Approximate Aggregate Queries Under Additive Inequalities

Mahmoud Abo-Khamis
Relational AI

Sungjin Im [*]
University of California, Merced

Benjamin Moseley [†]
Carnegie Mellon University

Kirk Pruhs [‡]
University of Pittsburgh

Alireza Samadian
University of Pittsburgh

November 10, 2020

Abstract

We consider the problem of evaluating certain types of functional aggregation queries on relational data subject to additive inequalities. Such aggregation queries, with a smallish number of additive inequalities, arise naturally/commonly in many applications, particularly in learning applications. We give a relatively complete categorization of the computational complexity of such problems. We first show that the problem is NP-hard, even in the case of one additive inequality. Thus we turn to approximating the query. Our main result is an efficient algorithm for approximating, with arbitrarily small relative error, many natural aggregation queries with one additive inequality. We give examples of natural queries that can be efficiently solved using this algorithm. In contrast, we show that the situation with two additive inequalities is quite different, by showing that it is NP-hard to evaluate simple aggregation queries, with two additive inequalities, with any bounded relative error.

1 Introduction

Kaggle surveys [1] show that the majority of learning tasks faced by data scientists involve relational data. Most commonly, the relational data is stored in tables in a relational database. So these data scientists want to compute something like

> **Data Science Query** = Standard_Learning_Task(Relational Tables $T_1, \ldots T_m$)

However, almost all standard algorithms for standard learning problems assume that the input consists of points in Euclidean space [10], and thus are not designed to operate directly on relational data. The current standard practice for a data scientist, confronted with a learning task on relational data, is:

1. Firstly, convert any non-numeric categorical data to numeric data. As there are standard methods to accomplish this [10], and as we do not innovate with respect to this process, we will assume that all data is a priori numerical, so we need not consider this step.
2. Secondly, issue a feature extraction query to extract the data from the relational database by joining (inner join) together the tables to materialize a design matrix $J = T_1 \bowtie \cdots \bowtie T_m$ with say N rows and d columns/features. Each row of this design matrix is then interpreted as a point in d-dimensional Euclidean space.
3. Finally this design matrix J is imported into a standard learning algorithm to train the model.

Thus conceptually, standard practice transforms a data science query to a query of the following form:

> **Data Science Query** = Standard_Learning_Algorithm(Design Matrix $J = T_1 \bowtie \cdots \bowtie T_m$)

where the joins are evaluated first, and the learning algorithm is then applied to the result. Note that if each table has n rows, the design matrix J can have as many as n^m entries. Thus, independent of the learning task, this standard practice necessarily has exponential worst-case time and space complexity as the design matrix can be exponentially larger than the underlying relational tables. Thus, a natural research question is what we call the relational learning question:

> **Relational Learning Question:** What standard learning problems admit relational algorithms, which informally are algorithms that are efficient when the data is in relational form?

Note that a relational algorithm can not afford to explicitly join the relational tables.

[*] Supported in part by NSF grants CCF-1409130, CCF-1617653, and CCF-1844939.

[†] Supported in part by NSF grants CCF-1725543, 1733873, 1845146, a Google Research Award, a Bosch junior faculty chair and an Infor faculty award.

[‡] Supported in part by NSF grants CCF-1421508, CCF-1535755, CCF-1907673, CCF-2036077 and an IBM Faculty Award.

In this paper, we consider the relational learning question in the context of the problem of evaluating Functional Aggregate Queries (FAQ's) subject to additive constraints, which we call FAQ-AI queries. Such queries/problems, with a smallish number of inequalities, arise naturally as subproblems in many standard learning algorithms. Before formally defining an FAQ-AI query, let us start with some examples. The first collection of examples are related to the classic Support Vector Machines problem (SVM), in which points are classified based on the side of a hyperplane that the point lies on [10, 25]. Each of the following examples can be reduced to FAQ-AI queries with one additive inequality, which we call FAQ-AI(1) queries:

- Counting the number of points correctly (or incorrectly) classified by a hyperplane.
- Finding the minimum distance of a correctly classified point to the boundary of a given hyperplane.
- Computing the gradient of the SVM objective function at a particular point.

And now we give some examples of problems related to the classic k-means clustering problem [25], in which the goal is to find locations for k centers so as to minimize the aggregate 2-norm squared distance from each point to its closest center. Each of the following examples can be reduced to FAQ-AI queries with $k - 1$ inequalities:

- Evaluating the k-means objective value for a particular collection of k centers.
- Computing the new centers in one iteration of the commonly used Lloyd's algorithm.
- Computing the furthest point in each cluster from the center of that cluster.

All of these problems are readily solvable in nearly linear time in the size of the input if the input was the design matrix. Our goal is to determine whether relational algorithms exist for such FAQ-AI problems when the input is in relational form.

One immediate difficulty that we run into is that if the tables have a sufficiently complicated structure, almost all natural problems/questions about the design matrix are NP-hard if the data is in relational form. For example, it is NP-hard to even determine whether or not the design matrix is empty (see for example [14, 21]). Thus, as we want to focus on the complexity of the functional aggregate query and the additive inequalities, we conceptually want to abstract out the complexity of the tables. The simplest way to accomplish this is to primarily focus on instances where the structure of the tables is simple, with the most natural candidate for "simplicity" being that the join is acyclic (see Definition 1). Acyclic joins are the norm in practice and are a commonly considered special case in the database literature. For example, there are efficient algorithms to compute the size of the design matrix for acyclic joins.

Formally defining what a "relational" algorithm is problematic, as for each natural candidate definition there are plausible scenarios in which that candidate definition is not the "right" definition. But for the purposes of this paper, it is sufficient to think of a "relational" algorithm as one whose runtime is polynomially bounded in n, m and d if the join is acyclic.

1.1 Informal Statement of Results

Here we informally state the results. The formal theorem statements are given in their respective sections.

Recall that FAQ-AI(1) is a FAQ-AI query with one additive inequality. We start by showing Theorem 4 in Section 3 that the FAQ-AI(1) problem is $\#P$-hard, even for the problem of counting the number of rows in the design matrix for a cross product join. Thus a relational algorithm for FAQ-AI(1) queries is extraordinarily unlikely as it would imply $P = \#P$, and therefore, $P = NP$.

Thus, we turn to approximately computing FAQ-AI(1) queries. An ideal result would be what we call a *Relational Approximation Scheme* (RAS), which is a collection $\{A_\epsilon\}$ of relational algorithms, one for each real $\epsilon > 0$, such that each A_ϵ is outputs a solution that has relative approximation error at most ϵ. Unfortunately, for FAQ-AI(2) we show that this problem does not admit a bounded approximation unless $P = NP$ using a relational algorithm. See Theorem 3.

Our main result is a RAS for FAQ-AI(1) queries that have certain natural properties. This is shown in Theorems 10 and 11. The formal definitions are cumbersome and we believe it would be more illuminating to list problems for which our techniques will give a RAS. We explain how to apply our results to get these results, as well as give some examples of problems for which our results are not applicable, in Section 7.

- Counting the number of points on one side of a hyperplane.
- Counting the number of points within a hypersphere of radius r centered at a point y.
- Counting the number of points in an axis-parallel ellipsoid.
- Sum of 1-norm distances from a point y of points on one side of a hyperplane.
- Sum of 2-norm squared of points in an axis-parallel ellipsoid.
- Number of nonzero entries of points on one side of a hyperplane.
- Finding the minimum 1-norm of any point in a hypersphere H of radius r centered at a point y.

- Finding the point on the specified side of a hyperplane H that has the maximum 2-norm distance from a point y.

To illustrate the results consider the first example above of counting points on one side of the hyperplane. This is a FAQ-AI(1) query. Intuitively, being on one side of a hyperplane can be represented as one additive constraint. For this problem, we can approximately count the number within $(1+\epsilon)$ factor for arbitrary $\epsilon > 0$ in time polynomial in the size of the tables input, parameterized by n, m and d, when the join is acyclic. If the join is not acyclic, then we can decompose the query to make it acyclic. This is the standard approach in the area and the running time depends polynomially on the size of the decomposition, a.k.a. the fractional hyper-treewidth (see Appendix C).

To summarize, we give a relatively complete characterization of FAQ-AI queries. FAQ-AI(1) problem is NP-Hard and admits a relational approximation scheme. Further, a FAQ-AI(k) for $k \geq 2$ is inapproximable.

1.2 Overview of the Algorithmic Techniques and Contributions

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$J = T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3

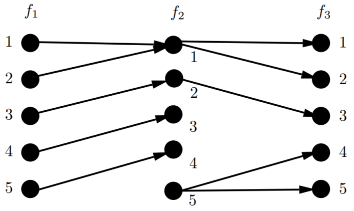

Figure 1: A specific instance in which $m = 2$ and $n = 5$. In particular, this shows T_1, T_2, the design matrix J, and the resulting layered directed graph G.

To help the reader understand our contributions without going through the formal definitions of the setup, including FAQ's, we first discuss the following very special case of the problem considered – it is special in both (i) *table structure* (how columns of the tables are correlated) and (ii) *query type* (the query outcome).

For (i), consider *path joins* which is a simple type of joins that can be modeled as a directed acyclic graph (DAG). A path join $J = T_1 \bowtie \cdots \bowtie T_m$, consists of m input tables where each input table T_i has two columns/features f_i and f_{i+1}. Then the join can be modeled as a layered DAG G in which there is one layer for each column and one vertex v in layer i for each entry value that appears in the f_i column in either table T_{i-1} or table T_i.

Further, in G, there is a directed edge between a vertex v in layer i and a vertex u in layer $i+1$ if and only if (v, u) is a row in table T_i. Then there is a one-to-one correspondence between full paths in G, which are paths from layer 1 to layer d, and rows in the design matrix (the outcome of the join). Each node v in G is associated with weight w_v which is an entry of a table where v appears. For simplicity think of the weights as being non-negative. For illustration of a path join and its analogy with DAGs, see Figure 1.

For (ii), consider the problem of counting the number of points in the design matrix on one side of a hyperplane, which we will call the Inequality Row Counting problem. This is equivalent to following problem assuming the above path join. You are given a value B and the goal is to count the distinct paths P in G from layer 1 to d such that $\sum_{v \in P} w_v \leq B$.

One can think of this as finding paths such that the summation of the node weights on the paths can fit into a knapsack of size B. This problem is a generalization of the knapsack counting problem. In the knapsack counting problem, one is given a knapsack of size B and n items of weight $k_1, k_2, \ldots k_n$. The goal is to count the number of feasible packings of the knapsack. To see this is a special case of problem, we can create a graph with n layers. Layer i corresponds to item i. The nodes have weight 0 and k_i. Every node in layer i each has edges to both nodes in layer $i+1$. A path now from the first to last layer describes a set of items. Going through the node of weight 0 in layer i corresponds to not choosing item i and going through the node of weight k_i is as if the item is chosen. It is easy to see now that counting the paths which total weight is less than B is precisely the solution to counting knapsack.

FAQ-AI(1) queries are generally closely related to the knapsack counting problem and techniques known for this problem will prove useful. It is known how to approximate this problem to a $1+\epsilon$ factor. This is done by writing a dynamic program. Order the items $1, 2, \ldots n$. The dynamic program is $D(i, N)$. This stores the minimum total weight needed to obtain at least N feasible distinct knapsack packings from items $1, \ldots, i$. Of course N could be exponential, and the idea is to approximate the dynamic program by only counting the number of solutions for the form $(1+\epsilon)^j$ for integer j. To solve Inequality Row Counting on a path join, we have to be more careful than when solving the knapsack counting. This is because the multi-layered DAG can be more complicated and therefore more refined subproblems should be considered. Further, this leads to accumulation of the approximation error of the subproblem objectives and we have to be more careful when applying the multiplication and summation operations in a sequence.

While adapting the key idea for knapsack counting problem to Inequality Row Counting for a path join requires non-trivial work, our main contribution is in introducing an algorithmic technique we call a *dynamic programming semiring*. Eventually, we want to join tables that are correlated in a more complicated way than the simple path join. Luckily there is a so-called Inside-Out algorithm for Sum-Product queries that can be used to process semiring type queries for arbitrary tables. Unfortunately, it is not obvious if we can find semiring structure in dynamic programming. In our dynamic programming semiring, the base elements can be thought of as arrays, and the summation and product operations in FAQ are designed so that the SumProd query computes a desired dynamic program. This conceptual contribution allows us to use the Inside-Out algorithm for joining arbitrary tables and for various queries. Further, to keep the computation compact, we extend the operations to be approximate while paying close attention to the order in which they are applied for approximation guarantees.

Given the widespread utility of dynamic programming as a algorithm design technique, it seems to us likely that dynamic programming semirings will be useful in designing relational algorithms for other problems.

1.3 Organization of the Paper

In Section 2 we give a formal definition of FAQ-AI. We also discuss the Inside-Out algorithm for SumProd queries over acyclic joins, as our algorithms will use the Inside-Out algorithm. We also formally define approximate operators to control the approximation quality. In Section 3 we give hardness results for FAQ-AI queries, which motivate us to study Approximate FAQ-AI(1). In Section 4, we explain how to obtain a RAS for a special type of FAQ-AI(1) query, an Inequality Row Counting Query, that counts the number of rows in the design matrix that satisfy a given additive inequality. This can be used to illustrate our algorithmic ideas. Then, in Sections 5 and 6 we show how to extend our algorithmic ideas to obtain RAS for two types of FAQ-AI(1) queries, namely SumSum and SumProd, which will be defined later. Finally, in Section 7 we show how some applications fit into our approximate FAQ-AI(1) framework.

1.4 Related Results

The Inside-Out algorithm [6] can evaluate a SumProd query in time $O(md^2n^h \log n)$, where h is the fractional hypertree width [15] of the query (assuming unit time per semiring operation). Note that $h = 1$ for the acyclic joins, and thus Inside-Out is a polynomial time algorithm for acyclic joins. One can reduce SumSum queries to m SumProd queries [2], and thus they can be solved in time $O(m^2d^2n^h \log n)$. Conceptually the Inside-Out algorithm is a generalization of Dijkstra's shortest path algorithm. We explain Inside-Out in more detail in Section 2.2. The Inside-Out algorithm builds on many earlier papers, including [8, 13, 17, 15].

FAQ-AI queries were introduced in [2]. [2] gave an algorithm with worst-case time complexity $O(md^2n^{m/2} \log n)$. So this is better than the standard practice of forming the design matrix, which has worst-case time complexity $\Omega(dn^m)$. Conceptually the improvement over standard practice arises because the algorithm in [2] avoids the last join.

Different flavors of queries with inequalities were also studied [16, 18, 5]. Relational algorithms for linear/polynomial regression are considered in [24, 3, 4, 19, 20]. An algorithm for k-means on relational data is given in [12].

2 Preliminaries

This section introduces the formal definitions of FAQ-AI and useful known algorithms for this and related problems.

2.1 FAQ-AI

The input to FAQ-AI problem consists of three components:

- A collection of relational tables $T_1, \dots T_m$ with real-valued entries. Let $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ be the design matrix that arises from the inner join of the tables. Let n be an upper bound on the number of rows in any table T_i, let N be the number of rows in J, and let d be the number of columns/features in J.

- An FAQ query $Q(J)$ that is either a SumProd query or a SumSum query. We define a SumSum query to be a query of the form:

$$Q(J) = \bigoplus_{x \in J} \bigoplus_{i=1}^{d} F_i(x_i)$$

where $(R, \oplus, I_0)$ is a commutative monoid over the arbitrary set R with identity I_0. We define a SumProd query to be a query of the form:

$$Q(J) = \bigoplus_{x \in J} \bigotimes_{i=1}^{d} F_i(x_i)$$

where $(R, \oplus, \otimes, I_0, I_1)$ is a commutative semiring over the arbitrary set R with additive identity I_0 and multiplicative identity I_1. In each case, x is a row in the design matrix J, x_i is the entry in column/feature i of x, and each F_i is an arbitrary (easy to compute) function with range R.

- A collection $\mathcal{L} = \{(G_1, L_1), \dots (G_b, L_b)\}$ of additive inequalities, where G_i is a collection $\{g_{i,1}, g_{i,2}, \dots g_{i,d}\}$ of d (easy to compute) functions that map the column domains to the reals, and each L_i is a real number. A row $x \in J$ satisfies the additive inequalities in $\mathcal{L}$ if for all $i \in [1, b]$, it is the case that $\sum_{j=1}^{d} g_{i,j}(x_j) \leq L_i$.

FAQ-AI(b) is a special case of FAQ-AI when the cardinality of $\mathcal{L}$ is at most b.

The output for the FAQ-AI problem is the result of the query on the subset of the design matrix that satisfies the additive inequalities. That is, the output for the FAQ-AI instance with a SumSum query is:

$$Q(\mathcal{L}(J)) = \bigoplus_{x \in \mathcal{L}(J)} \bigoplus_{i=1}^{d} F_i(x_i) \qquad (1)$$

And the output for the FAQ-AI instance with a SumProd query is:

$$Q(\mathcal{L}(J)) = \bigoplus_{x \in \mathcal{L}(J)} \bigotimes_{i=1}^{d} F_i(x_i) \qquad (2)$$

Here $\mathcal{L}(J)$ is the set of $x \in J$ that satisfy the additive inequalities in $\mathcal{L}$. To aid the reader in appreciating these definitions, we now illustrate how some of the SVM related problems in the introduction can be reduced to FAQ-AI(1).

Counting the number of negatively labeled points correctly classified by a linear separator: Here each row x of the design matrix J conceptually consists of a point in $\mathbb{R}^{d-1}$, whose coordinates are specified by the first $d-1$ columns in J, and a label in $\{1, -1\}$ in column d. Let the linear separator be defined by $\beta \in \mathbb{R}^{d-1}$. A negatively labeled point x is correctly classified if $\sum_{i=1}^{d-1} \beta_i x_i \leq 0$. The number of such points can be counted using SumProd query with one additive inequality as follows: $\oplus$ is addition, $\otimes$ is multiplication, $F_i(x_i) = 1$ for all $i \in [d-1]$, $F_d(x_d) = 1$ if $x_d = -1$, and $F_d(x_d) = 0$ otherwise, $g_{1,j}(x_j) = \beta_j x_j$ for $j \in [d-1]$, $g_{1,d}(x_d) = 0$, and $L_1 = 0$.

Finding the minimum distance to the linear separator of a correctly classified negatively labeled point: This distance can be computed using a SumProd query with one additive inequality as follows: $\oplus$ is the binary minimum operator, $\otimes$ is addition, $F_i(x_i) = \beta_i x_i$ for all $i \in [d-1]$, $F_d(x_d) = 1$ if $x_d = -1$, and $F_d(x_d) = 0$ otherwise, $g_{1,j}(x_j) = \beta_j x_j$ for $j \in [d-1]$, $g_{1,d}(x_d) = 0$, and $L_1 = 0$.

2.2 The Inside-Out Algorithm for Acyclic Queries

We give an overview of the well-known algorithm for FAQ, which is FAQ-AI(0), i.e., FAQ with no additive constraints. Here, we assume that the join is acyclic; otherwise we can decompose the join to make it acyclic (see Appendix C).

After giving some preliminary definitions, we explain how to obtain a hypertree decomposition of an acyclic join, and then explain the Inside-Out algorithm from [6] for evaluating a SumProd query $Q(J) = \oplus_{x \in J} \otimes_{i=1}^{d} F_i(x_i)$ over a commutative semiring $(R, \oplus, \otimes, I_0, I_1)$ for acyclic join $J = T_1 \bowtie \cdots \bowtie T_m$. We choose a specific implementation of the algorithm that is suitable for our purpose. A call of Inside-Out may optionally include a root table T_r.

Let C_i denote the set of columns in T_i and let $C = \bigcup_i C_i$. Furthermore, given a set of columns C_i and a tuple x, let $\Pi_{C_i}(x)$ be the projection of x onto C_i which is a tuple that has the values of x in columns C_i.

Definition 1. *A join query $J = T_1 \bowtie \cdots \bowtie T_m$ is acyclic if there exists a tree $G = (V, E)$ such that:*

- *The set of vertices are $V = \{v_1, \dots, v_m\}$ which has one vertex associated with each input table T_i, and*
- *for every column $c \in C$, the set of vertices $\{v_i | c \in C_i\}$ is a connected component of G.*

Note that the tree G used in Definition 1 is a hypertree decomposition of the join J [21, 6]. The definition of the hypertree decomposition for general joins (cyclic or acyclic) can be found in Appendix C.

Algorithm to Compute Hypertree Decomposition:

1. Initialize graph $G = (V, \emptyset)$ where $V = \{v_1, \dots, v_m\}$.
2. Repeat the following steps until $|T| = 1$:
   (a) Find T_i and T_j in T such that every column of T_i is either not in any other table of T or is in T_j. If there exists no T_i and T_j with this property, then the query is cyclic.
   (b) Remove T_i from T and add the edge (v_i, v_j) to G.

We are now ready to describe the inside-out algorithm.

Inside-out Algorithm:

1. Compute the hypertree decomposition $G = (V, E)$ of J.
2. Assign each column c in J to an arbitrary table T_i such that $c \in C_i$. Let A_i denote the columns assigned to T_i in this step.
3. For each table T_i, add a new column/feature Q_i. For all the tuples $x \in T_i$, initialize the value of column Q_i in row x to $Q_i^x = \bigotimes_{j \in A_i} F_j(x_j)$. Note that if $A_i = \emptyset$ then $Q_i^x = I_1$.

4. Repeat until G has only one vertex

   (a) Pick an arbitrary edge (v_i, v_j) in G such that v_i is a leaf and $i \neq r$.

   (b) Let $C_{ij} = C_i \cap C_j$ be the shared columns between T_i and T_j.

   (c) Construct a temporary table T_{ij} that has the columns $C_{ij} \cup \{Q_{ij}\}$.

   (d) If $C_{ij} = \emptyset$, then table T_{ij} only has the column/feature Q_{ij} and one row, and its lone entry is set to $\oplus_{x \in T_i} Q_i^x$. Otherwise, iterate through the y such that there exists an $x \in T_i$ for which $\Pi_{C_i}(x) = y$, and add the row (y, Q_{ij}^y) to table T_{ij} where:

      i. Q_{ij}^y is set to the sum, over all rows $x \in T_i$ such that $C_{ij}(x) = y$, of Q_i^x.

   (e) For all the tuples $(x, Q_j^x) \in T_j$, let $y = \Pi_{C_{ij}}(x)$, and update Q_j^x by

$$Q_j^x \leftarrow Q_j^x \otimes Q_{ij}^y.$$

   If $(y, Q_{ij}^y) \notin T_{ij}$, set $Q_j^x = I_0$.

   (f) Remove vertex v_i and edge (v_i, v_j) from G.

5. At the end, when there is one vertex v_r left in G, return the value

$$\bigoplus_{(x, Q_r^x) \in T_r} Q_r^x$$

When we use the Inside-Out algorithm in context of an approximation algorithm is important that the sum computed in step 4(d)i is computed using a balanced binary tree, so that if k items are being summed, the depth of the expression tree is at most $\lceil \log k \rceil$.

One way to think about step 4 of the algorithm is that it is updating the Q_j values to what they would be if what good old CLRS [11] calls a relaxation in the description of the Bellman-Ford shortest path algorithm was applied to every edge in a particular bipartite graph $G_{i,j}$. In $G_{i,j}$ one side of the vertices are the rows in T_i, and the other side are the rows in T_j, and there a directed edge (x, y) from a vertex/row in T_i to a vertex/row in T_j if they have equal projections onto $C_{i,j}$ – they have the same values in columns $C_{i,j}$. The length P_j^y of edge (x, y) is the original value of Q_j^y before the execution of step 4. A relaxation step on a directed edge (x, y) is then $Q_j^y = (P_j^y \otimes Q_i^x) \oplus Q_j^y$. So the result of step 4 of Inside-Out is the same as relaxing every edge in $G_{i,j}$. However, Inside-Out doesn't explicitly relax every edge; instead, Inside-Out exploits the structure of $G_{i,j}$, by grouping together rows in T_i that have the same projection onto $C_{i,j}$, to be more efficient.

2.3 Approximate Operators

Since we will mostly approximately answer FAQ-AI queries, we need to carefully control the errors accumulated over the query processing. Thus, we define approximate operators as follows.

Definition 2.

- *An operator $\odot$ has bounded error if it is the case that when $x/(1+\delta_1) \leq x' \leq (1+\delta_1)x$ and $y/(1+\delta_2) \leq y' \leq (1+\delta_2)y$ then $(x \odot y)/((1+\delta_1)(1+\delta_2)) \leq x' \odot y' \leq (1+\delta_1)(1+\delta_2)(x \odot y)$.*

- *An operator introduces no error if it is the case that when $x/(1+\delta_1) \leq x' \leq (1+\delta_1)x$ and $y/(1+\delta_2) \leq y' \leq (1+\delta_2)y$ then $(x \odot y)/(1+\max(\delta_1,\delta_2)) \leq x' \odot y' \leq (1+\max(\delta_1,\delta_2))(x \odot y)$.*

- *An operator $\odot$ is repeatable if for any two non-negative integers k and j and any non-negative real δ such that $k/(1+\delta) \leq j \leq (1+\delta)k$, it is the case that for every $x \in R$, $(\bigodot^k x)/(1+\delta) \leq \bigodot^j x \leq (1+\delta) \bigodot^k x$.*

- *An operator $\odot$ is monotone if it is either monotone increasing or monotone decreasing. The operator $\odot$ is monotone increasing if $x \odot y \geq \max(x, y)$. The operator $\odot$ is monotone decreasing if $x \odot y \leq \min(x, y)$.*

3 Hardness Results

We first show that our main problem cannot be solved even approximately with two or more additive constraints.

Theorem 3. *For all $c \geq 1$, it is NP-hard to c-approximate the number of rows in the design matrix (even for a cross product join) that satisfy two (linear) additive inequalities. So it is NP-hard to c-approximate FAQ-AI(2).*

Proof. We reduce from the Partition problem, where the input is a collection $W = \{w_1, w_2, ..., w_m\}$ of positive integers, and the question is whether one can partition W into two parts with equal aggregate sums. From this instance we create m tables, $T_1, T_2, \ldots, T_m$, where each T_i has a single columns and has two rows with entries w_i and $-w_i$. Let J be the cross product of these tables. Note that J has exactly 2^m rows and each row $x \in J$ contains either w_i or $-w_i$ for every i, which can be naturally interpreted as a partitioning that places each item i in one part or the other, depending on the sign of w_i. The two (linear) additive inequalities are $(1, 1, \ldots, 1) \cdot x \geq 0$ and $(-1, -1, \ldots, -1) \cdot x \geq 0$. Then the solution to the Row Counting SumProd query subject to these two constraints is the number of ways to partition W into two parts of equal aggregate sum. □

Unfortunately, the problem is still not solvable in polynomial time even with only one additive constriant.

Theorem 4. *The problem of evaluating a FAQ-AI(1) query is #P-Hard.*

Proof. We prove the #-hardness by a reduction from the $\#P$-hard Knapsack Counting problem. An instance for Knapsack consists of a collection $W = \{w_1, \ldots, w_d\}$ of nonnegative integer weights, and a nonnegative integer weight C. The output should be the number of subsets of W with aggregate weight at most C.

We construct the instance of FAQ-AI as follows. We creating d tables. Each table T_i has one column and two rows, with entries 0 and w_i. Then $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_d$ is the cross product join of the tables. We define β to be the d dimensional vector with 1's in all dimensions, and the additive inequality to $\beta \cdot x \leq C$. Then note that there is then a natural bijection between the rows in J that satisfy this inequality the subsets of W with aggregate weight at most C. □

In light of these hardness results, our goal is to develop approximation schemes for FAQ-AI(1).

4 Algorithm for Inequality Row Counting

The *Inequality Row Counting* problem is a special case of SumProd FAQ-AI(1) in which the SumProd query $Q(\mathcal{L}(J)) = \sum_{x \in J} \prod_{i=1}^{d} 1$ counts the number of rows in the design matrix that satisfy the constraints $\mathcal{L}$, which consists of one additive constraint $\sum_i g_i(x_i) \leq L$, over a join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$. We first present our approximate algorithm for the Inequality Row Counting because it illustrates many of the key ideas that are extendable to SumSum FAQ-AI(1) and SumProd FAQ-AI(1). In subsection 4.1, we design a SumProd query over a dynamic programming semiring that computes $Q(\mathcal{L}(J))$ exactly in exponential time. Then in subsection 4.2 we explain how to apply standard sketching techniques to obtain a RAS.

4.1 An Exact Algorithm

We first define a commutative semiring $(S, \sqcup, \sqcap, E_0, E_1)$ as follows:

- The elements of the base set S are finite multi-sets of real numbers. Let $\#A(e)$ denote the frequency of the real value e in the multi-set A and let it be 0 if e is not in A. Thus one can also think of A as a set of pairs of the form $(e, \#A(e))$.
- The additive identity E_0 is the empty set $\emptyset$.
- The addition operator $\sqcup$ is the union of the two multi-sets; that is $A = B \sqcup C$ if and only if for all real values e, $\#A(e) = \#B(e) + \#C(e)$.
- The multiplicative identity is $E_1 = \{0\}$.
- The multiplication operator $\sqcap$ contains the pairwise sums from the two input multi-sets; that is, $A = B \sqcap C$ if and only if for all real values e, $\#A(e) = \sum_{i \in \mathbb{R}} (\#B(e-i) \cdot \#C(i))$. Note that this summation is well-defined because there is a finite number of values for i such that $B(e-i)$ and $C(i)$ are nonzero.

Lemma 5. $(S, \sqcup, \sqcap, E_0, E_1)$ *is a commutative semiring.*

Lemma 6. *The SumProd query* $\widehat{Q}(J) = \sqcup_{x \in J} \sqcap_{i=1}^{d} F_i(x_i)$*, where* $F_i(x_i) = \{g_i(x_i)\}$*, evaluates to the multiset* $\{\sum_i g_i(x_i) \mid x \in J\}$ *the aggregate of the* g_i *functions over the rows of* J*.*

Proof. Based on the definition of $\sqcap$ we have,

$$\sqcap_{i=1}^{d} F_i(x_i) = \sqcap_{i=1}^{d} \{g_i(x_i)\} = \left\{ \sum_{i=1}^{d} g_i(x_i) \right\}$$

Then we can conclude:

$$\begin{aligned} \sqcup_{x \in J} \sqcap_{i=1}^{d} F_i(x_i) &= \sqcup_{x \in J} \left\{ \sum_{i=1}^{d} g_i(x_i) \right\} \\ &= \left\{ \sum_{i=1}^{d} g_i(x_i) \mid x \in J \right\} \end{aligned}$$

□

Thus the inequality row count is the number of elements in the multiset returned by $\widehat{Q}(J)$ that are at most L.

4.2 Applying Sketching

For a multiset A, let $\triangle A(t)$ denote the number of elements in A that are less than or equal to t. Then the ϵ-sketch $\mathbb{S}_\epsilon(A)$ of a multiset A is a multiset formed in the following manner: For each integer $k \in [1, \lfloor \log_{1+\epsilon} |A| \rfloor]$ there are $\lfloor (1+\epsilon)^k \rfloor - \lfloor (1+\epsilon)^{k-1} \rfloor$ copies of the $\lfloor (1+\epsilon)^k \rfloor$ smallest element $x_k \in A$; that is, $x_k = \triangle A(\lfloor (1+\epsilon)^k \rfloor)$. Note that $|\mathbb{S}_\epsilon(A)|$ may be less that $|A|$ as the maximum value of k is $\lfloor \log_{1+\epsilon} |A| \rfloor$. We will show in Lemma 7 that sketching preserves $\triangle A(t)$ within $(1+\epsilon)$ factor.

Lemma 7. *For all* $t \in \mathbb{R}$*, we have* $(1-\epsilon)\triangle A(t) \leq \triangle \mathbb{S}_\epsilon(A)(t) \leq \triangle A(t)$*.*

Then our algorithm runs the Inside-Out algorithm, with the operation $\sqcup$ replaced by an operation $\bigcirc$, defined by $A \bigcirc B = \mathbb{S}_\alpha(A \sqcup B)$, and with the operation $\sqcap$ replaced by an operation $\odot$, defined by $A \odot B = \mathbb{S}_\alpha(A \sqcap B)$, where $\alpha = \Theta(\frac{\epsilon}{m^2 \log(n)})$. That is, the operations $\bigcirc$ and $\odot$ are the sketched versions of $\sqcup$ and $\sqcap$. That is, Inside-Out is run on the query $\tilde{Q}(J) = \bigcirc_{x \in J} \odot_{i=1}^{d} F_i(x_i)$, where $F_i(x_i) = \{g_i(x_i)\}$.

Because $\bigcirc$ and $\odot$ do not necessarily form a semiring, Inside-Outside may not return $\tilde{Q}(J)$. However, Lemma 8 bounds the error introduced by each application of $\bigcirc$ and $\odot$. This makes it possible in Theorem 9 to bound the error of Inside-Out's output.

Lemma 8. *Let $A' = \mathbb{S}_\beta(A)$, $B' = \mathbb{S}_\gamma(B)$, $C = A \sqcup B$, $C' = A' \sqcup B'$, $D = A \sqcap B$, and $D' = A' \sqcap B'$. For all $t \in \mathbb{R}$, we have:*

1. $(1 - \max(\beta, \gamma))\triangle C(t) \leq \triangle C'(t) \leq \triangle C(t)$
2. $(1 - \beta - \gamma)\triangle D(t) \leq \triangle D'(t) \leq \triangle D(t)$

Theorem 9. *Our algorithm achieves an $(1 + \epsilon)$-approximation to the Row Count Inequality query $Q(\mathcal{L}(J))$ in time $O(\frac{1}{\epsilon^2}(m^3 \log^2(n))^2(d^2 mn^h \log(n)))$.*

Proof. We first consider the approximation ratio. Inside-Out on the query $\tilde{Q}(J)$ performs the same semiring operations as does on the query $\widehat{Q}(J)$, but it additionally applies the α-Sketching operation over each partial results, meaning the algorithm applies α-Sketching after steps 4d,4e, and 5. Lets look at each iteration of applying steps 4d and 4e. Each value produced in the steps 4d and 5 is the result of applying $\bigcirc$ over at most n^m different values (for acyclic queries it is at most n). Using Lemma 8 and the fact that the algorithm applies $\bigcirc$ first on each pair and then recursively on each pair of the results, the total accumulated error produced by each execution of steps 4d and 5 is $m \log(n)\alpha$. Then, since the steps 4d and 4e will be applied once for each table, and 4e accumulates the errors produced for all the tables, the result of the query will be $(m^2 \log(n) + m)\alpha$-Sketch of $\widehat{Q}(J)$.

We now turn to bounding the running time of our algorithm. The time complexity of Inside-Out is $O(md^2 n^h \log n)$ when the summation and product operators take a constant time [6]. Since each multi-set in a partial result has at most m^n members in it, an α-sketch of the partial results will have at most $O(\frac{m \log n}{\alpha})$ values, and we compute each of $\bigcirc$ and $\odot$ in time $O\left(\left(\frac{m \log n}{\alpha}\right)^2\right)$. Therefore our algorithm runs in time $O(\frac{1}{\epsilon^2}(m^3 \log^2(n))^2(d^2 mn^h \log(n)))$. □

5 SumSum FAQ-AI(1)

In this subsection we prove Theorem 10, that there is a RAS for SumSum FAQ-AI(1) queries covered by the theorem.

Theorem 10. *There is a RAS to compute a $(1 \pm \epsilon)$ approximation of a SumSum FAQ-AI(1) query over a commutative monoid $(R, \oplus, I_0)$ if:*

- *The domain R is a subset of reals $\mathbb{R}$.*
- *The operators $\oplus$ can be computed in polynomial time.*
- *$\oplus$ introduces no error.*
- *$\oplus$ is repeatable.*

Our algorithm for SumSum queries uses our approximation algorithm for Inequality Row Counting. Consider the SumSum $Q(\mathcal{L}(J)) = \oplus_{x \in \mathcal{L}(J)} \oplus_{i=1}^{d} F_i(x_i)$, where $\mathcal{L}$ consists of one additive constraint $\sum_i g_i(x_i) \leq L$, over a join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$.

SumSum Algorithm: For each table T_j, we run our Inside-Out on the Inequality Row Counting query $\tilde{Q}(J)$, with the root table being T_j, and let $\tilde{T}_j$ be the resulting table just before step 5 of Inside-Out is executed. From the resulting tables, one can compute, for every column $i \in [d]$ and for each possible value of x_i for $x \in J$, a $(1+\epsilon)$-approximation $U(x_i)$ to the number of rows in the design matrix that contain value x_i in column i, by aggregating over the row counts in any table $\tilde{T}_j$ that contains column i. Then one can evaluate $Q(\mathcal{L}(J))$ by

$$\bigoplus_{i=1}^{d} \bigoplus_{x_i \in D(i)} \bigoplus_{j=1}^{U(x_i)} F_i(x_i)$$

where $D(i)$ is the domain of column i.

Note that $\oplus$ operator is assumed to be repeatable, meaning if we have a $(1 \pm \epsilon)$ approximation of $U(x_i)$ then the approximation error of $\bigoplus_{j=1}^{U(x_i)} F_i(x_i)$ is also $(1 \pm \epsilon)$. Therefore, our SumSum algorithm is a $(1 + \epsilon)$-approximation algorithm because the only error introduced is by our Inequality Row Counting algorithm. The running time is $O(\frac{1}{\epsilon^2}(m^3 \log^2(n))^2(d^2 m^2 n^h \log(n)))$ because we run m Inequality Row Counting algorithm m times.

6 SumProd FAQ-AI(1)

In this section we prove Theorem 11, that there is a RAS for SumProd FAQ-AI(1) queries covered by the theorem.

Theorem 11. *There is a RAS to compute a SumProd FAQ-AI(1) query over a commutative semiring $(R, \oplus, \otimes, I_0, I_1)$ if:*

- *The domain R is $\mathbb{R}^+ \cup \{I_0\} \cup \{I_1\}$.*
- $I_0, I_1 \in \mathbb{R}^+ \cup \{+\infty\} \cup \{-\infty\}$
- *The operators $\oplus$ and $\otimes$ can be computed in polynomial time.*
- *$\oplus$ introduces no error.*
- *$\otimes$ has bounded error.*
- *$\oplus$ is monotone. An operator $\odot$ is monotone if it is either monotone increasing or monotone decreasing.*

- *The log of the aspect ratio of the query is polynomially bounded. The aspect ratio is the ratio of the maximum, over every possible submatrix of the design matrix, of the value of the query on that submatrix, to the minimum, over every possible submatrix of the design matrix, of the value of the query on that submatrix.*

Our RAS for such queries generalizes our RAS for Inequality Row Counting. Consider the SumProd query $Q(\mathcal{L}(J)) = \oplus_{x \in \mathcal{L}(J)} \otimes_{i=1}^{d} F_i(x_i)$. where $\mathcal{L}$ consists of the single additive constraint $\sum_{i=1}^{d} g_i(x_i) \leq L$. We again first give an exact algorithm that can viewed as a reduction to a SumProd query over a dynamic programming semiring, and then apply sketching.

6.1 An Exact Algorithm

We first define a structured commutative semiring $(S, \sqcup, \sqcap, E_0, E_1)$ derived from the $(R, \oplus, \otimes, I_0, I_1)$ as follows:

- The base set S are finite subsets A of $\mathbb{R} \times (R - \{I_0\})$ with the property that $(e, v) \in A$ and $(e, u) \in A$ implies $v = u$; so there is only one tuple in A of the form $(e, *)$. One can can interpret the value of v in a tuple $(e, v) \in A$ as a (perhaps fractional) multiplicity of e.
- The additive identity E_0 is the empty set $\emptyset$.
- The multiplicative identity E_1 is $\{(0, I_1)\}$.
- For all $e \in \mathbb{R}$, define $\#A(e)$ to be v if $(e, v) \in A$ and I_0 otherwise.
- The addition operator $\sqcup$ is defined by $A \sqcup B = C$ if and only if for all $e \in \mathbb{R}$, it is the case that $\#C(e) = \#A(e) \oplus \#B(e)$.
- The multiplication operator $\sqcap$ is defined by $A \sqcap B = C$ if and only if for all $e \in \mathbb{R}$, it is the case that $\#C(e) = \bigoplus_{i \in \mathbb{R}} \#A(e - i) \otimes \#B(i)$.

Lemma 12. *If $(R, \oplus, \otimes, I_0, I_1)$, is a commutative semiring then $(S, \sqcup, \sqcap, E_0, E_1)$ is a commutative semiring.*

For each column $i \in [d]$, we define the function $\mathcal{F}_i$ to be $\{(g_i(x_i), F_i(x_i))\}$ if $F_i(x_i) \neq I_0$ and the empty set otherwise. Our algorithm for computing $Q(\mathcal{L}(J))$ runs the Inside-Out algorithm on the SumProd query:

$$\widehat{Q} = \sqcup_{x \in J} \sqcap_{i=1}^{d} \mathcal{F}_i(x_i)$$

and returns $\bigoplus_{e \leq L} \#\widehat{Q}(e)$.

Lemma 13. *This algorithm correctly computes $Q(\mathcal{L}(J))$.*

6.2 Applying Sketching

For a set $A \in S$ define $\triangle A(\ell)$ to be $\oplus_{e \leq \ell} \#A(e)$. Note that $\triangle A(\ell)$ will be monotonically increasing if $\oplus$ is monotonically increasing, and it will be monotonically decreasing if $\oplus$ is monotonically decreasing.

Conceptually an ϵ-sketch $\mathbb{S}_\epsilon(A)$ of an element $A \in S$ rounds all multiplicities up to an integer power of $(1+\epsilon)$. Formally the ϵ-sketch $\mathbb{S}_\epsilon(A)$ of A is the element A' of S satisfying

$$\#A'(e) = \begin{cases} \bigoplus_{L_k < e \leq U_k} \#A(e) & \text{if } \exists k \; e = U_k \\ I_0 & \text{otherwise} \end{cases}$$

where

$$L_0 = \min\{e \in \mathbb{R} | \; \triangle A(e) \leq 0\}$$

and for $k \neq 0$

$$L_k = \min\{e \in \mathbb{R} | \; \rho(1+\epsilon)^{k-1} \leq \triangle A(e) \leq \rho(1+\epsilon)^k\}$$

and where

$$U_0 = \max\{e \in \mathbb{R} | \; \triangle A(e) \leq 0\}$$

and for $k \neq 0$

$$U_k = \max\{e \in \mathbb{R} | \; \rho(1+\epsilon)^{k-1} \leq \triangle A(e) \leq \rho(1+\epsilon)^k\}$$

where $\rho = \min\{\#A(e) \mid \#e \in \mathbb{R} \text{ and } \#A(e) > 0\}$. For the special case that $\#A(e) \leq 0$ for all $e \in \mathbb{R}$, we only have L_0 and U_0. Note that the only elements of R that can be zero or negative are I_0 and I_1; therefore, in this special case, $\#A(e)$ for all the elements e is either I_0 or I_1.

Lemma 14. *For all $A \in S$, for all $e \in \mathbb{R}^+$, if $A' = \mathbb{S}_\epsilon(A)$ then*

$$\triangle A(e)/(1+\epsilon) \leq \triangle A'(e) \leq (1+\epsilon)\triangle A(e)$$

Then our algorithm runs the Inside-Out algorithm, with the operation $\sqcup$ replaced by an operation $\bigcirc$, defined by $A \bigcirc B = \mathbb{S}_\alpha(A \sqcup B)$, and with the operation $\sqcap$ replaced by an operation $\odot$, defined by $A \odot B = \mathbb{S}_\alpha(A \sqcap B)$, where $\alpha = \frac{\epsilon}{m^2 \log(n) + m}$. That is, the operations $\bigcirc$ and $\odot$ are the sketched versions of $\sqcup$ and $\sqcap$. Our algorithm returns $\triangle A(L) = \oplus_{e \leq L} \#A(e)$.

Because $\bigcirc$ and $\odot$ do not necessarily form a semiring, Inside-Outside may not return $\bigcirc_{x \in J} \odot_{i=1}^{m} F_i(x_i)$. However, Lemma 15 bounds the error introduced by each application of $\bigcirc$ and $\odot$. This makes it possible in Theorem 11 to bound the error of Inside-Out's output.

Lemma 15. *Let $A' = \mathbb{S}_\beta(A)$, $B' = \mathbb{S}_\gamma(B)$, $C = A \sqcup B$, $C' = A' \sqcup B'$, $D = A \sqcap B$, and $D' = A' \sqcap B'$. Then, for all $e \in \mathbb{R}$ we have:*

1. $\frac{\triangle C(e)}{1+\max(\beta,\gamma)} \leq \triangle C'(e) \leq (1 + \max(\beta, \gamma))\triangle C(e)$
2. $\frac{\triangle D(e)}{(1+\beta)(1+\gamma)} \leq \triangle D'(e) \leq (1+\beta)(1+\gamma)\triangle D(e)$

Now we can prove the existence of an algorithm for approximating SumProd FAQ-AI(1) queries.

Proof of Theorem 11. We first consider the approximation ratio. Inside-Out on the query $\widetilde{Q}(J)$ performs the same semiring operations as it does on the query $\widehat{Q}(J)$, but it additionally applies the α-Sketching operation over each partial result, meaning the algorithm applies α-Sketching after steps 4d,4e, and 5. Lets look at each iteration of applying steps 4d and 4e. Each value produced in the steps 4d and 5 is the result of applying $\bigcirc$ over at most n^m different values (for acyclic queries it is at most n). Using Lemma 8 and the fact that the algorithm applies $\bigcirc$ first on each pair and then recursively on each pair of the results, the total accumulated error produced by each execution of steps 4d and 5 is $m\log(n)\alpha$. Then, since the steps 4d and 4e will be applied once for each table, and 4e accumulates the errors produced for all the tables, the result of the query will be $(m^2\log(n)+m)\alpha$-Sketch of $\widehat{Q}(J)$.

We now turn to bounding the running time of our algorithm. The time complexity of Inside-Out is $O(md^2n^h\log n)$ when the summation and product operators take a constant time [6]. The size of each partial result set $A \in S$, after applying α-sketching, will depend on the smallest positive value of $\triangle A(e)$ and the largest value of $\triangle A(e)$. Let β and γ be the minimum and maximum positive real value of SumProd query over all possible sub-matrices of the design matrix, the smallest and largest value of $\triangle A(e)$ for all partial results A would be β and γ respectively; therefore, the size of the partial results after applying α-Sketching is at most $O(\frac{\log(\gamma/\beta)}{\alpha})$. As a result, we compute each of $\bigcirc$ and $\odot$ in time $O\left(\left(\frac{\log(\gamma/\beta)}{\alpha}\right)^2\right)$. Therefore our algorithm runs in time $O(\frac{1}{\epsilon^2}(m^2\log(n)\log(\frac{\gamma}{\beta}))^2(d^2mn^h\log(n)))$ and the claim follows by the assumption that the log of the aspect ratio, $\log(\frac{\gamma}{\beta})$, is polynomially bounded. $\square$

7 Example Applications of Our General Results

Inequality Row Counting: Some example problems fow which we can use our Inequality Row Counting to obtain a RAS in a straightforward manner:

- Counting the number of points on one side of a hyperplane, say the points x satisfy $\beta \cdot x \leq L$.
- Counting the number of points within a hypersphere of radius r centered at a point y. The additive constraint is $\sum_{i=1}^{d}(x_i - y_i)^2 \leq r^2$.
- Counting the number of points in an axis-parallel ellispoid, say the points x such that $\sum_{i=1}^{d}\frac{x_i^2}{\alpha_i^2}$ for some d dimensional vector α.

SumSum FAQ-AI(1) Queries Some examples of problems that can be reduced to SumSum FAQ-AI(1) queries and an application of Theorem 10 gives a RAS:

- Sum of 1-norm distances from a point y of points on one side of a hyperplane. The SumSum query is $\sum_{x\in J}\sum_{i=1}^{d}|x_i - y_i|$. One can easily verify the addition introduces no error and is repeatable.
- Sum of 2-norm squared of points in an axis-parallel ellipsoid. The SumSum query is $\sum_{x\in J}\sum_{i=1}^{d}x_i^2$.
- Number of nonzero entries of points on one side of a hyperplane. The SumSum query is $\sum_{x\in J}\sum_{i=1}^{d}\mathbb{1}_{x_i\neq 0}$.

SumProd FAQ-AI(1) Queries Some examples of problems that can be reduced to SumProd FAQ-AI(1) queries and an application of Theorem 11 gives a RAS:

- Finding the minimum 1-norm of any point in a hypersphere H of radius r centered at a point y. The SumProd query is $\min_{x\in J}\sum_{i=1}^{d}|x_i|$. Note $(\mathbb{R}^+ \cup \{0\} \cup \{+\infty\}, \min, +, +\infty, 0)$ is a commutative semiring. The multiplication operator in this semiring, which is addition, has bounded error. The addition operator, which is minimum, introduces no error and is monotone. The aspect ratio is at most $(\max_{x\in J}\sum_{i=1}^{d}|x_i|)/(\min_{x\in J}\min_{i\in[d]|x_i\neq 0}|x_i|)$, and thus the log of the aspect ratio is polynomially bounded.
- Finding the point on the specified side of a hyperplane H that has the maximum 2-norm distance from a point y. The SumProd query is $\max_{x\in J}\sum_{i=1}^{d}(y_i - x_i)^2$. Note that this computes the point with the maximum 2-norm squared distance. One can not directly write a SumProd query to compute the point with the 2-norm distance; We need to appeal to the fact that the closest point is the same under both distance metrics. Note $(\mathbb{R}^+ \cup \{0\} \cup \{-\infty\}, \max, +, -\infty, 0)$ is a commutative semiring. The multiplication operator in this semiring, which is addition, has bounded error. The addition operator, which is maximum, introduces no error and is monotone. The aspect ratio is at most $(\max_{x\in J}\sum_{i=1}^{d}|x_i|)/(\min_{x\in J}\min_{i\in[d]|x_i\neq 0}|x_i|)$, and thus the log of the aspect ratio is polynomially bounded.

Snake Eyes: Some examples of problems for which our results apparently do not apply:

- Finding the minimum distance of any point on a specified side of a specified hyperplane H to H. So say the problem is to find a point x where $\beta \cdot x \geq L$ and $x \cdot \beta$. The natural SumProd query is $\min_{x\in J}\sum_{i=1}^{d}x_i\beta_i$. Note that some of the $x_i\beta_i$

terms maybe be negative, so this doesn't fulfill the condition that the domain has to be over the positive reals. And this appears to be a non-trivial issue basically because having good approximations of s and t does not in general allow one to compute a good approximation of $s - t$. We have been call this the subtraction problem. Using a variation of the proof of Theorem 3 one can show that approximating this query to within an $O(1)$ factor is NP-hard.

- Sum of entries of the points lying on one side of a hyperplane. The natural SumSum query is $\sum_{x \in J} \sum_{i=1}^{d} x_i$. Again as some of the x_i terms may be be negative, we run into the subtraction problem again.

- Aggregate 2-norms of the rows in the design matrix. The natural query is $\sum_{x \in J} \left(\sum_{i=1}^{d} x_i^2 \right)^{1/2}$, which is neither a SumSum or a SumProd query.

References

[1] Kaggle machine learning and data science survey. https://www.kaggle.com/kaggle/kaggle-survey-2018, 2018.

[2] M. Abo Khamis, R. R. Curtin, B. Moseley, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. On functional aggregate queries with additive inequalities. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 414–431. ACM, 2019.

[3] M. Abo Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Ac/dc: in-database learning thunderstruck. In *Second Workshop on Data Management for End-To-End Machine Learning*, page 8. ACM, 2018.

[4] M. Abo Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. In-database learning with sparse tensors. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 325–340, 2018.

[5] M. Abo Khamis, H. Q. Ngo, D. Olteanu, and D. Suciu. Boolean tensor decomposition for conjunctive queries with negation. In *ICDT*, pages 21:1–21:19, 2019.

[6] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: Questions asked frequently. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 13–28, 2016.

[7] I. Adler. *Width functions for hypertree decompositions.* 2006. Ph.D. Dissertation, Albert-Ludwigs-Universität Freiburg. 2006.

[8] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trans. Inf. Theor.*, 46(2):325–343, 2006.

[9] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 739–748. IEEE, 2008.

[10] A. Burkov. *The Hundred Page Machine Learning Book.*

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* 2009.

[12] R. R. Curtin, B. Moseley, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Rk-means: Fast clustering for relational data. *CoRR*, abs/1910.04939, 2019.

[13] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, 1996.

[14] M. Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006.

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

[16] A. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, Jan. 1988.

[17] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[18] P. Koutris, T. Milo, S. Roy, and D. Suciu. Answering conjunctive queries with inequalities. *Theory of Computing Systems*, 61(1):2–30, Jul 2017.

[19] A. Kumar, J. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *ACM SIGMOD International Conference on Management of Data*, pages 1969–1984, 2015.

[20] A. Kumar, J. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *International Conference on Management of Data*, pages 19–34, 2016.

[21] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM (JACM)*, 60(6):42, 2013.

[22] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms: [extended abstract]. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 37–48, 2012.

[23] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. In *SIGMOD Rec.*, pages 5–16, 2013.

[24] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18. ACM, 2016.

[25] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, New York, NY, USA, 2014.

[26] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

A Omitted Proofs from Section 4

A.1 Proof of Lemma 5

Proof. To prove the lemma, we prove the following claims in the order in which they appear.

1. $A \sqcup B = B \sqcup A$
2. $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$
3. $A \sqcup E_0 = A$
4. $A \sqcap B = B \sqcap A$
5. $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$
6. $A \sqcap E_0 = E_0$
7. $A \sqcap E_1 = A$
8. $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$

First we show $\sqcup$ is commutative and associative and $A \sqcup E_0 = A$. By definition of $\sqcup$, $C = A \sqcup B$ if and only if for all $e \in C$ we have $\#C(e) = \#A(e) + \#B(e)$. Since summation is commutative and associative, $\sqcup$ would be commutative and associative as well. Also note that if $B = E_0 = \emptyset$ then $\#B(e) = 0$ for all values of e and as a result $\#C(e) = \#A(e)$ which means $C = A$.

Now we can show that $\sqcap$ is commutative and associative. By definition of $\sqcap$, $C = A \sqcap B$ if and only if for all values of e, $\#C(e) = \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i))$, since we are taking the summation over all values:

$$\begin{aligned}\#C(e) &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i))\\ &= \sum_{i\in\mathbb{R}}(\#A(i)\cdot\#B(e-i))\end{aligned}$$

The last line is due to the definition of $B \sqcap A$, which means $\sqcap$ is commutative.

To show claim (5), let $D = A \sqcap (B \sqcap C)$ and $D' = (A \sqcap B) \sqcap C$:

$$\begin{aligned}\#D(e) &= \sum_{i\in\mathbb{R}}\#A(e-i)\cdot\Big(\sum_{j\in\mathbb{R}}\#B(i-j)\cdot\#C(j)\Big)\\ &= \sum_{i,j\in\mathbb{R}}\#A(e-i)\cdot\#B(i-j)\cdot\#C(j)\end{aligned}$$

By setting $i' = e-j$ and $j' = e-i$, we obtain:

$$\begin{aligned}\#D(e) &= \sum_{i',j'\in\mathbb{R}}\#A(j')\cdot\#B(i'-j')\cdot\#C(e-i')\\ &= \sum_{i'\in\mathbb{R}}\Big(\sum_{j'\in\mathbb{R}}\#A(j')\cdot\#B(i'-j')\Big)\cdot\#C(e-i')\\ &= \#D'(e),\end{aligned}$$

which means $\sqcap$ is associative, as desired.

Now we prove $A \sqcap E_0 = E_0$ and $A \sqcap E_1 = A$. The claim (6) is easy to show since for all e, $\#E_0(e) = 0$ then for all real values e $\sum_{i\in\mathrm{dist}(A)}(\#A(i)\cdot\#E_0(e-i)) = 0$; therefore, $A \sqcap E_0 = E_0$. For claim (7), we have $\sum_{i\in\mathrm{dist}(E_1)}(\#E_1(i)\cdot\#A(e-i)) = (\#E_1(0)\cdot\#A(e)) = \#A(e)$; therefore, $A \sqcap E_1 = A$.

At the end all we need to show is the distributive law which means we need to show $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$. Let $D = A \sqcap (B \sqcup C)$ and $D' = (A \sqcap B) \sqcup (A \sqcap C)$. We have,

$$\begin{aligned}&\#D(e)\\ &= \sum_{i\in\mathbb{R}}\#A(e-i)\cdot(\#B(i)+\#C(i))\\ &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i)) + (\#A(e-i)\cdot\#C(i))\\ &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i)) + \sum_{j\in\mathbb{R}}(\#A(e-j)\cdot\#C(j))\\ &= \#D'(e)\end{aligned}$$

□

A.2 Proof of Lemma 7

Proof. Let $A' = \mathbb{S}_\epsilon(A)$. Note that since we are always rounding the weights up, every item in A that is larger than t will be larger in A' as well. Therefore, $\triangle A'(t) \leq \triangle A(t)$. We now show the lower bound. Recall that in the sketch, every item in the sorted array A with an index in the interval $((1+\epsilon)^i, (1+\epsilon)^{i+1}]$ (or equivalently $((1+\epsilon)^i, \lfloor(1+\epsilon)^{i+1}\rfloor])$ will be rounded to $A[\lfloor(1+\epsilon)^{i+1}\rfloor]$. Let i be the integer such that $(1+\epsilon)^i < \triangle A(t) \leq (1+\epsilon)^{i+1}$, then the only items that are smaller or equal to t in A and are rounded to have a weight greater than t in A' are the ones with index between $(1+\epsilon)^i$ and $j = \triangle A(t)$. Therefore,

$$\begin{aligned}\triangle A(t) - \triangle A'(t) \leq& j - (1+\epsilon)^i \leq (1+\epsilon)^{i+1} - (1+\epsilon)^i\\ =&\epsilon(1+\epsilon)^i \leq \epsilon\triangle A(t),\end{aligned}$$

which shows the lower bound of $\#\triangle A(t)$ as claimed. □

A.3 Proof of Lemma 8

Proof. By the definition of $\sqcup$, we know $\#C'(t) = \#A'(t) + \#B'(t)$; thus we have:

$$\begin{aligned}\triangle C'(t) &= \sum_{\tau \le t} \#C'(\tau) = \sum_{\tau \le t} \#A'(\tau) + \sum_{\tau \le t} \#B'(\tau)\\ &= \triangle A'(t) + \triangle B'(t)\end{aligned}$$

Similarly, we have $\triangle C(t) = \triangle A(t) + \triangle B(t)$. Then by Lemma 7 we immediately have the first claim.

Let $D'' = A \sqcap B'$, Based on the definition of $\sqcap$ we have:

$$\begin{aligned}\triangle D''(t) &= \sum_{\tau \le t} \#D''(t) = \sum_{\tau \le t} \sum_{v \in \mathbb{R}} (\#A(v) \cdot \#B'(\tau - v))\\ &= \sum_{v \in \mathbb{R}} \sum_{\tau \le t} (\#A(v) \cdot \#B'(\tau - v))\\ &= \sum_{v \in \mathbb{R}} (\#A(v) \cdot \triangle B'(t - v))\end{aligned}$$

Therefore, using Lemma 7 we have: $(1-\gamma)\triangle D(t) \le \triangle D''(t) \le (1+\gamma)\triangle D(t)$. We can similarly replace A to A' in D'' and get $(1-\beta)\triangle D''(t) \le \triangle D'(t) \le (1+\beta)\triangle D''(t)$ which proves the second claim. □

B Omitted Proofs from Section 6

B.1 Proof of Lemma 12

Proof. we prove the following claims respectively:

1. $A \sqcup B = B \sqcup A$
2. $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$
3. $A \sqcup E_0 = A$
4. $A \sqcap B = B \sqcap A$
5. $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$
6. $A \sqcap E_0 = E_0$
7. $A \sqcap E_1 = A$
8. $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$

Based on the definition, $C = A \sqcup B$ if and only if $\#C(e) = \#A(e) \oplus \#B(e)$; since $\oplus$ is commutative and associative, $\sqcup$ will be commutative and associative as well. Furthermore, $\#A(e) \oplus \#E_0(e) = \#A(e) \oplus I_0 = \#A(e)$; therefore, $A \sqcup E_0 = A$.

Let $C = A \sqcap B$, using the commutative property of $\otimes$ and change of variables, we can prove the fourth claim:

$$\begin{aligned}\#C(e) &= \bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes \#B(i)\\ &= \bigoplus_{i \in \mathbb{R}} \#B(i) \otimes \#A(e-i)\\ &= \bigoplus_{j \in \mathbb{R}} \#B(e-j) \otimes \#A(j)\end{aligned}$$

Similarly, using change of variables $j' = i - j$ and $i' = j$, and semiring properties of the $\otimes$ and $\oplus$, we have:

$$\begin{aligned}&\bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes (\bigoplus_{j \in \mathbb{R}} \#B(i-j) \otimes \#C(j))\\ =&\bigoplus_{i \in \mathbb{R}} \bigoplus_{j \in \mathbb{R}} (\#A(e-i) \otimes \#B(i-j)) \otimes \#C(j)\\ =&\bigoplus_{i' \in \mathbb{R}} (\bigoplus_{j' \in \mathbb{R}} \#A(e-i'-j') \otimes \#B(j')) \otimes \#C(i')\end{aligned}$$

Therefore, $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$.

The claim $A \sqcap E_0 = E_0$ can be proved by the fact that $\#E_0(i) = I_0$ for all the elements and $\#A(e-i) \otimes I_0 = I_0$. Also we have $\bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes \#E_1(i) = \#A(e)$ because, $\#E_1(i) = I_0$ for all nonzero values of i and it is I_1 for $e = 0$; therefore, $A \sqcap E_1 = A$.

Let $D = A \sqcap (B \sqcup C)$, the last claim can be proved by the following:

$$\begin{aligned}&\#D(e)\\ =&\bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes (\#B(i) \oplus \#C(i))\\ =&\bigoplus_{i \in \mathbb{R}} \big((\#A(e-i) \otimes \#B(i)) \oplus (\#A(e-i) \otimes \#C(i))\big)\\ =&\big(\bigoplus_{i \in \mathbb{R}} (\#A(e-i) \otimes \#B(i))\big)\\ &\oplus \big(\bigoplus_{i \in \mathbb{R}} (\#A(e-i) \otimes \#C(i))\big)\end{aligned}$$

where the last line is the definition of $(A \sqcap B) \sqcup (A \sqcap C)$. □

B.2 Proof of Lemma 13

Proof. We can rewrite the generated FAQ as follow:

$$\begin{aligned}\hat{Q} &= \sqcup_{x \in J} \sqcap_{i=1}^{d} \mathcal{F}_i(x_i)\\ &= \sqcup_{x \in J} \sqcap_{i=1}^{d} \{(g_i(x_i), F_i(x_i))\}\\ &= \sqcup_{x \in J} \left\{ \left(\sum_{i=1}^{d} g_i(x_i), \bigotimes_{i=1}^{d} F_i(x_i) \right) \right\}\end{aligned}$$

Then the operator $\sqcup$ returns a set of pairs (e, v) such that for each value e, $v = \#\hat{Q}(e)$ is the aggregation using $\oplus$ operator over the rows of J where $\sum_{i=1}^{d} g_i(x_i) = e$. More formally,

$$\#\hat{Q}(e) = \bigoplus_{x \in J, \sum_i g_i(x_i) = e} \left(\bigotimes_{i=1}^{d} F_i(x_i) \right)$$

Therefore, the value returned by the algorithm is

$$\begin{aligned}\bigoplus_{e \le L} \#\hat{Q}(e) &= \bigoplus_{e \le L} \bigoplus_{x \in J, \sum_i g_i(x_i) = e} \left(\bigotimes_{i=1}^{d} F_i(x_i) \right)\\ &= \bigoplus_{x \in \mathcal{L}(J)} \bigotimes_{i=1}^{d} F_i(x_i)\end{aligned}$$

□

B.3 Proof of Lemma 14

Proof. Since $\triangle A(e)$ is monotone, the intervals $[L_k, U_k]$ do not have any overlap except over the points L_k and U_k, and if the $\triangle A(e)$ is monotonically increasing, then $L_k = U_{k+1}$; and if $\triangle A(e)$ is monotonically decreasing, then $L_k = U_{k-1}$.

For any integer j we have:

$$\begin{aligned}\triangle A(U_j) = \bigoplus_{i \le U_j} \#A(i) = \bigoplus_{k \le j} \bigoplus_{L_k < i \le U_k} \#A(i) = \bigoplus_{k \le j} \#A'(U_k)\\ = \bigoplus_{i \le U_j} \#A'(i) = \triangle A'(U_j)\end{aligned} \tag{3}$$

Now, first we assume $\triangle A(e)$ is monotonically increasing and prove the lemma. After that, we do the same for the monotonically decreasing case. Given a real value e, let k be the integer such that $e \in (L_k, U_k]$. Then using the definition of U_k and Equality (3) we have:

$$\begin{aligned}&\triangle A(e)/(1+\epsilon)\\ \le\ &\triangle A(U_k)/(1+\epsilon) = \triangle A(U_{k-1}) = \triangle A'(U_{k-1})\\ \le\ &\triangle A'(e) \le \triangle A'(U_k) = \triangle A(U_k)\\ =\ &(1+\epsilon)\triangle A(U_{k-1}) \le (1+\epsilon)\triangle A(e)\end{aligned}$$

Note that in the above inequalities, for the special case of $k = 0$, we can use L_k instead of U_{k-1}. Similarly for monotonically decreasing case we have:

$$\begin{aligned}&\triangle A(e)/(1+\epsilon)\\ \le\ &\triangle A(U_{k-1})/(1+\epsilon) = \triangle A(U_k) = \triangle A'(U_k)\\ \le\ &\triangle A'(e) \le \triangle A'(U_{k-1}) = \triangle A(U_{k-1})\\ =\ &(1+\epsilon)\triangle A(U_k) \le (1+\epsilon)\triangle A(e)\end{aligned}$$

□

B.4 Proof of Lemma 15

Proof. The first claim follows from the assumption that $\oplus$ does not introduce any error and it can be proved by the following:

$$\begin{aligned}&(\triangle A(e) \oplus \triangle B(e))/(1+\max(\beta,\gamma))\\ \le&\#C(e) = \triangle A'(e) \oplus \triangle B'(e)\\ \le&(1+\max(\beta,\gamma))(\triangle A(e) \oplus \triangle B(e))\end{aligned}$$

The second claim can be also proved similarly; based on definition of $\sqcap$, we have

$$\begin{aligned}\triangle D(e) &= \bigoplus_{j \le e} \bigoplus_{i \in \mathbb{R}} (\#A(j-i) \otimes \#B(i))\\ &= \bigoplus_{i \in \mathbb{R}} \bigoplus_{j \le e} (\#A(j-i) \otimes \#B(i))\\ &= \bigoplus_{i \in \mathbb{R}} (\#B(i) \otimes \bigoplus_{j \le e} \#A(j-i))\\ &= \bigoplus_{i \in \mathbb{R}} (\#B(i) \otimes \triangle A(e-i))\end{aligned}$$

Let $D'' = A' \sqcap B$, then based on the approximation guarantee of $\triangle A'(e)$ and the error properties of $\otimes$ and $\oplus$, we have

$$\triangle D(e)/(1+\beta) \le \triangle D''(e) \le (1+\beta)\triangle D(e)$$

Then the second claim follows by replacing B with B' in D'' and repeat the above step. □

C Background

We start by defining the joins.

Definition 16 (Join)**.** *Let $T_1, \dots, T_m$ be a set of tables and for each table T_i, let C_i denote the set of columns in T_i. The join of $T_1, \dots, T_m$ denoted by $T_1 \bowtie \cdots \bowtie T_m$ is a table J with set of column $C = \bigcup_j C_j$ such that a tuple/row x is in J if an only if its projection onto C_j is in T_j for all j.*

The structure of a join can be modeled as a hypergraph in which each vertex v_i of the hypergraph is associated with column f_i and each hyperedge S_j is associated with a table T_j such that $v_i \in S_j$ if and only if $f_i \in C_j$. In what the following, we use n to denote the size of the largest input table in the join query $Q = T_1 \bowtie \cdots \bowtie T_m$. We also use J to denote the output and $|J|$ to denote its size. We use the join query Q and its hypergraph $\mathcal{H}$ interchangeably.

C.1 Fractional edge cover number and output size bounds

Definition 1 (Fractional edge cover number ρ^*)**.** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph (of some query Q). Let $B \subseteq \mathcal{V}$ be any subset of vertices. A *fractional edge cover* of B using edges in $\mathcal{H}$ is a feasible solution $\vec{\lambda} = (\lambda_S)_{S \in \mathcal{E}}$ to the following linear program:

$$\begin{aligned}\min \quad & \sum_{S \in \mathcal{E}} \lambda_S\\ \text{s.t.} \quad & \sum_{S : v \in S} \lambda_S \ge 1, \quad \forall v \in B\\ & \lambda_S \ge 0, \quad \forall S \in \mathcal{E}.\end{aligned}$$

The optimal objective value of the above linear program is called the *fractional edge cover number* of B in $\mathcal{H}$ and is denoted by $\rho^*_{\mathcal{H}}(B)$. When $\mathcal{H}$ is clear from the context, we drop the subscript $\mathcal{H}$ and use $\rho^*(B)$.

Given a join query Q, the fractional edge cover number of Q is $\rho^*_{\mathcal{H}}(\mathcal{V})$ where $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is the hypergraph of Q.

Theorem 17 (AGM-bound [9, 15])**.** *Given a join query Q over a relational database instance I, the output size is bounded by*

$$|J| \leq n^{\rho^*},$$

where ρ^* is the fractional edge cover number of Q.

Theorem 18 (AGM-bound is tight [9, 15])**.** *Given a join query Q and a non-negative number n, there exists a database instance I whose relation sizes are upper-bounded by n and satisfies*

$$|J| = \Theta(n^{\rho^*}).$$

Worst-case optimal join algorithms [26, 22, 23] can be used to answer any join query Q in time

$$O(|\mathcal{V}| \cdot |\mathcal{E}| \cdot n^{\rho^*} \cdot \log n). \qquad (4)$$

C.2 Tree decompositions, acyclicity, and width parameters

Definition 2 (Tree decomposition)**.** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A *tree decomposition* of $\mathcal{H}$ is a pair (T, χ) where $T = (V(T), E(T))$ is a tree and $\chi : V(T) \to 2^{\mathcal{V}}$ assigns to each node of the tree T a subset of vertices of $\mathcal{H}$. The sets $\chi(t)$, $t \in V(T)$, are called the *bags* of the tree decomposition. There are two properties the bags must satisfy

(a) For any hyperedge $F \in \mathcal{E}$, there is a bag $\chi(t)$, $t \in V(T)$, such that $F \subseteq \chi(t)$.

(b) For any vertex $v \in \mathcal{V}$, the set $\{t \mid t \in V(T), v \in \chi(t)\}$ is not empty and forms a connected subtree of T.

Definition 3 (acyclicity)**.** A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is *acyclic* iff there exists a tree decomposition (T, χ) in which every bag $\chi(t)$ is a hyperedge of $\mathcal{H}$.

When $\mathcal{H}$ represents a join query, the tree T in the above definition is also called the *join tree* of the query. A query is acyclic if and only if its hypergraph is acyclic.

For non-acyclic queries, we often need a measure of how "close" a query is to being acyclic. To that end, we use *width* notions of a query.

Definition 4 (g-width of a hypergraph: a generic width notion [7])**.** Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, and $g : 2^{\mathcal{V}} \to \mathbb{R}^+$ be a function that assigns a non-negative real number to each subset of $\mathcal{V}$. The *g-width* of a tree decomposition (T, χ) of $\mathcal{H}$ is $\max_{t \in V(T)} g(\chi(t))$. The *$g$-width of $\mathcal{H}$* is the *minimum* g-width over all tree decompositions of $\mathcal{H}$. (Note that the g-width of a hypergraph is a *Minimax* function.)

Definition 5 (*Treewidth* and *fractional hypertree width* are special cases of *g-width*)**.** Let s be the following function: $s(B) = |B| - 1$, $\forall V \subseteq \mathcal{V}$. Then the *treewidth* of a hypergraph $\mathcal{H}$, denoted by $\mathsf{tw}(\mathcal{H})$, is exactly its s-width, and the *fractional hypertree width* of a hypergraph $\mathcal{H}$, denoted by $\mathsf{fhtw}(\mathcal{H})$, is the ρ^*-width of $\mathcal{H}$.

From the above definitions, $\mathsf{fhtw}(\mathcal{H}) \geq 1$ for any hypergraph $\mathcal{H}$. Moreover, $\mathsf{fhtw}(\mathcal{H}) = 1$ if and only if $\mathcal{H}$ is acyclic.

C.3 Algebraic Structures

In this section, we define some of the algebraic structures used in the paper. First, we discuss the definition of a monoid. A monoid is a semi-group with an identity element. Formally, it is the following.

Definition 19. *Fix a set S and let $\oplus$ be a binary operator $S \times S \to S$. The set S with $\oplus$ is a monoid if (1) the operator satisfies associativity; that is, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ for all $a, b, c \in S$ and (2) there is identity element $e \in S$ such that for all $a \in S$, it is the case that $e \oplus a = a \oplus e = a$.*

A commutative monoid is a monoid where the operator $\oplus$ is commutative. That is $a \oplus b = b \oplus a$ for all $a, b \in S$.

Next, we define a semiring.

Definition 20. *A semiring is a tuple $(R, \oplus, \otimes, I_0, I_1)$. The $\oplus$ operator is referred to as addition and the $\otimes$ is referred to as multiplication. The elements I_0 and I_1 are reffered as 0 element and 1 element and both are included in R. The tuple $(R, \oplus, \otimes, I_0, I_1)$ is a semiring if,*

1. *it is the case that R and $\oplus$ are a commutative monoid with I_0 as the identity.*

2. *R and $\otimes$ is a monoid with identity I_1.*

3. *the multiplication distributes over addition. That is for all $a, b, c \in R$ it is the case that $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$.*

4. *the I_0 element annihilates R. That is, $a \otimes I_0 = I_0$ and $I_0 \otimes a = I_0$ for all $a \in R$.*

A commutative semiring is a semiring where the multiplication is commutative. That is, $a \otimes b = b \otimes a$ for all $a, b \in S$.