Databases

Optimal Join Algorithms

Pierre Senellart

ENS
ÉCOLE NORMALE SUPÉRIEURE
PSL

18 March 2026

Plan

Joins everywhere

- Conjunctive queries are essentially joins plus projection
- Central operation in:
  - relational algebra
  - SQL query evaluation
  - Datalog engines
  - graph pattern matching
  - constraint satisfaction
- Basic implementation paradigm: evaluate an n-way join as a sequence of binary joins
- Question for today:
  is this always the right asymptotic strategy?

A running example: the triangle query

We will use throughout the lecture the Boolean conjunctive query

$$q_{\triangle}(x, y, z) \leftarrow R(x, y) \wedge S(y, z) \wedge T(x, z)$$

Interpretation:

- R, S, T are binary relations
- we look for triples (x, y, z) such that all three edges exist
- in graph terms: we are looking for triangles

This is one of the simplest examples where classical binary-join plans are not asymptotically optimal [Ngo et al., 2012, Veldhuizen, 2014].

The main questions

We will study three successive questions:

1. How are joins classically evaluated?
2. How large can the output of a join be in the worst case?
3. Can we evaluate joins in time matching that worst-case output size?

The answers will involve:

- binary join trees
- hypergraphs
- fractional edge covers
- worst-case optimal algorithms

Plan

What is an n-way join?

Let $R_1, \ldots, R_m$ be relation symbols.

The corresponding join is the natural join

$$\bowtie_{i=1}^{m} R_i$$

when relations share attribute names.

Equivalently, for a CQ

$$q(\mathbf{x}) \leftarrow \exists \mathbf{y} : R_1(\mathbf{z}_1) \wedge \cdots \wedge R_m(\mathbf{z}_m)$$

evaluation amounts to:

- compute the join of all atoms
- then project to the free variables $\mathbf{x}$

So the complexity of CQ evaluation is largely driven by the complexity of join evaluation.

Binary join plans

Classically, an n-way join is turned into a binary tree of joins.

Examples with three relations:

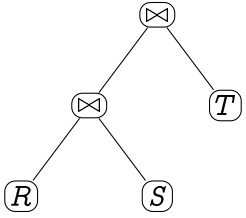
$$(R \bowtie S) \bowtie T \qquad\qquad R \bowtie (S \bowtie T)$$

Common shapes:

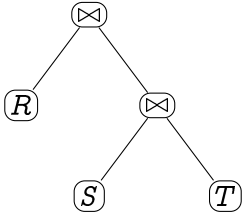
- Left-deep: every right child is a base relation
- Right-deep: every left child is a base relation
- Bushy: arbitrary binary tree

Whatever the physical implementation of each binary join, the logical paradigm is:
combine two relations at a time

Example of join trees



Left-deep plan: $((R \bowtie S) \bowtie T)$



Right-deep plan: $(R \bowtie (S \bowtie T))$

Complexity measure

For today, we ignore:

- detailed cost models
- choice of the join algorithm: hash join, sort-merge join, index nested loop join...
- low-level engineering issues

We focus on a more abstract question:

what is the asymptotic complexity of evaluating a join?

Relevant quantities:

- input size
- output size
- size of intermediate results
- structure of the query

Finding the best binary join plan

Assume for a moment a very simple cost model:

- the cost of a plan is the sum of the sizes of all intermediate results it materializes
- equivalently, we ignore low-level implementation choices and only ask: *in which order should we join the relations?*

Then the optimization problem is:

among all binary join trees for

$$R_1 \bowtie \ldots \bowtie R_n,$$

find the one of minimum estimated cost

Even with this very simple model, the search space is already very large.

How many binary join plans are there?

For n relations:

- the number of binary tree shapes with n leaves is the $(n-1)$-st Catalan number

$$C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1}$$

- if the leaves are labeled by $R_1, \ldots, R_n$, the number of binary join plans is

$$C_{n-1} \cdot n!$$

For left-deep plans only:

- the plan is entirely determined by the order in which relations are introduced
- hence there are only $n!$ left-deep plans

So left-deep plans are much fewer than arbitrary bushy plans, but still exponential in n.

How to search this space? What systems do

- Classical optimizers use dynamic programming:
  - for each subset $S \subseteq \{R_1, \ldots, R_n\}$, keep the best plan found for joining exactly the relations in S
  - build optimal plans bottom-up, from subsets of size 1 to subsets of size n
- May restrict to specific types of join (e.g., left-deep)
- May use heuristic or learning approaches for large plans (e.g., genetic algorithms in PostgreSQL, cf. GEQO)
- This is the right framework for binary join plans.
- But today's lecture asks a different question:
  what if the best binary plan is still asymptotically suboptimal?

A first crude bound on output size

For arbitrary relations $R_1, \ldots, R_m$,

$$|\bowtie_{i=1}^{m} R_i| \leqslant \prod_{i=1}^{m} |R_i|$$

This is immediate, but usually much too coarse.

Problems:

- ignores shared attributes
- ignores query structure
- can be exponentially larger than the real worst-case output
- says nothing precise about intermediate results in a binary plan

Triangle query under a binary plan

Consider again

$$R(x, y) \wedge S(y, z) \wedge T(x, z)$$

and the plan

$$(R \bowtie S) \bowtie T$$

If $|R| = |S| = |T| = N$, then:

- $R \bowtie S$ contains tuples (x, y, z) with $(x, y) \in R$ and $(y, z) \in S$, up to N^2 such tuples
- in the worst case, this intermediate relation can have size much larger than the final output (maybe no tuple or only 1 tuple in the output!)

This is the first sign that output size alone is not enough to understand the complexity of binary plans.

The Loomis–Whitney instance for triangles

Let A, B, C be sets of size n and define:

$$R = A \times B$$
$$S = B \times C$$
$$T = A \times C$$

Then:

- $|R| = |S| = |T| = n^2 (= N)$
- the output size is

$$|R \bowtie S \bowtie T| = |R \bowtie S| = n^3 = N^{3/2}$$

This already shows that the triangle query can have superlinear output size but still much smaller than the naive bound N^3.

A binary plan may still overpay

In the previous example,

$$|R \bowtie S| = n^3 = N^{3/2}$$

so the intermediate is not larger than the output.

However, on other instances, a binary plan may materialize many tuples of $R \bowtie S$ that will never match any tuple of T.

Main message:

- the binary decomposition is a restriction
- the right baseline (what we can hope at best) is not the "best binary plan"
- the right baseline is the largest possible output size

Main limitation of binary plans

Binary joins are local:

- first combine two relations
- then combine the result with another relation
- and so on

But a tuple may be relevant only if it satisfies all constraints simultaneously.

So the natural question is:

can we evaluate a join globally, without materializing large pairwise intermediates?

Plan

Hypergraph of a join query

Recall a CQ (or a join query) can be represented as a hypergraph $\mathcal{H} = (V, E)$:

- vertices V: the attributes / variables
- hyperedges E: one hyperedge per relation atom

Example

- The triangle query:
$$R(x, y) \wedge S(y, z) \wedge T(x, z)$$
gives the graph with edges $\{x, y\}$, $\{y, z\}$, $\{x, z\}$
- The star query:
$$R_1(x, y_1) \wedge \cdots \wedge R_k(x, y_k)$$
gives a star-shaped hypergraph

Why the hypergraph matters

Queries with the same number of atoms can behave very differently.

Compare:

- a star query:

$$R_1(x, y_1) \wedge \cdots \wedge R_k(x, y_k)$$

- a cycle query:

$$R_1(x_1, x_2) \wedge R_2(x_2, x_3) \wedge \cdots \wedge R_k(x_k, x_1)$$

They have the same arity pattern, but different structural properties, and different worst-case output sizes.

We need bounds that depend on the hypergraph, not just on the number of atoms.

Fractional edge covers

Definition

Let $\mathcal{H} = (V, E)$ be a hypergraph. A fractional edge cover is a family $(x_e)_{e \in E}$ of nonnegative reals such that for every vertex $v \in V$,

$$\sum_{e \ni v} x_e \geqslant 1$$

Intuition:

- each edge contributes a fractional weight
- every vertex must be covered up to weight at least 1

This is a relaxation of the usual notion of edge cover.

The AGM bound

Theorem (Atserias–Grohe–Marx [Atserias et al., 2013])

Let $Q = \bowtie_{e \in E} R_e$ be a natural join query. For every fractional edge cover $(x_e)_{e \in E}$ of the query hypergraph,

$$|Q| \leqslant \prod_{e \in E} |R_e|^{x_e}$$

The best such bound is obtained by solving the linear program:

$$\min \sum_{e \in E} x_e \log |R_e| \quad \text{subject to} \quad x_e \geqslant 0, \ \sum_{e \ni v} x_e \geqslant 1 \ \forall v$$

This is the AGM bound.

AGM bound on the triangle query

For

$$R(x, y) \wedge S(y, z) \wedge T(x, z)$$

the constraints are:

$$\begin{aligned} x_R + x_T &\geqslant 1 \qquad (\text{cover } x) \\ x_R + x_S &\geqslant 1 \qquad (\text{cover } y) \\ x_S + x_T &\geqslant 1 \qquad (\text{cover } z) \end{aligned}$$

If $|R| = |S| = |T| = N$, a symmetric optimal solution is

$$x_R = x_S = x_T = \frac{1}{2}$$

Hence

$$|R \bowtie S \bowtie T| \leqslant N^{1/2} N^{1/2} N^{1/2} = N^{3/2}$$

This bound is tight.

AGM bound on a star query

Consider

$$R_1(x, y_1) \wedge \cdots \wedge R_k(x, y_k)$$

with all relations of size N.

To cover each leaf variable y_i, edge R_i must get weight at least 1.

So the unique feasible solution is

$$x_{R_1} = \cdots = x_{R_k} = 1$$

Hence the AGM bound is

$$N^k$$

This is also tight.

Interpretation of the AGM bound

The AGM bound gives a tight worst-case upper bound on the output size:

- it depends on the relation sizes
- it depends on the hypergraph structure
- it is often much smaller than the naive product bound

Natural goal:

design algorithms whose running time is close to the AGM bound

This leads to the notion of worst-case optimal join algorithms.

Plan

Worst-case optimal join algorithms

Definition

A join algorithm is worst-case optimal if, for every fixed join query, its running time is bounded by

$$O(\mathrm{IN} + \mathrm{OUT}_{\max})$$

up to polylogarithmic factors, where $\mathrm{OUT}_{\max}$ denotes the AGM bound.

Informally:

- one cannot hope to do asymptotically better than reading the input
- one cannot hope to do asymptotically better than producing the output
- the AGM bound captures the largest possible output

Why binary joins are not worst-case optimal

On the triangle query with relations of size N:

- AGM bound: $O(N^{3/2})$
- any approach that is forced through certain large intermediates may pay $\Omega(N^2)$

So binary plans can be asymptotically worse than the best possible worst-case complexity [Ngo et al., 2012].

This is a structural limitation:

- not a problem of choosing the wrong physical join method
- not a problem of bad cardinality estimates
- a problem with the binary decomposition itself

A different paradigm

Instead of:

computing the join through pairwise materialization

we will:

search directly in the space of assignments to variables

We focus on the full join case, i.e., no projection.

Key ideas:

- fix an order on the attributes
- enumerate consistent partial assignments
- intersect candidate value sets from all relevant relations
- never materialize large binary intermediates

NPRR in one slide

The first general worst-case optimal join algorithm is due to Ngo, Porat, Ré, and Rudra [Ngo et al., 2012, 2018].

Main ideas:

- recursive decomposition of the query
- heavy/light partitioning of relation values (heavy have a lot of matches, but are rare; light can be numerous but have few matches)
- careful control of branching according to the AGM solution

Importance:

- first constructive proof that the AGM bound is algorithmically achievable
- conceptually foundational
- technically less simple to present than LFTJ which we will see next

Plan

Why focus on Leapfrog Triejoin?

Among worst-case optimal join algorithms, Leapfrog Triejoin (LFTJ) [Veldhuizen, 2014]:

- is conceptually simple
- has a very elegant structure
- is close to classical backtracking search
- has been implemented in practice

Ordered relations as tries

Fix a global variable order, for instance

$$x_1, \ldots, x_n$$

Each relation is stored according to the order induced on its attributes.

Example: for relation $R(x, y)$, ordered by (x, y), we can view the data as a trie:

- first level: values of x
- second level under each x: values of y

For $R(x, y, z)$ ordered by (x, y, z):

- level 1: possible x
- level 2: possible y for the chosen x
- level 3: possible z for the chosen (x, y)

A tiny trie example

Consider the relation

$$R(x, y) = \{(1, 2), (1, 4), (3, 1), (3, 5)\}$$

ordered by (x, y).

It can be seen as the trie:

$$1 \mapsto \{2, 4\}, \qquad 3 \mapsto \{1, 5\}$$

In other words:

- possible values of x are $1, 3$
- under $x = 1$, possible values of y are $2, 4$
- under $x = 3$, possible values of y are $1, 5$

LFTJ navigates these tries with synchronized iterators.

Primitive operation: intersection

- Suppose we have already fixed some prefix of variables.
- To choose the next variable x_i, we must ensure consistency with all relations containing x_i.
- Each such relation provides a sorted set of admissible values for x_i under the current prefix.
- The next values to explore are therefore in the intersection of these sorted sets.
- Core primitive of LFTJ:
  compute the intersection of several sorted iterators efficiently

The leapfrog idea

To intersect sorted sets $S_1, \ldots, S_k$:

- maintain one iterator per set
- compare current values
- advance the smallest one
- if all iterators point to the same value, output it

Because all sets are sorted, no value is visited too many times.

Thus multiway intersection can be done essentially in time linear in the total number of iterator advances.

LFTJ at a high level

Let the variables be ordered as

$$x_1, \ldots, x_n$$

LFTJ proceeds recursively:

1. determine all admissible values for x_1
2. for each such value, determine all admissible values for x_2
3. continue until x_n

At level i:

- only relations containing x_i matter
- their admissible value sets are intersected
- each resulting value extends the current partial assignment

Pseudocode sketch

```
Enumerate(i, partial assignment theta):
  if i > n:
    output theta
  else:
    compute for x_i the admissible values
      from all relations containing x_i
      under theta
    for each value a in their intersection:
      Enumerate(i+1, theta extended with x_i = a)
```

The key point is that the admissible values are not obtained by materializing intermediate joins, but by local intersections in the tries.

Triangle query with variable order x, y, z

Consider

$$R(x, y) \wedge S(y, z) \wedge T(x, z)$$

and choose the order

$$x,\ y,\ z$$

Relations are stored as:

- R ordered by (x, y)
- T ordered by (x, z)
- S ordered by (y, z)

We now see what LFTJ does.

Step 1: choosing x

Variable x occurs in:

- $R(x, y)$
- $T(x, z)$

At the first level, admissible values for x are:

$$\pi_x(R) \cap \pi_x(T)$$

So LFTJ starts by intersecting the sorted lists of values of x present in R and T.

Each common value a yields a partial assignment

$$x = a$$

Step 2: choosing y once $x = a$ is fixed

Now assume we fixed $x = a$.

Variable y occurs in:

- $R(x, y)$, which under $x = a$ yields the set

$$\{ y \mid (a, y) \in R \}$$

- $S(y, z)$, which contributes the set

$$\pi_y(S)$$

So admissible values for y are:

$$\{ y \mid (a, y) \in R \} \cap \pi_y(S)$$

Each common value b yields a partial assignment

$$x = a,\ y = b$$

Step 3: choosing z once $x = a$, $y = b$ are fixed

Now assume we fixed

$$x = a, \qquad y = b$$

Variable z occurs in:

- $S(y, z)$, which under $y = b$ yields

$$\{ z \mid (b, z) \in S \}$$

- $T(x, z)$, which under $x = a$ yields

$$\{ z \mid (a, z) \in T \}$$

So admissible values are:

$$\{ z \mid (b, z) \in S \} \cap \{ z \mid (a, z) \in T \}$$

Every common value c gives an output tuple (a, b, c).

What is missing compared with a binary plan?

LFTJ never builds the explicit relation

$$R \bowtie S$$

Instead, it only explores values that survive all local consistency tests.

In particular:

- no explicit ternary intermediate is materialized
- work is concentrated on the prefixes that can lead to actual answers
- this is why worst-case optimality becomes possible

Complexity result

Theorem (informal)

Algorithms such as NPRR and LFTJ evaluate any full conjunctive join in time matching the AGM bound up to logarithmic factors [Ngo et al., 2012, Veldhuizen, 2014].

For the triangle query with input relations of size N:

$$O(N^{3/2} \operatorname{polylog} N)$$

is achievable.

This matches the worst-case output size up to logarithmic overhead.

Plan

Acyclic queries as another structural tractability story

- Worst-case optimal joins are one structural answer.
- Another important answer is by restricting the shape of the query.
- For acyclic queries, Yannakakis's algorithm evaluates joins very efficiently [Yannakakis, 1981]:
  - semijoin reduction
  - then bottom-up evaluation on a join tree
- So there are at least two complementary viewpoints:
  - exploit special structure (acyclicity, bounded width...)
  - design general algorithms that are worst-case optimal

Beyond acyclicity

- More generally, many tractability results depend on width measures:
  - treewidth of the primal graph
  - hypertree width
  - fractional hypertree width
- These notions refine the idea that some cyclic queries are still "close to acyclic" and can be evaluated efficiently by decomposition methods [Gottlob et al., 2002].
- Worst-case optimal joins and width-based algorithms should be seen as two parts of the same broader story:

  query structure determines complexity

What this lecture did not cover

We intentionally ignored many important practical questions:

- detailed cost models
- statistics and cardinality estimation
- choice of physical join operator
- pipelining, caching, and memory management
- how to combine worst-case optimality with other relational operators

These belong more naturally to a systems-oriented lecture on query optimization.

Possible extensions

Topics that naturally follow:

- Joins with projection and aggregation: the FAQ framework and variable elimination [Khamis et al., 2016, 2020]
- Factorized query evaluation and representations: compact representations of query results and their complexity [Olteanu and Závodný, 2015, Bakibayev et al., 2016]
- Beyond worst-case analysis: degree-based bounds and finer instance-sensitive guarantees [Ngo et al., 2014, Khamis et al., 2024]

Conclusions

- The classical evaluation of an n-way join uses a binary join plan
- Binary plans may create intermediate results much larger than the final output
- The AGM bound gives a tight worst-case upper bound on the output size of a join, based on fractional edge covers [Atserias et al., 2013]
- Worst-case optimal join algorithms such as NPRR and LFTJ match this bound up to logarithmic factors [Ngo et al., 2012, Veldhuizen, 2014]
- Conceptually, they replace binary materialization by search over variable assignments

Bibliography I

Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. doi: 10.1137/110859440. URL https://doi.org/10.1137/110859440.

Nurzhan Bakibayev, Tomás Kociský, and Dan Olteanu. Aggregation and ordering in factorised databases. In *Proceedings of the 19th International Conference on Database Theory (ICDT)*, pages 6:1–6:18, 2016.

Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002. doi: 10.1006/JCSS.2001.1809. URL https://doi.org/10.1006/jcss.2001.1809.

Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 13–28. ACM, 2016. doi: 10.1145/2902251.2902280. URL https://doi.org/10.1145/2902251.2902280.

Bibliography II

Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Functional aggregate queries with additive inequalities. *ACM Trans. Database Syst.*, 45 (4):17:1–17:41, 2020. doi: 10.1145/3426865. URL https://doi.org/10.1145/3426865.

Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. Join size bounds using l_p-norms on degree sequences. *Proc. ACM Manag. Data*, 2(2): 96, 2024. doi: 10.1145/3651597. URL https://doi.org/10.1145/3651597.

Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 37–48. ACM, 2012. doi: 10.1145/2213556.2213565. URL https://doi.org/10.1145/2213556.2213565.

Bibliography III

Hung Q. Ngo, Dung T. Nguyen, Christopher Ré, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 234–245. ACM, 2014. doi: 10.1145/2594538.2594547. URL https://doi.org/10.1145/2594538.2594547.

Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018. doi: 10.1145/3180143. URL https://doi.org/10.1145/3180143.

Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi: 10.1145/2656335. URL https://doi.org/10.1145/2656335.

Bibliography IV

Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014. doi: 10.5441/002/ICDT.2014.13. URL `https://doi.org/10.5441/002/icdt.2014.13`.

Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

Licensing