Computing expected multiplicities for bag-TIDBs with bounded multiplicities

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

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

ABSTRACT

In this work, we study the problem of computing a tuple's expected multiplicity over probabilistic databases with bag semantics (where each tuple is associated with a multiplicity) exactly and approximately. We consider bag-TIDBs where we have a bound c on the maximum multiplicity of each tuple and tuples are independent probabilistic events (we refer to such databases as c-TIDBs). We are specifically interested in the fine-grained complexity of computing expected multiplicities and how it compares to the complexity of deterministic query evaluation algorithms — if these complexities are comparable, it opens the door to practical deployment of probabilistic databases. Unfortunately, our results imply that computing expected multiplicities for c-TIDBs based on the results produced by such query evaluation algorithms introduces super-linear overhead (under parameterized complexity hardness assumptions/conjectures). We proceed to study approximation of expected result tuple multiplicities for positive relational algebra queries ($\mathcal{RA}^+$) over c-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 expected multiplicity of an output tuple in time linear in the runtime of the corresponding deterministic query for any $\mathcal{RA}^+$ query.

CCS CONCEPTS

• **Information systems** → **Information systems applications**; **Data management systems**; • **Theory of computation** → **Probabilistic computation**; **Complexity classes**.

KEYWORDS

probabilstic data model, parameterized complexity, fine-grained complexity, lineage polynomial

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Su Feng, Boris Glavic, Aaron Huber, Oliver Kennedy, and Atri Rudra. 2022. Computing expected multiplicities for bag-TIDBs with bounded multiplicities. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 28 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

This work explores the problem of computing the expectation of the multiplicity of a tuple in the result of a query over a c-TIDB, a type of probabilistic database with bag semantics where the multiplicity of a tuple is a random variable with range $[0, c]$ for some fixed constant c and multiplicities assigned to any two tuples are independent of each other. Formally, a c-TIDB, $\mathcal{D} = \left(\{0, \ldots, c\}^D, \mathcal{P}\right)$ consists of a set of tuples D and a probability distribution $\mathcal{P}$ over all possible worlds generated by assigning each tuple $t \in D$ a multiplicity in the range $[0, c]$. Any such world can be encoded as a vector from $\{0, \ldots, c\}^D$, the set of all vectors of length $n = |D|$ such that each index corresponds to a distinct $t \in D$ storing its multiplicity. A given world $\mathbf{W} \in \{0, \ldots, c\}^D$ can be interpreted as follows: for each $t \in D$, $\mathbf{W}_t$ is the multiplicity of t in $\mathbf{W}$. Given that the multiplicities of tuples are independent events, the probability distribution $\mathcal{P}$ can be expressed compactly by assigning each tuple a (disjoint) probability distribution over $[0, c]$. Let $p_{t,j}$ denote the probability that tuple t is assigned multiplicity j. The probability of a particular world $\mathbf{W}$ is then $\prod_{t \in D} p_{t,\mathbf{W}_t}$.

Allowing for $\leq c$ multiplicities across all tuples gives rise to having $\leq (c+1)^n$ possible worlds instead of the usual 2^n possible worlds of a 1-TIDB, which (assuming set query semantics), is the same as the traditional set TIDB. In this work, since we are generally considering bag query input, we will only be considering bag query semantics. We denote by $Q(\mathbf{W})(t)$ the multiplicity of t in query Q over possible world $\mathbf{W} \in \{0, \ldots, c\}^D$.

We can formally state our problem of computing the expected multiplicity of a result tuple as:

Problem 1.1. *Given a c-TIDB $\mathcal{D} = \left(\{0, \ldots, c\}^D, \mathcal{P}\right)$, $\mathcal{RA}^+$ query[1] Q, and result tuple t, compute the expected multiplicity of t: $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}\left[Q(\mathbf{W})(t)\right]$.*

[1] An $\mathcal{RA}^+$ query is a query expressed in positive relational algebra, i.e., using only the relational algebra operators selection (σ), projection (π), natural join ($\bowtie$) and union ($\cup$).

$$\Phi[\pi_A(Q), \overline{D}, t] = \sum_{t':\pi_A(t')=t} \Phi[Q, \overline{D}, t']$$

$$\Phi[\sigma_\theta(Q), \overline{D}, t] = \begin{cases} \Phi[Q, \overline{D}, t] & \text{if } \theta(t) \\ 0 & \text{otherwise.} \end{cases}$$

$$\Phi[Q_1 \cup Q_2, \overline{D}, t] = \Phi[Q_1, \overline{D}, t] + \Phi[Q_2, \overline{D}, t]$$

$$\Phi[Q_1 \bowtie Q_2, \overline{D}, t] = \Phi[Q_1, \overline{D}, \pi_{attr(Q_1)} t] \cdot \Phi[Q_2, \overline{D}, \pi_{attr(Q_2)} t]$$

$$\Phi[R, \overline{D}, t] = X_t$$

Figure 1: Construction of the lineage (polynomial) for an $\mathcal{RA}^+$ query Q over an arbitrary deterministic database $\overline{D}$, where X consists of all X_t over all R in $\overline{D}$ and t in R. Here $\overline{D}.R$ denotes the instance of relation R in $\overline{D}$. Please note, after we introduce the reduction to 1-BIDB, the base case will be expressed alternatively.

It is natural to explore computing the expected multiplicity of a result tuple as this is the analog for computing the marginal probability of a tuple in a set PDB. In this work we will assume that $c = O(1)$ since this is what is typically seen in practice. Allowing for unbounded c is an interesting open problem.

Hardness of Set Query Semantics and Bag Query Semantics. Set query evaluation semantics over 1-TIDBs have been studied extensively, and the data complexity of the problem in general has been shown by Dalvi and Suicu to be #P-hard [14]. For our setting, there exists a trivial polytime algorithm to compute Problem 1.1 for any $\mathcal{RA}^+$ query over a c-TIDB due to linearity of expection (see Sec. 1.1). Since we can compute Problem 1.1 in polynomial time, the interesting question that we explore deals with analyzing the hardness of computing expectation using fine-grained analysis and parameterized complexity, where we are interested in the exponent of polynomial runtime.

Specifically, in this work we ask if Problem 1.1 can be solved in time linear in the runtime of an analogous deterministic query which we make more precise shortly. If this is true, then this would open up the way for deployment of c-TIDBs in practice. To analyze this question we denote by $T^*(Q, \mathcal{D})$ the optimal runtime complexity of computing Problem 1.1 over c-TIDB $\mathcal{D}$.

Let $T_{det}\left(Q, \overline{D}, c\right)$ (see Sec. 2.4 for further details) denote the runtime for query Q, deterministic database $\overline{D}$, and multiplicity bound c. This paper considers $\mathcal{RA}^+$ queries for which order of operations is *explicit*, as opposed to other query languages, e.g. Datalog, UCQ. Thus, since order of operations affects runtime, we denote the optimized $\mathcal{RA}^+$ query picked by an arbitrary production system as $\text{OPT}(Q) = \min_{Q' \in \mathcal{RA}^+, Q' \equiv Q} T_{det}\left(Q', \overline{D}, c\right)$. Then $T_{det}\left(\text{OPT}(Q), \overline{D}, c\right)$ is the runtime for the optimized query.[2]

Our lower bound results. Our question is whether or not it is always true that $T^*(Q, \mathcal{D}) \leq T_{det}(\text{OPT}(Q), D, c)$. Unfortunately this is not the case. Table 1 shows our results.

Specifically, depending on what hardness result/conjecture we assume, we get various weaker or stronger versions of *no* as an answer to our question. To make some sense of the other lower bounds in Table 1, we note that it is not too hard to show that $T^*(Q, \mathcal{D}) \leq O\left(\left(T_{det}(\text{OPT}(Q), D, c)\right)^k\right)$, where k is the join width (our notion of join width follows from Definition 2.2 and Fig. 1.) of the query Q over all result tuples t (and the parameter that defines our family of hard queries).

What our lower bound in the third row says is that one cannot get more than a polynomial improvement over essentially the trivial algorithm for Problem 1.1. However, this result assumes a hardness conjecture that is not as well studied as those in the first two rows of the table (see Sec. 3 for more discussion on the hardness assumptions). Further, we note that existing results[3] already imply the claimed lower bounds if we were to replace the $T_{det}(\text{OPT}(Q), D, c)$ by just n (indeed these results follow from known lower bounds for deterministic query processing). Our contribution is to then identify a family of hard queries where deterministic query processing is 'easy' but computing the expected multiplicities is hard.

Our upper bound results. We introduce a $(1 \pm \epsilon)$-approximation algorithm that computes Problem 1.1 in time $O_\epsilon(T_{det}(\text{OPT}(Q), D, c))$. This means, when we are okay with approximation, that we solve Problem 1.1 in time linear in the size of the deterministic query and bag PDBs are deployable in practice. In contrast, known approximation techniques ([32, 40]) in set-PDBs need time $\Omega(T_{det}(\text{OPT}(Q), D, c)^{2k})$ (see Appendix G). Further, our approximation algorithm works for a more general notion of bag PDBs beyond c-TIDBs (see Sec. 2.2).

1.1 Polynomial Equivalence

A common encoding of probabilistic databases (e.g., in [2, 5, 29, 30] and many others) relies on annotating tuples with lineages or propositional formulas that describe the set of possible worlds that the tuple appears in. The bag semantics analog is a provenance/lineage polynomial (see Fig. 1) $\Phi[Q, D, t]$ [27], a polynomial with non-zero integer coefficients and exponents, over variables $\mathbf{X}$ encoding input tuple multiplicities. Evaluating a lineage polynomial for a query result tuple t_{out} by, for each tuple t_{in}, assigning the variable $X_{t_{in}}$ encoding the tuple's multiplicity to the tuple's multiplicity in the possible world yields the multiplicity of the t_{out} in the query result for this world.

We drop Q, D, and t from $\Phi[Q, D, t]$ when they are clear from the context or irrelevant to the discussion. We now specify the problem of computing the expectation of tuple multiplicity in the language of lineage polynomials:

Problem 1.2 (Expected Multiplicity of Lineage Polynomials). *Given an $\mathcal{RA}^+$ query Q, c-TIDB $\mathcal{D}$ and result tuple t, compute the expected multiplicity of the polynomial $\Phi[Q, D, t]$ (i.e., $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}[\Phi[Q, D, t](\mathbf{W})]$, where $\mathbf{W} \in \{0, \ldots, c\}^D$).*

We note that computing Problem 1.1 is equivalent (yields the same result as) to computing Problem 1.2 (see Proposition 2.8).

[2] Note that our work applies to any $Q \in \mathcal{RA}^+$, which implies that specific heuristics for choosing an optimized query can be abstracted away, i.e., our work does not consider heuristic techniques.

[3] This claim follows from known results for the problem of counting k-cliques, where for a query Q over database D that counts the number of k-cliques. Specifically, a lower bound of the form $\Omega\left(n^{1+\epsilon_0}\right)$ for *some* $\epsilon_0 > 0$ follows from the triangle detection hypothesis (this like our result is for $k = 3$). Second, a lower bound of $\omega\left(n^{C_0}\right)$ for *all* $C_0 > 0$ under the assumption $\#\mathrm{W}[0] \neq \#\mathrm{W}[1]$ for counting k-clique [23]. Finally, a lower bound of $\Omega\left(n^{c_0 \cdot k}\right)$ for *some* $c_0 > 0$ was shown by [11] (under the strong exponential time hypothesis).

Lower bound on $T^*(Q_{hard}, \mathcal{D})$	**Num. $\mathcal{P}$s**	**Hardness Assumption**
$\Omega\left(\left(T_{det}\left(\text{OPT}\left(Q_{hard}\right), D, c\right)\right)^{1+\epsilon_0}\right)$ for *some* $\epsilon_0 > 0$	Single	Triangle Detection hypothesis
$\omega\left(\left(T_{det}\left(\text{OPT}\left(Q_{hard}\right), D, c\right)\right)^{C_0}\right)$ for *all* $C_0 > 0$	Multiple	$\#\mathsf{W}[0] \neq \#\mathsf{W}[1]$
$\Omega\left(\left(T_{det}\left(\text{OPT}\left(Q_{hard}\right), D, c\right)\right)^{c_0 \cdot k}\right)$ for *some* $c_0 > 0$	Multiple	Conjecture 3.2

Table 1: Our lower bounds for a specific hard query Q_{hard} parameterized by k. For $\mathcal{D} = \left\{\{0,\ldots,c\}^D, \mathcal{P}\right\}$ those with 'Multiple' in the second column need the algorithm to be able to handle multiple $\mathcal{P}$, i.e. probability distributions (for a given D). The last column states the hardness assumptions that imply the lower bounds in the first column (ϵ_o, C_0, c_0 are constants that are independent of k).

All of our results rely on working with a *reduced* form $\left(\widetilde{\Phi}\right)$ of the lineage polynomial Φ. In fact, it turns out that for the 1-TIDB case, computing the expected multiplicity (over bag query semantics) is *exactly* the same as evaluating this reduced polynomial over the probabilities that define the 1-TIDB. This is also true when the query input(s) is a block independent disjoint probabilistic database [40] (bag query semantics with tuple multiplicity at most 1), for which the proof of Lemma 1.4 (introduced shortly) holds .

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

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

It can be verified that $\Phi\left(A,B,C,E,X,Y,Z\right)$ for the sole result tuple of Q_1 is $AXB+BYE+BZC$. Now consider the product query $Q_1^2 = Q_1 \times Q_1$. The lineage polynomial for Q_1^2 is given by $\Phi_1^2\left(A,B,C,E,X,Y,Z\right)$

$$= A^2X^2B^2 + B^2Y^2E^2 + B^2Z^2C^2 + 2AXB^2YE + 2AXB^2ZC + 2B^2YEZC.$$

To compute $\mathbb{E}\left[\Phi_1^2\right]$ we can use linearity of expectation and push the expectation through each summand. To keep things simple, let us focus on the monomial $\Phi_1^{(ABX)^2} = A^2X^2B^2$ as the procedure is the same for all other monomials of Φ_1^2. Let W_X be the random variable corresponding to a lineage variable X. Because the distinct variables in the product are independent, we can push expectation through them yielding $\mathbb{E}\left[W_A^2W_X^2W_B^2\right] = \mathbb{E}\left[W_A^2\right]\mathbb{E}\left[W_X^2\right]\mathbb{E}\left[W_B^2\right]$. Since $W_A, W_B \in \{0,1\}$ we can further derive $\mathbb{E}\left[W_A\right]\mathbb{E}\left[W_X^2\right]\mathbb{E}\left[W_B\right]$ by the fact that for any $W \in \{0,1\}$, $W^2 = W$. Observe that if $X \in \{0,1\}$, then we further would have $\mathbb{E}\left[W_A\right]\mathbb{E}\left[W_X\right]\mathbb{E}\left[W_B\right] = p_A \cdot p_X \cdot p_B$ (denoting $Pr\left[W_A = 1\right] = p_A$) $= \widetilde{\Phi}_1^{(ABX)^2}\left(p_A, p_X, p_B\right)$ (see *ii*) of Definition 1.3). However, in this example, we get stuck with $\mathbb{E}\left[W_X^2\right]$, since $W_X \in \{0,1,2\}$ and for $W_X \leftarrow 2$, $W_X^2 \neq W_X$.

Denote the variables of Φ to be $\textsc{Vars}\left(\Phi\right)$. In the c-TIDB setting, $\Phi\left(\mathbf{X}\right)$ has an equivalent reformulation $\left(\Phi_R\left(\mathbf{X_R}\right)\right)$ that is of use to us, where $|\mathbf{X_R}| = c \cdot |\mathbf{X}|$. Given $X_t \in \textsc{Vars}\left(\Phi\right)$ and integer valuation $X_t \in \{0,\ldots,c\}$. We can replace X_t by $\sum_{j\in[c]} jX_{t,j}$ where the variables $\left(X_{t,j}\right)_{j\in[c]}$ are disjoint with integer assignments $X_t \in \{0,1\}$. Then for any $\mathbf{W} \in \{0,\ldots,c\}^D$ and corresponding reformulated world $\mathbf{W_R} \in \{0,1\}^{Dc}$, we set $\mathbf{W}_{\mathbf{R}_{t,j}} = 1$ for $\mathbf{W}_t = j$, while $\mathbf{W}_{\mathbf{R}_{t,j'}} = 0$ for all $j' \neq j \in [c]$. By construction then $\Phi\left(\mathbf{X}\right) \equiv \Phi_R\left(\mathbf{X_R}\right)$ ($\mathbf{X_R} = \textsc{Vars}\left(\Phi_R\right)$) since for any integer valuation $X_t \in [c]$, $X_j \in \{0,1\}$ we have the equality $X_t = j = \sum_{j\in[c]} jX_j$.

Considering again our example,

$$\Phi_{1,R}^{(ABX)^2}\left(A,X,B\right) = \Phi_1^{(AXB)^2}\left(\sum_{j_1\in[c]} j_1A_{j_1}, \sum_{j_2\in[c]} j_2X_{j_2}, \sum_{j_3\in[c]} j_3B_{j_3}\right)$$
$$= \left(\sum_{j_1\in[c]} j_1A_{j_1}\right)^2\left(\sum_{j_2\in[c]} j_2X_{j_2}\right)^2\left(\sum_{j_3\in[c]} j_3B_{j_3}\right)^2.$$

Since the set of multiplicities for tuple t by nature are disjoint we can drop all cross terms and have $\Phi_{1,R}^2 = \sum_{j_1,j_2,j_3\in[c]} j_1^2A_{j_1}^2j_2^2X_{j_2}^2j_3^2B_{j_3}^2$. Computing expectation we get $\mathbb{E}\left[\Phi_{1,R}^2\right] = \sum_{j_1,j_2,j_3\in[c]} j_1^2j_2^2j_3^2\,\mathbb{E}\left[W_{A_{j_1}}\right]\mathbb{E}\left[W_{X_{j_2}}\right]\mathbb{E}$ since we now have that all $W_{X_j} \in \{0,1\}$. This leads us to consider a structure related to the lineage polynomial.

Definition 1.3. *For any polynomial $\Phi\left(\left(X_t\right)_{t\in D}\right)$ define the reformulated polynomial $\Phi_R\left(\left(X_{t,j}\right)_{t\in D, j\in[c]}\right)$ to be the polynomial Φ_R $= \Phi\left(\left(\sum_{j\in[c]} j \cdot X_{t,j}\right)_{t\in D}\right)$ and ii) define the* reduced polynomial $\widetilde{\Phi}\left(\left(X_{t,j}\right)_{t\in D, j\in[c]}\right)$ *to be the polynomial resulting from converting Φ_R into the standard monomial basis (SMB),* [4] *removing all monomials containing the term $X_{t,j}X_{t,j'}$ for $t \in D$, $j \neq j' \in [c]$, and setting all* variable *exponents $e > 1$ to* 1.

Continuing with the example [5] $\Phi_1^2\left(A,B,C,E,X_1,X_2,Y,Z\right)$ we have

$$\widetilde{\Phi}_1^2(A,B,C,E,X_1,X_2,Y,Z) =$$
$$A\left(\sum_{j\in[c]} j^2X_j\right)B + BYE + BZC + 2A\left(\sum_{j\in[c]} j^2X_j\right)BYE + 2A\left(\sum_{j\in[c]} j^2X_j\right)BZC + 2BYEZC$$
$$ABX_1 + AB\left(2\right)^2X_2 + BYE + BZC + 2AX_1BYE + 2A\left(2\right)^2X_2BYE + 2AX_1BZC + 2A\left(2\right)^2X_2$$

Note that we have argued that for our specific example the expectation that we want is $\widetilde{\Phi}_1^2(Pr\left(A=1\right), Pr\left(B=1\right), Pr\left(C=1\right)), Pr\left(E=1\right), Pr\left(X_1 = 1\right.$ Lemma 1.4 generalizes the equivalence to *all* $\mathcal{RA}^+$ queries on c-TIDBs (proof in Appendix B.5).

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

1.2 Our Techniques

Lower Bound Proof Techniques. Our main hardness result shows that computing Problem 1.1 is $\#\mathsf{W}[1]-$*hard* for 1-TIDB. To prove

[4] This is the representation, typically used in set-PDBs, where the polynomial is reresented as sum of 'pure' products. See Definition 2.1 for a formal definition.

[5] To save clutter we do not show the full expansion for variables with greatest multiplicity = 1 since e.g. for variable A, the sum of products itself evaluates to $1^2 \cdot A^2 = A$.

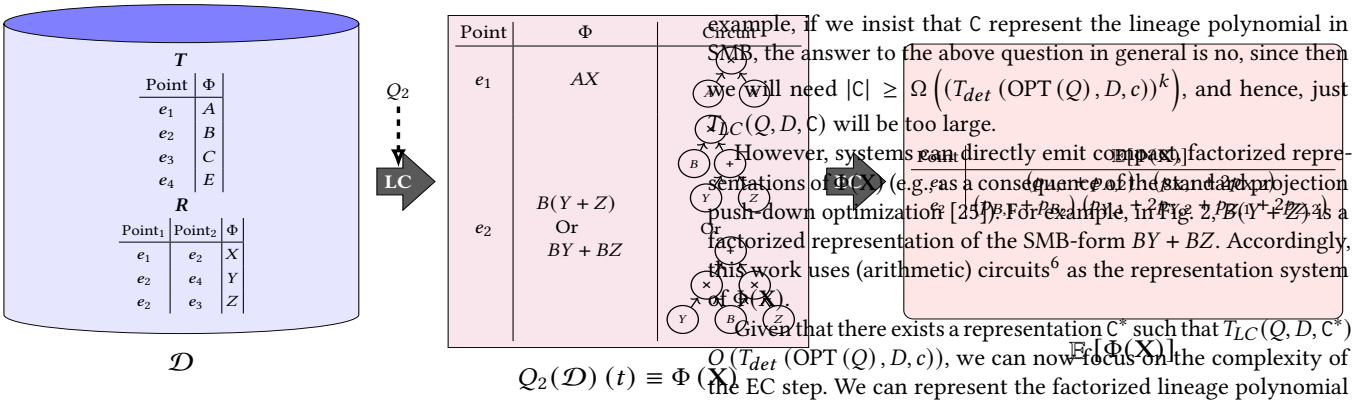


Figure 2: Intensional Query Evaluation Model ($Q_2 = \pi_{\textbf{Point}}\left(T \bowtie_{\textbf{Point=Point}_1} R\right)$ **where, for table** R, $c = 2$**, while for** T, $c = 1$.)

this result we show that for the same Q_1 from the example above, for an arbitrary 'product width' k, the query Q^k_{hard} is able to encode various hard graph-counting problems (assuming $O(n)$ tuples rather than the $O(1)$ tuples in Fig. 2). We do so by considering an arbitrary graph G (analogous to relation $\boldsymbol{R}$ of Q_1) and analyzing how the coefficients in the (univariate) polynomial $\widetilde{\Phi}(p, \ldots, p)$ relate to counts of subgraphs in G that are isomorphic to various graphs with k edges. E.g., we exploit the fact that the coefficient corresponding to the power of $2k$ in Φ of Q^k_{hard} is proportional to the number of k-matchings in G, a known hard problem in parameterized/fine-grained complexity literature.

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

We adopt a two-step intensional model of query evaluation used in set-PDBs, as illustrated in Fig. 2: (i) Lineage Computation (LC): Given input D and Q, output every tuple t that possibly satisfies Q, annotated with its lineage polynomial ($\Phi(\mathbf{X}) = \Phi[Q, D, t]\,(\mathbf{X})$); (ii) Expectation Computation (EC): Given $\Phi(\mathbf{X})$ for each tuple, compute $\mathbb{E}_{W\sim\mathcal{P}}[\Phi(\mathbf{W})]$. Let $T_{LC}(Q, D, \mathtt{C})$ denote the runtime of LC when it outputs $\mathtt{C}$ (which is a representation of Φ as an arithmetic circuit — more on this representation in Sec. 2.3). Denote by $T_{EC}(\mathtt{C}, \epsilon)$ (recall $\mathtt{C}$ is the output of LC) the runtime of EC, which we can leverage Definition 1.3 and Lemma 1.4 to address the next formal objective:

Problem 1.5 (c-TIDB linear time approximation). *Given c-TIDB $\mathcal{D}, \mathcal{RA}^+$ query Q, is there a $(1\pm\epsilon)$-approximation of $\mathbb{E}_{\mathbf{W}\sim\mathcal{P}}[Q(\mathbf{W})(t)]$ for all result tuples t where $\exists \mathtt{C} : T_{LC}(Q, D, \mathtt{C}) + T_{EC}(\mathtt{C}, \epsilon) \leq O_\epsilon(T_{det}(OPT(Q), D, c))$?*

We show in Appendix E.2.1 an $O(T_{det}(\mathrm{OPT}(Q), D, c))$ algorithm for constructing the lineage polynomial for all result tuples of an $\mathcal{RA}^+$ query Q (or more more precisely, a single circuit $\mathtt{C}$ with one sink per tuple representing the tuple's lineage). A key insight of this paper is that the representation of $\mathtt{C}$ matters. For example, if we insist that $\mathtt{C}$ represent the lineage polynomial in SMB, the answer to the above question in general is no, since then we will need $|\mathtt{C}| \geq \Omega\left((T_{det}(\mathrm{OPT}(Q), D, c))^k\right)$, and hence, just $T_{LC}(Q, D, \mathtt{C})$ will be too large.

However, systems can directly emit compact factorized representations of $\Phi(\mathbf{X})$ (e.g., as a consequence of standard projection push-down optimization [25]). For example, in Fig. 2, $B(Y + Z)$ is a factorized representation of the SMB-form $BY + BZ$. Accordingly, this work uses (arithmetic) circuits[6] as the representation system of $\Phi(\mathbf{X})$.

Given that there exists a representation $\mathtt{C}^*$ such that $T_{LC}(Q, D, \mathtt{C}^*) \leq O(T_{det}(\mathrm{OPT}(Q), D, c))$, we can now focus on the complexity of the EC step. We can represent the factorized lineage polynomial by its correspoding arithmetic circuit $\mathtt{C}$ (whose size we denote by $|\mathtt{C}|$). As we also show in Appendix E.2.2, this size is also bounded by $T_{det}(\mathrm{OPT}(Q), D, c)$ (i.e., $|\mathtt{C}^*| \leq O(T_{det}(\mathrm{OPT}(Q), D, c))$). Thus, the question of approximation can be stated as the following stronger (since Problem 1.5 has access to *all* equivalent $\mathtt{C}$ representing $Q(\mathbf{W})(t)$), but sufficient condition:

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

For an upper bound on approximating the expected count, it is easy to check that if all the probabilties are constant then (with an additive adjustment) $\Phi(p_1, \ldots, p_n)$

(i.e. evaluating the original lineage polynomial over the probability values) is a constant factor approximation . This is illustrated in the following example using Q^2_1 from earlier. To aid in presentation we assume $c = 2$ for variable X and $c = 1$ for all other variables. Let p_A denote $Pr[A = 1]$. In computing $\widetilde{\Phi}$, we have some cancellations to deal with:

$$\Phi^2_{1,R}(\mathbf{X}) = A^2\left(X_1^2 + 4X_1X_2 + 4X_2^2\right)B^2 + B^2Y^2E^2 + B^2Z^2C^2 + 2AX_1B^2YE + 2AX_2B^2YE + 2AX_1B^2ZC + 2AX_2B^2ZC + 2B^2YEZC$$

This then implies

$$\widetilde{\Phi}^2(\mathbf{X}) = AX_1B + 4AX_2B + BYE + BZC + 2AX_1BYE + 2AX_2BYE + 2AX_1BZC + 2AX_2BZC + 2BYEZC$$

Substituting $\mathbf{p}$ for $\mathbf{X}$,

$$\begin{aligned}\Phi^2_{1,R}(\mathbf{p}) &= p_A^2p_{X_1}^2p_B^2 + 4p_A^2p_{X_1}p_{X_2}p_B^2 + 4p_A^2p_{X_2}^2p_B^2 + p_B^2p_Y^2p_E^2 + p_B^2p_Z^2p_C^2 + 2p_Ap_{X_1}p_B^2p_Yp_E + 2p_Ap_{X_1}p_B^2p_Zp_C + 2p_Ap_{X_2}p_B^2p_Zp_C + 2p_B^2p_Yp_Ep_Zp_C \\ &\leq p_Ap_{X_1}p_B + 4p_A^2p_{X_1}p_{X_2}p_B^2 + 4p_Ap_{X_2}p_b + p_Bp_Yp_E + p_Bp_Zp_C + 2p_Ap_{X_1}p_Bp_Yp_E + 2p_Ap_{X_1}p_Bp_Zp_C + 2p_Ap_{X_2}p_Bp_Zp_C + 2p_Bp_Yp_Ep_Zp_C = \widetilde{\Phi}^2_1(\mathbf{p}) + 4p_A^2p_{X_1}p_{X_2}p_B^2.\end{aligned}$$

If we assume that all probability values are at least $p_0 > 0$, then given access to $\Phi^2_{1,R}(\mathbf{p}) - 4p_A^2p_{X_1}p_{X_2}p_B^2$ we get that $\Phi^2_{1,R}(\mathbf{p}) - 4p_A^2p_{X_1}p_{X_2}p_B^2$ is in the range $\left((p_0)^3 \cdot \left(\widetilde{\Phi}^2_1\mathbf{p}\right), \widetilde{\Phi}^2_1(\mathbf{p})\right]$. Note however, that this is *not a tight approximation*. In sec. 4 we demonstrate that a $(1\pm\epsilon)$ (multiplicative) approximation with competitive performance is achievable. To get an $(1\pm\epsilon)$-multiplicative approximation

[6] An arithmetic circuit is a DAG with variable and/or numeric source nodes and internal, each nodes representing either an addition or multiplication operator.

and solve Problem 1.6, using C we uniformly sample monomials from the equivalent SMB representation of Φ (without materializing the SMB representation) and 'adjust' their contribution to $\widetilde{\Phi}(\cdot)$.

Applications. Recent work in heuristic data cleaning [8, 42, 45, 45, 51] emits a PDB when insufficient data exists to select the 'correct' data repair. Probabilistic data cleaning is a crucial innovation, as the alternative is to arbitrarily select one repair and 'hope' that queries receive meaningful results. Although PDB queries instead convey the trustworthiness of results [37], they are impractically slow [18, 19], even in approximation (see Appendix G). Bags, as we consider, are sufficient for production use, where bag-relational algebra is already the default for performance reasons. Our results show that bag-PDBs can be competitive, laying the groundwork for probabilistic functionality in production database engines.

Paper Organization. We present 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. Finally, we discuss related work in Sec. 5 and conclude in Sec. 6. All proofs are in the appendix.

2 BACKGROUND AND NOTATION

2.1 Polynomial Definition and Terminology

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

$$\phi\left((X_t)_{t\in S}\right) = \sum_{\mathbf{d}\in\{0,\ldots,K\}^S} c_{\mathbf{d}} \cdot \prod_{t\in S} X_t^{d_t} \qquad \text{where } c_{\mathbf{d}} \in \mathbb{N}. \tag{1}$$

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

Unless othewise noted, we consider all polynomials to be in SMB representation. When it is unclear, we use SMB (ϕ) (SMB (Φ)) to denote the SMB form of a polynomial (lineage polynomial) ϕ (Φ).

Definition 2.2 (Degree). *The degree of polynomial $\phi(\mathbf{X})$ is the largest $\sum_{t\in S} d_t$ for all $\mathbf{d} \in \{0,\ldots,K\}^S$ such that $c_{(d_1,\ldots,d_n)} \neq 0$. We denote the degree of ϕ as* $\deg(\phi)$.

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

We call a polynomial $\Phi(\mathbf{X})$ a *c*-TIDB-*lineage polynomial* (or simply lineage polynomial), if it is clear from context that there exists an $\mathcal{RA}^+$ query Q, *c*-TIDB $\mathcal{D}$, and result tuple t such that $\Phi(\mathbf{X}) = \Phi[Q, D, t](\mathbf{X})$.

2.2 Binary-BIDB

A block independent database BIDB $\mathcal{D}'$ models a set of worlds each of which consists of a subset of the possible tuples D', where D' is partitioned into m blocks B_i and all B_i are independent random events. $\mathcal{D}'$ further constrains that all $t \in B_i$ for all $i \in [m]$ of D' be disjoint events. We refer to any monomial that includes $X_t X_{t'}$ for $t \neq t' \in B_i$ as a *cancellation*. We define next a specific construction of BIDB that is useful for our work.

$$\Phi'\left[\pi_A(Q), \overline{D}', t_j\right] = \sum_{\substack{t_{j'},\\ \pi_A(t_{j'})=t_j}} \Phi'\left[Q, \overline{D}', t_{j'}\right] \qquad \Phi'\left[Q_1 \cup\right.$$

$$\Phi'\left[\sigma_\theta(Q), \overline{D}', t_j\right] = \begin{cases} \theta = 1 & \Phi'\left[Q, \overline{D}', t_j\right] \\ \theta = 0 & 0 \end{cases} \qquad \Phi'\left[Q_1 \bowtie\right.$$

Figure 3: Construction of the lineage (polynomial) for an $\mathcal{RA}^+$ query Q over $\overline{D}'$.

Definition 2.3 (Binary-BIDB). *Define a* Binary-BIDB *to be the pair $\mathcal{D}' = \left(\bigtimes_{t\in D'}\{0, c_t\}, \mathcal{P}'\right)$, where D' is the set of possible tuples such that each $t \in D'$ has a multiplicity domain of $\{0, c_t\}$, with $c_t \in \mathbb{N}$. D' is partitioned into m independent blocks B_i, for $i \in [m]$, of disjoint tuples. $\mathcal{P}'$ is characterized by the vector $(p_t)_{t\in D'}$ where for every block B_i, $\sum_{t\in B_i} p_t \leq 1$. Given $W \in \bigtimes_{t\in D'}\{0, c_t\}$ and for $i \in [m]$,*

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

$\mathcal{P}'$ is the probability distribution across all worlds such that, given $W \in \bigtimes_{t\in D'}\{0, c_t\}$, $Pr[\mathbf{W} = W] = \prod_{i\in[m]} p_i(W)$. [7]

Fig. 3 shows the lineage construction of $\Phi'(\mathbf{X})$ given $\mathcal{RA}^+$ query Q for arbitrary deterministic $\overline{D}'$. Note that the semantics differ from Fig. 1 only in the base case.

Proposition 2.4 (*c*-TIDB reduction). *Given c-*TIDB $\mathcal{D} = \left(\{0,\ldots,c\}^D, \mathcal{P}\right)$, *let $\mathcal{D}' = \left(\bigtimes_{t\in D'}\{0, c_t\}, \mathcal{P}'\right)$ be the* Binary-BIDB *obtained in the following manner: for each $t \in D$, create block $B_t = \left\{\bigsqcup t, j_{j\in[c]}\right\}$ of disjoint tuples, for all $j \in [c]$. The probability distribution $\mathcal{P}'$ is the characterized by the vector $\mathbf{p} = \left((p_{t,j})_{t\in D, j\in[c]}\right)$. Then, the distributions $\mathcal{P}$ and $\mathcal{P}'$ are equivalent.*

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

Definition 2.5 ($\widetilde{\Phi}'$). *Given a polynomial $\Phi'(\mathbf{X})$ generated from a Binary-BIDB and let $\widetilde{\Phi}'(\mathbf{X})$ denote the reduced form of $\Phi'(\mathbf{X})$ derived as follows: i) compute SMB $(\Phi'(\mathbf{X}))$ eliminating all monomials with cross terms $X_t X_{t'}$ for $t \neq t' \in B_i$ and ii) reduce all* variable *exponents $e > 1$ to* 1.

Then given $\mathbf{W} \in \{0,1\}^{D'}$ over the reduced Binary-BIDB of Proposition 2.4, the disjoint requirement and the semantics for constructing the lineage polynomial over a Binary-BIDB, $\Phi'(\mathbf{W})$ is of the same structure as the reformulated polynomial $\Phi_R(\mathbf{W})$ of step i) from Definition 1.3, which then implies that $\widetilde{\Phi}'$ is the reduced polynomial that results from step ii) of both Definition 1.3 and Definition 2.5, and further that Lemma 1.4 immediately follows for Binary-BIDB polynomials.

Lemma 2.6. *Given any* Binary-BIDB $\mathcal{D}'$, $\mathcal{RA}^+$ *query Q, and lineage polynomial $\Phi'(\mathbf{X}) = \Phi'[Q, D', t](\mathbf{X})$, it holds that $\mathbb{E}_{\mathbf{W}\sim\mathcal{P}'}[\Phi'(\mathbf{W})] = \widetilde{\Phi}'(\mathbf{p})$.*

[7] We slightly abuse notation here, denoting a world vector as W rather than $\mathbf{W}$ to distinguish between the random variable and the world instance. When there is no ambiguity, we will denote a world vector as $\mathbf{W}$.

Let $|\Phi|$ be the number of operators in Φ.

Corollary 2.7. *If Φ is a* 1-BIDB *lineage polynomial already in SMB, then the expectation of Φ, i.e., $\mathbb{E}[\Phi] = \widetilde{\Phi}(p_1, \ldots, p_n)$ can be computed in $O(|\Phi|)$ time.*

2.2.1 Possible World Semantics. In this section, we show how the traditional possible worlds semantics corresponds to our setup. Readers can safely skip this part without missing anything vital to the results of this paper.

Queries over probabilistic databases are traditionally viewed as being evaluated using the so-called possible world semantics. A general bag-PDB can be defined as the pair $\mathcal{D} = (\Omega, \mathcal{P})$ where Ω is the set of possible worlds represented by $\mathcal{D}$ and $\mathcal{P}$ the probability distribution over Ω. Under the possible 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 $\omega \in \Omega$: $\{Q(\omega) : \omega \in \Omega\}$. The result of a query is the pair $(Q(\Omega), \mathcal{P}')$ where $\mathcal{P}'$ is a probability distribution that assigns to each possible query result the sum of the probabilites of the worlds that produce this answer: $Pr[\omega \in \Omega] = \sum_{\omega' \in \Omega, Q(\omega') = Q(\omega)} Pr[\omega']$.

Suppose that $\mathcal{D}'$ is a reduced Binary-BIDB from c-TIDB $\mathcal{D}$ as defined by **??**. Instead of looking only at the possible worlds of $\mathcal{D}'$, one can consider the set of all worlds, including those that cannot exist due to, e.g., disjointness. Since $|D| = n$ the all worlds set can be modeled by $\mathbf{W} \in \{0,1\}^{nc}$, such that $\mathbf{W}_{t,j} \in \mathbf{W}$ represents whether or not the multiplicity of t is j (*here and later, especially in Sec. 4, we will rename the variables as $X_1, \ldots, X_{n'}$, where $n' = \sum_{t \in D} |B_t|$*). [8] We can denote a probability distribution over all $\mathbf{W} \in \{0,1\}^{nc}$ as $\mathcal{P}'$. When $\mathcal{P}'$ is the one induced from each $p_{t,j}$ while assigning $Pr[\mathbf{W}] = 0$ for any $\mathbf{W}$ with $\mathbf{W}_{t,j}, \mathbf{W}_{t,j'} \neq 0$ for $j \neq j'$, we end up with a bijective mapping from $\mathcal{P}$ to $\mathcal{P}'$, such that each mapping is equivalent, implying the distributions are equivalent, and thus query results. Appendix B.2 has more details.

We now make a meaningful connection between possible world semantics and world assignments on the lineage polynomial.

Proposition 2.8 (Expectation of polynomials). *Given a* bag-PDB $\mathcal{D} = (\Omega, \mathcal{P})$, *$\mathcal{RA}^+$ query Q, and lineage polynomial $\Phi[Q, D, t]$ for arbitrary result tuple t, we have (denoting $\mathbf{D}$ as the random variable over Ω):* $\mathbb{E}_{\mathbf{D} \sim \mathcal{P}}[Q(\mathbf{D})(t)] = \mathbb{E}_{\mathbf{W} \sim \mathcal{P}}[\Phi[Q, D, t](\mathbf{W})]$.

A formal proof of Proposition 2.8 is given in Appendix B.3.[9]

2.3 Formalizing Problem 1.6

We focus on the problem of computing $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}[\Phi[Q, D, t](\mathbf{W})]$ from now on, assume implicit Q, D, t, and drop them from $\Phi[Q, D, t]$ (i.e., $\Phi(\mathbf{X})$ will denote a polynomial).

Problem 1.6 asks if there exists a linear time approximation algorithm in the size of a given circuit C which encodes $\Phi(\mathbf{X})$. Recall that in this work we represent lineage polynomials via *arithmetic circuits* [9], 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. Since we are specifically using circuits to model lineage polynomials, we can refer to these circuits as lineage circuits. However, when the meaning is clear, we will drop the term lineage and only refer to them as circuits.

Definition 2.9 (Circuit). *A circuit* C *is a Directed Acyclic Graph (DAG) whose source gates (in degree of 0) consist of elements in either $\mathbb{N}$ or $\mathbf{X} = (X_1, \ldots, X_n)$. For each result tuple there exists one sink gate. The internal gates have binary input and are either sum (+) or product ($\times$) gates. Each gate has the following members:* `type`, `input`, `val`, `partial`, `degree`, `Lweight`, *and* `Rweight`, *where* `type` *is the value type* $\{+, \times, \textit{VAR}, \textit{NUM}\}$ *and* `input` *the list of inputs. Source gates have an extra member* `val` *storing the value.* C_L (C_R) *denotes the left (right) input of* C.

When the underlying DAG is a tree (with edges pointing towards the root), the structure is an expression tree T. In such a case, the root of T is analogous to the sink of C. The fields `partial`, `degree`, `Lweight`, and `Rweight` are used in the proofs of Appendix D.

The circuits in Fig. 2 encode their respective polynomials in column Φ. Note that the ciricuit C representing AX and the circuit C' representing $B(Y + Z)$ each encode a tree, with edges pointing towards the root.

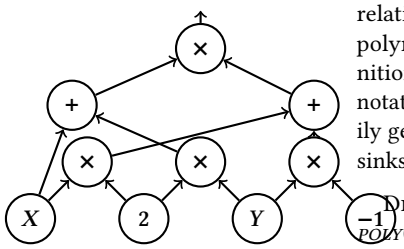


Figure 4: Circuit encoding of $(X + 2Y)(2X - Y)$

We next formally define the relationship of circuits with polynomials. While the definition assumes one sink for notational convenience, it easily generalizes to the multiple sinks case.

Definition 2.10 (poly($\cdot$)). *poly(C) maps the sink of circuit* C *to its corresponding polynomial (in SMB). poly($\cdot$) is recursively defined on* C *as follows, with addition and multiplication following the standard interpretation for polynomials:*

$$\textit{poly}(\mathsf{C}) = \begin{cases} \textit{poly}(\mathsf{C}_\mathsf{L}) + \textit{poly}(\mathsf{C}_\mathsf{R}) & \text{if } \mathsf{C}.\mathtt{type} = + \\ \textit{poly}(\mathsf{C}_\mathsf{L}) \cdot \textit{poly}(\mathsf{C}_\mathsf{R}) & \text{if } \mathsf{C}.\mathtt{type} = \times \\ \mathsf{C}.\mathtt{val} & \text{if } \mathsf{C}.\mathtt{type} = \textit{VAR OR NUM}. \end{cases}$$

C need not encode $\Phi(\mathbf{X})$ in the same, default SMB representation. For instance, C could encode the factorized representation $(X + 2Y)(2X - Y)$ of $\Phi(\mathbf{X}) = 2X^2 + 3XY - 2Y^2$, as shown in Fig. 4, while $\textit{poly}(\mathsf{C}) = \Phi(\mathbf{X})$ is always the equivalent SMB representation.

Definition 2.11 (Circuit Set). $\mathtt{CSet}(\Phi(\mathbf{X}))$ *is the set of all possible circuits* C *such that* $\textit{poly}(\mathsf{C}) = \Phi(\mathbf{X})$.

The circuit of Fig. 4 is an element of $\mathtt{CSet}(2X^2 + 3XY - 2Y^2)$. One can think of $\mathtt{CSet}(\Phi(\mathbf{X}))$ as the infinite set of circuits where for each element C, $\textit{poly}(\mathsf{C}) = \Phi(\mathbf{X})$.

We are now ready to formally state the final version of Problem 1.6.

Definition 2.12 (The Expected Result Multiplicity Problem). *Let $\mathcal{D}'$ be an arbitrary c-*TIDB *and $\mathbf{X}$ be the set of variables annotating tuples in D'. Fix an $\mathcal{RA}^+$ query Q and a result tuple t. The Expected Result Multiplicity Problem is defined as follows:*

Input: $\mathsf{C} \in \mathtt{CSet}(\Phi(\mathbf{X}))$ *for* $\Phi'(\mathbf{X}) = \Phi'[Q, D', t]$ ***Output***: $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}[\Phi'[Q, D', t](\mathbf{W})]$

[8] In this example, $|B_t| = c$ for all t.

[9] Although Proposition 2.8 follows, e.g., as an obvious consequence of [30]'s Theorem 7.1, we are unaware of any formal proof for bag-probabilistic databases.

2.4 Relationship to Deterministic Query Runtimes

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

Definition 2.13 (Join Cost). *Denote by $T_{join}(R_1, \ldots, R_m)$ the runtime of an algorithm for computing the m-ary join $R_1 \bowtie \ldots \bowtie R_m$. We require only that the algorithm must enumerate its output, i.e., that $T_{join}(R_1, \ldots, R_m) \geq |R_1 \bowtie \ldots \bowtie R_m|$. With this definition of $T_{join}(\cdot)$, worst-case optimal join algorithms are handled.*

Worst-case optimal join algorithms [38, 39] and query evaluation via factorized databases [41] (as well as work on FAQs [35]) can be modeled as $\mathcal{RA}^+$ queries (though the query size is data dependent). For these algorithms, $T_{join}(R_1, \ldots, R_n)$ is linear in the *AGM bound* [6]. Our cost model for general query evaluation follows from the join cost:

$$T_{det}\left(R, \overline{D}, c\right) = \left|Q(\overline{D})\right| \qquad T_{det}\left(\sigma_\theta(Q), \overline{D}, c\right) = T_{det}\left(Q, \overline{D}, c\right) + \left|Q(\overline{D})\right|$$

$$T_{det}\left(\pi_A(Q), \overline{D}, c\right) = T_{det}\left(Q, \overline{D}, c\right) + \left|Q(\overline{D})\right| \qquad T_{det}\left(Q_1 \cup Q_2, \overline{D}, c\right) = T_{det}\left(Q_1, \overline{D}, c\right) + T_{det}\left(Q_2, \overline{D}, c\right) + \left|Q(\overline{D})\right|$$

$$T_{det}\left(Q_1 \bowtie \ldots \bowtie Q_m, \overline{D}, c\right) = T_{det}\left(Q_1, \overline{D}, c\right) + \ldots + T_{det}\left(Q_m, \overline{D}, c\right) + T_{join}(Q_1(\overline{D}), \ldots, Q_m(\overline{D}))$$

Under this model, an $\mathcal{RA}^+$ query Q evaluated over database $\overline{D}$ has runtime $O(T_{det}\left(Q, \overline{D}, c\right))$. 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.

Lemma E.2 and Lemma E.3 show that for any $\mathcal{RA}^+$ query Q and D, there exists a circuit $\mathtt{C}^*$ such that $T_{LC}(Q, D, \mathtt{C}^*)$ and $|\mathtt{C}^*|$ are both $O(T_{det}(\text{OPT}(Q), D, c))$. Recall we assumed these two bounds when we moved from Problem 1.5 to Problem 1.6. Lastly, we can handle FAQs and factorized databases by allowing for optimization, i.e. OPT (Q).

3 HARDNESS OF EXACT COMPUTATION

In this section, we will prove the hardness results claimed in Table 1 for a specific (family) of hard instance $(Q_{hard}, \mathcal{D})$ for Problem 1.2 where $\mathcal{D}$ is a 1-TIDB. Note that this implies hardness for c-TIDBs $(c \geq 1)$, showing Problem 1.2 cannot be done in $O(T_{det}(\text{OPT}(Q), D, c))$ runtime. The results also apply to Binary-BIDB and other more general PDBs.

3.1 Preliminaries

Our hardness results are based on (exactly) counting the number of (not necessarily induced) subgraphs in G isomorphic to H. Let $\#(G, H)$ denote this quantity. 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 in its adjacency list representation): $\#(G, \text{△})$ (the number of triangles), $\#(G, \text{⫶})$ (the number of 3-matchings), and the latter's generalization $\#\left(G, \text{⫶}^k\right)$ (the number of k-matchings). We use $T_{match}(k, G)$ to denote the optimal runtime of computing $\#\left(G, \text{⫶}^k\right)$ exactly. Our hardness results in Sec. 3.2 are based on the following hardness results/conjectures:

Theorem 3.1 ([12]). *Given positive integer k and undirected graph $G = (V, E)$ with no self-loops or parallel edges, $T_{match}(k, G) \geq \omega(f(k) \cdot |E|^c)$ for any function f and any constant c independent of $|E|$ and k (assuming $\#\mathrm{W}[0] \neq \#\mathrm{W}[1]$).*

Conjecture 3.2. *There exists an absolute constant $c_0 > 0$ such that for every $G = (V, E)$, we have $T_{match}(k, G) \geq \Omega\left(|E|^{c_0 \cdot k}\right)$ for large enough k.*

We note that the above conjecture is somewhat non-standard. In particular, the best known algorithm to compute $\#\left(G, \text{⫶}^k\right)$ takes time $\Omega\left(|V|^{k/2}\right)$ (i.e. if this is the best algorithm then $c_0 = \frac{1}{4}$) [12]. What the above conjecture is saying is that one can only hope for a polynomial improvement over the state of the art algorithm to compute $\#\left(G, \text{⫶}^k\right)$.

Our hardness result in Section 3.3 is based on the following conjectured hardness result:

Conjecture 3.3. *There exists a constant $\epsilon_0 > 0$ such that given an undirected graph $G = (V, E)$, computing $\#(G, \text{△})$ exactly cannot be done in time $o\left(|E|^{1+\epsilon_0}\right)$.*

The so called *Triangle detection hypothesis* (cf. [36]), which states that detecting the presence of triangles in G takes time $\Omega\left(|E|^{4/3}\right)$, implies that in Conjecture 3.3 we can take $\epsilon_0 \geq \frac{1}{3}$.

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

Definition 3.4. *For any graph $G = (V, E)$ and $k \geq 1$, define*

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

Returning to Fig. 2, it can be seen that $\Phi_G^k(\mathbf{X})$ is the lineage polynomial from query Q_{hard}^k, which we define next (Q_2 from Sec. 1 is the same query with $k = 2$). Let us alias

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

as R. The query Q_{hard}^k then becomes

```
SELECT COUNT(*) FROM R JOIN R JOIN ··· JOIN R
                                  (k times)
```

Consider again the c-TIDB instance $\mathcal{D}$ of Fig. 2 and, for our hard instance, let $c = 1$. $\mathcal{D}$ generalizes to one compatible to Definition 3.4 as follows. Relation T has n tuples corresponding to each vertex for i in $[n]$, each with probability p and R has tuples corresponding to the edges E (each with probability of 1).[10] In other words, this instance D contains the set of n unary tuples in T (which corresponds to V) and m binary tuples in R (which corresponds to E). Note that this implies that Φ_G^k is indeed a 1-TIDB lineage polynomial.

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

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

Lemma 3.5. *Let Q^k_{hard} and D be as defined above. Then $T_{det}\left(Q^k_{hard}, D\right)$ is $O_k(m)$.*

3.2 Multiple Distinct p Values

We are now ready to present our main hardness result.

Theorem 3.6. *Let $p_0, \ldots, p_{2k}$ be $2k+1$ distinct values in $(0,1]$. Then computing $\widetilde{\Phi}^k_G(p_i, \ldots, p_i)$ (over all $i \in [2k+1]$) for arbitrary $G = (V, E)$ needs time $\Omega(T_{match}(k, G))$, assuming $T_{match}(k, G) \geq \omega(|E|)$.*

Note that the second row of Table 1 follows from Proposition 2.8, Theorem 3.6, Lemma 3.5, and Theorem 3.1 while the third row is proved by Proposition 2.8, Theorem 3.6, Lemma 3.5, and Conjecture 3.2. Since Conjecture 3.2 is non-standard, the latter hardness result should be interpreted as follows. Any substantial polynomial improvement for Problem 1.2 (over the trivial algorithm that converts Φ into SMB and then uses Corollary 2.7 for EC) would lead to an improvement over the state of the art *upper* bounds on $T_{match}(k, G)$. Finally, note that Theorem 3.6 needs one to be able to compute the expected multiplicities over $(2k+1)$ distinct values of p_i, each of which corresponds to distinct $\mathcal{P}$ (for the same D), which explain the 'Multiple' entry in the second column in the second and third row in Table 1. Next, we argue how to get rid of this latter requirement.

3.3 Single p value

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

Theorem 3.7. *Fix $p \in (0,1)$. Then assuming Conjecture 3.3 is true, any algorithm that computes $\widetilde{\Phi}^3_G(p, \ldots, p)$ for arbitrary $G = (V, E)$ exactly has to run in time $\Omega\left(|E|^{1+\epsilon_0}\right)$, where ϵ_0 is as defined in Conjecture 3.3.*

Note that Proposition 2.8 and Theorem 3.7 above imply the hardness result in the first row of Table 1. We note that Theorem 3.1 and Conjecture 3.2 (and the lower bounds in the second and third row of Table 1) need k to be large enough (in particular, we need a family of hard queries). But the above Theorem 3.7 (and the lower bound in first row of Table 1) holds for $k = 3$ (and hence for a fixed query).

4 $1 \pm \epsilon$ APPROXIMATION ALGORITHM

In Sec. 3, we showed that Problem 1.2 cannot be solved in $O(T_{det}(\mathrm{OPT}(Q), D, c))$ runtime. In light of this, we desire to produce an approximation algorithm that runs in time $O(T_{det}(\mathrm{OPT}(Q), D, c))$. We do this by showing the result via circuits, such that our approximation algorithm for this problem runs in $O(|\mathtt{C}|)$ for a very broad class of circuits, (thus affirming Problem 1.6); see the discussion after Lemma 4.9 for more. The following approximation algorithm applies to bag query semantics over both c-TIDB lineage polynomials and general BIDB lineage polynomials in practice, where for the latter we note that a 1-TIDB is equivalently a BIDB (blocks are size 1). Our experimental results (see Appendix D.11) which use queries from the PDBench benchmark [1] show a low γ (see Definition 4.6) supporting the notion that our bounds hold for general BIDB in practice.

Corresponding proofs and pseudocode for all formal statements and algorithms can be found in Appendix D.

4.1 Preliminaries and some more notation

We now introduce definitions and notation related to circuits and polynomials that we will need to state our upper bound results. First we introduce the expansion $\mathtt{E(C)}$ of circuit $\mathtt{C}$ which is used in our auxiliary algorithm SampleMonomial for sampling monomials when computing the approximation.

Definition 4.1 ($\mathtt{E(C)}$). *For a circuit $\mathtt{C}$, we define $\mathtt{E(C)}$ as a list of tuples $(\mathtt{v}, \mathtt{c})$, where $\mathtt{v}$ is a set of variables and $\mathtt{c} \in \mathbb{N}$. $\mathtt{E(C)}$ has the following recursive definition ($\circ$ is list concatenation).* $\mathtt{E(C)} =$

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

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

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

We will overload notation and use $|\mathtt{C}|(\mathbf{X})$ to mean $\textsc{poly}(|\mathtt{C}|)$. Conveniently, $|\mathtt{C}|(1, \ldots, 1)$ gives us $\sum_{(\mathtt{v},\mathtt{c}) \in \mathtt{E(C)}} |\mathtt{c}|$.

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

Definition 4.4 ($\deg(\cdot)$). [11] *$DEG(\mathtt{C})$ is defined recursively as follows:*

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

Next, we use the following notation for the complexity of multiplying integers:

Definition 4.5 ($\overline{M}(\cdot, \cdot)$). [12] *In a RAM model of word size of W-bits, $\overline{M}(M, W)$ denotes the complexity of multiplying two integers*

[11] Note that the degree of $\textsc{poly}(|\mathtt{C}|)$ 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 $\textsc{poly}(|\mathtt{C}|)$ is 0).

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

represented with M-bits. (We will assume that for input of size N, $W = O(\log N)$.)

Finally, to get linear runtime results, we will need to define another parameter modeling the (weighted) number of monomials in $\mathsf{E}(\mathsf{C})$ that need to be 'canceled' when monomials with dependent variables are removed (Sec. 2.2). Let $\textsc{isInd}(\cdot)$ be a boolean function returning true if monomial $\mathsf{v}_\mathtt{m}$ is composed of independent variables and false otherwise; further, let $\mathbb{1}_\theta$ also be a boolean function returning true if θ evaluates to true.

Definition 4.6 (Parameter γ). *Given a Binary-BIDB circuit* C *define*

$$\gamma(\mathsf{C}) = \frac{\sum_{(\mathsf{v},\mathsf{c})\in\mathsf{E}(\mathsf{C})} |\mathsf{c}| \cdot \mathbb{1}_{\neg \textsc{isInd}(\mathsf{v}_\mathtt{m})}}{|\mathsf{C}|\,(1,\ldots,1)}.$$

4.2 Our main result

We solve Problem 1.6 for any fixed $\epsilon > 0$ in what follows.

Algorithm Idea. Our approximation algorithm ($\textsc{ApproximateE}\widetilde{\Phi}$ pseudo code in Appendix D.1) is based on the following observation. Given a lineage polynomial $\Phi(\mathbf{X}) = \textsc{poly}(\mathsf{C})$ for circuit C over Binary-BIDB (recall that all c-TIDB can be reduced to Binary-BIDB by Proposition 2.4), we have:

$$\widetilde{\Phi}\,(p_1,\ldots,p_n) = \sum_{(\mathsf{v},\mathsf{c})\in\mathsf{E}(\mathsf{C})} \mathbb{1}_{\textsc{isInd}(\mathsf{v}_\mathtt{m})} \cdot \mathsf{c} \cdot \prod_{X_i\in\mathsf{v}} p_i. \tag{2}$$

Given the above, the algorithm is a sampling based algorithm for the above sum: we sample (via SampleMonomial) $(\mathsf{v},\mathsf{c}) \in \mathsf{E}(\mathsf{C})$ with probability proportional to $|\mathsf{c}|$ and compute $\mathsf{Y} = \mathbb{1}_{\textsc{isInd}(\mathsf{v}_\mathtt{m})} \cdot \prod_{X_i\in\mathsf{v}} p_i$. Repeating the sampling an appropriate number of times 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 D).

Runtime analysis. We can argue the following runtime for the algorithm outlined above:

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

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

can be computed in time

$$O\left(\left(\textsc{size}(\mathsf{C}) + \frac{\log\frac{1}{\delta}\cdot k\cdot\log k\cdot \textsc{depth}(\mathsf{C}))}{(\epsilon')^2\cdot(1-\gamma)^2\cdot p_0^{2k}}\right)\cdot \overline{\mathcal{M}}\left(\log\left(|\mathsf{C}|\,(1,\ldots,1)\right), \log\left(\textsc{size}(\mathsf{C})\right)\right)\right) \tag{4}$$

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

The restriction on γ is satisfied by any 1-TIDB (where $\gamma = 0$ in the equivalent 1-BIDB of Proposition 2.4) as well as for all three queries of the PDBench BIDB benchmark (see Appendix D.11 for experimental results). Further, we can also argue the following result, recalling from Sec. 1 for c-TIDB $\mathcal{D} = \left(\{0,\ldots,c\}^D, \mathcal{P}\right)$, where D is the set of possible tuples across all possible worlds of $\mathcal{D}$.

Lemma 4.8. *Given* $\mathcal{RA}^+$ *query* Q *and* c-TIDB $\mathcal{D}$, *let* C *be the circuit computed by* $Q\,(D)$. *Then, for the reduced Binary-BIDB* $\mathcal{D}'$ *there exists an equivalent circuit* C' *obtained from* $Q\,(D')$, *such that* $\gamma\,(\mathsf{C}') \leq 1 - c^{-(k-1)}$ *with* $\textsc{size}\,(\mathsf{C}') \leq \textsc{size}\,(\mathsf{C}) + O\,(nc)$ *and* $\textsc{depth}\,(\mathsf{C}') = \textsc{depth}\,(\mathsf{C}) + O\,(\log c)$.

We briefly connect the runtime in Eq. (4) to the algorithm outline earlier (where we ignore the dependence on $\overline{\mathcal{M}}\,(\cdot,\cdot)$, which is needed to handle the cost of arithmetic operations over integers). The $\textsc{size}(\mathsf{C})$ comes from the time taken to run OnePass once (OnePass essentially computes $|\mathsf{C}|\,(1,\ldots,1)$ using the natural circuit evaluation algorithm on C). We make $\frac{\log\frac{1}{\delta}}{(\epsilon')^2\cdot(1-\gamma)^2\cdot p_0^{2k}}$ many calls to SampleMonomial (each of which essentially traces $O(k)$ random sink to source paths in C all of which by definition have length at most $\textsc{depth}(\mathsf{C})$).

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

Lemma 4.9. *For any* Binary-BIDB *circuit* C *with* $\textsc{deg}(\mathsf{C}) = k$, *we have* $|\mathsf{C}|\,(1,\ldots,1) \leq 2^{2^k\cdot\textsc{depth}(\mathsf{C})}$. *Further, if* C *is a tree, then we have* $|\mathsf{C}|\,(1,\ldots,1) \leq \textsc{size}(\mathsf{C})^{O(k)}$.

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

Finally, note that by Proposition E.1 and Lemma E.2 for any $\mathcal{RA}^+$ query Q, there exists a circuit C^* for $\Phi[Q,D,t]$ such that $\textsc{depth}(\mathsf{C}^*) \leq O_{|Q|}(\log n)$ and $\textsc{size}(\mathsf{C}) \leq O_k\,(T_{det}\,(Q,D,c))$. Using this along with Lemma 4.9, Theorem 4.7 and the fact that $n \leq T_{det}\,(Q,D,c)$, we have the following corollary:

Corollary 4.10. *Let* Q *be an* $\mathcal{RA}^+$ *query and* $\mathcal{D}$ *be a* Binary-BIDB *with* $p_0 > 0$ *and* $\gamma < 1$ *(where* p_0, γ *as in Theorem 4.7) are absolute constants. Let* $\Phi(\mathbf{X}) = \Phi[Q,D,t]$ *for any result tuple* t *with* $\deg(\Phi) = k$. *Then one can compute an approximation satisfying Eq.* (3) *in time* $O_{k,|Q|,\epsilon',\delta}\,(T_{det}\,(OPT\,(Q)\,,D,c))$ *(given* Q, D *and* p_i *for each* $i \in [n]$ *that defines* $\mathcal{P}$*).*

Next, we note that the above result along with Lemma 4.8 answers Problem 1.5 in the affirmative as follows:

Corollary 4.11. *Let* Q *be an* $\mathcal{RA}^+$ *query and* $\mathcal{D}$ *be a* c-TIDB *with* $p_0 > 0$ *(where* p_0 *as in Theorem 4.7) is an absolute constant. Let* $\Phi(\mathbf{X}) = \Phi[Q,D,t]$ *for any result tuple* t *with* $\deg(\Phi) = k$. *Then one can compute an approximation satisfying Eq.* (3) *in time* $O_{k,|Q|,\epsilon',\delta,c}\,(T_{det}\,(OPT\,(Q)\,,D,c))$ *(given* Q, D *and* $p_{t,j}$ *for each* $t \in D, j \in [c]$ *that defines* $\mathcal{P}$*).*

Proof of Corollary 4.11. The proof follows by Lemma 4.8, and Corollary 4.10. □

If we want to approximate the expected multiplicities of all $Z = O(n^k)$ result tuples t simultaneously, we just need to run the above result with δ replaced by $\frac{\delta}{Z}$. Note this increases the runtime by only a logarithmic factor.

5 RELATED WORK

Probabilistic Databases (PDBs) have been studied predominantly for set semantics. Approaches for probabilistic query processing (i.e., computing marginal probabilities of tuples), fall into two broad categories. *Intensional* (or *grounded*) query evaluation computes the *lineage* of a tuple and then the probability of the lineage formula. It has been shown that computing the marginal probability of a tuple is #P-hard [48] (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. [15] and Olteanu et al. [22] proved dichotomies for UCQs and two classes of queries with negation, respectively. Amarilli et al. investigated tractable classes of databases for more complex queries [3]. Another line of work studies which structural properties of lineage formulas lead to tractable cases [33, 43, 46]. 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 [49] (TIDBs), block-independent databases (BIDBs) [44], and *PC-tables* [28]. Fink et al. [20] study aggregate queries over a probabilistic version of the extension of K-relations for aggregate queries proposed in [4] (*pvc-tables*) that supports bags, and has runtime complexity linear in the size of the lineage. However, this lineage is encoded as a tree; the size (and thus the runtime) are still superlinear in $T_{det}\ (Q, D, c)$. The runtime bound is also limited to a specific class of (hierarchical) queries, suggesting the possibility of a generalization of [15]'s dichotomy result to bag-PDBs.

Several techniques for approximating tuple probabilities have been proposed in related work [13, 16, 21, 40], relying on Monte Carlo sampling, e.g., [13], or a branch-and-bound paradigm [40]. Our approximation algorithm is also based on sampling.

Compressed Encodings are used for Boolean formulas (e.g, various types of circuits including OBDDs [31]) and polynomials (e.g., factorizations [41]) some of which have been utilized for probabilistic query processing, e.g., [31]. Compact representations for which probabilities can be computed in linear time include OBDDs, SDDs, d-DNNF, and FBDD. [17] studies circuits for absorptive semirings while [47] studies circuits that include negation (expressed as the monus operation). Algebraic Decision Diagrams [7] (ADDs) generalize BDDs to variables with more than two values. Chen et al. [10] introduced the generalized disjunctive normal form. Appendix H covers more related work on fine-grained complexity.

6 CONCLUSIONS AND FUTURE WORK

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

ACKNOWLEDGMENTS

7 ACKNOWLEDGEMENTS

We thank Virginia Williams for showing us Eq. (20), which greatly simplified our earlier proof of Lemma 3.8, and for graciously allowing us to use it.

REFERENCES

[1] pdbench. http://pdbench.sourceforge.net/. Accessed: 2020-12-15.
[2] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.
[3] Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Probabilities and provenance via tree decompositions. *PODS*, 2015.
[4] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011.
[5] Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data.
[6] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. doi:10.1137/110859440.
[7] 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.
[8] George Beskales, Ihab F. Ilyas, and Lukasz Golab. Sampling the repairs of functional dependency violations under hard constraints. *Proc. VLDB Endow.*, 3(1):197–207, 2010.
[9] Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*, volume 315. Springer, 1997.
[10] Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010.
[11] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. URL: https://www.sciencedirect.com/science/article/pii/S0022000006000675, doi:https://doi.org/10.1016/j.jcss.2006.04.007.
[12] Radu Curticapean. Counting matchings of size k is w[1]-hard. In *ICALP*, volume 7965, pages 352–363, 2013.
[13] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB*, 16(4):544, 2007.
[14] Nilesh Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.
[15] Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *JACM*, 59(6):30, 2012.
[16] 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.
[17] Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for datalog provenance. In *ICDT*, pages 201–212, 2014.
[18] Su Feng, Boris Glavic, Aaron Huber, and Oliver Kennedy. Efficient uncertainty tracking for complex queries with attribute-level bounds. In *SIGMOD*, 2021.
[19] Su Feng, Aaron Huber, Boris Glavic, and Oliver Kennedy. Uncertainty annotated databases - a lightweight approach for approximating certain answers. In *SIGMOD*, 2019.
[20] Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5):490–501, 2012.
[21] Robert Fink, Jiewen Huang, and Dan Olteanu. Anytime approximation in probabilistic databases. *VLDBJ*, 22(6):823–848, 2013.
[22] Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *TODS*, 41(1):4:1–4:47, 2016.
[23] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, page 538, USA, 2002. IEEE Computer Society.
[24] 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.
[25] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.).* Pearson Education, 2009.
[26] George Grätzer. *Universal algebra.* Springer Science & Business Media, 2008.

[27] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
[28] Todd J Green and Val Tannen. Models for incomplete and probabilistic information. In *EDBT*, pages 278–296. 2006.
[29] T. Imielinski and W. Lipski. Incomplete information in relational databases. 1989.
[30] Tomasz Imieliński and Witold Lipski Jr. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.
[31] Abhay Kumar Jha and Dan Suciu. Probabilistic databases with markoviews. *PVLDB*, 5(11):1160–1171, 2012.
[32] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.
[33] 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.
[34] Oliver Kennedy and Christoph Koch. Pip: A database system for great and small expectations. In *ICDE*, 2010.
[35] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *PODS*, pages 13–28, 2016.
[36] 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.
[37] Poonam Kumari, Said Achmiz, and Oliver Kennedy. Communicating data quality in on-demand curation. In *QDB*, 2016.
[38] Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*, 2018.
[39] 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.
[40] Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, pages 145–156, 2010.
[41] Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016.
[42] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, 2017.
[43] Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, 2011.
[44] C. Ré and D. Suciu. Materialized views in probabilistic databases: for information exchange and query optimization. In *VLDB*, pages 51–62, 2007.
[45] Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. Incremental knowledge base construction using deepdive. *VLDB J.*, 26(1):81–105, 2017.
[46] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Read-once functions and query evaluation in probabilistic databases. *PVLDB*, 3(1):1068–1079, 2010.
[47] Pierre Senellart. Provenance and probabilities in relational databases. *SIGMOD Record*, 46(4):5–15, 2018.
[48] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
[49] Guy Van den Broeck and Dan Suciu. Query processing on probabilistic data: A survey. 2017.
[50] Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *SIGACT News*, 49(4):29–35, 2018. doi:10.1145/3300150.3300158.
[51] Ying Yang, Niccolò Meneghetti, Ronny Fehling, Zhen Hua Liu, Dieter Gawlick, and Oliver Kennedy. Lenses: An on-demand approach to etl. *PVLDB*, 8(12):1578–1589, 2015.

A GENERALIZING BEYOND SET INPUTS

A.1 TIDBs

In our definition of TIDBs (Sec. 2.2), we assumed a model of TIDBs where each input tuple is assigned a probability p of having multiplicity 1. That is, we assumed inputs to be sets, but interpret queries under bag semantics. Other sensible generalizations of TIDBs from set semantics to bag semantics also exist.

One very natural such generalization is to assign each input tuple t a multiplicity m_t and probability p: the tuple has probability p to exists with multiplicity m_t, and otherwise has multiplicity 0. If the maximal multiplicity of all input tuples in the TIDB is bounded by some constant, then a generalization of our hardness results and approximation algorithm can be achieved by changing the construction of lineage polynomials (in Fig. 1) as follows (all other cases remain the same as in fig. 1):

$$\Phi[R, D_\Omega, t] = \begin{cases} m_t X_t & \text{if } D_\Omega.R(t) = m_t \\ 0 & \text{otherwise.} \end{cases}$$

That is the variable representing a tuple is multiplied by m_t to encode the tuple's multiplicity m_t. We note that our lower bounds still hold for this model since we only need $m_t = 1$ for all tuples t. Further, it can be argued that our proofs (as is) for approximation algorithms also work for this model. The only change is that since we now allow $m_t > 1$ some of the constants in the runtime analysis of our algorithms change but the overall asymptotic runtime bound remains the same.

Yet another option would be to assign each tuple a probability distribution over multiplicities. It seems very unlikely that our results would extend to a model that allows arbitrary probability distributions over multiplicities (our current proof techniques definitely break down). However, we would like to note that the special case of a Poisson binomial distribution (sum of independent but not necessarily identical Bernoulli trials) over multiplicities can be handled as follows: we add an additional identifier attribute to each relation in the database. For a tuple t with maximal multiplicity m_t, we create m_t copies of t with different identifiers. To answer a query over this encoding, we first project away the identifier attribute (note that as per Fig. 1, in Φ this would add up all the variables corresponding to the same tuple t).

A.2 BIDBs

The approach described above works for BIDBs as well if we define the bag version of BIDBs to associate each tuple t a multiplicity m_t. Recall that we associate each tuple in a block with a unique variable. Thus, the modified lineage polynomial construction shown above can be applied for BIDBs too (and our approximation results also hold).

B MISSING DETAILS FROM SECTION 2

B.1 $\mathcal{K}$-relations and $\mathbb{N}[\mathrm{X}]$-encoded PDBs

We can use $\mathcal{K}$-relations to model bags. A *$\mathcal{K}$-relation* [27] 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 R 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 defined similarly, where we view the $\mathcal{K}$-database (relation) as a function mapping tuples to their respective annotations. $\mathcal{RA}^+$ query semantics over $\mathcal{K}$-relations are analogous to the lineage construction semantics of Fig. 1, with the exception of replacing + with $\oplus_\mathcal{K}$ and $\cdot$ with $\otimes_\mathcal{K}$.

Consider the semiring $\mathbb{N} = \{\mathbb{N}, +, \times, 0, 1\}$ of natural numbers. $\mathbb{N}$-databases model bag semantics by annotating each tuple with its multiplicity. A probabilistic $\mathbb{N}$-database ($\mathbb{N}$-PDB) is a PDB where each possible world is an $\mathbb{N}$-database. We study the problem of computing statistical moments for query results over such databases. Given an $\mathbb{N}$-PDB $\mathcal{D} = (\Omega, \mathcal{P})$, ($\mathcal{RA}^+$) query Q, and possible result tuple t, we sum $Q(D)(t) \cdot \mathcal{P}(D)$ for all $D \in \Omega$ to compute the expected multiplicity of t. Intuitively, the expectation of $Q(D)(t)$ is the number of duplicates of t we expect to find in 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 (abusing notation) $\mathbb{N}[\mathbf{X}] = \{\mathbb{N}[\mathbf{X}], +, \cdot, 0, 1\}$ whose domain is $\mathbb{N}[\mathbf{X}]$, with the standard addition and multiplication of polynomials. We define an $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathrm{X}]}$ as the tuple $(D_{\mathbb{N}[\mathrm{X}]}, \mathcal{P})$, where $\mathbb{N}[\mathbf{X}]$-database $D_{\mathbb{N}[\mathrm{X}]}$ is paired with the probability distribution $\mathcal{P}$ across the set of possible worlds *represented* by $D_{\mathbb{N}[\mathrm{X}]}$, i.e. the one induced from $\mathcal{P}_{\mathbb{N}[\mathrm{X}]}$, the probability distribution over $\mathbf{X}$. Note that the notation is slightly abused since the first element of the pair is an encoded set of possible worlds, i.e. $D_{\mathbb{N}[\mathrm{X}]}$ is the deterministic bounding database. We denote by $\Phi[Q, D_{\mathbb{N}[\mathrm{X}]}, t]$ the annotation of tuple t in the result of $Q(D_{\mathbb{N}[\mathrm{X}]})(t)$, and as before, interpret it as a function $\Phi[Q, D_{\mathbb{N}[\mathrm{X}]}, t] : \{0,1\}^{|\mathbf{X}|} \to \mathbb{N}$ from vectors of variable assignments to the corresponding value of the annotating polynomial. $\mathbb{N}[\mathbf{X}]$-encoded PDBs and a function *Mod* (which transforms an $\mathbb{N}[\mathbf{X}]$-encoded 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 its respective $\mathbb{N}$-PDB. For that it will be opportune to define representation systems for $\mathbb{N}$-PDBs.

Definition B.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 $Q \in \mathcal{Q}$ and $M \in \mathcal{M}$ 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, referring to such databases as $\mathbb{N}[\mathbf{X}]$-encoded PDBs. Given $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathrm{X}]}$, one can think of the of $\mathcal{P}$ as the probability distribution

across all worlds $\{0,1\}^n$. Denote a particular world to be $\mathbf{w}$. For convenience let $\psi_{\mathbf{w}} : \mathcal{D}_{\mathbb{N}[\mathbf{X}]} \to \mathcal{D}_{\mathbb{N}}$ be a function that computes the corresponding $\mathbb{N}$-PDB upon assigning all values $w_i \in \mathbf{w}$ to $X_i \in \mathbf{X}$ of $D_{\mathbb{N}[\mathbf{X}]}$. Note the one-to-one correspondence between elements $\mathbf{w} \in \{0,1\}^n$ to the worlds encoded by $D_{\mathbb{N}[\mathbf{X}]}$ when $\mathbf{w}$ is assigned to $\mathbf{X}$ (assuming a domain of $\{0,1\}$ for each X_i). We can think of $\psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})(t)$ as the semiring homomorphism $\mathbb{N}[\mathbf{X}] \to \mathbb{N}$ that applies the assignment $\mathbf{w}$ to all variables $\mathbf{X}$ of a polynomial and evaluates the resulting expression in $\mathbb{N}$.

Definition B.2 (*Mod* $\left(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right)$). *Given an $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathbf{X}]}$, we compute its equivalent $\mathbb{N}$-PDB $\mathcal{D}_{\mathbb{N}} = Mod\left(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right) = (\Omega, \mathcal{P}')$ as:*

$$\Omega = \{\psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) \mid \mathbf{w} \in \{0,1\}^n\}$$
$$\forall D \in \Omega : Pr(D) = \sum_{\mathbf{w} \in \{0,1\}^n : \psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = D} Pr(\mathbf{w})$$

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

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

Importantly, as the following proposition shows, any finite $\mathbb{N}$-PDB can be encoded as an $\mathbb{N}[\mathbf{X}]$-encoded PDB and $\mathbb{N}[\mathbf{X}]$-encoded PDBs are closed under $\mathcal{RA}^+$[27].

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

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

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

The probability distribution $\mathcal{P}'$ assigns all world vectors zero probability except for $|\Omega|$ world vectors (representing the possible worlds) $\mathbf{w}_i$. 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 $Mod(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = \mathcal{D}$. Thus, $\mathbb{N}[\mathbf{X}]$-encoded PDBs are a complete representation system.

Since $\mathbb{N}[\mathbf{X}]$ is the free object in the variety of semirings, Birkhoff's HSP theorem implies that any assignment $\mathbf{X} \to \mathbb{N}$, which includes as a special case the assignments $\psi_{\mathbf{w}}$ used here, uniquely extends to the semiring homomorphism alluded to above, $\psi_{\mathbf{w}}\left(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right)(t) : \mathbb{N}[\mathbf{X}] \to \mathbb{N}$. For a polynomial $\psi_{\mathbf{w}}\left(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right)(t)$ substitutes variables based on $\mathbf{w}$ and then evaluates the resulting expression in $\mathbb{N}$. For instance, consider the polynomial $\mathcal{D}_{\mathbb{N}[\mathbf{X}]}(t) = \Phi = X + Y$ and assignment $\mathbf{w} := X = 0, Y = 1$. We get $\psi_{\mathbf{w}}\left(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right)(t) = 0 + 1 = 1$. Closure under $\mathcal{RA}^+$ queries follows from this and from [27]'s Proposition 3.5, which states that semiring homomorphisms commute with queries over $\mathcal{K}$-relations. □

B.2 TIDBs and BIDBs in the $\mathbb{N}[\mathbf{X}]$-encoded PDB model

Two important subclasses of $\mathbb{N}[\mathbf{X}]$-encoded PDBs that are of interest to us are the bag versions of tuple-independent databases (TIDBs) and block-independent databases (BIDBs). Under set semantics, a TIDB is a deterministic database D where each tuple t is assigned a probability p_t. The set of possible worlds represented by a TIDB D is all subsets of D. The probability of each world is the product of the probabilities of all tuples that exist with one minus the probability of all tuples of D that are not part of 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 at most 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.

In this work, we define TIDBs and BIDBs as subclasses of $\mathbb{N}[\mathbf{X}]$-encoded PDBs defined over variables $\mathbf{X}$ (Definition B.2) where $\mathbf{X}$ can be partitioned into blocks that satisfy the conditions of a TIDB or BIDB (stated formally in Sec. 2.2). 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 $\psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})$ (the ones where $w_i = 1$). For BIDBs specifically, note 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. Given PDB $\mathcal{D}$t $\mathcal{P}$ is the distribution induced by $\mathbf{p}$, which we will denote $\mathcal{P}^{(\mathbf{p})}$.

$$\mathop{\mathbb{E}}_{\mathbf{W} \sim \mathcal{P}^{(\mathbf{p})}}[\Phi(\mathbf{W})] = \sum_{\substack{\mathbf{w} \in \{0,1\}^n \\ s.t. w_j, w_{j'} = 1 \to \nexists b_i \supseteq \{t_{i,j}, t_{i',j}\}}} \Phi(\mathbf{w}) \prod_{\substack{j \in [n] \\ s.t. w_j = 1}} p_j \prod_{\substack{j \in [n] \\ s.t. w_j = 0}} (1 - p_i) \tag{5}$$

Recall that tuple blocks in a TIDB always have size 1, so the outer summation of eq. (5) is over the full set of vectors.

B.3 Proof of Proposition 2.8

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

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

From $Mod(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = \mathcal{D}$, we have that the range of $\psi_{\mathbf{w}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})}$ is Ω, so

$$= \sum_{D \in \Omega} \sum_{\mathbf{w} \in \{0,1\}^n : \psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = D} Pr(\mathbf{w}) \cdot Q(D_{\mathbb{N}[\mathbf{X}]})(t)(\mathbf{w})$$

The inner sum is only over $\mathbf{w}$ where $\psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = D$ (i.e., $Q(D_{\mathbb{N}[\mathbf{X}]})(t)(\mathbf{w}) = Q(D)(t)$)

$$= \sum_{D \in \Omega} \sum_{\mathbf{w} \in \{0,1\}^n : \psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = D} Pr(\mathbf{w}) \cdot Q(D)(t)$$

By distributivity of + over ×

$$= \sum_{D \in \Omega} Q(D)(t) \sum_{\mathbf{w} \in \{0,1\}^n : \psi_{\mathbf{w}}(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = D} Pr(\mathbf{w})$$

From the definition of $\mathcal{P}$ in definition B.2, given $Mod(\mathcal{D}_{\mathbb{N}[\mathbf{X}]}) = \mathcal{D}$, we get

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

□

B.4 Proposition B.4

Note the following fact:

Proposition B.4. *For any BIDB-lineage polynomial $\Phi(X_1, \ldots, X_n)$ and all $\mathbf{w}$ such that $Pr[\mathbf{W} = \mathbf{w}] > 0$, it holds that $\Phi(\mathbf{w}) = \widetilde{\Phi}(\mathbf{w})$.*

Proof. Note that any Φ 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 $e \geq 1$, $b^e = b$. Finally, note that there are exactly three cases where the expectation of a monomial term $\mathbb{E}\left[c_{\mathbf{d}} \prod_{i=n\ s.t.\ \mathbf{d}_i \geq 1} X_i\right]$ is zero: (i) when $c_{\mathbf{d}} = 0$, (ii) when $p_i = 0$ for some i where $\mathbf{d}_i \geq 1$, and (iii) when X_i and X_j are in the same block for some i, j where $\mathbf{d}_i, \mathbf{d}_j \geq 1$. □

B.5 Proof for Lemma 1.4

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

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

Let the boolean function $\textsc{isInd}(\cdot)$ take $\mathbf{d}$ as input and return true if there does not exist any dependent variables in $\mathbf{d}$, i.e., $\nexists\, B, i \neq j \mid d_{B,i}, d_{B,j} \geq 1$.[13]. Then in expectation we have

$$\mathop{\mathbb{E}}_{\mathbf{W}}[\Phi(\mathbf{W})] = \mathop{\mathbb{E}}_{\mathbf{W}}\left[\sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} W_i^{d_i} + \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \neg\textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} W_i^{d_i}\right] \tag{6}$$

$$= \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \mathop{\mathbb{E}}_{\mathbf{W}}\left[\prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} W_i^{d_i}\right] + \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \neg\textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \mathop{\mathbb{E}}_{\mathbf{W}}\left[\prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} W_i^{d_i}\right] \tag{7}$$

$$= \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \mathop{\mathbb{E}}_{\mathbf{W}}\left[\prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} W_i^{d_i}\right] \tag{8}$$

$$= \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} \mathop{\mathbb{E}}_{\mathbf{W}}\left[W_i^{d_i}\right] \tag{9}$$

$$= \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} \mathop{\mathbb{E}}_{\mathbf{W}}[W_i] \tag{10}$$

$$= \sum_{\substack{\mathbf{d} \in \{0,\ldots,K\}^n \\ \wedge\ \textsc{isInd}(\mathbf{d})}} c_{\mathbf{d}} \cdot \prod_{\substack{i=1 \\ s.t. d_i \geq 1}}^{n} p_i \tag{11}$$

$$= \widetilde{\Phi}(p_1, \ldots, p_n). \tag{12}$$

Eq. (6) is the result of substituting in the definition of Φ given above. Then we arrive at eq. (7) by linearity of expectation. Next, eq. (8) is the result of the independence constraint of BIDBs, specifically that any monomial composed of dependent variables, i.e., variables from the same block B, has a probability of 0. Eq. (9) is obtained by the fact that all variables in each monomial are independent, which allows for the expectation to be pushed through the product. In eq. (10), since $W_i \in \{0, 1\}$ it is the case that for any exponent $e \geq 1$, $W_i^e = W_i$. Next, in eq. (11) the expectation of a tuple is indeed its probability.

Finally, it can be verified that Eq. (12) follows since eq. (11) satisfies the construction of $\widetilde{\Phi}(p_1, \ldots, p_n)$ in Definition 1.3. □

B.6 Proof For Corollary 2.7

Proof. Note that Lemma 1.4 shows that $\mathbb{E}[\Phi] = \widetilde{\Phi}(p_1, \ldots, p_n)$. Therefore, if Φ is already in SMB form, one only needs to compute $\Phi(p_1, \ldots, p_n)$ ignoring exponent terms (note that such a polynomial is $\widetilde{\Phi}(p_1, \ldots, p_n)$), which indeed has $O(|\Phi|)$ computations. □

C MISSING DETAILS FROM SECTION 3

C.1 Lemma C.1

Lemma C.1. *Assuming that each $v \in V$ has degree ≥ 1,[14] the* PDB *relations encoding the edges for Φ_G^k of Definition 3.4 can be computed in $O(m)$ time.*

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

C.2 Proof of Lemma 3.5

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

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

[13] This BIDB notation is used and discussed in sec. 2.2

[14] This is WLOG, since any vertex with degree 0 can be dropped without affecting the result of our hard query.

We argue that $T_{join}(D.V, D.E, D.V)$ is at most $O(m)$ by noting that there exists an algorithm that computes $D.V \bowtie D.E \bowtie D.V$ in the same runtime[15]. Then by the assumption of Lemma C.1 (each $v \in V$ has degree ≥ 1), the sum of the first three terms is $O(m)$. We then obtain that $T_{det}\left(Q^1, D\right) = O(m) + O(m) = O(m)$. For $Q^k = Q_1^1 \times \cdots \times Q_k^1$, we have the recurrence $T_{det}\left(Q^k, D\right) = T_{det}\left(Q_1^1, D\right) + \cdots + T_{det}\left(Q_k^1, D\right) + T_{join}(Q_1^1, \cdots, Q_k^1)$. Since Q^1 outputs a count, computing the join $Q_1^1 \bowtie \cdots \bowtie Q_k^1$ is just multiplying k numbers, which takes $O(k)$ time. Thus, we have

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

as desired. □

C.3 Lemma C.2

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

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

C.4 Proof of Lemma C.2

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

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

We note that c_i is *exactly* the number of monomials in the SMB expansion of $\Phi_G^k(\mathbf{X})$ composed of i distinct variables.[16]

Given that we then have $2k+1$ distinct values of $\widetilde{\Phi}_G^k(p, \ldots, p)$ for $0 \leq i \leq 2k$, 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 $\mathbf{b}[i] = \widetilde{\Phi}_G^k(p_i, \ldots, p_i)$. 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 $\#\left(G, \text{⚷} \cdots \text{⚷}^k\right)$ from c_{2k} in $O(1)$ additional time. We claim that c_{2k} is $k! \cdot \#\left(G, \text{⚷} \cdots \text{⚷}^k\right)$. This can be seen intuitively by looking at the expansion of the original factorized representation

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

where a unique k-matching in the multi-set of product terms 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 $\Phi_G^k(\mathbf{X})$, with distinct indexes, and this implies that each distinct k-matching appears the exact number of permutations that exist for its particular set of k edges, or $k!$.

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

C.5 Proof of Theorem 3.6

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

$$O(m) + O(k) \cdot o\left(T_{match}(k, G)\right) + O(k^3) \tag{13}$$

$$\leq o\left(T_{match}(k, G)\right). \tag{14}$$

We obtain Eq. (14) from the facts that k is fixed (related to m) and the assumption that $T_{match}(k, G) \geq \omega(m)$. Thus we obtain the contradiction that we can achieve a runtime $o\left(T_{match}(k, G)\right)$ that is better than the optimal time $T_{match}(k, G)$ required to compute k-matchings. □

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

We need all the possible edge patterns in an arbitrary G with at most three distinct edges. We have already seen ⚷, ⚷⚷ 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.

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 C.7):

$$\#(G, \text{⚷}) = m, \tag{15}$$

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

[16] Since $\widetilde{\Phi}_G^k(\mathbf{X})$ 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.

$$\#(G, \text{2-star}) = \sum_{i \in V} \binom{d_i}{2} \tag{16}$$

$$\#(G, \text{2-matching}) = \sum_{(i,j) \in E} \frac{m - d_i - d_j + 1}{2} \tag{17}$$

$$\#(G, \text{3-star}) = \sum_{i \in V} \binom{d_i}{3} \tag{18}$$

$$\#(G, \text{edge} \sqcup \text{2-star}) + 3\#(G, \text{3-matching}) = \sum_{(i,j) \in E} \binom{m - d_i - d_j + 1}{2} \tag{19}$$

$$\#(G, \text{3-path}) + 3\#(G, \text{triangle}) = \sum_{(i,j) \in E} (d_i - 1) \cdot (d_j - 1) \tag{20}$$

C.7 Proofs of Eq. (15)-Eq. (20)

The proofs for Eq. (15), Eq. (16) and Eq. (18) are immediate.

Proof of Eq. (17). 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. (19). Eq. (19) 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. (17), 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, since it is indistinguishable to the closed form expression which of the remaining edges are either disjoint or connected to each of the edges in the *initial* set of edges disjoint to the edge under consideration. Observe that the disjoint case will be counted 3 times since each edge of a 3-path is visited once, and the same 3-path counted in each visitation. For the latter case however, it is true that since the two path in  is connected, there will be no multiple counting by the fact that the summation automatically disconnects the current edge, meaning that a two matching at the current vertex under consideration will not be counted. Thus,  will only be counted once, precisely when the single disjoint edge is visited in the summation. The sum over all such edge combinations is precisely then $\#(G, \text{edge} \sqcup \text{2-star}) + 3\#(G, \text{3-matching})$. Note that all factorials can be computed in $O(m)$ time, and then each combination $\binom{n}{2}$ can be performed with constant time operations, yielding the claimed $O(m)$ run time. □

Proof of Eq. (20). To compute $\#(G, \text{3-path})$, 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. (20) by observing that each triangle is counted thrice, while each 3-path is counted just once. For reasons similar to Eq. (17), 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. □

C.8 Tools to prove Theorem 3.7

Note that $\widetilde{\Phi}^3_G(p, \ldots, p)$ 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 C.3. *For any p, we have:*

$$\widetilde{\Phi}^3_G(p, \ldots, p) = \#(G, \text{edge})\, p^2 + 6\#(G, \text{2-star})\, p^3 + 6\#(G, \text{2-matching})\, p^4 + 6\#(G, \text{triangle})\, p^3 + 6\#(G, \text{3-star})\, p^4 + 6\#(G, \text{3-path})\, p^4 + 6\#(G, \text{edge} \sqcup \text{2-star})\, p^5 + 6\#(G, \text{3-matching})\, p^6. \tag{21}$$

C.8.1 Proof for Lemma C.3.

Proof. By definition we have that

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

Hence $\widetilde{\Phi}^3_G(\mathbf{X})$ 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 $\widetilde{\Phi}^3_G(\mathbf{X})$, where ν is the number of distinct variables in the monomial. Let $e_1 = (i_1, j_1), e_2 = (i_2, j_2)$, and $e_3 = (i_3, j_3)$. We compute $\widetilde{\Phi}^3_G(\mathbf{X})$ 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). When we have that $e_1 = e_2 = e_3$, then the monomial corresponds to $\#(G, \text{edge})$. There are exactly m such triples, each with a p^2 factor in $\widetilde{\Phi}^3_G(p, \ldots, p)$.

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 $\Phi^3_G(\mathbf{X})$. Analogously, there are three such occurrences in $\Phi^3_G(\mathbf{X})$ 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 $\widetilde{\Phi}^3_G$. Since $e \neq e'$, this case produces the following edge patterns: , , which contribute $6p^3$ and $6p^4$ respectively to $\widetilde{\Phi}^3_G(p, \ldots, p)$.

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

Since p is fixed, Lemma C.3 gives us one linear equation in $\#(G, \text{triangle})$ and $\#(G, \text{3-matching})$ (we can handle the other counts due to equations (15)-(20)). 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 C.4. *For $\ell \geq 1$, let graph $G^{(\ell)}$ be a graph generated from an arbitrary graph G, by replacing every edge e of G with an*

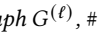
ℓ-path, such that all inner vertexes of an ℓ-path replacement edge are disjoint from all other vertexes.[17].

We will prove Theorem 3.7 by the following reduction:

Theorem C.5. *Fix $p \in (0,1)$. Let G be a graph on m edges. If we can compute $\widetilde{\Phi}^3_G(p,\ldots,p)$ exactly in $T(m)$ time, then we can exactly compute $\#(G, \text{  })$ in $O(T(m)+m)$ time.*

For clarity, we repeat the notion of $\#(G,H)$ to mean the count of subgraphs in G isomorphic to H. The following lemmas relate these counts in $G^{(2)}$ to $G^{(1)}$ (G). The lemmas are used to prove Lemma C.8.

Lemma C.6. *The 3-matchings in graph $G^{(2)}$ satisfy the identity:*

$$\#\left(G^{(2)}, \text{  }\right) = 8\cdot\#\left(G^{(1)}, \text{  }\right) + 6\cdot\#\left(G^{(1)}, \text{  }\right) + 4\cdot\#\left(G^{(1)}, \text{  }\right) + 4\cdot\#\left(G^{(1)}, \text{  }\right) + 2\cdot\#\left(G^{(1)}, \text{  }\right).$$

Lemma C.7. *For $\ell > 1$ and any graph $G^{(\ell)}$, $\#\left(G^{(\ell)}, \text{  }\right) = 0$.*

Finally, the following result immediately implies Theorem C.5:

Lemma C.8. *Fix $p \in (0,1)$. Given $\widetilde{\Phi}^3_{G^{(\ell)}}(p,\ldots,p)$ for $\ell \in [2]$, we can compute in $O(m)$ time a vector $\mathbf{b} \in \mathbb{R}^3$ such that*

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

allowing us to compute $\#(G, \text{  })$ and $\#(G, \text{})$ in $O(1)$ time.

C.9 Proofs for Lemma C.6, Lemma C.7, and Lemma C.8

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

Definition C.9 ($E^{(\ell)}$). *For $\ell > 1$, we use $E^{(\ell)}$ 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 $b \in \{0,\ldots,\ell-1\}$ where $(e,0),\ldots,(e,\ell-1)$ is the ℓ-path that replaces the edge e for $e \in E^{(1)}$.*

Definition C.10 ($E^{(\ell)}_S$). *Given an arbitrary subgraph $S^{(1)}$ of $G^{(1)}$, let $E^{(1)}_S$ denote the set of edges in $S^{(1)}$. Define then $E^{(\ell)}_S$ for $\ell > 1$ as the set of edges in the generated subgraph $S^{(\ell)}$ (i.e. when we apply Definition C.4 to S to generate $S^{(\ell)}$).*

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

Definition C.11 ($\binom{E}{t}$ and $\binom{E}{\leq t}$). *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 C.12. *Let $f_\ell : \binom{E^{(\ell)}}{3} \to \binom{E^{(1)}}{\leq 3}$ 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:*

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

[17] Note that $G \equiv G^{(1)}$.

Definition C.13 (f_ℓ^{-1}). *For an arbitrary subgraph $S^{(1)}$ of $G^{(1)}$ with at most $m \leq 3$ edges, the inverse function $f_\ell^{-1} : \binom{E^{(1)}}{\leq 3} \to 2^{\binom{E^{(\ell)}}{3}}$ takes $E^{(1)}_S$ and outputs the set of all elements $s \in \binom{E^{(\ell)}_S}{3}$ such that $f_\ell(s) = E^{(1)}_S$.*

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

We are now ready to prove the structural lemmas. To prove the structural lemmas, we will count the number of occurrences of and in $G^{(\ell)}$ we count for each $S \in \binom{E_1}{\leq 3}$, how many and subgraphs appear in $f_\ell^{-1}(E^{(1)}_S)$.

C.9.1 Proof of Lemma C.6.

Proof. For each subset $E^{(1)}_S \in \binom{E_1}{\leq 3}$, we count the number of 3-*matchings* in the 3-edge subgraphs of $G^{(2)}$ in $f_2^{-1}(E^{(1)}_S)$. We first consider the case of $E^{(1)}_S \in \binom{E_1}{3}$, where $E^{(1)}_S$ is composed of the edges e_1,e_2,e_3 and $f_2^{-1}(E^{(1)}_S)$ 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 $f_\ell(s) = \{e_1,e_2,e_3\}$. The size of the output is denoted $\left|f_2^{-1}(E^{(1)})\right|$. For the case where each set of edges of the form $\{(e_1,b_1),(e_2,b_2),(e_3,b_3)\}$ for $b_i \in [2], i \in [3]$ is present, we have $\left|f_2^{-1}(E^{(1)})\right| = 8$. We count the number of 3-matchings from the set $f_2^{-1}(E^{(1)}_S)$.

We do a case analysis based on the subgraph $S^{(1)}$ induced by $E^{(1)}_S$.

- 3-matching ()

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

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

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

Since these two selections can be made independently, $\left|f_2^{-1}\left(E^{(1)}_S\right)\right| = 2\cdot 3 = 6$ *distinct* 3-matchings in $f_2^{-1}(E^{(1)}_S)$.

- 3-star ()

When $S^{(1)}$ is isomorphic to , the inner edges $(e_i,1)$ of $S^{(2)}$ are all connected, and the outer edges $(e_i,0)$ are all disjoint. Note that for a valid 3-matching it must be the case that at most one inner edge can be part of the set of disjoint edges. For the case of when

exactly one inner edge is chosen, there exist 3 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, and $\left|f_2^{-1}\left(E_S^{(1)}\right)\right| = 4$.

- 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: $\{(e_1, 0), (e_2, 0), (e_3, 0)\}, \{(e_1, 0), (e_2, 0), (e_3, 1)\}, \{(e_1, 0), (e_2, 1), (e_3, 1)\}$, $\{(e_1, 1), (e_2, 1), (e_3, 1)\}$ and $\left|f_2^{-1}\left(E_S^{(1)}\right)\right| = 4$.

- 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. This rules out both subsets of $(e_1, 0), (e_2, 0), (e_3, 1)$ and $(e_1, 0), (e_2, 1), (e_3, 1)$, so that $\left|f_2^{-1}\left(E_S^{(1)}\right)\right| = 2$.

Let us now consider when $E_S^{(1)} \in \binom{E_1}{\leq 2}$, i.e. fixed subgraphs 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 $\{(e_2, 0), (e_2, 1)\}$. The third edge choice in $E_S^{(2)}$ will break the disjoint property of a 3-matching. Thus, a 3-matching cannot exist in $f_2^{-1}(E_S^{(1)})$. A similar argument holds for $|E_S^{(1)}| = 1$, where the output of f_2^{-1} is $\{\emptyset\}$ since there are not enough edges in the input to produce any other output.

Observe that all of the arguments above focused solely on the property of subgraph $S^{(1)}$ being 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. □

C.9.2 Proof of Lemma C.7.

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. □

C.9.3 Proof of Lemma C.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 linear system $\mathbf{M} \cdot \mathbf{x} = \mathbf{b}$.

To prove the first step, we use Lemma C.3 to derive the following equality (dropping the superscript and referring to $G^{(1)}$ as G):

$$\#(G,)\,p^2 + 6\#(G,)\,p^3 + 6\#(G,)\,p^4 + 6\#(G,)\,p^5 + 6\#(G,)\,p^6 = \widetilde{\Phi}^3_G(p, \ldots, p) \tag{22}$$

$$\#(G,) + \#(G,)\,p + \#(G,)\,p^2 + \#(G,)\,p^3 = \frac{\widetilde{\Phi}^3_G(p, \ldots, p)}{6p^3} - \frac{\#(G,)}{6p} - \#(G,) - \#(G,)\,p - \#(G,) \tag{23}$$

$$\#(G,)\,(1-3p) - \#(G,)\,(3p^2 - p^3) = \frac{\widetilde{\Phi}^3_G(p, \ldots, p)}{6p^3} - \frac{\#(G,)}{6p} - \#(G,) - \#(G,)\,p - \#(G,)\,p - [\#(G,)\,p + 3\#(G,)\,p] - \left[\#(G,)\,p^2 + 3\#(G,)\,p^2\right] \tag{24}$$

Eq. (22) is the result of Lemma C.3. We obtain the remaining equations through standard algebraic manipulations.

Note that the LHS of Eq. (24) is obtained using eq. (19) and eq. (20) and is indeed the product $\mathbf{M}[1] \cdot \mathbf{x}[1]$. Further note that this product is equal to the RHS of Eq. (24), where every term is computable in $O(m)$ time (by equations (15)-(20)). We set $\mathbf{b}[1]$ to the RHS of Eq. (24).

We follow the same process in deriving an equality for $G^{(2)}$. Replacing occurrences of G with $G^{(2)}$, we obtain an equation (below) of the form of eq. (24) for $G^{(2)}$. Substituting identities from lemma C.6 and Lemma C.7 we obtain

$$0 - (8\#(G,) + 6\#(G,) + 4\#(G,) + 4\#(G,) + 2\#(G,))\,(3p^2 - p^3) = \frac{\widetilde{\Phi}^3_{G^{(2)}}(p, \ldots, p)}{6p^3} - \frac{\#\left(G^{(2)}, \right)}{6p} - \#\left(G^{(2)}, \right) - \#\left(G^{(2)}, \right)p - \#\left(\right) - \left[\#\left(G^{(2)}, \right)p^2 + 3\#\left(G^{(2)}, \right)p^2\right] - \left[\#\left(G^{(2)}, \right)p + 3\#\left(\right.\right. \tag{25}$$

$$(10\#(G,) + 10G)(3p^2 - p^3) = \frac{\widetilde{\Phi}^3_{G^{(2)}}(p, \ldots, p)}{6p^3} - \frac{\#\left(G^{(2)}, \right)}{6p} - \#\left(G^{(2)}, \right) - \#\left(G^{(2)}, \right)p - \#\left(\right) - \left[\#\left(G^{(2)}, \right)p + 3\#\left(G^{(2)}, \right)p\right] - \left[\#\left(G^{(2)}, \right)p^2 - 3\#\left(G^{(\right.\right. + (4\#(G,) + [6\#(G,) + 18\#(G,)] + [4\#(G,) + 12\# \tag{26}$$

The steps to obtaining eq. (26) are analogous to the derivation immediately preceding. As in the previous derivation, note that the LHS of Eq. (26) is the same as $\mathbf{M}[2] \cdot \mathbf{x}[2]$. The RHS of Eq. (26) has terms all computable (by equations (15)-(20)) 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 $\#(G,)$ and $\#(G,)$ in $O(1)$ using Gaussian elimination.

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

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

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

$$= 10p^2(3-p) \cdot (1-p)^3 \tag{27}$$

From Eq. (27) it can easily be seen that the roots of $Det(\mathbf{M})$ are 0, 1, and 3. Hence there are no roots in $(0, 1)$ and Lemma C.8 follows. □

C.10 Proof of Theorem C.5

Proof. We can compute $G^{(2)}$ from $G^{(1)}$ in $O(m)$ time. Additionally, if in time $O(T(m))$, we have $\widetilde{\Phi}^3_{G^{(\ell)}}(p, \ldots, p)$ for $\ell \in [2]$, then the theorem follows by Lemma C.8. □

In other words, if Theorem C.5 holds, then so must Theorem 3.7.

C.11 Proof of Theorem 3.7

Proof. For the sake of contradiction, assume that for any G, we can compute $\widetilde{\Phi}^3_G(p, \ldots, p)$ in $o\left(m^{1+\epsilon_0}\right)$ time. Let G be the input graph. Then by Theorem C.5 we can compute $\#(G, \clubsuit)$ 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 3.3. □

D MISSING DETAILS FROM SECTION 4

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

$$\Phi(X, Y) = 2X^2 + 3XY - 2Y^2. \tag{28}$$

Definition D.1 (Pure Expansion). *The pure expansion of a polynomial Φ is formed by computing all product of sums occurring in Φ, without combining like monomials. The pure expansion of Φ generalizes Definition 2.1 by allowing monomials $m_i = m_j$ for $i \neq j$.*

Note that similar in spirit to **??**, $\mathtt{E(C)}$ Definition 4.1 reduces all variable exponents $e > 1$ to $e = 1$. Further, it is true that $\mathtt{E(C)}$ is the pure expansion of $\mathtt{C}$.

Example D.2 (Example of Pure Expansion). *Consider the factorized representation $(X + 2Y)(2X - Y)$ of the polynomial in Eq. (28). Its circuit $\mathtt{C}$ is illustrated in Fig. 4. The pure expansion of the product is $2X^2 - XY + 4XY - 2Y^2$. As an additional example of Definition 4.1, $\mathtt{E(C)} = [(X, 2), (XY, -1), (XY, 4), (Y, -2)]$.*

$\mathtt{E(C)}$ effectively[18] encodes the *reduced* form of 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 Φ to compute $\widetilde{\Phi}$, the input circuit $\mathtt{C}$ does not need to be in SMB/SOP form.

Example D.3 (Example for Definition 4.2). *Using the same factorization from Example D.2, $poly(|\mathtt{C}|) = (X + 2Y)(2X + Y) = 2X^2 + XY + 4XY + 2Y^2 = 2X^2 + 5XY + 2Y^2$. Note that this is not the same as the polynomial from Eq. (28). As an example of the slight abuse of notation we alluded to, $poly(|\mathtt{C}|(1, \ldots, 1)) = 2(1)^2 + 5(1)(1) + 2(1)^2 = 9$.*

Definition D.4 (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 $\mathtt{g}$.*

The following results assume input circuit $\mathtt{C}$ computed from an arbitrary $\mathcal{RA}^+$ query Q and arbitrary BIDB $\mathcal{D}$. We refer to $\mathtt{C}$ as a BIDB circuit.

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

Algorithm 1 Approximate$\widetilde{\Phi}(\mathtt{C}, \mathbf{p}, \delta, \epsilon)$

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**
    $\mathtt{Y_i} \leftarrow \prod_{X_j \in \mathtt{M}} p_j$ ▷ $\mathtt{M}$ is the sampled monomial's set of variables (cref. appendix D.9)
    $\mathtt{Y_i} \leftarrow \mathtt{Y_i} \times \mathtt{sgn_i}$
    $\mathtt{acc} \leftarrow \mathtt{acc} + \mathtt{Y_i}$ ▷ Store the sum over all samples
  end if
end for
$\mathtt{acc} \leftarrow \mathtt{acc} \times \frac{\mathtt{size}}{\mathtt{N}}$
return $\mathtt{acc}$

Theorem D.5. *Let $\mathtt{C}$ be an arbitrary* BIDB *circuit and define $\Phi(\mathbf{X}) = poly(\mathtt{C})$ and let $k = deg(\mathtt{C})$. Then an estimate $\mathcal{E}$ of $\widetilde{\Phi}(p_1, \ldots, p_n)$ can be computed in time*

$$O\left(\left(size(\mathtt{C}) + \frac{\log \frac{1}{\delta} \cdot |\mathtt{C}|^2(1, \ldots, 1) \cdot k \cdot \log k \cdot depth(\mathtt{C}))}{(\epsilon)^2 \cdot \widetilde{\Phi}^2(p_1, \ldots, p_n)}\right) \cdot \overline{\mathcal{M}}(\log(|\mathtt{C}|(1, \ldots, 1)), \log($$

such that

$$Pr\left(\left|\mathcal{E} - \widetilde{\Phi}(p_1, \ldots, p_n)\right| > \epsilon \cdot \widetilde{\Phi}(p_1, \ldots, p_n)\right) \leq \delta. \tag{29}$$

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

D.1 Proof of Theorem D.5

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

Lemma D.6. *The OnePass function completes in time:*

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

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 ($\mathtt{C_{mod}}$) output by OnePass, $\mathtt{C_{mod}.partial} = |\mathtt{C}|\,(1,\ldots,1)$.

Lemma D.7. *The function SampleMonomial completes in time*

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

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

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

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

$$Pr\left(\left|\mathtt{acc} - \widetilde{\Phi}(p_1,\ldots,p_n)\right| > \epsilon \cdot |\mathtt{C}|\,(1,\ldots,1)\right) \le \delta,$$

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

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

Proof of Theorem D.5. Set $\mathcal{E} = \textsc{ApproximateP}\widetilde{\Phi}(\mathtt{C}, (p_1,\ldots,p_n), \delta, \epsilon')$, where

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

which achieves the claimed error bound on $\mathcal{E}$ ($\mathtt{acc}$) trivially due to the assignment to ϵ' and theorem D.8, since $\epsilon' \cdot |\mathtt{C}|\,(1,\ldots,1) = \epsilon \cdot \frac{\widetilde{\Phi}(1,\ldots,1)}{|\mathtt{C}|\,(1,\ldots,1)} \cdot |\mathtt{C}|\,(1,\ldots,1) = \epsilon \cdot \widetilde{\Phi}(1,\ldots,1)$.

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

$$\frac{1}{(\epsilon')^2}\cdot\log\left(\frac{1}{\delta}\right) = \frac{\log\frac{1}{\delta}}{\epsilon^2\left(\frac{\widetilde{\Phi}(p_1,\ldots,p_N)}{|\mathtt{C}|\,(1,\ldots,1)}\right)^2} = \frac{\log\frac{1}{\delta}\cdot|\mathtt{C}|^2\,(1,\ldots,1)}{\epsilon^2\cdot\widetilde{\Phi}^2(p_1,\ldots,p_n)}.$$

□

Let us now prove Theorem D.8:

D.2 Proof of Theorem D.8

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

$$\mathtt{Y_i} = \mathbb{1}_{(\textsc{isInd}(\mathtt{v_m}))}\cdot\prod_{X_i\in\textsc{var}(v)} p_i,$$

where the indicator variable handles the check in Line 6 Then for random variable $\mathtt{Y}_i$, it is the case that

$$\mathbb{E}\left[\mathtt{Y_i}\right] = \sum_{(\mathtt{v},\mathtt{c})\in\mathtt{E(C)}}\frac{\mathbb{1}_{(\textsc{isInd}(\mathtt{v_m}))}\cdot c\cdot\prod_{X_i\in\textsc{var}(v)}p_i}{|\mathtt{C}|\,(1,\ldots,1)} = \frac{\widetilde{\Phi}(p_1,\ldots,p_n)}{|\mathtt{C}|\,(1,\ldots,1)},$$

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. (2) with X_i substituted by p_i.

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

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

Hoeffding's inequality states that if we know that each $\mathtt{Y}_i$ (which are all independent) always lie in the intervals $[a_i, b_i]$, then it is true that

$$Pr\left(\left|\overline{\mathtt{Y}} - \mathbb{E}\left[\overline{\mathtt{Y}}\right]\right| \ge \epsilon\right) \le 2\exp\left(-\frac{2N^2\epsilon^2}{\sum_{i=1}^{N}(b_i-a_i)^2}\right).$$

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

$$Pr\left(\left|\overline{\mathtt{Y}} - \mathbb{E}\left[\overline{\mathtt{Y}}\right]\right| \ge \epsilon\right) \le 2\exp\left(-\frac{2N^2\epsilon^2}{2^2N}\right) = 2\exp\left(-\frac{N\epsilon^2}{2}\right) \le \delta,$$

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

For the claimed probability bound of $Pr\left(\left|\mathtt{acc} - \widetilde{\Phi}(p_1,\ldots,p_n)\right| > \epsilon\cdot|\mathtt{C}|\,(1,\ldots,1)\right)$ δ, note that in the algorithm, $\mathtt{acc}$ is exactly $\overline{\mathtt{Y}}\cdot|\mathtt{C}|\,(1,\ldots,1)$. Multiplying the rest of the terms by the additional factor $|\mathtt{C}|\,(1,\ldots,1)$ yields the said bound.

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

Run-time Analysis. The runtime of the algorithm is dominated first by Line 3 (which by Lemma D.6 takes time $O\left(\textit{SIZE}(\mathtt{C})\cdot\overline{\mathcal{M}}\,(\log(|\mathtt{C}|\,(1,\ldots,1)),\right.$ and then by N iterations of the loop in Line 4. Each iteration's run time is dominated by the call to SampleMonomial in Line 5 (which by Lemma D.7 takes $O\left(\log k\cdot k\cdot\textsc{depth}(\mathtt{C})\cdot\overline{\mathcal{M}}\,(\log(|\mathtt{C}|\,(1,\ldots,1)),\log(\textsc{size}(\mathtt{C}))\right.$) and the check Line 6, which by the subsequent argument takes $O(k\log k)$ time. We sort the $O(k)$ variables by their block IDs and then check if there is a duplicate block ID or not. Combining all the times discussed here gives us the desired overall runtime. □

D.3 Proof of Theorem 4.7

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

$$\widetilde{\Phi}(1,\ldots,1) = (1-\gamma)\cdot|\mathtt{C}|\,(1,\ldots,1).$$

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

$$\widetilde{\Phi}(1,\ldots,1) \ge p_0^k\cdot\widetilde{\Phi}(1,\ldots,1).$$

The above two inequalities implies $\widetilde{\Phi}(1,\ldots,1) \ge p_0^k\cdot(1-\gamma)\cdot|\mathtt{C}|\,(1,\ldots,1)$. Applying this bound in the runtime bound in Theorem D.5 gives the first claimed runtime. The final runtime of $O_k\left(\frac{1}{\epsilon^2}\cdot\textsc{size}(\mathtt{C})\cdot\log\frac{1}{\delta}\cdot\overline{\mathcal{M}}\,(\log(|\mathtt{C}|\,(1,\ldots,1)),\log(\textsc{size}(\mathtt{C})))\right)$ follows by noting that $\textsc{depth}(\mathtt{C}) \le \textsc{size}(\mathtt{C})$ and absorbing all factors that just depend on k. □

D.4 Proof of Lemma 4.8

Proof. The circuit C' is built from C in the following manner. For each input gate $\mathtt{g}_i$ with $\mathtt{g}_i.\mathtt{val} = X_t$, replace $\mathtt{g}_i$ with the circuit S encoding the sum $\sum_{j=1}^{c} j \cdot X_{t,j}$. We argue that C' is a valid circuit by the following facts. Let $\mathcal{D} = \left(\{0,\ldots,c\}^D, \mathcal{P}\right)$ be the original c-TIDB C was generated from. Then, by Proposition 2.4 there exists a Binary-BIDB $\mathcal{D}' = (\bigtimes_{t\in D'} \{0, c_t\}, \mathcal{P}')$, with $D' = \left\{\int t, j \mid t \in D, j \in [c]\right\}$, from which the conversion from C to C' follows. Both poly (C) and poly (C′) have the same expected multiplicity since (by Proposition 2.4) the distributions $\mathcal{P}$ and $\mathcal{P}'$ are equivalent and each $j \cdot \mathbf{W}'_{t,j} = \mathbf{W}_t$ for $\mathbf{W}' \in \{0,1\}^{cn}$ and $\mathbf{W} \in \{0,\ldots,c\}^D$. Finally, note that because there exists a (sub) circuit encoding $\sum_{j=1}^{c} j \cdot X_{t,j}$ that is a *balanced* binary tree, the above conversion implies the claimed size and depth bounds of the lemma.

Next we argue the claim on γ (C′). Consider the list of expanded monomials E(C) for c-TIDB circuit C. Let $\mathtt{v}_\mathtt{m} = X_{t_1}^{d_1}, \ldots, X_{t_\ell}^{d_\ell}$ be an arbitrary monomial with ℓ variables. Then v yields the set of monomials $\mathtt{E}_\mathtt{v}(\mathtt{C}') = \left\{j_1^{d_1} \cdot X_{t,j_1}^{d_1} \times \cdots \times j_\ell^{d_\ell} \cdot X_{t,j_\ell}^{d_\ell}\right\}_{j_1,\ldots,j_\ell \in [c]}$ in E(C′). Recall that a cancellation occurs when we have a monomial v' such that there exists $t \neq t'$ in the same block B where variables $X_t, X_{t'}$ are in the set of variables $\mathtt{v}'_\mathtt{m}$ of v'. Observe that cancellations can only occur for each $X_t^{d_t} \in \mathtt{v}_\mathtt{m}$, where the expansion $\left(\sum_{j=1}^{c} j \cdot X_{t,j}\right)^{d_t}$ represents the monomial $X_t^{d_t}$ in D'. Consider the number of cancellations for $\left(\sum_{j=1}^{c} j \cdot X_{t,j}\right)^{d_t}$. Then $\gamma \leq 1 - c^{d_t - 1}$, since for each element in the set of cross products $\left\{\bigtimes_{i\in[d_t], j_i \in [c]} X_{t,j_i}\right\}$ there are *exactly* c surviving elements with $j_1 = \cdots = j_{d_t} = j$, i.e. $X_{t,j}^{d_t}$ for each $j \in [c]$. The rest of the $c^{d_t} - c$ cross terms cancel. Regarding the whole monomial v', it is the case that the proportion of non-cancellations across each $X_t^{d_t} \in \mathtt{v}'_\mathtt{m}$ multiply because non-cancelling terms for X_t can only be joined with non-cancelling terms of $X_{t'}^{d_{t'}} \in \mathtt{v}'_\mathtt{m}$ for $t \neq t'$. This then yields the fraction of cancelled monomials $\gamma \leq 1 - \prod_{i=1}^{\ell} c^{d_i - 1} \leq 1 - c^{-(k-1)}$ where the inequalities take into account the fact that $\sum_{i=1}^{\ell} d_i \leq k$.

Since this is true for arbitrary v, the bound follows for poly (C′). □

D.5 Proof of Lemma 4.9

We will prove Lemma 4.9 by considering the two cases separately. We start by considering the case when C is a tree:

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

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

Proof of Lemma D.9. For notational simplicity define $N = \text{size}(\mathtt{C})$ and $k = \text{deg}(\mathtt{C})$. We use induction on depth(C) to show that $|\mathtt{C}|(1,\ldots,1) \leq N^{k+1}$. For the base case, we have that depth (C) = 0, and there can only be one node which must contain a coefficient or constant. In this case, $|\mathtt{C}|(1,\ldots,1) = 1$, and size (C) = 1, and by Definition 4.4 it is the case that $0 \leq k = \text{deg}(\mathtt{C}) \leq 1$, and it is true that $|\mathtt{C}|(1,\ldots,1) = 1 \leq N^{k+1} = 1^{k+1} = 1$ for $k \in \{0,1\}$.

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

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

$$\begin{aligned}
|\mathtt{C}|(1,\ldots,1) &= |\mathtt{C}_\mathtt{L}|(1,\ldots,1) \cdot |\mathtt{C}_\mathtt{R}|(1,\ldots,1) \\
&\leq (N-1)^{k_\mathtt{L}+1} \cdot (N-1)^{k_\mathtt{R}+1} \\
&= (N-1)^{k+1} \qquad (30) \\
&\leq N^{k+1}.
\end{aligned}$$

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. (30) follows by definition 4.4 which states that for $k = \text{deg}(\mathtt{C})$ we have $k = k_\mathtt{L} + k_\mathtt{R} + 1$.

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

$$\begin{aligned}
|\mathtt{C}|(1,\ldots,1) &= |\mathtt{C}_\mathtt{L}|(1,\ldots,1) + |\mathtt{C}_\mathtt{R}|(1,\ldots,1) \\
&\leq N_\mathtt{L}^{k+1} + N_\mathtt{R}^{k+1} \\
&\leq (N-1)^{k+1} \qquad (31) \\
&\leq N^{k+1}.
\end{aligned}$$

In the above, the first inequality follows from the inductive hypothes and definition 4.4 (which implies 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 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 4.9 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 D.10. *Let* C *be a (general) circuit. Then we have*

$$|\mathtt{C}|(1,\ldots,1) \leq 2^{2^{\mathit{DEG}(\mathtt{C})} \cdot \mathit{DEPTH}(\mathtt{C})}.$$

Proof Sketch of Lemma D.10. We use the same notation as in the proof of Lemma D.9 and further define $d = \text{depth}(\mathtt{C})$. We will prove by induction on depth(C) that $|\mathtt{C}|(1,\ldots,1) \leq 2^{2^k \cdot d}$. The base case argument is similar to that in the proof of Lemma D.9. In the inductive case we have that $d_\mathtt{L}, d_\mathtt{R} \leq d - 1$.

For the case when the sink node is ×, we get that

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

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 $d_\mathtt{L}, d_\mathtt{R} \leq d - 1$, where we substitute the upperbound into every respective term.

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

$$|\mathtt{C}|(1,\ldots,1) = |\mathtt{C}_\mathtt{L}|(1,\ldots,1) + |\mathtt{C}_\mathtt{R}|(1,\ldots,1)$$

$$\leq 2^{2^{k_L} \cdot d_L} + 2^{2^{k_R} \cdot d_R}$$
$$\leq 2 \cdot 2^{2^k (d-1)}$$
$$\leq 2^{2^k d}.$$

In the above the first inequality follows from the inductive hypothesis while the second inequality follows from the facts that $k_L, k_R \leq k$ and $d_L, d_R \leq d-1$. The final inequality follows from the fact that $k \geq 0$. □

D.6 OnePass Remarks

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

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

$$|\mathsf{C}|(1,\ldots,1) = \begin{cases} |\mathsf{C}_L|(1,\ldots,1) \cdot |\mathsf{C}_R|(1,\ldots,1) & \textbf{if } \mathsf{C.type} = \times \\ |\mathsf{C}_L|(1,\ldots,1) + |\mathsf{C}_R|(1,\ldots,1) & \textbf{if } \mathsf{C.type} = + \\ |\mathsf{C.val}| & \textbf{if } \mathsf{C.type} = \textsc{num} \\ 1 & \textbf{if } \mathsf{C.type} = \textsc{var}. \end{cases} \tag{32}$$

It turns out that for proof of Lemma D.7, we need to argue that when C.type = +, we indeed have

$$\mathsf{C.Lweight} \leftarrow \frac{|\mathsf{C}_L|(1,\ldots,1)}{|\mathsf{C}_L|(1,\ldots,1) + |\mathsf{C}_R|(1,\ldots,1)}; \tag{33}$$

$$\mathsf{C.Rweight} \leftarrow \frac{|\mathsf{C}_R|(1,\ldots,1)}{|\mathsf{C}_L|(1,\ldots,1) + |\mathsf{C}_R|(1,\ldots,1)} \tag{34}$$

D.7 OnePass Example

Example D.11. *Let* T *encode the expression* $(X+Y)(X-Y)+Y^2$. *After one pass, Algorithm 2 would have computed the following weight distribution. For the two inputs of the sink gate* C, $\mathsf{C.Lweight} = \frac{4}{5}$ *and* $\mathsf{C.Rweight} = \frac{1}{5}$. *Similarly, for* S *denoting the left input of* C_L, $\mathsf{S.Lweight} = \mathsf{S.Rweight} = \frac{1}{2}$. *This is depicted in Fig. 5.*

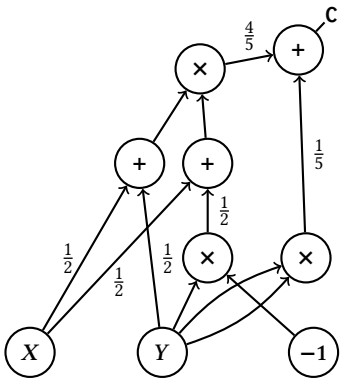


Figure 5: Weights computed by OnePass in Example D.11.

Algorithm 2 OnePass (C)

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

1: **for** g in TopOrd (C) **do** ▷ TopOrd (·) is the topological order of C
2: **if** g.type = var **then**
3: g.partial ← 1
4: **else if** g.type = num **then**
5: g.partial ← |g.val|
6: **else if** g.type = × **then**
7: g.partial ← $\mathsf{g_L}$.partial × $\mathsf{g_R}$.partial
8: **else**
9: g.partial ← $\mathsf{g_L}$.partial + $\mathsf{g_R}$.partial
10: g.Lweight ← $\frac{\mathsf{g_L.partial}}{\mathsf{g.partial}}$
11: g.Rweight ← $\frac{\mathsf{g_R.partial}}{\mathsf{g.partial}}$
12: **end if**
13: sum ← g.partial
14: **end for**
15: **return** (sum, C)

D.8 Proof of OnePass (Lemma D.6)

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

For the base case, we have only one gate, which by definition is a source gate and must be either var or num. In this case, as per eq. (32), lines 3 and 5 correctly compute C.partial as 1.

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

When g_{k+1}.type = +, then by line 9 g_{k+1}.partial = g_{k+1_L}.partial +g_{k+1_R}.partial, a correct computation, as per eq. (32). Further, lines 10 and 11 compute g_{k+1}.Lweight = $\frac{\mathsf{g}_{k+1_L}.\mathsf{partial}}{\mathsf{g}_{k+1}.\mathsf{partial}}$ and analogously for g_{k+1}.Rweight. All values needed for each computation have been correctly computed by the inductive hypothesis.

When g_{k+1}.type = ×, then line 7 computes g_{k+1}.partial = g_{k+1_L}.partial × g_{k+1_R}.partial, which indeed by eq. (32) is correct. This concludes the proof of correctness.

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

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 = num then                   ▷ The leaf is a coefficient
15:     return ({}, sgn(C.val))▷ sgn(·) outputs −1 for C.val ≥ 1 and −1 for C.val ≤ −1
16: else if C.type = var then
17:     return ({C.val}, 1)
18: end if
```

D.9 SampleMonomial Remarks

We briefly describe the top-down traversal of SampleMonomial. When C.type = +, the input to be visited is sampled from the weighted distribution precomputed by OnePass. When a C.type = × 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 C instead an expression tree T, we do not know that such a method exists when computing a sample of the input polynomial representation.

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

D.10 Proof of SampleMonomial (Lemma D.7)

Proof. We first need to show that SampleMonomial samples a valid monomial $\mathsf{v_m}$ by sampling and returning a set of variables v, such that (v, c) is in $\mathsf{E(C)}$ and $\mathsf{v_m}$ is indeed a monomial of the $\widetilde{\Phi}(\mathbf{X})$ encoded in C. We show this via induction over the depth of C. For the base case, let the depth d of C be 0. We have that the single gate is either a constant c for which by line 15 we return { }, or we have that C.type = var and C.val = x, and by line 17 we return $\{x\}$. By definition 4.1, both cases return a valid v for some (v, c) from $\mathsf{E(C)}$, 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 valid monomial.

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

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

For the base case $d = 0$, by definition 2.9 we know that the size (C) = 1 and C.type = num or var. For either case, the probability of the value returned is 1 since there is only one value to sample from. When C.val = x, the algorithm always return the variable set $\{x\}$. When C.type = num, SampleMonomial will always return $\emptyset$.

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

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

Consider the case when C.type = ×. For the term (v, c) from $\mathsf{E(C)}$ that is being sampled it is the case that $\mathsf{v} = \mathsf{v_L} \cup \mathsf{v_R}$, where $\mathsf{v_L}$ is coming from $\mathsf{C_L}$ and $\mathsf{v_R}$ from $\mathsf{C_R}$. The probability that SampleMonomial ($\mathsf{C_L}$) returns $\mathsf{v_L}$ is $\frac{|\mathsf{c_{v_L}}|}{|\mathsf{C_L}|(1,\ldots,1)}$ and $\frac{|\mathsf{c_{v_R}}|}{|\mathsf{C_R}|(1,\ldots,1)}$ for $\mathsf{v_R}$. Since both $\mathsf{v_L}$ and $\mathsf{v_R}$ are sampled with independent randomness, the final probability for sample v is then $\frac{|\mathsf{c_{v_L}}| \cdot |\mathsf{c_{v_R}}|}{|\mathsf{C_L}|(1,\ldots,1) \cdot |\mathsf{C_R}|(1,\ldots,1)}$. For (v, c) in $\mathsf{E(C)}$, by definition 4.1 it is indeed the case that $|\mathsf{c}| = |\mathsf{c_{v_L}}| \cdot |\mathsf{c_{v_R}}|$ and that (as shown in eq. (32)) $|\mathsf{C}|(1,\ldots,1) = |\mathsf{C_L}|(1,\ldots,1) \cdot |\mathsf{C_R}|(1,\ldots,1)$, and therefore v is sampled with correct probability $\frac{|\mathsf{c}|}{|\mathsf{C}|(1,\ldots,1)}$.

For the case when C.type = +, SampleMonomial will sample v from one of its inputs. By inductive hypothesis we know that any $\mathsf{v_L}$ in $\mathsf{E(C_L)}$ and any $\mathsf{v_R}$ in $\mathsf{E(C_R)}$ will both be sampled with correct probability $\frac{|\mathsf{c_{v_L}}|}{|\mathsf{C_L}|(1,\ldots,1)}$ and $\frac{|\mathsf{c_{v_R}}|}{|\mathsf{C_R}|(1,\ldots,1)}$, where either $\mathsf{v_L}$ or $\mathsf{v_R}$ will equal v, depending on whether $\mathsf{C_L}$ or $\mathsf{C_R}$ is sampled. Assume that v is sampled from $\mathsf{C_L}$, and note that a symmetric argument holds for the case when v is sampled from $\mathsf{C_R}$. Notice also that the probability of choosing $\mathsf{C_L}$ from C is $\frac{|\mathsf{C_L}|(1,\ldots,1)}{|\mathsf{C_L}|(1,\ldots,1)+|\mathsf{C_R}|(1,\ldots,1)}$ 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 C being independent from the

randomness used in its subtrees), the probability for v to be sampled from C is equal to the product of the probability that C_L is sampled from C and v is sampled in C_L, and

$$\begin{aligned}
&Pr(\textsc{SampleMonomial}(C) = v) = \\
&Pr(\textsc{SampleMonomial}(C_L) = v) \cdot Pr(\mathit{SampledChild}(C) = C_L) \\
&= \frac{|c_v|}{|C_L|(1,\ldots,1)} \cdot \frac{|C_L|(1,\ldots,1)}{|C_L|(1,\ldots,1) + |C_R|(1,\ldots,1)} \\
&= \frac{|c_v|}{|C|(1,\ldots,1)},
\end{aligned}$$

and we obtain the desired result.

Lastly, we show by simple induction of the depth d of C that SampleMonomial indeed returns the correct sign value of c in (v, c).

In the base case, C.type = NUM or VAR. For the former, SampleMonomial correctly returns the sign value of the gate. For the latter, SampleMonomial returns the correct sign of 1, since a variable is a neutral element, and 1 is the multiplicative identity, whose product with another sign element will not change that sign element.

For the inductive hypothesis, we assume for a circuit of depth $d \le k$ and $k \ge 0$ that the algorithm correctly returns the sign value of c.

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

Run-time Analysis. It is easy to check that except for lines 3 and 10, all lines take $O(1)$ time. Consider an execution of line 10. We note that we will be adding a given set of variables to some set at most once: since the sum of the sizes of the sets at a given level is at most DEG(C), each gate visited takes $O(\log \textsc{deg}(C))$. For Line 3, note that we pick C_L with probability $\frac{a}{a+b}$ where a = C.Lweight and b = C.Rweight. We can implement this step by picking a random number $r \in [a+b]$ and then checking if $r \le a$. It is easy to check that $a + b \le |C|(1,\ldots,1)$. This means we need to add and compare $\log |C|(1,\ldots,1)$-bit numbers, which can certainly be done in time $\overline{M}(\log(|C(1\ldots,1)|), \log \textsc{size}(C))$ (note that this is an overestimate). Denote Cost (C) (Eq. (35)) to be an upper bound of the number of gates visited by SampleMonomial. Then the runtime is $O\left(\textsc{Cost}(C) \cdot \log \textsc{deg}(C) \cdot \overline{M}(\log(|C(1\ldots,1)|), \log \textsc{size}(C))\right)$.

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

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

$$\textsc{Cost}(C) = \begin{cases} 1 + \textsc{Cost}(C_L) + \textsc{Cost}(C_R) & \textbf{if } C.\mathtt{type} = \times \\ 1 + \max(\textsc{Cost}(C_L), \textsc{Cost}(C_R)) & \textbf{if } C.\mathtt{type} = + \\ 1 & \textbf{otherwise} \end{cases} \quad (35)$$

First note that the number of gates visited in SampleMonomial is ≤ Cost(C). To show that eq. (35) upper bounds the number of nodes visited by SampleMonomial, note that when SampleMonomial visits a gate such that C.type = ×, line 8 visits each input of C, as defined in (35). For the case when 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 Cost (·) an upperbound. Finally, it is trivial to see that when C.type ∈ {VAR, NUM}, i.e., a source gate, that only one gate is visited.

We prove the following inequality holds.

$$2(\textsc{deg}(C)+1) \cdot \textsc{depth}(C) + 1 \ge \textsc{Cost}(C) \quad (36)$$

Note that eq. (36) implies the claimed runtime. We prove eq. (36) for the number of gates traversed in SampleMonomial using induction over DEPTH(C). Recall how degree is defined in definition 4.4.

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

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

$$2(\textsc{deg}(C_i)+1) \cdot \textsc{depth}(C_i) + 1 \ge \textsc{Cost}(C_i). \quad (37)$$

In particular, since for any i, eq. (37) holds, then it immediately follows that an inequality whose operands consist of a sum of the aforementioned inequalities must also hold. This is readily seen in the inequality of eq. (39) and eq. (40), where $2(\textsc{deg}(C_L)+1) \cdot \textsc{depth}(C_L) \ge \textsc{Cost}(C_L)$, likewise for C_R, and $1 \ge 1$. It is also true that $\textsc{depth}(C_L) \le \textsc{depth}(C) - 1$ and $\textsc{depth}(C_R) \le \textsc{depth}(C) - 1$.

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

If C.type = ×, then, by eq. (35), substituting values, the following should hold,

$$2(\textsc{deg}(C_L) + \textsc{deg}(C_R) + 2) \cdot (\max(\textsc{depth}(C_L), \textsc{depth}(C_R)) + 1) + 1 \quad (38)$$

$$\ge 2(\textsc{deg}(C_L)+1) \cdot \textsc{depth}(C_L) + 2(\textsc{deg}(C_R)+1) \cdot \textsc{depth}(C_R) + 3 \quad (39)$$

$$\ge 1 + \textsc{Cost}(C_L) + \textsc{Cost}(C_R) = \textsc{Cost}(C). \quad (40)$$

To prove (39), first, eq. (38) expands to,

$$2\textsc{deg}(C_L)\cdot\textsc{depth}_{\max} + 2\textsc{deg}(C_R)\cdot\textsc{depth}_{\max} + 4\textsc{depth}_{\max} + 2\textsc{deg}(C_L) + 2\textsc{deg}(C_R) + 4 + 1 \quad (41)$$

where $\textsc{depth}_{\max}$ is used to denote the maximum depth of the two input subcircuits. Eq. (39) expands to

$$2\textsc{deg}(C_L)\cdot\textsc{depth}(C_L) + 2\textsc{depth}(C_L) + 2\textsc{deg}(C_R)\cdot\textsc{depth}(C_R) + 2\textsc{depth}(C_R) + 3 \quad (42)$$

Putting Eq. (41) and Eq. (42) together we get

$$2\textsc{deg}(C_L) \cdot \textsc{depth}_{\max} + 2\textsc{deg}(C_R) \cdot \textsc{depth}_{\max} + 4\textsc{depth}_{\max} + 2\textsc{deg}(C_L) + 2\textsc{deg}(C_R)$$
$$\ge 2\textsc{deg}(C_L) \cdot \textsc{depth}(C_L) + 2\textsc{deg}(C_R) \cdot \textsc{depth}(C_R) + 2\textsc{depth}(C_L) + 2\textsc{depth}(C \quad (43)$$

Since the following is always true,

$$2\textsc{deg}(C_L) \cdot \textsc{depth}_{\max} + 2\textsc{deg}(C_R) \cdot \textsc{depth}_{\max} + 4\textsc{depth}_{\max} + 5$$

$$\geq 2\textsc{deg}(\mathsf{C_L}) \cdot \textsc{depth}(\mathsf{C_L}) + 2\textsc{deg}(\mathsf{C_R}) \cdot \textsc{depth}(\mathsf{C_R}) + 2\textsc{depth}(\mathsf{C_L}) + 2\textsc{depth}(\mathsf{C_R}) + 3,$$

then it is the case that Eq. (43) is *always* true.

Now to justify (40) which holds for the following reasons. First, eq. (40) is the result of Eq. (35) when C.type = ×. Eq. (39) is then produced by substituting the upperbound of (37) for each $\textsc{Cost}(\mathsf{C}_i)$, trivially establishing the upper bound of (40). This proves eq. (36) for the × case.

For the case when C.type = +, substituting values yields

$$2\left(\max(\textsc{deg}(\mathsf{C_L}), \textsc{deg}(\mathsf{C_R})) + 1\right) \cdot \left(\max(\textsc{depth}(\mathsf{C_L}), \textsc{depth}(\mathsf{C_R})) + 1\right) + 1 \tag{44}$$

$$\geq \max\left(2\left(\textsc{deg}(\mathsf{C_L}) + 1\right) \cdot \textsc{depth}(\mathsf{C_L}) + 1, 2\left(\textsc{deg}(\mathsf{C_R}) + 1\right) \cdot \textsc{depth}(\mathsf{C_R}) + 1\right) + 1 \tag{45}$$

$$\geq 1 + \max(\textsc{Cost}(\mathsf{C_L}), \textsc{Cost}(\mathsf{C_R})) = \textsc{Cost}(\mathsf{C}) \tag{46}$$

To prove (45), eq. (44) expands to

$$2\textsc{deg}_{\max}\textsc{depth}_{\max} + 2\textsc{deg}_{\max} + 2\textsc{depth}_{\max} + 2 + 1. \tag{47}$$

Since $\textsc{deg}_{\max} \cdot \textsc{depth}_{\max} \geq \textsc{deg}(\mathsf{C}_i) \cdot \textsc{depth}(\mathsf{C}_i)$, the following upper bound holds for the expansion of eq. (45):

$$2\textsc{deg}_{\max}\textsc{depth}_{\max} + 2\textsc{depth}_{\max} + 2 \tag{48}$$

Putting it together we obtain the following for (45):

$$2\textsc{deg}_{\max}\textsc{depth}_{\max} + 2\textsc{deg}_{\max} + 2\textsc{depth}_{\max} + 3 \geq 2\textsc{deg}_{\max}\textsc{depth}_{\max} + 2\textsc{depth}_{\max} + 2, \tag{49}$$

where it can be readily seen that the inequality stands and (49) follows. This proves (45).

Similar to the case of C.type = ×, (46) follows by equations (35) and (37).

This proves (36) as desired. □

D.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 Theorem 4.7, 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 [5], 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. 6 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. 6 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 Theorem 4.7 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 6: Number of Cancellations for Queries Over BIDB**.**

E CIRCUITS

E.1 Representing Polynomials with Circuits

E.1.1 Circuits for query plans. We now formalize circuits and the construction of circuits for $\mathcal{RA}^+$ queries. As mentioned earlier, we represent lineage polynomials as arithmetic circuits over $\mathbb{N}$-valued variables with +, ×. A circuit for query Q and $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathbf{X}]}$ is a directed acyclic graph $\left\langle V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}, E_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}, \phi_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}, \ell_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}} \right\rangle$ with vertices $V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}$ and directed edges $E_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}} \subset {V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}}^2$. The sink function $\phi_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}} : \mathcal{U}^n \to V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}$ is a partial function that maps the tuples of the n-ary relation $Q(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})$ to vertices. We require that $\phi_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}$'s range be limited to sink vertices (i.e., vertices with out-degree 0). A function $\ell_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}} : V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}} \to \{+, \times\} \cup \mathbb{N} \cup \mathbf{X}$ 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 ×. We require that vertices have an in-degree of at most two. Note that we can construct circuits for BIDBs in time linear in the time required for deterministic query processing over a possible world of the BIDB under the aforementioned assumption that $\left|\mathcal{D}_{\mathbb{N}[\mathbf{X}]}\right| \leq c \cdot |D|$.

E.2 Modeling Circuit Construction

We now connect the size of a circuit (where the size of a circuit is the number of vertices in the corresponding DAG) for a given $\mathcal{RA}^+$ query Q and $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathbf{X}]}$ to the runtime $T_{det}(Q, D_\Omega)$ of the PDB's deterministic bounding database D_Ω. 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 $v \in V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}$ in the arithmetic circuit for

$$\left\langle V_{Q,\mathcal{D}_{\mathbb{N}[X]}}, E_{Q,\mathcal{D}_{\mathbb{N}[X]}}, \phi_{Q,\mathcal{D}_{\mathbb{N}[X]}}, \ell_{Q,\mathcal{D}_{\mathbb{N}[X]}} \right\rangle$$

encodes a polynomial, realized as

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

We define the circuit for a $\mathcal{RA}^+$ query Q recursively by cases as follows. In each case, let $\left\langle V_{Q_i,\mathcal{D}_{\mathbb{N}[X]}}, E_{Q_i,\mathcal{D}_{\mathbb{N}[X]}}, \phi_{Q_i,\mathcal{D}_{\mathbb{N}[X]}}, \ell_{Q_i,\mathcal{D}_{\mathbb{N}[X]}} \right\rangle$ denote the circuit for subquery Q_i. We implicitly include in all circuits a global zero node v_0 s.t., $\ell_{Q,\mathcal{D}_{\mathbb{N}[X]}}(v_0) = 0$ for any $Q, \mathcal{D}_{\mathbb{N}[X]}$.

Algorithm 4 defines how the circuit for a query result is constructed. We quickly review the number of vertices emitted in each case.

Base Relation. This circuit has $|D_\Omega.R|$ vertices.

Selection. If we assume dead sinks are iteratively garbage collected, this circuit has at most $|V_{Q_1,\mathcal{D}_{\mathbb{N}[X]}}|$ vertices.

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

Union. This circuit has $|V_{Q_1,\mathcal{D}_{\mathbb{N}[X]}}| + |V_{Q_2,\mathcal{D}_{\mathbb{N}[X]}}| + |Q_1 \cap Q_2|$ vertices.

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

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

PROPOSITION E.1 (CIRCUIT DEPTH IS BOUNDED). *Let Q be a relational query and D_Ω be a deterministic bounding database with n tuples. There exists a (lineage) circuit $\mathtt{C}^*$ encoding the lineage of all tuples $t \in Q(D_\Omega)$ for which $\textsc{depth}(\mathtt{C}^*) \leq O(k|Q|\log(n))$.*

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

Algorithm 4 LC $(Q, D_\Omega, E, V, \ell)$

Input: Q: query
Input: D_Ω: a deterministic bounding database
Input: E, V, ℓ: accumulators for the edge list, vertex list, and vertex label list.
Output: $\mathtt{C} = \langle E, V, \phi, \ell \rangle$: a circuit encoding the lineage of each tuple in $Q(D_\Omega)$

if Q is R **then** ▷ **Case 1**: Q is a relation atom
  for $t \in D_\Omega.R$ **do**
    $V \leftarrow V \cup \{v_t\}$; $\ell \leftarrow \ell \cup \{(v_t, R(t))\}$ ▷ Allocate a fresh node v_t
    $\phi(t) \leftarrow v_t$
  end for
else if Q is $\sigma_\theta(Q')$ **then** ▷ **Case 2**: Q is a Selection
  $\langle V, E, \phi', \ell \rangle \leftarrow LC(Q', D_\Omega, V, E, \ell)$
  for $t \in \textsc{Dom}(\phi')$ **do**
    if $\theta(t)$ **then** $\phi(t) \leftarrow \phi'(t)$ **else** $\phi(t) \leftarrow v_0$
  end for
else if Q is $\pi_{\vec{A}}(Q')$ **then** ▷ **Case 3**: Q is a Projection
  $\langle V, E, \phi', \ell \rangle \leftarrow LC(Q', D_\Omega, V, E, \ell)$
  for $t \in \pi_{\vec{A}}(Q'(D_\Omega))$ **do**
    $V \leftarrow V \cup \{v_t\}$; $\ell \leftarrow \ell \cup \{(v_t, +)\}$ ▷ Allocate a fresh node v_t
    $\phi(t) \leftarrow v_t$
  end for
  for $t \in Q'(D_\Omega)$ **do**
    $E \leftarrow E \cup \{(\phi'(t), \phi(\pi_{\vec{A}} t))\}$
  end for
  Correct nodes with in-degrees > 2 by appending an equivalent fan-in two tree instead
else if Q is $Q_1 \cup Q_2$ **then** ▷ **Case 4**: Q is a Bag Union
  $\langle V, E, \phi_1, \ell \rangle \leftarrow LC(Q_1, D_\Omega, V, E, \ell)$
  $\langle V, E, \phi_2, \ell \rangle \leftarrow LC(Q_2, D_\Omega, V, E, \ell)$
  $\phi \leftarrow \phi_1 \cup \phi_2$
  for $t \in \textsc{Dom}(\phi_1) \cap \textsc{Dom}(\phi_2)$ **do**
    $V \leftarrow V \cup \{v_t\}$; $\ell \leftarrow \ell \cup \{(v_t, +)\}$ ▷ Allocate a fresh node v_t
    $\phi(t) \leftarrow v_t$
    $E \leftarrow E \cup \{(\phi_1(t), v_t), (\phi_2(t), v_t)\}$
  end for
else if Q is $Q_1 \bowtie \ldots \bowtie Q_m$ **then** ▷ **Case 5**: Q is a m-ary Join
  for $i \in [m]$ **do**
    $\langle V, E, \phi_i, \ell \rangle \leftarrow LC(Q_i, D_\Omega, V, E, \ell)$
  end for
  for $t \in \textsc{Dom}(\phi_1) \bowtie \ldots \bowtie \textsc{Dom}(\phi_m)$ **do**
    $V \leftarrow V \cup \{v_t\}$; $\ell \leftarrow \ell \cup \{(v_t, \times)\}$ ▷ Allocate a fresh node v_t
    $\phi(t) \leftarrow v_t$
    $E \leftarrow E \cup \left\{ (\phi_i(\pi_{sch(Q_i(D_\Omega))}(t)), v_t) \mid i \in [n] \right\}$
  end for
  Correct nodes with in-degrees > 2 by appending an equivalent fan-in two tree instead
end if

E.2.2 Circuit size vs. runtime.

Lemma E.2. *Given a $\mathbb{N}[\mathbf{X}]$-encoded PDB $\mathcal{D}_{\mathbb{N}[\mathbf{X}]}$ with deterministic bounding database D_Ω, and an $\mathcal{RA}^+$ query Q, the runtime of Q over D_Ω has the same or greater complexity as the size of the lineage of $Q(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})$. That is, we have $\left|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}\right| \leq kT_{det}(Q, D_\Omega) + 1$, where $k \geq 1$ is the maximal degree of any polynomial in $Q(\mathcal{D}_{\mathbb{N}[\mathbf{X}]})$.*

Proof. We prove by induction that $\left|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]} \setminus \{v_0\}}\right| \leq kT_{det}(Q, D_\Omega)$. For clarity, we implicitly exclude v_0 in the proof below.

The base case is a base relation: $Q = R$ and is trivially true since $|V_{R,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| = |D_\Omega.R| = T_{det}(R, D_\Omega)$ (note that here the degree $k = 1$). For the inductive step, we assume that we have circuits for subqueries $Q_1, \ldots, Q_m$ such that $|V_{Q_i,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| \leq k_i T_{det}(Q_i, D_\Omega)$ where k_i is the degree of Q_i.

Selection. Assume that $Q = \sigma_\theta(Q_1)$. In the circuit for Q, $|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| = |V_{Q_1,D_\Omega}|$ vertices, so from the inductive assumption and $T_{det}(Q, D_\Omega) = T_{det}(Q_1, D_\Omega)$ by definition, we have $|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| \leq kT_{det}(Q, D_\Omega)$.

Projection. Assume that $Q = \pi_{\mathbf{A}}(Q_1)$. The circuit for Q has at most $|V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |Q_1|$ vertices.

$$|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| \leq |V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |Q_1|$$

(From the inductive assumption)

$$\leq kT_{det}(Q_1, D_\Omega) + |Q_1|$$

(By definition of $T_{det}(Q, D_\Omega)$)

$$\leq kT_{det}(Q, D_\Omega).$$

Union. Assume that $Q = Q_1 \cup Q_2$. The circuit for Q has $|V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |V_{Q_2,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |Q_1 \cap Q_2|$ vertices.

$$|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| \leq |V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |V_{Q_2,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + |Q_1| + |Q_2|$$

(From the inductive assumption)

$$\leq k(T_{det}(Q_1, D_\Omega) + T_{det}(Q_2, D_\Omega)) + (|Q_1| + |Q_2|)$$

(By definition of $T_{det}(Q, D_\Omega)$)

$$\leq k(T_{det}(Q, D_\Omega)).$$

m-ary Join. Assume that $Q = Q_1 \bowtie \ldots \bowtie Q_m$. Note that $k = \sum_{i=1}^m k_i \geq m$. The circuit for Q has $|V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + \ldots + |V_{Q_k,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + (m-1)|Q_1 \bowtie \ldots \bowtie Q_k|$ vertices.

$$|V_{Q,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| = |V_{Q_1,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + \ldots + |V_{Q_k,\mathcal{D}_{\mathbb{N}[\mathbf{X}]}}| + (m-1)|Q_1 \bowtie \ldots \bowtie Q_k|$$

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

$$\begin{aligned}&\leq kT_{det}(Q_1, D_\Omega) + \ldots + kT_{det}(Q_k, D_\Omega) + \\ &\quad (m-1)|Q_1 \bowtie \ldots \bowtie Q_m| \\ &\leq k(T_{det}(Q_1, D_\Omega) + \ldots + T_{det}(Q_m, D_\Omega) + \\ &\quad |Q_1 \bowtie \ldots \bowtie Q_m|)\end{aligned}$$

(By definition of $T_{det}(Q, D_\Omega)$ and assumption on $T_{join}(\cdot)$)

$$\leq kT_{det}(Q, D_\Omega).$$

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

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

Lemma E.3. *Given a query Q over a deterministic bounding database D_Ω and the $\mathtt{C}^*$ output by Algorithm 4, the runtime $T_{LC}(Q, D_\Omega, \mathtt{C}^*) \leq O(T_{det}(Q, D_\Omega))$.*

Proof. By analysis of Algorithm 4, invoked as $\mathtt{C}^* \leftarrow LC(Q, D_\Omega, \{v_0\}, \emptyset, \{(v_0, 0)\}$

We assume that the vertex list V, edge list E, and vertex label list ℓ are mutable accumulators with $O(1)$ ammortized append. We assume that the tuple to sink mapping ϕ is a linked hashmap, with $O(1)$ insertions and retrievals, and $O(n)$ iteration over the domain of keys. We assume that the n-ary join $\textsc{Dom}(\phi_1) \bowtie \ldots \bowtie \textsc{Dom}(\phi_n)$ can be computed in time $T_{join}(\textsc{Dom}(\phi_1), \ldots, \textsc{Dom}(\phi_n))$ (Definition 2.13) and that an intersection $\textsc{Dom}(\phi_1) \cap \textsc{Dom}(\phi_2)$ can be computed in time $O(|\textsc{Dom}(\phi_1)| + |\textsc{Dom}(\phi_2)|)$ (e.g., with a hash table).

Before proving our runtime bound, we first observe that $T_{det}(Q, D) \geq \Omega(|Q(D)|)$. This is true by construction for the relation, projection, and union cases, by Definition 2.13 for joins, and by the observation that $|\sigma(R)| \leq |R|$.

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

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

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

Bag Union. As above, the recursive calls explicitly correspond to terms in the expansion of $T_{det}(Q_1 \cup Q_2, D_\Omega)$. Initializing ϕ (line 24) can be accomplished in $O(\textsc{Dom}(\phi_1) + \textsc{Dom}(\phi_2)) = O(|Q_1(D_\Omega)| + |Q_2(D_\Omega)|) \leq O(T_{det}(Q_1, D_\Omega) + T_{det}(Q_2, D_\Omega))$. The remainder requires computing $Q_1 \cup Q_2$ (line 25) and iterating over it (lines 25–29), which is $O(|Q_1| + |Q_2|)$ as noted above — this directly corresponds to terms in $T_{det}(Q_1 \cup Q_2, D_\Omega)$.

m-ary Join. As in the prior cases, recursive calls explicitly correspond to terms in our target runtime. The remaining logic involves (i) computing $\textsc{Dom}(\phi_1) \bowtie \ldots \bowtie \textsc{Dom}(\phi_m)$, (ii) iterating over the results, and (iii) creating a fan-in tree. Respectively, these are:
(i) $T_{join}(\textsc{Dom}(\phi_1), \ldots, \textsc{Dom}(\phi_m))$
(ii) $O(|Q_1(D_\Omega) \bowtie \ldots \bowtie Q_m(D_\Omega)|) \leq O(T_{join}(\textsc{Dom}(\phi_1), \ldots, \textsc{Dom}(\phi_m)))$ (Definition 2.13)

(iii) $O(m|Q_1(D_\Omega) \bowtie \ldots \bowtie Q_m(D_\Omega)|)$ (as (ii), noting that $m \leq k = O(1)$) □

F HIGHER MOMENTS

We make a simple observation to conclude the presentation of our results. So far we have only focused on the expectation of Φ. In addition, we could e.g. prove bounds of the probability of a tuple's multiplicity being at least 1. Progress can be made on this as follows: For any positive integer m we can compute the m-th moment of the multiplicities, allowing us to e.g. use the Chebyschev inequality or other high moment based probability bounds on the events we might be interested in. We leave further investigations for future work.

G THE KARP-LUBY ESTIMATOR

Computing the marginal probability of a tuple in the output of a set-probabilistic database query has been studied extensively. To the best of our knowledge, the current state of the art approximation algorithm for this problem is the Karp-Luby estimator [32], which first appeared in MayBMS/Sprout [40], and more recently as part of an online "anytime" approximation algorithm [16, 21].

The estimator works by observing that for any ℓ random binary (but not necessarily independent) events $\mathbf{W}_1, \ldots, \mathbf{W}_\ell$, the probability of at least one event occurring (i.e., $Pr(\mathbf{W}_1 \vee \ldots \vee \mathbf{W}_\ell)$) is bounded from above by the sum of the independent event probabilities (i.e., $Pr(\mathbf{W}_1 \vee \ldots \vee \mathbf{W}_\ell) \leq Pr(\mathbf{W}_1) + \ldots + Pr(\mathbf{W}_\ell)$). Starting from this ('easily' computable and large) value, the estimator proceeds to correct the estimate by estimating how much of an over-estimate it is. Specifically, if $\mathcal{P}$ is the joint distribution over $\mathbf{W}$, the estimator computes an approximation of:

$$O = \mathop{\mathbb{E}}_{\mathbf{W} \sim \mathcal{P}} \left[|\{ i \mid \mathbf{W}_i = 1, i \in [\ell] \}| \right].$$

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

$$Pr(\mathbf{W}_1 \vee \ldots \vee \mathbf{W}_\ell) = Pr(\mathbf{W}_1) + \ldots + Pr(\mathbf{W}_\ell) - O$$

The Karp-Luby estimator is employed on the SMB representation[19] of $\mathtt{C}$ (to solve the set-PDB version of Problem 1.6), where each W_i represents the event that one monomial is true. By simple inspection, if there are ℓ monomials, this estimator has runtime $\Omega(\ell)$. Further, a minimum of $\left\lceil \frac{3 \cdot \ell \cdot \log(\frac{2}{\delta})}{\epsilon^2} \right\rceil$ invocations of the estimator are required to achieve $1 \pm \epsilon$ approximation with probability at least $1 - \delta$ [40], entailing a runtime at least quadratic in ℓ. As an arbitrary lineage circuit $\mathtt{C}$ may encode $\Omega\left(|\mathtt{C}|^k\right)$ monomials, the worst case runtime is at least $\Omega\left(|\mathtt{C}|^{2k}\right)$ (where k is the 'degree' of lineage polynomial encoded by $\mathtt{C}$). By contrast note that by the discussion after Lemma 4.9 we can solve Problem 1.6 in time $O\left(|\mathtt{C}|^2\right)$ for all BIDB circuits *independent* of the degree k.

[19] Note that since we are in the set semantics, in the lineage polynomial/formula, addition is logical OR and multiplication is logical AND.

H PARAMETERIZED COMPLEXITY

In Sec. 3, we utilized common conjectures from fine-grained complexity theory. The notion of $\#\mathsf{W}[1] - hard$ is a standard notion in *parameterized complexity*, which by now is a standard complexity tool in providing data complexity bounds on query processing results [24]. E.g. the fact that k-matching is $\#\mathsf{W}[1] - hard$ 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. $\#\mathsf{W}[1] - hard$ for the general k-matching problem does not imply anything specific for the 3-matching problem. Similar questions have led to intense research into the new sub-field of *fine-grained complexity* (see [50]), where we care about the exponent in our hardness assumptions as well– e.g. Conjecture 3.3 is based on the popular *Triangle detection hypothesis* in this area (cf. [36]).