

Citation Gravity

A closed-corpus self-audit algorithm that discovers the bedrock authority a legal database is silently missing.

Author: Richard L. Sanders · Utah Bar #15728 **Role:** Sole inventor and implementer
Version: 1.0 — 2026-05-13 (engine v3, SQL-aggregated) **Source:** `scripts/citation-gravity-v3.js`, with consumer `scripts/landmark-harvest.js`, in the public repository at [<https://github.com/Benchslap/Benchslap>](https://github.com/Benchslap/Benchslap)

Abstract

Every legal-research corpus is incomplete in the same characteristic way: it has the cases that were easy to ingest, and it is missing the cases that everyone cites but nobody bothers to ingest because the cases are widely available elsewhere. *Smith v. Smith on alimony*. *Von Hake on contempt*. *Dahl on the standard of review*. The corpus has 3.2 million opinions and still cannot verify the seven cases everyone needs to verify on a Tuesday.

Citation Gravity is a closed-corpus self-audit that discovers exactly which cases the corpus is missing **by counting which cases the corpus's own opinions cite that the corpus does not itself contain**. The cases with the highest incoming-citation count from the corpus, but no row of their own, are the gravity wells: the bedrock authority the corpus needs and lacks.

The output is a ranked list of missing-bedrock citations, sorted by inbound citation count. The companion harvester (`landmark-harvest.js`) consumes the list and fetches each missing case from its public `.gov` source so the next pass of the engine returns a higher-coverage answer. The whole cycle is closed-loop: the corpus tells the harvester what to harvest next, and the corpus stays self-completing.

This paper describes the algorithm, the SQL-aggregated streaming implementation that makes it tractable on a 3.2-million-row corpus inside a 4 GB RAM budget, and the closed-loop coupling with the harvester that distinguishes it from any third-party citation-discovery tool.

1 · The problem

A 3-million-opinion corpus that does not contain *Marbury v. Madison* is, for the user looking for *Marbury v. Madison*, a 0-opinion corpus. Coverage on a closed corpus is not measured by the number of opinions it contains — it is measured by **which** opinions it contains.

The hard part is identifying the missing-bedrock cases. There is no list of "the cases every legal corpus must have." Different practice areas have different bedrock cases; different courts have different controlling authority; different research questions reach for different parts of the doctrine. A static reference list compiled by hand is wrong the day it is published.

The cases that are most-cited are not the same as the cases that are most-needed by the user, because most-cited is biased by the corpus's existing contents (more contracts opinions → more contracts citations). The cases that are most-needed are the ones the corpus's opinions cite **but the corpus itself does not contain**. That is a different set, and it is computable directly from the corpus's own opinion text.

I built Citation Gravity to compute it.

2 · The algorithm

Two sets, two passes, one filter.

Set H — the have-set. Every citation, neutral or reporter form, that already corresponds to an opinion row in `opinion_library`. This is the corpus's existing reach.

Set C — the cited-set. Every citation that appears in the `opinion_text` of any row in `opinion_library`. This is the universe of citations the corpus's own opinions point at.

Gravity = $C \setminus H$. Citations the corpus's opinions cite, that the corpus itself does not contain. The cardinality of each element in $C \setminus H$ is the count of how many times that missing citation is cited by the existing corpus. Sort descending. The top of the list is the gravity wells.

Filter: drop citations with fewer than `min-cites` inbound citations (default 5). Below that threshold the citation is either a one-off (genuinely rare and not bedrock) or a parser error (a sequence of digits and letters that happened to match a citation pattern). The threshold tunes the precision-recall trade.

3 · Pattern families

`PATTERNS` in `citation-gravity-v3.js` is a three-tuple of regex families covering the citation forms a U.S. legal corpus contains:

1. **Federal reporters + state reporters.** `<vol> <reporter>`
`<page>` for `U.S.`, `S.Ct.`, `L.Ed.2d`, `F.2d/3d`, `F.Supp.2d/3d`, `P.2d/3d`,
`So.2d/3d`, `N.E.2d/3d`, `N.W.2d/3d`, `S.E.2d/3d`, `S.W.2d/3d`, `A.2d/3d`,
`Cal.Rptr.2d/3d`, `N.Y.S.2d/3d`.
1. **Modern neutral cites.** `<year> <state-code> [App]`
`<number>` for the fourteen states that have adopted public-domain neutral citation: UT, WY, CO, NM, OH, VT, ME, MT, ND, SD, AK, NH, WV, ID.
1. **Older / longer-form state reporters.** `<vol> <reporter>`
`<page>` for `Wis. 2d`, `Minn.`, `Iowa`, `Neb.`, `Kan.`, `Ore.`, `Wash. 2d`,
`Conn.`, `Md.`, `Mass.`, `N.J.`, `Pa. Super.`, `Va.`, `Tex.`, `Fla.`, `Ill.`, `Ohio St.`.

The pattern families are deliberately overinclusive — they capture more than the ideal canonical form, on the theory that downstream normalization (lowercasing + whitespace collapse) will collapse near-duplicates into single entries, and the filter threshold will drop the noise. False positives cost nothing; missed citations cost everything.

4 · Memory-bounded SQL aggregation

The naive implementation — an in-memory Map keyed by citation — does not fit. A 3.2-million-row scan finds ~600,000 unique missing citations; at 200-400 bytes per Map entry plus V8 overhead, the memory pressure exceeds the 4 GB Node heap before the scan completes.

`citation-gravity-v3.js` moves the aggregation to PostgreSQL via an UNLOGGED temp table. The implementation is three phases:

Phase 1 — build the have-set in a JavaScript Set

Page through `opinion_library` ordering by `id`, batch size 5,000, accumulating `(neutral_citation, reporter_citation)` lowercased + whitespace-normalized into a JavaScript Set. The Set fits — every opinion row has at most two citation strings, and the set is bounded by the size of the corpus (3.2M opinions → ~6M strings → ~600 MB peak). Phase 1 typically completes in 60-180 seconds on the production droplet.

Phase 2 — stream `opinion_text` into a temp table

Create an UNLOGGED scratch table `citation_gravity_scratch(normalized text, raw text)`. Page through `opinion_library` again, batch 500, extracting every citation matching the three pattern families and buffering 10,000 `(normalized, raw)` tuples at a time before bulk-INSERTing them into the scratch table. The buffer flushes on hitting 10,000 entries, holding peak memory at ~50 MB regardless of corpus size. PostgreSQL bears the storage weight, which is what PostgreSQL is for. Phase 2 typically completes in 15-30 minutes on the production droplet under harvester load.

Phase 3 — aggregate + filter in a single SQL

A single CTE computes everything at the server side:

```
WITH ourcites AS (  
    SELECT lower(trim(neutral_citation)) AS n FROM opinion_library WHERE  
    neutral_citation IS NOT NULL  
    UNION  
    SELECT lower(trim(reporter_citation))      FROM opinion_library WHERE  
    reporter_citation IS NOT NULL  
) ,  
counts AS (  
    SELECT normalized, MAX(raw) AS raw, COUNT(*)::int AS n  
    FROM citation_gravity_scratch  
    GROUP BY normalized  
)  
SELECT counts.raw AS citation, counts.normalized, counts.n AS count  
FROM counts  
WHERE counts.n >= $1  
    AND counts.normalized NOT IN (SELECT n FROM ourcites)  
ORDER BY counts.n DESC  
LIMIT $2
```

This single query is bounded by the size of the scratch table (millions of rows but UNLOGGED so it does not touch WAL) and PostgreSQL's GROUP BY hash. It completes in 60-300 seconds on the production droplet. The Node side reads the result rows, writes a JSON file, drops the scratch table, exits.

End-to-end runtime on a 3.2-million-row corpus: 20-50 minutes. Peak Node heap: under 800 MB. Disk pressure: bounded to PostgreSQL's UNLOGGED storage, which is dropped at exit.

5 • Closed-loop coupling with the harvester

The output of Citation Gravity is the input to `landmark-harvest.js` — a companion script that consumes the JSON and, for each missing-bedrock citation, classifies the jurisdiction (Utah neutral / SCOTUS reporter / federal-circuit reporter / regional reporter),

searches the appropriate source (CourtListener citation lookup, the state's .gov opinion repository, the Caselaw Access Project bulk data), fetches the opinion text, AEGIS-hashes it (via `lib/authority-hash.js`), and INSERTs it into `opinion_library`.

The next pass of Citation Gravity sees the newly-ingested cases in `H` (the have-set), removes them from the output, and surfaces the next layer of gravity wells. The corpus self-completes: the cases the corpus most needs to harvest next are the cases the corpus is missing that the corpus's opinions are already citing most. The harvest pipeline points itself at its own gaps.

This is what makes the algorithm load-bearing in the long run. A static "top 100 cases by citation count" list is wrong the moment a new term of opinions arrives. A closed-loop self-audit is right by construction, because it derives the gap from the corpus the next pass will sample against.

6 · Why this is not a third-party citation-discovery tool

A commercial citation-discovery service (Shepard's, KeyCite, etc.) tells the user whether a citation has been overruled, cited positively, or treated negatively. That is a different problem. Those services answer "what should I think about this citation?" Citation Gravity answers "what citations does my corpus need to start containing in order to verify the arguments my users are going to make tomorrow?"

The third-party tools cannot answer Citation Gravity's question, because they do not know what citations are in your corpus. They know what citations are in their corpus. The gap between the two is the question Citation Gravity is built to answer; the third-party tools cannot help, by construction.

A commercial competitor wanting to copy Citation Gravity has to first build a corpus. They cannot use a third-party corpus because the gravity computation depends on the corpus being the same one the user is querying — running the algorithm against CourtListener tells you what CourtListener is missing, not what your tool is missing. The corpus dependency is the moat.

7 · Novelty claims

1. **Closed-corpus self-audit algorithm** in which the corpus's own opinions are used to compute the corpus's own missing-bedrock citations, ranked by inbound citation count from the corpus to the missing citation.
2. **Memory-bounded streaming + SQL-aggregated implementation** in which the citation accumulator lives in an UNLOGGED PostgreSQL temp table rather than an

in-memory Map, allowing the algorithm to run against a 3-million-row corpus inside a 4 GB Node heap.

3. **Three-pattern-family citation extractor** covering federal reporters, modern state neutral cites, and older / longer-form state reporters in a single regex family applied to the `opinion_text` scan.
4. **Closed-loop coupling with a downstream harvester** in which the gravity output is the harvester's next input, producing a self-completing corpus that converges on the bedrock authority its own opinions cite.

The four elements in conjunction are Citation Gravity. The canonical implementation lives at `scripts/citation-gravity-v3.js`; the consumer lives at `scripts/landmark-harvest.js`.

© Richard L. Sanders 2026. All rights reserved except those granted by the project's open license.