Computing Expected Multiplicities for Bag-TIDBs with Bounded Multiplicities

Su Feng
Boris Glavic
sfeng14@hawk.iit.edu
bglavic@iit.edu
Illinois Institute of Technology, USA
Chicago, New York, USA

Aaron Huber
Oliver Kennedy
Atri Rudra
ahuber@buffalo.edu
okennedy@buffalo.edu
atri@buffalo.edu
University at Buffalo, USA
Buffalo, New York, USA

ABSTRACT
We study the problem of computing a query result tuple’s expected multiplicity for probabilistic databases under bag semantics (where each tuple is associated with a multiplicity) exactly and approximately. Specifically, we are interested in the fine-grained complexity of this problem for c-TIDBs, i.e., probabilistic databases where tuples are independent probabilistic events and the multiplicity of each tuple is bound by a constant c. Unfortunately, our results imply that computing expected multiplicities for c-TIDBs introduces super-linear overhead over the corresponding deterministic query evaluation algorithms (under certain complexity hardness conjectures). Next, we develop a sampling algorithm that computes a (1 + ε)-approximation of the expected multiplicity of an output tuple in time linear in the runtime of the corresponding deterministic query for any positive relational algebra (RA) query over c-TIDBs and for a non-trivial subclass of block-independent databases.

ACM Reference Format:
Su Feng, Boris Glavic, Aaron Huber, Oliver Kennedy, and Atri Rudra. 2022. Computing Expected Multiplicities for Bag-TIDBs with Bounded Multiplicities. In Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym ‘XX’). ACM, New York, NY, USA, 34 pages. https://doi.org/XXXXXXX

1 INTRODUCTION
We explore the problem of computing the expectation of the multiplicity of a tuple in the result of a query over a c-TIDB (tuple independent database), a type of probabilistic database with bag semantics where the multiplicity of a tuple is a random variable with range [0, c] = {0, 1, . . . , c} for some fixed constant c, and multiplicities assigned to any two tuples are independent of each other. Formally, a c-TIDB, D = ([0, c]P, P) is defined over a deterministic bounding database (i.e., a ‘base’ set of tuples) D and a probability distribution P over all possible worlds generated by assigning each tuple t ∈ D a multiplicity in the range [0, c]. Any such world can be encoded as a vector (of length n = |D|) from [0, c]D, such that the multiplicity of each t ∈ D is stored at a distinct index. A given world W ∈ [0, c]D can be interpreted as follows: for each t ∈ D, Wt is the multiplicity of t in W. We note that encoding a possible world as a vector, while non-standard, is equivalent to encoding it as a bag of tuples (Appendix A.2). Given that tuple multiplicities are independent events, the probability distribution P can be expressed compactly by assigning each tuple a probability distribution over [0, c]. Let πt denote the probability that tuple t is assigned multiplicity j. The probability of a world W is then Πt∈D πtWt, where Wt is the multiplicity of a result tuple t in query Q over possible world W ∈ [0, c]D. We now formally state our problem of computing the expected multiplicity:

Figure 1: Lineage polynomial semantics for any RA query Q, arbitrary deterministic bounding database D (with variables X = (Xt)t∈D), where for R ∈ D, t ∈ R, the base case is Φ[¢, ¢, ¢]t(X) = Xt.

Problem 1.1. Given c-TIDB D = ([0, c]P, P), RA query Q, and result tuple t, compute the expectation Ew∼P [Q(W)(t)].

As also observed in [28, 29], computing the expected multiplicity of a result tuple in a bag probabilistic database is the analog of computing the marginal probability in a set PDB.

1 An RA query is a query expressed in positive relational algebra, i.e., using only the operators selection (σ), projection (π), natural join (⋈) and union (∪).
Hardness of Set Query Semantics and Bag Query Semantics.

Set query evaluation semantics over 1-TIDBs have been studied extensively, and their data complexity has, in general been shown to be #P-hard [13]. In an independent work, Grohe et. al. [28] studied bag-TIDBs with unbounded multiplicities, which requires a succinct representation of probability distributions over infinitely many multiplicities and proved a dichotomy for computing the probability of an output tuple’s multiplicity being bounded by given s.

Our Setup. In contrast to [28], we consider c-TIDBs, i.e., the multiplicity of input tuples is bound by a constant c. Then, there exists a trivial PTIME algorithm for computing the expectation of a result tuple’s multiplicity (Problem 1.1) for any fixed RA* query due to linearity of expectation (see Sec. 1.1). Since the data complexity of Problem 1.1 is in PTIME, we then explore the question of computing expectation using fine-grained and parameterized complexity, where we are interested in the exponent of polynomial runtime.2

Specifically, in this work we ask if Problem 1.1 can be solved in time linear in the runtime of an analogous deterministic query, which we make more precise shortly. If true, this opens up the way for deployment of c-TIDBs in practice. We expand on the practical implications of this problem later in the section but for now we stress that in practice, c is indeed constant and most often c = 1. That is, although production database systems use bag semantics for query evaluation, allowing duplicate intermediate or output tuples, input tuples in real world datasets are still frequently unique. To analyze this question we denote by $T^*(Q, D, c)$ the optimal runtime complexity of computing Problem 1.1 over c-TIDB $D$ and query $Q$.

Our lower bound results. Let $T_{det}(Q, D, c)$ (see Sec. 2.4 for a formal definition) denote the runtime for query $Q$ over any deterministic database that maps each tuple in $D$ to a multiplicity in $[0, c]$. Our question is whether or not it is always true that for every $Q$, $T^*(Q, D, c) \leq O(T_{det}(Q, D, c))$. We remark that the issue of query optimization is orthogonal to this question (recall that an RA* query explicitly encodes order of operations) since we want to answer the above question for all $Q$. Specifically, if there is an equivalent and more efficient query $Q'$, we allow both deterministic and probabilistic query processing access to $Q'$.

Unfortunately the answer to the above question is no- Table 1 shows our results. Specifically, depending on what hardness result/conjecture we assume, we get various weaker or stronger versions of no as an answer to our question. To make some sense of the lower bounds in Table 1, we note that it is not too hard to show that $T^*(Q, D, c) \leq O((T_{det}(Q, D, c))^k)$, where $k$ is the join width of $Q$ (our notion of join width is essentially the degree of the corresponding lineage polynomial we introduce in Sec. 1.1). What our lower bound in the third row says, is that for a specific family of hard queries, one cannot get more than a polynomial improvement (for fixed $k$) over essentially the trivial algorithm for Problem 1.1, assuming the Exponential Time Hypothesis (ETH) [31]. We also note that existing results3 imply the claimed lower bounds if we replace the $T_{det}(Q, D, c)$ by just $n = |D|$. Our contribution is to identify a family of hard queries where deterministic query processing is ‘easy’ but computing the expected multiplicities is hard.

Our upper bound results. We introduce a $(1 + \varepsilon)$-approximation algorithm that computes Problem 1.1 in time $O_e(T_{det}(Q, D, c))$.

This means, when we are okay with approximation, that we solve Problem 1.1 in time linear in the size of the deterministic query. In contrast, known approximation techniques ([34, 41]) in set-PDBs need time $\Omega(T_{det}(Q, D, c)^2)$ (see Appendix F). Further, our approximation algorithm works for a more general notion of bag PDBs beyond c-TIDBs (see Sec. 2.2).

1.1 Polynomial Equivalence

A common encoding of probabilistic databases (e.g., in [1, 4, 30] and many others) annotates tuples with lineages, propositional formulas that describe the set of possible worlds that the tuple appears in. The bag semantics is a provenance/lineage polynomial (see Fig. 1) $\Phi(Q, D, t)$ [26], a polynomial with non-zero integer coefficients and exponents, over variables $X$ encoding input tuple multiplicities. The lineage polynomial for result tuple $t_{out}$ evaluates to $t_{out}$’s multiplicity in a given possible world when each $X_{tm}$ is replaced by the multiplicity of $t_{in}$ in the possible world.

We now state the problem of computing the expectation of tuple multiplicity in terms of lineage polynomials (which is equivalent to Problem 1.2—see Proposition A.5):

**Problem 1.2 (Expected Multiplicity of Lineage Polynomials).** Given an RA* query $Q$, c-TIDB $D$ and result tuple $t_{out}$, compute $E_{W_{\Phi}}[\Phi(Q, D, t_{out})]$.

We drop $Q$, $D$, and $t$ from $\Phi(Q, D, t)$ when they are clear from the context or not relevant to the discussion. All of our results rely on working with a reduced form $\Phi$ of the lineage polynomial $\Phi$. As we show, for the 1-TIDB case, computing the expected multiplicity (over bag query semantics) is exactly the same as evaluating $\Phi$ over the probabilities that define the 1-TIDB. Further, only light extensions (see Definition 2.4) are required to support block independent disjoint probabilistic databases [41].

Next, we motivate the reduced polynomial $\Phi$. Consider the query $Q_1$ defined as follows over the bag relations of Fig. 2:

```
SELECT DISTINCT 1 FROM T t1, R r , T t2
WHERE t1.Point = r.Point1 AND t2.Point = r.Point2
```

It can be verified that $\Phi(A, B, C, E, U, Y, Z)$ for the sole result tuple of $Q_1$ is $AUB + BYE + BZC$. Now consider the product query $Q_1^2 = Q_1 \times Q_1$. The lineage polynomial for $Q_1^2$ is $\Phi_2^2 = \Phi_2(A, B, C, E, U, Y, Z)$

$$= A^2U^2 + B^2Y^2 + B^2Z^2 + 2AUUB + 2AYUE + 2BZC + 28^2YEZC.$$

To compute $E[\Phi_2^2]$, we can use linearity of expectation and push the expectation through each summand. To keep things simple, let

---

2While [28] also observes that computing the expectation of an output tuple multiplicity is in PTIME, it does not investigate the fine-grained complexity of this problem.

3This claim follows when we set $Q$ to the query that counts the number of $k$-cliques over database $D$ that encodes a graph. Precisely the same bounds as in the three rows of Table 1 (with $n$ replacing $T_{det}(Q, D, c)$) follow from the same complexity assumptions we make: triangle detection hypothesis (by definition), $\#W[0] \neq \#W[1]$ [23] and Strong ETH [10]. For the last result we can replace $k/\log k$ by just $k$. 
Thus, the idea is to replace the variable $x_i$ all corresponding to the event that $	ilde{b}$. Since $b$, we have $E_{[W_2]}$, since $W_{U_1}$, we get stuck with $E_{[W_2]}$, since $W_{U_2} \in \{0, 1\}$ and for $W_{U_1} \leftarrow 2, W_{U_2} \neq W_{U_2}$.

The simple insight to get around this issue is to note that the random variables $W_{U_1}$ and $W_{U_2} + 2W_{U_2}$ have exactly the same distribution, where $W_{U_1}, W_{U_2} \in \{0, 1\}$ and $Pr[W_{U_1} = 1] = Pr[W_{U_2} = j]$. Thus, the idea is to replace the variable $U$ by $U_1 + 2U_2$ (where $U_j$ corresponds to the event that $U$ has multiplicity $j$) yielding

$$M_{1, R}(A, U_1, U_2, B) \overset{\text{def}}{=} M_{1}(A, U_1 + 2U_2, W_{U_2}).$$

Given that $U$ can only have multiplicity of 1 or 2 but not both, given world vectors $(W_{U_1}, W_{U_2}, W_{U_2}, W_{U_2}) \in \{0, 1\}$, we have $E_{[W_{U_2}]}$. Further, since the world vectors are binary vectors, we have $E_{[W_{U_2}]} = E_{[W_{U_2}]} + 4E_{[W_{U_2}]} = E_{[W_{U_2}]}$.

We only did the argument for a single monomial but by linearity of expectation we can apply the same argument to all monomials in $\Phi^2$. Generalizing this argument to arbitrary $\Phi$ leads us to consider its following ‘reduced’ version:

**Definition 1.3.** For any polynomial $\Phi(X_i)_{i \in D}$ define the re-formulated polynomial $\Phi_R \left( X_{i,j} \right)_{i \in D, j \in [c]}$ to be the polynomial $\Phi_R = \Phi \left( \sum_{j \in [c]} \left( \sum_{i \in D} X_{i,j} \right) \right)$ and ii) define the reduced polynomial $\Phi_R \left( X_{i,j} \right)_{i \in D, j \in [c]}$ to be the polynomial $\Phi_R = \Phi \left( \sum_{j \in [c]} \left( \sum_{i \in D} X_{i,j} \right) \right)$ and setting each variable’s exponents $\geq 1$ to 1.

As we have essentially argued earlier, the expression for our specific example is $\Phi_R(Pr(A = 1), \ldots, Pr(Z = 1))$. This equivalence generalizes to all $\mathcal{RA}^+$ queries on c-TIDBs (proof in Appendix A.7):

**Lemma 1.4.** For any c-TIDB $D, \mathcal{RA}^+$ query $Q$, and lineage polynomial $\Phi(X)$, it holds that $\mathbb{E}_{W \sim P} \{ \Phi(W) \} = \Phi(p)$, where $p = \left( \langle \rho_{W,j} \rangle_{i \in D, j \in [c]} \right)$.

**1.2 Our Techniques**

**Lower Bound Proof Techniques.** To prove the lower bounds in Table 1 we show that for the same $Q_1$ from the example above, for an arbitrary ‘product width’ $k$, the query $Q^k_{\text{hard}}$ is able to encode various hard graph-counting problems (assuming $O(n)$ tuples rather than the $O(1)$ tuples in Fig. 2). We do so by considering an arbitrary graph $G$ (analogous to relation $R$ of $Q_1$) and analyzing how the coefficients in the (univariate) polynomial $\Phi(p_1, \ldots, p_r)$ relate to counts of subgraphs in $G$ that are isomorphic to various subgraphs with $k$ edges. E.g., for the last two rows in table 1, we exploit the fact that the coefficient corresponding to $p_1^2$, $\Phi(p_1, \ldots, p_r)$ of $Q^k_{\text{hard}}$ is proportional to the number of $k$-matchings in $G$, a known hard problem in parameterized/fine-grained complexity literature.

**Upper Bound Techniques.** Our negative results (Table 1) indicate that c-TIDBs (even for $c = 1$) cannot achieve comparable performance to deterministic databases for exact results (under complexity assumptions). In fact, under well established hardness conjectures, one cannot (drastically) improve upon the trivial algorithm to exactly compute the expected multiplicities for 1-TIDBs. A natural followup is whether we can do better if we are willing to settle for an approximation to the expected multiplicities.

We adopt the two-step intensional model of query evaluation used in set-PDBs, as illustrated in Fig. 2: (i) Lineage Computation (LC): Given input $D$ and $Q$, output every tuple that possibly satisfies $Q$, annotated with its lineage polynomial $\Phi(X)$; (ii) Expectation Computation (EC): Given $\Phi(X)$ for each tuple, compute a $(1 \pm \varepsilon)$-approximation $\mathbb{E}_{W \sim P} \{ \Phi(W) \}$. Let $T_{LC}(Q, D, C)$ denote the runtime of LC when it outputs $C$ (a representation of $\Phi$ as an arithmetic circuit — more on this representation in Sec. 2.3). Denote by $T_{EC}(C, \varepsilon)$ (recall $C$ is the output of LC) the runtime of EC when $\Phi$ is input as $C$. Then to answer if we can compute a $(1 \pm \varepsilon)$-approximation to the expected multiplicity in time linear to the deterministic query, it is enough to answer the following:

**Problem 1.5 (c-TIDB Linear Time Approximation).** Given c-TIDB $D, \mathcal{RA}^+$ query $Q$, is there a $(1 \pm \varepsilon)$-approximation of $\mathbb{E}_{W \sim P} \{ Q(W) \}$ for all result tuples $w$ where there exists $C : T_{LC}(Q, D, C) + T_{EC}(C, \varepsilon) \leq O(1)$?
in SMB, the answer to the above question in general is no, since then we will need $|C| \geq \Omega \left( (T_{det} (Q, D, c))^k \right)$, where $|C|$ is the size of circuit $C$. Hence, just $T_{LC} (Q, D, C)$ is too large. However, systems can directly emit compact, factorized representations of $\Phi (X)$, e.g., as a consequence of the standard projection push-down optimization [25]). Accordingly, this work uses (arithmetic) circuits\footnote{An arithmetic circuit is a DAG with variable/numeric source gates and multiplication/addition internal gates.} as the representation system of $\Phi (X)$, and we show in Appendix D.2.1 an $O (T_{det} (Q, D, c))$ algorithm for constructing the lineage polynomial for all result tuples of an $\mathcal{RA}^+$ query $Q$ (more precisely, a circuit $C$ with $n$ sinks, one per output tuple). Since a representation $C^*$ exists where $T_{LC} (Q, D, C^*) \leq O (T_{det} (Q, D, c))$ and the size of $C^*$ is bounded by $T_{det} (Q, D, c)$ (i.e., $|C^*| \leq O (T_{det} (Q, D, c)))$ (see Appendix D.2.2), we can focus on the complexity of $EC$. Thus, to solve Problem 1.5, it is sufficient to solve:

**Problem 1.6.** Given any circuit $C$ that encodes $\Phi (X)$ for all result tuples $t$ (one sink per $t$) for c-TIDB $\mathcal{D}$ and $\mathcal{RA}^+$ query $Q$, does there exist an algorithm that computes a $(1 \pm \varepsilon)$-approximation of $\mathbb{E}_{W \cdot \Phi} [\Phi (W) (t)]$ for all result tuples $t$ in $O (|C|)$ time?

We will formalize the notions of circuits and hence, Problem 1.6 in Sec. 2. For an upper bound on approximating the expected count, it is easy to check that if all the probabilities are constant then (with an additive adjustment) $\Phi (p_1, \ldots, p_n)$ is a constant factor approximation of $\Phi$ (where we assume $D = \{n\}$). This is illustrated in the following example using $Q_2^+$ from earlier. To aid in presentation we again limit our focus to $M_{1,R} (A, U, B)$, assume $c = 2$ for variable $U$ and $c = 1$ for all other variables. Recall $Pr_X$ denotes $Pr [X = 1]$. Then we have: $M_{1,R} (X) = A^2 (U_2^2 + 4U_1U_2 + 4U_2^2) B^2 = A^2U_1^2B^2 + 4A^2U_1U_2B^2 + 4A^2U_2B^4$, which in turn implies:

$$M_{1,R} (p) = 4p_2^2p_1p_2p_3 + p_3^2p_1p_3^2p_6 + 4p_2^2p_1p_3^2p_3.$$ 

Noting that $M_1 (X) = AU_1B + 4AU_2B$, if we assume that all probability values are in $[p_0, 1]$ for some $p_0 > 0$, we get that $M_{1,R} (p) = 4p_2^2p_1p_2p_3 + p_3^2p_1p_3^2p_6$ is in the range $[p_0^3 \cdot M_1 (p), p_1 (p)]$. Note however, that this is not a tight approximation. In Sec. 4 we demonstrate that a $(1 \pm \varepsilon)$ (multiplicative) approximation with competitive performance is achievable. To get an $(1 \pm \varepsilon)$-multiplicative approximation and solve Problem 1.6, using $C$ we uniformly sample monomials from the equivalent SMB representation of $\Phi$ (without materializing the SMB representation) and ‘adjust’ their contribution to $\Phi (p)$.

**Applications.** Recent work in heuristic data cleaning [7, 44, 47, 53] emits a PDB when insufficient data exists to select the ‘correct’ data repair. Probabilistic data cleaning is a crucial innovation, as the alternative is to arbitrarily select one repair and ‘hope’ that queries receive meaningful results. Although PDB queries instead convey the trustworthiness of results [38], they are imprecisely slow [18, 19], even in approximation (see Appendix F). Bags, as we consider, are sufficient for production use, where bag-relational algebra is already the default for performance reasons. Our results show that bag-PDBs can be competitive, laying the groundwork for probabilistic functionality in production database engines.

**Paper Organization.** We present background and notation in Sec. 2. We prove our main hardness results in Sec. 3 and present our approximation algorithm in Sec. 4. Finally, we discuss related work in Sec. 5 and conclude in Sec. 6. All proofs are in the appendix.

## 2 BACKGROUND AND NOTATION

### 2.1 Polynomial Definition and Terminology

Given an index set $S$ and variables $X_t$ for $t \in S$, a (general) polynomial $\phi$ over $(X_t)_{t \in S}$ with individual degree $K < \infty$ is formally defined as:

$$\phi ((X_t)_{t \in S}) = \sum_{d = (d_t)_{t \in S} \in \{0, K\}^S} c_{d_t} \cdot \prod_{t \in S} X_{d_t}^t \quad \text{where } c_{d_t} \in \mathbb{N}. \quad (1)$$

**Definition 2.1 (Standard Monomial Basis).** The term $\prod_{t \in S} X_{d_t}^t$ in Eq. (1) is a monomial. A polynomial $\phi (X)$ is in standard monomial basis (SMB) when we keep only the terms with $c_{d_t} \neq 0$ from Eq. (1).

Unless otherwise noted, we consider all polynomials to be in SMB representation. When it is unclear, we use SMB ($\phi$) to denote the SMB form of a polynomial $\phi$. We call a polynomial $\Phi (X)$ a c-TIDB-lineage polynomial (or simply lineage polynomial), if it is clear from context that there exists an $\mathcal{RA}^+$ query $Q$, c-TIDB $\mathcal{D}$, and result tuple $t$ such that $\Phi (X) = \Phi [Q, D, t] (X)$.

### 2.2 Binary-BIDB

A block independent database BIDB $\mathcal{D}'$ models a set of worlds each of which consists of a subset of the deterministic bounding database $\mathcal{D}'$, where $\mathcal{D}'$ is partitioned into $m$ blocks $B_i$ and the random
variables $W_t$ for $t \in B_1$ and $t \in B_j$ are independent for $i \neq j$. $D'$ further constrains that $W_t$ all $t \in B_i$ for the same $i \in [m]$ of $D'$ be disjoint events. We define next a specific construction of BIDB that is useful for our work.

**Definition 2.2 (Binary-BIDB).** Define a Binary-BIDB to be the pair $D' = (X_{i \in D'} \{0, c_i\}, P')$, where $D'$ is the deterministic bounding database such that each $t \in D'$ has a multiplicity in $\{0, c_i\}$, with $c_i \in N$. $D'$ is partitioned into $m$ independent blocks $B_i$, for $i \in [m]$, of disjoint tuples. $P'$ is characterized by the vector $(p_t)_{t \in D'}$ where for every block $B_i$, $\sum_{t \in B_i} p_t \leq 1$. For $W \in X_{i \in D'} \{0, c_i\}$ and $i \in [m]$, let

$$p_t(W) = \begin{cases} 1 - \sum_{t \in B_i} p_t & \text{if } W = 0 \text{ for all } t \in B_i; W_t, W_{t'} \neq 0 \\ p_t & \text{if there exists } t \neq t' \in B_i; W_t, W_{t'} \neq 0 \end{cases}$$

$P'$ is the probability distribution across all worlds such that, given $W \in X_{i \in D'} \{0, c_i\}$, $Pr[W = W'] = \prod_{i \in [m]} p_t(W)$.

Lineage polynomials for arbitrary deterministic bounding database $D'$ are constructed in a manner analogous to 1-TIDBs (see Fig. 1), differing only in the base case. In a 1-TIDB, each tuple contributes a multiplicity of 0 or 1, and $\Phi([R, D], t) = X_t$. In a Binary-BIDB, each tuple $t \in D'$ contributes its corresponding multiplicity: $\Phi([R, D], t) = c_t' - X_t$. See Appendix A.5 for full details.

**Binary-BIDB** are powerful enough to encode c-TIDB:

**Proposition 2.3 (c-TIDB reduction).** Given c-TIDB $D = \{(0, c), P\}$, let $D' = (X_{i \in D'} \{0, c_i\}, P')$ be the Binary-BIDB obtained in the following manner: for each $t \in D$, create block $B_t = \{(t, j)_{j \in [c]}\}$ of disjoint tuples, where $c(t, j) = j$ for each $(t, j) \in D'$. The probability distribution $P'$ is the parameterized by the vector $p = (p_t)_{t \in D, j \in [c]}$. Then, $P$ and $P'$ are equivalent.

We now define the reduced polynomial $\Phi'$ of a Binary-BIDB.

**Definition 2.4 (\Phi').** Given a polynomial $\Phi'(X)$ generated from a Binary-BIDB, let $\Phi'(X)$ denote the reduced $\Phi'(X)$ derived as follows: i) compute $SMB(\Phi'(X))$ eliminating monomials with cross terms $X_t X_{t'}$ for $t \neq t' \in B_i$ and ii) reduce all variable exponents $e > 1$ to 1.

We have the following generalization of Lemma 1.4:

**Lemma 2.5.** Given any Binary-BIDB $D', R^{\Phi'}$ query $Q$, and lineage polynomial $\Phi'(X) = \Phi'([Q, D'], t)(X)$, it holds that $E_{W \sim \rho^W}[\Phi'(W)] = \Phi'(p)$.

Let $[\Phi']$ be the number of operators in $\Phi'$. Then:

**Corollary 2.6.** If $\Phi'$ is a Binary-BIDB lineage polynomial already in SMB, then the expectation of $\Phi'$, i.e., $E_{W \sim \rho^W}[\Phi'(W)]$ can be computed in $O([\Phi'])$ time.

### 2.3 Formalizing Problem 1.6

We focus on the problem of computing $E_{W \sim \rho^W}[\Phi(W)]$ from now on. Problem 1.6 asks if there exists a linear time approximation algorithm in the size of a given circuit $C$ that encodes $\Phi(X)$. Recall that in this work we represent lineage polynomials via *arithmetic circuits* [8], a standard way to represent polynomials over fields (particularly in the field of algebraic complexity), which we use for polynomials over $\mathbb{N}$ in the obvious way. Since we are specifically using circuits to model lineage polynomials, we can refer to these circuits as lineage circuits. However, when the meaning is clear, we will drop the term lineage and only refer to them as circuits.

**Definition 2.7 (Circuit).** A circuit $C$ is a Directed Acyclic Graph (DAG) with source gates (in degree 0) drawn from either $\mathbb{N}$ or $X = (X_i)_{i \in D}$ and one sink gate for each result tuple. Internal gates have binary input and are either sum (+) or product (×) gates. Each gate has the following members: type, input, partial, degree, Lweight, and Rweight, where type is the value type (+, ×, VAR, NUM) and input the list of inputs. Source gates have an extra member val for the value $C_e(C_p)$ denotes the left (right) input of $C$.

We refer to the structure when the underlying DAG is a tree (with edges pointing towards the root) as an expression tree. The circuits (1) and (2) in column $\Phi$ of Fig. 2 are both expression trees. Members not described in Definition 2.7 are defined and used in the appendix.

The function $\text{poly}(\cdot)$ (Appendix A.9) maps a circuit to its corresponding polynomial. We next define its inverse:

**Definition 2.8 (Circuit Set).** $\text{CSet}(\Phi(X))$ is the set of all possible circuits $C$ such that $\text{poly}(C) = \Phi(X)$.

Fig. 3 depicts a circuit $C$ in $\text{CSet}(2X^2 + 3XY - 2Y^2)$. Light-text annotations and the colors can be ignored until Sec. 4. We are now ready to formally state the final version of Problem 1.6.

**Definition 2.9 (The Expected Result Multiplicity Problem).** Let $D$ be an arbitrary c-TIDB and $X$ be the set of variables annotating tuples in $D$. Fix an $R^{\Phi'}$ query $Q$ and a result tuple $t$. The Expected Result Multiplicity Problem is defined as follows:

**Input:** $C \in \text{CSet}(\Phi([Q, D], t)(X))$

**Output:** $\mathbb{E}_{W \sim \rho^W}[\Phi([Q, D], t)(W)]$.

### 2.4 Deterministic Query Runtimes

In Sec. 1, we introduced the function $T_{\text{det}}(\cdot)$ to analyze the runtime complexity of Problem 1.1. To decouple our results from any specific join algorithm, we first lower bound the cost of a join.

**Definition 2.10 (Join Cost).** Denote by $T_{\text{join}}(R_1, \ldots, R_m)$ the runtime of an algorithm for computing the $m$-ary join $R_1 \bowtie \ldots \bowtie R_m$. We require only that the algorithm must enumerate its output, i.e., that $T_{\text{join}}(R_1, \ldots, R_m) \geq |R_1 \bowtie \ldots \bowtie R_m|$.

Worst-case optimal join algorithms [39, 40] and query evaluation via factorized databases [42] (as well as work on FAQs [36] and AJaR [33]) can be modeled as $R^{\Phi'}$ queries (though the query size is data dependent). For these algorithms, $T_{\text{join}}(R_1, \ldots, R_m)$ is linear in the AGM bound [5]. Our cost model for general query evaluation follows:

$$T_{\text{det}}(\sigma_Q, t)(C) = |t| R$$

$$T_{\text{det}}(\sigma_Q, t)(C) = T_{\text{det}}(Q, t)(C)$$
Given positive integer $k$ and undirected graph $G = (V, E)$ with no self-loops or parallel edges, $T_{match}(k, G) \geq o\left(f(k) \cdot |E|^c\right)$ for any function $f$ and any constant $c$ independent of $|E|$ and $k$ (assuming $#W[0] \neq #W[1]$).

Theorem 3.2 ([12]). Given positive integer $k$ and undirected graph $G = (V, E)$, $T_{match}(k, G) \geq |V|^3/\log^2 k$ (assuming ETH).

The above result is saying that, assuming Exponential Time Hypothesis (ETH), one can only hope for a slightly super-polynomial improvement over the trivial algorithm to compute $\#(G, I \cdots I^k)$.

Our hardness result in Section 3.2 is based on the following conjectured hardness result:

Conjecture 3.3. There exists a constant $\alpha_0 > 0$ such that given an undirected graph $G = (V, E)$, computing $\#(G, A)$ exactly cannot be done in time $o\left(|E|^{1+\alpha_0}\right)$.

The so-called Triangle detection hypothesis (cf. [37]), which states that detecting the presence of triangles in $G$ takes time $O\left(|E|^{4/3}\right)$, implies that in Conjecture 3.3 we can take $\alpha_0 \geq \frac{1}{3}$.

All of our hardness results rely on a simple lineage polynomial encoding of the edges of a graph. To prove our hardness result, consider a graph $G = (V, E)$, where $|E| = m, V = [n]$. Our lineage polynomial has a variable $X_i$ for every $i \in [n]$. The hard polynomial for our problem will be a suitable power $k \geq 3$ of the polynomial above:

\[
\Phi_k^G(X_1, \ldots, X_n) = \left(\sum_{(i,j) \in E} X_i \cdot X_j\right)^k.
\]

Returning to Fig. 2, it can be seen that $\Phi_k^G(X)$ is the lineage polynomial from query $Q_{hard}^k$ which we define next.

Select count (*) from $Q_1$ join $Q_1$ join $\ldots$ join $Q_1$ $k$ times.

In the above, $Q_1$ is as defined in Sec. 1, which is the same as $Q_{hard}^k$. We next define the instances for $T$ and $R$ that lead to the lineage polynomial in Definition 3.4 as follows. Relation $T$ has $n$ tuples corresponding to each vertex in $i \in [n]$, each with probability $p$ and $R$ has tuples corresponding to the edges $E$ (each with a probability of $1$). In other words, the deterministic bounding database $D$ contains the set of $n$ unary tuples in $T$ (which corresponds to $V$) and $m$ binary tuples in $R$ (which corresponds to $E$). Note that this implies that $\Phi_k^G$ is indeed a 1-TIDB lineage polynomial.

Next, we note that the runtime for answering $Q_{hard}^k$ on deterministic database $D$, as defined above, is $O_k(m)$ (i.e. deterministic query processing is ‘easy’ for this query):

**Lemma 3.5.** For $Q_{hard}^k$, $D$ as above, $T_{det}\left(Q_{hard}^k, D, c\right)$ is $O_k(m)$.

### 3.1 Multiple Distinct $p$ Values

We are now ready to present one of our main hardness result.

**Theorem 3.6.** Let $p_0, \ldots, p_{2k}$ be $2k + 1$ distinct values in $(0, 1]$. Then computing $\Phi_k^G(p_1, p_2, \ldots, p_i)$ for all $i \in [2k + 1]$ for arbitrary $G = (V, E)$ needs time $\Omega(\min(k, \log k))$. If $T_{match}(k, G)$, $\geq o\left(|E|\right)$.

Note that the second (and third) row(s) of Table 1 follow from Theorem 3.6, Lemma 3.5, and Theorem 3.1 (Theorem 3.2 resp.). Theorem 3.6 follows by observing that $\Phi_k^G(p_1, p_2, \ldots, p_i) = p^{2k} \cdot \#(G, I \cdots I^k)$ and $r(p)$, where $r(p)$ is a polynomial of degree at most $2k - 1$ (with coefficients that just depend on $G$). By polynomial interpolation, knowing the values $\Phi_k^G(p_1, p_2, \ldots, p_i)$ (over all $i \in [2k + 1]$) allows us to compute all the coefficients, including $\#(G, I \cdots I^k)$.

Note that Theorem 3.6 needs one to be able to compute the expected multiplicities over $(2k + 1)$ distinct values of $p_i$, each of which corresponds to distinct $P$ (for the same $D$), which explain the ‘Multiple’ entries in the second column of the second and third rows in Table 1. Next, we argue how to get rid of this latter requirement. 

"Technically, $\Phi_k^G(X)$ should have variables corresponding to tuples in $R$ as well, but since they always are present with probability 1, we drop those. Our argument also works when all the tuples in $R$ also are present with probability $p$ but to simplify notation we assign probability 1 to edges."
3.2 Single \( p \) value

While Theorem 3.6 shows that computing \( \overline{\Phi}(p, \ldots, p) \) for multiple values of \( p \) in general is hard it does not rule out the possibility that one can compute this value exactly for a fixed value of \( p \). One can compute \( \overline{\Phi}(p, \ldots, p) \) exactly in linear time for \( p \in \{0, 1\} \). Next, we show that these are the only two possibilities:

**Theorem 3.7.** Fix \( p \in (0, 1) \). Then assuming Conjecture 3.3, any algorithm that computes \( \overline{\Phi}(p, \ldots, p) \) for arbitrary \( G \in (V, E) \) exactly has to run in time \( \Omega(|E|^{1+\epsilon_0}) \), where \( \epsilon_0 \) is as in Conjecture 3.3.

Note that Lemma 3.5 and Theorem 3.7 above imply the hardness result in the first row of Table 1. We note that Theorem 3.1 and Theorem 3.2 (and the lower bounds in the second and third rows) need \( k \) to be large enough (in particular, we need a family of hard queries). But the above Theorem 3.7 (and the lower bound in first row of Table 1) holds for \( k = 3 \) (and hence for a fixed query).

Unlike the proof of Theorem 3.6, in this case we have to pay close attention to all the coefficients of \( \overline{\Phi}_G(p, \ldots, p) \):

**Lemma 3.8.** For any \( p \), we have:
\[
\overline{\Phi}_G(p, \ldots, p) = \# (G, \Delta_1) p + 6 \# (G, \Delta_2) p^2 + 6 \# (G, \Delta_3) p^3 + 6 \# (G, \Delta_4) p^4 + 6 \# (G, \Delta_5) p^5 + 6 \# (G, \Delta_6) p^6 + 6 \# (G, \Delta_7) p^7 + 6 \# (G, \Delta_8) p^8.
\]

Since \( p \) is fixed, the earlier polynomial interpolation based argument does not work anymore. Next, we use the fact that the algorithm still has to compute \( \overline{\Phi}_G(p, \ldots, p) \) for all graphs \( G \). We focus on the graphs \( G^{(1)} \) obtained from \( G \) by replacing each edge by path of length \( \ell \) (Definition B.3). We then show

**Lemma 3.9.** Fix \( p \in (0, 1) \). Given \( \overline{\Phi}_G^{(1)}(p, \ldots, p) \) for \( \ell \in [2] \), we can compute in \( O(m) \) time a vector \( b \in \mathbb{R}^2 \) such that
\[
\begin{pmatrix}
1 - 3p \\
10(3p^2 - p^3)
\end{pmatrix}
= b,
\]
allowing us to compute \( \# (G, \Delta_1) \) and \( \# (G, \Delta_2) \) in \( O(1) \) time.

Note that eq. (3) only depends on subgraph counts on \( G = G^{(1)} \). It can be verified that the coefficient matrix in eq. (3) is full rank for all \( p \in (0, 1) \). Then by solving the linear equations in eq. (3) we can compute \( \overline{\Phi}_G(p, \Delta, \ldots) \), from where Conjecture 3.3 implies Theorem 3.7.

4 1 ± \( \epsilon \) APPROXIMATION ALGORITHM

We showed in Sec. 3 that a runtime of \( O(T_{det}(Q, D, c)) \) cannot be achieved for Problem 1.2. In light of this, we desire to produce an approximation algorithm that runs in time \( O(T_{det}(Q, D, c)) \). We do this by showing the result via circuits, such that our 1 ± \( \epsilon \) approximation algorithm for this problem runs in \( O(|C|) \) for a very broad class of circuits, (thus solving Problem 1.6); see the discussion after Lemma C.11 for more. The following approximation algorithm applies to bag query semantics over both \( c \)-TIDB lineage polynomials and general BIDB lineage polynomials in practice. Our experimental results (see Appendix C.12), which use queries from the PDBench benchmark [43] support the notion that our bounds hold for general BIDB in practice. Proofs and pseudocode for all formal statements and algorithms are in Appendix C.

4.1 Preliminaries and some more notation

For notational convenience, in this section we will assume that deterministic bounding database \( D' = [n] \). We now introduce definitions related to circuits and polynomials that we will need to state our upper bound results. First, we introduce the expansion \( E(C) \) of circuit \( C \) which is used in our auxiliary algorithm \textsc{SampleMonomial} for sampling monomials when computing the approximation.

**Definition 4.1 (E(C)).** For a circuit \( C \), we define \( E(C) \) as a list of tuples \((v, c)\), where \( v \) is a set of variables and \( c \in \mathbb{N} \). \( E(C) \) has the following recursive definition (\( \epsilon \) is list concatenation):
\[
E(C) = \begin{cases}
E(C_0) \circ E(C_1) & \text{if } C \text{. type } = + \\
((v_L \cup v_R, c_L \cdot c_R) \in E(C_L), (v_R, c_R) \in E(C_R)) & \text{if } C \text{. type } = \times \\
\text{List } \{(\emptyset, C \text{. val})\} & \text{if } C \text{. type } = \text{NUM} \\
\text{List } \{(C \text{. val}, 1)\} & \text{if } C \text{. type } = \text{VAR}.
\end{cases}
\]

Later on, we will denote the monomial composed of the variables \( v \) as \( v_\mathcal{B} \). As an example of \( E(C) \), consider \( C \) illustrated in Fig. 3. \( E(C) \) is then \( [ (X, 2), (X, Y, -1), (Y, 4), (Y, -2) ] \). This helps us redefine \( \overline{\Phi} \) (see Eq. (4)) in a way that makes our algorithm more transparent.

Next, we present a sequence of definitions that will be useful for our algorithm and its analysis.

**Definition 4.2 (\(|C|\)).** For any circuit \( C \), the corresponding positive circuit, denoted \( |C| \), is obtained from \( C \) as follows. For each leaf node \( t \) of \( C \) where \( t \text{. type } \) is \text{NUM}, update \( t \text{. value } \) to \( |t \text{. value } | \).

We will overload notation and use \(|C| (X)\) to mean \( \text{POLY } (|C|) \). Convenitely, \( |C| (1, \ldots, 1) \) gives us \( \sum_{(v, c) \in E(C)} c \).

**Definition 4.3 (size (\( \_ \)), depth (\( \_ \))).** The functions \text{SIZE (\( \_ \))}, \text{DEPTH (\( \_ \))}. The functions \text{SIZE and DEPTH output the number of gates and levels respectively for input C}.

**Definition 4.4 (deg (\( \_ \))).** \( \text{deg}(C) \) is defined recursively:
\[
\text{deg}(C) = \begin{cases}
\max(\text{deg}(C_L), \text{deg}(C_R)) & \text{if } C \text{. type } = + \\
\text{deg}(C_L) + \text{deg}(C_R) + 1 & \text{if } C \text{. type } = \times \\
1 & \text{if } C \text{. type } = \text{VAR} \\
0 & \text{otherwise.}
\end{cases}
\]

We use the following notation for integer multiplication complexity:

**Definition 4.5 (\( \text{Mul} (\_ , \_ ) \)).** \( \text{Mul}(M, W) \) denotes the complexity of multiplying two integers represented with \( M \)-bits. (For input of size \( N \), we make the standard assumption \( W = O(\log N) \)).

Finally, to get linear runtime results, we will need to define another parameter modeling the (weighted) number of monomials in \( E(C) \) that need to be ‘canceled’ when monomials with dependent variables are removed (Sec. 2.2). Let \( \text{ISN}\text{D (\( \_ , \_ \))} \) be a boolean function

\footnote{Note that the degree of \text{POLY (\( |C| \))} is always upper bounded by \text{deg}(C) and the latter can be strictly larger (e.g. consider the case when \( C \) multiplies two copies of the constant 1 – here we have \text{deg}(C) = 1 but degree of \text{POLY (\( |C| \))} is 0).}

\footnote{We note that when doing arithmetic operations on the RAM model for input of size \( N \), we have that \( \text{Mul}(O(\log N), O(\log N)) = O(1) \).}
returning true if monomial \( v_0 \) is composed of independent variables and false otherwise; further, let \( \mathbb{I}_g \) also be a boolean function returning true if \( \bar{0} \) evaluates to true.

**Definition 4.6 (Parameter \( \gamma \)).** Given Binary-BIDB circuit \( C \), let

\[
y(C) = \sum_{(v, c) \in E(C)} \mathbb{1}_{\text{isBIND}(v_0)} \cdot \frac{\mathbb{1}_{\text{isBIND}(v_0)}}{|C| (1, \ldots, 1)}
\]

### 4.2 Our main result

In what follows, we solve Problem 1.6 for any fixed \( \epsilon > 0 \).

**Algorithm Idea.** Our approximation algorithm (\( \text{APPROXIMATE}\)) is based on the above observation. Given a lineage polynomial \( \Phi(X) = \text{poly}(C) \) for circuit \( C \) over Binary-BIDB (recall that all \( c \)-TIDB can be reduced to Binary-BIDB by Proposition 2.3), we have:

\[
\tilde{\Phi}(p_1, \ldots, p_n) = \sum_{(v, c) \in E(C)} \mathbb{1}_{\text{isBIND}(v_0)} \cdot \prod_{X \in \text{ev}} p_i.
\]

Given the above, the algorithm is a sampling based algorithm for the above sum: we sample (via \text{SAMPLEMONOMIAL}) \( (v, c) \in E(C) \) with probability proportional to \( |c| \) and compute \( Y = \mathbb{1}_{\text{isBIND}(v_0)} \cdot \prod_{X \in \text{ev}} p_i \). Repeating the sampling an appropriate number of times and computing the average of \( Y \) gives us our final estimate.

We illustrate \text{SAMPLEMONOMIAL} (Algorithm 3) using the circuit \( C \) in Fig. 3. As a pre-processing step, \text{OnePass} (Algorithm 2) recursively (for each sub-circuit) computes \( |C| (1, \ldots, 1) \) in the obvious manner. The gray values in Fig. 3 represent the value \( |C'| (1, \ldots, 1) \) for each sub-circuit \( C' \) rooted at the corresponding node. E.g. in the bottom right x (blue) gate, the value is \( 1 \times 1 = 1 \) (where the left child has \( Y \leftarrow 1 \) and the right child has \( |1| = 1 \)).

We now consider a partial run of \text{SAMPLEMONOMIAL} that samples \( (X, Y, 1) \) in Fig. 3. It recursively traverses both children of the sink \( \times \) gate. For the red and green children, which are both + gates, we randomly choose one of their children. Specifically \text{SAMPLEMONOMIAL} then randomly picks the right child of the green + gate (i.e. the blue \( \times \) gate that represents \( \gamma - \gamma \)) and is computed by recursing on both children of the blue \( \times \) gate) with probability \( \frac{1}{3} \) (where the numerator and denominator are the values computed by \text{OnePass} for the blue \( \times \) and green + gates respectively). Similarly at the left red + gate we sample the left child (representing \( X \) and is computed by recursing to the leaf node \( X \)) with probability \( \frac{1}{3} \). Note that the probability for choosing \( (XY, -1) \) overall is \( \frac{1}{3} \), which is indeed the ratio of the coefficient of \( (XY, -1) \) to the sum of all coefficients in \( |C| \), as needed. All algorithm details, including those for \( \text{APPROXIMATE} \) (Algorithm 1), are in Appendix C.

**Runtime analysis.** We can argue the following runtime for the \( \text{APPROXIMATE} \) (which solves Problem 1.6):

**Theorem 4.7.** Let \( C \) be an arbitrary Binary-BIDB circuit, define \( \Phi(X) = \text{poly}(C) \), let \( k = \text{dec}(C) \), and let \( \gamma = y(C) \). Further let it be the case that \( p_i \geq p_0 \) for all \( i \in [n] \). Then for all \( \epsilon' > 0 \), an estimate \( \mathcal{E} \) of \( \Phi(p_1, \ldots, p_n) \) satisfying

\[
\Pr\left( |\mathcal{E} - \tilde{\Phi}(p_1, \ldots, p_n)| > \epsilon' \cdot \tilde{\Phi}(p_1, \ldots, p_n) \right) \leq \delta \quad (5)
\]
can be computed in time

\[
O \left( (\text{size}(C) + \log \frac{1}{\delta} \cdot \log \frac{1}{\epsilon'} \cdot |C| / (|C|) \cdot (1 - y)^2 \cdot p_0^2) \cdot \mathcal{M} \left( \log (|C|) (1, \ldots, 1), \log (\text{size}(C)) \right) \right).
\]

In particular, if \( p_0 > 0 \) and \( \gamma < 1 \) are absolute constants then the above runtime simplifies to

\[
O_k \left( \left( \frac{1}{(1 - \epsilon')^2} \cdot \text{size}(C) \cdot \log \frac{1}{\delta} \right) \cdot \mathcal{M} \left( \log (|C|) (1, \ldots, 1), \log (\text{size}(C)) \right) \right).
\]

The restriction on \( \gamma \) is satisfied by any 1-TIDB (where \( \gamma = 0 \) in the equivalent 1-BIDB of Proposition 2.3) as well as for all three queries of the PDBench BIDB benchmark (Appendix C.12).

Next, by Proposition D.1 and Lemma D.2 for any \( \mathcal{R}\mathcal{A}^* \) query \( Q \), there exists a circuit \( C \) for \( \Phi(Q, D, t) \) such that \( \text{depth}(C') \leq O_Q(\log n) \) and \( \text{size}(C') \leq O_k(\text{depth}(Q, D, c)) \). Then, we note that Proposition 2.3 gives us an equivalent \( C \) with essentially the same size/depth and has \( \gamma \leq 1 - c^{-\omega(k)} \) (Lemma C.10). Finally, we argue (using the fact \( C \) has low depth) that \( |C| (1, \ldots, 1) \leq \text{size}(C) \cdot O_k(1) \) (Lemma C.11). The above sequence of arguments results in the following result (which answers Problem 1.5 in the affirmative):

**Corollary 4.8.** Let \( Q \) be an \( \mathcal{R}\mathcal{A}^* \) query and \( D \) be a c-TIDB with \( p_0 > 0 \), where \( p_0 \) as in Theorem 4.7, is an absolute constant. Let \( \Phi(X) = \Phi(Q, D, t) \) for any result tuple \( t \) with \( \text{deg}(\Phi) = k \). Then one can compute an approximation satisfying Eq. (5) in time

\[
O_k(\text{size}(C) \cdot \text{size}(C)) \cdot \mathcal{M} \left( \log (|C|) (1, \ldots, 1), \log (\text{size}(C)) \right).
\]

If we want to approximate the expected multiplicities of all \( Z = O(n^k) \) result tuples \( t \) simultaneously, we just need to run the above result with \( \delta \) replaced by \( \frac{1}{2} \), which increases the runtime by a factor of \( O_k(\log n) \).

## 5 RELATED WORK

**Probabilistic Databases (PDBs)** have been studied predominantly for set semantics. Approaches for probabilistic query processing (i.e., computing marginal probabilities of tuples), fall into two broad categories. Intensional (or grounded) query evaluation computes the lineage of a tuple and then the probability of the lineage formula. It has been shown that computing the marginal probability of a tuple is \#P-hard [50] (by reduction from weighted model counting). The second category, extensional query evaluation, is in \( \text{P}/\text{TIME} \), but is limited to certain classes of queries. Dalvi et al. [15] and Olteanu et al. [22] proved dichotomies for UQOs and two classes of queries with negation, respectively. Amarilli et al. investigated tractable classes of databases for more complex queries [2]. Another line of work studies which structural properties of lineage formulas lead to tractable cases [35, 45, 48]. In this paper we focus on intensional query evaluation with polynomials.

Many data models have been proposed for encoding PDBs more compactly than as sets of possible worlds. These include tuple-independent databases [51] (TIDBs), block-independent databases (BIDBs) [46], and \( \text{PC-tables} \) [27]. Fink et al. [20] study aggregate queries over a probabilistic version of the extension of K-relations for aggregate queries proposed in [3] (pre-tables) that supports bags, and has runtime complexity linear in the size of the lineage.
However, this lineage is encoded as a tree; the size (and thus the runtime) are still superlinear in $T_{det}(Q, D, c)$. The runtime bound is also limited to a specific class of (hierarchical) queries, suggesting the possibility of a generalization of [15]'s dichotomy result to bag-PDBs for our problem (22) presents a dichotomy result for a related problem.

Several techniques for approximating tuple probabilities have been proposed in related work [14, 16, 21, 41], relying on Monte Carlo sampling, e.g., [14], or a branch-and-bound paradigm [41]. Our approximation algorithm is also based on sampling.

**Compressed Encodings** are used for Boolean formulas (e.g., various types of circuits including OBDDs [32] and polynomials (e.g., factorizations [42]) some of which have been utilized for probabilistic query processing, e.g., [32]. Compact representations for which probabilities can be computed in linear time include OBDDs, SDDs, d-DNNF, and FBDD. [17] studies circuits for absorptive semirings while [49] studies circuits that include negation (expressed as the monus operation). Algebraic Decision Diagrams [6] (ADDs) generalize BDDs to variables with more than two values. Chen et al. [9] introduced the generalized disjunctive normal form. Appendix G covers more related work on fine-grained complexity.

**6 CONCLUSIONS AND FUTURE WORK**

We have studied the problem of calculating the expected multiplicity of a bag-query result tuple, a problem that has a practical application in probabilistic databases over multisets. We show that under various parameterized complexity hardness results/conjectures computing the expected multiplicities exactly is not possible in linear time in the corresponding deterministic query processing time. We prove that it is possible to approximate the expectation of a lineage polynomial in linear time in the deterministic query processing over TIDBs and BIDBs (assuming that there are few cancellations). Interesting directions for future work include development of a dichotomy for bag PDBs. While we can handle higher moments (this follows fairly easily from our existing results– see Appendix E), more general approximations are an interesting area for exploration, including those for more general data models.

**ACKNOWLEDGMENTS**

We thank Virginia Williams for showing us Eq. (22), which greatly simplified our earlier proof of Lemma 3.8, and for graciously allowing us to use it. This work is supported by NSF grants IIS-1956149, IIS-1750460, and IIS-2107107.

**REFERENCES**

[1] Parag Agrawal, Omar Benjelloun, Amish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. 2006. Trio: A System for Data, Uncertainty, and Lineage. In VLDB 1151–1154.

[2] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. 2015. Probabilities and provenance via tree decompositions. PODS (2015).

[3] Yael Amirav, Martin Grohe, and Dániel Marx. 2012. Dichotomies for Queries with Negation in Probabilistic Databases. TODS 41, 1 (2016), 41–447.

[4] Jörg Flum and Martin Grohe. 2002. The Parameterized Complexity of Counting Problems. In Proceedings of the 4th Symposium on Foundations of Computer Science (FOCS ’02). IEEE Computer Society. USA, 538.

[5] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. Database Systems - The Complete Book (2. ed.) Pearson Education.

[6] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abdelardo Pardo, and Fabio Somenzi. 1993. Algebraic Decision Diagrams and Their Applications. In IEEE CAD.

[7] H. Becker, Bab H. Iyyun, and Luizak Golab. 2010. Sampling the Repairs of Functional Dependency Violations under Hard Constraints. Proc. VLDB Endow. 3, 1 (2010), 197–207.

[8] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. 1997. Algebraic complexity theory. Vol. 315. Springer.

[9] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abdelardo Pardo, and Fabio Somenzi. 1993. Algebraic Decision Diagrams and Their Applications. In IEEE CAD.

[10] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. 2006. Strong computational lower bounds via parameterized complexity. J. Comput. System Sci. 72, 8 (2006), 1346–1367. https://doi.org/10.1016/j.jcss.2006.04.007

[11] Radu Curticeanu. 2013. Counting Matchings of Size k in W[1]-Hard. In ICAILP, Vol. 7965. 352–363.

[12] Radu Curticeanu and Dániel Marx. 2014. Complexity of Counting Subgraphs: Only the Boundedness of the Vertex-Cover Number Counts. In Proceedings of the 2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS ’14). IEEE Computer Society. USA, 130–139. https://doi.org/10.1109/FOCS.2014.22

[13] Nílsoles Dalvi and Dan Suciu. 2007. The Dichotomy of Conjunctive Queries on Probabilistic Structures. In PODS 293–302.

[14] N. Dalvi and D. Suciu. 2007. Efficient query evaluation on probabilistic databases.

[15] N. Dalvi and D. Suciu. 2007. Efficient query evaluation on probabilistic databases.

[16] Nílsoles Dalvi and Dan Suciu. 2012. The dichotomy of probabilistic inference for unions of conjunctive queries. JACM 59, 6 (2012), 30.

[17] Maarit Van den Heuwel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerke, and Martin Theobald. 2019. Anytime Approximation in Probabilistic Databases via Scaled Dissociations. In SIGMOD: 1295–1312.

[18] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. 2014. Circuits for Datalog Provenance. In ICDT: 201–212.

[19] Su Feng, Boris Glavic, Aaron Huber, and Oliver Kennedy. 2021. Efficient Uncertainty Tracking for Complex Queries with Attribute-level Bounds. In SIGMOD.

[20] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. 2019. Uncertainty Annotated Databases - A Lightweight Approach for Approximating Certain Answers. In SIGMOD.

[21] Robert Fink, Larisa Han, and Dan Olteanu. 2012. Aggregation in probabilistic databases via knowledge compilation. PVLDB 5, 5 (2012), 490–501.

[22] Robert Fink, Jiewen Huang, and Dan Olteanu. 2013. Anytime approximation in probabilistic databases. VLDB 16, 4 (2007), 544.

[23] Robert Fink and Dan Olteanu. 2016. Dichotomies for Queries with Negation in Probabilistic Databases. TODS 41, 1 (2016), 41–447.

[24] Jörg Flum and Martin Grohe. 2002. The Parameterized Complexity of Counting Problems. In Proceedings of the 4th Symposium on Foundations of Computer Science (FOCS ’02). IEEE Computer Society. USA, 538.

[25] Todd J. Green, Gregory Karpovits, and Val Tannen. 2007. Provenance semirings. In PODS 31–40.

[26] Todd J. Green, Gregory Karpovits, and Val Tannen. 2006. Models for incomplete and probabilistic information. In EDBT.: 278–296.

[27] Martin Grohe, Peter Lindner, and Christoph Stanké. 2022. Probabilistic Query Evaluation with Bag Semantics. https://doi.org/10.48550/ARXIV.220111524

[28] Paolo Guagliardo and Leonid Libkin. 2017. Correctness of SQL Queries on Databases with Nulls. SIGMOD Rec. 46, 3 (2017), 5–16.

[29] Tomasz Imieliński and Witold Lipski Jr. 1984. Incomplete Information in Relational Databases. JACM 31, 4 (1984), 761–791.

[30] Russell Impagliazzo, Ramanan Paturi, and Francis Zane. 2001. Which Problems Have Strongly Exponential Complexity? J. Comput. System Sci. 63, 4 (2001), 512–530. https://doi.org/10.1006/jcss.2001.1774

[31] Abhay Kumar Jha and Dan Suciu. 2012. Probabilistic Databases With Markovkries. PVLDB 5, 11 (2012), 1160–1171.

[32] Manas R. Joslekar, Rohan Puttagunta, and Christopher Ré. 2016. AJAR: Aggregations and joins over annotated relations. In Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 – July 01, 2016, Tova Milo and Wang-Chiew Tan (Eds.). ACM, 91–106. https://doi.org/10.1145/2902251.2902293

[33] Richard M. Karp, Michael Luby, and Neal Madras. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. J. Algorithms 10, 3 (1989), 429–448.

[34] Latastha Kenig, Arvindr Gal, and Ofir Strichman. 2013. A New Class of Linear Expressions over Probabilistic Databases Computable in F-Time. In SUM, Vol. 8078. 218–232.

[35] Mahmoud Abu Khamis, Huan Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In PODS 13–28.
[37] Tsvi Kopelowitz and Virginia Vassilevska Williams. 2020. Towards Optimal Set-Disjointness and Set-Intersection Data Structures. In ICALP, Vol. 168. 74:1–74:16.
[38] Poonam Kumari, Said Achniz, and Oliver Kennedy. 2016. Communicating Data Quality in On-Demand Curation. In QDB.
[39] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In PODS.
[40] Hung Q. Ngo, Christopher Ré, and Atri Rudra. 2013. Skew strikes back: new developments in the theory of join algorithms. SIGMOD Rec. 42, 4 (2013), 5–16.
[41] Dan Olteanu, Jiewen Huang, and Christoph Koch. 2010. Approximate confidence computation in probabilistic databases. In ICDE. 145–156.
[42] Dan Olteanu and Maximilian Schleich. 2015. Factorized Databases. SIGMOD Rec. 45, 2 (2016), 5–16.
[43] pdbench utility 2008. pdbench. http://pdbench.sourceforge.net/ Accessed: 2020-12-15.
[44] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. Holoclean: Holistic Data Repairs with Probabilistic Inference. Proc. VLDB Endow. 10, 11 (2017), 1190–1201.
[45] Sudeepa Roy, Vittorio Perduca, and Val Tannen. 2011. Faster query answering in probabilistic databases using read-once functions. In ICDT.
[46] C. Ré and D. Suciu. 2007. Materialized views in probabilistic databases: for information exchange and query optimization. In VLDB 51–62.
[47] Christopher De Sa, Alexander Rainer, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. 2017. Incremental knowledge base construction using DeepDive. VLDB J. 26, 1 (2017), 81–105.
[48] Prithviraj Sen, Amol Deshpande, and Lise Getoor. 2010. Read-Once Functions and Query Evaluation in Probabilistic Databases. PVLDB 3, 1 (2010), 1068–1079.
[49] Pierre Senellart. 2018. Provenance and Probabilities in Relational Databases. SIGMOD Record 46, 4 (2018), 5–15.
[50] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. SIAM J. Comput. 8, 3 (1979), 410–421.
[51] Guy Van den Broeck and Dan Suciu. 2017. Query Processing on Probabilistic Data: A Survey. (2017).
[52] Virginia Vassilevska Williams. 2018 Some Open Problems in Fine-Grained Complexity. SIGACT News 49, 4 (2018), 29–35. https://doi.org/10.1145/3300150.3300158
[53] Ying Yang, Niccolò Meneghetti, Ronny Fehling, Zhen Hua Liu, Dieter Gawlick, and Oliver Kennedy. 2015. Lenses: An On-Demand Approach to ETL. PVLDB 8, 12 (2015), 1578–1589.
A  MISSING DETAILS FROM SECTION 2

A.1  Polynomials

**Definition A.1 (Degree).** The degree of polynomial \( \phi(X) \) is the largest \( \sum d_i \) for all \( d_i \in \{0, \ldots, K\}^S \) such that \( c_{(d_1, \ldots, d_n)} \neq 0 \). We denote the degree of \( \phi \) as deg \( \phi \).

As an example, the degree of the polynomial \( X^2 + 2XY^2 + Y^2 \) is 3. Product terms in lineage arise only from join operations (Fig. 1), so intuitively, the degree of a lineage polynomial is analogous to the largest number of joins needed to produce a result tuple.

A.2  Background details for proof of Proposition A.5

A.2.1  \( \mathcal{K} \)-relations and \( \mathbb{N}[X] \)-encoded PDBs. We can use \( \mathcal{K} \)-relations to model bags. A \( \mathcal{K} \)-relation [26] is a relation whose tuples are annotated with variables from a commutative semiring \( \mathcal{K} = \{0, \mathcal{K}, \odot, \oplus, \ominus, \overline{1}_K\} \). A commutative semiring is a structure with a domain \( K \) and associative and commutative binary operations \( \odot, \oplus \) such that \( \odot K \) distributes over \( \odot K, \overline{1}_K \) is the identity of \( \odot K, K \) is the identity of \( \odot K, \overline{1}_K \) annihilates all elements of \( K \) when combined by \( \odot K \). Let \( \mathcal{U} \) be a countable domain of values. Formally, an n-ary \( \mathcal{K} \)-relation \( R \) over \( \mathcal{U} \) is a function \( R : \mathcal{U}^n \rightarrow K \) with finite support supp \( (R) = \{ t | R(t) \neq 0_K \} \). A \( \mathcal{K} \)-database is defined similarly, where we view the \( \mathcal{K} \)-database (relation) as a function mapping tuples to their respective annotations. \( \mathcal{R} \), \( \mathcal{R}^+ \) query semantics over \( \mathcal{K} \)-relations are analogous to the lineage construction semantics of Fig. 1, with the exception of replacing \( + \) with \( \oplus \) and \( \cdot \) with \( \odot \).

Consider the semiring \( \mathbb{N} = \{\mathbb{N}, +, \times, 0, 1\} \) of natural numbers. \( \mathbb{N} \)-databases model bag semantics by annotating each tuple with its multiplicity. A probabilistic \( \mathbb{N} \)-database (\( \mathbb{N} \)-PDB) is a PDB where each possible world is an \( \mathbb{N} \)-database. We study the problem of computing statistical moments for query results over such databases. Given an \( \mathbb{N} \)-PDB \( D = (\Omega, P, (\mathcal{R} \mathcal{A}^+)) \) query \( Q \), and possible result tuple \( t \), we sum \( Q(D)(t) \cdot P(D) \) for all \( D \in \Omega \) to compute the expected multiplicity of \( t \). Intuitively, the expectation of \( Q(D)(t) \) is the number of duplicates of \( t \) we expect to find in results of query \( Q \).

Let \( \mathbb{N}[X] \) denote the set of polynomials over variables \( X = (X_1, \ldots, X_n) \) with natural number coefficients and exponents. Consider now the semiring (abusing notation) \( \mathbb{N}[X] = \mathbb{N}[X, +, \times, 0, 1] \) whose domain is \( \mathbb{N}[X] \), with the standard addition and multiplication of polynomials. We define an \( \mathbb{N}[X] \)-encoded PDB \( \mathbb{D}_\mathbb{N}[X] \) as the tuple \( (\mathbb{D}_\mathbb{N}[X], P) \), where \( \mathbb{N}[X] \)-database \( \mathbb{D}_\mathbb{N}[X] \) is paired with the probability distribution \( P \) across the set of possible worlds represented by \( \mathbb{D}_\mathbb{N}[X] \), i.e. the one induced from \( P_\mathbb{N}[X] \), the probability distribution over \( X \). Note that the notation is slightly abused since the first element of the pair is an encoded set of possible worlds, i.e. \( \mathbb{D}_\mathbb{N}[X] \) is the deterministic bounding database. We denote by \( \phi(D_\mathbb{N}[X], t) \) the annotation of tuple \( t \) in the result of \( Q(D_\mathbb{N}[X])(t) \), and as before, interpret it as a function \( \phi(t) : (\mathbb{N}[X], t) \rightarrow \mathbb{N} \) from vectors of variable assignments to the corresponding value of the annotating polynomial. \( \mathbb{N}[X] \)-encoded PDBs and a function \( \text{Mod} \) (which transforms an \( \mathbb{N}[X] \)-encoded PDB to an equivalent \( \mathbb{N} \)-PDB) are both formalized next.

To justify the use of \( \mathbb{N}[X] \)-databases, we need to show that we can encode any \( \mathbb{N} \)-PDB in this way and that the query semantics over this representation coincides with query semantics over its respective \( \mathbb{N} \)-PDB. For that it will be opportune to define representation systems for \( \mathbb{N} \)-PDBs.

**Definition A.2 (Representation System).** A representation system for \( \mathbb{N} \)-PDBs is a triple \( (M, \text{Mod}, \text{Mod}) \) where \( M \) is a set of representations and \( \text{Mod} \) associates with each \( M \in M \) an \( \mathbb{N} \)-PDB \( D \). We say that a representation system is closed under a class of queries \( Q \) if for any query \( Q \in Q \) and \( M \in M \) we have:

\[
\text{Mod}(Q(M)) = Q(\text{Mod}(M))
\]

A representation system is complete if for every \( \mathbb{N} \)-PDB \( D \) there exists \( M \in M \) such that:

\[
\text{Mod}(M) = D
\]

As mentioned above we will use \( \mathbb{N}[X] \)-databases paired with a probability distribution as a representation system, referring to such databases as \( \mathbb{N}[X] \)-encoded PDBs. Given \( \mathbb{N}[X] \)-encoded PDB \( \mathbb{D}_\mathbb{N}[X] \), one can think of the \( D_\mathbb{N}[X] \) as the probability distribution across all worlds \( \{0, 1\}^n \). Denote a particular world to be \( W \). For convenience let \( \psi_W : \mathbb{D}_\mathbb{N}[X] \rightarrow D_\mathbb{N}[X] \) be a function that computes the corresponding \( \mathbb{N} \)-PDB upon assigning all values \( W_i \in W \) to \( X_i \in X \) of \( D_\mathbb{N}[X] \). Note the one-to-one correspondence between elements \( W \in \{0, 1\}^n \) to the worlds encoded by \( D_\mathbb{N}[X] \) when \( W \) is assigned to \( X \) (assuming a domain of \( \{0, 1\} \) for each \( X_i \)). We can think of \( \psi_W(D_\mathbb{N}[X]) (t) \) as the semiring homomorphism \( \mathbb{N}[X] \rightarrow \mathbb{N} \) that applies the assignment \( W \) to all variables \( X \) of a polynomial and evaluates the resulting expression in \( \mathbb{N} \).

**Definition A.3 (Mod \( \mathcal{D}_\mathbb{N}[X] \)).** Given an \( \mathbb{N}[X] \)-encoded PDB \( \mathbb{D}_\mathbb{N}[X] \), we compute its equivalent \( \mathbb{N} \)-PDB \( \mathbb{D}_\mathbb{N} = \text{Mod}(\mathcal{D}_\mathbb{N}[X]) \in (\Omega, P') \) as:

\[
\mathcal{D}_\mathbb{N}[X] = \{ \psi_W(D_\mathbb{N}[X]) | W \in \{0, 1\}^n \}
\]

\[
\forall D \in \Omega : Pr(D) = \sum_{W \in \{0, 1\}^n : \psi_W(D_\mathbb{N}[X]) = D} Pr(W)
\]

For instance, consider a \( D_\mathbb{N}[X] \) consisting of a single tuple \( t_1 = (1) \) annotated with \( X_1 + X_2 \) with probability distribution \( Pr(\{0, 1\}) = 0, Pr(\{0, 1\}) = 0, Pr(\{1, 0\}) = 0.3 \) and \( Pr(\{1, 1\}) = 0.7 \). This \( \mathbb{N}[X] \)-encoded PDB encodes two possible worlds (with non-zero probability).
that we denote using their world vectors.

\[ D_{[0,1]}(t_1) = 1 \quad \text{and} \quad D_{[1,1]}(t_1) = 2 \]

Importantly, as the following proposition shows, any finite \( \mathbb{N} \)-PDB can be encoded as an \( \mathbb{N}[X] \)-encoded PDB and \( \mathbb{N}[X] \)-encoded PDBs are closed under \( \mathcal{RA}^+ \)[26].

**Proposition A.4.** \( \mathbb{N}[X] \)-encoded PDBs are a complete representation system for \( \mathbb{N} \)-PDBs that is closed under \( \mathcal{RA}^+ \) queries.

**Proof.** To prove that \( \mathbb{N}[X] \)-encoded PDBs are complete consider the following construction that for any \( \mathbb{N} \)-PDB \( D = (\Omega, \mathcal{P}) \) produces an \( \mathbb{N}[X] \)-encoded PDB \( D_{\mathbb{N}[X]} = (D_{\mathbb{N}[X]}, \mathcal{P}') \) such that \( \text{Mod}(D_{\mathbb{N}[X]}) = D \). Let \( \Omega = \{D_1, \ldots, D_{|\Omega|} \} \). For each world \( D_i \) we create a corresponding variable \( X_i \). In \( D_{\mathbb{N}[X]} \) we assign each tuple \( t \) the polynomial:

\[ D_{\mathbb{N}[X]}(t) = \sum_{i=1}^{|\Omega|} D_i(t) \cdot X_i \]

The probability distribution \( \mathcal{P}' \) assigns all world vectors zero probability except for \( |\Omega| \) world vectors (representing the possible worlds) \( W_i \). All elements of \( W_i \) are zero except for the position corresponding to variables \( X_i \) which is set to 1. Unfolding definitions it is trivial to show that \( \text{Mod}(D_{\mathbb{N}[X]}) = D \). Thus, \( \mathbb{N}[X] \)-encoded PDBs are complete representation system.

Since \( \mathbb{N}[X] \) is the free object in the variety of semirings, Birkhoff’s HSP theorem implies that any assignment \( X \rightarrow \mathbb{N} \), which includes as a special case the assignments \( \psi_W \) used here, uniquely extends to the semiring homomorphism alluded to above, \( \psi_W(D_{\mathbb{N}[X]}) \). For a polynomial \( \psi_W(D_{\mathbb{N}[X]}) \) substitutes variables based on \( W \) and then evaluates the resulting expression in \( \mathbb{N} \). For instance, consider the polynomial \( D_{\mathbb{N}[X]}(t) = \Phi = X \cdot Y + Y \) and assignment \( W : X = 0, Y = 1 \). We get \( \psi_W(D_{\mathbb{N}[X]}(t)) = 0 + 1 = 1 \).

\[ \mathcal{P} \text{ closure under } \mathcal{RA}^+ \text{ queries follows from this and from [26]’s Proposition 3.5, which states that semiring homomorphisms commute with queries over } \mathcal{K}\text{-relations.} \]

\[ \square \]

A.2.2 TIDBs and BIDBs in the \( \mathbb{N}[X] \)-encoded PDB model. Two important subclasses of \( \mathbb{N}[X] \)-encoded PDBs that are of interest to us are the bag versions of tuple-independent databases (TIDBs) and block-independent databases (BIDBs). Under set semantics, a TIDB is a deterministic database \( D \) where each tuple \( t \) is assigned a probability \( p_t \). The set of possible worlds represented by a TIDB \( D \) is all subsets of \( D \). The probability of each world is the product of the probabilities of all tuples that exist with one minus the probability of all tuples of \( D \) that are not part of the world, i.e., tuples are treated as independent random events. In a BIDB, we also assign each tuple a probability, but additionally partition \( D \) into blocks. The possible worlds of a BIDB \( D \) are all subsets of the probability that contain at most one tuple from each block. Note then that the tuples sharing the same block are disjoint, and the sum of the probabilities of all the tuples in the same block \( B \) is at most 1. The probability of such a world is the product of the probabilities of all tuples present in the world and the product of the probabilities that no tuple is present in each block \( B \) for which no tuple exists in that world.

In this work, we define TIDBs and BIDBs as subclasses of \( \mathbb{N}[X] \)-encoded PDBs defined over variables \( X \) (Definition A.3) where \( X \) can be partitioned into blocks that satisfy the conditions of a TIDB or BIDB (stated formally in Section 2.2). In this work, we considered one further deviation from the standard: We use bag semantics for queries. Even though tuples cannot occur more than once in the input TIDB or BIDB, they can occur with a multiplicity larger than one in the result of a query. Since in TIDBs and BIDBs, there is a one-to-one correspondence between tuples in the database and variables, we can interpret a vector \( W \in \{0,1\}^m \) as denoting which tuples exist in the possible world \( \psi_W(D_{\mathbb{N}[X]}) \) (the ones where \( W_i = 1 \)). For BIDBs specifically, note that at most one of the bits corresponding to tuples in each of the \( m \) blocks will be set (i.e., for any pair of bits \( W_j, W_j' \) that are part of the same block \( B_i \), \( j \neq j' \), at most one of them will be set). Denote the vector \( p \) to be a vector whose elements are the individual probabilities \( p_t \) of each tuple \( t_i \). Given PDB \( D \) where \( \mathcal{P} \) is the distribution induced by \( p \), which we will denote \( \mathcal{P}^{(p)} \).

\[ \mathbb{E}_{W \sim \mathcal{P}^{(p)}}[\Phi(W)] = \sum_{W \in \{0,1\}^n, s.t. W_j, W_j' \rightarrow \mathbb{B}_{i=1} \geq (t_i, t_j)} \Phi(W) \prod_{j \in \{m\}} p_j \prod_{i \in \{m\}, s.t. \forall j \neq j, W_j = 0} (1 - \sum p_j) \]

Recall that tuple blocks in a TIDB always have size 1 so, the outer summation of eq. (7) is over the full set of vectors.

**A.3 Proof of Proposition 2.3.** We first need to prove that any \( c \)-TIDB \( D \) can be reduced to the Binary-BIDB created by Proposition 2.3. By definition, any \( t \in D \) can be present \( c' \in [c] \) times in the possible worlds it appears in, with a disjoint probability distribution across the multiplicities \( c' \). Then the construction of Proposition 2.3 of the block of tuples \( \{(t, j, j) \mid j \in [c]\} \) for each \( t \in D \) indeed encodes the disjoint behavior across multiplicities. \( D \) further requires that all \( t \in D \) are independent, a property which is enforced by independence constraint across all blocks of tuples in Definition 2.2. Then the construction of \( D' \) in Proposition 2.3 is an equivalent representation of \( D \).
Next we need to show that the distributions over \( \mathcal{D} \) and \( \mathcal{D}' \) are equivalent. The distribution \( \mathcal{P} \) is a distribution disjoint across the set of multiplicities \([c]\) and independent across all \( t \in \mathcal{D} \). By definition, Proposition 2.3 creates \( \mathcal{D}' \) by producing a block of disjoint tuples \( t_j = (t, j) \) for each \( t \in \mathcal{D} \) and \( j \in [c] \), where \( \mathbf{p} = \{p_{t,j} \mid t \in \mathcal{D}, j \in [c]\} \). Since the probability vector \( \mathbf{p}_D = \{p_{t,j} \mid t \in \mathcal{D}, j \in [c]\} \) and each \( p_{t,j} \) for \( j \neq j' \in [c] \) are disjoint, the distributions are hence the same.

### A.4 Equating Expectations of Tuple Multiplicities and Polynomials

**Proposition A.5 (Expectation of polynomials).** Given a bag-FDB \( \mathcal{D} = (\Omega, \mathcal{P}) \), \( \mathcal{RA}^* \) query \( Q \), and lineage polynomial \( \Phi[Q, D \in \mathcal{D}] \) for arbitrary result tuple \( t \), we have (denoting \( D \) as the random variable over \( \Omega \)): \( \mathbb{E}_{D \sim \mathcal{P}}[Q(D)] = \mathbb{E}_{W \sim \mathcal{P}}[\Phi[Q, D \in \mathcal{D}, t](W)] \).

Although Proposition A.5 follows, e.g., as an obvious consequence of [30]'s Theorem 7.1, we are unaware of any prior formal proof for bag-probabilistic databases.

**Proof.** We need to prove for \( \mathbb{N} \)-FDB \( \mathcal{D} = (\Omega, \mathcal{P}) \) and \( \mathbb{N}[X] \)-encoded PDB \( \mathcal{D}_{\mathbb{N}[X]} = (\mathcal{D}_{\mathbb{N}[X]}, \mathcal{P}') \) where \( \text{Mod}(\mathcal{D}_{\mathbb{N}[X]}) = \mathcal{D} \) that \( \mathbb{E}_{D \sim \mathcal{P}}[Q(D)] = \mathbb{E}_{W \sim \mathcal{P}'}[\Phi[Q, \mathcal{D}_{\mathbb{N}[X]}, t](W)] \). By expanding \( \Phi[Q, \mathcal{D}_{\mathbb{N}[X]}, t] \) and the expectation we have:

\[
\mathbb{E}_{W \sim \mathcal{P}'}[\Phi(W)] = \sum_{W \in \{0,1\}^n} \mathbb{P}(W) \cdot Q(\mathcal{D}_{\mathbb{N}[X]})(t)(W)
\]

From \( \text{Mod}(\mathcal{D}_{\mathbb{N}[X]}) = \mathcal{D} \), we have that the range of \( \Phi_W(\mathcal{D}_{\mathbb{N}[X]}) \) is \( \Omega \), so

\[
\mathbb{E}_{W \sim \mathcal{P}'}[\Phi(W)] = \sum_{D \in \Omega} \sum_{W \in \{0,1\}^n : \Phi_W(\mathcal{D}_{\mathbb{N}[X]})(t) = D} \mathbb{P}(W) \cdot Q(D)(t)(W)
\]

The inner sum is only over \( W \) where \( \Phi_W(\mathcal{D}_{\mathbb{N}[X]}) = D \) (i.e., \( Q(\mathcal{D}_{\mathbb{N}[X]})(t)(W) = Q(D)(t) \))

\[
= \sum_{D \in \Omega} \sum_{W \in \{0,1\}^n : \Phi_W(\mathcal{D}_{\mathbb{N}[X]})(t) = D} \mathbb{P}(W) \cdot Q(D)(t)
\]

By distributivity of + over \( \times \)

\[
= \sum_{D \in \Omega} Q(D)(t) \cdot \sum_{W \in \{0,1\}^n : \Phi_W(\mathcal{D}_{\mathbb{N}[X]})(t) = D} \mathbb{P}(W)
\]

From the definition of \( \mathcal{P} \) in Definition A.3, given \( \text{Mod}(\mathcal{D}_{\mathbb{N}[X]}) = \mathcal{D} \), we get

\[
= \sum_{D \in \Omega} Q(D)(t) \cdot Pr(D) = \mathbb{E}_{D \sim \mathcal{P}}[Q(D)](t).
\]

\( \square \)

### A.5 Construction of the lineage (polynomial) for an \( \mathcal{RA}^* \) query \( Q \) over \( \overline{\mathcal{D}} \)

The following rules are analogous to Fig. 1, differing only in the base case. They are presented here for completeness.

\[
\Phi'[\pi_A(Q), \overline{\mathcal{D}'}, t_j] = \sum_{\pi_A(t_j) = t_j} \Phi'[Q, \overline{\mathcal{D}'}, t_j]
\]

\[
\Phi'[Q_1 \cup Q_2, \overline{\mathcal{D}'}, t_j] = \Phi'[Q_1, \overline{\mathcal{D}'}, t_j] + \Phi'[Q_2, \overline{\mathcal{D}'}, t_j]
\]

\[
\Phi'[\sigma_{\theta}(Q), \overline{\mathcal{D}'}, t_j] = \begin{cases} \Phi'[Q, \overline{\mathcal{D}'}, t_j] & \theta = 1 \\ 0 & \theta = 0 \end{cases}
\]

\[
\Phi'[Q_1 \bowtie Q_2, \overline{\mathcal{D}'}, t_j] = \Phi'[Q_1, \overline{\mathcal{D}'}, \pi_{\text{attr}(Q_1)}(t_j)] \cdot \Phi'[Q_2, \overline{\mathcal{D}'}, \pi_{\text{attr}(Q_2)}(t_j)]
\]

\[
\Phi'[R, \overline{\mathcal{D}'}, t_j] = \pi_{X_{t,j}}
\]
A.6 Proposition A.6

Note the following fact:

**Proposition A.6.** For any BIDB-lineage polynomial $\Phi(X_1, \ldots, X_n)$ and all $W$ such that $P_{\tau}[W] > 0$, it holds that $\Phi(W) = \bar{\Phi}(W)$.

**Proof.** Note that any $\Phi$ in factorized form is equivalent to its SMB expansion. For each term in the expanded form, further note that for all $b \in \{0, 1\}$ and all $\varepsilon \geq 1$, $b^\varepsilon = b$. By definition (see Definition 2.4), $\Phi(X)$ is the SMB expansion of $\Phi(X)$ followed by reducing every exponent $\varepsilon > 1$ to 1 and eliminating all cross terms for the BIDB case. Note that it must be that no cross terms exist in $\Phi(X)$, since by the proposition statement, $P_{\tau}[W] > 0$. Thus, since all monomials are indeed the same, it follows that $\Phi(W) = \bar{\Phi}(W)$. □

A.7 Proof for Lemma 2.5

**Proof.** Let $\Phi$ be a polynomial of $n$ variables with highest degree $= K$, defined as follows:

$$
\Phi(X_1, \ldots, X_n) = \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{i \in D'} X_{i,d},
$$

where $D'$ has $n$ tuples, we can equivalently write $\prod_{t \in D'} X_{t,d}$ for the product term.

Let the boolean function $\text{isInd}()$ take $d$ as input and return true if there does not exist any dependent variables in the monomial encoded by $d$, i.e., for any block $B \in D'$, $t \in B | d_l, d_r \geq 1$.

Then, given Binary-BIDB $D$, query $Q$, and polynomial $\Phi(W) = \Phi(Q, D, t)$, in expectation we have

$$
\mathbb{E}_{W}[\Phi(W)] = \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right] + \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \mathbb{E}_{W}\left[ \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \mathbb{E}_{W}\left[ \sum_{d \in \{(0, \ldots, K)^n \cap \lnot \text{isInd}(d)} c_d \cdot \prod_{t \in D'} W_{t,d} \right]
$$

$$
= \Phi(p_1, \ldots, p_n).
$$

Eq. (8) is the result of substituting in the definition of $\Phi$ given above. Then we arrive at eq. (9) by linearity of expectation. Next, eq. (10) is the result of the independence constraint of BIDBs, specifically that any monomial composed of dependent variables, i.e., variables from the same block $B$, has a probability of $0$. Eq. (11) is obtained by the fact that all variables in each monomial are independent, which allows for the expectation to be pushed through the product. In eq. (12), recall the lineage construction semantics of Appendix A.5 for a Binary-BIDB, where the annotation of a tuple $t$ with multiplicity $j$ is written as $j^{\delta_{t}} \cdot X_{t,d}$ such that $\text{Dom}(W_{t}) = \{0, 1\}$. Then $c_d$ absorbs all such $j^{\delta_{t}}$ factors. Since $W_{t} \in \{0, 1\}$ it is the case that for any exponent $\varepsilon \geq 1$, $W_{t}^\varepsilon = W_{t}$. Next, in eq. (13) the expectation of a tuple is indeed its probability.

Finally, it can be verified that Eq. (14) follows since eq. (13) satisfies the construction of $\bar{\Phi}(p_1, \ldots, p_n)$ in Definition 1.3. □

A.8 Proof For Corollary 2.6

**Proof.** Note that Lemma 1.4 shows that $\mathbb{E}[\Phi] = \bar{\Phi}(p_1, \ldots, p_n)$. Therefore, if $\Phi$ is already in SMB form, one only needs to compute $\Phi(p_1, \ldots, p_n)$ ignoring exponent terms (note that such a polynomial is $\bar{\Phi}(p_1, \ldots, p_n)$), which indeed has $O(|\Phi|)$ computations. □
A.9 Definition of POLY(⋅)
We formally define the relationship of circuits with polynomials. While the definition assumes one sink for notational convenience, it easily generalizes to the multiple sinks case.

POLY(C) maps the sink of circuit C to its corresponding polynomial in SMB. POLY(⋅) is recursively defined on C as follows, with addition and multiplication following the standard interpretation for polynomials:

\[
\text{POLY}(C) = \begin{cases} 
\text{POLY}(C_L) + \text{POLY}(C_R) & \text{if } C\text{.type }= + \\
\text{POLY}(C_L) \cdot \text{POLY}(C_R) & \text{if } C\text{.type }= \times \\
\text{C.val} & \text{if } C\text{.type }= \text{VAR OR NUM.}
\end{cases}
\]

C need not encode \(\Phi(X)\) in the same, default SMB representation. For instance, C could encode the factorized representation \((X + 2Y)(2X - Y)\) of \(\Phi(X) = 2X^2 + 3XY - 2Y^2\), as shown in Fig. 3, while \(\text{POLY}(C) = \Phi(X)\) is always the equivalent SMB representation.

B MISCELLANEOUS DETAILS FROM SECTION 3

B.1 Lemma B.1

Lemma B.1. Assuming that each \(v \in V\) has degree \(\geq 1\), the PDB relations encoding the edges for \(\Phi_G^k\) of Definition 3.4 can be computed in \(O(m)\) time.

Proof of Lemma B.1. Only two relations need be constructed, one for the set \(V\) and one for the set \(E\). By a simple linear scan, each can be constructed in time \(O(m + n)\). Given that the degree of each \(v \in V\) is at least 1, we have that \(m \geq \Omega(n)\), and this yields the claimed runtime. \(\square\)

B.2 Proof of Lemma B.3

Proof. By the recursive definition of \(T_{det}(\cdot, \cdot)\) (see Sec. 2.4), we have the following equation for our hard query \(Q\) when \(k = 1\), (we denote this as \(Q^1\)).

\[
T_{det}\left(Q^1, D\right) = |D.V| + |D.E| + |D.| + T_{join}(D.V, D.E, D.V).
\]

We argue that \(T_{join}(D.V, D.E, D.V)\) is at most \(O(m)\) by noting that there exists an algorithm that computes \(D.V \bowtie D.E \bowtie D.V\) in the same runtime. Then by the assumption of Lemma B.1, the sum of the first three terms is \(O(m)\). We then obtain that \(T_{det}\left(Q^1, D\right) = O(m) + O(m) = O(m)\). For \(Q^k = Q^1 \times \cdots \times Q^k\), we have the recurrence \(T_{det}\left(Q^k, D\right) = T_{det}\left(Q^1, D\right) + \cdots + T_{det}\left(Q^k, D\right) + T_{join}(Q^1 \times \cdots \times Q^k)\). Since \(Q^1\) outputs a count, computing the join \(Q^1 \times \cdots \times Q^k\) is just multiplying \(k\) numbers, which takes \(O(k)\) time. Thus, we have

\[
T_{det}\left(Q^k, D\right) \leq k \cdot O(m) + O(k) \leq O(km),
\]

as desired. \(\square\)

B.3 Lemma B.2

The following lemma reduces the problem of counting \(k\)-matchings in a graph to our problem (and proves Theorem 3.6):

Lemma B.2. Let \(p_0, \ldots, p_{2k}\) be distinct values in \((0, 1]\). Then given the values \(\hat{\Phi}_G^k(p_0, \ldots, p_{2k})\) for \(0 \leq i \leq 2k\), the number of \(k\)-matchings in \(G\) can be computed in \(O(k^3)\) time.

B.4 Proof of Lemma B.2

Proof. We first argue that \(\hat{\Phi}_G^k(p, \ldots, p) = \sum^k_{i=0} c_i \cdot p^i\). First, since \(\Phi_G(X)\) has degree 2, it follows that \(\Phi_G^k(X)\) has degree 2k. By definition, \(\hat{\Phi}_G^k(X)\) sets every exponent \(e > 1\) to \(e = 1\), which means that \(\deg(\hat{\Phi}_G^k) \leq \deg(\Phi_G^k) = 2k\). Thus, if we think of \(p\) as a variable, then \(\hat{\Phi}_G^k(p, \ldots, p)\) is a univariate polynomial of degree at most \(\deg(\hat{\Phi}_G^k) \leq 2k\). Thus, we can write

\[
\hat{\Phi}_G^k(p, \ldots, p) = \sum^k_{i=0} c_i p^i
\]

We note that \(c_i\) is exactly the number of monomials in the SMB expansion of \(\hat{\Phi}_G^k(X)\) composed of \(i\) distinct variables.\(^{11}\)

\(^{11}\)This is WLOG, since any vertex with degree 0 can be dropped without affecting the result of our hard query.

\(^{10}\)Indeed the trivial algorithm that computes the obvious pair-wise joins has the claimed runtime. That is, we first compute \(D.V \bowtie D.E\), which takes \(O(m)\) (assuming \(D.V\) is stored in hash map) since tuples in \(D.V\) can only filter tuples in \(D.E\). The resulting subset of tuples in \(D.E\) are then again joined (on the right) with \(D.V\), which by the same argument as before also takes \(O(m)\) time, as desired.

\(^{11}\)Since \(\Phi_G^k(X)\) does not have any monomial with degree < 2, it is the case that \(c_0 = c_1 = 0\) but for the sake of simplicity we will ignore this observation.
Given that we then have $2k + 1$ distinct values of $\tilde{\Phi}_G^k(p_i, \ldots, p_i)$ for $0 \leq i \leq 2k$, it follows that we have a linear system of the form $M \cdot c = b$ where the $i$th row of $M$ is $\left( p_i^0, \ldots, p_i^{2k} \right)$. $c$ is the coefficient vector $(c_0, \ldots, c_{2k})$, and $b$ is the vector such that $b[i] = \tilde{\Phi}_G^k(p_i, \ldots, p_i)$. In other words, matrix $M$ is the Vandermonde matrix, from which it follows that we have a matrix with full rank (the $p_i$’s are distinct), and we can solve the linear system in $O(k^3)$ time (e.g., using Gaussian Elimination) to determine $c$ exactly. Thus, after $O(k^3)$ work, we know $c$ and in particular, $c_{2k}$ exactly.

Next, we show why we can compute $\# \left(G, I \cdots I^k\right)$ from $c_{2k}$ in $O(1)$ additional time. We claim that $c_{2k}$ is $k! \cdot \# \left(G, I \cdots I^k\right)$. This can be seen intuitively by looking at the expansion of the original factorized representation

$$\Phi_G^k(X) = \sum_{(i_1,j_1), \ldots, (i_k,j_k) \in E} X_{i_1}X_{j_1} \cdots X_{i_k}X_{j_k},$$

where a unique $k$-matching in the multi-set of product terms has $k$ distinct $(i_l,j_l)$ index pairs. Further, any monomial composed of such a distinct set of $2k$ variables will be produced $k!$ times in a $k$-wise product of the sum of a set of elements with itself. This is true because each (identical) product term contains each of the $k$ distinct elements, giving us $k \cdot k - 1 \cdot 1 = k!$ permutations of a distinct $k$-matching.

Since, as noted earlier, $c_{2k}$ represents the number of monomials with $2k$ distinct variables, then it must be that $c_{2k}$ is the overall number of $k$-matchings. And since we have $k!$ copies of each distinct $k$-matching, it follows that $c_{2k} = k! \cdot \# \left(G, I \cdots I^k\right)$. Thus, simply dividing $c_{2k}$ by $k!$ gives us $\# \left(G, I \cdots I^k\right)$, as needed.

**Proof of Theorem 3.6**

Proof. For the sake of contradiction, assume we can solve our problem in $o\left(T_{\text{match}}(k, G)\right)$ time. Given a graph $G$ by Lemma B.1 we can compute the PDB encoding in $O(m)$ time. Then after we run our algorithm on $\tilde{\Phi}_G^k$, we get $\tilde{\Phi}_G^k(p_i, \ldots, p_i)$ for every $0 \leq i \leq 2k$ in additional $O(k) \cdot o\left(T_{\text{match}}(k, G)\right)$ time. Lemma B.2 then computes the number of $k$-matchings in $G$ in $O(k^3)$ time. Adding the runtime of all of these steps, we have an algorithm for computing the number of $k$-matchings that runs in time

$$O(m) + O(k) \cdot o\left(T_{\text{match}}(k, G)\right) + O(k^3)$$

$$\leq o\left(T_{\text{match}}(k, G)\right).$$

We obtain Eq. (16) from the facts that $k$ is fixed (related to $m$) and the assumption that $T_{\text{match}}(k, G) \geq o(m)$. Thus we obtain the contradiction that we can achieve a runtime $o\left(T_{\text{match}}(k, G)\right)$ that is better than the optimal time $T_{\text{match}}(k, G)$ required to compute $k$-matchings.

**B.6 Subgraph Notation and $O(1)$ Closed Formulas**

We need all the possible edge patterns in an arbitrary $G$ with at most three distinct edges. We have already seen $\Delta, \Lambda, \Xi$ and $\Xi\Xi$, so we define the remaining patterns:

- Single Edge ($\{\}$)
- 2-path ($\langle\rangle$)
- 2-matching ($\bigotimes$)
- 3-star ($\langle\rangle\langle\rangle$)—this is the graph that results when all three edges share exactly one common endpoint. The remaining endpoint for each edge is disconnected from any endpoint of the remaining two edges.
- Disjoint Two-Path ($\langle\rangle\langle\rangle$)—this subgraph consists of a two-path and a remaining disjoint edge.

For any graph $G$, the following formulas for $\# \left(G, H\right)$ compute their respective patterns exactly in $O(m)$ time, with $d_i$ representing the degree of vertex $i$ (proofs are in Appendix B.7):

$$\# \left(G, I\right) = m,$$

$$\# \left(G, \langle\rangle\right) = \sum_{i \in V} \binom{d_i}{2}$$

$$\# \left(G, \bigotimes\right) = \sum_{(i,j) \in E} \frac{m - d_i - d_j + 1}{2}$$

$$\# \left(G, \langle\rangle\langle\rangle\right) = \sum_{i \in V} \binom{d_i}{3}$$

$$\# \left(G, \langle\rangle\langle\rangle\right) + 3\# \left(G, \bigotimes\bigotimes\right) = \sum_{(i,j) \in E} \binom{m - d_i - d_j + 1}{2}$$

$$\# \left(G, \bigotimes\bigotimes\right) + 3\# \left(G, \langle\rangle\langle\rangle\langle\rangle\right) = \sum_{(i,j) \in E} (d_i - 1) \cdot (d_j - 1)$$
B.7 Proofs of Eq. (17)-Eq. (22)

The proofs for Eq. (17), Eq. (18) and Eq. (20) are immediate.

Proof of Eq. (19). For edge \((i, j)\) connecting arbitrary vertices \(i\) and \(j\), finding all other edges in \(G\) disjoint to \((i, j)\) is equivalent to finding all edges that are not connected to either vertex \(i\) or \(j\). The number of such edges is \(m - d_i - d_j + 1\), where we add 1 since edge \((i, j)\) is removed twice when subtracting both \(d_i\) and \(d_j\). Since the summation is iterating over all edges such that a pair \(((i, j), (k, \ell))\) will also be counted as \(((k, \ell), (i, j))\), division by 2 then eliminates this double counting. Note that \(m\) and \(d_i\) for all \(i \in V\) can be computed in one pass over the set of edges by simply maintaining counts for each quantity. Finally, the summation is also one traversal through the set of edges where each operation is either a lookup (\(O(1)\) time) or an addition operation (also \(O(1)\) time).

Proof of Eq. (21). Eq. (21) is true for similar reasons. For edge \((i, j)\), it is necessary to find two additional edges, disjoint or connected. As in our argument for Eq. (19), once the number of edges disjoint to \((i, j)\) have been computed, then we only need to consider all possible combinations of two edges from the set of disjoint edges, since it doesn’t matter if the two edges are connected or not. Note, the factor 3 of \(\Phi_i\) is necessary to account for the triple counting of 3-matchings, since it is indistinguishable to the closed form expression which of the remaining edges are either disjoint or connected to each of the edges in the initial set of edges disjoint to the edge under consideration. Observe that the disjoint case will be counted 3 times since each edge of a 3-path is visited once, and the same 3-path counted in each visitation. For the latter case however, it is true that since the two path in \(I_\Lambda\) is connected, there will be no multiple counting by the fact that the summation automatically disconnects the current edge, meaning that a two matching at the current vertex under consideration will not be counted. Thus, \(I_\Lambda\) will only be counted once, precisely when the single disjoint edge is visited in the summation. The sum over all such edge combinations is precisely then \(#(G, I_\Lambda) + 3\#(G, II)\). Note that all factorials can be computed in \(O(m)\) time, and then the remaining operations in computing each combination \(\binom{n}{3}\) are constant, thus yielding the claimed \(O(m)\) run time.

Proof of Eq. (22). To compute \(#(G, II)\), note that for an arbitrary edge \((i, j)\), a 3-path exists for edge pair \((i, \ell)\) and \((j, k)\) where \(i, j, \ell, k\) are distinct. Further, the quantity \((d_i - 1) \cdot (d_j - 1)\) represents the number of 3-edge subgraphs with middle edge \((i, j)\) and outer edges \((i, \ell)\) and \((j, k)\) such that \(\ell \neq j\) and \(k \neq i\). When \(k = \ell\), the resulting subgraph is a triangle, and when \(k \neq \ell\), the subgraph is a 3-path. Summing over all edges \((i, j)\) gives Eq. (22) by observing that each triangle is counted thrice, while each 3-path is counted just once. For reasons similar to Eq. (19), all \(d_i\) can be computed in \(O(m)\) time and each summand can then be computed in \(O(1)\) time, yielding an overall \(O(m)\) run time.

B.8 Tools to prove Theorem 3.7

Note that \(\Phi_3^i(p, \ldots, p)\) as a polynomial in \(p\) has degree at most six. Lemma 3.8 shows the exact coefficients of \(\Phi_3^i(p, \ldots, p)\), which we prove next.

B.8.1 Proof for Lemma 3.8.

Proof. By definition we have that

\[
\Phi_3^i(x) = \sum_{(i_1, j_1), (i_2, j_2), (i_3, j_3) \in E} \prod_{\ell = 1}^{3} x_{i_{\ell}} x_{j_{\ell}}.
\]

Hence \(\Phi_3^i(x)\) has degree six. Note that the monomial \(\prod_{\ell=1}^{3} x_{i_{\ell}} x_{j_{\ell}}\) will contribute to the coefficient of \(p^{ev}r^n\) in \(\Phi_3^i_G(X)\), where \(n\) is the number of distinct variables in the monomial. Let \(e_1 = (i_1, j_1), e_2 = (i_2, j_2),\) and \(e_3 = (i_3, j_3)\). We compute \(\Phi_3^i_G(x)\) by considering each of the three forms that the triple \((e_1, e_2, e_3)\) can take.

Case 1: \(e_1 = e_2 = e_3\) (all edges are the same). When we have that \(e_1 = e_2 = e_3\), then the monomial corresponds to \(#(G, \lambda)\). There are exactly \(m\) such triples, each with a \(p^2\) factor in \(\Phi_3^i_G(p, \ldots, p)\).

Case 2: This case occurs when there are two distinct edges of the three, call them \(e\) and \(e'\). When there are two distinct edges, there is then the occurrence when 2 variables in the triple \((e_1, e_2, e_3)\) are bound to \(e\). There are three combinations for this occurrence in \(\Phi_3^i_G(x)\). Analogously, there are three such occurrences in \(\Phi_3^i_G(x)\) when there is only one occurrence of \(e\), i.e. 2 of the variables in \((e_1, e_2, e_3)\) are \(e'\). This implies that all \(3 + 3 = 6\) combinations of two distinct edges \(e\) and \(e'\) contribute to the same monomial in \(\Phi_3^i_G\). Since \(e \neq e'\), this case produces the following edge patterns: \(\lambda, \lambda', \lambda\), which contribute \(6p^3, 6p^4\) respectively to \(\Phi_3^i_G(p, \ldots, p)\).

Case 3: All \(e_1, e_2\) and \(e_3\) are distinct. For this case, we have \(3! = 6\) permutations of \((e_1, e_2, e_3)\), each of which contribute to the same monomial in the SMB representation of \(\Phi_3^i_G(x)\). This case consists of the following edge patterns: \(\lambda, \lambda, \lambda, \lambda, \lambda, \lambda\), which contribute \(6p^3, 6p^4, 6p^5, 6p^5\) and \(6p^6\) respectively to \(\Phi_3^i_G(p, \ldots, p)\).

Since \(p\) is fixed, Lemma 3.8 gives us one linear equation in \(#(G, \lambda)\) and \(#(G, \lambda')\) (we can handle the other counts due to equations (17)-(22)). However, we need to generate one more independent linear equation in these two variables. Towards this end we generate another graph related to \(G\):
Definition B.3. For \( t \geq 1 \), let graph \( G^{(t)} \) be a graph generated from an arbitrary graph \( G \), by replacing every edge \( e \) of \( G \) with an \( t \)-path, such that all inner vertices of an \( t \)-path replacement edge have degree 2.\(^{12}\)

We will prove Theorem 3.7 by the following reduction:

Theorem B.4. Fix \( p \in (0, 1) \). Let \( G \) be a graph on \( m \) edges. If we can compute \( \tilde{E}_G^3(p, \ldots, p) \) exactly in \( T(m) \) time, then we can exactly compute \( \#(G, \Delta) \) in \( O(T(m) + m) \) time.

For clarity, we repeat the notion of \( \#(G, H) \) to mean the count of subgraphs in \( G \) isomorphic to \( H \). The following lemmas relate these counts in \( G^{(2)} \) to \( G^{(1)} \) and are useful in proving Lemma 3.9.

Lemma B.5. The 3-matchings in graph \( G^{(2)} \) satisfy the identity:

\[
\# \left( G^{(2)}, \text{III} \right) = 8 \cdot \# \left( G^{(1)}, \text{III} \right) + 6 \cdot \# \left( G^{(1)}, \text{I} \Delta \right) + 4 \cdot \# \left( G^{(1)}, \text{I} \right) + 2 \cdot \# \left( G^{(1)}, \Delta \right).
\]

Lemma B.6. For \( t > 1 \) and any graph \( G^{(t)} \), \( \# \left( G^{(t)}, \Delta \right) = 0 \).

B.9 Proofs for Lemma B.5, Lemma B.6, and Lemma 3.9

Before proceeding, let us introduce a few more helpful definitions.

Definition B.7 (\( E^{(t)} \)). For \( t > 1 \), we use \( E^{(t)} \) to denote the set of edges in \( G^{(t)} \). For any graph \( G^{(t)} \), its edges are denoted by the a pair \( (e, b) \), such that \( b \in \{0, \ldots, t – 1\} \) where \( (e, 0), \ldots, (e, t – 1) \) is the \( t \)-path that replaces the edge \( e \) for \( e \in E^{(1)} \).

Definition B.8 (\( E_S^{(t)} \)). Given an arbitrary subgraph \( S^{(1)} \) of \( G^{(1)} \), let \( E_S^{(t)} \) denote the set of edges in \( S^{(1)} \). Define then \( E_S^{(t)} \) for \( t > 1 \) as the set of edges in the generated subgraph \( S^{(t)} \) (i.e. when we apply Definition B.3 to \( S \) to generate \( S^{(t)} \)).

For example, consider \( S^{(1)} \) with edges \( E_S^{(1)} = \{e_1\} \). Then the edge set of \( S^{(2)} \) is defined as \( E_S^{(2)} = \{(e_1, 0), (e_1, 1)\} \).

Definition B.9 (\( \ell \) and \( \ell \)). Let \( \ell \) denote the set of subsets in \( E \) with exactly \( t \) edges. In a similar manner, \( \ell \) is used to mean the subsets of \( E \) with \( t \) or fewer edges.

The following function \( f_t \) is a mapping from every \( 3 \)-edge shape in \( G^{(t)} \) to its ‘projection’ in \( G^{(1)} \).

Definition B.10. Let \( f_t : \left( E^{(t)}_{S \leq 3} \right) \) be defined as follows. For any element \( s \in \left( E^{(t)}_{S \leq 3} \right) \) such that \( s = \{(e_1, b_1), (e_2, b_2), (e_3, b_3)\} \), define:

\[
f_t \left( (e_1, b_1), (e_2, b_2), (e_3, b_3) \right) = \{e_1, e_2, e_3\}.
\]

Definition B.11 (\( f_t^{-1} \)). For an arbitrary subgraph \( S^{(1)} \) of \( G^{(1)} \) with at most \( m \leq 3 \) edges, the inverse function \( f_t^{-1} : \left( E^{(t)}_{S \leq 3} \right) \to \left( E^{(1)}_{S \leq 3} \right) \) takes \( E_S^{(t)} \) and outputs the set of all elements \( s \in \left( E^{(t)}_{S \leq 3} \right) \) such that \( f_t(s) = E_S^{(1)} \).

Note, importantly, that when we discuss \( f_t^{-1} \), that each edge present in \( E_S^{(1)} \) must have an edge in each \( s \in f_t^{-1}(E_S^{(1)}) \) that projects down to it. In particular, if \( |E_S^{(1)}| = 3 \), then it must be the case that each \( s \in f_t^{-1}(E_S^{(1)}) \) consists of the following set of edges: \( \{e_1, e_1, e_2, e_2, e_3, e_3\} \), where \( i, j, m \) are distinct.

We are now ready to prove the structural lemmas. To prove the structural lemmas, we will count the number of occurrences of \( \Delta \) and \( \text{III} \) in \( G^{(t)} \). This is accomplished by computing how many \( \text{III} \) and \( \Delta \) subgraphs appear in \( f_t^{-1}(E_S^{(1)}) \) for each \( S \in \left( E^{(t)}_{S \leq 3} \right) \).

B.9.1 Proof of Lemma B.5

Proof. For each subset \( E_S^{(1)} \in \left( E^{(1)}_{S \leq 3} \right) \), we count the number of 3-matchings in the 3-edge subgraphs of \( G^{(2)} \) in \( f_2^{-1}(E_S^{(1)}) \). Denote the set of 3-matchings in \( f_2^{-1}(E_S^{(1)}) \) as \( f_2^{-1}(E_S^{(1)}) \) and its size as \( \left| f_2^{-1}(E_S^{(1)}) \right| \). We first consider the case of \( E_S^{(1)} \in \left( E_{S \leq 3}^{(1)} \right) \), where \( E_S^{(1)} \) is composed of the edges \( e_1, e_2, e_3 \) and \( f_2^{-1}(E_S^{(1)}) \) is the set of all 3-edge subsets of edges \( e_i \in \{(e_1, 0), (e_1, 1), (e_2, 0), (e_2, 1), (e_3, 0), (e_3, 1)\} \) such that \( f_t(s) = \{e_1, e_2, e_3\} \). For the case that \( f_2^{-1}(E_S^{(1)}) = f_2^{-1}(E_S^{(1)}) \) we have that \( \left| f_2^{-1}(E_S^{(1)}) \right| = 8 \).

We do a case analysis based on the subgraph \( S^{(1)} \) induced by \( E_S^{(1)} \).

- 3-matching \( \text{III} \)
  When \( S^{(1)} \) is isomorphic to \( \text{III} \), it is the case that edges in \( E_S^{(2)} \) are not disjoint only for the pairs \( (e_i, 0), (e_i, 1) \) for \( i \in \{1, 2, 3\} \). By definition, each set of edges in \( f_2^{-1}(E_S^{(1)}) \) is a three matching and \( \left| f_2^{-1}(E_S^{(1)}) \right| = 8 \).

\(^{12}\)Note that \( G = G^{(1)} \).
• Disjoint Two-Path (I.Δ)

For $S^{(1)}$ isomorphic to $I.Δ$, edges $e_2, e_3$ form a 2-path with $e_1$ being disjoint. This means that in $S^{(2)}$ edges $(e_2, 0), (e_2, 1), (e_3, 0), (e_3, 1)$ form a 4-path while $(e_1, 0), (e_1, 1)$ is its own disjoint 2-path. We can pick either $(e_1, 0)$ or $(e_1, 1)$ for the first edge in the 3-matching, while it is necessary to have a 2-matching from path $(e_2, 0), \ldots, (e_3, 1)$. Note that the 4-path allows for three possible 2-matchings, specifically,

$$\{ (e_2, 0), (e_3, 0) \}, \{ (e_2, 0), (e_1, 1) \}, \{ (e_2, 1), (e_3, 1) \}.$$

Since these two selections can be made independently, \(|E_{S}^{-1}(f_{S}^{(1)})| = 2 \cdot 3 = 6.

• 3-star (Δ)

When $S^{(1)}$ is isomorphic to $Δ$, the inner edges $(e_1, 1)$ of $S^{(2)}$ are all connected, and the outer edges $(e_i, 0)$ are all disjoint. Note that for a valid 3-matching it must be the case that at most one inner edge can be part of the set of disjoint edges. For the case of when exactly one inner edge is chosen, there exist 3 possibilities, based on which inner edge is chosen. Note that if $(e_1, 1)$ is chosen, the matching has to choose $(e_j, 0)$ for $j \neq i$ and $(e_j, 0)$ for $j' \neq i, j' \neq j$. The remaining possible 3-matching occurs when all 3 outer edges are chosen, and

$$|E_{S}^{-1}(f_{S}^{(1)})| = 4.$$

• 3-path (Σ)

When $S^{(1)}$ is isomorphic to $Σ$ it is the case that all edges beginning with $e_1$ and ending with $e_3$ are successively connected. This means that the edges of $E_{S}^{(2)}$ form a 6-path. For a 3-matching to exist in $f_{S}^{-1}(E_{S}^{(1)})$, we cannot pick both $(e_1, 0)$ and $(e_1, 1)$ or both $(e_1, 1)$ and $(e_j, 0)$ where $j = i + 1$. There are four such possibilities: \(\{(e_1, 0), (e_2, 0), (e_3, 0)\}, \{(e_1, 0), (e_2, 0), (e_3, 1)\}, \{(e_1, 0), (e_2, 1), (e_3, 1)\}, \{(e_1, 1), (e_2, 1), (e_3, 1)\}\) and

$$|E_{S}^{-1}(f_{S}^{(1)})| = 4.$$

• Triangle (A)

For $S^{(1)}$ isomorphic to $A$, note that it is the case that the edges in $E_{S}^{(2)}$ are connected in a successive manner, but this time in a cycle, such that $(e_1, 0)$ and $(e_3, 1)$ are also connected. While this is similar to the discussion of the three path above, the first and last edges are not disjoint. This rules out both subsets of $(e_1, 0), (e_2, 0), (e_3, 1)$ and $(e_1, 0), (e_2, 1), (e_3, 1)$, so that

$$|E_{S}^{-1}(f_{S}^{(1)})| = 2.$$

Let us now consider when $E_{S}^{(1)} \in \{ E_{S}^{(1)} \}_{S \in \mathcal{S}}$, i.e. fixed subgraphs among

• 2-matching (Σ), 2-path (Δ), 1 edge (I)

When $|E_{S}^{-1}(f_{S}^{(1)})| = 2$, we can only pick one from each of two pairs, \(\{(e_1, 0), (e_1, 1)\}\) and \(\{(e_2, 0), (e_2, 1)\}\). The third edge choice in $E_{S}^{(2)}$ will break the disjoint property of a 3-matching. Thus, a 3-matching cannot exist in $f_{S}^{-1}(E_{S}^{(1)})$. A similar argument holds for $|E_{S}^{(1)}| = 1$, where the output of $f_{S}^{-1}$ is \(\emptyset\) since there are not enough edges in the input to produce any other output.

Observe that all of the arguments above focused solely on the property of subgraph $S^{(1)}$ being isomorphic. In other words, all $E_{S}^{(1)}$ of a given “shape” yield the same number of 3-matchings in $f_{S}^{-1}(E_{S}^{(1)})$, and this is why we get the required identity using the above case analysis. $\square$

B.9.2 Proof of Lemma B.6.

**Proof.** The number of triangles in $G^{(t)}$ for $t \geq 2$ will always be 0 for the simple fact that all cycles in $G^{(t)}$ will have at least six edges. $\square$

B.9.3 Proof of Lemma 3.9.

**Proof.** The proof consists of two parts. First we need to show that a vector $b$ satisfying the linear system exists and further can be computed in $O(m)$ time. Second we need to show that # $(G, \Delta), # (G, \Sigma)$ can indeed be computed in time $O(1)$.

The lemma claims that for $M = \begin{pmatrix} 1 - 3p & -(3p^2 - p^3) \\ 10(3p^2 - p^3) & 10(3p^2 - p^3) \end{pmatrix}$, $x = \begin{pmatrix} # (G, \Delta) \\ # (G, \Sigma) \end{pmatrix}$ satisfies the linear system $M \cdot x = b$.

To prove the first step, we use Lemma 3.8 to derive the following equality (dropping the superscript and referring to $G^{(t)}$ as $G$):

$$# (G, I) p^2 + 6# (G, \Delta) p^3 + 6# (G, I) p^4 + 6# (G, \Sigma) p^5 + 6# (G, \Delta) p^6 + 6# (G, \Sigma) p^7 + 6# (G, \Delta) p^8 + 6# (G, \Sigma) p^9 = \Phi_G(p, \ldots, p)$$

(23)

$$# (G, \Delta) + # (G, \Sigma) p + # (G, I) p^2 + # (G, \Sigma) p^3 \quad \frac{\Phi_G(p, \ldots, p)}{6p^3} = \frac{# (G, I)}{6p} - \frac{# (G, \Delta)}{6p} - \frac{# (G, \Sigma)}{6p}$$

(24)

$$# (G, \Delta) (1 - 3p) - # (G, \Sigma) (3p^2 - p^3) =$$

$$\frac{\Phi_G(p, \ldots, p)}{6p^3} = \frac{# (G, I)}{6p} - \frac{# (G, \Delta)}{6p} - \frac{# (G, \Sigma)}{6p}$$
The steps to obtaining eq. (27) are analogous to the derivation of Eq. (25). As in the previous derivation, note that the LH S of Eq. (27) is the form of eq. (25) for is equal to the RHS of Eq. (25), where every term is computable in O(m) time (by equations (17)-(22)). We set b[1] to the RHS of Eq. (25).

We follow the same process in deriving an equality for G^{(2)}. Replacing occurrences of G with G^{(2)}, we obtain an equation (below) of the form of eq. (25) for G^{(2)}. Substituting identities from lemma B.5 and Lemma B.6 we obtain

\[ 0 - (8\# (G, \mathbf{1}) + 4\# (G, \Delta) + 4\# (G, \mathbf{1}) + 2\# (G, \Delta)) (3p^2 - p^3) = \]

\[ \frac{3}{6p^3} \cdot \left[ \# (G^{(2)} \mathbf{1}) - p - \# (G^{(2)} \Delta) \right] \]

\[ - \left[ \# (G^{(2)} \mathbf{1}) p^2 + 3\# (G^{(2)} \mathbf{1}) p^2 \right] - \left[ \# (G^{(2)} \Delta) p + 3\# (G^{(2)} \Delta) \right] \]

\[ (10\# (G, \Delta) + 10G^{(2)}) (3p^2 - p^3) = \]

\[ \frac{3}{6p^3} \cdot \left[ \# (G^{(2)} \mathbf{1}) - \# (G^{(2)} \Delta) \right] - \# (G^{(2)} \mathbf{1}) p^2 - 3\# (G^{(2)} \Delta) p^2 \]

\[ + (4\# (G, \Delta) + 6\# (G, \mathbf{1}) + 18\# (G, \mathbf{1}) + 12\# (G, \Delta)) (3p^2 - p^3) \]

The steps to obtaining eq. (27) are analogous to the derivation of Eq. (25). As in the previous derivation, note that the LHS of Eq. (27) is the same as M[2] - x[2]. The RHS of Eq. (27) has terms all computable (by equations (17)-(22)) in O(m) time. Setting b[2] to the RHS then completes the proof of step 1.

Note that if M has full rank then one can compute # (G, \Delta) and # (G, \mathbf{1}) in O(1) using Gaussian elimination.

To show that M indeed has full rank, we show in what follows that Det (M) \neq 0 for every p \in (0, 1). Det (M) =

\[ \frac{1 - 3p}{10(3p^2 - p^3)} \cdot \frac{-3p^2 - p^3}{10(3p^2 - p^3)} = (1 - 3p) \cdot 10(3p^2 - p^3) + 10(3p^2 - p^3) \cdot (3p^2 - p^3) \]

\[ = 10(3p^2 - p^3) \cdot (1 - 3p + 3p^2 - p^3) = 10(3p^2 - p^3) \cdot (3p^2 - p^3 - 3p + 1) \]

\[ = 10p^2(1 - p) \cdot (1 - p)^3 \]

From Eq. (28) it can easily be seen that the roots of Det (M) are 0, 1, and 3. Hence there are no roots in (0, 1) and Lemma 3.9 follows.

\[ \square \]

### B.10 Proof of Theorem B.4

Proof. We can compute G^{(2)} from G^{(1)} in O(m) time. Additionally, if in time O(T(m)), we have \( \bar{F}_{G^1}(p, \ldots, p) \) for \( t \in [2] \), then the theorem follows by Lemma 3.9.

\[ \square \]

### B.11 Proof of Theorem 3.7

Proof. For the sake of contradiction, assume that for any G, we can compute \( \bar{F}_{G^1}(p, \ldots, p) \) in \( o(m^{1+\epsilon}) \) time. Let G be the input graph. Then by Theorem B.4 we can compute \# (G, \Delta) in further time \( o(m^{1+\epsilon}) + O(m) \). Thus, the overall reduction takes \( o(m^{1+\epsilon}) + O(m) = o(m^{1+\epsilon}) \) time, which violates Conjecture 3.3.

\[ \square \]

\[ \text{In other words, if Theorem B.4 holds, then so must Theorem 3.7}. \]

### C  MISSING DETAILS FROM SECTION 4

In the following definitions and examples, we use the following polynomial as an example:

\[ \Phi(X, Y) = 2X^2 + 3XY - 2Y^2. \]  

### Definition C.1 (Pure Expansion).

The pure expansion of a polynomial \( \Phi \) is formed by computing all product of sums occurring in \( \Phi \), without combining like monomials. The pure expansion of \( \Phi \) generalizes Definition 2.1 by allowing monomials \( m_1 = m_j \) for \( i \neq j \).

Note that similar in spirit to Definition 2.4, E(C) Definition 4.1 reduces all variable exponents \( e > 1 \) to \( e = 1 \). Further, it is true that E(C) encodes the pure expansion of C.

### Example C.2 (Example of Pure Expansion).

Consider the factorized representation \((X + 2Y)(2X - Y)\) of the polynomial in Eq. (29). Its circuit C is illustrated in Fig. 3. The pure expansion of the product is \( 2X^2 - XY + 4XY - 2Y^2 \). As an additional example of Definition 4.1, E(C) = \[ (X, 2), (XY, -1), (XY, 4), (Y, -2) \].
Algorithm 1 \textsc{Approximate}{}(\textit{C}, \textit{p}, \textit{δ}, \textit{ε})

\textbf{Input}: \textit{C}: Circuit
\textbf{Input}: \textit{p} = (p_1, \ldots, p_n) ∈ [0, 1]^N
\textbf{Input}: \textit{δ} ∈ [0, 1]
\textbf{Input}: \textit{ε} ∈ [0, 1]
\textbf{Output}: \textit{acc} ∈ \mathbb{R}

1: \textit{acc} ← 0
2: \textit{N} ← \left\lceil \frac{\log \frac{2}{\textit{ε}}}{\log \frac{1}{\textit{δ}}} \right\rceil
3: (C_{\text{mod}}, \text{size}) ← \textsc{OnePass}{}(\textit{C}) \quad \triangleright \text{OnePass is Algorithm 2}
4: \textbf{for} \textit{i} ∈ 1 \text{ to } \textit{N} \textbf{do}
5: \quad (\textit{M}, \text{sgn}_1) ← \textsc{SampleMonomial}{}(C_{\text{mod}}) \quad \triangleright \text{SampleMonomial is Algorithm 3}
6: \quad \text{if } \textit{M} \text{ has at most one variable from each block} \textbf{then}
7: \quad \quad \textit{Y}_1 ← \prod_{\text{en} j} p_j
8: \quad \quad \textit{Y}_1 ← \textit{Y}_1 \times \text{sgn}_1
9: \quad \quad \text{acc} ← \text{acc} + \textit{Y}_1 \quad \triangleright \text{Store the sum over all samples}
10: \quad \textbf{end if}
11: \textbf{end for}
12: \textit{acc} ← \text{acc} \times \text{size} \quad \text{N}
13: \textbf{return} \textit{acc}

\textit{E}(\textit{C}) \text{ effectively}^{13} \text{ encodes the reduced form of } \text{poly}{}(\textit{C}), \text{ decoupling each monomial into a set of variables } \text{v} \text{ and a real coefficient } \text{c}. \text{ However, unlike the constraint on the input } \textit{Φ} \text{ to compute } \tilde{\textit{Φ}}, \text{ the input circuit } \textit{C} \text{ does not need to be in SMB/SOP form.}

\textbf{Example C.3 (Example for Definition 4.2).} Using the same factorization from Example C.2, \text{poly}{}(|\textit{C}|) = (X + 2Y)(2X + Y) = 2X^2 + XY + 4XY + 2Y^2 = 2X^2 + 5XY + 2Y^2. \text{ Note that this is not the same as the polynomial from Eq. (29). As an example of the slight abuse of notation we alluded to in Sec. 4, } |\textit{C}|(1, \ldots, 1) = 2(1)^2 + 5(1)(1) + 2(1)^2 = 9.

\textbf{Definition C.4 (Subcircuit).} A subcircuit of a circuit \textit{C} is a circuit \textit{S} such that \textit{S} is a DAG subgraph of the DAG representing \textit{C}. The sink of \textit{S} has exactly one gate \textit{g}.

The following results assume input circuit \textit{C} computed from an arbitrary \textit{R,ÂA,} \textit{query } \textit{Q} \text{ and arbitrary BIDB } \mathcal{D}. \text{ We refer to } \textit{C} \text{ as a BIDB circuit.}

\textbf{Theorem C.5.} Let \textit{C} be an arbitrary BIDB circuit and define \( \Phi(\textit{X}) = \text{poly}{}(\textit{C}) \) and let \( k = \text{deg}{}(\textit{C}) \). Then an estimate \( \varepsilon \) of \( \tilde{\Phi}(p_1, \ldots, p_n) \) can be computed in time

\[ O \left( \frac{\text{size}(\textit{C}) \cdot \log \frac{1}{\varepsilon} (1, \ldots, 1) \cdot k \cdot \log k \cdot \text{depth}(\textit{C}))}{\log (\text{size}(\textit{C}))} \right) \cdot \text{O} \left( \log (|\textit{C}|(1, \ldots, 1)) \right) \]

such that

\[ \Pr \left( \left| \varepsilon - \tilde{\Phi}(p_1, \ldots, p_n) \right| > \varepsilon \cdot \tilde{\Phi}(p_1, \ldots, p_n) \right) \leq \delta. \quad (30) \]

The slight abuse of notation seen in \( |\textit{C}|(1, \ldots, 1) \) is explained after Definition 4.2 and an example is given in Example C.3. The only difference in the use of this notation in Theorem C.5 is that we include an additional exponent to square the quantity.

\subsection{Proof of Theorem C.5}

We prove Theorem C.5 constructively by presenting an algorithm \textsc{Approximate}{}(Algorithm 1) which has the desired runtime and computes an approximation with the desired approximation guarantee. Algorithm \textsc{Approximate}{} uses auxiliary algorithm \textsc{OnePass}{} to compute weights on the edges of a circuit. These weights are then used to sample a set of monomials of \( \Phi(\textit{C}) \) from the circuit \textit{C} by traversing the circuit using the weights to ensure that monomials are sampled with an appropriate probability. The correctness of \textsc{Approximate}{} relies on the correctness (and runtime behavior) of auxiliary algorithms \textsc{OnePass}{} and \textsc{SampleMonomial}{} that we state in the following lemmas (and prove later in this part of the appendix).

\textbf{Lemma C.6.} The \textsc{OnePass}{} function completes in time:

\[ O \left( \text{size}(\textit{C}) \cdot \text{O} \left( \log (|\textit{C}|(1, \ldots, 1)) \right) \right) \]

\footnote{The minor difference here is that \( \text{E}(\textit{C}) \) encodes the reduced form of the pure expansion of the compressed representation, as opposed to the SMB representation}
OnePass guarantees two post-conditions: First, for each subcircuit $S$ of $C$, we have that $S\text{.partial}$ is set to $|S| (1, \ldots, 1)$. Second, when $S\text{.type} = +$, $S\text{.weight} = \left| \frac{S_{i \cdot N_{i \cdot E \cdot M_{i \cdot P \cdot S \cdot I \cdot D \cdot C \cdot 2}}}{} \right|$ and likewise for $S\text{.weight}$. 

To prove correctness of Algorithm 1, we use the following fact that follows from the above lemma: for the modified circuit $(C_{\text{mod}})$ output by OnePass, $C_{\text{mod}\text{-partial}} = |C| (1, \ldots, 1)$.

**Lemma C.7.** The function $\text{SAMPLE\text{-}MONOMIAL}$ completes in time

$$O(\log k \cdot k \cdot \text{DEPTH}(C) \cdot \overline{M} (\log (|C| (1, \ldots, 1)), \log \text{SIZE}(C)))$$

where $k = \text{DEG}(C)$. The function returns every $(v, \text{sign}(c))$ for $(v, c) \in E(C)$ with probability $\frac{|c|}{|C| (1, \ldots, 1)}$.

With the above two lemmas, we are ready to argue the following result:

**Theorem C.8.** For any $C$ with $\text{DEG(POLY}(|C|)) = k$, algorithm 1 outputs an estimate $\tilde{\Phi}(p_1, \ldots, p_n)$ such that

$$\Pr \left( \left| \text{acc} - \tilde{\Phi}(p_1, \ldots, p_n) \right| \geq \varepsilon \cdot |C| (1, \ldots, 1) \right) \leq \delta,$$

in $O \left( \text{SIZE}(C) + \frac{\log \frac{1}{\delta}}{\varepsilon^2} \cdot k \cdot \log k \cdot \text{DEPTH}(C) \cdot \overline{M} (\log (|C| (1, \ldots, 1)), \log \text{SIZE}(C)) \right)$ time.

Before proving Theorem C.8, we use it to argue the claimed runtime of our main result, Theorem C.5.

**Proof of Theorem C.5.** Set $E = \text{APPROXIMATE\text{-}\tilde{\Phi}(C, (p_1, \ldots, p_n), \delta, \varepsilon')}$, where

$$\varepsilon' = \varepsilon \cdot \frac{\tilde{\Phi}(p_1, \ldots, p_n)}{|C| (1, \ldots, 1)},$$

which achieves the claimed error bound on $E$ (acc) trivially due to the assignment to $\varepsilon'$ and Theorem C.8, since $\varepsilon' \cdot |C| (1, \ldots, 1) = \varepsilon \cdot \frac{\tilde{\Phi}(p_1, \ldots, p_n)}{|C| (1, \ldots, 1)} |C| (1, \ldots, 1) = \varepsilon \cdot \tilde{\Phi}(p_1, \ldots, p_n)$.

The claim on the runtime follows from Theorem C.8 since

$$\frac{1}{(\varepsilon')^2} \cdot \log \left( \frac{1}{\delta} \right) = \frac{\log \frac{1}{\delta}}{\varepsilon^2 \left( \frac{\tilde{\Phi}(p_1, \ldots, p_n)}{|C| (1, \ldots, 1)} \right)^2} = \frac{\log \frac{1}{\delta} \cdot |C|^2 (1, \ldots, 1)}{\varepsilon^2 \cdot \tilde{\Phi}(p_1, \ldots, p_n)}$$

\[ \square \]

Let us now prove Theorem C.8:

**C.2 Proof of Theorem C.8**

**Proof.** Consider now the random variables $Y_1, \ldots, Y_N$, where each $Y_1$ is the value of $Y_1$ in algorithm 1 after line 8 is executed. Overloading $\text{ISIND} (\cdot)$ to receive monomial input (recall $v_N$ is the monomial composed of the variables in the set $v$), we have

$$Y_1 = \mathbb{I}_{(\text{ISIND}(v_1))} \cdot \prod_{X_{i \cdot \text{EVAR}(p) \cdot I}} p_i$$

where the indicator variable handles the check in Line 6 Then for random variable $Y_i$, it is the case that

$$\mathbb{E} \left[ Y_i \right] = \sum_{(v, c) \in E(C)} \mathbb{I}_{(\text{ISIND}(v_1))} \cdot c \cdot \prod_{X_{i \cdot \text{EVAR}(p) \cdot I}} p_i$$

where in the first equality we use the fact that $\text{sgn}_1 \cdot |C| = c$ and the second equality follows from Eq. (4) with $X_i$ substituted by $p_i$. Let $Y = \frac{1}{N} \sum_{i=1}^{N} Y_i$. It is also true that

$$\mathbb{E} \left[ \sum_{i=1}^{N} Y_i \right] = \frac{\tilde{\Phi}(p_1, \ldots, p_n)}{|C| (1, \ldots, 1)}.$$
Hoeffding’s inequality states that if we know that each $Y_i$ (which are all independent) always lie in the intervals $[a_i, b_i]$, then it is true that

$$Pr \left( |\bar{Y} - E[\bar{Y}]| \geq \epsilon \right) \leq 2 \exp \left( -\frac{2\epsilon^2}{\sum_{i=1}^{N} (b_i - a_i)^2} \right).$$

Line 5 shows that $sgn_1$ has a value in $\{-1, 1\}$ that is multiplied with $O(k) \cdot p_i \in [0, 1]$, which implies the range for each $Y_i$ is $[-1, 1]$. Using Hoeffding’s inequality, we then get:

$$Pr \left( |\bar{Y} - E[\bar{Y}]| \geq \epsilon \right) \leq 2 \exp \left( -\frac{2\epsilon^2}{2^2N} \right) = 2 \exp \left( -\frac{Ne^2}{2} \right) \leq \delta,$$

where the last inequality dictates our choice of $N$ in Line 2. For the claimed probability bound of $Pr \left( |acc - \Phi(p_1, \ldots, p_n)\bar{C}| \geq \epsilon \cdot |C| (1, \ldots, 1)\right) \leq \delta$, note that in the algorithm, acc is exactly $\bar{Y} \cdot |C| (1, \ldots, 1)$. Multiplying the rest of the terms by the additional factor $|C| (1, \ldots, 1)$ yields the said bound.

This concludes the proof for the first claim of theorem C.8. Next, we prove the claim on the runtime.

**Run-time Analysis.** The runtime of the algorithm is dominated first by Line 3 which has $O \left( \text{size}(C) \cdot \bar{M} \left( \log (|C| (1, \ldots, 1)), \log (\text{size}(C)) \right) \right)$ runtime by Lemma C.6. There are then $N$ iterations of the loop in Line 4. Each iteration’s run time is dominated by the call to SAMPLEMOMENT in Line 5 (which by Lemma C.7 takes $O \left( \log k \cdot k \cdot \text{DEPTH}(C) \cdot \bar{M} \left( \log (|C| (1, \ldots, 1)), \log (\text{size}(C)) \right) \right)$) and the check Line 6, which by the subsequent argument takes $O(k \log k)$ time. We sort the $O(k)$ variables by their block IDs and then check if there is a duplicate block ID or not. Combining all the times discussed here gives us the desired overall runtime.

**C.3 Proof of Theorem 4.7**

**Proof.** The result follows by first noting that by definition of $y$, we have

$$\Phi(1, \ldots, 1) = (1 - y) \cdot |C| (1, \ldots, 1).$$

Further, since each $p_i \geq p_0$ and $\Phi(X)$ (and hence $\Phi(X))$ has degree at most $k$, we have that

$$\Phi(1, \ldots, 1) \geq p_0^k \cdot \Phi(1, \ldots, 1).$$

The above two inequalities implies $\Phi(1, \ldots, 1) \geq p_0^k \cdot (1 - y) \cdot |C| (1, \ldots, 1)$. Applying this bound in the runtime bound in Theorem C.5 gives the first claimed runtime. The final runtime of $O_k \left( \frac{1}{\delta^2} \cdot \text{size}(C) \cdot \log \frac{1}{\delta} \cdot \bar{M} \left( \log (|C| (1, \ldots, 1)), \log (\text{size}(C)) \right) \right)$ follows by noting that $\text{DEPTH}(C) \leq \text{size}(C)$ and absorbing all factors that just depend on $k$.

**C.4 ONEPass Remarks**

Please note that it is assumed that the original call to ONEPass consists of a call on an input circuit $C$ such that the values of members partial, Lweight and Rweight have been initialized to Null across all gates.

The evaluation of $|C| (1, \ldots, 1)$ can be defined recursively, as follows (where $C_L$ and $C_R$ are the ‘left’ and ‘right’ inputs of $C$ if they exist):

$$|C| (1, \ldots, 1) = \begin{cases} |C_L| (1, \ldots, 1) \cdot |C_R| (1, \ldots, 1) & \text{if } C\text{-type} = \times \\ |C_L| (1, \ldots, 1) + |C_R| (1, \ldots, 1) & \text{if } C\text{-type} = + \\ |C|\text{val} & \text{if } C\text{-type} = \text{NUM} \\ 1 & \text{if } C\text{-type} = \text{VAR}. \end{cases}$$

(31)

It turns out that for proof of Lemma C.7, we need to argue that when $C\text{-type} = +$, we indeed have

$$C\text{Lweight} \leftarrow \frac{|C_L| (1, \ldots, 1)}{|C_L| (1, \ldots, 1) + |C_R| (1, \ldots, 1)};$$

$$C\text{Rweight} \leftarrow \frac{|C_R| (1, \ldots, 1)}{|C_L| (1, \ldots, 1) + |C_R| (1, \ldots, 1)}.$$  

(32)

(33)

**C.5 ONEPass Example**

**Example C.9.** Let $T$ encode the expression $(X + Y)(X - Y) + Y^2$. After one pass, Algorithm 2 would have computed the following weight distribution. For the two inputs of the sink gate $C$, $C\text{Lweight} = \frac{3}{5}$ and $C\text{Rweight} = \frac{1}{5}$. Similarly, for $S$ denoting the left input $C_L$ of $C$, $S\text{Lweight} = S\text{Rweight} = \frac{1}{5}$. This is depicted in Fig. 4.
Algorithm 2 OnePass (C)

Input: C: Circuit
Output: C: Annotated Circuit
Output: sum ∈ N

1: for g in TopOrd (C) do
2:   if g.type = VAR then
3:     g.partial ← 1
4:   else if g.type = NUM then
5:     g.partial ← |g.val|
6:   else if g.type = × then
7:     g.partial ← g_l.partial × g_r.partial
8:   else
9:     g.partial ← g_l.partial + g_r.partial
10:    g_l.weight ← g_l.partial
11:    g_r.weight ← g_r.partial
12: end if
13: sum ← g.partial
14: end for
15: return (sum, C)

C.6 Proof of OnePass (Lemma C.6)

Proof. We prove the correct computation of partial, Lweight, Rweight values on C by induction over the number of iterations in the topological order TopOrd (line 1) of the input circuit C. TopOrd follows the standard definition of a topological ordering over the DAG structure of C.

For the base case, we have only one gate, which by definition is a source gate and must be either VAR or NUM. In this case, as per eq. (31), lines 3 and 5 correctly compute C.partial as 1.

For the inductive hypothesis, assume that OnePass correctly computes S.partial, S.Lweight, and S.Rweight for all gates g in C with k ≥ 0 iterations over TopOrd. We now prove for k + 1 iterations that OnePass correctly computes the partial, Lweight, and Rweight values for each gate g_i in C for i ∈ [k + 1]. The g_{k + 1} must be in the last ordering of all gates g_i. When size (C) > 1, if g_{k+1} is a leaf node, we are back to the base case. Otherwise g_{k+1} is an internal node which requires binary input.

When g_{k+1}.type = +, then by line 9 g_{k+1}.partial = g_{k+1}_L.partial + g_{k+1}_R.partial, a correct computation, as per eq. (31). Further, lines 10 and 11 compute g_{k+1}.Lweight = \frac{g_{k+1}_L.partial}{g_{k+1}.partial} and analogously for g_{k+1}.Rweight. All values needed for each computation have been correctly computed by the inductive hypothesis.

When g_{k+1}.type = ×, then line 7 computes g_{k+1}.partial = g_{k+1}_L.partial × g_{k+1}_R.partial, which indeed by eq. (31) is correct. This concludes the proof of correctness.
We briefly describe the top-down traversal of \texttt{SampleMonomial}. When \texttt{C.type} = +, the input to be visited is sampled from the weighted distribution precomputed by \texttt{OnePass}. When \texttt{C.type} = \times node is visited, both inputs are visited. The algorithm computes two properties: the set of all variable leaf nodes visited, and the product of the signs of visited coefficient leaf nodes. We will assume the TreeSet data structure to maintain sets with logarithmic time insertion and linear time traversal of its elements. While we would like to take advantage of the space efficiency gained in using a circuit \texttt{C} instead of an expression tree \texttt{T}, we do not know that such a method exists when computing a sample of the input polynomial representation.

The efficiency gains of circuits over trees is found in the capability of circuits to only require space for each gate. For the base case, let the depth \( \pi \) be 1. We have that the single gate is either a constant \( \pi \) for which by line 15 we return \{\}, or we have that \texttt{C.type} = \texttt{var} and \texttt{C.val} = \pi, and by line 17 we return \{\pi\}. By definition 4.1, both cases return a valid \pi for some \( (\pi,\pi) \) from \texttt{E(C)}, and the base case is proven.

For the inductive hypothesis, assume that for \( d \leq k \) for some \( k \geq 1 \), that it is indeed the case that \texttt{SampleMonomial} returns a valid monomial.

For the inductive step, let us take a circuit \texttt{C} with \( d = k + 1 \). Note that each input has depth \( d - 1 \leq k \), and by inductive hypothesis both of them sample a valid monomial. Then the sink can be either a \( + \) or \( \times \) gate. For the case when \texttt{C.type} = +, line 3 of \texttt{SampleMonomial} will choose one of the inputs of the source. By inductive hypothesis it is the case that some valid monomial is being randomly sampled from each of the inputs. Then it follows when \texttt{C.type} = + that a valid monomial is sampled by \texttt{SampleMonomial}. When the \texttt{C.type} = \times, line 10 computes the set union of the monomials returned by the two inputs of the sink, and it is trivial to see by definition 4.1 that \( v_\pi \) is a valid monomial encoded by some \( (\pi,\pi) \) of \texttt{E(C)}.

### Algorithm 3 \texttt{SampleMonomial}(\texttt{C})

\begin{verbatim}
Input: \texttt{C}: Circuit
Output: \texttt{vars}: TreeSet
Output: \texttt{sgn} \in \{-1,1\}
1: \texttt{vars} \leftarrow \emptyset
2: if \texttt{C.type} = + then
3: \hspace{1em} \texttt{C samp} \leftarrow Sample from left input (\texttt{C_L}) and right input (\texttt{C_R}) w.p. \texttt{C.L.weight} and \texttt{C.R.weight}.
4: \hspace{1em} return \texttt{ SAMPLEMONOMIAL(C samp)}
5: else if \texttt{C.type} = \times then
6: \hspace{1em} return \{\texttt{vars}, \texttt{sgn}\}
7: \hspace{1em} \texttt{sgn} \leftarrow 1
8: \hspace{1em} for \texttt{input} in \texttt{C.input} do
9: \hspace{2em} \texttt{(v, s)} \leftarrow \texttt{ SAMPLEMONOMIAL(input)}
10: \hspace{2em} \texttt{vars} \leftarrow \texttt{vars} \cup \{v\}
11: \hspace{2em} \texttt{sgn} \leftarrow \texttt{sgn} \times s
12: \hspace{1em} end for
13: return \texttt{ (vars, sgn)}
14: else if \texttt{C.type} = \texttt{num} then
15: \hspace{1em} return \{\}, \texttt{sgn(C.val)}
16: else if \texttt{C.type} = \texttt{var} then
17: \hspace{1em} return \{(\texttt{C.val}, 1)\}
18: end if
\end{verbatim}

**Runtime Analysis.** It is known that \texttt{TopOrd}(\texttt{G}) is computable in linear time. There are \texttt{size}(\texttt{C}) iterations. Each iteration has runtime \( O(\overline{M}(\log (|\texttt{C}(1\ldots,1)|), \log (\texttt{size}(\texttt{C})))) \) time. This can be seen since each of all the numbers which the algorithm computes is at most \(|\texttt{C}(1\ldots,1)|\). Hence, by definition each such operation takes \( \overline{M}(\log (|\texttt{C}(1\ldots,1)|), \log (\texttt{size}(\texttt{C}))) \) time, which proves the claimed runtime. 

### C.7 \textbf{SampleMonomial} Remarks

We briefly describe the top-down traversal of \texttt{SampleMonomial}. When \texttt{C.type} = +, the input to be visited is sampled from the weighted distribution precomputed by \texttt{OnePass}. When a \texttt{C.type} = \times node is visited, both inputs are visited. The algorithm computes two properties: the set of all variable leaf nodes visited, and the product of the signs of visited coefficient leaf nodes. We will assume the TreeSet data structure to maintain sets with logarithmic time insertion and linear time traversal of its elements. While we would like to take advantage of the space efficiency gained in using a circuit \texttt{C} instead of an expression tree \texttt{T}, we do not know that such a method exists when computing a sample of the input polynomial representation.

The efficiency gains of circuits over trees is found in the capability of circuits to only require space for each distinct term in the compressed representation. This saves space in such polynomials containing non-distinct terms multiplied or added to each other, e.g., \( x^4 \). However, to avoid biased sampling, it is imperative to sample from both inputs of a multiplication gate, independently, which is indeed the approach of \texttt{SampleMonomial}.

### C.8 Proof of \texttt{SampleMonomial} (Lemma C.7)

**Proof.** We first need to show that \texttt{SampleMonomial} samples a valid monomial \( v_\pi \) by sampling and returning a set of variables \( v \), such that \( (v,\pi) \) is in \texttt{E(C)} and \( v_\pi \) is indeed a monomial of the \( \Phi(X) \) encoded in \texttt{C}. We show this via induction over the depth of \texttt{C}. For the base case, let the depth \( \pi \) be 1. We have that the single gate is either a constant \( \pi \) for which by line 15 we return \{\}, or we have that \texttt{C.type} = \texttt{var} and \texttt{C.val} = \pi, and by line 17 we return \{\pi\}. By definition 4.1, both cases return a valid \pi for some \( (\pi,\pi) \) from \texttt{E(C)}, and the base case is proven.

For the inductive hypothesis, assume that for \( d \leq k \) for some \( k \geq 1 \), that it is indeed the case that \texttt{SampleMonomial} returns a valid monomial.

For the inductive step, let us take a circuit \texttt{C} with \( d = k + 1 \). Note that each input has depth \( d - 1 \leq k \), and by inductive hypothesis both of them sample a valid monomial. Then the sink can be either a \( + \) or \( \times \) gate. For the case when \texttt{C.type} = +, line 3 of \texttt{SampleMonomial} will choose one of the inputs of the source. By inductive hypothesis it is the case that some valid monomial is being randomly sampled from each of the inputs. Then it follows when \texttt{C.type} = + that a valid monomial is sampled by \texttt{SampleMonomial}. When the \texttt{C.type} = \times, line 10 computes the set union of the monomials returned by the two inputs of the sink, and it is trivial to see by definition 4.1 that \( v_\pi \) is a valid monomial encoded by some \( (\pi,\pi) \) of \texttt{E(C)}. 

We will next prove by induction on the depth $d$ of $C$ that for $(v, c) \in E(C)$, $v$ is sampled with a probability $\frac{|c|}{|c|(1, \ldots, 1)}$.

For the base case $d = 1$, by definition 2.7 we know that the size $(C) = 1$ and $C.type = \text{NUM or VAR}$. For either case, the probability of the value returned is 1 since there is only one value to sample from. When $C.val = x$, the algorithm always return the variable set $\{x\}$. When $C.type = \text{NUM}$, $\text{SAMPLEMONOMIAL}$ will always return the variable set $\emptyset$.

For the inductive hypothesis, assume that for $d \leq k$ and $k \geq 1$ $\text{SAMPLEMONOMIAL}$ indeed returns $v$ in $(v, c)$ of $E(C)$ with probability $\frac{|c|}{|c|(1, \ldots, 1)}$.

We prove now for $d = k + 1$ the inductive step holds. It is the case that the sink of $C$ has two inputs $C_L$ and $C_R$. Since $C_L$ and $C_R$ are both depth $d - 1 \leq k$, by inductive hypothesis, $\text{SAMPLEMONOMIAL}$ will return $v_L$ in $(v_L, c_L)$ of $E(C_L)$ and $v_R$ in $(v_R, c_R)$ of $E(C_R)$, from $C_L$ and $C_R$ with probability $\frac{|c_L|}{|c_L|(1, \ldots, 1)}$ and $\frac{|c_R|}{|c_R|(1, \ldots, 1)}$.

Consider the case when $C.type = \times$. For the term $(v, c)$ from $E(C)$ that is being sampled it is the case that $v = v_L \cup v_R$, where $v_L$ is coming from $C_L$ and $v_R$ from $C_R$. The probability that $\text{SAMPLEMONOMIAL}((C_L))$ returns $v_L$ is $\frac{|c_L|}{|c_L|(1, \ldots, 1)}$ and $\frac{|c_R|}{|c_R|(1, \ldots, 1)}$ for $v_R$. Since both $v_L$ and $v_R$ are sampled with independent randomness, the final probability for sample $v$ is then $\frac{|c_L|}{|c_L|(1, \ldots, 1)} \cdot \frac{|c_R|}{|c_R|(1, \ldots, 1)}$. For $(v, c)$ in $E(C)$, by definition 4.1 it is the case that $|c| = |c_L| \cdot |c_R|$ and that (as shown in eq. (31)) $|c|(1, \ldots, 1) = |c_L|(1, \ldots, 1) \cdot |c_R|(1, \ldots, 1)$, and therefore $v$ is sampled with correct probability $\frac{|c|}{|c|(1, \ldots, 1)}$.

For the case when $C.type = +$, $\text{SAMPLEMONOMIAL}$ will sample $v$ from one of its inputs. By inductive hypothesis we know that any $v_L$ in $E(C_L)$ and any $v_R$ in $E(C_R)$ will both be sampled with correct probability $\frac{|c_L|}{|c_L|(1, \ldots, 1)}$ and $\frac{|c_R|}{|c_R|(1, \ldots, 1)}$, where either $v_L$ or $v_R$ will equal $v$, depending on whether $C_L$ or $C_R$ is sampled. Assume that $v$ is sampled from $C_L$, and note that a symmetric argument holds for the case when $v$ is sampled from $C_R$. Notice also that the probability of choosing $C_L$ from $C$ is $\frac{|c_L|}{|c_L|(1, \ldots, 1)} \cdot \frac{|c_R|}{|c_R|(1, \ldots, 1)}$ as computed by $\text{OnePass}$. Then, since $\text{SAMPLEMONOMIAL}$ goes top-down, and each sampling choice is independent (which follows from the randomness in the root of $C$ being independent from the randomness used in its subtrees), the probability for $v$ to be sampled from $C$ is equal to the product of the probability that $C_L$ is sampled from $C$ and $v$ is sampled in $C_L$, and

$$Pr(\text{SAMPLEMONOMIAL}(C) = v) =$$

$$Pr(\text{SAMPLEMONOMIAL}(C_L) = v) \cdot Pr(\text{SampledChild}(C) = C_L) = \frac{|c_L|}{|c_L|(1, \ldots, 1)} \cdot \frac{|c_L|(1, \ldots, 1) + |c_R|(1, \ldots, 1)}{|c|(1, \ldots, 1)} = \frac{|c|}{|c|(1, \ldots, 1)}.$$ 

and we obtain the desired result.

Lastly, we show by simple induction of the depth $d$ of $C$ that $\text{SAMPLEMONOMIAL}$ indeed returns the correct sign value of $c$ in $(v, c)$.

In the base case, $C.type = \text{NUM or VAR}$. For the former by Line 15, $\text{SAMPLEMONOMIAL}$ correctly returns the sign value of the gate. For the latter by Line 17, $\text{SAMPLEMONOMIAL}$ returns the correct sign of 1, since a variable is a neutral element, and 1 is the multiplicative identity, whose product with another sign element will not change that sign element.

For the inductive hypothesis, we assume for a circuit of depth $d \leq k$ and $k \geq 1$ that the algorithm correctly returns the sign value of $c$. Similar to before, for a depth $d \leq k + 1$, it is true that $C_L$ and $C_R$ both return the correct sign of $c$. For the case that $C.type = \times$, the sign value of both inputs are multiplied, which is the correct behavior by definition 4.1. When $C.type = +$, only one input of $C$ is sampled, and the algorithm returns the correct sign value of $c$ by inductive hypothesis.

**Run-time Analysis.** It is easy to check that except for lines 3 and 10, all lines take $O(1)$ time. Consider an execution of line 10. We note that we will be adding a given set of variables to some set at most once: since the sum of the sizes of the sets at a given level is at most $\deg(C)$, each gate visited takes $O(\log \deg(C))$. For Line 3, note that we pick $C_L$ with probability $\frac{a}{a+b}$, where $a = C.L\text{weight}$ and $b = C.R\text{weight}$. We can implement this step by picking a random number $r \in [a+b]$ and then checking if $r \leq a$. It is easy to check that $a+b \leq |C|(1, \ldots, 1)$. This means we need to add and compare $\log(|C|(1, \ldots, 1))$-bit numbers, which can certainly be done in time $\mathcal{M}(\log(|C|(1, \ldots, 1))), \log \deg(C)$ (note that this is an over-estimate). Denote $\text{Cost}(\cdot)$ (Eq. (34)) to be an upper bound of the number of gates visited by $\text{SAMPLEMONOMIAL}$. Then the runtime is $O\left(\text{Cost}(\cdot) \cdot \log \deg(C) \cdot \mathcal{M}(\log(|C|(1, \ldots, 1))), \log \deg(C)\right)$.

Since there can be at most $\deg(C)$ nodes visited at every level of the circuit, and each of the first $d - 1$ levels (going from the sink to the source nodes) will contain at least one recursive call, we can upperbound the number of recursive calls in $\text{SAMPLEMONOMIAL}$ by $O(\deg(C) + 1) \cdot \text{depth}(C)$, which by the above will prove the claimed runtime of Lemma C.7.

Let $\text{Cost}(\cdot)$ be a function that models an upper bound on the number of gates that can be visited in the run of $\text{SAMPLEMONOMIAL}$. We define $\text{Cost}(\cdot)$ recursively as follows.
First note that the number of gates visited in \textsc{SampleMonomial} is \( \leq \text{Cost}(C) \). To show that eq. (34) upper bounds the number of nodes visited by \textsc{SampleMonomial}, note that when \textsc{SampleMonomial} visits a gate such that \( C\text{.type} = \times \), line 8 visits each input of \( C \), as defined in (34). For the case when \( C\text{.type} = + \), line 3 visits exactly one of the input gates, which may or may not be the subcircuit with the maximum number of gates traversed, which makes \( \text{Cost}(\cdot) \) an upperbound. Finally, it is trivial to see that when \( C\text{.type} \in \{\text{VAR, NUM}\} \), i.e., a source gate, that only one gate is visited.

We prove the following inequality holds.

\[
2(\text{deg}(C) + 1) \cdot \text{depth}(C) + 1 \geq \text{Cost}(C) \tag{35}
\]

Note that eq. (35) implies the claimed runtime.

We prove eq. (35) for the number of gates traversed in \textsc{SampleMonomial} using induction over \text{depth}(C). Recall how degree is defined in definition 4.4. For the base case \( \text{deg}(C) \in \{0, 1\} \), \( \text{depth}(C) = 1 \), \( \text{Cost}(C) = 1 \), and it is trivial to see that the inequality \( 2 \text{deg}(C) \cdot \text{depth}(C) + 1 \geq \text{Cost}(C) \) holds.

For the inductive hypothesis, we assume the bound holds for any circuit where \( \ell \geq \text{depth}(C) \geq 0 \). Now consider the case when \textsc{SampleMonomial} has an arbitrary circuit \( C \) with \( \text{depth}(C) = \ell + 1 \). By definition \( C\text{.type} \in \{+, \times\} \). Note that since \( \text{depth}(C) \geq 2 \), \( C \) must have input(s). Further we know that by the inductive hypothesis the inputs \( C_i \) for \( i \in \{L, R\} \) of the sink gate \( C \) uphold the bound

\[
2(\text{deg}(C_i) + 1) \cdot \text{depth}(C_i) + 1 \geq \text{Cost}(C_i). \tag{36}
\]

In particular, since for any \( i \), eq. (36) holds, then it immediately follows that an inequality whose operands consist of a sum of the aforementioned inequalities must also hold. This is readily seen in the inequality of eq. (38) and eq. (39), where \( 2(\text{deg}(C_L) + 1) \cdot \text{depth}(C_L) \geq \text{Cost}(C_L) \), likewise for \( C_R \), and \( 1 \geq 1 \). It is also true that \( \text{depth}(C_L) \leq \text{depth}(C) - 1 \) and \( \text{depth}(C_R) \leq \text{depth}(C) - 1 \).

If \( C\text{.type} = + \), then \( \text{deg}(C) = \max(\text{deg}(C_L), \text{deg}(C_R)) \). Otherwise \( C\text{.type} = \times \) and \( \text{deg}(C) = \text{deg}(C_L) + \text{deg}(C_R) + 1 \). In either case it is true that \( \text{depth}(C) = \max(\text{depth}(C_L), \text{depth}(C_R)) + 1 \).

If \( C\text{.type} = \times \), then, by eq. (34), substituting values, the following should hold,

\[
2(\text{deg}(C_L) + 1) \cdot \text{depth}(C_L) + 1 \geq 2(\text{deg}(C_L) + 1) \cdot \text{depth}(C_L) + 2(\text{deg}(C_R) + 1) \cdot \text{depth}(C_R) + 3 \tag{37}
\]

To prove (38), first, eq. (37) expands to,

\[
2\text{deg}(C_L) \cdot \text{depth}_{\text{max}} + 2\text{deg}(C_R) \cdot \text{depth}_{\text{max}} + 4\text{depth}_{\text{max}} + 2\text{deg}(C_L) + 2\text{deg}(C_R) + 4 + 1 \tag{40}
\]

where \( \text{depth}_{\text{max}} \) is used to denote the maximum depth of the two input subcircuits. Eq. (38) expands to

\[
2\text{deg}(C_L) \cdot \text{depth}(C_L) + 2\text{deg}(C_R) \cdot \text{depth}(C_R) + 3 \tag{41}
\]

Putting Eq. (40) and Eq. (41) together we get

\[
2\text{deg}(C_L) \cdot \text{depth}_{\text{max}} + 2\text{deg}(C_R) \cdot \text{depth}_{\text{max}} + 4\text{depth}_{\text{max}} + 2\text{deg}(C_L) + 2\text{deg}(C_R) + 5 \geq 2\text{deg}(C_L) \cdot \text{depth}(C_L) + 2\text{deg}(C_R) \cdot \text{depth}(C_R) + 3. \tag{42}
\]

Now to justify (39) which holds for the following reasons. First, eq. (39) is the result of Eq. (34) when \( C\text{.type} = \times \). Eq. (38) is then produced by substituting the upperbound of (36) for each \( \text{Cost}(C_i) \), trivially establishing the upper bound of (39). This proves eq. (35) for the \( \times \) case.

For the case when \( C\text{.type} = + \), substituting values yields

\[
2(\max(\text{deg}(C_L), \text{deg}(C_R)) + 1) \cdot (\max(\text{depth}(C_L), \text{depth}(C_R)) + 1) + 1 \tag{43}
\]

\[
\geq \max(2(\text{deg}(C_L) + 1) \cdot \text{depth}(C_L) + 1, 2(\text{deg}(C_R) + 1) \cdot \text{depth}(C_R) + 1) + 1 \tag{44}
\]

\[
\geq 1 + \max(\text{Cost}(C_L), \text{Cost}(C_R)) = \text{Cost}(C) \tag{45}
\]

To prove (44), eq. (43) expands to

\[
2\text{deg}_{\text{max}} \cdot \text{depth}_{\text{max}} + 2\text{deg}_{\text{max}} + 2\text{depth}_{\text{max}} + 2 + 1. \tag{46}
\]

Since \( \text{deg}_{\text{max}} \cdot \text{depth}_{\text{max}} \geq \text{deg}(C_L) \cdot \text{depth}(C_L) \), the following upperbounds the expansion of eq. (44):

\[
2\text{deg}_{\text{max}} \cdot \text{depth}_{\text{max}} + 2\text{deg}_{\text{max}} + 2\text{depth}_{\text{max}} + 2. \tag{47}
\]

Putting it together we obtain the following for (44):

\[
2\text{deg}_{\text{max}} \cdot \text{depth}_{\text{max}} + 2\text{deg}_{\text{max}} + 2\text{depth}_{\text{max}} + 3 \geq 2\text{deg}_{\text{max}} \cdot \text{depth}_{\text{max}} + 2\text{depth}_{\text{max}} + 2. \tag{48}
\]
where it can be readily seen that the inequality stands and (48) follows. This proves (44).

Similar to the case of $C$, type $\otimes$, (45) follows by equations (34) and (36).

This proves (35) as desired.

\[ \square \]

### C.9 Lemma C.10, Lemma C.11, Corollary C.12, and Proof of Corollary 4.8

**Lemma C.10.** Given $RA^*$ query $Q$ and $c$-TIDB $D$, let $C$ be the circuit computed by $Q(D)$. Then, for the reduced Binary-BIDB $D'$ there exists an equivalent circuit $C'$ obtained from $Q(D')$, such that $\gamma(C') \leq 1 - c^{-(k-1)}$ with $\text{size}(C') \leq \text{size}(C) + O(nc)$ and $\text{depth}(C') = \text{depth}(C) + O(\log c)$.

We briefly connect the runtime in Eq. (6) to the algorithm outline earlier (where we ignore the dependence on $\mathcal{M}(\ldots)$, which is needed to handle the cost of arithmetic operations over integers). The size simplifies to $O(\text{depth}(C))$ term in the runtime.

**Lemma C.11.** For any Binary-BIDB circuit $C$ with $\deg(C) = k$, we have $|C| (1, \ldots, 1) \leq 2^{\text{depth}(C)}$. Further, if $C$ is a tree, then we have $|C| (1, \ldots, 1) \leq \text{size}(C)^{O(k)}$.

Note that the above implies that with the assumption $p_0 > 0$ and $\gamma < 1$ are absolute constants from Theorem 4.7, then the runtime there simplifies to $O_k\left(\frac{1}{\gamma^r} \cdot \text{size}(C)^2 \cdot \log \frac{1}{\delta}\right)$ for general circuits $C$. If $C$ is a tree, then the runtime simplifies to $O_k\left(\frac{1}{(c)^r} \cdot \text{size}(C) \cdot \log \frac{1}{\delta}\right)$, which then answers Problem 1.6 with yes for such circuits.

Finally, note that by Proposition D.1 and Lemma D.2 for any $RA^*$ query $Q$, there exists a circuit $C'$ for $\Phi[Q, D, t]$ such that $\text{depth}(C') \leq O_Q(\log n)$ and $\text{size}(C') \leq O_k(T_{det}(Q, D, c))$. Using this along with Lemma C.11, Theorem 4.7 and the fact that $n \leq T_{det}(Q, D, c)$, we have the following corollary:

**Corollary C.12.** Let $Q$ be an $RA^*$ query and $D$ be a Binary-BIDB with $p_0 > 0$ and $\gamma < 1$, where $p_0, \gamma$ as in Theorem 4.7, are absolute constants. Let $\Phi(X) = \Phi[Q, D, t]$ for any result tuple $t$ with $\deg(\Phi) = k$. Then one can compute an approximation satisfying Eq. (5) in time $O_{k, |Q|, c', \gamma}(T_{det}(Q, D, c))$ (given $Q, D$ and $p_k$ for each $i \in [n]$ that defines $\mathcal{P}$).

### C.10 Proof of Lemma C.10

**Proof.** The circuit $C'$ is built from $C$ in the following manner. For each input gate $g_i$ with $g_i.val = X_{i,j}$, replace $g_i$ with the circuit $S$ encoding the sum $\sum_{j=1}^{d_i} X_{i,j}$. We argue that $C'$ is a valid circuit by the following manner. Let $D' = \{(0, c_i) \}\}$ be the original $c$-TIDB $C$ was generated from. Then, by Proposition 2.3 there exists a Binary-BIDB $D'' = \{X_{i,e}^{D''} | (0, c_i) \} \} \{t, j \ | t \in D, j \in [c]\}$, from which the conversion from $C$ to $C'$ follows. Both $\text{poly}(C)$ and $\text{poly}(C')$ have the same expected multiplicity since (by Proposition 2.3) the distributions $\mathcal{P}$ and $\mathcal{P}'$ are equivalent and $\sum_{j=1}^{d_i} W_{t,j} = W_t$ for $W' \in [1, c]^c$ and $W \in [1, c]^D$ such that $W_t \equiv W_t'$. Finally, note that because there exists a (sub) circuit encoding $\sum_{j=1}^{d_i} X_{i,j}$ that is a balanced binary tree, the above conversion implies the claimed size and depth bounds of the lemma.

Next we argue the claim on $\gamma(C')$. Consider the list of expanded monomials $E(C)$ for $c$-TIDB circuit $C$. Let $v_n = X_{t_1,j_1}^{d_1} \cdots X_{t_d,f}^{d_d}$ be an arbitrary monomial with $f$ variables and let $\otimes$ (abusing notation) $v_n' = \left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right) \cdots \left(\sum_{j=1}^{d_d} X_{t_d,j}^{d_d}\right)$. Then, for $f_t = \sum_{i=1}^{d_i} d_i$, $v_n$ induces the set of monomials $\{ \prod_{i=1}^{f_t} j_i \cdot X_{t_i,j_i} \}_{j_i \in [c]}$ke in the pure expansion of $v_n'$. Recall that a cancellation occurs in $v_n'$ when there exists $t_i,j \neq t_i,j'$ in the same block $B$ where variables $X_{t_i,j}, X_{t_i,j'}$ are in the set of variables $v_n'$ of $v_n'$. Observe that cancellations can only occur for each $X_{t_i,j} = v_n$, where the expansion $\left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right)$ represents the monomial $X_{t_1,j}^{d_1}$ in $D'$. Consider the number of cancellations for $\left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right)$. Then $\gamma \leq 1 - c^{-(d-1)}$, since for each element in the set of cross products $\{X_{t_i,j} \in [d_i], j_i \in [c] : X_{t_i,j}\}$ there are exactly $c$ surviving elements with $j_1 = \cdots = j_{d_i} = j$, i.e. $X_{t_i,j}^{d_1}$ for each $j \in [c]$. The rest of the $c^{d_i} - c$ cross terms cancel. Considering all of $v_n'$, it is the case that the proportion of non-cancellations for each $\left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right)$ $v_n'$ multiply because non-cancelling terms for $\left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right)$ can only be joined with non-cancelling terms of $\left(\sum_{j=1}^{d_1} X_{t_1,j}^{d_1}\right)$ $v_n'$ for $t \neq t'$. This then yields the fraction of cancelled monomials $\gamma \leq 1 - \prod_{i=1}^{d_i} c^{-(d-1)} \leq 1 - c^{-(k-1)}$ where the inequalities take into account the fact that $f_t \leq k$.

Since this is true for arbitrary $v$, the bound follows for $\text{poly}(C')$. \[ \square \]

### C.11 Proof of Lemma C.11

We will prove Lemma C.11 by considering the two cases separately. We start by considering the case when $C$ is a tree:
**Lemma C.13.** Let $C$ be a tree (i.e. the sub-circuits corresponding to two children of a node in $C$ are completely disjoint). Then we have

$$|C| (1, \ldots, 1) \leq \text{SIZE}(C)^{\text{DEG}(C)+1}.$$

**Proof of Lemma C.13.** For notational simplicity define $N = \text{SIZE}(C)$ and $k = \text{DEG}(C)$. We use induction on $\text{DEPTH}(C)$ to show that $|C| (1, \ldots, 1) \leq N^{k+1}$. For the base case, we have that $\text{DEPTH}(C) = 0$, and there can only be one node which must contain a coefficient or constant. In this case, $|C| (1, \ldots, 1) = 1$, and by Definition 4.4 it is the case that $0 \leq k = \text{DEG}(C) \leq 1$, and it is true that $|C| (1, \ldots, 1) = 1 \leq N^{k+1} = 1^{k+1} = 1$ for $k \in \{0, 1\}$.

Assume for $\ell > 0$ an arbitrary circuit $C$ of $\text{DEPTH}(C) = \ell$ that it is true that $|C| (1, \ldots, 1) \leq N^{k+1}$.

For the inductive step we consider a circuit $C$ such that $\text{DEPTH}(C) = \ell + 1$. The sink can only be either a $\times$ or $+$ gate. Let $k_L, k_R$ denote $\text{DEG}(C_L)$ and $\text{DEG}(C_R)$ respectively. Consider when sink node is $\times$. Then note that

$$|C| (1, \ldots, 1) = |C_L| (1, \ldots, 1) \cdot |C_R| (1, \ldots, 1)$$

$$\leq (N-1)^{k_L+1} \cdot (N-1)^{k_R+1}$$

$$= (N-1)^{k+1}$$

$$\leq N^{k+1}.$$  (49)

In the above the first inequality follows from the inductive hypothesis (and the fact that the size of either subtree is at most $N-1$) and Eq. (49) follows by definition 4.4 which states that for $k = \text{DEG}(C)$ we have $k = k_L + k_R + 1$.

For the case when the sink gate is a $+$ gate, then for $N_L = \text{SIZE}(C_L)$ and $N_R = \text{SIZE}(C_R)$ we have

$$|C| (1, \ldots, 1) = |C_L| (1, \ldots, 1) + |C_R| (1, \ldots, 1)$$

$$\leq N_L^{k_L+1} + N_R^{k_R+1}$$

$$\leq (N-1)^{k+1}$$

$$\leq N^{k+1}.$$  (50)

In the above, the first inequality follows from the inductive hypothesis and definition 4.4 (which implies the fact that $k_L, k_R \leq k$). Note that the RHS of this inequality is maximized when the base and exponent of one of the terms is maximized. The second inequality follows from this fact as well as the fact that since $C$ is a tree we have $N_L + N_R = N - 1$ and, lastly, the fact that $k \geq 0$. This completes the proof.

The upper bound in Lemma C.11 for the general case is a simple variant of the above proof (but we present a proof sketch of the bound below for completeness):

**Lemma C.14.** Let $C$ be a (general) circuit. Then we have

$$|C| (1, \ldots, 1) \leq 2^{2^{|\text{IN}(C)| \cdot \text{DEPTH}(C)}}.$$

**Proof Sketch of Lemma C.14.** We use the same notation as in the proof of Lemma C.13 and further define $d = \text{DEPTH}(C)$. We will prove by induction on $\text{DEPTH}(C)$ that $|C| (1, \ldots, 1) \leq 2^{2^k \cdot d}$. The base case argument is similar to that in the proof of Lemma C.13. In the inductive case we have that $d_L, d_R \leq d - 1$.

For the case when the sink node is $\times$, we get that

$$|C| (1, \ldots, 1) = |C_L| (1, \ldots, 1) \times |C_R| (1, \ldots, 1)$$

$$\leq 2^{2^k \cdot d_L} \times 2^{2^k \cdot d_R}$$

$$\leq 2^{2^k \cdot (d-1)}$$

$$\leq 2^{2^k \cdot d}.$$  

In the above the first inequality follows from inductive hypothesis while the second inequality follows from the fact that $k_L, k_R \leq k - 1$ and $d_L, d_R \leq d - 1$, where we substitute the upper bound into every respective term.

Now consider the case when the sink node is $+$, we get that

$$|C| (1, \ldots, 1) = |C_L| (1, \ldots, 1) + |C_R| (1, \ldots, 1)$$

$$\leq 2^{2^k \cdot d_L} + 2^{2^k \cdot d_R}$$

$$\leq 2 \cdot 2^{2^k \cdot (d-1)}$$

$$\leq 2^{2^k \cdot d}.$$  

In the above the first inequality follows from the inductive hypothesis while the second inequality follows from the facts that $k_L, k_R \leq k$ and $d_L, d_R \leq d - 1$. The final inequality follows from the fact that $k \geq 0$. □
Proof of Corollary 4.8. By Lemma C.10 and Corollary C.12, the proof follows.

\[ \text{□} \]

C.12 Experimental Results

Recall that by definition of BIDB, a query result cannot be derived by a self-join between non-identical tuples belonging to the same block. Note, that by Theorem 4.7, \( y \) must be a constant in order for Algorithm 1 to achieve linear time. We would like to determine experimentally whether queries over BIDB instances in practice generate a constant number of cancellations or not. Such an experiment would ideally use a database instance with queries both considered to be typical representations of what is seen in practice.

We ran our experiments using Windows 10 WSL Operating System with an Intel Core i7 2.40GHz processor and 16GB RAM. All experiments used the PostgreSQL 13.0 database system.

For the data we used the MayBMS data generator [43] tool to randomly generate uncertain versions of TPCH tables. The queries computed over the database instance are \( Q_1 \), \( Q_2 \), and \( Q_3 \) from [4], all of which are modified versions of TPC-H queries \( Q_3 \), \( Q_6 \), and \( Q_7 \) where all aggregations have been dropped.

As written, the queries disallow BIDB cross terms. We first ran all queries, noting the result size for each. Next the queries were rewritten so as not to filter out the cross terms. The comparison of the sizes of both result sets should then suggest in one way or another whether or not there exist many cross terms in practice. As seen, the experimental query results contain little to no cancelling terms. Fig. 5 shows the result sizes of the queries, where column CF is the result size when all cross terms are filtered out, column CI shows the number of output tuples when the cancelled tuples are included in the result, and the last column is the value of \( y \). The experiments show \( y \) to be in a range between \([0, 0.1]\)\%, indicating that only a negligible or constant (compare the result sizes of \( Q_1 \) < \( Q_2 \) and their respective \( y \) values) amount of tuples are cancelled in practice when running queries over a typical BIDB instance. Interestingly, only one of the three queries had tuples that violated the BIDB constraint.

To conclude, the results in Fig. 5 show experimentally that \( y \) is negligible in practice for BIDB queries. We also observe that (i) tuple presence is independent across blocks, so the corresponding probabilities (and hence \( p_q \)) are independent of the number of blocks, and (ii) BIDBs model uncertain attributes, so block size (and hence \( y \)) is a function of the “messiness” of a dataset, rather than its size. Thus, we expect Theorem 4.7 to hold in general.

| Query | CF   | CI   | \( y \) |
|-------|------|------|--------|
| \( Q_1 \) | 46,714 | 46,768 | 0.1%   |
| \( Q_2 \) | 179,917 | 179,917 | 0%     |
| \( Q_3 \) | 11,535 | 11,535 | 0%     |

Figure 5: Number of Cancellations for Queries Over BIDB.

D CIRCUITS

D.1 Representing Polynomials with Circuits

D.1.1 Circuits for query plans. We now formalize circuits and the construction of circuits for \( \mathcal{RA}^+ \) queries. As mentioned earlier, we represent linear equations as arithmetic circuits over \( \mathbb{N} \)-valued variables with +, \( \times \). A circuit for query \( Q \) and \( \mathbb{N}[X] \)-encoded PDB \( \mathcal{D}_Q[X] \) is a directed acyclic graph \( \langle \mathcal{V}_Q, \mathcal{E}_Q, \mathcal{D}_Q[X], \phi_Q, \mathcal{D}_Q[X], \iota_Q, \mathcal{D}_Q[X] \rangle \) with vertices \( \mathcal{V}_Q, \mathcal{D}_Q[X] \) and directed edges \( \mathcal{E}_Q, \phi_Q [X], \iota_Q [X] \) such that \( \mathcal{V}_Q, \mathcal{D}_Q[X] \subset \mathcal{V}_Q [X] \). The sink function \( \phi_Q [X] : \mathbb{N} [X] \rightarrow \mathcal{V}_Q \) is a partial function that maps the tuples of the \( n \)-ary relation \( Q(\mathcal{D}_Q[X]) \) to vertices. We require that \( \phi_Q[X] \)'s range be limited to sink vertices (i.e., vertices with out-degree 0). A function \( \iota_Q, \mathcal{D}_Q[X] : \mathbb{V}_Q \mathcal{D}_Q[X] \rightarrow \{+,-,\times\} \cup \mathbb{N} \cup X \) assigns a label to each node: Source nodes (i.e., vertices with in-degree 0) are labeled with constants or variables (i.e., \( \mathbb{N} \cup X \)), while the remaining nodes are labeled with the symbol + or \( \times \). We require that vertices have an in-degree of at most two. Note that we can construct circuits for BIDBs in time linear in the time required for deterministic query processing over a possible world of the BIDB under the aforementioned assumption that \( |\mathcal{D}_Q[X]| \leq c \cdot |\mathcal{D}| \).

D.2 Modeling Circuit Construction

We now connect the size of a circuit (where the size of a circuit is the number of vertices in the corresponding DAG) for a given \( \mathcal{RA}^+ \) query \( Q \) and \( \mathbb{N}[X] \)-encoded PDB \( \mathcal{D}_Q[X] \) to the runtime \( T_{det} \) of the PDB's deterministic bounding database \( D \). We do this formally by showing that the size of the circuit is asymptotically no worse than the corresponding runtime of a large class of deterministic query processing algorithms.

Each vertex \( v \in \mathcal{V}_Q \mathcal{D}_Q[X] \) in the arithmetic circuit for

\[ \langle \mathcal{V}_Q, \mathcal{E}_Q, \mathcal{D}_Q[X], \phi_Q, \mathcal{D}_Q[X], \iota_Q, \mathcal{D}_Q[X] \rangle \]

For background on \( \mathbb{N}[X] \)-encoded PDBs, see Appendix A.2
encodes a polynomial, realized as

\[
\text{lin}(v) = \begin{cases} 
\sum_{(v',v) \in E_Q,D_N(X)} \text{lin}(v') & \text{if } \ell(v) = + \\
\prod_{(v',v) \in E_Q,D_N(X)} \text{lin}(v') & \text{if } \ell(v) = x \\
\ell(v) & \text{otherwise}
\end{cases}
\]

We define the circuit for a \( R.A^* \) query \( Q \) recursively by cases as follows. In each case, let \( \left\{ V_{Q_1,D_N(X)}, E_{Q_1,D_N(X)}, \Phi_{Q_1,D_N(X)}, I_{Q_1,D_N(X)} \right\} \) denote the circuit for subquery \( Q_1 \). We implicitly include in all circuits a global node \( v_0 \) s.t., \( I_{Q,D_N(X)}(v_0) = 0 \) for any \( Q, D_N(X) \).

Algorithm 4 defines how the circuit for a query result is constructed. Denote the set of active output tuples as \( \text{Dom}(\phi) \). We quickly review the number of vertices emitted in each case.

**Base Relation.** This circuit has \([D.R] \) vertices.

**Selection.** If we assume dead sinks are iteratively garbage collected, this circuit has at most \(|V_{Q_1,D_N(X)}| \) vertices.

**Projection.** This formulation will produce vertices with an in-degree greater than two, a problem that we correct by replacing every vertex with an in-degree over two by an equivalent fan-in two tree. The resulting structure has at most \(|Q_1| - 1 \) new vertices. The corrected circuit thus has at most \(|V_{Q_1,D_N(X)}| + |Q_1| \) vertices.

**Union.** This circuit has \(|V_{Q_1,D_N(X)}| + |V_{Q_2,D_N(X)}| + |Q_1 \cap Q_2| \) vertices.

**\( k \)-ary Join.** As in projection, newly created vertices will have an in-degree of \( k \), and a fan-in two tree is required. There are \(|Q_1 \bowtie \ldots \bowtie Q_k| \) such vertices, so the corrected circuit has \(|V_{Q_1,D_N(X)}| + \ldots + |V_{Q_k,D_N(X)}| + (k-1)|Q_1 \bowtie \ldots \bowtie Q_k| \) vertices.

**D.2.1 Bounding circuit depth.** We first show that the depth of the circuit (Depth; Definition 4.3) is bounded by the size of the query. Denote by \(|Q|\) the number of relational operators in query \( Q \), which recall we assume is a constant.

**Proposition D.1 (Circuit depth is bounded).** Let \( Q \) be a relational query and \( D \) be a deterministic bounding database with \( n \) tuples. There exists a (lineage) circuit \( C^* \) encoding the lineage of all tuples \( t \in Q(D) \) for which \( \text{Depth}(C^*) \leq O(k|Q| \log(n)) \).

**Proof.** We show that the bound of Proposition D.1 holds for the circuit constructed by Algorithm 4. First, observe that Algorithm 4 is (recursively) invoked exactly once for every relational operator or base relation in \( Q \); it thus suffices to show that a call to Algorithm 4 adds at most \( O_k(\log(n)) \) to the depth of a circuit produced by any recursive invocation. Second, observe that modulo the logarithmic fan-in of the projection and join cases, the depth of the output is at most one greater than the depth of any input (or at most 1 in the base case of relation atoms). For the inductive step, the number of in-edges can be no greater than the join width, which itself is bounded by \( k \). The depth thus increases by at most a constant factor of \(|\log(k)| = O_k(1) \). For the projection case, observe that the fan-in is bounded by \( |Q'(D_{Q_1})| \), which is in turn bounded by \( n^k \). The depth increase for any projection node is thus at most \(|\log(n^k)| = O(k \log(n)) \), as desired. \( \square \)

**D.2.2 Circuit size vs. runtime.**

**Lemma D.2.** Given a \( [N,X] \)-encoded PDB \( D_N(X) \) with deterministic bounding database \( D \), and an \( R.A^* \) query \( Q \), the runtime of \( Q \) over \( D \) has the same or greater complexity as the size of the lineage of \( Q(D_N(X)) \). That is, we have \(|V_{Q,D_N(X)}| \leq kT_{\text{det}}(Q,D,c) + 1\), where \( k \geq 1 \) is the maximal degree of any polynomial in \( Q(D_N(X)) \).

**Proof.** We prove by induction that \(|V_{Q,D_N(X)} \setminus \{v_0\}| \leq kT_{\text{det}}(Q,D,c)\). For clarity, we implicitly exclude \( v_0 \) in the proof below.

The base case is a base relation: \( Q = R \) and is trivially true since \(|V_R,D_N(X)| = |D.R| = T_{\text{det}}(R,D,c)\) (note that here the degree \( k = 1 \)). For the inductive step, we assume that we have circuits for subqueries \( Q_1, \ldots, Q_m \) such that \(|V_{Q_i,D_N(X)}| \leq k_iT_{\text{det}}(Q_i,D,c)\) where \( k_i \) is the degree of \( Q_i \).

**Selection.** Assume that \( Q = \sigma_{Q_1}(Q) \). In the circuit for \( Q, |V_{Q,D_N(X)}| = |V_{Q_1,D}| \) vertices, so from the inductive assumption and \( T_{\text{det}}(Q,D,c) = T_{\text{det}}(Q_1,D,c) \) by definition, we have \(|V_{Q,D_N(X)}| \leq kT_{\text{det}}(Q,D,c)\).

**Projection.** Assume that \( Q = \pi_{A}(Q_1) \). The circuit for \( Q \) has at most \(|V_{Q_1,D_N(X)}| + |Q_1| \) vertices.

\(|V_{Q,D_N(X)}| \leq |V_{Q_1,D_N(X)}| + |Q_1|\)

(by definition of \( T_{\text{det}}(Q,D,c)\))

\(|V_{Q,D_N(X)}| \leq kT_{\text{det}}(Q_1,D,c) + |Q_1|\)

**Union.** Assume that \( Q = Q_1 \cup Q_2 \). The circuit for \( Q \) has \(|V_{Q_1,D_N(X)}| + |V_{Q_2,D_N(X)}| + |Q_1 \cap Q_2| \) vertices.

\(|V_{Q,D_N(X)}| \leq |V_{Q_1,D_N(X)}| + |V_{Q_2,D_N(X)}| + |Q_1| + |Q_2|\)

(by definition of \( T_{\text{det}}(Q,D,c)\))
Algorithm 4 LINELAGECIRCUIT (Q, D, E, V, ℓ)

Input: Q: query
Input: D: a deterministic bounding database
Input: E, V, ℓ: accumulators for the edge list, vertex list, and vertex label list.
Output: C = (V, E, φ, ℓ): a circuit encoding the lineage of each tuple in Q(D)

1: if Q is R then
2:   for t ∈ D.R do
3:     V ← V ∪ {uℓ}; ℓ ← ℓ ∪ \{(uℓ, R(t))\}  \rightarrow Allocate a fresh node uℓ; note that when R(t) = c · X₁ for c > 1, we assume the algorithm generates a 3 node circuit encoding the multiplication of c · X₁, adding the new vertices, edges, and vertex/label pairs to their respective sets.
4:     φ(t) ← uℓ
5:   end for
6:   return (V, E, φ, ℓ)

7: else if Q is σ₁(Q₁) then
8:   (V, E, φ′, ℓ) ← LINELAGECIRCUIT(Q₁, D, V, E, ℓ)
9:   for t ∈ DOM(φ′) do
10:     if ∃(t) then φ′(t) ← φ′(t) else φ′(t) ← u₀
11:   end for
12:   return (V, E, φ, ℓ)

13: else if Q is π₁(Q₁) then
14:   (V, E, φ′, ℓ) ← LINELAGECIRCUIT(Q₁, D, V, E, ℓ)
15:   for t ∈ π₁(Q₁(D)) do
16:     V ← V ∪ {uℓ}; ℓ ← ℓ ∪ \{(uℓ, +)\}  \rightarrow Allocate a fresh node uℓ
17:     φ(t) ← uℓ
18:   end for
19:   for t ∈ Q₁(D) do
20:     E ← E ∪ \{(φ′(t), π₁(t))\}
21:   end for
22:   Correct nodes with in-degrees > 2 by appending an equivalent fan-in two tree instead
23:   return (V, E, φ, ℓ)

24: else if Q is Q₁ ∪ Q₂ then
25:   (V', E', φ₁, ℓ') ← LINELAGECIRCUIT(Q₁, D, V, E, ℓ)
26:   (V, E, φ₂, ℓ) ← LINELAGECIRCUIT(Q₂, D, V', E', ℓ')
27:   φ ← φ₁ ∪ φ₂
28:   for t ∈ DOM(φ₁) ∩ DOM(φ₂) do
29:     V ← V ∪ {uℓ}; ℓ ← ℓ ∪ \{(uℓ, +)\}  \rightarrow Allocate a fresh node uℓ
30:     φ(t) ← uℓ
31:   end for
32:   return (V, E, φ, ℓ)

33: else if Q is Q₁ ≃ ... ≃ Qₘ then
34:   for i ∈ [m] do
35:     (V, E, φᵢ, ℓ) ← LINELAGECIRCUIT(Qᵢ, D, V, E, ℓ)
36:   end for
37:   for t ∈ DOM(φ₁) ≃ ... ≃ DOM(φₘ) do
38:     V ← V ∪ {uℓ}; ℓ ← ℓ ∪ \{(uℓ, X)\}  \rightarrow Allocate a fresh node uℓ
39:     φ(t) ← uℓ
40:     E ← E ∪ \{(φᵢ(πᵢ(Qᵢ(D))(t)), uᵢ) | i ∈ [m] \}
41:   end for
42:   Correct nodes with in-degrees > 2 by appending an equivalent fan-in two tree instead
43:   return (V, E, φ, ℓ)
44: end if
(From the inductive assumption)
\[ \leq k(T_{det}(Q_1, D, c) + T_{det}(Q_2, D, c)) + (|Q_1| + |Q_2|) \]

(By definition of \( T_{det}(Q, D, c) \))
\[ \leq k(T_{det}(Q, D, c)). \]

**m-ary Join.** Assume that \( Q = Q_1 \bowtie \ldots \bowtie Q_m \). Note that \( k = \sum_{i=1}^{m} k_i \geq m \). The circuit for \( Q \) has \(|V_{Q_1,D_{ni}[X]}| + \ldots + |V_{Q_k,D_{ni}[X]}| + (m-1)|Q_1 \bowtie \ldots \bowtie Q_k| \)
vertices.
\[ |V_{Q,D_{ni}[X]}| = |V_{Q_1,D_{ni}[X]}| + \ldots + |V_{Q_k,D_{ni}[X]}| + (m-1)|Q_1 \bowtie \ldots \bowtie Q_k| \]
From the inductive assumption and noting \( \forall i : k_i \leq k \) and \( m \leq k \)
\[ \leq kT_{det}(Q_1, D, c) + \ldots + kT_{det}(Q_k, D, c) + (m-1)|Q_1 \bowtie \ldots \bowtie Q_m| \]
\[ \leq k(T_{det}(Q_1, D, c) + \ldots + T_{det}(Q_1, D, c) + |Q_1 \bowtie \ldots \bowtie Q_m|) \]
(By definition of \( T_{det}(Q, D, c) \) and assumption on \( T_{join}(\cdot) \))
\[ \leq kT_{det}(Q, D, c). \]

The property holds for all recursive queries, and the proof holds. \( \square \)

### D.2.3 Runtime of LineageCircuit.
We next need to show that we can construct the circuit in time linear in the deterministic runtime.

**Lemma D.3.** Given a query \( Q \) over a deterministic bounding database \( D \) and the \( C^* \) output by Algorithm 4, the runtime \( T_{LineageCircuit}(Q, D, C^*) \leq O(T_{det}(Q, D, c)) \).

**Proof.** By analysis of Algorithm 4, invoked as \( C^* \leftarrow LineageCircuit(Q, D, \emptyset, \{v_0\}, \{(v_0, 0)\}) \).
We assume that the vertex list \( V \), edge list \( E \), and vertex label list \( \ell \) are mutable accumulators with \( O(1) \) ammortized append. We assume that the tuple to sink mapping \( \phi \) is a linked hash map, with \( O(1) \) insertions and retrievals, and \( O(n) \) iteration over the domain of keys. We assume that the \( m \)-ary join \( \text{Dom}(\phi_1) \bowtie \ldots \bowtie \text{Dom}(\phi_n) \) can be computed in time \( T_{join}(\text{Dom}(\phi_1), \ldots, \text{Dom}(\phi_n)) \) (Definition 2.10) and that an intersection \( \text{Dom}(\phi_1) \cap \text{Dom}(\phi_2) \) can be computed in time \( O(|\text{Dom}(\phi_1)| + |\text{Dom}(\phi_2)|) \) (e.g., with a hash table).

Before proving our runtime bound, we first observe that \( T_{det}(Q, D, c) \geq \Omega(|Q(D)|) \). This is true by construction for the relation, projection, and union cases, by Definition 2.10 for joins, and by the observation that \( |\sigma(R)| \leq |R| \).

We show that \( T_{det}(Q, D, c) \) is an upper-bound for the runtime of Algorithm 4 by recursion. The base case of a relation atom requires only an \( O(|D.R)| \) iteration over the source tuples. For the remaining cases, we make the recursive assumption that for every subquery \( Q' \), it holds that \( O(T_{det}(Q', D, c)) \) bounds the runtime of Algorithm 4.

**Selection.** Selection requires a recursive call to Algorithm 4, which by the recursive assumption is bounded by \( O(T_{det}(Q', D, c)) \). Algorithm 4 requires a loop over every element of \( Q'(D) \). By the observation above that \( T_{det}(Q, D, c) \geq \Omega(|Q(D)|) \), this iteration is also bounded by \( O(T_{det}(Q', D, c)) \).

**Projection.** Projection requires a recursive call to Algorithm 4, which by the recursive assumption is bounded by \( O(T_{det}(Q', D)) \), which in turn is a term in \( T_{det}(\pi_AQ', D, c) \). What remains is an iteration over \( \pi_A(Q(D)) \) (lines 13–16), an iteration over \( Q'(D) \) (lines 17–19), and the construction of a fan-in tree (line 20). The first iteration is \( O(|Q(D)|) \leq O(T_{det}(Q, D, c)) \). The second iteration and the construction of the bounded fan-in tree are both \( O(|Q'(D)|) \leq O(T_{det}(Q', D, c)) \), by the observation above that \( T_{det}(Q, D, c) \geq \Omega(|Q(D)|) \).

**Bag.** As above, the recursive calls explicitly correspond to terms in the expansion of \( T_{det}(Q_1 \cup Q_2, D, c) \). Initializing \( \phi \) (Line 27) can be accomplished in \( O(\text{Dom}(\phi_1) + \text{Dom}(\phi_2)) = O(|Q_1(D)| + |Q_2(D)|) \leq O(T_{det}(Q_1, D) + T_{det}(Q_2, D, c)) \). The remainder requires computing \( Q_1 \cap Q_2 \) (Line 28) and iterating over it (Lines 29 to 31), which is \( O(|Q_1| + |Q_2|) \) as noted above — this directly corresponds to terms in \( T_{det}(Q_1 \cup Q_2, D, c) \).

**m-ary Join.** As in the prior cases, recursive calls explicitly correspond to terms in our target runtime. The remaining logic involves (i) computing \( \text{Dom}(\phi_1) \bowtie \ldots \bowtie \text{Dom}(\phi_m) \), (ii) iterating over the results, and (iii) creating a fan-in tree. Respectively, these are:
(i) \( T_{join}(\text{Dom}(\phi_1), \ldots, \text{Dom}(\phi_m)) \)
(ii) \( O(|Q(D)| \bowtie \ldots \bowtie M(D)) \leq O(T_{join}(\text{Dom}(\phi_1), \ldots, \text{Dom}(\phi_m))) \) (Definition 2.10)
(iii) \( O(|Q(D)| \bowtie \ldots \bowtie M(D)) \) (as (ii), noting that \( m \leq k = O(1) \)) \( \square \)
E HIGHER MOMENTS
We make a simple observation to conclude the presentation of our results. So far we have only focused on the expectation of \( \Phi \). In addition, we could e.g. prove bounds of the probability of a tuple’s multiplicity being at least 1. Progress can be made on this as follows: For any positive integer \( m \) we can compute the \( m \)-th moment of the multiplicities, allowing us to e.g. use the Chebyshev inequality or other high moment based probability bounds on the events we might be interested in. We leave further investigations for future work.

F THE KARP-LUBY ESTIMATOR
Computing the marginal probability of a tuple in the output of a set-probabilistic database query has been studied extensively. To the best of our knowledge, the current state of the art approximation algorithm for this problem is the Karp-Luby estimator [34], which first appeared in MayBMS/Sprout [41], and more recently as part of an online “anytime” approximation algorithm [16, 21].

The estimator works by observing that for any \( \ell \) random binary (but not necessarily independent) events \( W_1, \ldots, W_\ell \), the probability of at least one event occurring (i.e., \( \Pr (W_1 \lor \ldots \lor W_\ell) \)) is bounded from above by the sum of the independent event probabilities (i.e., \( \Pr (W_1 \lor \ldots \lor W_\ell) \leq \Pr (W_1) + \ldots + \Pr (W_\ell) \)). Starting from this (easily computable and large) value, the estimator proceeds to correct the estimate by estimating how much of an over-estimate it is. Specifically, if \( P \) is the joint distribution over \( W \), the estimator computes an approximation of:

\[
O = \mathbb{E}_{W \sim P} \left[ | \{ i \mid W_i = 1, i \in [\ell] \} | \right].
\]

The accuracy of this estimate is improved by conditioning \( P \) on a \( W_i \) chosen uniformly at random (which ensures that the sampled count will be at least 1) and correcting the resulting estimate by \( \Pr (W_i) \). With an estimate of \( O \), it can easily be verified that the probability of the disjunction can be computed as:

\[
\Pr (W_1 \lor \ldots \lor W_\ell) = \Pr (W_1) + \ldots + \Pr (W_\ell) - O
\]

The Karp-Luby estimator is employed on the SMB representation\(^\text{15}\) of \( C \) (to solve the set-PDB version of Problem 1.6), where each \( W_i \) represents the event that one monomial is true. By simple inspection, if there are \( \ell \) monomials, this estimator has runtime \( \Omega (\ell) \). Further, a minimum of \( \frac{3 \cdot \ell \cdot \log (|C|)}{\varepsilon^2} \) invocations of the estimator are required to achieve \( 1 \pm \varepsilon \) approximation with probability at least \( 1 - \delta \) [41], entailing a runtime at least quadratic in \( \ell \). As an arbitrary lineage circuit \( C \) may encode \( \Omega \left( |C|^k \right) \) monomials, the worst case runtime is at least \( \Omega \left( |C|^{2k} \right) \) (where \( k \) is the ’degree’ of lineage polynomial encoded by \( C \)). By contrast note that by the discussion after Lemma C.11 we can solve Problem 1.6 in time \( O \left( |C|^2 \right) \) for all BIDB circuits independent of the degree \( k \).

G PARAMETERIZED COMPLEXITY
In Sec. 3, we utilized common conjectures from fine-grained complexity theory. The notion of \( \#W[1] \)-hard is a standard notion in parameterized complexity, which by now is a standard complexity tool in providing data complexity bounds on query processing results [24]. E.g. the fact that \( k \)-matching is \( \#W[1] \)-hard implies that we cannot have an \( n^{O(1)} \) runtime. However, these results do not carefully track the exponent in the hardness result. E.g. \( \#W[1] \)-hard for the general \( k \)-matching problem does not imply anything specific for the \( 3 \)-matching problem. Similar questions have led to intense research into the new sub-field of fine-grained complexity (see [52]), where we care about the exponent in our hardness assumptions as well– e.g. Conjecture 3.3 is based on the popular Triangle detection hypothesis in this area (cf. [37]).

\(^{15}\)Note that since we are in the set semantics, in the lineage polynomial/formula, addition is logical OR and multiplication is logical AND.