You should NEVER use a term before defining it.

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 Q [1], and result tuple t, compute the expected multiplicity of t: $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}\left[Q(\mathbf{W})(t)\right]$.*

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 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 [13]. 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 by simply computing the expectation over a 'sum-of-products' representation of the query operations of $Q(\mathcal{D})(t)$. 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 equivalent deterministic query. 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 $T_{det}\left(\text{OPT}(Q), \overline{D}, c\right)$ (see Sec. 2.4 for further details) denote the runtime for query $\text{OPT}(Q)$, deterministic database $\overline{D}$, and multiplicity bound c. Being we consider $\mathcal{RA}^+$ queries in which order of operators can impact runtime, we denote the optimal query as $\text{OPT}(Q) = \min_{Q' \in \mathcal{RA}^+, Q' \equiv Q} T_{det}\left(Q', \overline{D}, c\right)$.

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

Table 1 Our lower bounds for a specific hard query Q 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).

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

[1] A query Q is an $\mathcal{RA}^+$ query if it is composed entirely of one or more of the positive relational operators $\{\sigma, \pi, \bowtie, \cup\}$.

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

For an upper bound on approximating the expected count, it is easy to check that if all the probabilties are constant then $\Phi\left(p_1,\ldots,p_n\right)$ (i.e. evaluating the original lineage polynomial over the probability values) is a constant factor approximation. For example, using Q^2 from above, using p_A to denote $Pr\left[A=1\right]$ (and similarly for the other variables), we can see that

$$\begin{aligned}\Phi_1^2\left(\mathbf{p}\right) &= p_A^2p_X^2p_B^2+p_B^2p_Y^2p_E^2+p_B^2p_Z^2p_C^2+2p_Ap_Xp_B^2p_Yp_E+2p_Ap_Xp_B^2p_Zp_C+2p_B^2p_Yp_Ep_Zp_C\\ &\le p_Ap_Xp_B+p_Bp_Yp_E+p_Bp_Zp_C+2p_Ap_Xp_Bp_Yp_E+2p_Ap_Xp_Bp_Zp_C+2p_Bp_Yp_Ep_Zp_C=\widetilde{\Phi}_1^2\left(\mathbf{p}\right)\end{aligned}$$

If we assume that all seven probability values are at least $p_0>0$, we get that $\Phi_1^2\left(\mathbf{p}\right)$ is in the range $\left[\left(p_0\right)^3\cdot\widetilde{\Phi}_1^2\left(\mathbf{p}\right),\widetilde{\Phi}_1^2\left(\mathbf{p}\right)\right]$. 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 and solve Problem 1.6, using $\mathtt{C}$ we uniformly sample monomials from the equivalent SMB representation of Φ (without materializing the SMB representation) and 'adjust' their contribution to $\widetilde{\Phi}\left(\cdot\right)$.

Applications. Recent work in heuristic data cleaning [49, 43, 40, 8, 43] 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 [35], they are impractically slow [18, 17], 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.

If the idea is you want to preserve X for Φ use Y or Z.

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

Why are you using S instead of X?

A polynomial over a set of variables $\mathbf{S}$ with $|S|=m$ and individual degree $B<\infty$ is formally defined as (where $c_{\mathbf{d}}\in\mathbb{N}$):

$$\Phi\left(S_1,\ldots,S_m\right)=\sum_{\mathbf{d}\in\{0,\ldots,B\}^D}c_{\mathbf{d}}\cdot\prod_{i\in[m]}S_i^{d_i}. \tag{1}$$

Perhaps better to use some other notation than Φ for a general poly & use Φ only for lineage polys.

don't match

▶ **Definition 2.1** (Standard Monomial Basis). *The term* $\prod_{i\in[m]}S_i^{d_i}$ *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 $\mathrm{SMB}\left(\Phi\right)$ to denote the SMB form of a polynomial Φ.

▶ **Definition 2.2** (Degree). *The degree of polynomial* $\Phi(\mathbf{X})$ *is the largest* $\cancel{d}=\sum_{i\in[m]}d_i$ *such that* $c_{(d_1,\ldots,d_n)}\neq 0$. *We denote the degree of* Φ *as* $\deg\left(\Phi\right)$.

~~Can just say that D' is disjoint union of sets of tuples B1, …, Bn~~

~~Also uppercase for blocks is better~~

~~if clear from the context~~ PROPAGATE

~~I think this is for confusing. If you want you can say 1-BIDB is a gen. of 1-TIDB~~

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 there exists a $\mathcal{RA}^+$ query Q, c-TIDB $\mathcal{D}$, and result tuple t such that $\Phi(\mathbf{X}) = \Phi[Q,D,t](\mathbf{X})$.

not defined

~~D' is partitioned not defined.~~

2.2 1-BIDB

A block independent database BIDB $\mathcal{D}'$ can viewed as a 1-TIDB $\mathcal{D}$ with the added flexibility that each $t \in D$ has multiple disjoint alternatives, i.e., all $t \in D'$ are partitioned into m independent blocks with the condition that tuples $t \in b_i$ for $i \in [m]$ are disjoint events. We define next a specific construction of BIDB that is useful for our work.

Actually use times

~~but you sentence seems to suggest t is partitioned~~

▶ **Definition 2.3** (1-BIDB). *Define a 1-BIDB to be the pair $\mathcal{D}' = \left(\prod_{t\in D'}\{0,c_t\}, \mathcal{P}'\right)$, where D' is the set of possible tuples such that each $t \in D'$ has a multiplicity domain of $\{0,c_t\}$, with $c_t \in \mathbb{N}$. ~~The operation $\prod_{t\in D'}$ is the direct product of all such multiplicity domain pairs.~~ The tuples $t \in D'$ are partitioned into m independent blocks b_i, $i \in [m]$, of disjoint tuples. $\mathcal{P}'$ is the probability distribution across all worlds such that, given $\mathbf{W} \in \prod_{t\in D'}\{0,c_t\}$, $t, t' \in b_i$: $Pr[\mathbf{W}_t, \mathbf{W}'_t > 0] = 0$.* This is NOT the full defn.

Update like this

~~As X is standard for cartesian product the product is not specified~~

W?

We now present a reduction that is useful in deriving our results:

~~Actually this should be stated as a lemma/prop.~~

▶ **Definition 2.4** (c-TIDB reduction). *Given c-TIDB $\mathcal{D} = \left(\{0,\ldots,c\}^D, \mathcal{P}\right)$, let $\mathcal{D}' = \left(\prod_{t\in D'}\{0,c_t\}^{D'}, \mathcal{P}'\right)$ be the 1-BIDB obtained in the following manner: for each $t \in D$, create block $b_t = \left\{\langle t,j\rangle_{j\in[c]}\right\}$ of disjoint tuples, for all $j\in[c]$. The probability distribution $\mathcal{P}'$ is the one induced by $\mathbf{p} = \left((p_{t,j})_{t\in D, j\in[c]}\right)$ and the BIDB disjoint requirement, where given any $\mathbf{W} \in \prod_{t\in D'}\{0,c_t\}^{D'}$, $Pr[\mathbf{W}_{t,j}, \mathbf{W}_{t,j'} > 0] = 0$ for any $j \neq j' \in [c]$, such that for any $W \in \prod_{t\in D'}\{0,c_t\}^{D'}$, $Pr[\mathbf{W} = W] = \prod_{t\in D', j\in[c]} W_{t,j}\cdot j \cdot p_t$ if $\forall t \in D'$ $\nexists j \neq j' \in [c]$, $W_{t,j}, W_{t,j'} \geq 1$; otherwise $Pr[\mathbf{W} = W] = 0$.*[4]

~~truly generalized~~

there are for Φ not the PDB

~~This part should be in Def 2-3. Here you are defining what the prob distrib of D' should be~~

For the c-TIDB $\mathcal{D}$, each $X_t \in [c]$, while in the reduced 1-BIDB $\mathcal{D}'$, each $X_{t,j} \in \{0,1\}$. Hence, in the setting of 1-BIDB, we have the following semantics for generating lineage polynomials in $\mathcal{RA}^+$ queries: $\Phi'[\pi_A(Q), D', t_j] = \sum_{t_{j'} \in \pi_A(Q(D')) : t_{j'} = t_j} \Phi'[Q, D', t_{j'}]$, $\Phi'[\sigma_\theta(Q), D', t_j] = \begin{cases} \theta = 1 & \Phi'[Q, D', t_j] \\ \theta = 0 & 0 \end{cases}$, $\Phi'[Q_1 \bowtie Q_2, D', t_j] = \Phi'\left[Q_1, D', \pi_{attr(Q_1)}(t_j)\right] \cdot \Phi'\left[Q_2, D', \pi_{attr(Q_2)}(t_j)\right]$, $\Phi'[Q_1 \cup Q_2, D', t_j] = \Phi'[Q_1, D', t_j] + \Phi'[Q_2, D', t_j]$, and the base case now becomes $\Phi'[R, D', t_j] = j \cdot X_{t,j}$ (c.f. Fig. 1). Then given the disjoint requirement and the semantics for constructing the lineage polynomial over a 1-BIDB, $\Phi'[R, D', t]$ is of the same structure as the reformulated polynomial Φ_R of step i) from Definition 1.3, which then implies that $\widetilde{\Phi}'$ is the reduced polynomial that results from step ii) of Definition 1.3, and further that Lemma 1.4 immediately follows for 1-BIDB polynomials: $\mathbb{E}_{\mathbf{W}\sim\mathcal{P}'}\left[\Phi'(\mathbf{W})\right] = \widetilde{\Phi}'(\mathbf{p})$.

Put this in it's own fig.

$\widetilde{\Phi}'$ should have it's own defn

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

Sorry should have said this before

▶ **Corollary 2.5.** *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.*

---

[4] 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 $\mathbf{W}$.

Here's how to package this para. First give fig to define Φ'. Then give definition of $\widetilde{\Phi}'$. THEN compare the two to Φ & $\widetilde{\Phi}$ for TIDB. Finally state this as a lemma as well.

Need a filler sentence say readers can skip this section w/o losing anything

Connect this defn to the defn we have used for c-TIDB & 1-BIDB

2.2.1 Possible World Semantics

specify what is $\mathcal{P}$ here

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 $\mathcal{D}$. 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: $Pr[\omega \in \Omega] = \sum_{\omega' \in \Omega, Q(\omega') = Q(\omega)} Pr[\omega']$.

Aaron says: I am not sure the following paragraph is needed, since the reduction definition says pretty much the same thing. Unless that definition changes, we can get rid of this paragraph.

Suppose that $\mathcal{D}$ is a 1-BIDB. Instead of looking only at the possible worlds of $\mathcal{D}$, one can consider all worlds, including those that cannot exist due to, e.g., disjointness. The all worlds set can be modeled by $\mathbf{W} \in \{0,1\}^{cn}$, 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* $n = \sum_{t \in D} |b_t|$).We can denote a probability distribution over all $\mathbf{W} \in \{0,1\}^{cn}$ 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 $\mathbf{W}_{t,j}, \mathbf{W}_{t,j'} \geq 1$ 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. Appendix B.2 has more details.

This para can do this instead

Technically only for c-TIDB & not bag PDBs in general.

Recall Fig. 1 again, which defines the lineage polynomial $\Phi[Q, D, t]$ for any $\mathcal{RA}^+$ query. We now make a meaningful connection between possible world semantics and world assignments on the lineage polynomial.

▶ **Proposition 2.6** (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*):* $\mathbb{E}_{\mathbf{D} \sim \mathcal{P}}[Q(\mathbf{D})(t)] = \mathbb{E}_{\mathbf{W} \sim \mathcal{P}}\left[\Phi[Q, D, t](\mathbf{W})\right]$.

A formal proof of Proposition 2.6 is given in Appendix B.3.[5] We focus on the problem of computing $\mathbb{E}_{\mathbf{W} \sim \mathcal{P}}\left[\Phi[Q, D, t](\mathbf{W})\right]$ 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).

2.3 Formalizing Problem 1.6

Recall that

Problem 1.6 asks if there exists a linear time approximation algorithm in the size of a given circuit C which encodes $\Phi(\mathbf{X})$. 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 ~~particularly~~ specifically using circuits to model lineage polynomials, we can refer to these circuits as lineage circuits. However, when the meaning is clear, we will drop the term lineage and only refer to them as circuits.

▶ **Definition 2.7** (Circuit). *A circuit* C *is a Directed Acyclic Graph (DAG) 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*

[5] Although Proposition 2.6 follows, e.g., as an obvious consequence of [28]'s Theorem 7.1, we are unaware of any formal proof for bag-probabilistic databases.

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`, `partial`, `input`, `degree`, `Lweight`, *and* `Rweight`, *where* `type` *is the value type* $\{+, \times, \text{VAR}, \text{NUM}\}$ *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.

Aaron says: Does the following matter, i.e., does it point anything out special for our research? **EDIT**: Lemma 4.8 does use this (when C is a tree) to answer Problem 1.6 with a yes.

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 last one is not a tree

The circuits in Fig. 2 encode their respective polynomials in column Φ. Note that each circuit C encodes a tree, with edges pointing towards the root.

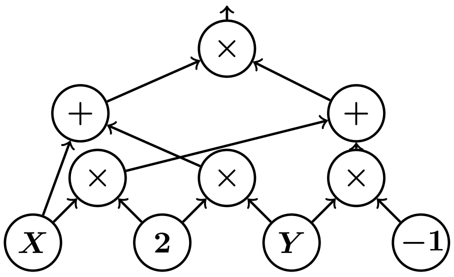

Figure 3 Circuit encoding of $(X + 2Y)(2X - Y)$

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.

maps

▶ **Definition 2.8** (POLY$(\cdot)$). ~~*Denote*~~ *POLY*(C) ~~*to be the function from*~~ *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:*

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

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. 3, while POLY$(\texttt{C}) = \Phi(\mathbf{X})$ is always the equivalent SMB representation.

▶ **Definition 2.9** (Circuit Set). `CSet`$(\Phi(\mathbf{X}))$ *is the set of all possible circuits* C *such that POLY*$(C) = \Phi(\mathbf{X})$.

The circuit of Fig. 3 is an element of `CSet`$\left(2X^2 + 3XY - 2Y^2\right)$. One can think of `CSet`$(\Phi(\mathbf{X}))$ as the infinite set of circuits where for each element C, POLY$(\texttt{C}) = \Phi(\mathbf{X})$.

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

▶ **Definition 2.10** (The Expected Result Multiplicity Problem). *Let* $\mathcal{D}$ *be an arbitrary* BIDB-*PDB and* $\mathbf{X}$ *be the set of variables annotating tuples in* D_Ω. *Fix an* $\mathcal{RA}^+$ *query* Q *and a result tuple* t. *The* EXPECTED RESULT MULTIPLICITY PROBLEM *is defined as follows:*

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

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 abstract the cost of a join.

▶ **Definition 2.11** (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 $T_{join}(R_1, \ldots, R_m) \geq |R_1 \bowtie \ldots \bowtie R_m|$.*

needs to be updated. OPT(Q) needs to be brought in here'

Worst-case optimal join algorithms [37, 36] and query evaluation via factorized databases [39] (as well as work on FAQs [33]) 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:

$$T_{det}\left(R, \overline{D}, c\right) = |\overline{D}.R| \quad T_{det}\left(\sigma Q, \overline{D}, c\right) = T_{det}\left(Q, \overline{D}\right) \quad T_{det}\left(\pi Q, \overline{D}, c\right) = T_{det}\left(Q, \overline{D}, c\right) + \left|Q(\overline{D})\right|$$
$$T_{det}\left(Q \cup Q', \overline{D}, c\right) = T_{det}\left(Q, \overline{D}, c\right) + T_{det}\left(Q', \overline{D}, c\right) + \left|Q\left(\overline{D}\right)\right| + \left|Q'\left(\overline{D}\right)\right|$$
$$T_{det}\left(Q_1 \bowtie \ldots \bowtie Q_m, \overline{D}, c\right) = T_{det}\left(Q_1, \overline{D}, c\right) + \ldots + T_{det}\left(Q_m, \overline{D}, c\right) + T_{join}(Q_1(\overline{D}), \ldots, Q_m(\overline{D}))$$

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

Finally, 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 $O(T_{det}(Q, D, c))$. Recall we assumed these two bounds when we moved from Problem 1.5 to Problem 1.6.

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 $(Q, \mathcal{D})$ for Problem 1.2 where $\mathcal{D}$ is a 1-TIDB. Note that this implies hardness for c-TIDBs ($c \geq 1$), BIDBs and general bag-PDB, showing Problem 1.2 cannot be done in $O\left(T_{det}\left(\mathrm{OPT}\left(Q\right), D, c\right)\right)$ runtime.

Problem 1.2 only talks about c-TIDBs.

you should definitely mention the results also work for 1-BIDB & other more general bag 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): $\#(G, ⧊)$ (the number of triangles), $\#(G, ⫶⫶⫶)$ (the number of 3-matchings), and the latter's generalization $\#\left(G, ⫶ \cdots ⫶^{k}\right)$ (the number of k-matchings). We use $T_{match}\left(k, G\right)$ to denote the optimal runtime of computing $\#\left(G, ⫶ \cdots ⫶^{k}\right)$ exactly. Our hardness results in Sec. 3.2 are based on the following hardness results/conjectures:

▶ **Theorem 3.1** ([11]). *Given positive integer k and undirected graph $G = (V, E)$ with no self-loops or parallel edges, $T_{match}\left(k, G\right) \geq \omega\left(f(k) \cdot |E|^c\right)$ for any function f and ~~fixed~~ any constant c independent of $|E|$ and k (assuming $\#\mathsf{W}[0] \neq \#\mathsf{W}[1]$).*

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

We note that the above conjecture is somewhat non-standard. In particular, the best known algorithm to compute $\#\left(G, ⫶ \cdots ⫶^{k}\right)$ takes time $\Omega\left(|V|^{k/2}\right)$ (i.e. if this is the best algorithm then $c_0 = \frac{1}{4}$) [11]. 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 $\#\left(G, ⫶ \cdots ⫶^{k}\right)$.

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

Also why do you have different P_i for each i? We only use same p for all :)

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

The so called *Triangle detection hypothesis* (cf. [34]), 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*

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

Spell it out. Could be as simple as saying apply Fig. 1 to the Q_2 we set the domain

Returning to Fig. 2, it is easy to see that $\Phi_G^k(\mathbf{X})$ is the lineage polynomial whose structure mirrors the query Q_2 from Sec. 1. Let us alias

```
SELECT 1 FROM T t_1, R r, T t_2
WHERE t_1.city = r.city1 AND t_2.city = r.city2
```

as R_i for each $i \in [k]$. The query Q^k then becomes

```
SELECT COUNT(*) FROM R_1 JOIN R_2 JOIN···JOIN R_k
```

make sure notation is consistent w/ def /notation from Sec 1.

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_i and R has tuples corresponding to the edges E (each with probability of 1).[6] In other words, for 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 c-TIDB-lineage polynomial.

1

> **Aaron says: @atri**, we discussed this last meeting, but I am not sure if we really pinpointed how we want to treat (*in a consistent manner*) the runtime of Lemma 3.5 since k is a constant and m is growing. Would it be a good idea to be consistent with the O_ϵ notation of Problem 1.5 and say $O_k(m)$

Sure!

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

▶ **Lemma 3.5.** *Let Q^k and D be as defined above. Then $T_{det}\left(Q^k, D\right)$ is $O(km)$.*

3.2 Multiple Distinct p Values

We are now ready to present our main hardness result.

---

[6] 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.

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

Note that the second row of Table 1 follows from Proposition 2.6, Theorem 3.6, Lemma 3.5, and Theorem 3.1 while the third row is proved by Proposition 2.6, 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 2.5 for EC) would lead to an improvement over the state of the art *upper* bounds on $T_{match}\left(k, G\right)$. 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 $\widetilde{\Phi}_G^3(p, \ldots, p)$ 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 2.6 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).

First say we want approx in ↑ time. THEN say we do that by changing the result via circuits

4 $1 \pm \epsilon$ Approximation Algorithm

In Sec. 3, we showed that Problem 1.2 cannot be solved in $O\left(T_{det}\left(\text{OPT}\left(Q\right), D, c\right)\right)$ runtime. With this result, we now design an approximation algorithm for our problem that runs in $O\left(|\mathtt{C}|\right)$ for a very broad class of circuits, (thus affirming Problem 1.6 see the discussion after Lemma 4.8 for more). The following approximation algorithm applies to c-TIDB lineage polynomials and general BIDB (over bag-$\mathcal{RA}^+$ query semantics) lineage polynomials in practice, where for the latter we note that a 1-TIDB is equivalently a 1-BIDB (blocks are size 1) and our experimental results (see Appendix D.10) using 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.

This sentence is too long

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 $\mathtt{E}(\mathtt{C})$ of circuit $\mathtt{C}$ which is used in our auxiliary algorithm for sampling monomials when computing the approximation.

↑ give flow ptr

Did not read rest of sec 4 super carefully

You should lead with the main result here.

i.e. we solve P 1.6 from any fixed $\epsilon > 0$

4.2 Our main result

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

$$\widetilde{\Phi}\left(p_1, \ldots, p_n\right) = \sum_{(\mathtt{v},\mathtt{c}) \in \mathtt{E}(\mathtt{C})} \mathbb{1}_{\text{ISIND}(\mathtt{v_m})} \cdot \mathtt{c} \cdot \prod_{X_i \in \mathtt{v}} p_i. \tag{2}$$

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

Where is the corollary for c-TIDB?? That is, the main results of this section

▶ **Theorem 4.7.** *Let C be an arbitrary 1-BIDB circuit, define $\Phi(\mathbf{X}) = \text{POLY}(C)$, let $k = \text{DEG}(C)$, and let $\gamma = \gamma(C)$. Further let it be the case that $p_i \geq p_0$ for all $i \in [n]$. Then an estimate $\mathcal{E}$ of $\widetilde{\Phi}(p_1, \ldots, p_n)$ 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

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

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

The restriction on γ is satisfied by any 1-TIDB (where $\gamma = 0$ in the equivalent 1-BIDB of Definition 2.4) as well as for all three queries of the PDBench BIDB benchmark (see Appendix D.10 for experimental results).

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 $\text{SIZE}(\mathtt{C})$ comes from the time take to run ONEPASS once (ONEPASS essentially computes $|\mathtt{C}|\left(1, \ldots, 1\right)$ using the natural circuit evaluation algorithm on $\mathtt{C}$). 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 $\mathtt{C}$ all of which by definition have length at most $\text{DEPTH}(\mathtt{C})$).

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

▶ **Lemma 4.8.** *For any BIDB circuit C with $\text{DEG}(C) = k$, we have $|C|\left(1, \ldots, 1\right) \leq 2^{2^k \cdot \text{DEPTH}(C)}$. Further, if C is a tree, then we have $|C|\left(1, \ldots, 1\right) \leq \text{SIZE}(C)^{O(k)}$.*

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

Query size. No if is not exactly k

why should ε & δ be constants?

Results hold for any $\varepsilon, \delta > 0$

Aaron says: Is it standard to assume that in the asymptotic notation above ϵ and δ are constant? Otherwise this does not uphold Problem 1.6.

Finally, note that by Proposition E.1 and Lemma E.2 for any $\mathcal{RA}^+$ query Q, there exists a circuit $\mathtt{C}^*$ for $\Phi[Q, D, t]$ such that $\text{DEPTH}(\mathtt{C}^*) \le O_{|Q|}(\log n)$ and $\text{SIZE}(\mathtt{C}) \le O_k\left(T_{det}\left(Q, D_\Omega\right)\right)$. Using this along with Lemma 4.8, Theorem 4.7 and the fact that $n \le T_{det}\left(Q, D_\Omega\right)$, we answer Problem 1.5 in the affirmative as follows:

▶ **Corollary 4.9.** *Let Q be an $\mathcal{RA}^+$ query and $\mathcal{D}$ be a* 1-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 $O_{k,|Q|,\epsilon',\delta}\left(T_{det}\left(Q, D, c\right)\right)$ (given Q, D and p_i for each $i \in [n]$ that defines $\mathcal{P}$).*

Aaron says: What is $|Q|$? Isn't that just k?

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 [46] (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. [14] and Olteanu et al. [21] 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 [31, 41, 44]. 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 [47] (TIDBs), block-independent databases (BIDBs) [42], and *PC-tables* [26]. Fink et al. [19] 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 $T_{det}\left(Q, D, c\right)$. The runtime bound is also limited to a specific class of (hierarchical) queries, suggesting the possibility of a generalization of [14]'s dichotomy result to bag-PDBs.

Several techniques for approximating tuple probabilities have been proposed in related work [20, 15, 38, 12], relying on Monte Carlo sampling, e.g., [12], or a branch-and-bound paradigm [38]. Our approximation algorithm is also based on sampling.

Compressed Encodings are used for Boolean formulas (e.g, various types of circuits including OBDDs [29]) and polynomials (e.g., factorizations [39]) some of which have been utilized for probabilistic query processing, e.g., [29]. Compact representations for which probabilities can be computed in linear time include OBDDs, SDDs, d-DNNF, and FBDD. [16] studies circuits for absorptive semirings while [45] 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.