channel == prev_channel else cost.switch_s,
42         action_s=cost.measure_s,
43     )
44
45
46 def clear_cost(prev_pos: Vec2, pos: Vec2, success: bool,
47               cost: CostConfig = CostConfig()) -> CostBreakdown:
48     """合法 /clear 的耗时。
49
50     /clear 的 channel 是目标干扰源频道, 不切换测向机频道, 也不产生换频道耗时 [题面]。
51     """
52     return CostBreakdown(
53         move_s=dist(prev_pos, pos) / cost.speed_mps,
54         switch_s=0.0,
55         action_s=cost.clear_success_s if success else cost.clear_fail_s,
56     )
57
58
59 def move_time(a: Vec2, b: Vec2, cost: CostConfig = CostConfig()) -> float:
60     """任意两点间的移动耗时 (规划用)。"""
61     return dist(a, b) / cost.speed_mps

```

src/mcm26b/protocol/transport.py

```

1     """传输抽象: HTTP 细节与协议客户端解耦, 便于故障注入与本地模拟。
2
3     真实运行用 :class:`RequestsTransport`; 自建模拟器经 :class:`LocalSimTransport`
4     (sim 模块) 挂载; 回放用 PlaybackTransport。连接失败与读超时统一抛
5     :class:`TransportError` —— 此类失败后重试"同一动作"必须复用原 request_id
6     与原请求体 [题面], 由 client 层保证。
7     """
8
9     from __future__ import annotations
10
11     from dataclasses import dataclass
12
13     import requests
14
15
16     class TransportError(Exception):
17         """网络层失败: 连接不上、读写超时。动作可能未送达, 也可能结果未知。"""
18
19
20     @dataclass(frozen=True)
21     class TransportResponse:
22         status: int
23         body: bytes

```

```

24
25
26 class Transport:
27     """最小 POST 接口。同步阻塞，一次一个请求（协议要求串行）。"""
28
29     def post(self, url: str, body: bytes, headers: dict[str, str],
30             timeout: tuple[float, float]) -> TransportResponse:
31         raise NotImplementedError
32
33
34 class RequestsTransport(Transport):
35     """基于 requests 的默认实现。timeout=(连接, 读取)。"""
36
37     def post(self, url, body, headers, timeout):
38         try:
39             response = requests.post(url, data=body, headers=headers, timeout=timeout)
40         except (requests.exceptions.Timeout, requests.exceptions.ConnectionError) as exc:
41             raise TransportError(str(exc)) from exc
42         except requests.exceptions.RequestException as exc: # 其它请求级异常同样按网络失败处理
43             raise TransportError(str(exc)) from exc
44         return TransportResponse(response.status_code, response.content)

```

src/mcm26b/routing.py

```

1  """图论/动态规划工具箱：开放路径排序、下界与路由度量（S1/S5）。
2
3  设计原则（docs/第三四问图论与动态规划优化方案.md §1、§3.2、§3.4）：
4
5  - **开放路径**（起点固定为原点、终点自由）是本题的真实图问题：一次作业回合
6    = 依次停靠若干作业点，代价 =  $\Sigma$  移动时间 +  $\Sigma$  动作耗时；
7  - 小规模（ $n \leq \text{EXACT\_LIMIT}$ ）直接用 Held-Karp 精确 DP，规模更大时用
8    最近邻打底 + 2-opt / Or-opt 改进，并始终给出 **MST 下界**用于在线后悔值审计；
9  - 所有函数只做“排序与度量”，不承担任何安全责任，也不改变覆盖/清除语义。
10
11  约定：边权为**欧氏距离**（米）；时间折算由调用方按 ``speed_mps`` 处理。
12  2-opt / Or-opt 用  $O(1)$  增量代价评估，使  $n \approx 10^2$  量级仍可在毫秒内完成。
13  """
14
15  from __future__ import annotations
16
17  import math
18
19  from .config import Vec2
20
21  EXACT_LIMIT = 13          # Held-Karp 状态数  $2^n \cdot n$ :  $n=13 \rightarrow 106\,496$ ，纯 Python 可接受
22  OR_OPT_LIMIT = 60        # 作业点数超过此值时跳过高阶 Or-opt（见 improve_open_path）
23
24
25  def path_length(order: list[Vec2], start: Vec2 = (0.0, 0.0)) -> float:
26     """开放路径长度（起点 start，终点自由）。"""
27     total = 0.0
28     position = start
29     for point in order:
30         total += math.dist(position, point)
31         position = point
32     return total
33
34
35  def nearest_neighbor_order(points: list[Vec2],
36                             start: Vec2 = (0.0, 0.0)) -> list[Vec2]:
37     """最近邻开放路径（ $O(n^2)$ ，启发式上界）。"""
38     remaining = list(points)
39     order: list[Vec2] = []
40     position = start
41     while remaining:
42         nxt = min(remaining, key=lambda q: math.dist(position, q))
43         remaining.remove(nxt)
44         order.append(nxt)
45         position = nxt

```

```

46     return order
47
48
49 def two_opt_open(order: list[Vec2], start: Vec2 = (0.0, 0.0),
50                 max_passes: int = 8) -> list[Vec2]:
51     """开放路径 2-opt (终点自由: 只反转内部片段, 不做首尾相接)。
52
53     增量代价 0(1): 反转 [i..j] 只影响 (prev, i) 与 (j, next) 两条边。
54     """
55     best = list(order)
56     n = len(best)
57     if n < 3:
58         return best
59     for _ in range(max_passes):
60         improved = False
61         for i in range(n - 1):
62             prev = best[i - 1] if i > 0 else start
63             pi = best[i]
64             d_prev_i = math.dist(prev, pi)
65             for j in range(i + 1, n):
66                 pj = best[j]
67                 nxt = best[j + 1] if j + 1 < n else None
68                 d_prev_j = math.dist(prev, pj)
69                 removed = d_prev_i
70                 added = d_prev_j
71                 if nxt is not None:
72                     removed += math.dist(pj, nxt)
73                     added += math.dist(pi, nxt)
74                 if added < removed - 1e-9:
75                     best[i:j + 1] = best[i:j + 1][::-1]
76                     improved = True
77                     pi = best[i]
78                     d_prev_i = math.dist(prev, pi)
79             if not improved:
80                 break
81     return best
82
83
84 def or_opt_open(order: list[Vec2], start: Vec2 = (0.0, 0.0),
85                 max_passes: int = 4, max_segment: int = 3) -> list[Vec2]:
86     """Or-opt: 把长度 ≤ max_segment 的片段搬家 (开放路径, 增量代价 0(1))。"""
87     best = list(order)
88     n = len(best)
89     if n < 4:
90         return best
91     for _ in range(max_passes):
92         improved = False
93         for length in range(1, min(max_segment, n - 1) + 1):
94             for i in range(0, n - length + 1):
95                 segment = best[i:i + length]
96                 rest = best[:i] + best[i + length:]
97                 for j in range(len(rest) + 1):
98                     if j == i:
99                         continue
100                     candidate = rest[:j] + segment + rest[j:]
101                     if _segment_move_gain(best, i, length, j, start) > 1e-9:
102                         best = candidate
103                         improved = True
104                     break
105             if improved:
106                 break
107         if improved:
108             break
109     if not improved:
110         break
111     return best
112
113
114 def _segment_move_gain(order: list[Vec2], i: int, length: int, j: int,
115                       start: Vec2) -> float:

```

```

116     """把 order[i:i+length] 插到去掉该段后的第 j 位所省下的长度（正=更省）。"""
117     rest = order[:i] + order[i + length:]
118     segment = order[i:i + length]
119
120     def length_of(seq: list[Vec2]) -> float:
121         return path_length(seq, start)
122
123     return length_of(order) - length_of(rest[:j] + segment + rest[j:])
124
125
126 def improve_open_path(points: list[Vec2], start: Vec2 = (0.0, 0.0),
127                        max_passes: int = 8) -> list[Vec2]:
128     """最近邻 + 2-opt + Or-opt 的开放路径改进（大规模兜底）。
129
130     Or-opt 的搬移评估是  $O(n^3)$ ，作业点数超过 :data:`OR_OPT_LIMIT` 时只做
131      $O(n^2)$  的 2-opt（度量口径下 Or-opt 的相对收益远小于其墙钟代价）。
132     """
133     order = nearest_neighbor_order(points, start)
134     order = two_opt_open(order, start, max_passes)
135     if len(order) > OR_OPT_LIMIT:
136         return order
137     return or_opt_open(order, start, max_passes)
138
139
140 def held_karp_open(points: list[Vec2], start: Vec2 = (0.0, 0.0)) -> list[Vec2]:
141     """Held-Karp 精确开放路径（起点固定、终点自由）。
142
143     状态 ``dp[mask][j]`` = 访问集合 mask、当前停在 j 的最小长度；
144     规模  $2^n \cdot n$ ，仅用于  $n \leq$  :data:`EXACT_LIMIT`。返回最优访问顺序。
145     """
146     n = len(points)
147     if n == 0:
148         return []
149     if n == 1:
150         return list(points)
151     dist = [[math.dist(points[i], points[j]) for j in range(n)] for i in range(n)]
152     from_start = [math.dist(start, points[j]) for j in range(n)]
153     full = (1 << n) - 1
154     inf = float("inf")
155     dp: list[list[float]] = [[inf] * n for _ in range(1 << n)]
156     parent: list[list[int]] = [[-1] * n for _ in range(1 << n)]
157     for j in range(n):
158         dp[1 << j][j] = from_start[j]
159     for mask in range(1 << n):
160         row = dp[mask]
161         for j in range(n):
162             current = row[j]
163             if current == inf or not (mask >> j) & 1:
164                 continue
165             rest = full ^ mask
166             while rest:
167                 bit = rest & -rest
168                 k = bit.bit_length() - 1
169                 rest ^= bit
170                 candidate = current + dist[j][k]
171                 nxt = mask | bit
172                 if candidate < dp[nxt][k]:
173                     dp[nxt][k] = candidate
174                     parent[nxt][k] = j
175     best_end = min(range(n), key=lambda j: dp[full][j])
176     order: list[Vec2] = []
177     mask, j = full, best_end
178     while j != -1:
179         order.append(points[j])
180         previous = parent[mask][j]
181         mask ^= 1 << j
182         j = previous
183     order.reverse()
184     return order
185

```

```

186
187 def optimal_open_path(points: list[Vec2], start: Vec2 = (0.0, 0.0),
188                        exact_limit: int = EXACT_LIMIT) -> list[Vec2]:
189     """离线最优（或近优）开放路径：小规模精确，大规模 2-opt + Or-opt。"""
190     unique = _dedupe(points)
191     if len(unique) <= exact_limit:
192         return held_karp_open(unique, start)
193     return improve_open_path(unique, start)
194
195
196 def mst_weight(points: list[Vec2], start: Vec2 | None = (0.0, 0.0)) -> float:
197     """最小生成树权重 (Prim,  $O(n^2)$ )；含 start 时给出开放路径最优值的下界。
198
199     开放路径删除其最后一条边即得一条生成树，故 MST 权重  $\leq$  最优路径长度。
200     """
201     nodes = ([start] if start is not None else []) + list(points)
202     n = len(nodes)
203     if n <= 1:
204         return 0.0
205     in_tree = [False] * n
206     best = [float("inf")] * n
207     best[0] = 0.0
208     total = 0.0
209     for _ in range(n):
210         u, u_cost = -1, float("inf")
211         for i in range(n):
212             if not in_tree[i] and best[i] < u_cost:
213                 u, u_cost = i, best[i]
214         if u < 0:
215             break
216         in_tree[u] = True
217         total += u_cost
218         ux, uy = nodes[u]
219         for v in range(n):
220             if in_tree[v]:
221                 continue
222             distance = math.hypot(nodes[v][0] - ux, nodes[v][1] - uy)
223             if distance < best[v]:
224                 best[v] = distance
225     return total
226
227
228 def _dedupe(points, cell_m: float = 0.5) -> list[Vec2]:
229     """按 ``cell_m`` 网格去重（同一物理点只算一次作业）。"""
230     seen: set[tuple[int, int]] = set()
231     unique: list[Vec2] = []
232     for point in points:
233         key = (int(round(point[0] / cell_m)), int(round(point[1] / cell_m)))
234         if key in seen:
235             continue
236         seen.add(key)
237         unique.append((float(point[0]), float(point[1])))
238     return unique
239
240
241 def route_metrics(service_points, actual_move_m: float,
242                  speed_mps: float = 5.0) -> dict:
243     """路由度量：离线最优开放回路、MST 下界与相对 gap（S1 必报列）。"""
244     unique = _dedupe(service_points)
245     offline = path_length(optimal_open_path(unique)) if unique else 0.0
246     lower = mst_weight(unique)
247     gap = ((actual_move_m - offline) / actual_move_m) if actual_move_m > 0 else 0.0
248     return {
249         "distinct_service_points": len(unique),
250         "offline_opt_open_path_m": round(offline, 1),
251         "offline_opt_open_path_s": round(offline / speed_mps, 1),
252         "route_lower_bound_s": round(lower / speed_mps, 1),
253         "route_gap_vs_offline": round(gap, 4),
254     }

```

scripts/audit_paper_formal.py

```
1  """Audit only the formal exports selected in the official simulator's test folder.
2
3  Match the public activation-ticket digest to the official statistics queue,
4  and the complete package digest to its result sidecar. Export the selected
5  statistics and reproducible paper tables using the Python standard library.
6  """
7  from __future__ import annotations
8
9  import argparse
10 import base64
11 import csv
12 import hashlib
13 import json
14 import sqlite3
15 import struct
16 from decimal import Decimal, ROUND_HALF_UP
17 from pathlib import Path
18
19 ROOT = Path(__file__).resolve().parents[1]
20 FIELDS = (
21     "entered", "end_reason", "cleared_jammer_count", "measure_accepted_count",
22     "virtual_time_us", "program_run_duration_ms", "channel_switch_count",
23     "clear_failure_count", "state",
24 )
25
26
27 def sha(data: bytes) -> str:
28     return hashlib.sha256(data).hexdigest()
29
30
31 def public_header(path: Path) -> tuple[dict, str]:
32     data = path.read_bytes()
33     if data[:8] != b"JMBFLOG1" or len(data) < 14:
34         raise ValueError(f"Unsupported formal envelope: {path.name}")
35     version, size = struct.unpack(">HI", data[8:14])
36     if version != 1 or not 0 < size <= len(data) - 14:
37         raise ValueError(f"Invalid envelope header: {path.name}")
38     header = json.loads(data[14:14 + size])
39     if header["package_type"] != "formal_behavior_log":
40         raise ValueError("Practice logs are not admitted")
41     return header, sha(data)
42
43
44 def fixed(value: Decimal, digits: int = 3) -> str:
45     return str(value.quantize(Decimal(1).scaleb(-digits), rounding=ROUND_HALF_UP))
46
47
48 def derive(row: dict) -> dict:
49     count = row["cleared_jammer_count"]
50     total = Decimal(row["virtual_time_us"]) / 1_000_000
51     movement = total - (5 * row["measure_accepted_count"]
52         + row["channel_switch_count"] + 5 * count
53         + 3 * row["clear_failure_count"])
54     if count <= 0 or movement < 0:
55         raise ValueError("Invalid counts or negative inferred movement time")
56     return {
57         "virtual_time_s": str(total),
58         "average_clear_time_s": fixed(total / count, 6),
59         "program_run_duration_s": fixed(Decimal(row["program_run_duration_ms"]) / 1000),
60         "inferred_movement_time_s": str(movement),
61         "inferred_distance_m": str(5 * movement),
62         "movement_share_percent": fixed(100 * movement / total, 6),
63         "source_total": None,
64         "clear_fraction": None,
65     }
66
67
68 def collect(sim: Path) -> list[dict]:
```

```

69     selected = sorted((sim / "test").rglob("*.jlog"))
70     if not selected:
71         raise ValueError("No formal exports in test/")
72     queue = sim / "JammersSimulatorData/formal-statistics-queue.sqlite3"
73     original = sim / "JammersSimulatorData/behavior-logs"
74     # Read the live queue including its WAL, without writing to the simulator.
75     with sqlite3.connect(queue.resolve().as_uri() + "?mode=ro", uri=True) as db:
76         db.row_factory = sqlite3.Row
77         query = "SELECT activation_ticket_sha256," + ",".join(FIELDS) + " FROM statistics_tasks"
78         statistics = {r["activation_ticket_sha256"]: dict(r) for r in db.execute(query)}
79     records = []
80     for path in selected:
81         header, digest = public_header(path)
82         encoded = header["server_ticket_b64"]
83         ticket_hash = sha(base64.urlsafe_b64decode(encoded + "=" * (-len(encoded) % 4)))
84         sidecar = json.loads((original / (path.stem + ".result.json")).read_text("utf-8"))
85         for field in ("problem_no", "formal_index", "case_code"):
86             if header[field] != sidecar[field]:
87                 raise ValueError(f"Mismatched case: {path.name}")
88         if digest != sidecar["package_sha256"] or digest != sha((original / path.name).read_bytes()):
89             raise ValueError(f"Mismatched formal package: {path.name}")
90         matched = statistics[ticket_hash]
91         row = {name: header[name] for name in ("problem_no", "formal_index", "case_code", "
client_version")}
92         row.update({name: matched[name] for name in FIELDS})
93         row.update({
94             "file": path.relative_to(sim).as_posix(), "size_bytes": path.stat().st_size,
95             "package_sha256": digest, "activation_ticket_sha256": ticket_hash,
96             "created_at_utc": header["created_at_utc"],
97             "ended_at_utc": sidecar["ended_at_utc"],
98             "content_encryption": header["content_encryption"],
99             "identity_and_ticket_content_omitted": True,
100             "same_as_official_original": True,
101         })
102         row["derived"] = derive(row)
103         records.append(row)
104     keys = [(r["problem_no"], r["formal_index"]) for r in records]
105     if len(set(keys)) != len(keys):
106         raise ValueError("Duplicate formal attempt")
107     return records
108
109
110 def write_outputs(rows: list[dict], out: Path) -> None:
111     out.mkdir(parents=True, exist_ok=True)
112     snapshot = {
113         "schema": "paper-formal-audit-v1",
114         "selection": "Only Jammers-simulator-full/test/**/*.jlog",
115         "statistics_source": "Official formal-statistics-queue.sqlite3; matched by SHA-256 of the public
activation ticket",
116         "coverage": {str(p): sorted(r["formal_index"] for r in rows if r["problem_no"] == p)
for p in (3, 4)},
117         "measurement_scope": "Per-case official action counts, virtual duration and program duration",
118         "limitations": [
119             "Source totals, source types, solver commit and trajectories require additional evidence.",
120             "Queue state confirmed records a statistics submission acknowledgement.",
121         ],
122         "records": rows,
123     }
124
125     (out / "formal-results.json").write_text(json.dumps(snapshot, ensure_ascii=False, indent=2) + "\n",
"utf-8")
126     columns = ["problem_no", "formal_index", "case_code", *FIELDS, "package_sha256"]
127     with (out / "formal-results.csv").open("w", encoding="utf-8-sig", newline="") as handle:
128         writer = csv.DictWriter(handle, fieldnames=columns + list(rows[0]["derived"]))
129         writer.writeheader()
130         for row in rows:
131             writer.writerow({**{key: row[key] for key in columns}, **row["derived"]})
132     for problem in (3, 4):
133         by_index = {r["formal_index"]: r for r in rows if r["problem_no"] == problem}
134         lines = ["% Generated by scripts/audit_paper_formal.py; seconds; half-up rounding."]
135         for index in (1, 2, 3):

```

```

136         row = by_index.get(index)
137         if row is None:
138             lines.append(f"测试 {index} & 未提供记录 & -- & -- & -- " + r"\n")
139         else:
140             avg = Decimal(row["virtual_time_us"]) / 1_000_000 / row["cleared_jammer_count"]
141             lines.append(f"测试 {index} & \texttt{{{row['case_code']}}} & {row['cleared_jammer_count']} & {fixed(avg)} & {row['derived']['program_run_duration_s']} " + r"\n")
142             (out / f"formal-q{problem}-rows.tex").write_text("\n".join(lines) + "\n", "utf-8")
143     cost_lines, constants = [], []
144     for row in rows:
145         label = f"Q{row['problem_no']}-{row['formal_index']}"
146         total = Decimal(row["virtual_time_us"]) / 1_000_000
147         move = Decimal(row["derived"]["inferred_movement_time_s"])
148         cost_lines.append(f"{label} & {row['measure_accepted_count']} & {row['channel_switch_count']} & {row['clear_failure_count']} & {fixed(total)} & {fixed(move)} & {fixed(100*move/total, 2)}\% " + r"\n")
149     (out / "formal-cost-rows.tex").write_text("\n".join(cost_lines) + "\n", "utf-8")
150     for problem, word in ((3, "Three"), (4, "Four")):
151         group = [r for r in rows if r["problem_no"] == problem]
152         count = sum(r["cleared_jammer_count"] for r in group)
153         total = sum(Decimal(r["virtual_time_us"]) / 1_000_000 for r in group)
154         mean = sum(Decimal(r["virtual_time_us"]) / 1_000_000 / r["cleared_jammer_count"] for r in group) / len(group)
155         constants.extend([f"\newcommand{{\Q{word}Count}}{{{count}}}",
156                           f"\newcommand{{\Q{word}Total}}{{{fixed(total, 6)}}}",
157                           f"\newcommand{{\Q{word}Weighted}}{{{fixed(total/count)}}}",
158                           f"\newcommand{{\Q{word}Mean}}{{{fixed(mean)}}}"])
159     shares = [Decimal(r["derived"]["movement_share_percent"]) for r in rows]
160     runtimes = [Decimal(r["program_run_duration_ms"]) / 1000 for r in rows]
161     constants.extend([
162         f"\newcommand{{\FormalClearCount}}{{{sum(r['cleared_jammer_count'] for r in rows)}}}",
163         f"\newcommand{{\FormalMovementMin}}{{{fixed(min(shares), 2)}}}",
164         f"\newcommand{{\FormalMovementMax}}{{{fixed(max(shares), 2)}}}",
165         f"\newcommand{{\FormalRunMin}}{{{fixed(min(runtimes))}}}",
166         f"\newcommand{{\FormalRunMax}}{{{fixed(max(runtimes))}}}",
167     ])
168     (out / "formal-macros.tex").write_text("\n".join(constants) + "\n", "utf-8")
169     audit = ["# 正式测试数据核对", "", f"本次汇总 `Jammers-simulator-full/test` 中的 {len(rows)} 份正式导出包。",
170            "利用公开头部的案例信息、包哈希及票据哈希，与原始导出、结果侧文件和官方正式统计数据库建立对应关系。", "",
171            "| 案例 | 正式序号 | 原件一致 | 包 SHA-256 |", "| --- | --- | --- | --- |"]
172     for row in rows:
173         audit.append(f"| {row['case_code']} | Q{row['problem_no']}-{row['formal_index']} | 是 | `{row['package_sha256']}` |")
174     audit += ["", "虚拟时间保留官方整数微秒，程序运行时间采用官方毫秒。平均定位清除时间 = 总虚拟秒数 / 清除数。",
175            "移动时间由总时长减去检测、换频、成功及失败清除耗时计算。",
176            "`confirmed` 表示统计提交确认，`user_exit` 表示主动退出。",
177            "实验评价采用清除数量、逐案例平均定位清除时间、程序运行时间和动作耗时分解。",
178            "扩展评价所需的源总数与配对场景设计见论文未来工作。", ""]
179     (out / "audit.md").write_text("\n".join(audit), "utf-8")
180
181
182 def main() -> None:
183     parser = argparse.ArgumentParser(description=__doc__)
184     parser.add_argument("--sim-root", type=Path, default=ROOT / "Jammers-simulator-full")
185     parser.add_argument("--out", type=Path, default=ROOT / "docs/paper/data")
186     parser.add_argument("--snapshot", type=Path, help="Regenerate tables from the audited anonymous snapshot, without the simulator database")
187     args = parser.parse_args()
188     if args.snapshot:
189         rows = json.loads(args.snapshot.read_text("utf-8"))["records"]
190         for row in rows:
191             if derive(row) != row["derived"]:
192                 raise ValueError("Snapshot arithmetic changed")
193             path = args.sim_root / row["file"]
194             if path.exists() and public_header(path)[1] != row["package_sha256"]:
195                 raise ValueError("Snapshot and formal export disagree")
196     else:

```

```

197         rows = collect(args.sim_root)
198     write_outputs(rows, args.out)
199     for row in rows:
200         print(f"Q{row['problem_no']}--{row['formal_index']} {row['case_code']}: "
201               f"cleared={row['cleared_jammer_count']} average={row['derived']['average_clear_time_s']} s
202               "
203               f"real={row['derived']['program_run_duration_s']} s")
204
205 if __name__ == "__main__":
206     main()

```

scripts/build_paper_support.py

```

1  """Build vector figure sources, complete source listings, and a compact support ZIP.
2
3  Formal figures use the audited snapshot only. Geometric figures use declared
4  algorithm parameters, not fabricated observations or hidden simulator data.
5  """
6  from __future__ import annotations
7
8  import argparse
9  import hashlib
10 import json
11 import zipfile
12 from decimal import Decimal
13 from pathlib import Path
14
15 ROOT = Path(__file__).resolve().parents[1]
16 PAPER = ROOT / "docs/paper"
17
18 # Display copies only: executable Python files remain byte-for-byte unchanged.
19 ESCAPE_OPEN = "(" + "*"
20 ESCAPE_CLOSE = "@*" + ")"
21 LATEX_SYMBOLS = {
22     "\u03b1": r"\alpha", "\u03b4": r"\delta", "\u03b8": r"\theta",
23     "\u03bb": r"\lambda", "\u03c0": r"\pi", "\u03c1": r"\rho",
24     "\u03c6": r"\varphi", "\u2308": r"\lceil", "\u2309": r"\rceil",
25     "\u2200": r"\forall", "\u2203": r"\exists", "\u2205": r"\varnothing",
26     "\u2208": r"\in", "\u2227": r"\wedge", "\u2229": r"\cap",
27     "\u222a": r"\cup", "\u2248": r"\approx", "\u2264": r"\le",
28     "\u2265": r"\ge", "\u2286": r"\subseteq", "\u2287": r"\supseteq",
29     "\u27f9": r"\Longrightarrow",
30 }
31
32
33 def escaped_listing(source: str) -> str:
34     if ESCAPE_OPEN in source or ESCAPE_CLOSE in source:
35         raise ValueError("Source collides with the LaTeX escape delimiters")
36     return "".join(
37         ESCAPE_OPEN + r"\ensuremath{" + LATEX_SYMBOLS[c] + "}" + ESCAPE_CLOSE
38         if c in LATEX_SYMBOLS else c for c in source
39     )
40
41
42 def digest(data: bytes) -> str:
43     return hashlib.sha256(data).hexdigest()
44
45
46 def figures(rows: list[dict]) -> None:
47     out = PAPER / "figures"
48     out.mkdir(parents=True, exist_ok=True)
49     plan = json.loads((ROOT / "src/mcm26b/guarantee/plans/q4_mincover_v2.json").read_text("utf-8"))
50     lines = [
51         r"\begin{figure}[htbp]\centering",
52         r"\begin{tikzpicture}[x=0.00145cm,y=0.00145cm,font=\small]",
53         r"\fill[inkblue!5] (0,0) circle (1800);",
54         r"\draw[thick] (0,0) circle (1800);",
55         r"\draw[->,warmgray] (-2900,0)--(2950,0) node[right] {$x$/m};",

```

```

56         r"\draw[->,warmgray] (0,-2800)--(0,2850) node[above] {$y$/m};",
57     ]
58     for x in (-2000, -1000, 1000, 2000):
59         lines += [fr"\draw ({x},-35)--({x},35);",
60                 fr"\node[below,font=\scriptsize] at ({x},-50) {{{x}}};"]
61     for x, y in plan["points"]:
62         lines.append(fr"\fill[inkblue] ({x},{y}) circle (33);")
63     lines += [
64         r"\node[anchor=west,align=left] at (3250,900) {圆域: 源位置先验\蓝点: 31个检测站};",
65         r"\node[anchor=west,align=left] at (3250,-700) {部分站点在圆域外\用于接收向外发射的源};",
66         r"\end{tikzpicture}",
67         r"\caption{第四问集合覆盖模型得到的31个检测站及源位置区域。}",
68         r"\label{fig:q4plan}\end{figure}",
69     ]
70     (out / "q4-plan.tex").write_text("\n".join(lines) + "\n", "utf-8")
71     colors = ["inkblue", "inkblue!50", "warmgray!70", "black!18"]
72     bars = [
73         r"\begin{figure}[htbp]\centering",
74         r"\begin{tikzpicture}[x=0.00108cm,y=0.85cm,font=\small]",
75     ]
76     for j, row in enumerate(rows):
77         y = len(rows) - 1 - j
78         components = [Decimal(row["derived"]["inferred_movement_time_s"]),
79                       Decimal(5 * row["measure_accepted_count"]),
80                       Decimal(row["channel_switch_count"]),
81                       Decimal(5 * row["cleared_jammer_count"] + 3 * row["clear_failure_count"])]
82         label = f"Q{row['problem_no']}-{row['formal_index']}"
83         bars.append(fr"\node[anchor=east] at (-100,{y + 0.3}) {{{label}}};")
84         current = Decimal(0)
85         for amount, color in zip(components, colors):
86             bars.append(fr"\fill[{color}] ({current},{y}) rectangle ({current + amount},{y + 0.6});")
87             current += amount
88         bars.append(fr"\node[anchor=west,font=\scriptsize] at ({current + 80},{y + 0.3}) {{{current:.1f}}};")
89     bars.append(r"\draw[->] (0,-0.25)--(10500,-0.25);")
90     for tick in range(0, 10001, 2000):
91         bars.append(fr"\draw ({tick},-0.25)--({tick},-0.35) node[below,font=\scriptsize] {{{tick}}};")
92     bars.append(r"\node[anchor=west,font=\scriptsize] at (10400,-0.65) {s};")
93     for x, color, label in zip((0, 2600, 5200, 7600), colors, ("移动", "检测", "换频", "清除")):
94         bars += [fr"\fill[{color}] ({x},-1.3) rectangle ({x + 350},-1.05);",
95                 fr"\node[anchor=west] at ({x + 450},-1.175) {{{label}}};"]
96     bars += [
97         r"\end{tikzpicture}",
98         r"\caption{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。}",
99         r"\label{fig:costbars}\end{figure}",
100    ]
101    (out / "formal-time-bars.tex").write_text("\n".join(bars) + "\n", "utf-8")
102
103
104    def listings(rows: list[dict]) -> None:
105        data = PAPER / "data"
106        files = sorted((ROOT / "src/mcm26b").rglob("*.py"))
107        files += [ROOT / "scripts/audit_paper_formal.py",
108                 ROOT / "scripts/build_paper_support.py"]
109        text = [
110            r"\begin{group}",
111            r"\linespread{1.0}\selectfont",
112            r"\lstset{basicstyle=\ttfamily\fontsize{8}\fontsize{10}\selectfont,numberstyle=\tiny,"
113            r"aboveskip=6pt,belowskip=8pt,xleftmargin=1.5em,frame=none}",
114        ]
115        for path in files:
116            rel = path.relative_to(ROOT).as_posix()
117            display = data / "listing-source" / (rel + ".tex")
118            display.parent.mkdir(parents=True, exist_ok=True)
119            display.write_text(escaped_listing(path.read_text("utf-8")), "utf-8")
120            display_rel = display.relative_to(PAPER).as_posix()
121            escape_option = "escapeinside={" + ESCAPE_OPEN + "}" + "}" + ESCAPE_CLOSE + "}"
122            text += [fr"\subsection*{{\pathtext{{{rel}}}}}",
123                    fr"\lstinputlisting[{escape_option}]{{{display_rel}}}"
124            ]
125        text.append(r"\end{group}")

```

```

125 (data / "source-listings.tex").write_text("\n".join(text) + "\n", "utf-8")
126 table = [
127     r"\begin{longtable}{p{1.15cm}p{13.3cm}}",
128     r"\caption{所选正式导出文件与完整包哈希}\label{tab:packages}\\",
129     r"\toprule 案例 & 原始文件名及 SHA-256\\midrule\endfirsthead",
130     r"\toprule 案例 & 原始文件名及 SHA-256\\midrule\endhead",
131 ]
132 for row in rows:
133     h = row["package_sha256"]
134     label = f"Q{row['problem_no']}-{row['formal_index']}"
135     table += [
136         fr"{label} & \path{{{{Path(row['file']).name}}}}\\",
137         fr"& {{{\scriptsize\texttt{{{h[:32]}}}}}\allowbreak\texttt{{{h[32:]}}}}}\[4pt]",
138     ]
139     table += [r"\bottomrule\end{longtable}"]
140 (data / "formal-file-list.tex").write_text("\n".join(table) + "\n", "utf-8")
141
142
143 SUPPORT_README = """# 论文支撑材料
144
145 汇总官方模拟器 test 中的六份正式导出及哈希匹配的官方统计。
146 问题三、四各包含三次正式测试。支撑材料保留原始文件名及文件校验值。
147
148 ## 环境和运行
149
150 Python >=3.10; 安装 requests, 可选 numpy 加速。
151 在本目录使用 PowerShell:
152
153 $env:PYTHONPATH = (Resolve-Path src).Path
154 python -m mcm26b.cli --help
155 python -m mcm26b.cli run --problem 3 --strategy bayes --robot-id <模拟器登录队号>
156 python -m mcm26b.cli run --problem 4 --strategy directional --robot-id <模拟器登录队号>
157
158 在线命令需要先由使用者在官方模拟器中选择测试、待接口就绪。
159 策略的自检命令另需可选 referee 包。
160
161 ## 重算论文数据和图表
162
163 python scripts/audit_paper_formal.py --snapshot docs/paper/data/formal-results.json
164 python scripts/build_paper_support.py
165
166 上述命令在本地重算统计表、图表及源码附录。统计快照来自官方正式
167 统计队列，通过票据哈希匹配所选包。重新验证来源匹配需原电脑的统计队列。
168 连续覆盖与混合类型推断的扩展见论文未来工作。
169
170 ## 编译
171
172 安装 TeX Live (XeLaTeX, ctex, TikZ), 进入 docs/paper:
173
174 latexmk -xelatex -interaction=nonstopmode -halt-on-error main.tex
175
176 版式使用原项目模板。编译前请将原项目 docs/paper/fonts 目录放回
177 同一相对位置。使用 latexmk 自动完成交叉引用所需的多遍编译。
178 support-manifest.json 列出包内全部材料 (清单自身除外) 及 SHA-256。
179 """
180
181
182 def package(rows: list[dict]) -> Path:
183     selected = set((ROOT / "src/mcm26b").rglob("*.py"))
184     selected.update((ROOT / "src/mcm26b/guarantee/plans").glob("*.json"))
185     selected.update((PAPER / "data").rglob("*."))
186     selected.update((PAPER / "figures").glob("*.tex"))
187     selected.update((PAPER / "sections").glob("*.tex"))
188     selected.update([PAPER / "main.tex", PAPER / "mcm26b-paper.cls",
189                     PAPER / "README.md", PAPER / "REWRITE_NOTES.md",
190                     ROOT / "scripts/audit_paper_formal.py",
191                     ROOT / "scripts/build_paper_support.py"])
192     for row in rows:
193         path = ROOT / "Jammers-simulator-full" / row["file"]
194         if digest(path.read_bytes()) != row["package_sha256"]:

```

```

195         raise ValueError("Original formal package changed")
196     selected.add(path)
197     blobs = {p.relative_to(ROOT).as_posix(): p.read_bytes() for p in sorted(selected) if p.is_file()}
198     blobs["README.md"] = SUPPORT_README.encode("utf-8")
199     blobs["requirements.txt"] = b"requests>=2.31\nnumpy>=1.24\n"
200     manifest = [{"file": name, "bytes": len(value), "sha256": digest(value)} for name, value in sorted(
        blobs.items())]
201     blobs["support-manifest.json"] = (json.dumps(manifest, ensure_ascii=False, indent=2) + "\n").encode(
        "utf-8")
202     out = PAPER / "support-materials.zip"
203     with zipfile.ZipFile(out, "w", zipfile.ZIP_DEFLATED, compresslevel=9) as archive:
204         for name, value in sorted(blobs.items()):
205             entry = zipfile.ZipInfo(name, (2026, 9, 13, 0, 0, 0))
206             entry.compress_type = zipfile.ZIP_DEFLATED
207             archive.writestr(entry, value)
208     if out.stat().st_size > 20_000_000:
209         raise ValueError("Support archive exceeds 20 MB")
210     print(f"Support archive: {len(blobs)} files, {out.stat().st_size} bytes")
211     return out
212
213
214 def main() -> None:
215     parser = argparse.ArgumentParser(description=__doc__)
216     parser.add_argument("--package", action="store_true")
217     args = parser.parse_args()
218     rows = json.loads((PAPER / "data/formal-results.json").read_text("utf-8"))["records"]
219     figures(rows)
220     listings(rows)
221     if args.package:
222         package(rows)
223
224
225 if __name__ == "__main__":
226     main()

```