# Worst-case Optimal Join Algorithms

HUNG Q. NGO, University at Buffalo, SUNY
ELY PORAT, Bar-Ilan University
CHRISTOPHER RÉ, Stanford University
ATRI RUDRA, University at Buffalo, SUNY

Efficient join processing is one of the most fundamental and well-studied tasks in database research. In this work, we examine algorithms for natural join queries over many relations and describe a new algorithm to process these queries optimally in terms of worst-case data complexity. Our result builds on recent work by Atserias, Grohe, and Marx, who gave bounds on the size of a natural join query in terms of the sizes of the individual relations in the body of the query. These bounds, however, are not constructive: they rely on Shearer's entropy inequality, which is information-theoretic. Thus, the previous results leave open the question of whether there exist algorithms whose runtimes achieve these optimal bounds. An answer to this question may be interesting to database practice, as we show in this article that any project-join style plans, such as ones typically employed in a relational database management system, are asymptotically slower than the optimal for some queries. We present an algorithm whose runtime is worst-case optimal for all natural join queries. Our result may be of independent interest, as our algorithm also yields a constructive proof of the general fractional cover bound by Atserias, Grohe, and Marx without using Shearer's inequality. This bound implies two famous inequalities in geometry: the Loomis-Whitney inequality and its generalization, the Bollobás-Thomason inequality. Hence, our results algorithmically prove these inequalities as well. Finally, we discuss how our algorithm can be used to evaluate full conjunctive queries optimally, to compute a relaxed notion of joins and to optimally (in the worst-case) enumerate all induced copies of a fixed subgraph inside of a given large graph.

CCS Concepts: • **Information systems → Relational database model**; • **Theory of computation → Database query processing and optimization (theory)**;

Additional Key Words and Phrases: Join Algorithms, fractional cover bound, Loomis-Whitney inequality, Bollobás-Thomason inequality



A preliminary version of this article was presented at PODS'12 as Reference [62]. We thank Georg Gottlob for sending us a full version of his work [30]. We thank XuanLong Nguyen for introducing us to the Loomis-Whitney inequality. We thank Dung Nguyen for catching some errors in the earlier statement of our algorithm. We thank the anonymous PODS'12 and JACM referees for many helpful comments that have greatly improved the presentation of the article. H.N.'s work is partly supported by NSF Grants No. CCF-1161196 and No. CCF-1319402. C.R. acknowledges the National Science Foundation (NSF) under CAREER Awards No. IIS-1353606 and No. CCF-1356918, the Office of Naval Research (ONR) under Awards No. N000141210041 and No. N000141310129, the Sloan Research Fellowship, the Moore Foundation Data Driven Investigator award, and gifts from American Family Insurance, Google, Lightspeed Ventures, and Toshiba. A.R.'s work on this project is supported by NSF Grants No. CCF-0844796 and No. CCF-1319402.
Authors' addresses: H. Q. Ngo and A. Rudra, 338 Davis Hall, University at Buffalo, Buffalo, NY, 14214. USA; emails: {hungngo, atri}@buffalo.edu; E. Porat, Bar-Ilan University, Ramat-Gan, 5290002 Israel; email: porately@cs.biu.ac.il; C. Ré, Gates Computer Science Building, 353 Serra Mall, Stanford, CA 94305. USA; email: chrismre@cs.stanford.edu.

https://doi.org/10.1145/3180143

**ACM Reference format:**
Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3, Article 16 (March 2018), 40 pages.
https://doi.org/10.1145/3180143

# 1 INTRODUCTION

## 1.1 The Natural Join Query

A (natural) join query $q$ is a relational query of the form $q = R_1(\bar{A}_1) \bowtie R_2(\bar{A}_2) \bowtie \cdots \bowtie R_m(\bar{A}_m)$, where $\bar{A}_j$ is the attribute set of relation $R_j$, $j \in [m]$. Given a database instance $I$ on the schema $\{R_1, \ldots, R_m\}$, the join query evaluation problem is to compute $q(I)$, which is the set of all tuples $\mathbf{t}$ over attribute set $\bigcup_{j\in[m]} \bar{A}_j$ such that the projection of $\mathbf{t}$ onto the attributes $\bar{A}_j$ belongs to $R_j$, for each $j \in [m]$.

Efficient evaluation of join queries is at the heart of relational database management system [1, 55, 76]. Furthermore, the join operation can be used to capture a wide variety of problems in other domains in computer science, large graph analytics, coding theory, and combinatorics. For example, join query is essentially equivalent to constraint satisfaction [49], which is itself ubiquitous. Another class of examples concerns pattern extractions from complex networks such as biological networks (e.g., transcription and protein interaction networks), and social networks. The simplest example is in social network analysis, where computing and listing the number of triangles in a graph is key to computing the clustering coefficients [60, 73] and transitivity ratio [66, 74]. We can reduce the triangle enumeration problem to a three way cyclic join problem as follows. We evaluate the join

$$q = R(A, B) \bowtie S(B, C) \bowtie T(A, C) \tag{1}$$

where the database instance is the input graph $G = (V, E)$, and the three relations $R^G, S^G$, and $T^G$ are identical to the binary symmetric relation $E$. (Due to symmetry, each triangle in the graph will be listed 6 times in the join output.) Similarly, we can express the general subgraph listing problem as join queries. (We will address this latter point in a bit.)

The *list recovery problem* in coding theory [38] can also be stated a join evaluation problem, where roughly speaking we join the code and all the lists at all code positions. We have used our join evaluation algorithm applied to a particular join query to design new list recoverable codes that were crucial in designing new sub-linear time decodable compressed sensing schemes [28].

For the above reasons, efficient join query processing is a cornerstone algorithmic problem and has been extensively studied and implemented in the past 40 years [1, 18, 42, 55, 76].

## 1.2 The Optimal Join Problem and Our Main Result

Ideally, given $m$ relations $R_1, \ldots, R_m$ over $n$ attributes on a specific database instance $I$, we would like to evaluate the join in time

$$\tilde{O}\left(f(m,n) \cdot \sum_{j\in[m]} \left|R_j^I\right| + g(m,n) \cdot \left|\bowtie_{j\in[m]} R_j^I\right|\right),$$

where $f$ and $g$ are polynomial functions of the query size $(m, n)$, and $|R|$ denotes the number of tuples of relation $R$. Unfortunately, this ideal solution does not exist, modulo well-accepted complexity theoretic assumptions. For example, with a simple well-known reduction from the 3UniqueSAT problem, one can show that the ideal solution does not exist unless NP = RP. In fact, even when we relax $f$ and $g$ to be computable functions, in effect ignoring the complexity dependency on the

query size, such an instance-optimal result in terms of data-complexity is still unattainable unless W[1] = RFPT. (See Section 7.2 for details.)

Consequently, we formulate and solve an easier problem:

> **Optimal Worst-case Join Evaluation Problem** (Optimal Join Problem). *Given a fixed database schema* $\bar{R} = \{R_j(\bar{A}_j)\}_{j\in[m]}$ *and an* $m$*-tuple of integers* $\bar{N} = (N_1, \ldots, N_m)$*. Let* $q$ *be the natural join query joining the relations in* $\bar{R}$ *and let* $I(\bar{N})$ *be the set of all instances such that* $|R_j^I| = N_j$ *for* $j = 1, \ldots, m$*. Then, the worst-case optimal join evaluation problem is to evaluate* $q$ *in time*
>
> $$O\left(f(m,n)\cdot\sum_{j\in[m]} N_j + g(m,n)\cdot\sup_{I\in I(\bar{N})}|q(I)|\right).$$

Since any algorithm to produce $q(I)$ requires time at least $|q(I)|$, an algorithm that solves the above problem would have an optimal worst-case data-complexity. Implicitly, variants of optimal join problem has been studied for over three decades: a modern relational database management system (RDBMS) uses decades of highly tuned algorithms to efficiently produce query results. Nevertheless, as we shall show in this article, existing join-tree-based query evaluation systems are asymptotically suboptimal—even in the simple example of Equation (1).

The main result of this article is a solution to the above problem with $f(m,n) = n^2$ and $g(m,n) = mn$. Here, the term $f(m,n)\cdot\sum_{j\in[m]} N_j$ is the *pre-processing time*, and $g(m,n)\cdot\sup_{I\in I(\bar{N})}|q(I)|$ is the *worst-case query evaluation time*. In a RDBMS, one computes information, e.g., indexes, offline that may obviate the need to read the entire input relations to produce the output. In a similar spirit, we can extend our results to evaluate any query $q$ in time $O(g(m,n)\cdot\sup_{I\in I(\bar{N})}|q(I)|)$, removing the pre-processing time by precomputing some indices.

### 1.3 Atserias-Grohe-Marx's Fractional Cover Bound

The first question we need to answer to solve the optimal join problem is to compute exactly the bound $\sup_{I\in I(\bar{N})}|q(I)|$. Luckily, Grohe and Marx [36] and Atserias, Grohe, and Marx [9] (AGM's results henceforth) derived tight bounds on the number of output tuples of a (natural) join query in terms of the sizes of the input relations. Query output size estimation is also fundamentally important for efficient query processing; these results have generated a great deal of excitement and directly paved the way for our results.

To understand the spirit of AGM's results, consider the triangle query example Equation (1), and let $q(I)$ denote the set of tuples that is output from applying the query to a database instance $I$. Henceforth, we will drop the superscript $I$ if the input database instance is clear from context. By definition, $q(I)$ is the set of triples of constants $(a,b,c)$ such that $R(ab)$, $S(bc)$, and $T(ac)$ are in $I$. Our goal is to bound the number of tuples returned by $q$ on $I$, denoted by $|q(I)|$, in terms of $|R|$, $|S|$, and $|T|$. For simplicity, let us consider the case when $|R| = |S| = |T| = N$. A straightforward bound is $|q(I)| \le N^3$. One can obtain a better bound by noticing that any pair-wise join (say $R \bowtie S$) will contain $q(I)$ in it as $R$ and $S$ together contain all attributes (or they "cover" all the attributes). This leads to the bound $|q(I)| \le N^2$. AGM showed that one can get a better upper bound of $|q(I)| \le N^{3/2}$ by generalizing the notion of cover to a so-called "fractional cover" (see Section 3). Moreover, this estimate is tight in the sense that for infinitely many values of $N$, one can find a database instance $I$ that for which $|q(I)| = N^{3/2}$.

These non-trivial estimates are exciting to database researchers as they offer previously unknown, nontrivial methods to estimate the cardinality of a query result—a fundamental problem

to support efficient query processing. For example, when we use the query (1) to model the triangle enumeration problem on an input graph $G = (V, E)$, AGM's bound gives an upper bound of $(2|E|)^{3/2}/6$. This bound is the same as the tight bound obtained by Alon [6] on the number of triangles in a graph in terms of the number of edges of the graph. Using join queries to model subgraph enumeration, AGM's bound also yields very good bounds similar to those in References [6, 23, 27]. (We will come back to this problem in Section 8.4.)

More generally than the triangle example, given an arbitrary natural-join query $q$ and given the sizes of input relations, the AGM method can generate an upper bound $U$ such that $|q(I)| \leq U$, where $U$ depends on the "best" fractional cover of the attributes. This "best" fractional cover can be computed by a linear program (see Section 3 for more details). Henceforth, we refer to this inequality as the *AGM's fractional cover inequality*, and the bound $U$ as the *AGM's fractional cover bound* (or just *AGM bound*). They also show that the bound is essentially optimal in the sense that for infinitely many sizes of input relations, there exists an instance $I$ such that each relation in $I$ is of the prescribed size and $|q(I)| = \Omega(U)$.

AGM's results leave open whether one can compute the actual set $q(I)$ in time $O(U)$. In fact, AGM observe this issue and presented an algorithm that computes $q(I)$ with a runtime of $O(|q|^2 \cdot U \cdot N)$ where $N$ is the cardinality of the largest input relation and $|q|$ denotes the size of the query $q$. AGM establish that their join-project plan can in some cases be super-polynomially better than any join-only plan. However, AGM's join-project plan is not optimal. Even on the triangle query example of Equation (1), we can construct a family of database instances $I_1, I_2, \ldots, I_N, \ldots$, such that in the $N$th instance $I_N$ we have $|R| = |S| = |T| = N$ and *any* join-project and any join-only plan take $\Omega(N^2)$-time even though from AGM's bound we know that $|q(I)| \leq U = N^{3/2}$, which is the best worst-case runtime one can hope for.

### 1.4 Connections to Geometric Inequalities by Loomis-Whitney and Bollóbas-Thomason

Our contribution begins with a connection between AGM's inequality and a family of inequalities in geometry. In particular, we show that the AGM's inequality is *equivalent* to the discrete version of a geometric inequality proved by Bollóbas and Thomason ([14], Theorem 2). This equivalence is shown in Section 4.

Our ideas for an algorithm solving the optimal join problem begin by examining a special case of the Bollóbas-Thomason (BT) inequality: the classic Loomis-Whitney (LW) inequality [53]. The LW inequality bounds the measure of an $n$-dimensional set in terms of the measures of its $(n-1)$-dimensional projections onto the coordinate hyperplanes. We can bound the size of query Equation (1) with $|q(I)| \leq \sqrt{|R||S||T|}$ by applying *exactly* the $n = 3$-dimensional LW inequality to the discrete measure. Our algorithmic development begins with a slight generalization of the query $q$ in Equation (1). We describe an algorithm for join queries that has the same format as in the LW inequality setup with $n \geq 3$. In particular, we consider "LW instances" of the optimal join problem, where the query is to join $n$ relations whose attribute sets are all the distinct $(n-1)$-subsets of a universe of $n$ attributes. Since the LW inequality is tight, and our join algorithm has runtime that is asymptotically data-optimal for this class of queries (e.g., $O(N^{3/2})$ in our motivating example), our algorithm is data-complexity optimal in the worst case for LW instances.

Our algorithm for LW instances exhibits a key twist compared to a conventional join algorithm. The twist is that the join algorithm partitions the values of the join key on each side of the join into two sets: those values that are *heavy* and those values that are *light*. Intuitively, a value of a join key is heavy if its fanout is high enough so that joining all such join keys could violate the size bound (e.g., $N^{3/2}$ above). The art is selecting the precise fanout threshold for when a join key

is heavy. This per-tuple choice of join strategy is, to the best of our knowledge, not typically done in standard RDBMS join processing.

Building on the algorithm for LW instances, we next describe the main result: an algorithm to solve the optimal join problem for all join queries. In particular, we design an algorithm for evaluating join queries that not only *proves* AGM's fractional cover inequality *without* using the information-theoretic Shearer's inequality, but also has a runtime that is linear in the bound (modulo a linear time pre-processing scan over input relations). As AGM's inequality implies the BT and LW inequalities, our result is the first algorithmic proof of these geometric inequalities as well. To do this, we must carefully select which projections of relations to join and in which order. Our algorithm joins relations on a "per tuple" basis, and it also makes use of the light/heavy check idea but in a different way from the LW case.

### 1.5 Other Results

It is easy to show that any join-only plan is suboptimal for some queries. A natural question is, *when do classical RDBMS algorithms have higher worst-case runtime than our proposed approach?* AGM's analysis of their join-project algorithm leads to a worst case runtime complexity that is a factor of the largest relation worse than the AGM's bound. To investigate whether AGM's analysis is tight, we ask a sharper variant of this question: *Given a query $q$ does there exist a family of instances $I$ such that our algorithm runs asymptotically faster than a standard binary-join-based plan or AGM's join-project plan?* We give a partial answer to this question by describing a sufficient syntactic condition for the query $q$ such that for each $k \geq 2$, we can construct a family of instances where each relation is of size $N$ such that any project-join plan (which as a special case includes AGM's algorithm) will need time $\Omega(N^2/k^2)$, while the fractional cover bound is $O(N^{1+1/(k-1)})$—an asymptotic gap. We then show through a more detailed analysis that our algorithm on these instances takes $O(k^2N)$-time.

We consider several extensions and improvements of our main result. We show that our algorithm is also data-complexity optimal for full conjunctive queries, based on the generalization of AGM's bound by Gottlob, Lee, and Valiant [30] to conjunctive queries. In terms of the dependence on query size, our algorithms are also efficient (at most linear in $|q|$, which is better than the quadratic dependence in AGM) for full queries, but they are not necessarily optimal. In particular, if each relation in the schema has arity 2, we are able to give an algorithm with better query complexity than our general algorithm. This shows that in general our algorithm's dependence on the factors of the query is not the best possible. We also consider computing a relaxed notion of joins and give worst-case optimal algorithms for this problem as well. Finally, we consider the algorithmic version of a very well studied combinatorial problem [6, 27]: given a fixed graph $H$, enumerate all of its induced copies in a much larger graph $G$. Our main observation here, once we formulate the correct join query version of the subgraph enumeration problem, is to show that the AGM bound is still tight.

### 1.6 Outline of the Rest of the Paper

The remainder of the article is organized as follows. Section 2 describes related and subsequent work. In Section 3, we describe our notation and formulate the main problem. Section 4 proves the connection between AGM's inequality and BT inequality. In Section 5, we present a data-optimal join algorithm for LW instances, and then extend this to arbitrary join queries in Section 6. (A reader who is only interested in the optimal join algorithm for general join queries can skip Sections 4 and 5.) We discuss the limits of performance of prior approaches and our approach in more detail in Section 7. In Section 8, we describe several extensions. We conclude in Section 9.

## 2 RELATED WORK

Grohe and Marx [36] made the first (implicit) connection between fractional edge cover and the output size of a join query. Their results were stated for constraint satisfaction problems, but the close ties between CSPs and join queries are well-known [49]. Atserias, Grohe, and Marx [9] extended Grohe and Marx's results in a database-friendly language, hence, we will discuss AGM's results even though the bound and the algorithm most related to our work already appeared (implicitly) in Grohe and Marx.

Consider a join query over $m$ relations $R_e$, $e \in E$, where $E$ is a collection of subsets of an $n$ attribute universe $V$, and relation $R_e$ is on attribute set $e$. Then, AGM show that the number of output tuples is bounded above by $\prod_{e\in E} |R_e|^{x_e}$, where $\mathbf{x} = (x_e)_{e\in E}$ is an *arbitrary* fractional edge cover of the hypergraph $H = (V, E)$. They also show that this bound is tight. In particular, for infinitely many positive integers $N$ there is a database instance with $|R_e| = N$, $\forall e \in E$, and the best upper bound gives the actual number of output tuples. When the sizes $|R_e|$ were given as inputs to the (output size estimation) problem, obviously the best upper bound is obtained by picking the fractional cover $\mathbf{x}$ that minimizes the linear objective function $\sum_{e\in E}(\log |R_e|) \cdot x_e$. In this size constrained case, however, their lower bound is off from the upper bound by a factor of $2^n$, where $n$ is the total number of attributes. AGM also present an inapproximability result that justifies this gap. Note, however, that the gap is only dependent on the query size and the bound is still asymptotically optimal in the data-complexity sense. The second relevant result from AGM is a join-project query evaluation plan with runtime $O(|q|^2 N_{\max}^{1+\sum_{e\in E} x_e})$, where $N_{\max}$ is the maximum size of input relations and $|q| = nm$ is the query size.

As shall be shown later, the AGM's inequality contains as a special case the discrete versions of two well-known inequalities in geometry: the *Loomis-Whitney* (LW) inequality [53] and its generalization the *Bollobás-Thomason* (BT) inequality [14]. There are two typical proofs of the discrete LW and BT inequalities. The first proof is by induction using Hölder's inequality [14]. The second proof (see Lyons and Peres [54]) essentially uses equivalent entropy inequalities by Han [40] and its generalization by Shearer [21], which was also the route Grohe and Marx [36] took to prove AGM's bound. All of these proofs are non-constructive.

There are many applications of the discrete LW and BT inequalities. The $n = 3$ case of the LW inequality was used to prove communication lower bounds for matrix multiplication on distributed memory parallel computers [45]. The inequality was used to prove submultiplicativity inequalities regarding sums of sets of integers [39]. In Reference [52], a special case of BT inequality was used to prove a network-coding bound. Recently, some of the authors of this article have used our *algorithmic* version of the LW inequality to design a new sub-linear time decodable compressed sensing matrices [28].

Inspired by AGM's results, Gottlob, Lee, and Valiant [30] provided bounds for conjunctive queries with and without functional dependencies. For these bounds, they defined a new notion of "coloring number" that comes from the dual linear program of the fractional cover linear program. The coloring number was introduced to deal with functional dependencies. (See Section 8.3 for more details.)

Related to the problem of estimating the size of an output is cardinality estimation. A large number of structures have been proposed for cardinality estimation [7, 22, 43, 47, 50, 68], they have all focused on various sub-classes of queries and deriving estimates for arbitrary query expressions and has involved making statistical assumptions such as the independence and containment assumptions that result in large estimation errors [44]. Follow-up work has considered sophisticated probability models, Entropy-based models [57, 72] and graphical models [75]. In contrast, in this work, we examine the *worst case behavior* of algorithms in terms of its cardinality estimates.

Join processing algorithms are one of the most studied algorithms in database research. A staggering number of variants have been considered, we list a few: Block-Nested loop join, Hash-Join, Grace, Sort-merge (see Grafe [34] for a survey). Conceptually, it is interesting that none of the classical algorithms consider performing a per-tuple cardinality estimation as our algorithm does.

Join query is a special case of conjunctive query. Conjunctive query evaluation is a fundamental problem in database theory, which has a very long history and deep connections to logic and constraint satisfaction [17, 19, 24, 32, 33, 49, 67, 78]. It was recognized early on that the complexity dependency on the query plays a very different role from the complexity dependency on the data, because in typical databases the size of the data is many orders of magnitude larger than the size of the query. This phenomenon is magnified in the large graph problems on complex networks, because the size of the relations (complex network edge set) is many orders of magnitude larger than the size of the input query (the small subgraph pattern).

In the 1980s, starting with the seminal work of Vardi [78], database researchers started to analyze three types of query complexities: expression complexity, data complexity, and combined complexity. The expression complexity of a query is the computational complexity of the query evaluation problem when the data is fixed. In data complexity, we fix the query and the database is the input. In combined complexity, the input size is the sum of the query expression size and the input data size. While data complexity is insensitive to the way the query is expressed, the query complexity is highly dependent on the query language. (Consider, for example, $R \bowtie \cdots \bowtie R$ one hundred times vs. $R^{100}$). This is also the reason why "expression complexity" is used instead of "query complexity." Also, to remove the dependency on the output size, most works focused on the Boolean case where were ask whether the output of a given conjunctive query is empty. These queries are called *Boolean conjunctive queries*. Since the classic work of Chandra and Merlin in 1977 [17], showing that the expression complexity and the combined complexity of conjunctive queries are NP-hard, there has been a long line of work on the expression, data, and combined complexity of conjunctive queries [15, 16, 67, 78, 79]. The important phenomenon to notice is that there is often an exponential complexity gap between the data complexity and the expression complexity [78]. Due to the obvious asymmetry between data size and query size, Yannakakis [85] suggested that *parameterized complexity* is the correct way to study query complexity. Roughly, a query evaluation problem is *fixed parameter tractable* (FPT) if there is an algorithm for it with run time $f(|q|)n^c$ for some constant $c$, where $|q|$ is the size of the query expression and $n$ is the size of the database. Then, Papadimitriou and Yannakakis [67] showed that the conjunctive query evaluation problem is $W[1]$-complete.

Since the general conjunctive query evaluation problem is computationally hard, researchers have looked at the complexity of restricted classes of queries. The most important sub-class of conjunctive queries is the class of *acyclic* queries. The body of a conjunctive query can be modeled by a hypergraph where the vertex set is the set of all variables appearing in the query's body and the hyperedges are the sets of variables, one set per atom. The query is acyclic if the corresponding hypergraph is acyclic (see References [1, 24, 55, 76] for the definition of query hypergraph acyclicity). In 1981, Yannakakis [84] showed that acyclic conjunctive queries can be evaluated in polynomial time. There have been an extensive body of work on the complexity and algorithms for acyclic conjunctive queries [12, 13, 19, 24, 25, 29, 31, 49, 56, 65, 67, 69, 70, 81, 82, 86, 87]. Another large class of queries called the *bounded tree-width* queries are also poly-time computable [19, 31]. These results are essentially the best we can hope for in terms of combined complexity and parameterized complexity [37].

On a technical level, the work *adaptive query processing* is related, e.g., Eddies [10] and RIO [11]. The main idea is that to compensate for bad statistics, the query plan may adaptively be changed (as it better understands the properties of the data). While both our method and the methods

proposed here are adaptive in some sense, our focus is different: this body of work focuses on heuristic optimization methods, while our focus is on provable worst-case runtime bounds. A related idea has been considered in practice: heuristics that split tuples based on their fanout have been deployed in modern parallel databases to handle skew [83]. This idea was not used to theoretically improve the runtime of join algorithms.

### 2.1 Subsequent Work

We now present some of the work that followed the initial publications of our results in [62]. Shortly after the initial publication of our work, Veldhuizen showed that another join algorithm called Leapfrog TrieJoin (henceforth, LFTJ) is also worst-case optimal [80]. What is remarkable about this result is that LFTJ was already implemented in the commercial database engine of LogicBlox and the optimality of LFTJ was proved *after* the algorithm had already been used in practice. LFTJ is similar but a bit different from our algorithm. We then observed in Reference [63] that both LFTJ and our algorithm are special cases of a general join algorithm that is also worst-case optimal. We simplified the argument in Reference [63], and we follow the simplified analysis of our algorithm in this article.

A natural question left open by our work was to design join algorithms that have some per-instance guarantee. Ngo et al. [61] designed a join algorithm with beyond worst-case complexity guarantees. Both this algorithm and LFTJ are very competitive with respect to existing database engines especially on graph datasets [64]. Abo Khamis et al. later showed that both these beyond worst-case results and the worst-case optimal results can be recovered in a single simple framework that has close connections to logical resolution [2]. Recently Abo Khamis et al. [3] were able to generalize the worst-case join algorithms to obtain algorithms for certain sums of products over semi-rings (which in addition to modeling conjunctive queries also capture problems in probabilistic graphical models, matrix vector multiplication and various problem in logic). Also, see the followup work [48].

Under queries with functional dependencies or degree bounds (which generalize both cardinality bounds and functional dependencies), where the AGM bound is no-longer tight nor is the best possible bound. In these general settings, there have been exciting recent progress both in terms of output-size bounds, algorithms meeting the output size bounds and stronger notions of query widths. These new results tightly connect information theoretic inequalities and database algorithms. The new developments vastly generalize much of the bounds and algorithmic results discussed in this article [4, 5].

## 3 NOTATION AND FORMAL PROBLEM STATEMENT

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 $R_1, R_2, \ldots$. A relational schema for the symbol $R_i$ of arity $k$ is a tuple $\bar{A}_i = (A_{i_1}, \ldots, A_{i_k})$ of distinct attributes[1] that defines the attributes of the relation. A relational database schema is a set of relational symbols and associated schemas denoted by $R_1(\bar{A}_1), \ldots, R_m(\bar{A}_m)$. 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 $I$ is a collection of instances, one for each relational symbol in schema, denoted by $R_i^I$.

A *natural join* query (or simply query) $q$ is specified by a finite subset of relational symbols $q \subseteq \mathbb{N}$, denoted by $\bowtie_{i \in q} R_i$. Let $\bar{A}(q)$ denote the set of all attributes that appear in some relation in $q$, that is $\bar{A}(q) = \{A \mid A \in \bar{A}_i \text{ for some } i \in q\}$. 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

[1]When handling more general conjunctive queries, the attributes need not be distinct. We handle this case in Section 8.3.

database instance $I$, the output of the query $q$ on the database instance $I$ is denoted by $q(I)$ and is defined as

$$q(I) \stackrel{\text{def}}{=} \left\{\mathbf{t} \in \mathbf{D}^{\bar{A}(q)} \mid \mathbf{t}_{\bar{A}_i} \in R_i^I \text{ for each } i \in q\right\},$$

where $\mathbf{D}^{\bar{A}(q)}$ is a shorthand for $\times_{i:A_i \in \bar{A}(q)} \mathbf{D}_i$.

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 tuple $\mathbf{t}_{\bar{S}}$, define the $\mathbf{t}_{\bar{S}}$*-section* of $R$ as

$$R[\mathbf{t}_{\bar{S}}] = \pi_{\bar{A}\setminus\bar{S}}(R \ltimes \{\mathbf{t}_{\bar{S}}\}).$$

*From Join Queries to Hypergraphs.* A query $q$ on attributes $\bar{A}(q)$ can be viewed as a hypergraph $H = (V, E)$ where $V = \bar{A}(q)$ and there is an edge $e_i = \bar{A}_i$ for each $i \in q$. Henceforth, we will often use the edges of the hypergraph to index the relations. Hence, the join query $q$ represented by the hypergraph $H = (V, E)$ is the query $q = \bowtie_{e\in E} R_e$, with $n = |V|$ and $m = |E|$. When the input database instance $I$ is implicit, we will drop the superscript $I$, and let $N_e = |R_e|$ be the number of tuples in $R_e$ (in $I$).

We use the hypergraph setting to introduce the *fractional edge cover polyhedron* that plays a central role in our technical developments. The fractional edge cover polyhedron defined by $H$ is the set of all points $\mathbf{x} = (x_e)_{e\in E} \in \mathbb{R}^E$ such that

$$\sum_{v\in e} x_e \geq 1, \text{ for any } v \in V$$
$$x_e \geq 0, \text{ for any } e \in E.$$

Note that the solution $x_e = 1$ for $e \in E$ is always feasible for hypergraphs representing join queries. A point $\mathbf{x}$ in the polyhedron is also called a *fractional (edge) cover* of the hypergraph $H$.

Grohe and Marx [36], and Atserias, Grohe, and Marx [9] establish the following bound.

Theorem 3.1 (AGM's bound). *Let $q$ be a join query represented by the hypergraph $H = (V, E)$. Let $N_e$ be the input relation sizes. Then, for* any *fractional cover $\mathbf{x} = (x_e)_{e\in E}$, the following upper-bound holds*

$$\left|\bowtie_{e\in E} R_e\right| \leq \prod_{e\in E} N_e^{x_e}. \tag{2}$$

*Furthermore, there are arbitrarily large database instances such that $N_e = N$ for all $e \in E$ and the* best *upper-bound is tight for such instances. If the input sizes $N_e$ are given, then there exists a database instance for which $|\bowtie_{e\in E} R_e| \geq 2^{-|V|} \cdot \prod_{e\in E} N_e^{x_e^*}$, where $(x_e^*)_e$ is an optimal solution to LP with the objective* $\min \sum_{e\in E} (\log N_e) x_e$ *subject to the edge covering constraints above.*

The bound is proved nonconstructively using Shearer's entropy inequality [21]. However, AGM provide an algorithm based on join-project plans that runs in time $O(|q|^2 \cdot N_{\max}^{1+\sum_{e\in E} x_e})$ where $N_{\max} = \max_{e\in E} N_e$. They observe that for a fixed hypergraph $H$ and given sizes $N_e$ the bound (2) can be minimized by solving the linear program that minimizes the linear objective $\sum_{e\in E} (\log N_e) \cdot x_e$ over fractional edge cover solutions $\mathbf{x}$. (Since in linear time we can figure out if we have an empty relation, and hence an empty output, for the rest of the article, we are always going to assume that $N_e \geq 1$.) Thus, the formal problem that we consider recast in this language is:

*Definition 3.2 (OJ Problem—Optimal Join Problem).* With the notation above, let $\mathbf{x}$ be an arbitrary fractional cover of the hypergraph $H$, design an algorithm to compute $\bowtie_{e\in E} R_e$ with runtime

$$O\left(f(n,m)\cdot\sum_{e\in E}N_e + g(n,m)\cdot\prod_{e\in E}N_e^{x_e}\right).$$

Here $f(\cdot)$ and $g(\cdot)$ are ideally polynomials with (small) constant degree, which only depend on the query size. The linear term $\sum_{e\in E} N_e$ is to read the input. Hence, such an algorithm would be data-optimal in the worst case.[2]

*Example 3.1.* We recast the triangle query example (1) from the introduction in the hypergraph language. Recall, we are given three relations $R(A,B)$, $S(B,C)$, and $T(A,C)$, so $V = \{A,B,C\}$ and three edges corresponding each to $R$, $S$, and $T$ are $E = \{\{A,B\},\{B,C\},\{A,C\}\}$ respectively. Thus, $n = 3$ and $m = 3$. If we are given that $N_e = N$ for all $e \in E$, then one can check that the optimal fractional cover that minimizes $x_{\{A,B\}}\log N + x_{\{B,C\}}\log N + x_{\{A,C\}}\log N$ is $x_e = 1/2$ for $e \in E$, which has the objective value $3/2\cdot\log N$; in turn, this gives $\sup_{I\in I(\bar{N})}|q(I)| \le N^{3/2}$ (recall $I(\bar{N}) = \{I : |R_e^I| = N_e \text{ for } e \in E\}$). The join-project algorithm of AGM on this example gives a time bound of $O(N^{5/2})$, though a closer analysis gives a time bound of $O(N^2)$.

Given an even integer $N$, we construct a database instance $I_N$ such that (1) $|R^{I_N}| = |S^{I_N}| = |T^{I_N}| = N$, (2) $|R \bowtie S| = |R \bowtie T| = |S \bowtie T| = N^2/4 + N/2$, and (3) $|R \bowtie S \bowtie T| = 0$. The following instance satisfies all three properties:

$$R^{I_N} = S^{I_N} = T^{I_N} = \{(0,j)\}_{j\in[N/2]} \bigcup \{(j,0)\}_{j\in[N/2]}.$$

For example,

$$R \bowtie S = \{(i,0,j)\}_{i,j\in[N/2]} \bigcup \{(0,i,0)\}_{i\in[N/2]}$$

and $R \bowtie S \bowtie T = \emptyset$. Thus, any standard join-based algorithm takes time $\Omega(N^2)$. We show later that any join-project plan (including AGM's algorithm) takes $\Omega(N^2)$-time too. Recall that the AGM bound for this instance is $O(N^{3/2})$, and our algorithm thus takes time $O(N^{3/2})$. In fact, as shall be shown later, on this particular family of instances our algorithm takes only $O(N)$ time.

## 4 CONNECTIONS TO GEOMETRIC INEQUALITIES

In this section, we describe the Bollobás-Thomason (BT) inequality from discrete geometry and prove that BT inequality is equivalent to AGM's inequality. We then look at a special case of BT inequality, the Loomis-Whitney (LW) inequality, from which our algorithmic development starts in the next section.

Theorem 4.1 (Discrete Bollobás-Thomason (BT) Inequality). *Let $S \subset \mathbb{Z}^n$ be a finite set of $n$-dimensional grid points. Let $\mathcal{F}$ be a collection of subsets of $[n]$ in which every $i \in [n]$ occurs in exactly $d$ members of $\mathcal{F}$. Let $S_F$ be the set of projections $\mathbb{Z}^n \to \mathbb{Z}^F$ of points in $S$ onto the coordinates in $F$. Then, $|S|^d \le \prod_{F\in\mathcal{F}} |S_F|$.*

To prove the equivalence between BT inequality and the AGM bound, we first need a simple observation.

Lemma 4.2. *Consider an instance of the OJ problem consisting of a hypergraph $H = (V,E)$, a fractional cover $\mathbf{x} = (x_e)_{e\in E}$ of $H$, and relations $R_e$ for $e \in E$. Then, in linear time, we can transform the*

[2]Following Reference [26], we assume in this work that given relations $R$ and $S$ one can compute $R \bowtie S$ in time $O(|R| + |S| + |R \bowtie S|)$. This only holds in an amortized sense using hashing, or in the worst-case sense using the "lazy array" technique at the cost of large space overhead. To achieve true worst-case results with low-space, one can use sorting operations that result in a log factor increase in runtime.

*instance into another instance* $H' = (V, E'), \mathbf{x}' = (x'_e)_{e\in E'}, (R'_e)_{e\in E'}$, *such that the following properties hold:*

- *(a)* $\mathbf{x}'$ *is a "tight" fractional edge cover of the hypergraph* $H'$, *namely* $\mathbf{x}' \geq 0$ *and*
$$\sum_{e\in E': v\in e} x'_e = 1, \quad \text{for every } v \in V.$$
- *(b) The two problems have identical join output:*
$$\bowtie_{e\in E} R_e = \bowtie_{e\in E'} R'_e.$$
- *(c) AGM's bound on the transformed instance is at least as good as that of the original instance:*
$$\prod_{e\in E'} |R'_e|^{x'_e} \leq \prod_{e\in E} N_e^{x_e}.$$

Proof. We describe the transformation in steps. At each step properties (b) and (c) are invariants. After all steps are done, (a) holds.

While there still exists some vertex $v \in V$ such that $\sum_{e\in E: v\in e} x_e > 1$, i.e. $v$'s constraint is not tight, let $f$ be an arbitrary hyperedge $f \in E$ such that $v \in f$. Partition $f$ into two parts $f = f_t \cup f_{\neg t}$, where $f_t$ consists of all vertices $u \in f$ such that $u$'s constraint is tight, and $f_{\neg t}$ consists of vertices $u \in f$ such that $u$'s constraint is not tight. Note that $v \in f_{\neg t}$.

Define $\rho = \min\{x_f, \min_{u\in f_{\neg t}}\{\sum_{e: u\in e} x_e - 1\}\}$. This is the amount, which if we were able to reduce $x_f$ by $\rho$, then we will either turn $x_f$ to 0 or make some constraint for $u \in f_{\neg t}$ tight. However, reducing $x_f$ might violate some already tight constraint for $u \in f_t$. The trick is to break $f$ into two parts.

We will set $E' = E \cup \{f_t\}$, create a "new" relation $R'_{f_t} = \pi_{f_t}(R_f)$, and keep all the old relations $R'_e = R_e$ for all $e \in E$. Set the variables $x'_e = x_e$ for all $e \in E - \{f\}$ also. The only two variables that have not been set are $x'_f$ and $x'_{f_t}$. We set them as follows.

- When $x_f \leq \min_{u\in f_{\neg t}} \{\sum_{e: u\in e} x_e - 1\}$, set $x'_f = 0$ and $x'_{f_t} = x_f$.
- When $x_f > \min_{u\in f_{\neg t}} \{\sum_{e: u\in e} x_e - 1\}$, set $x'_f = x_f - \rho$ and $x'_{f_t} = \rho$.

Either way, it can be readily verified that the new instance is a legitimate OJ instance satisfying properties (b) and (c). In the first case, some positive variable in some non-tight constraint has been reduced to 0. In the second case, at least one non-tight constraint has become tight. Thus, each step makes progress in terms of either having one more tight constraint or one more positive variable in a non-tight constraint becoming 0. The new variable $x'_{f_t}$ that we introduce only participates in tight constraints, and hence, we will not change it in future steps. Hence, after a linear number of steps in $n + m$, we will have all tight constraints. □

With this technical observation, we can now connect the two families of inequalities:

Theorem 4.3. *AGM's inequality implies BT inequality.*

Proof. To see that AGM's inequality implies BT inequality, we think of each coordinate as an attribute, and the projections $S_F$ as the input relations. Set $x_F = 1/d$ for each $F \in \mathcal{F}$. It follows that $\mathbf{x} = (x_F)_{F\in\mathcal{F}}$ is a fractional cover for the hypergraph $H = ([n], \mathcal{F})$. AGM's bound then implies that $|S| \leq \prod_{F\in\mathcal{F}} |S_F|^{1/d}$. □

Theorem 4.4. *BT inequality implies AGM's inequality.*

Proof. Consider an instance of the OJ problem with hypergraph $H = (V, E)$ and a fractional cover $\mathbf{x} = (x_e)_{e\in E}$ of $H$. With standard analytic arguments, we can assume that the $x_e$ are rational

values (otherwise, we can replace each $x_e$ by a rational sequence $x'_e$ tending to $x_e$ from above, i.e., $x'_e > x_e$). First, by Lemma 4.2, we can assume that all cover constraints are tight, i.e.,

$$\sum_{e:v\in e} x_e = 1, \text{ for any } v \in V.$$

Second, by writing all variables $x_e$ as $d_e/d$ for a positive common denominator $d$, we obtain

$$\sum_{e:v\in e} d_e = d, \text{ for any } v \in V.$$

Now, create $d_e$ copies of each relation $R_e$. Call the new relations $R'_e$. We obtain a new hypergraph $H' = (V, E')$ where every attribute $v$ occurs in exactly $d$ hyperedges. This is precisely the Bollóbas-Thomason's setting of Theorem 4.1. Hence, the size of the join is bounded above by $\prod_{e\in E'} |R'_e|^{1/d} = \prod_{e\in E} N_e^{d_e/d} = \prod_{e\in E} N_e^{x_e}$. □

*The Loomis-Whitney Inequality.* We now consider a special case of BT (or AGM), the discrete version of a classic geometric inequality called the *Loomis-Whitney inequality* [53]. The setting is that for $n \geq 2$, $V = [n]$ and $E = \binom{V}{n-1}$. In this case $x_e = 1/(n-1), \forall e \in E$ is a fractional cover solution for $(V, E)$, and Loomis-Whitney showed the following:

Theorem 4.5 (Discrete Loomis-Whitney (LW) inequality). *Let $S \subset \mathbb{Z}^n$ be a finite set of $n$-dimensional grid points. For each dimension $i \in [n]$, let $S_{[n]\setminus\{i\}}$ denote the $(n-1)$-dimensional projection of $S$ onto the coordinates $[n] \setminus \{i\}$. Then, $|S|^{n-1} \leq \prod_{i\in[n]} |S_{[n]\setminus\{i\}}|$.*

It is clear from our discussion above that LW is a special case of BT (and so AGM), and it is with this special case that we begin our algorithmic development in the next section.

## 5 ALGORITHM FOR LOOMIS-WHITNEY INSTANCES

We first consider queries whose forms are slightly more general than that in our motivating example 3.1. This class of queries has the same setup as in LW inequality of Theorem 4.5. In this spirit, we define a *Loomis-Whitney (LW) instance* of the OJ problem to be a hypergraph $H = (V, E)$ such that $E$ is the collection of all subsets of $V$ of size $n-1$. When the LW inequality is applied to this setting, it guarantees that $|\bowtie_{e\in E} R_e| \leq (\prod_{e\in E} N_e)^{1/(n-1)}$, and the bound is tight in the worst case.

The reader who is interested in the result for general joins can safely skip this section and move on to the next section.

The main result of this section is the following:

Theorem 5.1 (Loomis-Whitney Instance). *Let $n \geq 2$ be an integer. Consider a Loomis-Whitney instance $H = (V = [n], E)$ of the OJ problem with input relations $R_e$ for $e \in E$. Then the join $\bowtie_{e\in E} R_e$ can be computed in time*

$$O\left(n^2 \sum_{e\in E} N_e + n^2 \cdot \left(\prod_{e\in E} N_e\right)^{1/(n-1)}\right).$$

Before proving this result, we give an example that illustrates the intuition behind our algorithm and solve the motivating example Equation (1) from the introduction.

*Example 5.1.* Recall that our input has three relations $R(A, B)$, $S(B, C)$, $T(A, C)$ and a database instance $I$ such that $|R| = |S| = |T| = N$. (Again, we drop the superscript $I$ as the instance is implicitly understood.) Let $J = R \bowtie S \bowtie T$. Our goal is to construct $J$ in time $O(N^{3/2})$. For exposition,

define a parameter $\tau \geq 0$ that we will choose below. We use $\tau$ to define two sets that effectively partition the tuples in $R$.

$$D = \{\mathbf{t}_B \in \pi_B(R) : |R[\mathbf{t}_B]| > \tau\} \text{ and } G = \{(\mathbf{t}_A, \mathbf{t}_B) \in R : \mathbf{t}_B \notin D\}$$

Intuitively, $D$ contains the heavy join keys in $R$. Note that $|D| < N/\tau$. Observe that $J \subseteq (D \times T) \cup (G \bowtie S)$, and that this union is disjoint. Our algorithm will construct $D \times T$ (resp. $G \bowtie S$) in time $O(N^{3/2})$, then it will filter out those tuples in both $S$ and $R$ (resp. $T$) using the hash tables on $S$ and $R$ (resp. $T$); this process produces exactly $J$. Since the runtime is linear in the above sets' sizes, the key question is how big are these two sets?

Observe that $|D \times T| \leq (N/\tau)N = N^2/\tau$, while $|G \bowtie S| = \sum_{t_B \in \pi_B(G)} |R[t_B]| \cdot |S[t_B]| \leq \tau N$. Setting $\tau = \sqrt{N}$ to minimize the maximum of the upper-bounds will make both terms at most $N^{3/2}$, establishing the runtime of our algorithm. One can check that if the relations are of different cardinalities, then we can still use the same algorithm; moreover, by setting $\tau = \sqrt{|R| \cdot |T| \cdot |S|}$, we achieve a runtime of $O(\sqrt{|R| \cdot |S| \cdot |T|} + |R| + |S| + |T|)$.

We would like to point out that an LW instance with $n > 3$ is a more complicated join query than the triangle query but our algorithm has to run faster than the algorithm above. To describe the general algorithm underlying Theorem 5.1, we need to introduce some data structures and notation.

*Data Structures and Notation.* Let $H = (V, E)$ be an LW instance. Algorithm 1 begins by constructing a labeled, binary tree $\mathcal{T}$ whose set of leaves is exactly $V$ and each internal node has exactly two children. Any binary tree over this leaf set can be used. We denote the left child of any internal node $x$ as $\text{LC}(x)$ and its right child as $\text{RC}(x)$. Each node $x \in \mathcal{T}$ is labeled by a function LABEL, where $\text{LABEL}(x) \subseteq V$ are defined inductively as follows: $\text{LABEL}(x) = V \setminus \{x\}$ for a leaf node $x \in V$, and $\text{LABEL}(x) = \text{LABEL}(\text{LC}(x)) \cap \text{LABEL}(\text{RC}(x))$ if $x$ is an internal node of the tree. It is immediate that for any internal node $x$, we have $\text{LABEL}(\text{LC}(x)) \cup \text{LABEL}(\text{RC}(x)) = V$ and that $\text{LABEL}(x) = \emptyset$ if and only if $x$ is the root of the tree. Let $J$ denote the output set of tuples of the join, i.e. $J = \bowtie_{e \in E} R_e$. For any node $x \in \mathcal{T}$, let $\mathcal{T}(x)$ denote the subtree of $\mathcal{T}$ rooted at $x$, and $\mathcal{L}(\mathcal{T}(x))$ denote the set of leaves under this subtree. For any three relations $R$, $S$, and $T$, define $R \bowtie_S T = (R \bowtie T) \ltimes S$.

*Algorithm for LW Instances.* Algorithm 1 works in two stages. Let $u$ be the root of the tree $\mathcal{T}$. First, we compute a tuple set $C(u)$ containing the output $J$ such that $C(u)$ has a relatively small size (at most the size bound times $n$). Second, we prune those tuples that cannot participate in the join (which takes only linear time in the size of $C(u)$). The interesting part is how we compute $C(u)$. Inductively, we compute a set $C(x)$ that at each stage contains candidate tuples and an auxiliary set $D(x)$, which is a superset of the projection $\pi_{\text{LABEL}(x)}(J \setminus C(x))$. The set $D(x)$ will intuitively allow us to deal with those tuples that would blow up the size of an intermediate relation. The key novelty in Algorithm 1 is the construction of the set $G$ that contains all those tuples (join keys) that are in some sense *light*, i.e., joining over them would not exceed the size/time bound $P$ by much. The elements that are not light are postponed to be processed later by pushing them to the set $D(x)$. This is in full analogy to the sets $G$ and $D$ defined in Example 5.1.

Proof of Theorem 5.1. We claim that the following three properties hold for every node $x \in \mathcal{T}$:

- (1) $\pi_{\text{LABEL}(x)}(J \setminus C(x)) \subseteq D(x)$;
- (2) $|C(x)| \leq (|\mathcal{L}(\mathcal{T}(x))| - 1) \cdot P$; and
- (3) $|D(x)| \leq \min\left\{\min_{l \in \mathcal{L}(\mathcal{T}(x))}\{N_{[n]\setminus\{l\}}\}, \frac{\prod_{l \in \mathcal{L}(\mathcal{T}(x))} N_{[n]\setminus\{l\}}}{P^{|\mathcal{L}(\mathcal{T}(x))|-1}}\right\}$.

**ALGORITHM 1:** Algorithm for Loomis-Whitney Instances

An LW instance: $R_e$ for $e \in \binom{V}{n-1}$ and $N_e = |R_e|$.
$P \leftarrow \prod_{e \in E} N_e^{1/(n-1)}$ (the size bound from LW inequality)
$u \leftarrow \text{root}(\mathcal{T})$; $(C(u), D(u)) \leftarrow \text{LW}(u)$
"Prune" $C(u)$ and return

$\text{LW}(x) : x \in \mathcal{T}$ returns $(C, D)$

**if** $x$ is a leaf **then**
  **return** $(\emptyset, R_{\text{LABEL}(x)})$
$(C_L, D_L) \leftarrow LW(\text{LC}(x))$ and $(C_R, D_R) \leftarrow LW(\text{RC}(x))$
$F \leftarrow \pi_{\text{LABEL}(x)}(D_L) \cap \pi_{\text{LABEL}(x)}(D_R)$
$G \leftarrow \{\mathbf{t} \in F : |D_L[\mathbf{t}]| + 1 \le \lceil P/|D_R| \rceil\}$ //*Note that* $F = G = \emptyset$ *if* $|D_R| = 0$
**if** $x$ is the root of $\mathcal{T}$ **then**
  $C \leftarrow (D_L \rtimes D_R) \cup C_L \cup C_R$
  $D \leftarrow \emptyset$
**else**
  $C \leftarrow (D_L \rtimes_G D_R) \cup C_L \cup C_R$
  $D \leftarrow F \setminus G$.
**return** $(C, D)$

Assuming these three properties hold, let us first prove that that Algorithm 1 correctly computes the join, $J$. Let $u$ denote the root of the tree $\mathcal{T}$. By property (1),

$$\pi_{\text{LABEL}(\text{LC}(u))}(J \setminus C(\text{LC}(u))) \subseteq D(\text{LC}(u)),$$
$$\pi_{\text{LABEL}(\text{RC}(u)))}(J \setminus C(\text{RC}(u))) \subseteq D(\text{RC}(u.))$$

Hence,

$$J \setminus (C(\text{LC}(u)) \cup C(\text{RC}(u))) \subseteq D(\text{LC}(u)) \times D(\text{RC}(u)) = D(\text{LC}(u)) \rtimes D(\text{RC}(u)).$$

This implies $J \subseteq C(u)$. Thus, from $C(u)$, we can compute $J$ by keeping only tuples in $C(u)$ whose projection on any attribute set $e \in E = \binom{[n]}{n-1}$ is contained in $R_e$ (the "pruning" step).

We next show that properties 1–3 hold by induction on each step of Algorithm 1. For the base case, consider $\ell \in \mathcal{L}(\mathcal{T})$. Recall that in this case $C(\ell) = \emptyset$ and $D(\ell) = R_{[n]\setminus\{\ell\}}$; thus, properties 1–3 hold.

Now assume that properties 1–3 hold for all children of an internal node $v$. We first verify properties 2 and 3 for $v$. From the definition of $G$,

$$|D(\text{RC}(v)) \rtimes_G D(\text{LC}(v))| \le \left( \left\lceil \frac{P}{|D(\text{RC}(v))|} \right\rceil - 1 \right) \cdot |D(\text{RC}(v))| \le P.$$

From the inductive upper bounds on $C(\text{LC}(v))$ and $C(\text{RC}(v))$, property 2 holds at $v$. By definition of $G$ and a straightforward counting argument, note that

$$|D(v)| = |F \setminus G| \le |D(\text{LC}(v))| \cdot \frac{1}{\lceil P/|D(\text{RC}(v))| \rceil} \le \frac{|D(\text{LC}(v))| \cdot |D(\text{RC}(v))|}{P}.$$

From the induction hypotheses on $\text{LC}(v)$ and $\text{RC}(v)$, we have

$$|D(\text{LC}(v))| \le \frac{\prod_{\ell \in \mathcal{L}(\mathcal{T}(\text{LC}(v)))} N_{[n]-\{\ell\}}}{P^{|\mathcal{L}(\mathcal{T}(\text{LC}(v)))|-1}},$$
$$|D(\text{RC}(v))| \le \frac{\prod_{\ell \in \mathcal{L}(\mathcal{T}(\text{RC}(v)))} N_{[n]-\{\ell\}}}{P^{|\mathcal{L}(\mathcal{T}(\text{RC}(v)))|-1}},$$

which implies that

$$|D(v)| \le \frac{\prod_{\ell \in \mathcal{L}(\mathcal{T}(v))} N_{[n]\setminus\{\ell\}}}{P^{|\mathcal{L}(\mathcal{T}(v))|-1}}.$$

Further, it is easy to see that

$$|D(v)| \le \min(|D(\textsc{lc}(v))|, |D(\textsc{rc}(v))|),$$

which by induction implies that

$$|D(v)| \le \min_{\ell \in \mathcal{L}(\mathcal{T}(v))} N_{[n]-\{\ell\}}.$$

Property 3 is thus verified.

Finally, we verify property 1. By induction, we have

$$\pi_{\textsc{label}(\textsc{lc}(v))}(J \setminus C(\textsc{lc}(v))) \subseteq D(\textsc{lc}(v)),$$
$$\pi_{\textsc{label}(\textsc{rc}(v))}(J \setminus C(\textsc{rc}(v))) \subseteq D(\textsc{rc}(v)).$$

This along with the fact that $\textsc{label}(\textsc{lc}(v)) \cap \textsc{label}(\textsc{rc}(v)) = \textsc{label}(v)$ implies that

$$\pi_{\textsc{label}(v)}(J \setminus C(\textsc{lc}(v)) \cup C(\textsc{rc}(v))) \subseteq D(\textsc{lc}(v))_{\textsc{label}(v)} \cap D(\textsc{rc}(v))_{\textsc{label}(v)} = G \uplus D(v).$$

Further, every tuple in $(J \setminus C(\textsc{lc}(v)) \cup C(\textsc{rc}(v)))$ whose projection onto $\textsc{label}(v)$ is in $G$ also belongs to $D(\textsc{rc}(v)) \bowtie_G D(\textsc{lc}(v))$. This implies that $\pi_{\textsc{label}(v)}(J \setminus C(v)) = D(v)$, as desired.

For the run time complexity of Algorithm 1, we claim that for every node $x$, we need time $O(n|C(x)| + n|D(x)|)$. To see this note that for each node $x$, the lines 4, 5, 7, and 9 of the algorithm can be computed in that much time using hashing. Using property (3) above, we have a (loose) upper bound of $O(nP + n \min_{l \in \mathcal{L}(\mathcal{T}(x))} N_{[n]\setminus\{l\}})$ on the run time for node $x$. Summing the run time over all the nodes in the tree gives the claimed run time. □

## 6 AN ALGORITHM FOR ALL JOIN QUERIES

This section presents our algorithm for proving the AGM's inequality whose runtime matches the bound.

Theorem 6.1. *Let $H = (V, E)$ be a hypergraph representing a natural join query. Let $\mathbf{x} = (x_e)_{e \in E}$ be an arbitrary point in the fractional cover polyhedron*

$$\sum_{e: v \in e} x_e \ge 1, \text{ for any } v \in V$$
$$x_e \ge 0, \text{ for any } e \in E.$$

*For each $e \in E$, let $R_e$ be a relation of size $N_e$ (number of tuples in the relation). Then,*

- *(a) The join $\bowtie_{e \in E} R_e$ has size (number of tuples) bounded by*

$$\left|\bowtie_{e \in E} R_e\right| \le \prod_{e \in E} N_e^{x_e}.$$

- *(b) Furthermore, the join $\bowtie_{e \in E} R_e$ can be computed in time*

$$O\left(n^2 m + n^2 \sum_{e \in E} N_e + mn \prod_{e \in E} N_e^{x_e}\right).$$

*Remark 6.2.* In the runtime above, $n^2 m$ is the query preprocessing time, $n^2 \sum_{e \in E} N_e$ is the data preprocessing time, and $mn \prod_{e \in E} N_e^{x_e}$ is the query evaluation time. If all relations in the database are indexed in advance to satisfy three conditions (HTw), $w \in \{1, 2, 3\}$, in Section 6.3, then we can remove the term $n^2 \sum_{e \in E} N_e$ from the runtime. Also, the fractional cover $\mathbf{x}$ should be set to the best

fractional cover in terms of the linear objective $\sum_{e\in E}(\log N_e)\cdot x_e$. The data-preprocessing time of $\tilde{O}(n^2\sum_{e\in E}N_e)$ is for a single known query. If we were to index all relations in advance without knowing which queries to be evaluated, then the advance-indexing takes $\tilde{O}(n\cdot n!\sum_{e\in E}N_e)$-time. This price is paid once, up-front, for an arbitrary number of future queries.

Before turning to our algorithm and proof of this theorem, we observe that a consequence of this theorem is the following algorithmic version of the discrete version of BT inequality.

Corollary 6.3. *Let $S\subset\mathbb{Z}^n$ be a finite set of $n$-dimensional grid points. Let $\mathcal{F}$ be a collection of subsets of $[n]$ in which every $i\in[n]$ occurs in exactly $d$ members of $\mathcal{F}$. Let $S_F$ be the set of projections $\mathbb{Z}^n\to\mathbb{Z}^F$ of points in $S$ onto the coordinates in $F$. Then,*

$$|S|^d\le\prod_{F\in\mathcal{F}}|S_F|. \tag{3}$$

*Furthermore, given the projections $S_F$, we can compute $S$ in time*

$$O\left(n^2|\mathcal{F}|+n^2\sum_{F\in\mathcal{F}}|S_F|+|\mathcal{F}|n\left(\prod_{F\in\mathcal{F}}|S_F|\right)^{1/d}\right).$$

Recall that the LW inequality is a special case of the BT inequality. Hence, our algorithm proves the LW inequality as well.

### 6.1 Main Ingredients of the Algorithm

We will first describe the algorithm using B-tree-like indices for input relations. This choice will add a log-factor to the claimed runtime of Theorem 6.1. This log-factor will be hidden in the $\tilde{O}(\cdot)$ in this and the next subsection. The runtime *is* worst-case, however. Then, in Section 6.3, we describe a simple replacement of the search tree indices by a collection of hash indices that knocks off the log-factor, proving Theorem 6.1. However, the runtime using typical hash indices is in an amortized sense. For a true worst-case runtime as claimed in Theorem 6.1, we will have to use the lazy-array technique as shown in Reference [26], at the cost of enormous space overhead. Another advantage of describing the algorithm using the search tree structures is that it will be clear how the total order of attributes affects the overall runtime.

There are three key ingredients in the algorithm (Algorithm 4) and its analysis:

(1) We first build a *search tree* for each relation $R_e$, which will be used throughout the algorithm. This step is responsible for the (near-) linear term $\tilde{O}(n^2\sum_{e\in E}N_e)$ in the runtime. The search tree for each relation is built using a particular ordering of attributes in the relation called *the total order*.
(2) Suppose we have two relations $R$ and $S$ on the same set of attributes and we want to compute $R\bowtie S=R\cap S$. If the search trees for $R$ and $S$ have already been built, then the intersection can be computed in time $\tilde{O}(k\min\{|R|,|S|\})$ where $k$ is the number of attributes in $R$, because we can traverse every tuple of the smaller relation and check (using binary-search) into the search structure for the larger relation. (Later, when hash indices are used, the intersection can be computed in time $O(k\min\{|R|,|S|\})$.) Also note that, for any two non-negative numbers $r$ and $s$ such that $r+s\ge 1$, we have $\min\{|R|,|S|\}\le|R|^r|S|^s$.
(3) The third ingredient is based on *unrolling* sums using generalized Hölder inequality Equation (4).

We make use of the following form of Hölder's inequality, which was also attributed to Jensen. (See the classic book "Inequalities" by Hardy et al. [41], Theorem 22 on page 29.)

Lemma 6.4 (Hardy ET AL. [41]). *Let $m, n$ be positive integers. Let $y_1, \ldots, y_n$ be non-negative real numbers such that $y_1 + \cdots + y_n \geq 1$. Let $a_{ij} \geq 0$ be non-negative real numbers, for $i \in [m]$ and $j \in [n]$. With the convention $0^0 = 0$, we have:*

$$\sum_{i\in[m]} \prod_{j\in[n]} a_{ij}^{y_j} \leq \prod_{j\in[n]} \left( \sum_{i\in[m]} a_{ij} \right)^{y_j}. \tag{4}$$

Recall the following notation from Section 3. 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 tuple $\mathbf{t}_{\bar{S}}$, define the $\mathbf{t}_{\bar{S}}$*-section* of $R$ as

$$R[\mathbf{t}_{\bar{S}}] = \pi_{\bar{A}\setminus\bar{S}} \left( R \ltimes \{\mathbf{t}_{\bar{S}}\} \right).$$

In particular, $R[\mathbf{t}_\emptyset] = R$.

Finally, we remark that the presentation of both our algorithm and its analysis follows that from Reference [63] instead of our original presentation [62], since the former is a bit cleaner.

### 6.2 Description and Analysis of the Algorithm

We begin with an informal description of our main algorithm that computes any join query $q$. Let $q$ be defined by the hypergraph $H = (V, E)$; i.e., we have relations $R_e$ for each $e \in E$ and $q = \bowtie_{e\in E} R_e$. Let $f \in E$ be any relation in $q$. Let

$$\bar{f} = V \setminus f$$

be the set of attributes not in $R_f$. Define

$$L = \pi_{\bar{f}}(q).$$

Then, note that we have

$$q = \bigcup_{\mathbf{t}\in L} q[\mathbf{t}].$$

Further, note that both $L$ and $q[\mathbf{t}]$ are join problems with strictly smaller join queries (and hence can be computed recursively). Indeed, one can verify that

$$L = \bowtie_{e\in E: e\cap\bar{f}\neq\emptyset} \pi_{\bar{f}}(R_e)$$

and for every $\mathbf{t} \in L$,

$$q[\mathbf{t}] = \bowtie_{e\in E: e\cap f\neq\emptyset} \pi_f(R_e[\mathbf{t}]).$$

We then note that

$$q[\mathbf{t}] = R_f \cap Z, \tag{5}$$

where

$$Z = \left( \bowtie_{e\in E\setminus\{f\}: e\cap f\neq\emptyset} \pi_f(R_e[\mathbf{t}]) \right).$$

For the base case of recursion, we have $|V| = 1$ in which case we note that

$$\bowtie_{e\in E} R_e = \bigcap_{e\in E} R_e. \tag{6}$$

Next, we note that we can compute the intersection of $m$ sets $S_1, \ldots, S_m$ in time $O(m \cdot \min_{i\in[m]} |S_i|)$, assuming we can perform membership query in any set in time $O(1)$: indeed the algorithm is to check if every element in the smallest set is also present in the other $m - 1$ sets.

**ALGORITHM 2:** Recursive-Join$(H = (V, E), \bowtie_{e \in E} R_e, \mathbf{y})$

1: **if** $|V| = 1$ **then**
2: **return** $\cap_{e \in E} R_e$
3: $Q \leftarrow \emptyset$ // $Q$ is the set of tuples to be returned
4: Pick $f \in E$ such that $f$ is a suffix of the total order on $V$
5: $\bar{f} \leftarrow V \setminus f$
6: $E_1 \leftarrow \{e \in E | e \cap \bar{f} \neq \emptyset\}$
7: $E_2 \leftarrow \{e \in E | e \cap f \neq \emptyset\}$
8: $L \leftarrow$ Recursive-Join$(H_1 = (\bar{f}, E_1 \setminus f), \bowtie_{e \in E_1} \pi_{\bar{f}}(R_e), (y_e)_{e \in E_1})$ //$E_1 \setminus f \stackrel{def}{=} \{e \setminus f | e \in E_1\}$
9: **for** every $\mathbf{t} \in L$ **do**
10: $Q[\mathbf{t}] \leftarrow \emptyset$
11: **if** $y_f < 1$ and $|R_f| \geq \prod_{e \in E_2 \setminus \{f\}} |R_e[\mathbf{t}]|^{\frac{y_e}{1-y_f}}$ **then**
12: $Z \leftarrow$ Recursive-Join$\left(H_2 = (f, (E_2 \cap f) \setminus \{f\}), \bowtie_{e \in E_2 \setminus \{f\}} \pi_f(R_e[\mathbf{t}]), \left(\frac{y_e}{1-y_f}\right)_{e \in E_2 \setminus \{f\}}\right)$ //
$E_2 \cap f \stackrel{def}{=} \{e \cap f | e \in E_2\}$
13: **for** each $\mathbf{t}' \in Z$ **do** //Filter tuples in $Z$ against $R_f$
14: **if** $\mathbf{t}' \in R_f$ **then**
15: $Q[\mathbf{t}] \leftarrow Q[\mathbf{t}] \cup \{\mathbf{t}'\}$
16: **else**
17: **for** every $\mathbf{t}' \in R_f$ **do** //Filter each tuple in $R_f$ against other relations
18: **if** $\pi_{e \cap f}(\mathbf{t}') \in (R_e[\mathbf{t}])$ for every $e \in E_2$ **then**
19: $Q[\mathbf{t}] \leftarrow Q[\mathbf{t}] \cup \{\mathbf{t}'\}$
20: $Q \leftarrow Q \cup \{\mathbf{t}\} \times Q[\mathbf{t}]$
21: **return** $Q$

This is enough to compute Equation (6) (assuming we know the sizes of the relations $N_e$ for every $e \in E$).

The main idea in the general case is to apply the same trick as above to Equation (5) but this looks a bit problematic as $Z$ is defined recursively, and hence, we do not know $|Z|$ upfront. Of course, one can materialize $Z$ and then figure out $\min(|R_f|, |Z|)$. However, this approach leads to an unacceptable runtime. Here is the trick we use—even though we do not know $|Z|$, we do have an upper bound on it, namely, the AGM bound. Thus, if we know all the sizes $|\pi_f(R_e[\mathbf{t}])|$, then we can use the AGM bound to obtain a bound $\tilde{Z} \geq |Z|$. In our algorithm, we use $\tilde{Z}$ to determine which of $R$ or $Z$ we use as a filter. In particular, note that if $|R_f| < \tilde{Z}$, then we do not even need to compute $Z$ (since checking for membership in $Z$ can be done in $O(m)$ time by checking for membership in each $\pi_f(R_e[\mathbf{t}])$).

Algorithm 2 present the details of the outline above. We would like to remark that Algorithm 2 is not the obvious generalization of Algorithm 1. In particular, the division of light and heavy keys happens implicitly in the second comparison in line 11 in Algorithm 2 (as opposed to Algorithm 1, where this division happens explicitly).

In the informal description of the algorithm, we stated certain assumption on how the relations are stored. Before we present the runtime analysis of Algorithm 2, we present a sufficient condition that is enough to satisfy all those assumption on the data structures (as well as define the notion of the *suffix* relation as used in line 4 in Algorithm 2).

We will assume that before Algorithm 2 is run, we have computed an ordering of the attributes $\sigma = A_1, A_2, \ldots, A_n$ such that every relation is stored in a B-Tree whose sort-order is *consistent* with

$\sigma$. In particular if the sort order for $R_e$ is $A_1^e, \ldots, A_k^e$, then $A_i^e$ comes before $A_j^e$ in $\sigma$ for any $i \leq j$. Further, this means that the first level of a B-Tree for $R_e$ has all the distinct values of $A_1^e$ and for each value $a$ in the first level of the B-Tree a sub-tree is define recursively for $R_e[a]$.

Informally, we will call $\sigma$ to be a *compatible total ordering* if the B-Trees as described above are enough to support all the queries issued by Algorithm 2. More formally, $\sigma$ is compatible if there exists an $f = \{A_{k+1}, \ldots, A_n\}$ for some $0 \leq k < n$. We will call such an $f$ to be *suffix* of $\sigma$: if there are multiple ones, then we will have one designated suffix. Line 4 can pick any such $f$: each choice will define a different compatible total ordering. Further, $\sigma$ restricted to $\bar{f}$ and $f$ are compatible total orderings for $\rhd\lhd_{e \in E_1} \pi_{A_1,\ldots,A_k}(R_e)$ and $\rhd\lhd_{e \in E_2 \setminus \{f\}} \pi_f(R_e[\mathbf{t}])$ (for every possible $\mathbf{t} \in L$), respectively. We illustrate this definition with an example:

*Example 6.1.* Consider a join query with $H = (\{1,2,3,4,5\}, \{e_1 = (1,3,5), e_2 = (2,3,4)\})$. Note that in this case the total ordering $\sigma = 1,2,3,4,5$ is not compatible, because neither $e_1$ nor $e_2$ are suffixes of $\sigma$. On the other hand, $2,4,1,3,5$ and $1,5,2,3,4$ are compatible orderings (as are all ordering where one permutes the first two and last three attributes independently in both the orderings).

We defer the computation of such a compatible total ordering (a greedy choice suffices) to Section 6.2.2, where we also observe (see Lemma 6.6) that such a total ordering $A_1, \ldots, A_n$ implies that all the queries issued by Algorithm 2 are of the following form (and the claimed runtimes follows from the fact that we are using B-Tree data structures for each $R_e$ that is consistent with $A_1, \ldots, A_n$):

- (ST1) Decide whether $\mathbf{u}_{\{A_1,\ldots,A_i\}} \in \pi_{\{A_1,\ldots,A_i\}}(R_e)$ in $\tilde{O}(i)$-time (by "stepping down" the tree along the $u_{A_1}, \ldots, u_{A_i}$ path).
- (ST2) Query the size $|\pi_{\{A_{i+1},\ldots,A_j\}}(R_e[\mathbf{u}_{\{A_1,\ldots,A_i\}}])|$ in $O(i)$ time.
- (ST3) List all tuples in the set $\pi_{\{A_{i+1},\ldots,A_j\}}(R_e[\mathbf{u}_{\{A_1,\ldots,A_i\}}])$ in time linear in the output size if the output is not empty.

*6.2.1 Runtime Analysis of Algorithm 2.* Assuming that the relations are stored in B-Trees according to a compatible total ordering and hence, satisfies (ST1)–(ST3), we will prove the following bounds on the runtime of Algorithm 2.

Theorem 6.5. *Consider a call* Recursive-Join$(H, q, \mathbf{y})$ *to Algorithm 2. Then,*

- *(a) The procedure outputs a relation $Q$ with at most the following number of tuples*
$$AGM(H, q, \mathbf{y}) := \prod_{e \in E} |R_e|^{y_e}.$$
- *(b) Furthermore, the procedure runs in time $\tilde{O}(mn \cdot AGM(H, q, \mathbf{y}))$.*

Proof. We will prove this result by induction on $|V|$.

We remark that the proof presented here follows more closely the presentation in Reference [63] than the argument in Reference [62] (the former is a simplification of the latter).

For the base case, we have $|V| = 1$. In this case, by line 2, we have that $Q = \cap_{e \in E} R_e$. As discussed earlier, this implies that
$$|Q| \leq \min_{e \in E} |R_e| \leq \prod_{e \in E} |R_e|^{y_e},$$
where the second inequality follows from the fact that $\mathbf{y}$ is a valid edge cover (and hence $\sum_{e \in E} y_e \geq 1$). Also as discussed earlier, the intersection can be computed in time $O(m \min_{e \in E} |R_e|)$. The same argument proves the claimed runtime for the base case.

For the inductive hypothesis, we assume that the result holds for every $|V| \leq n-1$. Lines 4–7 can be implemented in $O(mn)$ time. We next consider line 8. We first note that since $\mathbf{y}$ is a valid edge cover for $H$, the vector $(y_e)_{e \in E_1}$ is also a valid edge cover for $H_1$. Since $V(H_1) = \overline{f}$ and $|\overline{f}| < n$, by the inductive hypothesis, we can compute $L$ in time

$$\tilde{O}\left(m \cdot (n - |f|) \cdot \prod_{e \in E_1} |\pi_{\overline{f}}(R_e)|^{y_e}\right) \leq \tilde{O}\left(m \cdot (n - |f|) \cdot \prod_{e \in E} |R_e|^{y_e}\right). \tag{7}$$

Now, we consider the iteration for each $\mathbf{t} \in L$. We claim that for each such $\mathbf{t}$, we have that

$$|Q[\mathbf{t}]| \leq \prod_{e \in E_2} |R_e[\mathbf{t}]|^{y_e} \tag{8}$$

and can be computed in time to within an $\tilde{O}(m \cdot |f|)$ factor of the above bound. We defer the proofs of these claims till later and use these bounds to complete the proof. We begin with a bound on the final output size.

To facilitate the proof, we will need the following new notation. For any subset $h \subseteq V$, define

$$\bar{h} = V \setminus h,$$

$$E_1(h) = \{e | e \cap \bar{h} \neq \emptyset\},$$

$$E_2(h) = \{e | e \cap h \neq \emptyset\},$$

and

$$L(h) = \bowtie_{e \in E_1(h)} \pi_{\bar{h}}(R_e).$$

Note that in Algorithm 2, we have $E_1 = E_1(f), E_2 = E_2(f)$ and $L = L(f)$. For notational simplicity, we assume that the total order is $1, 2, \ldots, n$ and that $f = \{k+1, \ldots, n\}$. Define

$$g = f \cup \{k\}.$$

Now, note that any $\mathbf{t} \in L(f)$ can be represented as $(\mathbf{t}', t_k)$ such that $\mathbf{t}' \in L(g)$. (Below the sum over $t_k$ is over all $t_k$ such that $(\mathbf{t}', t_k) \in L(f)$. For clarity, we do not explicitly state this condition.) Note that

$$\begin{aligned}
|Q| &= \sum_{\mathbf{t} \in L(f)} |Q[\mathbf{t}]| \\
&\leq \sum_{\mathbf{t} \in L(f)} \prod_{e \in E_2(f)} |R_e \ltimes \mathbf{t}|^{y_e} && (9) \\
&= \sum_{\mathbf{t}' \in L(g)} \sum_{t_k} \prod_{e \in E_2(f)} |R_e \ltimes (\mathbf{t}', t_k)|^{y_e} && (10) \\
&\leq \sum_{\mathbf{t}' \in L(g)} \sum_{t_k} \prod_{e \in E_2(g)} |R_e \ltimes (\mathbf{t}', t_k)|^{y_e} && (11) \\
&= \sum_{\mathbf{t}' \in L(g)} \prod_{e \in E_2(g): k \notin e} |R_e \ltimes \mathbf{t}'|^{y_e} \sum_{t_k} \prod_{e \in E_2(g): k \in e} |R_e \ltimes (\mathbf{t}', t_k)|^{y_e} && (12) \\
&\leq \sum_{\mathbf{t}' \in L(g)} \prod_{e \in E_2(g): k \notin e} |R_e \ltimes \mathbf{t}'|^{y_e} \prod_{e \in E_2(g): k \in e} \left(\sum_{t_k} |R_e \ltimes (\mathbf{t}', t_k)|\right)^{y_e} && (13)
\end{aligned}$$

$$\leq \sum_{\mathbf{t}' \in L(g)} \prod_{e \in E_2(g): k \notin e} |R_e \ltimes \mathbf{t}'|^{y_e} \prod_{e \in E_2(g): k \in e} (|R_e \ltimes \mathbf{t}'|)^{y_e} \tag{14}$$

$$= \sum_{\mathbf{t}' \in L(g)} \prod_{e \in E_2(g)} |R_e \ltimes \mathbf{t}'|^{y_e}. \tag{15}$$

In the above, Equation (9) follows from Equation (8), and the fact that for any $R_e$ and $\mathbf{t}$, $|R[\mathbf{t}]| = |R_e \ltimes \mathbf{t}|$. Equation (10) follows from the definition of $g$ and $L(g)$. Equation (11) follows from the fact that $E_2(f) \subseteq E_2(g)$.[3] Equation (12) follows from the fact that for any $e \in E_2(g)$ such that $k \notin e$, $R_e \ltimes (\mathbf{t}', t_k) = R_e \ltimes \mathbf{t}'$. Equation (13) follows from Hölder's inequality (and the fact that by the edge covering constraint for $k$, we have $\sum_{e \in E_2(g): k \in e} y_e = \sum_{e \in E: k \in e} y_e \geq 1$). Equation (14) follows from the fact that the sum over $t_k$ is only for those $t_k$ such that $(\mathbf{t}', t_k) \in L(f)$.

Now note that Equation (15) has the same form as Equation (9), except we have replaced $f$ by $g$ that is one bigger than $f$. Applying the same argument $k - 1$ times proves the claimed bound[4] on the size of $Q$ (i.e., part (a) of the theorem). Finally, it can be verified that each iteration of the for loop in line 9 (that corresponds to a $\mathbf{t} \in L$) can be implemented in time $O(m|f||Q[\mathbf{t}]|)$. This with the time bound in Equation (7) proves part (b) of the theorem.

To complete the proof, we argue Equation (8). (We will only argue the claimed bound on the size of $Q[\mathbf{t}]$– the claim on the runtime follows by a similar argument and noting that each step in the outer for loop in Algorithm 2 takes time $O(m \cdot |f|)$.) We first consider the case that $y_f \geq 1$. In that case Algorithm 2 computes $Q[\mathbf{t}]$ inside the for loop in line 17. In this case it is easy to see that $|Q[\mathbf{t}]| \leq |R_f|$ (which along with the fact that $y_f \geq 1$) implies Equation (8).

Next, we consider the case of $y_f < 1$. We claim that

$$|Q[\mathbf{t}]| \leq \min\left( |R_f|, \prod_{e \in \mathrm{E}_2 \setminus \{f\}} |R_e[\mathbf{t}]|^{\frac{y_e}{1-y_f}} \right). \tag{16}$$

The above along with the fact that for $a, b \geq 0$ we have $\min(a, b) \leq a^{y_f} \cdot b^{1-y_f}$ proves Equation (8). Finally, to complete the proof, we argue Equation (16). Towards this end note that if Algorithm 2 runs the for loop in line 17, then $|Q[\mathbf{t}]| \leq |R_f|$ (note that since $y_f < 1$ and we have executed line 17, in this case the second inequality in line 11 is not satisfied). This implies the bound in Equation (16). Now consider the case when Algorithm 2 runs the for loop in line 13. Note that in this case we have $Q[\mathbf{t}] = Z \cap R_f$, which implies that $|Q[\mathbf{t}]| \leq \min(|Z|, |R_f|)$. To prove Equation (16), we claim that by induction $|Z| \leq \prod_{e \in \mathrm{E}_2 \setminus \{f\}} |R_e[\mathbf{t}]|^{y_e/(1-y_f)}$. To see the latter claim, we only need to argue that $\mathbf{y}' = (y_e/(1-y_f))_{e \in E_2 \setminus \{f\}}$ is a valid edge cover for $H_2$ (since $V(H_2) = f$ and $|f| < n$). Indeed, since $\mathbf{y}$ is a valid edge cover for $H$, we have that for every $u \in f$, we have

$$1 \leq \sum_{e \in E: u \in e} y_e = \sum_{e \in E_2: u \in e} y_e = y_f + \sum_{e \in E_2 \setminus \{f\}} y_e.$$

Rearranging the above gives

$$1 \leq \sum_{e \in E_2 \setminus \{f\}} \frac{y_e}{1 - y_f},$$

[3] There is one small technicality that we did not explicitly address: in particular, Equation (11) is not true if for some $(\mathbf{t}', t_k) \in L(f)$ and $e \in E_2(g)$ we have that $R_e \ltimes (\mathbf{t}', t_k) = \emptyset$. To be technically correct the product over $e \in E_2(g)$ should only consider those $e$ with non-empty $R_e \ltimes (\mathbf{t}', t_k)$. The same holds for the product over $e \in E_2(g) : k \in e$ in Equation (12). However, in Equation (13), we no longer require this qualification and rest of the steps are all fine. For the sake of clarity, we do explicitly add the qualification on $e$ in Equations (11) and (12).

[4] Indeed, at the end, we have the bound $\sum_{\mathbf{t}' \in L([n])} \prod_{e \in E_2([n])} |R_e \ltimes \mathbf{t}'|^{y_e}$, which is $\prod_{e \in E} |R_e|^{y_e}$, since we have $E_1([n]) = \emptyset$ and $E_2([n]) = E$.

which shows that $\mathbf{y}'$ is a valid edge cover for $H_2$, as desired. □

*6.2.2 More on Compatible Total Orderings.* We now go back to the assumption that Algorithm 2 has access to a compatible total ordering on $V$. In this section, we describe how we can compute such an ordering. For the LFTJ algorithm [80], it turns out that *all* total orderings of $V$ are compatible. Thus, one might hope that this might also be the case for Algorithm 2. However, as we showed in Example 6.1, this is *not* the case.

We note that, as stated, the definition of a compatible total ordering is a sufficient condition on the total ordering for Algorithm 2. However, we argue now that Example 6.1 also shows that it is in some sense also necessary. For example, if one picks $\sigma = 1, 2, 3, 4, 5$ as the total order (for the instance in Example 6.1) then no matter which of $e_1 = (1, 3, 5)$ nor $e_2 = (2, 3, 4)$ Algorithm 2 picks as the suffix in line 4, one cannot make the runtime analysis from Section 6.1 to work. To see this assume that $e_1$ is chosen as the suffix in line 4 (the argument for $e_2$ being the suffix is similar and omitted). In this case, note that when computing $L$, we have to access $\pi_{2,4}(R_{e_2})$. However, in this case (ST3) is no longer satisfied (since the B-Tree for $R_{e_2}$ has the sort order $2, 3, 4$ and thus the attribute 3 is in between 2 and 4).

*Compatible Ordering Satisfies (ST1)-(ST3).* Before presenting an algorithm to compute a compatible total ordering, we record a simple observation that makes compatible total orderings effective for our purposes. In particular, we note that

Lemma 6.6. *Let $\sigma$ be a compatible total ordering and let all the relations be stored in B-Trees consistent with $\sigma$. Then the B-Trees satisfy (ST1), (ST2), and (ST3).*

Sketch. The claims on the runtimes in (ST1)–(ST3) follow immediately from the kinds of queries they represent and the fact that all relations are stored in B-Trees consistent with $\sigma$ (queries in (ST2) can be answered if the B-Trees also store the number of tuples stored under each sub-tree, which is easy to do during the stage when the B-Trees are built in near-linear time). Thus, all we need to argue is that all queries issued by Algorithm 2 falls into one of three categories. One can argue this by induction on recursion depth. We now informally argue that this is the case. In Algorithm 2, we make queries to the B-Trees at the following locations:

- (Line 2) This will need queries of type (ST2) to figure out the smallest list, then queries of type (ST3) to list all elements in the smallest set and finally filtering queries will be of type (ST1).
- (Line 11) The queries are of the form (ST2).
- (Line 14) The queries are of the form (ST1).
- (Line 17) The queries are of the form (ST3).
- (Line 18) The queries are of the form (ST1). Note that checking if $\pi_{e\cap f}(\mathbf{t}') \in R_e[\mathbf{t}]$ is equivalent to checking if $(\pi_{e\cap f}(\mathbf{t}'), \pi_{e\setminus f}(\mathbf{t})) \in R_e$ so this claim is correct.

Finally, we note that in any recursive call a relation $R_e$ is $R'_e[\mathbf{u}_{A_i,\ldots,A_j}]$, where $j \geq i$ and $R'_e$ is one of the original relations for some $i \in [n]$. □

*An Algorithm to Compute a Compatible Total Ordering.* If one looks at Example 6.1, then it suggests the following natural greedy algorithm to compute such an ordering: pick any edge $f \in E$; place all attributes in $\bar{f}$ before those in $f$ and recurse on both parts. Further, designate $f$ as the suffix on $V$. Algorithm 3 presents the details.

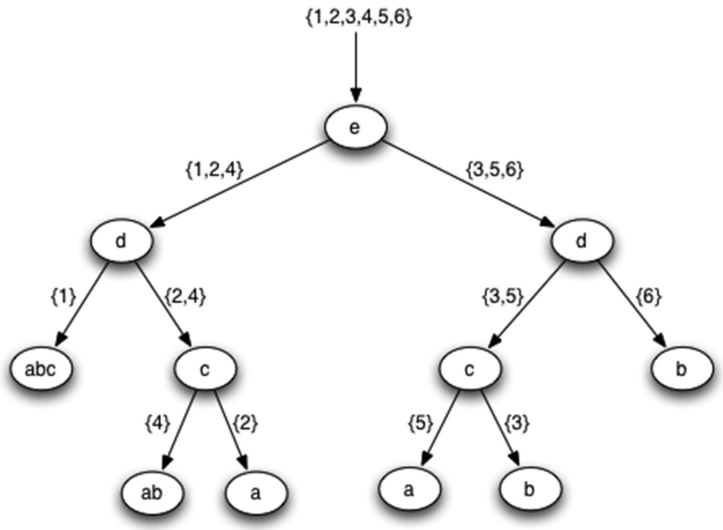


Fig. 1. A run of Algorithm 3 on the instance from Example 6.2. The compatible total ordering is 1, 4, 2, 5, 3, 6.

**ALGORITHM 3:** PRINT-ATTRIBS($H = (V, E)$)

```
1:  if |V| = 1 then
2:      print the attribute in V
3:  else if |E| is just a multiset where each edge is V then
4:      print the attributes in V in arbitrary order
5:      Designate V as the suffix for H
6:  else
7:      Pick an arbitrary f ∈ E and designate it as suffix for H
8:      f̄ ← V \ f
9:      E1 ← {e ∈ E | e ∩ f̄ ≠ ∅}
10:     E2 ← {e ∈ E | e ∩ f ≠ ∅}
11:     PRINT-ATTRIBS(H1 = (f̄, E1 \ f))
12:     PRINT-ATTRIBS(H2 = (f, E2 ∩ f))
```

*Example 6.2.* Consider an example where the hypergraph $H$ has 6 attributes $V = \{1, \ldots, 6\}$, and 5 relations $R_a, R_b, R_c, R_d, R_e$ defined by the following vertex-edge incident matrix **M**.

**M**

| | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 |
| 5 | 1 | 0 | 0 | 0 | 1 |
| 6 | 0 | 1 | 0 | 1 | 1 |

Figure 1 shows the run of Algorithm 3 on the hypergraph $H$ above. Each internal node corresponds to a recursive call and is labeled by the corresponding suffix. The edges in the tree are labeled by the corresponding set of vertices. The total order is 1, 4, 2, 5, 3, 6, which can be read off (in order during a traversal of the tree) as the label of the edges pointing to the leaves.

It is easy to argue by induction on size of $H$ that Algorithm 3 indeed prints a valid compatible total ordering (the main observation is that the recursive calls to $H_1$ and $H_2$ are on graphs that are completely disjoint and hence their total orderings are independent).

Further, it can be verified that Algorithm 3 has at most $n$ recursive calls (as each recursive call partitions the set of vertices $V$ into two non-trivial subsets. Finally, each recursive calls needs $O(mn)$ time. Thus, the overall runtime if $O(mn^2)$. Thus, we have argued that

THEOREM 6.7. *Algorithm 3 computes a compatible total ordering in time $O(mn^2)$.*

*The Overall Algorithm.* For completeness, we present the final overall algorithm that starts from the query $q$ and the relations instance $R_e$ and computes the answer for $q$ in Algorithm 4.

**ALGORITHM 4:** Computing the join $\bowtie_{e\in E} R_e$

**Input:** Hypergraph $H = (V, E)$
**Input:** Fractional cover $\mathbf{x} = (x_e)_{e\in E}$
**Input:** Relations $R_e, e \in E$
1: Compute a total order $\sigma$ of attributes using PRINT-ATTRIBS($H$)
2: Compute a collection of search trees (or hash indices) for all relations that is consistent with $\sigma$
3: **return** RECURSIVE-JOIN($H, \bowtie_{e\in E} R_e, \mathbf{x}$)

Algorithm 4 along with the discussions above proves Theorem 6.1 except the time bounds need an extra log factor. Next, we outline how we can get rid of this log factor to truly prove Theorem 6.1.

### 6.3 Removing the log-factor with Hash Indices

We can build a family of hash indices so that they enforce the same properties as (ST1), (ST2), and (ST3); and, the queries to the various $R_e$ in Algorithm 2 can be performed efficiently. The advantage is the runtime reduction by a log-factor, as we no longer need to do binary search with the search trees. We describe the form of the hash tables constructed by our algorithm. Each hash table is described by a triple $(e, \bar{K}, \bar{A})$, where $e \in E$, $\bar{K} \subseteq e$ is the search key, and $\bar{A} \subseteq e \setminus \bar{K}$ are the value attributes. For each such triple, our algorithm builds three hash tables that each map hash keys $\mathbf{t} \in D^{\bar{K}}$ to one of three data types below.

- (HT1) A hash table that maps $\mathbf{t}$ to a Boolean that is true if $\mathbf{t} \in \pi_{\bar{K}}(R_e)$. Thus, we can decide for any fixed tuple $\mathbf{t} \in D^{\bar{K}}$ whether $\mathbf{t} \in \pi_{\bar{K}}(R_e)$ in time $O(|\bar{K}|)$.
- (HT2) A hash table that maps each $\mathbf{t}$ to $|\pi_{\bar{A}}(R_e[\mathbf{t}])|$, i.e., the number of tuples $\mathbf{u} \in \pi_{\bar{K}\cup\bar{A}}(R_e)$ such that $\mathbf{t}_{\bar{K}} = \mathbf{u}_{\bar{K}}$. For a fixed $\mathbf{t} \in D^{\bar{K}}$, this can be queried in time $O(|\bar{K}|)$.
- (HT3) A hash table that returns all tuples $\mathbf{u} \in \pi_{\bar{A}}(R_e[\mathbf{t}])$ in time linear in the output size (if the output is not empty).

Any such hash table for relation $R_e$ can be built in total time $O(N_e)$. We denote this hash table $\mathrm{HTw}(e, \bar{K}, \bar{A})$ for $w \in \{1, 2, 3\}$. We abuse notation slightly and write $\mathrm{HTw}(e, U, \bar{A})$ for $w \in \{1, 2, 3\}$ when $U \setminus e \neq \emptyset$ by defining $\mathrm{HTw}(i, U, \bar{A}) = \mathrm{HTw}(i, U \cap e, (\bar{A} \setminus U) \cap e)$.

We need only $O(n^2)$ number of hash indices per input relation, bringing the total number of hash indices down to $O(n^2 m)$. We will only need to build 3 hash tables for every triple $(e, \bar{K}, \bar{A})$ such that $\bar{K}$ precede $\bar{A}$ in the total order. Thus, for edge $d$ in Example 6.2, we need to build at most 21 indexes, i.e., three indexes for each of the following nine different key pairs: $[(), (4)]$, $[(), (4, 2)]$, $[(), (4, 2, 6)]$, $[(4), (2)]$, $[(4), (2, 6)]$, $[(4, 2), (6)]$, and $[(4, 2, 6)()]$. (It is less than 21, because some indices are trivial or not defined, e.g., HT1 when $\mathbf{t} = ()$.)

## 7 LIMITS OF STANDARD APPROACHES

### 7.1 Join-project Plans Cannot be Worst-case Optimal

For a given join query $q$, we describe a sufficient syntactic condition for $q$ so that when computed by any join-project plan is asymptotically slower than the worst-case bound. Our algorithm runs within this bound, and so for such $q$ there is an asymptotic runtime gap.

*LW Instances.* Recall that an *LW instance* of the OJ problem is a join query $q$ represented by the hypergraph $(V, E)$, where $V = [n]$, and $E = \binom{[n]}{n-1}$ for some integer $n \geq 2$. Our main result in this section is the following lemma[5]:

Lemma 7.1. *Let $n \geq 2$ be an arbitrary integer. Given any LW-query $q$ represented by a hypergraph $([n], \binom{[n]}{n-1})$, and any positive integer $N \geq 2$, there exist 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 runs in time $\Omega(N^2/n^2)$.*

Before proving the lemma, we note that both the traditional join-tree algorithm and AGM's algorithm are join-project plans, and thus their runtimes are asymptotically worse than the best AGM bound for this instance, which is $|\bowtie_{i\in[n]} R_i| \leq \prod_{i\in[n]} |R_i|^{1/(n-1)} = N^{1+1/(n-1)}$. On the other hand, both Algorithm 1 and Algorithm 4 take $O(N^{1+1/(n-1)})$-time as we have analyzed. In fact, for Algorithm 4, we are able to demonstrate a stronger result: its runtime on this instance is $O(n^2N)$, which is better than what we can analyze for a general instance of this type. In particular, the runtime gap between Algorithm 4 and AGM's algorithm is $\Omega(N)$ for constant $n$.

Proof of Lemma 7.1. In the instances below the domain of any attribute will be $\mathbf{D} = \{0, 1, \ldots, (N-1)/(n-1)\}$. For the sake of clarity, we ignore the integrality issue. For any $i \in [n]$, let $R_i$ be the set of *all* tuples in $\mathbf{D}^{[n]\setminus\{i\}}$ each of which has at most one non-zero value. Then, it is not hard to see that $|R_i| = (n-1)[(N-1)/(n-1)+1] - (n-2) = N$, for all $i \in [n]$; and, $|\bowtie_{i=1}^{n} R_i| = n[(N-1)/(n-1)+1] - (n-1) = N + (N-1)/(n-1) > N$.

A relation $R$ on attribute set $\bar{A} \subseteq [n]$ is called *simple* if $R$ is the set of *all* tuples in $\mathbf{D}^{\bar{A}}$ 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 $|S \bowtie T| \geq (1 + (N-1)/(n-1))^2 = \Omega(N^2/n^2)$.

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. □

Finally, we analyze the runtime of Algorithm 4 directly on this instance without resorting to Lemma 6.4. In our analysis, Hölder's inequality lost some information about the runtime. The following lemma shows that our algorithm and our bound can be better than what we were able to analyze.

Lemma 7.2. *On the collection of instances from the previous lemma, Algorithm 4 runs in time $O(n^2N)$.*

Proof. Without loss of generality, assume that the suffix $f$ in line 4 of Algorithm 2 is $[n] \setminus \{n\}$. Note that in this case, $\bar{f} = \{n\}$. Below, we will assume that the edge cover vector is the all $1/(n-1)$'s vector.

[5]We thank an anonymous PODS'12 referee for sketching to us the argument showing that our example works for all join-project plans rather than just the AGM algorithm and a join-tree algorithm.

The first thing Algorithm 4 does is that it computes the join $L_n = \bowtie_{i\in[n-1]} \pi_{\{n\}}(R_i)$, in time $O(nN)$. Note that $L_n = \mathbf{D}$, the domain. Next, Algorithm 4 goes through each value $a \in L_n$ and decides whether to solve a subproblem. First, consider the case $a > 0$. Here Algorithm 4 estimates a bound for the join $\bowtie_{j\in[n-1]} \pi_{[n-1]}(R_j[a])$. The estimate is 1, because $|\pi_{[n-1]}(R_j[a])| = 1$ for all $a > 0$. Hence, the algorithm will recursively compute this join, which takes time $O(n^2)$ and filter the result against $R_n$. Overall, solving the sub problems for $a > 0$ takes $O(n^2N)$ time. Second, consider the case when $a = 0$. In this case $|\pi_{[n-1]}(R_j[0])| = ((n-2)N-1)/(n-1)$. The subproblem's estimated size bound is

$$\prod_{i\in[n-1]} |\pi_{[n-1]}(R_j[0])|^{\frac{1/(n-1)}{1-1/(n-1)}} = \left[\frac{(n-2)N-1}{(n-1)}\right]^{(n-1)/(n-2)} > N,$$

if $N \geq 4$ and $n \geq 4$. Hence, in this case $R_n$ will be filtered against the $\pi_{[n-1]}(R_j[0])$, which takes $O(n^2N)$ time. □

*Extending Beyond LW Instances.* Using the above results, we give a sufficient condition for when there exist a family of instances $\mathcal{I} = I_1, \ldots, I_N, \ldots$, such that on instance $I_N$ every binary join strategy takes time at least $\Omega(N^2)$, but our algorithm takes $o(N^2)$. Given a hypergraph $H = (V, E)$, we first define some notation. Fix $U \subseteq V$ and call an attribute $v \in V \setminus U$ to be *$U$-relevant* if for all $e$ such that $v \in e$ then $e \cap U \neq \emptyset$; call $v$ *$U$-troublesome* if for all $e \in E$, if $v \in e$ then $U \subseteq e$. Now, we can state our result:

Lemma 7.3. *Given a join query $H = (V, E)$ and some $U \subseteq V$ where $|U| \geq 2$, if there exists $F \subseteq E$ such that $|F| = |U|$ that satisfies the following three properties: (1) each $u \in U$ occurs in exactly $|U| - 1$ elements in $F$, (2) each $v \in V$ that is $U$-relevant appears in at least $|U| - 1$ edges in $F$, (3) there are no $U$-troublesome attributes. Then, there is some family of instances $\mathcal{I}$ such that (a) computing the join query represented by $H$ with a join tree takes time $\Omega(N^2/|U|^2)$ while (b) the algorithm from Section 6 takes time $O(N^{1+1/(|U|-1)})$.*

Given a $(U, F)$ as in the lemma, the idea is to simply to set all those edges in $f \in F$ to be the instances from Lemma 7.1 and extend all attributes with a single value, say $c_0$. Since there are no $U$-troublesome attributes, to construct the result set at least one of the relations in $F$ must be joined. Since any pair $F$ must take time $\Omega(N^2/|U|^2)$ by the above construction, this establishes (a). To establish (b), we need to describe a particular feasible solution to the cover LP whose objective value is $N^{1+1/(|U|-1)}$, implying that the runtime of our proposed algorithm is upper bounded by this value. To do this, we first observe that any attribute not in $U$ takes the value only $c_0$. Then, we observe that any node $v \in V$ that is not $U$-relevant is covered by some edge $e$ whose size is exactly 1 (and so we can set $x_e = 1$). Thus, we may assume that all nodes are $U$-relevant. Then, observe that all relevant attributes can be set by the cover $x_e = 1/(|U| - 1)$ for $e \in F$. This is a feasible solution to the LP and establishes our claim.

### 7.2 Impossibility of Instance Optimality

Given the promising results in the worst case, it is natural wonder if one can obtain a join algorithm whose run time is polynomial in both the size of the query *as well* as the size of the output. More precisely, given a join query $q$ and an instance $I$, can one compute the result of query $q$ on instance $I$ in time $\text{poly}(|q|, |q(I)|, |I|)$. Unfortunately, this is not possible unless NP = RP. We briefly present a proof of this fact below.

The proof is fairly standard: we use the standard reduction of 3SAT to conjunctive queries but with two simple specializations: (i) We reduce from the 3UniqueSAT, where the input formula is either unsatisfiable or has *exactly* one satisfying assignment and (ii) $q$ is a full join query instead

of a general conjunctive query. It is known that 3UniqueSAT cannot be solved in deterministic polynomial time unless NP = RP [77].

For the sake of completeness, we sketch the reduction here. Let $\phi = C_1 \wedge C_2 \wedge \ldots C_m$ be a 3UniqueSAT CNF formula on $n$ variables $a_1, \ldots, a_n$. (W.l.o.g. assume that a clause does not contain both a variable and its negation.) For each clause $C_j$ for $j \in [m]$, create a relation $R_j$ on the variables that occur in $C_j$. The query $q$ is

$$\bowtie_{j \in [m]} R_j.$$

Now define the database instance $I$ as follows: for each $j \in [m]$, $R_j^I$ contains the seven assignments to the variables in $C_j$ that makes it true. Note that $q(I)$ contains all the satisfying assignments for $\phi$: in other words, $q(I)$ has one element if $\phi$ is satisfiable otherwise $q(I) = \emptyset$. In other words, we have $|q(I)| \leq 1$, $|q| = O(m + n)$ and $|I| = O(m)$. Thus an instance optimal algorithm with time complexity $\text{poly}(|q|, |q(I)|, |I|)$ for $q$ would be able to determine if $\phi$ is satisfiable or not in time $\text{poly}(n, m)$, which would imply NP = RP.

The above negative result is in the combined complexity sense, where we take both the query size and input relation sizes as input to the problem. As we have alluded to in Section 2, the more appropriate complexity measure in the database setting is data complexity. In this sense, is it possible to compute $q(I)$ in time $f(|q|) \cdot \text{poly}(|I|, |q(I)|)$, where $f$ is any computable function? The answer is likely no, too, unless W[1] = RFPT. We use the standard reduction from the (unique) $k$-CLIQUE problem [67] to the join query evaluation problem. The (unique) $k$-CLIQUE problem is known to be W[1]-hard under the W[$P$]-randomized reduction with one-sided error [59].

## 8 EXTENSIONS

In Section 8.1, we describe some results on the combined complexity of our approach. In Section 8.2, we observe that our algorithm can be used to compute a relaxed notion of join. In Section 8.3, we show how to extend our algorithm to work for full conjunctive queries. Finally, in Section 8.4, we show how the AGM bound (and hence, our algorithm) is tight for the problem of enumerating induced copies of a fixed subgraph in a given large graph.

### 8.1 Combined Complexity

Given that our algorithms are data-optimal for worst-case inputs it is tempting to wonder if one can obtain an join algorithm whose run time is both query and data optimal in the worst-case. We show that in the special case when each input relation has arity at most 2, we can attain a data-optimal algorithm that is simpler than Algorithm 4 with an asymptotically better query complexity.

*Each Relation has at Most 2 Attributes.* As was mentioned in the introduction, our algorithm in Theorem 6.1 not only has better data complexity than AGM's algorithm (in fact, we showed our algorithm has optimal worst-case data complexity), it has a better query complexity. In this section, we show that for the special case when the join query $q$ is on relations with at most two attributes (i.e., the corresponding hypergraph $H$ is a graph), we can obtain an even better query complexity as in Theorem 6.1 (with the same optimal data complexity).

Without loss of generality, we can assume that each relation contains exactly 2 attributes, because a 1-attribute relation $R_e$ needs to have $x_e = 1$ in the corresponding LP and thus, contributes a separate factor $N_e$ to the final product. Thus, $R_e$ can be joined with the rest of the query with any join algorithm (including the naive Cartesian product based algorithm). In this case, the hypergraph $H$ is a graph that can be assumed to be simple.

We first prove an auxiliary lemma for the case when $H$ is a cycle. We assume that all relations are indexed in advanced, which takes $O(\sum_{e\in E} N_e)$ time. In what follows, we will not include this preprocessing time in the analysis. The following lemma essentially reduces the case when $H$ is a cycle to the case when $H$ is a triangle, a Loomis-Whitney instance with $n = 3$.

Lemma 8.1 (Cycle Lemma). *If $H$ is a cycle, then $\bowtie_{e\in E} R_e$ can be computed in time $O(m\sqrt{\prod_{e\in H} N_e})$.*

Proof. First suppose $H$ is an even cycle, consisting of consecutive edges $e_1 = (1, 2)$, $e_2 = (2, 3), \ldots, e_{2k'} = (2k', 1)$. Without loss of generality, assume

$$N_{e_1} N_{e_3} \cdots N_{e_{2k'-1}} \le N_{e_2} N_{e_4} \cdots N_{e_{2k'}}.$$

In this case, we compute the (cross-product) join

$$R = R_{e_1} \bowtie R_{e_3} \bowtie \cdots \bowtie R_{e_{2k'-1}}.$$

Note that $R$ contains all the attributes. Then, sequentially join $R$ with each of $R_{e_2}$ to $R_{e_{2k'}}$. The total runtime is

$$O\left(k' N_{e_1} N_{e_3} \cdots N_{e_{2k'-1}}\right) \le O\left(m\sqrt{\prod_{e\in H} N_e}\right).$$

Second, suppose $H$ is an odd cycle consisting of consecutive edges $e_1 = (1, 2)$, $e_2 = (2, 3), \ldots,$ $e_{2k'+1} = (2k' + 1, 1)$. If $k' = 1$, then by the Loomis-Whitney algorithm for the $n = 3$ case (Algorithm 1), we can compute $R_{e_1} \bowtie R_{e_2} \bowtie R_{e_3}$ in time $O(\sqrt{N_{e_1} N_{e_2} N_{e_3}})$. Suppose $k' > 1$. Without loss of generality, assume

$$N_{e_1} N_{e_3} \cdots N_{e_{2k'-1}} \le N_{e_2} N_{e_4} \cdots N_{e_{2k'}}.$$

In particular, $N_{e_1} N_{e_3} \cdots N_{e_{2k'-1}} \le \sqrt{\prod_{e\in H} N_e}$, which means the following join can be computed in time $O(m\sqrt{\prod_{e\in H} N_e})$:

$$X = R_{e_1} \bowtie R_{e_3} \bowtie \cdots \bowtie R_{e_{2k'-1}}.$$

Note that $X$ spans the attributes in the set $[2k']$. Let $S = \{2, 3, \ldots, 2k' - 1\}$, and $X_S$ denote the projection of $X$ down to coordinates in $S$; and define

$$W = (\ldots (X_S \bowtie R_{e_2}) \bowtie R_{e_4}) \cdots \bowtie R_{e_{2k'-2}}).$$

Since $R_{e_2} \bowtie R_{e_4} \cdots \bowtie R_{e_{2k'-2}}$ spans precisely the attributes in $S$, the relation $W$ can be computed in time $O(m|X_S|) = O(m|X|) = O(m\sqrt{\prod_{e\in H} N_e})$. Note that

$$|W| \le \min\{N_{e_1} N_{e_3} \cdots N_{e_{2k'-1}}, N_{e_2} N_{e_4} \cdots N_{e_{2k'-2}}\}.$$

We claim that one of the following inequalities must hold:

$$|W| \cdot N_{e_{2k'}} \le \sqrt{\prod_{e\in H} N_e}, \text{ or}$$

$$|W| \cdot N_{e_{2k'+1}} \le \sqrt{\prod_{e\in H} N_e}.$$

Suppose both of them do not hold, then

$$\begin{aligned}\prod_{e\in H} N_e &= (N_{e_1}N_{e_3}\cdots N_{e_{2k'-1}})\cdot(N_{e_2}N_{e_4}\cdots N_{e_{2k'-2}})\cdot N_{e_{2k'}}\cdot N_{e_{2k'+1}}\\ &\geq |W|^2 N_{e_{2k'}}N_{e_{2k'+1}}\\ &= (|W|\cdot N_{e_{2k'}})\cdot(|W|\cdot N_{e_{2k'+1}})\\ &> \prod_{e\in H} N_e,\end{aligned}$$

which is a contradiction. Hence, without loss of generality, we can assume $|W|\cdot N_{2k'} \leq \sqrt{\prod_{e\in H} N_e}$. Now, compute the relation

$$Y = W \bowtie R_{e_{2k'}},$$

which spans the attributes $S \cup \{2k', 2k'+1\}$. Finally, by thinking of all attributes in the set $S \cup \{2k'\}$ as a "bundled attribute," we can use the Loomis-Whitney algorithm for $n = 3$ to compute the join

$$X \bowtie Y \bowtie R_{e_{2k'+1}}$$

in time linear in

$$\begin{aligned}\sqrt{|X|\cdot|Y|\cdot N_{e_{2k'+1}}} &\leq \sqrt{(N_{e_1}N_{e_3}\cdots N_{e_{2k'-1}})\cdot(|W|\cdot N_{e_{2k'}})\cdot N_{e_{2k'+1}}}\\ &\leq \sqrt{(N_{e_1}N_{e_3}\cdots N_{e_{2k'-1}})\cdot(N_{e_2}N_{e_4}\cdots N_{e_{2k'-2}}\cdot N_{e_{2k'}})\cdot N_{e_{2k'+1}}}\\ &= \sqrt{\prod_{e\in H} N_e}.\end{aligned}$$

□

With the help of Lemma 8.1, we can now derive a solution for the case when $H$ is an arbitrary graph. Consider any *basic feasible solution* $\mathbf{x} = (x_e)_{e\in E}$ of the fractional cover polyhedron

$$\begin{aligned}\sum_{v\in e} x_e &\geq 1,\ v \in V\\ x_e &\geq 0,\ e \in E.\end{aligned}$$

It is known that $\mathbf{x}$ is *half-integral*, i.e., $x_e \in \{0, 1/2, 1\}$ for all $e \in E$ (see Schrijver's book [71], Theorem 30.10). However, we will also need a graph structure associated with the half-integral solution; hence, we adapt a known proof [71] of the half-integrality property with a slightly more specific analysis. It should be noted, however, that the following is already implicit in the existing proof.

Lemma 8.2. *For any basic feasible solution $\mathbf{x} = (x_e)_{e\in E}$ of the fractional cover polyhedron above, $x_e \in \{0, 1/2, 1\}$ for all $e \in E$. Furthermore, the collection of edges $e$ for which $x_e = 1$ is a union $\mathcal{S}$ of stars. And, the collection of edges $e$ for which $x_e = 1/2$ form a set $C$ of vertex-disjoint odd-length cycles that are also vertex disjoint from $\mathcal{S}$.*

Proof. First, if some $x_e = 0$, then we remove $e$ from the graph and recurse on $G - e$. The new $\mathbf{x}$ is still an extreme point of the new polyhedron. So, we can assume that $x_e > 0$ for all $e \in E$.

Second, we can also assume that $H$ is connected. Otherwise, we consider each connected component separately.

As usual, let $n = |V|$ and $m = |E|$. The polyhedron is defined on $m$ variables and $n + m$ inequality constraints. The extreme point must be the intersection of exactly $m$ (linearly independent) tight constraints. But the constraints $\mathbf{x} \geq \mathbf{0}$ are not tight as we have assumed $x_e > 0, \forall e$. Hence, there must be $m$ vertices $v$ for which the constraints $\sum_{v\in e} x_e \geq 1$ are tight. In particular, $m \leq n$. Since $H$ is connected, it is either a tree, or has exactly one cycle.

Suppose $H$ is a tree, then it has at least two leaves and at most one non-tight constraint (as there must be $m = n - 1$ tight constraints). Consider the leaf $u$ whose constraint is tight. Let $v$ be $u$'s neighbor. Then $x_{uv} = 1$, because $u$ is tight. If $v$ is tight, then we are done, the graph $H$ is just an edge $uv$. (If there was another edge $e$ incident to $v$, then $x_e = 0$.) If $v$ is not tight, then $v$ is not a leaf. We start from another tight leaf $w \neq u$ of the tree and reason in the same way. Then, $w$ has to be connected to $v$. Overall, the graph is a star.

Next, consider the case when $H$ is not a tree. All $n = m$ vertices has to be tight in this case. Thus, there cannot be a degree-1 vertex for the same reasoning as above. Thus, $H$ is a cycle. If $H$ is an odd cycle, then it is easy to show that the only solution for which all vertices are tight is the all-1/2 solution. If $H$ is an even cycle, then $\mathbf{x}$ cannot be an extreme point, because it can be written as $\mathbf{x} = (\mathbf{y} + \mathbf{z})/2$ for feasible solutions $\mathbf{y}$ and $\mathbf{z}$ (just add and subtract $\epsilon$ from alternate edges to form $\mathbf{y}$ and $\mathbf{z}$). □

Now, let $\mathbf{x}^*$ be an *optimal* basic feasible solution to the following linear program.

$$\begin{aligned} \min & \sum_{e \in E} (\log N_e) \cdot x_e \\ \text{s.t.} & \sum_{v \in e} x_e \geq 1, v \in V \\ & x_e \geq 0, e \in E. \end{aligned}$$

Then $\prod_{e \in E} N_e^{x_e^*} \leq \prod_{e \in E} N_e^{x_e}$ for any feasible fractional cover $\mathbf{x}$. Let $S$ be the set of edges on the stars and $\mathcal{C}$ be the collection of disjoint cycles as shown in the above lemma, applied to $\mathbf{x}^*$. Then,

$$\prod_{e \in E} N_e^{x_e^*} = \left( \prod_{e \in S} N_e \right) \prod_{C \in \mathcal{C}} \sqrt{\prod_{e \in C} N_e}.$$

Consequently, we can apply Lemma 8.1 to each cycle $C \in \mathcal{C}$ and take a cross product of all the resulting relations with the relations $R_e$ for $e \in S$. We just proved the following theorem.

Theorem 8.3. *When each relation has at most two attributes, we can compute the join* $\bowtie_{e \in E} R_e$ *in time* $O(m \prod_{e \in E} N_e^{x_e})$.

### 8.2 Relaxed Joins

We observe that our algorithm can actually evaluate a relaxed notion of join queries. Say we are given a query $q$ represented by a hypergraph $H = (V, E)$ where $V = [n]$. The $m$ input relations are $R_e$, $e \in E$. We are also given a "relaxation" number $0 \leq r \leq m$. Our goal is to output all tuples that agree with at least $m - r$ input relations. In other words, we want to compute $\cup_{S \subseteq E, |S| \geq m-r} \bowtie_{e \in S} R_e$. However, we need to modify the problem to avoid the case that the set of attributes of relations indexed by $S$ does not cover all the attributes in the universe $V$. Toward this end, define the set

$$C(q, r) = \left\{ S \subseteq E \mid |S| \geq m - r \text{ and } \bigcup_{e \in S} e = V \right\}.$$

With the notations established above, we are now ready to define the relaxed join problem.

*Definition 8.4 (Relaxed Join Problem).* Given a query $q$ represented by the hypergraph $H = (V = [n], E)$, and an integer $0 \leq r \leq m$, evaluate

$$q_r \stackrel{\text{def}}{=} \bigcup_{S \in C(q,r)} \left( \bowtie_{e \in S} R_e \right).$$

Before we proceed, we first make the following simple observation: given any two sets $S, T \in C(q, r)$ such that $S \subseteq T$, we have $\bowtie_{e \in T} R_e \subseteq \bowtie_{e \in S} R_e$. This means in the relaxed join problem,

we only need to consider subsets of relations that are not contained in any other subset. In particular, define $\hat{C}(q,r) \subseteq C(q,r)$ to be the largest subset of $C(q,r)$ such that for any $S \neq T \in \hat{C}(q,r)$ neither $S \subset T$ nor $T \subset S$. We only need to evaluate $q_r = \bigcup_{S \in \hat{C}(q,r)} (\rhd\!\lhd_{e \in S} R_e)$.

Given an $S \in \hat{C}(q,r)$, let $\mathsf{LPOpt}(S)$ denote the optimal size bound given by the AGM's fractional cover inequality Equation (2) on the join query represented by the hypergraph $(V,S)$. In particular, $\mathsf{LPOpt}(S) = \prod_{e \in S} N_e^{x_e^*}$, where $\mathbf{x}_S^* = (x_e^*)_{e \in S}$ is an optimal solution to the following linear program called $\mathrm{LP}(S)$:

$$\begin{aligned} \min & \sum_{e \in S} (\log N_e) \cdot x_e \\ \text{subject to} & \sum_{e \in S : i \in e} x_e \geq 1 && \text{for any } i \in V \\ & x_e \geq 0 && \text{for any } e \in S. \end{aligned} \tag{17}$$

*Upper Bounds.* We start with a straightforward upper bound.

Proposition 8.5. *Let $q$ be a join query on $m$ relations and let $0 \leq r \leq m$ be an integer. Then, given sizes of the input relations, the number of output tuples for query $q_r$ is upper bounded by*

$$\sum_{S \in \hat{C}(q,r)} \mathsf{LPOpt}(S).$$

Further, Algorithm 4 evaluates $q_r$ with data complexity linear in the bound above. The next natural question is to determine how good the upper bound is. Before we answer the question, we prove a stronger upper bound.

Given a subset of hyperedges $S \subseteq E$ that *covers* $V$, i.e., $\cup_{e \in S} e = V$, let $\mathsf{BFS}(S) \subseteq S$ be the subset of hyperedges in $S$ that gets a *positive* $x_e^*$ value in an *optimal* basic feasible solution to the linear program $\mathrm{LP}(S)$ defined in Equation (17). (If there are multiple such solutions, then pick any one in a consistent manner.) Call two subsets $S, T \subseteq E$ *bfs-equivalent* if $\mathsf{BFS}(S) = \mathsf{BFS}(T)$. Finally, define $C^*(q,r) \subseteq \hat{C}(q,r)$ as the collection of sets from $\hat{C}(q,r)$, which contains exactly one arbitrary representative from each bfs-equivalence class.

Theorem 8.6. *Let $q$ be a join query represented by $H = (V,E)$, and let $0 \leq r \leq m$ be an integer. The number of output tuples of $q_r$ is upper bounded by $\sum_{S \in C^*(q,r)} \mathsf{LPOpt}(S)$. Further, the query $q_r$ can be evaluated in time*

$$O\left( \sum_{S \in C^*(q,r)} (mn \cdot \mathsf{LPOpt}(S) + \mathrm{poly}(n,m)) \right),$$

*plus the time needed to compute $C^*(q,r)$ from $q$.*

Note that since $C^*(q,r) \subseteq \hat{C}(q,r)$, the bound in Theorem 8.6 is no worse than that in Proposition 8.5. We will show later that the bound in Theorem 8.6 is indeed tight.

Proof of Theorem 8.6. We will prove the result by presenting the algorithm to compute $q_r$. A simple yet key idea is the following. Let $S \neq S' \in \hat{C}(q,r)$ be two different sets of hyperedges with the following property. Define $T \stackrel{\text{def}}{=} \mathsf{BFS}(S) = \mathsf{BFS}(S')$ and let $\mathbf{x}_T^* = (x_i^*)_{i \in T}$ be the projection of the corresponding optimal basic feasible solution to the $(V,S)$ and the $(V,S')$ problems projected down to $T$. (The two projections result in the same vector $\mathbf{x}_T^*$.) The outputs of the joins on $S$ and on $S'$ are both subsets of the output of the join on $T$. We can simply run Algorithm 4 on inputs $(V,T)$ and $\mathbf{x}_T^*$, then prune the output against relations $R_e$ with $e \in S \setminus T$ or $S' \setminus T$. In particular, we only need to compute $\rhd\!\lhd_{e \in T} R_e$ once for both $S$ and $S'$. □

**ALGORITHM 5:** Computing Relaxed Join $q_r$

1: Compute $C^*(q, r)$.
2: $Q \leftarrow \emptyset$.
3: **for** every $S \in C^*(q, r)$ **do**
4: Let $\mathbf{x}^*_S$ be an optimal BFS for LP($S$)
5: Let $T = \{e \in S \mid x^*_e > 0\}$. (Note that $T = \mathsf{BFS}(S)$.)
6: Run Algorithm 4 on $\{x^*_e\}_{e\in T}$ to compute $\phi_T = \bowtie_{e\in T} R_e$.
7: **for** every tuple $\mathbf{t} \in \phi_T$ **do**
8: **if** for at least $m - r$ hyperedges $e \in E$, $\mathbf{t}_e \in R_e$ **then**
9: $Q \leftarrow Q \cup \{\mathbf{t}\}$
10: **return** $Q$

Other than the time to compute $C^*(q, r)$ in the line 1, line 4 needs poly$(n, m)$ time to solve the LP, line 5 needs $O(m)$ time, while by Theorem 6.1, line 6 will take $O(mn \cdot \mathsf{LPOpt}(S) + n^2 m)$ time. Finally, Theorem 6.1 shows that $|\phi_T| \le \mathsf{LPOpt}(S)$,[6] which shows that the loop in line 7 is repeated $\mathsf{LPOpt}(S)$ times and lines 8 and 9 can be implemented in $O(m)$ time and thus, lines 7–9 will take time $O(m \cdot \mathsf{LPOpt}(S))$.

Finally, we argue the correctness of the algorithm. We first note that by line 8, every tuple $\mathbf{t}$ that is output is indeed a correct one. Thus, we have to argue that we do not miss any tuple $\mathbf{t}$ that needs to be output. For the sake of contradiction assume that there exists such a tuple $\mathbf{t}$. Note that by definition of $\hat{C}(q, r)$, this implies that there exists a set $S' \in \hat{C}(q, r)$ such that for every $e \in S'$, $\mathbf{t}_e \in R_e$. However, note that by definition of $C^*(q, r)$, for some execution of the loop in line 3, we will consider $T$ such that $T = \mathsf{BFS}(S')$. Further, by the correctness of Algorithm 4, we have that $\mathbf{t} \in \phi_T$. This implies (along with the definition of $\hat{C}(q, r)$) that $\mathbf{t}$ will be retained in line 8, which is a contradiction. □

It is easy to check that one can compute $C^*$ in time $m^{O(r)}$ (by going through all subsets of $E$ of size at least $m - r$ and performing all the required checks). We leave open the question of whether this time bound can be improved.

*Lower Bound.* We now show that the bound in Theorem 8.6 is tight for some query and some database instance $I$.

We first define the query $q$. The hypergraph is $H = (V = [n], E)$ where $m = n + 1$. The hyperedges are $E = \{e_1, \ldots, e_{n+1}\}$ where $e_i = \{i\}$ for $i \in [n]$ and $e_{n+1} = [n]$. The database instance $I$ consists of relations $R_e$, $e \in E$, all of which are of size $N$. For each $i \in [n]$, $R_{e_i} = [N]$. And, $R_{e_{n+1}} = \bigcup_{i\in[N]} \{N + i\}^n$.

It is easy to check that for any $r > 0$, $q_r(I)$ is the set $R_{e_{n+1}} \cup [N]^n$, i.e. $|q_r(I)| = N + N^n$. Next, we claim that for this query instance $C^*(q, r) = \{\{n + 1\}, [n]\}$. Note that $\mathsf{BFS}(\{n + 1\}) = \{n + 1\}$ and $\mathsf{BFS}([n]) = [n]$, which implies that $\mathsf{LPOpt}(\{n + 1\}) = N$ and $\mathsf{LPOpt}([n]) = N^n$. This along with Theorem 8.6 implies that $|q_r(I)| \le N + N^n$, which proves the tightness of the size bound in Theorem 8.6, as desired.

Finally, we argue that $C^*(q, r) = \{\{n + 1\}, [n]\}$. Toward this end, consider any $T \in \hat{C}(q, r)$. Note that if $(n + 1) \notin T$, then we have $T = [n]$, and since $\mathsf{BFS}(T) = T$ (and we will see soon that for any other $T \in \hat{C}(q, r)$, we have $\mathsf{BFS}(T) \ne [n]$), which implies that $[n] \in C^*(q, r)$. Now consider the case when $(n + 1) \in T$. Note that in this case $T = \{n + 1\} \cup T'$ for some $T' \subset [n]$ such that $|T'| \ge n - r$. Now note that all the relations in $T$ cannot cover the $n$ attributes but $R_{n+1}$ by itself does include

[6] This also proves the claimed bound on the size of $q_r$.

all the $n$ attributes. This implies that $\mathrm{BFS}(T) = \{n+1\}$ in this case. This proves that $\{n+1\}$ is the other element in $C^*(q, r)$, as desired.

### 8.3 Dealing with Full Queries and Simple Functional Dependencies

*Processing Full Conjunctive Queries.* Our goal in this section is to handle a more general class of queries that may contain selections and joins to the same table, which we describe now.

In this section, we follow the notation from Reference [30], and we reproduce it here for the sake of completeness. A *database instance* $I = (\mathcal{U}, \mathcal{R})$ consists of a finite universe of constants $\mathcal{U}$ and a set of relation names $\mathcal{R}$ each over $\mathcal{U}$. A *conjunctive query* has the form $q = R(\bar{A}_0) \leftarrow R_1(\bar{A}_1) \wedge \cdots \wedge R_m(\bar{A}_m)$, where each $\bar{A}_j$ is a tuple of (not necessarily distinct) variables of length $|\bar{A}_j|$. The $R_j$ are relation names (in $\mathcal{R}$), which are also not necessarily distinct. Each variable that occurs in the head $R(\bar{A}_0)$ must also appear in the body $R_j(\bar{A}_j)$ for some $j \in [m]$. We call a conjunctive query *full* if each variable that appears in the body also appears in the head. The set of all variables in $q$ is denoted $\mathrm{var}(q)$. A single relation may occur several times in the body, and so we may have $i_j = i_k$ for some $j \neq k$. The answer of a query $q$ over a database instance $I$ is a set of tuples of arity $|\bar{A}_0|$, which is denoted $q(I)$, and is defined to contain exactly those tuples $\theta(\bar{A}_0)$ where $\theta : \mathrm{var}(q) \to \mathcal{U}$ is any substitution such that for each $j = 1, \ldots, m$, $\theta(\bar{A}_j) \in R_j$.

Gottlob et al. [30] (GLVV henceforth) derived upper and lower bounds on the size of a conjunctive query building on the ideas of AGM. Since we are only dealing with full conjunctive queries without functional dependencies, we will use the linear programming form of GLVV's bound that was explicitly derived in Grohe [35]. To obtain an upper bound, we construct a hypergraph $H = (V, E)$ as before, where $V = \mathrm{var}(q)$. There is an edge $e_j$ for each relation name $R_j$, where $e_j$ is the set of variables in $\bar{A}_j$.

Let $\mathbf{x} = (x_e)_{e \in E}$ be an arbitrary fractional cover for $H$. Let $N_j = |R_j|$ be the input relation sizes. (Note that since the $R_j$ are not necessarily distinct, the $N_j$ are not independent.) GLVV's result for full conjunctive queries can be stated as follows: for any input database instance $I$, we have $|q(I)| \leq \prod_{e \in E} N_e^{x_e}$.

Now, Algorithm 4 can be applied to full conjunctive queries after a simple pre-processing step. The run time matches GLVV's upperbound. We call a full conjunctive query *reduced* if no variable is repeated in the same subgoal. We can assume without loss of generality that a full conjunctive query is reduced, since we can create an equivalent reduced query within the time bound. In time $O(|R_j|)$ for each $j = 1, \ldots, m$, we create a new relation $R'_j$ with arity equal to the number of distinct variables. In one scan over $R_j$ we can produce $R'_j$ by keeping only those tuples that satisfy constants (selections) in the query and any repeated variables. We then construct $q'$ a query over the $R_j$ in the obvious way. Clearly, $q(I) = q'(I)$, and we can construct both in a single scan over the input.

The runtime of our algorithm on full conjunctive queries is also worst-case optimal in terms of data complexity, because GLVV proved the following lower bound. There are arbitrary large integers $N_1, \ldots, N_m$, and database instance $I$ with $|R_j| \leq \sum_{i:R_i=R_j} N_i$ for which the output size satisfies $|q(I)| \geq \prod_{e \in E} N_e^{x^*_e}$. Here, $(x^*_e)$ is the optimal fractional cover for $H$, minimizing the linear objective $\sum_{e \in E} x_e \log N_e$.

*Simple Functional Dependencies.* Given a join query $(V, E)$, a (simple) functional dependency (FD) is a triple $(e, u, v)$ where $u, v \in V$ and $e \in E$ and is written as $e.u \to e.v$. It is a constraint in that the FD $(e, u, v)$ implies that for any pair of tuples $\mathbf{t}, \mathbf{t}' \in R_e$, if $t_u = t'_u$ then $t_v = t'_v$. Fix a set of functional dependencies $\Gamma$, construct a directed (multi-)graph $G(\Gamma)$ where the nodes are the attributes $V$ and there is an edge $(u, v)$ for each functional dependency. The set of all nodes reachable from a node $u$ is a set $U$ of nodes; this relationship is denoted $u \to^* U$.

Given a set of functional dependencies, we propose an algorithm to process a join query. The first step is to compute for each relation $R_e$ for $e \in E$, a new relation $R'_{e'}$ whose attributes are the

union of the closure of each element of $v \in E$, i.e., $e' = \{u \mid v \to u \text{ for } v \in e\}$. Using the closure this can be computed in time $|E||V|$. Then, we compute the contents of $R'_e$. Walking the graph induced by the FDs in a breadth first manner, we can expand $R_e$ to contain all the attributes $R_{e'}$ in time linear in the input size. Finally, we solve the LP from previous section and use our algorithm. It is clear that this algorithm is a strict improvement over our previous algorithm that is FD-unaware. It is an open question to understand its data optimality. We are, however, able to give an example that suggests this algorithm can be substantially better than algorithms that are not FD aware.

Consider the following family of instances on $k+2$ attributes $A, B_1, \ldots, B_k, C$ parameterized by $N$:

$$q = \left( \bowtie_{i \in [k]} R_i(A, B_i) \right) \bowtie \left( \bowtie_{i \in [k]} S_i(B_i, C) \right).$$

Now, we construct a family of instances such that $|R_i| = |S_i| = N$ for $i = 1, \ldots, k$. Suppose there are functional dependencies $A \to B_i$.

Our algorithm will first produce a relation $R'(A, B_1, \ldots, B_k)$, which can then be joined in time $N$ with each relation $S_i$ for $i = 1, \ldots, k$. When we solve the LP, we get a bound of of $|q(I)| \le N^2$—and our algorithm runs within this time.

Now consider the original instance without functional dependencies. Then, the AGM bound is $|q(I)| \le N^k$. More interestingly, one can construct a simple instance where half of the join has a huge size, that is $|\bowtie_{i \in [k]} S_i(B_i, C)| = N^k$. Thus, if we choose the wrong join ordering our algorithms runtime will blow up.

### 8.4 Enumerating Copies of a Fixed Subgraph

Let $H$ be the subgraph on $k$ vertices and $s$ edges that we are interested in. Let $G$ be the host graph on $n$ vertices and $m$ edges; i.e., we want to find all induced occurrences of $H$ in $G$. For this section, we will think of $k$ and $s$ to be $\ll n$: in particular, we will sometimes ignore terms that do not depend on $n$. We will also assume that $H$ and $G$ are undirected and connected (and thus $m \ge n - 1$ and $s \ge k - 1$).

*8.4.1 The Algorithm.* The algorithm is actually straightforward: we represent the sub-graph enumeration problem as a join problem and then use our join algorithm to solve the join query. We begin by stating the join query. Let $R$ denote the set of edges of $E(G)$ (when thought of as directed edges (thus, if $(u, v)$ is in $R$ then so is $(v, u)$). Further, let $T$ denote the set of vertices in $V(G)$. Now consider the join query (where $R(i, j) = R$ and $T(i) = T$):

$$q(H) = \left( \bowtie_{(i,j) \in E(H)} R(i, j) \right) \bowtie \left( \bowtie_{i \in V(H)} T(i) \right). \tag{18}$$

As mentioned above, since we already have a worst-case optimal algorithm to solve join queries, the only interesting part is to show that our general purpose algorithm to solve arbitrary join algorithms is optimal for this special case of subgraph enumeration. In particular, we need to show that the AGM bound is tight when $R$ and $T$ come from graphs as discussed above. In particular, AGM's argument to prove that their upper bound is tight when applied to the query above need not necessarily give relations $R$ and $T$ that are *consistent* with some fixed graph $G$. In Section 8.4.2, we show that we can indeed come up with relations $R$ and $T$ that come from the same graph $G$.

We would like to make a few more comments:

(1) First, the output of the join query actually returns all homomorphisms of subgraph $H$ in $G$ instead of all the induced copies of $H$ in $G$. However, as we will show in Section 8.4.2, for every fixed $H$, there is an infinite family of graph $G$ such that the number of induced

copies of $H$ in $G$ matches the AGM bound. Thus, we can always have a post-processing step that can prune out subgraphs returned by $q(H)$ that are not induced copies of $H$ in $G$.

(2) At first glance, it might seem a bit odd that we include the $T$ relations in the join query. However, as we will see in Section 8.4.2, this extra relation (along with its bound of $n$ on its size) helps us prove more refined lower bounds.

(3) We note that a similar bound on the number of copies of $H$ in $G$ was obtained by [6]. We are re-stating the earlier bounds in terms of AGM, since that is more closely related to our algorithm.

(4) Finally, we remark that our methods can easily be extended to the case when $H$ and $G$ are hypergraphs. For the hypergraph case, a bound similar to one one given by AGM for that case was derived by Reference [27]. In this work, we only concentrate on the graph case, since our main objective is to show the power of our algorithm in solving problems related to *specific* join queries.

*8.4.2 A Tight Lower Bound.* We now consider the $q(H)$ in Equation (18). In what follows, we will think of the parameter $1 \le b \le 2$ such that $m = n^b$ (we will also overload it and think of $m = \Theta(n^b)$).

For what follows let $\mathbf{x} = (x_e)_{e \in E(H)}$ be the optimal solution to the following (primal) LP

$$\min \quad b \cdot \sum_{e \in E(H)} x_e + \sum_{v \in V(H)} x_v,$$

subject to

$$\sum_{e \ni v} x_e + x_v \ge 1 \text{ for all } v \in V(H),$$
$$x_e \ge 0 \text{ for all } e \in E(H),$$
$$x_u \ge 0 \text{ for all } u \in V(H).$$

Also in what follows, unless stated otherwise, let $\mathbf{y} = (y_v)_{v \in V(H)}$ be the optimal solution to the following dual LP:

$$\max \sum_{v \in V(H)} y_v,$$

subject to

$$y_u + y_v \le b \text{ for all } (u, v) \in E(H), \tag{19}$$

$$y_u \le 1 \text{ for all } u \in V(H), \tag{20}$$

$$y_v \ge 0 \text{ for all } v \in V(H). \tag{21}$$

Lemma 8.7. *Let $H$ be a fixed subgraph with $|V(H)| = k$ and $|E(H)| = s$. Let $\mathbf{x}$ and $\mathbf{y}$ be optimal primal and dual solutions to the LPs above. Then for every large enough $N$ (compared to $k$ and $s$), there is a graph $G$ with*

$$N \le n = |V(G)| \le (k+1)N, \tag{22}$$

$$N^b \le m = |E(G)| \le (s+3) \cdot N^b, \tag{23}$$

*such that $G$ has at least*

$$N^{b \cdot \sum_{e \in E(H)} x_e + \sum_{u \in V(H)} x_v} \tag{24}$$

*distinct induced copies of $H$ in it.*

Proof. Consider $k$ disjoint vertex sets $\{V_u\}_{u \in V(H)}$ such that $|V_u| = \lceil N^{y_u} \rceil$. The set $\cup_{u \in V(H)} V_u$ will be in $V(G)$. Thus, the number of vertices we have created so far is

$$n = \sum_{u \in V(H)} \lceil N^{y_u} \rceil \leq k \cdot N,$$

where the inequality follows from the constraint Equation (20). If $n < N$, then we add $N - n$ "dummy" vertices. This proves (22)—we might end up adding an extra $N$ vertices, which will lead to the $(k+1)N$ as the upper bound.

Now for each edge $(u, w) \in E(H)$, add in all possible edges between $V_u$ and $V_w$. Note that we have added $\lceil N^{y_u} \rceil \cdot \lceil N^{y_w} \rceil$ edges here. Thus, we have

$$m \leq \sum_{(u,w) \in E(H)} (N^{y_u} + 1) \cdot (N^{y_w} + 1) \leq s \cdot N^b + 2N + 1 \leq (s+2) \cdot N^b,$$

where the inequality follows from the constraint Equations (19) and (20). If $m < N^b$, then we add extra dummy nodes, so that we have a total of $N$ dummy nodes (note that this does not affect Equation (22)) and add $N^b - m$ dummy edges within the dummy codes (note that since $b \leq 2$, this is possible). This proves Equation (23).

Finally, note that if we pick exactly one vertex from each of $V_u$ for $u \in V(H)$, then the induced sub-graph in $G$ is $H$. Since we get a distinct subgraph for each such choice, $G$ has at least

$$\prod_{u \in V(H)} N^{y_u} = N^{\sum_{u \in V(H)} y_u}$$

copies of $H$ in it. (The dummy edges might result in more copies of $H$, but since we only care about a lower bound, that is not an issue.) Strong LP duality then proves Equation (24). □

Lemma 8.7 implies the following:

Corollary 8.8. *Given any fixed $H$ and a real $1 \leq b \leq 2$, then there exists an infinite family of graphs $G$ with $m = \Theta_{k,s}(n^b)$ such that the number of copies of $H$ in $G$ and the AGM upper bounded are off by a factor of at most $(ks)^{O(k)}$.*

Proof. The claim on the relation between $m$ and $n$ follows from Equations (22) and (23). Note that when we apply the AGM bound to Equation (18), we get an upper bound of

$$\begin{aligned}\left|q^G(H)\right| \leq m^{\sum_{e \in E(H)} x_e} \cdot n^{\sum_{u \in V(H)} x_u} &\leq (s+3)^k \cdot N^{b \cdot \sum_{e \in E(H)} x_e} \cdot (k+1)^k \cdot N^{\sum_{u \in V(H)} x_u} \\ &\leq ((s+3)(k+1))^k \cdot N^{b \cdot \sum_{e \in E(H)} x_e + \sum_{u \in V(H)} x_u},\end{aligned}$$

which with Equation (24) gives the gap between the upper and lower bounds. The second inequality follows from Equations (22) and (23) and the following easily verifiable facts[7]:

$$\sum_{e \in E(H)} x_e \leq k,$$

and

$$\sum_{u \in V(H)} x_u \leq k.$$ □

[7]Recall that $\mathbf{x}$ is an optimal solution to the primal LP. Further, the following is a feasible solution: $x'_v = 1$ for every $v \in V(H)$ and $x'_e = 0$ for every $e \in E(H)$, which gives an objective value of $k$ and hence, the optimal objective value (which will be obtained by $\mathbf{x}$) has to be smaller.

## 9 CONCLUSION AND FUTURE WORK

In this work, we established optimal algorithms for the worst-case behavior of join algorithms. We also demonstrated that the join algorithms employed in RDBMS do not achieve these optimal bounds—and we demonstrated families of instances where they were asymptotically worse by factors close to the size of the largest relation. It is interesting to ask similar questions for average case complexity. Our work offers a fundamentally different way to approach join optimization rather than the traditional binary-join/dynamic-programming-based approach. It has been shown that the ideas developed in this article and subsequent works can be competitive in real DBMS settings [64].

Another interesting direction is to extend these results to a larger classes of queries and to database schemata that have constraints. We presented some preliminary results on full conjunctive queries and simple functional dependencies (FDs). Not surprisingly, using dependency information one can obtain tighter bounds compared to the (FD-unaware) fractional cover technique. It is interesting to investigate whether our algorithm for computing relaxed joins can be useful in related context such as those considered in Koudas et al. [51].

There are potentially interesting connections between our work and several inter-related topics, which are all great subjects to further explore. We algorithmically proved AGM's bound, which is equivalent to BT inequality, which in turn is essentially equivalent to Shearer's entropy inequality. There are known combinatorial interpretations of entropy inequalities of which Shearer's is a special case; for example, Alon et al. [8] derived some such connections using a notion of "sections" similar to what we used in this article. An analogous partitioning procedure was used in Reference [58] to compute joins by relating the number of solutions to submodular functions. Our lead example (the LW inequality with $n = 3$) is equivalent to the problem of enumerating all triangles in a tri-partite graph. It was known that this can be done in time $O(N^{3/2})$ [46]. Interestingly, if we order the vertices by their degree then one can compute all triangles in $O(N^{3/2})$ time by doing pair-wise joins [20]. Thus, one can get around the lower bound arguments of Section 7.1 if one is allowed to re-order the elements of the domain. This suggests an interesting open question: for what class of join queries does re-ordering of domain elements allow traditional join-tree algorithm to be worst-case optimal?

## REFERENCES


[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley.

[2] Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2016. Joins via geometric resolutions: Worst case and beyond. *ACM Trans. Database Syst.* 41, 4 (2016), 22:1–22:45. DOI: https://doi.org/10.1145/2967101

[3] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions asked frequently. In *Proceedings of PODS'16*. 13–28. DOI: https://doi.org/10.1145/2902251.2902280

[4] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2016. Computing join queries with functional dependencies. In *Proceedings of PODS'16*, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 327–342. DOI: https://doi.org/10.1145/2902251.2902289

[5] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of PODS'17*, Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts (Eds.). ACM, 429–444. DOI: https://doi.org/10.1145/3034786.3056105

[6] Noga Alon. 1981. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel J. Math.* 38, 1-2 (1981), 116–130. DOI: https://doi.org/10.1007/BF02761855

[7] Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. 1999. Tracking join and self-join sizes in limited storage. In *Proceedings of PODS*. 10–20.

[8] Noga Alon, Ilan Newman, Alexander Shen, Gábor Tardos, and Nikolai K. Vereshchagin. 2007. Partitioning multi-dimensional sets in a small number of "uniform" parts. *Eur. J. Comb.* 28, 1 (2007), 134–144.

[9] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size bounds and query plans for relational joins. In *Proceedings of FOCS*. IEEE Computer Society, 739–748.

[10] Ron Avnur and Joseph M. Hellerstein. 2000. Eddies: Continuously adaptive query processing. In *Proceedings of the SIGMOD Conference*. 261–272.
[11] Shivnath Babu, Pedro Bizarro, and David J. DeWitt. 2005. Proactive re-optimization. In *Proceedings of the SIGMOD Conference*. 107–118.
[12] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. 1983. On the desirability of acyclic database schemes. *J. ACM (JACM)* 30, 3 (1983), 479–513.
[13] Philip A. Bernstein and Nathan Goodman. 1981. Power of natural semijoins. *SIAM J. Comput.* 10, 4 (1981), 751–771.
[14] Béla Bollobás and Andrew Thomason. 1995. Projections of bodies and hereditary properties of hypergraphs. *Bull. London Math. Soc.* 27, 5 (1995), 417–424. DOI: https://doi.org/10.1112/blms/27.5.417
[15] Ashok K. Chandra and David Harel. 1979. Computable queries for relational data bases (Preliminary Report). In *Proceedings of STOC*, Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho (Eds.). ACM, 309–318.
[16] Ashok K. Chandra and David Harel. 1980. Structure and complexity of relational queries. In *Proceedings of FOCS*. IEEE Computer Society, 333–347.
[17] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of STOC*, John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison (Eds.). ACM, 77–90.
[18] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In *Proceedings of PODS'98*. ACM, New York, NY, 34–43. DOI: https://doi.org/10.1145/275487.275492
[19] Chandra Chekuri and Anand Rajaraman. 2000. Conjunctive query containment revisited. *Theor. Comput. Sci.* 239, 2 (2000), 211–229.
[20] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223. DOI: https://doi.org/10.1137/0214017
[21] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. 1986. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A* 43, 1 (1986), 23–37. DOI: https://doi.org/10.1016/0097-3165(86)90019-1
[22] Antonios Deligiannakis, Minos N. Garofalakis, and Nick Roussopoulos. 2007. Extended wavelets for multiple measures. *ACM Trans. Database Syst.* 32, 2 (2007), 10.
[23] David Eppstein. 1993. Connectivity, graph minors, and subgraph multiplicity. *J. Graph Theory* 17, 3 (1993), 409–416. DOI: https://doi.org/10.1002/jgt.3190170314
[24] Ronald Fagin. 1983. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM* 30, 3 (1983), 514–550.
[25] Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. 1982. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst.* 7, 3 (1982), 343–360.
[26] Jörg Flum, Markus Frick, and Martin Grohe. 2002. Query evaluation via tree-decompositions. *J. ACM* 49, 6 (2002), 716–752.
[27] Ehud Friedgut and Jeff Kahn. 1998. On the number of copies of one hypergraph in another. *Israel J. Math.* 105 (1998), 251–256. DOI: https://doi.org/10.1007/BF02780332
[28] Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. 2013. $\ell_2/\ell_2$-foreach sparse recovery with low risk. In *Proceedings of ICALP'13*. 461–472. DOI: https://doi.org/10.1007/978-3-642-39206-1_39
[29] Nathan Goodman and Oded Shmueli. 1982. Tree queries: A simple class of relational queries. *ACM Trans. Database Syst.* 7, 4 (1982), 653–677.
[30] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. 2012. Size and treewidth bounds for conjunctive queries. *J. ACM* 59, 3 (2012), 16. DOI: https://doi.org/10.1145/2220357.2220363
[31] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2001. The complexity of acyclic conjunctive queries. *J. ACM* 48, 3 (2001), 431–498.
[32] Georg Gottlob, Nicola Leone, and Francesco Scarcello. 2002. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 579–627.
[33] Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. 2009. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM* 56, 6 (2009).
[34] Goetz Graefe. 1993. Query evaluation techniques for large databases. *Comput. Surveys* 25, 2 (June 1993), 73–170.
[35] M. Grohe. 2013. Bounds and algorithms for joins via fractional edge covers. In *Search of Elegance in the Theory and Practice of Computation, Essays Dedicated to Peter Buneman*, V. Tannen, L. Wong, L. Libkin, W. Fan, W.-C. Tan, and M. Fourman (Eds.). Lecture Notes in Computer Science 8000, 321–338.
[36] Martin Grohe and Dániel Marx. 2006. Constraint solving via fractional edge covers. In *Proceedings of SODA*. ACM Press, 289–298.
[37] Martin Grohe, Thomas Schwentick, and Luc Segoufin. 2001. When is the evaluation of conjunctive queries tractable? In *Proceedings of STOC*, Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis (Eds.). ACM, 657–666.

[38] Venkatesan Guruswami. 2004. *List Decoding of Error-Correcting Codes (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition)*. Lecture Notes in Computer Science, Vol. 3282. Springer.
[39] Katalin Gyarmati, Máté Matolcsi, and Imre Z. Ruzsa. 2010. A superadditivity and submultiplicativity property for cardinalities of sumsets. *Combinatorica* 30, 2 (2010), 163–174. DOI: https://doi.org/10.1007/s00493-010-2413-6
[40] Te Sun Han. 1978. Nonnegative entropy measures of multivariate symmetric correlations. *Info. Control* 36, 2 (1978), 133–156.
[41] G. H. Hardy, J. E. Littlewood, and G. Pólya. 1988. *Inequalities*. Cambridge University Press, Cambridge. xii+324 pages. Reprint of the 1952 edition.
[42] Yannis E. Ioannidis. 1996. Query optimization. *ACM Comput. Surv.* 28 (March 1996), 121–123. Issue 1. DOI: https://doi.org/10.1145/234313.234367
[43] Yannis E. Ioannidis. 2003. The history of histograms (abridged). In *Proceedings of VLDB*. 19–30.
[44] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the propagation of errors in the size of join results. In *Proceedings of the SIGMOD Conference*. 268–277.
[45] Dror Irony, Sivan Toledo, and Alexandre Tiskin. 2004. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.* 64, 9 (2004), 1017–1026.
[46] Alon Itai and Michael Rodeh. 1978. Finding a minimum circuit in a graph. *SIAM J. Comput.* 7, 4 (1978), 413–423. DOI: https://doi.org/10.1137/0207033
[47] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. 1998. Optimal histograms with quality guarantees. In *Proceedings of VLDB*.
[48] Manas R. Joglekar, Rohan Puttagunta, and Christopher Ré. 2016. AJAR: Aggregations and joins over annotated relations. In *Proceedings of PODS'16*. 91–106. DOI: https://doi.org/10.1145/2902251.2902293
[49] Phokion G. Kolaitis and Moshe Y. Vardi. 2000. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.* 61, 2 (2000), 302–332.
[50] A. C. König and G. Weikum. 1999. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In *Proceedings of VLDB*.
[51] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. 2006. Relaxing join and selection queries. In *Proceedings of VLDB'06*.
[52] April Rasala Lehman and Eric Lehman. 2005. Network coding: Does the model need tuning? In *Proceedings of SODA'05*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 499–504.
[53] L. H. Loomis and H. Whitney. 1949. An inequality related to the isoperimetric inequality. *Bull. Amer. Math. Soc* 55 (1949), 961–962.
[54] Russell Lyons. 2011. Probability on Trees and Networks. Retrieved from http://php.indiana.edu/rdlyons/prbtree/prbtree.html.
[55] David Maier. 1983. *The Theory of Relational Databases*. Computer Science Press.
[56] Francesco M. Malvestuto. 1986. Modelling large bases of categorial data with acyclic schemes. In *Proceedings of ICDT (Lecture Notes in Computer Science)*, Giorgio Ausiello and Paolo Atzeni (Eds.), Vol. 243. Springer, 323–340.
[57] Volker Markl, Nimrod Megiddo, Marcel Kutsch, Tam Minh Tran, Peter J. Haas, and Utkarsh Srivastava. 2005. Consistently estimating the selectivity of conjuncts of predicates. In *Proceedings of VLDB*. 373–384.
[58] Dániel Marx. 2010. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of STOC*. 735–744.
[59] Moritz Müller. 2008. Valiant-Vazirani lemmata for various logics. *Electron. Colloq. Comput. Complex. (ECCC)* 15, 063 (2008).
[60] M. E. J. Newman. 2003. The structure and function of complex networks. *SIAM Rev.* 45, 2 (2003), 167–256 (electronic). DOI: https://doi.org/10.1137/S003614450342480
[61] Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. 2014. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of PODS'14*. 234–245. DOI: https://doi.org/10.1145/2594538.2594547
[62] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2012. Worst-case optimal join algorithms. In *Proceedings of PODS*, Maurizio Lenzerini and Michael Benedikt (Eds.). ACM, 37–48.
[63] Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2013. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.* 42, 4 (2013), 5–16. DOI: https://doi.org/10.1145/2590989.2590991
[64] Dung T. Nguyen, Molham Aref, Martin Bravenboer, George Kollias, Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2015. Join processing for graph patterns: An old dog with new tricks. In *Proceedings of GRADES'15*. DOI: https://doi.org/10.1145/2764947.2764948
[65] Anna Pagh and Rasmus Pagh. 2006. Scalable computation of acyclic joins. In *Proceedings of PODS*. 225–232.
[66] Rasmus Pagh and Charalampos E. Tsourakakis. 2012. Colorful triangle counting and a MapReduce implementation. *Info. Process. Lett.* 112, 7 (March 2012), 277–281. DOI: https://doi.org/10.1016/j.ipl.2011.12.007

[67] Christos H. Papadimitriou and Mihalis Yannakakis. 1997. On the complexity of database queries. In *Proceedings of PODS*, Alberto O. Mendelzon and Z. Meral Özsoyoglu (Eds.). ACM Press, 12–19.
[68] V. Poosala, Y. E. Ioannidis, P. J. Haas, and Eugene J. Shekita. 1996. Improved histograms for selectivity estimation of range predicates. In *Proceedings of SIGMOD*. 294–305.
[69] Domenico Saccà. 1985. Closures of database hypergraphs. *J. ACM* 32, 4 (1985), 774–803.
[70] Yehoshua Sagiv and Oded Shmueli. 1993. Solving queries by tree projections. *ACM Trans. Database Syst.* 18, 3 (1993), 487–511.
[71] Alexander Schrijver. 2003. *Combinatorial Optimization. Polyhedra and Efficiency. Vol. A*. Algorithms and Combinatorics, Vol. 24. Springer-Verlag, Berlin.
[72] Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. 2006. ISOMER: Consistent histogram construction using query feedback. In *Proceedings of ICDE*. 39.
[73] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of WWW*. 607–614.
[74] Charalampos E. Tsourakakis. 2008. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Proceedings of* ICDM'08. IEEE Computer Society, Washington, DC, USA, 608–617. DOI: https://doi.org/10.1109/ICDM.2008.72
[75] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB* 4, 11 (2011), 852–863.
[76] Jeffrey D. Ullman. 1989. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press.
[77] Leslie G. Valiant and Vijay V. Vazirani. 1986. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.* 47, 3 (1986), 85–93.
[78] Moshe Y. Vardi. 1982. The complexity of relational query languages (Extended Abstract). In *Proceedings of STOC*, Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber (Eds.). ACM, 137–146.
[79] Moshe Y. Vardi. 1995. On the complexity of bounded-variable queries. In *Proceedings of PODS*, Mihalis Yannakakis (Ed.). ACM Press, 266–276.
[80] Todd L. Veldhuizen. 2014. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of ICDT*. 96–106.
[81] Dan E. Willard. 1996. Applications of range query theory to relational data base join and selection operations. *J. Comput. Syst. Sci.* 52, 1 (1996), 157–169.
[82] Dan E. Willard. 2002. An algorithm for handling many relational calculus queries efficiently. *J. Comput. Syst. Sci.* 65, 2 (2002), 295–331.
[83] Yu Xu, Pekka Kostamaa, Xin Zhou, and Liang Chen. 2008. Handling data skew in parallel joins in shared-nothing systems. In *Proceedings of the SIGMOD Conference*. 1043–1052.
[84] Mihalis Yannakakis. 1981. Algorithms for acyclic database schemes. In *Proceedings of VLDB*. IEEE Computer Society, 82–94.
[85] Mihalis Yannakakis. 1995. Perspectives on database theory. In *Proceedings of FOCS*. IEEE Computer Society, 224–246.
[86] C. Yu and M. Ozsoyoglu. 1979. An algorithm for tree query membership for a distributed query. In *Proceedings of IEEE COMPSAC*. 306–312.
[87] C. T. Yu and M. Z. Ozsoyoglu. 1984. On determining tree-query membership of a distributed query. *Informatica* 22, 3 (1984), 261–282.

Received January 2013; revised November 2017; accepted January 2018