Skew Strikes Back: New Developments in the Theory of Join Algorithms

Hung Q. Ngo
University at Buffalo, SUNY
hungngo@buffalo.edu

Christopher Ré
Stanford University
chrismre@cs.stanford.edu

Atri Rudra
University at Buffalo, SUNY
atri@buffalo.edu

October 17, 2013

1 Introduction

Evaluating the relational join is one of the central algorithmic and most well-studied problems in database systems. A staggering number of variants have been considered including Block-Nested loop join, Hash-Join, Grace, Sort-merge (see Grafe [20] for a survey, and [5, 8, 27] for discussions of more modern issues). Commercial database engines use finely tuned join heuristics that take into account a wide variety of factors including the selectivity of various predicates, memory, IO, etc. In spite of this study of join queries, the textbook description of join processing is *suboptimal*. This survey describes recent results on join algorithms that have provable worst-case optimality runtime guarantees. We survey recent work and provide a simpler and unified description of these algorithms that we hope is useful for theory-minded readers, algorithm designers, and systems implementors.

Much of this progress can be understood by thinking about a simple join evaluation problem that we illustrate with the so-called *triangle query*, a query that has become increasingly popular in the last decade with the advent of social networks, biological motifs, and graph databases [39, 40]

> *Suppose that one is given a graph with N edges, how many distinct triangles can there be in the graph?*

A first bound is to say that there are at most N edges, and hence at most $O(N^3)$ triangles. A bit more thought suggests that every triangle is indexed by any two of its sides and hence there at most $O(N^2)$ triangles. However, the correct, tight, and non-trivial asymptotic is $O(N^{3/2})$. An example of the questions this survey is how do we list all the triangles in time $O(N^{3/2})$? Such an algorithm would have a worst-case optimal running time. In contrast, traditional databases evaluate joins pairwise, and as has been noted by several authors, this forces them to run in time $\Omega(N^2)$ on some instance of the triangle query. This survey gives an overview of recent developments that establish such non-trivial bounds for *all* join queries and algorithms that meet these bounds, which we call worst-case optimal join algorithms.

Estimates on the output size of join have been known since the 1990s, thanks to the work of Friedgut and Kahn [14] in the context of bounding the number of occurrences of a given small hypergraph inside a large hypergraph. More recently and more generally, tight estimates for the natural join problem were derived by Grohe-Marx [23] and Atserias-Grohe-Marx [3] (AGM henceforth). In fact, similar bounds can be traced back to the 1940s in geometry, where it was known as the famous Loomis-Whitney inequality [29]. The most general geometric bound is by Bollobás-Thomason in the 1990s [6]. We proved (with Porat) that AGM and the discrete version of Bollobás-Thomason are *equivalent* [32], and so the connection between these areas is deep.

Connections of join size to arcane geometric bounds may reasonably lead a practitioner to believe that the cause of suboptimality is a mysterious force wholly unknown to them—but it is not; it is the old enemy of the database optimizer, skew. We hope to highlight two conceptual messages with this survey:

- The main ideas of the the algorithms presented here are an optimal way of avoiding skew – something database practitioners have been fighting with for decades. We describe a theoretical basis for one family of techniques to cope with skew by relating them to geometry.
- The second idea is a challenge to the database dogma of doing *"one join at a time,"* as is done in traditional database systems. We show that there are classes of queries for which *any* join-project plan is destined to be slower than the best possible run time by a polynomial factor *in the data size*.

Outline of the Survey. We begin with a short (and necessarily incomplete) history of join processing with a focus on recent history. In Section 2, we describe how these algorithms work for the triangle query. In Section 3, we describe how to use these new size bounds for join queries. In Section 4, we provide new simplified proofs of these bounds and join algorithms. Finally, we describe two open questions in Section 5

A Brief History of Join Processing

Conjunctive query evaluation in general and join query evaluation in particular have a very long history and deep connections to logic and constraint satisfaction [7, 9, 13, 17, 19, 28, 33, 41]. Most of the join algorithms with provable performance guarantees work for specific classes of queries.[1] As we describe, there are two major approaches for join processing: using *structural information of the query* and *using cardinality information*. As we explain, the AGM bounds are exciting because they bring together both types of information.

The Structural Approaches On the theoretical side, many algorithms use some structural property of the query such as *acyclicity* or *bounded "width."* For example, when the query is acyclic, the classic algorithm of Yannakakis [45] runs in time essentially linear in the input plus output size. A query is acyclic if and only if it has a *join tree*, which can be constructed using the textbook *GYO-reduction* [21, 46]

Subsequent work further expand the classes of queries which can be evaluated in polynomial time. These work define progressively more general notions of "width" for a query, which intuitively measures how far a query is from being acyclic. Roughly, these results state that if the corresponding notion of "width" is bounded by a constant, then the query is "tractable," i.e. there is a polynomial time algorithm to evaluate it. For example, Gyssens et al. [24, 25] showed that queries with bounded "degree of acyclicity" are tractable. Then come *query width* (qw) from Chekuri and Rajaraman [9], *hypertree width* and *generalized hypertree width* (ghw) from Gottlob et al. [18, 37]. These are related to the *treewidth* (tw) of a query's hypergraph, rooted in Robertson and Seymour on graph minors [36]. Acyclic queries are exactly those with qw $= 1$.

Cardinality-based Approaches Width only tells half of the story, as was wonderfully articulated in Scarcello's SIGMOD Record paper [37]:

> *decomposition methods focus "only" on structural features, while they completely disregard "quantitative" aspects of the query, that may dramatically affect the query-evaluation time.*

[1] Throughout this survey, we will measure the run time of join algorithms in terms of the input data, assuming the input query has constant size; this is known as the *data complexity* measure, which is standard in database theory [41].

Said another way, the width approach disregards the input relation sizes and summarizes them in a single number, N. As a result, the run time of these structural approaches is $O(N^{w+1} \log N)$, where N is the input size and w is the corresponding width measure. On the other hand, commercial RDBMSs seem to place little emphasis on the structural property of the query and tremendous emphasis on the cardinality side of join processing. Commercial databases often process a join query by breaking a complex multiway join into a series of pairwise joins; an approach first described in the seminal System R, Selinger-style optimizer from the 1970 [38]. However, throwing away this structural information comes at a cost: *any* join-project plan is destined to be slower than the best possible run time by a polynomial factor *in the data size*.

Bridging This Gap A major recent result from AGM [3, 23] is the key to bridging this gap: AGM derived a *tight* bound on the output size of a join query as a function of individual input relation sizes *and* a much finer notion of "width". The AGM bound leads to the notion of *fractional query number* and eventually *fractional hypertree width* (fhw) which is strictly more general than all of the above width notions [31]. To summarize, for the same query, it can be shown that

$$\text{fhw} \leqslant \text{ghw} \leqslant \text{qw} \leqslant \text{tw} + 1,$$

and the join-project algorithm from AGM runs in time $O(N^{\text{fhw}+1} \log N)$. AGM's bound is sharp enough to take into account cardinality information, and they can be *much* better when the input relation sizes vary. The bound takes into account *both* the input relation statistics *and* the structural properties of the query. The question is whether it is possible and how to turn the bound into join algorithms, with runtime $O(N^{\text{fwh}})$ and much better when input relations do not have the same size.

The first such worst-case optimal join algorithm was designed by the authors (and Porat) in 2012 [32]. Soon after, an algorithm (with a simpler description) with a similar optimality guarantee was presented soon after called "Leapfrog Triejoin" [42]. Remarkably this algorithm was implemented in a commercial database system *before* its optimality guarantees were discovered. A key idea in the algorithms is handling skew in a theoretically optimal way, and uses many of the same techniques that database management systems have used for decades heuristically [12, 43, 44]

A technical contribution of this survey is to describe the algorithms from [32] and [42] and their analyses in one unifying (and simplified) framework. In particular, we make the observation that these join algorithms are in fact special cases of a *single* join algorithm. This result is new and serves to explain the common link between these join algorithms. We also illustrate some unexpected connections with geometry, which we believe are interesting in their own right and may be the basis for further theoretical development.

2 Much ado about the triangle

We begin with the triangle query

$$Q_{\triangle} = R(A, B) \bowtie S(B, C) \bowtie T(A, C).$$

The above query is the simplest cyclic query and is rich enough to illustrate most of the ideas in the new join algorithms.[2] We first describe the traditional way to evaluate this query and how skew impacts this query. We then develop two closely related algorithmic ideas allowing us to mitigate the impact of skew in these examples; they are the key ideas behind the recent join processing algorithms.

[2]This query can be used to list all triangles in a given graph $G = (V, E)$, if we set R, S and T to consist of all pairs (u, v) and (v, u) for which uv is an edge. Due to symmetry, each triangle in G will be listed 6 times in the join.

2.1 Why traditional join plans are suboptimal

The textbook way to evaluate any join query, including $Q_\triangle$, is to determine the best pair-wise join plan [35, Ch. 15]. Figure 1 illustrates three plans that a conventional RDBMS would use for this query. For example, the first plan is to compute the intermediate join $P = R \bowtie T$ and then compute $P \bowtie S$ as the final output.

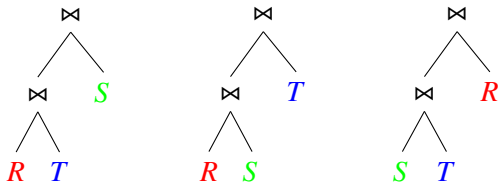

Figure 1: The three pair-wise join plans for $Q_\triangle$.

We next give a family of instances for which any of the above three join plans must run in time $\Omega(N^2)$ because the intermediate relation P is too large. Let $m \geqslant 1$ be a positive integer. The instance family is illustrated in Figure 2, where the domain of the attributes A, B and C are $\{a_0, a_1, \ldots, a_m\}$, $\{b_0, b_1, \ldots, b_m\}$, and $\{c_0, c_1, \ldots, c_m\}$ respectively. In Figure 2, the unfilled circles denote the values a_0, b_0 and c_0 respectively while the black circles denote the rest of the values.

For this instance each relation has $N = 2m + 1$ tuples and $|Q_\triangle| = 3m + 1$; however, any pair-wise join has size $m^2 + m$. Thus, for large m, any of the three join plans will take $\Omega(N^2)$ time. In fact, it can be shown that even if we allow projections in addition to joins, the $\Omega(N^2)$ bound still holds. (See Lemma 4.2.) By contrast, the two algorithms shown in the next section runs in time $O(N)$, which is optimal because the output itself has $\Omega(N)$ tuples!

2.2 Algorithm 1: The Power of Two Choices

Inspecting the bad example above, one can see a root cause for the large intermediate relation: a_0 has "high degree" or in the terminology to follow it is *heavy*. In other words, it is an example of *skew*. To cope with skew, we shall take a strategy often employed in database systems: we deal with nodes of high and low skew using different join techniques [12, 44]. The first goal then is to understand when a value has high skew. To shorten notations, for each a_i define

$$Q_\triangle[a_i] := \pi_{B,C}(\sigma_{A=a_i}(Q_\triangle)).$$

We will call a_i *heavy* if

$$|\sigma_{A=a_i}(R \bowtie T)| \geqslant |Q_\triangle[a_i]|.$$

In other words, the value a_i is *heavy* if its contribution to the size of intermediate relation $R \bowtie S$ is *greater* than its contribution to the size of the output. Since

$$|\sigma_{A=a_i}(R \bowtie S)| = |\sigma_{A=a_i}R| \cdot |\sigma_{A=a_i}S|,$$

we can easily compute the left hand side of the above inequality from an appropriate index of the input relations. Of course, we do not know $|Q_\triangle[a_i]|$ until after we have computed $Q_\triangle$. However, note that we always have $Q_\triangle[a_i] \subseteq S$. Thus, we will use $|S|$ as a proxy for $|Q_\triangle[a_i]|$. The two choices come from the following two ways of computing $Q_\triangle[a_i]$:

(i) Compute $\sigma_{A=a_i}(R) \bowtie \sigma_{A=a_i}(T)$ and filter the results by probing against S or

$$R = \{a_0\} \times \{b_0, \ldots, b_m\} \cup \{a_0, \ldots, a_m\} \times \{b_0\}$$

$$S = \{b_0\} \times \{c_0, \ldots, c_m\} \cup \{b_0, \ldots, b_m\} \times \{c_0\}$$

$$T = \{a_0\} \times \{c_0, \ldots, c_m\} \cup \{a_0, \ldots, a_m\} \times \{c_0\}$$

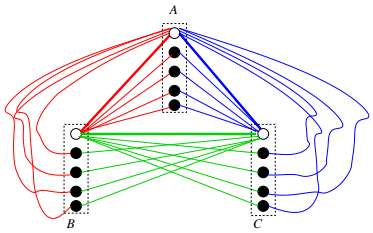

Figure 2: Counter-example for join-project only plans for the triangles (left) and an illustration for $m = 4$ (right). The pairs connected by the red/green/blue edges form the tuples in the relations $R/S/T$ respectively. Note that the in this case each relation has $N = 2m + 1 = 9$ tuples and there are $3m + 1 = 13$ output tuples in $Q_\triangle$. Any pair-wise join however has size $m^2 + m = 20$.

(ii) Consider each tuple in $(b, c) \in S$ and check if $(a_i, b) \in R$ and $(a_i, c) \in T$.

We pick option (i) when a_i is light (low skew) and pick option (ii) when a_i is heavy (high skew).

Example 1. Let us work through the motivating example from Figure 2. When we compute $Q_\triangle[a_0]$, we realize that a_0 is heavy and hence, we use option (ii) above. Since here we just scan tuples in S, computing $Q_\triangle[a_0]$ takes $O(m)$ time. On the other hand, when we want to compute $Q_\triangle[a_i]$ for $i \geqslant 1$, we realize that these a_i's are light and so we take option (i). In these cases $|\sigma_{A=a_i}R| = |\sigma_{A=a_i}T| = 1$ and hence the algorithm runs in time $O(1)$. As there are m such light a_i's, the algorithm overall takes $O(m)$ each on the heavy and light vertices and thus $O(m) = O(N)$ overall which is best possible since the output size is $\Theta(N)$.

Algorithm and Analysis Algorithm 1 fully specifies how to compute $Q_\triangle$ using the above idea of two choices. Given that the relations R, S, and T are already indexed appropriately, computing L in line 2 can easily be done in time $O(\min\{|R|, |S|, |T|\})$. Then, for each $a \in L$, the body of the for loop from line 4 to line 11 clearly takes time in the order of

$$\min\left(|\sigma_{A=a}R| \cdot |\sigma_{A=a}T|, |S|\right),$$

thanks to the power of two choices! Thus, the overall time spent by the algorithm is up to constant factors

$$\sum_{a \in L} \min\left(|\sigma_{A=a}R| \cdot |\sigma_{A=a}T|, |S|\right). \tag{1}$$

We bound the sum above by using two inequalities. The first is the simple observation that for any $x, y \geqslant 0$

$$\min(x, y) \leqslant \sqrt{xy}. \tag{2}$$

The second is the famous Cauchy-Schwarz inequality[3]:

$$\sum_{a \in L} x_a \cdot y_a \leqslant \sqrt{\sum_{a \in L} x_a^2} \cdot \sqrt{\sum_{a \in L} y_a^2}, \tag{3}$$

[3]The inner product of two vectors is at most the product of their length

Algorithm 1 Computing $Q_\triangle$ with power of two choices.

Input: $R(A,B), S(B,C), T(A,C)$ in sorted order

```
1:  Q△ ← ∅
2:  L ← π_A(R) ∩ π_A(T)
3:  For each a ∈ L do
4:      If |σ_{A=a}R| · |σ_{A=a}T| ⩾ |S| then
5:          For each (b,c) ∈ S do
6:              If (a,b) ∈ R and (a,c) ∈ T then
7:                  Add (a,b,c) to Q△
8:      else
9:          For each b ∈ π_B(σ_{A=a}R) ∧ c ∈ π_C(σ_{A=a}T) do
10:             If (b,c) ∈ S then
11:                 Add (a,b,c) to Q△
12: Return Q
```

where $(x_a)_{a\in L}$ and $(y_a)_{a\in L}$ are vectors of real values. Applying (2) to (1), we obtain

$$\sum_{a\in L} \sqrt{|\sigma_{A=a}R| \cdot |\sigma_{A=a}T| \cdot |S|} \tag{4}$$

$$= \sqrt{|S|} \cdot \sum_{a\in L} \sqrt{|\sigma_{A=a}R|} \cdot \sqrt{|\sigma_{A=a}T|} \tag{5}$$

$$\leqslant \sqrt{|S|} \cdot \sqrt{\sum_{a\in L} |\sigma_{A=a}R|} \cdot \sqrt{\sum_{a\in L} |\sigma_{A=a}T|}$$

$$\leqslant \sqrt{|S|} \cdot \sqrt{\sum_{a\in \pi_A(R)} |\sigma_{A=a}R|} \cdot \sqrt{\sum_{a\in \pi_A(T)} |\sigma_{A=a}T|}$$

$$= \sqrt{|S|} \cdot \sqrt{|R|} \cdot \sqrt{|T|}.$$

If $|R| = |S| = |T| = N$, then the above is $O(N^{3/2})$ as claimed in the introduction. We will generalize the above algorithm beyond triangles to general join queries in Section 4. Before that, we present a second algorithm that has exactly the same worst-case run-time and a similar analysis to illustrate the recursive structure of the generic worst-case join algorithm described in Section 4.

2.3 Algorithm 2: Delaying the Computation

Now we present a second way to compute $Q_\triangle[a_i]$ that differentiates between heavy and light values $a_i \in A$ in a different way. We don't try to estimate the heaviness of a_i right off the bat. Algorithm 2 "looks deeper" into what pair (b,c) can go along with a_i in the output by computing c for each candidate b.

Algorithm 2 works as follows. By computing the intersection $\pi_B(\sigma_{A=a_i}R) \cap \pi_B S$, we only look at the candidates b that can possibly participate with a_i in the output (a_i,b,c). Then, the candidate set for c is $\pi_C(\sigma_{B=b}S) \cap \pi_C(\sigma_{A=a_i}T)$. When a_i is really skewed toward the heavy side, the candidates b and then c help gradually reduce the skew toward building up the final solution $Q_\triangle$.

Example 2. Let us now see how delaying computation works on the bad example. As we have observed in using the power of two choices, computing the intersection of two sorted sets takes time at most the *minimum* of the two sizes.

Algorithm 2 Computing $Q_\triangle$ by delaying computation.

Input: $R(A,B), S(B,C), T(A,C)$ in sorted order

```
1: Q ← ∅
2: L_A ← π_A R ∩ π_A T
3: For each a ∈ L_A do
4:     L_B^a ← π_B σ_{A=a} R ∩ π_B S
5:     For each b ∈ L_B^a do
6:         L_C^{a,b} ← π_C σ_{B=b} S ∩ π_C σ_{A=a} T
7:         For each c ∈ L_C^{a,b} do
8:             Add (a, b, c) to Q
9: Return Q
```

For a_0, we consider all $b \in \{b_0, b_1, \ldots, b_m\}$. When $b = b_0$, we have

$$\pi_C(\sigma_{B=b_0} S) = \pi_C(\sigma_{A=a_0} T) = \{c_0, \ldots, c_m\},$$

so we output the $m+1$ triangles in total time $O(m)$. For the pairs (a_0, b_i) when $i \geqslant 1$, we have $|\sigma_{B=b_i} S| = 1$ and hence we spend $O(1)$ time on each such pair, for a total of $O(m)$ overall.

Now consider a_i for $i \geqslant 1$. In this case, $b = b_0$ is the only candidate. Further, for (a_i, b_0), we have $|\sigma_{A=a_i} T| = 1$, so we can handle each such a_i in $O(1)$ time leading to an overall run time of $O(m)$. Thus on this bad example Algorithm 2 runs in $O(N)$ time.

Appendix B has the full analysis of Algorithm 2: its worst-case runtime is exactly the same as that of Algorithm 1. What is remarkable is that both of these algorithms follow exactly the same recursive structure and they are special cases of a generic worst-case optimal join algorithm.

3 A User's Guide to the AGM bound

We now describe one way to generalize the bound of the output size of a join (mirroring the $O(N^{3/2})$ bound we saw for the triangle query) and illustrate its use with a few examples.

3.1 AGM Bound

To state the AGM bound, we need some notation. The natural join problem can be defined as follows. We are given a collection of m relations. Each relation is over a collection of attributes. We use $\mathcal{V}$ to denote the set of attributes; let $n = |\mathcal{V}|$. The join query Q is modeled as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where for each hyperedge $F \in \mathcal{E}$ there is a relation R_F on attribute set F. Figure 3 shows several example join queries, their associated hypergraphs, and illustrates the bounds below.

Atserias-Grohe-Marx [3] and Grohe-Marx [23] proved the following remarkable inequality, which shall be referred to as the *AGM's inequality* henceforth. Let $\mathbf{x} = (x_F)_{F \in \mathcal{E}}$ be *any* point in the following polyhedron:

$$\left\{ \mathbf{x} \mid \sum_{F : v \in F} x_F \geqslant 1, \forall v \in \mathcal{V}, \mathbf{x} \geqslant \mathbf{0} \right\}.$$

Such a point $\mathbf{x}$ is called a *fractional edge cover* of the hypergraph $\mathcal{H}$. Then, AGM's inequality states that the join size can be bounded by

$$|Q| = |\bowtie_{F \in \mathcal{E}} R_F| \leqslant \prod_{F \in \mathcal{E}} |R_F|^{x_F}. \tag{6}$$

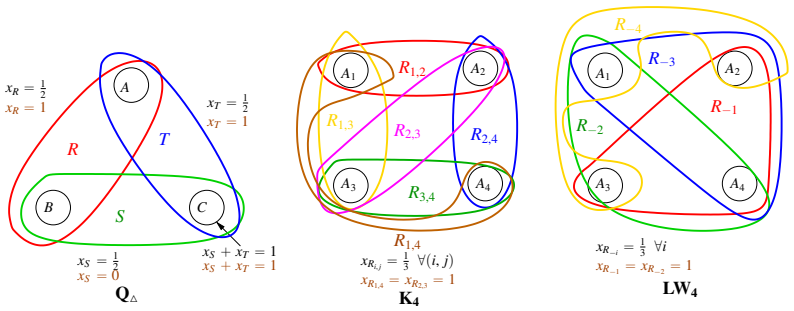

Figure 3: A handful of queries and their covers.

3.2 Example Bounds

We now illustrate the AGM bound on some specific join queries. We begin with the triangle query $Q_\triangle$. In this case the corresponding hypergraph $\mathcal{H}$ is as in the left part of Figure 3. We consider two covers (which are also marked in Figure 3). The first one is $x_R = x_T = x_S = \frac{1}{2}$. This is a valid cover since the required inequalities are satisfied for every vertex. For example, for vertex C, the two edges incident on it are S and T and we have $x_S + x_T = 1 \geqslant 1$ as required. In this case the bound (6) states that

$$|Q_\triangle| \leqslant \sqrt{|R| \cdot |S| \cdot |T|}. \tag{7}$$

Another valid cover is $x_R = x_T = 1$ and $x_S = 0$ (this cover is also marked in Figure 3). This is a valid cover, e.g. since for C we have $x_S + x_T = 1 \geqslant 1$ and for vertex A, we have $x_R + x_T = 2 \geqslant 1$ as required. For this cover, bound (6) gives

$$|Q_\triangle| \leqslant |R| \cdot |T|. \tag{8}$$

These two bounds can be better in different scenarios. E.g. when $|R| = |S| = |T| = N$, then (7) gives an upper bound of $N^{3/2}$ (which is the tight answer) while (8) gives a bound of N^2, which is worse. However, if $|R| = |T| = 1$ and $|S| = N$, then (7) gives a bound of $\sqrt{N}$, which has a lot of slack; while (8) gives a bound of 1, which is tight.

For another class of examples, consider the "clique" query. In this case there are $n \geqslant 3$ attributes and $m = \binom{n}{2}$ relations: one $R_{i,j}$ for every $i < j \in [n]$: we will call this query K_n. Note that K_3 is $Q_\triangle$. The middle part of Figure 3 considers the K_4 query. We highlight one cover: $x_{R_{i,j}} = \frac{1}{n-1}$ for every $i < j \in [n]$. This is a valid cover since every attribute is contained in $n-1$ relations. Further, in this case (6) gives a bound of $\sqrt[n-1]{\prod_{i<j} |R_{i,j}|}$, which simplifies to $N^{n/2}$ for the case when every relation has size N.

Finally, we consider the Loomis-Whitney LW_n queries. In this case there are n attributes and there are $m = n$ relations. In particular, for every $i \in [n]$ there is a relation $R_{-i} = R_{[n]\setminus\{i\}}$. Note that LW_3 is $Q_\triangle$. See the right of Figure 3 for the LW_4 query. We highlight one cover: $x_{R_{i,j}} = \frac{1}{n-1}$ for every $i < j \in [n]$. This is a valid cover since every attribute is contained in $n-1$ relations. Further, in this case (6) gives a bound of $\sqrt[n-1]{\prod_i |R_{-i}|}$, which simplifies to $N^{1+\frac{1}{n-1}}$ for the case when every relation has size N. Note that this bound approaches N as n becomes larger.

3.3 The Tightest AGM Bound

As we just saw, the optimal edge cover for the AGM bound depends on the relation sizes. To minimize the right hand side of (6), we can solve the following linear program:

$$\begin{aligned} \min \quad & \sum_{F \in \mathcal{E}} (\log_2 |R_F|) \cdot x_F \\ \text{s.t.} \quad & \sum_{F : v \in F} x_F \geqslant 1, v \in \mathcal{V} \\ & \mathbf{x} \geqslant \mathbf{0} \end{aligned}$$

Implicitly, the objective function above depends on the database instance $\mathcal{D}$ on which the query is applied. Let $\rho^*(Q, \mathcal{D})$ denote the optimal objective value to the above linear program. We refer to $\rho^*(Q, \mathcal{D})$ as the *fractional edge cover number* of the query Q with respect to the database instance $\mathcal{D}$, following Grohe [22]. The AGM's inequality can be summarized simply by $|Q| \leqslant 2^{\rho^*(Q,\mathcal{D})}$.

3.4 Applying AGM bound on conjunctive queries with simple functional dependencies

Thus far we have been describing bounds and algorithms for natural join queries. A super-class of natural join queries called *conjunctive queries*. A conjunctive query is a query of the form

$$C = R_0(\bar{X}_0) \leftarrow R_1(\bar{X}_1) \wedge \cdots \wedge R_m(\bar{X}_m)$$

where $\{R_1, \ldots, R_m\}$ is a multi-set of relation symbols, i.e. some relation might occur more than once in the query, and $\bar{X}_0, \ldots, \bar{X}_m$ are tuples of variables, and each variable occurring in the query's head $R(\bar{X}_0)$ must also occur in the body. It is important to note that the same variable might occur more than once in the same tuple $\bar{X}_i$.

We will use $\text{vars}(C)$ to denote the set of all variables occurring in C. Note that $\bar{X}_0 \subseteq \text{vars}(C)$ and it is entirely possible for $\bar{X}_0$ to be *empty* (Boolean conjunctive query). For example, the following are conjunctive queries:

$$\begin{aligned} R_0(WXYZ) &\leftarrow S(WXY) \wedge S(WWW) \wedge T(YZ) \\ R_0(Z) &\leftarrow S(WXY) \wedge S(WWW) \wedge T(YZ). \end{aligned}$$

The former query is a *full conjunctive query* because the head atom contains all the query's variable.

Following Gottlob, Lee, Valiant, and Valiant (GLVV hence forth) [15, 16], we also know that the AGM bound can be extended to general conjunctive queries even with simple functional dependencies.[4] In this survey, our presentation closely follows Grohe's presentation of GLVV [22].

To illustrate what can go "wrong" when we are moving from natural join queries to conjunctive queries, let us first consider a few example conjunctive queries, introducing one issue at a time. In all examples below, relations are assumed to have the same size N.

Example 3 (Projection)**.** Consider

$$C_1 = R_0(W) \leftarrow R(WX) \wedge S(WY) \wedge T(WZ).$$

In the (natural) join query, $R(WX) \wedge S(WY) \wedge T(WZ)$ AGM bound gives N^3; but because $R_0(W) \subseteq \pi_W(R) \bowtie \pi_W(S) \bowtie \pi_W(T)$, AGM bound can be adapted to the instance restricted only to the output variables yielding an upper bound of N on the output size.

[4] GLVV also have fascinating bounds for the general functional dependency and composite keys cases, and characterization of treewidth-preserving queries; both of those topics are beyond the scope of this survey, in particular because they require different machinery from what we have developed thus far.

Example 4 (Repeated variables)**.** Consider the query

$$C_2 = R_0(WY) \leftarrow R(WW) \wedge S(WY) \wedge T(YY).$$

This is a full conjunctive query as all variables appear in the head atom R_0. In this case, we can replace $R(WW)$ and $T(YY)$ by keeping only tuples $(t_1, t_2) \in R$ for which $t_1 = t_2$ and tuples $(t_1, t_2) \in T$ for which $t_1 = t_2$; essentially, we turn the query into a natural join query of the form $R'(W) \wedge S(WY) \wedge T'(Y)$. For this query, $x_{R'} = x_{T'} = 0$ and $x_S = 1$ is a fractional cover and thus by AGM bound N is an upperbound on the output size.

Example 5 (Introducing the chase)**.** Consider the query

$$C_3 = R_0(WXY) \leftarrow R(WX) \wedge R(WW) \wedge S(XY).$$

Without additional information, the best bound we can get for this query is $O(N^2)$: we can easily turn it into a natural join query of the form $R(WX) \wedge R'(W) \wedge S(XY)$, where R' is obtained from R by keeping all tuples $(t_1, t_2) \in R$ for which $t_1 = t_2$. Then, $(x_R, x_{R'}, x_S)$ is a fractional edge cover for this query if and only if $x_R + x_{R'} \geqslant 1$ (to cover W), $x_R + x_S \geqslant 1$ (to cover X), $x_S \geqslant 1$ (to cover Y); So, $x_S = x_{R'} = 1$ and $x_R = 0$ is a fractional cover, yielding the $O(N^2)$ bound. Furthermore, it is easy to construct input instances for which the output size is $\Omega(N^2)$:

$$\begin{aligned} R &= \{(i,i) \mid i \in [N/2]\} \bigcup \{(i,0) \mid i \in [N/2]\} \\ S &= \{(0,j) \mid j \in [N]\}. \end{aligned}$$

Every tuple $(i, 0, j)$ for $i \in [N/2]$, $j \in [N]$ is in the output.

Next, suppose we have an additional piece of information that the first attribute in relation R is its *key*, i.e. if (t_1, t_2) and (t_1, t_2') are in R, then $t_2 = t_2'$. Then we can significantly reduce the output size bound because we can infer the following about the output tuples: (w, x, y) is an output tuple iff (w, x) and (w, w) are in R, and (x, y) are in S. The functional dependency tells us that $x = w$. Hence, the query is equivalent to

$$C_3' = R_0(WY) \leftarrow R(WW) \wedge S(WY).$$

The AGM bound for this (natural) join query is N. The transformation from C_3 to C_3' we just described is, of course, the famous *chase* operation [2, 4, 30], which is much more powerful than what conveyed by this example.

Example 6 (Taking advantage of FDs)**.** Consider the following query

$$C_4 = R_0(XY_1, \ldots, Y_k, Z) \leftarrow \bigwedge_{i=1}^{k} R_i(XY_i) \wedge \bigwedge_{i=1}^{k} S_i(Y_i Z).$$

First, without any functional dependency, AGM bound gives N^k for this query, because the fractional cover constraints are

$$\begin{aligned} \sum_{i=1}^{k} x_{R_i} &\geqslant 1 \text{ (cover } X) \\ x_{R_i} + x_{S_i} &\geqslant 1 \text{ (cover } Y_i) \; i \in [k] \\ \sum_{i=1}^{k} x_{S_i} &\geqslant 1 \text{ (cover } Z). \end{aligned}$$

The AGM bound is $N^{\sum_i (x_{R_i} + x_{S_i})} \geqslant N^k$.

Second, suppose we know $k+1$ functional dependencies: each of the first attributes of relations $R_1, \ldots, R_k$ is a key for the corresponding relation, and the first attribute of S_1 is its key. Then, we have the following sets of functional dependencies: $X \to Y_i$, $i \in [k]$, and $Y_1 \to Z$. Now, construct a fictitious relation $R'(X, Y_1, \ldots, Y_k, Z)$ as follows: $(x, y_1, \ldots, y_k, z) \in R'$ iff $(x, y_i) \in R_i$ for all $i \in [k]$ and $(y_1, z) \in S_1$. Then, obviously $|R'| \leqslant N$. More importantly, the output does not change if we add R' to the body query C_4 to obtain a new conjunctive query C_4'. However, this time we can set $x_{R'} = 1$ and all other variables in the fractional cover to be 0 and obtain an upper bound of N.

We present a more formal treatment of the steps needed to convert a conjunctive query with simple functional dependencies to a join query in Appendix E.

4 Worst-case-optimal algorithms

We first show how to analyze the upper bound that proves AGM and from which we develop a generalized join algorithm that captures both algorithms from Ngo-Porat-Ré-Rudra [32] (henceforth NPRR) and Leapfrog Triejoin [42]. Then, we describe the limitation of any join-project plan.

Henceforth, we need the following notation. Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be any hypergraph and $I \subseteq \mathcal{V}$ be an arbitrary subset of vertices of $\mathcal{H}$. Then, we define

$$\mathcal{E}_I := \{F \in \mathcal{E} \mid F \cap I \neq \varnothing\}.$$

Example 7. For the query $Q_\triangle$ from Section 2, we have $\mathcal{H}_\triangle = (\mathcal{V}_\triangle, \mathcal{E}_\triangle)$, where

$$\begin{aligned} \mathcal{V}_\triangle &= \{A, B, C\}, \\ \mathcal{E}_\triangle &= \Big\{\{A, B\}, \{B, C\}, \{A, C\}\Big\}. \end{aligned}$$

Let $I_1 = \{A\}$ and $I_2 = \{A, B\}$, then $\mathcal{E}_{I_1} = \{\{A, B\}, \{A, C\}\}$, and $\mathcal{E}_{I_2} = \mathcal{E}_\triangle$.

4.1 A proof of the AGM bound

We prove the AGM inequality in two steps: a query decomposition lemma, and then a succinct inductive proof, which we then use to develop a generic worst-case optimal join algorithm.

4.1.1 The query decomposition lemma

Ngo-Porat-Ré-Rudra [32] gave an inductive proof of AGM bound (6) using Hölder inequality. (AGM proved the bound using an entropy based argument: see Appendix D for more details.) The proof has an inductive structure leading naturally to recursive join algorithms. NPRR's strategy is a generalization of the strategy in [6] to prove the Bollobás-Thomason inequality, shown in [32] to be *equivalent* to AGM's bound.

Implicit in NPRR is the following key lemma, which will be crucial in proving bounds on general join queries (as well as proving upper bounds on the runtime of the new join algorithms).

Lemma 4.1 (Query decomposition lemma)**.** *Let $Q = \bowtie_{F \in \mathcal{E}} R_F$ be a natural join query represented by a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, and $\mathbf{x}$ be any fractional edge cover for $\mathcal{H}$. Let $\mathcal{V} = I \uplus J$ be an arbitrary partition of $\mathcal{V}$ such that $1 \leqslant |I| < |\mathcal{V}|$; and,*

$$L = \bowtie_{F \in \mathcal{E}_I} \pi_I(R_F).$$

Then,

$$\sum_{\mathbf{t}_I \in L} \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F} \leqslant \prod_{F \in \mathcal{E}} |R_F|^{x_F}. \tag{9}$$

Before we prove the lemma above, we outline how we have already used the lemma above specialized to $Q_\triangle$ in Section 2 to bound the runtime of Algorithm 1. We use the lemma with $\mathbf{x} = (1/2, 1/2, 1/2)$, which is a valid fractional edge cover for $\mathcal{H}_\triangle$.

For Algorithm 1 we use Lemma 4.1 with $I = I_1$. Note that L in Lemma 4.1 is the same as

$$\pi_A(R) \bowtie \pi_A(T) = \pi_A(R) \cap \pi_A(T),$$

i.e. this L is exactly the same as the L in Algorithm 1. We now consider the left hand side (LHS) in (9). Note that we have $\mathcal{E}_J = \Big\{\{A, B\}, \{B, C\}, \{A, C\}\Big\}$. Thus, the LHS is the same as

$$\begin{aligned}
&\sum_{a \in L} \sqrt{|R \ltimes (a)|} \cdot \sqrt{|T \ltimes (a)|} \cdot \sqrt{|S \ltimes (a)|} \\
=&\sum_{a \in L} \sqrt{|\sigma_{A=a} R|} \cdot \sqrt{|\sigma_{A=a} T|} \cdot \sqrt{|S|}.
\end{aligned}$$

Note that the last expression is exactly the same as (4), which is at most $\sqrt{|R| \cdot |S| \cdot |T|}$ by Lemma 4.1. This was what shown in Section 2.

Proof of Lemma 4.1. The plan is to "unroll" the sum of products on the left hand side using Hölder inequality as follows. Let $j \in I$ be an arbitrary attribute. Define

$$\begin{aligned}
I' &= I - \{j\} \\
J' &= J \cup \{j\} \\
L' &= \bowtie_{F \in \mathcal{E}_{I'}} \pi_{I'}(R_F).
\end{aligned}$$

We will show that

$$\sum_{\mathbf{t}_I \in L} \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F} \leqslant \sum_{\mathbf{t}_{I'} \in L'} \prod_{F \in \mathcal{E}_{J'}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F}. \tag{10}$$

Then, by repeated applications of (10) we will bring I' down to empty and the right hand side is that of (9).

To prove (10) we write $\mathbf{t}_I = (\mathbf{t}_{I'}, t_j)$ for some $\mathbf{t}_{I'} \in L'$ and decompose a sum over L to a double sum over L' and t_j, where the second sum is only over t_j for which $(\mathbf{t}_{I'}, t_j) \in L$.

$$\begin{aligned}
\sum_{\mathbf{t}_I \in L} \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F} &= \sum_{\mathbf{t}_{I'} \in L'} \sum_{t_j} \prod_{F \in \mathcal{E}_J} |R_F \ltimes (\mathbf{t}_{I'}, t_j)|^{x_F} \\
&= \sum_{\mathbf{t}_{I'} \in L'} \sum_{t_j} \left(\prod_{F \in \mathcal{E}_J} |R_F \ltimes (\mathbf{t}_{I'}, t_j)|^{x_F} \right) \cdot \left(\prod_{F \in \mathcal{E}_{J'} - \mathcal{E}_J} 1^{x_F} \right) \\
&= \sum_{\mathbf{t}_{I'} \in L'} \sum_{t_j} \prod_{F \in \mathcal{E}_{J'}} |R_F \ltimes (\mathbf{t}_{I'}, t_j)|^{x_F} \\
&= \sum_{\mathbf{t}_{I'} \in L'} \prod_{F \in \mathcal{E}_{J'} - \mathcal{E}_{\{j\}}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F} \sum_{t_j} \prod_{F \in \mathcal{E}_{\{j\}}} |R_F \ltimes (\mathbf{t}_{I'}, t_j)|^{x_F} \\
&\leqslant \sum_{\mathbf{t}_{I'} \in L'} \prod_{F \in \mathcal{E}_{J'} - \mathcal{E}_{\{j\}}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F} \prod_{F \in \mathcal{E}_{\{j\}}} \left(\sum_{t_j} |R_F \ltimes (\mathbf{t}_{I'}, t_j)| \right)^{x_F} \\
&\leqslant \sum_{\mathbf{t}_{I'} \in L'} \prod_{F \in \mathcal{E}_{J'} - \mathcal{E}_{\{j\}}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F} \prod_{F \in \mathcal{E}_{\{j\}}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F} \\
&= \sum_{\mathbf{t}_{I'} \in L'} \prod_{F \in \mathcal{E}_{J'}} |R_F \ltimes \mathbf{t}_{I'}|^{x_F}.
\end{aligned}$$

In the above, the third equality follows from fact that for any $F \in \mathcal{E}_{J'} - \mathcal{E}_J$, we have $F \subseteq I' \cup \{j\}$. The first inequality is an application of Hölder inequality, which holds because $\sum_{F \in \mathcal{E}_{\{j\}}} x_F \geqslant 1$. The second inequality is true because the sum is only over t_j for which $(\mathbf{t}_{I'}, t_j) \in L$. □

It is quite remarkable that from the query decomposition lemma, we can prove AGM inequality (6), and describe and analyze two join algorithms succinctly.

4.1.2 An inductive proof of AGM inequality

Base case. In the base case $|\mathcal{V}| = 1$, we are computing the join of $|\mathcal{E}|$ unary relations. Let $\mathbf{x} = (x_F)_{F \in \mathcal{E}}$ be a fractional edge cover for this instance. Then,

$$\begin{aligned} |\bowtie_{F \in \mathcal{E}} R_F| &\leqslant \min_{F \in \mathcal{E}} |R_F| \\ &\leqslant \left(\min_{F \in \mathcal{E}} |R_F|\right)^{\sum_{F \in \mathcal{E}} x_F} \\ &= \prod_{F \in \mathcal{E}} \left(\min_{F \in \mathcal{E}} |R_F|\right)^{x_F} \\ &\leqslant \prod_{F \in \mathcal{E}} |R_F|^{x_F}. \end{aligned}$$

Inductive step. Now, assume $n = |\mathcal{V}| \geqslant 2$. Let $\mathcal{V} = I \uplus J$ be any partition of $\mathcal{V}$ such that $1 \leqslant |I| < |\mathcal{V}|$. Define $L = \bowtie_{F \in \mathcal{E}_I} \pi_I(R_F)$ as in Lemma 4.1. For each tuple $\mathbf{t}_I \in L$ we define a new join query

$$Q[\mathbf{t}_I] := \bowtie_{F \in \mathcal{E}_J} \pi_J(R_F \ltimes \mathbf{t}_I).$$

Then, obviously we can write the original query Q as

$$Q = \bigcup_{\mathbf{t}_I \in L} (\{\mathbf{t}_I\} \times Q[\mathbf{t}_I]). \tag{11}$$

The vector $(x_F)_{F \in \mathcal{E}_J}$ is a fractional edge cover for the hypergraph of $Q[\mathbf{t}_I]$. Hence, the induction hypothesis gives us

$$|Q[\mathbf{t}_I]| \leqslant \prod_{F \in \mathcal{E}_J} |\pi_J(R_F \ltimes \mathbf{t}_I)|^{x_F} = \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F}. \tag{12}$$

From (11), (12), and (9) we obtain AGM inequality:

$$|Q| = \sum_{\mathbf{t}_I \in L} |Q[\mathbf{t}_I]| \leqslant \prod_{F \in \mathcal{E}} |R_F|^{x_F}.$$

4.2 Worst-case optimal join algorithms

From the proof of Lemma 4.1 and the query decomposition (11), it is straightforward to design a class of recursive join algorithms which is optimal in the worst case. (See Algorithm 3.)

A mild assumption which is not very crucial is to pre-index all the relations so that the inputs to the sub-queries $Q[\mathbf{t}_I]$ can readily be available when the time comes to compute it. Both NPRR and Leapfrog Triejoin algorithms do this by fixing a global attribute order and build a B-tree-like index structure for each input relation consistent with this global attribute order. NPRR also described an hash-based indexing structure so as to remove a log-factor from the final run time. We will not delve on this point here, except to emphasize the fact that we do not include the linear time pre-processing step in the final runtime formula.

Algorithm 3 Generic-Join($\bowtie_{F\in\mathcal{E}} R_F$)

Input: Query Q, hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$

$Q \leftarrow \emptyset$
If $|\mathcal{V}| = 1$ **then**
    return $\bigcap_{F\in\mathcal{E}} R_F$
Pick I arbitrarily such that $1 \leqslant |I| < |\mathcal{V}|$
$L \leftarrow$ Generic-Join($\bowtie_{F\in\mathcal{E}_I} \pi_I(R_F)$)
For every $\mathbf{t}_I \in L$ **do**
    $Q[\mathbf{t}_I] \leftarrow$ Generic-Join($\bowtie_{F\in\mathcal{E}_J} \pi_J(R_F \ltimes \mathbf{t}_I)$)
    $Q \leftarrow Q \cup \{\mathbf{t}_I\} \times Q[\mathbf{t}_I]$
Return Q

Given the indices, when $|\mathcal{V}| = 1$ computing $\bigcap_{F\in\mathcal{E}} R_F$ can easily be done in time

$$\tilde{O}(m \min |R_F|) = \tilde{O}(m \prod_{F\in\mathcal{E}} |R_F|^{x_F}).$$

Then, given this base-case runtime guarantee, we can show by induction that the overall runtime of Algorithm 3 is $\tilde{O}(mn \prod_{F\in\mathcal{E}} |R_F|^{x_F})$, where $\tilde{O}$ hides a potential log-factor of the input size. This is because, by induction the time it takes to compute L is $\tilde{O}(m|I| \prod_{F\in\mathcal{E}_I} |R_F|^{x_F})$, and the time it takes to compute $Q[\mathbf{t}_I]$ is

$$\tilde{O}\left(m(n-|I|) \prod_{F\in\mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F}\right)$$

Hence, from Lemma 4.1, the total run time is $\tilde{O}$ of

$$\begin{aligned} & m|I| \prod_{F\in\mathcal{E}_I} |R_F|^{x_F} + m(n-|I|) \sum_{\mathbf{t}_I\in L} \prod_{F\in\mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F} \\ \leqslant\ & m|I| \prod_{F\in\mathcal{E}_I} |R_F|^{x_F} + m(n-|I|) \prod_{F\in\mathcal{E}} |R_F|^{x_F} \\ \leqslant\ & mn \prod_{F\in\mathcal{E}} |R_F|^{x_F}. \end{aligned}$$

The NPRR algorithm is an instantiation of Algorithm 3 where it picks $J \in \mathcal{E}$, $I = \mathcal{V} - J$, and solves the sub-queries $Q[\mathbf{t}_I]$ in a different way, making use of the power of two choices idea. Since $J \in \mathcal{E}$, we write

$$Q[\mathbf{t}_I] = R_J \bowtie (\bowtie_{F\in\mathcal{E}_J - \{J\}} \pi_J(R_F \ltimes \mathbf{t}_I)).$$

Now, if $x_J \geqslant 1$ then we solve for $Q[\mathbf{t}_I]$ by checking for every tuple in R_J whether it can be part of $Q[\mathbf{t}_I]$. The run time is $\tilde{O}$ of

$$(n-|I|)|R_J| \leqslant (n-|I|) \prod_{F\in\mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F}.$$

When $x_J < 1$, we will make use of an extremely simple observation: for any real numbers $p, q \geqslant 0$ and $z \in [0,1]$, $\min\{p,q\} \leqslant p^z q^{1-z}$ (note that (2) is the special case of $z = 1/2$). In particular, define

$$\begin{aligned} p &= |R_J| \\ q &= \prod_{F\in\mathcal{E}_J - \{J\}} |\pi_J(R_F \ltimes \mathbf{t}_I)|^{\frac{x_F}{1-x_J}} \end{aligned}$$

Then,

$$\begin{aligned}\min\{p,q\} &\leqslant |R_J|^{x_J} \prod_{F\in\mathcal{E}_J-\{J\}} |\pi_J(R_F \ltimes \mathbf{t}_I)|^{x_F} \\ &= \prod_{F\in\mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F}.\end{aligned}$$

From there, when $x_J < 1$ and $p \leqslant q$, we go through each tuple in R_J and check as in the case $x_J \geqslant 1$. And when $p > q$, we solve the subquery $\bowtie_{F\in\mathcal{E}_J-\{J\}} \pi_J(R_F \ltimes \mathbf{t}_I)$ first using $\left(\frac{x_F}{1-x_J}\right)_{F\in\mathcal{E}_J-\{J\}}$ as its fractional edge cover; and then check for each tuple in the result whether it is in R_J. In either case, the run time $\tilde{O}(\min\{p,q\})$ which along with the observation above completes the proof.

Next we outline how Algorithm 1 is Algorithm 3 with the above modification for NPRR for the triangle query $Q_\triangle$. In particular, we will use $\mathbf{x} = (1/2, 1/2, 1/2)$ and $I = \{A\}$. Note that this choice of I implies that $J = \{B, C\}$, which means in Step 5 Algorithm 3 computes

$$L = \pi_A(R) \bowtie \pi_A(T) = \pi_A(R) \cap \pi_A(T),$$

which is exactly the same L as in Algorithm 1. Thus, in the remaining part of Algorithm 3 one would cycle through all $a \in L$ (as one does in Algorithm 1). In particular, by the discussion above, since $x_S = 1/2 < 1$, we will try the best of two choices. In particular, we have

$$\begin{aligned}\bowtie_{F\in\mathcal{E}_J-\{J\}} \pi_J(R_F \ltimes (a)) &= \pi_B(\sigma_{A=a}R) \times \pi_C(\sigma_{A=a}T), \\ p &= |I|, \\ q &= |\sigma_{A=a}R| \cdot |\sigma_{A=a}T|.\end{aligned}$$

Hence, the NPRR algorithm described exactly matches Algorithm 1.

The Leapfrog Triejoin algorithm [42] is an instantiation of Algorithm 3 where $\mathcal{V} = [n]$ and $I = \{1, \ldots, n-1\}$ (or equivalently $I = \{1\}$). To illustrate, we outline how Algorithm 2 is Algorithm 3 with $I = \{A, B\}$ when specialized to $Q_\triangle$. Consider the run of Algorithm 3 on $\mathcal{H}_\triangle$, and the first time Step 4 is executed. The call to Generic-Join in Step 5 returns $L = \{(a,b) | a \in L_A, b \in L_B^a\}$, where L_A and L_B^a are as defined in Algorithm 2. The rest of Algorithm 3 is to do the following for every $(a,b) \in L$. $Q[(a,b)]$ is computed by the recursive call to Algorithm 3 to obtain $\{(a,b) \times L_C^{a,b}$, where

$$L_C^{a,b} = \pi_C\sigma_{B=b}S \bowtie \pi_C\sigma_{A=a}T = \pi_C\sigma_{B=b}S \cap \pi_C\sigma_{A=a}T,$$

exactly as was done in Algorithm 2. Finally, we get back to L in Step 5 being as claimed above. Note that in during the recursive call of Algorithm 3 on $Q_{\bowtie} = R \bowtie \pi_B(S) \bowtie \pi_A(T)$. The claim follows by picking $I = \{A\}$ in Step 4 when Algorithm 3 is run on $Q_{\bowtie}$ (and tracing through rest of Algorithm 3).

4.3 On the limitation of any join-project plan

AGM proved that there are classes of queries for which join-only plans are significantly worse than their join-project plan. In particular, they showed that for every $M, N \in \mathbb{N}$, there is a query Q of size at least M and a database $\mathcal{D}$ of size at least N such that $2^{\rho^*(Q,\mathcal{D})} \leqslant N^2$ and every join-only plan runs in time at least $N^{\frac{1}{5}\log_2|Q|}$.

NPRR continued with the story and noted that for the class of LW_n queries from Section 3.2 every join-project plan runs in time polynomially worse than the AGM bound. The proof of the following lemma can be found in Appendix F.

Lemma 4.2. *Let $n \geqslant 2$ be an arbitrary integer. For any LW-query Q with corresponding hypergraph $\mathcal{H} = ([n], \binom{[n]}{n-1})$, and any positive integer $N \geqslant 2$, there exist n relations R_i, $i \in [n]$ such that $|R_i| = N, \forall i \in [n]$, the attribute set for R_i is $[n] - \{i\}$, and that* any *join-project plan for Q on these relations has run-time at least $\Omega(N^2/n^2)$.*

Note that both the traditional join-tree-based algorithms and AGM's algorithm described in Appendix D.2 are join-project plans. Consequently, they run in time asymptotically worse than the best AGM bound for this instance, which is

$$|\bowtie_{i=1}^{n} R_i| \leqslant \prod_{i=1}^{n} |R_i|^{1/(n-1)} = N^{1+1/(n-1)}.$$

On the other hand, both algorithms described in Section 4.2 take $O(N^{1+1/(n-1)})$-time because their run times match the AGM bound. In fact, the NPRR algorithm in Section 4.2 can be shown to run in linear data-complexity time $O(n^2N)$ for this query [32].

5 Open Questions

We conclude this survey with two open questions: one for systems researchers and one for theoreticians:

1. A natural question to ask is whether the algorithmic ideas that were presented in this survey can gain runtime efficiency in databases systems. This is an intriguing open question: on one hand we have shown asymptotic improvements in join algorithms, but on the other there are several decades of engineering refinements and research contributions in the traditional dogma.

2. Worst-case results, as noted by several authors, may only give us information about pathological instances. Thus, there is a natural push toward more refined measures of complexity. For example, current complexity measures are too weak to explain why indexes are used or give insight into the average case. For example, could one design an adaptive join algorithm whose run time is somehow dictated by the "difficulty" of the input instance (instead of the input size as in the currently known results)?

Acknowledgements

HN's work is partly supported by NSF grant CCF-1319402 and a gift from Logicblox. CR's work on this project is generously supported by NSF CAREER Award under No. IIS-1353606, NSF award under No. CCF-1356918, the ONR under awards No. N000141210041 and No. N000141310129, Sloan Research Fellowship, Oracle, and Google. AR's work is partly supported by NSF CAREER Award CCF-CCF-0844796, NSF grant CCF-1319402 and a gift from Logicblox.

References

[1] Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Aho, A. V., Beeri, C., and Ullman, J. D. The theory of joins in relational databases. *ACM Trans. Database Syst. 4*, 3 (1979), 297–314.

[3] Atserias, A., Grohe, M., and Marx, D. Size bounds and query plans for relational joins. In *FOCS* (2008), IEEE Computer Society, pp. 739–748.

[4] Beeri, C., and Vardi, M. Y. A proof procedure for data dependencies. *J. ACM 31*, 4 (1984), 718–741.

[5] Blanas, S., Li, Y., and Patel, J. M. Design and evaluation of main memory hash join algorithms for multi-core cpus. In *SIGMOD* (2011), ACM, pp. 37–48.

[6] Bollobás, B., and Thomason, A. Projections of bodies and hereditary properties of hypergraphs. *Bull. London Math. Soc. 27*, 5 (1995), 417–424.

[7] Chandra, A. K., and Merlin, P. M. Optimal implementation of conjunctive queries in relational data bases. In *STOC* (1977), J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, Eds., ACM, pp. 77–90.

[8] Chaudhuri, S. An overview of query optimization in relational systems. In *PODS* (1998), ACM, pp. 34–43.

[9] Chekuri, C., and Rajaraman, A. Conjunctive query containment revisited. *Theor. Comput. Sci. 239*, 2 (2000), 211–229.

[10] Chung, F. R. K., Graham, R. L., Frankl, P., and Shearer, J. B. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A 43*, 1 (1986), 23–37.

[11] Cover, T. M., and Thomas, J. A. *Elements of information theory*, second ed. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, 2006.

[12] DeWitt, D. J., Naughton, J. F., Schneider, D. A., and Seshadri, S. Practical skew handling in parallel joins. In *Proceedings of the 18th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1992), VLDB '92, Morgan Kaufmann Publishers Inc., pp. 27–40.

[13] Fagin, R. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM 30*, 3 (1983), 514–550.

[14] Friedgut, E., and Kahn, J. On the number of copies of one hypergraph in another. *Israel J. Math. 105* (1998), 251–256.

[15] Gottlob, G., Lee, S. T., and Valiant, G. Size and treewidth bounds for conjunctive queries. In *PODS* (2009), J. Paredaens and J. Su, Eds., ACM, pp. 45–54.

[16] Gottlob, G., Lee, S. T., Valiant, G., and Valiant, P. Size and treewidth bounds for conjunctive queries. *J. ACM 59*, 3 (2012), 16.

[17] Gottlob, G., Leone, N., and Scarcello, F. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci. 64*, 3 (2002), 579–627.

[18] Gottlob, G., Leone, N., and Scarcello, F. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci. 66*, 4 (2003), 775–808.

[19] Gottlob, G., Miklós, Z., and Schwentick, T. Generalized hypertree decompositions: Np-hardness and tractable variants. *J. ACM 56*, 6 (2009).

[20] Graefe, G. Query evaluation techniques for large databases. *ACM Computing Surveys 25*, 2 (June 1993), 73–170.

[21] Graham, M. H. On the universal relation, 1980. Tech. Report.

[22] Grohe, M. Bounds and algorithms for joins via fractional edge covers. `http://www.automata.rwth-aachen.de/~grohe/pub/gro12+.pdf`, 2012. Manuscript.

[23] Grohe, M., and Marx, D. Constraint solving via fractional edge covers. In *SODA* (2006), ACM Press, pp. 289–298.

[24] Gyssens, M., Jeavons, P., and Cohen, D. A. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell. 66*, 1 (1994), 57–89.

[25] Gyssens, M., and Paredaens, J. A decomposition methodology for cyclic databases. In *Advances in Data Base Theory* (1982), pp. 85–122.

[26] Hardy, G. H., Littlewood, J. E., and Pólya, G. *Inequalities*. Cambridge University Press, Cambridge, 1988. Reprint of the 1952 edition.

[27] Kim, C., Kaldewey, T., Lee, V. W., Sedlar, E., Nguyen, A. D., Satish, N., Chhugani, J., Di Blas, A., and Dubey, P. Sort vs. hash revisited: fast join implementation on modern multi-core cpus. *Proc. VLDB Endow. 2*, 2 (Aug. 2009), 1378–1389.

[28] Kolaitis, P. G., and Vardi, M. Y. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci. 61*, 2 (2000), 302–332.

[29] Loomis, L. H., and Whitney, H. An inequality related to the isoperimetric inequality. *Bull. Amer. Math. Soc 55* (1949), 961–962.

[30] Maier, D., Mendelzon, A. O., and Sagiv, Y. Testing implications of data dependencies. *ACM Trans. Database Syst. 4*, 4 (Dec. 1979), 455–469.

[31] Marx, D. Approximating fractional hypertree width. *ACM Trans. Algorithms 6*, 2 (Apr. 2010), 29:1–29:17.

[32] Ngo, H. Q., Porat, E., Ré, C., and Rudra, A. Worst-case optimal join algorithms: [extended abstract]. In *PODS* (2012), pp. 37–48.

[33] Papadimitriou, C. H., and Yannakakis, M. On the complexity of database queries. In *PODS* (1997), A. O. Mendelzon and Z. M. Özsoyoglu, Eds., ACM Press, pp. 12–19.

[34] Radhakrishnan, J. Entropy and counting. *Computational Mathematics, Modelling and Algorithms* (2003), 146.

[35] Ramakrishnan, R., and Gehrke, J. *Database Management Systems*, 3 ed. McGraw-Hill, Inc., New York, NY, USA, 2003.

[36] Robertson, N., and Seymour, P. D. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms 7*, 3 (1986), 309–322.

[37] Scarcello, F. Query answering exploiting structural properties. *SIGMOD Record 34*, 3 (2005), 91–99.

[38] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1979), SIGMOD '79, ACM, pp. 23–34.

[39] Suri, S., and Vassilvitskii, S. Counting triangles and the curse of the last reducer. In *WWW* (2011), pp. 607–614.

[40] Tsourakakis, C. E. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *ICDM* (2008), IEEE Computer Society, pp. 608–617.

[41] Vardi, M. Y. The complexity of relational query languages (extended abstract). In *STOC* (1982), H. R. Lewis, B. B. Simons, W. A. Burkhard, and L. H. Landweber, Eds., ACM, pp. 137–146.

[42] Veldhuizen, T. L. Leapfrog Triejoin: a worst-case optimal join algorithm. *ArXiv e-prints* (Oct. 2012).

[43] Walton, C. B., Dale, A. G., and Jenevein, R. M. A taxonomy and performance model of data skew effects in parallel joins. In *Proceedings of the 17th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1991), VLDB '91, Morgan Kaufmann Publishers Inc., pp. 537–548.

[44] Xu, Y., Kostamaa, P., Zhou, X., and Chen, L. Handling data skew in parallel joins in shared-nothing systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2008), SIGMOD '08, ACM, pp. 1043–1052.

[45] Yannakakis, M. Algorithms for acyclic database schemes. In *VLDB* (1981), IEEE Computer Society, pp. 82–94.

[46] Yu, C., and Ozsoyoglu, M. On determining tree-query membership of a distributed query. *Informatica 22*, 3 (1984), 261–282.

A Relation Algebra Notation

We assume the existence of a set of attribute names $\mathcal{A} = A_1, \ldots, A_n$ with associated domains $\mathbf{D}_1, \ldots, \mathbf{D}_n$ and infinite set of relational symbols $\mathcal{R}$. A relational schema for the symbol $R \in \mathcal{R}$ of arity k is a tuple $\bar{A}(R) = (A_{i_1}, \ldots, A_{i_k})$ of distinct attributes that defines the attributes of the relation. A relational database schema is a set of relational symbols and associated schemas denoted by $R(\bar{A}(R)), R \in \mathcal{R}$. A relational instance for $R(A_{i_1}, \ldots, A_{i_k})$ is a subset of $\mathbf{D}_{i_1} \times \cdots \times \mathbf{D}_{i_k}$. A relational database $\mathcal{D}$ is a collection of instances, one for each relational symbol in schema, denoted by $R^{\mathcal{D}}$.

A *natural join* query (or simply join query) Q is specified by a finite subset of relational symbols $\text{atoms}(Q) \subseteq \mathcal{R}$, denoted by $\bowtie_{R \in \text{atoms}(Q)} R$. Let $\bar{A}(Q)$ denote the set of all attributes that appear in some relation in Q, that is $\bar{A}(Q) = \{A \mid A \in \bar{A}(R) \text{ for some } R \in \text{atoms}(Q)\}$. Given a tuple $\mathbf{t}$ we will write $\mathbf{t}_{\bar{A}}$ to emphasize that its support is the attribute set $\bar{A}$. Further, for any $\bar{S} \subset \bar{A}$ we let $\mathbf{t}_{\bar{S}}$ denote $\mathbf{t}$ restricted to $\bar{S}$. Given a database instance $\mathcal{D}$, the output of the query Q on the database instance $\mathcal{D}$ is denoted $Q(\mathcal{D})$ and is defined as

$$Q(\mathcal{D}) \stackrel{\text{def}}{=} \left\{\mathbf{t} \in \mathbf{D}^{\bar{A}(Q)} \mid \mathbf{t}_{\bar{A}(R)} \in R^{\mathcal{D}} \text{ for each } R \in \text{atoms}(Q)\right\}$$

where $\mathbf{D}^{\bar{A}(Q)}$ is a shorthand for $\times_{i:A_i \in \bar{A}(Q)} \mathbf{D}_i$. When the instance is clear from the context we will refer to $Q(\mathcal{D})$ by just Q.

We also use the notion of a *semijoin*: Given two relations $R(\bar{A})$ and $S(\bar{B})$ their semijoin $R \ltimes S$ is defined by

$$R \ltimes S \stackrel{\text{def}}{=} \{\mathbf{t} \in R : \exists \mathbf{u} \in S \text{ s.t. } \mathbf{t}_{\bar{A} \cap \bar{B}} = \mathbf{u}_{\bar{A} \cap \bar{B}}\}.$$

For any relation $R(\bar{A})$, and any subset $\bar{S} \subseteq \bar{A}$ of its attributes, let $\pi_{\bar{S}}(R)$ denote the *projection* of R onto $\bar{S}$, i.e.

$$\pi_{\bar{S}}(R) = \left\{\mathbf{t}_{\bar{S}} \mid \exists \mathbf{t}_{\bar{A} \setminus \bar{S}}, (\mathbf{t}_{\bar{S}}, \mathbf{t}_{\bar{A} \setminus \bar{S}}) \in R\right\}.$$

For any relation $R(\bar{A})$, any subset $\bar{S} \subseteq \bar{A}$ of its attributes, and a vector $\mathbf{s} \in \prod_{i \in \bar{S}} \mathbf{D}_i$, let $\sigma_{\bar{S}=\mathbf{s}}(R)$ denote the *selection* of R with $\bar{S}$ and $\mathbf{s}$, i.e.

$$\sigma_{\bar{S}=\mathbf{s}}(R) = \{\mathbf{t} \mid \mathbf{t}_{\bar{S}} = \mathbf{s}\}.$$

B Analysis of Algorithm 2

It turns out that the run time of Algorithm 2 is dominated by the time spent in computing the set $L_C^{a,b}$ for every $a \in L_A$ and $b \in L_B^a$. We now analyze this time "inside out." We first note that for a given $a \in L_A$ and $b \in L_B^a$, the time it takes to compute $L_C^{a,b}$ is at most

$$\min\{|\pi_C\sigma_{B=b}S|, |\pi_C\sigma_{A=a}T|\} \leqslant \sqrt{|\pi_C\sigma_{B=b}S| \cdot |\pi_C\sigma_{A=a}T|},$$

where the inequality follows from (2). We now go one level up and sum the run time over all $b \in L_B^a$ (but for the same fixed $a \in L_A$ as above). This leads to the time taken to compute $L_C^{a,b}$ over all $b \in L_B^a$ to be upper bounded by

$$\begin{aligned}
&\sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sum_{b\in L_B^a} \sqrt{|\pi_C\sigma_{B=b}S|} \\
\leqslant &\sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sqrt{|L_B^a|} \cdot \sqrt{\sum_{b\in L_B^a} |\pi_C\sigma_{B=b}S|} \\
\leqslant &\sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sqrt{|L_B^a|} \cdot \sqrt{\sum_{b\in \pi_B S} |\pi_C\sigma_{B=b}S|} \\
= &\sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sqrt{|L_B^a|} \cdot \sqrt{|S|} \\
= &\sqrt{|S|} \cdot \sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sqrt{|L_B^a|} \\
\leqslant &\sqrt{|S|} \cdot \sqrt{|\pi_C\sigma_{A=a}T|} \cdot \sqrt{|\pi_B\sigma_{A=a}R|},
\end{aligned}$$

where the first inequality follows from (3) (where we have $L = L_B^a$ and the vectors are the all 1's vector and the vector $(\sqrt{|\pi_C\sigma_{B=b}S|})_{b\in L_B^a}$). The second inequality follows from the fact that $L_B^a \subseteq \pi_B S$. The final inequality follows from the fact that $L_B^a \subseteq \pi_B\sigma_{A=a}R$.

To complete the runtime analysis, we need to sum up the last expression above for every $a \in L_A$. However, note that this sum is exactly the expression (5), which we have already seen is $N^{3/2}$, as desired.

For completeness, we show how the analysis of Algorithm 2 above follows directly from Lemma 4.1. In this case we use Lemma 4.1 with $I = I_2$, which implies

$$L = R \bowtie \pi_B S \bowtie \pi_A T = \{(a,b) | a \in L_A, b \in L_B^a\},$$

where L_A and L_B^a is as defined in Algorithm 2. Note that in this case $\mathcal{E}_J = \{\{B,C\},\{A,C\}\}$. Thus, we have that the LHS in (9) is the same as

$$\begin{aligned}
&\sum_{a\in L_A}\sum_{b\in L_B^a} \sqrt{|S \ltimes (a,b)|} \cdot \sqrt{|T \ltimes (a,b)|} \\
= &\sum_{a\in L_A}\sum_{b\in L_B^a} \sqrt{|\pi_C\sigma_{B=b}S|} \cdot \sqrt{|\pi_C\sigma_{A=a}T|}.
\end{aligned}$$

Note that the last expression is the same as the one in (13). Lemma 4.1 argues that the above is at most $N^{3/2}$, which is exactly what we proved above.

C Technical Tools

The following form of Hölder's inequality (also historically attributed to Jensen) can be found in any standard texts on inequalities. The reader is referred to the classic book "Inequalities" by Hardy, Littlewood, and Pólya [26] (Theorem 22 on page 29).

Lemma C.1 (Hölder inequality)**.** *Let m, n be positive integers. Let $y_1, \ldots, y_n$ be non-negative real numbers such that $y_1 + \cdots + y_n \geqslant 1$. Let $a_{ij} \geqslant 0$ be non-negative real numbers, for $i \in [m]$ and $j \in [n]$. With the convention $0^0 = 0$, we have:*

$$\sum_{i=1}^{m} \prod_{j=1}^{n} a_{ij}^{y_j} \leqslant \prod_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} \right)^{y_j}. \tag{13}$$

D Entropy and Alternate Derivations

D.1 Entropy and Shearer's inequality

For basic background knowledge on information theory and the entropy function in particular, the reader is referred to Thomas and Cover [11]. We are necessarily brief in this section.

Let X be a discrete random variable taking on values from a domain $\mathcal{X}$ with probability mass function p_X. The (binary) *entropy function* $H[X]$ is a measure of the degree of uncertainty associated with X, and is defined by

$$H[X] := -\sum_{x \in \mathcal{X}} p_X(x) \log_2 p_X(x).$$

We can replace X by a tuple $\mathbf{X} = (X_1, \ldots, X_n)$ of random variables and define the *joint entropy* in exactly the same way. The only difference in the formula above is the replacement of p_X by the joint probability mass function.

Let X, Y be two discrete random variables on domains $\mathcal{X}, \mathcal{Y}$, respectively. The *conditional entropy function* of X given Y measures the degree of uncertainty about X given that we knew Y:

$$H[X \mid Y] := -\sum_{\substack{x \in \mathcal{X} \\ y \in \mathcal{Y}}} \mathrm{P}[X = x, Y = y] \log_2 \mathrm{P}[X = x \mid Y = y].$$

Again, we extend the above definition in the natural way when X and Y are replaced by tuples of random variables. Many simple relations regarding entropy and conditional entropy can be derived from first principle, and they often have very intuitive interpretation. For example, the inequality

$$H[X \mid Y, Z] \leqslant H[X \mid Y]$$

can be intuitively "explained" by thinking that knowing less (only Y as opposed to knowing both Y and Z) leads to more uncertainty about X. Similarly, the following formula can easily be derived from first principles.

$$H[X_1, \ldots, X_n] = \sum_{j=1}^{n} H[H_j \mid X_1, \ldots, X_{j-1}]. \tag{14}$$

The above basic observation can be used to prove Shearer's inequality below. Our proof here follows the conditional entropy approach from Radhakrishnan [34]. The version of Shearer's lemma below is slightly more direct than the version used in [3, 22], leading to a shorter proof of AGM's inequality in Section D.2.

Lemma D.1 (Shearer [10]). *Let $X_1, \ldots, X_n$ be n random variables. For each subset $F \subseteq [n]$, let $\mathbf{X}_F = (X_i)_{i \in F}$; and, let $\mathbf{X} = X_{[n]}$. Let $\mathcal{H} = (\mathcal{V} = [n], \mathcal{E})$ be a hypergraph, and $\mathbf{x} = (x_F)_{F \in \mathcal{E}}$ be any fractional edge cover of the hypergraph. Then, the following inequality holds*

$$H[\mathbf{X}] \leqslant \sum_{F \in \mathcal{E}} x_F \cdot H[\mathbf{X}_F] \tag{15}$$

Proof. For every $F \in \mathcal{E}$ and $j \in F$, we have

$$H[X_j \mid X_i, i < j] \leqslant H[X_j \mid X_i, i < j, i \in F]$$

because the entropy on left hand side is conditioned on a (perhaps non-strict) superset of the variables conditioned on the right hand side. Additionally, because the vector $\mathbf{x} = (x_F)_{F \in \mathcal{E}}$ is a fractional edge cover, for every $j \in [n]$ we have $\sum_{F \in \mathcal{E}, j \in F} x_F \geqslant 1$. It follows that, for every $j \in [n]$,

$$H[X_j \mid X_i, i < j] \leqslant \sum_{F \in \mathcal{E}, j \in F} x_F \cdot H[X_j \mid X_i, i < j, i \in F].$$

From this inequality and formula (14), we obtain

$$\begin{aligned} H[\mathbf{X}] &= \sum_{j=1}^{n} H[X_j \mid X_i, i < j] \\ &\leqslant \sum_{j=1}^{n} \sum_{F \in \mathcal{E}, j \in F} x_F \cdot H[X_j \mid X_i, i < j, i \in F] \\ &= \sum_{F \in \mathcal{E}} x_F \cdot \sum_{j \in F} H[X_j \mid X_i, i < j, i \in F] \\ &= \sum_{F \in \mathcal{E}} x_F \cdot H[\mathbf{X}_F]. \end{aligned}$$

□

D.2 AGM's proof based on Shearer's entropy inequality and a join-project plan

AGM's inequality was shown in [3, 23] as follows. A similar approach was used in [14] to prove an essentially equivalent inequality regarding the number of copies of a hypergraph in another. Let $\mathbf{X} = (X_v)_{v \in \mathcal{V}}$ denote a uniformly chosen random tuple from the output of the query Q. Then, $H[\mathbf{X}] = \log_2 |Q|$. Note that each X_v, $v \in \mathcal{V}$ is a random variable. For each $F \in \mathcal{E}$, let $\mathbf{X}_F = (X_v)_{v \in F}$. Then, the random variables $\mathbf{X}_F$ takes on values in R_F. Because the uniform distribution has the maximum entropy, we have $H[\mathbf{X}_F] \leqslant \log_2 |R_F|$, for all $F \in \mathcal{E}$.

Let $\mathbf{x}$ denote any fractional edge cover for the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ of the query Q, then from Shearer's inequality (15) we obtain

$$\log_2 |Q| = H[\mathbf{X}] \leqslant \sum_{F \in \mathcal{E}} x_F \cdot H[\mathbf{X}_F] \leqslant \sum_{F \in \mathcal{E}} x_F \cdot \log_2 |R_F|,$$

which is exactly AGM's inequality (6).

Proposition D.2. *For any query Q, there is a join-project plan with runtime $O(|Q|^2 \cdot 2^{\rho^*(Q,\mathcal{D})} \cdot N)$ for evaluating Q, where N is the input size.*

Proof. We define the join-project plan recursively. Let $A_1, \ldots, A_n$ be the attributes and $R_1, \ldots, R_m$ be the relations of Q. Let $\bar{B}_{n-1} = (A_1, \ldots, A_{n-1})$. We first recursively compute the join $P = \bowtie_{F \in \mathcal{E}} \pi_{\bar{B}_{n-1}}(R_F)$. Then, we output

$$Q = (\cdots((P \bowtie \pi_{A_n}(R_1)) \bowtie \pi_{A_n}(R_2)) \bowtie \cdots \bowtie \pi_{A_n}(R_m)).$$

The base case is when $n = 1$ which is the simple m set intersection problem. It is easy to see that $\rho^*(P, \mathcal{D}) \leqslant \rho^*(Q, \mathcal{D})$, because any fractional edge cover for Q is also a fractional edge cover for P. Hence, all the intermediate results in computing Q from P has sizes at most $|P| \cdot N \leqslant 2^{\rho^*(P,\mathcal{D})} N \leqslant 2^{\rho^*(Q,\mathcal{D})} \cdot N$. From there, the claimed run-time follows. □

E A more formal description of GLVV results

From examples in Section 3.4, we can describe GLVV's strategy for obtaining size bounds for general conjunctive queries with simple functional dependencies. Let C be such query. The general idea is to construct a natural join query Q such that $|C| \leqslant |Q|$ and thus we can simply apply AGM bound on Q.

- The first step is to turn the conjunctive query C into $C_1 = \text{chase}(C)$.

  This step can be done in $O(|C|^4)$-time [1, 2, 4, 30]. This is the step in the spirit of Example 5.

- Next, let C_2 be obtained from C_1 by replacing each repeated relation symbol by a fresh relation symbol. Thus, in C_2 there is no duplicate relation. (In the join algorithm itself, all we have to do is to have them point to the same underlying index.)

- The next step is in the spirit of Example 6. Recall that a simple functional dependency (FD) is of the form $R[i] \to R[j]$, which means the jth attribute of R is functionally determined by the ith attribute. From the set of given simple FDs, we can obtain a set of FDs in terms of the variables in the query C_2. For example, if $R[3] \to R[1]$ is a given FD and $R(XYZW)$ occurs in the body of the query C_2, then we obtain a FD of the form $Z \to X$. Now, WLOG assume that $\text{vars}(C_2) = \{X_1, \ldots, X_n\}$. We repeatedly apply the following steps for each (variable) FD of the form $X_i \to X_j$:

  - For every atom R in which X_i appears but X_j does not, add X_j to R. (This is a fictitious relation – and in the join algorithm we do not need to physically add a new field to the data; rather, we keep a pointer to where X_j can be found when needed.)
  - For every FD of the form $X_k \to X_i$, we add $X_k \to X_j$ as a new FD.
  - Remove the FD $X_i \to X_j$.

  It is easy to see that performing the above in a canonical order will terminate in about $O(n^2)$ time, and the new query C_3 (with some fictitious relations "blown up" with more attributes) is exactly equivalent to the the old query C_2.

- Next, we can remove repeated variables from relations by keeping only tuples that match the repetition patterns. For example, if there is an atom $R(XXY)$ in the query C_3, then we only have to keep tuples (t_1, t_2, t_3) in R for which $t_1 = t_2$. After this step, we can replace R by a relation $R'(XY)$. This step is in the spirit of Example 4. We call the resulting query C_4.

- Finally, we turn C_4 into Q by "projecting out" all the attributes not in the head atom as was shown in Example 3. Since Q is now a join query, AGM bounds applies.

What is remarkable is that GLVV showed that the bound obtained this way is essentially tight up to a factor which is data-independent.

Note again that our description above is essentially what was obtained from GLVV, without the coloring number machinery. Furthermore, if C was a full conjunctive query, then we don't have to project away any attribute and thus any worst-case join algorithm described in the previous section can be applied which is still worst-case optimal!

F Proof of Lemma 4.2

Proof of Lemma 4.2. In the instances below the domain of any attribute is

$$\mathcal{D} = \{0, 1, \ldots, (N-1)/(n-1)\},$$

where we ignore the integrality issue for the sake of clarify. For $i \in [n]$, let R_i denote the set of *all* tuples in $\mathcal{D}^{[n]-\{i\}}$ each of which has at most one non-zero value. It follows that, for all $i \in [n]$,

$$|R_i| = (n-1)[(N-1)/(n-1)+1] - (n-2) = N,$$

and that

$$\begin{aligned} |\bowtie_{i=1}^{n} R_i| &= n[(N-1)/(n-1)+1] - (n-1) \\ &= N + (N-1)/(n-1). \end{aligned}$$

(We remark that the instance above was specialized to $n = 3$ in Section 2.)

A relation R on attribute set $\bar{A} \subseteq [n]$ is called "simple" if R is the set of *all* tuples in $\mathcal{D}^{\bar{A}}$ each of which has at most one non-zero value. Then, we observe the following properties.

(a) The input relations R_i are simple.

(b) An arbitrary projection of a simple relation is simple.

(c) Let S and T be any two simple relations on attribute sets $\bar{A}_S$ and $\bar{A}_T$, respectively. If $\bar{A}_S$ is contained in $\bar{A}_T$ or vice versa, then $S \bowtie T$ is simple. If neither $\bar{A}_S$ nor $\bar{A}_T$ is contained in the other, then $|S \bowtie T| \geqslant (1 + (N-1)/(n-1))^2 = \Omega(N^2/n^2)$.

For an arbitrary join-project plan starting from the simple relations R_i, we eventually must join two relations whose attribute sets are not contained in one another, which right then requires $\Omega(N^2/n^2)$ run time. □