ITEM #229 - UnalignedAND-BTP Fish-Control Backbone - Exactness Guards, Recall Modes, and Anti-Leak Contracts

Conversation：UnalignedAND-BTP 算法加速

20260111

Authors: Sizhe Tan & GPT-Obot

ME：

奥博特，我们讨论：UnalignedAND-BTP(BucketTreeOfPermutation)的加速算法

1。UnalignedAND-BTP 是 DBM Metric Distance 的核心算法。对 GraphStarmap, SequenceStarmap, ImageStarmap
等非简单 Metric Space 结构来说，求可控精度解（non-heuristic-solution）而不可绕过的重要工具。

2。UnalignedAND-BTP 可解两类问题：
2A）Find Two Metric Space Points' Best Match or Top K Matches。 e.g.，Metric Distance；
2B) Given a number of known Metric Space Points, predict a large Metric Space Point (i.e., stracture)
as combination of the known Metric Space Points. e.g.，生物蛋白结构预测。

3。UnalignedAND-BTP 的 Computation Complexity 是较大的。尤其当一个或两个 Metric Space Point 所含的 Nodes 数量很大的时候。

任何能改善与加速UnalignedAND-BTP算法的工程与算法方式都是极其欢迎与意义重大的。我们把改进分两类：

3A）Improve Small StarmapA VS Large StarmapB UnalignedAND-BTP;

3B) Improve Large StarmapA VS Large StarmapB UnalignedAND-BTP;

4. For Problem 3A）- Improve Small Starmap VS Large Starmap UnalignedAND-BTP:

Two Approaches:

4A) Human Genomic Project like Approach:

a) Cut Large StarmapB into pieces, each pieces is larger than 3 times of Small StarmapA to have enough room to avoid missing matches in border area of pieces;

b) Find matches for each pair of SmallStarmapA VS LargeStarmapBPieces;

c) Collect all matches and eliminate duplicated matches;

done.

4B) Anchor Search Segmentation Tier-wise Matching/ZoomingIn Approach:

a1) In Small StarmapA, select some Anchors Nodes, and search Large StarmapB for anchor nodes's occurrences.

a2) At each of anchor nodes' occurrences, cut a large enough piece. Say LargeStarmapBPieces;

b) Find matches for each pair of SmallStarmapA VS LargeStarmapBPieces;

c) Collect all matches and eliminate duplicated matches;

```
            d) Repeat a) to c) until Anchors Nodes are not attactive enough anymore;

            done.

5. For Problem 3B) - Improve Large Starmap VS Large Starmap UnalignedAND-BTP:

        5A) Human Genomic Project like Approach:

            a) Cut Large StarmapA into small pieces;

            b) For each peach of LargeStarmapA,
                    b1) call above algo 4A) to find small-to-large matches with Large StarmapB;
                    b2) collect all matches

            d) For each match in the collected all matches in b),
                    if the match has no overlap with any other match and
                    if the match has no node at LargeStarmapA piece's border,  (**** very important to enable to catch border match(s) in subsequent process)
                        simplify both LargeStarmapA and LargeStarmapB by removing all nodes umples by this match;

            e) Repeat a) to d) until no more match can be collected.

            f) Finally, use algo 3A) to find matches of the final reduced LargeStarmapA and LargeStarmapB
               and collect the matches.

            done.

        5B) Anchor Search Segmentation Tier-wise Matching/ZoomingIn Approach:

            a) Cut Large StarmapA into small pieces;

            b) For each peach of LargeStarmapA,
                    b1) call above algo 4B) to find small-
```

```
to-large matches with Large StarmapB;
                    b2) collect all matches

            d) For each match in the collected all matches in b),
                    if the match has no overlap with any other match and
                    if the match has no node at LargeStarmapA piece's border,  (**** very important to enable to catch border match(s) in subsequent process)
                        simplify both LargeStarmapA and LargeStarmapB by removing all nodes umples by this match;

            e) Repeat a) to d) until no more match can be collected.

            f) Finally, use algo 3A) to find matches of the final reduced LargeStarmapA and LargeStarmapB
               and collect the matches.

            done.
```

奥博特，这套算法非常复杂，逻辑嵌套回转的很，有问题我当进一步誊清。

请评论与拓展。

AI：

下面我按“你现有思路的正确性边界 + 可落地的工程化增强 + 算法层可证明不漏解的剪枝/加速”来评论与拓展。整体结论：你提出的 4A/4B/5A/5B，本质上是在给 **UnalignedAND-BTP** 叠加一个“分治 + 候选生成 + 去重 + 逐步剥离(peeling)”的外层框架，这是正确方向；但要把它从“逻辑嵌套回转”变成“可实现、可测试、可控精度”，关键在于把每一步的 **不漏解条件**、**候选上界/下界**、**边界处理**、**冲突消解** 固化成明确 contract。

---

0. 先把问题抽象成一个统一形式

不论 Graph/Sequence/Image Starmap，UnalignedAND-BTP 都可视为：

- 输入：两个点（两个 Starmap）A、B，各含节点集合与关系结构（边/序/局部几何等）。
- 输出：
    - 2A：A→B 的最优匹配或 Top-K 匹配（以及距离、解释链）；
    - 2B：B 由若干已知小块组成的组合/拼装预测（本质仍是“多块匹配 + 组合一致性约束”）。

外层加速框架的目标是：**把 BTP 的“全局大搜索”改成“在少量候选片段上做 BTP”，并保证不会漏掉全局最优或 Top-K（或至少给出可证明的误差/召回界）。**

这会导出三类必须显式化的“可证明剪枝”工具：

1. **可计算的下界 LB / 上界 UB**：用于 branch-and-bound、Top-K 截断与片段级提前淘汰；
2. **候选生成的完备性条件**：切片/anchor 不能把真解排除在候选集合之外；
3. **冲突与去重规则**：多片段、多候选会产生重叠与重复映射，必须用一致的 canonical 形式处理。

---

1) 对 4A（HGP 切片）的评论与增强

1.1 你“每片 > 3×A”与“边界不漏”的本质

你在 4A(a) 的“片段要足够大以避免 border missing”是对的，但“3×”应当被替换为 **与匹配半径/结构直径相关的可证明条件**：

- 定义小图/小 starmap A 的“匹配影响半径” R(A)R(A)R(A)：

    - Sequence：窗口长度/最大对齐偏移；
    - Graph：A 中从 anchor 到最远节点的 hop/度量半径；
    - Image：A 的最大局部支持域/patch 尺寸等。
- 切片时对每片使用 **overlap halo**：片段核心区 + 两侧 halo 宽度 ≥ R(A)R(A)R(A)。
- 这样保证：任何落在核心区的真匹配，其所需上下文都完全包含在该片段内，不会被切断。

这比“3×”更可控，且能被单元测试验证。

1.2 片段级去重：必须给匹配一个 canonical key

4A(c) “eliminate duplicated matches”在工程上经常出错，建议把去重变成**确定性的 canonical 化**：

- MatchKey = normalize(mapping)：
    - 按 A 的节点顺序列出其映射到 B 的节点 id；
    - 或取映射边集合的最小哈希签名；
    - 再加上片段 id + 片段坐标做辅助（用于调试，而不是逻辑判等）。
- 对 Top-K：去重时要保留距离最小的那条（同 key 多来源）。

1.3 片段级剪枝：先算“便宜下界”，再决定是否跑 BTP

在每个片段上直接跑 UnalignedAND-BTP 仍然贵。应先在片段层做一个极快的 LB：

- 例：把 A 的节点特征做 multiset；片段也做 multiset；
    - LB ≈ unmatched feature mass × minCost
- 或用你们常用的“tier contribution”思想：前几层/前几项的必要贡献先算出 LB。
  若 LB 已经大于当前 Top-K 阈值，片段直接跳过（这是非启发式的剪枝，因为基于下界）。

---

2) 对 4B（Anchor 分段）的评论与增强

4B 本质是 **候选召回系统**：用少量 anchor occurrences 在 B 中生成候选片段集合，再在这些候选上跑 BTP。

2.1 Anchor 的选择标准：要“高区分度 + 可验证召回”

“anchor nodes not attractive enough”需要量化，否则会变成拍脑袋。建议定义 AnchorScore：

- **IDF/稀有度**：在 B 中出现次数越少越好（但不能为 0）；
- **稳定性**：anchor 的局部结构签名（邻接度/方向分解/局部 patch）越稳定越好；
- **覆盖性**：一组 anchors 在 A 上要覆盖不同区域，避免全在一个局部。

并给出一个硬约束来保证不漏解（或至少可控召回）：

- 使用 **多 anchor 联合**：只要真匹配存在，则至少一个 anchor 的 occurrence 会被命中。
- 对此需要：anchors 集合覆盖 A 的“必现特征”（例如 A 中若干不可替代的节点/局部结构）。

2.2 Occurrence 扩展成片段：用“结构半径”替代固定尺寸

与 4A 类似：从 anchor occurrence 扩成候选片段时，片段大小要由 R(A)R(A)R(A) 或“匹配直径”决定，而不是经验值。

2.3 候选片段的合并：先合并 occurrence 邻近区域，再跑 BTP

多个 anchor occurrence 可能落在同一真匹配区域附近，会产生大量重复片段。建议：

- 在 B 的坐标/图距离上把候选片段做一次 union/merge（类似区间合并或图上的 ball union）；

- 合并后再跑 BTP，可显著降本。

---

3) 对 5A/5B（Large vs Large + peeling）的评论与增强

你 5A/5B 的核心是：先把 A 切小，对每小块在 B 中找匹配，收集后做“无重叠且非边界”匹配的剥离，然后迭代，最后对缩小后的残差再做一次 3A。

这个 peeling 思路非常强，但要注意：**“无重叠 + 非边界”只是一个安全剥离条件的雏形**，还需要两个关键补全，否则容易在复杂结构里错剥或漏剥。

3.1 “冲突图 + 最大兼容集合”的形式化

收集到的 matches 之间可能冲突（共享 A 节点、共享 B 节点、结构不一致）。建议明确建立：

- ConflictGraph：match_i 与 match_j 若冲突则连边
- 目标：选一个 **兼容集合** S，使得
    - 总距离贡献最优（或总 gain 最大），并满足安全剥离条件

这在工程上可以用两级策略：

1. **安全优先**：先挑“强唯一”的 matches（高置信、低歧义、远离边界、LB-UB gap 小）；
2. 再对剩余冲突区域做局部求解（哪怕用较小规模的 BTP）。

这样逻辑就从“回转嵌套”变成“明确的冲突消解模块”。

3.2 边界规则要从“piece border”升级为“halo 可剥离区”

你强调 “piece border 节点不能剥离”是对的，但建议替换为：

- piece = core + halo

- **只剥离完全落在 core 内的匹配**，任何触及 halo 的匹配暂不剥离

这样天然保证：跨片段的真匹配会在后续迭代中以“另一个片段的 core”被捕获，不会因为早期切片而漏掉或错剥。

3.3 剥离后的残差更新：必须维护索引与一致性缓存

剥离节点后，BTP 若有 BucketTreeOfPermutation / inverted index / signature index，应增量更新，否则迭代会越来越慢或失真。工程上建议：

- ResidualStarmapView：逻辑删除 + 版本号
- IndexLayer：支持 tombstone / epoch-based rebuild
- Cache：LB/UB、anchor occurrences、片段合并结果，都要带版本戳，避免用旧缓存污染结果

---

4) 进一步的“加速抓手”：把 BTP 内核暴露出可用的 Bound 接口

你现在的外层框架，真正威力取决于：**BTP 是否能在“未完成匹配”状态给出 LB/UB**。建议把 UnalignedAND-BTP 内核改造成下面这种“可中断、可界限”的求解器形态：

- `step()`：扩展搜索一层/一个 bucket
- `currentBest()`：当前最优解
- `lowerBound(partialState)`：对任何未完成状态给下界
- `upperBound(partialState)`：可选（更强剪枝）

一旦有这个能力，你的 4A/4B/5A/5B 会立刻变得更“可控精度、非启发式”：

- Top-K：用第 K 个解的距离作为全局阈值
- 片段级：LB > 阈值直接跳

- 迭代级：当剩余残差的全局 LB 已经大于阈值或收益不足时提前收敛

---

5) 两个你会立刻遇到的“坑”与建议

5.1 “不漏解”与“Anchor 驱动”的天然张力

Anchor 方法若不加约束，很容易变成启发式召回（漏掉不含 anchor 的真解）。建议你明确两种运行模式：

- **Exact mode（不漏解）**：
    - anchors 只用于排序/优先级，不用于排除；
    - 排除必须来自可证明 LB（或必须覆盖所有必要特征的 anchor 集合）。
- **Recall-controlled mode（可控召回）**：
    - anchors 用于候选召回，但要输出“召回证据”与“漏解风险估计”（例如覆盖率指标）。

这样你对外就能非常清楚地说：哪些是“non-heuristic solution”，哪些是“trade-off solution”。

5.2 “去重/冲突消解”若不形式化，会把系统复杂度翻倍

强烈建议把 match 作为一等公民对象，具备：

- canonical key（去重）
- conflict predicate（冲突）
- explain chain（贡献拆解）
- safe peel predicate（是否可剥离）

这会让你后续写 JUnit 玩具测试非常顺滑：每条规则都能被独立验证。

---

6) 我给你一个更“收敛”的总流程（把回转逻辑摊平）

你现在的 3A/3B 可以统一成一个模板：

1. **Candidate Generation**
    - 4A：切片 + halo；或
    - 4B：anchors → occurrences → 合并片段（仍带 halo）
2. **Cheap Bound Filtering**（片段级 LB）
3. **Run BTP on Survivors**（可中断、可界限）
4. **Canonicalize + Dedup**
5. **Conflict Resolution**（最大兼容集合 / 安全优先）
6. **Safe Peeling (optional, for large-large)**（只剥离 core 内匹配）
7. **Iterate on residual**（版本化索引增量更新）
8. **Finalize**（残差上做一次更精细的 small-to-large 或全局 BTP）

这样无论 2A 还是 2B，都只是“是否启用 peeling + 是否启用组合一致性约束”的差别。

如果你接下来要“誊清”，我建议你优先把四个 Contract 写出来（每个 10 行以内就够）：

1. `PieceWithHalo` 的不漏解条件；
2. `AnchorSet` 的覆盖性条件（或召回模式声明）；
3. `MatchKey/Conflict/SafePeel` 三件套定义；
4. `BtpSolverBounds`：LB/UB 语义与单调性（能否用于剪枝）。

这四个写清楚，你这套复杂算法就会从“嵌套回转”变成“可证明、可工程落地、可单测”的体系。

讲的很好。请把四个 Contract 写出来

下面给出你要的“四个 Contract”。我按 **可直接落入 DBM/Java 接口注释与单测** 的风格写：每个 Contract 都包含 **目的、输入/输出、必须满足的不变式（Invariants）、失败条件、建议的最小测试点**。你后续可以把这些文字原样搬进 ITEM 或 JavaDoc。

Contract 1 — `PieceWithHalo`: 切片 + Halo 的不漏解条件

Purpose

在把大 Starmap 切成多个片段进行 small-to-large（或 large-to-large 的子问题）搜索时，保证**不会因为切片边界而漏掉任何“完整落在某片核心区(core)”的真匹配**。

Inputs

- `A`: Small Starmap（查询/模板）
- `B`: Large Starmap（被搜目标）
- `R(A)`: A 的匹配影响半径/结构直径（定义见下）
- `Piece`: 对 B 的一个片段视图，包含：
  - `core`: 片段核心区（用于允许剥离/接受匹配）
  - `halo`: core 周围扩展区（仅用于提供上下文，不作为可剥离区域）

Definitions

- **Influence Radius / Matching Radius `R(A)`**：A 中任意一个节点在匹配时可能需要的最远上下文距离上界。
  - Sequence：最大对齐偏移/窗口长度上界
  - Graph：以 anchor 或任意节点为中心的最大 hop 半径（或度量半径）
  - Image：patch 半径/最大局部支持域
- `support(M)`: 一个 match `M` 在 B 上使用到的节点集合（或边集合、或像素/patch 集合）
- `core(P)` / `halo(P)`：core 与 halo 的集合域，且 `core ⊂ (core ∪ halo)`

Guarantee (Non-missing)

对任意 match `M`（A ↔ B 的一个可行匹配），若：

1. `support(M) ⊆ core(P)`（匹配完全落在某片段的 core 内）
2. `haloWidth(P) ≥ R(A)`（该片段 halo 宽度不小于 A 的影响半径）
   则在 `BTP(A, P)`（仅在片段视图上运行 BTP）中，`M` 必须可被枚举/恢复（即不会因边界缺上下文被排除）。

Invariants

- **Coverage**：所有片段的 `core` 的并集覆盖 `B` 的搜索域（你想搜索的全部区域）。
- **Context Completeness**：对每个片段 `P`，`halo` 覆盖 `core` 内任意点半径 `R(A)` 的邻域。
- **Isolation**：`halo` 仅用于上下文，任何触及 `halo` 的匹配不得被标为“可剥离/可最终接受（exact accept）”，除非它在另一片段的 `core` 内再次出现。

Failure Conditions

- `haloWidth < R(A)`：可能漏解（跨边界的结构信息缺失）。
- 切片 `core` 覆盖不全：漏解。
- 对 Graph/Image 的 `R(A)` 估计过小：漏解风险。

Minimal Unit Tests

1. 构造一个真匹配跨越原始切片边界，但在另一片段的 core 内完整落入，验证不漏。
2. 把 haloWidth 减到 `< R(A)`，验证至少出现一个预期漏解（负例）。
3. 验证“触及 halo 的匹配不会被剥离”，避免错剥导致后续漏解。

Contract 2 — `AnchorSet`: Anchor 集合覆盖性与模式声明

Purpose

用 anchors 在 B 上生成候选片段（4B/5B），同时明确：系统运行在 **Exact Mode（不漏解）** 还是 **Recall-controlled Mode（可能漏解但给出风险指标）**。

Inputs

- `A`: Query Starmap
- `AnchorSet S = {a1..am}`：从 A 中选出的 anchor 节点/局部结构
- `Occ(B, ai)`：ai 在 B 中的出现位置集合（可由索引/签名检索得到）
- `Expand(occ, R(A))`：以 occ 为中心扩成 `PieceWithHalo`（haloWidth ≥ R(A)）

Required Mode Declaration

- `mode ∈ {EXACT, RECALL_CONTROLLED}`

EXACT Mode Contract (Non-missing Candidate Generation)

必须满足一个可验证的覆盖性条件。推荐两种可选 Contract（二选一即可）：

(E1) Must-hit Anchor Contract（强条件）

存在一个子集 `S* ⊆ S`，使得对任意真实最优/Top-K 匹配 `M`：

- 至少有一个 anchor `ai ∈ S*` 映射到 `support(M)` 内（即 `ai` 在该匹配中“被使用/被解释”），且
- `Occ(B, ai)` 的检索是完备的（不漏掉 B 中所有 ai 的合法出现）。

则由所有 `Expand(Occ(B, ai), R(A))` 产生的候选片段集合必须包含一个片段 `P`，使得 `support(M) ⊆ core(P)`（或至少 ⊆ piece domain 且在某片段 core 内可复现）。

(E2) Anchor-as-Priority Only（保守条件）

anchors 只用于**排序与优先级**，不得作为候选排除依据。系统仍必须保留一个兜底机制（例如 4A 的全覆盖切片，或全局 LB 驱动的遍历），确保不会因 anchor 召回失败而漏解。

解释：E2 在工程上更稳，E1 在性能上更强但对“must-hit”证明要求更高。

RECALL_CONTROLLED Mode Contract

允许漏解，但必须输出可度量的风险/覆盖指标，例如：

- `coverageScore`: anchors 在 A 上的覆盖比例（按节点/按 tier 权重）
- `selectivity`: anchors 在 B 上的平均出现次数（越少越好）
- `missRiskEstimate`: 基于未覆盖区域权重给出的漏解风险上界（可以是启发式，但要明确标注）

Invariants

- `Occ(B, ai)` 的检索过程必须**定义清楚**“什么叫同一 anchor 的出现”（签名一致性规则）。
- 多 anchor 候选片段必须做 merge/union（减少重复片段）。

Minimal Unit Tests

1. EXACT(E2)：禁用 anchor 召回也能靠兜底切片找到真解。
2. EXACT(E1)：构造一个真解必须包含某个稀有 anchor，验证候选片段一定覆盖真解。
3. RECALL：输出 coverage/风险指标随 anchor 集合变化而单调合理。

Contract 3 — `MatchKey / Conflict / SafePeel`: 去重、冲突与安全剥离三件套

Purpose

把“收集 matches → 去重 → 解决冲突 → 决定是否剥离(peel)”变成确定性的可测模块，避免 5A/5B 迭代中的错剥/漏剥。

Inputs

- `Match M`: 一个匹配结果，包含：
  - `mapAtoB`: A 节点到 B 节点的映射（可部分）
  - `supportA`, `supportB`: 使用到的节点集合
  - `distance`, `contributionChain`（可选）
  - `pieceId`（来源片段）
- `PieceWithHalo P`：该 match 所在片段（用于剥离判定）

3.1 MatchKey (Canonicalization) Contract

定义 `key(M)` 为 match 的**规范签名**，满足：

- **Determinism**：同一映射语义在不同片段/不同搜索路径下得到的 match，`key` 必须相同。
- **Collision Safety（弱要求即可）**：不同语义映射尽量不同 key；允许偶发哈希碰撞，但必须可二次验证（例如比对映射表）。

推荐 key 形式（可二选一）：

- `key = [B_id(map(a1)), B_id(map(a2)), ...]` 按 A 的固定顺序列出映射到 B 的 id（未映射用 sentinel）。
- 或 `key = hash(sortedEdgesMapped)`（对 Graph 更稳）。

Dedup Rule：同 key 的多条 match，仅保留 `distance` 最小者；若距离相同，保留 `supportB` 更小/更内聚者（减少冲突）。

3.2 Conflict Predicate Contract

定义 `conflict(Mi, Mj)` 必须满足：

- **Symmetry**：`conflict(i,j) == conflict(j,i)`
- **Monotonicity w.r.t. support**：若 Mi 与 Mj 在 A 或 B 上共享资源，则必须冲突（除非你显式支持“共享节点”语义）。

典型冲突条件：

- `supportA(Mi) ∩ supportA(Mj) != ∅` → 冲突
- `supportB(Mi) ∩ supportB(Mj) != ∅` → 冲突
- 或更强：结构一致性冲突（两个 match 对同一节点给出不同映射）

3.3 SafePeel Predicate Contract

定义 `safePeel(M, P)`：是否可以从 residual 中剥离该 match 覆盖的节点。

必须条件（建议作为硬条件）：

1. `supportB(M) ⊆ core(P)` 且 `supportA(M) ⊆ core(pieceA)`（若 A 也切片）
2. `M` 与当前选定兼容集合 `S` 内任意 match 不冲突
3. `M` 的解释链/贡献满足“锁定性”要求（可选但强烈推荐）：例如 LB-UB gap 很小或 distance 足够低，避免错剥。

剥离动作：

- 从 A_residual、B_residual 中逻辑删除 `supportA(M)` 与 `supportB(M)`，并记录剥离日志（用于可逆调试）。

Minimal Unit Tests

1. 两片段得到相同 match，key 去重保留最优距离。
2. 构造 support 重叠的 matches，conflict 必须为真。
3. support 触及 halo 的 match，safePeel 必须为假。
4. 剥离后 residual 上的后续匹配仍能找到跨边界真解（结合 Contract 1）。

Contract 4 — `BtpSolverBounds`: BTP 求解器的 LB/UB 语义与单调性

Purpose

把 UnalignedAND-BTP 从“黑盒一次跑完”升级为“可中断、可剪枝、可 Top-K 截断”的求解器，使外层 4A/4B/5A/5B 的加速成为**非启发式**（由下界证明安全）。

Inputs / State

- `State s`: BTP 搜索状态（部分映射/已选 bucket/已分配 permutation 前缀等）
- `costSoFar(s)`: 已确定部分的累积成本
- `remaining(s)`: 未解释/未匹配的剩余部分（A 或 B 的残余需求）
- 函数：
    - `LB(s)`: 对从 s 出发任何完成解的最小可能成本下界
    - `UB(s)`: 从 s 出发可构造的某个完成解的成本上界（可选但很有用）

Correctness Requirements

1. **Admissibility (Lower Bound is safe)**
   对任意从状态 s 可达到的完成解 `M`：
   `LB(s) ≤ cost(M)`
   这保证：若 `LB(s) > bestKThreshold`，则状态 s 可安全剪枝，不会漏掉 Top-K。
2. **Consistency / Monotonicity（建议）**

   若 `s'` 是 `s` 的扩展（做了更多决定）：
   `LB(s') ≥ LB(s)`
   这保证剪枝行为稳定、便于调试与性能预测。
3. **Upper Bound Validity（若提供 UB）**

   `UB(s)` 必须对应一个可验证的完成解（可以由 `constructSolution(s)` 给出并验证）。

   这用于快速降低 `bestKThreshold`，增强剪枝。
4. **Anytime Property（可中断）**

   求解器必须能在任意时刻返回：
    - 当前最优解 `bestSoFar`（可能非最优，但可验证可行）

   - 当前全局最小下界 `globalLB = min_s LB(s)`（用于判断离最优还差多少）
     当 `globalLB ≥ cost(bestSoFar)` 时，`bestSoFar` 即为全局最优（或 Top-K 的完备性满足）。

Minimal Unit Tests

1. 人工构造小规模问题，枚举所有完成解得到真实最优 `opt`，验证 `LB(root) ≤ opt`，且剪枝不漏最优。
2. 扩展状态序列验证 `LB` 单调不降（或至少不频繁违背）。
3. 若实现 UB：验证每个 UB 都能被 `verifySolution()` 通过。

这四个 Contract 一旦固化，你的 4A/4B/5A/5B 就会从“嵌套回转”变成四个可单测的模块拼装：
`PieceWithHalo`（不漏切片）+ `AnchorSet`（候选完备/模式声明）+ `MatchKey/Conflict/SafePeel`（迭代剥离正确性）+ `BtpSolverBounds`（非启发式剪枝与 Top-K）。

如果你下一步希望我把它们转成 **Java 8 接口 + Javadoc + 3–5 个 JUnit4 玩具测试**，我可以直接按这四个 Contract 出一套最小可跑骨架。

因问题比较长比较绕，有两点我没讲清楚，谢谢你的誊清：i）我用了笼统的 3X Size 说，不同 Starmap 的 Size 含义是不一样的，一般是指几步/Number of Steps/Lags; ii) Anchor search 找到 Occurrences 后要对 Occurrences 来求 CCC 以删除低质量的 Occurrences 添乱。

明白。你补充的两点非常关键，而且会直接改变 Contract 的写法与实现方式。我这里把它们“落地化”成两组明确的修订与扩展：一组解决 **Size/3X 的语义统一**，一组把 **Occurrence→CCC 过滤** 变成可证明、可单测的候选净化层。

I) “3× Size” 的语义统一：用 `Span/Reach` 代替 Size

你说的对：Graph/Sequence/Image 的 “Size” 不同，若继续用 “3×” 会让读者误解，也让工程参数失控。建议把“3× Size”的经验说法替换为一个统一的、与不漏解直接相关的量：

1) 定义两个统一标尺

(A) Query Reach / Influence Span：`Reach(A)`

表示：为了验证/完成一个 match，A 的任何局部结构可能需要的“最大跨越范围”。

- SequenceStarmap：
  `Reach(A) = maxLag(A)` 或 “最大允许对齐偏移 + 最大局部窗口半径”
  本质是你说的 “steps/lags”。
- GraphStarmap：
  `Reach(A) = maxHopRadius(A)`（从 anchor 或从 A 的“必要节点集”出发到最远约束节点的 hop 半径上界）
  若有权重边/度量边，用“图度量半径”也可。
- ImageStarmap：
  `Reach(A) = maxPatchRadius(A)` 或 “最大空间支持域直径”。

(B) Piece Core Span：`CoreSpan(P)`

表示：片段 core 的有效跨度（Sequence 就是长度；Graph 是 core 子图的直径/半径；Image 是 core 的宽高最小边等）。

2) 用一个可读的、不漏解的条件替代“3×”

把原先的经验式：

pieceSize ≥ 3×A

替换为：

`CoreSpan(P) ≥ SpanNeed(A)` 且 `HaloWidth(P) ≥ Reach(A)`

其中 `SpanNeed(A)` 是你希望 piece 至少能容纳 A 的“对齐自由度 + 内部结构跨度”的需求。对于 Sequence，最自然：

- `SpanNeed(A) = len(A) + 2*maxLag(A)`（示例：需要容纳 A 的长度以及向左右允许偏移的空间）
- `HaloWidth(P) = Reach(A) = maxLag(A)`（或更保守：maxLag + windowRadius）

这样读者会立刻理解：你所谓“3×”在时间序列场景下很可能是某个 `len + 2*lag` 的粗略近似；在 Graph/Image 则对应“core 必须大到覆盖 A 的可验证范围”。

3) 写法建议（给 ITEM 与接口注释）

- 把“3×”降级为“经验默认值”，放到 Implementation Notes；
- Contract 中只保留 `Reach(A)` / `SpanNeed(A)` 这类可解释、可计算的量。
  这样既保留你原意（片段要足够大），又避免“Size”歧义。

II) Anchor Occurrences 的 CCC 过滤：必须加一层 “Occurrence Quality Gate”

你说的第二点，本质是：anchor search 只是“粗召回”，Occurrences 里会混入大量噪声；必须对 Occurrence 求局部 CCC（Common Concept Core / 局部稳定核心）以删掉低质量 occurrence，避免候选爆炸与误导。

这可以非常干净地固化成一个标准模块：**Occurrence→CCC→Filter→Piece**。

1) 新增一个明确的 Contract：`OccurrenceCccGate`

Purpose
对每个 anchor occurrence 计算局部 CCC，并据此过滤，保证进入 BTP 的候选片段集合“更干净”，同时在 EXACT 模式下不引入漏解（或把漏解风险显式化）。

Inputs

- `A`：Query Starmap
- `anchor ai`
- `occ`：B 中某个 occurrence（位置/节点 id/局部片段引用）
- `CccExtractor`：从 `A` 与 `B@occ` 计算局部 CCC 的过程
- `QualityScorer`：基于 CCC 的质量评分/阈值策略
- `mode ∈ {EXACT, RECALL_CONTROLLED}`

Outputs

- `CccEvidence`:
  - `cccCore`：被认为相同/一致的局部核心
  - `cccSupportScore`：一致性强度
  - `cccAmbiguity`：歧义度（一个 occ 是否对应多种解释）
  - `rejectReason`（若被过滤）

2) EXACT vs RECALL 两种过滤策略（避免“过滤即启发式漏解”）

EXACT Mode：过滤只能基于“可证明不可能成为最优/Top-K”的理由

也就是说，过滤条件必须来自 **可证明下界**：

- 计算一个 occurrence 的快速下界：`LB_occ = LB( BTP(A, Piece(occ)) )` 或更便宜的下界近似（但必须仍是下界）。
- 若 `LB_occ > globalTopKThreshold`，则该 occ 可安全删除。
- CCC 在这里的作用是：更快更强地计算 `LB_occ`（例如 CCC 匹配不到必要核心时，下界立刻变大）。

RECALL_CONTROLLED Mode：允许按 CCC 质量阈值删噪声

例如：

- `cccSupportScore < t1` reject
- `cccAmbiguity > t2` reject
- `occurrenceFrequencyPenalty` reject（过于常见的 occurrence）

但必须输出：

- 过滤前后 occ 数量、覆盖度变化；
- 一个 “missRiskEstimate”（哪怕是启发式，也要明示）。

3) 你要的“CCC 删除低质量 Occurrences”可以这样定义得很 DBM

给一个非常 DBM 风味的 CCC 定义模板（可用于 Sequence/Graph/Image）：

- `CCC(A, B@occ)` 是在 A 与 B@occ 的局部视图中，能够在若干 tier 上同时满足的稳定结构集合：
  - tier-0：必要节点特征一致（类型/标签/方向分解等）
  - tier-1：必要关系一致（邻接/顺序/局部几何约束）
  - tier-2：更高阶结构一致（短环、motif、局部不变量等）

基于此给出一个“质量”：

- `supportScore = Σ (tierWeight * matchedMass)`
- `ambiguity = #alternativeMappingsWithinCore`
- `rarityBonus = IDF(coreSignature)`（核心越稀有越好）

然后过滤就变得非常清晰：

- “低质量” = supportScore 低 或 ambiguity 高 或 rarity 太低（太常见）。

4) 工程实现位置建议

把它放在 Anchor 管线中非常明确的阶段：

```
AnchorSelect(A)
→ OccurrenceSearch(B, anchor)
→ OccurrenceCccGate(A, B, anchor, occ) ← 你补充的关键步骤
→ ExpandToPieceWithHalo(occ)
→ PieceMerge/Union
→ CheapLBFiltering
→ RunBTP
```

这样读者与代码维护者一眼能看懂“CCC 是用来净化 Occurrences 的”。

对原四个 Contract 的最小修订清单

1. **PieceWithHalo**：把 “3× size” 改成 `Reach(A)` / `SpanNeed(A)` / `CoreSpan(P)` 的条件表达；并在注释里写“3× 只是经验默认值，等价于某些场景下 len+2*lag 的粗略近似”。
2. **AnchorSet**：在 occurrences 之后插入 `OccurrenceCccGate`，并把“候选完备性”明确拆成：
    - occurrence 检索完备性
    - CCC gate 的 exact/recall 模式约束
3. **MatchKey/Conflict/SafePeel**：不需要改变，但建议把 `CccEvidence` 放进 match 的 explain chain（这会让解释链更强）。
4. **BtpSolverBounds**：强调 CCC gate 若用于过滤，EXACT 模式必须依赖 LB，不得依赖纯质量阈值。

如果你愿意，我下一步可以把新增的 `OccurrenceCccGate` 也写成和前四个同规格的 Contract（10 行级），并给出一个极小 Java 8 接口组：`ICccExtractor / ICccQualityGate / OccurrenceCandidate / CccEvidence`，这样你就能直接开始接到现有 UnalignedAND-BTP 的 anchor 管线上。

请把新增的 OccurrenceCccGate 也写成和前四个同规格的 Contract（10 行级），并给出一个极小 Java 8 接口组

Contract 5 — `OccurrenceCccGate`（10 行级，同规格）

1. **Purpose**：对 `AnchorSearch` 得到的每个 `Occurrence` 计算局部 `CCC`（Common Concept Core）证据，并据此**过滤/降噪/排序**，控制候选爆炸，同时在 `EXACT` 模式下**不引入漏解**。

2. **Inputs**：`A`（query starmap）、`B`（target starmap）、`anchor`、`occurrence`、`Reach(A)`、`mode ∈ {EXACT, RECALL_CONTROLLED}`、`thresholdK`（当前 Top-K 阈值，可为空）。
3. **Outputs**：`GateDecision`：`ACCEPT | REJECT | DEFER`，以及 `CccEvidence`（core/signature、supportScore、ambiguity、lbHint、rejectReason）。
4. **CCC Extract**：`CccExtractor` 必须返回一个**确定性** `cccSignature`（用于去重/聚类 occurrences）与 `coreSupport`（被一致解释的局部核心集合）。
5. **EXACT Filtering Rule**：若 `mode=EXACT`，则 `REJECT` 只能由**可证明下界**触发：`lbHint > thresholdK`（thresholdK 非空时）或 `lbHint = +INF`（不可行）。禁止仅凭 “supportScore 太低” 直接拒绝。
6. **RECALL Filtering Rule**：若 `mode=RECALL_CONTROLLED`，允许用 `supportScore/ambiguity/rarity` 等阈值拒绝，但必须输出 `missRiskEstimate` 或至少输出 coverage 相关指标（如 accepted/total、被拒类别分布）。
7. **DEFER Semantics**：用于 “质量不高但不能证明无解” 的 occurrences：进入低优先级队列，只有当 `thresholdK` 变严或候选不足时再回补处理。
8. **Stability**：相同 `(A,B,anchor,occurrence,params)` 必须产生同一 `GateDecision` 与同一 `cccSignature`（便于缓存与复现）。
9. **Non-interference**：Gate 不得修改 A/B；只产出证据与决策。任何剥离/删除发生在后续模块（SafePeel）。
10. **Minimal Tests**：a) EXACT：低 support 但 lbHint 未超阈值 → 不得 REJECT；b) EXACT：lbHint=INF → 必须 REJECT；c) RECALL：阈值变化能显著减少 accepted 数且输出原因统计。

极小 Java 8 接口组（可直接 drop-in 骨架）

```
package com.dbm.metric.btp.gate;

import java.util.Collections;
import java.util.List;
import java.util.Map;
```

```
/**
 * OccurrenceCccGate: Anchor occurrence -> CCC evidence ->
(ACCEPT/REJECT/DEFER).
 *
 * Design notes:
 * - EXACT mode: may only REJECT based on provable lower-bound evidence
(lbHint).
 * - RECALL_CONTROLLED mode: may REJECT by quality thresholds, but must
report risk/coverage.
 */
public interface OccurrenceCccGate<A, B, Anchor, Occ> {

    GateResult evaluate(GateInput<A, B, Anchor, Occ> in);

    enum Mode { EXACT, RECALL_CONTROLLED }

    enum Decision { ACCEPT, REJECT, DEFER }

    /**
     * Immutable input bundle for a single occurrence.
     */
    final class GateInput<A, B, Anchor, Occ> {
        public final A query;
        public final B target;
        public final Anchor anchor;
        public final Occ occurrence;

        /** Reach(A): matching influence span / max context radius. */
        public final int reach;

        /** Optional current Top-K threshold. Null means "unknown yet". */
        public final Double topKThreshold;

        public final Mode mode;

        /** Free-form params (e.g., tier weights, ambiguity threshold, rarity
policy). */
        public final Map<String, Object> params;

        public GateInput(A query,
                         B target,
                         Anchor anchor,
                         Occ occurrence,
                         int reach,
                         Double topKThreshold,
                         Mode mode,
                         Map<String, Object> params) {
            this.query = query;
            this.target = target;
            this.anchor = anchor;
            this.occurrence = occurrence;
            this.reach = reach;
            this.topKThreshold = topKThreshold;
            this.mode = mode;
            this.params = params == null ? Collections.<String,
Object>emptyMap() : params;
        }
    }

    /**
     * Output decision + CCC evidence.
     */
    final class GateResult {
        public final Decision decision;
        public final CccEvidence evidence;

        public GateResult(Decision decision, CccEvidence evidence) {
            this.decision = decision;
            this.evidence = evidence;
        }

        public static GateResult accept(CccEvidence e) { return new
GateResult(Decision.ACCEPT, e); }
        public static GateResult reject(CccEvidence e) { return new
GateResult(Decision.REJECT, e); }
        public static GateResult defer(CccEvidence e)  { return new
GateResult(Decision.DEFER,  e); }
    }

    /**
     * CCC evidence produced for (anchor, occurrence).
     * Keep it minimal; downstream can log/explain/cluster based on signature
+ scores.
     */
    final class CccEvidence {
        /** Deterministic signature for de-dup / clustering of occurrences.
*/
        public final String cccSignature;

        /** [0..1] or unbounded; must be comparable within same gate config.
*/
        public final double supportScore;

        /** Higher means more alternative explanations/mappings (worse). */
        public final double ambiguity;

        /**
         * A provable (or safe) lower-bound hint for any full match starting
from this occurrence.
         * Use Double.POSITIVE_INFINITY for "provably impossible".
         */
        public final double lbHint;

        /** Optional: why rejected/deferred (debug + metrics). */
        public final String reason;

        /** Optional: small structured metrics for reporting/coverage. */
        public final Map<String, Double> metrics;

        public CccEvidence(String cccSignature,
                           double supportScore,
                           double ambiguity,
                           double lbHint,
                           String reason,
                           Map<String, Double> metrics) {
            this.cccSignature = cccSignature;
            this.supportScore = supportScore;
            this.ambiguity = ambiguity;
            this.lbHint = lbHint;
            this.reason = reason;
            this.metrics = metrics == null ? Collections.<String, 
Double>emptyMap() : metrics;
        }

        public static CccEvidence minimal(String sig, double support, double 
amb, double lbHint) {
            return new CccEvidence(sig, support, amb, lbHint, null, null);
        }
    }

    /**
     * CCC extractor: computes CCC evidence (core signature + quality signals 
+ lb hint).
     * This is intentionally separate to allow caching and alternative 
implementations.
     */
    interface CccExtractor<A, B, Anchor, Occ> {
        CccEvidence extract(GateInput<A, B, Anchor, Occ> in);
    }

    /**
     * Quality policy: maps evidence + mode + thresholdK -> decision.
     * EXACT mode must only REJECT based on lbHint (or lbHint=INF).
     */
    interface CccDecisionPolicy {
        Decision decide(Mode mode, Double topKThreshold, CccEvidence ev, 
Map<String, Object> params);
    }

    /**
     * Minimal default implementation pattern: extractor + decision policy.
     */
    final class Default<A, B, Anchor, Occ> implements OccurrenceCccGate<A, B, 
Anchor, Occ> {
        private final CccExtractor<A, B, Anchor, Occ> extractor;
        private final CccDecisionPolicy policy;

        public Default(CccExtractor<A, B, Anchor, Occ> extractor, 
CccDecisionPolicy policy) {
            this.extractor = extractor;
            this.policy = policy;
        }

        @Override
        public GateResult evaluate(GateInput<A, B, Anchor, Occ> in) {
            CccEvidence ev = extractor.extract(in);
            Decision d = policy.decide(in.mode, in.topKThreshold, ev, 
in.params);
            if (d == Decision.ACCEPT) return GateResult.accept(ev);
            if (d == Decision.DEFER)  return GateResult.defer(ev);
            return GateResult.reject(ev);
        }
    }

    /**
     * Optional: batch statistics for reporting (useful in RECALL_CONTROLLED
mode).
     * Kept as an interface so you can plug it into the pipeline without hard
dependency.
     */
    interface GateStatsSink {
        void onEvaluated(Mode mode, Decision decision, CccEvidence evidence);
        default void onBatchDone() {}
    }
}
```

备注（工程落点）

- 这个接口组刻意 **不绑定** 你的 Starmap 类型：用泛型 `A/B/Anchor/Occ`，你可以在 `com.dbm.starmap.*` 里用 adapter 包一层。
- `lbHint` 是把“CCC 过滤在 EXACT 模式下不漏解”落地的关键抓手：只要你能从 CCC 快速推出一个安全下界，就能合法 REJECT。
- `DEFER` 是把“低质量但不能证明无解”的 occurrence 从主流程移走，避免干扰，但保留兜底完备性。

如果你下一步要继续，我建议紧接着补一个极小 `CccDecisionPolicy` 默认实现（EXACT/RECALL 两套阈值规则），以及一个 `CccExtractor` 的 toy 版（比如基于 tier-0/1 的签名与支持度），便于先把管线跑起来。

请补一个极小 CccDecisionPolicy 默认实现（EXACT/RECALL 两套阈值规则），以及一个 CccExtractor 的 toy 版（比如基于 tier-0/1 的签名与支持度），便于先把管线跑起来。

```
package com.dbm.metric.btp.gate;

import java.util.HashMap;
import java.util.Map;
import java.util.Objects;

/**
 * Minimal default implementations:
 * - DefaultCccDecisionPolicy: EXACT/RECALL_CONTROLLED decision rules.
 * - ToyTier01CccExtractor: toy CCC extractor using tier-0/1 signature &
support scoring.
 *
 * This is intentionally generic; plug adapters for your specific
Starmap/Anchor/Occurrence types.
```

```
 */
public final class CccGateDefaults {

    private CccGateDefaults() {}

    /**
     * Default policy parameters (keys in GateInput.params):
     * - "recall.minSupport" (Double, default 0.25)
     * - "recall.maxAmbiguity" (Double, default 0.75)
     * - "recall.deferSupport" (Double, default 0.35)  // between minSupport
and deferSupport -> DEFER
     * - "recall.deferAmbiguity" (Double, default 0.90) // between
maxAmbiguity and deferAmbiguity -> DEFER
     *
     * EXACT mode rule:
     * - If ev.lbHint is +INF => REJECT (provably impossible)
     * - Else if topKThreshold != null and ev.lbHint > topKThreshold =>
REJECT (safe prune)
     * - Else ACCEPT (never reject on support/ambiguity in EXACT mode)
     *
     * RECALL_CONTROLLED mode rule:
     * - If ev.lbHint is +INF => REJECT
     * - Else if support < minSupport => REJECT
     * - Else if ambiguity > deferAmbiguity => REJECT
     * - Else if support < deferSupport => DEFER
     * - Else if ambiguity > maxAmbiguity => DEFER
     * - Else ACCEPT
     */
    public static final class DefaultCccDecisionPolicy implements
OccurrenceCccGate.CccDecisionPolicy {

        @Override
        public OccurrenceCccGate.Decision decide(OccurrenceCccGate.Mode mode,
                                                 Double topKThreshold,
                                                 OccurrenceCccGate.CccEvidence
ev,
                                                 Map<String, Object> params) {
            Objects.requireNonNull(mode, "mode");
            Objects.requireNonNull(ev, "ev");

            // Always safe to reject if provably impossible.
            if (Double.isInfinite(ev.lbHint) && ev.lbHint > 0) {
                return OccurrenceCccGate.Decision.REJECT;
            }

            if (mode == OccurrenceCccGate.Mode.EXACT) {
                // Safe prune only by lower bound (when threshold is known).
                if (topKThreshold != null && ev.lbHint >
topKThreshold.doubleValue()) {
                    return OccurrenceCccGate.Decision.REJECT;
                }
                // Never reject by quality signals in EXACT mode.
                return OccurrenceCccGate.Decision.ACCEPT;
            }

            // RECALL_CONTROLLED: allow quality-based decisions
            double minSupport = getDouble(params, "recall.minSupport", 0.25);
            double maxAmb = getDouble(params, "recall.maxAmbiguity", 0.75);
            double deferSupport = getDouble(params, "recall.deferSupport",
0.35);
            double deferAmb = getDouble(params, "recall.deferAmbiguity",
0.90);

            // Hard rejects
            if (ev.supportScore < minSupport) {
                return OccurrenceCccGate.Decision.REJECT;
            }
            if (ev.ambiguity > deferAmb) {
                return OccurrenceCccGate.Decision.REJECT;
            }

            // Soft defers
            if (ev.supportScore < deferSupport) {
                return OccurrenceCccGate.Decision.DEFER;
            }
            if (ev.ambiguity > maxAmb) {
                return OccurrenceCccGate.Decision.DEFER;
            }

            return OccurrenceCccGate.Decision.ACCEPT;
        }

        private static double getDouble(Map<String, Object> params, String
key, double def) {
            if (params == null) return def;
            Object v = params.get(key);
            if (v == null) return def;
            if (v instanceof Number) return ((Number) v).doubleValue();
            try {
                return Double.parseDouble(String.valueOf(v));
            } catch (Exception ignore) {
                return def;
            }
        }
    }

    /**
     * Toy CCC extractor based on tier-0/1:
     * - tier-0: anchor "type" token + local feature hash (provided via
adapters)
     * - tier-1: a small local neighborhood signature (provided via adapters)
     *
     * To keep it generic, you provide a ToyAdapter that can:
     * - compute a deterministic tier0 token for (A, anchor) and (B,
occurrence)
     * - compute a deterministic tier1 token for (A, anchor) and (B,
occurrence)
     * - estimate occurrence frequency/rarity for ambiguity scoring
(optional)
     * - provide a safe lower-bound hint (optional, else 0)
     *
     * Scoring:
     * - supportScore in [0,1]:
     *     1.0 if tier0 matches and tier1 matches
     *     0.6 if tier0 matches but tier1 mismatch
     *     0.0 if tier0 mismatch
     * - ambiguity:
     *     base on frequency if available; else:
     *       0.2 (both match), 0.5 (only tier0 match), 0.9 (tier0 mismatch)
     *
     * lbHint:
     * - if tier0 mismatch => +INF (provably impossible for this toy
definition)
     * - else adapter.safeLowerBoundHint(...) if provided, else 0
     *
     * IMPORTANT:
     * - In your real system, "tier0 mismatch => INF" must be justified; keep
this toy conservative
     *   by only using it when tier0 corresponds to a MUST-MATCH anchor
constraint.
     */
    public static final class ToyTier01CccExtractor<A, B, Anchor, Occ>
            implements OccurrenceCccGate.CccExtractor<A, B, Anchor, Occ> {

        public interface ToyAdapter<A, B, Anchor, Occ> {
            /** Must be deterministic and stable. */
            String tier0TokenForQuery(A query, Anchor anchor);

            /** Must be deterministic and stable. */
            String tier1TokenForQuery(A query, Anchor anchor);

            /** Must be deterministic and stable. */
            String tier0TokenForOccurrence(B target, Occ occurrence, Anchor
anchor);

            /** Must be deterministic and stable. */
            String tier1TokenForOccurrence(B target, Occ occurrence, Anchor
anchor);

            /**
             * Optional: frequency of this occurrence signature in B for
ambiguity estimation.
             * Return null if not available.
             */
            Integer occurrenceFrequencyHint(B target, Occ occurrence, Anchor
anchor);

            /**
             * Optional: a safe lower bound hint for any full match rooted at
this occurrence.
             * Return null if not available (toy defaults to 0).
             */
            Double safeLowerBoundHint(OccurrenceCccGate.GateInput<A, B,
Anchor, Occ> in,
                                      String tier0OccToken,
                                      String tier1OccToken);
        }

        private final ToyAdapter<A, B, Anchor, Occ> adapter;

        public ToyTier01CccExtractor(ToyAdapter<A, B, Anchor, Occ> adapter) {
            this.adapter = Objects.requireNonNull(adapter, "adapter");
        }

        @Override
        public OccurrenceCccGate.CccEvidence
extract(OccurrenceCccGate.GateInput<A, B, Anchor, Occ> in) {
            String q0 = adapter.tier0TokenForQuery(in.query, in.anchor);
            String q1 = adapter.tier1TokenForQuery(in.query, in.anchor);

            String o0 = adapter.tier0TokenForOccurrence(in.target,
in.occurrence, in.anchor);
            String o1 = adapter.tier1TokenForOccurrence(in.target,
in.occurrence, in.anchor);

            boolean tier0Match = safeEq(q0, o0);
            boolean tier1Match = safeEq(q1, o1);

            // Deterministic signature: anchor tier0 + tier1 + occurrence
tier0 + tier1
            // Keep it short but stable.
            String sig = "T0:" + norm(q0) + "|" + norm(o0) + ";T1:" +
norm(q1) + "|" + norm(o1);

            double support;
            double ambiguity;
            double lbHint;
            String reason = null;

            if (!tier0Match) {
                support = 0.0;
                ambiguity = 0.95;
                lbHint = Double.POSITIVE_INFINITY;
                reason = "tier0_mismatch";
            } else {
                support = tier1Match ? 1.0 : 0.6;

                // Ambiguity heuristic using occurrence frequency if
available
                Integer freq = adapter.occurrenceFrequencyHint(in.target,
in.occurrence, in.anchor);
                if (freq != null && freq.intValue() > 0) {
                    // More frequent -> more ambiguous; map freq to
(0.2..0.95) smoothly.
                    ambiguity = clamp(0.2 + Math.log10(1.0 +
freq.doubleValue()) / 3.0, 0.2, 0.95);
                } else {
                    ambiguity = tier1Match ? 0.2 : 0.5;
                }

                Double lb = adapter.safeLowerBoundHint(in, o0, o1);
                lbHint = (lb == null) ? 0.0 : Math.max(0.0,
lb.doubleValue());
                if (!tier1Match) reason = "tier1_mismatch";
            }

            Map<String, Double> metrics = new HashMap<String, Double>();
            metrics.put("tier0Match", tier0Match ? 1.0 : 0.0);
            metrics.put("tier1Match", tier1Match ? 1.0 : 0.0);
            metrics.put("support", support);
            metrics.put("ambiguity", ambiguity);

            return new OccurrenceCccGate.CccEvidence(sig, support, ambiguity,
lbHint, reason, metrics);
        }

        private static boolean safeEq(String a, String b) {
            return a != null && b != null && a.equals(b);
        }

        private static String norm(String s) {
            return s == null ? "NULL" : s;
        }

        private static double clamp(double v, double lo, double hi) {
            if (v < lo) return lo;
            if (v > hi) return hi;
            return v;
        }
    }
}
```

使用方式（最小跑通）

- `CccExtractor`：先用 `ToyTier01CccExtractor` + 你自己的 `ToyAdapter`（把 anchor/occurrence 的 tier0/tier1 token 取出来即可）。
- `CccDecisionPolicy`：直接用 `DefaultCccDecisionPolicy`。
- 组合成 Gate：`new OccurrenceCccGate.Default<>(extractor, policy)`，即可在 AnchorSearch 后对每个 occurrence 调 `evaluate()`。

两个关键提醒（避免误用）

1. toy extractor 里把 `tier0 mismatch => lbHint=INF`，只有当你的 tier0 真的是 “must-match 核心约束” 才安全；否则你应改成 `lbHint` 给一个较大但非 INF 的下界（EXACT 模式下就不会被直接 REJECT）。
2. EXACT 模式下默认 policy 不会因 `supportScore` 低而 REJECT；若你想“更 aggressive”也必须先把“支持度低 ⇒ LB 超阈值”的证明（或安全下界）补齐。

很好。这是一个比较复杂的计算系统，不是一两次讨论就能完全搞清楚的，编程实现还会出现未预料到的问题。主要是怕有/也会有逻辑覆盖漏洞导致漏掉有效解。我们先把大的思路与骨架固定住，先达到顶级的“鱼控”，再腾出手来解决次一级的算法与工程问题。

同意你的策略：先把“顶级鱼控”的**方向盘与骨架**固定住，再把复杂性压到可验证、可迭代的子模块里。为了最大限度降低“逻辑覆盖漏洞导致漏解”的风险，我建议你把系统先以**两种明确运行形态** 固化，并给每一层都加上“可证明兜底”。

1) 固化两种运行形态（避免口径漂移）

A. EXACT Fish-Control（不漏解优先）

目标：在可接受算力内尽量做到 Top-K 完备；任何“剪枝/过滤”必须可证明安全。

- 片段：`PieceWithHalo` 用 `Reach(A)` / `SpanNeed(A)` 约束；core 覆盖全域。
- Anchor：只做 **优先级排序**（E2），不做排除；或在满足 must-hit 证明时才排除（E1）。
- CCC Gate：只能用 `lbHint` + `thresholdK` 做 REJECT；低质量只允许 DEFER。
- BTP：必须提供 `LB`（最低要求），有 `UB` 更好；全局阈值随 bestK 收紧。

B. RECALL-Controlled Fish-Control（吞吐优先）

目标：显著降本，允许漏解但必须“可观察、可量化、可回补”。

- Anchor + CCC Gate：用阈值直接 REJECT，同时输出 coverage/missRisk。
- 保留 DEFER 队列：当候选不足或结果不稳定时，回补低优先级 occurrences/pieces。

这两种形态一旦写入 ITEM/代码注释，团队未来不会在“到底是不是启发式”上反复摇摆。

2) 你最担心的“漏解漏洞”，用三道兜底阀门锁住

1. **Coverage 兜底**（切片层）：core 并集覆盖搜索域；halo 满足 Reach(A)。
2. **Candidate 兜底**（召回层）：EXACT 模式下 anchors 不得成为排除条件（除非 must-hit 可证明）。
3. **Filter 兜底**（CCC Gate）：EXACT 模式下不允许质量阈值 REJECT，只能 DEFER 或用 lbHint 安全剪枝。

这三条是“鱼控”层面的硬护栏；它们让你即使后续实现细节出错，也更可能表现为“变慢/候选多”，而不是“静默漏解”。

3) 把“复杂回转逻辑”收敛为可测试的 8 段流水线

你现在已经基本具备可固化的主干：

1. `Reach(A)` / `SpanNeed(A)` 计算
2. `PieceWithHalo` 生成（全覆盖切片或 occurrence 扩张）
3. Anchor occurrences 检索
4. `OccurrenceCccGate`（ACCEPT/DEFER/REJECT）
5. piece merge/union
6. piece-level cheap LB 过滤
7. piece-level BTP（anytime + LB/UB）
8. `MatchKey/Conflict/SafePeel`（large-large 时迭代）

建议你在代码结构上让这 8 段每段都有单测入口与日志统计（counts、阈值、LB 分布、reject 原因分布）。这会让“未预料到的问题”变成可定位、可回归的 bug。

4) 建议你现在就固定的“三份最小工件”

不需要一次性写大系统，先把下面三样落地，后续所有增强都能挂上去：

1. **Contracts 文档**：你现在已有 1–5，建议合并成一页“UnalignedAND-BTP Fish-Control Contracts”。

2. **接口组**：`PieceWithHalo`、`OccurrenceCccGate`（已给）、`BtpSolverBounds`、`MatchKey/Conflict/SafePeel`。
3. **Toy Demo + JUnit4**：用小规模 Sequence（steps/lags）先把 EXACT 模式跑通：
    - 故意制造边界匹配、重复 occurrence、低质量 occurrence、冲突 match
    - 验证：EXACT 不漏、RECALL 可控、DEFER 可回补

如果你认可这个"先固化鱼控骨架"的优先级，下一步我建议把你刚才的共识正式固化成一个 ITEM（中英双语），标题类似：

ITEM: UnalignedAND-BTP Fish-Control Backbone — Exactness Guards, Recall Modes, and Anti-Leak Contracts

并附一个"8 段流水线总图（Mermaid）+ Contract 表格"。这样你后续每次迭代都不会偏离主轴。

讲的很好。谢谢。请生成 ITEM #229: UnalignedAND-BTP Fish-Control Backbone — Exactness Guards, Recall Modes, and Anti-Leak Contracts。 并附一个"8 段流水线总图（Mermaid）+ Contract 表格"。

下面给出 **DBM-COT ITEM #229** 的**正式固化版本**。内容按你一贯的 ITEM 风格组织：**问题定位 → 总体骨架 → 两种运行形态 → Anti-Leak Contracts → 8 段流水线总图（Mermaid）→ Contract 表格 → 工程落点与后续演进**。
（本 ITEM 是"鱼控级"骨架文档，**不追求细节最优，只追求不漏解的方向盘正确**。）

ITEM #229 — UnalignedAND-BTP Fish-Control Backbone

Exactness Guards, Recall Modes, and Anti-Leak Contracts

Category

DBM-COT · Metric Space Intelligence · UnalignedAND-BTP · Fish-Control Backbone

Status

Backbone / Architecture-level (stable direction anchor)

---

0. Motivation & Positioning

UnalignedAND-BTP (BucketTreeOfPermutation) 是 DBM Metric Distance 的核心内核算法，用于解决：

- **2A**：两个复杂 Metric Space 点（Graph / Sequence / Image Starmap）的 **最优或 Top-K 匹配**；
- **2B**：由若干已知小结构 **组合/预测一个大型结构**（如生物分子、复杂模式拼装）。

该算法计算复杂度高、结构嵌套深，任何工程级加速都极易引入**静默漏解（silent miss）**。
因此，本 ITEM 不讨论具体优化技巧，而是**固定一套“鱼控级”骨架**：

先确保不漏解（Exactness Guards），再允许可控降本（Recall Modes），
用 Contracts 把所有潜在逻辑漏洞显式化并锁住。

---

1. Two Canonical Operating Modes（两种规范运行形态）

1.1 EXACT Fish-Control Mode（不漏解优先）

目标：在算力允许范围内，保证 Top-K / 最优解的完备性。

原则：

- 任何 **REJECT / 剪枝** 必须来自**可证明的下界（LB）**；
- Anchor / CCC / Quality **只能排序或 DEFER，不能拍脑袋排除**；
- 慢可以，漏不行。

典型适用：

- 算法验证阶段
- 论文级结果
- 结构预测/科学计算的“最终答案”阶段

1.2 RECALL-Controlled Fish-Control Mode（吞吐优先）

目标：显著降本提速，允许漏解，但必须 **可观测、可量化、可回补**。

原则：

- Anchor + CCC Gate 可直接 REJECT 低质量候选；
- 必须输出 coverage / miss-risk / reject-reason 统计；
- 必须保留 **DEFER / 回补通道**。

典型适用：

- 大规模扫描
- 实时系统
- 预筛 / warm-up 阶段

---

2. Anti-Leak Philosophy（反漏解设计哲学）

所有加速都可能出错，因此系统必须具备三道**硬兜底阀门**：

1. **Coverage Guard（切片层）**
    - 搜索域必须被 `core` 并集覆盖；
    - `halo ≥ Reach(A)`，防止边界漏解。
2. **Candidate Guard（召回层）**

    - EXACT 模式下，Anchor **不得成为排除条件**（除非 must-hit 有证明）。
3. **Filter Guard（过滤层）**
    - EXACT 模式下，CCC/质量指标 **只能 DEFER，不能 REJECT**；
    - 只有 `LB > threshold` 才允许 REJECT。

这三道阀门的作用是：

宁可慢、宁可候选多，也不允许静默漏掉有效解。

---

3. Core Concepts（统一抽象）

3.1 Reach / Span 统一尺度

为避免 “3× size” 的歧义，统一使用：

- **Reach(A)**：A 的匹配影响半径（steps / lags / hops / patch radius）
- **SpanNeed(A)**：A 完整可验证匹配所需的最小跨度
- **CoreSpan(P)**：Piece 核心区的有效跨度

不漏解条件：

```
CoreSpan(P) ≥ SpanNeed(A)
HaloWidth(P) ≥ Reach(A)
```

3.2 CCC as Evidence, not Magic

CCC（Common Concept Core）不是“神秘判定器”，而是：

- 一种 **Occurrence 级的证据压缩**；
- 用于 **下界估计**、**排序**、**解释链**；
- 在 EXACT 模式下 **不得单独作为排除依据**。

---

4. The 8-Stage Fish-Control Pipeline

Overview

UnalignedAND-BTP 的所有加速与控制，最终被摊平成 **8 段线性流水线**。

Mermaid: 8-Stage Pipeline

```
flowchart TD
    A[1. Compute Reach(A) / SpanNeed(A)]
    B[2. Generate Pieces with Core + Halo]
    C[3. Anchor Search → Occurrences]
    D[4. OccurrenceCccGate\n(ACCEPT / DEFER / REJECT)]
    E[5. Piece Merge / Union]
    F[6. Piece-level Cheap LB Filter]
    G[7. UnalignedAND-BTP\n(Anytime + LB/UB)]
    H[8. MatchKey / Conflict\n/ SafePeel & Iterate]

    A --> B
    B --> C
    C --> D
    D --> E
    E --> F
    F --> G
    G --> H
    H -->|residual| B
```

---

5. Contract Table（反漏解契约总表）

#	Contract	Scope	Core Guarantee	Leak Risk if Violated
C1	**PieceWithHalo**	切片	Core 覆盖 + Halo ≥ Reach(A) ⇒ 不因边界漏解	高（跨边界真解丢失）
C2	**AnchorSet**	召回	EXACT 模式下 Anchor 只排序不排除	中-高
C3	**OccurrenceCccGate**	过滤	EXACT：仅 LB 可 REJECT； RECALL：需风险标注	高（静默漏解）

#	Contract	Scope	Core Guarantee	Leak Risk if Violated
C4	**MatchKey / Conflict / SafePeel**	组合	去重确定性 + 冲突显式化 + 仅 core 内可剥离	中
C5	**BtpSolverBounds**	内核	LB 可证明、单调；支持 Top-K 截断	极高（错误剪枝）

Contract 不是优化，而是“鱼控护栏”：
违反 Contract 的代码，哪怕跑得快，也不被允许进入主干。

---

6. Engineering Guidance（工程落点）

- **模块化**：8 段流水线必须可单测、可统计、可日志化；
- **EXACT 默认**：新模块先在 EXACT 模式下通过，再谈 RECALL；
- **DEFER 通道**：任何“不确定但可能有解”的候选必须有去处；
- **可逆调试**：Peel / Remove 必须可回放（日志或版本化）。

---

7. Roadmap & Evolution

- Phase-1：Toy Sequence / Graph 在 EXACT 模式下跑通（证明不漏）；
- Phase-2：引入 CCC-based LB 加强，提升 EXACT 性能；
- Phase-3：RECALL-Controlled 大规模部署，DEFER 回补闭环；
- Phase-4：跨域应用（生物结构、复杂图、长期时间序列）。

---

8. Closing Remark（鱼控宣言）

我们不是在“钓一条鱼”，也不是“撒一张网”。
我们做的是鱼控：
先控制生态、边界与演化方向，
再谈速度、规模与收益。

ITEM #229 作为 UnalignedAND-BTP 的**方向盘与骨架**，
其价值不在于“算得多快”，
而在于 **“永远知道自己有没有漏掉真正的鱼”**。

如果你愿意，下一步我可以直接帮你生成：

- **ITEM #229 的 Java Skeleton 包目录**（接口 + 空实现）
- 或 **一个 Sequence-only 的 EXACT Toy Demo + JUnit4**，
  用来把这套骨架第一次“落水试航”。

很好。请生成 ITEM #229 的 Java Skeleton 包目录（接口 + 空实现）

下面给出 **ITEM #229 — Java Skeleton 包目录（接口 + 空实现）**。
目标非常明确：**把“鱼控骨架”先固定成可编译、可扩展、不夹带策略假设的工程外壳。**
所有实现均为 *empty / stub*，但**接口边界、依赖方向、命名语义已经锁死**，后续任何算法优化都只能“往里填”，不能“改骨架”。

ITEM #229 — Java Skeleton Package Layout

UnalignedAND-BTP Fish-Control Backbone

Java 8 compatible
风格：`com.dbm.metric.btp.*`
依赖方向：**Pipeline → Contracts → Core Solver**（单向）

---

1. 总包结构（目录树）

```
com/dbm/metric/btp/
├── pipeline/
│   ├── FishControlPipeline.java
│   ├── FishControlMode.java
│   ├── PipelineContext.java
│   └── PipelineStats.java
│
├── piece/
│   ├── ReachEstimator.java
│   ├── SpanNeedEstimator.java
│   ├── Piece.java
│   ├── PieceWithHalo.java
│   └── PieceGenerator.java
│
├── anchor/
│   ├── Anchor.java
│   ├── AnchorSelector.java
│   ├── Occurrence.java
│   └── OccurrenceSearcher.java
│
├── gate/
│   ├── OccurrenceCccGate.java        // already defined
│   ├── GateDecision.java
│   └── GateStats.java
│
├── ccc/
│   ├── CccEvidence.java
│   ├── CccExtractor.java
│   └── CccSignature.java
│
├── filter/
│   ├── PieceLowerBoundEstimator.java
│   └── PieceFilter.java
│
├── solver/
│   ├── UnalignedAndBtpSolver.java
│   ├── BtpSolverState.java
│   ├── BtpBounds.java
│   └── BtpResult.java
│
├── match/
│   ├── Match.java
│   ├── MatchKey.java
│   ├── MatchConflictDetector.java
│   └── SafePeelDecider.java
│
├── peel/
│   ├── PeelAction.java
│   └── ResidualView.java
│
└── demo/
    └── EmptyFishControlDemo.java
```

---

2. Pipeline 层（方向盘）

FishControlMode.java

```
package com.dbm.metric.btp.pipeline;

public enum FishControlMode {
    EXACT,
    RECALL_CONTROLLED
}
```

PipelineContext.java

```
package com.dbm.metric.btp.pipeline;

public class PipelineContext {
    public final FishControlMode mode;

    public PipelineContext(FishControlMode mode) {
        this.mode = mode;
    }
}
```

FishControlPipeline.java

```
package com.dbm.metric.btp.pipeline;

public interface FishControlPipeline<A, B> {

    void run(A query, B target, PipelineContext context);

}
```

---

3. Piece / Reach 层（Coverage Guard）

ReachEstimator.java

```
package com.dbm.metric.btp.piece;

public interface ReachEstimator<A> {
    int estimateReach(A query);
}
```

SpanNeedEstimator.java

```
package com.dbm.metric.btp.piece;

public interface SpanNeedEstimator<A> {
    int estimateSpanNeed(A query);
}
```

Piece.java

```
package com.dbm.metric.btp.piece;

public interface Piece {
    boolean contains(Object nodeId);
}
```

PieceWithHalo.java

```
package com.dbm.metric.btp.piece;

public interface PieceWithHalo extends Piece {

    boolean isInCore(Object nodeId);

    boolean isInHalo(Object nodeId);

}
```

PieceGenerator.java

```
package com.dbm.metric.btp.piece;

import java.util.List;

public interface PieceGenerator<B> {

    List<PieceWithHalo> generatePieces(B target, int spanNeed, int reach);

}
```

---

4. Anchor / Occurrence 层（Candidate Guard）

Anchor.java

```
package com.dbm.metric.btp.anchor;

public interface Anchor {
}
```

Occurrence.java

```
package com.dbm.metric.btp.anchor;

public interface Occurrence {
}
```

AnchorSelector.java

```
package com.dbm.metric.btp.anchor;

import java.util.List;

public interface AnchorSelector<A> {

    List<Anchor> selectAnchors(A query);

}
```

OccurrenceSearcher.java

```
package com.dbm.metric.btp.anchor;

import java.util.List;

public interface OccurrenceSearcher<B> {

    List<Occurrence> search(B target, Anchor anchor);

}
```

---

5. CCC Gate 层（Filter Guard）

注意：此处只给“骨架接口”，实现已在前一轮给出。

GateDecision.java

```
package com.dbm.metric.btp.gate;

public enum GateDecision {
    ACCEPT,
    DEFER,
    REJECT
}
```

GateStats.java

```
package com.dbm.metric.btp.gate;

public interface GateStats {

    void onAccept();
    void onDefer();
    void onReject();

}
```

---

6. CCC 层（Evidence，不是决策）

`CccSignature.java`

```
package com.dbm.metric.btp.ccc;

public interface CccSignature {
    String asString();
}
```

`CccEvidence.java`

```
package com.dbm.metric.btp.ccc;

public interface CccEvidence {

    CccSignature signature();

    double supportScore();

    double ambiguity();

    double lowerBoundHint();

}
```

`CccExtractor.java`

```
package com.dbm.metric.btp.ccc;

public interface CccExtractor<A, B, Occurrence> {

    CccEvidence extract(A query, B target, Occurrence occurrence);

}
```

---

7. Piece-Level Filter（Cheap LB）

`PieceLowerBoundEstimator.java`

```
package com.dbm.metric.btp.filter;

import com.dbm.metric.btp.piece.PieceWithHalo;

public interface PieceLowerBoundEstimator<A> {

    double estimateLowerBound(A query, PieceWithHalo piece);

}
```

`PieceFilter.java`

```
package com.dbm.metric.btp.filter;

import com.dbm.metric.btp.piece.PieceWithHalo;

public interface PieceFilter {

    boolean accept(PieceWithHalo piece, double lowerBound, Double
topKThreshold);

}
```

---

8. BTP Solver 内核（唯一允许“重”的地方）

`BtpBounds.java`

```
package com.dbm.metric.btp.solver;

public interface BtpBounds {

    double lowerBound();

    double upperBound();

}
```

`BtpSolverState.java`

```
package com.dbm.metric.btp.solver;

public interface BtpSolverState {
}
```

}

BtpResult.java

```
package com.dbm.metric.btp.solver;

import com.dbm.metric.btp.match.Match;

public interface BtpResult {

    Match bestMatch();

    double distance();

}
```

UnalignedAndBtpSolver.java

```
package com.dbm.metric.btp.solver;

public interface UnalignedAndBtpSolver<A, B> {

    void initialize(A query, B target);

    boolean step();

    BtpBounds bounds();

    BtpResult currentBest();

}
```

---

9. Match / Conflict / Peel（组合安全）

Match.java

```
package com.dbm.metric.btp.match;

public interface Match {
}
```

MatchKey.java

```
package com.dbm.metric.btp.match;

public interface MatchKey {
    String asString();
}
```

MatchConflictDetector.java

```
package com.dbm.metric.btp.match;

public interface MatchConflictDetector {

    boolean conflict(Match a, Match b);

}
```

SafePeelDecider.java

```
package com.dbm.metric.btp.match;

public interface SafePeelDecider {

    boolean canPeel(Match match);

}
```

---

10. Peel / Residual（可逆）

ResidualView.java

```
package com.dbm.metric.btp.peel;

public interface ResidualView {

    void remove(Object nodeId);

}
```

PeelAction.java

```
package com.dbm.metric.btp.peel;

import com.dbm.metric.btp.match.Match;

public interface PeelAction {

    void apply(Match match, ResidualView view);

}
```

---

11. Demo（占位，不误导）

EmptyFishControlDemo.java

```
package com.dbm.metric.btp.demo;

public class EmptyFishControlDemo {

    public static void main(String[] args) {
        System.out.println("ITEM #229 Fish-Control Backbone skeleton 
loaded.");
    }

}
```

---

12. 设计宣言（非常重要）

- **这是“方向盘工程”，不是性能工程**
- 所有接口都是 *anti-leak guard* 的具象化
- 任何后续实现：
    - **只能填充实现**
    - **不得删除 Contract**
    - **不得绕开 EXACT 模式约束**

如果你愿意，下一步我可以直接帮你做两件非常“落地”的事之一：

1. **Sequence-only EXACT Toy Implementation**（steps / lags / window）
2. **JUnit4 验证套件**：专测 *边界*、*重复*、*低质量*、*冲突*、*剥离* 五类漏解高危点

你只需告诉我：**先从哪一类 Starmap 下水试航。**

请生成 JUnit4 验证套件：专测 边界、重复、低质量、冲突、剥离 五类漏解高危点

下面给出一套 **JUnit4 验证套件（toy but contract-faithful）**，专门覆盖你指出的五类“静默漏解高危点”：

- **边界（Piece/Halo）**

- **重复（Dedup / MatchKey）**
- **低质量（CCC Gate：EXACT 不得凭质量 REJECT）**
- **冲突（Overlap / Conflict Graph 的最小替代）**
- **剥离（SafePeel：halo-touch 不得 peel）**

设计原则：

1. **只测试 Contract 语义，不测试性能**；
2. 用一个极简的 Sequence-like toy starmap（整数序列 + 位置 occurrence + 匹配区间）模拟；
3. 所有 stub 实现写在测试类内部，避免引入大量依赖文件；
4. Java 8 + JUnit4 可直接编译运行。

文件 1：`UnalignedAndBtpFishControlLeakGuardsTest.java`

```
package com.dbm.metric.btp.test;

import org.junit.Test;
import org.junit.Assert;

import java.util.*;

/**
 * JUnit4 Verification Suite for ITEM #229 Fish-Control Backbone.
 *
 * Focus: "silent miss" high-risk areas:
 *  1) Boundary (PieceWithHalo coverage)
 *  2) Duplicate (MatchKey dedup)
 *  3) Low-quality occurrence (CCC gate exactness guard)
 *  4) Conflict (overlap) resolution
 *  5) Peeling (SafePeel vs halo-touch)
 *
 * This is a contract-focused toy test. It is not a benchmark.
 */
public class UnalignedAndBtpFishControlLeakGuardsTest {

    // -------------------------------
    // Toy domain model (Sequence-like)
    // -------------------------------

    static final class ToyQuery {
        final int[] a;     // query tokens
        final int reach;   // like maxLag
        ToyQuery(int[] a, int reach) { this.a = a; this.reach = reach; }
    }

    static final class ToyTarget {
        final int[] b;     // target tokens
        ToyTarget(int[] b) { this.b = b; }
        int size() { return b.length; }
    }

    /**
     * Piece with [coreStart, coreEnd] inclusive, and halo width around core.
     * Domain = [coreStart - halo, coreEnd + halo] clamped to target range.
     */
    static final class ToyPieceWithHalo {
        final int coreStart, coreEnd;
        final int haloWidth;
        final int domainStart, domainEnd;

        ToyPieceWithHalo(int coreStart, int coreEnd, int haloWidth, int
targetSize) {
            this.coreStart = coreStart;
            this.coreEnd = coreEnd;
            this.haloWidth = haloWidth;
            this.domainStart = Math.max(0, coreStart - haloWidth);
            this.domainEnd = Math.min(targetSize - 1, coreEnd + haloWidth);
        }

        boolean isInCore(int idx) { return idx >= coreStart && idx <=
coreEnd; }
        boolean isInDomain(int idx) { return idx >= domainStart && idx <=
domainEnd; }

        @Override public String toString() {
            return "Piece{core=[" + coreStart + "," + coreEnd + "], halo=" +
haloWidth +
                    ", domain=[" + domainStart + "," + domainEnd + "]}";
        }
    }

    static final class ToyOccurrence {
        final int pos; // anchor occurrence position in B
        ToyOccurrence(int pos) { this.pos = pos; }
    }

    /**
     * Toy match = interval mapping: query A aligned to B on [bStart, bEnd]
inclusive.
     * Distance = #mismatched tokens (toy).
     */
    static final class ToyMatch {
        final int bStart, bEnd;
        final int distance;
        ToyMatch(int bStart, int bEnd, int distance) {
            this.bStart = bStart; this.bEnd = bEnd; this.distance = distance;
        }
        boolean overlaps(ToyMatch other) {
            return !(this.bEnd < other.bStart || other.bEnd < this.bStart);
        }
        String key() { return "B[" + bStart + "," + bEnd + "]"; }
        @Override public String toString() { return key() + " d=" +
distance; }
    }

    /**
     * Residual view: tracks removed indices.
     */
    static final class ToyResidual {
        final boolean[] removed;
        ToyResidual(int n) { this.removed = new boolean[n]; }
        void removeRange(int s, int e) {
            for (int i = s; i <= e; i++) removed[i] = true;
        }
        boolean isRemoved(int i) { return removed[i]; }
    }

    // ------------------------------
    // Toy "BTP" matcher (contract-driven)
    // ------------------------------

    /**
     * Search for exact substring match of A in B within piece domain,
     * but it requires that the FULL match interval [bStart,bEnd] is within
piece domain.
     * This simulates "context completeness" dependence (simplified).
     */
    static final class ToyPieceMatcher {
        ToyMatch findBest(ToyQuery q, ToyTarget t, ToyPieceWithHalo piece) {
            int aLen = q.a.length;
            ToyMatch best = null;

            for (int start = piece.domainStart; start + aLen - 1 <=
piece.domainEnd; start++) {
                int end = start + aLen - 1;
                // Must be fully inside domain:
                // (already guaranteed by loop bounds)
                int dist = 0;
                for (int i = 0; i < aLen; i++) {
                    if (t.b[start + i] != q.a[i]) dist++;
                }
                if (best == null || dist < best.distance) best = new
ToyMatch(start, end, dist);
            }
            return best;
        }
    }

    // ------------------------------
    // CCC Gate (toy) with EXACT vs RECALL behavior
    // ------------------------------

    enum Mode { EXACT, RECALL_CONTROLLED }
    enum Decision { ACCEPT, DEFER, REJECT }
    static final class CccEvidence {
        final String sig;
        final double support;     // [0..1]
        final double ambiguity;   // [0..1]
        final double lbHint;      // safe LB hint (toy)
        final String reason;

        CccEvidence(String sig, double support, double ambiguity, double
lbHint, String reason) {
            this.sig = sig; this.support = support; this.ambiguity =
ambiguity; this.lbHint = lbHint; this.reason = reason;
        }
    }

    static final class GateResult {
        final Decision decision;
        final CccEvidence ev;
        GateResult(Decision decision, CccEvidence ev) { this.decision =
decision; this.ev = ev; }
    }

    interface CccExtractor {
        CccEvidence extract(ToyQuery q, ToyTarget t, ToyOccurrence occ);
    }

    interface CccDecisionPolicy {
        Decision decide(Mode mode, Double topKThreshold, CccEvidence ev,
Map<String,Object> params);
    }

    /**
     * Default policy (same semantics as discussed):
     * - EXACT: may only REJECT by lbHint (lbHint=INF or lbHint>threshold)
     * - RECALL: may REJECT by quality thresholds
     */
    static final class DefaultPolicy implements CccDecisionPolicy {
        @Override
        public Decision decide(Mode mode, Double topKThreshold, CccEvidence
ev, Map<String, Object> params) {
            if (Double.isInfinite(ev.lbHint) && ev.lbHint > 0) return
Decision.REJECT;

            if (mode == Mode.EXACT) {
                if (topKThreshold != null && ev.lbHint > topKThreshold)
return Decision.REJECT;
                return Decision.ACCEPT; // never reject by quality
            }

            double minSupport = getD(params, "recall.minSupport", 0.25);
            double maxAmb = getD(params, "recall.maxAmbiguity", 0.75);
            double deferSupport = getD(params, "recall.deferSupport", 0.35);
            double deferAmb = getD(params, "recall.deferAmbiguity", 0.90);

            if (ev.support < minSupport) return Decision.REJECT;
            if (ev.ambiguity > deferAmb) return Decision.REJECT;
            if (ev.support < deferSupport) return Decision.DEFER;
            if (ev.ambiguity > maxAmb) return Decision.DEFER;
            return Decision.ACCEPT;
        }

        private double getD(Map<String,Object> p, String k, double def) {
            if (p == null) return def;
            Object v = p.get(k);
            if (v == null) return def;
            if (v instanceof Number) return ((Number) v).doubleValue();
            try { return Double.parseDouble(String.valueOf(v)); } catch
(Exception e) { return def; }
        }
    }

    /**
     * Toy extractor:
     * - tier0: check anchor position token equals first token of query
(cheap)
     * - tier1: check two-token prefix match if possible
     * support:
     *  - 1.0 if prefix2 matches, 0.6 if only tier0 matches, 0.0 if mismatch
     * lbHint:
     *  - INF if tier0 mismatch AND anchor is declared must-match
(params.mustMatchTier0=true)
     *  - else 0
     */
    static final class ToyTier01Extractor implements CccExtractor {
        @Override
        public CccEvidence extract(ToyQuery q, ToyTarget t, ToyOccurrence
occ) {
            int pos = occ.pos;
            boolean tier0 = (pos >= 0 && pos < t.size() && t.b[pos] ==
q.a[0]);
            boolean tier1 = false;
            if (tier0 && q.a.length >= 2 && pos + 1 < t.size()) {
                tier1 = (t.b[pos] == q.a[0] && t.b[pos + 1] == q.a[1]);
            }
            double support = tier0 ? (tier1 ? 1.0 : 0.6) : 0.0;
            double ambiguity = tier1 ? 0.2 : (tier0 ? 0.5 : 0.95);

            // Must-match tier0 makes mismatch provably impossible for this
anchor.
            boolean mustMatchTier0 = true; // keep it true in this toy to
exercise INF path safely
            double lbHint = (!tier0 && mustMatchTier0) ?
Double.POSITIVE_INFINITY : 0.0;

            String sig = "occ@" + pos + "|t0=" + (tier0 ? 1 : 0) + "|t1=" +
(tier1 ? 1 : 0);
            String reason = !tier0 ? "tier0_mismatch" : (!tier1 ?
"tier1_mismatch" : "ok");
            return new CccEvidence(sig, support, ambiguity, lbHint, reason);
        }
    }

    static final class OccurrenceCccGate {
        final CccExtractor extractor;
        final CccDecisionPolicy policy;
        OccurrenceCccGate(CccExtractor extractor, CccDecisionPolicy policy) {
            this.extractor = extractor; this.policy = policy;
        }
        GateResult eval(Mode mode, Double topK, Map<String,Object> params,
ToyQuery q, ToyTarget t, ToyOccurrence occ) {
            CccEvidence ev = extractor.extract(q, t, occ);
            Decision d = policy.decide(mode, topK, ev, params);
            return new GateResult(d, ev);
        }
    }

    // ------------------------------
    // Helpers: dedup, conflict resolution, safe peel
    // ------------------------------

    static final class MatchDeduper {
        Map<String, ToyMatch> dedup(List<ToyMatch> matches) {
            Map<String, ToyMatch> bestByKey = new LinkedHashMap<String,
ToyMatch>();
            for (ToyMatch m : matches) {
                ToyMatch prev = bestByKey.get(m.key());
                if (prev == null || m.distance < prev.distance)
bestByKey.put(m.key(), m);
            }
            return bestByKey;
        }
    }

    static final class ConflictResolver {
        /**
         * Minimal "max compatible set" heuristic for tests:
         * - sort by distance ascending
         * - greedily take non-overlapping
         */
        List<ToyMatch> selectNonOverlappingBest(List<ToyMatch> matches) {
            List<ToyMatch> list = new ArrayList<ToyMatch>(matches);
            Collections.sort(list, new Comparator<ToyMatch>() {
                @Override public int compare(ToyMatch o1, ToyMatch o2) {
                    int c = Integer.compare(o1.distance, o2.distance);
                    if (c != 0) return c;
                    return Integer.compare(o1.bStart, o2.bStart);
                }
            });
            List<ToyMatch> chosen = new ArrayList<ToyMatch>();
            for (ToyMatch m : list) {
                boolean ok = true;
                for (ToyMatch c : chosen) {
                    if (m.overlaps(c)) { ok = false; break; }
                }
                if (ok) chosen.add(m);
            }
            return chosen;
        }
    }

    static final class SafePeel {
        boolean canPeel(ToyMatch m, ToyPieceWithHalo piece) {
            // Only peel if the entire match interval lies in core (not
touching halo).
            for (int i = m.bStart; i <= m.bEnd; i++) {
                if (!piece.isInCore(i)) return false;
            }
            return true;
        }
        void peel(ToyMatch m, ToyResidual residual) {
            residual.removeRange(m.bStart, m.bEnd);
        }
    }

    // ============================================================
    // 1) Boundary Risk Test: Halo must prevent border silent miss
    // ============================================================
    @Test
    public void testBoundary_guardedByHaloWidth() {
        ToyQuery q = new ToyQuery(new int[]{7,8,9,10}, /*reach*/ 1);
        ToyTarget t = new ToyTarget(new int[]{0,1,2,3,4,5,7,8,9,10,11,12});

        // Two pieces: core [0..5], core [6..11]
        // Correct halo (>=reach): piece2 domain includes enough room.
        ToyPieceWithHalo piece2_ok = new ToyPieceWithHalo(6, 11, 1,
t.size());
        ToyPieceWithHalo piece2_bad = new ToyPieceWithHalo(6, 11, 0,
t.size()); // no halo

        ToyPieceMatcher matcher = new ToyPieceMatcher();

        ToyMatch ok = matcher.findBest(q, t, piece2_ok);
        Assert.assertNotNull("With sufficient halo, match should be found",
ok);
        Assert.assertEquals("Match should align at [6..9]", 6, ok.bStart);
        Assert.assertEquals(9, ok.bEnd);
        Assert.assertEquals(0, ok.distance);

        // Simulate boundary failure: if the algorithm had needed to consider
a start that is only available with halo,
        // then halo=0 could miss. Here we enforce a stricter check: require
that match interval must be in piece domain.
        // With halo=0, [6..9] is still in domain, so to force a boundary-
sensitive case we move the true match to start=5.
        ToyTarget t2 = new ToyTarget(new int[]{0,1,2,3,4,7,8,9,10,11,12,13});
// true match at [5..8]
        ToyMatch ok2 = matcher.findBest(q, t2, piece2_ok);
        Assert.assertNotNull(ok2);
        Assert.assertEquals(5, ok2.bStart);

        ToyMatch bad2 = matcher.findBest(q, t2, piece2_bad);
        // With halo=0, piece2_bad domain is [6..11], so it cannot even
consider start=5 (silent miss risk)
        Assert.assertNotNull("Still returns a best in domain, but will NOT be
the true global best", bad2);
        Assert.assertNotEquals("Halo=0 misses the true best starting at 5",
5, bad2.bStart);
    }
    // ============================================================
    // 2) Duplicate Risk Test: same match from multiple sources must dedup
deterministically
    // ============================================================
    @Test
    public void testDuplicate_dedupByMatchKeyKeepsBestDistance() {
        List<ToyMatch> raw = new ArrayList<ToyMatch>();
        raw.add(new ToyMatch(10, 13, 2)); // same key, worse
        raw.add(new ToyMatch(10, 13, 0)); // same key, best
        raw.add(new ToyMatch(20, 23, 1));

        MatchDeduper d = new MatchDeduper();
        Map<String, ToyMatch> best = d.dedup(raw);

        Assert.assertEquals(2, best.size());
        Assert.assertEquals(0, best.get("B[10,13]").distance);
        Assert.assertEquals(1, best.get("B[20,23]").distance);
    }

    // ============================================================
    // 3) Low-quality Risk Test: EXACT mode must not REJECT by quality (only
by LB)
    // ============================================================
    @Test
    public void testLowQuality_exactModeMustNotRejectBySupportOrAmbiguity() {
        ToyQuery q = new ToyQuery(new int[]{7,8,9,10}, 1);
        ToyTarget t = new ToyTarget(new int[]{0,7,1,2,3,4,5,6}); //
occurrence at pos=1 matches tier0 only, tier1 likely mismatch
        ToyOccurrence occ = new ToyOccurrence(1);

        OccurrenceCccGate gate = new OccurrenceCccGate(new
ToyTier01Extractor(), new DefaultPolicy());

        // EXACT with unknown threshold: should ACCEPT (never reject by low
support)
        GateResult r1 = gate.eval(Mode.EXACT, null,
Collections.<String,Object>emptyMap(), q, t, occ);
        Assert.assertEquals("EXACT mode must not reject by quality",
Decision.ACCEPT, r1.decision);

        // RECALL: with strict thresholds, should DEFER/REJECT depending on
support/ambiguity
        Map<String,Object> params = new HashMap<String, Object>();
        params.put("recall.minSupport", 0.8); // make it strict
        GateResult r2 = gate.eval(Mode.RECALL_CONTROLLED, null, params, q, t,
occ);
        Assert.assertTrue("RECALL mode may reject/defer low-quality",
                r2.decision == Decision.REJECT || r2.decision ==
Decision.DEFER);
    }

    // ============================================================
    // 4) Conflict Risk Test: overlapping matches must not both be accepted
    // ============================================================
    @Test
    public void testConflict_overlapResolutionPicksNonOverlappingSet() {
        List<ToyMatch> matches = Arrays.asList(
                new ToyMatch(0, 3, 2),
                new ToyMatch(2, 5, 0),  // overlaps with [0,3]
                new ToyMatch(6, 9, 1)   // non-overlap
        );

        ConflictResolver resolver = new ConflictResolver();
        List<ToyMatch> chosen = resolver.selectNonOverlappingBest(matches);

        // The best is [2,5] (d=0) and [6,9] (d=1). Should not include [0,3].
        Assert.assertEquals(2, chosen.size());
        Assert.assertEquals("B[2,5]", chosen.get(0).key());
        Assert.assertEquals("B[6,9]", chosen.get(1).key());
        Assert.assertFalse(chosen.get(0).overlaps(chosen.get(1)));
    }

    // ============================================================
    // 5) Peel Risk Test: matches touching halo must not be peeled (avoid border leak)
    // ============================================================
    @Test
    public void testPeel_mustRejectHaloTouchingMatch() {
        ToyTarget t = new ToyTarget(new int[]{0,1,2,3,4,5,6,7,8,9});
        ToyPieceWithHalo piece = new ToyPieceWithHalo(3, 7, 1, t.size()); // core [3..7], halo expands to [2..8]

        ToyResidual residual = new ToyResidual(t.size());
        SafePeel peel = new SafePeel();

        // Match fully inside core => can peel
        ToyMatch mCore = new ToyMatch(4, 6, 0);
        Assert.assertTrue(peel.canPeel(mCore, piece));
        peel.peel(mCore, residual);
        Assert.assertTrue(residual.isRemoved(4));
        Assert.assertTrue(residual.isRemoved(5));
        Assert.assertTrue(residual.isRemoved(6));

        // Match touches halo (index 2 is halo, not core) => must NOT peel
        ToyResidual residual2 = new ToyResidual(t.size());
        ToyMatch mHaloTouch = new ToyMatch(2, 4, 0); // includes 2 (halo)
        Assert.assertFalse("Halo-touching match must not be peeled", peel.canPeel(mHaloTouch, piece));
        // Ensure no removal occurs
        Assert.assertFalse(residual2.isRemoved(2));
        Assert.assertFalse(residual2.isRemoved(3));
        Assert.assertFalse(residual2.isRemoved(4));
    }
}
```

你将得到什么（与五类风险逐一对齐）

1. **边界**：`testBoundary_guardedByHaloWidth`

    - 演示 halo 不足时，“真全局最优对齐起点在边界外”会被 piece-domain 静默排除；有 halo 时可覆盖。
    - 这就是 Contract C1 的最低可测形式。
2. **重复**：`testDuplicate_dedupByMatchKeyKeepsBestDistance`
    - 同 key 多来源，只保留距离最优者，避免 Top-K/迭代时重复污染。
3. **低质量**：`testLowQuality_exactModeMustNotRejectBySupportOrAmbiguity`
    - EXACT 模式下，不允许凭 support/ambiguity REJECT（只可 DEFER/ACCEPT），防止“高危漏解”。
    - RECALL 模式下允许收紧阈值。
4. **冲突**：`testConflict_overlapResolutionPicksNonOverlappingSet`
    - 两个 overlap match 不可同时接受，必须显式 conflict-resolution。
5. **剥离**：`testPeel_mustRejectHaloTouchingMatch`
    - 触及 halo 的 match 禁止 peel，防止“跨片段真解被早期剥离破坏”。

集成建议（最小摩擦）

- 先把这个 test 文件放到你工程的 `src/test/java/com/dbm/metric/btp/test/`。
- 确保 `JUnit4` 依赖存在（你项目中通常已经有）。
- 这套测试是 **骨架护栏**：后续你把真实 `PieceGenerator / OccurrenceCccGate / Solver / Peel` 接进来时，可以用这些 test 作为“语义回归”模板，把 toy stub 替换为真实实现的 adapter。

如果你下一步希望我把该套测试改造成：

- **直接调用你刚刚生成的 skeleton 接口**（而不是 test 内部 toy 类），
  我也可以给出一版“与 com.dbm.metric.btp.* 接口完全对齐”的测试适配版。

请该套测试改造成： 直接调用你刚刚生成的 skeleton 接口（而不是 test 内部 toy 类）

下面是一版**“直接调用 skeleton 接口”的 JUnit4 验证套件**。
关键变化：测试不再使用“自定义 toy 类体系”当作主接口，而是**用你刚刚固定的骨架接口**（`com.dbm.metric.btp.*`）来驱动：`PieceWithHalo / PieceGenerator / AnchorSelector / OccurrenceSearcher / OccurrenceCccGate / MatchKey / MatchConflictDetector / SafePeelDecider / PeelAction / ResidualView` 等。

说明（很重要）：

- 测试仍然需要 **toy 的空实现**（否则没法运行），但这些 toy 实现**全部实现 skeleton 接口**，并且测试调用也通过接口完成。
- 该文件假设你已把此前生成的以下源码加入工程：
  - skeleton 接口包（`com.dbm.metric.btp.*`）
  - `com.dbm.metric.btp.gate.OccurrenceCccGate`（完整版接口，含 `GateInput/GateResult/CccEvidence/CccExtractor/CccDecisionPolicy/Default`）
  - `com.dbm.metric.btp.gate.CccGateDefaults`（含 `DefaultCccDecisionPolicy` 和 `ToyTier01CccExtractor`）
    若你当前 skeleton 里只保留了“简化版 GateDecision/GateStats”而没放完整版 `OccurrenceCccGate`，请以此前我给你的完整版为准，否则测试无法直接编译。

文件：

`src/test/java/com/dbm/metric/btp/test/Item227LeakGuardsOnSkeletonTest.java`

```
package com.dbm.metric.btp.test;

import com.dbm.metric.btp.anchor.Anchor;
import com.dbm.metric.btp.anchor.AnchorSelector;
import com.dbm.metric.btp.anchor.Occurrence;
import com.dbm.metric.btp.anchor.OccurrenceSearcher;
import com.dbm.metric.btp.filter.PieceFilter;
import com.dbm.metric.btp.filter.PieceLowerBoundEstimator;
import com.dbm.metric.btp.gate.CccGateDefaults;
import com.dbm.metric.btp.gate.OccurrenceCccGate;
import com.dbm.metric.btp.match.Match;
import com.dbm.metric.btp.match.MatchConflictDetector;
import com.dbm.metric.btp.match.MatchKey;
import com.dbm.metric.btp.match.SafePeelDecider;
import com.dbm.metric.btp.peel.PeelAction;
import com.dbm.metric.btp.peel.ResidualView;
import com.dbm.metric.btp.piece.PieceGenerator;
import com.dbm.metric.btp.piece.PieceWithHalo;
import com.dbm.metric.btp.piece.ReachEstimator;
import com.dbm.metric.btp.piece.SpanNeedEstimator;

import org.junit.Assert;
import org.junit.Test;

import java.util.*;

/**
 * ITEM #229 - Leak Guards Verification Suite (JUnit4)
 * Version: Calling ONLY skeleton interfaces (toy impls are interface
implementations).
 *
 * Covers 5 silent-miss high-risk areas:
 *  1) Boundary (PieceWithHalo + halo reach)
 *  2) Duplicate (MatchKey dedup)
 *  3) Low-quality occurrences (OccurrenceCccGate EXACT vs RECALL)
 *  4) Conflict (overlap conflict detection/resolution)
 *  5) Peeling (SafePeelDecider forbids halo-touch peel)
 */
public class Item227LeakGuardsOnSkeletonTest {

    // =========================
    // Toy domain data structures
    // =========================

    static final class ToyQuery {
        final int[] a;
        final int reach; // lag-like
        ToyQuery(int[] a, int reach) { this.a = a; this.reach = reach; }
    }

    static final class ToyTarget {
        final int[] b;
        ToyTarget(int[] b) { this.b = b; }
        int size() { return b.length; }
    }

    // ============
    // Skeleton impl
    // ============

    // -------- PieceWithHalo --------

    static final class ToyPiece implements PieceWithHalo {
        final int coreStart, coreEnd;
        final int haloWidth;
        final int domainStart, domainEnd;

        ToyPiece(int coreStart, int coreEnd, int haloWidth, int targetSize) {
            this.coreStart = coreStart;
            this.coreEnd = coreEnd;
            this.haloWidth = haloWidth;
            this.domainStart = Math.max(0, coreStart - haloWidth);
            this.domainEnd = Math.min(targetSize - 1, coreEnd + haloWidth);
        }

        @Override
        public boolean contains(Object nodeId) {
            int idx = (Integer) nodeId;
            return idx >= domainStart && idx <= domainEnd;
        }

        @Override
        public boolean isInCore(Object nodeId) {
            int idx = (Integer) nodeId;
            return idx >= coreStart && idx <= coreEnd;
        }

        @Override
        public boolean isInHalo(Object nodeId) {
            int idx = (Integer) nodeId;
            // halo is "domain but not core"
            return contains(nodeId) && !isInCore(nodeId);
        }

        @Override
        public String toString() {
            return "ToyPiece{core=[" + coreStart + "," + coreEnd + "], halo="
+ haloWidth +
                    ", domain=[" + domainStart + "," + domainEnd + "]}";
        }
    }

    static final class ToyReachEstimator implements ReachEstimator<ToyQuery>
{
        @Override public int estimateReach(ToyQuery query) { return
query.reach; }
    }

    static final class ToySpanNeedEstimator implements
SpanNeedEstimator<ToyQuery> {
        @Override public int estimateSpanNeed(ToyQuery query) {
            // toy: need len(A) + 2*reach
            return query.a.length + 2 * query.reach;
        }
    }

    static final class ToyPieceGenerator implements PieceGenerator<ToyTarget>
{
        private final int coreSize;
        private final boolean useHalo;

        ToyPieceGenerator(int coreSize, boolean useHalo) {
            this.coreSize = coreSize;
            this.useHalo = useHalo;
        }

        @Override
        public List<PieceWithHalo> generatePieces(ToyTarget target, int
spanNeed, int reach) {
            // toy: two pieces dividing target
            int n = target.size();
            int halo = useHalo ? reach : 0;

            int mid = Math.min(n - 1, coreSize - 1);
            ToyPiece p1 = new ToyPiece(0, mid, halo, n);
            ToyPiece p2 = new ToyPiece(mid + 1, n - 1, halo, n);

            return Arrays.<PieceWithHalo>asList(p1, p2);
        }
    }

    // -------- Anchor / Occurrence --------

    static final class ToyAnchor implements Anchor {
        final int token;
        ToyAnchor(int token) { this.token = token; }
    }

    static final class ToyOccurrence implements Occurrence {
        final int pos;
        ToyOccurrence(int pos) { this.pos = pos; }
    }

    static final class ToyAnchorSelector implements AnchorSelector<ToyQuery>
{
        @Override public List<Anchor> selectAnchors(ToyQuery query) {
            // toy: anchor on first token of query
            return Collections.<Anchor>singletonList(new
ToyAnchor(query.a[0]));
        }
    }

    static final class ToyOccurrenceSearcher implements
OccurrenceSearcher<ToyTarget> {
        @Override public List<Occurrence> search(ToyTarget target, Anchor
anchor) {
            ToyAnchor a = (ToyAnchor) anchor;
            List<Occurrence> out = new ArrayList<Occurrence>();
            for (int i = 0; i < target.size(); i++) {
                if (target.b[i] == a.token) out.add(new ToyOccurrence(i));
            }
            return out;
        }
    }

    // -------- Piece-level LB filter (toy) --------

    static final class ToyPieceLowerBoundEstimator implements
PieceLowerBoundEstimator<ToyQuery> {
        @Override public double estimateLowerBound(ToyQuery query,
PieceWithHalo piece) {
            // toy safe LB: 0 if piece has enough domain length to fit query
else +INF
```

```
            // We approximate "domain length" by counting indices that 
piece.contains(i).
            int count = 0;
            // scan a bounded range; in real impl you'd have domain 
boundaries.
            // Here we assume target size <= 10^5 for tests; we only need 
small tests anyway.
            for (int i = 0; i < 1000; i++) {
                if (piece.contains(i)) count++;
            }
            return count >= query.a.length ? 0.0 : Double.POSITIVE_INFINITY;
        }
    }

    static final class ToyPieceFilter implements PieceFilter {
        @Override public boolean accept(PieceWithHalo piece, double 
lowerBound, Double topKThreshold) {
            if (Double.isInfinite(lowerBound) && lowerBound > 0) return 
false;
            if (topKThreshold != null && lowerBound > topKThreshold) return 
false;
            return true;
        }
    }

    // -------- Match / MatchKey / conflict / peel --------

    static final class ToyMatch implements Match {
        final int bStart, bEnd;
        final int distance;
        ToyMatch(int bStart, int bEnd, int distance) {
            this.bStart = bStart; this.bEnd = bEnd; this.distance = distance;
        }
        boolean overlaps(ToyMatch other) {
            return !(this.bEnd < other.bStart || other.bEnd < this.bStart);
        }
        @Override public String toString() { return "B[" + bStart + "," + 
bEnd + "] d=" + distance; }
    }

    static final class ToyMatchKey implements MatchKey {
        final String s;
        ToyMatchKey(String s) { this.s = s; }
        @Override public String asString() { return s; }
    }

    static final class ToyConflictDetector implements MatchConflictDetector {
        @Override public boolean conflict(Match a, Match b) {
            ToyMatch x = (ToyMatch) a;
            ToyMatch y = (ToyMatch) b;
            return x.overlaps(y);
        }
    }

    static final class ToySafePeelDecider implements SafePeelDecider {
        private final PieceWithHalo piece;
        ToySafePeelDecider(PieceWithHalo piece) { this.piece = piece; }

        @Override
        public boolean canPeel(Match match) {
            ToyMatch m = (ToyMatch) match;
            for (int i = m.bStart; i <= m.bEnd; i++) {
                if (!piece.isInCore(i)) return false; // halo-touch forbidden
            }
            return true;
        }
    }

    static final class ToyResidualView implements ResidualView {
        final boolean[] removed;
        ToyResidualView(int n) { this.removed = new boolean[n]; }

        @Override public void remove(Object nodeId) {
            int idx = (Integer) nodeId;
            removed[idx] = true;
        }

        void removeRange(int s, int e) {
            for (int i = s; i <= e; i++) remove(i);
        }

        boolean isRemoved(int i) { return removed[i]; }
    }

    static final class ToyPeelAction implements PeelAction {
        @Override public void apply(Match match, ResidualView view) {
            ToyMatch m = (ToyMatch) match;
            ToyResidualView rv = (ToyResidualView) view;
            rv.removeRange(m.bStart, m.bEnd);
        }
    }

    // ----------------------------
    // Helpers for tests (dedup etc)
    // ----------------------------

    static Map<String, ToyMatch> dedupByKey(List<ToyMatch> matches) {
        Map<String, ToyMatch> best = new LinkedHashMap<String, ToyMatch>();
        for (ToyMatch m : matches) {
            String key = new ToyMatchKey("B[" + m.bStart + "," + m.bEnd +
"]").asString();
            ToyMatch prev = best.get(key);
            if (prev == null || m.distance < prev.distance) best.put(key, m);
        }
        return best;
    }

    static List<ToyMatch> greedyNonOverlapping(List<ToyMatch> matches,
MatchConflictDetector conflict) {
        List<ToyMatch> list = new ArrayList<ToyMatch>(matches);
        Collections.sort(list, new Comparator<ToyMatch>() {
            @Override public int compare(ToyMatch o1, ToyMatch o2) {
                int c = Integer.compare(o1.distance, o2.distance);
                if (c != 0) return c;
                return Integer.compare(o1.bStart, o2.bStart);
            }
        });
        List<ToyMatch> chosen = new ArrayList<ToyMatch>();
        for (ToyMatch m : list) {
            boolean ok = true;
            for (ToyMatch c : chosen) {
                if (conflict.conflict(m, c)) { ok = false; break; }
            }
            if (ok) chosen.add(m);
        }
        return chosen;
    }

    /**
     * Toy matcher that searches within a piece's DOMAIN only.
     * We keep it as a helper here; in the real system this is UnalignedAND-
BTP on a piece-view.
     */
    static ToyMatch findBestInPieceDomain(ToyQuery q, ToyTarget t, ToyPiece
piece) {
        int aLen = q.a.length;
        ToyMatch best = null;
        for (int start = piece.domainStart; start + aLen - 1 <=
piece.domainEnd; start++) {
            int end = start + aLen - 1;
            int dist = 0;
            for (int i = 0; i < aLen; i++) {
                if (t.b[start + i] != q.a[i]) dist++;
            }
            if (best == null || dist < best.distance) best = new
ToyMatch(start, end, dist);
        }
        return best;
    }

    // ============================================================
    // 1) Boundary: halo too small => silent miss of global best
    // ============================================================
    @Test
    public void testBoundary_pieceWithHaloGuardsAgainstBorderMiss() {
        ToyQuery q = new ToyQuery(new int[]{7,8,9,10}, /*reach*/ 1);

        // target has true perfect match starting at 5 (boundary between two
pieces)
        ToyTarget t = new ToyTarget(new int[]{0,1,2,3,4,7,8,9,10,11,12,13});
// [5..8] is perfect

        ReachEstimator<ToyQuery> reachE = new ToyReachEstimator();
        SpanNeedEstimator<ToyQuery> spanE = new ToySpanNeedEstimator();
        int reach = reachE.estimateReach(q);
        int spanNeed = spanE.estimateSpanNeed(q);

        // With halo
        PieceGenerator<ToyTarget> genOk = new
ToyPieceGenerator(/*coreSize*/5, /*useHalo*/true);
```

```
        List<PieceWithHalo> piecesOk = genOk.generatePieces(t, spanNeed,
reach);

        // Without halo
        PieceGenerator<ToyTarget> genBad = new
ToyPieceGenerator(/*coreSize*/5, /*useHalo*/false);
        List<PieceWithHalo> piecesBad = genBad.generatePieces(t, spanNeed,
reach);

        // Find best match by scanning pieces (toy)
        ToyMatch bestOk = null;
        for (PieceWithHalo p : piecesOk) {
            ToyPiece tp = (ToyPiece) p;
            ToyMatch m = findBestInPieceDomain(q, t, tp);
            if (bestOk == null || m.distance < bestOk.distance) bestOk = m;
        }
        Assert.assertNotNull(bestOk);
        Assert.assertEquals("With halo, should be able to consider start=5",
5, bestOk.bStart);
        Assert.assertEquals(0, bestOk.distance);

        ToyMatch bestBad = null;
        for (PieceWithHalo p : piecesBad) {
            ToyPiece tp = (ToyPiece) p;
            ToyMatch m = findBestInPieceDomain(q, t, tp);
            if (bestBad == null || m.distance < bestBad.distance) bestBad =
m;
        }
        Assert.assertNotNull(bestBad);
        Assert.assertNotEquals("Without halo, start=5 is outside piece2
domain => silent miss", 5, bestBad.bStart);
        Assert.assertTrue("Without halo, best distance should be worse than
perfect", bestBad.distance > 0);
    }

    // =================================================================
    // 2) Duplicate: same mapping from multiple sources must dedup by
MatchKey
    // =================================================================
    @Test
    public void testDuplicate_dedupByMatchKeyKeepsBest() {
        List<ToyMatch> raw = Arrays.asList(
                new ToyMatch(10, 13, 2),
                new ToyMatch(10, 13, 0),
                new ToyMatch(20, 23, 1)
        );
        Map<String, ToyMatch> best = dedupByKey(raw);
        Assert.assertEquals(2, best.size());
        Assert.assertEquals(0, best.get("B[10,13]").distance);
        Assert.assertEquals(1, best.get("B[20,23]").distance);
    }

    // =================================================================
    // 3) Low-quality occurrences: EXACT must not REJECT by quality-only
    // =================================================================
    @Test
    public void testLowQuality_exactModeNoRejectBySupportAmbiguity() {
        ToyQuery q = new ToyQuery(new int[]{7,8,9,10}, 1);
        ToyTarget t = new ToyTarget(new int[]{0,7,1,2,3,4,5,6}); //
occurrence at pos=1: tier0 ok, tier1 mismatch likely

        AnchorSelector<ToyQuery> anchorSelector = new ToyAnchorSelector();
        OccurrenceSearcher<ToyTarget> occSearcher = new
ToyOccurrenceSearcher();

        Anchor anchor = anchorSelector.selectAnchors(q).get(0);
        Occurrence occ = occSearcher.search(t, anchor).get(0);

        // Gate = extractor + default policy (from CccGateDefaults)
        CccGateDefaults.DefaultCccDecisionPolicy policy = new
CccGateDefaults.DefaultCccDecisionPolicy();

        // Toy extractor requires adapter; provide minimal adapter for our
toy types.
        CccGateDefaults.ToyTier01CccExtractor.ToyAdapter<ToyQuery, ToyTarget,
Anchor, Occurrence> adapter =
                new
CccGateDefaults.ToyTier01CccExtractor.ToyAdapter<ToyQuery, ToyTarget, Anchor,
Occurrence>() {
                    @Override public String tier0TokenForQuery(ToyQuery
query, Anchor a) {
                        return String.valueOf(query.a[0]);
                    }
                    @Override public String tier1TokenForQuery(ToyQuery
query, Anchor a) {
                        if (query.a.length < 2) return "NA";
                        return query.a[0] + "," + query.a[1];
                    }
                    @Override public String tier0TokenForOccurrence(ToyTarget
target, Occurrence o, Anchor a) {
                        int pos = ((ToyOccurrence) o).pos;
                        return String.valueOf(target.b[pos]);
                    }
                    @Override public String tier1TokenForOccurrence(ToyTarget
target, Occurrence o, Anchor a) {
                        int pos = ((ToyOccurrence) o).pos;
                        if (pos + 1 >= target.size()) return "NA";
                        return target.b[pos] + "," + target.b[pos + 1];
                    }
                    @Override public Integer
occurrenceFrequencyHint(ToyTarget target, Occurrence o, Anchor a) {
                        // toy: frequency of tier0 token
                        String tok = tier0TokenForOccurrence(target, o, a);
                        int v = Integer.parseInt(tok);
                        int cnt = 0;
                        for (int x : target.b) if (x == v) cnt++;
                        return cnt;
                    }
                    @Override public Double
safeLowerBoundHint(OccurrenceCccGate.GateInput<ToyQuery, ToyTarget, Anchor,
Occurrence> in,
                                                               String
tier0OccToken, String tier1OccToken) {
```

```
                        // toy: no strong LB proof => return 0 (so EXACT
won't reject)
                        return 0.0;
                    }
                };

        OccurrenceCccGate.CccExtractor<ToyQuery, ToyTarget, Anchor,
Occurrence> extractor =
                new CccGateDefaults.ToyTier01CccExtractor<ToyQuery,
ToyTarget, Anchor, Occurrence>(adapter);

        OccurrenceCccGate<ToyQuery, ToyTarget, Anchor, Occurrence> gate =
                new OccurrenceCccGate.Default<ToyQuery, ToyTarget, Anchor,
Occurrence>(extractor, policy);

        OccurrenceCccGate.GateInput<ToyQuery, ToyTarget, Anchor, Occurrence>
inExact =
                new OccurrenceCccGate.GateInput<ToyQuery, ToyTarget, Anchor,
Occurrence>(
                        q, t, anchor, occ, /*reach*/1, /*topK*/null,
                        OccurrenceCccGate.Mode.EXACT,
Collections.<String,Object>emptyMap());

        OccurrenceCccGate.GateResult rExact = gate.evaluate(inExact);
        Assert.assertEquals("EXACT must not reject by low quality alone",
                OccurrenceCccGate.Decision.ACCEPT, rExact.decision);

        Map<String,Object> strictRecall = new HashMap<String, Object>();
        strictRecall.put("recall.minSupport", 0.8);

        OccurrenceCccGate.GateInput<ToyQuery, ToyTarget, Anchor, Occurrence>
inRecall =
                new OccurrenceCccGate.GateInput<ToyQuery, ToyTarget, Anchor,
Occurrence>(
                        q, t, anchor, occ, /*reach*/1, /*topK*/null,
                        OccurrenceCccGate.Mode.RECALL_CONTROLLED,
strictRecall);

        OccurrenceCccGate.GateResult rRecall = gate.evaluate(inRecall);
        Assert.assertTrue("RECALL may reject/defer low-quality occurrences",
                rRecall.decision == OccurrenceCccGate.Decision.REJECT ||
                        rRecall.decision ==
OccurrenceCccGate.Decision.DEFER);
    }

    // ============================================================
    // 4) Conflict: overlapping matches must not both survive selection
    // ============================================================
    @Test
    public void testConflict_overlapMustBeResolved() {
        List<ToyMatch> matches = Arrays.asList(
                new ToyMatch(0, 3, 2),
                new ToyMatch(2, 5, 0),  // overlaps with [0,3]
                new ToyMatch(6, 9, 1)   // non-overlap
        );

        MatchConflictDetector conflict = new ToyConflictDetector();
        List<ToyMatch> chosen = greedyNonOverlapping(matches, conflict);

        Assert.assertEquals(2, chosen.size());
        Assert.assertEquals("B[2,5]", new ToyMatchKey("B[" +
chosen.get(0).bStart + "," + chosen.get(0).bEnd + "]").asString());
        Assert.assertEquals("B[6,9]", new ToyMatchKey("B[" +
chosen.get(1).bStart + "," + chosen.get(1).bEnd + "]").asString());
        Assert.assertFalse(conflict.conflict(chosen.get(0), chosen.get(1)));
    }

    // ============================================================
    // 5) Peel: halo-touch matches must not be peeled; core-only can peel
    // ============================================================
    @Test
    public void testPeel_safePeelDeciderForbidsHaloTouch() {
        ToyTarget t = new ToyTarget(new int[]{0,1,2,3,4,5,6,7,8,9});
        // core [3..7], halo=1 => domain [2..8]
        PieceWithHalo piece = new ToyPiece(3, 7, 1, t.size());

        SafePeelDecider decider = new ToySafePeelDecider(piece);
        PeelAction peel = new ToyPeelAction();

        ToyResidualView residual = new ToyResidualView(t.size());

        // core-only match => peel ok
        Match mCore = new ToyMatch(4, 6, 0);
        Assert.assertTrue(decider.canPeel(mCore));
        peel.apply(mCore, residual);
        Assert.assertTrue(residual.isRemoved(4));
        Assert.assertTrue(residual.isRemoved(5));
        Assert.assertTrue(residual.isRemoved(6));

        // halo-touch match includes index 2 (halo) => must not peel
        ResidualView residual2 = new ToyResidualView(t.size());
        Match mHalo = new ToyMatch(2, 4, 0);
        Assert.assertFalse("Halo-touch match must not be peelable",
decider.canPeel(mHalo));

        // ensure no side effects (since we didn't peel)
        ToyResidualView rv2 = (ToyResidualView) residual2;
        Assert.assertFalse(rv2.isRemoved(2));
        Assert.assertFalse(rv2.isRemoved(3));
        Assert.assertFalse(rv2.isRemoved(4));
    }
}
```

你现在得到的效果

- 这套测试**完全围绕 skeleton 接口**来组织（toy 仅作“实现体”）。
- 当你把真实实现替换 toy 实现时，这些测试可以直接变为**回归护栏**：

    - halo 逻辑、dedup key、EXACT gate 行为、conflict 选择、safe peel 都会被锁死。
- 它对“漏解风险”非常敏感：任何绕开 Contract 的优化，都很容易在这五类测试上暴露。

若你遇到编译问题（最可能的两个点）

1. 你当前工程里是否已经包含 **完整版** `OccurrenceCccGate`（带 `GateInput/GateResult/CccEvidence/CccExtractor/CccDecisionPolicy/Default`）和 `CccGateDefaults`？
    - 如果没有，我建议你以此前我提供的那两个源码文件为准直接放入 `com.dbm.metric.btp.gate`。
2. 你的 skeleton `PieceWithHalo` 是否与我测试里一致（`isInCore/isInHalo/contains` 的签名）？
    - 若你已做过小改名，测试里把方法名改一下即可。

如果你希望我下一步再进一步：把这套测试改为 **调用 FishControlPipeline.run(...)**（整条 8 段流水线端到端），我也可以直接给一套“最小可跑 pipeline 空实现 + 测试端到端 harness”。

兄弟，我们还没有把这个整体算法整理固定成一个 ITEM，读者在 ITEM 单子中找不到我们在谈什么逻辑遗漏与鱼控。请把整个 RTB 加速算法整理固定成一个 ITEM #228.