Standard Operating Procedure in Bag PDB Queries Considered Harmful

Su Feng ✉
Illinois Institute of Technology, Chicago, USA

Boris Glavic ✉
Illinois Institute of Technology, USA

Aaron Huber ✉
University at Buffalo, USA

Oliver Kennedy ✉
University at Buffalo, USA

Atri Rudra ✉
University at Buffalo, USA

Abstract

The problem of computing the marginal probability of a tuple in the result of a query over set-probabilistic databases (PDBs) can be reduced to calculating the probability of the *lineage formula* of the result, a Boolean formula over random variables representing the existence of tuples in the database's possible worlds. The analog for bag semantics is a natural number-valued polynomial over random variables that evaluates to the multiplicity of the tuple in each world. In this work, we study the problem of calculating the expectation of such polynomials (a tuple's expected multiplicity) exactly and approximately. For tuple-independent databases (TIDBs), the expected multiplicity of a query result tuple can trivially be computed in linear time in the size of the tuple's lineage, if this polynomial is encoded as a sum of products (the standard operating procedure for Set-PDBs). However, using a reduction from the problem of counting k-matchings, we demonstrate that calculating the expectation is #W[1]-hard when the polynomial is compressed, for example through factorization. Such factorized representations are exploited by modern join algorithms (e.g., worst-case optimal joins), and so our results imply that computing probabilities for Bag-PDB based on the results produced by such algorithms introduces super-linear overhead. The problem stays hard even for polynomials generated by conjunctive queries (CQs) if all input tuples have a fixed probability p (s.t. $p \in (0, 1)$). We proceed to study polynomials of result tuples of union of conjunctive queries (UCQs) over TIDBs and for a non-trivial subclass of block-independent databases (BIDBs). We develop a sampling algorithm that computes a $1 \pm \epsilon$-approximation of the expectation of polynomial circuits in linear time in the size of the polynomial. By removing Bag-PDB's reliance on the sum-of-products representation of polynomials, this result paves the way for future work on PDBs that are competitive with deterministic databases.

2012 ACM Subject Classification Information systems → Incomplete data

Keywords and phrases PDB, bags, polynomial, boolean formula, etc.

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

A *probabilistic database* $\mathcal{D} = (\Omega, \mathbf{P})$ is set of deterministic databases $\Omega = \{D_1, \ldots, D_n\}$ called possible worlds, paired with a probability distribution $\mathbf{P}$ over these worlds. A well-studied problem in probabilistic databases is to take a query Q and a probabilistic database $\mathcal{D}$, and compute the *marginal probability* of a tuple t (i.e., its probability of appearing in the result of query Q over $\mathcal{D}$). This problem is #P-hard for set semantics, even for *tuple-independent probabilistic databases* [35] (TIDBs), which are a subclass of probabilistic databases where

OnTime	City$_\ell$	Φ	$\mathbf{p}$
	Buffalo	L_a	0.9
	Chicago	L_b	0.5
	Bremen	L_c	0.5
	Zurich	L_d	1.0

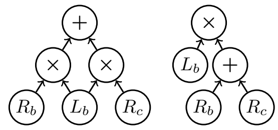
(a) Relation *OnTime*

Route	City$_1$	City$_2$	Φ	$\mathbf{p}$
	Buffalo	Chicago	R_a	1.0
	Chicago	Zurich	R_b	1.0
	Chicago	Bremen	R_c	1.0

(b) Relation *Route*

Q_1	City	Φ	$\mathbb{E}_{\Omega \sim \mathbf{P}}[Q(D)(t)]$
	Buffalo	$L_a \cdot R_a$	0.9
	Chicago	$L_b \cdot R_b + L_b \cdot R_c$	$0.5 \cdot 1.0 + 0.5 \cdot 1.0 = 1.0$

(c) Q_1's Result



(d) Two circuits for $Q_1(Chicago)$

Figure 1 TIDB instance and query results for Example 1.1.

tuples are independent events. The dichotomy of Dalvi and Suciu [10] separates the hard cases, from cases that are in PTIME for unions of conjunctive queries (UCQs). In this work we consider bag semantics, where each tuple is associated with a multiplicity $D_i(t)$ in each possible world D_i and study the analogous problem of computing the expectation of the multiplicity of a query result tuple t (denoted $Q(D)(t)$):

$$\underset{\overline{\mathbf{D}} \sim \mathbf{P}}{\mathbb{E}}[Q(\overline{\mathbf{D}})(t)] = \sum_{D \in \Omega} Q(D)(t) \cdot \mathbf{P}(D) \qquad \textbf{(Expected Result Multiplicity)} \quad (1)$$

▶ **Example 1.1.** *Consider the bag-TIDB relations shown in Fig. 1. We define a TIDB under bag semantics analogously to the set case: each input tuple is associated with a probability of having a multiplicity of one (and otherwise multiplicity zero), and tuples are independent random events. Ignore column Φ for now. In this example, we have shipping routes that are certain (probability 1.0) and information about whether shipping at locations is on time (with a certain probability). Query Q_1, shown below returns starting points of shipping routes where shipment processing is on time.*

$$Q_1(City) :- OnTime(City), Route(City, \_)$$

Fig. 1c shows the possible results of this query. For example, there is a 90% probability there is a single route starting in Buffalo that is on time, and the expected multiplicity of this result tuple is 0.9*. There are two shipping routes starting in Chicago. Since the Chicago location has a 50% probability of being on schedule (we assume that delays are linked), the expected multiplicity of this result tuple is* $0.5 + 0.5 = 1.0$.

A well-known result in probabilistic databases is that under set semantics, the marginal probability of a query result t can be computed based on the tuple's lineage. The lineage of a tuple is a Boolean formula (an element of the semiring PosBool$[\mathbf{X}]$ [19] of positive Boolean expressions) over random variables ($\mathbf{X} = (X_1, \ldots, X_n)$) that encode the existence of input tuples. Each possible world D corresponds to an assignment $\{0,1\}^n$ of the variables in $\mathbf{X}$ to either true (the tuple exists in this world) or false (the tuple does not exist in this world). Importantly, the following holds: if the lineage formula for t evaluates to true under the assignment for a world D, then $t \in Q(D)$. Thus, the marginal probability of tuple t is equal to the probability that its lineage evaluates to true (with respect to the obvious analog of probability distribution $\mathbf{P}$ defined over $\mathbf{X}$).

For bag semantics, the lineage of a tuple is a polynomial over variables $\mathbf{X} = (X_1, \ldots, X_n)$ with coefficients in the set of natural numbers $\mathbb{N}$ (an element of semiring $\mathbb{N}[\mathbf{X}]$). Analogously to sets, evaluating the lineage for t over an assignment corresponding to a possible world yields the multiplicity of the result tuple t in this world. Thus, instead of using Eq. (1) to compute the expected result multiplicity of a tuple t, we can equivalently compute the expectation of the lineage polynomial of t, which for this example we denote as $\Phi^t_{Q,\mathcal{D}}$ or Φ if the parameters are clear from the context[1]. In this work, we study the complexity of computing the expectation of such polynomials encoded as arithmetic circuits.

▶ **Example 1.2.** *Associating a lineage variable with every input tuple as shown in Fig. 1, we can compute the lineage of every result tuple as shown in Fig. 1b. For example, the tuple Chicago is in the result, because L_b joins with both R_b and R_c. Its lineage is $\Phi = L_b{\cdot}R_b{+}L_b{\cdot}R_c$. The expected multiplicity of this result tuple is calculated by summing the multiplicity of the result tuple, weighted by its probability, over all possible worlds. In this example, Φ is a sum of products (SOP), and so we can use linearity of expectation to solve the problem in linear time (in the size of Φ). The expectation of the sum is the sum of the expectations of monomials. The expectation of each monomial is then computed by multiplying the probabilities of the variables (tuples) occurring in the monomial. The expected multiplicity for Chicago is* 1.0*.*

The expected multiplicity of a query result can be computed in linear time (in the size of the result's lineage) if the lineage is in SOP form. However, this need not be true for compressed representations of polynomials, including factorized polynomials or arithmetic circuits. For instance, Fig. 1d shows two circuits encoding the lineage of the result tuple (*Chicago*) from Example 1.2. The left circuit encodes the lineage as a SOP while the right circuit uses distributivity to push the addition gate below the multiplication, resulting in a smaller circuit. Given that there is a large body of work (on, e.g., deterministic bag-relational query processing) that can output such compressed representations [24, 30], an interesting question is whether computing expectations is still in linear time for such compressed representations. If the answer is in the affirmative, then probabilities over bag-PDBs can be computed with linear overhead (in the size of the compressed representation) using any algorithm that computes compressed lineage polynomials. Unfortunately, we prove that this is not the case: computing the expected count of a query result tuple is super-linear under standard complexity assumptions (#W[1]-hard) in the size of a lineage circuit.

Concretely, we make the following contributions: (i) We show that the expected result multiplicity problem (Definition 2.14) for conjunctive queries for bag-$TIDB$s is #W[1]-hard in the size of a lineage circuit by reduction from counting the number of k-matchings over an arbitrary graph; (ii) We present an $(1 \pm \epsilon)$-*multiplicative* approximation algorithm for bag-$TIDB$s and show that for typical database usage patterns (e.g. when the circuit is a tree or is generated by recent worst-case optimal join algorithms or their FAQ followups [24]) its complexity is linear in the size of the compressed lineage encoding; (iii) We generalize the approximation algorithm to bag-$BIDB$s, a more general model of probabilistic data; (iv) We further prove that for $\mathcal{RA}^+$ queries (an equivalently expressive, but factorizable form of UCQs), we can approximate the expected output tuple multiplicities with only $O(\log Z)$ overhead (where Z is the number of output tuples) over the runtime of a broad class of query processing algorithms. We also observe that our results trivially extend to higher moments of the tuple multiplicity (instead of just the expectation).

Overview of our Techniques. All of our results rely on working with a *reduced* form of the lineage polynomial Φ. In fact, it turns out that for the TIDB (and BIDB) case, computing the expected multiplicity is *exactly* the same as evaluating this reduced polynomial over the

[1] In later sections, where we focus on a single lineage polynomial, we will simply refer to $\Phi^t_{Q,\mathcal{D}}$ as Q.

probabilities that define the TIDB/BIDB. Next, we motivate this reduced polynomial by continuing Example 1.1.

Consider the query $Q() :- \mathit{OnTime}(\text{City}), \mathit{Route}(\text{City}, \text{City}'), \mathit{OnTime}(\text{City}')$ over the bag relations of Fig. 1. It can be verified that Φ for Q is $L_aL_b + L_bL_d + L_bL_c$. Now consider the product query $Q^2() :- Q(), Q()$. The lineage polynomial for Q^2 is given by Φ^2:

$$\left(L_aL_b + L_bL_d + L_bL_c\right)^2 = L_a^2L_b^2 + L_b^2L_d^2 + L_b^2L_c^2 + 2L_aL_b^2L_d + 2L_aL_b^2L_c + 2L_b^2L_dL_c.$$

The expectation $\mathbb{E}\left[\Phi^2\right]$ then is:

$$\mathbb{E}\left[L_a\right]\mathbb{E}\left[L_b^2\right] + \mathbb{E}\left[L_b^2\right]\mathbb{E}\left[L_d^2\right] + \mathbb{E}\left[L_b^2\right]\mathbb{E}\left[L_c^2\right] + 2\,\mathbb{E}\left[L_a\right]\mathbb{E}\left[L_b^2\right]\mathbb{E}\left[L_d\right]$$
$$+ 2\,\mathbb{E}\left[L_a\right]\mathbb{E}\left[L_b^2\right]\mathbb{E}\left[L_c\right] + 2\,\mathbb{E}\left[L_b^2\right]\mathbb{E}\left[L_d\right]\mathbb{E}\left[L_c\right]$$

If the domain of a random variable W is $\{0,1\}$, then for any $k > 0$, $\mathbb{E}\left[W^k\right] = \mathbb{E}\left[W\right]$, which means that $\mathbb{E}\left[\Phi^2\right]$ simplifies to:

$$\mathbb{E}\left[L_a^2\right]\mathbb{E}\left[L_b\right]+\mathbb{E}\left[L_b\right]\mathbb{E}\left[L_d\right]+\mathbb{E}\left[L_b\right]\mathbb{E}\left[L_c\right]+2\,\mathbb{E}\left[L_a\right]\mathbb{E}\left[L_b\right]\mathbb{E}\left[L_d\right]+2\,\mathbb{E}\left[L_a\right]\mathbb{E}\left[L_b\right]\mathbb{E}\left[L_c\right]+2\,\mathbb{E}\left[L_b\right]\mathbb{E}\left[L_d\right]\mathbb{E}\left[L_c\right]$$

This property leads us to consider a structure related to the lineage polynomial.

▶ **Definition 1.3.** *For any polynomial $Q(\mathbf{X})$, define the* reduced polynomial $\widetilde{Q}(\mathbf{X})$ *to be the polynomial obtained by setting all exponents $e > 1$ in the SOP form of $Q(\mathbf{X})$ to 1.*

With Φ^2 as an example, we have:

$$\widetilde{\Phi^2}(L_a, L_b, L_c, L_d) = L_aL_b + L_bL_d + L_bW_c + 2L_aL_bL_d + 2L_aL_bL_c + 2L_bL_cL_d$$

It can be verified that the reduced polynomial is a closed form of the expected count (i.e., $\mathbb{E}\left[\Phi^2\right] = \widetilde{\Phi^2}(P\left[L_a = 1\right], P\left[L_b = 1\right], P\left[L_c = 1\right]), P\left[L_d = 1\right]))$. In fact, we show in Lemma 2.8 that this equivalence holds for *all* UCQs over TIDB/BIDB.

To prove our hardness result we show that for the same Q considered in the running example, the query Q^k is able to encode various hard graph-counting problems. We do so by analyzing how the coefficients in the (univariate) polynomial $\widetilde{\Phi}\left(p, \ldots, p\right)$ relate to counts of various sub-graphs on k edges in an arbitrary graph G (which is used to define the relations in Q). For the upper bound it is easy to check that if all the probabilties are constant then $\Phi\left(P\left[X_1 = 1\right], \ldots, P\left[X_n = 1\right]\right)$ (i.e. evaluating the original lineage polynomial over the probability values) is a constant factor approximation. To get an $(1 \pm \epsilon)$-multiplicative approximation we sample monomials from Φ and 'adjust' their contribution to $\widetilde{\Phi}\left(\cdot\right)$.

Paper Organization. We present relevant background and notation in Sec. 2. We then prove our main hardness results in Sec. 3 and present our approximation algorithm in Sec. 4. We present some (easy) generalizations of our results in Sec. 5 and also discuss extensions from computing expectations of polynomials to the expected result multiplicity problem (Definition 2.14). Finally, we discuss related work in Sec. 6 and conclude in Sec. 7.

2 Background and Notation

2.1 Probabilistic Databases (PDBs)

An *incomplete database* Ω is a set of deterministic databases D called possible worlds. Denote the schema of D as $sch(D)$. A *probabilistic database* $\mathcal{D}$ is a pair $(\Omega, \mathbf{P})$ where Ω is an incomplete database and $\mathbf{P}$ is a probability distribution over Ω. Queries over probabilistic databases are evaluated using the so-called possible world semantics. Under the possible

$$\llbracket \pi_A(R) \rrbracket_D(t) = \sum_{t':\pi_A(t')=t} \llbracket R \rrbracket_D(t') \qquad \llbracket (R_1 \cup R_2) \rrbracket_D(t) = \llbracket R_1 \rrbracket_D(t) + \llbracket R_2 \rrbracket_D(t)$$

$$\llbracket \sigma_\theta(R) \rrbracket_D(t) = \begin{cases} \llbracket R \rrbracket_D(t) & \text{if } \theta(t) \\ \mathbb{0}_{\mathcal{K}} & \text{otherwise.} \end{cases} \qquad \llbracket (R_1 \bowtie R_2) \rrbracket_D(t) = \llbracket R_1 \rrbracket_D(\pi_{sch(R_1)}(t)) \cdot \llbracket R_2 \rrbracket_D(\pi_{sch(R_2)}(t))$$

$$\llbracket R \rrbracket_D(t) = R(t)$$

Figure 2 Evaluation semantics $\llbracket \cdot \rrbracket_D$ for $\mathbb{N}[\mathbf{X}]$-DBs [19].

world semantics, the result of a query Q over an incomplete database Ω is the set of query answers produced by evaluating Q over each possible world: $Q(\Omega) = \{\ Q(D) \mid D \in \Omega\ \}$.

For a probabilistic database $\mathcal{D} = (\Omega, \mathbf{P})$, the result of a query is the pair $(Q(\Omega), \mathbf{P}')$ where $\mathbf{P}'$ is a probability distribution over $Q(\Omega)$ that assigns to each possible query result the sum of the probabilities of the worlds that produce this answer:

$$\forall D \in Q(\Omega) : \mathbf{P}'(D) = \sum_{D' \in \Omega : Q(D') = D} \mathbf{P}(D')$$

Let $\mathbb{N}[\mathbf{X}]$ denote the set of polynomials over variables $\mathbf{X} = (X_1, \ldots, X_n)$ with natural number coefficients and exponents. We model incomplete relations using Green et. al.'s $\mathbb{N}[\mathbf{X}]$-databases [19], discussed in detail in Appendix A.1 and summarized here. In an $\mathbb{N}[\mathbf{X}]$-database, relations are defined as functions from tuples to elements of $\mathbb{N}[\mathbf{X}]$, typically called annotations. We write $R(t)$ to denote the polynomial annotating tuple t in relation R. Note that $R(t)$ is the lineage polynomial for t. Each possible world is defined by an assignment of N binary values $\mathbf{W} \in \{0,1\}^{|\mathbf{X}|}$ to $\mathbf{X}$. The multiplicity of $t \in R$ in this possible world, denoted $R(t)(\mathbf{W})$, is obtained by evaluating the polynomial annotating t on $\mathbf{W}$. $\mathbb{N}[\mathbf{X}]$-relations are closed under $\mathcal{RA}^+$ (Fig. 2).

We will use $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$, defined as the tuple $(\Omega_{\mathbb{N}[\mathbf{X}]}, \mathbf{P})$, where $\mathbb{N}[\mathbf{X}]$-database $\Omega_{\mathbb{N}[\mathbf{X}]}$ is paired with probability distribution $\mathbf{P}$. We denote by Q_t the annotation of tuple t in the result of Q on an implicit $\mathbb{N}[\mathbf{X}]$-PDB (i.e., $Q_t = Q(\mathbf{D})(t)$ for some $\mathbf{D}$) and as before, interpret it as a function $Q_t : \{0,1\}^{|\mathbf{X}|} \to \mathbb{N}$ from vectors of variable assignments to the corresponding value of the annotating polynomial. $\mathbb{N}[\mathbf{X}]$-PDBs and a function Mod (which transforms an $\mathbb{N}[\mathbf{X}]$-PDB to classical, or $\mathbb{N}$-PDB [19, 14]) are both formalized in Appendix A.1.

▶ **Proposition 2.1** (Expectation of polynomials). *Given an $\mathbb{N}$-PDB $\mathcal{D} = (\Omega, \mathbf{P})$ and $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D} = (\Omega'_{\mathbb{N}[\mathbf{X}]}, \mathbf{P}')$ where $Mod(\mathbf{D}) = \mathcal{D}$, we have: $\mathbb{E}_{\Omega \sim \mathbf{P}}[Q(\Omega)(t)] = \mathbb{E}_{\mathbf{W} \sim \mathbf{P}'}[Q_t(\mathbf{W})]$.* [2]

A formal proof of Proposition 2.1 is given in Appendix A.3. This proposition shows that computing expected tuple multiplicities is equivalent to computing the expectation of a polynomial (for that tuple) from a probability distribution over all possible assignments of variables in the polynomial to $\{0,1\}$. We focus on this problem from now on, assume an implicit result tuple, and so drop the subscript from Q_t (i.e., Q will denote a polynomial).

[2] Although assumed by most prior work on set-probabilistic databases, e.g., as an obvious consequence of [21]'s Theorem 7.1, we are unaware of any formal proof for bag-probabilistic databases.

2.1.1 TIDBs and BIDBs

In this paper, we focus on two popular forms of PDBs: Block-Independent (BIDB) and Tuple-Independent (TIDB) PDBs. A BIDB $\mathbf{D} = (\Omega_{\mathbb{N}[\mathbf{X}]}, \mathbf{P})$ is an $\mathbb{N}[\mathbf{X}]$-PDB such that (i) every tuple is annotated with either 0 (i.e., the tuple does not exist) or a unique variable X_i and (ii) that the tuples t of $\mathbf{D}$ for which $\mathbf{D}(t) \neq 0$ can be partitioned into a set of blocks such that variables from separate blocks are independent of each other and variables from the same block are disjoint events. In other words, each random variable corresponds to the event of a single tuple's presence. A *TIDB* is a BIDB where each block contains exactly one tuple. Appendix A.2 explains TIDBs and BIDBs in greater detail. In a BIDB (and by extension a TIDB) $\mathbf{D}$, tuples are partitioned into ℓ blocks $b_1, \ldots, b_\ell$ where tuple $t_{i,j} \in b_i$ is associated with a probability $p_{t_{i,j}} = \mathbf{P}[X_{i,j} = 1]$, and is annotated with a unique variable $X_{i,j}$.[3] Because blocks are independent and tuples from the same block are disjoint, the probabilities $p_{t_{i,j}}$ and the blocks induce the probability distribution $\mathbf{P}$ of $\mathbf{D}$. We will write a BIDB-lineage polynomial $Q(\mathbf{X})$ for a BIDB with ℓ blocks as $Q(\mathbf{X}) = Q(X_{1,1}, \ldots, X_{1,|b_1|}, \ldots, X_{\ell,|b_\ell|})$, where $|b_i|$ denotes the size of b_i.[4]

2.2 Reduced Polynomials and Equivalences

We now introduce some terminology and develop a reduced form (a closed form of the polynomial's expectation) for polynomials over probability distributions derived from a BIDB or TIDB. Note that a polynomial over $\mathbf{X} = (X_1, \ldots, X_n)$ is formally defined as:

$$Q(X_1, \ldots, X_n) = \sum_{\mathbf{d}=(d_1,\ldots,d_n)\in\mathbb{N}^n} c_{\mathbf{d}} \cdot \prod_{i=1}^{n} X_i^{d_i}. \tag{2}$$

▶ **Definition 2.2** (Standard Monomial Basis). *From above, the term $\prod_{i=1}^{n} X_i^{d_i}$ is a* monomial. *A polynomial $Q(\mathbf{X})$ is in standard monomial basis (SMB) when we keep only the terms with $c_{\mathbf{i}} \neq 0$ from Eq.* (2)*.*

We consider SMB as the default representation of a polynomial. We use SMB(Q) to denote the SMB form of a polynomial Q.

▶ **Definition 2.3** (Degree). *The degree of polynomial $Q(\mathbf{X})$ is the largest $\sum_{i=1}^{n} d_i$ such that $c_{(d_1,\ldots,d_n)} \neq 0$.*

The degree of the polynomial $X^2 + 2XY + Y^2$ is 2. Product terms in lineage arise only from join operations (Fig. 2), so intuitively, the degree of a lineage polynomial is analogous to the largest number of joins in any clause of the UCQ query that created it. In this paper we consider only finite degree polynomials. We call a polynomial $Q(\mathbf{X})$ a *BIDB-lineage polynomial* (resp., *TIDB-lineage polynomial*, or simply lineage polynomial), if there exists a $\mathcal{RA}^+$ query Q, BIDB $\mathbf{D}$ (TIDB $\mathbf{D}$, or $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$), and tuple t such that $Q(\mathbf{X}) = Q(\mathbf{D})(t)$.

▶ **Definition 2.4** (Modding with a set). *Let S be a* set *of polynomials over $\mathbf{X}$. Then $Q(\mathbf{X})$* mod *S is the polynomial obtained by taking the mod of $Q(\mathbf{X})$ over* all *polynomials in S (order does not matter).*

---

[3] Although only a single independent, $[|b_i|+1]$-valued variable is customarily used per block, we decompose it into $|b_i|$ correlated $\{0,1\}$-valued variables per block that can be used directly in polynomials (without an indicator function). For $t_j \in b_i$, the event $(X_{i,j} = 1)$ corresponds to the event $(X_i = j)$ in the customary annotation scheme.

[4] Later on in the paper, especially in Sec. 4, we will overload notation and rename the variables as $X_1, \ldots, X_n$, where $n = \sum_{i=1}^{\ell} |b_i|$.

For example for a set of polynomials $S = \left\{X^2 - X, Y^2 - Y\right\}$, taking the polynomial $2X^2 + 3XY - 2Y^2 \mod S$ yields $2X + 3XY - 2Y$.

▶ **Definition 2.5** ($\mathcal{B}$, $\mathcal{T}$)**.** *Given the set of BIDB variables $\{X_{i,j}\}$, define*

$$\mathcal{B} = \left\{\ X_{i,j} \cdot X_{i,j'} \mid i \in [\ell], j \neq j' \in [\ |b_i|\ ]\ \right\} \qquad \mathcal{T} = \left\{\ X_{i,j}^2 - X_{i,j} \mid i \in [\ell], j \in [\ |b_i|\ ]\ \right\}$$

▶ **Definition 2.6** (Reduced BIDB Polynomials)**.** *Let $Q(\mathbf{X})$ be a BIDB-lineage polynomial. The reduced form $\widetilde{Q}(\mathbf{X})$ of $Q(\mathbf{X})$ is: $\widetilde{Q}(\mathbf{X}) = Q(\mathbf{X}) \mod (\mathcal{T} \cup \mathcal{B})$*

All exponents $e > 1$ in $\text{SMB}(Q(\mathbf{X}))$ are reduced to $e = 1$ via mod $\mathcal{T}$. Performing the modulus of $\widetilde{Q}(\mathbf{X})$ with $\mathcal{B}$ ensures the disjoint condition of BIDB, removing monomials with lineage variables from the same block. For the special case of TIDBs, the second step is not necessary since every block contains a single tuple.

▶ **Definition 2.7** (Valid Worlds)**.** *For probability distribution $\mathbf{P}$, the set of valid worlds η consists of all the worlds with probability value greater than 0; i.e., for variable vector $\mathbf{W}$*

$$\eta = \{\ \mathbf{w} \mid P[\mathbf{W} = \mathbf{w}] > 0\ \}$$

Next, we show why the reduced form is useful for our purposes:

▶ **Lemma 2.8.** *Let $\mathbf{D}$ be a BIDB over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ and with probability distribution $\mathbf{P}$ produced by the tuple probability vector $\mathbf{p} = (p_1, \ldots, p_n)$ over all $\mathbf{w}$ in η. For any BIDB-lineage polynomial $Q(\mathbf{X})$ based on $\mathbf{D}$ and query Q we have:*

$$\mathop{\mathbb{E}}_{\mathbf{W} \sim \mathbf{P}}\left[Q(\mathbf{W})\right] = \widetilde{Q}(\mathbf{p}).$$

Note that in the preceding lemma, we have assigned $\mathbf{p}$ to the variables $\mathbf{X}$. Intuitively, Lemma 2.8 states that when we replace each variable X_i with its probability p_i in the reduced form of a BIDB-lineage polynomial and evaluate the resulting expression in $\mathbb{R}$, then the result is the expectation of the polynomial.

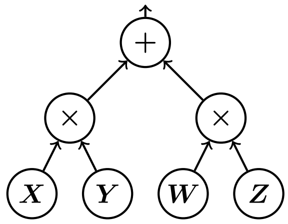
▶ **Corollary 2.9.** *If Q is a BIDB-lineage polynomial, then the expectation of Q, i.e., $\mathbb{E}\left[Q\right] = \widetilde{Q}\left(p_1, \ldots, p_n\right)$ can be computed in $O(\textsc{size}\left(SMB(Q)\right))$, where $\textsc{size}\left(Q\right)$ (Definition 4.4) is proportional to the total number of multiplication/addition operators in Q.*

2.3 Problem Definition

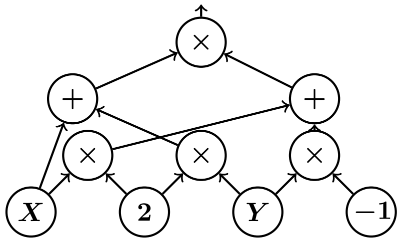
We first formally define circuits, an encoding of polynomials that we use throughout the paper. Since we are particularly using *lineage* circuits, we drop the term lineage and only refer to them as circuits. For illustrative purposes consider the polynomial $Q(\mathbf{X}) = 2X^2 + 3XY - 2Y^2$ over $\mathbf{X} = [X, Y]$.

We represent query polynomials via *arithmetic circuits* [6], a standard way to represent polynomials over fields (particularly in the field of algebraic complexity) that we use for polynomials over $\mathbb{N}$ in the obvious way.

▶ **Definition 2.10** (Circuit)**.** *A circuit $\mathtt{C}$ is a Directed Acyclic Graph (DAG) whose source nodes (in degree of 0) consist of elements in either $\mathbb{R}$ or $\mathbf{X}$. The internal nodes and (the single) sink node of $\mathtt{C}$ (corresponding to the result tuple t) have binary input and are either sum ($+$) or product ($\times$) gates. Each node in a circuit $\mathtt{C}$ has the following members: `type`, `val`, `partial`, `input`, `degree` and `Lweight`, `Rweight`, where `type` is the type of value stored in the node (one of $\{+, \times, \textsc{var}, \textsc{num}\}$), `val` is the value stored (a constant or variable), and `input` is the list of the nodes inputs. We use $\mathtt{C}_L$ to denote the left input and $\mathtt{C}_R$ the right input*



(a) Circuit encoding $XY + WZ$, a special case of an expression tree



(b) Circuit encoding of $(X + 2Y)(2X - Y)$

Figure 3 Example circuit encodings

or the sink of circuit $\texttt{C}$. *When the underlying DAG is a tree (with edges pointing towards the root), we will refer to the structure as an expression tree* $\texttt{T}$. *Note that in such a case, the root of* $\texttt{T}$ *is analogous to the sink of* $\texttt{C}$.

As stated in Definition 2.10, every internal node has at most two in-edges, is labeled as an addition or a multiplication node, and has no limit on its outdegree. Note that if we limit the outdegree to one, then we get expression trees. We ignore the fields `partial`, `Lweight`, and `Rweight` until Sec. 4.

▶ **Example 2.11.** *The circuit* $\texttt{C}$ *in Fig. 3a encodes the polynomial* $XY + WZ$. *Note that circuit* $\texttt{C}$ *encodes a tree, with edges pointing towards the root.*

The semantics of circuits follows the obvious interpretation. We next define its relationship with polynomials formally:

▶ **Definition 2.12** ($\textsc{poly}(\cdot)$). *Denote* $\textsc{poly}(\texttt{C})$ *to be the function from circuit* $\texttt{C}$ *to its corresponding polynomial.* $\textsc{poly}(\cdot)$ *is recursively defined on* $\texttt{C}$ *as follows, with addition and multiplication following the standard interpretation for polynomials:*

$$\textsc{poly}(\texttt{C}) = \begin{cases} \textsc{poly}(\texttt{C}_\texttt{L}) + \textsc{poly}(\texttt{C}_\texttt{R}) & \text{if } \texttt{C.type} = + \\ \textsc{poly}(\texttt{C}_\texttt{L}) \cdot \textsc{poly}(\texttt{C}_\texttt{R}) & \text{if } \texttt{C.type} = \times \\ \texttt{C.val} & \text{if } \texttt{C.type} = \textsc{var} \text{ OR } \textsc{num}. \end{cases}$$

Note that $\texttt{C}$ need not encode an expression in SMB. For instance, $\texttt{C}$ could represent a compressed form of the running example, such as $(X + 2Y)(2X - Y)$, as shown in Fig. 3b, while $\textsc{poly}(\texttt{C}) = 2X^2 + 3XY - 2Y^2$.

▶ **Definition 2.13** (Circuit Set). $\texttt{CSet}(Q(\mathbf{X}))$ *is the set of all possible circuits* $\texttt{C}$ *such that* $\textsc{poly}(\texttt{C}) = Q(\mathbf{X})$.

The circuit of Fig. 3b is an element of $\texttt{CSet}(2X^2 + 3XY - 2Y^2)$. One can think of $\texttt{CSet}(Q(\mathbf{X}))$ as the infinite set of circuits each of which model an encoding (factorization) equal to $\textsc{poly}(\texttt{C})$. Note that Definition 2.13 implies that $\texttt{C} \in \texttt{CSet}(\textsc{poly}(\texttt{C}))$.

We are now ready to formally state our **main problem**.

▶ **Definition 2.14** (The Expected Result Multiplicity Problem). *Let* $\mathbf{X} = (X_1, \ldots, X_n)$, *and* $\mathcal{D}$ *be an* $\mathbb{N}[\mathbf{X}]$*-PDB over* $\mathbf{X}$ *with probability distribution* $\mathbf{P}$ *over assignments* $\mathbf{X} \to \{0, 1\}$, Q *an* n*-ary query, and* t *an* n*-ary tuple. The* Expected Result Multiplicity Problem *is defined as follows:*

Input: *A circuit* $\texttt{C} \in \texttt{CSet}(Q(\mathbf{X}))$ *for* $Q(\mathbf{X}) = Q(\mathbf{D})(t)$ ***Output:*** $\mathbb{E}_{\mathbf{W} \sim \mathbf{P}}[Q(\mathbf{W})]$

3 Hardness of exact computation

In this section, we will prove that computing $\mathop{\mathbb{E}}_{\mathbf{W}\sim\mathbf{P}}\left[Q(\mathbf{W})\right]$ exactly for a TIDB-lineage polynomial $Q(\mathbf{X})$ generated from a project-join query (even an expression tree representation) is #W[1]-hard. Note that this implies hardness for BIDBs and general $\mathbb{N}[\mathbf{X}]$-PDBs under bag semantics. Furthermore, we demonstrate in Sec. 3.3 that the problem remains hard, even if $P[X_i = 1] = p$ for all X_i and any fixed valued $p \in (0, 1)$ as long as certain popular hardness conjectures in fine-grained complexity hold.

3.1 Preliminaries

Our hardness results are based on (exactly) counting the number of occurrences of a subgraph H in G. Let $\#(G, H)$ denote the number of occurrences of H in graph G. We can think of H as being of constant size and G as growing. In particular, we will consider the problems of computing the following counts (given G as an input and its adjacency list representation): $\#(G, \triangle)$ (the number of triangles), $\#(G, {:}{:}{:})$ (the number of 3-matchings), and the latter's generalization $\#\left(G, {:}\cdots{:}^{k}\right)$ (the number of k-matchings). Our hardness result in Sec. 3.2 is based on the following result:

▶ **Theorem 3.1** ([8]). *Given positive integer k and undirected graph G with no self-loops or parallel edges, computing $\#\left(G, {:}\cdots{:}^{k}\right)$ exactly is #W[1]-hard (parameterization is in k).*

The above result means that we cannot hope to count the number of k-matchings in $G = (V, E)$ in time $f(k) \cdot |V|^c$ for any function f and constant c independent of k. In fact, all known algorithms to solve this problem take time $|V|^{\Omega(k)}$. Our hardness result in Section 3.3 is based on the following conjectured hardness result:

▶ **Conjecture 3.2.** *There exists a constant $\epsilon_0 > 0$ such that given an undirected graph $G = (V, E)$, computing exactly $\#(G, \triangle)$ cannot be done in time $o\left(|E|^{1+\epsilon_0}\right)$.*

Based on the so called *Triangle detection hypothesis* (cf. [25]), which states that detection of whether G has a triangle or not takes time $\Omega\left(|E|^{4/3}\right)$, implies that in Conjecture 3.2 we can take $\epsilon_0 \geq \frac{1}{3}$.

Both of our hardness results rely on a simple query 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 query polynomial has a variable X_i for every i in $[n]$. Consider the polynomial $Q_G(\mathbf{X}) = \sum_{(i,j)\in E} X_i \cdot X_j$. The hard polynomial for our problem will be a suitable power $k \geq 3$ of the polynomial above:

▶ **Definition 3.3.** *For any graph $G = ([n], E)$ and $k \geq 1$, define*

$$Q_G^k(X_1, \ldots, X_n) = \left(\sum_{(i,j)\in E} X_i \cdot X_j\right)^k$$

Our hardness results only need a TIDB instance; We also consider the special case when all the tuple probabilities (probabilities assigned to X_i by $\mathbf{p}$) are the same value. Note that our hardness results even hold for the expression trees.
Returning to Fig. 1, it is easy to see that $Q_G^k(\mathbf{X})$ generalizes our running example query:

$$Q_G^k :- OnTime(C_1), Route(C_1, C_1'), OnTime(C_1'), \ldots, OnTime(C_k), Route(C_k, C_k'), OnTime(C_k')$$

where adapting the PDB instance in Fig. 1, relation *OnTime* has n tuples corresponding to each vertex in $V = [n]$ each with probability p and *Route*$(\text{City}_1, \text{City}_2)$ has tuples corresponding to the edges E (each with probability of 1).[5] Note that this implies that our hard query polynomial can be represented as an expression tree produced by a project-join query with same probability value for each input tuple p_i.

3.2 Multiple Distinct p Values

We are now ready to present our main hardness result.

▶ **Theorem 3.4.** *Computing $\widetilde{Q}^k_G(p_i, \ldots, p_i)$ for arbitrary G and any $(2k+1)$ distinct values p_i $(0 \leq i \leq 2k)$ is #W[1]-hard (parameterization is in k).*

We will prove the above result by reducing from the problem of computing the number of k-matchings in G. Given the current best-known algorithm for this counting problem, our results imply that unless the state-of-the-art k-matching algorithms are improved, we cannot hope to solve our problem in time better than $\Omega_k\left(m^{k/2}\right)$ where $m = |E|$, which is only quadratically faster than expanding $Q^k_G(\mathbf{X})$ into its SMB form and then using Corollary 2.9. By contrast the approximation algorithm we present in Sec. 4 has runtime $O_k(m)$ for this query.
The following lemma reduces the problem of counting k-matchings in a graph to our problem (and proves Theorem 3.4):

▶ **Lemma 3.5.** *Let $p_0, \ldots, p_{2k}$ be distinct values in $(0, 1]$. Then given the values $\widetilde{Q}^k_G(p_i, \ldots, p_i)$ for $0 \leq i \leq 2k$, the number of k-matchings in G can be computed in $O\left(k^3\right)$ time.*

3.3 Single p value

While Theorem 3.4 shows that computing $\widetilde{Q}(p, \ldots, p)$ in general is hard it does not rule out the possibility that one can compute this value exactly for a *fixed* value of p. Indeed, it is easy to check that one can compute $\widetilde{Q}(p, \ldots, p)$ exactly in linear time for $p \in \{0, 1\}$. In this section, we show that these two are the only possibilities:

▶ **Theorem 3.6.** *Fix $p \in (0, 1)$. Then assuming Conjecture 3.2 is true, any algorithm that computes $\widetilde{Q}^3_G(p, \ldots, p)$ from G exactly has to run in time $\Omega\left(m^{1+\epsilon_0}\right)$, where ϵ_0 is as defined in Conjecture 3.2.*

The above shows the hardness for a very specific query polynomial but it is easy to come up with an infinite family of hard query polynomials by 'embedding' $\widetilde{Q}^3_G$ into an infinite family of trivial query polynomials. Unlike Theorem 3.4 the above result does not show that computing $\widetilde{Q}^3_G(p, \ldots, p)$ for a fixed $p \in (0, 1)$ is #W[1]-hard. However, in Sec. 4 we show that if we are willing to compute an approximation that this problem (and indeed solving our problem for a much more general setting) is in linear time.

We will prove the above result by the following reduction:

▶ **Theorem 3.7.** *Fix $p \in (0, 1)$. Let G be a graph on m edges. If we can compute $\widetilde{Q}^3_G(p, \ldots, p)$ exactly in $T(m)$ time, then we can exactly compute $\#\left(G, \right)$ in $O\left(T(m) + m\right)$ time.*

[5] Technically, $Q^k_G(\mathbf{X})$ should have variables corresponding to tuples in *Route* as well, but since they always are present with probability 1, we drop those. Our argument also works when all the tuples in *Route* also are present with probability p but to simplify notation we assign probability 1 to edges.

The following result immediately implies Theorem 3.7:

▶ **Lemma 3.8.** *Fix $p \in (0,1)$. Given $\widetilde{Q}^3_{G^{(\ell)}}(p,\ldots,p)$ for $\ell \in [2]$, we can compute in $O(m)$ time a vector $\mathbf{b} \in \mathbb{R}^3$ such that*

$$\begin{pmatrix} 1-3p & -(3p^2-p^3) \\ 10(3p^2-p^3) & 10(3p^2-p^3) \end{pmatrix} \cdot \begin{pmatrix} \#\left(G, \;\right)] \\ \#\left(G, \;\right) \end{pmatrix} = \mathbf{b},$$

allowing us to compute $\#\left(G, \;\right)$ and $\#\left(G, \;\right)$ in $O(1)$ time.

4 $1 \pm \epsilon$ Approximation Algorithm

In Sec. 3, we showed that computing the expected multiplicity of a compressed lineage polynomial for TIDB (even just based on project-join queries), and by extension BIDB (or any $\mathbb{N}[\mathbf{X}]$-PDB) is unlikely to be possible in linear time (Theorem 3.4), even if all tuples have the same probability (Theorem 3.6). Given this, we now design an approximation algorithm for our problem that runs in *linear time*.[6] The folowing approximation algorithm applies to BIDB, though our bounds are more meaningful for a non-trivial subclass of BIDBs that contains both TIDBs, as well as the PDBench benchmark [1].

4.1 Preliminaries and some more notation

We now introduce useful definitions and notation related to circuits and polynomials. All proofs and missing pseudocode can be found in Appendix C.

▶ **Definition 4.1** (Variables in a monomial)**.** *Given a monomial v, we use $\textsc{var}(v)$ to denote the set of variables in v.*

For example the monomial XY has $\textsc{var}(XY) = \{X, Y\}$.

▶ **Definition 4.2** ($\mathtt{E(C)}$)**.** *The logical view of $E(C)$ is a list of tuples (v, c), where v is a set of variables and c is in $\mathbb{R}$. $E(C)$ has the following recursive definition ($\circ$ is list concatenation).*

$$E(C) = \begin{cases} E(C_L) \circ E(C_R) & \textbf{\textit{if }} C.\mathtt{type} = + \\ \{(v_L \cup v_R, c_L \cdot c_R) \mid (v_L, c_L) \in E(C_L), (v_R, c_R) \in E(C_R)\} & \textbf{\textit{if }} C.\mathtt{type} = \times \\ \mathtt{List}\,[(\emptyset, C.\mathtt{val})] & \textbf{\textit{if }} C.\mathtt{type} = \textsc{num} \\ \mathtt{List}\,[(\{C.\mathtt{val}\}, 1)] & \textbf{\textit{if }} C.\mathtt{type} = \textsc{var}. \end{cases}$$

For further explanation, please refer to Example C.2.

▶ **Definition 4.3** ($|\mathtt{C}|(\mathbf{X})$)**.** *For any circuit C, the corresponding* positive circuit*, denoted $|C|$, is obtained from C as follows. For each leaf node ℓ of C where $\ell.\mathtt{type}$ is $\textsc{num}$, update $\ell.\mathtt{value}$ to $|\ell.\mathtt{value}|$.*

Please see Example C.3 for an illustration.

▶ **Definition 4.4** ($\textsc{size}(\cdot)$)**.** *The function $\textsc{size}$ takes a circuit C as input and outputs the number of gates (nodes) in C.*

▶ **Definition 4.5** ($\textsc{depth}(\cdot)$)**.** *The function $\textsc{depth}$ has circuit C as input and outputs the number of levels in C.*

[6] For a very broad class of circuits: please see the discussion after Lemma 4.11 for more.

▶ **Definition 4.6** ($\text{DEG}(\cdot)$). [7] $\textit{DEG}(C)$ *is defined recursively as follows:*

$$\textit{DEG}(C) = \begin{cases} \max(\textit{DEG}(C_L), \textit{DEG}(C_R)) & \textit{if } C.\textit{type} = + \\ \textit{DEG}(C_L) + \textit{DEG}(C_R) + 1 & \textit{if } C.\textit{type} = \times \\ 0 & \textit{otherwise.} \end{cases}$$

Finally, we will need the following notation for the complexity of multiplying large integers:

▶ **Definition 4.7** ($\overline{\mathcal{M}}(\cdot,\cdot)$). [8] *In a RAM model of word size of W-bits, $\overline{\mathcal{M}}(M, W)$ denotes the complexity of multiplying two integers represented with M-bits. (We will assume that for input of size N, $W = O(\log N)$.*

4.2 Our main result

▶ **Theorem 4.8.** *Let C be a circuit for a UCQ over BIDB and define $Q(\mathbf{X}) = \textit{POLY}(C)$ and let $k = \textit{DEG}(C)$. Then an estimate $\mathcal{E}$ of $\widetilde{Q}(p_1, \ldots, p_n)$ can be computed in time*

$$O\left(\left(\textit{SIZE}(C) + \frac{\log\frac{1}{\delta} \cdot |C|^2(1,\ldots,1) \cdot k \cdot \log k \cdot \textit{DEPTH}(C))}{(\epsilon')^2 \cdot \widetilde{Q}^2(p_1,\ldots,p_n)}\right) \cdot \overline{\mathcal{M}}\left(\log\left(|C|(1,\ldots,1)\right), \log\left(\textit{SIZE}(C)\right)\right)\right)$$

such that

$$P\left(\left|\mathcal{E} - \widetilde{Q}(p_1,\ldots,p_n)\right| > \epsilon' \cdot \widetilde{Q}(p_1,\ldots,p_n)\right) \leq \delta. \tag{3}$$

To get linear runtime results from Theorem 4.8, we will need to define another parameter modeling the (weighted) number of monomials in $\mathtt{E}(\mathtt{C})$ to be 'canceled' when it is modded with $\mathcal{B}$ (Definition 2.5).

▶ **Definition 4.9** (Parameter γ). *Given an expression tree C, define*

$$\gamma(C) = \frac{\sum_{(v,c)\in E(C)} |c| \cdot \mathbb{1}\left(v \mod \mathcal{B} \equiv 0\right)}{|C|(1,\ldots,1)}$$

We next present a few corollaries of Theorem 4.8.

▶ **Corollary 4.10.** *Let $Q(\mathbf{X})$ be as in Theorem 4.8 and let $\gamma = \gamma(C)$. Further let it be the case that $p_i \geq p_0$ for all $i \in [n]$. Then an estimate $\mathcal{E}$ of $\widetilde{Q}(p_1, \ldots, p_n)$ satisfying Eq. (3) can be computed in time*

$$O\left(\left(\textit{SIZE}(C) + \frac{\log\frac{1}{\delta} \cdot k \cdot \log k \cdot \textit{DEPTH}(C))}{(\epsilon')^2 \cdot (1-\gamma)^2 \cdot p_0^{2k}}\right) \cdot \overline{\mathcal{M}}\left(\log\left(|C|(1,\ldots,1)\right), \log\left(\textit{SIZE}(C)\right)\right)\right)$$

In particular, if $p_0 > 0$ and $\gamma < 1$ are absolute constants then the above runtime simplifies to $O_k\left(\left(\frac{1}{(\epsilon')^2} \cdot \textit{SIZE}(C) \cdot \log\frac{1}{\delta}\right) \cdot \overline{\mathcal{M}}\left(\log\left(|C|(1,\ldots,1)\right), \log\left(\textit{SIZE}(C)\right)\right)\right)$.

The restriction on γ is satisfied by any TIDB (where $\gamma = 0$) as well as for all three queries of the PDBench BIDB benchmark (see Appendix C.11 for experimental results).

Finally, we address the $\overline{\mathcal{M}}\left(\log\left(|\mathtt{C}|(1,\ldots,1)\right), \log\left(\text{SIZE}(\mathtt{C})\right)\right)$ term in the runtime.

---

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

[8] We note that when doing arithmetic operations on the RAM model for input of size N, we have that $\overline{\mathcal{M}}(O(\log N), O(\log N)) = O(1)$. More generally we have $\overline{\mathcal{M}}(N, O(\log N)) = O(N \log N \log\log N)$.

Algorithm 1 ApproximateQ̃$(\mathtt{C}, \mathbf{p}, \delta, \epsilon)$

Input: $\mathtt{C}$: Circuit
Input: $\mathbf{p} = (p_1, \ldots, p_n) \in [0,1]^N$
Input: $\delta \in [0,1]$
Input: $\epsilon \in [0,1]$
Output: $\mathtt{acc} \in \mathbb{R}$

$\mathtt{acc} \leftarrow 0$
$\mathtt{N} \leftarrow \left\lceil \frac{2\log\frac{2}{\delta}}{\epsilon^2} \right\rceil$
$(\mathtt{C_{mod}}, \mathtt{size}) \leftarrow$ OnePass $(\mathtt{C})$ ▷ OnePass is Algorithm 2
for $\mathtt{i} \in 1$ to $\mathtt{N}$ **do** ▷ Perform the required number of samples
  $(\mathtt{M}, \mathtt{sgn_i}) \leftarrow$ SampleMonomial $(\mathtt{C_{mod}})$ ▷ SampleMonomial is Algorithm 3. Note that $\mathtt{sgn_i}$ is the *sign* of the monomial's coefficient and *not* the coefficient itself
  if $\mathtt{M}$ has at most one variable from each block **then**
    $\mathtt{Y_i} \leftarrow \prod_{X_j \in \text{VAR}(\mathtt{M})} p_j$
    $\mathtt{Y_i} \leftarrow \mathtt{Y_i} \times \mathtt{sgn_i}$
    $\mathtt{acc} \leftarrow \mathtt{acc} + \mathtt{Y_i}$ ▷ Store the sum over all samples
  end if
end for
$\mathtt{acc} \leftarrow \mathtt{acc} \times \frac{\mathtt{size}}{\mathtt{N}}$
return $\mathtt{acc}$

▶ **Lemma 4.11.** *For any circuit C with* $\text{DEG}(C) = k$*, we have* $|C|(1, \ldots, 1) \leq 2^{2^k \cdot \text{SIZE}(C)}$*. Further, under either of the following conditions:*

1. *C is a tree,*
2. *C encodes the run of the algorithm in [24] on an FAQ query,*

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 Corollary 4.10, then the runtime there simplies to $O_k\left(\frac{1}{(\epsilon')^2} \cdot \text{SIZE}(\mathtt{C})^2 \cdot \log\frac{1}{\delta}\right)$ for general circuits $\mathtt{C}$ and to $O_k\left(\frac{1}{(\epsilon')^2} \cdot \text{SIZE}(\mathtt{C}) \cdot \log\frac{1}{\delta}\right)$ for the case when $\mathtt{C}$ satisfies the specific conditions in Lemma 4.11. In Appendix C.4 we argue that these conditions are very general and encompass many interesting scenarios, including query evaluation under $\mathcal{RA}^+$ or FAQ.

4.3 Approximating $\widetilde{Q}$

The algorithm (ApproximateQ̃ detailed in Algorithm 1) to prove Theorem 4.8 follows from the following observation. Given a query polynomial $Q(\mathbf{X}) = \text{POLY}(\mathtt{C})$ for circuit $\mathtt{C}$ over $BIDB$, we can exactly represent $\widetilde{Q}(\mathbf{X})$ as follows:

$$\widetilde{Q}(X_1, \ldots, X_n) = \sum_{(\mathtt{v},\mathtt{c}) \in \mathtt{E}(\mathtt{C})} \mathbb{1}(\mathtt{v} \mod \mathcal{B} \not\equiv 0) \cdot \mathtt{c} \cdot \prod_{X_i \in \text{VAR}(\mathtt{v})} X_i \tag{4}$$

Given the above, the algorithm is a sampling based algorithm for the above sum: we sample (via SampleMonomial) $(\mathtt{v},\mathtt{c}) \in \mathtt{E}(\mathtt{C})$ with probability proportional to $|\mathtt{c}|$ and compute $Y = \mathbb{1}(\mathtt{v} \mod \mathcal{B} \not\equiv 0) \cdot \prod_{X_i \in \text{VAR}(\mathtt{v})} p_i$. Taking $\mathtt{N}$ samples and computing the average of Y gives us our final estimate. OnePass is used to compute the sampling probabilities needed in SampleMonomial (details are in Appendix C).

5 More on Circuits and Moments

We formalize our claim from Sec. 1 that a linear approximation algorithm for our problem implies that PDB queries (under bag semantics) can be answered (approximately) in the same runtime as deterministic queries under reasonable assumptions. Lastly, we generalize our result for expectation to other moments.

The cost model. So far our analysis of $\text{Approximate}\widetilde{Q}$ has been in terms of the size of the lineage circuits. We now show that this model corresponds to the behavior of a deterministic database by proving that for any $\mathcal{RA}^+$ query Q, we can construct a compressed circuit for Q and BIDB $\mathbf{D}$ of size (and in runtime) linear in that of a general class of query processing algorithms for the same query Q on a deterministic database D. We assume a linear relationship between input sizes $|\mathbf{D}|$ and $|D|$ (i.e., $\exists c, D \in \mathbf{D}$ s.t. $|\mathbf{D}| \leq c \cdot |D|)$). [9] We adopt a minimalistic compute-bound model of query evaluation drawn from the worst-case optimal join literature [28, 26].

$$\mathbf{cost}(R, D) = |R| \quad \mathbf{cost}(\sigma Q, D) = \mathbf{cost}(Q, D) \quad \mathbf{cost}(\pi Q, D) = \mathbf{cost}(Q, D) + |Q(D)|$$
$$\mathbf{cost}(Q \cup Q', D) = \mathbf{cost}(Q, D) + \mathbf{cost}(Q', D) + |Q(D)| + |Q'(D)|$$
$$\mathbf{cost}(Q_1 \bowtie \ldots \bowtie Q_n, D) = \mathbf{cost}(Q_1, D) + \ldots + \mathbf{cost}(Q_n, D) + |Q_1(D) \bowtie \ldots \bowtie Q_n(D)|$$

Under this model a query Q evaluated over database D has runtime $O(\mathbf{cost}(Q, D))$. We assume that full table scans are used for every base relation access. We can model index scans by treating an index scan query $\sigma_\theta(R)$ as a base relation.

It can be verified that worst-case optimal join algorithms [28, 26], as well as query evaluation via factorized databases [30] (and work on FAQs [24]) can be modeled as select-union-project-join queries (though these queries can be data dependent).[10] Further, it can be verified that the above cost model on the corresponding SPJU join queries correctly captures their runtime.

We are now ready to formally state our claim from Sec. 1:

▶ **Corollary 5.1.** *Given an SPJU query Q over a TIDB $\mathbf{D}$ and let D_{max} denote the world containing all tuples of $\mathbf{D}$, we can compute a $(1 \pm \epsilon)$-approximation of the expectation for each output tuple in $Q(\mathbf{D})$ with probability at least $1 - \delta$ in time*

$$O_k\left(\frac{1}{\epsilon^2} \cdot \boldsymbol{cost}(Q, D_{max}) \cdot \log\frac{1}{\delta} \cdot \log(n)\right)$$

Proof. This follows from Lemma D.1 (Appendix D.1.2) and Corollary 4.10 (where the latter is used with δ being substituted[11] with $\frac{\delta}{n^k}$).

Higher Moments. We make a simple observation to conclude the presentation of our results. So far we have only focused on the expectation of Q. In addition, we could e.g. prove bounds of probability of the 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,

[9] This is a reasonable assumption because each block of a BIDB represents entities with uncertain attributes. In practice there is often a limited number of alternatives for each block (e.g., which of five conflicting data sources to trust). Note that all TIDBs trivially fulfill this condition (i.e., $c = 1$).

[10] This claim can be verified by e.g. simply looking at the *Generic-Join* algorithm in [28] and *factorize* algorithm in [30].

[11] Recall that Corollary 4.10 is stated for a single output tuple so to get the required guarantee for all (at most n^k) output tuples of Q we get at most $\frac{\delta}{n^k}$ probability of failure for each output tuple and then just a union bound over all output tuples.

allowing us to e.g. use Chebyschev inequality or other high moment based probability bounds on the events we might be interested in. We leave further investigations for future work.

6 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 [36] (by reduction from weighted model counting). The second category, *extensional* query evaluation, is in PTIME, but is limited to certain classes of queries. Dalvi et al. [11] and Olteanu et al. [17] proved dichotomies for UCQs 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 [23, 31, 33]. 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 [37] (TIDBs), block-independent databases (BIDBs) [32], and *PC-tables* [20]. Fink et al. [15] study aggregate queries over a probabilistic version of the extension of K-relations for aggregate queries proposed in [3] (*pvc-tables*). As an extension of K-relations, this approach supports bags. In contrast, we study a less general data model ($\mathbb{N}[\mathbf{X}]$-PDBs) and query class, but provide a linear time approximation algorithm and provide new insights into the complexity of computing expectations while [15] computes probabilities for individual output annotations.

Several techniques for approximating tuple probabilities have been proposed in related work [16, 12, 29, 9], relying on Monte Carlo sampling, e.g., [9], or a branch-and-bound paradigm [29]. Our approximation algorithm is also based on sampling.

Compressed Encodings are used for Boolean formulas (e.g, various types of circuits including OBDDs [22]) and polynomials (e.g., factorizations [30]) some of which have been utilized for probabilistic query processing, e.g., [22]. Compact representations for which probabilities can be computed in linear time include OBDDs, SDDs, d-DNNF, and FBDD. [13] studies circuits for absorptive semirings while [34] studies circuits that include negation (expressed as the monus operation). Algebraic Decision Diagrams [5] (ADDs) generalize BDDs to variables with more than two values. Chen et al. [7] introduced the generalized disjunctive normal form. Appendix E covers more related work on fine-grained complexity.

7 Conclusions and Future Work

We have studied the problem of calculating the expectation of lineage polynomials over BIDBs. This problem has a practical application in probabilistic databases over multisets, where it corresponds to calculating the expected multiplicity of a query result tuple. While the expectation of a polynomial can be calculated in linear time for polynomials in SOP form, the problem is #W[1]-hard for factorized polynomials (proven through a reduction from the problem of counting k-matchings). We prove that it is possible to approximate the expectation of a lineage polynomial in linear time UCQs over TIDBs and BIDBs (under the assumption that there are few cancellations). Interesting directions for future work include development of a dichotomy for bag PDBs and approximations for more general data models.

References

1 pdbench. `http://pdbench.sourceforge.net/`. Accessed: 2020-12-15.
2 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Probabilities and provenance via tree decompositions. *PODS*, 2015.
3 Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.
4 Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data.
5 R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *IEEE CAD*, 1993.
6 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315. Springer, 1997.
7 Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010.
8 Radu Curticapean. Counting matchings of size k is w[1]-hard. In *ICALP*, volume 7965, pages 352–363, 2013.
9 N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB*, 16(4):544, 2007.
10 Nilesh Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.
11 Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *JACM*, 59(6):30, 2012.
12 Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. Anytime approximation in probabilistic databases via scaled dissociations. In *SIGMOD*, pages 1295–1312, 2019.
13 Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for datalog provenance. In *ICDT*, pages 201–212, 2014.
14 Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. Uncertainty annotated databases - a lightweight approach for approximating certain answers. In *SIGMOD*, 2019.
15 Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5):490–501, 2012.
16 Robert Fink, Jiewen Huang, and Dan Olteanu. Anytime approximation in probabilistic databases. *VLDBJ*, 22(6):823–848, 2013.
17 Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *TODS*, 41(1):4:1–4:47, 2016.
18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. `doi:10.1007/3-540-29953-X`.
19 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
20 Todd J Green and Val Tannen. Models for incomplete and probabilistic information. In *EDBT*, pages 278–296. 2006.
21 Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.
22 Abhay Kumar Jha and Dan Suciu. Probabilistic databases with markoviews. *PVLDB*, 5(11):1160–1171, 2012.
23 Batya Kenig, Avigdor Gal, and Ofer Strichman. A new class of lineage expressions over probabilistic databases computable in p-time. In *SUM*, volume 8078, pages 219–232, 2013.
24 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *PODS*, pages 13–28, 2016.
25 Tsvi Kopelowitz and Virginia Vassilevska Williams. Towards optimal set-disjointness and set-intersection data structures. In *ICALP*, volume 168, pages 74:1–74:16, 2020.

26 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*, 2018.

27 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018.

28 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.

29 Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.

30 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.

31 Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, 2011.

32 C. Ré and D. Suciu. Materialized views in probabilistic databases: for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.

33 Prithviraj Sen, Amol Deshpande, and Lise Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.

34 Pierre Senellart. Provenance and probabilities in relational databases. *SIGMOD Record*, 46(4):5–15, 2018.

35 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

36 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

37 Guy Van den Broeck and Dan Suciu. Query processing on probabilistic data: A survey. 2017.

38 Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *SIGACT News*, 49(4):29–35, 2018. `doi:10.1145/3300150.3300158`.

8 Acknowledgements

We thank Virginia Williams for showing us Eq. (17), which greatly simplified our earlier proof of Lemma 3.8, and for graciously allowing us to use it.

A Missing details from Section 2

A.1 $\mathcal{K}$-relations and $\mathbb{N}[\mathbf{X}]$-PDBs

We use K-relations to model bags. A *K-relation* [19] is a relation whose tuples are annotated with elements from a commutative semiring $\mathcal{K} = (K, \oplus_{\mathcal{K}}, \otimes_{\mathcal{K}}, \mathbb{0}_{\mathcal{K}}, \mathbb{1}_{\mathcal{K}})$. A commutative semiring is a structure with a domain K and associative and commutative binary operations $\oplus_{\mathcal{K}}$ and $\otimes_{\mathcal{K}}$ such that $\otimes_{\mathcal{K}}$ distributes over $\oplus_{\mathcal{K}}$, $\mathbb{0}_{\mathcal{K}}$ is the identity of $\oplus_{\mathcal{K}}$, $\mathbb{1}_{\mathcal{K}}$ is the identity of $\otimes_{\mathcal{K}}$, and $\mathbb{0}_{\mathcal{K}}$ annihilates all elements of K when combined by $\otimes_{\mathcal{K}}$. Let $\mathcal{U}$ be a countable domain of values. Formally, an n-ary $\mathcal{K}$-relation over $\mathcal{U}$ is a function $R : \mathcal{U}^n \to K$ with finite support $supp(R) = \{t \mid R(t) \neq \mathbb{0}_{\mathcal{K}}\}$. A $\mathcal{K}$-database is a set of $\mathcal{K}$-relations. It will be convenient to also interpret a $\mathcal{K}$-database as a function from tuples to annotations. Thus, $R(t)$ (resp., $D(t)$) denotes the annotation associated by $\mathcal{K}$-relation R ($\mathcal{K}$-database D) to t.

For completeness, we briefly review the semantics for $\mathcal{RA}^+$ queries over $\mathcal{K}$-relations [19] illustrated in Fig. 2. In Fig. 2, we use $[\![\cdot]\!]_D$ to denote the result of evaluating query Q over $\mathcal{K}$-database D, assume that tuples are of appropriate arity, use $sch(R)$ to denote the attributes of R, and use $\pi_A(t)$ to denote the projection of tuple t on a list of attributes A. Furthermore, $\theta(t)$ denotes the (Boolean) result of evaluating condition θ over t.

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. Specifically, given a probabilistic $\mathbb{N}$-database $\mathcal{D} = (\Omega, \mathbf{P})$, query Q, and possible result tuple t, we use $Q(D)(t)$ for $D \in \Omega$ as input in RHS of Eq. (1) to compute the expected multiplicity of t. Note that the tables of Fig. 1 have an implicit 1 $\mathbb{N}$-valued annotation for each tuple in tables *OnTime* and *Route*. Intuitively, the expectation of $Q(D)(t)$ is the number of duplicates of t we expect to find in result of query Q.

Let $\mathbb{N}[\mathbf{X}]$ denote the set of polynomials over variables $\mathbf{X} = (X_1, \ldots, X_n)$ with natural number coefficients and exponents. Consider now the semiring $(\mathbb{N}[\mathbf{X}], +, \cdot, 0, 1)$ whose domain is $\mathbb{N}[\mathbf{X}]$, with the standard addition and multiplication of polynomials. We will use $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$, defined as the tuple $(\Omega_{\mathbb{N}[\mathbf{X}]}, \mathbf{P})$, where $\mathbb{N}[\mathbf{X}]$-database $\Omega_{\mathbb{N}[\mathbf{X}]}$ is paired with probability distribution $\mathbf{P}$. We denote by Q_t the annotation of tuple t in the result of Q on an implicit $\mathbb{N}[\mathbf{X}]$-PDB (i.e., $Q_t = Q(\mathbf{D})(t)$ for some $\mathbf{D}$) and as before, interpret it as a function $Q_t : \{0,1\}^{|\mathbf{X}|} \to \mathbb{N}$ from vectors of variable assignments to the corresponding value of the annotating polynomial. $\mathbb{N}[\mathbf{X}]$-PDBs and a function *Mod* (which transforms an $\mathbb{N}[\mathbf{X}]$-PDB to an equivalent $\mathbb{N}$-PDB) are both formalized next.

To justify the use of $\mathbb{N}[\mathbf{X}]$-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 $\mathbb{N}$-PDB. For that it will be opportune to define representation systems for $\mathbb{N}$-PDBs.

▶ **Definition A.1** (Representation System). *A representation system for $\mathbb{N}$-PDBs is a tuple $(\mathcal{M}, Mod)$ where $\mathcal{M}$ is a set of representations and Mod associates with each $M \in \mathcal{M}$ an $\mathbb{N}$-PDB $\mathcal{D}$. We say that a representation system is* closed *under a class of queries $\mathcal{Q}$ if for any query $Q \in \mathcal{Q}$ we have:*

$$Mod(Q(M)) = Q(Mod(M))$$

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

$$Mod(M) = \mathcal{D}$$

As mentioned above we will use $\mathbb{N}[\mathbf{X}]$-databases paired with a probability distribution as a representation system. We refer to such databases as $\mathbb{N}[\mathbf{X}]$-PDBs and use bold symbols to distinguish them from possible worlds (which are $\mathbb{N}$-databases). Formally, an $\mathbb{N}[\mathbf{X}]$-PDB is an $\mathbb{N}[\mathbf{X}]$-database $\Omega_{\mathbb{N}[\mathbf{X}]}$ and a probability distribution $\mathbf{P}$ over assignments φ of the variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ occurring in annotations of $\Omega_{\mathbb{N}[\mathbf{X}]}$ to $\{0,1\}$. Note that an assignment $\varphi : \mathbf{X} \rightarrow \{0,1\}^n$ can be represented as a vector $\mathbf{w} \in \{0,1\}^n$ where $\mathbf{w}[i]$ records the value assigned to variable X_i. Thus, from now on we will solely use such vectors which we refer to as *world vectors* and implicitly understand them to represent assignments. Given an assignment φ we use $\varphi(\mathbf{D})$ to denote the semiring homomorphism $\mathbb{N}[\mathbf{X}] \rightarrow \mathbb{N}$ that applies the assignment φ to all variables of a polynomial and evaluates the resulting expression in $\mathbb{N}$.

▶ **Definition A.2** ($\mathbb{N}[\mathbf{X}]$-PDBs)**.** *An $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$ over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a tuple $(\Omega_{\mathbb{N}[\mathbf{X}]}, \mathbf{P})$ where D is an $\mathbb{N}[\mathbf{X}]$-database and $\mathbf{P}$ is a probability distribution over $\mathbf{w} \in \{0,1\}^n$. We use $\varphi_{\mathbf{w}}$ to denote the assignment corresponding to $\mathbf{w} \in \{0,1\}^n$. The $\mathbb{N}$-PDB $Mod(\mathbf{D}) = (\Omega, \mathbf{P}')$ encoded by $\mathbf{D}$ is defined as:*

$$\Omega = \{\varphi_{\mathbf{w}}(\mathbf{D}) \mid \mathbf{w} \in \{0,1\}^n\}$$

$$\forall D \in \Omega : P'(D) = \sum_{\mathbf{w} \in \{0,1\}^n : \varphi_{\mathbf{w}}(\mathbf{D}) = D} P(\mathbf{w})$$

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

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

Importantly, as the following proposition shows, any finite $\mathbb{N}$-PDB can be encoded as an $\mathbb{N}[\mathbf{X}]$-PDB and $\mathbb{N}[\mathbf{X}]$-PDBs are closed under positive relational algebra queries, the class of queries we are interested in in this work.

▶ **Proposition A.3.** *$\mathbb{N}[\mathbf{X}]$-PDBs are a complete representation system for $\mathbb{N}$-PDBs that is closed under $\mathcal{RA}^+$ queries.*

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

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

The probability distribution $\mathbf{P}'$ assigns all world vectors zero probability except for $|\Omega|$ world vectors (representing the possible worlds) $\mathbf{w_i}$. All elements of $\mathbf{w_i}$ are zero except for the position corresponding to variables X_i which is set to 1. Unfolding definitions it is trivial to show that $Mod(\mathbf{D}) = \mathcal{D}$. Thus, $\mathbb{N}[\mathbf{X}]$ are a complete representation system. The closure under $\mathcal{RA}^+$ queries follows from the fact that an assignment $\mathbf{X} \rightarrow \{0,1\}$ is a semiring homomorphism and that semiring homomorphisms commute with queries over $\mathcal{K}$-relations.

Now let us consider computing the expected multiplicity of a tuple t in the result of a query Q over an $\mathbb{N}$-PDB $\mathcal{D}$ using the annotation of t in the result of evaluating Q over an

$\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$ for which $Mod(\mathbf{D}) = \mathcal{D}$. The expectation of the polynomial $Q = Q(\mathbf{D})(t)$ based on the probability distribution of $\mathbf{D}$ over the variables in $\mathbf{D}$ is:

$$\mathop{\mathbb{E}}_{\mathbf{W}\sim\mathbf{P}}[Q(\mathbf{W})] = \sum_{\mathbf{w}\in\{0,1\}^n} \varphi_{\mathbf{w}}(Q(\mathbf{D})(t)) \cdot P(\mathbf{w}) \tag{5}$$

Since $\mathbb{N}[\mathbf{X}]$-PDBs $\mathbf{D}$ are a complete representation system for $\mathbb{N}$-PDBs which are closed under $\mathcal{RA}^+$, computing the expectation of the multiplicity of a tuple t in the result of an $\mathcal{RA}^+$ query over the $\mathbb{N}$-PDB $Mod(\mathbf{D})$, is the same as computing the expectation of the polynomial $Q(\mathbf{D})(t)$. ◀

A.2 TIDBs and BIDBs in the $\mathbb{N}[\mathrm{X}]$-PDB model

Two important subclasses of $\mathbb{N}[\mathbf{X}]$-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 this 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 D 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 probabilitites of all the tuples in the same block b is 1. The probability of such a world is the product of the probabilities of all tuples present in the world. For bag TIDBs and BIDBs, we define the probability of a tuple to be the probability that the tuple exists with multiplicity at least 1.

As already noted above, in this work, we define TIDBs and BIDBs as subclasses of $\mathbb{N}[\mathbf{X}]$-PDBs. In this work, we consider 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 $\mathbf{w} \in \{0,1\}^n$ as denoting which tuples exist in the possible world $\varphi_{\mathbf{w}}(\mathbf{D})$ (the ones where $\mathbf{w}[j] = 1$). For BIDBs specifically, note that that at most one of the bits corresponding to tuples in each block will be set (i.e., for any pair of bits w_j, $w_{j'}$ that are part of the same block $b_i \supseteq \{t_{i,j}, t_{i,j'}\}$, at most one of them will be set). Denote the vector $\mathbf{p}$ to be a vector whose elements are the individual probabilities p_i of each tuple t_i. Let $\mathbf{P}^{(\mathbf{p})}$ denote the distribution induced by $\mathbf{p}$.

$$\mathop{\mathbb{E}}_{\mathbf{W}\sim\mathbf{P}^{(\mathbf{p})}}[Q(\mathbf{W})] = \sum_{\substack{\mathbf{w}\in\{0,1\}^n\\ s.t. w_j, w_{j'}=1 \rightarrow \nexists b_i \supseteq \{t_{i,j}, t_{i',j}\}}} Q(\mathbf{w}) \prod_{\substack{j\in[n]\\ s.t. w_j=1}} p_j \prod_{\substack{j\in[n]\\ s.t. w_j=0}} (1-p_i) \tag{6}$$

Recall that tuple blocks in a TIDB always have size 1, so the outer summation of eq. (6) is over the full set of vectors.

A.3 Proof of Proposition 2.1

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

$$\mathop{\mathbb{E}}_{\mathbf{W}\sim\mathbf{P}'}[Q_t(\mathbf{W})] = \sum_{\mathbf{w}\in\{0,1\}^n} P'(\mathbf{w})\cdot Q(\mathbf{D})(t)(\mathbf{w})$$

From $Mod(\mathbf{D}) = \mathcal{D}$, we have that the range of $\varphi_{\mathbf{w}(\mathbf{D})}$ is Ω, so

$$= \sum_{D\in\Omega} \sum_{\mathbf{w}\in\{0,1\}^n:\varphi_{\mathbf{w}}(\mathbf{D})=D} P'(\mathbf{w})\cdot Q(\mathbf{D})(t)(\mathbf{w})$$

In the inner sum, $\varphi_{\mathbf{w}}(\mathbf{D}) = D$, so by distributivity of $+$ over $\times$

$$= \sum_{D\in\Omega} Q(D)(t) \sum_{\mathbf{w}\in\{0,1\}^n:\varphi_{\mathbf{w}}(\mathbf{D})=D} P'(\mathbf{w})$$

From the definition of P, given $Mod(\mathbf{D}) = \mathcal{D}$, we get

$$= \sum_{D\in\Omega} Q(D)(t)\cdot P(D) \quad = \mathop{\mathbb{E}}_{D\sim\mathbf{P}}[Q(D)(t)]$$

◀

A.4 Lemma A.4

▶ **Lemma A.4.** *If* $Q(X_1,\ldots,X_n) = \sum_{\mathbf{d}\in\{0,\ldots,B\}^n} q_{\mathbf{d}}\cdot \prod_{\substack{i=1\\ s.t.d_i\geq 1}}^{n} X_i^{d_i}$ *then* $\widetilde{Q}(X_1,\ldots,X_n) = \sum_{\mathbf{d}\in\eta} q_{\mathbf{d}}\cdot \prod_{\substack{i=1\\ s.t.d_i\geq 1}}^{n} X_i$

Proof. Follows by the construction of $\widetilde{Q}$ in definition 2.6. ◀

A.5 Proposition A.5

Note the following fact:

▶ **Proposition A.5.** *For any BIDB-lineage polynomial* $Q(X_1,\ldots,X_n)$ *and all* $\mathbf{w}\in\eta$*, it holds that* $Q(\mathbf{w}) = \widetilde{Q}(\mathbf{w})$*.*

Proof. Note that any Q 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 $e\geq 1$, $b^e = b$. ◀

A.6 Proof for Lemma 2.8

Proof. Let Q be the generalized polynomial, i.e., the polynomial of n variables with highest degree $= B$:

$$Q(X_1,\ldots,X_n) = \sum_{\mathbf{d}\in\{0,\ldots,B\}^n} q_{\mathbf{d}}\cdot \prod_{\substack{i=1\\ s.t.d_i\geq 1}}^{n} X_i^{d_i}$$

. Then, in expectation we have

$$\mathop{\mathbb{E}}_{\mathbf{W}}[Q(\mathbf{W})] = \sum_{\mathbf{d}\in\eta} q_{\mathbf{d}}\cdot \mathop{\mathbb{E}}_{\mathbf{w}}\left[\prod_{\substack{i=1\\ s.t.d_i\geq 1}}^{n} w_i^{d_i}\right] \tag{7}$$

$$= \sum_{\mathbf{d}\in\eta} q_{\mathbf{d}} \cdot \prod_{\substack{i=1\\ s.t. d_i\geq 1}}^{n} \mathop{\mathbb{E}}_{\mathbf{w}}\left[w_i^{d_i}\right] \tag{8}$$

$$= \sum_{\mathbf{d}\in\eta} q_{\mathbf{d}} \cdot \prod_{\substack{i=1\\ s.t. d_i\geq 1}}^{n} \mathop{\mathbb{E}}_{\mathbf{w}}\left[w_i\right] \tag{9}$$

$$= \sum_{\mathbf{d}\in\eta} q_{\mathbf{d}} \cdot \prod_{\substack{i=1\\ s.t. d_i\geq 1}}^{n} p_i \tag{10}$$

$$= \widetilde{Q}(p_1, \ldots, p_n) \tag{11}$$

In steps eq. (7) and eq. (8), by linearity of expectation (recall the variables are independent, or the monomial expectation is 0), the expecation can be pushed all the way inside of the product. In eq. (9), note that $w_i \in \{0, 1\}$ which further implies that for any exponent $e \geq 1$, $w_i^e = w_i$. Next, in eq. (10) the expectation of a tuple is indeed its probability.

Finally, observe Eq. (11) by construction in Lemma A.4, that $\widetilde{Q}(p_1, \ldots, p_n)$ is exactly the product of probabilities of each variable in each monomial across the entire sum. ◀

A.7 Proof For Corollary 2.9

Proof. Note that lemma 2.8 shows that $\mathbb{E}[Q] = \widetilde{Q}(p_1, \ldots, p_n)$. Therefore, if Q is already in SMB form, one only needs to compute $Q(p_1, \ldots, p_n)$ ignoring exponent terms (note that such a polynomial is $\widetilde{Q}(p_1, \ldots, p_n)$), which indeed has $O(\text{SMB}(|Q|))$ computations. ◀

B Missing details from Section 3

We use Lemma 3.5 to prove Theorem 3.4:

B.1 Proof of Theorem 3.4

Proof. For the sake of contradiction, let us assume we can solve our problem in $f(k) \cdot m^c$ time for some absolute constant c. Then given a graph G we can compute the query polynomial or rather, expression tree representation of $\widetilde{Q}_G^k$ (in the obvious way) in $O(km)$ time. Then after we run our algorithm on $\widetilde{Q}_G^k$, we get $\widetilde{Q}_G^k(p_i, \ldots, p_i)$ for $0 \leq i \leq 2k$ in additional $f(k) \cdot m^c$ time. Lemma 3.5 then computes the number of k-matchings in G in $O(k^3)$ time. Thus, overall we have an algorithm for computing the number of k-matchings in time

$$\begin{aligned} O(km) + f(k) \cdot m^c + O(k^3) &\leq \left(O(k^3) + f(k)\right) \cdot m^{c+1} \\ &\leq \left(O(k^3) + f(k)\right) \cdot n^{2c+2}, \end{aligned}$$

which contradicts Theorem 3.1. ◀

B.2 Proof of Lemma 3.5

Proof. We first argue that $\widetilde{Q}_G^k(p, \ldots, p) = \sum_{i=0}^{2k} c_i \cdot p^i$. First, since $Q_G(\mathbf{X})$ has degree 2, it follows that $Q_G^k(\mathbf{X})$ has degree $2k$. By definition, $\widetilde{Q}_G^k(\mathbf{X})$ sets every exponent $e > 1$ to $e = 1$, which means that $\text{DEG}(\widetilde{Q}_G^k) \leq \text{DEG}(Q_G^k) = 2k$. Thus, if we think of p as a variable, then

$\widetilde{Q}^k_G(p,\ldots,p)$ is a univariate polynomial of degree at most $\textsc{deg}(\widetilde{Q}^k_G) \le 2k$. Thus, we can write

$$\widetilde{Q}^k_G(p,\ldots,p) = \sum_{i=0}^{2k} c_i p^i$$

We note that c_i is *exactly* the number of monomials in the SMB expansion of $Q^k_G(\mathbf{X})$ composed of i distinct variables.[12]

Given that we then have $2k+1$ distinct values of $\widetilde{Q}^k_G(p,\ldots,p)$ for $0 \le i \le 2k$, it follows that we have a linear system of the form $\mathbf{M} \cdot \mathbf{c} = \mathbf{b}$ where the ith row of $\mathbf{M}$ is $\left(p_i^0 \ldots p_i^{2k}\right)$, $\mathbf{c}$ is the coefficient vector $(c_0,\ldots,c_{2k})$, and $\mathbf{b}$ is the vector such that $\mathbf{b}[i] = \widetilde{Q}^k_G(p_i,\ldots,p_i)$. In other words, matrix $\mathbf{M}$ 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 $\mathbf{c}$ exactly. Thus, after $O(k^3)$ work, we know $\mathbf{c}$ and in particular, c_{2k} exactly. Next, we show why we can compute $\#\left(G, \cdots^k\right)$ from c_{2k} in $O(1)$ additional time. We claim that c_{2k} is $k! \cdot \#\left(G, \cdots^k\right)$. This can be seen intuitively by looking at the original factorized representation

$$Q^k_G(\mathbf{X}) = \sum_{(i_1,j_1),\cdots,(i_k,j_k)\in E} X_{i_1}X_{j_1}\cdots X_{i_k}X_{j_k},$$

where across each of the k products, an arbitrary k-matching can be selected $\prod_{i=1}^k i = k!$ times. Indeed, note that each k-matching $(i_1,j_1)\ldots(i_k,j_k)$ in G corresponds to the monomial $\prod_{\ell=1}^k X_{i_\ell}X_{j_\ell}$ in $Q^k_G(\mathbf{X})$, with distinct indexes. Second, the only surviving monomials $\prod_{\ell=1}^k X_{i_\ell}X_{j_\ell}$ of degree exactly $2k$ in $\widetilde{Q}^k_G(\mathbf{X})$ must have that all of $i_1,j_1,\ldots,i_k,j_k$ are distinct in $Q^k_G(\mathbf{X})$. By the last two statements, only monomials composed of $2k$ distinct variables in $Q^k_G(\mathbf{X})$ (and hence of degree $2k$ in $\widetilde{Q}^k_G(\mathbf{X})$) correspond to a k-matching in G.

Notice that each of the $k!$ permutations of an arbitrary monomial maps to the same distinct k-matching in G, and this implies a $k!$ to 1 mapping between degree $2k$ monomials in $\widetilde{Q}^k_G(\mathbf{X})$ and k-matchings in G. It then follows that $c_{2k} = k! \cdot \#\left(G, \cdots^k\right)$. Thus, simply dividing c_{2k} by $k!$ gives us $\#\left(G, \cdots^k\right)$, as needed. ◀

B.3 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 , and , so we define the remaining patterns:

- Single Edge ( )
- 2-path ( )
- 2-matching ( )
- 3-star ( )–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 ( )–this subgraph consists of a two-path and a remaining disjoint edge.

[12] Since $\widetilde{Q}^k_G(\mathbf{X})$ does not have any monomial with degree < 2, it is the case that $c_0 = c_1 = 0$ but for the sake of simplcity we will ignore this observation.

For any graph G, the following formulas for $\#(G, H)$ compute their respective patterns exactly in $O(m)$ time, with d_i representing the degree of vertex i (proofs are in Appendix B.4):

$$\#(G, \;) = m, \tag{12}$$

$$\#(G, \;) = \sum_{i \in V} \binom{d_i}{2} \tag{13}$$

$$\#(G, \;) = \sum_{(i,j) \in E} \frac{m - d_i - d_j + 1}{2} \tag{14}$$

$$\#(G, \;) = \sum_{i \in V} \binom{d_i}{3} \tag{15}$$

$$\#(G, \;) + 3\#(G, \;) = \sum_{(i,j) \in E} \binom{m - d_i - d_j + 1}{2} \tag{16}$$

$$\#(G, \;) + 3\#(G, \;) = \sum_{(i,j) \in E} (d_i - 1) \cdot (d_j - 1) \tag{17}$$

B.4 Proofs of Eq. (12)-Eq. (17)

The proofs for Eq. (12), Eq. (13) and Eq. (15) are immediate.

Proof of Eq. (14). 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. (16). Eq. (16) 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. (14), 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 is necessary to account for the triple counting of 3-matchings. It is also the case that, since the two path in is connected, that there will be no double counting by the fact that the summation automatically disconnects the current edge, meaning that a two matching at the current vertex will not be counted. The sum over all such edge combinations is precisely then $\#(G, \;) + 3\#(G, \;)$. Note that all d_i and $d_i - 3$ factorials can be computed in $O(m)$ time, and then each combination $\binom{n}{3}$ can be performed with constant time operations, yielding the claimed $O(m)$ run time. ◀

Proof of Eq. (17). To compute $\#(G, \;)$, note that for an arbitrary edge (i, j), a 3-path exists for edge pair (i, ℓ) and (j, k) where i, j, k, ℓ 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 $(i, \ell), (j, k)$ 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. (17) by observing that each triangle is counted thrice, while each 3-path is counted just once. For reasons similar to Eq. (14), 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.5 Proof of Theorem 3.6

Proof. For the sake of contradiction, assume that for any G, we can compute $\widetilde{Q}^3_G(p,\ldots,p)$ in $o\left(m^{1+\epsilon_0}\right)$ time. Let G be the input graph. It is easy to see that one can compute the expression tree for $Q^3_G(\mathbf{X})$ in $O(m)$ time. Then by Theorem 3.7 we can compute $\#\left(G, \;\right)$ in further time $o\left(m^{1+\epsilon_0}\right)+O(m)$. Thus, the overall, reduction takes $o\left(m^{1+\epsilon_0}\right)+O(m) = o\left(m^{1+\epsilon_0}\right)$ time, which violates Conjecture 3.2. ◀

B.6 Tools to prove Lemma 3.8

Note that $\widetilde{Q}^3_G(p,\ldots,p)$ as a polynomial in p has degree at most six. Next, we figure out the exact coefficients since this would be useful in our arguments:

▶ **Lemma B.1.** *For any p, we have:*

$$\begin{aligned}\widetilde{Q}^3_G(p,\ldots,p) &= \#\left(G, \;\right)p^2 + 6\#\left(G, \;\right)p^3 + 6\#\left(G, \;\right)p^4 + 6\#\left(G, \;\right)p^3\\ &\quad + 6\#\left(G, \;\right)p^4 + 6\#\left(G, \;\right)p^4 + 6\#\left(G, \;\right)p^5 + 6\#\left(G, \;\right)p^6. \end{aligned} \tag{18}$$

B.6.1 Proof for Lemma B.1

Proof. By definition we have that

$$Q^3_G(\mathbf{X}) = \sum_{(i_1,j_1),(i_2,j_2),(i_3,j_3)\in E}\;\prod_{\ell=1}^{3} X_{i_\ell}X_{j_\ell}.$$

Hence $\widetilde{Q}^3_G(\mathbf{X})$ has degree six. Note that the monomial $\prod_{\ell=1}^{3} X_{i_\ell}X_{j_\ell}$ will contribute to the coefficient of p^ν in $\widetilde{Q}^3_G(\mathbf{X})$, where ν is the number of distinct variables in the monomial. Let $e_1=(i_1,j_1), e_2=(i_2,j_2), e_3=(i_3,j_3)$. We compute $\widetilde{Q}^3_G(\mathbf{X})$ 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). There are exactly $m=\#\left(G, \;\right)$ such triples, each with a p^2 factor in $\widetilde{Q}^3_G(p,\ldots,p)$.

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 occurence when 2 variables in the triple (e_1,e_2,e_3) are bound to e. There are three combinations for this occurrence in $Q^3_G(\mathbf{X})$. Analogusly, there are three such occurrences in $Q^3_G(\mathbf{X})$ 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 $\widetilde{Q}^3_G$. Since $e\neq e'$, this case produces the following edge patterns: , , which contribute $6p^3$ and $6p^4$ respectively to $\widetilde{Q}^3_G(p,\ldots,p)$.

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 $Q^3_G(\mathbf{X})$. This case consists of the following edge patterns: , , , , , which contribute $6p^3, 6p^4, 6p^4, 6p^5$ and $6p^6$ respectively to $\widetilde{Q}^3_G(p,\ldots,p)$. ◀

Since p is fixed, Lemma B.1 gives us one linear equation in $\#\left(G, \;\right)$ and $\#\left(G, \;\right)$ (we can handle the other counts due to equations (12)-(17)). 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.2.** *For $\ell > 1$, let graph $G^{(\ell)}$ be a graph generated from an arbitrary graph $G^{(1)}$, by replacing every edge e of $G^{(1)}$ with a ℓ-path, such that all inner vertexes of an ℓ-path replacement edge are disjoint from the inner vertexes of any other ℓ-path replacement edge.*

Next, we relate the various sub-graph counts in $G^{(2)}$ to $G^{(1)}$ (G).

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

$\#\Big(G^{(2)},$  $\Big) = 8 \cdot \#\Big(G^{(1)},$  $\Big) + 6 \cdot \#\Big(G^{(1)},$  $\Big)$

$+ 4 \cdot \#\Big(G^{(1)},$  $\Big) + 4 \cdot \#\Big(G^{(1)},$  $\Big) + 2 \cdot \#\Big(G^{(1)},$  $\Big).$

▶ **Lemma B.4.** *For $\ell > 1$ and any graph $G^{(\ell)}$, $\#\big(G^{(\ell)},$  $\big) = 0$.*

B.7 Proof of Theorem 3.7

Proof. We can compute $G^{(2)}$ from $G^{(1)}$ in $O(m)$ time. Additionally, if in time $O(T(m))$, we have $\widetilde{Q}^3_{G^{(\ell)}}(p, \ldots, p)$ for $\ell \in [2]$, then the theorem follows by Lemma 3.8. ◀ In other words, if Theorem 3.7 holds, then so must Theorem 3.6.

B.8 Proofs for Lemma B.3, Lemma B.4, and Lemma 3.8

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

▶ **Definition B.5.** *For $\ell > 1$, we use E_ℓ to denote the set of edges in $G^{(\ell)}$. For any graph $G^{(\ell)}$, its edges are denoted by the a pair (e, b), such that $b \in \{0, \ldots, \ell - 1\}$ and $e \in E_1$, where $(e, 0), \ldots, (e, \ell - 1)$ is the ℓ-path that replaces the edge e.*

▶ **Definition B.6** ($E_S^{(\ell)}$)**.** *Given an arbitrary subgraph $S^{(1)}$ of $G^{(1)}$, let $E_S^{(1)}$ denote the set of edges in $S^{(1)}$. Define then $E_S^{(\ell)}$ for $\ell > 1$ as the set of edges in the generated subgraph $S^{(\ell)}$ (i.e. when we apply Definition B.2 to $S^{(1)}$).*

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.7.** *Let $\binom{E}{t}$ denote the set of subsets in E with exactly t edges. In a similar manner, $\binom{E}{\leq t}$ is used to mean the subsets of E with t or fewer edges.*

The following function f_ℓ is a mapping from every 3-edge shape in $G^{(\ell)}$ to its 'projection' in $G^{(1)}$.

▶ **Definition B.8.** *Let $f_\ell : \binom{E_\ell}{3} \mapsto \binom{E_1}{\leq 3}$ be defined as follows. For any element $s \in \binom{E_\ell}{3}$ such that $s = \{(e_1, b_1), (e_2, b_2), (e_3, b_3)\}$, define:*

$$f_\ell\left(\{(e_1, b_1), (e_2, b_2), (e_3, b_3)\}\right) = \{e_1, e_2, e_3\}.$$

▶ **Definition B.9** (f_ℓ^{-1})**.** *For an arbitrary subgraph $S^{(1)}$ of $G^{(1)}$ with at most $m \leq 3$ edges, the inverse function $f_\ell^{-1} : \binom{E_1}{\leq 3} \mapsto 2^{\binom{E_\ell}{3}}$ takes $E_S^{(1)}$ and outputs the set of all elements $s \in \binom{E_S^{(\ell)}}{3}$ such that $f_\ell(s) = E_S^{(1)}$.*

Note, importantly, that when we discuss f_ℓ^{-1}, that each *edge* present in $E_S^{(1)}$ must have an edge in $s \in f_\ell^{-1}(S)$ that projects down to it. In particular, if $|E_S^{(1)}| = 3$, then it must be the case that each $s \in f_\ell^{-1}(S)$ consists of the following set of edges: $\{(e_i, b), (e_j, b'), (e_m, b'')\}$, where i, j and m are distinct.

We first note that f_ℓ is well-defined:

▶ **Lemma B.10.** *f_ℓ is a function.*

Proof. Note that f_ℓ is properly defined. For any $S \in \binom{E_\ell}{3}$, $|f(S)| \leq 3$, since it has to be the case that any subset of 3 edges in E_ℓ will map to at most three edges in E_1. All mappings are in the required range. Then, since for any $b \in \{0, \ldots, \ell - 1\}$ the map $(e, b) \mapsto e$ is a function and has exactly one mapping, which implies that f_ℓ is a function. ◀

We are now ready to prove the structural lemmas. Note that f_ℓ maps subsets of three edges in $G^{(\ell)}$ to a subset of at most three edges in E_1. To prove the structural lemmas, we will use the map f_ℓ^{-1}. In particular, to count the number of occurrences of , , in $G^{(\ell)}$ we count for each $S \in \binom{E_1}{\leq 3}$, how many of // subgraphs appear in $f_\ell^{-1}(S)$.

B.8.1 Proof of Lemma B.3

Proof. For each subset $E_S^{(1)} \in \binom{E_1}{\leq 3}$, we count the number of 3-matchings in the 3-edge subgraphs of $G^{(2)}$ in $f_2^{-1}(E_S^{(1)})$. We first consider the case of $E_S^{(1)} \in \binom{E_1}{3}$, 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 $s \in \{(e_1, 0), (e_1, 1), (e_2, 0), (e_2, 1), (e_3, 0), (e_3, 1)\}$ such that $f_\ell(s) = \{e_1, e_2, e_3\}$.

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

- 3-matching ( )

When $S^{(1)}$ is isomorphic to , it is the case that edges in $E_S^{(2)}$ are *not* disjoint only for the pairs $(e_i, 0), (e_i, 1)$ for $i \in \{1, 2, 3\}$. All choices for $b_1, b_2, b_3 \in \{0, 1\}$, $(e_1, b_1), (e_2, b_2), (e_3, b_3)$ will compose a 3-matching. One can see that we have a total of two possible choices for b_i for each edge e_i in $G^{(1)}$ yielding $2^3 = 8$ possible 3-matchings in $f_2^{-1}(E_S^{(1)})$.

- Disjoint Two-Path ( )

For $S^{(1)}$ isomorphic to edges e_2, e_3 form a 2-path with e_1 being disjoint. This means that $(e_2, 0), (e_2, 1), (e_3, 0), (e_3, 1)$ form a 4-path while $(e_1, 0), (e_1, 1)$ is its own disjoint 2-path. We can only pick either $(e_1, 0)$ or $(e_1, 1)$ for $f_2^{-1}(E_S^{(1)})$, and then we need to pick a 2-matching from e_2 and e_3. Note that the four path allows there to be 3 possible 2 matchings, specifically,

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

Since these two selections can be made independently, there are $2 \cdot 3 = 6$ *distinct* 3-matchings in $f_2^{-1}(E_S^{(1)})$.

- 3-star ( )

When $S^{(1)}$ is isomorphic to , the inner edges $(e_i, 1)$ of $E_S^{(2)}$ 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 possiblities, based on which inner edge is chosen. Note that if $(e_i, 1)$ 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. Thus, there are four 3-matchings in $f_2^{-1}(E_S^{(1)})$.

- 3-path ( )

When $S^{(1)}$ is isomorphic to 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 $f_2^{-1}(E_S^{(1)})$, we cannot pick both $(e_i, 0)$ and $(e_i, 1)$ or both $(e_i, 1)$ and $(e_j, 0)$ where $j = i+1$. There are four such possibilities: $\{(e_1,0),(e_2,0),(e_3,0)\}$, $\{(e_1,0),(e_2,0),(e_3,1)\}$, $\{(e_1,0),(e_2,1),(e_3,1)\}$, $\{(e_1,1),(e_2,1),(e_3,1)\}$, a total of four 3-matchings in $f_2^{-1}(E_S^{(1)})$.

- Triangle ( )

For $S^{(1)}$ isomorphic to , 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, since they are connected. This rules out both subsets of $(e_1,0),(e_2,0),(e_3,1)$ and $(e_1,0),(e_2,1),(e_3,1)$, yielding two 3-matchings.

Let us now consider when $E_S^{(1)} \in \binom{E_1}{\leq 2}$, i.e. patterns among

- 2-matching ( ), 2-path ( ), 1 edge ( )

When $|E_S^{(1)}| = 2$, we can only pick one from each of two pairs, $\{(e_1,0),(e_1,1)\}$ and $\{(e_2,0),(e_2,1)\}$. This implies that a 3-matching cannot exist in $f_2^{-1}(E_S^{(1)})$. The same argument holds for $|E_S^{(1)}| = 1$, where we can only pick one edge from the pair $\{(e_1,0),(e_1,1)\}$. Trivially, no 3-matching exists in $f_2^{-1}(E_S^{(1)})$.

Observe that all of the arguments above focused solely on the subgraph $S^{(1)}$ is isomorphmic. In other words, all $E_S^{(1)}$ of a given "shape" yield the same number of 3-matchings in $f_2^{-1}(E_S^{(1)})$, and this is why we get the required identity using the above case analysis. ◀

B.8.2 Proof of Lemma B.4

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

B.8.3 Proof of Lemma 3.8

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

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

To prove the first step, we use Lemma B.1 to derive the following equality (dropping the superscript and referring to $G^{(1)}$ as G):

$$\#(G, \;)\, p^2 + 6\#(G, \;)\, p^3 + 6\#(G, \;)\, p^4 + 6\#(G, \;)\, p^3 + 6\#(G, \;)\, p^4 + 6\#(G, \;)\, p^4 + 6\#(G, \;\;)\, p^5 + 6\#(G, \;)\, p^6 = \widetilde{Q}^3_G(p, \ldots, p) \tag{19}$$

$$\#(G, \;) + \#(G, \;)\, p + \#(G, \;\;)\, p^2 + \#(G, \;)\, p^3 = \frac{\widetilde{Q}^3_G(p, \ldots, p)}{6p^3} - \frac{\#(G, \;)}{6p} - \#(G, \;) - \#(G, \;)\, p - \#(G, \;)\, p \tag{20}$$

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

$$\frac{\widetilde{Q}^3_G(p,\ldots,p)}{6p^3} - \frac{\#\left(G, \text{ed}\right)}{6p} - \#\left(G, \text{twopath}\right) - \#\left(G, \text{twodis}\right)p - \#\left(G, \text{threestar}\right)p$$
$$- \left[\#\left(G, \text{ed twopath}\right)p^2 + 3\#\left(G, \text{threedis}\right)p^2\right] - \left[\#\left(G, \text{threepath}\right)p + 3\#\left(G, \text{tri}\right)p\right] \tag{21}$$

Eq. (19) is the result of Lemma B.1. We obtain the remaining equations through standard algebraic manipulations.

Note that the LHS of Eq. (21) is indeed the product $\mathbf{M}[1]\cdot\mathbf{x}[1]$. Further note that this product is equal to the RHS of Eq. (21), where every term is computable in $O(m)$ time (by equations (12)-(17)). We set $\mathbf{b}[1]$ to the RHS of Eq. (21).

We follow the same process in deriving an equality for $G^{(2)}$. Replacing occurrences of G with $G^{(2)}$, we obtain Eq. (21) for $G^{(2)}$. Substituting identities from Lemma B.3 and Lemma B.4 we obtain

$$0 - \left(8\#\left(G, \text{threedis}\right) + 6\#\left(G, \text{ed twopath}\right) + 4\#\left(G, \text{threestar}\right) + 4\#\left(G, \text{threepath}\right) + 2\#\left(G, \text{tri}\right)(3p^2 - p^3)\right) =$$
$$\frac{\widetilde{Q}^3_{G^{(2)}}(p,\ldots,p)}{6p^3} - \frac{\#\left(G^{(2)}, \text{ed}\right)}{6p} - \#\left(G^{(2)}, \text{twopath}\right) - \#\left(G^{(2)}, \text{twodis}\right)p - \#\left(G^{(2)}, \text{threestar}\right)p$$
$$- \left[\#\left(G^{(2)}, \text{ed twopath}\right)p^2 + 3\#\left(G^{(2)}, \text{threedis}\right)p^2\right] - \left[\#\left(G^{(2)}, \text{threepath}\right)p + 3\#\left(G^{(2)}, \text{tri}\right)p\right] \tag{22}$$

$$\left(10\#\left(G, \text{tri}\right) + 10G\,\text{threedis}\right)(3p^2 - p^3) =$$
$$\frac{\widetilde{Q}^3_{G^{(2)}}(p,\ldots,p)}{6p^3} - \frac{\#\left(G^{(2)}, \text{ed}\right)}{6p} - \#\left(G^{(2)}, \text{twopath}\right) - \#\left(G^{(2)}, \text{twodis}\right)p - \#\left(G^{(2)}, \text{threestar}\right)p$$
$$- \left[\#\left(G^{(2)}, \text{threepath}\right)p + 3\#\left(G^{(2)}, \text{tri}\right)p\right] - \left[\#\left(G^{(2)}, \text{ed twopath}\right)p^2 - 3\#\left(G^{(2)}, \text{threedis}\right)p^2\right]$$
$$+ \left(4\#\left(G, \text{threestar}\right) + \left[6\#\left(G, \text{ed twopath}\right) + 18\#\left(G, \text{threedis}\right)\right] + \left[4\#\left(G, \text{threepath}\right) + 12\#\left(G, \text{tri}\right)\right]\right)(3p^2 - p^3) \tag{23}$$

As in the previous equality derivation for G, note that the LHS of Eq. (23) is the same as $\mathbf{M}[2]\cdot\mathbf{x}[2]$. The RHS of Eq. (23) has terms all computable (by equations (12)-(17)) in $O(m)$ time. Setting $\mathbf{b}[2]$ to the RHS then completes the proof of step 1.

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

To show that $\mathbf{M}$ indeed has full rank, we will show that $Det\left(\mathbf{M}\right) \neq 0$ for every $p \in (0,1)$. Let $\mathbf{M} =$

$$\begin{vmatrix} 1-3p & -(3p^2-p^3) \\ 10(3p^2-p^3) & 10(3p^2-p^3) \end{vmatrix} = (1-3p)\cdot 10(3p^2-p^3) + 10(3p^2-p^3)\cdot(3p^2-p^3)$$
$$= 10(3p^2-p^3)\cdot(1-3p+3p^2-p^3) = 10(3p^2-p^3)\cdot(-p^3+3p^2-3p+1)$$
$$= 10p^2(3-p)\cdot(1-p)^3 \tag{24}$$

From Eq. (24) it can easily be seen that the roots of $Det\left(\mathbf{M}\right)$ are $0, 1$, and 3. Hence there are no roots in $(0,1)$ and Lemma 3.8 follows. ◀

C Missing Details from Section 4

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

$$Q(X,Y) = 2X^2 + 3XY - 2Y^2. \tag{25}$$

▶ **Definition C.1** (Pure Expansion). *The pure expansion of a polynomial Q is formed by computing all product of sums occurring in Q, without combining like monomials. The pure expansion of Q generalizes Definition 2.2 by allowing monomials $m_i = m_j$ for $i \neq j$.*

Note that similar in spirit to Definition 2.6, $\mathtt{E}(\mathtt{C})$ Definition 4.2 reduces all variable exponents $e > 1$ to $e = 1$.

In the following, we abuse notation and write $\mathtt{v}$ to denote the monomial obtained as the products of the variables in the set.

▶ **Example C.2** (Example for Definition 4.2). *Consider the factorized representation $(X + 2Y)(2X - Y)$ of the polynomial in Eq. (25). Its circuit $\mathtt{C}$ is illustrated in Fig. 3b. The pure expansion of the product is $2X^2 - XY + 4XY - 2Y^2$ and the $\mathtt{E}(\mathtt{C})$ is $[(X, 2), (XY, -1), (XY, 4), (Y, -2)]$.*

$\mathtt{E}(\mathtt{C})$ effectively[13] encodes the *reduced* form of $\text{POLY}(\mathtt{C})$, decoupling each monomial into a set of variables $\mathtt{v}$ and a real coefficient $\mathtt{c}$. However, unlike the constraint on the input to compute $\widetilde{Q}$, the input circuit $\mathtt{C}$ does not need to be in SMB/SOP form.

▶ **Example C.3** (Example for Definition 4.3). *Using the same factorization from Example C.2, $\text{POLY}(|\mathtt{C}|) = (X + 2Y)(2X + Y) = 2X^2 + XY + 4XY + 2Y^2 = 2X^2 + 5XY + 2Y^2$. Note that this is not the same as the polynomial from Eq. (25).*

▶ **Definition C.4** (Evaluation). *Given a circuit $\mathtt{C}$ and a valuation $\mathbf{a} \in \mathbb{R}^n$, we define the evaluation of $\mathtt{C}$ on $\mathbf{a}$ as $\mathtt{C}(\mathbf{a}) = \text{POLY}(\mathtt{C})(\mathbf{a})$.*

▶ **Definition C.5** (Subcircuit). *A subcircuit of a circuit $\mathtt{C}$ is a circuit $\mathtt{S}$ such that $\mathtt{S}$ is a DAG subgraph of the DAG representing $\mathtt{C}$. The sink of $\mathtt{S}$ has exactly one gate $\mathtt{g}$.*

C.1 Proof of Theorem 4.8

In order to prove Theorem 4.8, we will need to argue the correctness of $\text{APPROXIMATE}\widetilde{Q}$, which relies on the correctness of auxiliary algorithms OnePass and SampleMonomial.

▶ **Lemma C.6.** *The OnePass function completes in time:*

$$O\left(\text{SIZE}(\mathtt{C}) \cdot \overline{\mathcal{M}}\left(\log\left(|\mathtt{C}(1\ldots,1)|\right), \log \text{SIZE}(\mathtt{C})\right)\right)$$

OnePass guarantees two post-conditions: First, for each subcircuit $\mathtt{S}$ of $\mathtt{C}$, we have that $\mathtt{S.partial}$ is set to $|\mathtt{S}|(1,\ldots,1)$. Second, when $\mathtt{S.type} = +$, $\mathtt{S.Lweight} = \frac{|\mathtt{S}_L|(1,\ldots,1)}{|\mathtt{S}|(1,\ldots,1)}$ and likewise for $\mathtt{S.Rweight}$.

To prove correctness of Algorithm 1, we only use the following fact that follows from the above lemma: for the modified circuit ($\mathtt{C}_{\mathtt{mod}}$), $\mathtt{C}_{\mathtt{mod}}.\mathtt{partial} = |\mathtt{C}|(1,\ldots,1)$.

▶ **Lemma C.7.** *The function SampleMonomial completes in time*

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

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

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

[13] The minor difference here is that $\mathtt{E}(\mathtt{C})$ encodes the *reduced* form over the SOP expansion of the compressed representation, as opposed to the SMB representation

▸ **Theorem C.8.** *For any C with* $\text{DEG}(poly(|C|)) = k$*, algorithm 1 outputs an estimate* `acc` *of* $\widetilde{Q}(p_1,\ldots,p_n)$ *such that*

$$P\left(\left|\texttt{acc} - \widetilde{Q}(p_1,\ldots,p_n)\right| > \epsilon \cdot |C|(1,\ldots,1)\right) \leq \delta,$$

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

Before proving Theorem C.8, we use it to argue our main result, Theorem 4.8.

Proof. Set $\mathcal{E} = \textsc{Approximate}\widetilde{Q}(\mathtt{C}, (p_1,\ldots,p_n), \delta, \epsilon')$, where

$$\epsilon' = \epsilon \cdot \frac{\widetilde{Q}(p_1,\ldots,p_n)\cdot(1-\gamma)}{|\mathtt{C}|(1,\ldots,1)},$$

which achieves the claimed accuracy bound on $\mathcal{E}$ due to Theorem C.8.

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

$$\begin{aligned}\frac{1}{(\epsilon')^2}\cdot\log\left(\frac{1}{\delta}\right) &= \frac{\log\frac{1}{\delta}}{\epsilon^2\left(\frac{\widetilde{Q}(p_1,\ldots,p_N)}{|\mathtt{C}|(1,\ldots,1)}\right)^2}\\ &= \frac{\log\frac{1}{\delta}\cdot|\mathtt{C}|^2(1,\ldots,1)}{\epsilon^2\cdot\widetilde{Q}^2(p_1,\ldots,p_n)},\end{aligned}$$

which completes the proof. ◂

We now return to the proof of Theorem C.8:

C.2 Proof of Theorem C.8

Proof. Consider now the random variables $Y_1,\ldots,Y_n$, where each Y_i is the value of $\mathtt{Y_i}$ after Line 8 is executed. In particular, note that we have

$$Y_i = \mathbb{1}\left(\mathtt{v} \mod \mathcal{B} \not\equiv 0\right)\cdot\prod_{X_i\in\text{VAR}(v)} p_i,$$

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

$$\begin{aligned}\mathbb{E}\left[Y_i\right] &= \sum_{(\mathtt{v},\mathtt{c})\in\mathtt{E}(\mathtt{C})}\frac{\mathbb{1}\left(\mathtt{v} \mod \mathcal{B}\not\equiv 0\right)\cdot c\cdot\prod_{X_i\in\text{VAR}(v)}p_i}{|\mathtt{C}|(1,\ldots,1)}\\ &= \frac{\widetilde{Q}(p_1,\ldots,p_n)}{|\mathtt{C}|(1,\ldots,1)},\end{aligned}$$

where in the first equality we use the fact that $\mathtt{sgn_i}\cdot|\mathtt{c}| = \mathtt{c}$ and the second equality follows from Eq. (4) with X_i substituted by p_i.

Let $\overline{\mathbf{Y}} = \frac{1}{N}\sum_{i=1}^{N} Y_i$. It is also true that

$$\mathbb{E}\left[\overline{\mathbf{Y}}\right] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[Y_i\right] = \frac{\widetilde{Q}(p_1,\ldots,p_n)}{|\mathtt{C}|(1,\ldots,1)}.$$

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

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

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

$$P\left(\,\left|\overline{\mathbf{Y}} - \mathbb{E}\left[\overline{\mathbf{Y}}\right]\,\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{2N^2\epsilon^2}{2^2N}\right) = 2\exp\left(-\frac{N\epsilon^2}{2}\right) \leq \delta,$$

where the last inequality follows from our choice of N in Line 2.

For the claimed probability bound of $P\left(\left|\texttt{acc} - \widetilde{Q}(p_1,\dots,p_n)\right| > \epsilon \cdot |\texttt{C}|(1,\dots,1)\right) \leq \delta$, note that in the algorithm, $\texttt{acc}$ is exactly $\overline{\mathbf{Y}} \cdot |\texttt{C}|(1,\dots,1)$. Multiplying the rest of the terms by the same factor yields the said bound.

This concludes the proof for the first claim of theorem C.8. We prove the claim on the runtime next.

Run-time Analysis

The runtime of the algorithm is dominated by Line 3 (which by Lemma C.6 takes time $O\left(\text{SIZE}(\texttt{C}) \cdot \overline{\mathcal{M}}\left(\log\left(|\texttt{C}|^2(1,\dots,1)\right), \log\left(\text{SIZE}(\texttt{C})\right)\right)\right)$) and the N iterations of the loop in Line 4. Each iteration's run time is dominated by the call to Line 5 (which by Lemma C.7 takes $O\left(\log k \cdot k \cdot \text{DEPTH}(\texttt{C}) \cdot \overline{\mathcal{M}}\left(\log\left(|\texttt{C}|^2(1,\dots,1)\right), \log\left(\text{SIZE}(\texttt{C})\right)\right)\right)$) and 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. Adding up all the times discussed here gives us the desired overall runtime. ◀

C.3 Proof of Corollary 4.10

Proof. The result follows by first noting that by definition of γ, we have

$$\widetilde{Q}(1,\dots,1) = (1-\gamma) \cdot |\texttt{C}|(1,\dots,1).$$

Further, since each $p_i \geq p_0$ and $Q(\mathbf{X})$ (and hence $\widetilde{Q}(\mathbf{X})$) has degree at most k, we have that

$$\widetilde{Q}(1,\dots,1) \geq p_0^k \cdot \widetilde{Q}(1,\dots,1).$$

The above two inequalities implies $\widetilde{Q}(1,\dots,1) \geq p_0^k \cdot (1-\gamma) \cdot |\texttt{C}|(1,\dots,1)$. Applying this bound in the runtime bound in Theorem 4.8 gives the first claimed runtime. The final runtime of $O_k\left(\frac{1}{\epsilon^2} \cdot \text{SIZE}(\texttt{C}) \cdot \log\frac{1}{\delta} \cdot \overline{\mathcal{M}}\left(\log\left(|\texttt{C}|^2(1,\dots,1)\right), \log\left(\text{SIZE}(\texttt{C})\right)\right)\right)$ follows by noting that $\text{DEPTH}(\texttt{C}) \leq \text{SIZE}(\texttt{C})$ and absorbing all factors that just depend on k. ◀

C.4 Proof of Lemma 4.11

We will prove Lemma 4.11 by considering the three cases separately. We start by considering the case when $\texttt{C}$ is a tree:

▶ **Lemma C.9.** *Let $\texttt{C}$ be a tree (i.e. the sub-circuits corresponding to two children of a node in $\texttt{C}$ are completely disjoint). Then we have*

$$|\texttt{C}|(1,\dots,1) \leq \left(\mathit{SIZE}(\texttt{C})\right)^{\mathit{DEG}(\texttt{C})+1}.$$

Proof. For notational simplicity define $N = \text{SIZE}(\texttt{C})$ and $k = \text{DEG}(\texttt{C})$. To prove this result, we by prove by induction on $\text{DEPTH}(\texttt{C})$ that $|\texttt{C}|(1,\dots,1) \leq N^{k+1}$. For the base case, we have that $\text{DEPTH}(\texttt{C}) = 0$, and there can only be one node which must contain a coefficient (or constant) of 1. In this case, $|\mathtt{C}|(1,\ldots,1) = 1$, and $\textsc{size}(\mathtt{C}) = 1$, and it is true that $|\mathtt{C}|(1,\ldots,1) = 1 \leq N^{k+1} = 1^1 = 1$.

Assume for $\ell > 0$ an arbitrary circuit $\mathtt{C}$ of $\textsc{depth}(\mathtt{C}) \leq \ell$ that it is true that $|\mathtt{C}|(1,\ldots,1) \leq N^{\deg(\mathtt{C})+1}$.

For the inductive step we consider a circuit $\mathtt{C}$ such that $\textsc{depth}(\mathtt{C}) = \ell + 1$. The sink can only be either a $\times$ or $+$ gate. Consider when sink node is $\times$. Let $k_{\mathtt{L}}, k_{\mathtt{R}}$ denote $\textsc{deg}(\mathtt{C}_{\mathtt{L}})$ and $\textsc{deg}(\mathtt{C}_{\mathtt{R}})$ respectively. Then note that

$$\begin{aligned}
|\mathtt{C}|(1,\ldots,1) &= |\mathtt{C}_{\mathtt{L}}|(1,\ldots,1) \cdot |\mathtt{C}_{\mathtt{R}}|(1,\ldots,1) \\
&\leq (N-1)^{k_{\mathtt{L}}+1} \cdot (N-1)^{k_{\mathtt{R}}+1} \\
&= (N-1)^{k+1} \qquad (26)\\
&\leq N^{k+1}.
\end{aligned}$$

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. (26) follows by nothing that for a $\times$ gate we have $k = k_{\mathtt{L}} + k_{\mathtt{R}} + 1$.

For the case when the sink gate is a $+$ gate, then for $N_{\mathtt{L}} = \textsc{size}(\mathtt{C}_{\mathtt{L}})$ and $N_{\mathtt{R}} = \textsc{size}(\mathtt{C}_{\mathtt{R}})$ we have

$$\begin{aligned}
|\mathtt{C}|(1,\ldots,1) &= |\mathtt{C}_{\mathtt{L}}|(1,\ldots,1) + |\mathtt{C}_{\mathtt{R}}|(1,\ldots,1) \\
&\leq N_{\mathtt{L}}^{k+1} + N_{\mathtt{R}}^{k+1} \\
&\leq (N-1)^{k+1} \qquad (27)\\
&\leq N^{k+1}.
\end{aligned}$$

In the above, the first inequality follows from the inductive hypothesis (and the fact that $k_{\mathtt{L}}, k_{\mathtt{R}} \leq k$). 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 $\mathtt{C}$ is a tree we have $N_{\mathtt{L}} + N_{\mathtt{R}} = N - 1$ and, lastly, the fact that $k \geq 0$. This completes the proof.

The upper bound in Lemma 4.11 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.10.** *Let $\mathtt{C}$ be a (general) circuit. Then we have*

$$|\mathtt{C}|(1,\ldots,1) \leq 2^{2^{\textsc{deg}(\mathtt{C})} \cdot \textsc{size}(\mathtt{C})}.$$

Proof Sketch. We use the same notation as in the proof of Lemma C.9. We will prove by induction on $\textsc{depth}(\mathtt{C})$ that $|\mathtt{C}|(1,\ldots,1) \leq 2^{2^k \cdot N}$. The base case argument is similar to that in the proof of Lemma C.9. In the inductive case we have that $N_{\mathtt{L}}, N_{\mathtt{R}} \leq N - 1$.

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

$$\begin{aligned}
|\mathtt{C}|(1,\ldots,1) &= |\mathtt{C}_{\mathtt{L}}|(1,\ldots,1) \times |\mathtt{C}_{\mathtt{R}}|(1,\ldots,1) \\
&\leq 2^{2^{k_{\mathtt{L}}} \cdot N_{\mathtt{L}}} \times 2^{2^{k_{\mathtt{R}}} \cdot N_{\mathtt{R}}} \\
&\leq 2^{2 \cdot 2^{k-1} \cdot (N-1)} \\
&\leq 2^{2^k N}.
\end{aligned}$$

In the above the first inequality follows from inductive hypothesis while the second inequality follows from the fact that $k_{\mathtt{L}}, k_{\mathtt{R}} \leq k - 1$ and $N_{\mathtt{L}}, N_{\mathtt{R}} \leq N - 1$.

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

$$\begin{aligned}
|\mathtt{C}|\,(1,\ldots,1) &= |\mathtt{C}_\mathtt{L}|\,(1,\ldots,1) + |\mathtt{C}_\mathtt{R}|\,(1,\ldots,1)\\
&\le 2^{2^{k_\mathtt{L}}\cdot N_\mathtt{L}} + 2^{2^{k_\mathtt{R}}\cdot N_\mathtt{R}}\\
&\le 2\cdot 2^{2^{k}(N-1)}\\
&\le 2^{2^{k}N}.
\end{aligned}$$

In the above the first inequality follows from the inductive hypothesis while the second inequality follows from the facts that $k_\mathtt{L}, k_\mathtt{R} \le k$ and $N_\mathtt{L}, N_\mathtt{R} \le N-1$. The final inequality follows from the fact that $k \ge 0$. ◀

Finally, we consider the case when $\mathtt{C}$ encodes the run of the algorithm from [24] on an FAQ query. We cannot handle the full generality of an FAQ query but we can handle an FAQ query that has a "core" join query on k relations and then a subset of the k attributes are "summed" out (e.g. the sum could be because of projecting out a subset of attributes from the join query). While the algorithm [24] essentially figures out when to 'push in' the sums, in our case since we only care about $|\mathtt{C}|\,(1,\ldots,1)$ we will consider the obvious circuit that computes the "inner join" using a worst-case optimal join (WCOJ) algorithm like [27] and then adding in the addition gates. The basic idea is very simple: we will argue that the there are at most $\textsc{size}(\mathtt{C})^k$ tuples in the join output (each with having a value of 1 in $|\mathtt{C}|\,(1,\ldots,1)$). Then the largest value we can see in $|\mathtt{C}|\,(1,\ldots,1)$ is by summing up these at most $\textsc{size}(\mathtt{C})^k$ values of 1. Note that this immediately implies the claimed bound in Lemma 4.11.

We now sketch the argument for the claim about the join query above. First, we note that the computation of a WCOJ algorithm like [27] can be expressed as a circuit with *multiple* sinks (one for each output tuple). Note that annotation corresponding to $\mathbf{t}$ in $\mathtt{C}$ is the polynomial $\prod_{e\in E} R(\pi_e(\mathbf{t}))$ (where E indexes the set of relations). It is easy to see that in this case the value of $\mathbf{t}$ in $|\mathtt{C}|\,(1,\ldots,1)$ will be 1 (by multiplying 1 k times). The claim on the number of output tuples follow from the trivial bound of multiplying the input size bound (each relation has at most $n \le \textsc{size}(\mathtt{C})$ tuples and hence we get an overall bound of $n^k \le \textsc{size}(\mathtt{C})^k$. Note that we did not really use anything about the WCOJ algorithm except for the fact that $\mathtt{C}$ for the join part only is built only of multiplication gates. In fact, we do not need the better WCOJ join size bounds either (since we used the trivial n^k bound). As a final remark, we note that we can build the circuit for the join part by running say the algorithm from [24] on an FAQ query that just has the join query but each tuple is annotated with the corresponding variable X_i (i.e. the semi-ring for the FAQ query is $\mathbb{N}[\mathbf{X}]$).

C.5 OnePass Remarks

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

The evaluation of $|\mathtt{C}|\,(1,\ldots,1)$ can be defined recursively, as follows (where $\mathtt{C}_\mathtt{L}$ and $\mathtt{C}_\mathtt{R}$ are the 'left' and 'right' inputs of $\mathtt{C}$ if they exist):

$$|\mathtt{C}|\,(1,\ldots,1) = \begin{cases} |\mathtt{C}_\mathtt{L}|\,(1,\ldots,1)\cdot|\mathtt{C}_\mathtt{R}|\,(1,\ldots,1) & \textbf{if } \mathtt{C.type} = \times\\ |\mathtt{C}_\mathtt{L}|\,(1,\ldots,1)+|\mathtt{C}_\mathtt{R}|\,(1,\ldots,1) & \textbf{if } \mathtt{C.type} = +\\ |\mathtt{C.val}| & \textbf{if } \mathtt{C.type} = \textsc{num}\\ 1 & \textbf{if } \mathtt{C.type} = \textsc{var}. \end{cases} \tag{28}$$

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

$$\texttt{C.Lweight} \leftarrow \frac{|\texttt{C}_\texttt{L}|(1,\ldots,1)}{|\texttt{C}_\texttt{L}|(1,\ldots,1) + |\texttt{C}_\texttt{R}|(1,\ldots,1)}; \tag{29}$$

$$\texttt{C.Rweight} \leftarrow \frac{|\texttt{C}_\texttt{R}|(1,\ldots,1)}{|\texttt{C}_\texttt{L}|(1,\ldots,1) + |\texttt{C}_\texttt{R}|(1,\ldots,1)} \tag{30}$$

C.6 OnePass Example

▶ **Example C.11.** *Let* $\texttt{T}$ *encode the expression* $(X_1 + X_2)(X_1 - X_2) + X_2^2$*. After one pass, Algorithm 2 would have computed the following weight distribution. For the two inputs of the root* $+$ *node* $\texttt{T}$*,* $\texttt{T.Lweight} = \frac{4}{5}$ *and* $\texttt{T.Rweight} = \frac{1}{5}$*. Similarly, let* $\texttt{S}$ *denote the left-subtree of* $\texttt{T}_\texttt{L}$*,* $\texttt{S.Lweight} = \texttt{S.Rweight} = \frac{1}{2}$*. This is depicted in Fig. 4.*

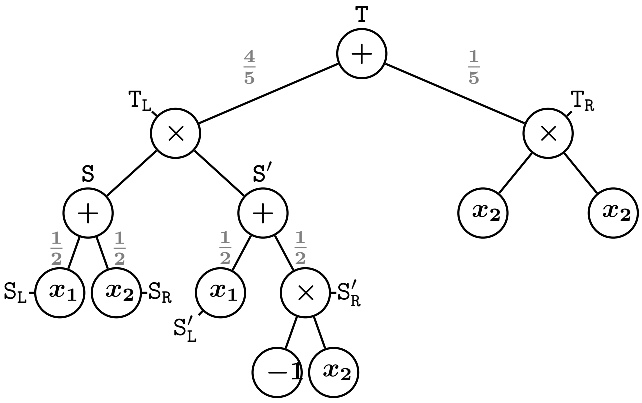


Figure 4 Weights computed by OnePass in Example C.11.

C.7 OnePass

C.8 Proof of Lemma C.6

Proof. We prove the correct computation of $\texttt{partial}$, $\texttt{Lweight}$, $\texttt{Rweight}$ values on $\texttt{C}$ by induction over the number of iterations in line 2 over the topological order TopOrd of the input circuit $\texttt{C}$. Note that TopOrd is the standard definition of a topological ordering over the DAG structure of $\texttt{C}$.

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. (28), lines 4 and 6 correctly compute $\texttt{C.partial}$ as 1 and $\texttt{C.val}$ respectively.

For the inductive hypothesis, assume that OnePass correctly computes $\texttt{S.partial}$, $\texttt{S.Lweight}$, and $\texttt{S.Rweight}$ for all gates $\texttt{g}$ in $\texttt{C}$ with $k \geq 0$ iterations over TopOrd.

We now prove for $k+1$ iterations that OnePass correctly computes the $\texttt{partial}$, $\texttt{Lweight}$, and $\texttt{Rweight}$ values for each gate $\texttt{g}_\texttt{i}$ in $\texttt{C}$ for $i \in [k+1]$. Note that the $\texttt{g}_{\texttt{k + 1}}$ must be in the last ordering of all gates $\texttt{g}_\texttt{i}$. It is also the case that $\texttt{g}_{k+1}$ has two inputs. Finally, note that for $\text{SIZE}(\texttt{C}) > 1$, if $\texttt{g}_{k+1}$ is a leaf node, we are back to the base case. Otherwise $\texttt{g}_{k+1}$ is an internal node $\texttt{g}_\texttt{s}.\texttt{type} = +$ or $\texttt{g}_\texttt{s}.\texttt{type} = \times$.

Algorithm 2 ONEPASS ($\texttt{C}$)

Input: $\texttt{C}$: Circuit
Output: $\texttt{C}$: Annotated Circuit
Output: $\texttt{sum} \in \mathbb{R}$

1: $\texttt{C}' \leftarrow$ REDUCE($\texttt{C}$)
2: **for** $\texttt{g}$ in TOPORD ($\texttt{C}'$) **do** ▷ TOPORD $(\cdot)$ is the topological order of $\texttt{C}$
3:     **if** $\texttt{g.type} =$ VAR **then**
4:         $\texttt{g.partial} \leftarrow 1$
5:     **else if** $\texttt{g.type} =$ NUM **then**
6:         $\texttt{g.partial} \leftarrow |\texttt{g.val}|$
7:     **else if** $\texttt{g.type} = \times$ **then**
8:         $\texttt{g.partial} \leftarrow \texttt{g}_\texttt{L}\texttt{.partial} \times \texttt{g}_\texttt{R}\texttt{.partial}$
9:     **else**
10:         $\texttt{g.partial} \leftarrow \texttt{g}_\texttt{L}\texttt{.partial} + \texttt{g}_\texttt{R}\texttt{.partial}$
11:         $\texttt{g.Lweight} \leftarrow \frac{\texttt{g}_\texttt{L}\texttt{.partial}}{\texttt{g.partial}}$
12:         $\texttt{g.Rweight} \leftarrow \frac{\texttt{g}_\texttt{R}\texttt{.partial}}{\texttt{g.partial}}$
13:     **end if**
14:     $\texttt{sum} \leftarrow \texttt{g.partial}$
15: **end for**
16: **return** $(\texttt{sum}, \texttt{C}')$

When $\texttt{g}_{k+1}\texttt{.type} = +$, then by line 10 $\texttt{g}_{k+1}\texttt{.partial} = \texttt{g}_{k+1_\texttt{L}}\texttt{.partial} + \texttt{g}_{k+1_\texttt{R}}\texttt{.partial}$, a correct computation, as per Eq. (28). Further, lines 11 and 12 compute $\texttt{g}_{k+1}\texttt{.Lweight} = \frac{\texttt{g}_{k+1_\texttt{L}}\texttt{.partial}}{\texttt{g}_{k+1}\texttt{.partial}}$ and analogously for $\texttt{g}_{k+1}\texttt{.Rweight}$. Note that all values needed for each computation have been correctly computed by the inductive hypothesis.

When $\texttt{g}_{k+1}\texttt{.type} = \times$, then line 8 computes $\texttt{g}_{k+1}\texttt{.partial} = \texttt{g}_{k+1_\texttt{L}.\texttt{partial}} \times \texttt{g}_{k+1_\texttt{R}}\texttt{.partial}$, which indeed is correct, as per Eq. (28).

Runtime Analysis

It is known that TOPORD(G) is computable in linear time. Next, each of the $\textsc{size}(\texttt{C})$ iterations of the loop in Line 2 take $O\left(\overline{\mathcal{M}}\left(\log\left(|\texttt{C}(1\ldots,1)|\right), \log \textsc{size}(\texttt{C})\right)\right)$ time. It is easy to see that each of all the numbers which the algorithm computes is at most $|\texttt{C}|(1,\ldots,1)$. Hence, by definition each such operation takes $\overline{\mathcal{M}}\left(\log\left(|\texttt{C}(1\ldots,1)|\right), \log \textsc{size}(\texttt{C})\right)$ time, which proves the claimed runtime. ◀

C.9 SampleMonomial Remarks

We briefly describe the top-down traversal of SAMPLEMONOMIAL. For a parent $+$ gate, the input to be visited is sampled from the weighted distribution precomputed by ONEPASS. When a parent $\times$ 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 an expression tree $\texttt{T}$, we do not know that such a method exists when computing a sample of the input polynomial representation.

Algorithm 3 SampleMonomial (C)

```
Input: C: Circuit
Output: vars: TreeSet
Output: sgn ∈ {−1, 1}                ▷ Algorithm 2 should have been run before this one
1: vars ← ∅
2: if C.type = + then                                        ▷ Sample at every + node
3:     C_samp ← Sample from left input (C_L) and right input (C_R) w.p. C.Lweight and
   C.Rweight.                    ▷ Each call to SampleMonomial uses fresh randomness
4:     (v, s) ← SampleMonomial(C_samp)
5:     return (v, s)
6: else if C.type = × then                  ▷ Multiply the sampled values of all inputs
7:     sgn ← 1
8:     for input in C.input do
9:         (v, s) ← SampleMonomial(input)
10:        vars ← vars ∪ {v}
11:        sgn ← sgn × s
12:    end for
13:    return (vars, sgn)
14: else if C.type = numeric then                            ▷ The leaf is a coefficient
15:    return ({}, sign(C.val))
16: else if C.type = var then
17:    return ({C.val}, 1)
18: end if
```

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 SampleMonomial.

C.10 Proof of Lemma C.7

Proof. We first need to show that SampleMonomial indeed returns a monomial $\mathtt{v}$,[14] such that $(\mathtt{v}, \mathtt{c})$ is in $\mathtt{E}(\mathtt{C})$, which we do by induction on the depth of $\mathtt{C}$.

For the base case, let the depth d of $\mathtt{C}$ be 0. We have that the root node is either a constant $\mathtt{c}$ for which by line 15 we return $\{\ \}$, or we have that $\mathtt{C.type} = \textsc{var}$ and $\mathtt{C.val} = x$, and by line 17 we return $\{x\}$. Both cases sample a monomial, and the base case is proven.

For the inductive hypothesis, assume that for $d \leq k$ for some $k \geq 0$, that it is indeed the case that SampleMonomial returns a monomial.

For the inductive step, let us take a circuit $\mathtt{C}$ with $d = k + 1$. Note that each input has depth $d \leq k$, and by inductive hypothesis both of them return a valid monomial. Then the root can be either a $+$ or $\times$ node. For the case of a $+$ root node, line 3 of SampleMonomial will choose one of the inputs of the root. By inductive hypothesis it is the case that a monomial in $\mathtt{E}\ (\mathtt{C})$ is being returned from either input. Then it follows that for the case of $+$ root node a valid monomial is returned by SampleMonomial. When the root is a $\times$ node,

[14] Technically it returns $\textsc{var}(\mathtt{v})$ but for less cumbersome notation we will refer to $\textsc{var}(\mathtt{v})$ simply by $\mathtt{v}$ in this proof.

line 10 computes the set union of the monomials returned by the two inputs of the root, and it is trivial to see by Definition 4.2 that $\mathtt{v}$ is a valid monomial in some $(\mathtt{v},\mathtt{c}) \in \mathtt{E}(\mathtt{C})$.

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

For the base case $d = 0$, by definition 2.10 we know that the root has to be either a coefficient or a variable. For either case, the probability of the value returned is 1 since there is only one value to sample from. When the root is a variable x the algorithm correctly returns $(\{x\}, 1)$. When the root is a coefficient, SampleMonomial correctly returns $(\{\ \}, sign(\mathtt{c}_i))$.

For the inductive hypothesis, assume that for $d \leq k$ and $k \geq 0$ SampleMonomial indeed samples $\mathtt{v}$ in $(\mathtt{v},\mathtt{c})$ in $\mathtt{E}(\mathtt{C})$ with probability $\frac{|\mathtt{c}|}{|\mathtt{C}|(1,\ldots,1)}$.

We prove now for $d = k + 1$ the inductive step holds. It is the case that the root of $\mathtt{C}$ has up to two inputs $\mathtt{C}_\mathtt{L}$ and $\mathtt{C}_\mathtt{R}$. Since $\mathtt{C}_\mathtt{L}$ and $\mathtt{C}_\mathtt{R}$ are both depth $d \leq k$, by inductive hypothesis, SampleMonomial will sample both monomials $\mathtt{v}_\mathtt{L}$ in $(\mathtt{v}_\mathtt{L},\mathtt{c}_\mathtt{L})$ of $\mathtt{E}(\mathtt{C}_\mathtt{L})$ and $\mathtt{v}_\mathtt{R}$ in $(\mathtt{v}_\mathtt{R},\mathtt{c}_\mathtt{R})$ of $\mathtt{E}(\mathtt{C}_\mathtt{R})$, from $\mathtt{C}_\mathtt{L}$ and $\mathtt{C}_\mathtt{R}$ with probability $\frac{|\mathtt{c}_\mathtt{L}|}{|\mathtt{C}_\mathtt{L}|(1,\ldots,1)}$ and $\frac{|\mathtt{c}_\mathtt{R}|}{|\mathtt{C}_\mathtt{R}|(1,\ldots,1)}$.

The root has to be either a $+$ or $\times$ node.

Consider the case when the root is $\times$. Note that we are sampling a term from $\mathtt{E}(\mathtt{C})$. Consider $(\mathtt{v},\mathtt{c})$ in $\mathtt{E}(\mathtt{C})$, where $\mathtt{v}$ is the sampled monomial. Notice also that it is the case that $\mathtt{v} = \mathtt{v}_\mathtt{L} \times \mathtt{v}_\mathtt{R}$, where $\mathtt{v}_\mathtt{L}$ is coming from $\mathtt{C}_\mathtt{L}$ and $\mathtt{v}_\mathtt{R}$ from $\mathtt{C}_\mathtt{R}$. The probability that SampleMonomial $(\mathtt{C}_\mathtt{L})$ returns $\mathtt{v}_\mathtt{L}$ is $\frac{|\mathtt{c}_{\mathtt{v}_\mathtt{L}}|}{|\mathtt{C}_\mathtt{L}|(1,\ldots,1)}$ and $\frac{|\mathtt{c}_{\mathtt{v}_\mathtt{R}}|}{|\mathtt{C}_\mathtt{R}|(1,\ldots,1)}$ for $\mathtt{v}_\mathtt{R}$. Since both $\mathtt{v}_\mathtt{L}$ and $\mathtt{v}_\mathtt{R}$ are sampled with independent randomness, the final probability for sample $\mathtt{v}$ is then $\frac{|\mathtt{c}_{\mathtt{v}_\mathtt{L}}|\cdot|\mathtt{c}_{\mathtt{v}_\mathtt{R}}|}{|\mathtt{C}_\mathtt{L}|(1,\ldots,1)\cdot|\mathtt{C}_\mathtt{R}|(1,\ldots,1)}$. For $(\mathtt{v},\mathtt{c})$ in $\mathtt{E}$ $(\mathtt{C})$, it is indeed the case that $|\mathtt{c}| = |\mathtt{c}_{\mathtt{v}_\mathtt{L}}| \cdot |\mathtt{c}_{\mathtt{v}_\mathtt{R}}|$ and that $|\mathtt{C}|(1,\ldots,1) = |\mathtt{C}_\mathtt{L}|(1,\ldots,1)\cdot|\mathtt{C}_\mathtt{R}|(1,\ldots,1)$, and therefore $\mathtt{v}$ is sampled with correct probability $\frac{|\mathtt{c}|}{|\mathtt{C}|(1,\ldots,1)}$.

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

$$
\begin{aligned}
&P(\textsc{SampleMonomial}(\mathtt{C}) = \mathtt{v}) = \\
&P(\textsc{SampleMonomial}(\mathtt{C}_\mathtt{L}) = \mathtt{v}) \cdot P(\mathit{SampledChild}(\mathtt{C}) = \mathtt{C}_\mathtt{L}) \\
&\quad = \frac{|\mathtt{c}_\mathtt{v}|}{|\mathtt{C}_\mathtt{L}|(1,\ldots,1)} \cdot \frac{|\mathtt{C}_\mathtt{L}|(1,\ldots,1)}{|\mathtt{C}_\mathtt{L}|(1,\ldots,1) + |\mathtt{C}_\mathtt{R}|(1,\ldots,1)} \\
&\quad = \frac{|\mathtt{c}_\mathtt{v}|}{|\mathtt{C}|(1,\ldots,1)},
\end{aligned}
$$

and we obtain the desired result.

Run-time Analysis

It is easy to check that except for lines 10 and 3, all lines take $O(1)$ time. For Line 10, 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 $\textsc{deg}(\mathtt{C})$, each gate visited takes $O(\log \textsc{deg}(\mathtt{C}))$. For Line 3, note that we pick $\mathtt{C}_\mathtt{L}$ with probability $\frac{a}{a+b}$ where

$a = \texttt{C.Lweight}$ and $b = \texttt{C.Rweight}$. 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 $a+b \leq |\texttt{C}|(1,\ldots,1)$. This means we need to add and compare $\log |\texttt{C}|(1,\ldots,1)$-bit numbers, which can certainly be done in time $\overline{\mathcal{M}}\left(\log\left(|\texttt{C}(1\ldots,1)|\right), \log \textsc{size}(\texttt{C})\right)$ (note that this is an over-estimate). Denote $\textsc{Cost}(\texttt{C})$ (Eq. (31)) to be an upper bound of the number of nodes visited by $\textsc{SampleMonomial}$. Then the runtime is $O\left(\textsc{Cost}(\texttt{C}) \cdot \log \textsc{deg}(\texttt{C}) \cdot \overline{\mathcal{M}}\left(\log\left(|\texttt{C}(1\ldots,1)|\right), \log \textsc{size}(\texttt{C})\right)\right)$.

We now bound the number of recursive calls in $\textsc{SampleMonomial}$ by $O\left((\textsc{deg}(\texttt{C})+1) \cdot \textsc{depth}(\texttt{C})\right)$, which by the above will prove the claimed runtime.

Let $\textsc{Cost}(\cdot)$ be a function that models an upper bound on the number of gates that can be visited in the run of $\textsc{SampleMonomial}$. We define $\textsc{Cost}(\cdot)$ recursively as follows.

$$\textsc{Cost}(\texttt{C}) = \begin{cases} 1 + \textsc{Cost}(\texttt{C}_\texttt{L}) + \textsc{Cost}(\texttt{C}_\texttt{R}) & \textbf{if } \texttt{C.type} = \times \\ 1 + \max\left(\textsc{Cost}(\texttt{C}_\texttt{L}), \textsc{Cost}(\texttt{C}_\texttt{R})\right) & \textbf{if } \texttt{C.type} = + \\ 1 & \textbf{otherwise} \end{cases} \tag{31}$$

First note that the number of gates visited in $\textsc{SampleMonomial}$ is $\leq \textsc{Cost}(\texttt{C})$. To show that Eq. (31) upper bounds the number of nodes visited by $\textsc{SampleMonomial}$, note that when $\textsc{SampleMonomial}$ visits a gate such that $\texttt{C.type} = \times$, line 8 visits each input of $\texttt{C}$, as defined in (31). For the case when $\texttt{C.type} = +$, 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 $\textsc{Cost}(\cdot)$ an upperbound. Finally, it is trivial to see that when $\texttt{C.type} \in \{\textsc{var}, \textsc{num}\}$, i.e., a source gate, that only one gate is visited.

We prove the following inequality holds.

$$2\left(\textsc{deg}(\texttt{C})+1\right) \cdot \textsc{depth}(\texttt{C}) + 1 \geq \textsc{Cost}(\texttt{C}) \tag{32}$$

Note that Eq. (32) implies the claimed runtime. We prove Eq. (32) for the number of gates traversed in $\textsc{SampleMonomial}$ using induction over $\textsc{depth}(\texttt{C})$. Recall how degree is defined in Definition 4.6.

For the base case $\textsc{deg}(\texttt{C}) = \textsc{depth}(\texttt{C}) = 0$, $\textsc{Cost}(\texttt{C}) = 1$, and it is trivial to see that the inequality $2\textsc{deg}(\texttt{C}) \cdot \textsc{depth}(\texttt{C}) + 1 \geq \textsc{Cost}(\texttt{C})$ holds.

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

$$2\left(\textsc{deg}(\texttt{C}_i)+1\right) \cdot \textsc{depth}(\texttt{C}_i) + 1 \geq \textsc{Cost}(\texttt{C}_i). \tag{33}$$

It is also true that $\textsc{depth}(\texttt{C}_\texttt{L}) \leq \textsc{depth}(\texttt{C}) - 1$ and $\textsc{depth}(\texttt{C}_\texttt{R}) \leq \textsc{depth}(\texttt{C}) - 1$.

If $\texttt{C.type} = +$, then $\textsc{deg}(\texttt{C}) = \max\left(\textsc{deg}(\texttt{C}_\texttt{L}), \textsc{deg}(\texttt{C}_\texttt{R})\right)$. Otherwise $\texttt{C.type} = \times$ and $\textsc{deg}(\texttt{C}) = \textsc{deg}(\texttt{C}_\texttt{L}) + \textsc{deg}(\texttt{C}_\texttt{R}) + 1$. In either case it is true that $\textsc{depth}(\texttt{C}) = \max(\textsc{depth}(\texttt{C}_\texttt{L}), \textsc{depth}(\texttt{C}_\texttt{R})) + 1$.

If $\texttt{C.type} = \times$, then, substituting values, the following should hold,

$$\begin{aligned} &2\left(\textsc{deg}(\texttt{C}_\texttt{L}) + \textsc{deg}(\texttt{C}_\texttt{R}) + 2\right) \cdot \left(\max(\textsc{depth}(\texttt{C}_\texttt{L}), \textsc{depth}(\texttt{C}_\texttt{R})) + 1\right) + 1 \\ &\quad \geq 2\left(\textsc{deg}(\texttt{C}_\texttt{L}) + 1\right) \cdot \textsc{depth}(\texttt{C}_\texttt{L}) + 2\left(\textsc{deg}(\texttt{C}_\texttt{R}) + 1\right) \cdot \textsc{depth}(\texttt{C}_\texttt{R}) + 3 && (34) \\ &\quad \geq 1 + \textsc{Cost}(\texttt{C}_\texttt{L}) + \textsc{Cost}(\texttt{C}_\texttt{R}) = \textsc{Cost}(\texttt{C}). && (35) \end{aligned}$$

To prove (34), first, the LHS expands to,

$$2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}_{\max}+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}_{\max}+4\textsc{depth}_{\max}+2\textsc{deg}(\mathtt{C}_\mathtt{L})+2\textsc{deg}(\mathtt{C}_\mathtt{R})+4+1 \tag{36}$$

where $\textsc{depth}_{\max}$ is used to denote the maximum depth of the two input subcircuits. The RHS expands to

$$2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}(\mathtt{C}_\mathtt{R})+2\textsc{depth}(\mathtt{C}_\mathtt{R})+3 \tag{37}$$

Putting Eq. (36) and Eq. (37) together we get

$$\begin{aligned}&2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}_{\max}+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}_{\max}+4\textsc{depth}_{\max}+2\textsc{deg}(\mathtt{C}_\mathtt{L})+2\textsc{deg}(\mathtt{C}_\mathtt{R})+5\\&\quad\geq 2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}(\mathtt{C}_\mathtt{R})+2\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{depth}(\mathtt{C}_\mathtt{R})+3\end{aligned} \tag{38}$$

Since the following is always true,

$$\begin{aligned}&2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}_{\max}+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}_{\max}+4\textsc{depth}_{\max}+5\\&\quad\geq 2\textsc{deg}(\mathtt{C}_\mathtt{L})\cdot\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{deg}(\mathtt{C}_\mathtt{R})\cdot\textsc{depth}(\mathtt{C}_\mathtt{R})+2\textsc{depth}(\mathtt{C}_\mathtt{L})+2\textsc{depth}(\mathtt{C}_\mathtt{R})+3,\end{aligned}$$

then it is the case that Eq. (38) is *always* true.

Now to justify (35) which holds for the following reasons. First, the RHS is the result of Eq. (31) when $\mathtt{C.type} = \times$. The LHS is then produced by substituting the upperbound of (33) for each $\textsc{Cost}(\mathtt{C}_i)$, trivially establishing the upper bound of (35). This proves Eq. (32) for the $\times$ case.

For the case when $\mathtt{C.type} = +$, substituting values yields

$$\begin{aligned}&2\left(\max(\textsc{deg}(\mathtt{C}_\mathtt{L}),\textsc{deg}(\mathtt{C}_\mathtt{R}))+1\right)\cdot\left(\max(\textsc{depth}(\mathtt{C}_\mathtt{L}),\textsc{depth}(\mathtt{C}_\mathtt{R}))+1\right)+1\\&\quad\geq \max\left(2\left(\textsc{deg}(\mathtt{C}_\mathtt{L})+1\right)\cdot\textsc{depth}(\mathtt{C}_\mathtt{L})+1, 2\left(\textsc{deg}(\mathtt{C}_\mathtt{R})+1\right)\cdot\textsc{depth}(\mathtt{C}_\mathtt{R})+1\right)+1 &(39)\\&\quad\geq 1+\max(\textsc{Cost}(\mathtt{C}_\mathtt{L}),\textsc{Cost}(\mathtt{C}_\mathtt{R}))=\textsc{Cost}(\mathtt{C}) &(40)\end{aligned}$$

To prove (39), the LHS expands to

$$2\textsc{deg}_{\max}\textsc{depth}_{\max}+2\textsc{deg}_{\max}+2\textsc{depth}_{\max}+2+1. \tag{41}$$

Since $\textsc{deg}_{\max}\cdot\textsc{depth}_{\max}\geq\textsc{deg}(\mathtt{C}_i)\cdot\textsc{depth}(\mathtt{C}_i)$, the following upper bound holds for the expanded RHS of (39):

$$2\textsc{deg}_{\max}\textsc{depth}_{\max}+2\textsc{depth}_{\max}+2 \tag{42}$$

Putting it together we obtain the following for (39):

$$\begin{aligned}&2\textsc{deg}_{\max}\textsc{depth}_{\max}+2\textsc{deg}_{\max}+2\textsc{depth}_{\max}+3\\&\quad\geq 2\textsc{deg}_{\max}\textsc{depth}_{\max}+2\textsc{depth}_{\max}+2,\end{aligned} \tag{43}$$

where it can be readily seen that the inequality stand and (43) follows. This proves (39).

Similar to the case of $\mathtt{C.type} = \times$, (40) follows by equations (31) and (33).

This proves (32) as desired. ◀

C.11 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 Corollary 4.10, γ must be a constant in order for Algorithm 1 to acheive 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 [1] 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 γ. The experiments show γ 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 γ 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 γ 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_0) are independent of the number of blocks, and (ii) BIDBs model uncertain attributes, so block size (and hence γ) is a function of the "messiness" of a dataset, rather than its size. Thus, we expect the corollary to hold in general.

Query	CF	CI	γ
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 SPJU queries. As mentioned earlier, we represent lineage polynomials as arithmetic circuits over $\mathbb{N}$-valued variables with $+$, $\times$. A circuit for query Q and $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$ is a directed acyclic graph $\langle V_{Q,\mathbf{D}}, E_{Q,\mathbf{D}}, \phi_{Q,\mathbf{D}}, \ell_{Q,\mathbf{D}}\rangle$

with vertices $V_{Q,\mathbf{D}}$ and directed edges $E_{Q,\mathbf{D}} \subset {V_{Q,\mathbf{D}}}^2$. The sink function $\phi_{Q,\mathbf{D}} : \mathcal{U}^n \to V_{Q,\mathbf{D}}$ is a partial function that maps the tuples of the n-ary relation $Q(\mathbf{D})$ to vertices. We require that $\phi_{Q,\mathbf{D}}$'s range be limited to sink vertices (i.e., vertices with out-degree 0). A function $\ell_{Q,\mathbf{D}} : V_{Q,\mathbf{D}} \to \{ +, \times \} \cup \mathbb{N} \cup \mathbf{X}$ assigns a label to each node: Source nodes (i.e., vertices with in-degree 0) are labeled with constants or variables (i.e., $\mathbb{N} \cup \mathbf{X}$), while the remaining nodes are labeled with the symbol $+$ or $\times$. We require that vertices have an in-degree of at most two. For the specifics on how to construct a circuit to encode the polynomials of all result tuples for a query and $\mathbb{N}[\mathbf{X}]$-PDB see Appendix D.1. 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 $|\mathbf{D}| \leq c \cdot |D|$.

D.1.2 Circuit size vs. runtime

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 SPJU query Q and $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$ to its $\mathbf{cost}(Q, D)$ where D is one of the possible worlds of $\mathbf{D}$. 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 $v \in V_{Q,\mathbf{D}}$ in the arithmetic circuit for

$$\langle V_{Q,\mathbf{D}}, E_{Q,\mathbf{D}}, \phi_{Q,\mathbf{D}}, \ell_{Q,\mathbf{D}} \rangle$$

encodes a polynomial, realized as

$$\mathbf{lin}\,(v) = \begin{cases} \sum_{v' : (v',v) \in E_{Q,\mathbf{D}}} \mathbf{lin}\,(v') & \textbf{if } \ell(v) = + \\ \prod_{v' : (v',v) \in E_{Q,\mathbf{D}}} \mathbf{lin}\,(v') & \textbf{if } \ell(v) = \times \\ \ell(v) & \textbf{otherwise} \end{cases}$$

We define the circuit for a select-union-project-join Q recursively by cases as follows. In each case, let $\langle V_{Q_i,\mathbf{D}}, E_{Q_i,\mathbf{D}}, \phi_{Q_i,\mathbf{D}}, \ell_{Q_i,\mathbf{D}} \rangle$ denote the circuit for subquery Q_i.

Base Relation. Let Q be a base relation R. We define one node for each tuple. Formally, let $V_{Q,\mathbf{D}} = \{ v_t \mid t \in R \}$, let $\phi_{Q,\mathbf{D}}(t) = v_t$, let $\ell_{Q,\mathbf{D}}(v_t) = R(t)$, and let $E_{Q,\mathbf{D}} = \emptyset$. This circuit has $|R|$ vertices.

Selection. Let $Q = \sigma_\theta\,(Q_1)$. We re-use the circuit for Q_1. Formally, let $V_{Q,\mathbf{D}} = V_{Q_1,\mathbf{D}}$, let $\ell_{Q,\mathbf{D}}(v_0) = 0$, and let $\ell_{Q,\mathbf{D}}(v) = \ell_{Q_1,\mathbf{D}}(v)$ for any $v \in V_{Q_1,\mathbf{D}}$. Let $E_{Q,\mathbf{D}} = E_{Q_1,\mathbf{D}}$, and define

$$\phi_{Q,\mathbf{D}}(t) = \phi_{Q_1,\mathbf{D}}(t) \text{ for } t \text{ s.t. } \theta(t).$$

Dead sinks are iteratively removed, and so this circuit has at most $|V_{Q_1,\mathbf{D}}|$ vertices.

Projection. Let $Q = \pi_{\mathbf{A}} Q_1$. We extend the circuit for Q_1 with a new set of sum vertices (i.e., vertices with label $+$) for each tuple in Q, and connect them to the corresponding sink nodes of the circuit for Q_1. Naively, let $V_{Q,\mathbf{D}} = V_{Q_1,\mathbf{D}} \cup \{ v_t \mid t \in \pi_{\mathbf{A}} Q_1 \}$, let $\phi_{Q,\mathbf{D}}(t) = v_t$, and let $\ell_{Q,\mathbf{D}}(v_t) = +$. Finally let

$$E_{Q,\mathbf{D}} = E_{Q_1,\mathbf{D}} \cup \{ (\phi_{Q_1,\mathbf{D}}(t'), v_t) \mid t = \pi_{\mathbf{A}} t', t' \in Q_1, t \in \pi_{\mathbf{A}} Q_1 \}$$

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 tree.

The resulting structure has at most $|Q_1| - 1$ new vertices. The corrected circuit thus has at most $|V_{Q_1,\mathbf{D}}| + |Q_1|$ vertices.

Union. Let $Q = Q_1 \cup Q_2$. We merge graphs and produce a sum vertex for all tuples in both sides of the union. Formally, let $V_{Q,\mathbf{D}} = V_{Q_1,\mathbf{D}} \cup V_{Q_2,\mathbf{D}} \cup \{\, v_t \mid t \in Q_1 \cap Q_2 \,\}$, let $\ell_{Q,\mathbf{D}}(v_t) = +$, and let

$$E_{Q,\mathbf{D}} = E_{Q_1,\mathbf{D}} \cup E_{Q_2,\mathbf{D}} \cup \{\, (\phi_{Q_1,\mathbf{D}}(t), v_t), (\phi_{Q_2,\mathbf{D}}(t), v_t) \mid t \in Q_1 \cap Q_2 \,\}$$

$$\phi_{Q,\mathbf{D}}(t) = \begin{cases} v_t & \textbf{if } t \in Q_1 \cap Q_1 \\ \phi_{Q_1,\mathbf{D}}(t) & \textbf{if } t \notin Q_2 \\ \phi_{Q_2,\mathbf{D}}(t) & \textbf{if } t \notin Q_1 \end{cases}$$

This circuit has $|V_{Q_1,\mathbf{D}}| + |V_{Q_2,\mathbf{D}}| + |Q_1 \cap Q_2|$ vertices.

***k*-ary Join.** Let $Q = Q_1 \bowtie \ldots \bowtie Q_k$. We merge graphs and produce a multiplication vertex for all tuples resulting from the join Naively, let $V_{Q,\mathbf{D}} = V_{Q_1,\mathbf{D}} \cup \ldots \cup V_{Q_k,\mathbf{D}} \cup \{\, v_t \mid t \in Q_1 \bowtie \ldots \bowtie Q_k \,\}$, let

$$E_{Q,\mathbf{D}} = E_{Q_1,\mathbf{D}} \cup \ldots \cup E_{Q_k,\mathbf{D}} \cup \Big\{\, (\phi_{Q_1,\mathbf{D}}(\pi_{sch(Q_1)}t), v_t), \\ \ldots, (\phi_{Q_k,\mathbf{D}}(\pi_{sch(Q_k)}t), v_t) \;\mid t \in Q_1 \bowtie \ldots \bowtie Q_k \,\Big\}$$

Let $\ell_{Q,\mathbf{D}}(v_t) = \times$, and let $\phi_{Q,\mathbf{D}}(t) = v_t$ As in projection, newly created vertices will have an in-degree of k, and a fan-in tree is required. There are $|Q_1 \bowtie \ldots \bowtie Q_k|$ such vertices, so the corrected circuit has $|V_{Q_1,\mathbf{D}}| + \ldots + |V_{Q_k,\mathbf{D}}| + (k-1)|Q_1 \bowtie \ldots \bowtie Q_k|$ vertices.

▶ **Lemma D.1.** *Given a $\mathbb{N}[\mathbf{X}]$-PDB $\mathbf{D}$ and query plan Q, the runtime of Q over $\mathbf{D}$ has the same or better complexity as the size of the lineage of $Q(\mathbf{D})$. That is, we have $|V_{Q,\mathbf{D}}| \leq (k-1)\boldsymbol{cost}(Q)$, where k is the maximal degree of any polynomial in $Q(\mathbf{D})$.*

The proof is shown in in Appendix D.2. We now have all the pieces to argue that using our approximation algorithm, the expected multiplicities of a SPJU query can be computed in essentially the same runtime as deterministic query processing for the same query.

D.2 Proof for Lemma D.1

Proof. Proof by induction. The base case is a base relation: $Q = R$ and is trivially true since $|V_{R,\mathbf{D}}| = |R|$. For the inductive step, we assume that we have circuits for subplans $Q_1, \ldots, Q_n$ such that $|V_{Q_i,\mathbf{D}}| \leq (k_i - 1)\mathbf{cost}(Q_i, \mathbf{D})$ where k_i is the degree of Q_i.

Selection. Assume that $Q = \sigma_\theta(Q_1)$. In the circuit for Q, $|V_{Q,\mathbf{D}}| = |V_{Q_1,\mathbf{D}}|$ vertices, so from the inductive assumption and $\mathbf{cost}(Q, \mathbf{D}) = \mathbf{cost}(Q_1, \mathbf{D})$ by definition, we have $|V_{Q,\mathbf{D}}| \leq (k-1)\mathbf{cost}(Q, \mathbf{D})$. **Projection.** Assume that $Q = \pi_{\mathbf{A}}(Q_1)$. The circuit for Q has at most $|V_{Q_1,\mathbf{D}}| + |Q_1|$ vertices.

$$|V_{Q,\mathbf{D}}| \leq |V_{Q_1,\mathbf{D}}| + |Q_1|$$

(From the inductive assumption)

$$\leq (k-1)\mathbf{cost}(Q_1, \mathbf{D}) + |Q_1|$$

(By definition of $\mathbf{cost}(Q, \mathbf{D})$)

$$\leq (k-1)\mathbf{cost}(Q, \mathbf{D}).$$

Union. Assume that $Q = Q_1 \cup Q_2$. The circuit for Q has $|V_{Q_1,\mathbf{D}}| + |V_{Q_2,\mathbf{D}}| + |Q_1 \cap Q_2|$ vertices.

$$|V_{Q,\mathbf{D}}| \leq |V_{Q_1,\mathbf{D}}| + |V_{Q_2,\mathbf{D}}| + |Q_1| + |Q_2|$$

(From the inductive assumption)

$$\leq (k-1)(\mathbf{cost}(Q_1,\mathbf{D}) + \mathbf{cost}(Q_2,\mathbf{D})) + (b_1 + b_2)$$

(By definition of $\mathbf{cost}(Q,\mathbf{D})$)

$$\leq (k-1)(\mathbf{cost}(Q,\mathbf{D})).$$

k-ary Join. Assume that $Q = Q_1 \bowtie \ldots \bowtie Q_k$. The circuit for Q has $|V_{Q_1,\mathbf{D}}| + \ldots + |V_{Q_k,\mathbf{D}}| + (k-1)|Q_1 \bowtie \ldots \bowtie Q_k|$ vertices.

$$|V_{Q,\mathbf{D}}| = |V_{Q_1,\mathbf{D}}| + \ldots + |V_{Q_k,\mathbf{D}}| + (k-1)|Q_1 \bowtie \ldots \bowtie Q_k|$$

From the inductive assumption and noting $\forall i : k_i \leq k-1$

$$\begin{aligned}&\leq (k-1)\mathbf{cost}(Q_1,\mathbf{D}) + \ldots + (k-1)\mathbf{cost}(Q_k,\mathbf{D}) +\\ &\quad (k-1)|Q_1 \bowtie \ldots \bowtie Q_k|\\ &\leq (k-1)(\mathbf{cost}(Q_1,\mathbf{D}) + \ldots + \mathbf{cost}(Q_k,\mathbf{D}) +\\ &\quad |Q_1 \bowtie \ldots \bowtie Q_k|)\end{aligned}$$

(By definition of $\mathbf{cost}(Q,\mathbf{D})$)

$$= (k-1)\mathbf{cost}(Q,\mathbf{D}).$$

The property holds for all recursive queries, and the proof holds. ◀

E 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 [18]. E.g. the fact that k-matching is $\#W[1] - hard$ implies that we cannot have an $n^{\Omega(1)}$ 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 has led to intense research into the new sub-field of *fine-grained complexity* (see [38]), where we care about the exponent in our hardness assumptions as well– e.g. Conjecture 3.2 is based on the popular *Triangle detection hypothesis* in this area (cf. [25]).