The VLDB Journal (2009) 18:1091–1116
DOI 10.1007/s00778-009-0151-4

SPECIAL ISSUE PAPER

The trichotomy of HAVING queries on a probabilistic database

Christopher Ré · Dan Suciu

Received: 16 September 2008 / Revised: 31 March 2009 / Accepted: 1 May 2009 / Published online: 15 July 2009

Abstract We study the evaluation of positive conjunctive queries with Boolean aggregate tests (similar to HAVING in SQL) on probabilistic databases. More precisely, we study conjunctive queries with predicate aggregates on probabilistic databases where the aggregation function is one of MIN, MAX, EXISTS, COUNT, SUM, AVG, or COUNT(DISTINCT) and the comparison function is one of $=, \neq, \geq, >, \leq$, or $<$. The complexity of evaluating a HAVING query depends on the aggregation function, α, and the comparison function, θ. In this paper, we establish a set of trichotomy results for conjunctive queries with HAVING predicates parametrized by (α, θ). For such queries (without self-joins), one of the following three statements is true: (1) the exact evaluation problem has $\mathcal{P}$-time data complexity. In this case, we call the query *safe*. (2) The exact evaluation problem is $\sharp\mathcal{P}$-hard, but the approximate evaluation problem has (randomized) $\mathcal{P}$-time data complexity. More precisely, there exists an FPTRAS for the query. In this case, we call the query *apx-safe*. (3) The exact evaluation problem is $\sharp\mathcal{P}$-hard, and the approximate evaluation problem is also hard. We call these queries *hazardous*. The precise definition of each class depends on the aggregate considered and the comparison function. Thus, we have queries that are (MAX, $\geq$)-safe, (COUNT, $\leq$)-apx-safe, (SUM, $=$)-hazardous, etc. Our trichotomy result is a significant extension of a previous dichotomy result for Boolean conjunctive queries into safe and not safe. For each of the three classes we present novel techniques. For safe queries, we describe an evaluation algorithm that uses random variables over semirings. For apx-safe queries, we describe an FPTRAS that relies on a novel algorithm for generating a random possible world satisfying a given condition. Finally, for hazardous queries we give novel proofs of hardness of approximation. The results for safe queries were previously announced (in Ré, C., Suciu, D. Efficient evaluation of. In: DBPL, pp. 186–200, 2007), but all other results are new.

Keywords Probabilistic databases · Query evaluation · Sampling algorithms · Semirings · Safe plans

This work was partially supported by NSF Grants IIS-0513877 and IIS-0713576.

C. Ré (✉) · D. Suciu
Department of Computer Science and Engineering,
University of Washington, Seattle, USA
e-mail: chrisre@cs.washington.edu

D. Suciu
e-mail: suciu@cs.washington.edu

1 Introduction

We study the complexity of evaluating aggregate queries on probabilistic databases. Our motivation is to manage data produced by integration applications, e.g., data from object reconciliation [24,52,53] or information extraction [8,23,27,35]. 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 uncertainty is captured using probabilities has attracted renewed interest [10,15,19,51].

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 the 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 that we call HAVING queries. Prior art [6,26] has defined a semantic for *value aggregation* that returns the expected value of an aggregate query (e.g., the expected MAX price) and has demonstrated its utility

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

Fig. 1 A probabilistic database with a Profit relation that contains the profit an analyst forecasts for each item sold. Prior Art [26] has considered a semantic similar to the query in **a**, which returns the expected value of an aggregate. In contrast, we study queries similar to **b** which computes the probability of a HAVING style predicate, e.g., that the SUM of profits exceeds a value (here, 0.0)

for 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 Figure 1 illustrates a probabilistic database that contains a single relation, Profit. Intuitively, a tuple in Profit records the profit that one of our analysts forecasts if we continue to sell that item. We are not certain in our prediction, and so Profit records a confidence with each prediction. For example, Alice is quite sure that we will lose money if we continue selling widgets; this is captured by the tuple $(Widget, Alice, \$-99K, 0.99)$ in Profit. Intuitively, 0.99 is the marginal probability of the fact $(Widget, Alice, \$-99k)$.

An example of a value aggregate is shown in Fig. 1a. In this approach, the answer to an aggregation query is the *expected value of the aggregate function*. Using linearity of expectation, the value of the query in Fig. 1a is $100\text{M} \times 0.01 + -99K \times 0.99 \approx 900$ K. Intuitively, this large value suggests that we should continue selling widgets because we expect to make money. A second approach (that we propose and study in this paper), is akin to HAVING style aggregation in standard SQL. An example is the query in Fig. 1b that intuitively says: "*What is the probability that we will make a profit?*". The answer to this query is the probability that the value of the SUM is greater than 0. Here, the answer is only 0.01: this small probability tells us that we should stop selling widgets or risk going out of business.

Our technical starting point is the observation that we can evaluate a query q with an aggregate α on a deterministic database using a two step process: (1) annotate the database with values from some semiring, S_α, e.g., if $\alpha = \texttt{COUNT}$, then we can take S_α to be the natural numbers, and (2) propagate these annotations during query processing (using the rules in Green et al. [21]). In this scheme, each tuple output by the query is annotated with a value in the semiring S_α that is exactly the *value* of the aggregate, e.g., the COUNT of the tuples returned by q. Thus, it is easy to check if the HAVING query is true: simply test the predicate aggregate on the value returned by the query, e.g., is the SUM returned by the query greater than 0? If the answer is yes, return true.

To evaluate aggregate queries on probabilistic databases, we generalize this approach. On a probabilistic database, the output of an aggregate query Q is described by a *random variable*, denoted s_Q, that takes values in S_α. A HAVING query Q whose predicate is, say, $\texttt{COUNT}(*) < k$, can be computed over a probabilistic database in two stages: (1) compute the distribution of the random variable, s_Q; and (2) apply a *recovery function* that computes the probability that $s_Q < k$, i.e., sum over all satisfying values of s_Q. The cost of this algorithm depends on the space required to represent the random variable s_Q, which may be exponential in the size of the database. This cost is prohibitively high for many applications.[1] In general, this cost is unavoidable, as prior art has shown that for SELECT-PROJECT-JOIN (SPJ) queries (without HAVING), computing a query's probability is $\sharp\mathcal{P}$-Complete [2][12,20].

Although evaluating general SPJ queries on a probabilistic database is hard, there is a class of SPJ queries (called *safe queries*) that can be computed efficiently and exactly [12,40]. A safe query has a relational plan P, called a *safe plan*, that is augmented to compute the output probability of a query by manipulating the marginal probabilities associated with tuples. The manipulations performed by the safe plan are standard multiplications and additions. These manipulations are correct because the safe plan "knows" the correlations of the tuples that the probabilities represent, e.g., the plan only multiplies probabilities when the events are independent. To generalize safe plans to compute HAVING queries, we provide analogous operations for semiring random variables. First, we describe *marginal vectors* that are analogous to marginal probabilities: a marginal vector is a succinct, but lossy, representation of a random variable. We then show that

[1] A probabilistic database represents a distribution over standard, deterministic instances, called *possible worlds* [18]. A probabilistic database with n tuples can encode 2^n possible worlds, i.e., one for each subset of tuples. We defer to Sect. 2.1 for more details.

[2] $\sharp\mathcal{P}$ defined by Valiant [49] is the class of functions that contains the problem of counting the number of solutions to $\mathcal{NP}$-Hard problems (e.g., $\sharp$3-SAT). Formally, we mean here that there is a polynomial reduction from a $\sharp\mathcal{P}$-Hard problem, and to any problem in $\sharp\mathcal{P}$. Since technically, the query evaluation problem itself is not in $\sharp\mathcal{P}$.

the operation analogous to multiplying marginal probabilities is a kind of *semiring convolution*. Informally, we show that substituting multiplications with convolutions is correct precisely when the plan is safe.

As we show, the running time of safe plans with convolutions is proportional to the number of elements in the semiring, S_α. Thus, to compute `HAVING` queries with an aggregate α efficiently, we need S_α to be small, i.e., S_α should contain at most polynomially many elements in the size of the instance. This condition is met when the aggregate α is one of {`EXISTS`, `MIN`, `MAX`, `COUNT`}. For $\alpha \in$ {`SUM`,`AVG`, `COUNT(DISTINCT)`}, the condition is not met. In these cases, our algorithm is efficient only for a restricted type of safe plans that we call α-*safe*. For α-safe plans, a `HAVING` query with α can be computed efficiently and exactly. Further, we show that α-safe plans capture tractable exact evaluation for queries without self-joins.[3] More precisely, for each aggregate α above, there is a dichotomy for queries without self-joins: either (1) Q is α-safe, and so has a $\mathcal{P}$-time algorithm, or (2) Q is not α-safe and evaluating Q exactly is $\sharp\mathcal{P}$-Hard. Further, we can decide whether a query is α-safe in $\mathcal{P}$-time. The techniques for exact evaluation were described in the preliminary version of this paper [43].

Exact evaluation is the gold standard, but in many applications, *approximately* computing probabilities suffices. For example, if the input probabilities are obtained heuristically, then computing the precise value of the output probability may be overkill. Alternatively, even if the probabilities are obtained precisely, a user may not care about the difference between a query that returns a probability score of 0.9 versus 0.90001. Leveraging this observation, we show that there are some queries that can be efficiently *approximated*, even though they are not α-safe (and so cannot be computed exactly). More precisely, we study when there exists a *Fully Polynomial Time Randomized Approximation Scheme* (FPTRAS) for approximating the value of a `HAVING` query.[4] Our key result is that there is a second dichotomy for approximate evaluation for queries without self-joins: either (1) an approximation scheme in this paper can approximate a `HAVING` query efficiently, or (2) there is no such efficient approximation scheme. Interestingly, we show that the introduction of self-joins *raises* the complexity of approximation: we show a stronger inapproximability result for queries involving self-joins.

In general, the complexity of evaluating a `HAVING` query Q depends on the predicate that Q uses. More precisely, the hardness depends on both the aggregate function, α, and the comparison function, θ, which together are called an *aggregate-test pair*, e.g., in Fig. 1b the aggregate-test pair is (`COUNT`, $>$). For many such aggregate test pairs (α, θ), we show a *trichotomy result*: For `HAVING` queries using (α, θ) without self-joins over tuple-independent probabilistic databases, exactly one of the following three statements is true: (1) The exact evaluation problem has $\mathcal{P}$-time data complexity. In this case we call the query *safe*. (2) The exact evaluation problem is $\sharp\mathcal{P}$-hard, but the approximate evaluation problem has (randomized) $\mathcal{P}$-time data complexity (there exists an FPTRAS to evaluate the query). In this case, we call the query *apx-safe*. (3) The exact evaluation problem is $\sharp\mathcal{P}$-hard and the approximate evaluation problem is also hard (no FPTRAS exists). We call these queries *hazardous*. It is interesting to note that the third class is empty for `EXISTS`, which are the class extensively studied by prior work [12]: That is, all Boolean conjunctive queries have an efficient approximation algorithm.

A key step in many Monte–Carlo-style approximation algorithms based on sampling (including those in this paper) is randomly generating instances (called possible worlds). Computing a random possible world is straightforward in a probabilistic database: we select each tuple with its corresponding marginal probability taking care never to select two disjoint tuples. However, to support efficient techniques like *importance sampling* [29], we need to do something more: we need to generate a random possible world from the set of worlds that *satisfy a constraint that is specified by an aggregate query*. For example, we need to generate a random world, $\tilde{W}$, such that the `MAX` price returned by a query q on $\tilde{W}$ is equal to 30. We call this the *random possible world generation problem*. Our key technical result is that when q is safe (without aggregation) and the number of elements in the semiring S is small, then we can solve this problem efficiently, i.e., with randomized polynomial time data complexity. The novel technical insight is that *we can use safe plans as a guide to sample the database*. This use is in contrast to the traditional use for safe plans of computing query probabilities exactly. We apply our novel sampling technique to provide an FPTRAS to approximately evaluate some `HAVING` queries that have $\sharp\mathcal{P}$-hard exact complexity. Thus, the approaches described in this paper can efficiently answer strictly more queries than our previous, exact approach (albeit only in an approximate sense).

Contributions and outline

We study conjunctive queries with `HAVING` predicates on common representations of probabilistic databases [4,41,51] where the aggregation function is one of `EXISTS`, `MIN`, `MAX`, `COUNT`, `SUM`, `AVG`, or `COUNT(DISTINCT)`; and the aggregate test is one of $=, \neq, <, \leq, >$, or $\geq$. In Sect. 2, we formalize `HAVING` queries, our choice of representation,

[3] A self-join is a join between a relation and itself. The query $R(x, y), S(y)$ does not have a self-join, but $R(x, y), R(y, z)$ does.

[4] An FPTRAS can be thought of as a form of sampling that is guaranteed to rapidly converge and so is efficient. We refer to Definition 19 for formal details.

```
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

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

(b) Extended Syntax (Not Boolean)

$Q[\texttt{COUNT(DISTINCT}\ r) \geq 2]$:–
  $\texttt{MovieMatch}(t, \text{‘Fletch’})$,
  $\texttt{Reviewer}(-, r, t)$

(c) Syntax of this paper

Fig. 2 A translation of the query “*Which movies have been reviewed by at least two distinct reviewers?*” into **a** SQL; **b** an extended syntax of this paper, which is not Boolean; and **c** the syntax of this paper, which is Boolean and is a HAVING query

and define efficient evaluation. In Sect. 3, we review the relevant technical background (e.g., semirings and safe plans). In Sect. 4, we give our main results for exact computation: 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.
- We can decide in polynomial time (in the size of Q) if Q is α-safe.

In Sect. 5, which is completely new, we state and solve the problem of generating a random possible world when the query defining the constraint is safe. In Sect. 6, we discuss approximation schemes for queries that have $\alpha \in \{$MIN,MAX, COUNT, SUM$\}$. The hardness of an approximation algorithm for a HAVING query depends on the aggregate, α, but also on the predicate test, θ. We show:

- If Q is (α, θ)-apx-safe then Q has an FPTRAS.
- If Q has no self-joins and is not (α, θ)-apx-safe then, Q does not have an FPTRAS and is (α, θ)-hazardous.
- We can decide in polynomial time (in the size of Q) if Q is (α, θ)-apx-safe.

We show that the trichotomy holds for all combinations of α and $\theta \in \{=, \leq, <, \geq, >\}$, but leave open the case of COUNT and SUM with either of $\{\geq, >\}$. Additionally, we also show that queries with self-joins belong to a complexity class that is believed to be as hard to approximate as any problem in $\sharp\mathcal{P}$. This suggests that the complexity for HAVING query approximation is perhaps more subtle than for Boolean queries.

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.

2.1 Semantics

We consider the aggregate functions EXISTS, 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 for $i = 1, \ldots, m$, g_i is a distinct, positive extensional database predicate (EDB), that is, a relational symbol.[5] 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,[6] $\theta \in \{=, \neq, <, \leq, >, \geq\}$, 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 and is denoted $\mathbf{sk}(Q) = q$. In the above syntax, θ is called the **predicate test**; k is called the **predicate operand**; and the pair (α, θ) is called an **aggregate test**.

Figure 2a 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. 2b. The translated query is not a Boolean HAVING query because it has a head variable (m). In this paper, we discuss only Boolean HAVING queries. As is standard, to study the complexity of non-Boolean queries, we can substitute constants for head variables. For example, if we substitute ‘Fletch’ for m, then the result is Fig. 2c which is a Boolean HAVING query.

Definition 2 Given a HAVING query $Q[\alpha(y)\ \theta\ k]$ and a world W (a standard relational instance), we define $\mathcal{Y}$ to be the multiset of values $v(y)$ where y is distinguished variable in Q and v is a valuation of $q = \mathbf{sk}(Q)$ that is contained in W. In symbols,

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

[5] Since all relational symbols are distinct, HAVING queries do not contain self-joins: $q = R(x, y), R(y, z)$ has a self-join, while $R(x, y), S(y)$ does not.

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

Title	Matched	P	
‘Fletch’	‘Fletch’	0.95	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(ReviewID, Reviewer ; ReviewTitle ; P)

Fig. 3 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*

Here, $\mathbf{im}(v) \subseteq W$ denotes that image of $\mathbf{sk}(Q)$ under the valuation v is contained in the world W. 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.

In the above definition, we follow SQL semantics and require that $\mathcal{Y} \neq \emptyset$ in order to say that $W \models Q$. For example, $Q[\texttt{COUNT}(*) < 10]$:–$R(x)$ is false in SQL if $R^W = \emptyset$, i.e., the interpretation of R in the world W is the empty table. This, however, is a minor technicality and our results are unaffected by the alternate choice that $\texttt{COUNT}(*) < 10$ is true on the empty database.

2.2 Probabilistic databases

In this paper, we use probabilistic databases described in the block-independent disjoint (BID) representation [41,42] that generalizes many representations in the literature including p-?-sets and p-or-sets [22], ?- and x-relations [45], and tuple independent databases [12,33]. The BID representation is essentially the same as Barbara et al. [4].

Syntax. We think of a BID schema as a relational schema where the attributes are partitioned into three disjoint sets. We write a BID schema as $\mathrm{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ where the sets are separated by semicolons. Here, $\boldsymbol{K}$ is a set of attributes called the **possible worlds key**; $\boldsymbol{A}$ is a set of attributes called the set of **value attributes**; and P is a single, distinguished attribute that stores the marginal probability of the tuple called the **probability attribute**. The values of $\boldsymbol{K}$ and $\boldsymbol{A}$ come from some discrete domain, and the values in the attribute P are numbers in the interval (0, 1]. For example, the BID schema in Fig. 1 is Profit(Item;Forecaster,Profit;P): {Item} is the possible worlds key, {Forecaster,Profit} is the set of value attributes, and P is the probability attribute. Also pictured in Fig. 1 is an instance of this schema.

Semantics. We think of an instance of a BID schema as representing a distribution over instances called *possible worlds*. The schema of these possible worlds is $R(\boldsymbol{K}, \boldsymbol{A})$, i.e., the same schema without the attribute P. Let J be an instance of a BID schema. We denote by $t[\boldsymbol{K}\boldsymbol{A}P]$ a tuple in J, emphasizing its three kinds of attributes, and call $t[\boldsymbol{K}\boldsymbol{A}]$, its projection on the $\boldsymbol{K}\boldsymbol{A}$ attributes, a *possible tuple*. Define a *possible world*, W, to be any instance of $R(\boldsymbol{K}, \boldsymbol{A})$ consisting of possible tuples such that $\boldsymbol{K}$ is a key in W. Note that the key constraints do not hold in the BID instance J, but must hold in any possible world W. Let $\mathcal{W}_J$ be the set of all possible worlds associated to J. In Fig. 1, one possible world W_1 is $R^{W_1} = \{(\text{Widget,Alice}, -99k), (\text{Whatsit, Alice}, 1M)\}$.

We define the semantics of BID instances only for *valid* instances, which are BID instances such that the values in P can be interpreted as a valid probability distribution, i.e., such that for every tuple $t \in R^J$ in any BID relation $\mathrm{R}(\boldsymbol{K}; \boldsymbol{A}; P)$ the inequality $\sum_{s\in \mathrm{R}:s[\boldsymbol{K}]=t[\boldsymbol{K}]} s[P] \leq 1$ holds. A valid instance J defines a finite probability space $(\mathcal{W}_J, \mu_J)$. First note that any possible tuple $t[\boldsymbol{K}\boldsymbol{A}]$ can be viewed as an event in the probability space $(\mathcal{W}_J, \mu_J)$, namely the event that a world contains $t[\boldsymbol{K}\boldsymbol{A}]$. Then we define the semantics of J to be the probability space $(\mathcal{W}_J, \mu_J)$ such that (a) the marginal probability of any possible tuple $t[\boldsymbol{K}\boldsymbol{A}]$ is $t[P]$, (b) any two tuples from the same relation $t[\boldsymbol{K}\boldsymbol{A}]$, $t'[\boldsymbol{K}\boldsymbol{A}]$ such that $t[\boldsymbol{K}] = t'[\boldsymbol{K}]$ are *disjoint events* (or *exclusive events*), and (c) for any set of tuples $\{t_1, \ldots, t_n\}$ such that all tuples from the same relation have distinct keys, the events defined by these tuples are independent. From above, we see that $\mu(W_1) = 0.99$.

Example 2 The data in Fig. 3 shows an example of a BID database that stores data from integrating extracted movie reviews from USENET with a movie database from IMDB. The MovieMatch table is uncertain because it is the result of an automatic matching procedure (or fuzzy-join [9]). For example, the probability a review title ‘Fletch’ matches a movie titled ‘Fletch’ is very high (0.95), but it is not certain (1.0) 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*. That is, we have extracted the title from 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 tuples, say $t[KA]$ and $t'[KA]$, are disjoint if $t[K] = t'[K]$ and $t[A] \neq t'[A]$.

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* [12], which is also known as a *?-table* [39] or a *p-?-table* [22].

Finally, we generalize to probabilistic databases that contain many BID tables in the standard way: tuples in distinct tables are independent.

Query semantics. Users write queries on the possible worlds schema, i.e., their queries do not explicitly mention the probability attributes of relations. In this paper, all queries are Boolean so the answer to a query is a probability score (the marginal probability that the query is true). We define this formally:

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:

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

In general, for a Boolean conjunctive query q, we write $\mu_J(q)$ to denote the marginal probability that q is true.

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

2.3 Notions of complexity for HAVING queries

In the database tradition, we would like to measure the data complexity [50], i.e., treat the query as fixed, but allow the data to grow. This assumption makes sense in practice because the query is generally orders of magnitude smaller than the size of the database. Hence, a running time for query evaluation of $O(n^{f(|Q|)})$ where $|Q|$ is the size of a conjunctive query Q is $\mathcal{P}$-time. In our setting, this introduces a minor technical problem: by fixing a HAVING query q, we also fix k (the predicate operand); this means that we should accept a running time $n^{f(k)}$ as efficient. Clearly this is undesirable: because k can be large.[7] For example, $Q[\text{SUM}(y) > 200]$:–$R(x, y)$. 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 part 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 a BID representation J *and* a parameter $k > 0$, calculate $\mu_J(Q)$ where $Q[\alpha(y)\ \theta\ k]$ is such that $\mathbf{sk}(Q) = q$.

The technical problem that we address in this work is the complexity of the query evaluation problem. Later, we will see that the query evaluation problem for the query in Example 3 is hard for $\sharp\mathcal{P}$, and moreover, that this is the general complexity for all HAVING queries.

3 Preliminaries

We review some basic facts about semirings (for a reference see Lang [34]). Then, we introduce random variables over semirings.

3.1 Background: queries on databases with semiring annotations

In this section, we review material from Green et al. [21] that tells us how to compute queries on a database whose tuples are annotated with elements of a semiring. To get there, we need some classical definitions.

A **monoid** is a triple $(S, +, 0)$ where S is a set, $+$ is an associative binary operation on S, and 0 is the identity of $+$, i.e., $s + 0 = 0$ for each $s \in S$. For example, $S = \mathbb{N}$ (the natural numbers) with addition is the canonical example of a monoid.

A **semiring** is a structure $(S, +, \cdot, 0, 1)$ where $(S, +, 0)$ forms a commutative monoid with identity 0; $(S, \cdot, 1)$ is a monoid with identity 1; $\cdot$ distributes over $+$, i.e., $s \cdot (t + u) = (s \cdot t) + (s \cdot u)$ where $s, t, u \in S$; and 0 annihilates S, i.e., $0 \cdot s = 0$ for any $s \in S$.

A **commutative semiring** is one in which $(S, \cdot, 1)$ is a commutative monoid. As is standard, we abbreviate either structure with the set S when the associated operations and distinguished constants are clear from the context. In this paper, all semirings will be commutative semirings.

Example 4 [Examples of Semirings] For an 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, denoted $\mathbb{B}$. For $k = 1, 2, \ldots,$ another set of semirings we consider are $\mathbb{S}_k = (\mathbb{Z}_k, +_k, \cdot_k, 0, 1)$ where $+_k(x, y) = \min(x + y, k)$ and $\cdot_k = \min(xy, k)$ where addition and multiplication are in $\mathbb{Z}$.

The idea is that database elements will be annotated with elements from the semiring (defined next) and then these

[7] If we fix the query than k is assumed to be a constant, and so we can take even double exponential time in k. Thus, we would like to take k as part of the input.

annotations will be propagated during query processing. For us, the important point is that aggregation queries can be viewed as doing computation in these semirings.

Definition 5 Given a commutative semiring S and a Boolean conjunctive query $q = g_1, \ldots, g_n$, an annotation is a set of functions indexed by subgoals such that for $i = 1, \ldots, n$, τ_{g_i} is a function from tuples that unify with g_i to S. We denote the set of annotation functions with τ.

Remark 2 In the above definition, we restrict τ to assigning values to tuples that unify with g_i, since g_i may incorporate selections. For example, if $g_i = R(x, \text{'a'})$ then τ does not need to assign values to tuples whose second component is 'b'. Implicitly, τ should assign all such tuples 0.

We now define the syntax of relational plans and some related notions. This is completely standard, except for the minor issue that we view projection as removing attributes instead of the traditional view of projection as keeping attributes.

Definition 6 (Query Plan Syntax)

- a **plan** P is inductively defined as (1) a single subgoal that may include selections, (2) $\pi_{-x} P_1$ if P_1 is a plan and x is a variable, and (3) $P_1 \bowtie P_2$ if P_1, P_2 are plans.
- $\mathbf{var}(P)$, the variables output by P, is defined inductively as (1) $\mathbf{var}(g)$, the variables in the subgoal g, if $P = g$; (2) $\mathbf{var}(\pi_{-x} P) = \mathbf{var}(P) - \{x\}$; and (3) $\mathbf{var}(P_1 \bowtie P_2) = \mathbf{var}(P_1) \cup \mathbf{var}(P_2)$.
- $\mathbf{goal}(P)$, the set of subgoals in P, is defined inductively as (1) $\mathbf{goal}(g) = \{g\}$; (2) $\mathbf{goal}(\pi_{-x} P_1) = \mathbf{goal}(P_1)$; and (3) $\mathbf{goal}(P_1 \bowtie P_2) = \mathbf{goal}(P_1) \cup \mathbf{goal}(P_2)$.

A graphical example query plan is shown in Fig. 4a along with its description in the above syntax.

We view relational plans as computing relational tuples that are annotated with elements of a semiring (following Green et al. [21]). To be precise, fix a domain $\mathbb{D}$, and denote the **value** of a plan P on a deterministic instance W as ω_P^W, which is a function $\mathbb{D}^{|\mathbf{var}(P)|} \to S$. Informally, the value of a plan maps each standard tuple returned by the plan to an element of the semiring S. We define ω_P^W inductively:

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

An example of a plan computing a value in a semiring is shown in Fig. 4a. The value of the plan in the figure is 6:

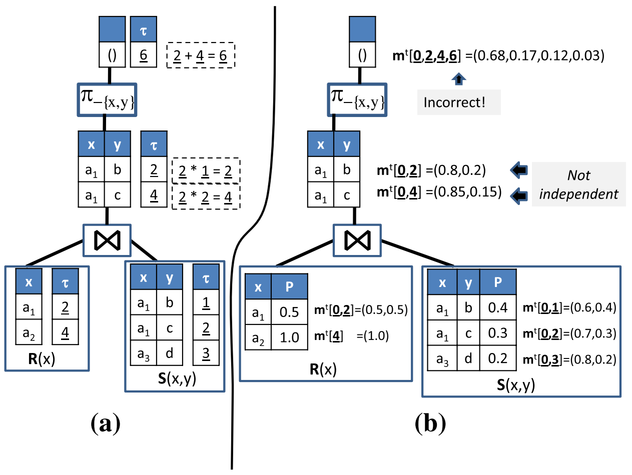


Fig. 4 **a** This is a query plan $P = \pi_{-x}(\pi_{-y}(R(x) \bowtie S(x, y)))$ for the query $q = R(x), S(x, y)$ over some database annotated in $\mathbb{N}$. The value of the query is $q(W, \tau) = 6$. **b** This is an extensional plan (Definition 12) for P $(\pi^I_{-x}(\pi^I_{-y}(R(x) \bowtie S(x, y)))$. This plan is not safe, since intermediate values may be neither independent nor disjoint. Thus, the extensional value computed by this plan is not the correct marginal probability of the query. For readability, we underline elements of the semiring

Since the plan is Boolean, it returns the empty tuple which is annotated with 6, more succinctly, $\omega_P^W() = 6$.

For a standard conjunctive query q, there may be many distinct, but logically equivalent, relational plans to compute q. Green et al. [21] show that ω_P^W *does not depend* on the particular choice of logically equivalent plan P for q. In turn, this justifies the notation $q(W, \tau)$, as the value of a conjunctive query q on a deterministic instance W under annotation τ. Formally, we define this value as $q(W, \tau) \stackrel{\text{def}}{=} \omega_P^W()$ where P is any plan for q and where ω_P^W is applied to the empty tuple. This notion is well defined precisely because the value of q does not depend on the choice of plan, P. When τ is clear from the context, we drop it and write simply $q(W)$ to denote the value of q on a world W.

3.2 Background: random variables on semirings

In this section, we extend the idea of semirings on a standard database to probabilistic databases. Intuitively, in each possible world, every tuple is annotated with a (potentially different) semiring element. Hence, we think of each tuple as being associated with a *semiring random variable* (defined formally below). A naive representation of these random variables can be large, which motivates us to define an efficient (small) representation called *marginal vectors* (in full analogy with marginal probabilities). In addition, we define (efficient) operations on these marginal vectors that are fully analogous with multiplying and adding marginal probabilities. In the remainder of this section, we fix a BID instance

J, and denote by $(\mathcal{W}, \mu)$ the distribution on possible worlds induced by J (Sect. 2.1).

Definition 7 Given a semiring S, an **S-random variable**, r, is a function $r : \mathcal{W} \to S$. Given two S-random variables r, t then $r + t$ and $r \cdot t$ denote random variables defined in the obvious way:

$$(r + t)(W) = r(W) + t(W) \text{ and } (r \cdot t)(W) = r(W) \cdot t(W)$$

We write $r = s$ as a shorthand for the event that the random variable r takes value s. We denote the probability of this event as $\mu(r = s)$. More precisely, $\mu(r = s) = \mu(\{W \in \mathcal{W} \mid r(W) = s\})$. Two basic notions on random variables are independence and disjointness:

Definition 8 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 $s_1, \dots, s_n \in S$, we have

$$\mu\left(\bigwedge_{i \in N} r_i = s_i\right) = \prod_{i \in N} \mu(r_i = s_i)$$

We say that R is **disjoint** if for any $i \neq j$ we have:

$$\mu((r_i \neq 0) \wedge (r_j \neq 0)) = 0$$

If r and t are two disjoint random variables[8] then $\mu(r = 0 \vee t = 0) = \mu(r = 0) + \mu(t = 0) - 1$.

To represent a single S-random variable, we may need space as large as the number of possible worlds ($|\mathcal{W}|$). This can be exponential in the size of the database J, and so, is prohibitive for most applications. We now define an alternative representation called *marginal vectors* that have size proportional to the size of the semiring, i.e., $|S|$.

Definition 9 Given a random variable r on S, the **marginal vector** (or simply, the marginal) of r is denoted $\boldsymbol{m}^r$ and is a real-valued vector indexed by S defined by $\forall s \in S$ $\mu(r = s) = \boldsymbol{m}^r[s]$.

Two simple facts immediate from the definition are $\forall s \in S$ $m^r[s] \geq 0$ (all entries are positive) and $\sum_{s \in S} m^r[s] = 1$ (total probability). We use the following notation $\boldsymbol{m}^r[s_1, \dots, s_k]$ where $s_1, \dots, s_k$ are semiring elements to be a shorthand for the tuple of marginal probabilities $(\boldsymbol{m}^r[s_1], \dots, \boldsymbol{m}^r[s_k])$.

Marginal vectors for semiring random variables are the analog of marginal probabilities for Boolean events: they are a means to write down a simple, succinct (but lossy) representation of a random variable. In the case of a Boolean semiring (i.e., $\mathbb{B} = (\{0, 1\}, \max, \min, 0, 1)$), a random variable r is an event that is true (when $r = 1$) or false (when $r = 0$). Suppose that the marginal probability that r is true is p_r (and so it is false with probability $1 - p_r$). Then, the marginal vector has two entries one for each of the semiring elements, 0 and 1:

$$\boldsymbol{m}^r[0] = 1 - p_r \text{ and } \boldsymbol{m}^r[1] = p_r$$

If r and t are independent Boolean events, then their conjunction $r \wedge t$ has marginal probability given by the simple formula $\Pr[r \wedge t] = \Pr[r]\Pr[t]$. We generalize the idea of multiplying marginal probabilities to marginal vectors of semiring elements; the resulting operation is called a *monoid convolution*. In full analogy, when when r, t are disjoint semiring random variables, we introduce a *disjoint operation* that is analogous to the rule $\Pr[r \vee t] = \Pr[r] + \Pr[t]$ for disjoint Boolean events.

Definition 10 Given a monoid $(S, +, 0)$, the **monoid convolution** is a binary operation on marginal vectors denoted $\oplus$. For any marginals $\boldsymbol{m}^r$ and $\boldsymbol{m}^t$ we define the s-entry (for $s \in S$) of $\boldsymbol{m}^r \oplus \boldsymbol{m}^t$ by the equation:

$$(\boldsymbol{m}^r \oplus \boldsymbol{m}^t)[s] \overset{\text{def}}{=} \sum_{i,j : i+j=s} \boldsymbol{m}^r[i]\boldsymbol{m}^t[j]$$

That is, the sum ranges over all pairs of elements from the semiring S whose sum (computed in the semiring S) is exactly s. We emphasize that since the entries of the marginal vectors are in $\mathbb{R}$, the arithmetic operations on $\boldsymbol{m}$ in the above equation are performed in $\mathbb{R}$ as well.

The **disjoint operation** for $(S, 0, +)$ is denoted $\boldsymbol{m}^r \coprod \boldsymbol{m}^t$ and is defined by

$$\begin{aligned} &\text{if } s \neq 0\ (\boldsymbol{m}^r \textstyle\coprod \boldsymbol{m}^t)[s] \overset{\text{def}}{=} \boldsymbol{m}^r[s] + \boldsymbol{m}^t[s] \\ &\quad \text{else} \quad (\boldsymbol{m}^r \textstyle\coprod \boldsymbol{m}^t)[0] \overset{\text{def}}{=} (\boldsymbol{m}^r[0] + \boldsymbol{m}^t[0]) - 1. \end{aligned}$$

In a semiring $(S, +, \cdot, 0, 1)$ we use $\oplus$ to mean the convolution over addition, i.e., over the monoid $(S, +, 0)$, and $\otimes$ to mean the convolution over multiplication, i.e., over the monoid $(S, \cdot, 1)$. Notice that the disjoint operation is always paired with $+$ (not $\cdot$).

Example 5 Consider the Boolean semiring $\mathbb{B}$ and two random variables r and t taking values in $\mathbb{B}$ with marginal probabilities p_r and p_t, respectively. Then $\boldsymbol{m}^r = (1 - p_r, p_r)$ and $\boldsymbol{m}^t = (1 - p_t, p_t)$. If r and t are independent, then the distribution of $r \vee t$ can be computed using $r \oplus t$ (in $\mathbb{B}$, $r \vee t = r + t$). From the definition, we see that $(r \oplus t)[0] = (1 - p_r)(1 - p_t)$ and $(r \oplus t)[1] = (1 - p_t) + (1 - p_r)p_t + p_r p_t$.

If r and t are disjoint, then $\boldsymbol{m}^{r+t}[1] = (\boldsymbol{m}^r \coprod \boldsymbol{m}^t)[1] = (p_r + p_t)$ and $\boldsymbol{m}^{r+t}[0] = (\boldsymbol{m}^r \coprod \boldsymbol{m}^t)[0] = 1 - \boldsymbol{m}^{r+t}[1]$.

The next proposition restates that the two operations in the previous definition yield the correct results, and states bounds on their running time:

[8] A more illustrative way to write this computation is $\Pr[r = 0 \vee t = 0] = 1 - \Pr[r \neq 0 \wedge t \neq 0] = 1 - (1 - \mu(r = 0)) + (1 - \mu(t = 0))$.

`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. 5 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. `COUNT(DISTINCT)` is omitted because it uses two different algebras together. One important point to note is that, in the case of `SUM`, if t is outside the semiring (i.e., larger) than $\tau(t)$ is set to the largest element of the semiring. Since all values are present, once this value is present it forces the value of the predicate θ, e.g., if $\theta = \geq$ then the predicate is trivially satisfied

Proposition 1 *Let r and s be random variables on the monoid $(S, +, 0)$ with marginal vectors $\boldsymbol{m}^r$ and $\boldsymbol{m}^t$, respectively. Then let $\boldsymbol{m}^{r+t}$ denote the marginal of $r + t$. If r and t are independent then $\boldsymbol{m}^{r+t} = \boldsymbol{m}^r \oplus \boldsymbol{m}^t$. If r and t are disjoint then $\boldsymbol{m}^{r+t} = \boldsymbol{m}^r \coprod \boldsymbol{m}^t$. Further, the convolution is associative, so the convolution of n variables $r_1, \ldots, r_n$ can be computed in time $O(n\,|S|^2)$:*

$$\bigoplus_{i=1,\ldots,n} \boldsymbol{m}^{r_i} \overset{\text{def}}{=} \boldsymbol{m}^{r_1} \oplus \cdots \oplus \boldsymbol{m}^{r_n}$$

and disjoint operation applied to $r_1, \ldots, r_n$ denoted below can be computed in $O(n\,|S|)$.

$$\coprod_{i=1,\ldots,n} \boldsymbol{m}^{r_1} \overset{\text{def}}{=} \boldsymbol{m}^{r_1} \coprod \cdots \coprod \boldsymbol{m}^{r_n}$$

Proof We include the proof of the convolution since it is illustrative. We assume that $\boldsymbol{m}^x[i] = \mu(x = i)$ for $x \in \{r, t\}$ and $i \in S$, i.e., the marginal vectors are correct, and that r and t are independent. We show that $\left(\boldsymbol{m}^r \oplus \boldsymbol{m}^t\right)[s] = \mu(r + t = s)$. Since $s \in S$ is arbitrary, this proves the correctness claim.

$$\begin{aligned}
\left(\boldsymbol{m}^r \oplus \boldsymbol{m}^t\right)[s] &= \sum_{i,j \in S: i+j=s} \boldsymbol{m}^r[i]\boldsymbol{m}^t[j] \\
&= \sum_{i,j \in S: i+j=s} \mu(r = i)\mu(t = j) \\
&= \sum_{i,j \in S: i+j=s} \mu(r = i \wedge t = j) \\
&= \mu(r + t = s) = \boldsymbol{m}^{r+t}[s]
\end{aligned}$$

The first equality is the definition. The second equality is by assumption that the marginal vectors are correct. The third line is by the independence assumption. The final line is because the sum is exhaustive. To see the time bound, observe that we can simply consider all $|S|^2$ pairs to compute the convolution (which we assume has unit cost). Since the semiring is associative, and so is the convolution. This also means that we can compute the n-fold convolutions pairwise.

The importance of this proposition is that if the number of elements in the semiring is small, then each operation can be done efficiently. We will use this proposition as the basis of our efficient exact algorithms.

4 Approaches for HAVING

We define α-safe `HAVING` queries for $\alpha \in \{$ `EXISTS`, `MIN`, `MAX`, `COUNT`$\}$ in Sect. 4.3, for $\alpha =$ `COUNT(DISTINCT)` in Sect. 4.4, and $\alpha \in \{$`AVG`, `SUM`$\}$ in Sect. 4.5.

4.1 Aggregates and semirings

We explain how to compute `HAVING` queries using semirings on deterministic databases, which we then generalize to probabilistic databases. Since `HAVING` queries are Boolean, we use a function $\rho : S \to \{\text{true}, \text{false}\}$, called the **recovery function**, that maps a semiring value s to true if that value satisfies the predicate in the having query Q, e.g., when checking `COUNT`$(*) \geq 4$, $\rho(4)$ is true, but $\rho(3)$ is false. Figure 5 lists the semirings for the aggregates in this paper, their associated annotation functions τ, and an associated Boolean recovery function ρ. The aggregation function `EXISTS` essentially yields the safe plan algebra of Dalvi and Suciu [12,13,40].

Example 6 Consider the query $Q[$`MIN`$(y) > 10] :\!- R(y)$ where $R = \{t_1, \ldots, t_n\}$ is a tuple independent database. Figure 5 tells us that we should 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 $t_i[y] \leq 10$. Let $q_\tau = \sum_{i=1,\ldots,m} \tau(t_i)$, the sum is in S, and so, $q_\tau = \max_{i=1,\ldots,m} \tau(t_i)$. Now, $\rho(q_t)$ is satisfied only when q_τ is 1. In turn, this occurs if and only if all $t_i[y]$ are greater than 10 as required.

A careful reader may have noticed that we could have used $\mathbb{Z}_2$ to compute this example (instead of $\mathbb{Z}_3$). When we generalize to probabilistic databases, we may have to account for a tuple being absent (for which we use the value 0).

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. 5, then for any deterministic instance* W*:*

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

Proof Let q, the skeleton of Q, have n subgoals. We show only `MIN` with $\leq$ in full detail. All other aggregate-test pairs follow by similar arguments. We observe the equation

$$q(W, \tau) = \sum_{v:\mathbf{im}(v)\subseteq W}\ \prod_{i=1,\dots,n} v(g_i)$$

Further, $W \models Q[\mathtt{MIN}(y) \leq k]$ if and only if there there is some valuation such that $\prod_{i=1,\dots,n} v(g_i) = 2$. Since, $2+s = 2$ for any $s \in S$ the existence of such a valuation implies $q(W, \tau) = 2$. Conversely, if $q(W, \tau) = 2$ then there must be some such valuation since $x + y = 2$ implies that either x or y is 2 in this semiring. Hence, the claim holds.

Similarly, $W \models Q[\mathtt{MIN}(y) \geq k]$, the query is satisfied if and only if *all* elements are $\geq k$ and so each term (valuation) in the summation must evaluate to 0 or 1. Similar arguments are true for $=, \neq$. In the case of `COUNT`, if we want to count from $1, \dots, k$ we also need two elements, 0 and $k + 1$: 0 encodes that a tuple is absent and $k + 1$ encodes that the value is "bigger than k".

In probabilistic databases, we view $q(W, \tau)$ as a random variable by fixing τ (the semiring annotation functions), i.e., we view $q(W, \tau)$ as a function of W alone. We denote this random variable q_τ. Our goal is to compute the marginal vector of q_τ. The marginal vector of q_τ, denoted $\boldsymbol{m}^{q_\tau}$, is sufficient to compute the value of any `HAVING` query since we can simply examine those entries in $\boldsymbol{m}^{q_\tau}$ for which the recovery function, ρ, is true. Said another way, a simple corollary of Proposition 2 is the following generalization to probabilistic databases:

Corollary 1 *Given a* `HAVING` *query* Q*, let* $q = \mathbf{sk}(Q)$*,* S*,* ρ*, and* τ *be as in Proposition 2, then for any BID instance* J *we have the following equalities:*

$$\mu_J(Q) = \sum_{k\,:\,\rho(k)\text{ is true}} \boldsymbol{m}^{q_\tau}[k]$$

Corollary 1 tells us that we can compute $\mu(Q)$ by examining the entries of the marginal vector $\boldsymbol{m}^{q_\tau}$. Hence, our goal is to compute $\boldsymbol{m}^{q_\tau}[s]$ for each such index, $s \in S$.

4.2 Computing safely in semirings

We now extend safe plans to compute a marginal vector instead of a Boolean value. Specifically, we compute $\boldsymbol{m}^{q_\tau}$, the marginal vector for q_τ using the operations defined in Sect. 3.2.

Definition 11 An **extensional plan** for a Boolean conjunctive query q is defined recursively as a subgoal g and if P_1, P_2 are extensional plans then so are $\pi^I_{-x} P_1$ (independent project), $\pi^D_{-x} P_1$ (disjoint project), and $P_1 \bowtie P_2$ (join). An extensional plan P is **safe** if, assuming P_1 and P_2 are safe, the following conditions are met:

- $P = g$ is always 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 $i = 1, 2$, $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$, i.e., we may not project away variables that are shared in two subgoals before they are joined.

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

Intuitively, a safe plan tells us that the correlations of tuples produced by intermediate stages of the plan are either independent or disjoint, as opposed to correlated in some unknown way. In particular, $P = \pi^I_{-x}(P_1)$ is a safe plan whenever those tuples produced by P_1 on any instance are independent (provided the tuples differ on the variable x). Hence, we call π^I an independent project. Similarly, if $P = \pi^D_{-x}(P_1)$ is safe, then the tuples produced by P_1 are disjoint whenever they differ on the variable x. Further, a join is safe if the branches do not contain any common subgoals, i.e., any tuple produced by P_1 is independent of any tuple produced by P_2. For completeness, we state and prove a formal version of this discussion in Appendix A.

Computing with safe plans

We now augment safe plans to compute marginal vectors. Intuitively, we generalize the operation of multiplying marginal probabilities (as done in safe plans) to semiring convolutions of marginal vectors, and we generalize the operation of adding the marginal probabilities of disjoint events to disjoint operations on marginal vectors. We think of a plan as computing a marginal vector: the marginal vector computed by a plan P on a BID instance J is called the *extensional value* of P and is denoted as $\hat{\omega}^J_{P,S}$ and is defined below.

Definition 12 Given a BID instance J and a semiring S. Let P be a safe plan. Denote the **extensional value** of P in S on J as $\hat{\omega}^J_{P,S}$. $\hat{\omega}^J_{P,S}$ is a function that maps each tuple to a marginal vector. To emphasize the recursion, we fix J and S and denote $\hat{\omega}^J_{P,S}$ as $\hat{\omega}_P$. We define the value of $\hat{\omega}_P$ inductively:

- If $P = g$ then $\hat{\omega}_P(t) = \boldsymbol{m}^t$ where $\boldsymbol{m}^t[0] = 1 - t[P]$ and $\boldsymbol{m}^t[\tau_g(t)] = t[P]$ and all other entries are 0.

- If $P = \pi^{I}_{-x} P_1$ then $\hat{\omega}_P(t) = \bigoplus_{t':t'[\mathbf{var}(P_1)]=t} \hat{\omega}_{P_1}(t)$ where $\bigoplus$ denotes the convolution over the monoid $(S, +, 0)$.
- If $P = \pi^{D}_{-x} P_1$ then $\hat{\omega}_P(t) = \coprod_{t':t'[\mathbf{var}(P_1)]=t} \hat{\omega}_{P_1}(t)$ where $\coprod$ denotes the disjoint operation over the monoid $(S, +, 0)$.
- If $P = P_1 \bowtie P_2$ then $\hat{\omega}_P(t) = \hat{\omega}_{P_1}(t_1) \otimes \hat{\omega}_{P_2}(t_2)$ where for $i = 1, 2$ t_i is t restricted to $\mathbf{var}(P_i)$ and $\otimes$ denotes the convolution over the monoid $(S, \cdot, 1)$.

Figure 4b gives an example of computing the extensional value of a plan: the plan shown is not safe, meaning that the extensional value it computes is not correct, i.e., equal to $\boldsymbol{m}^{q_\tau}$. This illustrates that any plan may be converted to an extensional plan, but we need additional conditions (safety) to ensure that the computation is correct. Interestingly, in this case, there is an alternate safe plan: $P_0 = \pi_{-x}(R(x) \bowtie \pi_{-y}(S(x, y)))$, i.e., we move the projection early.

The next lemma states that for safe plans, the extensional value is computed correctly, i.e., the conditions insured by the safe plan and the operator used in Definition 12 make exactly the same correlation assumptions. For example, π^I indicates independence, which ensures that $\oplus$ correctly combines two input marginal vectors. The proof of the following lemma is a straightforward induction and is omitted.

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

A safe plan (in the terminology of this paper) ensures that the convolutions and disjoint operations output the correct results, but it is *not sufficient to ensure that the plan is efficient*. In particular, the operations in a safe plan on S take time (and space) 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*. As we will see, this happens for SUM in most cases. As we show in the next section, if α is one of MIN, MAX, or COUNT, the number of elements in the needed semiring is small enough, so the safety of $\mathbf{sk}(Q)$ and Q coincide.

4.3 EXISTS-,MIN-, MAX- and COUNT-safe

We now give optimal algorithms when α is one of EXISTS, MIN, MAX, or COUNT. The results on EXISTS are exactly the results of Dalvi and Suciu [12]. We include them to clarify our generalization.

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

Theorem 1 *Let $Q[\alpha(y)\ \theta\ k]$ be a HAVING query for $\alpha \in$ { EXISTS, MIN, MAX, COUNT} such that Q is α-safe then the exact evaluation problem for Q is in polynomial time in the size of the data.*

Correctness is straightforward from Lemma 1. Efficiency follows because the semiring is of constant size for EXISTS, MIN, and MAX. For COUNT, observe that an upper bound on $|S|$ is number of tuples returned by the query plus one (for empty), thus count is polynomially bounded as well. Thus, the entire plan has polynomial time data complexity.

Complexity

The results of Dalvi and Suciu [12,13,40] show that either a conjunctive query without self-joins has a safe plan or it is $\sharp\mathcal{P}$-hard. The idea is to show that a HAVING query Q is satisfied only if $\mathbf{sk}(Q)$ is satisfied, which implies that computing Q is at least as hard as computing $\mathbf{sk}(Q)$. Formally, we have:

Theorem 2 [Exact Dichotomy for MIN, MAX, and COUNT] *If $\alpha \in$ {MIN, MAX, COUNT} and $Q[\alpha(y)\ \theta\ k]$ does not contain self-joins, then either (1) Q is α-safe and so Q has data complexity in $\mathcal{P}$, or (2) Q has $\sharp\mathcal{P}$-hard data complexity. Further, we can find an α-safe plan in $\mathcal{P}$.*

Proof The first part of the dichotomy is Theorem 1. We show the matching negative result. Consider the predicate test MIN$(y) \geq 1$; assuming that Q is not MIN-safe, we have (by above) that $\mathbf{sk}(Q) = q$ is not safe in the sense of Dalvi and Suciu, we show that this query can be used to compute $\Pr[q]$ on an BID instance J. To see this, create a new instance J' that contains exactly the same tuples as J, but recode all values in attributes referenced by y as integers with values greater than 1: this query is true precisely when at least one tuple exists and hence with $\Pr[q]$. We show below that this is sufficient to imply that all tests θ are hard as well. The proof for MAX is symmetric. COUNT is similar.

Lemma 2 *Let $\alpha \in$ {MIN,MAX, COUNT, SUM, COUNT (DISTINCT), if computing $Q[\alpha(y) = k]$ exactly is $\sharp\mathcal{P}$-hard, then it is $\sharp\mathcal{P}$-hard for* all *$\theta \in \Theta$. Furthermore, if q_τ takes at most polynomially many values then the converse also holds: if computing $Q[\alpha(y)\ \theta\ k]$ exactly is $\sharp\mathcal{P}$-hard for any $\theta \in \Theta$, then it is $\sharp\mathcal{P}$-hard for all $\theta \in \Theta$.*

Proof We first observe that all aggregate functions α in the statement are positive, integer-valued functions. We show that we can use $\leq, \geq, >, <, \neq$ as a black box to compute $=$ efficiently. We then show that we can compute the inequalities in time $O(k)$ (using $=$), thus proving both parts of the claim.

First, observe that $q(W, \tau) = s$ is a function on worlds, i.e., the events are disjoint for different values of s. Hence,

$$\mu(Q[\alpha(y) \leq k]) = \sum_{k' \leq k} \mu(Q[\alpha(y) = k']$$

From this equation it follows that we can compute any inequality using $=$ in time proportional to the number of possible values. To see the forward direction, we compute

$$\mu(Q[\alpha(y) \leq k+1]) - \mu(Q[\alpha(y) \leq k]) = \mu(Q[\alpha(y) = k])$$

similarly for a strict inequality. And, $1 - \mu(Q[\alpha(y) \neq k]) - \mu(Q[\alpha(y) \neq 0]) = \mu(Q[\alpha(Y) = k])$. The $\neq 0$ statement is only necessary with SQL semantics.

The exact $\sharp\mathcal{P}$-hardness proofs in the remainder of this section satisfy the requirement of this lemma. Interestingly, this lemma *does not hold for approximation hardness*.

4.4 COUNT(DISTINCT)-safe queries

Intuitively, we compute COUNT(DISTINCT) in two stages: (1) For the subplan rooted at π_{-y}, we first compute the probability that each value is returned by the plan (i.e., we compute the DISTINCTpart using EXISTS). (2) Then, since we have removed duplicates implicitly using EXISTS, we count the number of distinct values using the COUNT algorithm from Sect. 4.3.

The ordering of the operators, first EXISTS and then COUNT, is important. As we show in Theorem. 4, this ordering exactly captures tractable evaluation. First, we need a small technical proposition to state our characterization:

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

Proof 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 $i = 1, 2$ $\mathbf{var}(\mathbf{goal}(P_1)) \cap \mathbf{var}(\mathbf{goal}(P_2)) \subseteq \mathbf{var}(P_i)$.

Thus, it makes sense to talk about the unique node in the plan tree where a variable x is removed, as we do in the next definition:

Definition 14 A query Q[COUNT(DISTINCT y) θ k] is **COUNT(DISTINCT)-safe** if there is a safe plan P for the skeleton of Q such that if P_1 is the unique node in the plan where y is removed, i.e., either π^I_{-y} or π^D_{-y} in P, then no proper ancestor of P_1 is π^I_{-x} for any x.

This definition exactly insists on the ordering of operators that we highlighted above.

Example 7 Fix a BID instance J. Consider

Q[COUNT(DISTINCT y) ≥ 2] :–$R(y, x), S(y)$

A COUNT(DISTINCT)-safe plan for the skeleton of Q is $P = \pi^I_{-y}((\pi^I_{-x} R(y, x)) \bowtie S(y))$. The subquery $P_1 = (\pi^I_{-x} R(y, x)) \bowtie S(y)$ returns tuples (values for y). We use the EXISTS algebra to compute the probability that each distinct value appears.

Now, we must count the number of distinct values: since we have eliminated duplicates, all y values are trivially distinct and we can use the COUNT algebra. To do this, we map each EXISTS marginal vector to a vector suitable for computing COUNT, i.e., a vector in $\mathbb{Z}_k$ (here $k = 2$). In other words, $(1 - p, p) = \hat{\omega}^J_{P,\mathrm{EXISTS}}(t) = \boldsymbol{m}^t$ is mapped to $\hat{\tau}(\boldsymbol{m}^t) = (1 - p, p, 0)$. In general, this vector would be of length $k + 1$.

Since $P = \pi^I_{-y} P_1$, we know that all tuples returned by P_1 are independent. Thus, the correct distribution is given by convolution over all such t', each one corresponding to a distinct y value, i.e., $\oplus_t \hat{\tau}(t')$. To compute the final result, use the recovery function, ρ defined by $\rho(s) = s \geq 2$

The proof of the following theorem is a generalization of Example 7, whose proof we include in Appendix B.

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

Complexity. We now establish that for COUNT (DISTINCT) queries without self-joins, COUNT (DISTINCT)-safe captures efficient computation. We do this in two stages: first, we exhibit some canonical hard patterns for COUNT(DISTINCT), and second, in the appendix, we reduce any other non- COUNT(DISTINCT)-safe pattern to one of these hard patterns.

Proposition 4 *The following HAVING queries are $\sharp\mathcal{P}$-hard for $i = 1, 2, \ldots$:*

Q_1[*COUNT(DISTINCT* y) θ k] :–R(x), S(x, y)

and,

$Q_{2,i}$[*COUNT(DISTINCT* y) θ k] :–R$_1(x; y), \ldots,$ R$_i(x; y)$

Proof We prove Q_1 is hard and defer $Q_{2,i}$ to the Appendix B. To see that Q_1 is hard, we reduce from counting the number of independent sets in a graph (V, E) which is $\sharp\mathcal{P}$-hard. We let k be the number of edges ($|E|$) and $\theta =$ ‘ $\geq$’. Intuitively, with these choices Q will be satisfied only when all edges are present. For each node $u \in V$, create a tuple R(u) with probability 0.5. For edge $e = (u, v)$ create two tuples S(u, e), S(v, e), each with probability 1. For any set $V' \subseteq V$, let $W_{V'}$ denote the world where the tuples corresponding to V' are present. For any subset of nodes, V', we show that V' is an independent set if and only if $W_{V-V'}$

satisfies Q_1, i.e., all edges are present in its node-complement. 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, thus completing the reduction. Now, if N is an independent set, then for any edge (u, v), it must be the case that at least one of u or v is in $V - N$, else the set would not be independent, since it would contain an induced edge. 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. This completes the reduction. 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 showing that the patterns in the previous theorem capture the boundary of hardness.

Theorem 4 [COUNT(DISTINCT) Dichotomy] *Let $Q[\alpha(y)\ \theta\ k]$ be a HAVING such that α is COUNT(DISTINCT), then either (1) Q is COUNT(DISTINCT)-safe and so has $\mathcal{P}$ data complexity or (2) Q is not COUNT(DISTINCT)-safe and has $\sharp\mathcal{P}$-hard data complexity.*

Proof Part (1) of the result is Theorem 3. We sketch the proof of (2) in the simpler case when only tuple independent probabilistic tables are used in Q and defer a full proof to Appendix B. 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 $S(z, y)$ such that $y \notin \boldsymbol{x} \cup z$ and $\boldsymbol{x} \cap z \neq \emptyset$. Given a graph (V, E), we then construct a BID instance J exactly as in the proof of Proposition 4. Only the R relation is required to have probabilistic tuples, all others can set their probabilities to 1.

Extending to BID databases requires more work because our technique of adding extra tuples with probability 1 does not work: doing so naively may violate a possible worlds key constraint. The full proof appears in Appendix B. It is straightforward to decide if a plan is COUNT(DISTINCT)-safe: the safe plan algorithm of Dalvi and Suciu [13,40] 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 need to further restrict the class of allowable plans. For example, there are queries involving SUM on a single table that are $\sharp\mathcal{P}$-hard, e.g., the query $Q[\text{SUM}(y) = k]$:–$R(y)$ is already $\sharp\mathcal{P}$-hard. There are, however, some queries that can be evaluated efficiently:

Definition 15 A HAVING query $Q[\alpha(y)\ \theta\ k]$ for $\alpha \in \{\text{SUM}, \text{AVG}\}$ 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.

The idea of the positive algorithm is that if the plan contains π^D_{-y}, i.e., each value for y is present disjointly. Let $a_1, \ldots, a_n$ be the y values returned by running the standard query $q(y)$ [adding y to the head of $\mathbf{sk}(Q)$]. Now consider the query Q' where $\mathbf{sk}(Q') = q[y \to a_i]$ (substitute y with a_i). On this query, the value of y is fixed, so we only need to compute the multiplicity of a_i figure out if Q' is true. To do this, we use the COUNT algebra of Sect. 4.3 whenever q is safe.

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

Proof [Sketch] Since Q is α-safe, then there is a plan P satisfying Definition 15. The consequence of this definition is that on any possible world W, we have that the conjunctive query $q(y)$ ($q = \mathbf{sk}(Q)$) returns a single tuple (i.e., a single binding for y). This implies that the values are *disjoint*. So for a fixed positive integer a returned by $q(y)$, the predicate $\text{SUM}(y)\ \theta\ k$ depends only on the *multiplicity* of a. Hence, we can write:

$$\Pr[Q] = \sum_{a \in S} \Pr[Q_a \left[\text{COUNT}(*)\ \theta\ \frac{k}{a}\right]$$

Here, Q_a denotes that $\mathbf{sk}(Q_a) = q[y \to a]$, i.e., y is substituted with a in the body of Q_a. Since Q is α-safe, we have that $q[y \to a]$ is safe, and so by Theorem 1, each term can be computed with the COUNT algebra. Hence, we can compute the entire sum in polynomial time and so $\Pr[Q]$. For AVG, it is slightly simpler: Since we are taking the value of m copies of a, we have that the AVG is a if $m > 0$ (else the query is false). Thus, we simply need to compute the probability that the value a exists with multiplicity greater than 1 (which can be handled by the standard EXISTS algebra).

Example 8 Consider $Q[\text{SUM}(y) > 10]$:–R('a'; y), S(y, u). This query is SUM-safe, with plan π^D_{-y}(R('a'; y) $\bowtie$ π^I_{-u} 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 *Let $\alpha \in \{\text{SUM}, \text{AVG}\}$ and $\theta \in \{\leq, <, =, >, \geq\}$ then $Q[\alpha(y)\ \theta\ k]$:–$R(y)$ has $\sharp\mathcal{P}$-data complexity.*

Proof We only show SUM, deferring AVG to the appendix. 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 $S \subseteq 1, \ldots, n$ such that $\sum_{s \in S} x_i = B$. We create the representation with schema $R(X;;P)$ satisfying $\mathtt{R} = \{(x_1; 0.5), \ldots, (x_n; 0.5)\}$, i.e., each tuple present with probability 0.5. Thus, $\mu(Q) * 2^n$ is number of such S. Showing hardness for other aggregate tests follows from Lemma 2.

Theorem 6 *Let $\alpha \in \{\mathtt{SUM}, \mathtt{AVG}\}$ and let $Q[\alpha(y)\ \theta\ k]$ be a* HAVING *query, then either (1) Q is α-safe and hence has $\mathcal{P}$-time data complexity, or (2) Q is not α-safe and Q has $\sharp\mathcal{P}$-data complexity.*

We prove this theorem in Appendix C.

5 Generating a random world

In this section, we give an algorithm (Algorithm 5.2.1) to solve the *random possible world generation problem*, which informally asks us to generate a possible world $\tilde{W}$ such that $q(\tilde{W}, \tau) = s$, i.e., such that the value of q on $\tilde{W}$ is s. The probability that we generate a fixed world $\tilde{W}$ is exactly the probability of $\tilde{W}$ *conditioned* on the value of q being equal to s. Our solution to this problem is a key a subroutine in our FPTRAS for SUM (in Sect. 6), but it is also an interesting problem in its own right. As pointed out by Cohen et al. [11], a random world satisfying some constraints is useful for many debugging and related tasks.

5.1 Problem Definition

Definition 16 Let J be a BID instance, q be a conjunctive query, and τ be an annotation function. A *BID random world generator* (simply, a *random generator*) is a randomized algorithm $\mathcal{A}$ that generates a possible world $\tilde{W} \in \mathcal{W}_J$ such that for any $s \in S$ we have:[9]

$$\Pr_{\mathcal{A}}[\tilde{W} = W] = \mu(W \mid q(W, \tau) = s)$$

where $\Pr_{\mathcal{A}}$ emphasizes that the probability is taken over the random choices of the algorithm $\mathcal{A}$. Further, we require that $\mathcal{A}$ run in time poly$(|J|, |S|)$.

This definition says that the probability a world is generated is *exactly* the conditional probability of that instance (conditioned on the value of the query q being s). In this section, we show that when $\mathbf{sk}(Q)$ is safe then we can solve create a random generator for any BID instance and any annotation function.

5.2 Possible world generation algorithm

[9] Formally, if $\mathcal{W} = \emptyset$, then we require that that a random generator return a special value, $\perp$. This value is like an exception and will not be returned during the course of normal execution.

Algorithm 5.2.1 A random world generator for J_ϕ

```
Decl: RWHELPER($\phi$ : semiring parse tree,
                $s$ : a semiring value)
    returns a random world denoted $\tilde{W} \subseteq J_\phi$.

if $\phi$ is a leaf, i.e., $\phi = (t, \boldsymbol{m}^t)$ for some tuple $t$ then
  (* If $s \neq 0$ then this implies the tuple must be present. *)
  if $s \neq \tau(t)$ then return $\{t\}$
  elif $s = 0$ then return $\emptyset$ else return $\perp$
(* Inductive case *)
Let $\phi$ have label $(\text{OP}, m^r)$ and children $\phi_1$ and $\phi_2$
    with marginal vectors $m^{\phi_1}$ and $m^{\phi_2}$, respectively.
if OP $= \oplus$ then
  Choose $(s_1, s_2)$ s.t. $s_1 + s_2 = s$ with probability $m^{\phi_1}[s_1]m^{\phi_1}[s_2]\frac{1}{m^{\phi}[s]}$
if OP $= \otimes$ then
  Choose $(s_1, s_2)$ s.t. $s_1 \cdot s_2 = s$ with probability $m^{\phi_1}[s_1]m^{\phi_1}[s_2]\frac{1}{m^{\phi}[s]}$
if OP $= \coprod$ then
  Choose $(s_1, s_2) = (s, 0)$ with probability $m^{\phi_1}[s_1]\frac{1}{m^{\phi}[s]}$
      or $(s_1, s_2) = (0, s)$ with probability $m^{\phi_1}[s_2]\frac{1}{m^{\phi}[s]}$

(* Union the results of the recursive calls *)
return RWHELPER($\phi_1, s_1$) $\cup$ RWHELPER($\phi_2, s_2$)
```

Algorithm 5.2.2 A random world generator for J

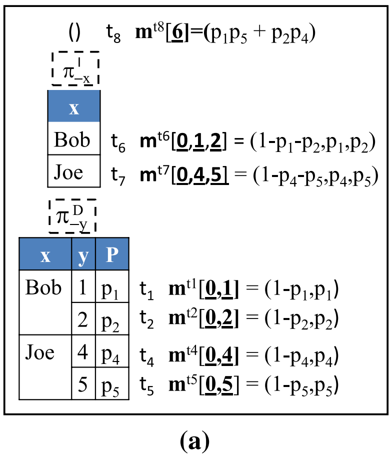
```
Decl: RANDOMWORLD($\phi$ : semiring parse tree,
                  $J$ : A BID instance, $s$ a semiring element
    returns a random world of $J$ denoted $\tilde{W}$.

Let $\tilde{W} \leftarrow$ RWHELPER($\phi, s$) and $T = J - J_\phi$
for each $t \in T$ do
  Let $K(t) = \{t' \mid t[K] = t'[K]\} = \{t_1, \ldots, t_m\}$ with $p_i = \Pr[t_i]$.
  Let $\{t_{k+1}, \ldots, t_m\} = K(t) \cap J_\phi$
  if $K(t) \cap \tilde{W} = \emptyset$ then
    select $t_i$ from $i = 1, k$ with $\frac{p_i}{1 - \sum_{j=k+1,m} p_j}$ and $\tilde{W} \leftarrow \tilde{W} \cup \{t_i\}$
  $T \leftarrow T - K(t)$
return $\tilde{W}$
```

To describe our algorithm, we need a notation to record the intermediate operations of the safe plan on the marginal vectors, i.e., a kind of *lineage* or *provenance* for the semiring computation. Here, we view a safe plan as computing the marginal vectors *and* as computing a symbolic semiring expression (essentially, a parse tree of the extensional computation performed by the plan).

Definition 17 A *semiring parse tree* ϕ is a binary tree where a leaf is labeled with a pair $(t, \boldsymbol{m}^t)$ where t is a tuple and $\boldsymbol{m}$ is a marginal vector on S; and an internal node is labeled with a pair $(\text{OP}, \boldsymbol{m})$ where OP $\in \{\oplus, \otimes, \coprod\}$ and $\boldsymbol{m}$ is a marginal vector.

Given a safe plan P and a BID instance J with annotation τ, the parse tree associated to P and J is denoted $\phi(P, J, \tau)$. We think of $\phi(P, J, \tau)$ as a record of the computation of P on J. More precisely, $\phi(P, J, \tau)$ is a parse tree for the semiring expression that we compute given P and J using the rules of Definition 12. The operations in a safe plan are n-ary: we can, however, transform these n-ary operations into


()
t_8 $m^{t8}[6]=(p_1p_5+p_2p_4)$
π^I_{-x}
x
Bob
Joe
t_6 $m^{t6}[0,1,2]=(1-p_1-p_2,p_1,p_2)$
t_7 $m^{t7}[0,4,5]=(1-p_4-p_5,p_4,p_5)$
π^D_{-y}
x y P
Bob 1 p_1
2 p_2
Joe 4 p_4
5 p_5
t_1 $m^{t1}[0,1]=(1-p_1,p_1)$
t_2 $m^{t2}[0,2]=(1-p_2,p_2)$
t_4 $m^{t4}[0,4]=(1-p_4,p_4)$
t_5 $m^{t5}[0,5]=(1-p_5,p_5)$
(a)

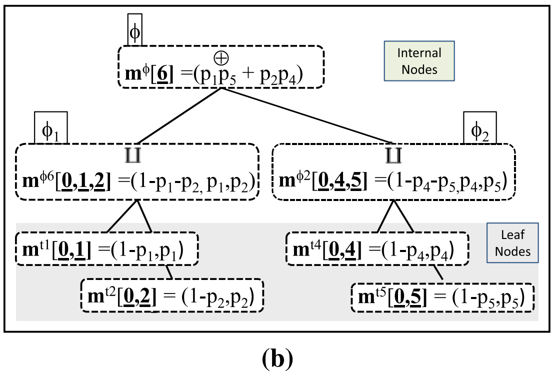

ϕ
$\oplus$
$m^\phi[6]=(p_1p_5+p_2p_4)$
Internal Nodes
ϕ_1
ϕ_2
$\coprod$
$m^{\phi 6}[0,1,2]=(1-p_1-p_2,p_1,p_2)$
$m^{\phi 2}[0,4,5]=(1-p_4-p_5,p_4,p_5)$
$m^{t1}[0,1]=(1-p_1,p_1)$
$m^{t4}[0,4]=(1-p_4,p_4)$
Leaf Nodes
$m^{t2}[0,2]=(1-p_2,p_2)$
$m^{t5}[0,5]=(1-p_5,p_5)$
(b)

Fig. 6 **a** A BID relation $R(A;B)$ used in Example 9 along with a safe plan $P=\pi^I_{-x}(\pi^D_{-y}(R))$. The extensional computation is in the semiring $\mathbb{N}$. **b** $\phi(P,\{R\})$ is shown where $P=\pi^I_{-x}(\pi^D_{-y}(R))$. The dotted boxes map to the nodes in the tree, described in Definition 17. In the figure, for readability, we only show the entry for 6 in the root. Also, the entries in the marginal vectors are real (rational) numbers and are written as expressions *only* for the sake of readability. There is a one-to-one correspondence between intermediate marginal vectors and nodes in the parse tree $\phi(P,J,\tau)$

a binary parse tree in an arbitrary way, since the operations are associative. An example parse tree is shown in Fig. 6. We observe that any safe plan can be mapped to a parse tree.

Example 9 Figure 6 illustrates how $\phi(P,J,\tau)$ is constructed for a simple example based on SUM. Figure 6a shows a relation $R(A;B)$ where A is a possible worlds key. Our goal is to generate a random world such that the query $Q[\mathtt{SUM}(y)=6]$:–$R(x;y)$ is true. The skeleton of Q is safe and so has a safe plan, $P=\pi^I_{-x}(\pi^D_{-y}(R))$. Figure 6a also shows the intermediate tuples that are computed by the plan, along with their associated marginal vectors. For example, the marginal vector associated to t_1 is $\boldsymbol{m}^{t_1}[0,1]=(1-p_1,p_1)$. Similarly, the marginal vector for intermediate tuples like t_6 is $\boldsymbol{m}^{t_6}[1]=p_2$. At the top of the plan is the empty tuple, t_8, and one entry in its associated marginal vector, i.e., $\boldsymbol{m}^{t_8}[6]=p_1p_5+p_2p_4$.

The parse tree $\phi(P,J,\tau)$ corresponding to P on the instance $J=\{R\}$ is illustrated in Fig. 6b. The bottom-most level of internal nodes have $\mathrm{OP}=\coprod$, since they encode the action of the disjoint projection π^D_{-y}. In contrast, the root has $\mathrm{OP}=\oplus$, since it records the computation of the independent project, π^I_{-x}. As we can see, the parse tree simply records the computation and the intermediate results.

Algorithm Overview. Our algorithm has two phases: (1) we first build a random generator for the tuples in the parse tree ϕ (defined formally below); this is Algorithm 5.2.1. (2) Using the tuples generated in step (1), we select those tuples not in the parse tree and complete the generator for J; this is Algorithm 5.2.2. To make this precise, we need the following notation:

Definition 18 Given a semiring parse tree ϕ, we define $\mathbf{tup}(\phi)$ inductively: if ϕ is a leaf corresponding to a tuple t, then $\mathbf{tup}(\phi)=\{t\}$. Otherwise, ϕ has two child parse trees ϕ_1 and ϕ_2, then $\mathbf{tup}(\phi)=\mathbf{tup}(\phi_1)\cup\mathbf{tup}(\phi_2)$. We also consider $\mathbf{tup}^+(\phi)=\mathbf{tup}(\phi)-\{t\mid\tau(t)=0\}$, i.e., $\mathbf{tup}^+(\phi)$ is the set of tuples with non-zero annotations contained in ϕ.

If P is a safe plan, then **tup** has a particular simple form:

Proposition 6 *Let P be a safe plan for q and J be a BID instance, then for any internal node ϕ_0 in $\phi(P,J,\tau)$ with children ϕ_1 and ϕ_2, we have that* $\mathbf{tup}(\phi_1)\cap\mathbf{tup}(\phi_2)=\emptyset$ *and* $\mathbf{tup}^+(\phi_1)\cap\mathbf{tup}^+(\phi_2)=\emptyset$.

Proof We observe that an $\oplus$ or an $\coprod$ node is introduced only if there is a projection removing a variable x (Definition 11), in which case the tuples in $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ disagree on x, hence, are disjoint sets of tuples. Case two is that $\mathrm{OP}=\otimes$, which is introduced only as a join of two tuples. In this case, $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ come from different relations (since there are no self-joins in q). Thus, $\mathbf{tup}(\phi_1)$ and $\mathbf{tup}(\phi_2)$ have an empty intersection. The second statement follows since $\mathbf{tup}^+(\phi_i)\subseteq\mathbf{tup}(\phi_i)$ for $i=1,2$.

For any parse tree ϕ, we can view the tuples in $\mathbf{tup}^+(\phi)$ as a BID instance that we denote J_ϕ (any subset of a BID instance is again, a BID instance). For a deterministic world W and a semiring expression ϕ, we write $\phi(W)$ to mean the semiring value of ϕ on world W, which is computed in the obvious way.

Step (1): A generator for J_ϕ. We now define precisely the first step of our algorithm: Our goal is to construct a random world generator for the worlds induced by the BID instance J_ϕ. This is captured by the following lemma:

Lemma 3 *Let P be a safe plan for a query q, $\phi=\phi(P,J,\tau)$, and $J_\phi=\mathbf{tup}^+(\phi)$ then Algorithm 5.2.1 is a random generator for J_ϕ for any annotation function τ.*

Proof Let ϕ_0 be a subtree of $\phi(P, J, \tau)$. Then, given any $s \in S$, Algorithm 5.2.1 is a random generator for J_{ϕ_0}. We induct on the structure of the parse tree ϕ. In the base case, ϕ_0 is a leaf node and our claim is straightforward: If $s = 0$, then we return the empty world. If $\tau(t) = s$, then we simply return a singleton world $\{t\}$ if $\tau(t) = s$. Otherwise, we have that $\tau(t) \neq s$, then the input is not well-formed and we return an exception $(\perp)$ as required. This is a correct random generator, because our input is conditioned to be deterministic (i.e., μ has all the mass on a single instance).

We now write the probability that $\phi(W) = s$ in a way that shows that if we recursively can randomly generate worlds for subtrees, then we can make a random generator. Inductively, we consider an internal node ϕ with children ϕ_1 and ϕ_2. Assume for concreteness that $\text{OP} = \oplus$ (the argument for $\text{OP} = \otimes$ is identical and for $\text{OP} = \coprod$ is only a slight variation). Let W denote a world of J_ϕ. Then,

$$\begin{aligned}&\Pr[\phi(W) = s]\\&\quad = \Pr[\phi_1(W) = s_1 \wedge \phi_2(W) = s_2 \mid s_1 + s_2 = s]\end{aligned}$$

This equality follows from the computation of ϕ. We then simplify this expression using the fact that for $i = 1, 2$, ϕ_i's value is a function $\mathbf{tup}^+(\phi_i)$. Let $W_i = W \cap \mathbf{tup}^+(\phi_i)$, we get:

$$\Pr[\phi_1(W_1) = s_1 \wedge \phi_2(W_2) = s_2 \mid s_1 + s_2 = s]$$

Observe that $\Pr[s_1 + s_2 = s] = \Pr[\phi(W) = s]$. Then, for any fixed s_1, s_2 such that $s_1 + s_2 = s$, we can then apply Bayes's rule and independence to get:

$$\frac{\Pr[\phi_1(W_1) = s_1]\Pr[\phi_2(W_2) = s_2]}{\Pr[\phi(W) = s]}$$

Notice that W_1 (respectively, W_2) is a possible world of J_{ϕ_1} (respectively, J_{ϕ_2}) and so the inductive hypothesis applies. Now, by Proposition 6, the worlds returned by these worlds do not make conflicting choices. Since the recursive calls are correct, we just need to ensure that we pick (s_1, s_2) with the above probability. Examining Algorithm 5.2.1, we see that we pick (s_1, s_2) with exactly this probability, since

$$\begin{aligned}&\Pr[\phi_1(W) = s_1 \wedge \phi_2(W) = s_2 \mid s_1 + s_2 = s]\\&\qquad = \frac{m^{\phi_1}[s_1]m^{\phi_2}[s_2]}{m^{\phi}[s]}\end{aligned}$$

This completes the proof.

Example 10 We illustrate Algorithm 5.2.1 using the data of Example 9. Our goal is to generate a random world such that the query $Q[\texttt{SUM}(y) = 6]$:–$R(x; y)$ is true. The algorithm proceeds top-down from the root ϕ. The entry for 6 is selected with probability equal to $p_1 p_5 + p_2 p_4$.

Assume we have selected 6, then we look at the child parse trees, ϕ_1 and ϕ_2: there are two ways to derive 6 with non-zero probability (1) the subtree ϕ_1 takes value 1 and ϕ_2 takes value 5, written $(\phi_1, \phi_2) = (1, 5)$ or (2) we set $(\phi_1, \phi_2) = (2, 4)$. We choose between these options randomly; we select $(\phi_1, \phi_2) = (1, 5)$ with probability equal to $\frac{p_1 p_5}{p_1 p_5 + p_2 p_4}$ (the conditional probability). Otherwise, we select $(\phi_1, \phi_2) = (2, 4)$. Suppose we have selected $(\phi_1, \phi_2) = (1, 5)$, we then recurse on the subtree ϕ_1 with value $s_1 = 1$ and the subtree ϕ_2 with value $s_2 = 5$.

Recursively, we can see that to set $\phi_1 = 1$, it must be that t_1 is present and t_2 is absent. Similarly, we conclude that t_4 must be absent and t_5 must be present. Hence, our random world is $\tilde{W} = \{t_1, t_5\}$. If we had instead chosen $(\phi_1, \phi_2) = (2, 4)$ then we would selected $\tilde{W} = \{t_2, t_4\}$. Notice that our algorithm never selects $(\phi_1, \phi_2) = (3, 3)$ (i.e., this occurs with probability 0). More generally, this algorithm *never* selects any invalid combination of tuple values.

Step (2): A generator for J. We randomly include tuples in J that are not mentioned in ϕ, i.e., tuples in $J - J_\phi$. These are tuples that do not match any selection condition in the query, and can be freely added to $\tilde{W}$ without affecting the query result. Here, we need to exercise some care to not insert two tuples with the same key into $\tilde{W}$, and so, we only consider tuples whose possible worlds key differs from those returned by Step (1). Formally, we prove the following lemma:

Lemma 4 *Let $\phi(P, J, \tau)$ be a parse tree for a safe plan P, a BID instance J, and an annotation τ. Then, given a random generator for J_ϕ, Algorithm 5.2.2 is a random generator for J.*

Proof We first use the random generator to produce a random world of J_ϕ, call it W_ϕ. Now, consider a tuple $t \in J - J_\phi$, let $K(t) = \left\{t' \mid t'[K] = t[K]\right\} = \{t_1, \ldots, t_m\}$, i.e., tuples distinct from t that share a key with t. If $K(t) \cap W_\phi \neq \emptyset$, then t cannot appear in this world because it is disjoint from the set of $K(t)$. Otherwise, $K(t) \cap W_\phi = \emptyset$, and let $K(t) - J_\phi = \{t_1, \ldots, t_k\}$ (without loss) with marginal probabilities $p_1, \ldots, p_k$, i.e., those key tuples not in $\mathbf{tup}^+(\phi)$. These tuples do not affect the value so all that matters is adding them with the correct probability, which is easily seen to be the conditional probability:

$$\Pr[t_i \text{ is included }] = \frac{p_i}{1 - \sum_{j=k+1,\ldots,m} p_j}$$

This conditional simply says that it is conditioned on *none* of the tuples in $K(t) \cap J_\phi$ appearing. This is exactly Algorithm 5.2.2

The main result We now state the main technical result of this section: It follows directly from the lemma above:

Theorem 7 *Let q be a safe conjunctive query, then Algorithm 5.2.1 is a random generator for any BID instance J and annotation τ.*

An immediate consequences of Theorem. 7 is that if the semiring S does not contain too many elements, then Algorithm 5.2.1 solves the random possible world generation problem.

Corollary 2 *If q is safe and $|S| = \text{poly}(|J|)$, then Algorithm 5.2.1 solves the random possible world generation problem in time* $\text{poly}(|J|)$.

We use this corollary in the next section to design an FPTRAS for SUM.

6 Approximating **HAVING** queries with **MIN**, **MAX** and **SUM**

In this section, we study the problem of approximating HAVING queries. First, we describe an FPTRAS for having queries that have $\alpha = \text{MIN}$ where the test condition is $<$ or $\leq$, or $\alpha = \text{MAX}$ where the condition is one of $\{\geq, >\}$. This first FPTRAS applies to arbitrary such HAVING queries, including queries whose skeleton is unsafe. Second, we describe an FPTRAS for HAVING queries whose skeleton is safe, whose aggregate is SUM, and where the test condition is any of $<$, $\leq$, $>$, or $\geq$.

Our FPTRAS for SUM uses the random possible world generator of the previous section. These FPTRASes apply to a class of queries that we call (α, θ)-*apx-safe*. Additionally, we study the limits of any approach, and prove an approximation dichotomy for many (α, θ) pairs of HAVING queries without self-joins: either the above scheme is able to provide an FPTRAS and so the query is (α, θ)-apx-safe, or there is no FPTRAS: we call these queries (α, θ)-*hazardous*.[10]

6.1 Background: approximation of $\sharp\mathcal{P}$-hard problems

Although $\sharp\mathcal{P}$-problems are unlikely to be able to be solved exactly and efficiently, some problems have a strong approximation called a *Fully Polynomial Time Randomized Approximation Scheme* or FPTRAS [36], which is intuitively like a $1 + \varepsilon$ approximation.

Definition 19 Given function f that takes an input J and returns a number $\mathsf{f}(J) \in [0, 1]$, where J is a BID instance, we say that an algorithm $\mathcal{A}$ is an FPTRAS for f if given any $\delta > 0$, a confidence, and any $\varepsilon > 0$, an error, $\mathcal{A}$ takes J, ε, and δ as input and produces a number denoted $\tilde{\mathsf{f}}(J)$ such that

$$\Pr_{\mathcal{A}}[\left|\mathsf{f}(J) - \tilde{\mathsf{f}}(J)\right| \leq \varepsilon\, \mathsf{f}(J)] > 1 - \delta$$

where $\Pr_{\mathcal{A}}$ is taken over the random choices of the algorithm, $\mathcal{A}$. Further, $\mathcal{A}$ runs in time polynomial in ε^{-1}, $|W|$, and $\log \frac{1}{\delta}$.

[10] Formally, we mean that the $\sharp$BIS problem would have an FPTRAS, an unlikely outcome [16,17].

This definition asks for a *relative approximation* [36], which means that if f is exponentially small, but non-zero, our algorithm is required to return a non-zero value. This is in contrast to an *absolute approximation*, that is allowed to return 0 (and could be constructed using naïve random sampling). In this section, we fix a query Q and consider the function $f(J) = \mu_J(Q)$, where J is a BID instance. We study whether this function admits an FPTRAS.

We define three counting-like problems that are all $\sharp\mathcal{P}$-hard and will be of use later in this section:

Definition 20 The $\sharp$CLIQUE problem is given a graph (V, E), compute the fraction of the subsets of V that are cliques. The $\sharp$BIS problem is given a bipartite graph (U, V, E), compute the fraction of of the subsets of $U \times V$ that are independent sets. The $\sharp$KNAPSACK problem is given a set of positive integers $Y = \{y_1, \ldots, y_n\}$ and a positive integer value k, compute the fraction of sets $W \subseteq Y$ such that $\sum_{i \in W} y_i \leq k$.

All three problems are $\sharp\mathcal{P}$-hard[11]. In a celebrated result, Jerrum and Sinclair [47] showed that $\sharp$KNAPSACK *does* have an FPTRAS using a sophisticated Markov Chain Monte Carlo technique. It is believed that neither $\sharp$CLIQUE nor $\sharp$BIS have an FPTRAS. Interestingly, they are not equally hard to approximate (see Dyer et al. [16]). In particular, $\sharp$BIS is a complete problem with respect to *approximation preserving reductions*. We do not need these reductions in their full generality, and simply observe that polynomial time computable 1–1 reductions (bijections) are approximation preserving. In this section, we say that a problem is $\sharp$BIS-hard if there is a 1–1, polynomial–time reduction to $\sharp$BIS.

The $\sharp$KNAPSACK problem is related to the problem of computing HAVING queries with the aggregate functions SUM on a single table.

6.2 An FPTRAS for MIN with $\{\leq, <\}$ and MAX with $\{\geq, >\}$

Consider a query $Q[\text{MIN}(y) \leq k] :- g_1, \ldots, g_l$ then an equivalent condition to $W \models Q$ is that $W \models q'$ where $q' :- g_1, \ldots, g_l, y \leq k$. In other words, Q is equivalent to a conjunctive query, q', that contains an inequality predicate. As such, the standard algorithm for conjunctive queries on probabilistic databases [12,20,41] based on Karp–Luby [29] can be used. A symmetric argument can be used to find an FPTRAS for the aggregate test (MAX, $\geq$). Thus, we get essentially for free the following theorem:

Theorem 8 *If $(\alpha, \theta) \in \{(\mathit{MIN}, \leq), (\mathit{MIN}, <), (\mathit{MAX}, \geq)\}$, $\{(\mathit{MAX}, >)\}$ then $Q[\alpha(y)\ \theta\ k]$ has an* FPTRAS.

[11] We mean here that there is a 1–1 correspondence with the counting variants of these problems, which are canonical $\sharp\mathcal{P}$-complete problems.

Although this theorem is easy to obtain, it is interesting to note that $Q[\texttt{MIN}(y) > k]$ has an FPTRAS *only* if $\mathbf{sk}(Q)$ is safe (as we show in Lemma 8). If $\mathbf{sk}(Q)$ is safe, then, we can compute its value exactly, so the FPTRAS is not very helpful. In contrast, Theorem 8 has no such restriction – $\mathbf{sk}(Q)$ can be an arbitrary conjunctive query. This is a striking example that approximation complexity may be more subtle than exact evaluation. In particular, an analog of Lemma 2 does not hold.

6.3 An FPTRAS for safe queries using SUM with $\{<, \leq, \geq, >\}$

The key idea of the FPTRAS is based on a generalization of Dyer's observation: for some $k \geq 0$, the query $Q[\texttt{SUM}(y) \leq k]$ is only hard to compute if k is very large. If k is small, i.e., polynomial in the instance size, then we can compute Q exactly. Dyer's idea is to *scale and round down the values, so that the y-values are small enough for exact computation*. The cost of rounding is that it introduces some spurious solutions, but not too many. In particular, the fraction of rounded solutions is large enough that if we can sample from the rounded solutions, then we can efficiently estimate the fraction of original solutions inside the rounded solutions.

To perform the sampling, we use Algorithm 5.2.1 from the previous section (via Algorithm 6.3.2). Pseudo-code for the entire FPTRAS is shown in Fig. 6.3.1. We show only $(\texttt{SUM}, \leq)$ in detail, and explain informally how to extend to the other inequalities at the end of the section.

Theorem 9 *Let Q be a* HAVING *query $Q[\texttt{SUM}(y)\ \theta\ k]$ such that $\theta \in \{\leq, <\}$ and $\mathbf{sk}(Q)$ is safe, then Algorithm 6.3.1 is an* FPTRAS *for Q.*

It is interesting to note that Theorem 9 implies that we can efficiently evaluate a *much larger* set of queries than the previous, complete exact algorithm (albeit only in an approximate sense). In particular, only a very restricted class of SUM-safe queries can be processed efficiently and exactly (cf. Definition 15).

Our algorithm makes two technical assumptions: (1) in this section, unlike the rest of the paper, our semantics differ from SQL: In standard SQL, for a Boolean HAVING query q, if no tuples are returned by $\mathbf{sk}(Q)$ then $Q[\texttt{SUM}(y) \leq k]$ is false. In contrast, in this section, we assume that $Q[\texttt{SUM}(y) \leq 1]$ is true, even if $\mathbf{sk}(Q) = q$ is false, i.e., we choose the mathematical convention $\sum_{y \in \mathcal{Y}} = 0$, over SQL's choice, and (2) we make a *bounded odds* assumption:[12] for any tuple t there is exists $\beta > 1$ such that $\beta^{-1} \leq \frac{p_t}{1-p_t} \leq \beta$. These technical restrictions can be relaxed, but are chosen to simplify our analysis.

[12] For example, that this rules our $p_t = 1$ for any tuple and allows any tuple to not be present with some probability.

Algorithm 6.3.1 An FPTRAS for SUM

```
Decl: SAMPLE($Q$: a query $Q[\texttt{SUM}(y) \leq k]$ with a safe skeleton $q$,
        an instance $I$, a confidence $\delta$ and error $\varepsilon$)
    returns estimate of $\mu_I(Q)$.

Let body$(q) = \{g_1, \ldots, g_l\}$ and $n_i = |\mathbf{pred}(g_i)|$, i.e., the size of the $i$th relation.
Let $n = \prod_{i=1,\ldots,l} n_i$.
Let $Q^R[\texttt{SUM}(y) \leq n^2]$ with the same body as $q$ (see below).
Let $\tau^R(y) = \lfloor \frac{n^2 y}{k} \rfloor$ and $y > k \mapsto n^2 + 1$
Construct an expression parse tree, $\phi = \phi(P, I, \tau^R)$ where $P$ is a plan for $q^R$.
For $i = 1, \ldots m$ $W_i \leftarrow$ SAMPLEHELPER$(\phi, k)$
(* Run $m$ samples, for $m$ a polynomial in $\delta, \varepsilon, n$ *)
return $\frac{|\{W_i | W_i \models Q^O\}|}{m} * \mu(Q^R)$ (* $\approx \frac{\mu(Q^O)}{\mu(Q^R)}\mu(Q^R) = \mu(Q^O)$ *)
(* Compute fraction of $W_i$ that satisfy the original query $Q^O$. *)
```

Algorithm 6.3.2 Sampling Helper Routine

```
Decl: SAMPLEHELPER($\phi$: safe aggregate expression, $b$: a bound)
    returns a world

Select $s \in 0, \ldots, b$ with probability $\frac{m^\phi[s]}{\sum_{s'} m^\phi[s']}$.
(* Select a final value for the query that is less than the bound $b$ *)
return RANDOMWORLD$(\phi, s)$
```

6.3.1 The rounding phase

The goal of the rounding phase is to produce a query and an annotation function that rounds the values in the instance down enough so that (1) the exact processing algorithms of Sect. 4.2 for SUM queries can be used, and (2) we can randomly generate a world using the algorithm of Sect. 5. Algorithm 6.3.2 shows pseudo code for how these two steps are put together. The main result of this section is that the resulting algorithm is efficient (runs in time polynomial in the size of the BID instance J).

To get there, we construct two things: (1) an annotation function, τ^R, to do the rounding and (2) a query, Q^R, that uses the annotation function τ^R and the semiring $\mathbb{S}_{n^2+1}$ to compute the exact distribution of the rounded sum in polynomial time.

The Annotation Function. Let g be the first subgoal of q such that $\mathbf{var}(g) \ni y$ and $R = \mathbf{pred}(g)$, i.e., R is some relation containing y values. We scale down the values of y in R via the (rounding) annotation function denoted τ^R. Let $n = \prod_{g \in \mathbf{goal}(q)} |\mathbf{pred}(g)|$, i.e., the product of the sizes of all relations in q. Observe that n is polynomial in the instance size.[13] The rounded annotation function maps into the much smaller, rounded semiring $S^R = \mathbb{S}_{n^2+1}$. We define the rounded annotation function τ^R to agree everywhere with the original annotation function τ^O, *except* on g (the R rela-

[13] Recall that the query is fixed so if the database contains m tuples then $n = m^{O(1)}$ where $O(1)$ hides a constant depending only on q, e.g., the number of subgoals suffices.

tion): Here, $\tau_g^O(t) = t[y]$. In the rounded annotation function, we have $\tau_g^R(t) = \lfloor \frac{n^2}{k} t[y] \rfloor$, i.e., the y values are scaled down by a factor of n^2/k and rounded-down to the next highest integer. Additionally, if $t[y]$ is greater than k, then $\tau^R(t) = n^2 + 1$. Intuitively, this mapping is correct since if such a tuple is in the output of the query, then we are sure the summation is greater than k.

The Query. We construct a rounded query $Q^R[\mathtt{SUM}(y) \leq n^2]$ with the same body as Q^O. Let q be the skeleton of both Q^O and Q^R, i.e., $q = \mathbf{sk}(Q^R) = \mathbf{sk}(Q^O)$. We observe that since n^2 is polynomial in the instance size, the generic semiring algorithm of Sect. 4.2 can be used to compute the entire distribution $q(W, \tau^R)$ *exactly* in time polynomial in the size of the instance. Since we will always use Q^R with the rounded annotation function it makes sense to write $W \models Q^R$ if $q(W, \tau_R) \leq n^2$. Similarly, we will always use Q^O with the original annotation function so that it makes sense to write $W \models Q^O$ if $q(W, \tau_O) \leq k$.

Definition 21 Let W be a possible world from some BID instance J. If $W \models Q^O$, then we call W an *original solution*. If $W \models Q^R$ then we call W a *rounded solution*. Further, denote the set of original solutions with W_J^O and rounded solutions with W_J^R:

$$W_J^O = \left\{ W \in \mathcal{W}_J \mid W \models Q^O \right\}$$
$$\text{and } W_J^R = \left\{ W \in \mathcal{W}_J \mid W \models Q^R \right\}$$

We drop the subscript J when the BID instance is clear from the context.

We observe an essential property of our scheme: all original solutions are rounded solutions, i.e., $W_J^O \subseteq W_J^R$. Formally,

Lemma 5 *For any possible world W, $W \models Q^O \implies W \models Q^R$, and more precisely, there exists a $\delta \in [0, n)$ such that $q(W, \tau^R) = q(W, \tau^O) - \delta$.*

Proof Let $q = \mathbf{sk}(Q^O) = \mathbf{sk}(Q^R)$ and $\mathcal{V}$ be the set of all valuations for q. Let W be a possible world:

$$\begin{aligned} W \models Q^O &\iff \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \tau^O(v(g)) \leq k \\ &\iff \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \frac{n^2}{k} \tau^O(v(g)) \leq n^2 \\ &\implies \sum_{v \in \mathcal{V}: \mathbf{im}(v) \subseteq W} \tau^R(v(g)) + \delta_v \leq n^2 \\ &\iff W \models Q^R \end{aligned}$$

Here, $\delta_v \in [0, 1)$ and accounts for the round-off of the floor function. Since $0 \leq \sum_v \delta_v < n$, we have the more precise statement.

The importance of this lemma is that by sampling within the rounded solutions, we have a chance of hitting *any* original solution. Let $\tilde{W}$ be a random rounded solution created using Algorithm 6.3.2, then let f be the Boolean-valued estimator (random variable) that takes value 1 iff $\tilde{W} \models Q^O$. It is not hard to see that this estimator satisfies:

$$\mathbf{E}_{\mathcal{A}}[\mathsf{f}] = \frac{\mu_J\left(W^O\right)}{\mu_J\left(W^R\right)}$$

Here, $\mathcal{A}$ is written to emphasize that the expectation is taken with respect to the (random) choices of Algorithm 6.3.2. Importantly, this is exactly an individual trial of Algorithm 6.3.1.

6.3.2 Analysis of the convergence

Using Algorithm 6.3.2, we can efficiently conduct an individual (random) trial. The last technical piece to show that Algorithm 6.3.1 is an FPTRAS, is to show that the number of trials m that are needed to guarantee that the estimator converges is small enough, i.e., $m = \text{poly}(|J|)$. The first lemma that we need is the standard $\{0, 1\}$-estimator lemma [36], which is an application of a Chernoff Bound.

Lemma 6 **([36])** *Let $m > 0$ be an integer. Given a sequence of independent Boolean-valued ($\{0, 1\}$) random variables $\mathsf{f}_1, \ldots, \mathsf{f}_m$ with mean $\mathbf{E}[\mathsf{f}]$, then the estimator*

$$\mathsf{f}_m = \frac{1}{m} \sum_{i=1,m} \mathsf{f}_i$$

achieves a relative error of ε with probability $1 - \delta$ for some $m = O(\mathbf{E}[\mathsf{f}]^{-1} \varepsilon^{-2} \log \delta^{-1})$.

Observe that the estimator used in Algorithm 6.3.1 is exactly of this type. The second lemma that we need is that the probability mass of the original solutions contained in the rounded solutions is "big enough" so that our sampling scheme will converge quickly.

Lemma 7 *Let Q^R and Q^O defined as above, J be a BID instance, and μ_J be J's induced probability measure, then,*

$$(n+1)^{-1} \beta^{-1} \leq \frac{\mu_J(W^O)}{\mu_J(W^R)} \leq 1$$

where $n = \prod_{g \in \mathbf{goal}(q)} |\mathbf{pred}(g)|$.

This lemma is the technical heart of the argument: it intuitively places bounds on the variance of our estimate. We give a full proof in Appendix D. The importance of Lemma 7 is that it shows that $\mathbf{E}[\mathsf{f}] = \frac{\mu_J(W^O)}{\mu_J(W^R)} \geq n^{-1}\beta$, and so, applying Lemma 6, we see that we need at most $m = O(n\beta^{-1}\varepsilon^{-2} \log \delta^{-1})$ samples. We observe that a relative estimate for $\mathbf{E}[\mathsf{f}]$ implies that we have a relative estimate for $\mathbf{E}[\mathsf{f}]\mu_J(W^R) = \mu_J(W^O)$, the probability that we want to estimate. Thus, the algorithm is efficient as long as the the number of samples is bounded by a polynomial in $|J|$; a sufficient condition for this to hold is $\beta^{-1} = \text{poly}(|J|)$ which follows from the bounded odds assumption. Thus, under the bounded odds assumption with $\beta = \text{poly}(|J|)$, we have:

Theorem 10 *Let Q be a* `HAVING` *query $Q[\alpha\ \theta\ k]$ with $\alpha =$* `SUM` *and $\theta \in \{<, \leq, >, \geq\}$, if the skeleton of Q is safe then Q has an* FPTRAS.

Extending to Other Inequalities. A virtually identical argument shows that $\theta = `<'$ has an FPTRAS. To see that $\geq$ has an FPTRAS with `SUM` on tuple independent database, the key observation is that we can compute a number $M = \max_W q(W, \tau)$. Then, we create a new BID instance $\bar{J}$ where each tuple $t \in J$, we map t to t' where $t = t'$ except that $t[P] = 1 - p$. We then ask the query $Q[\texttt{SUM}(y) < M - k]$, which is satisfied precisely on a world W when $Q[\texttt{SUM}(y) \geq k]$.

6.4 The limits of any approach and a dichotomy

We now study the limit of any approach to approximating `HAVING` queries. We see two interesting phenomenon: (1) the approximation depends not only the aggregate, but also the test. For example, $Q[\texttt{MIN} \leq k]$ has an FPTRAS while, in general, $Q[\texttt{MIN}(y) \geq k]$ does not. (2) The introduction of self-joins results in problems that are believed to be harder to approximate than those without self-joins; this suggests a more interesting complexity landscape for approximate query evaluation than exact query evaluation [17].

In this section, we only consider $\theta \in \{=, <, \leq, >, \geq\}$, i.e., we omit $\neq$ from consideration. To compactly specify aggregate tests, e.g., $(\texttt{MIN}, >)$, we write (α, Θ_0) where α is an aggregate and Θ_0 is a set of tests, i.e., $\Theta_0 \subseteq \{=, <, \leq, >, \geq\} = \Theta$; (α, Θ_0) is a short hand for the set $\bigcup_{\theta \in \Theta_0} \{(\alpha, \theta)\}$. We let $\Theta_{\leq} = \{\leq, <, =\}$ and $\Theta_{\geq} = \{\geq, >, =\}$. With this notation, we can state our first lemma.

Lemma 8 *Let (α, θ) be in $\{(\mathit{MIN}, \Theta_{\geq}),\ (\mathit{MAX}, \Theta_{\leq}),\ (\mathit{COUNT}, \Theta_{\leq}),\ (\mathit{SUM}, \Theta_{\leq})\}$ then the following* `HAVING` *query is $\sharp$BIS-hard:*

$$Q_{BIS}[\alpha(y)\ \theta\ k] \text{ :- } R(x), S(x, y), T(y)$$

Let $Q[\alpha(y)\ \theta\ k]$ be a `HAVING` *query such that* $\mathbf{sk}(Q)$ *is not safe and consider only tuple-independent databases, then Q is $\sharp$BIS-hard.*

The second statement identifies the precise boundary of hardness for approximation over tuple independent databases.

Proof We give a general construction that will be used in every reduction used to prove that $Q_{\texttt{BIS}}$ is $\sharp$`BIS`-hard. Given a bipartite graph (U, V, E), we create an instance of three relations R, S and T. The skeleton of our query in the reduction is $R(x), S(x, y), T(y)$. Without loss, we assume that U, V are labeled from $1, \ldots |U| + |V|$. Here, we encode a bipartite graph with $u \in U \mapsto R(u)$ and $v \in V \mapsto T(v)$, we assign each of these tuples probability 0.5. We let S encode E. It is not hard to see that there is a bijection between possible worlds and subsets of the graph. In particular, if a possible world corresponds to an independent set then no tuples are returned. We now add in a deterministic set of tuples, i.e., all probabilities are 1, as $\{R(a), S(a, a), T(a)\}$ for some a that we will set below. These tuples are always present in the answer. Actually, *only* these tuples are present in the output if and only if this world encodes a bipartite independent set.

To see the reduction for `MAX`, set $a = 0$. We observe that $\texttt{MAX}(y) \leq 0$ if and only if the only tuple returned are the 0 tuples, i.e., a bipartite independent set. For `MIN` let $a = |U| + |V| + 1$, now check if $\texttt{MIN}(y) \geq a$. The $\texttt{COUNT}(y) \leq 1$ if only the a tuples are present. Similarly, `SUM` follows by setting $a = 1$ and ensuring all values are encoded higher. Thus, the bijection of the solution sets is the same.

Claim (2), that this reduction works for any unsafe query, follows by a result of Dalvi and Suciu [12] that shows that if a skeleton is not safe over tuple independent databases, it must *always* contain the $R(x), S(x, y), T(y)$ pattern used in this reduction. All other relations can contain a single tuple. This works because our reductions do not care about where the distinguished variable y falls, so we can set everything to 1 (or 0) in another relation.

As a consequence of this lemma, the positive results of this paper, and the completeness of the safe plan algorithm of Dalvi and Suciu [12], we have the following:

Theorem 11 *Assume that $\sharp$BIS does* not *have an* FPTRAS. *Let (α, θ) be in $\{(\mathit{MIN}, \Theta), (\mathit{MAX}, \Theta), (\mathit{COUNT}, \Theta_{\leq})\}$, $\{(\mathit{SUM}, \Theta_{\leq})\}$ then for any* `HAVING` *query $Q[\alpha(y)\ \theta\ k]$ over a tuple independent database J, either* (1) *the query evaluation problem can be approximated in randomized polynomial time and we call it* (α, θ)-apx-safe or (2) *the query evaluation problem does not have an* FPTRAS *and we call it* (α, θ)-hazardous. *Further, we can decide in which case Q falls in polynomial time.*

In some cases deciding in which case a query falls is trivial, e.g., a $(\texttt{MIN}, \leq)$ `HAVING` query is always (α, θ)-safe. In the cases that the decision is non-trivial, we can reuse the safe plan algorithm of Dalvi and Suciu [12] [applied to $\mathbf{sk}(Q)$]. An immediate consequence of this dichotomy is a trichotomy which is obtained by combining the relevant theorem from Sect. 4 with the above theorem. For example, to get a trichotomy for the class of $(\texttt{COUNT}, \leq)$-`HAVING` queries, we combine Theorem 2 with the above theorem.

It is interesting to note that our positive algorithms work for arbitrary safe plans over BID databases. However, it is

$\mathbf{sk}(Q)$	$(\texttt{MIN}, \Theta_<), (\texttt{MAX}, \Theta_>)$	$(\texttt{MIN}, \Theta_>), (\texttt{MAX}, \Theta_<), (\texttt{COUNT}, \Theta)$
safe	safe ($\mathcal{P}$, Thm. 1)	safe ($\mathcal{P}$, Thm. 1)
not safe	apx-safe (FPTRAS, Thm. 8))	hazardous (no FPTRAS, Thm. 11)

Fig. 7 Summary of results for MIN, MAX and COUNT. They form a trichotomy over tuple independent databases

not immediately clear that the known hardness reductions (based on polynomial interpolation [13]), can be used to prove approximate hardness. Further, we leave the case of COUNT and SUM with open.

7 Summary of results

Figure 7 summarizes our results for MIN, MAX and COUNT. If we restrict to HAVING queries over tuple independent instances the lines of the table are crisp and form a trichotomy: any such HAVING query cleanly falls into exactly one bucket. The positive results we have shown hold for all BID database. Over general BID databases, however, we have only established the weaker negative result that there exists *some* hard query when the skeleton is unsafe[14]. For example, the query $R(x; y), S(y)$ is known to be $\sharp\mathcal{P}$-hard [13]. Our results show that $Q[\texttt{COUNT}(y) \geq y]$:–$R(x; y), S(y)$ is $\sharp\mathcal{P}$-hard, but leave open whether it has an FPTRAS.

The state of the results with $(\texttt{SUM}, <)$ over tuple-independent databases is more interesting: if Q is SUM-safe, then its evaluation is in $\mathcal{P}$-time (Theorem 5). If Q is not SUM-safe, but $\mathbf{sk}(Q)$ is safe then Q is $\sharp\mathcal{P}$-hard (Theorem 6), but does admit an FPTRAS (Theorem 10). We call Q $(\texttt{SUM}, <)$-apx-safe. If $\mathbf{sk}(Q)$ is not safe, then evaluating Q is $\sharp$BIS-hard (Theorem 11), and so likely has no FPTRAS. We call Q $(\texttt{SUM}, <)$-hazardous. We now show that with the addition of self-joins, even a simple pattern becomes as hard to *approximate* as $\sharp$CLIQUE, which is as hard to approximate as any problem in $\sharp\mathcal{P}$. This is interesting because it points out that the complexity of approximation may be richer than we have explored in this paper:

Lemma 9 *Let* (α, θ) *be in* $\{(MIN, \Theta_{\leq}), (MAX, \Theta_{\geq}), (COUNT, \Theta_{\leq}), (SUM, \leq)\}$ *and consider the HAVING query:*

$$Q_{CLIQUE}[\alpha(y)\ \theta\ k] \text{ :–} R(x), S(x, y), R(x)$$

then Q_{CLIQUE} *is as hard to approximate as* $\sharp$*CLIQUE.*

Proof The input instance of $\sharp$CLIQUE is $G = (V, E)$: for each $v \in V$, we create a tuple $R(v)$ that has probability $\frac{1}{2}$ and E encodes exactly the *complement* (symmetrically closed) edge relation; here, $(v, v) \notin E$. Notice that a possible world is simply a subset of R. If in a possible world, $q = \mathbf{sk}(Q)$ is satisfied then, this implies there is some pair of nodes (u, v) that are not connected by an edge in G and so W does not represent a clique. Hence the query is false precisely when $\sharp$CLIQUE is true. Using exactly the same encoding as we used in the previous proof, we can then test the probability of this condition.

[14] It is an open problem to extend these results to all BID databases.

8 Related work

Probabilistic relational databases have been discussed by Barbara et al. [4], the ProbView system [44], and more recently, by Dalvi and Suciu [12], Ré et al. [41], Sen et al. [46], MayBMS [2], MCDB [25], Orion [10], and Trio [51]. Trio, Mystiq, and MayBMS approaches have representations that are similar to BID tables, so our results could be applied to these systems. Currently, all of these approaches omit HAVING style aggregation. MCDB [25] does have richer aggregation queries, but takes a statistical approach and so does not provide formal guarantees.

Koch [30] formalizes a language that allows predication on probabilities and discusses approximation algorithms for this richer language, though he does not consider HAVING aggregation. This is in part due to the fact that his aim is to create a fully compositional language for probabilistic databases [31]. Extending our style of aggregation to a fully compositional language is an interesting open problem.

Soliman et al. [48] consider combining top-k with measures, such as SUM, which is similar in spirit to HAVING. Their correlation model allows more complex distributions to be specified much more succinctly, but they do not focus on complex queries involving joins. Combining ranking with HAVING queries is a powerful, but currently unexplored idea.

Cheng et al. [10] and Desphande et al. [15] consider probabilistic databases resulting from sensor networks so that the database models continuous values, such as temperature. The focus here is on rich correlation models, but simpler querying. In this settings, the natural aggregation queries are effectively over a singe relation. In this work, we consider a richer class of aggregation queries, but with simpler probabilistic models.

The problem of generating a random world that satisfies a constraint is fundamental and is considered by Cohen et al. [11]. They point out that many applications for this task, and use it to answer rich queries on probabilistic XML databases. In this paper, we differ in the constraint language we choose and that we use our sampling algorithm as a basis for an FPTRAS. There is also a connection to the recent work of Koch and Olteanu on *conditioning a probabilistic database* [32] who recognize the fundamental importance of finding worlds conditioned on constraints, though they do not consider constraints with SQL aggregation.

In the OLAP setting, Burdick et al. [6,7] give efficient algorithms for *value aggregation* in a model that is equivalent to the single table model. Their focus is on the semantics of the problem. As such, they consider how to assign the correct probabilities, called *the allocation problem*, and handling constraints in the data. The allocation problem is an interesting and important problem. Our problem is orthogonal: we assume the database has been specified and focus on query evaluation.

Ross et al. [44] describe an approach to computing aggregates on a probabilistic database, by computing bounding intervals (e.g., the AVG is between $[5, 600, 5, 700]$). They consider a richer class of aggregation functions than we discuss, but with an incomparable semantics. Their complexity results show that computing bounding intervals exactly is $\mathcal{NP}$-Hard. In contrast, we are interested in a more fine-grained static analysis: our goal is to find the syntactic boundary of hardness. Trio also uses a bounded interval style approach [37].

There is work on value aggregation on a streaming probabilistic databases [26]. In addition, they consider computing value approximations aggregates, such as AVG, in a streaming manner. In contrast, computing the AVG for predicate aggregates (as we do in this paper) on a single table is $\sharp\mathcal{P}$-Hard. One way to put these results together is that computing a value aggregate is the first moment (expectation) while a HAVING aggregate allows us to capture the complete distribution (in the exact case). Kanagal and Deshpande [28] also work in the streaming context of aggregation that computes an expected value style of aggregation. This work does not look at complex queries, like joins.

Arenas et al. [3] consider the closely related problem of the complexity of aggregate queries, similar to HAVING queries, over data which violates functional dependencies. They do not consider a probabilistic semantic, but instead consider a semantic based on incomplete databases based on repairs [5]. Hence, the query semantic of this work is greatest lower bound or least upper bound on the set of all minimal repairs. They also consider multiple predicates, which we leave for future work. There is a deep relationship between the repair semantics and probabilistic approaches. A representative work in this direction is Andristos et al. [1].

Our trichotomy results are based on the conjecture that $\sharp$BIS does not have an FPTRAS. Evidence of this conjecture is given by Dyer [16,17] by establishing that this problem is complete for a class of problems with respect to *approximation preserving reductions*. At this point, it would be fair to say that this conjecture is less well established than $\sharp\mathcal{P} \neq \mathcal{P}$. Any positive progress, i.e., showing that $\sharp$BIS does have an FPTRAS, could be adapted to our setting. As we have shown, some problems are as hard to approximate as any problem in $\sharp\mathcal{P}$, e.g., as hard as $\sharp$CLIQUE. An interesting open problem is to find if there is a corresponding syntactic boundary of hardness: is it true that either a query is $\sharp$BIS-easy or $\sharp$CLIQUE-hard? We conjecture that such a syntactic boundary exists, though it remains open.

The EXISTS results of this paper are from Dalvi and Suciu [12], who later proved a more general dichotomy results (allowing queries with self-joins) [14]. Recently, Olteanu et al. [38] showed a dichotomy for conjunctive queries with inequality predicates. All of these results rely intimately on the BID model. This model is complete if views are added [22,41], but alternate approaches (such as graphical models) may express some distributions much more succinctly. A more succinct representation tends usually raise the complexity of any problem. It is an open question how to extend these results to more succinct models such as considered by Sen et al. [46], Kanagal et al. [28], or to sample-based models models [25].

9 Conclusion

In this paper, we examine the complexity of evaluating positive conjunctive queries with predicate aggregates over probabilistic databases called HAVING queries. For each aggregate, we discuss a novel method to evaluate these queries. Our method is based on computing the distribution of random variables in a semiring. We prove that for conjunctive queries without self-joins our methods are optimal. Additionally, we study the problem of generating a random world that satisfies a semiring element and provide an efficient solution. We apply this sampling algorithm as a subroutine to design an approximation for queries that are hard to compute exactly, thus expanding the border of known tractable cases. We show that our approximations capture efficient approximation.

Appendix A: Properties of safe plans

We formalize the properties that hold in safe extensional plans in the following proposition:

Proposition 7 *Let $P = \pi^I_{-x} P_1$ be a safe plan then for any tuples $t_1, \ldots, t_n \in \mathbb{D}^{|\mathbf{var}(P_1)|}$ that disagree on x, i.e., such that $i \neq j$ implies that $t_i[\mathbf{var}(P)] = t_j[\mathbf{var}(P)]$ and $t_i[x] \neq t_j[x]$ and then for any $s_1, \ldots, s_n \in S$ we have independence, that is the following equation holds:*

$$\mu\left(\bigwedge_{i=1,\ldots,n} \omega_{P_1,S}(t_i) = s_i\right) = \prod_{i=1,\ldots,n} \mu\left(\omega_{P_1,S}(t_i) = s_i\right) \tag{1}$$

Similarly, let $P = \pi^D P_1$ then we have disjointness:

$$\mu\left(\bigwedge_{i=1,\ldots,n} \omega_{P_1,S}(t_i) = 0\right) = \sum_{i=1,\ldots,n} \mu\left(\omega_{P_1,S}(t_i) = 0\right) - (n-1) \tag{2}$$

Let $P = P_1 \bowtie P_2$, then for any tuples $t_i \in \mathbb{D}^{|\mathbf{var}(P_i)|}$ for $i = 1, 2$ then and $s_1, s_2 \in S$, we have independence:

$$\mu\left(\omega_{P_1,S}(t_1) = s_1 \wedge \omega_{P_1,S}(t_2) = s_2\right) \\ = \mu\left(\omega_{P_1,S}(t_1) = s_1\right)\mu\left(\omega_{P_1,S}(t_2) = s_2\right) \qquad (3)$$

Proof We prove Eq. 1. To see this observe that, directly from the definition, the set of tuples that contribute to t_i and t_j ($i \neq j$) do not share the same value for a key in *any* relation. It is not hard to see that t_i and t_j are functions of independent tuples, hence are independent. The equation then follows by definition of independence.

We prove Eq. 2. Assume for contradiction that the tuples are not disjoint, that is there exists some possible world W such that for some $i \neq j$ $\{t_i, t_j\} \subseteq W$. By the definition, there must exist some key goal g such that $\mathbf{key}(g) \subseteq \mathbf{var}(P)$. Thus, for t_i and t_j to be present in W it must be that there are two distinct tuples with the same key value—but different values for the attribute corresponding to x. This is a contradiction to the key condition, hence the tuples are disjoint and the equation follows.

We prove Eq. 3. In a safe plan, $\mathbf{goal}(P_1) \cap \mathbf{goal}(P_2) = \emptyset$ and distinct relations are independent. As a result, the tuples themselves are independent.

Appendix B: Full proof for COUNT(DISTINCT)

Theorem 12 [Restatement of Theorem. 3] *Let Q be 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 that P_1 is a descendant in P of P_2 (alternatively, containment). Let P_y be a subplan which satisfies $P_{-y} = \pi^I_{-y}(P') \prec P$ or $P_{-y} = \pi^D_{-y}(P') \prec P$. P_{-y} is a safe plan, hence S-safe for $S = \mathbb{Z}_2$, i.e., the EXISTS algebra. For each t, 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 $\hat{\omega}^J_{P_0}(t) = \coprod_{t' \in P_1} \hat{\omega}^J_{P_1}(t)$; 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:

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

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 annihilator 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.

Complexity

Proposition 8 [Second Half of Proposition 4] *The following HAVING queries are $\sharp\mathcal{P}$-hard for $i \geq 1$:*

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

Proof We start with $i = 1$. The hardness of $Q_{2,i}$ 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 observe a simple fact:

Proposition 9 *Let Q be 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 that there is some projection independent π^I_{-x} on all variables.

Definition 22 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

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

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

Proposition 10 *If q is safe and $x \in F^q_\infty$ then there is a safe plan P such that $\pi^D_{-x} \in P$ and for all ancestors of π^D_{-x} they are either $\pi^D_{-z} P_1$ for some z or $P_1 \bowtie P_2$.*

Proof Consider the smallest query q such that the proposition 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 10 *Let Q be a* HAVING *query using* COUNT(DISTINCT) *such that $q = \mathbf{sk}(Q)$ is safe, but Q is not* COUNT(DISTINCT)*-safe; and let there be 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 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 11 *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. Similarly, 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 [Proposition 10] By Proposition 9, 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 $R(x; y)$ which is the hard pattern. Further, it is easy to see that $g \Rightarrow^* R_i(x; y)$ for some i.

We can now prove the main result:

Lemma 11 *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 Lemma 10. 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.

Appendix C: Full Proofs for SUM and AVG

AVG hardness

Definition 23 Given a set of nonnegative integers $a_1, \ldots, a_n$, the $\sharp$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 12 *$\sharp$NONNEGATIVE SUBSET-AVG is $\sharp\mathcal{P}$-hard.*

Proof We first observe that if we allow arbitrary integers, then we can reduce any $\sharp$NONNEGATIVE SUBSET-SUM with $B = 0$, which is $\sharp\mathcal{P}$-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 $\sum_{s \in S} a_s = 0$.

Proof of Theorem. 6

It is sufficient to show the following lemma:

Lemma 12 *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 $y_1, \dots, y_n$ let $q_i = q[y \to y_i]$ and $S \subseteq 1, \dots, n$ 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 $\mathbf{im}(q_i) \subseteq W$.*

Armed with his lemma we can always construct the distribution used in Proposition 5.

Proof We observe that $y \notin F_\infty^q$ else there would be a SUM- and AVG-safe plan by Proposition 10. Now consider the rewriting $q[x \to$ '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. Since there are no self-joins, this implies in any valuation either it must use a tuple containing y_i or the relation contains a single tuple with a for every attribute. Hence, the multiplicity of y_i is ≤ 1 in any possible world. Since there is only one relation with probabilistic tuples and all tuples have $\mu(t) = 0.5$, we have $\mu(\wedge_{s \in S} q_s) = 2^{-|S|}$ as required.

Proposition 13 *If $Q[$SUM$(y) = k]$ is not* SUM*-safe and on a tuple independent instance, then Q does not have an* FPTRAS.

Proof We observe that (SUM, $=$) is hard to approximate on even a single tuple-independent as a consequence of the previous reduction, which gives a one-to-one reduction showing (SUM, $=$) is as hard as
$\sharp$SUBSET-SUM, an $\mathcal{NP}$-hard problem and so has no FPTRAS.

Appendix D: Convergence Proof of Lemma 7

In the proof of this lemma, we need a technical proposition:

Proposition 14 *Let q be a conjunctive query without self-joins and R any relation contained in* $\mathbf{goal}(q)$*, then $q(W, \tau) = \sum_{t \in R} q((W - R) \cup \{t\}, \tau)$. Here, the summation is in the semiring S.*

Proof By definition, the value of the query $q(W)$ can be written as $q(W, \tau) = \sum_{v \in \mathcal{V}} \prod_{g \in g} \tau(v(g))$. Since q does not contain self-joins, each valuation contains exactly one member of R. Hence, there is a bijection between the between the two sums. Since semirings are associative, this completes the proof.

We can now prove Lemma 7.

Proof of Lemma 7 We first observe that $W^O \subseteq W^R$, by Lemma 5, which shows $\frac{\mu(W^O)}{\mu(W^R)} \leq 1$. To see the other inequality, we construct a function $f : W^R \to W^O$ such that for any $W \in W^O$, $\frac{\mu(W)}{\mu(f^{-1}(W))} \geq (n+1)^{-1}\beta^{-1}$. This is sufficient to prove the claim. We describe f: if $W \in W^O$ then $f(W) = W$ else, $W \in W^R - W^O$ then we show that there is a tuple $t \in W$ such that $W - \{t\} \in W^O$, $f(W) = W - \{t\}$. Since there are at most n possible tuples to remove, this shows that $\left|f^{-1}(W)\right| \leq (n+1)$, Using the bounded odds equation, we have that $\frac{\mu(W)}{\mu(f^{-1}(W))} \geq (n+1)^{-1}\beta^{-1}$. Thus, all that remains to be shown is that we can always find such a tuple, t.

Consider $W \in W^R - W^O$, which means that $q(W, \tau^O) > k$ and $q(W, \tau^R) \leq n^2$. There must exist a tuple t such that $q(W, \tau^O) - q(W - \{t\}, \tau^O) > k/n$ otherwise $q(W, \tau^O) \leq k$, which is a contradiction. To see this, consider any relation R in the query, we apply the above proposition to observe that:

$$\sum_{t \in R} q(W - \{t\}, \tau^O) = \sum_{t \in R} \sum_{s \in R : s \neq t} q(W - R \cup \{s\}, \tau^O)$$
$$= (|R| - 1) \sum_{t \in R} q(W - R \cup \{t\}, \tau^O)$$
$$= (|R| - 1) q(W, \tau^O)$$

The second to last equality follows by counting how many times each term appears in the summation and that the semiring is embeddable in the rational numbers ($\mathbb{Q}$).

$$q(W, \tau^O) - q(W - \{t\}, \tau^O) \leq k/n$$
$$\implies |R|\, q(W, \tau^O) + \sum_{t \in R} q(W - \{t\}, \tau^O) \leq k$$
$$\implies |R|\, q(W, \tau^O) + (|R| - 1) q(W, \tau^O) = q(W, \tau^O) \leq k$$

The second line follows by summing over t in R, using the previous equation, and using that $|R| \leq n$. Thus, we can conclude there is some t such that $q(W, \tau^O) - q(W - \{t\}, \tau^O) > k/n$. By Lemma 5 we have:

$$q(W, \tau^R) \leq n^2 \implies \frac{n^2}{k} q(W, \tau^R) - \delta \leq n^2$$

Where $\delta \in [0, n)$. In turn, this implies

$$q(W, \tau^O) \leq \frac{k}{n^2}\delta + k \leq k + \frac{k}{n}$$

Since, $q(W, \tau^O) - q(W - \{t\}, \tau^O) > k/n$, we have that $q(W - \{t\}, \tau^O) \leq k$ and so $W - \{t\} \models Q^O$ and hence, $W - \{t\} \in W^O$.

References

1. Andritsos, P., Fuxman, A., Miller, R.J.: Clean answers over dirty databases. In: ICDE (2006)
2. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: ICDE, pp. 983–992 (2008)

3. Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V., Spinrad, J.: Scalar aggregation in inconsistent databases. Theor. Comp. Sci. (2003)
4. Barbara, D., Garcia-Molina, H., Porter, D.: The management of probabilistic data. IEEE Trans. Knowl. Data Eng. **4**(5), 487–502 (1992)
5. Bertossi, L., Chomicki, J., Cortes, A., Gutierrez, C.: Consistent answers from integrated data sources. In: International Conference on Flexible Query Answering Systems (2002)
6. Burdick, D., Deshpande, P.M., Jayram, T.S., Ramakrishnan, R., Vaithyanathan, S.: Olap over uncertain and imprecise data. VLDB J. **16**(1), 123–144 (2007)
7. Burdick, D., Deshpande, P., Jayram, T.S., Ramakrishnan, R., Vaithyanathan, S.: Olap over uncertain and imprecise data. In VLDB, pp. 970–981 (2005)
8. Cafarella, M.J., Ré, C., Suciu, D., Etzioni, O.: Structured querying of web text data: a technical challenge. In: CIDR, pp. 225–234. http://www.crdrdb.org (2007)
9. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: ACM SIGMOD, San Diego, CA (2003)
10. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proceedings of SIGMOD03 (2003)
11. Cohen, S., Kimelfeld, B., Sagiv, Y.: Incorporating constraints in probabilistic xml. In: PODS, pp. 109–118 (2008)
12. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB, Toronto, Canada (2004)
13. Dalvi, N., Suciu, D.: Management of probabilisitic data: foundations and challenges. In: PODS, pp. 1–12 (2007)
14. Dalvi, N.N., Suciu, D.: The dichotomy of conjunctive queries on probabilistic structures. In: PODS, pp. 293–302 (2007)
15. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks (2004)
16. Dyer, M.E., Goldberg, L.A., Greenhill, C.S., Jerrum, M.: On the relative complexity of approximate counting problems. In: APPROX, pp. 108–119 (2000)
17. Dyer, M.E., Goldberg, L.A., Jerrum, M.: An approximation trichotomy for boolean #csp. CoRR, abs/0710.4272 (2007)
18. Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. In: Vardi, M.Y. (ed.) Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge, pp. 277–293. Morgan Kaufmann, San Francisco (1988)
19. Fuxman, A., Miller, R.J.: First-order query rewriting for inconsistent databases. In: ICDT, pp. 337–351 (2005)
20. Gradel, E., Gurevich, Yu., Hirch, C.: The complexity of query reliability. In: Symposium on Principles of Database Systems, pp. 227–234 (1998)
21. Green, T., Karvounarakis, G., Tannen, V.: Provenance semirings. In: PODS (2007)
22. Green, T.J., Tannen, V.: Models for incomplete and probabilistic information. IEEE Data Engineering Bulletin, vol 29 (2006)
23. Gupta, R., Sarawagi, S.: Curating probabilistic databases from information extraction models. In; Proceedings of the 32nd International Conference on Very Large Databases (VLDB) (2006)
24. Hernandez, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: SIGMOD Conference, pp. 127–138 (1995)
25. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a monte carlo approach to managing uncertain data. In: SIGMOD Conference, pp. 687–700 (2008)
26. Jayram, T.S., Kale, S., Vee, E.: Efficient aggregation algorithms for probabilistic data. In: SODA (2007)
27. Jayram, T.S., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar information extraction system. IEEE Data Engineering Bulletin, vol. **29**(1), (2006)
28. Kanagal, B., Deshpande, A.: Online filtering, smoothing and probabilistic modeling of streaming data. In: ICDE, pp. 1160–1169, (2008)
29. Karp, R.M., Luby, M.: Monte–carlo algorithms for enumeration and reliability problems. In: FOCS, pp. 56–64, (1983)
30. Koch, C.: Approximating predicates and expressive queries on probabilistic databases. In: PODS, pp. 99–108 (2008)
31. Koch, C.: A compositional query algebra for second-order logic and uncertain databases. In: Proceedings of ICDT (2009)
32. Koch, C., Olteanu, D.: Conditioning probabilistic databases. In: VLDB (2008)
33. Lakshmanan, L., Leone, N., Ross, R., Subrahmanian, V.S.: Probview: a flexible probabilistic database system. ACM Trans. Database Syst. 22(3), (1997)
34. Lang, S.: Algebra. Springer, Heidelberg (2002)
35. Mansuri, I., Sarawagi, S.: A system for integrating unstructured data into relational databases. In: Proceedings of ICDE 2006 (2006)
36. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1997)
37. Murthy, R., Ikeda, R., Widom, J.: Making aggregation work in uncertain and probabilistic databases. Technical Report 2007-7, Stanford InfoLab, June 2007
38. Olteanu, D., Huang, J., Koch, C.: SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases. In: Proc. of ICDE 2009 (2009)
39. Parag, A., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: a system for data uncertainty and lineage. In: VLDB (2006)
40. Ré, C., Dalvi, N., Suciu, D.: Query evaluation on probabilistic databases. IEEE Data Eng. Bull. **29**(1), 25–31 (2006)
41. Ré, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proceedings of ICDE (2007)
42. Ré, C., Suciu, D.: Materialized views in probabilsitic databases for information exchange and query optimization. In: VLDB (2007)
43. Ré, C., Suciu, D. Efficient evaluation of. In: DBPL, pp. 186–200 (2007)
44. Ross, R., Subrahmanian, V.S., Grant, J.: Aggregate operators in probabilistic databases. J. ACM **52**(1), 54–101 (2005)
45. Sarma, A.D., Benjelloun, O., Halevy, A.Y., Widom, J.: Working models for uncertain data. In: Liu, L., Reuter, A., Whang, K.-Y., Zhang, J. (eds.) ICDE, p. 7. IEEE Computer Society, Los Alamitos (2006)
46. Sen, P., Deshpande, A.: Representing and querying correlated tuples in probabilistic databases. In: Proceedings of ICDE (2007)
47. Sinclair, A., Jerrum, M.: Approximate counting, uniform generation and rapidly mixing markov chains. Inf. Comput. **82**(1), 93–133 (1989)
48. Soliman, M., Ilyas, I.F., Chang, K.C.-C.: Top-k query processing in uncertain databases. In: Proceedings of ICDE (2007)
49. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. **8**(3), 410–421 (1979)
50. Vardi, M.Y.: The complexity of relational query languages. In: Proceedings of 14th ACM SIGACT Symposium on the Theory of Computing, pp. 137–146, San Francisco, California (1982)
51. Widom, J.: Trio: a system for integrated management of data, accuracy, and lineage. In: CIDR, pp 262–276 (2005)
52. Winkler, W.E.: Improved decision rules in the fellegi-sunter model of record linkage. Technical report, Statistical Research Division, U.S. Census Bureau, Washington, DC (1993)
53. Winkler, W.E.: The state of record linkage and current research problems. Technical report, Statistical Research Division, US Bureau of the Census (1999)