Efficient Evaluation of `HAVING` Queries on a Probabilistic Database University of Washington TR: #2007-06-01

Christopher Ré and Dan Suciu

Department of Computer Science and Engineering
University of Washington
{chrisre,suciu}@cs.washington.edu

Abstract. We study the evaluation of positive conjunctive queries with Boolean aggregate tests (similar to `HAVING` queries in SQL) on probabilistic databases. Our motivation is to handle aggregate queries over imprecise data resulting from information integration or information extraction. More precisely, we study conjunctive queries with predicate aggregates using `MIN`, `MAX`, `COUNT`, `SUM`, `AVG` or `COUNT(DISTINCT)` on probabilistic databases. Computing the precise output probabilities for positive conjunctive queries (without `HAVING`) is $\sharp\mathcal{P}$-hard, but is in $\mathcal{P}$ for a restricted class of queries called safe queries. Further, for queries without self-joins either a query is safe or its data complexity is $\sharp\mathcal{P}$-Hard, which shows that safe queries exactly capture tractable queries without self-joins. In this paper, for each aggregate above, we find a class of queries that exactly capture efficient evaluation for `HAVING` queries without self-joins. Our algorithms use a novel technique to compute the marginal distributions of elements in a semiring, which may be of independent interest.

1 Introduction

We study the complexity of evaluating aggregate queries on probabilistic databases. Our motivation is managing data resulting from integration applications, e.g. object reconciliation [**?**,**?**,**?**] and information extraction [**?**,**?**,**?**,**?**]. Standard approaches require that we eliminate all uncertainty before any querying can begin, which is expensive in both man-hours to perform the integration and in lost revenue due to down time. An alternative approach where data are allowed to be uncertain but we capture uncertainty using probabilities has attracted renewed interest [**?**,**?**,**?**,**?**,**?**,**?**]. In such systems, individual tuples are allowed to be incorrect, but aggregations of tuples still provide meaningful information.

In SQL, aggregates come in two forms: *value aggregates* that are returned to the user in the `SELECT` clause (e.g. the `MAX` price) and *predicate aggregates* that appear in a `HAVING` clause (e.g. is the `MAX` price greater than $10.00?). In this paper, we focus on positive conjunctive queries with a single predicate aggregate which we call `HAVING` queries. Prior art [**?**,**?**] has defined a semantic for *value aggregation* that returns the expected value of an aggregate query (e.g. the expected `MAX` price) and have demonstrated its utility for Decision Support or OLAP style applications. In this paper, we propose a complementary semantic for predicate aggregates inspired by `HAVING` (e.g. what is the *probability* that the `MAX` price is bigger than \$10.00?). We illustrate the difference between the approaches with a simple example:

Example 1. Consider a probabilistic database with a `Profit` relation that contains the profit forecasted for each item if we continue to sell it:

Item	Forecaster	Profit	P
Widget	Alice	\$−99K	0.99
	Bob	\$100M	0.01
Whatsit	Alice	\$1M	1

`Profit`(Item;Forecaster,Profit;P)

```
SELECT SUM(PROFIT)
FROM PROFIT
WHERE ITEM='Widget'
```

(a) Expectation Style

```
SELECT ITEM
FROM PROFIT
WHERE ITEM='Widget'
HAVING SUM(PROFIT) > 0.0
```

(b) `HAVING` Style

`Profit` is an example of BID relation which captures our uncertain about contradictory and incompatible forecasts. Here, we trust Alice's forecast (probability 0.99) more than Bob's (0.01). Prior art [?] considered aggregate queries in the `SELECT` clause such as (a). Their semantic computes the *expected profit*. Using linearity of expectation, the value of query (a) is 100M * 0.01 + $-99K$ * 0.99 $\approx$ 900K. This large value suggests that we should continue selling widgets because we expect to make money. However, if we asked the `HAVING` style query (b), which says: What is the chance that we will make a profit? The answer is only 0.01, which tells us that we should immediately stop selling widgets or risk going out of business.

Prior work [?,?,?] has shown that for Boolean conjunctive queries without `HAVING`, computing a query's probability is $\sharp\mathcal{P}$-Complete[1]. Although in general evaluating conjunctive queries on a probabilistic database is hard, there are a class of queries that can be computed efficiently called *safe queries* [?,?]. In this paper, for each aggregate α, we find a class of `HAVING` queries called **α-safe** that can be evaluated efficiently. Further, we show that there is a dichotomy for queries without self-joins: Either Q is α-safe, and has an algorithm in $\mathcal{P}$, or Q is not α-safe and is $\sharp\mathcal{P}$-Hard.

Our starting observation is that evaluating a query with a value aggregate with aggregate α on a traditional database can be computed by annotating the database with values from some semiring, S_α, then computing the annotations returned by a the query by evaluating any algebra plan P over the semiring, using the rules in [?]. Therefore the output of an aggregate function α on a probabilistic database is described by a random variable s_Q with values in S_α, and a `HAVING` query Q whose predicate is, say, `COUNT`$(*) < k$, can be computed over the probabilistic database in two stages: first compute the random variable s_Q, second apply some *recovery function* that computes the probability $s_Q < k$. The cost of this algorithm depends on the space required to represent random variables s_Q, which is proportional to the set of possible worlds of the probabilistic database and hence is prohibitively high. Our key technical insight is that if the plan P is a *safe plan* then the random variable s_Q can be computed using a

[1] $\sharp\mathcal{P}$ defined by Valiant [?] is the class of functions that contains the problem of counting the number of solutions to $\mathcal{NP}$-Hard problems (e.g. `#3-SAT`).

much more concise representation called a *marginal vector*, because P can guarantee that the random variables being combined are either independent or disjoint. In addition we need to impose an extra condition on P to ensure that the recovery function can be computed from the marginal vector representation of s_Q, and we call this condition plus the safety condition for the plan P, which depends on α, α-safety.

Contributions and Overview We study conjunctive queries with `HAVING` predicates where the aggregation function is given by `MIN`, `MAX`, `COUNT`, `SUM`, `AVG` or `COUNT(DISTINCT)` and the aggregate test is one of $=, \neq, <, \leq, >$, or $\geq$ on common representations of probabilistic databases [**?**,**?**,**?**]. In Sec. 2, we formalize `HAVING` queries, our choice of representation and define efficient evaluation. In Sec. 3, we review the relevant technical material (e.g. semirings and safe plans). In Sec. 4, we give our main results: For each aggregate α, we find a class of `HAVING` queries, called α-safe, such that for any Q using α:

– If Q is α-safe then Q's data complexity is in $\mathcal{P}$.
– If Q has no self-joins and is not α-safe then, Q has $\sharp\mathcal{P}$-hard data complexity.
– It can be decided in polynomial time in the size of Q if Q is α-safe.

2 Formal Problem Description

We first define the syntax and semantics of `HAVING` queries on probabilistic databases and then define the problem of evaluating `HAVING` queries.

Semantics We consider the aggregates `MIN`, `MAX`, `COUNT`, `SUM`, `AVG` and `COUNT(DISTINCT)` as functions on multisets with the obvious semantics.

Definition 1. *A Boolean conjunctive query is a single rule $q = g_1, \ldots, g_m$ where each g_i is a positive EDB predicate. A Boolean `HAVING` query is a single rule:*

$$Q[\alpha(y)\ \theta\ k]\ \coloneq\ g_1, \ldots, g_n$$

where for each i, g_i is a positive EDB predicate, $\alpha \in \{$`MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, `COUNT(DISTINCT)`$\}$, y is a single variable[2], $\theta \in \{=, \neq, <, \leq, >, \geq\}$ and k is a constant. The set of variables in the body of Q is denoted $\mathbf{var}(Q)$*. We assume that* $y \in \mathbf{var}(Q)$*. The conjunctive query $q = g_1, \ldots, g_n$, is called the* ***skeleton*** *of Q, denoted* $\mathbf{sk}(Q) = q$*. We call θ the* ***predicate test****, k, the* ***predicate operand*** *and a pair (α, θ) an* ***aggregate test****.*

Fig. 1(a) shows a SQL query with a `HAVING` predicate, that asks for all movies reviewed by at least two distinct reviewers. A translation of this query into an extension of our syntax is shown in Fig. 1(b). The translated query is not a Boolean `HAVING` query because it has a head variable (d). In this paper, we discuss only Boolean `HAVING` queries. However, as is standard, to study the complexity of non-boolean queries, we can substitute constants for head variables. For example, if we substitute 'M. Ritchie' for d, then the result is Fig. 1(c) which is a Boolean `HAVING` query.

[2] For `COUNT`, we will omit y and write the more familiar `COUNT(*)` instead.

```
SELECT m.Title
FROM MovieMatch m, Reviewer r
WHERE m.ReviewTitle = r.ReviewTitle
GROUP BY m.Title
HAVING COUNT(DISTINCT r.reviewer) ≥ 2
```

(a) SQL Query

$Q(m)[\texttt{COUNT(DISTINCT}\ y) \geq 2]$:–
 $\texttt{MovieMatch}^p(t, m)$,
 $\texttt{Reviewer}^p(-, r, t)$

(b) Extended Sytanx (Non Boolean)

$Q[\texttt{COUNT(DISTINCT}\ y) \geq 2]$:–
 $\texttt{MovieMatch}^p(\text{`Fletch'}, m)$,
 $\texttt{Reviewer}^p(-, r, t)$

(c) Syntax of this paper

Fig. 1. A translation of the query "Which moviews have been reviewed by at least 2 distinct reviewers?" into SQL and the syntax of this paper.

Definition 2. *Given a* `HAVING` *query* $Q[\alpha(y)\ \theta\ k]$ *and a relational instance* W*, let*

$$\mathcal{Y} = \{\!|\ v(y) \mid v \text{ is a valuation for } Q \text{ and } \mathbf{im}(v) \subseteq W\ |\!\}$$

i.e. $\mathcal{Y}$ *is the multiset of all valuations of* Q *applied to* y*. We say that* Q *is satisfied on* W *and write* $W \models Q[\alpha(y)\ \theta\ k]$ *(or simply* $W \models Q$*) if* $\mathcal{Y} \neq \emptyset$ *and* $\alpha(\mathcal{Y})\ \theta\ k$ *holds.*

Probabilistic Databases In this paper, we will use probabilistic databases described in the block-independent disjoint (BID) representation [**?**,**?**] which generalizes many representations in the literature including p-?-sets and p-or-sets [**?**], ?- and x-relations [**?**] and tuple independent databases [**?**,**?**] and is similar to [**?**].

Syntax A **BID schema** has relational schemas of the form $\texttt{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ with its attributes partitioned into three classes separated by semicolons: the **possible worlds key** ($\boldsymbol{K}$), the **value attributes** ($\boldsymbol{A}$), and a single distinguished probability attribute P taking values in $(0, 1]$. The corresponding **possible worlds schema** has relations of the form $\texttt{R}(\boldsymbol{K}; \boldsymbol{A})$, i.e. the same schema without the attribute P.

Semantics Let J be an instance of a BID schema. We denote by $t[KAP]$ a tuple in J, emphasizing its three kinds of attributes, and call $t[KA]$, its projection on the KA attributes, a *possible tuple*. Define a *possible world*, W, to be any instance consisting of possible tuples s.t. K is a key in W. Note that the key constraints do not hold in J, but do hold in any possible world. Let $\mathcal{W}_J$ be the set of all possible worlds. We define the semantics of BID instances only for *valid* instances, which are instances J s.t. for every tuple $t \in R^J$ in any BID relation $\texttt{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ the inequality $\sum_{s \in \texttt{R}: s[\boldsymbol{K}]=t[\boldsymbol{K}]} s[P] \leq 1$ holds. For a valid instance J its semantics is a finite probability space $(\mathcal{W}_J, \mu_J)$. First note that any possible tuple $t[KA]$ can be viewed as an event in the probability space $(\mathcal{W}_J, \mu_J)$, namely the event that a world contains $t[KA]$. Then we define the semantics of J to be the probability space $(\mathcal{W}_J, \mu_J)$ s.t. (a) the marginal probability of any possible tuple $t[KA]$ is $t[P]$, (b) any two tuples from the same relation $t[KA]$, $t'[KA]$ s.t. $t[K] = t'[K]$ are *disjoint events* (a.k.a. exclusive events), and (c) for any set of tuples $\{t_1, \ldots, t_n\}$ s.t. all tuples from the same relation have distinct keys, the events defined by these tuples are independent.

Example 2. The data in Fig. 2 shows an example of a BID representation that stores data from integrating extracted movie reviews (e.g. from USENET) with a movie database

Title	Matched	P	
'Fletch'	'Fletch'	0.98	m_1
'Fletch'	'Fletch 2'	0.9	m_2
'Fletch 2'	'Fletch'	0.4	m_3
'The Golden Child'	'The Golden Child'	0.95	m_4
'The Golden Child'	'Golden Child'	0.8	m_5
'The Golden Child'	'Wild Child'	0.2	m_6

`MovieMatch`(CleanTitle,ReviewTitle;P)

ReviewID	Reviewer	Title	P	
231	'Ryan'	'Fletch'	0.7	t_{231a}
		'Spies Like Us'	0.3	t_{231b}
232	'Ryan'	'European Vacation'	0.90	t_{232a}
		'Fletch 2'	0.05	t_{232b}
235	'Ben'	'Fletch'	0.8	t_{235a}
		'Wild Child'	0.2	t_{235b}

`Reviews`(Reviewer,ReviewTitle;Rating;P)

Fig. 2. Sample Data arising from integrating automatically extracted reviews from a movie database. `MovieMatch` is a probabilistic relation, we are uncertain which review title matches with which movie in our clean database. `Reviews` is uncertain because it is the result of *information extraction* and *sentiment analysis*.

(e.g. IMDB). The `MovieMatch` table is uncertain because it is the result of an automatic matching procedure. For example, the probability a review title 'Fletch' matches a movie titled 'Fletch' is very high 0.95, but not 1 because the title is extracted from text and so may contain errors: For example, from 'The second Fletch movie', our extractor will likely extract just 'Fletch' although this review actually refers to 'Fletch 2'. The review table is uncertain because it is the result of *information extraction* and so we have extracted the title from free text (e.g. 'Fletch is a great movie, just like Spies Like Us'). Notice that $t_{232a}[P] + t_{232b}[P] = 0.95 < 1$, which indicates that there is some probability reviewid 232 is actually not a review at all.

Remark 1. Recall that two distinct possible $t[KA]$ and $t'[KA]$ are disjoint if $t[K] \neq t'[K]$ and $t[A] = t'[A]$. But what happens if $A = \emptyset$, i.e. all attributes are part of the possible worlds key ? In that case all possible tuples become independent, and we sometime call a table $R(K; ; P)$ a *tuple independent table* [**?**] or a ?*-table* [**?**] or a *p-?-table* [**?**].

Queries are posed over the possible worlds schema. For clarity, we denote such relations with a superscripted p (e.g. $\texttt{Reviews}^p$).

Definition 3 (Query Semantics). *The marginal probability of a* `HAVING` *query Q on BID database J is denoted $\mu_J(Q)$ (or simply $\mu(Q)$) and is defined by:*

$$\mu_J(Q) = \sum_{W \in \mathcal{W}_J : W \models Q} \mu_J(W)$$

We also make use of $\mu_J(q)$ where q is a Boolean conjunctive query with the standard semantics.

Example 3. Fig. 1(c) shows a query which asks for all movies that were reviewed by at least 2 different reviewers. The movie 'Fletch' is present when the formula is satisfied $(m_1 \wedge t_{231a}) \vee (m_2 \wedge t_{232b}) \vee (m_1 \wedge t_{235a})$ and the multiplicity of tuples is exactly the number of disjuncts satisfied. Thus, $\mu(Q)$ is the probability that at least two of these disjuncts are true, which semantically can be computed by summing over all possible worlds.

Notions of complexity for `HAVING` queries In the database tradition, we would like to measure the data complexity [**?**], i.e. treat the query as fixed, but allow the data to grow. This assumption makes sense in practice because the running time for query evaluation can be $O(n^{f(|Q|)})$ where $|Q|$ is the size of a conjunctive query Q. This exponential running time is considered to be tenable in practice, because database queries are generally small. However, in our setting this introduces a problem. By fixing a `HAVING` query q we also fix k, which means that we should accept a running time n^k as efficient. Clearly this is undesirable: because k can be large. For example, $Q()[\texttt{SUM}(y) > 200] \coloneq R(x, y)$. For that reason we consider in this paper an alternative definition of the data complexity of `HAVING` queries, where both the database and k are port of the input.

Definition 4. *Fix a skeleton q, an aggregate α, and a comparison operator θ. The **query evaluation problem** is: given as input the encoding of a BID representation J,* and *a binary representation of $k > 0$, calculate $\mu_J(Q)$, where $Q[\alpha\ \theta\ k]$ is such that $\mathbf{sk}(Q) = q$.*

The technical problem we address in this work is the complexity of query evaluation. We shall see for the query in Ex. 3, that the query evaluation problem is hard for $\sharp\mathcal{P}$. And moreover, this is the general case for all `HAVING` queries.

3 Preliminaries

We review here some basic facts on semirings (mostly from [**?**] and introduce random variables over semirings.

3.1 Background: Random Variables on Semirings

Definition 5. *A **monoid** $(S, +, 0)$ is a set S and $+$ is an associative binary operation with identity, 0. A **semiring** is a structure $(S, +, \cdot, 0, 1)$ where $(S, +, 0)$ is a commutative monoid with identity 0, $(S, \cdot, 1)$ is a monoid with identity 1, $\cdot$ distributes over $+$ and 0 annihilates S. A commutative semiring is one in which $(S, \cdot, 1)$ forms a commutative monoid. We abbreviate either structure with the set S when clear from the context.*

We shall consider only commutative semirings in this paper.

Example 4. For integer $k \geq 0$, let $\mathbb{Z}_{k+1} = \{0, 1, \ldots, k\}$ then for every such k, $(\mathbb{Z}_k, \max, \min, 0, k)$ is a semiring. In particular, $k = 2$ is the Boolean semiring. Another set of semirings we consider, $\mathbb{S}_k = (\mathbb{Z}_k, +_k, \cdot_k, 0, 1)$ where $+_k(x, y) = \min(x + y, k)$ and $\cdot_k = \min(xy, k)$ where addition and multiplication are in $\mathbb{Z}$.

For the rest of this section fix a BID instance J, and denote $(\mathcal{W}, \mu) = (\mathcal{W}_J, \mu_J)$ the probability space induced by J on possible worlds.

Definition 6. *Given a semiring $(S, +_S, \cdot_S)$, an S**-random variable**, r, is a function $r : \mathcal{W} \to S$. Given two random variables r, s then $r +_S s$ and $r \cdot_S s$ are random variables defined as $(r +_S s)(W) = r(W) +_S s(W)$ and $(r \cdot_S s)(W) = r(W) \cdot_S s(W)$.*

In the sequel, we make use of the following fact:

Fact 1 *The set of S-random variables on a fixed BID instance J induces a semiring, denoted S^J, with the operations in Def. 6.*

Definition 7. *Given a semiring S and a set of random variables $R = \{r_1, \dots, r_n\}$ on S, R is* ***independent*** *if $\forall N \subseteq 1, \dots, n$ and any set $k_1, \dots, k_n \in S$, we have $\mu(\bigwedge_{i \in N} r_i = k_i) = \prod_{i \in N} \mu(r_i = k_i)$. We say that R is* ***disjoint*** *if for any $i \neq j$ we have $\mu((r_i \neq 0) \wedge (r_j \neq 0)) = 0$.*

To represent a single random variable we need space as large as $|\mathcal{W}|$, which is exponential in the size of the J and thus prohibitive for most applications. However, there exists an alternative representation in terms of *marginal vectors*, which only takes size S.

Definition 8. *Given a random variable r on S, the* ***marginal vector*** *(or simply, the marginal) of r is denoted $\boldsymbol{m}^r$ and is a vector indexed by S defined by $\forall s \in S \mu(r = s) = \boldsymbol{m}^r[s]$. Given a monoid $(S, +_S, 0)$, the* ***monoid convolution*** *is a binary operation on marginals denoted $*^{+_S}$, and for any marginals $\boldsymbol{m}^r$ and $\boldsymbol{m}^t$ is defined by*

$$\forall s \in S \ (\boldsymbol{m}^r *^{+_S} \boldsymbol{m}^t)[s] \overset{\text{def}}{=} \sum_{\substack{i +_S j = k \\ i, j \in S}} \boldsymbol{m}^r[i] \boldsymbol{m}^t[j]$$

The disjoint operation for $(S, 0)$ is denoted $\boldsymbol{m}^r \perp \boldsymbol{m}^s$ and is defined by

$$\textit{if } s \neq 0 \ (\boldsymbol{m}^r \perp \boldsymbol{m}^s)[s] \overset{\text{def}}{=} \boldsymbol{m}^r[s] + \boldsymbol{m}^s[s] \ \textit{else} \ (\boldsymbol{m}^r \perp \boldsymbol{m}^s)[0] \overset{\text{def}}{=} (\boldsymbol{m}^r[0] + \boldsymbol{m}^s[0]) - 1$$

The next proposition tells us when the operations defined in the previous definition yield the correct results:

Proposition 1. *If r and s are random variables on the monoid $(S, +_S, 0)$ with marginal vectors $\boldsymbol{m}^r$ and $\boldsymbol{m}^s$ then let $\boldsymbol{m}^{r +_S s}$ denote the marginal of $r +_S s$. If r and s are independent then $\boldsymbol{m}^{r +_S s} = \boldsymbol{m}^r *^{+_S} \boldsymbol{m}^s$. If r and s are disjoint then $\boldsymbol{m}^{r +_S s} = \boldsymbol{m}^r \perp \boldsymbol{m}^s$. Further, the n-fold convolution can be computed in time $O(n|S|^2)$ and the n-fold disjoint operation can be computed in $O(n|S|)$.*

The importance of this proposition is that if the semiring is small, then each operation can be done efficiently.

Example 5. Consider the Boolean semiring, and two random variables r and s with marginal probabilities (of truth) p_r and p_s. Then $\boldsymbol{m}^r = (1 - p_r, p_r)$ and $\boldsymbol{m}^s = (1 - p_s, p_s)$. If r and s are independent then, the distribution of $r \vee s = r +_S s$ which can be computed using $r *^+ s$. This satisfies $(r *^+ s)[1] = p_r(1 - p_s) + (1 - p_r)p_s + p_r p_s$ or in a more familiar form, $\boldsymbol{m}^r *^+ \boldsymbol{m}^s = ((1 - p_r)(1 - p_s), 1 - (1 - p_r)(1 - p_s))$.

3.2 Background: Queries on databases annotated from a semiring

In this section, we review material from [?] on computing queries on databases annotated with semiring elements. A slight twist on prior art is that we allow the query to induce the annotations as a first step, rather than being part of the data.

Definition 9. *Given a commutative semiring S and a Boolean conjunctive query $q = g_1, \dots, g_n$, an annotation is a set of functions indexed by subgoals, such that for $i = 1, \dots, n$, τ_{g_i} is a function on tuples that unify with g_i to S. We denote the set of annotation functions with τ.*

As in [?], we compute the annotation using a modified relational algebra which we define below:

Definition 10.

- *a **plan** P is inductively defined as (a) a single subgoal (b) $\pi_{-x}P_1$ if P_1 is a plan (b) $P_1 \bowtie P_2$ if P_1, P_2 are plans.*
- $\mathbf{var}(P)$*, the variables output by P, is defined as* $\mathbf{var}(g)$ *if* $P = g$, $\mathbf{var}(\pi_{-x}P) = \mathbf{var}(P) - \{x\}$ *and* $\mathbf{var}(P_1 \bowtie P_2) = \mathbf{var}(P_1) \cup \mathbf{var}(P_2)$.
- $\mathbf{goal}(P)$*, the set of subgoals in P, is defined as (a)* $\mathbf{goal}(g) = \{g\}$, *(b)* $\mathbf{goal}(\pi_{-x}P_1) = \mathbf{goal}(P_1)$ *(c)* $\mathbf{goal}(P_1 \bowtie P_2) = \mathbf{goal}(P_1) \cup \mathbf{goal}(P_2)$.

The **value** of a plan P on a standard instance W is a set of tuples with attributes corresponding to the variables in $\mathbf{var}(P)$ each annotated with a semiring element, denoted $\omega_P^W(t)$, which is defined inductively below. Concurrently, we define the support of a tuple $\mathbf{supp}_{P,V}(t) = \{t' \mid \forall y \in V\ t[y] = t'[y] \ \wedge\ \omega_P^W(t') \neq 0\}$ where P is a plan, V is a set of variables such that $V \subseteq \mathbf{var}(P)$, t is a tuple with attributes corresponding to V and t' is a tuple with attributes corresponding to $\mathbf{var}(P)$.

- If $P = g$ then if t unifies with g then $\omega_P^W(t) = \tau_g(t)$ else $\omega_P^W(t) = 0$.
- If $P = \pi_{-x}P_1$, then $\omega_{\pi_{-x}P_1}^W(t) = \sum_{t' \in \mathbf{supp}^W_{P_1,\mathbf{var}(P)}(t)} \omega_{P_1}^W(t')$.
- else $P = P_1 \bowtie P_2$ and for $i = 1, 2$ let t_i be t restricted to $\mathbf{var}(P_i)$ then $\omega_{P_1 \bowtie P_2}^W(t) = \omega_{P_1}^W(t_1) \cdot \omega_{P_2}^W(t_2)$

A result of [?] shows that ω_P^W is independent of the choice of plan P, which justifies the notation $s_{\tau,W,q}$, the value of a conjunctive query q on a determinsitic instance W under annotation τ defined as $s_{\tau,W,q} \overset{\text{def}}{=} \omega_P^W()$ where P is any plan for q where and ω_P^W is applied to the empty tuple.

4 Approaches for `HAVING`

In this section, we define the α-safe `HAVING` queries for $\alpha \in \{$`MIN`, `MAX`, `COUNT`$\}$ in Sec. 4.3, for $\alpha =$ `COUNT(DISTINCT)` in Sec. 4.4 and $\alpha \in \{$`AVG`, `SUM`$\}$ in Sec. 4.5.

4.1 Aggregates and semirings

We explain the details of computing HAVING queries using semirings on deterministic databases, which immediately generalizes to probabilistic databases. Since HAVING queries are Boolean, we use a function ρ, the **recovery function**, which maps semiring values to true if that value would satisfy the HAVING query. Fig. 3 lists the (commutative) semirings for the aggregates in this paper, their annotations τ and a Boolean recovery function ρ. EXIST is similar to the safe plan algebra of [**?**,**?**,**?**].

HAVING Predicate	Semiring	Annotation $\tau_{g^*}(t)$	Recovery $\rho(s)$
EXISTS	$(\mathbb{Z}_2, \max, \min)$	1	$s = 1$
MIN(y) $\{<, \leq\}$ k	$(\mathbb{Z}_3, \max, \min)$	if $t\ \theta\ k$ then 2 else 1	$s = 2$
MIN(y) $\{>, \geq\}$ k	$(\mathbb{Z}_3, \max, \min)$	if $t\ \theta\ k$ then 1 else 2	$s = 1$
MIN(y) $\{=, \neq\}$ k	$(\mathbb{Z}_4, \max, \min)$	if $t < k$ then 3 else if $t = k$ then 2 else 1	if = then $s = 2$ if $\neq$ then $s \neq 2$
COUNT$(*)$ θ k	$\mathbb{S}_{k+1}$	1	$(s \neq 0) \wedge (s\ \theta\ k)$
SUM(y) θ k	$\mathbb{S}_{k+1}$	$t[y]$	$(s \neq 0) \wedge (s\ \theta\ k)$

Fig. 3. Semirings for the operators MIN, COUNT and SUM. Let g^* be the lowest indexed subgoal such that contains y. For all $g \neq g^*$, $\forall t$, $\tau_g(t)$ equals the multiplicative identity of the semiring. Let $\mathbb{Z}_{k+1} = \{0, 1, \ldots, k\}$ and $+_k(x, y) \overset{\text{def}}{=} \min(x + y, k)$ and $\cdot_k \overset{\text{def}}{=} \min(xy, k)$, where $x, y \in \mathbb{Z}$. Let $\mathbb{S}_k \overset{\text{def}}{=} (Z_{k+1}, +_k, \cdot_k, 0, 1)$. MAX and MIN are symmetric. AVG and COUNT(DISTINCT) are omitted.

Example 6. Consider the query $Q[\texttt{MIN}(y) \geq 10]$:– $R(y)$ where $R = \{t_1, \ldots, t_n\}$. Fig. 3 tells us to use the semiring $(\mathbb{Z}_3, \max, \min)$. We first apply τ: $\tau(t_i) = 1$ represents that $t_i[y] > 10$ while $\tau(t_i) = 2$ represents that $z_i[y] \leq 10$. Let $s = \sum_{i=1,\ldots,m}^{S} \tau(t_i) = \max_{i=1,\ldots,m} \tau(t_i)$. $\rho(s)$ is satisfied only when s is 1, i.e. all $z_i[y]$ are greater than 10.

More generally, we have the following proposition:

Proposition 2. *Given a HAVING query Q, let $q = \mathbf{sk}(Q)$ and S, ρ and τ be chosen as in Fig. 3, then for any deterministic instance W and $s_{\tau,W,q}$ (Sec. 3.2):*

$$W \models Q \iff \rho(s_{\tau,W,q})$$

In probabilistic databases, we want to compute the random variable $s_{\tau,q}$ defined as $s_{\tau,q}(W) \overset{\text{def}}{=} s_{\tau,W,q}$. A simple corollary of Prop. 2 is the following generalization to probabilistic databases:

Corollary 1. *Given Q, let q, S ,ρ and τ be as in Prop. 2 then for any BID instance J we have the following equalities:*

$$\mu_J(Q) = \mu_J(\rho(s_{\tau,W,q})) = \sum_{k\,:\,\rho(k)} \boldsymbol{m}^{s_{\tau,q}}[k]$$

Cor. 1 tells us that examining the entries of the marginal vector at index s such that $\rho(s)$ is true is sufficient to answer Q. Hence our goal is to compute $\boldsymbol{m}^{s_{\tau,q}}$.

4.2 Computing safely in semirings

We now extend safe plans to compute a marginal vector instead of a Boolean value. Specifically, we compute $\boldsymbol{m}^{s_{\tau,q}}$, the marginal vector for $s_{\tau,q}$ using the operations defined in Sec. 3.1.

Definition 11. *An extensional plan for a Boolean conjunctive query q is a subgoal g and if P_1, P_2 are extensional plans then so are $\pi^I_{-x}P_1$, $\pi^D_{-x}P_1$ and $P_1 \bowtie P_2$. An extensional plan P is **safe** if P_1 and P_2 are safe then*

- *$P = g$ is safe*
- *$P = \pi^I_{-x}P_1$ is safe if $x \in \mathbf{var}(P_1)$ and $\forall g \in \mathbf{goal}(P_1)$ then $x \in \mathbf{key}(g)$*
- *$P = \pi^D_{-x}P_1$ is safe if $x \in \mathbf{var}(P_1)$ and $\exists g \in \mathbf{goal}(P_1)$, $\mathbf{key}(g) \subseteq \mathbf{var}(P)$, $x \in \mathbf{var}(g)$.*
- *$P = P_1 \bowtie P_2$ is safe if $\mathbf{goal}(P_1) \cap \mathbf{goal}(P_2) = \emptyset$*
  - *and for $i = 1, 2$ $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$*

An extensional plan P is a safe plan for q if P is safe and $\mathbf{goal}(P) = q$.

Proposition 3. *If P is a safe plan for q, then for $x \in \mathbf{var}(q)$ there is exactly one of π^I_{-x} or π^D_{-x} in P.*

At least one of the two projections must be present, because we must remove the variable x (q is boolean). If there were more than one in the plan, then they cannot be descendants of each other because $x \notin \mathbf{var}(P_1)$ for the ancestor and they cannot be joined afterward because of the join condition for $i = 1, 2$ $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$

Definition 12. *Given a BID instance J. Let P be an safe plan, then denote the **extensional value** of P in S on J as $\hat{\omega}^J_{P,S}$ which is a marginal vector on S defined inductively:*

- *$\hat{\omega}^J_{g,S}(t) = \hat{\tau}_g(t)$ where $\hat{\tau}_g(t)$ is the marginal vector $\boldsymbol{m}^t$ given by $\boldsymbol{m}^t[0] = 1 - t[P]$ and $\boldsymbol{m}^t[\tau_g(t)] = t[P]$, i.e. the (probabilistic) image of τ.*
- *$\hat{\omega}^J_{\pi^I_{-x}P_1,S}(t) = \ast^{+S}_{t' \in \mathbf{supp}_{P_1,\mathbf{var}(P)}(t)} \hat{\omega}^J_{P_1,S}(t)$.*
- *$\hat{\omega}^J_{\pi^D_{-x}P_1,S}(t) = \perp_{t' \in \mathbf{supp}_{P_1,\mathbf{var}(P)}(t)} \hat{\omega}^J_{P_1,S}(t)$.*
- *$\hat{\omega}^J_{P_1 \bowtie P_2}(t) = \hat{\omega}^J_{P_1,S}(t_1) \ast^{\cdot} \hat{\omega}^J_{P_2,S}(t_2)$ where for $i = 1, 2$ t_i is t restricted to $\mathbf{var}(P_i)$*

The next lemma uses the observation that an operator in a safe plan and the operator used to compute the value have the same correlation assumptions. For example, π^I assumes independence, which is required for $\ast^+$.

Lemma 1. *If P is a safe plan for a Boolean query q then for any $s_i \in S$ on any BID instance J, we have $\hat{\omega}^J_P()[s_i] = \mu_J(s_{\tau,q} = s_i)$.*

Remark 2. A safe plan is not necessarily efficient for any S (Def. 4). In particular, the operations in a safe plan on S take time polynomial in $|S|$. Thus, if the size of S grows super-polynomially in $|J|$, the size of the BID instance, the plan will not be efficient.

4.3 MIN, MAX and COUNT-safe

We can now formalize the class of queries which are efficient for MIN, MAX and COUNT.

Definition 13. *If $\alpha \in \{MIN, MAX, COUNT\}$ and $Q[\alpha(t)\ \theta\ k]$ is a HAVING query, then Q is α-**safe** if the skeleton of Q is safe.*

Theorem 1. *If $Q[\alpha(y)\ \theta\ k]$ is a HAVING query for $\alpha \in \{MIN, MAX, COUNT\}$ and Q is α-safe then the exact evaluation problem for Q is in polynomial time.*

Correctness is straightforward from Lem. 1. Efficiency follows because the semiring is of polynomial size. Hence the translation and each operation in the evaluation is of polynomial size for each aggregate in the theorem. In particular, S is constant for MIN and MAX and upper bounded by n for COUNT.

Remark 3. We remark that for SUM, we can only guarantee that $|S| = O(k)$, which implies a running time of $O(kn^{|Q|})$, which is not efficient (Def. 4).

The results of [**?**,**?**,**?**] show that either a conjunctive query without self-joins has a safe plan or it is $\sharp\mathcal{P}$-hard. It is not hard to see that each aggregate has at least one test so that a HAVING query Q is satisfied only when the skeleton of Q is satisfied. It is then straightforward to extend to all predicate tests. Formally, we have:

Theorem 2. *If $\alpha \in \{MIN, MAX, COUNT\}$ and $Q[\alpha(y)\ \theta\ k]$ does not contain self-joins, if Q is α-safe then Q has data complexity in $\mathcal{P}$ else Q has $\sharp\mathcal{P}$-hard data complexity. Further, we can find an α-safe plan in $\mathcal{P}$.*

The algorithm to find a safe plan is identical to [**?**,**?**].

4.4 COUNT(DISTINCT)-safe queries

Intuitively, we compute COUNT(DISTINCT) in two stages: proceeding bottom-up, we first compute the probability a y value appears (i.e. DISTINCT), we then count the number of distinct values (i.e. COUNT) using the techniques of the previous section. However, one caveat is that the representation is lossy: We do not know *which* values are present, only the distribution of their count. This implies that not all operations on these lossy marginal vectors are correct, which restricts the class of allowable plans:

Definition 14. *A query $Q[COUNT(DISTINCT\ y)\ \theta\ k]$ is **COUNT(DISTINCT)-safe** if there is a safe plan P for the skeleton of Q such that π^I_{-y} or π^D_{-y} is in P and no proper ancestor is π^I_{-x} for any x.*

Example 7. Fix a BID instance J. Consider $Q[\texttt{COUNT(DISTINCT}\ y) > 2] \coloneq R(y,x), S(y)$, a safe plan for the skeleton of Q is $P = \pi^I_{-y}((\pi^I_{-x}R(y,x)) \bowtie S(y))$. For the subquery $P_1 = (\pi^I_{-x}R(y,x)) \bowtie S(y)$, calculate the probability that each value EXISTS in the subplan P_1, i.e. for each t, $\hat{\omega}^J_{P,\texttt{EXISTS}}(t)$. Intuitively, all tuples returned by the plan are independent because π^I_{-y} is correct, and all y values are trivially distinct.

Intuitively, we map each EXISTS marginal vector to a vector suitable for computing COUNT, i.e. a vector in $\mathbb{Z}_k$ where $k = 2$, in this problem. In other words, $(1-p, p) =$

$\hat{\omega}^J_{P,\texttt{EXISTS}}(t) = \boldsymbol{m}^t$ is mapped to $\hat{\tau}(\boldsymbol{m}^t) = (1-p, p, 0)$. Thus, the correct distribution is given by $\ast_{t' \in \mathbf{supp}_P(())} \hat{\tau}(t')$. To compute the final result, use the recovery function, ρ defined by $\rho(s) \iff s > 2$

The proof of the following theorem is a generalization of Ex. 7, whose proof we include in the appendix:

Theorem 3. *If Q is COUNT(DISTINCT)-safe then its evaluation problem is in $\mathcal{P}$.*

Complexity We now establish that for COUNT(DISTINCT) queries without self-joins, COUNT(DISTINCT)-safe captures efficient computation. We first show the canonical hard patterns for COUNT(DISTINCT) and then extend this to show that the evaluation of any COUNT(DISTINCT) query without self-joins that is not COUNT(DISTINCT)-safe can be reduced to one of these hard patterns.

Proposition 4. *The following HAVING queries are $\sharp\mathcal{P}$-hard for $i \geq 1$:*

$$Q_1[\mathit{COUNT}(\mathit{DISTINCT}\, y)\,\theta\, k] \text{ :- } \mathtt{R}^p(x), \mathtt{S}(x,y) \text{ and } Q_{2,i}[\mathit{COUNT}(\mathit{DISTINCT}\, y)\,\theta\, k] \text{ :- } \mathtt{R}^p_1(x;y), \ldots, \mathtt{R}^p_i(x;y)$$

Proof (Sketch). To see that Q_1 is hard, we reduce from counting the number of independent sets in a graph (V, E). We let k be the number of edges, intuitively Q is satisfied only when all edges are present. For each node $u \in V$, create a tuple $\mathtt{R}(u)$ with probability 0.5. For edge $e = (u, v)$ create two tuples $\mathtt{S}(e, u), \mathtt{S}(e, v)$, each with probability 1. For any set $V' \subseteq V$, let $W_{V'}$ denote the world corresponding where the tuples corresponding to V' are present. For any subset of nodes, N, we show that if N is an independent set if and only if W_{V-N} satisfies Q_1. Since $f(N) = V - N$ is one-to-one, the number of possible worlds that satisfy Q_1 are exactly the number of independent sets. If N is independent, then for any edge (u, v), it must be the case that at least one of u or v is in $V - N$, thus every edge is present and Q is satisfied. If N is not independent, then there must be some edge (u, v) such that $u, v \in N$, hence neither of u, v is in $V - N$. Since this edge is missing, Q_1 cannot be satisfied. The hardness of Q_2 is based on a reduction from counting the set covers of a fixed size and is in the appendix.

There is some work in generalizing the previous theorem, we show in the appendix:

Theorem 4. *If Q is not COUNT(DISTINCT)-safe and does not contain self-joins, then Q has $\sharp\mathcal{P}$-hard data complexity.*

Proof (Sketch). We sketch the proof in the simpler case when only tuple independent probabilistic tables are used in Q. Assume the theorem fails, let Q be the minimal counter example in terms of subgoals; this implies we may assume that Q is connected and the skeleton of Q is safe. Since there is no safe plan projecting on y and only independent projects are possible, the only condition that can fail is that some subgoal does not contain y. Thus, there are at least two subgoals $R(\boldsymbol{x})$ and $S(\boldsymbol{z}y)$ such that $y \notin \boldsymbol{x} \cup \boldsymbol{z}$ and $\boldsymbol{x} \cap \boldsymbol{z} \neq \emptyset$. Given a graph (V, E), we then construct a BID instance J exactly as in the proof of Prop. 4. Only the R relation is required to have probabilistic tuples, all others can set their probabilities to 1.

Extending to tuple-disjoint databases requires slightly more work, because adding multiple tuples with probability 1 may violate a possible world key constraint. The full proof appears in the appendix. It is straightforward to decide if a plan is COUNT(DISTINCT)-safe, the safe plan algorithm of [**?**,**?**] simply tries only disjoint projects and joins until it is able to project away y or it fails.

4.5 SUM-safe and AVG-safe queries

To find SUM- and AVG-safe queries, we further restrict the class of allowable plans. Intuitively, if each value for y is present disjointly, then computing the multiplicity of each value is sufficient to compute the query, which we accomplish using the COUNT algebra of Sec. 4.3. Since computing SUM and AVG for a HAVING query with a single tuple independent table is already $\sharp\mathcal{P}$-Hard, our safe plans we must not contain an independent project for y (π^I_{-y}).

Definition 15. *A HAVING query $Q[\alpha(y)\ \theta\ k]$ for $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ is α-safe, if there is a safe plan P for the skeleton of Q such that π^D_{-y} in P and no proper ancestor of π^D_{-y} is π^I_{-x} for any x.*

Theorem 5. *If $Q[\alpha(y)\ \theta\ k]$ for $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ is α-safe, then Q's evaluation problem is in $\mathcal{P}$.*

Proof (Sketch). Since Q is α-safe, there is a plan P satisfying Def. 15. We may assume without loss of generality that $P = \pi^D_{-y}(P_1)$. We compute P_1 using the algebra for COUNT, $\hat{\omega}^J_{\texttt{COUNT},P_1}(t)$, which is correct because P_1 is safe. We then have the following equation which can be computed efficiently:

$$\mu(Q) = \sum_{t \in \mathbf{supp}_{P_1}()} \sum_{s : s * t[y]\ \theta\ k} \hat{\omega}^J_{P_1,\texttt{COUNT}}[s]$$

Example 8. Consider $Q[\texttt{SUM}(y) > 10]$:– $\texttt{R}(\text{‘a’}; y), \texttt{S}(y, u)$. This query is SUM-safe, with plan $\pi^D_{-y}(\texttt{R}(\text{‘a’}; y) \bowtie \pi^I_{-u}\texttt{S}(y, u))$.

Complexity We show that if a HAVING query without self-joins is not SUM-safe then, it has $\sharp\mathcal{P}$-data complexity. AVG follows by essentially the same construction.

Proposition 5. *If $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ and $\theta \in \{\leq, <, =, >, \geq\}$ then $Q[\alpha(y)\ \theta\ k]$:– $R^p(y)$ has $\sharp\mathcal{P}$-data complexity.*

Proof (Sketch). Consider when θ is $=$. An instance of $\sharp$SUBSET-SUM is a set of integers $x_1, \ldots, x_n$ and our goal is to count the number of subsets $\emptyset \neq S \subseteq 1, \ldots, n$ such that $\sum_{s \in S} x_i = B$. We create the representation with schema $R(X; ; P)$ satisfying $\texttt{R} = \{(x_1; 0.5), \ldots, (x_n; 0.5)\}$, i.e. each tuple present with probability 0.5. Thus, $\mu(Q) * 2^n$ is number of such S. Showing hardness for other aggregate tess is straightforward.

Theorem 6. *If $\alpha \in \{\texttt{SUM}, \texttt{AVG}\}$ and $\theta \in \{=, \neq, \leq, <, >, \geq\}$ then let $Q[\alpha(y)\ \theta\ k]$ be a HAVING query if Q does not contain self-joins and is not α-safe, then Q has $\sharp\mathcal{P}$-data complexity. Further, there is an algorithm to decide if Q is α-safe in $\mathcal{P}$.*

If Q's skeleton is unsafe, then it is straightforward that Q has $\sharp\mathcal{P}$-hard data complexity. So we may assume that Q is safe, but not SUM-safe. Given any set of constants $y_1, \ldots, y_n$, let $q = \mathbf{sk}(Q)$ and for $i = 1, \ldots, n$, q_i be $q[y \to y_i]$. We construct an instance J such that the q_i are satisfied independently with multiplicity 1[3]. This allows us to construct the reduction above. A formal proof is in the appendix.

5 Related Work

Probabilistic relational databases have been discussed by Barbara et. al [**?**] and more recently Dalvi and Suciu [**?**], Ré et. al [**?**], Sen et. al [**?**] and Widom [**?**], though all omit HAVING style aggregation. Cheng et al. [**?**] and Desphande et. al [**?**] consider probabilistic databases resulting from sensor networks and handle continuous distributions with more general correlations, while we handle only the discrete case. In their settings, aggregate queries are effectively value aggregates over a singe relation.

In the OLAP setting [**?**] and streaming setting [**?**] give efficient algorithms for *value aggregation* in a model which is equivalent to the single table model and focuses on scaling such computation (e.g. using streaming techniques). In contrast, computing the AVG for predicate aggregates on a single table is already $\sharp\mathcal{P}$-Hard. Ross et al. [**?**] describe an approach to computing aggregates on a probabilistic database, by computing bounding intervals (e.g. the AVG is between $[5600, 5700]$). For more aggregate functions than we discuss, they show computing bounding intervals exactly is $\mathcal{NP}$-Hard but do not offer any results on the boundary of hardness.

A closely related work is Arenas et. al, [**?**] which considers the complexity of aggregate queries, similar to HAVING queries, over data which violates functional dependencies. Their semantic is greatest lower bound or least upper bound on the set of all minimal repairs, i.e. not probabilistic. They consider multiple predicates, which we do not. In this paper, we deal with more general types of value inconsistency.

6 Conclusion

In this paper we have examined the complexity of evaluating positive conjunctive queries with predicate aggregates over probabilistic databases. For each aggregate, we discussed a novel method based on computing the distribution of elements in a semiring to evaluate such queries. We proved that for conjunctive queries without self-joins our methods are optimal.

Acknowledgements This work was partially supported by NSF Grants IIS-0428168 and IIS-0454425.

[3] By multiplicity, we mean that for any $W \in \mathcal{W}_J$ and any q_i there is at most one valuation v such that $\mathbf{im}(v) \subseteq W$

7 Appendix: Full Proof for `COUNT(DISTINCT)`

Theorem 7 (Restatement of Thm. 3). *If Q is `COUNT(DISTINCT)`-safe then its evaluation problem is in $\mathcal{P}$.*

Proof. Since Q is `COUNT(DISTINCT)`-safe, then there is a safe plan P for the skeleton of Q. In the following let $P_1 \prec P_2$ denote the relationship P_1 is a descendant in P of P_2 (alternatively, containment). Let P_y be a subplan which satisfies $P_y = \pi_y^I(P') \prec P$ or $P_y = \pi_y^D(P') \prec P$. P_y is a safe plan, hence S-safe for $S = \mathbb{Z}_2$, i.e. the `EXISTS` algebra. For each t such that $\mathbf{supp}_{P_y}(t) \neq \emptyset$, we can write $\hat{\omega}^I_{P_y}(t) = (1-p, p)$, i.e. t is present with probability p. From this, create a marginal vector in $\mathbb{Z}^{k+1}$, as in `COUNT`, $\boldsymbol{m}^t$ such that $\boldsymbol{m}^t[0] = 1-p$ and $\boldsymbol{m}^t[1] = p$ and all other entries 0. Notice that if $t \neq t'$ then $t[y] \neq t[y']$. Informally, this means all y values are distinct "after" P_y.

Compute the remainder of P as follows: If P_0 is not a proper ancestor or descendant of P_y, then compute P_0 as if you were using the `EXISTS` algebra. To emphasize that P_0 should be computed this way, we shall denote the value of t under P_0 as $\hat{\omega}^J_{P_0,\texttt{EXISTS}}(t)$. Since P is `COUNT(DISTINCT)`-safe, any proper ancestor P_0 of P_y is of the form $P_0 = \pi^D_{-x} P_1$ or $P_0 = P_1 \bowtie P_2$. If $P_0 = \pi^D_{-x} P_1$ then $\hat{\omega}^J_{P_0}(t) = \perp_{t' \in \mathbf{supp}_{P_1}} \hat{\omega}^J_{P_1}(t)$; this is correct because the tuples we are combining are disjoint, so which values are present does not matter. Else, we may assume $P_0 = P_1 \bowtie P_2$ and without loss we assume that $P_y \prec P_1$, thus we compute:

$$\hat{\omega}^J_{P_1,\texttt{COUNT(DISTINCT)}}(t) = \hat{\omega}^J_{P_1}(t_1) \ast^{\cdot} \hat{\omega}^J_{P_2,\texttt{EXISTS}}(t_2)$$

This is an abuse of notation since we intend that $\hat{\omega}^J_{P_2} \in \mathbb{Z}_2$ is first mapped into $\mathbb{Z}_{k+1}$ and then the convolution is performed. Since we are either multiplying our lossy vector by the annihalator or the multiplicative identity, this convolution has the effect of multiplying by the probability that t is in P_2, since these events are independent this is exactly the value of their conjunction.

7.1 Complexity

Proposition 6 (Second Half of Prop. 4). *The following `HAVING` queries are $\sharp\mathcal{P}$-hard for $i \geq 1$:*

$$Q_{2,i}[\mathit{COUNT(DISTINCT}\, y)\ \theta\ k] \coloneq \mathrm{R}_1^p(x;y), \ldots, \mathrm{R}_i^p(x;y)$$

Proof. We start with $i = 1$. The hardness of Q_2 is shown by a reduction counting the number of set covers of size k. The input is a set of elements $U = \{u_1, \ldots, u_n\}$ and a family of sets $\mathcal{F} = \{S_1, \ldots, S_m\}$. A cover is a subset of $\mathcal{F}$ such that for each $u \in U$ there is $S \in \mathcal{S}$ such that $u \in S$. For each element $u \in U$, let $S_u = \{S \in \mathcal{F} \mid u \in S\}$, add a tuple $R(u; S; |S_u|^{-1})$ where $S \in S_u$. Every possible world corresponds to a set cover and hence, if W_k is the number of covers of size k then $\mu(Q) = W_k(\prod_{u \in U} |S_u|^{-1})$. Notice that if use the same reduction $i > 1$, we have that $\mu(Q) = W_k(\prod_{u \in U} |S_u|^{-i})$.

We show that if Q contains self-joins and is not `COUNT(DISTINCT)`-safe, then Q has $\sharp\mathcal{P}$ data complexity. First, we oberve a simple fact:

Proposition 7. *If Q is a `HAVING` query with an unsafe skeleton then Q has $\sharp\mathcal{P}$-hard data complexity. Further, if Q is connected and safe but not `COUNT(DISTINCT)`-safe then there must exist $x \neq y$ such that $\forall g \in \mathbf{goal}(Q)$, $x \in \mathbf{key}(g)$.*

Proof. We simply observe that the count of distinct variables is ≥ 1 exactly when the query is satisfied, which is $\sharp\mathcal{P}$-hard. The other aggregates follow easily. Since the skeleton of Q is safe, there is a safe plan for Q that is not `COUNT(DISTINCT)`-safe. This implies there is some projection independent π^I_{-y} on all variables.

Definition 16. *For a conjunctive query q let F^q_∞ be the least fixed point of $F^q_0, F^q_1, \ldots,$ where*

$$F^q_0 = \{x \mid \exists g \in \mathbf{goal}(Q) \text{ s.t. } \mathbf{key}(g) = \emptyset \ \wedge \ x \in \mathbf{var}(g)\}$$

and

$$F^q_{i+1} = \{x \mid \exists g \in \mathbf{goal}(Q) \text{ s.t. } \mathbf{key}(g) \subseteq F_i \ \wedge \ x \in \mathbf{var}(g)\}$$

Intuitevely, F^q_∞ is the set of variables "fixed" in a possible world.

Proposition 8. *If q is safe and $x \in F^q_\infty$ then there is a safe plan P such that $\pi_{-x} \in P$ and for all ancestors of π_{-x} they are either $\pi_{-z}P_1$ for some z or $P_1 \bowtie P_2$.*

Proof. Consider the smallest query q such that the propositon fails where the order is given by number of subgoals then number of variables variables. Let $x_1, \ldots, x_n$ according to the partial order $x_i \prec x_j$ if exists F^q_k such that $x_i \in F^q_k$ but $x_j \notin F^q_k$. If $q = q_1 q_2$ such that $x \in \mathbf{var}(q_1)$ and $\mathbf{var}(q_1) \cap \mathbf{var}(q_2) = \emptyset$ then P_1 satisfies the claim and $P_1 \bowtie P_2$ is a safe plan. Otherwise let P_1 be a safe plan for $q[x_1 \to a]$ for some fresh constant a. Since this has fewer variables P_1 satisfies the claim and $\pi_{-x}P_1$ is safe immediately from the definition.

We now define a set of rewrite rules $\Rightarrow$ which transform the skeleton and preserve hardness. We use these rewrite rules to show the following lemma:

Lemma 2. *Let $q = \mathbf{sk}(Q)$, if q is safe but Q is not `COUNT(DISTINCT)`-safe and there is some g such that $y \notin \mathbf{key}(g)$ and $y \notin f^q_\infty$ then Q has $\sharp\mathcal{P}$-hard data complexity.*

For notational convenience, we shall simple work with the skeleton of a `HAVING` query $Q[\alpha(y)\ \theta\ k]$ and assume that y is a distinguished variable.

1) $q \Rightarrow q[z \to c]$	if $z \in F^q_\infty$
2) $q \Rightarrow q_1$	if $q = q_1 q_2$ and $\mathbf{var}(q_1) \cap \mathbf{var}(q_2) = \emptyset$ and $y \in \mathbf{var}(q_1)$
3) $q \Rightarrow q[z \to x]$	if $x, z \in \mathbf{key}(g)$ and $z \neq y$
4) $q, g \Rightarrow q, g'$	if $\mathbf{key}(g) = \mathbf{key}(g')$, $\mathbf{var}(g) = \mathbf{var}(g')$ and $\mathbf{arity}(g) < \mathbf{arity}(g)'$
5) $q, g \Rightarrow q$	if $\mathbf{key}(g) = \mathbf{var}(g)$

We let $q \Rightarrow^* q'$ denote that q' is the result of any finite sequence of rewrite rules applied to q.

Proposition 9. *If $q \Rightarrow^* q'$ and q' has $\sharp\mathcal{P}$-hard data complexity, then so does q.*

Proof. For rule 1, we can simply restrict to instances where $z \to c$. For rule 2, if q_1 is hard then q is hard because we can fill out each relation in q_2 with a single tuple and use q to answer q_1. Similiarily, for rule 3 we can consider instances where $z = x$ so q will answer q_1. For rule 4, we apply the obvious mapping on instances (to the new subgoal). For rule 5, we fill out g with tuples of probability 1 and use this to answer q.

Proof (Prop. 2). By Prop. 7, there is some x such that $x \in \mathbf{key}(g)$ for any $g \in \mathbf{goal}(Q)$. Let $q = \mathbf{sk}(Q)$, we apply rule 1 and 2 to a fixed point, which removes any products. We then apply the rule 3 as $\forall z \neq y$, $q[z \to x]$. Thus, all subgoals have two variables, x and y. We then apply rule 4 to a fixed point and finally rule 5 to a fixed point. It is easy to see that all remaining subgoals are of the form $R^p(x; y)$ which is the hard pattern. Further, it is easy to see that $g \Rightarrow^* R^p(x; y)$.

We can now prove the main result:

Lemma 3. *If Q is a `HAVING` query without self-joins and Q is not `COUNT(DISTINCT)`-safe then the evaluation problem for Q is $\sharp\mathcal{P}$-hard.*

Proof. If q is unsafe, then Q has $\sharp\mathcal{P}$-hard data complexity. Thus, we may assume that q is safe but Q is not COUNT(DISTINCT)-safe. If Q contains $g \in \mathbf{goal}(Q)$ such that $y \in \mathbf{var}(g)$ but $y \notin \mathbf{key}(g)$ then Q has $\sharp\mathcal{P}$-hard data complexity by Lem. 2. Thus, we may assume that y appears only in key positions.

First apply rewrite rule 2, to remove any products and so we may assume Q is connected. If Q is a connected and $y \in \mathbf{key}(g)$ for every g then Q is COUNT(DISTINCT)-safe. Thus, there are at least two subgoals and one contains a variable x distinct from y call them g and g' respectively. Apply the rewrite rule 3 as $q[z \to x]$ for each $z \in \mathbf{var}(q) - \{x, y\}$. Using rules 4 and 5, we can then drop all subgoals but g, g' to obtain the pattern $R(x), S(x, y)$, which is hard.

8 Appendix: Full Proofs for `SUM` and `AVG`

8.1 `AVG` hardness

Definition 17. *Given a set of nonnegative integers $a_1, \ldots, a_n$, the ♯NONNEGATIVE SUBSET-AVG problem is to count the number of non-empty subsets $S \subseteq 1, \ldots, n$ such that $\sum_{s \in S} a_s |S|^{-1} = B$ for some fixed integer B.*

Proposition 10. *♯NONNEGATIVE SUBSET-AVG is ♯$\mathcal{P}$-hard.*

Proof. We first observe that if we allow aribitrary integers, then we can reduce any ♯NONNEGATIVE SUBSET-SUM with $B = 0$, which is ♯$\mathcal{P}$-hard. Since the summation of any set is 0 if and only if their average is 0. Thus, we reduce from this unrestricted version of the problem. Let $B = \min_i a_i$ then we simply make $a'_i = a_i + B$, now all values are positive, we then ask if the average is B. For any set S we have :

$$\sum_{s \in S} a'_s |S|^{-1} = \sum_{s \in S} (a_s + B)|S|^{-1} = \sum_{s \in S} (a_s + B)|S|^{-1} = \sum_{s \in S} |S|^{-1} a_s + B$$

Thus, it is clear that the average is satisfied only when $\sum_{s \in S} a_s = 0$.

8.2 Proof of Thm. 6

It is sufficient to show the following lemma:

Lemma 4. *Let $q = \mathbf{sk}(Q)$, if If q is safe, but Q is not `SUM`-safe then there is an instance I then for any set of values $y_1, \ldots, y_n$ let $q_i = q[y \to y_i]$ and $S \subseteq 1, \ldots, n$ we have $\mu(\bigwedge_{s \in S}^n q_s) = \prod_{s \in S} \mu(q_s) = 2^{-|S|}$. Further, on any world W and q_i there is a single valuation v for q_i such that $\mathbf{im}(q_i) \subseteq W$.*

This lemma that we can always construct the same distribution used in Prop. 5.

Proof. We observe that $y \notin F_\infty^q$ else there would be a `SUM`- and `AVG`-safe plan by Prop. 8. Now consider the rewriting $q[x \to \text{'a'}]$ for any $x \in F_\infty$ and $q[x \to y]$ if $x \notin F_\infty$. Thus, in any subgoal $y = \mathbf{var}(g)$. Pick one and add each y_1 value with probability $\frac{1}{2}$ independently. Notice that every relation either contains y_i in each tuple or the constant a. It is not hard to see that in any world that all valuations are distinct.