# Queries with Difference on Probabilistic Databases

Sanjeev Khanna
University of Pennsylvania
Philadelphia, PA, USA
sanjeev@cis.upenn.edu

Sudeepa Roy
University of Pennsylvania
Philadelphia, PA, USA
sudeepa@cis.upenn.edu

Val Tannen
University of Pennsylvania
Philadelphia, PA, USA
val@cis.upenn.edu

## ABSTRACT

We study the feasibility of the exact and approximate computation of the probability of relational queries with difference on tuple-independent databases. We show that even the difference between two "safe" conjunctive queries without self-joins is "unsafe" for exact computation. We turn to approximation and design an FPRAS for a large class of relational queries with difference, limited by how difference is nested and by the nature of the subtracted subqueries. We give examples of inapproximable queries outside this class.

## 1. INTRODUCTION

Query evaluation on probabilistic databases has been studied for a long time (we refer to the recent overviews [12, 28, 8]). It eventually became known [20, 10] that there exist simple queries for which the exact computation of the probability of the answers is hard (specifically, $\#\mathcal{P}$-hard) even when database instances follow the simple *tuple-independent* model. In [10], Dalvi and Suciu showed a remarkable dichotomy for the class of *conjunctive queries* (which we denote CQ) but *without self-joins* (which we denote CQ$^-$) by separating them into "safe" queries for which answer probabilities can be computed in polynomial time and "unsafe" ones for which the computation is $\#\mathcal{P}$-hard. For CQ$^-$ the safe queries have a simple characterization. We do not need the characterization in this paper but it is instructive to note that it does not take much to destroy safety. The query $q$ in Figure 1b is safe but the query $q'$ in Figure 1c is not.

In an important series of papers they have then extended the dichotomy results both to the class of all *positive* queries (which can be described equivalently as the select-project-join-union (SPJU) queries or as the unions of conjunctive queries (UCQ)), on independent-tuple databases [11, 9] and to CQ$^-$ on disjoint-independent databases [12].

In this paper we work only with tuple-independent probabilistic databases. These are defined by tables such as those in Figure 1, where each tuple $t_i$ is associated with a probability $p_i \in [0, 1]$. We will also need to associate each tuple $t_i$ with a distinct boolean variable $u_i$ which can be thought of as a notation for the event that $t_i$ appears in a random instance, *independently* of the other tuples. Then $p_i = \Pr[u_i]$. We call the $u_i$'s *tuple variables*. For example in Figure 1a, the tuple variable $u_1$ denotes the event that the tuple $R(b_1, c_1)$ appears in a random instance defined by $I$ and the fraction 0.7 inside parentheses is the probability of that event (i.e. $\Pr[u_1] = 0.7$).

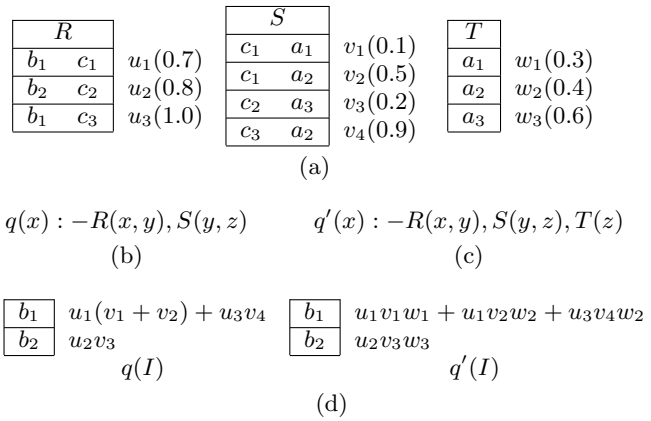


Figure 1: **(a) A tuple-independent probabilistic database $I$ with three relations $R, S, T$, the tuples are annotated with probabilities and tuple variables; (b) a *safe* query $q$; (c) an *unsafe* query $q'$; (d) answer tuples annotated with their boolean provenance.**

For all queries $q$ (safe or unsafe), given a probabilistic database $I$, the computation of the probabilities for the tuples in the answer $q(I)$ can be presented in two stages [18, 44]. In the *first stage* we compute for each tuple in $q(I)$ a *boolean expression* in the tuple variables from $I$. The boolean expression computed for a tuple $t_o \in q(I)$, which we call the *boolean provenance* of $t_o$, describes the event that $t_o$ appears in $q(I)$ as a combination of the events denoted by the input tuple variables. This first stage is not a source of computational hardness, provided we are concerned, as is the case throughout this paper, only with *data complexity* (i.e., we assume that the size of the query is a constant). Indeed, every boolean provenance in $q(I)$ is of poly size and can be computed in poly time, in the size of $I$. A further observation that will play a fundamental role in this paper is that for positive queries (UCQ, equivalently SPJU), the DNF of each boolean provenance is also of poly size and can be computed in poly time, again in the size of $I$[1].

[1]Through this paper we use DNF for the *irredundant* DNF


*Proceedings of the VLDB Endowment,* Vol. 4, No. 11

In the *second stage*, the probability of each boolean provenance is computed from the probabilities that the model associates with the tuples in $I$, i.e., with the tuple variables. For example, in Figure 1d the event that tuple $q'(b_2)$ appears in $q'(I)$ for a random instance described by $I$ is described by its boolean provenance $u_2v_3w_3$. By the independence assumptions it has probability $\Pr[u_2v_3w_3] = \Pr[u_2]\Pr[v_3]\Pr[w_3] = 0.4 \times 0.2 \times 0.6 = 0.048$. This break into two stages, called *intensional semantics* in [18, 10]) and described as the correct way of computing probabilities for unsafe queries gives a better understanding of the source of computational hardness. Indeed, the probability of a boolean expression is related to counting the number of its satisfying assignments which leads to #SAT, Valiant's first $\#\mathcal{P}$-complete problem [42].

But even for safe queries the separate study of the boolean provenance is beneficial. Olteanu and Huang have shown that if $q \in \mathrm{CQ}^-$ is safe then for any $I$ the boolean provenance of any tuple in $q(I)$ is read-once [31]. A boolean expression is in *read-once form* if each of its variables appears exactly once, and is *read-once* if it is equivalent to a boolean expression in read-once form. It is easy to see that the probability of an expression whose variables denote independent events can be computed in linear time if the expression is in read-once form (basic trick: $\Pr[\varphi \vee \psi] = 1-(1-\Pr[\varphi])(1-\Pr[\psi])$, because $\varphi$ and $\psi$ have disjoint sets of variables and hence are independent). With hindsight, the quest in [10] for plans for safe queries that on *all* inputs compute probability in poly time can be seen as a quest for plans that compute boolean provenance directly in read-once form. In our example, the boolean provenance of both answers to the safe query $q$ is shown already in read-once form in Figure 1d. It is in fact computed in this form by the safe plan $\Pi_x(R \bowtie \Pi_y S)$ found through the algorithm given in [10]. On the other hand, for the unsafe query $q'$ the boolean provenance of the tuple $q'(b_1)$ is not read-once (there is no equivalent read-once form). Using *knowledge compilation* techniques [15] such as BDDs and d-DNNFs the analysis of boolean provenance can also explain safety, i.e., poly time computation of probabilities, for classes of queries larger than safe $\mathrm{CQ}^-$s, as shown in [31, 25].

What of *unsafe positive queries*? As we saw from the example above, they can be quite common and cannot be ignored. Given the $\#\mathcal{P}$ lower bound, researchers have dealt with such queries in various ways (see Section 5). One observation relevant for this paper is that an unsafe query $q$ can produce on *some* probabilistic database $I$ boolean provenance for tuples in $q(I)$ whose probability can be computed efficiently and hopefully this can be decided in poly time based on $I$. For instance, modify our example in Figure 1a by deleting the tuple $S(c_3, a_2)$. To obtain the modified boolean provenance just set Figure 1d the corresponding tuple variable $v_4$ to 0. The query $q'$ is unsafe, but on this modified instance the boolean provenance of the answers is read-once because $u_1v_1w_1 + u_1v_2w_2 = u_1(v_1w_1 + v_2w_2)$ and thus the efficient way of computing probabilities applies. This leads to an "instance-by-instance" approach (taken in [39, 36]) in which both queries and instances are examined before choosing a probability computation algorithm.

Another observation relevant for this paper is that giving up on exact computation of the probabilities completely solves the problem for unsafe positive queries. This is because the boolean provenance of positive queries has IDNF of poly size and, as shown in [10] (see also [33]), an FPRAS (Fully Polynomial Randomized Approximation Scheme) is given by slightly adapting the one for DNF counting in [26].

of an expression, i.e., a disjunction of minterms that cannot be further shrunk by idempotence or absorption.

| | | $\Pr[q_1 - q_2]$ | |
|---|---|---|---|
| $\mathbf{q_1}$ | $\mathbf{q_2}$ | **Exact** | **Approximate** |
| UCQ | safe $\mathrm{CQ}^-$ | $\#\mathcal{P}$-complete | FPRAS |
| UCQ | UCQ | | Inapproximable |

Table 1: **Complexity of computation of probability for differences of positive queries.**

To summarize so far, there has been considerable work done on positive queries. But what of queries that include the relational *difference operation*? The concept of boolean provenance and the intensional semantics extend to difference. This was essentially shown in an early seminal paper by Imieliński and Lipski, and sets of tuples annotated by boolean provenance form particular cases of their *c*-tables [23]. However, there are some new and considerable difficulties with such queries. For one, the IDNF of the boolean provenance may be of exponential size and this precludes using either the read-onceness testing algorithm of [19] and the FPRAS for DNF counting in [26]. The latter is already observed by Koch [28] who also discusses techniques for avoiding the difference operation in some practical applications. Nonetheless, queries with difference are important. Practical SQL queries make use of difference not just explicitly (the EXCEPT construct) but also implicitly as when SELECT subqueries are nested within WHERE clauses with logically complex conditions.

In a recent and independent work [17], Fink et. al. proposed a framework to compute exact and approximate probabilities for answers of arbitrary relational algebra queries that allow difference operations. They extended the approach in [32] that computes a *d-tree* given a DNF expression by repeated use of independent AND, independent OR and Shannon's expansion, and showed that the boolean provenance of the answer is not required to be converted to DNF for queries with difference. Though this approach always returns an exact or approximate probability of the answer, the size of the d-tree is not guaranteed to be of poly-size in the input expression and therefore the running time may not be polynomial in the worse case. In fact, in this paper we show that it is not possible to get any non-trivial poly-time approximation of the answer probability for very simple queries with difference under standard complexity assumptions.

**Our Contributions.** We study the complexity of computing the probability of relational queries *with difference* on tuple-independent databases. Our *first result* is negative, and a somewhat surprising one. We exhibit two boolean queries $q_1$ and $q_2$, both of which are very nice by themselves, namely they are *safe* $\mathrm{CQ}^-$s, but such that computing the probability of $q_1 - q_2$ is $\#\mathcal{P}$-hard. The proof is complex, involving reductions from counting independent sets in 3-regular bipartite graphs to counting edge covers in another special class of bipartite graphs to counting satisfying assignments of certain configurations of boolean expressions[2].

[2]It has recently been brought to our attention that this result also follows from a hardness result in [9]. However, our proof uses completely different ideas and also shows $\#\mathcal{P}$-hardness of some natural graph problems which may be of

This hardness of exact computation result suggests that any class of interesting queries with difference which are *safe* in the spirit of [10, 11, 9] would have to be severely restricted[3].

In view of this lower bound for exact computation we turn to approximate computation of the probabilities. Our *second result* gives an FPRAS for computing probabilities for large classes of queries with difference. In particular, a corollary of our result applies to queries of the form $q_1 - q_2$ where $q_1$ is an arbitrary positive query (UCQ/SPJU) while $q_2$ is a safe query in $\text{CQ}^-$. The latter restriction is important because our *third result* shows the *inapproximability* of computing the probability of "$True - q$" where *True* is the boolean query that is always true while $q$ is the boolean CQ $q() :- S(x), R(x,y), S(y)$ which has a self-join. The three results give a simple summary of the situation for differences of positive queries, see Table 1.

In fact, our full FPRAS result is not restricted to a single differences of positive queries. In particular, it gives an FPRAS for any relational algebra query $q$ with multiple uses of difference as long as they are restricted as follows: (1) If $q_1 \bowtie q_2$ is a subquery of $q$ then at least one of $q_1, q_2$ must be positive, (2) If $q_1 - q_2$ is a subquery of $q$ then $q_1$ must be positive and $q_2$ must be a safe query in $\text{CQ}^-$ (in an instance-by-instance approach, the last requirement can be relaxed to $q_2(I)$ having read-once boolean provenance on a given $I$). To allow for instance-by-instance approaches we state our full FPRAS result in terms of requirements on the boolean provenance of the queries, using combinations of DNFs and d-DNNFs. The proof uses a new application of the Karp-Luby framework [26] that goes well beyond DNF counting.

**Roadmap.** In Section 2 we review boolean provenance, read-onceness and d-DNNFs. We also introduce the notion of difference rank and describe the connection between graph configurations (such as cliques or independent sets) and truth assignments that satisfy the boolean provenance of certain queries. In Section 3 we give our $\#\mathcal{P}$-hardness result for exact computation of probability of tuples produced by difference queries $q_1 - q_2$, where both $q_1, q_2$ are safe for UCQ. Section 4 gives our inapproximability results for general SPJUD queries with difference rank 1, as well as an FPRAS when the boolean provenance are in a special form. We also discuss classes of queries that produce this special form. In Section 5 we review some of the related work. Finally we conclude in Section 6 with directions for future work.

# 2. PRELIMINARIES

In this section we review some preliminary notions relevant to the rest of the paper.

## 2.1 Boolean provenance

The annotation of tuples with boolean expressions goes back to the *c*-tables of Imieliński and Lipski [23] who used them to describe *incomplete* databases. It was then used in [18, 44] and ever since for probabilistic databases. This is a particular case of *provenance* annotation [21] which is why in this paper we use the terminology "boolean provenance".

[3]There is always the obvious easy case when the events associated with $q_1$ and with $q_2$ are independent of each other because, for example, they operate on separate parts of the input database. We consider this case uninteresting.

The algorithm [23] for computing boolean provenance is quite well-known so we only illustrate it here on an example, namely for the query $q$ in Figure 1b and the instance $I$ in the same figure. We use the relational algebra expression $\Pi_x(R \bowtie \Pi_y S)$ which is equivalent to $q$ and we show the steps of the computation in Figure 2. First, to compute $\Pi_y S$, the boolean provenances of tuples in $S$ with the same value of $y$ are combined using disjunction ($+$) (see Figure 2a); disjunction is also used with union. For join, the boolean provenance of two tuples are combined using conjunction (see Figure 2b). Thus, the boolean provenance of the answers of positive queries is a *monotone* boolean expression. Moreover, an easy induction shows:

PROPOSITION 1. *For any positive (UCQ/SPJU) query $q$ and for any probabilistic database $I$ where $|I| = n$, the DNF of the boolean provenance of the tuples in $q(I)$ can be computed in time polynomial in $n$ (and so it also has poly size).*

Now consider $R = R_1 - R_2$ where the tuples in $R_1, R_2$ are annotated with boolean provenance. For a tuple $t$ to appear in $R$, $t$ must appear in $R_1$, let's say with boolean provenance $\phi$. If $t$ does not appear in $R_2$, then the boolean provenance of $t$ in $R$ will be also $\phi$; on the other hand if $t$ appears in $R_2$, let's say with boolean provenance $\psi$, then the boolean provenance of $t$ in $R$ will be $\phi \wedge \overline{\psi}$. Figure 2d shows the boolean provenance for the difference query $q' - q$ where $q, q'$ are given in Figure 1

When difference is added, Proposition 1 fails. Indeed, it is easy to see that if $U$ and $V$ each have $n$ tuples and no common join attributes, then the DNF of the boolean provenance of $True - \Pi_\emptyset(U \times V)$ has $2^{n^2}$ minterms. (here *True* is the boolean query that is always true, for example a 0-ary base relation containing one "dummy" tuple).

## 2.2 Difference Rank

As we have seen, for queries with difference the DNF of the boolean provenance can have exponential size. Thus, it is natural to study queries in which difference is used in restricted ways. The first step toward this is to limit the amount of "nesting" involving difference in a relational algebra expression. We do it through the following definition.

DEFINITION 1. *The* difference rank *of a relational algebra expression is a function $\delta : \mathcal{RA} \to \mathbb{N}$ defined inductively as:*

- (Base relation) $\delta(R) = 0$,
- (Project) $\delta(\Pi q) = \delta(q)$,
- (Select) $\delta(\sigma q) = \delta(q)$,
- (Join) $\delta(q_1 \bowtie q_2) = \delta(q_1) + \delta(q_2)$,
- (Union) $\delta(q_1 \cup q_2) = \max(\delta(q_1), \delta(q_2))$,
- (Difference) $\delta(q_1 - q_2) = \delta(q_1) + \delta(q_2) + 1$.

The queries of difference rank 0 are exactly the positive (SPJU) queries. The positive results in this paper will be for queries of difference rank 1, subject to further restrictions. There are natural queries of difference rank 2 however (see subsection 2.4). Proposition 2 below (the proof is given in Appendix A) is an extension of Proposition 1 and it justifies our focus on queries of difference rank 1. It shows that their boolean provenance has a certain structure that is potentially more manageable, and indeed, we will exploit this structure in designing approximation algorithms (Section 4).

| | |
|---|---|
| $c_1$ | $v_1 + v_2$ |
| $c_2$ | $v_3$ |
| $c_3$ | $v_4$ |

(a)

| | | |
|---|---|---|
| $b_1$ | $c_1$ | $u_1(v_1 + v_2)$ |
| $b_2$ | $c_2$ | $u_2v_3$ |
| $b_1$ | $c_3$ | $u_3v_4$ |

(b)

| | |
|---|---|
| $b_1$ | $u_1(v_1 + v_2) + u_3v_4$ |
| $b_2$ | $u_2v_3$ |

(c)

| | |
|---|---|
| $b_1$ | $(u_1v_1w_1 + u_1v_2w_2 + u_3v_4w_2) \cdot \overline{(u_1(v_1 + v_2) + u_3v_4)}$ |
| $b_2$ | $(u_2v_3w_3) \cdot \overline{(u_2v_3)}$ |

(d)

Figure 2: **Boolean provenance for (a) $\Pi_y(S)$; (b) $R \bowtie \Pi_y(S)$; (c) $\Pi_x(R \bowtie \Pi_y S)$; (d) $\Pi_x((R \bowtie S) \bowtie T) - \Pi_x(R \bowtie \Pi_y(S))$.**

PROPOSITION 2. *For any relational algebra query $q$ such that $\delta(q) \leq 1$, and for any probabilistic database $I$ where $|I| = n$, the boolean provenance of any tuple $t \in q(I)$ can be computed in poly-time in $n$ in the form*

$$\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$$

*where each of $\alpha_0, \cdots, \alpha_r$ and $\beta_1, \cdots, \beta_r$ is a monotone DNF in poly-size in $n$ while $r$ is also polynomial in $n$; moreover, if $\delta(q) = 0$, then $r = 0$ (we have a single DNF $\alpha_0$).*

## 2.3 Read-Once and d-DNNF

A boolean expression $\phi$ is said to be in *read-once form* if every variable $x \in \texttt{Var}(\phi)$ appears *exactly once*, where $\texttt{Var}(\phi)$ denotes the set of variables in $\phi$. An expression having an equivalent read-once form is called *read-once*. For example, $\overline{x_1}(x_2 + \overline{x_3}x_4) + x_5x_6$ is read-once but $\overline{x_1}(x_2 + \overline{x_3}x_4) + x_4x_6$ is not. The expression $xy + yz$ is read-once but $xy + yz + zx$ is not. A read-once expression $\phi$ can be naturally represented by a read-once-tree (see Figure 3 (a)).

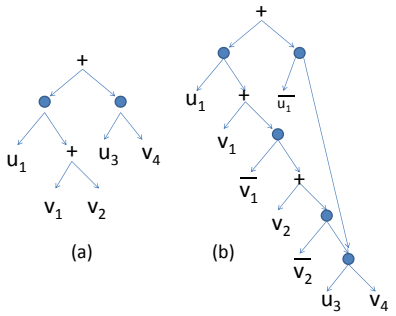


Figure 3: $\phi = u_1(v_1+v_2)+u_3v_4$**: (a) as a read-once tree, (b) as a d-DNNF.**

A *d-DNNF* (for *deterministic decomposable negation normal form*), introduced by Darwiche [13, 15], is a rooted DAG, where the leaves are labeled with positive or negative literals $x$ or $\overline{x}$, and internal nodes are $+$-nodes or $\cdot$-node (see Figure 3). The $\cdot$-nodes are *decomposable*, in the sense that, for any two children $u_i$ and $u_j$ of a $\cdot$-node $u$, $\texttt{Var}(\phi_{u_i}) \cap \texttt{Var}(\phi_{u_j}) = \emptyset$ (i.e., $\phi_{u_i}$ and $\phi_{u_j}$ are independent). On the other hand, $+$-nodes are *deterministic*, i.e., for any $+$-node $u$, and for any two children $u_i, u_j$ of $u$, the set of assignments of $\texttt{Var}(\phi)$ that satisfy $\phi_{u_i}$ and the set of assignments that satisfy $\phi_{u_j}$ are disjoint. Similar to read-once expressions, for a boolean expression $\phi$ represented as a d-DNNF, $\Pr[\phi]$ can be computed in linear time in the size of the d-DNNF repeatedly using $\Pr[\phi_u] = \Pi_{i=1}^{\ell} \Pr[\phi_{u_i}]$ (if $u$ is a $\cdot$-node) and $\Pr[\phi_u] = \sum_{i=1}^{\ell} \Pr[\phi_{u_i}]$ (if $u$ is a $+$-node), where $u_1, \cdots, u_\ell$ are children of $u$. The following proposition shows containment of these knowledge compilation forms which we will exploit while designing our approximation algorithms (the containment is strict [25]):

PROPOSITION 3. *If $\phi$ is in read-once form, then $\phi$ has a d-DNNF representation of size $O(n)$ ($n = |\texttt{Var}(\phi)|$) which can be computed in $O(n)$ time from the read-once tree of $\phi$.*

The proof of the above proposition follows from [13, 31, 25] using the construction of an OBDD [6] as an intermediate step. We illustrate the read-once-tree and d-DNNF representation of the read-once boolean provenance $u_1(v_1 + v_2) + u_3v_4$ of the tuple $q(b_1)$ from Figure 1d (see, for instance, [25] for the exact procedure).

Whether a boolean expression is read-once was known to be decidable in poly-time (along with the computation of the equivalent read-once form), when $\phi$ is in DNF [22, 19]; recently [39, 36] gave poly-time algorithm to achieve the same when a boolean expression $\phi$ produced by a $\text{CQ}^-$ query is given in *any* arbitrary form along with the query plan that produced $\phi$. An equivalent d-DNNF for a given boolean expression may not be compact, and also it is not known whether the boolean expressions having poly-size d-DNNF expressions are closed under negation [14, 15]. Nevertheless, since read-once boolean expressions are obviously closed under negation it follows from Proposition 3 that if $\phi$ is read-once, then $\overline{\phi}$ has a d-DNNF representation of size $O(|\texttt{Var}(\phi)|)$.

## 2.4 From Graphs to Queries

There is a tight connection between graph properties that assert the existence of vertex configurations such as cliques, vertex covers, independent sets, etc., and certain (boolean) relational algebra queries. Indeed, consider the relational schema $(V, E, S)$ where $V(A), S(A)$ are unary relations while $E(A_1, A_2)$ is binary. The idea is that $(V, E)$ encodes a simple undirected graph while the vertex subset $S \subseteq V$ and describes a *configuration* such as a vertex cover, a clique, etc. in the graph. Not all $(V, E)$-relational databases encode simple graphs so we restrict attention to those satisfying the constraints $E \subseteq V \times V$, $\forall v_1 v_2\ E(v_1, v_2) \rightarrow E(v_2, v_1)$ and $\forall v\ \neg E(v, v)$.

To express the relational algebra queries we begin with formulating the clique, cover, etc. properties as first-order sentences which we then transform into queries (making sure of domain independence).

**Examples**

1. $S$ induces a *clique*: $\forall x, y\ S(x) \wedge S(y) \wedge x \neq y \rightarrow E(x, y)$
   $q_{clique} = True - [\sigma_{A_1 \neq A_2}(\rho_{A_1/A}S \times \rho_{A_2/A}S) - E]$
2. $S$ is a *vertex cover*: $\forall x, y\ E(x, y) \rightarrow S(x) \vee S(y)$
   $q_{v-cover} = True - [E \bowtie \rho_{A_1/A}(V - S) \bowtie \rho_{A_2/A}(V - S)]$
3. $S$ is an *independent set*: $\forall x, y\ S(x) \wedge S(y) \rightarrow \neg E(x, y)$
   $q_{ind-set} = True - [E \bowtie \rho_{A_1/A}S \bowtie \rho_{A_2/A}S]$

Note that $\delta(q_{ind-set}) = 1$ while $\delta(q_{v-cover}) = \delta(q_{clique}) = 2$.

Consider probabilistic databases $I$ with schema $(V, E, S)$ such that the tuples in $V$ (the vertices) and $E$ (the edges) have probability 1 while $S$ has the same tuples as $V$ but

with arbitrary probabilities. The tuple variables that matter are those for $S$ and they correspond 1-1 to the vertices of the graph. A random instance of $S$ is subset of $V$ and these are in 1-1 correspondence with the truth assignments to the tuple variables. Hence there is a 1-1 correspondence between configurations satisfying the query (e.g., forming a clique, or an independent set) and the truth assignments that make the boolean provenance of the query true. This gives a reduction from counting the configurations to counting the corresponding satisfying assignments (this is further reduced to computing probability by choosing all probabilities in $S$ to be $\frac{1}{2}$).

# 3. HARDNESS OF EXACT COMPUTATION

In this section we give a hardness result for SPJUD queries. We will consider boolean SPJUD queries of the form $q = q_1 - q_2$, where both $q_1, q_2$ are CQ$^-$ queries; hence $\delta(q) = 1$. We show that the problem of exact probability evaluation for this class of simple-looking query is hard even in very restricted cases.

Given an SPJUD query $q = q_1 - q_2$ and a probabilistic database instance $I$, let $\phi_1$ and $\phi_2$ denote the two boolean provenance-s produced by $q_1$ and $q_2$ on $I$ respectively. Hence the boolean provenance of the unique tuple in $q(I)$ will be $\phi = \phi_1 \cdot \overline{\phi_2}$. As we discussed in Section 2.3, if $\phi$ (and therefore $\overline{\phi}$) is a read-once (RO) boolean expression on independent random variables, then $\Pr[\phi]$ and $\Pr[\overline{\phi}]$ can be computed efficiently. In this section we show that, the exact computation of $\Pr[\phi_1 \cdot \overline{\phi_2}]$ is $\#\mathcal{P}$-hard, even when both $\phi_1, \phi_2$ are RO. In fact, we show that the expressions $\phi_1, \phi_2$ can be generated by simple queries $q_1$ and $q_2$ of constant size that are individually *safe* for the class CQ$^-$ [10]. The following theorem states the main result proved in this section.

THEOREM 1. *There exists a fixed-size SPJUD query $q$ of the form $q = q_1 - q_2$ where both $q_1, q_2$ are safe CQ$^-$ queries, such that the exact computation of the probabilities of the answers is $\#\mathcal{P}$-hard.*

Let $\mathtt{C}(\phi)$ be the number of satisfying assignments of a boolean expression $\phi$. A tuple-variable $x$ in a boolean expression or in a probabilistic database instance will be called *uncertain* if the variable $x$ is present with some uncertainty, i.e., $\Pr[x] \notin \{0, 1\}$. In our reduction, we will construct a probabilistic database instance where for all uncertain tuple variables $x$, $\Pr[x] = \frac{1}{2}$. In this scenario, computation of $\Pr[\phi_1.\overline{\phi_2}]$ is equivalent to the computation of the number of satisfying assignments of $\phi_1.\overline{\phi_2}$ (since $\Pr[\phi_1.\overline{\phi_2}] = \frac{\mathtt{C}(\phi_1.\overline{\phi_2})}{2^N}$, where $N$ = the number of uncertain tuple variables); hence we focus on the counting version from now on.

To prove Theorem 1, intuitively, (i) first we need to construct a boolean expression $\phi$ which is expressible as the product of two RO expressions $\phi_1$ and $\overline{\phi_2}$, and where the computation of $\Pr[\phi]$ is $\#\mathcal{P}$-hard; (ii) then we need to construct two safe boolean queries $q_1, q_2$ and a database instance $I$ such that $q_1, q_2$ produce $\phi_1, \phi_2$ as the boolean provenance of the unique answer tuples in $q_1(I)$ and $q_2(I)$ respectively. Note that if $\phi_1$ and $\phi_2$ do not share any variables and are RO, then $\Pr[\phi_1 \cdot \overline{\phi_2}] = \Pr[\phi_1] \cdot (1 - \Pr[\phi_2])$ can be easily computed. Hence the challenge in the first step is to find a "hard" expression which can be factored into two "easy " expressions which share variables, whereas, the challenge in the second step is to construct a query that produces these expressions without using self join operation (since $\phi_1$ and $\phi_2$ share variables, we need to ensure that no two variables from the same relation join in either of them).

Before we describe the steps of the proof of Theorem 1, we first define two counting versions of satisfiability problems, one is the product of two RO CNF expressions, and, the other is the product of two RO DNF expressions.

DEFINITION 2. *We are given two CNF (resp. DNF) expressions $\psi_1$ and $\psi_2$, such that (i) $\mathtt{Var}(\psi_1) = \mathtt{Var}(\psi_2) = V$ (say), (ii) all literals are positive in both $\psi_1$ and $\psi_2$, (iii) both $\psi_1, \psi_2$ are RO, (iv) every clause (resp. term) in both $\psi_1$ and $\psi_2$ has* at most *four variables, and, (v) the variables in $V$ can be partitioned into four groups $V_1, \cdots, V_4$ such that no two variables from any group $V_i$ co-occur in any clause (resp. term) of $\psi_1$ and $\psi_2$. The goal is to compute $\mathtt{C}(\psi_1 \cdot \psi_2)$. We call this problem $\#$RO$\times$RO-4Partite-4CNF (resp. $\#$RO$\times$RO-4Partite-4DNF).*

An example of a RO$\times$RO-4Partite-4CNF expression is $\psi = \psi_1 \cdot \psi_2$, where $\psi_1 = (x_1+y_1+z_2)(x_2+y_2+z_1+w_1)$ and $\psi_1 = (x_2+y_1+z_2+w_1)(x_1+y_2+z_1)$ (both $\psi_1$ and $\psi_2$ read-once, and the variables $\{x_i\}$, $\{y_i\}$, $\{z_i\}$ and $\{w_i\}$ form a partition into four groups). Note that $\#$RO$\times$RO-4Partite-4CNF is a special case of the #Twice-SAT problem[4] which has been proved to be $\#\mathcal{P}$-hard in [7] without the requirements of bipartiteness of clauses in the expression $\psi_1 \cdot \psi_2$ in terms of sharing variables and 4-partiteness of variables in terms of belonging to the same clause.

Now we can give a proof sketch of Theorem 1, the details are deferred to Appendix B due to space constraint.

**Proof Sketch of Theorem 1:** The theorem is proved by a sequence of reductions that preserve $\#\mathcal{P}$-hardness. Our starting point is the problem of counting independent sets in a 3-regular graphs which is known to be $\#\mathcal{P}$-complete [43]. The steps are as follows:

Step1 Show that counting edge covers in bipartite graphs of degree at most four where the edge set can be partitioned into four matchings (called $\#$4Partite-4BEC) is $\#\mathcal{P}$-hard by a reduction from counting independent sets in 3-regular bipartite graphs.

Step2 Show that $\#$RO$\times$RO-4Partite-4CNF is $\#\mathcal{P}$-hard by a reduction from $\#$4Partite-4BEC.

Step3 Show that $\#$RO$\times$RO-4Partite-4DNF is $\#\mathcal{P}$-hard by a reduction from $\#$RO$\times$RO-4Partite-4CNF.

Step4 Show that $\Pr[q(I)]$ is $\#\mathcal{P}$-hard by a reduction from $\#$RO$\times$RO-4Partite-4DNF, where $q = q_1 - q_2$ satisfies the desired properties stated in Theorem 1.

In Step4, the queries we construct are $q_1() := R_1(x, y_1)\ R_2(x, y_2)\ R_3(x, y_3)\ R_4(x, y_4)$ and $q_2() := R_1(x_1, y)\ R_2(x_2, y)\ R_3(x_3, y)\ R_4(x_4, y)$. Clearly, $q_1, q_2$ do not use self-join and hence belong to CQ$^-$. Further, both of them are *hierarchical* (i.e. for every two variables $x, y$, the sets of subgoals that contain $x, y$ are either disjoint or one is contained in the other) and therefore are safe for the class CQ$^-$ [11].

[4]Twice-SAT is an instance of SAT where every variable appears in at most two clauses.

# 4. ESTIMATING TUPLE PROBABILITIES

Since the exact computation of the probabilities of the result tuples of a simple class of SPJUD queries has shown to be $\#\mathcal{P}$-hard in Section 3, now we focus on the question whether we can efficiently approximate the probabilities to a desired accuracy level. In other words, we attempt to get a *fully polynomial randomized approximation scheme* (or *FPRAS*). An FPRAS is a randomized algorithm that runs in time polynomial in the input size and $1/\epsilon$ ($\epsilon$ is the *accuracy parameter*) and produces a result that is correct to within relative error $(1 \pm \epsilon)$ with high probability. In particular, for the boolean provenance $\phi_t$ of a tuple $t$ in the answer $q(I)$ for an SPJUD query $q$ and database instance $I$, the approximation algorithm should run in time polynomial in $n = |I|$, and $1/\epsilon$, and output a value $\widehat{P_t}$ such that

$$\Pr[|\Pr[\phi_t] - \widehat{P_t}| \leq \epsilon \Pr[\phi_t]] \geq 3/4$$

The success probability can be boosted to $1 - \delta$ for any given *confidence parameter* $\delta$ by repeating the experiment $\log(1/\delta)$ times and then taking the median of the values.

The results in this section are summarized in the following theorems. First we state the inapproximability result in Theorem 2 (proof is given in Appendix C.3).

THEOREM 2. *There exists a fixed size SPJUD query of difference rank 1 such that the computation of probabilities of the answers does not have any FPRAS unless $\mathcal{P} = \mathcal{NP}$.*

Nevertheless, we show that an FPRAS can be achieved in some cases where the boolean provenance of the answers can be computed in a certain *probability friendly form* (PFF) as defined below.

DEFINITION 3. *A boolean expression $\phi$ on $n$ variables is said to be in PFF if $\phi$ is in form*

$$\phi = \alpha_0 + \sum_{i=1}^{r} \alpha_i \gamma_i$$

*where $\alpha_0, \alpha_1, \cdots, \alpha_r$ are DNFs, $\gamma_1, \cdots, \gamma_r$ are d-DNNFs[5], all of polynomial size in $n$, and $r$ is polynomial in $n$.*

Note that we are allowed to have negative literals in $\alpha_i$-s and $\gamma_j$-s in the above definition. The next theorem shows that if the boolean provenance of the answers of the SPJUD query can be expressible in PFF, then the probability computation of the answers has an FPRAS.

THEOREM 3. *There is an FPRAS for approximating the probabilities of the answers of any SPJUD query $q$ on probabilistic databases $I$ such that the boolean provenance of the tuples in $q(I)$ have PFF-s that can be computed in polynomial time in the size of $I$.*

We prove Theorem 3 in Section 4.1. In Section 4.2 we show that, indeed there is a natural class of queries of difference rank 1 that will produce boolean provenance expressions in PFF which can also be computed in poly-time.

[5]To be specific, each $\gamma_i$ is represented by d-DNNF DAGs.

## 4.1 FPRAS for $\Pr[\phi]$ when $\phi$ is in PFF

Consider a PFF $\phi$ of the form $\phi = \alpha_0 + \sum_{i=1}^{r} \alpha_i \gamma_i$, Where $r$ is polynomial in $n = |\mathtt{Var}(\phi)|$, all $\alpha_i$-s are represented in DNF, and $\gamma_i$-s are represented in d-DNNF, all having size polynomial in $n$. By expanding the minterms in the $\alpha_i$-s, $\phi$ can be expressed in the form

$$\phi = A_1 + A_2 + \cdots + A_s + B_1 \cdot \gamma_1 + \cdots + B_q \cdot \gamma_q \qquad (1)$$

where $A_j$-s and $B_j$-s are minterms (conjunction of positive and negative literals) from the DNFs $\alpha_i$-s and some of the $\gamma_i$-s may be the same. Moreover, $s, q$ will be polynomial in $n$. The equivalent DNF of $\phi$ can be of exponential size in $n$. However, we show that the *general Karp-Luby framework* to estimate the size of union of sets [26, 30] can still be used to obtain an FPRAS for $\Pr[\phi]$.

Given $\Pr[x]$ for all (independent) variables in $\mathtt{Var}(\phi)$, the general Karp-Luby framework works for estimating $\Pr[\phi]$ where $\phi = \phi_1 + \phi_2 + \cdots + \phi_m$, and $\phi_i$-s are boolean expressions *(not necessarily DNF minterms)* satisfying the following three properties:

(Q1) For each $\phi_i$, $\Pr[\phi_i]$ can be computed in poly-time,

(Q2) For each $\phi_i$, a random satisfying assignment $\sigma$ (of variables in $\mathtt{Var}(\phi)$) of $\phi_i$ can be generated in poly-time (i.e. $\sigma$ is sampled with probability $\Pr[\sigma|\phi] = \frac{\Pr[\sigma]}{\Pr[\phi]}$).

(Q3) For each assignment $\sigma$ and each $\phi_i$, it can be decided in poly-time whether $\sigma$ satisfies $\phi_i$.

The framework is presented in Appendix C.1 (Algorithm 2) for the sake of completeness. It is well-known that a set of samples of size $O(\frac{m}{\epsilon^2}\log(\frac{1}{\delta}))$ suffices to estimate $\Pr[\phi]$ within accuracy $(1 \pm \epsilon)$ with probability $\geq 1 - \delta$.

Hence it remains to show that all (Q1), (Q2), (Q3) hold when $\phi = \phi_1 + \cdots + \phi_m$ is in PFF. The property (Q3) trivially holds for all assignments $\sigma$ and for all $\phi_i$. Again, if $\phi_i$ in (1) is one of $A_1, \cdots, A_s$ (conjunction of literals), then properties (Q1) and (Q2) easily hold. Therefore we focus on proving (Q1) and (Q2), when $\phi_i$ is of the form $B_i\gamma_i$, where $B_i$ is conjunction of literals and $\gamma_i$ is in d-DNNF.

**Restricted d-DNNFs for $\phi_i = B_i \cdot \gamma_i$:** It is easy to see that if a boolean expression $f$ is represented in a poly-size d-DNNF, then any partial assignments of the variables in $f$ also has a poly-size d-DNNF, which can be computed in poly-time by replacing some of the variables by their unique assignments and reducing the d-DNNF with repeated use of $\textsc{True} + f' = \textsc{True}$, $\textsc{False} + f' = f'$, etc. Consider the boolean expression $B_i \cdot \gamma_i$. If $B_i \cdot \gamma_i$ is true, then $B_i$ must be true, which forces a unique assignment of the variables in $B_i$. The d-DNNF for the expression $\gamma_i$ with this partial assignment of the variables in $\mathtt{Var}(B_i) \cap \mathtt{Var}(\gamma_i)$ can be computed in poly-time. Let us call this d-DNNF as $\mathcal{D}_i$ (on the variable set $\mathtt{Var}(\gamma_i) \setminus \mathtt{Var}(B_i)$).

**(Q1): Computation of $\Pr[B_i \cdot \gamma_i]$.** The value of $\Pr[B_i \cdot \gamma_i]$ can be computed from its d-DNNF $\mathcal{D}_i$ in time linear in the size of $\mathcal{D}_i$ as discussed in Section 2.3, and multiplying this probability with the probability of the unique satisfying assignments of the variables in $\mathtt{Var}(B_i)$.

**(Q2): Uniform Sampling from d-DNNF.** Uniform sampling of a satisfying assignment of $\phi_i = B_i \cdot \gamma_i$, will be done by uniformly sampling a satisfying assignment $\sigma$ of $\mathtt{Var}(\gamma_i) \setminus \mathtt{Var}(B_i)$ using d-DNNF $\mathcal{D}_i$, extending $\sigma$ to include

the unique satisfying assignment of variables in $\texttt{Var}(B_i)$ and then further extending that assignment to an assignment $\sigma'$ of $\texttt{Var}(\phi)$ by a random assignment to the variables in $\texttt{Var}(\phi) \setminus \texttt{Var}(\phi_i)$ (for every variable $x \in \texttt{Var}(\phi) \setminus \texttt{Var}(\phi_i)$, assign $x = 1$ with probability $\Pr[x]$).

For a node $u$ in the d-DNNF $\mathcal{D}_i$, we will use $\phi_u$ to denote the sub-expression induced by the node $u$. Uniform sampling from a d-DNNF critically uses the *determinism* of the $+$-nodes: If a $+$-node $u$ has children $u_1, \cdots, u_k$, then for every pair of $j, \ell$, where $j \neq \ell$, the set of satisfying assignments for $\phi_{u_j}$ and that for $\phi_{u_k}$ are disjoint. The procedure for uniform sampling is given in Algorithm 1. It processes the nodes in the d-DNNF DAG $\mathcal{D}_i$ in reverse topological order (i.e., all the children of a node $u$ in $\mathcal{D}_i$ is processed before $u$ is processed). After a node $u$ is processed by the procedure, a satisfying assignment $\sigma_u$ of $\texttt{Var}(\phi_u)$ is returned. For a $\cdot$-node $u$, $\sigma_u$ is computed by concatenating the assignments $\sigma_{u_j}$, $j = 1$ to $k$, where $u_1, \cdots, u_k$ are the children of $u$. This works because $\texttt{Var}(\phi_{u_i}) \cap \texttt{Var}(\phi_{u_j}) = \emptyset$, for every two distinct children $u_i, u_j$. On the other hand, for every $+$-node $u$, one of the children $u_j$ is chosen at random, and $\sigma_u$ is assigned to be $\sigma_{u_j}$. It is easy to check that the sampling procedure runs in time linear in the size of $\mathcal{D}_i$, the formal analysis of its correctness is given in Appendix C.2.

**Algorithm 1** *Uniform satisfying assignment generation from a d-DNNF.*

**Input: A d-DNNF $\mathcal{D}$ with root $\texttt{r}$.**
**Output: A satisfying assignment $\sigma$ of $\texttt{Var}(\phi_\texttt{r})$ output with probability $\Pr[\sigma]/\Pr[\phi_\texttt{r}]$ .**

– Compute $\Pr[\phi_u]$ for every node $u$ in $\mathcal{D}$ (see Section 2.3).
– Compute a reverse topological order $\pi$ of the nodes in the d-DNNF DAG $\mathcal{D}$. Process the nodes in the order $\pi$.
**for** each node $u$ in $\mathcal{D}$ **do**
  **if** $u$ is a sink-node marked with variable $x$ **then**
    **if** $\phi_u = x$, set $\sigma_u$ to be $x = 1$; **else** set $\sigma_u$ to be $x = 0$.
  **else** {/* *$u$ is an internal node* */}
    – Let $u_1, \cdots, u_k$ be the children of $u$.
    **if** $u$ is a $\cdot$-node **then**
      – Set $\sigma_u$ to be concatenation of $\sigma_{u_j}$, $j = 1$ to $k$.
    **else** {/* *$u$ is a $+$-node* */}
      – Choose child $u_j$ with probability $\frac{\Pr[u_j]}{\sum_{\ell=1}^{k} \Pr[u_j]}$.
      – Extend $\sigma_{u_j}$ to $\sigma'_{u_j}$ by randomly assigning the variables in $\texttt{Var}(\phi_u) \setminus \texttt{Var}(\phi_{u_j})$.
      – Set $\sigma_u$ to be $\sigma'_{u_j}$.
    **end if**
  **end if**
**end for**
**return** – $\sigma = \sigma_\texttt{r}$.

## 4.2 Classes of Queries Producing PFF

For an SPJUD query $q$ with difference rank $\delta(q) = 1$, we call a *difference sub-query* to be the sub-query that *immediately* appears on the right hand side of a difference operation. Since we allow union, there may be more than one difference sub-query of a query $q$ with $\delta(q) = 1$. It is well-known that the boolean provenance $\phi_t$ of answers $t$ of a safe $\text{CQ}^-$ query on tuple-independent databases are read-once [31], furthermore, if $\phi_t$ is read-once, $\overline{\phi_t}$ is also read-once, and there is a poly-size d-DNNF for $\overline{\phi_t}$ which can be computed in poly-time (see Proposition 3). Consider Proposition 2. From the proof of this proposition it follows that only the boolean provenance $\beta_i$ produced by difference sub-queries appear as $\overline{\beta_i}$ of the boolean provenance $\phi$ of result tuples of an SPJUD query $q$ with $\delta(q) = 1$. Since these $\beta_i$-s are read-once when the corresponding difference sub-query is safe for $\text{CQ}^-$, we have the following corollary:

Corollary 1. *Given an SPJUD query $q$ with $\delta(q) = 1$, if all the difference sub-query-s in $q$ are safe for the class $CQ^-$, then for any probabilistic database instance $I$, the boolean provenance of all answers in $q(I)$ will have a PFF that can be computed in poly-time. Therefore there is an FPRAS for this class of queries on any instance $I$.*

Corollary 1 can be extended to the instance-by-instance approach taken in [39, 36]: for the query-instance pairs $(q, I)$ such that for every difference sub-query $q'$ of $q$ the boolean provenance-s in $q'(I)$ are read-once, the probability of the answers in $q(I)$ can be approximated. On the other hand, a similar result can be obtained when the difference sub-queries are UCQ queries such that for all instances $I$, the boolean provenance-s have a poly-size OBDD (*ordered binary decision diagrams*) representation [25]; this can also be decided from the query. Since OBDD-s are closed under negation, and a d-DNNF of poly-size in OBDD can be constructed in poly-time [25, 13], the boolean provenance of the answer tuples for all such queries on all database instances will have PFF-s computable in poly-time, and therefore again an FPRAS can be obtained using our result in the previous section.

# 5. RELATED WORK

There has been significant progress since 2004 in probabilistic databases. We have already mentioned the *dichotomy* results that have identified exactly the safe positive queries [10, 11, 11, 9]. Ré and Suciu show a *trichotomy* result [35] for queries in $\text{CQ}^-$ extended with *aggregate* operations. Such queries can be divided into safe [34] which have efficient exact computation, *approx-safe* which have an FPRAS, and *hazardous*, which are inapproximable. This is a stronger kind of result than our lower bounds because we do not show that *every* query for which we don't give an FPRAS is inapproximable, just that some such queries exist. Approximation techniques for queries on probabilistic databases are also studied in [27, 32]. We have also mentioned work explaining the tractability of safe queries through knowledge compilation applied to their boolean provenance [31, 25] and a recent work proposing a framework for exact and approximate probability evaluation for queries with difference, but without a guarantee of polynomial running time [17].

Exploiting boolean provenance through knowledge compilation has been used for unsafe queries, in an instance-by-instance approach for the class of $\text{CQ}^-$s [39, 36]. An earlier, completely different approach compiles queries *and* databases into *probabilistic graphical models* and then performs *inference* on these [37, 38], taking advantage of the considerable theoretical and practical arsenal developed in Machine Learning. Knowledge compilation is combined with Bayesian network inference in [24]. Significant work has also been done on top-$k$ queries on probabilistic databases [33, 29, 40]. Finally, several systems for query processing on probabilistic databases have been developed, including MistiQ [5], Trio [2], and MayBMS [1].

# 6. CONCLUSIONS AND FUTURE WORK

We have examined the theoretical difficulty of computing exactly, and of approximating, the answers to relational queries with difference on probabilistic databases. The obvious next step is to assess the practical validity of the algorithms that we have presented as part of our FPRAS.

Desirable extensions of this work include using more flexible probabilistic models, e.g. disjoint-independent databases, and continuing to improve the sharpness of the divisions we have discovered, including dichotomy or trichotomy results as discussed before.

**Acknowledgements.** This work was supported in part by the US National Science Foundation grants IIS-0803524, IIS-0904314 and CCF-0635084. The authors thank Dan Suciu for many useful discussions.

# 7. REFERENCES

[1] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992, 2008.

[2] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[3] B. Bollobás. *Modern Graph Theory*. Springer, corrected edition, July 1998.

[4] M. Bordewich, M. Dyer, and M. Karpinski. Path coupling using stopping times and counting independent sets and colorings in hypergraphs. *Random Struct. Algorithms*, 32:375–399, May 2008.

[5] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.

[6] R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[7] R. Bubley and M. Dyer. Graph orientations with no sink and an approximation for a hard case of #sat. In *SODA*, pages 248–257, 1997.

[8] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[9] N. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214, 2010.

[10] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[11] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[12] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[13] A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001.

[14] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *AAAI*, pages 627–634, 2002.

[15] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.

[16] M. E. Dyer, L. A. Goldberg, C. S. Greenhill, and M. Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2003.

[17] R. Fink, D. Olteanu, and S. Rath. Providing support for full relational algebra in probabilistic databases. In *ICDE*, pages 315–326, 2011.

[18] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[19] M. C. Golumbic, A. Mintz, and U. Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial $k$-trees. *Discrete Appl. Math.*, 154(10):1465–1477, 2006.

[20] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *PODS*, pages 227–234, 1998.

[21] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[22] V. A. Gurvich. Criteria for repetition-freeness of functions in the algebra of logic. *Soviet Math. Dokl.*, 43(3):721–726, 1991.

[23] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, September 1984.

[24] A. Jha, D. Olteanu, and D. Suciu. Bridging the gap between intensional and extensional query evaluation in probabilistic databases. In *EDBT*, pages 323–334, 2010.

[25] A. Jha and D. Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *ICDT*, pages 162–173, 2011.

[26] R. M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64, 1983.

[27] C. Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108, 2008.

[28] C. Koch. On query algebras for probabilistic databases. *SIGMOD Rec.*, 37:78–85, March 2009.

[29] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[30] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28:33–37, March 1996.

[31] D. Olteanu and J. Huang. Using obdds for efficient query evaluation on probabilistic databases. In *SUM*, pages 326–340, 2008.

[32] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

[33] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[34] C. Ré and D. Suciu. Efficient evaluation of having queries on a probabilistic database. In *DBPL*, pages 186–200, 2007.

[35] C. Ré and D. Suciu. The trichotomy of having queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009.

[36] S. Roy, V. Perduca, and V. Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.

[37] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[38] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. *PVLDB*, 1(1):809–820, 2008.

[39] P. Sen, A. Deshpande, and L. Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.

[40] M. A. Soliman, I. F. Ilyas, and S. Ben-David. Supporting ranking queries on uncertain and incomplete data. *VLDB J.*, 19(4):477–501, 2010.

[41] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

[42] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8(2):189 – 201, 1979.

[43] M. Xia and W. Zhao. #3-regular bipartite planar vertex cover is #p-complete. In *TAMC*, pages 356–364, 2006.

[44] E. Zimányi. Query evaluation in probabilistic relational databases. *Theor. Comput. Sci.*, 171(1-2):179–219, 1997.

# APPENDIX

# A. PROOFS FROM SECTION 2

## A.1 Proof Sketch of Proposition 1 and Proposition 2

For simplicity, we combine the proofs of these two propositions together since they use similar induction argument.

PROOF. Let us denote by $\mathcal{N}(q, I)$, the number of tuples in the answer output by query $q$ on instance $I$. We prove by induction on the size of the $\mathcal{RA}$ expression $q$ that (1) $\mathcal{N}(q, I)$ is bounded by $n^{|q|}$, (b) the provenance expression $\phi_t$ of all result tuples $t$ can be computed in total time $O(n^{c|q|})$, for some constant $c \geq 3$, such that (a) each expression $\phi_t$ has the form $\phi_t = \alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$, (b) for every such $\phi_t$, for all $\alpha_i$ and $\beta_j$, $|\alpha_i|$ and $\beta_j$ are bounded by $n^{|q|+1}$, (c) $\sum_{t \in q[I]} r_t \leq n^{|q|}$, and, (d) for each $t$, $r_t = 0$ if $\delta(q) = 0$ (it may be the case that $r_t = 0$ even when $\delta(q) = 1$). Since $|q|$ is considered a constant, this will imply the statements of both Proposition 1 and Proposition 2.

The base case follows for queries $q$ such that $|q| = 1$, which must be a single base relation $R$; hence $\delta(q) = 0$. Here $\mathcal{N}(q, I) \leq n$, the provenance expressions are the tuple variable themselves, so they can be computed in $O(n) = O(n^c)$ time. The tuple variables are trivially IDNF expression $\alpha_0$ with $r_t = 0$ (hence $\sum_{t \in q[I]} r_t = 0 \leq n^{|q|}$) and $|\alpha_0| = 1 \leq n^{|q|}$.

Suppose the induction hypothesis holds for all $\mathcal{RA}$ expressions $q$ such that $|q| < k$, where $\delta(q) \leq 1$. Now consider a query $q$ such that $|q| = k$, $k > 1$. The hypothesis is proved for all possible operations in $q$, however due to space constraint, we describe the proof outline for project, join and difference operations; select and union can be similarly handled.

1. (Project) If $q$ is of the form $\Pi q_1$, then $|q_1| = |q| - 1 < k$. Hence $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I) \leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$. The provenance expression for every tuple $t$ will equal to $\phi_{t_1} + \cdots + \phi_{t_\ell}$, for some tuples $t_1, \cdots, t_\ell$ in $q_1[I]$. Since $\ell \leq n^{|q_1|}$, this can be computed in time $O(n^{3|q_1|}) + O(n^{c|q_1|}) = O(n^{c|q_1|}) = O(n^{c|q|})$ (the total time complexity to compute the project operation even by brute-force method is $O(\mathcal{N}(q_1, I) \times \mathcal{N}(q_1, I) \times \max_{t_1 \in q_1[I]} |\phi_{t_1}| = O(n^{3|q_1|})$, and the time complexity to compute $q_1[I]$ is $O(n^{c|q_1|})$) . Also, $\sum_{t \in q[I]} r_t$ is bounded by $\sum_{t \in q_1[I]} r_{t_1} \leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$. This is because every tuple in $q_1[I]$ contributes to exactly one tuple in $q[I]$.
   $\phi_t = \phi_{t_1} + \cdots + \phi_{t_\ell}$ will have the form $\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$ by IH. Size of every DNF $\alpha_i$-s and $\beta_j$-s remains the same, hence the sizes are bounded by $n^{|q_1|} \leq n^{|q|}$. Further, if $\delta(q) = 0$, $\delta(q_1) = 0$ as well. So the value of $r = r(t_j)$ is 0 for all $j \in [1, \ell]$. Hence $r_t$ is also 0.
2. (Join) Let $q = q_1 \bowtie q_2$, i.e. $|q| = |q_1| + |q_2| + 1$. In the worst case, every tuple in $q_1$ can join with every tuple in $q_2$. Therefore, $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I) \times \mathcal{N}(q_2, I)$, $\leq n^{|q_1|} \times n^{|q_2|}$ (by IH), $\leq n^{|q|}$. If $\delta(q) = 0$, both $\delta(q_1), \delta(q_2)$ are 0, therefore we have the required form of $\phi_t$ for a tuple $t$, where $r_t = 0$. Hence $\sum_{t \in q[I]} r_t = 0$ as well.
   If $\delta(q) = 1$, exactly one of $\delta(q_1), \delta(q_2)$ is 1; wlog. let $\delta(q_1) = 1, \delta(q_2) = 0$. Then also for every tuple $\phi_t$ in $q[I]$, $\phi_t = \phi_{t_1} \times \phi_{t_2}$. Let $\phi_{t_1} = \alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}$, $\phi_{t_2} = \alpha'_0$. Then $\phi_t = \left(\alpha_0 + \sum_{i=1}^{r} \alpha_i \overline{\beta_i}\right) \times \alpha'_0 = = \alpha''_0 + \sum_{i=1}^{r} \alpha''_i \overline{\beta_i}$, where $\alpha''_i$, $i \in [0, r]$, are obtained by computing the IDNF of $\alpha_i \times \alpha'_0$. Hence $\sum_{t \in q[I]} r_t = (\sum_{t \in q[I]} r_t) \times \mathcal{N}(q_2, I)$ (since every tuple in $q_1[I]$ can join with at most $\mathcal{N}(q_2, I)$ tuples in $q_2[I]$, and every tuple $t_2$ in $q_2[I]$ have $r_{t_2} = 0$) $\leq n^{|q_1|} \times n^{|q_2|}$ (by IH) $\leq n^{|q_1|+|q_2|} \leq n^{|q|}$. Size of $|\beta_j|$ remains the same, and $|\alpha''_i| \leq n^{|q_1|} \times n^{|q_2|} \leq n^{|q|}$. The time complexity of the brute-force method can be shown to be $O(n^{c|q|})$.
3. (Difference) Let $q = q_1 - q_2$, i.e. $|q| = |q_1| + |q_2| + 1$. Therefore, $\mathcal{N}(q, I) \leq \mathcal{N}(q_1, I)$, $\leq n^{|q_1|}$ (by IH), $\leq n^{|q|}$. For difference, $\delta(q)$ can not be 0, i.e., $\delta(q) = 1$, and it must be the case that $\delta(q_1) = \delta(q_2) = 0$. For a tuple $t$ in $q[I]$, either (i) $\phi_t = \phi_{t_1}$, for some $t_1$ in $q_1[I]$ (when there is no tuple in $q_2[I]$ that has the same value as in $t$), or, (ii) $\phi_t = \phi_{t_1} \times \phi_{t_2}$, for some $t_1$ in $q_1[I]$ and $t_2$ in $q_2[I]$. In case (i), $\phi_t$ obviously has the required form, $r_t = 0$, and $|\alpha_i|, |\beta_j|$ have the required size bound. In case (ii), since $\delta(q_1) = \delta(q_2) = 0$, we have by IH, $\phi_{t_1} = \alpha_0$ and $\phi_{t_2} = \alpha'_0$, where both $\alpha_0, \alpha'_0$ are positive IDNF. Hence $\phi_t = \phi_{t_1} \times \overline{\phi_{t_2}} = \alpha_0 \overline{\alpha'_0}$. Hence again $\phi_t$ has the required form, and again $|\alpha_0|$ and $|\alpha'_0|$ individually have the required size bound. For both $t_1, t_2$, $r_{t_1} = r_{t_2} = 0$, so $r_t = 1$. Hence $\sum_{t \in q[I]} r_t \leq \mathcal{N}(q, I) \leq n^{|q|}$. The difference can be computed in time $O(n^{|q_1|} \times n^{|q_2|}) + O(n^{c|q_1|}) + O(n^{c|q_2|})$ (by IH), $= O(n^{c|q|})$.

This completes the proof of the proposition. □

# B. PROOFS FROM SECTION 3

In this section we prove Theorem 1, by proving the sequence of four steps stated in Section 3. The following lemma shows a nice property of product of two RO expressions and will be useful in proving Step3 and Step4.

LEMMA 1. *Let $\phi_1, \phi_2$ be two read-once boolean expression on the same set of variables. Then either each of $\mathtt{C}(\phi_1 \cdot \overline{\phi_2})$, $\mathtt{C}(\phi_1 \cdot \phi_2)$, $\mathtt{C}(\overline{\phi_1} \cdot \phi_2)$ and $\mathtt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$ is poly-time computable or none of them is.*

PROOF. Let $n = |\mathtt{Var}(\phi_1)| = |\mathtt{Var}(\phi_2)|$. If $\phi$ is an RO expression, then $\mathtt{C}(\phi)$ can be exactly computed in time linear in the size of $\phi$. Therefore, $\mathtt{C}(\overline{\phi_1}) = 2^n - \mathtt{C}(\phi)$ can also be efficiently computed. Now we have, (i) $\mathtt{C}(\phi_1) = \mathtt{C}(\phi_1 \cdot \phi_2) + \mathtt{C}(\phi_1 \cdot \overline{\phi_2})$, (ii) $\mathtt{C}(\overline{\phi_1}) = \mathtt{C}(\overline{\phi_1} \cdot \phi_2) + \mathtt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$, (iii) $\mathtt{C}(\phi_2) = \mathtt{C}(\phi_1 \cdot \phi_2) + \mathtt{C}(\overline{\phi_1} \cdot \phi_2)$, and, (iv) $\mathtt{C}(\overline{\phi_2}) = \mathtt{C}(\phi_1 \cdot \overline{\phi_2}) + \mathtt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$. Using the above equations, if any one of $\mathtt{C}(\phi_1 \cdot \phi_2)$, $\mathtt{C}(\phi_1 \cdot \overline{\phi_2})$, $\mathtt{C}(\overline{\phi_1} \cdot \phi_2)$, or $\mathtt{C}(\overline{\phi_1} \cdot \overline{\phi_2})$ can be exactly computed in poly-time, all the others can be exactly computed in poly-time. □

## B.1 Step1: #$\mathcal{P}$-hardness of #4Partite-4BEC

For a graph $G(V, E)$, a subset $U \subseteq E$ is an *edge cover* if every vertex of $G$ has at least one incident edge in $U$. Theorem 4 shows the #$\mathcal{P}$-hardness of the #4Partite-4BEC problem. The proof uses similar idea as given in [4] which reduces #independent set problem in a constant degree graph to the #edge-cover problem in a graph of minimum degree $\Delta$, and shows that by constructing a number of #edge-cover instances, and using the *interpolation technique* [41], independent set in a constant degree graph can be exactly computed. However, the edge-cover instances constructed in [4]

are non-bipartite and the vertices in those instances do not have constant degrees; hence we need to modify the construction.

We reduce the problem of #independent set in 3-regular bipartite graphs (#3-reg-BIS) to #4Partite-4BEC. #vertex-cover in 3-regular bipartite graphs is known to be #$\mathcal{P}$-hard [43]. In a graph $G(V, E)$, $S \subseteq V$ is a vertex cover if and only if $V \setminus S$ is an independent set. Hence the set of vertex cover has a 1-1 correspondence with the set of independent sets in any graph which shows that #3-reg-BIS is #$\mathcal{P}$-hard.

THEOREM 4. *#4Partite-4BEC is #$\mathcal{P}$-hard.*

PROOF. Let $G(V_1, V_2, E)$ be a bipartite 3-regular graph, where $|V_1| = |V_2| = n'$. We will use $V = V_1 \cup V_2$ to denote the vertex set of $G$, and and $n = 2n'$ to denote the total number of vertices in $G$. Let $I_j(G)$ be the number of independent sets of size $j$, $j \in [0, n]$. Form $G'$ by inserting a vertex $u_e$ on every edge $e \in E$. Let $U = \{x_e : e \in E\}$ ($|U| = 3n/2$). Let $N_i(G')$ be the number of edge subsets in $G'$ which leave exactly $i$ vertices in $V$ uncovered, but no vertex in $U$ uncovered($N_0(G')$ is the number of edge covers in $G'$). The set of vertices in $G'$ which are not covered by such a subset of edges forms an independent set in $G$: if a vertex $u \in V$ is uncovered in $G'$, for every incident edge $e$ on $u$, $e = (u, v) \in E$, $x_e$ and hence $v$ must be covered.

Let $W \subset V$ be an independent set of size $j$ in $G$. Let us count the number of edge subsets in $G'$ that (i) do not cover $W$, (ii) cover all members of $U$, and (iii) may or may not cover the vertices in $V \setminus W$. The $3j$ edges incident on $W$ must not be chosen in such a subset, hence the $3j$ edges next to these edges must be chosen to cover all members in $U$. Hence we are left with $3n - 6j$ edges which can be paired up into $\frac{3n-6j}{2}$ groups according to the edges in $E$ they came from. At least one edge in each such pair must be chosen to cover vertices in $U$, so we have 3 choices for each pair. Hence for a fixed independent set $W$ of size $j$, the number of choices is $3^{(3n-6j)/2}$. If we multiply this quantity with the number of independent sets of size $j$ in $G$, i.e. $I_j(G)$, we double count some of the edge subsets which cover more than one independent sets of size $j$. An edge subset which covers exactly $i$ vertices from $V$ and all vertices from $U$ is counted $\binom{i}{j}$ times, once for every choice of $j$ out of those $i$ vertices the subset covers. Hence it follows that

$$3^{(3n-6j)/2} I_j(G) = \sum_{i=j}^{n} \binom{i}{j} N_i(G') \tag{2}$$

Hence if we can compute all $N_i(G')$, $i \in [0, n]$, we can compute the number of all independent sets $\sum_{j=0}^{n} I_j(G)$ in $G$.

So focus on computing all $N_i(G')$ in $G'$. Let $C_k$ denote a chain with $k$ nodes. We attach a copy $C_s^v$ of $C_s$ to each vertex $v \in V$ by identifying $v$ with an endpoint of $C_s$. Let us call the resulting graph $G'_s$. Since we started with a regular bipartite graph $G$ of degree 3, $G'_s$ is a bipartite graph with maximum degree 4 (vertices in $V$ have degree 4, vertices in $U$ have degree 2, and the vertices in the chains $C_s^v$, $v \in V$ have degree at most 2).

Let $M_k$ denote the number of edge covers in $C_k$. Now we count the number of edge covers in $G'_s$. Each edge cover of $G'_s$ induces an edge subset of $G'$. To extend an edge subset of $G'$ which leaves exactly $i$ vertices in $V$ uncovered, and no vertex in $U$ uncovered, to an edge cover of $G'_s$, we must select an edge cover for $i$ copies of $C_s$, but either an edge cover or a cover missing the first vertex (identified with the corresponding $x_e$) for the remaining $n - i$ copies of $C_s$. Thus the total number of edge covers $N_0(G'_s)$ is given by:

$$\begin{aligned} N_0(G'_s) &= \sum_{i=0}^{n} M_s^i (M_s + M_{s-1})^{n-i} N_i(G') \\ &= M_s^n \sum_{i=0}^{n} (1 + \frac{M_{s-1}}{M_s})^{n-i} N_i(G') \end{aligned} \tag{3}$$

In Lemma 2 we show that the edge set can indeed be partitioned into four disjoint matchings, which shows that each $G'_s$, for any value of $s$, is an instance of the #4Partite-4BEC problem.

Suppose we are given an oracle for the #4Partite-4BEC problem. That oracle can be used to compute $N_0(G'_s)$ for any $s$ value. From Lemma 4 the values of $\frac{M_{s-1}}{M_s}$ are distinct for all distinct $s$, and therefore from Fact 1, all the coefficients $N_i(G')$ can be computed by calling the edge cover oracle on $G'_s$ graphs for $n + 1$ distinct values of $s$ (in (3). From these $N_i(G')$ values the number of independent sets in a 3-regular graph can be exactly computed using (2). This completes the reduction. □

LEMMA 2. *The set of edges in each $G'_s$ in the proof of Theorem 4 can be partitioned into set of four disjoint matchings.*

PROOF. Recall that we started with a 3-regular bipartite graph $G(V, E)$, inserted the vertex set $U = \{x_e : e \in E\}$ into the edges in $E$ to form the graph $G'$, and attached chains with $s$ vertices, $C_s$ with every vertex in $V$ to form the graph $G'_s$.

It is well-known that a 3-regular graph can be partitioned into 3 disjoint perfect matchings (see, for instance [3]). Let a set of 3 disjoint matchings for $G$ be $E_1, E_2, E_3$, where $E = E_1 \cup E_2 \cup E_3$. Let $V_1, V_2$ be the bipartition of vertices in $G$ ($V = V_1 \cup V_2$), and let $E_i^1, E_i^2$, $i \in [1, 3]$, denote the set of edges in $G'$ which are incident on $V_1$ and $V_2$ respectively. Note that $\bigcup_{i=1}^{3} (E_i^1 \cup E_i^2)$ is exactly the edge set in $G'$.

Further, any chain $C_s$ can be partitioned into two disjoint matchings (comprising the alternating edges). For a copy of the chain $C_s^v$, $v \in V$, let $M_1(C_s^v)$ the matching which includes the edge incident on $v$, and $M_2(C_s^v)$ be the other matching.

Now we construct four disjoint group of matchings $P_1$, $P_2$, $P_3$, $P_4$ as follows (there may be several other ways): (i) $P_1 = E_1^1 \cup E_2^2 \cup \bigcup_{s \in V} \{M_2(C_s^v)\}$, (ii) $P_2 = E_2^1 \cup E_3^2$, (iii) $P_3 = E_3^1 \cup E_1^2$, (iv) $P_4 = \bigcup_{s \in V} \{M_1(C_s^v)\}$.

Clearly, $P_1 \cup P_2 \cup P_3 \cup P_4$ is exactly the edge set of $G'_s$. The fact that $P_4$ is a matching follows easily, since all the copies of the chains $C_s^v$ are disjoint. Each $E_j^i$, $i \in [1, 2], j \in [1, 3]$ forms a matching themselves, since they are taken from the matchings $E_1, E_2, E_3$ in $G$. In $P_1, P_2$ or $P_3$, the edges taken from $G'$ are of the form $E_j^i \cup E_{j'}^{i'}$, where $i \neq i'$ and $j \neq j'$. Since $i \neq i'$, if the edges $E_j^i$ are incident on $V_1$, $E_{j'}^{i'}$ are incident on $V_2$ (or vice versa). Since $j \neq j'$, no two edges incident on some $x_e \in U$ are ever included together. Moreover, $P_1$ includes the edges from the chains $\bigcup_{s \in V} \{M_2(C_s^v)\}$ which themselves are matchings, and are not incident on any vertex in $V_1 \cup V_2$ Therefore, the $P_i$-s, $i \in [1, 4]$ partition the edge set in $G'_s$ in four disjoint matchings. □

The following lemma counts the number of edge covers in a chain (similar to vertex cover counting in a chain [41]).

LEMMA 3. *Let $M_k$ denote the number of edge covers in a chain graph with $k$ nodes $C_k$. Then $M_k$ can be computed by the recurrence $M_k = M_{k-1} + M_{k-2}$, for $k \geq 3$, and $M_2 = M_3 = 1$ (there is no edge cover for a singleton node).*

PROOF. In $C_2$, the unique edge must be chosen, so $M_2 = 1$. Again in $C_3$, both the edges must be chosen to cover two endpoints, hence $M_3 = 1$.

Let $v_0, \cdots, v_k$ be the vertices in the chain $C_k$, $k \geq 3$. The first edge $(v_0, v_1)$ in the chain must be included in any edge cover which also covers $v_1$. Hence $v_1$ may or may not be covered by the edge $(v_1, v_2)$. The number of edge subsets in the chain $v_1, \cdot, v_k$ which covers $v_1$ (and the other vertices $v_2, \cdots, v_k$), i.e. includes $(v_1, v_2)$, is $M_{k-1}$, whereas the number of edge subsets which do not cover $v_1$ (does not include $(v_1, v_2)$), but covers $v_2, \cdots, v_k$ is $M_{k-2}$. Hence $M_k = M_{k-1} + M_{k-2}$. □

COROLLARY 2. *The value of $M_k$ is the same as the $(k-1)$-th number in the Fibonacci series.*

The following Fact 1 and Lemma 4 used in the proof of Theorem 4 have been proved in [41] (Fact 4.1 and Lemma A.1 respectively).

FACT 1. *[41] Let $f(x) = \sum_{i=0}^{d} a_i x^i$ be a polynomial with rational coefficients. If $(\alpha_0, \beta_0), \cdots, (\alpha_d, \beta_d)$ are such that $f(\alpha_j) = \beta_j$, $j = 1$ to $d$, and all $\alpha_j$-s are* distinct*, then all the coefficients $a_i$-s of $f$ can be recovered in time polynomial in $d$ and the maximum number of bits needed to represent $\alpha_j$-s and $\beta_j$-s.*

LEMMA 4. *[41] Let $F_N$ denote the $N$-th Fibonacci number and let $r_N = \frac{F_{N+1}}{F_N}$. Then $r_i \neq r_j$ for any $i \neq j$.*

## B.2 Step2: #$\mathcal{P}$-hardness of #RO×RO-4Partite-4CNF

Here we prove the following theorem.

THEOREM 5. *#RO×RO-4Partite-4CNF is #$\mathcal{P}$-hard.*

Given an instance of #4Partite-4BEC, we construct an instance of #RO×RO-4Partite-4CNF as follows (as done in [7]): Let the #4Partite-4BEC instance be $G(V_1, V_2, E)$. There is a variable $x_e$ for every edge $e \in E$. The two DNF expressions $\psi_1, \psi_2$ correspond to bipartite vertex sets $V_1, V_2$ respectively. There is a clause $\sum_{e \ni v} x_e$ in $\psi_1$ (resp. $\psi_2$) for every vertex $v \in V_1$ (resp $v \in V_2$). Note that $E' \subseteq E$ is an edge-cover of $G$ if and only if by assigning 1 to the variables $x_e$, $e \in E'$ and assigning 0 to the variables $x_e$, $e \notin E'$ we get a satisfying assignments of $\psi_1 \cdot \psi_2$. Hence given an oracle for computing $\mathtt{C}(\psi_1 \cdot \psi_2)$, the number of edge covers in $G$ can be exactly computed.

By construction, both $\psi_1, \psi_2$ are positive and RO (since $G$ is bipartite). Moreover, since degree of every vertex in $V_1, V_2$ is bounded by 4, the number of variables in every clause is also bounded by 4. The set of edges $E$ can be partitioned into four disjoint matchings $P_i$ ($i \in [1, 4]$). However, two variables $x_e, x_{e'}$ co-occur in a clause in $\psi_1$ or $\psi_2$ if and only if $e$ and $e'$ have a common endpoint. Therefore, the variable set $\{x_e : e \in E\}$ can be partitioned into four groups $X_i$, where $X_i = \{x_e : e \in P_i\}$ ($i \in [1, 4]$), such that no two variables from the same $X_i$ will co-occur in a clause in $\psi_1$ or $\psi_2$. Finally, each edge has an endpoint in both $V_1$ and $V_2$, hence $\mathtt{Var}(V_1) = \mathtt{Var}(V_2)$. Hence $\psi_1 \cdot \psi_2$ is an instance of the #RO×RO-4Partite-4CNF problem. Since #4Partite-4BEC is #$\mathcal{P}$-hard, #RO×RO-4Partite-4CNF is also #$\mathcal{P}$-hard.

## B.3 Step3: #$\mathcal{P}$-hardness of #RO×RO-4Partite-4DNF

Here we show that the #RO×RO-4Partite-4DNF problem is #$\mathcal{P}$-hard by a reduction from #RO×RO-4Partite-4CNF that uses the properties of RO expressions given by Lemma 1.

THEOREM 6. *#RO×RO-4Partite-4DNF is #$\mathcal{P}$-hard.*

PROOF. Consider an instance of #RO×RO-4Partite-4CNF, where we are given two positive RO CNF expressions $\psi_1, \psi_2$ that are defined on the variable set, such that every clause in $\psi_1, \psi_2$ has at most four variables. The goal is to compute $\mathtt{C}(\psi_1 \cdot \psi_2)$. Then using Lemma 1, $\mathtt{C}(\overline{\psi_1} \cdot \overline{\psi_2})$ is #$\mathcal{P}$-hard (otherwise given an oracle to compute $\mathtt{C}(\overline{\psi_1} \cdot \overline{\psi_2})$, $\mathtt{C}(\psi_1 \cdot \psi_2)$ can be exactly computed). Since $\psi_1, \psi_2$ are positive CNF RO expression where each clause has at most four positive variables, both $\overline{\psi_1}, \overline{\psi_2}$ are DNF RO expressions, where each term has at most four variables (all are negated literals).

From $\overline{\psi_1}, \overline{\psi_2}$, we construct two positive DNF RO expressions $\eta_1, \eta_2$, by making each negated variable in each term of $\overline{\psi_1}, \overline{\psi_2}$ positive in $\eta_1, \eta_2$ respectively. Hence $\eta_1, \eta_2$ are positive DNF RO expressions, where each term has at most four positive variables, and therefore $\eta_1 \cdot \eta_2$ is an instance of the #RO×RO-4Partite-4DNF problem. Let $N = |\mathtt{Var}(\psi_1)| = |\mathtt{Var}(\psi_2)| = |\mathtt{Var}(\eta_1)| = |\mathtt{Var}(\eta_2)|$. It is easy to verify that, $\mathbf{x} = \langle x_1, \cdots, x_N \rangle$ is a satisfying assignment of $\overline{\psi_1} \cdot \overline{\psi_2}$, if and only if $\mathbf{y} = \langle \overline{x_1}, \cdots, \overline{x_N} \rangle$ is a satisfying assignment of $\eta_1 \cdot \eta_2$. Hence there is a one-one correspondence between the satisfying assignments of $\overline{\psi_1} \cdot \overline{\psi_2}$ and $\eta_1 \cdot \eta_2$, and therefore $\mathtt{C}(\overline{\psi_1} \cdot \overline{\psi_2}) = \mathtt{C}(\eta_1 \cdot \eta_2)$. Combining the above two steps, if there is an oracle to solve #RO×RO-4Partite-4DNF, we can also solve #RO×RO-4Partite-4CNF. Since #RO×RO-4Partite-4CNF is #$\mathcal{P}$-hard, #RO×RO-4Partite-4DNF is also #$\mathcal{P}$-hard. □

## B.4 Step4: #$\mathcal{P}$-hardness of $\Pr[q_1 - q_2]$

Finally we reduce #RO×RO-4Partite-4DNF to probability computation of a query with difference and thus prove Step4.

PROOF. Consider an instance of #RO×RO-4Partite-4DNF: $\eta_1 \cdot \eta_2$. To prove the #$\mathcal{P}$-hardness of exact computation of $\Pr[\phi_1 \cdot \overline{\phi_2}]$ (equivalently $\mathtt{C}(\phi_1 \cdot \overline{\phi_2})$ since all the uncertain variables $x$ will have $\Pr[x] = \frac{1}{2}$) it suffices to construct two queries $q_1, q_2$ along with an instance of probabilistic database such that $\phi_1 = q_1(I) = \eta_1$ and $\phi_2 = q_2(I) = \eta_2$. This again follows from Lemma 1, since given an oracle for $\mathtt{C}(\phi_1 \cdot \phi_2)$, $\mathtt{C}(\phi_1 \cdot \overline{\phi_2})$ can be exactly computed.

(1) First we extend the minterms (any term in an RO DNF expression is a minterm) with one, two or three variables to have exactly four variables by inserting dummy variables which always assume value 1 (in probabilistic database they correspond to deterministic tuples). Let the corresponding RO DNF expressions be $\eta_1'$ and $\eta_2'$. Always fresh variables are inserted in every clause of $\eta_1$ and $\eta_2$ and therefore $\mathtt{Var}(\eta_1') \neq \mathtt{Var}(\eta_2')$. However, $\mathtt{C}(\eta_1 \cdot \eta_2) = \mathtt{C}(\eta_1' \cdot \eta_2')$ (the satisfying assignments have a one-one correspondence).

(2) Now recall that the variables $V = \mathtt{Var}(\eta_1) = \mathtt{Var}(\eta_2)$ can be partitioned into four disjoint groups $V_1, \cdots, V_4$ such that no two variables $x, y$ from the same group $V_i$, $i \in [1, 4]$ do not co-occur in a minterm in $\eta_1$ or $\eta_2$. Since we always inserted fresh variables in the minterms of $\eta_1, \eta_2$, this property

also holds for $\eta_1'$ and $\eta_2'$ by arbitrarily assigning new variables to different groups. Hence, every minterm will have a variable from each of the groups $V_1, \cdots, V_4$.

(3) The probabilistic database instance $I$ has four relations $R_1(A,B), R_2(A,B), R_3(A,B), R_4(A,B)$, all with two attributes $A$ and $B$, that contain tuples annotated with the variables in $X_1, \cdots, X_4$ respectively. Let the number of minterms in $\eta_1'$ (resp. $\eta_2'$) be $m_1$ (resp. $m_2$). We fill out the attribute values s of the four tables in the following way:

(a) Let the $i$-th minterm in $\eta_1'$ be $\langle x_{i_1} y_{i_2} z_{i_3}, w_{i_4}\rangle$, where $x_{i_1} \in R_1$, $y_{i_2} \in R_2$, $z_{i_3} \in R_3$, and $w_{i_4} \in R_4$. Assign $a_i$ as the values of attribute $A$ in the $i_1$-th row of $R_1$, $i_2$-th row of $R_2$, $i_3$-th row of $R_3$ and $i_4$-th row of $R_4$, for all $i \in [1, m_1]$.

(b) Similarly, let the $j$-th minterm in $\eta_2'$ be $\langle x_{j_1} y_{j_2} z_{j_3}, w_{j_4}\rangle$, where $x_{j_1} \in R_1$, $y_{j_2} \in R_2$, $z_{j_3} \in R_3$, and $w_{j_4} \in R_4$. Assign $b_j$ as the values of $B$ in the $j_1$-th row of $R_1$, $j_2$-th row of $R_2$, $j_3$-th row of $R_3$ and $j_4$-th row of $R_4$, for all $j \in [1, m_2]$.

(c) Now, due to the introduction of dummy variable, and possible unequal values of $m_1, m_2$, some of the attribute values in the tables $R_1, \cdots, R_4$ may be unassigned after the above two steps. Every such unassigned positions are filled with a fresh attribute value not used before.

(4) Let $N_i = |X_i|$ $(i \in [1,4])$. Let the value of the tuple in the $j$-th row of $R_1, \cdots, R_4$ be $a_j, b_j, c_j$ and $d_j$ respectively, $j \in [1, N_i]$. The original variables in $V = \texttt{Var}(\eta_1) = \texttt{Var}(\eta_2)$ appear with probability $\frac{1}{2}$, whereas the new variables inserted in $\eta_1'$ and $\eta_2'$ appear with probability 1.

(5) Finally, $q_1() := R_1(x,y_1)R_2(x,y_2)R_3(x,y_3)R_4(x,y_4)$ and $q_2() := R_1(x_1,y)R_2(x_2,y)R_3(x_3,y)R_4(x_4,y)$.

Since exactly the tuples appearing in the same minterms of $\eta_1', \eta_2'$ join in $q_1, q_2$ respectively, it easily follows that $q_1(I) = \eta_1'$ and $q_2(I) = \eta_2'$. Clearly, $q_1$ and $q_2$ are queries in $\text{CQ}^-$. Further, both boolean queries $q_1, q_2$ are hierarchical (i.e. for every two variables $x, y$, the sets of subgoals that contain $x, y$ are either disjoint or one is contained in the other) and therefore are safe [11]. □

This completes the proof of Theorem 1.

# C. PROOFS FROM SECTION 4

## C.1 Karp-Luby General Framework

The framework is presented in Algorithm 2.

**Algorithm 2** *Karp-Luby algorithm*

**Input: A boolean expression $\phi = \phi_1 + \cdots + \phi_m$ where properties (Q1), (Q2), (Q3) hold for each $\phi_i$ ($\phi_i$ are not necessarily DNF minterms), accuracy parameter $\epsilon$, confidence parameter $\delta$**

**Output: An estimation of** $\Pr[\phi]$.

Initialize $C = 0$.
**for** $t = 1$ to $M$ do **do** {/* $M$ = *number of samples* */}
– Sample $\phi_i$ w.p. $\frac{\Pr[\phi_i]}{\sum_j \Pr[\phi_j]}$.
– Sample a random satisfying assignment $\sigma$ of $\phi_i$.
**if** $\sigma$ does not satisfy any of $\phi_1, \phi_2, \cdots, \phi_{i-1}$ **then**
– $C = C + 1$.
**end if**
**end for**
– Output $\frac{C}{M} \cdot \sum_j \Pr[\phi_j]$.

It is well-known that $\mathbb{E}[\frac{C}{M} \cdot \sum_j \Pr[\phi_j]] = \Pr[\phi]$, and a set of samples of size $O(\frac{m}{\epsilon^2}\log(\frac{1}{\delta}))$ suffices to estimate $\Pr[\phi]$ within accuracy $(1 \pm \epsilon)$ with probability $\geq 1 - \delta$ [26, 30].

## C.2 Correctness of the sampling procedure in Algorithm 1

A d-DDNNF $\mathcal{D}$ with root $\texttt{r}$ represents the expression $\phi_{\texttt{r}}$, ($\phi_u$ is the sub-expression at node $u$ of $\mathcal{D}$). Here we prove that, for every node $u \in \texttt{Var}(\phi_{\texttt{r}})$, Algorithm 1 assigns a random assignment $\sigma_u$ of the variables $\texttt{Var}(\phi_u)$ with probability $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]}$ (which shows that at the end, a random satisfying assignment of $\texttt{Var}(\phi_{\texttt{r}})$ will be output with probability $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]}$). The proof is by induction on the reverse topological order $\pi$ on $\texttt{Var}(\phi_u)$. The first node $u$ in $\pi$ must be a sink node, and if $u$ is labeled with $x$ (resp. $\overline{x}$), the *unique* satisfying assignment $\sigma_u$ will be $x = 1$ (resp. 0) which is assigned with probability 1. Assume that the induction holds up to the $i$-th node in order $\pi$ and consider the $i+1$-th node $u$ with children $u_1, \cdots, u_\ell$. If $u$ is a $\cdot$-node, then by the *disjointness* of $\cdot$-nodes, $\texttt{Var}(u_j) \cap \texttt{Var}(u_\ell) = \emptyset$. Hence $\phi_{u_j}$ and $\phi_{u_\ell}$ are independent, and $\Pr[\phi_u] = \Pi_{j=1}^k \Pr[\phi_{u_j}]$. Each satisfying assignment $\sigma_u$ of $\phi_u$ must be a concatenation of satisfying assignments $\sigma_{u_j}$, $j = 1$ to $k$, where $\sigma_{u_j}$ is the projection of the assignment $\sigma_u$ on variables $\texttt{Var}(\phi_{u_j})$, and since the variables are disjoint in all $\phi_{u_j}$, $\Pr[\sigma_u] = \Pi_{j=1}^k \Pr[\sigma_{u_j}]$. By induction hypothesis, $\sigma_{u_j}$ is assigned with probability $\frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$. Therefore $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]} = \Pi_{j=1}^k \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$. On the other hand, if $u$ is a $+$-node, $\Pr[\phi_u] = \sum_{j=1}^k \Pr[\phi_{u_j}]$ (satisfying assignments of every $\phi_{u_j}, \phi_{u_\ell}$ are disjoint). For a satisfying assignment $\sigma_u$ of $\phi_u$, let $\sigma_u$ satisfies $\phi_{u_j}$ ($j$ is unique) which is assigned with probability $\frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]}$ (even after the extension in Step 12). Therefore $\frac{\Pr[\sigma_u]}{\Pr[\phi_u]} = \frac{\Pr[\phi_{u_j}]}{\sum_\ell \Pr[\phi_{u_\ell}]} \cdot \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_{u_j}]} = \frac{\Pr[\sigma_{u_j}]}{\sum_\ell \Pr[\phi_{u_\ell}]} = \frac{\Pr[\sigma_{u_j}]}{\Pr[\phi_u]}$.

## C.3 Hardness for General SPJUD Queries with Difference Rank 1

The reduction is from counting the number of independent sets in a graph. Given a graph $G(V,E)$, consider the relational schema $(V, E, S)$ where $V(A), S(A)$ are unary relations while $E(A_1, A_2)$ is binary. $V$ and $E$ capture the vertex and edges in $G$, whereas $S$ captures a subset of vertices, in particular, the set of possible worlds of $I$ which correspond to an independent set in $G$. The tuple variables in $V, E$ are deterministic, and appear with probability 1, whereas, every tuple variable in $S$ appears with probability $\frac{1}{2}$. As we discussed in Section 2.4, the independence sets in a graph can be captured by the SPJUD query $q_{ind-set} = \mathit{True} - [E \bowtie \rho_{A_1/A} S \bowtie \rho_{A_2/A} S]$. Clearly, $q_{ind-set} = 1$.

Let $\phi$ be the boolean provenance of the unique tuple in $q_{ind-set}(I)$. Clearly, $\Pr[\phi] = \frac{N_{IS}}{2^n}$, where $n = |V|$ and $N_{IS}$ is the number of independent sets in $G$. It is known that counting independent sets in an arbitrary graph does not have any non-trivial approximation unless $\mathcal{P} = \mathcal{NP}$ [16]. This shows the inapproximability of tuple probabilities generated by SPJUD queries even if the query has difference rank 1 and proves Theorem 2.

**Remark:** The above query $q_{ind-set}$ uses self-join. The hardness can be extended to queries without self-join under the weaker assumption that counting independent sets in bipartite graphs do not have any approximation [16].