Worst-case Optimal Join Algorithms

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

Ely Porat
Bar-Ilan University
porately@cs.biu.ac.il

Christopher Ré
University of Wisconsin–Madison
chrisre@cs.wisc.edu

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

ABSTRACT

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 novel 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 full conjunctive 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 running time achieve these optimal bounds. An answer to this question may be interesting to database practice, as we show in this paper that any project-join plan is polynomially slower than the optimal bound for some queries. We construct an algorithm whose running time 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. In addition, we show that this bound is equivalent to a geometric inequality by Bollobás and Thomason, one of whose special cases is the famous Loomis-Whitney inequality. Hence, our results algorithmically prove these inequalities as well. Finally, we discuss how our algorithm can be used to compute a relaxed notion of joins.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms

Algorithms, Theory

Keywords

Join Algorithms, fractional cover bound, Loomis-Whitney inequality, Bollobás-Thomason inequality

PODS'12, May 21–23, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Recently, Grohe and Marx [17] and Atserias, Grohe, and Marx [5] (AGM's results henceforth) derived tight bounds on the number of output tuples of a *full conjunctive query*[1] in terms of the sizes of the relations mentioned in the query's body. As query output size estimation is fundamentally important for efficient query processing, these results have generated a great deal of excitement.

To understand the spirit of AGM's results, consider the following example where we have a schema with three attributes, A, B, and C, and three relations, $R(A,B)$, $S(B,C)$ and $T(A,C)$, defined over those attributes. Consider the following natural join query:

$$q = R \bowtie S \bowtie T \tag{1}$$

Let $q(I)$ denote the set of tuples that is output from applying q to a database instance I, i.e. $q(I)$ 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 trivial bound is $|q(I)| \leq N^3$. One can obtain a better bound by noticing that the output of any pair-wise join (say $R \bowtie S$) will be a superset of $q(I)$, since the union of the attributes in R and S together contain (or "cover") all attributes. This leads to the bound $|q(I)| \leq N^2$. AGM showed that one can get a better upper bound of $|q(I)| \leq N^{3/2}$ by generalizing the notion of cover to a so-called "fractional cover" (see Section 2). Moreover, this estimate is tight in the sense that for infinitely many values of N, one can find a database instance I for which $|R| = |S| = |T| = N$ and $|q(I)| = N^{3/2}$. 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.

More generally, 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 2 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*. 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 $|q(I)| = U$.

AGM's results leave open whether one can compute the actual set $q(I)$ in time $O(U)$. In fact, AGM observed this issue and presented an algorithm that computes $q(I)$ with a running time of $O(|q|^2 \cdot$

[1]A full conjunctive query is a conjunctive query where every variable in the body appears in the head.

$U \cdot N$) where N is the cardinality of the largest input relation and $|q|$ denotes the size of the query q. AGM established that their join-project plan can in some cases be super-polynomially better than any join-only plan. However, AGM's join algorithm is not optimal. Even on query (1), we can construct a family of database instances $I_1, I_2, \ldots, I_N, \ldots$, such that in the Nth instance I_N we have $|R| = |S| = |T| = N$ and any join-project plan (which includes AGM's algorithm) takes $\Omega(N^2)$-time even though from AGM's bound we know that $|q(I)| \leq U = N^{3/2}$, which is the best worst-case run-time that one can hope for.

The $\sqrt{N}$-gap on a small example motivates our central question. In what follows, *natural join queries* are defined as the join of a set of relations $R_1, \ldots, R_m$.

> **Optimal Worst-case Join Evaluation Problem** (Optimal Join Problem). *Given a fixed database schema $\bar{R} = \left\{R_i(\bar{A}_i)\right\}_{i=1}^{m}$ 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 $|R_i^I| = N_i$ for $i = 1, \ldots, m$. Define $U = \sup_{I \in I(\bar{N})} |q(I)|$. Then, the optimal worst-case join evaluation problem is to evaluate q in time $O(U + \sum_{i=1}^{m} N_i)$.*

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.[2] (Note that we are mainly concerned with data complexity and thus the $O(U)$ bound above ignores the dependence on $|q|$. Our results have a small $O(|q|)$ factor.)

Implicitly, this problem has been studied for over three decades: a modern RDBMS uses decades of highly tuned algorithms to efficiently produce query results. Nevertheless, as we described above, such systems are asymptotically suboptimal – even for query (1). Our main result is an algorithm that achieves asymptotically optimal worst-case running times for all join queries.

We begin by describing connections 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 Bollobás and Thomason [8, Theorem 2]. This equivalence is shown in Section 2.2.

Our ideas for an algorithm solving the optimal join problem begin by examining a special case of the Bollobás-Thomason (BT) inequality: the classic Loomis-Whitney (LW) inequality [26]. 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. The bound $|q(I)| \leq \sqrt{|R||S||T|}$ for query (1) is *exactly* the LW inequality with $n = 3$ applied to the discrete measure. Our algorithmic development begins with a slight generalization of query (1). We describe an algorithm for join queries which have 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 running time that is asymptotically data-optimal for this class of queries (e.g., $O(N^{3/2})$ in our motivating example), our algorithm is worst-case data-complexity optimal for LW instances.

Our algorithm for LW instances exhibits a key twist compared to a conventional join algorithm. The twist is that our 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 not typically done in standard RDBMS join processing.

Building on the algorithm for LW instances, we next describe our main result: an algorithm to solve the optimal join problem for all join queries. In particular, we design an algorithm for evaluating join queries which not only *proves* AGM's fractional cover inequality *without* using the information-theoretic Shearer's inequality, but also has a running time that is linear in the bound (modulo pre-processing time). As AGM's inequality is equivalent to BT inequality and thus implies LW inequality, 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 as in the LW-instance case. Our algorithm computes these orderings, and then at each stage it performs a heavy/light tuple check that is similar to the strategy used for the LW instances earlier.

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 run-time than our proposed approach?* AGM's analysis of their join-project algorithm leads to a worst case run-time 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 $O(k^2 N)$-time.

We consider several extensions and improvements of our main result. 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.

Outline. The remainder of the paper is organized as follows: in the rest of this section, we describe related work. In Section 2 we describe our notation, formulate the main problem, and prove the connection between AGM's inequality and BT inequality. Our main results are in Section 3. We first present a data-optimal join algorithm for LW instances, and then present the optimal algorithm for arbitrary join queries. We also discuss the limits of performance of prior approaches and our approach in more detail. In Section 4, we describe several extensions. We conclude in Section 5. Due to space constraints some proofs are deferred to the full version [30].

Related Work

Grohe and Marx [17] made the first (implicit) connection between fractional edge cover and the output size of a conjunctive query. (Their results were stated for constraint satisfaction problems.) At-

[2] In an 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 $O(U)$, removing the term $\sum_i N_i$ by precomputing some indices.

serias, Grohe, and Marx [5] extended Grohe and Marx's results in the database setting.

The first relevant result of AGM is the following inequality. Consider a join query over relations R_e, $e \in E$, where E is a collection of subsets of an attribute "universe" V, and relation R_e is on attribute set e. Then, 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 cover of the hypergraph $H = (V, E)$.

They also showed 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 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 $\mathbf{x}$ which 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 presented an inapproximability result which 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 plan with running time $O\left(|q|^2 N_{\max}^{1+\sum x_e}\right)$, where $N_{\max}$ is the maximum size of input relations and $|q| = |V| \cdot |E|$ is the query size.

The AGM's inequality contains as a special case the discrete versions of two well-known inequalities in geometry: the *Loomis-Whitney* (LW) inequality [26] and its generalization the *Bollobás-Thomason* (BT) inequality [8]. There are two typical proofs of the discrete LW and BT inequalities. The first proof is by induction using Hölder's inequality [8]. The second proof (see Lyons and Peres [27]) essentially uses "equivalent" entropy inequalities by Han [19] and its generalization by Shearer [9], which was also the route Grohe and Marx [17] 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 [22]. The inequality was used to prove submultiplicativity inequalities regarding sums of sets of integers [18]. In [25], a special case of BT inequality was used to prove a network-coding bound. Recently, some of the authors of this paper have used our *algorithmic* version of the LW inequality to design a new sub-linear time decodable compressed sensing matrices [12] and efficient pattern matching algorithms [31].

Inspired by AGM's results, Gottlob, Lee, and Valiant [13] generalized AGM's results to conjunctive queries with functional dependencies. Their key idea was a new notion, *the "coloring number"*, which is derived from the dual linear program of the fractional cover linear program.

Join processing is one of the most studied problems in database research. On the theoretical side, that acyclic queries can be computed in polynomial time is one of the classic results in database theory [1, Ch. 6.4]. When the join graph is acyclic, there are several known results which achieve (near) optimal run time with respect to the output size [32, 38]. One direction to extend the reach of these positive results is using *hypertree decompositions* that capture the idea that many queries are nearly acyclic [14, 15]; This work has culminated in efficient algorithms for broad classes of conjunctive queries – a more general class of queries than we consider here. The algorithms in this work are complementary: our algorithms are most interesting when the queries are cyclic. In practice, a staggering number of variants have been considered, we list a few: Block-Nested loop join, Hash-Join, Grace, Sort-merge (see Grafe [16] for a survey). Conceptually, it is interesting that none of the classical algorithms consider performing a per-tuple cardinality estimation as our algorithm does. It is interesting future work to implement our algorithm to better understand its performance.

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 [2, 10, 20, 23, 24, 33]. Often, deriving estimates for arbitrary query expressions involves making statistical assumptions, such as the independence or containment assumptions, which may result in large estimation errors [21]. Follow-up work has considered sophisticated probability models, entropy-based models [28, 35] and graphical models [36]. In contrast, in this work we examine the *worst case behavior* of algorithms in terms of its cardinality estimates.

On a technical level, the work *adaptive query processing* is related, e.g., Eddies [6] and RIO [7]. The main idea is that to compensate for erroneous 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 running time 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 [39]. This idea was not used to theoretically improve the running time of join algorithms. We are excited that a key mechanism used by our algorithm is implemented in a modern commercial system.

2. PRELIMINARIES

We first describe our notation and formal problem statement. Then, we describe the connection between AGM's result and the BT inequality.

2.1 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 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 an instance 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 database instance I, the output of the query q on I is denoted $q(I)$ 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 $q = \bowtie_{i\in q} R_i$ 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 $i \in q$. Let $N_e = |R_e|$ be the number of tuples in R_e. *From now on we use the hypergraph and the original notation for the query interchangeably.*

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

$$\begin{aligned} \sum_{e:v\in e} x_e &\geq 1, \text{ for any } v \in V \\ x_e &\geq 0, \text{for any } e \in E \end{aligned}$$

Note that the solution $x_e = 1$ for $e \in E$ is always a feasible solution for hypergraphs representing join queries (since each vertex appears in some edge, $\sum_{e:v\in e} x_e \geq 1$). A point $\mathbf{x}$ in the polytope is also called a *fractional (edge) cover solution* of the hypergraph H.

Atserias, Grohe, and Marx [5] establish that for *any* point $\mathbf{x} = (x_e)_{e\in E}$ in the fractional edge cover polytope

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

The bound is proved nonconstructively using Shearer's entropy inequality [9]. However, AGM provide an algorithm based on join-project plans that runs in time $O(|q|^2 \cdot N_{\max}^{1+\sum_e x_e})$ where $N_{\max} = \max_{e\in E} N_e$. They observed that for a fixed hypergraph H and given sizes N_e the bound (2) can be minimized by solving the linear program which minimizes the linear objective $\sum_e (\log N_e)\cdot x_e$ over fractional edge cover solutions $\mathbf{x}$. (Since in linear time we can determine if we have an empty relation, and hence an empty output, for the rest of the paper we are always going to assume that $N_e \geq 1$.) We recast our problem using the above language.

Definition 2.1 (OJ Problem – Optimal Join Problem). *With the notation above, design an algorithm to compute $\bowtie_{e\in E} R_e$ with running time*

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

Here $f(|V|,|E|)$ and $g(|V|,|E|)$ are ideally polynomials with (small) constant degrees, which only depend on the query size. The linear term $\sum_{e\in E} N_e$ is to read and index the input (in a specific way). Such an algorithm would be data-optimal in the worst case.[3]

We recast our motivating example from the introduction in our notation. Recall that we are given as input, $R(A,B), S(B,C), T(A,C)$. The resulting hypergraph (V, E) is such that $V = \{A, B, C\}$ and E contains three edges corresponding to each of R, S, and T. More explicitly, we have $E = \{\{A,B\}, \{B,C\}, \{A,C\}\}$. Thus, $|V| = 3$ and $|E| = 3$. If $N_e = N$, one can check that the optimal solution to the LP is $x_e = \frac{1}{2}$ for $e \in E$ which has the objective value $\frac{3}{2}\log N$; in turn, this gives $\sup_{I\in I(\bar{N})} |q(I)| \leq N^{3/2}$ (recall $I(\bar{N}) = \{I : |R_e^I| = N_e \text{ for } e \in E\}$).

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

$$R^{I_N} = S^{I_N} = T^{I_N} = \{(0,j)\}_{j=0}^{(N-1)/2} \cup \{(j,0)\}_{j=0}^{(N-1)/2}.$$

For example,

$$R \bowtie S = \{(i,0,j)\}_{i,j=0}^{(N-1)/2} \cup \{(0,i,0)\}_{i=1,\ldots,(N-1)/2}$$

and $R \bowtie S \bowtie T = \{(0,0,j)\}_{j=0}^{(N-1)/2} \cup \{(0,j,0)\}_{j=1}^{(N-1)/2} \cup \{(j,0,0)\}_{j=1}^{(N-1)/2}$. Thus, any standard join-based algorithm takes time $\Omega(N^2)$. We show later that any project-join plan (which includes 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.

2.2 Connections to Geometric Inequalities

We describe the Bollobás-Thomason (BT) inequality from discrete geometry and prove that the BT inequality is equivalent to the AGM inequality. We then look at a special case of the BT inequality called the Loomis-Whitney (LW) inequality, from which our algorithmic development starts in the next section. The BT inequality can be stated as follows.

Theorem 2.2 (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, $|S|^d \leq \prod_{F\in\mathcal{F}} |S_F|$.*

To prove the equivalence between BT inequality and the AGM bound, we first need a simple observation, whose proof can be found in the full version [30].

Lemma 2.3. *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 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 the same answer:

$$\bowtie_{e\in E} R_e = \bowtie_{e\in E'} R'_e.$$

(c) AGM's bound on the transformed instance is at least as good as that of the original instance:

$$\prod_{e\in E'} |R'_e|^{x'_e} \leq \prod_{e\in E} |R_e|^{x_e}.$$

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

Proposition 2.4. *BT inequality and AGM's fractional cover bound are equivalent.*

[3] As shall be seen later, the worst-case preprocessing time is linear in the RAM model using the *"lazy array"* technique of Lemma A.3 of Flum, Frick, and Grohe [11], at the expense of potentially huge space overhead. To remove this excess space, we can build a set of hash indices in expected linear time using any perfect hashing scheme with worst-case constant time, e.g., *Cuckoo hashing*. Also, one can build a search tree for each relation to ensure a worst-case guarantee but with an extra log factor in the running time.

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 $|S| \le \prod_{F\in\mathcal{F}} |S_F|^{1/d}$.

Conversely, consider an instance of the OJ problem with hypergraph $H = (V, E)$ and a rational fractional cover $\mathbf{x} = (x_e)_{e\in E}$ of H. First, by Lemma 2.3, we can assume that all cover constraints are tight, i.e., $\sum_{e:v\in e} x_e = 1$, for any $v \in V$. Second, when all variables x_e are rational we can write x_e as d_e/d for a positive common denominator d. Consequently,

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

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 2.2. Hence, the size of the join is bounded above by $\prod_{e\in E'} |R'_e|^{1/d} = \prod_{e\in E} |R_e|^{d_e/d} = \prod_{e\in E} |R_e|^{x_e}$. When some of the x_e are not rational, we can replace each irrational x_e by a rational $x'_e > x_e$ with a sufficiently small difference and apply the above analysis. □

Loomis-Whitney. We now consider a special case of the BT inequality, the discrete version of a classic geometric inequality called the *Loomis-Whitney inequality* [26]. The setting is that for $n \ge 2$, $V = [n]$ and $E = \binom{V}{|V|-1}$,[4] where in this case $x_e = 1/(|V| - 1), \forall e \in E$ is a fractional cover solution for (V, E). LW showed the following:

Theorem 2.5 (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, $|S|^{n-1} \le \prod_{i=1}^{n} |S_{[n]\setminus\{i\}}|$.*

The LW inequality is a special case of the BT inequality (and so the AGM inequality), and it is with this special case that we begin our algorithmic development in the next section.

3. MAIN RESULTS

We first describe our algorithm for the LW inequality. We then describe our main algorithmic result, which is an algorithm that proves the AGM bound and whose running time matches the bound. Finally, we observe some limitations of project-join plans, which include as special cases both standard binary join-based algorithms and AGM's join algorithm.

3.1 Algorithm for Loomis-Whitney Instances

We first consider queries whose forms are slightly more general than that in our motivating example (1). This class of queries has the same setup as in LW inequality of Theorem 2.5. 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 $|V| - 1$. When the LW inequality is applied to this setting, it guarantees that $|\bowtie_{e\in E} R_e| \le (\prod_{e\in E} N_e)^{1/(n-1)}$, and the bound is tight in the worst case. The main result of this section is the following:

Theorem 3.1 (Loomis-Whitney instance). *Let $n \ge 2$ be an integer. Consider a Loomis-Whitney instance $H = (V = [n], E)$ of the OJ problem with input relations R_e, where $|R_e| = N_e$ for $e \in E$. Then the join $\bowtie_{e\in E} R_e$ can be computed in time*

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

Before describing our algorithm, we give an example that illustrates the intuition behind our algorithm and solves the motivating example (1) from the introduction.

Example 2. Recall that our input has three relations $R(A, B), S(B, C), T(A, C)$ and an instance I such that $|R^I| = |S^I| = |T^I| = N$. Let $J = R \bowtie S \bowtie T$. Our goal is to construct J in time $O(N^{3/2})$. For exposition, define a parameter $\tau \ge 0$ that we will choose below. We use τ to define two sets that effectively partition the tuples in R^I.

$$D = \{t_B \in \pi_B(R) : |R^I[t_B]| > \tau\} \text{ and } G = \{(t_A, t_B) \in R^I : t_B \notin D\}$$

Intuitively, D contains the heavy join keys in R. Note that $|D| < N/\tau$. Observe that $J \subseteq (D \times T) \cup (G \bowtie S)$ (also note 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 our running time is linear in the above sets, the key question is how big are these two sets?

Observe that $|D \times T| \le (N/\tau)N = N^2/\tau$ while $|G \bowtie S| = \sum_{t_B\in\pi_B(G)} |R[t_B]||S[t_B]| \le \tau N$. Setting $\tau = \sqrt{N}$ makes both terms at most $N^{3/2}$, establishing the running time 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 $\tau = \sqrt{\frac{|R||T|}{|S|}}$, we achieve a running time of $O(\sqrt{|R||S||T|} + |R| + |S| + |T|)$.

To describe the general algorithm underlying Theorem 3.1, 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 $D(x)$ intuitively allows 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

[4] We use $E = \binom{V}{k}$ to denote the set of all undirected hyperedges (subsets of nodes) of size exactly k.

Algorithm 1 Algorithm for Loomis-Whitney Instances

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

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

1: **if** x is a leaf **then**
2: **return** $(\emptyset, R_{\text{LABEL}(x)})$
3: $(C_L, D_L) \leftarrow LW(\text{LC}(x))$ and $(C_R, D_R) \leftarrow LW(\text{RC}(x))$
4: $F \leftarrow \pi_{\text{LABEL}(x)}(D_L) \cap \pi_{\text{LABEL}(x)}(D_R)$
5: $G \leftarrow \{\mathbf{t} \in F : |D_L[\mathbf{t}]| + 1 \le \lceil P/|D_R| \rceil\}$ // $F = G = \emptyset$ *if* $|D_R| = 0$
6: **if** x is the root of $\mathcal{T}$ **then**
7: $C \leftarrow (D_L \bowtie D_R) \cup C_L \cup C_R$
8: $D \leftarrow \emptyset$
9: **else**
10: $C \leftarrow (D_L \bowtie_G D_R) \cup C_L \cup C_R$
11: $D \leftarrow F \setminus G$.
12: **return** (C, D)

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 2.

By induction on each step of the algorithm, we establish in the full version of this paper 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) $|C(x)| \le (|\mathcal{L}(\mathcal{T}(x))| - 1) \cdot P$; and (3)

$$|D(x)| \le \min\left\{ \min_{l \in \mathcal{L}(\mathcal{T}(x))} \{N_{[n]\setminus\{l\}}\}, \frac{\prod_{l \in \mathcal{L}(\mathcal{T}(x))} N_{[n]\setminus\{l\}}}{P^{|\mathcal{L}(\mathcal{T}(x))|-1}} \right\}.$$

Assuming the above three properties, we next prove that our algorithm correctly computes the join J. Let u denote the root of the tree $\mathcal{T}$. By property (1),

$$\begin{aligned} \pi_{\text{LABEL}(\text{LC}(u))}(J \setminus C(\text{LC}(u))) &\subseteq D(\text{LC}(u)) \\ \pi_{\text{LABEL}(\text{RC}(u))}(J \setminus C(\text{RC}(u))) &\subseteq D(\text{RC}(u)) \end{aligned}$$

Hence,

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

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).

Running Time. For the run time complexity of the above algorithm, 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, 10, and 11 of Algorithm 1 can be computed within the time bound using hashing. Using property (3) above, we have a (loose) upper bound of $O\left(nP + n \min_{l \in \mathcal{L}(\mathcal{T}(x))} N_{[n]\setminus\{l\}}\right)$ on the run time for node x. Summing the run time over all the nodes in the tree gives the claimed run time.

3.2 An Algorithm for All Join Queries

This section presents our algorithm for proving the AGM inequality that has a running time that matches the bound.

THEOREM 3.2. *Let $H = (V, E)$ be a hypergraph representing a natural join query. Let $n = |V|$ and $m = |E|$. Let $\mathbf{x} = (x_e)_{e \in E}$ be an arbitrary point in the fractional cover polytope*

$$\begin{aligned} \sum_{e : v \in e} x_e &\ge 1, \text{ for any } v \in V \\ x_e &\ge 0, \text{ for any } e \in E \end{aligned}$$

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

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

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

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

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

REMARK 3.3. *In the running time above, $m^2 n$ 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 (HT1), (HT2), and (HT3) from Section 3.2.3, then we can remove the term $n^2 \sum_{e \in E} N_e$ from the running time. To make the bound tight, the fractional cover solution $\mathbf{x}$ should be the best fractional cover in terms of the linear objective $\sum_e (\log N_e) \cdot x_e$. The data-preprocessing time of $O(n^2 \sum_e N_e)$ 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 $O(n \cdot n! \sum_e N_e)$-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 3.4. *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

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

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

3.2.1 The Algorithm and Terminology

Algorithm 2 has three main phases: (1) We first construct a labeled binary tree that we call a *query plan tree* or QP tree. Then, we construct a total order of attributes to be used in the next step. (2) Using the total order from phase (1), we construct a set of hash indices for various probing operations in the next step. In step (3), we give a recursive algorithm to compute the required join (whose recursion is based on the QP tree). The algorithm in (3) is similar to our LW algorithm: it uses a notion of heavy and light join keys, it computes a superset of the join and uses hash tables to filter this set. It does have some key technical differences: the structure of the recursion is different and the handling of heavy/light join keys is more general.

Algorithm 2 Computing the join $\bowtie_{e\in E} R_e$

Input: Hypergraph $H = (V, E)$, $|V| = n$, $|E| = m$
Input: Fractional cover solution $\mathbf{x} = (x_e)_{e\in E}$
Input: Relations $R_e, e \in E$
1: Compute the query plan tree $\mathcal{T}$, let u be $\mathcal{T}$'s root node
2: Compute a total order of attributes
3: Compute a collection of hash indices for all relations
4: **return** RECURSIVE-JOIN($u, \mathbf{x}$, NIL)

To make this section self-contained, we repeat some terminology and notation. For each tuple $\mathbf{t}$ on attribute set A, we will write $\mathbf{t}$ as $\mathbf{t}_A$ to emphasize the support of $\mathbf{t}$: $\mathbf{t}_A = (t_a)_{a\in A}$. Consider any relation R with attribute set S. Let $A \subset S$ and $\mathbf{t}_A$ be a fixed tuple. Then, $\pi_A(R)$ denotes the projection of R on to attributes in A and,

$$R[\mathbf{t}_A] := \pi_{S\setminus A}(R \ltimes \{\mathbf{t}_A\}) = \{\mathbf{t}_{S\setminus A} \mid (\mathbf{t}_A, \mathbf{t}_{S\setminus A}) \in R\}.$$

In particular, $R[\mathbf{t}_\emptyset] = R$. There is a complete worked example of our algorithm in the full version of this paper.

3.2.2 Step (1): Build a query plan tree

Given a query $H = (V, E)$, fix an arbitrary order $e_1, e_2, \ldots, e_m$ of *all* the hyperedges in E. We construct a labeled binary tree $(\mathcal{T}, \text{LC}, \text{RC})$ where LC (resp. RC) maps an internal node to their left child (resp. right child) and to a special constant NIL if no such child exists. Each node $x \in \mathcal{T}$ is equipped with a pair of functions $\text{LABEL}(x) \in [m]$ and $\text{UNIV}(x) \subseteq V$. Very roughly, each node x and the sub-tree below it forms the "skeleton" of a sub-problem. There will be many sub-problems that correspond to each skeleton. The value $\text{LABEL}(x)$ points to an "anchor" relation for the sub-problem and $\text{UNIV}(x)$ is the set of attributes that the sub-problem is joining on. The anchor relation divides the universe $\text{UNIV}(x)$ into two parts to further sub-divide the recursion structure.

Algorithm 3 Constructing the query plan tree $\mathcal{T}$

1: Fix an arbitrary order $e_1, e_2, \ldots, e_m$ of all the hyperedges in E.
2: $\mathcal{T} \leftarrow$ BUILD-TREE(V, m)

BUILD-TREE(U, k)
1: **if** $e_i \cap U = \emptyset, \forall i \in [k]$ **then**
2: **return** NIL
3: Create a node u with $\text{LABEL}(u) \leftarrow k$ and $\text{UNIV}(u) = U$
4: **if** $k > 1$ and $\exists i \in [k]$ such that $U \not\subseteq e_i$ **then**
5: $\text{LC}(u) \leftarrow$ BUILD-TREE$(U \setminus e_k, k-1)$
6: $\text{RC}(u) \leftarrow$ BUILD-TREE$(U \cap e_k, k-1)$
7: **return** u

Algorithm 3 builds the query plan tree $\mathcal{T}$. Note that line 5 and 6 will not be executed if $U \subseteq e_i, \forall i \in [k]$, in which case u is a leaf node. When u is not a leaf node, if $U \subseteq e_k$ then u will not have a left child ($\text{LC}(u) = \text{NIL}$). If $e_i \cap U \cap e_k = \emptyset$ for all $i \in [k-1]$ then u will not have a right child ($\text{RC}(u) = \text{NIL}$). The running time for this pre-processing step is $O(m^2 n)$. Figure 1 shows a query plan tree produced by Algorithm 3 on an example query.

From $\mathcal{T}$, we compute a total order on V in two steps. First, we define a partial order of all attributes by traversing the tree $\mathcal{T}$ in post-order. If a node u is visited before a node v, then all elements of $\text{UNIV}(u)$ precede elements $\text{UNIV}(v) \setminus \text{UNIV}(u)$ in this partial order. Second, we take an arbitrary linear extension of this partial order and call it *the total order*. A complete pseudocode listing of this routine can be found in the full version of this paper, along with a few properties of the total order. In the example of Figure 1, the total order is $1, 4, 2, 5, 3, 6$.

$$\begin{aligned} q = \quad & R_1(A_1, A_2, A_4, A_5) \\ \bowtie \quad & R_2(A_1, A_3, A_4, A_6) \\ \bowtie \quad & R_3(A_1, A_2, A_3) \\ \bowtie \quad & R_4(A_2, A_4, A_6) \\ \bowtie \quad & R_5(A_3, A_5, A_6) \end{aligned}$$

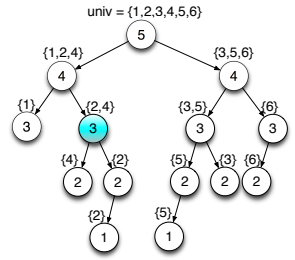

Figure 1: (a) A query q and (b) a sample QP tree for q.

3.2.3 Step (2): Build a Family of Indexes

We describe the form of the hash tables constructed by our algorithm. Each hash table is described by a triple $(i, \bar{K}, \bar{A})$ where $i \in [m]$, $\bar{K} \subseteq e_i$ is the search key, and $\bar{A} \subseteq e_i \setminus \bar{K}$ are the value attributes. For each such triple, our algorithm builds three hash tables that 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).

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

We will describe later how the total order allows us to reduce the total number hash indices down to $O(n^2 m)$. We will only need to build 3 hash tables for every triple $(i, \bar{K}, \bar{A})$ such that $\bar{K}$ precede $\bar{A}$ in the total order. Thus, for R_4 in Figure 1 we need to build at most 21 indexes, i.e., three indexes for each of the following nine different key pairs: $[(), (A_4)]$, $[(), (A_4, A_2)]$, $[(), (A_4, A_2, A_6)]$, $[(A_4), (A_2)]$, $[(A_4), (A_2, A_6)]$, $[(A_4, A_2), (A_6)]$, and $[(A_4, A_2, A_6), ()]$. (It is less than 21 because some indices are trivial or not defined, e.g. HT1 when $\mathbf{t} = ()$.) We group the pairs by brackets to make them easier to visually parse.

3.2.4 Step (3): Compute the Join Recursively

We are ready to present the heart of Algorithm 2 which computes the join recursively in a nested-loop-like fashion. The input to the algorithm consists of the hypergraph $H = (V, E)$ with $|V| = n$, $|E| = m$, and a point $\mathbf{x} = (x_e)_{e\in E}$ in the fractional cover polytope

$$\begin{aligned} \sum_{e: i\in e} x_e &\geq 1, \text{ for any } i \in V \\ x_e &\geq 0, \text{ for any } e \in E \end{aligned}$$

Throughout this section, we denote the final output by J which is defined to be $J = \bowtie_{e\in E} R_e$.

The crux of Algorithm 2 is a procedure called RECURSIVE-JOIN (Procedure 4) that takes as inputs three arguments: (1) a node u from the QP-tree $\mathcal{T}$ whose label is k for some $k \in [m]$. (2) A tuple $\mathbf{t}_S \in \mathbf{D}^S$ where S is the set of *all* attributes preceding $\text{UNIV}(u)$ in the total order, and (3) a fractional cover solution $\mathbf{y}_{E_k} = (y_{e_1}, \ldots, y_{e_k})$ of the hypergraph instance $(\text{UNIV}(u), E_k)$. (Here, $E_k = \{e_1, \ldots, e_k\}$ and

Procedure 4 RECURSIVE-JOIN($u, \mathbf{y}, \mathbf{t}_S$)

1: Let $U = \text{UNIV}(u)$, $k = \text{LABEL}(u)$
2: Ret $\leftarrow \emptyset$ // *Ret is the returned tuple set*
3: **if** u is a leaf node of $\mathcal{T}$ **then** // *note that* $U \subseteq e_i, \forall i \le k$
4: $\quad j \leftarrow \text{argmin}_{i\in[k]} \left\{|\pi_U(R_{e_i}[\mathbf{t}_{S\cap e_i}])|\right\}$
5: $\quad$ // *By convention,* $R_e[\text{NIL}] = R_e$ *and* $R_e[\mathbf{t}_\emptyset] = R_e$
6: $\quad$ **for** each tuple $\mathbf{t}_U \in \pi_U(R_{e_j}[\mathbf{t}_{S\cap e_j}])$ **do**
7: $\quad\quad$ **if** $\mathbf{t}_U \in \pi_U(R_{e_i}[\mathbf{t}_{S\cap e_i}])$, for all $i \in [k] \setminus \{j\}$ **then**
8: $\quad\quad\quad$ Ret $\leftarrow$ Ret $\cup \{(\mathbf{t}_S, \mathbf{t}_U)\}$
9: $\quad$ **return** Ret
10: **if** $\text{LC}(u) = \text{NIL}$ **then** // *u is not a leaf node of* $\mathcal{T}$
11: $\quad L \leftarrow \{\mathbf{t}_S\}$
12: $\quad$ // *note that* $L \neq \emptyset$ *and* $\mathbf{t}_S$ *could be* NIL *(when* $S = \emptyset$*)*
13: **else**
14: $\quad L \leftarrow$ RECURSIVE-JOIN($\text{LC}(u), (y_1, \ldots, y_{k-1}), \mathbf{t}_S$)
15: $W \leftarrow U \setminus e_k$, $W^- \leftarrow e_k \cap U$
16: **if** $W^- = \emptyset$ **then**
17: $\quad$ **return** L
18: **for** each tuple $\mathbf{t}_{S\cup W} = (\mathbf{t}_S, \mathbf{t}_W) \in L$ **do**
19: $\quad$ **if** $y_{e_k} \ge 1$ **then**
20: $\quad\quad$ **go to** line 27
21: $\quad$ **if** $\left(\prod_{i=1}^{k-1} |\pi_{e_i\cap W^-}(R_{e_i}[\mathbf{t}_{(S\cup W)\cap e_i}])|^{\frac{y_{e_i}}{1-y_{e_k}}} < |\pi_{W^-}(R_{e_k}[\mathbf{t}_{S\cap e_k}])|\right)$ **then**
22: $\quad\quad Z \leftarrow$ RECURSIVE-JOIN$\left(\text{RC}(u), \left(\frac{y_{e_i}}{1-y_{e_k}}\right)_{i=1}^{k-1}, \mathbf{t}_{S\cup W}\right)$
23: $\quad\quad$ **for** each tuple $(\mathbf{t}_S, \mathbf{t}_W, \mathbf{t}_{W^-}) \in Z$ **do**
24: $\quad\quad\quad$ **if** $\mathbf{t}_{W^-} \in \pi_{W^-}(R_{e_k}[\mathbf{t}_{S\cap e_k}])$ **then**
25: $\quad\quad\quad\quad$ Ret $\leftarrow$ Ret $\cup \{(\mathbf{t}_S, \mathbf{t}_W, \mathbf{t}_{W^-})\}$
26: $\quad$ **else**
27: $\quad\quad$ **for** each tuple $\mathbf{t}_{W^-} \in \pi_{W^-}(R_{e_k}[\mathbf{t}_{S\cap e_k}])$ **do**
28: $\quad\quad\quad$ **if** $\mathbf{t}_{e_i\cap W^-} \in \pi_{e_i\cap W^-}(R_{e_i}[\mathbf{t}_{(S\cup W)\cap e_i}])$ for all e_i such that $i < k$ and $e_i \cap W^- \neq \emptyset$ **then**
29: $\quad\quad\quad\quad$ Ret $\leftarrow$ Ret $\cup \{(\mathbf{t}_S, \mathbf{t}_W, \mathbf{t}_{W^-})\}$
30: **return** Ret

we only take the restrictions of hyperedges in E_k onto the universe $\text{UNIV}(u)$). More precisely, $\mathbf{y}_{E_k}$ is a point in the following polytope:

$$\sum_{e\in E_k : i\in e} y_e \quad \ge \quad 1, \text{ for any } i \in \text{UNIV}(u)$$
$$y_e \quad \ge \quad 0, \text{ for any } e \in E_k$$

The goal of RECURSIVE-JOIN is to compute a superset of the relation $\{\mathbf{t}_S\} \times \pi_{\text{UNIV}(u)}(J[\mathbf{t}_S])$, i.e., a superset of the output tuples that start with $\mathbf{t}_S$ on the attributes $S \cup \text{UNIV}(u)$. This intermediate output is analogous to the set C in Algorithm 1 for LW instances. A second similarity to Algorithm 1 is that our algorithm makes a choice per tuple based on the output's estimated size.

Theorem 3.2 is a special case of the following lemma where we set u to be the root of the QP-tree $\mathcal{T}$, $\mathbf{y} = \mathbf{x}$, and $S = \emptyset$ ($\mathbf{t}_S = \text{NIL}$). Finally, we observe that we need only $O(n^2)$ number of hash indices per input relation, which completes the proof.

LEMMA 3.5. *Consider a call* RECURSIVE-JOIN$(u, \mathbf{y}, \mathbf{t}_S)$ *to Procedure 4. Let* $k = \text{LABEL}(u)$ *and* $U = \text{UNIV}(u)$*. Then,*

(a) The procedure outputs a relation Ret on attributes $S \cup U$ *with at most the following number of tuples*

$$B(u, \mathbf{y}, \mathbf{t}_S) := \prod_{i=1}^{k} |\pi_{U\cap e_i}(R_{e_i}[\mathbf{t}_{S\cap e_i}])|^{y_i}.$$

(For the sake of presentation, we use the convention that when $U \cap e_i = \emptyset$ *we set* $|\pi_{U\cap e_i}(R_{e_i}[\mathbf{t}_{S\cap e_i}])| = 1$ *so that the factor does not contribute anything to the product.)*

(b) Furthermore, the procedure runs in time $O(mn \cdot B(u, \mathbf{y}, \mathbf{t}_S))$.

The lemma is proved by induction on the height of the sub-tree of $\mathcal{T}$ rooted at u. We include a full formal proof in the full version of this paper, but give the main ideas here.

Base Case. In the base case, the node u is a leaf, and $\text{UNIV}(u) \subseteq e_i, \forall i \in [\text{LABEL}(u)]$. Observe that

$$\min_{i=1,\ldots,k} |\pi_U(R_{e_i}[\mathbf{t}_S])| \le \prod_{i=1}^{k} |\pi_U(R_{e_i}[\mathbf{t}_S])|^{y_i} = B(u, \mathbf{y}, \mathbf{t}_S).$$

since $\sum_{i=1}^{k} y_i \ge 1$ ($\mathbf{y}$ is a fractional cover solution). Because the left-hand-side of the above inequality is clearly an upper bound on the number of output tuples, so is the right-hand-side. Hence, (a) holds. To control the running time and prove (b), a nested-loop strategy works: we first find the j that achieves the left-hand side of the inequality, i.e., for which $\pi_U(R_{e_j}[\mathbf{t}_S])$ is smallest among $j \in [k]$. To find this minimum, we probe HT2(i, S, U) with search key $\mathbf{t}_S$ for $i \in [k]$ (i.e., once for each relation). Since u is a leaf node, $\text{UNIV}(u) \subseteq e_i$ for each $i \in [k]$ and hence $U = U \cap e_i$. Thus, we can query HT3(j, S, U) to find all tuples in relation $\pi_U(R_{e_j}[\mathbf{t}_S])$ in time $O(|\pi_U(R_{e_j}[\mathbf{t}])|)$. Then, for each such tuple $\mathbf{v} \in \pi_U(R_{e_j}[\mathbf{t}])$, and for each relation R_{e_i} with $i \in [k] \setminus \{j\}$ we probe into R_{e_i} with HT1(i, S, U); the tuple $\mathbf{v}$ is returned iff this probe returns true for all $i \in [k] \setminus \{j\}$. This procedure takes $O(kn + kn|\pi_U(R_{e_j}[\mathbf{t}_S])|)$ where j is the minimum as above.

Induction Step. In this case, u is an internal node. The key challenge is that we have to make a cost-based decision about which lower subproblems to solve. The interesting case is when there are both a left- and a right-child problem. We recursively solve the left subproblem, from which we get back a relation on $S \cup U \setminus e_k$ (each of whose tuples has values $\mathbf{t}_S$ on attributes S), which we then store in a variable L (formally, $L \supseteq \{\mathbf{t}_S\} \times \pi_{U\setminus e_k}(J[\mathbf{t}_S])$). For example, consider the highlighted node in Figure 1(b). We will refer to this node throughout this section as our example. Here, $S = \{1\}$ and so we have a fixed tuple $\mathbf{t}_S$ as input. The left-subproblem is the leaf that computes the tuples in $\{\mathbf{t}_S\} \times \pi_{\{4\}}(J[\mathbf{t}_S])$, which have support $\{1, 4\}$.

Next, for each tuple $\mathbf{t} = \mathbf{t}_{S\cup(U\setminus e_k)} \in L$, we will make a decision on whether to solve an associated "right subproblem." There are $|L|$ such subproblems and thus $|L|$ decisions to make. Each decision is based on our estimation of the running time if we were to solve the subproblem. The run-time estimation is the AGM's bound on the output size of the sub-problem. To obtain the estimation, we define for each right subproblem a fractional cover solution. The relation R_{e_k} is used as an "anchor" for the entire process.

Specifically, we construct a hypergraph $(\text{UNIV}(\text{RC}(u)), E_{k-1})$ with an associated fractional cover $\mathbf{y}'_{E_{k-1}}$ where $y'_{e_i} = y_{e_i}/(1 - y_{e_k})$ for $i \in [k-1]$. (When $y_{e_k} \ge 1$ we will not solve this subproblem and directly take option (1) below.) For each $\mathbf{t} \in L$, the input relation sizes for this sub-problem are $|\pi_{e_i\cap U\cap e_k}(R_{e_i}[\mathbf{t}])|$ for $i \in [k-1]$.

For each $\mathbf{t}_{S\cup W} = (\mathbf{t}_S, \mathbf{t}_W) \in L$, where $W = U \setminus e_k$, our algorithm considers two options, and we use the estimated run-time of the projected subproblem to choose between these options.

Option (1) Our algorithm loops over each tuple in $\pi_{U\cap e_k}(R_{e_k}[\mathbf{t}_{S\cap e_k}])$ and filters it against all projections that are below it (lines 27–29). In this case our running time is $O(|\pi_{U\cap e_k}(R_{e_k})[\mathbf{t}_{S\cap e_k}]|)$. In our running example, given the tuple $\mathbf{t}_{\{1,4\}} = (t_1, t_4) \in L$, we would loop over each tuple $\mathbf{t}_{\{2\}} = (t_2) \in \pi_{\{2\}}(R_3[t_1])$.

For each such tuple, we add (t_1, t_4, t_2) to the output Ret if $t_2 \in \pi_{\{2\}}(R_1[(t_1, t_4)])$. This check can be done by probing R_1 using HT1$(1, (A_1, A_4, A_2), ())$.

Option (2) Our algorithm solves the right subproblem recursively and filters the result with $\pi_{U\cap e_k}(R_{e_k}[\mathbf{t}_{S\cap e_k}])$ (lines 22–25). In our running example, given $(t_1, t_4) \in L$ the right subproblem will compute those tuples (t_2) in $\pi_{\{2\}}(R_1[\mathbf{t}_{\{1,4\}}])$ and then filter them with HT1$(3, (A_1, A_2), ())$. The important property of option (2) is that its running time does *not* depend on $|\pi_{U\cap e_k}(R_{e_k}[\mathbf{t}_{S\cap e_k}])|$. In particular, option (2)'s running time only depends on the output size of the right subproblem.

To decide between these two options, we compare the following two quantities:

$$\text{LHS} = |\pi_{U\cap e_k}(R_{e_k}[\mathbf{t}_{S\cap e_k}])| \text{ versus RHS} = \prod_{i=1}^{k-1} \left|\pi_{e_i\cap U\cap e_k}(R_{e_i}[\mathbf{t}_{(S\cup W)\cap e_i}])\right|^{y'_{e_i}}$$

We choose option (1) if either $y_{e_k} \geq 1$ or the LHS is less than the RHS and option (2) otherwise. Observe that we can compute both quantities given our indices in time proportional to $O(kn)$. Our overall running time is proportional to the minimum of these two quantities (plus the inconsequential term $O(kn)$). Summing over all tuples $\mathbf{t} \in L$ the minimum of the above two quantities and "unroll" the sum by applying a generalized Hölder's inequality many times we can then prove both the output size and the running time.

Used Search Keys. Finally, we need to understand which search keys are used in the hash table. Observe that whenever an attribute v is used in a search key (e.g., HTw(i, S, U) for $w \in \{1, 2, 3\}$), all attributes that come before v in the total order and are in e_i are *bound*. Thus, if $e_i = (v_{i_1}, \ldots, v_{i_{|e_i|}})$ and the attributes are ordered as above, then the search key and the returned keys is always a *prefix* of $v_{i_1}, \ldots, v_{i_k}$. Hence, we only need to have $3n \sum_{e\in E} |e|$ indices. In the full version of this paper, we describe a slightly more complex data structure that combines all hash tables for one relation into a single "search tree" structure. The search trees have the advantage that their building time is deterministic (unlike a typically perfect hashing scheme which has a constant worst-case lookup time but only expected linear building time). However, the search trees necessitate a log-factor blow up in the total run time of our algorithm.

3.3 Limits of Standard Approaches

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 running-time gap.

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 3.6. *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 $[n] \setminus \{i\}$, 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 running times are asymptotically worse than the best AGM bound for this instance which is $|\bowtie_{i=1}^n R_i| \leq \prod_{i=1}^n |R_i|^{1/(n-1)} = N^{1+1/(n-1)}$. On the other hand, both Algorithm 1 and Algorithm 2 take $O(N^{1+1/(n-1)})$-time as we have analyzed. In fact, for Algorithm 2, we are able to demonstrate a stronger result: its run-time 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 run-time gap between Algorithm 2 and AGM's algorithm is $\Omega(N)$ for constant n.

Proof of Lemma 3.6. 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 $\mathbf{D}^{[n]-\{i\}}$ 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, $|\bowtie_{i=1}^n R_i| = n[(N-1)/(n-1)+1] - (n-1) = N + (N-1)/(n-1)$.

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 and this step alone requires $\Omega(N^2/n^2)$ run time. □

Finally, we analyze the run-time of Algorithm 2 directly on this instance without resorting to Theorem 3.2.

Lemma 3.7. *On the collection of instances from the previous lemma, Algorithm 2 runs in time $O(n^2N)$.*

Proof. Without loss of generality, assume the hyperedge order Algorithm 2 considers is $[n] - \{1\}, \ldots, [n] - \{n\}$. In this case, the universe of the left-child of the root of the QP-tree is $\{n\}$, and the universe of the right-child of the root is $[n-1]$.

The first thing Algorithm 2 does is that it computes the join $L_n = \bowtie_{i=1}^{n-1} \pi_{\{n\}}(R_i)$, in time $O(nN)$. Note that $L_n = \mathbf{D}$, the domain. Next, Algorithm 2 goes through each value $a \in L_n$ and decides whether to solve a subproblem or not. First, consider the case $a > 0$. Here Algorithm 2 estimates a bound for the join $\bowtie_{j=1}^{n-1} \pi_{[n-1]}(R_j[a])$. 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 $|\pi_{[n-1]}(R_j[0])| = \frac{(n-2)N-1}{(n-1)}$. The subproblem's estimated size bound is

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

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 $U \subseteq V$ then call an attribute $v \in V \setminus U$ *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:

[5] We thank an anonymous PODS'12 referee for showing us that our example works for all join-project plans rather than just the AGM algorithm and arbitrary join-only algorithms.

Lemma 3.8. *Given a join query $H = (V, E)$ and some $U \subseteq V$ where $|U| \geq 2$, then 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 3.2 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 3.6 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 running time 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.

4. EXTENSIONS

4.1 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 2 with an asymptotically better query complexity.

Further, given promising results in the worst case, it is natural to wonder whether or not 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.

Each relation has at most 2 *attributes.* As is mentioned in the introduction, our algorithm in Theorem 3.2 not only has better data complexity than AGM's algorithm (in fact we showed our algorithm has optimal worst-case data complexity), it has 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 better query complexity compared to the algorithm in Theorem 3.2 (while retaining 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 which can be assumed to be simple. We assume that all relations are indexed in advanced, which takes $O(\sum_e N_e)$ time. In what follows we will not include this preprocessing time in the analysis.

We first state a lemma for the case when H is a cycle.

Lemma 4.1 (Cycle Lemma). *If H is a cycle, then $\bowtie_{e \in E} R_e$ can be computed in time $O(m \sqrt{\prod_{e \in H} N_e})$.*

The proof of the lemma shows that we can reduce the computation of the case when H is a cycle to our previous algorithm for Loomis-Whitney instances with $n = 3$.

With the help of Lemma 4.1, 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

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

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 [34], Theorem 30.10). However, we will also need a graph structure associated with the half-integral solution; hence, we adapt a known proof of the half-integrality property with a slightly more specific analysis [34].

Lemma 4.2. *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 $\mathcal{C}$ of vertex-disjoint odd-length cycles that are also vertex disjoint from the union S of stars.*

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

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

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 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 4.1 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 summarize the above discussion in the following theorem.

Theorem 4.3. *When each relation has at most two attributes, we can compute the join $\bowtie_{e \in E} R_e$ in time $O(m \prod_{e \in E} N_e^{x_e})$.*

Impossibility of Instance Optimality. 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 [37].

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 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 ϕ: in other words, $q(I)$ has one element if ϕ is satisfiable otherwise $q(I) = \emptyset$. In other words, we have $|q(I)| \le 1$, $|q| = O(m+n)$ 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 ϕ is satisfiable or not in time $\text{poly}(n, m)$, which would imply NP = RP.

4.2 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 $V = [n]$ and $|E| = m$. The m input relations are R_e, $e \in E$. We are also given a "relaxation" number $0 \le r \le m$. Our goal is to output all tuples that agree with at least $m - r$ input relations. In other words, we want to compute $\cup_{S \subseteq E, |S| \ge m-r} \bowtie_{e \in S} R_e$. 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. Towards this end, define the set

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

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

Definition 4.4 (Relaxed join problem). *Given a query q represented by the hypergraph $H = (V = [n], E)$, and an integer $0 \le r \le m$, evaluate*

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

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 $S \ne T \in \hat{C}(q, r)$ neither $S \subset T$ nor $T \subset S$. We only need to evaluate $q_r = \bigcup_{S \in \hat{C}(q,r)} (\bowtie_{e \in S} R_e)$.

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

$$\begin{array}{lll} \min & \sum_{e \in S} (\log |R_e|) \cdot x_e & \\ \text{subject to} & \sum_{e \in S : v \in e} x_e \ge 1 & \text{for any } v \in V \\ & x_e \ge 0 & \text{for any } e \in S. \end{array} \tag{4}$$

Upper bounds. We start with a straightforward upper bound.

Proposition 4.5. *Let q be a join query on m relations and let $0 \le r \le m$ 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 2 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 $\text{LP}(S)$ defined in (4). (If there are multiple such solutions, 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 4.6. *Let q be a join query represented by $H = (V, E)$, and let $0 \le r \le m$ 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*

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

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 4.6 is no worse than that in Proposition 4.5. We will show later that the bound in Theorem 4.6 is indeed tight.

We defer the proof of Theorem 4.6 to the full version and mention the main idea here. Let $S \ne S' \in \hat{C}(q, r)$ be two different sets of hyperedges with the following property. Define $T \stackrel{def}{=} \mathsf{BFS}(S) = \mathsf{BFS}(S')$ 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 2 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 $\bowtie_{e \in T} R_e$ once for both S and S'.

Lower bound. We now show that the bound in Theorem 4.6 is (almost) tight for some query and some database instance I.

We first define the query q. The hypergraph is $H = (V = [n], E)$ where $m = |E| = n + 1$. 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, $R_{e_{n+1}} = \bigcup_{i=1}^{N} \{N + i\}^n$.

It is easy to check that for any $r \ge n$, $q_r(I)$ is the set $R_{e_{n+1}} \cup [N]^n$, i.e. $|q_r(I)| = N + N^n$. (For $0 < r < n$, we have $|q_r(I)| = N^n$.) Next, we claim that for this query instance for any $r > 0$, $C^*(q, r) = \{\{n+1\}, [n]\}$. 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 4.6 implies that $|q_r(I)| \le N + N^n$, which proves the tightness of the size bound in Theorem 4.6 for $(r \ge n)$, as desired. (For $0 < r < n$, the bound is almost tight.)

Finally, we argue that $C^*(q, r) = \{\{n+1\}, [n]\}$. Towards this end, consider any $T \in \hat{C}(q, r)$. Note that if $(n+1) \notin T$, 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 all the n attributes. This implies that $\mathsf{BFS}(T) = \{n+1\}$ in this case. This proves that $\{n+1\}$ is the other element in $C^*(q, r)$, as desired.

5. CONCLUSION AND FUTURE WORK

We establish optimal algorithms for the worst-case behavior of join algorithms. We also demonstrate that the join algorithms employed in RDBMSs do not achieve these optimal bounds. Moreover, we demonstrate families of instances where join-project algorithms are 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 different way to approach join optimization rather than the traditional binary-join/dynamic-programming-based approach. Thus, our immediate future work is to implement these ideas to see how they compare in real RDBMS settings to the algorithms in a modern RDBMS.

Another interesting direction is to extend these results to a larger classes of queries and to database schemata that have constraints. We include in the full version 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.

There are potentially interesting connections between our work and several inter-related topics. We algorithmically prove that the AGM inequality is equivalent to the BT inequality; in turn both inequalities are essentially equivalent to Shearer's entropy inequality. There are known combinatorial interpretations of entropy inequalities (which include Shearer's as a special case); for example, Alon et al. [3] derived some such connections using a notion of "sections" similar to what we used in this paper. An analogous partitioning procedure is used by Marx [29] to compute joins by relating the number of solutions to submodular functions. Query (1) is essentially equivalent to the problem of enumerating all triangles in a tri-partite graph, which can be solved in time $O(N^{3/2})$ [4].

6. ACKNOWLEDGMENTS

We thank Georg Gottlob for sending us a full version of his work [13] and XuanLong Nguyen for introducing us to the Loomis-Whitney inequality. We thank the anonymous referees for many helpful comments that greatly improved the presentation of the paper. In particular, we thank a reviewer for pointing out the current proof (and statement) of Lemma 3.6 and an error in previous lower bound argument in Section 4.2. AR's work on this project is supported the NSF CAREER Award under CCF-0844796. CR's work on this project is generously supported by the NSF CAREER Award under IIS-1054009, the ONR under N000141210041, and gifts from Google, Greenplum, LogicBlox, and Oracle.

7. REFERENCES

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

[2] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *PODS*, pages 10–20, 1999.

[3] N. Alon, I. Newman, A. Shen, G. Tardos, and N. K. Vereshchagin. Partitioning multi-dimensional sets in a small number of "uniform" parts. *Eur. J. Comb.*, 28(1):134–144, 2007.

[4] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[5] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748. IEEE, 2008.

[6] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD Conference*, pages 261–272, 2000.

[7] S. Babu, P. Bizarro, and D. J. DeWitt. Proactive re-optimization. In *SIGMOD Conference*, pages 107–118, 2005.

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

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

[10] A. Deligiannakis, M. N. Garofalakis, and N. Roussopoulos. Extended wavelets for multiple measures. *TODS*, 32(2):10, 2007.

[11] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[12] A. C. Gilbert, H. Q. Ngo, E. Porat, A. Rudra, and M. J. Strauss. Efficiently decodable ℓ_2/ℓ_2 for each compressed sensing with tiny failure probability, November 2011. Manuscript.

[13] G. Gottlob, S. T. Lee, and G. Valiant. Size and treewidth bounds for conjunctive queries. In *PODS*, pages 45–54, 2009.

[14] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: A survey. In *MFCS*, 2001.

[15] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized hypertree decompositions: np-hardness and tractable variants. In *PODS*, 2007.

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

[17] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[18] K. Gyarmati, M. Matolcsi, and I. Z. Ruzsa. A superadditivity and submultiplicativity property for cardinalities of sumsets. *Combinatorica*, 30(2):163–174, 2010.

[19] T. S. Han. Nonnegative entropy measures of multivariate symmetric correlations. *Information and Control*, 36(2):133–156, 1978.

[20] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[21] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *SIGMOD Conference*, 1991.

[22] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[23] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. In *VLDB*, 1998.

[24] A. C. König and G. Weikum. Combining Histograms and Parametric Curve Fitting for Feedback-Driven Query Result-size Estimation. In *VLDB*, 1999.

[25] A. R. Lehman and E. Lehman. Network coding: does the model need tuning? In *SODA*, pages 499–504, 2005.

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

[27] R. Lyons. Probability on trees and networks, jun 2011. with Yuval Peres url: http://php.indiana.edu/ rdlyons/prbtree/prbtree.html.

[28] V. Markl, N. Megiddo, M. Kutsch, T. M. Tran, P. J. Haas, and U. Srivastava. Consistently estimating the selectivity of conjuncts of predicates. In *VLDB*, pages 373–384, 2005.

[29] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC*, pages 735–744, 2010.

[30] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms, 2012. arXiv:1203.1952 [cs.DB].

[31] H. Q. Ngo, E. Porat, and A. Rudra. Personal Communciation.

[32] A. Pagh and R. Pagh. Scalable computation of acyclic joins. In *PODS*, pages 225–232, 2006.

[33] V. Poosala, Y. Ioannidis, P. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, pages 294–305, 1996.

[34] A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency. Vol. A*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.

[35] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. In *ICDE*, page 39, 2006.

[36] K. Tzoumas, A. Deshpande, and C. S. Jensen. Lightweight graphical models for selectivity estimation without independence assumptions. *PVLDB*, 4(11):852–863, 2011.

[37] L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[38] D. E. Willard. Applications of range query theory to relational data base join and selection operations. *J. Comput. Syst. Sci.*, 52(1), 1996.

[39] Y. Xu, P. Kostamaa, X. Zhou, and L. Chen. Handling data skew in parallel joins in shared-nothing systems. In *SIGMOD*, 2008.